PERIYAR INSTITUTE OF DISTANCE EDUCATION (PRIDE)

PERIYAR UNIVERSITY

SALEM - 636 011.

B.Sc. COMPUTER SCIENCE

THIRD YEAR

PAPER - VIII : INTERNET AND PROGRAMMING JAVA

Prepared By
M.KALADEVI
9, Royal Gardens,
Meyyanur Bye-pass Road,
Salem-4

B.Sc. COMPUTER SCIENCE
THIRD YEAR
PAPER – VIII : INTERNET AND PROGRAMMING JAVA

Unit – I:

Internet Connection Concepts: Internet Communication Protocols – Types Of Internet Connections - Internet Service Providers - Security Issues On The Internet.

E-Mail Concepts: How Do You Get Your E-Mail? - E-Mail Addressing - Message Headers – Downloading E-Mail – Formatted E-Mail – Attaching Files To Messages – Web Based E-Mail – Mail Away From Home- Avoiding Viruses. **E-Mail Security:** Reasons To Secure Messages, Public Key Cryptography, Using Cryptography With E-Mail.

Online Chatting And Conferencing Concepts: Forms Of Chat, Messaging And Conference – How The Chat Work. **WWW Concepts:** Elements Of The Web, Web Browsers, Security And Privacy Issues.

Unit - II:

Fundamentals Of Object Oriented Programming: Introduction - Object Oriented Paradigm- Basic Concepts Of Object Oriented Programming – Benefits Of OOP – Applications Of OOP. **JAVA Evolution:** JAVA History – JAVA Features – How JAVA Differs From C And C++ - JAVA And Internet – JAVA And World Wide Web – Web Browsers – Hardware And Software Requirements – JAVA Support Systems – JAVA Environment. **Overview Of JAVA Language:** Introduction Simple JAVA Program – More Of JAVA - An Application With Two Classes – JAVA Program Structure – JAVA Tokens – JAVA Statements – Implementing A JAVA Program – JAVA Virtual Machine – Command Line Arguments – Programming Style.

Constants, Variables And Data Types: Constants - Variables - Data Types – Declaration Of Variables – Giving Values To Variables – Scope Of Variables – Symbolic Constants – Type Casting – Getting Values Of Variables.

Operators And Expressions: Introduction – Arithmetic Operators- Relational Operators – Logical Operators – Assignment Operators – Increment And Decrement Operators – Conditional Operators – Bit Wise Operators – Special Operators – Arithmetic Expressions - Evaluation Of Expressions - Precedence Of Arithmetic Operators – Type Conversions In Expressions – Operator Precedence And Associativity – Mathematical Functions.

Decision Making And Branching: Introduction – Decision Making With If Statement – Simple If Statement – The If…Else Statement – Nesting Of If…Else Statement –The If…Else Ladder – The Switch Statement. **Decision Making And Looping:** Introduction – The While Statement – The Do Statement – The For Statement – Jump In Loops – Labeled Loops.

Unit – III:

Classes, Objects And Methods: Introduction – Defining A Class – Adding Variables – Adding Methods – Creating Objects – Accessing Class

Members – Constructors – Methods Overloading – Static Members – Nesting Of Methods. **Inheritance:** Extending A Class – Overriding Methods – Final Variables And Methods – Final Classes – Finalizer Methods – Abstract Methods And Classes – Visibility Control. **Arrays String And Vectors:** Arrays – One Dimensional Arrays – Creating An Array – Two Dimensional Arrays – Strings – Vectors – Wrapper Classes. **Interface:Multiple Inheritance:** Introduction – Defining Interfaces – Extending Interfaces – Implementing Interfaces – Accessing Interface Variables.

Unit - IV:

Packages: Putting Classes Together: Introduction – JAVA API Packages – Using System Packages – Naming Conventions – Creating Packages – Accessing A Packages – Using A Packages – Adding Class To A Package – Hiding Classes.

Multithreaded Programming: Introduction – Creating Threads – Extending The Thread Class – Stopping And Blocking A Thread - Life Cycle Of A Thread – Using Thread Methods - Thread Exceptions - Thread Priority – Synchronization – Implementing The Runnable Interface.

Managing Errors And Exceptions: Introduction – Types Of Errors – Exceptions – Syntax Of Exception Handling Code – Multiple Catch Statements – Using Finally Statement – Throwing Our Own Exceptions For Debugging.

Unit - V:

Applet Programming: Introduction – How Applets Differ From Applications – Preparing To Write Applets – Building Applet Code – Applet Life Cycle – Creating An Executable Applet – Designing A Web Page – Applet Tag – Adding Applet To HTML File – Running The Applet – More About Applet To Passing Parameters To Applets – Aligning The Display –More About HTML Tags – Displaying Numerical Values – Getting Input From The User.

Graphics Programming: Introduction – The Graphics Class – Lines And Rectangles – Circles And Ellipses – Drawing Arcs – Drawing Polygons – Line Graphs – Using Control Loops In Applet – Drawing Bar Charts.

Managing Input / Output Files: Introduction – Concepts Of Streams – Stream Classes – Byte Stream Classes – Character Stream Classes – Using Stream – Other Useful I/O Classes – Using The File Classes – Input / Output Exceptions – Creation Of Files – Reading / Writing Characters – Reading / Writing Bytes – Handling Primitive Data Types- Concatenation And Buffering Files – Random Access Files – Interactive Input And Output – Other Stream Classes.

TEXT BOOKS:

“The Complete reference – Internet Millennium Edition”, Margaret Levine Young T.M.H, New Delhi. (Unit – I)

“Programming with JAVA”, E.Balagurusamy. T.M.H, New Delhi. 2nd Edition. (Unit – II to Unit – V)

INTRODUCTION

Dear Students,

This package consists of five units dealing with concepts of **Internet and Programming Language JAVA**. In first unit, we introduce the need for studying Internet; this unit mainly highlights the concepts of Internet, e-mail, Online Chatting & Conferencing and World Wide Web.

The second unit deals with the fundamentals of object oriented programming and the evolution of Java programming. This unit also consists of elaborate discussion on object oriented programming and basic concept of Java programming and its application.

The third unit explores the process of classes & objects and discuss about the different inheritance concept and its usage. You can also learn about arrays & vectors and its applications.

The fourth unit deals with the common usage of the packages, multithreaded programming and how can manage the errors & exceptions.

In the fifth unit, we introduce the applet & graphics programming and its usages. An applet programming concepts mainly used to create web pages in the Internet. We can also explore the concepts of managing input & output files.

All the above said five units of lesson in **SIM pattern** of this package have been prepared by **Mrs. M.Kaladevi, M.C.A.,M.Phil.,** to make your task much easier while going through it.

PRIDE would be happy if you make use of this learning material to enrich your knowledge and skills.

UNIT – I

UNIT STRUCTURE

UNIT – I

1.1 Introduction

Internet is the world **largest computer network**, the *network of networks* that connects computers *all over the world*. Internet is a worldwide collection of computer networks connecting academic, governmental, commercial, and organizational and individual's sites. It was created nearly a *quarter centaury* ago as a project for the department of defense, U.S.A. It's aim was to create a method for widely separated computers to transfer the data efficiently even in the event of nuclear attack. It provides access to communication services and information resources to millions of users around the globe.

Internet service include *direct communication (e-mail, chat, etc.,), online conferencing (Usenet news group, email discussions lists), distributed information resources (world wide web, gopher), remote login and file transfer (Telnet, FTP) and many other valuable tools and resources.* From a limited number of computers and users, the Internet today has grown to thousands of regional networks that connect millions of users. Any single individual, company or country does not own this global network.

1.2 Objectives

In this unit, we shall explore the process of Internet connection concepts. It is fully covered as basic concepts of the Internet, E-mail and E-mail Security concepts. After studying this lesson, you should be able to:

- ❖ Describe the Internet connection concepts.
- ❖ Describe the Internet and email concepts.
- ❖ Major elements of public key cryptography and using cryptography with email.
- ❖ Describe various chatting and conferencing concepts.
- ❖ Describe various WWW concepts.

1.3 Internet Connection Concepts

1.3.1 Internet Communication Protocols

Computers connected to the Internet communicate by using the **Internet Protocol (IP),** which slices information into packets (Chunks of data to be transmitted separately) and routes them to their destination. Along with IP, most computers on the **Internet communicate with Transmission Control Protocol (TCP),** and **the combination** is called **TCP/IP**.

Before Windows 95, Windows users had to get a separate TCP/IP communications program, and several free and commercial programs were available. All versions of Windows starting with Windows 95 come with TCP/IP connection software called **Dial-up Networking**. A similar standard exists for the Macintosh: **MacTCP. MacOS 7.6.1** or later come with **MacTCP- compatible Internet connection software**. In 7.1.1 through 9.2, it's called **Open Transport/PPP** or **Apple Remote Access**. In Mac OS X, Internet connection software is built right into the underlying UNIX operating system.

Internet Hosts

Each computer on the Internet is called **a host computer** or **host**. The computers on the Internet and there are now millions of Internet hosts-are connected by **cables, phone lines,** and **satellite connections**. They include large mainframe computers, smaller minicomputers, and personal computers. When your PC or Mac dials into an Internet account, your computer is an Internet hosts, too.

Internet Protocol (IP) Addresses

Each host computer on the Internet has a unique number, called its **IP address**. IP addresses are in the format **xxx.xxx.xxx.xxx**, where each **xxx** is a number from **0** to **255**. IP addresses identify the host computers, so that packets of information reach the correct computer. You may have to type IP addresses when you configure your computer for connection to the Internet.

If you connect to the Internet by using a dial-up account, your *Internet service provider (ISP)* assigns your computer an ***IP address*** each time that you connect. This system enables your ISP to get along with fewer IP addresses, because it needs only enough IP addresses for the number of users who are connected simultaneously (as opposed to assigning a permanent IP address to each customer of the ISP). If you use a high-speed DSL or cable Internet account, you may have static (unchanging) IP address, or your ISP may assign you an address each time you connect. Static IP addresses get rare every year and usually cost extra.

On windows XP systems, you can find out your computer's IP address by choosing **Start | Control Panel | Network Connections** to see your Internet and LAN connections. **Right-click** your Internet or LAN connections; choose **Status** from the menu that appears, and **click** the support **tab**.

Domain and Host Names

The name of each host computer consists of a series of words separated by dots. The last part of the domain name is called the **top-level domain (TLD).** The TLDs of *three or more letters* are used mainly in the United States and indicate the *type of organization* that owns the domain. The original sever **three-letter TLDs** are the following.

Com	Originally for commercial organizations, but now used by individuals, government agencies, and nonprofits as well
Net	Internet service providers and other network-related companies
Org	Noncommercial (often nonprofit) organizations
Gov	U.S. Government agencies
Mil	U.S. military
Edu	Educational domains
Int	International organizations like NATO and the International Red Cross

Aero	Airlines
Arpa	Internet infrastructure
Biz	Businesses
Coop	Cooperatives
Info	Anyone
Museum	Museum
Name	Individuals

Two-letter TLDs indicate the country in which the organization that owns the computer is located. U.S. organizations can register domains that end with us.

The **last two parts** of a host computer name constitute the **domain**. The **second-to-last part** of the name is chosen by the **organization** that owns the computer and is usually some variant of the organization's name. **For example**, computers at the U.S. president's offices at the **White House** have the domain **whitehouse.gov**. Computers at the **McGraw-Hill** publishing company are named with the domain **mcgraw-hill.com**.

Because most organizations own more than one computer on the Internet, most host computer names have at least **one more part**, preceding the domain name and called a **third-level domain**. This *additional part* is assigned by the *organization* itself. **For example**, the **gurus.com** domain has several host names, including www.gurus.com (the main web site), **net.gurus.com** (the Internet Gurus web site), and **wine.gurus.com** (the web site of the society of Wine Educators). By far, the most widely used addition to domain names is www, because it is frequently used for an organization's web serve (the computer that stores web pages).

Servers and Clients

Many of the host computers on the *Internet offer services to other computers on the Internet*. **For example**, your ISP probably has a host computer that handles your incoming and outgoing mail. Computers that provide services for other computers to use are called **servers**. The software run by server computers to provide services is called server software.

Many of the computers on the *Internet use servers to get information*. **For example**, when your computer dials into an Internet account, your e-mail program downloads your incoming messages from your ISP's mail server. Programs that ask servers for services are called **clients**. Your e-mail program is more properly called an **e-mail client**.

Types of servers and **clients** that you may encounter:

❖ **Mail servers** handle incoming and outgoing e-mail. Specifically, **post office protocol (POP or POP3)** and **IMAP (Internet Message Access Protocol)** servers store incoming e-mail, whereas **Simple Mail Transfer Protocol (SMTP)** servers relay outgoing e-mail. Mail clients get incoming messages from, and send outgoing messages to, a mail server, and enable you to read, write, save, and print messages.

- **Web servers** store web pages and transmit them in response to requests from web clients, which are usually called **browsers.**
- **FTP servers** store files that you can transfer to or from your computer if you have an FTP client.
- **News servers** store Usenet newsgroup articles that you can read and send if you have a **news client** or **newsreader**.
- **IRC servers** act as a switchboard for the **Internet Relay Chat (IRC)** channels. To participate, you use an IRC client.

Ports and Port Numbers

One host computer can run more than one server program. **For example**, a small ISP might have one computer running a POP sever, SMTP server, web server, and news server. TO keep requests for information straight, each type of server responds to packets sent to specific ports (input for a specific Internet service). Ports are numbered and standard port numbers used throughout the Internet. You almost never need to type post numbers, but here are some widely used port numbers in case you do:

Port Number	**Internet Service**
21	FTP (file transfer)
22	Telnet (remote login)
25	SMTP (mail relaying)
80	World Wide Web
110	POP3 (Storage of incoming mail)
194	IRC (online chat)
532	Usenet newsgroups

The Domain Name System and DNS Servers

A **Domain Name System (DNS)** server translates between the numeric IP addresses that identify each host computer on the Internet and the corresponding domain names. People prefer to use host names because they are easier to type and remember, but actual Internet communications use the numeric addresses. **For example**, if your browse requests a web page from the Yahoo! Web site, which has the host name www.yahoo.com, a DNS server translates that name to 204.71.200.69, one of Yahoo's web servers, and then sends the request to that IP address.

1.3.2 Types Of Internet Connections

To connect to the Internet, you connect your computer to a computer that is on the Internet, usually one run by an ISP. You can connect your computer by using a dial-up phone line, which is how most home users connected to the Internet during the 1990s. If you need to connect at higher speeds than a regular phone line allows, you can get a high-speed phone line, assuming that your phone company offers them. You may have **three options**, depending on what your phone company offers: **DSL, ISDN,** or **a leased line**.

If your cable TV company offers an Internet service, you can connect your computer to the Internet by using a cable TV connection. Rural users may consider installing a satellite dish for Internet connections, whereas urban users may have access to a wireless connection.

High-speed Internet connections, including **DSL, ISDN, leased lines, cable Internet, and satellite**, are all called **broadband connections**.

Dial-Up Connections

A **dial-up connection** to the Internet works over an **ordinary phone line**. Dial-up connections use the **Point-to-point Protocol (PPP)** and are also called **PPP accounts**.

To use a dial-up account, you need a *modem*. Most computers come with an internal modem. You connect only when you want to use Internet services and disconnect when you are done.

DSL Connections

Digital Subscriber Line (DSL) is a family of all-digital, high-speed lines that use your normal phone wires with special modems on wither end. Most DSL lines are actually ADSL (Asymmetric Digital Subscriber Line). ADSL is optimized for the way many people use the Internet: more downloads than uploads. The line is asymmetric, because it has more capacity for data received by your computer than for data that you send (such as e-mail and browser commands). The downstream bandwidth (data transfer speed from the Internet to your computer) can range from 384 Kbps to 8 megabits per second (Mbps). The upstream bandwidth (Speed from your computer to the Internet) can range from 90 Kbps to 768 Kbps.

With a **DSL line**, you can connect your computer to the Internet and talk on the phone at the same time on the same phone line. The speed of your Internet connection may drop while you are talking on the phone.

Costs for DSL lines ate higher than for regular phone lines. The **phone company** or **ISP** usually provides the **DSL modem**, which match the DSL modem installed at their end. Because there are several competing DSL modem standards, not all DSL modems work with all DSL lines. DSL modems usually *connect to your computer* through an **Ethernet** or other **network card** in your computer. When you sign up for a DSL line, the phone company comes to your location, installs the DSL modem, and configures your computer to use it.

ISDN Connections

Integrated Services Digital Network (ISDN) lines are also available from many local telephone companies. **ISDN** is and upgraded phone line that can be used for faster Internet access and for regular voice calls.

To connect your computer to an ISDN line, you need an **ISDN adapter**. Your *Phone Company or ISP* usually provides the *ISDN adapter* as part of the sign-up fee.

ISDN adapters may be

- **Internal ISDN adapters :**Internal adapters bypass any serial port bottleneck, so you can get full 128 Kbps out of your ISDN line.
- **External ISDN adapters**: External adapters usually connect to your computer's serial port.

You plug in your existing equipment to the adapter. Also, check to see whether the adapter has *full ringing support*. That is, when a phone call comes in, does the phone actually ring, or do you just get some flashing lights on the front of the adapter? Flashing lights are easy to miss when you're busy poring over your computer screen.

Leased Lines

If you need *to transfer very large amounts of data* or *run Internet server software*, contact your telephone company for a **leased line**, the same type of phone line that organizations use to connect corporate offices.

Cable TV Internet connections

Cable modem service is the competitive threat that's caused phone companies to accelerate their ADSL efforts. The same network that brings you dozens of TV channels can now bring you millions of web sites. Downstream speeds are impressive-the line can theoretically bring you data as fast as 30 Mbps, much faster than your computer can handle it-bu upstream speed depends on line quality. Large cable companies are spending money to upgrade their networks to **hybrid fiber-coaxial (HFC)** to better handle **two-way traffic**.

Satellite Internet Connections

Digital Satellite Systems (DSS), or **direct broadcast satellite**, lets you get Internet information by satellite. To connect, you can use a 24-inch antenna, a coaxial cable, a PC adapter card, and Windows-based software. With early satellite systems, you received data from the Internet at a high speed via the satellite, but so send data to the Internet, you needed a *dial-up connection and an ISP*.

Wireless Internet Connections

In a few urban areas, you can use *wireless Internet access*. TO set it up, you attach a radio modem, about the size of a deck of cards, to your laptop.

Another way to connect to the Internet via wireless is by using a digital cell phone that includes Internet connectivity. Because these devices have tiny screens, limited keyboards, no mouse, and slow connection rates, Internet content mist be tailored to them and is usually limited to text. Check with your cell phone company to find out whether they offer cellular Internet access.

Some **ISPs** offer wireless connections to *personal data assistants* (PDAs) such as the **BlackBerry, Palm, CompaqiPaq, or HandSpring Visor**. These small devices have small screens, but you can use them to read your e-mail and browse the Web. Like cell phones, PDAs require web content to be tailored to their small screens and their slow connections rates. Check with your local ISPs to find out what they offer.

Connecting Local Area Networks to the Internet

Homes or **organizations** that have many PCs can connect the computers in network and then connect that network to the Internet. This method is more efficient than connecting each PC to the Internet. This method is more efficient than connecting each PC to the Internet by using its own modem and phone line. Colleges, universities, and large corporations have used the Internet this way for a decade, and smaller offices and even homes increasingly do, too.

1.3.3 Internet Service Providers

An **Internet service provider (ISP)** is an organization that provides **Internet accounts**, whether **dial-in, DSL, ISDN, cable, satellite, or wireless**. Thousands of ISPs exist in the United States, including dozens of ISPs with dial-up access phone numbers throughout the country, and many with phone numbers in limited regions. **For example**, EarthLink (www.earthlink.com) has access phone numbers in all major U.S. cities.

In addition to connecting you to the Internet, here are some other features that your Internet account may provide:

- **E-mail mailboxes:** Your account almost certainly with at least one e-mail mailbox on a POP or IMAP server.
- **Web server space:** Most Internet accounts include a modest amount of disk space on web server, so that you can make your own web pages accessible to the Internet.
- **Domain hosting:** If you want your own domain name, most ISPs can host your domain, so that e-mail to the domain lands in your mailbox, and web addresses in your domain refer to pages that you store on your ISP's web server.

1.3.4 Security Issues On The Internet

Internet has become widely used by the *public*; many security issues have arisen, including *viruses*, *cookies*, and *firewalls*.

Protecting Your Computer from Viruses

A **virus** is *self-replicating program*, frequently with destructive side effects. Viruses that spread via e-mail attachments are called **worms**. When the Internet was young (ten years ago), viruses were spread only in programs that were downloaded from FTP servers or passed around on floppy disks. Now your computer can catch a virus from an infected e-mail message.

Viruses can't travel in plain text, like e-mail messages. You **receive** a virus **as an *e-mail attachment*** or **in a *file you download***. Always run a virus-checking program on all computers that connect to the Internet. Don't just install the program; you also need to sign up to receive updates as new viruses appear.

If a virus arrives attached to e-mail messages, your **virus-checker** should pop up a message asking what you want to do. You usually have the option of deleting the entire attachment, deleting the virus from the attachment but saving the rest of the file.

Background checking also provides the best protection against the subtle (and rare) viruses that arrive, not as files, but as infections of your computer's memory that can travel through your web browser. In some instances, however, background checking can slow down your system, cause conflict with other background programs.

Protecting Your Computer from Intruders with Firewalls

A **firewall** is a program that *controls what information passes from one network to another*. You can use a firewall between your PC and the Internet to stop outsiders from getting access to your PC via the Internet.

How Firewall Work

Each packet on the Internet is addressed to a specific port number, and you can control access by port. You do not want outsiders to be able to use this port, so you may want to block anyone on the Internet from accessing this port.

A firewall controls which ports are open, refusing to respond to packets addressed to other ports. Some firewalls control only incoming information.

A good firewall program *monitors both incoming and outgoing packets* and makes sure that outgoing packets come from a program that you know about. If it doesn't recognize program, it alerts you and asks what to do.

Microsoft's Internet Connection Firewall

Windows XP comes with the **Internet Connection Firewall**, a firewall program that protects your incoming, but no outgoing, Internet traffic. Your only configuration option is to turn the firewall on or off there are no other settings.

Virtual Private Networks

Many large organizations have LANs that enable people within the organization to share files. Although the LAN is connected to the Internet, most organizations install a firewall to block Internet users from accessing information on the LAN. You can connect to the Internet through an Internet provider, but how can you access your organization's LAN?

Virtual Private Networking (VPN) provides a way for an authorized computer on the Internet to tunnel through a firewall and connects to a computer on a LAN. When you are on the road and you need to connect to computer at your office, VPN is the way to make the connection.

To connect to a LAN through a firewall, the firewall must support **point-point Tunneling Protocol (PPTP)**, which lets VPN connect you through the firewall. Your organization's LAN administrator sets up a VPN server, the program on the LAN that provides PPPTP. Both the VPN client and the VPN server must have Internet connections.

Choosing Passwords

On the Internet, you need a password for your account, a password for your e-mail mailbox and passwords for the many web sites with which you do business.

- If you're protecting anything important, don't use any English word or common name.
- Use your brain sludge
- Use acronym
- Stick a number in, spell it wrong or glue a few words together.
- Don't write down passwords. Write down hints.

1.3.5 Self Assessment Questions

Fill in the blank

1. Each computer on the Internet is called as _________________.

2.A ______________ is self-replicating program, frequently with destructive side effects

True / False

1. ISDN is used for faster Internet access.

2. Each host computer on the Internet has a unique IP address.

Multiple Choices

1. ISP stands for

a) Intranet service provider	b) Internet service provider
c) Internet service policy	d) Information service provider

2. Expansion for DSS is

a) Digital Satellite Systems	b) direct broadcast satellite
c) both a) & b)	d) none of the above

Short Answer

1. What is an Internet?

____

____

2. Define server & client

____

____

3. What is the purpose of firewall?

____

____

1.4.E-Mail Concepts

1.4.1. How Do You Get Your E-Mail?

You receive Internet e-mail when it's sent to your unique e-mail address. E-mail messages are passed through the Internet by using a protocol called **Simple Mail Transfer Protocol (SMTP).**

Receiving Incoming Messages

E-mail can arrive at any time, you need an e-mail mailbox that resides on mail server, a computer that is permanently connected to the Internet and that is set up to handle your incoming e-mail. Like your postal service mailbox, the mail server is able to accept e-mail at any time and store it until you delete it. Depending on the type of connection that you have, you may download e-mail from the mail server to your computer, or you may read your e-mail while it sits on the mail server.

Mail servers receive and store e-mail messages in mailboxes by using a protocol called **Post Office Protocol (POP)** or **POP3** or **IMAP (Internet Message Access Protocol).**

To read your e-mail, you need a **mail client** or **e-mail application** such as *outlook Express, Netscape Mail, Netscape Messenger, or Eudora*. A client application works in concert with a server – in the case of e-mail, a mail server collects your e-mail, and your mail client enables you to read it.

If you have POP mail, you need a POP mail client that copies your mail from the mail server to your local computer. If you have an IMAP server, the mail stays on the remote server. You use your mail client to read it and do all your mail manipulations.

Sending Outgoing Messages

You write e-mail on your own computer by using your e-mail application. Then, you transfer the messages to an SMTP server-a mail server that accepts outgoing e-mail. Your **Internet service provider (ISP)** probably runs both an SMTP server and a POP or IMAP server for its customers; the SMTP server that takes care of sending your e-mail messages.

Ways of Accessing E-mail

- ❖ You may use mail client that downloads your incoming messages from the POP server to your computer and uploads your incoming messages from the POP server to your computer and uploads your outgoing messages to the SMTP server. This may occur through a *local area network (LAN)* or *through a dial-up, DSL, ISDN,* or *cable connection.*
- ❖ You may use a web-based e-mail service
- ❖ You may use an online service, such as America Online, which has its own e-mail program
- ❖ You may get your e-mail through a LAN, a common system at large organizations. If your organization has some sort of Internet connection, e-mail arrives in the company's POP or IMAP server. You then read your e-mail either on the server, using an e-mail application, or on your

own computer, by downloading your e-mail from the server through the LAN by using an e-mail application.

1.4.2. E-Mail Addressing

Internet e-mail address consists of two parts joined by @

- ❖ **User name:** User names can contain characters (Letters, Numbers, Underscores, periods, and Special Characters). They cannot contain commas, spaces, or parentheses.
- ❖ **Host or domain name:** The host name provides the Internet location of the mailbox, usually the name of a company or Internet service. The host name may include a period and a sub domain name. For **example :** sneezy@gromm.com

Rules for forming the e-mail address.

- ❖ Capitalization usually is not important in e-mail addresses.
- ❖ E-mail address do not have punctuation marks around them
- ❖ E-mail addresses do not have spaces in them
- ❖ Most e-mail programs allow you to type angle brackets (< >) around e-mail addresses.
- ❖ You can also precede an e-mail address with the person's name in quotes.

For **example:**

“Inventory details” <price@inventory.com>

1.4.3. Message Headers

Every e-mail message sent starts with headers-lines of text that tell you about the message. The headers are like the envelope for the message and include the addresses of the recipient and the sender.

Each header consists of the type of header, a colon, and the content of the header. For example, the header that shows who the message is addressed to consists of “To:” followed by one or more e-mail addresses. Headers that start with X are always optional headers, and many e-mail applications ignore them.

1.4.4. Downloading E-Mail

Your e-mail accumulates on a POP or IMAP server and your e-main application downloads the messages to your computer, so that you can read them. You have the following options:

- ❖ You can usually work either offline or online. If you have a dial-up connection to the Internet, you compose your messages before you dial in, queuing them
- ❖ You can choose either to leave downloaded messages on the server or to delete them from the server.

Working Offline

Most POP e-mail clients allow you to work offline. Working offline means that you read your e-mail by doing the following.

1. Connect to the Internet
2. Download your e-mail
3. Disconnect from the Internet
4. Read your e-mail, delete messages you don’t want to keep, compose replies, and write new messages
5. When you are ready, connect to the Internet
6. Send your new messages and download any new messages that may have arrived
7. Disconnect from the Internet

IMAP mail generally stays on the server and you use your e-mail client to read it there, so the process for working on IMAP e-mail offline is different.

Check the instructions for your e-mail to download folders from IMAP server to your local computer.

Deleting Messages from the Server

If a POP server stores your messages until you download them, your e-mail program usually deletes them from the POP server after downloading. Your e-mail application may have setting that enables you to choose whether to delete the e-mail from the server and, if so, when to delete it. Some ISPs limit the size of your mailbox.

1.4.5. Formatted E-Mail

Formatted document was to send it as attachment. If both your and recipient’s e-mail support it, you can sent formatted e-mail. Formatted e-mails are

- ❖ **HTML:** This is formatted with HTML tags. The HTML formatting can include text formatting, numbering, bullets, alignments, horizontal lines, backgrounds, hyperlinks, and HTML styles. HTML formatted e-mail is actually sent using MIME protocol.
- ❖ **Rich Text format:** This format can be read by most word processing applications. Rich text formatting can include text formatting, bullets, and alignments.
- ❖ **MIME (Multi purpose Internet Mail Extensions):** This is formatting created just for e-mail. MIME is also used for attachments. Formatting can include text formatting, pictures, video, and sound.

1.4.6. Attaching Files To Messages

By attaching files to e-mail, you can exchange documents for revision, pass on spreadsheets for data entry, or send a presentation for review. You can also attach electronic pictures, sounds, or movies-anything that can be put in file form.

1.4.7. Web Based E-Mail

Web-based e-mail provides both advantages and disadvantages. The main advantage is that if you can access the Web, you can read your e-mail. You don't have to be at your own computer to access your e-mail application. In addition, most web-based e-mail is free.

You can *read two kinds of messages* on the Web:

- **Messages sent to web-only account:** For example, the Yahoo Mail web site at **mail.yahoo.com** lets you sign up for a free e-mail mailbox, with a user name that you pick. Your address is username@yahoo.com. You can read messages sent to your Yahoo Mail address at the Yahoo Mail web site or with an e-mail application.
- **Messages stored in your POP mailbox:** Some web sites allow you to enter the name of your POP server, your user name, and your password. The site then retrieves the messages from your mailbox and displays them on web page, enabling you to read and respond to them.

1.4.8. Mail Away From Home

When you're away from your computer, you may still be able to read your e-mail, even if you don't regularly use a web-based e-mail service. You may be able to dial into your e-mail provider or use a web-based service.

If you use e-mail as part of your job and you travel on business, your e-mail administrator may have created a way for you to get your e-mail when you're out of the office.

Dialing or Telnetting In

Even if you usually download e-mail to your computer, you may also be able to telnet into the mail server to read your e-mail when you're not at your computer, but do have access to someone else's computer. When you telnet in, you connect to the Internet and then use a telnet program to connect to your company's computer over the Internet. If your company's computer allows you to telnet in, you may be able to use a UNIX mail program like Pine or Elm.

1.4.9. Avoiding Viruses

Viruses spread via e-mail are a real danger in today's Internet world. Although it's true that many e-mail virus warnings are hoaxes, viruses certainly exist, and some viruses can do significant damage. It is worth some cost and effort to avoid them.

The Simplest way to avoid viruses is to avoid using the e-mail programs that most viruses target: Microsoft Outlook and Outlook Express.

Viruses cannot be contained in a purely text-based message. A virus must be an executable file (usually with the extension .exe or .com) attached to a message- however, file extensions can be hidden. The biggest danger is opening an infected attachment; it's good policy to always use your virus checker before opening an attachment. Simply saving an infected file will not infect your computer, but it's safer still to permanently delete the message and the attachment.

Some antivirus software, such as Norton AntiVirus, McAfee VirusScan, and MailDefence, scans downloading e-mail for viruses. If e-mail is not automatically scanned, you may wish to manually scan attached files: save the file to your hard drive, right-click it, and choose to scan it with your virus checker. Some viruses infect files and can be cleaned by antivirus software, whereas other files need to be deleted.

Your computer can also be infected with a virus if your e-mail software allows scripts to run in w-mail messages. Viruses can also spread via macros in Microsoft Word or Microsoft Excel files.

1.4.10. Self Assessment Questions

Fill in the blank

1. The _______________ provides the Internet location of the mailbox.
2. Viruses infect files and can be cleaned by _____________software.

True / False

1. Outlook express is a one of the mail client.
2. Accessing web-based mails, we need not Internet connection.

Multiple Choices

1. MIME stands for

a) Multi instruction mail extensions b)Multi instruction and multi extensions

c) Multi purpose Internet Mail Extensions d) All of the above

Short Answer

1. What is the purpose of SMTP?

____

____

2. How mail servers receive and store e-mail messages in mailboxes?

____

____

E-Mail Security

1.4.11. Reasons To Secure Messages

All information sent out over the Internet is *distributed in packets*, or *small blocks* that are directed by your ISP's server, sent across the Internet, and reassembled by your recipient's server. These packages pass through many servers as they travel to their destination, and along the way, they may be detected by a packet sniffer, a program designed to identify certain groups of numbers or letters, such as credit card numbers or passwords.

Your e-mail can also be intercepted if you send or receive e-mail through a Web-based setup, because someone can attempt to guess your password and tamper with your account. And if you send e-mail by using your employer's e-mail programs and mail servers, all of your e-mail –sent and received – may be backed up and stored in the company's achieves, without your knowledge or approval.

1.4.12. Public Key Cryptography

To send e-mail that unauthorized people can’t read, you use encryption, which protects your information by encoding it-substituting letters and numbers with different characters, based on a code. With the right password, the recipient of your e-mail can decode the message and read it.

One number-your private key is stored on the head drive of your computer and is imprinted on every encrypted e-mail that you send. The public number-your public key-can be freely distributed, because only the two halves working together can decrypt your e-mail. In fact, many people who use secure e-mail routinely include the public part of their key pair in their signature files. This system of *paired keys* is called **public key cryptography.**

You can also use public key cryptography to digitally sign your messages, to prove that no one else could have sent them. By encrypting the message with your private key anyone can decrypt it by using your public key. If the recipient’s e-mail program can handle public key cryptography, it can inform the recipient that the sender definitely is you; otherwise, the digital signature just appears as an attachment.

Two standard systems of public key cryptography have emerged for use on the Internet: *digital certificates* and *PGP*. Both can be used to send and receive secure e-mail.

Digital Certificates

One system of public key cryptography uses a digital certificate to protect the information that you send, by attaching a secure signature. A digital certificate acts like a passport or a driver’s license; it authenticates your identity uniquely so that people who receive your e-mail know for sure that you’re the sender. You keep the private part of your digital certificate on your hard disk, and the public part is available in several ways: if someone sends you a signed message, you can save the public part of that person’s digital certificate, or you can sear h for a person’s digital ID on one of several Web sites. You keep the public parts of your correspondents’ digital certificates in your e-mail program’s address book, so that you can decrypt encrypted messages from those people.

Like passwords or driver’s licenses, digital certificates are issued by a trusted agency – a certificate authority. VeriSign is the most popular and recognizable vendor of digital certificates.

VerSign offers *several types of digital certificates*:

- Server IDs for secure Web servers
- Developer IDs for software developers
- Personal Digital IDs for secure Web servers
- Developer IDs for software developers and
- Personal Digital IDs for individuals and organizations who want to use secure e-mail.

PGP

PGP (Pretty Good Privacy) is another type of public key cryptography. PGP programs are available on the Internet that enables you to create your own key pairs. Many e-mail programs work with PGP, including Eudora and Outlook 97.

1.4.13. Using Cryptography With E-Mail

Both digital certificates and PGP can be used with Internet e-mail. When you send a message, you can choose to encrypt it or sign it. Before you send an encrypted message, you need to have the public key for the recipient. E-mail programs that can handle security usually also let you store the public keys of your correspondents in an address book. If the person to whom you want to send an encrypted file has a digital ID, you can assume that he or she also has an e-mail program that can handle the same type of security that your program uses.

Before you can send a signed message, you need to have your own private key installed in your e-mail program. You can send a signed message to anyone, because the signature doesn't affect the text of the message. If the recipient's e-mail program can't handle the type of security that you used, the signature just appears as an attachment at the end of the message.

1.4.14. Self Assessment Questions

Fill in the blank

1. Two standard systems of public key cryptography have emerged for use on the Internet are ___________________ and _________________.

True / False

1.All information sent out over the Internet is distributed in packets

Short Answer

1.Define cryptography.

____

____

1.5.Online Chatting And Conferencing Concepts

1.5.1. Forms Of Chat

In some types of chatting and conferencing, messages are sent immediately after they are complete. This type of communication is called *real-time communication*. Other ways of chatting deliver messages more slowly; for **example**, via e-mail. These other types of communication are also called *asynchronous*, because participants do not all read and respond to messages at the same time (synchronously).

❖ **Real-time chat**

Allows dialog to happen quickly, since each participant sees each message within seconds of when it is sent.

- **Asynchronous chat**

  Allow participants to consider their responses, gather information and formulate a response carefully. It also allows people from different time zones or with different schedules to participate. For **example**, there may not be a time when all the members of committee are available for a real-time meeting.

1.5.2. Messaging And Conference

E-mail messages are usually addressed and delivered to only one or two people. An e-mail mailing list allows messages to be distributed to a large list of people. Depending on how the m ailing list is set up, wither one, or all subscribers can post messages, so that mailing lists can be used to *distribute newsletters or press releases or to allow large group discussions.*

Usenet Newsgroups

Usenet is a system that allows messages to be distributed throughout the Internet. Because of the volume of messages, the messages are divided into newsgroups, or topics. You use a newsreader program to subscribe to a newsgroup, read the messages posted to that newsgroup, and post your own messages.

Internet Relay Chat (IRC)

Internet Relay Chat (IRC) allows thousands of Internet users to participant in real-time text-based chat. When you use an IRC program to connect to a central IRC server and join a conversation, you see all the messages that are typed in that channel within seconds of when the messages are sent. The IRC program enables you to type and send your own messages, too.

Web-Based Chat

Many Web sites now provide a Web-based way to send and receive IRC messages. Other Web sites provide their own real-time or asynchronous chat pages.

AOL Chat Rooms and other Proprietary Services

America Online users spend most of their online hours in char rooms, AOL services that allow real-time chat on a wide variety of subjects. To participate in AOL chat rooms, you must have an AOL account and use AOL's proprietary software to connect to your account. Other Internet users cannot participate in AOL chat rooms.

Similarly, CompuServe (a business-oriented online service now owned by AOL) offers forums and conferences on many different topics. You need a CompuServe account and CompuServe software to participate.

Direct Chat Systems

ICQ (Pronounced "I seek you"), AOL Instant Messenger, and *other systems* enable you to send messages to other people when both you and they are connected to the Internet. You create a list of the people who you want to chat with. When one of the people on your list connects to the Internet, your direct chat program informs you that your friends are online, and you can then exchange messages.

Online Conferencing

If text isn't enough, you can use one of several Internet-based *online conferencing programs* that enable you to confer via text, voice, and video with one or more other people. To use one of these programs, your computer requires a microphone, speakers, and a video camera. Some conferencing programs also allow all the participants to see or edit a document on their screens and to see or write on digital whiteboard.

MUDs and MOOs

In addition to unstructured chats and discussions, many multi-user games are in progress on the Internet at any hour of the day or night. Multi-user dimensions (MUDs) are text-based chats in which the participants play a game by following a set of rules enforced by the central server computer. The game is usually a fantasy game, but may be an online university or other group event. MUDs object oriented (MOOs) are user-programmable games that are similar to MUDs: by programming, participants can create objects in the shared world of the MOO.

1.5.3. How The Chat Work

Identifying yourself

In mailing lists and newsgroups, you are identified by your name and e-mail address. When you join an IRC channel, Web-based chat, or AOL chat room, you choose a name to go by-variously referred to as your nickname, handle, or screen name. If someone is already using the name that you planned to use, you must choose another one. On systems that allow you to choose a nickname each time that you join, remember that the person who has a particular nickname today may no be the same person who had it yesterday.

Topics, Newsgroups, Channels, and Rooms

Tens of thousands of people can simultaneously participate in e-mail mailing lists, Usenet newsgroups, Internet Relay Chat, Web-based chat, and AOL chat rooms. The discussions are categorized by topic, enabling people who are interested in a particular topic to communicate with each other. Topics may include hobbies, personal problems, sports, research areas, religious beliefs, or other areas of interest, or chat participants may simply be grouped together by geographical area or age. Some discussions consist entirely of people looking for partners for romance, sex, or simple banter.

Depending on the system, topic groups may be called newsgroups (in Usenet), channels (in IRC), or rooms (in AOL). E-mail mailing lists are already divided by topic (one list per topic), although some mailing lists ask you to include topic keywords in the subject lines of your messages, too.

Following the Discussion

In IRC channels and AOL chat rooms, the discussion consists of short messages from many participants, with each message preceded by the name of the person who sent it. The messages are displayed on your screen in the order in which your computer received them, and several conversations may be happening at the same time.

Following a chat discussion can be tricky. When you join a channel or chat room, stay quiet for a few minutes until your screen fills up with messages. Start by reading one interesting-looking message and then read down through the messages for responses to that message and for other messages from the same person who sent the original message. When you have something to say, jump in.

1.5.4. Self Assessment Questions

Fill in the blank

1. _________ is a system that allows messages to be distributed throughout the Internet.

2. _____________________ programs that enable you to confer via text, voice, and video with one or more other people.

True / False

1. Asynchronous communication allows the participants to read and respond to messages at the same time .

2. The users of other Internet cannot participate in AOL chat rooms

Multiple Choices

1. What is the expansion for IRC?

a) Intranet Relay Chat	b) Internet Receive Chat
c) Internet Relay Chat	d) none of the above

Short Answer

1. What are the types of chatting?

____

____

2. How can you identify yourself in internet?

____

____

1.6.WWW Concepts

1.6.1. Elements Of The Web

Clients and Servers

A Web server is a computer connected to the Internet that runs a program that takes responsibility for storing, retrieving, and distributing some of the Web's files. A Web or Web browser is a computer that requests files from the Web. When a client computer wants access to one of the files on the Web, the network directs the requests to the Web server that is responsible for that file. The server then retrieves the file from its storage media and sends it to the client computer that requested it.

The Web's Languages and Protocols

When a client computer requests a file from the Web, it can assume very little about the server that stores the file, or the various other computers that might handle the file as it is transmitted from the server to the client. For such a system to work, it must have a well-defined set of languages and protocols that are independent of the hardware or operating systems on which they run.

URLs and Transfer Protocols

Each file on the Internet has an address, called a Uniform Resource Locator (URL). For example, the URL of the ESPN Sportzone Web site is http://espnet.sportzone.com

The first part of a URL specifies the transfer protocol, the method that a computer uses to access this file. Most Web pages are accessed with the Hypertext Transfer Protocol (the language of Web communication), which is why Web addresses typically begin with http (or its sucure version, https or shttp). The http:// at the beginning of a Web page's URL is so common that it often goes without saying; if you simply type espnet.sportzone.com into the address window of Internet Explorer or Navigator, the browser fills in the http:// for itself. In common usage, the http:// at the beginning of a URL often is left out.

The next part of the address denotes the host name of the Web server. The URL doesn't tell you where the Web server is actually located. The domain name system routes your Web page request to the Web server regardless of its physical location. As users, you don't need to deal with this level of detail, and that's a good thing.

Some URLs contain information following the host name of the Web server. This information specifies exactly which file you want to see, and what directory it is stored in. If the directory name and filename aren't specified, you get the default Web page for that Web server.

HTML

The **Hyper Text Markup Language (HTML)** is the universal language of the Web. It is a language that you use to *lay out pages that are capable of displaying all the diverse kinds of information that the Web contains.*

While various software companies own and sell HTML reading and HTML-writing programs, no one owns the language HTML itself. The World Wide Web Consortium (W3C), at http://www.w3c.org, manages the HTML standard.

Java and JavaScript

Java is a language for *sending small applications (called applets) over the Web*, so that they can be executed by your computer.

JavaScript is a language for extending HTML to embed small programs called *scripts in Web pages*. The main purpose of applets and scripts is to speed up the interactivity of Web pages; you interact with an applet or script that the Web server runs on your computer, instead of interacting with a distant Web server. Java and JavaScript are also used for animation; the Web server sends an animation constructing applet or script that runs on your computer, instead of transmitting the frames of an animation over the Internet. Typically, this process is invisible to the user-the interaction or the animation just happens, without calling your attention to how it happens.

VBScript and ActiveX Controls

VBScript and ActiveX Controls are Microsoft systems (not Web standards) that work with Internet Explorer.

ActiveX controls (AXCs), like Java, are used to embed executable programs into a Web Page. When Internet Explorer encounters a Web page that uses ActiveX controls, it checks whether that particular control is already installed on your computer, and if it isn't, IE installs it.

XML and other Advanced Web Languages

The Extensible Markup Language (XML) is a very powerful language that may replace HTML as the language of the Web. Currently, XML is little more than a specification at the W3C, but it is expected to be implement in the fifth-generation browsers.

XML is a language for writing languages (such as HTML), which is what makes it so powerful. XML gives document designers a greatly increased capability to attach explanatory tags to data.

The W3C is working on two style sheet language specifications: the Extensible Style Language (XSL) and Cascading Style Sheets (CSS). Another extension of HTML is Dynamic HTML (DHTML), which consists of three components: HTML, JavaScript, and cascading style sheets (CSS).

Image Formats

Pictures, drawings, charts, and diagrams are available on the Web in a variety of formats. The most popular formats for displaying graphical information are JPEG and GIF.

Audio and Video Formats

Some files on the Web represent audio or video, and they can be played by browser plug-ins. Web audio and video come in ***two flavors***:

- ❖ Your browser can download whether the entire file and play it (which can take a long time, because audio files are large and video files are huge) or
- ❖ Only the part of the file that it needs to play next, discarding the parts that it has played already. The second technique is called streaming audio or streaming video.

VRML

The **Virtual Reality Modeling Language** (VRML) is the Web's way of describing three-dimensional scenes and objects. Given a VRML file, a browser can display a scene or object as it would appear from any particular viewing location. You can rotate an object or move through a scene, using the controls that a browser provides.

Like **Java**, VRML moves some computational burden from the network to your computer. Rather than having Web servers store and transmit all the possible 2-D views of a scene, the scene is described in VRML and downloaded to your machine. Your computer then figures out what you would see if you stand in a particular place and look in a particular direction. Given the speed of most Internet connections, computing a view on your machine is much faster than downloading one from a Web server.

Like **HTML pages**, VRML scenes can contain many kinds of information. A scene of a city square, for example, might contain a kiosk, and each face of the kiosk might display a different picture or text document. The objects in a scent might also be links to URLs, which are accessed when you click the object. The University of Essex, for example, has posted a 3-D campus model on the Web at the following site:

http://esewww.essex.ac.uk/campus-model.wrl

Web Pages and Web Sites

A Web page is an HTML document that is stored on a Web server and that has a URL so that it can be accessed via the Web.

A Web site is a collection of Web pages belonging to a particular person or organization. Typically, the URLs of these pages share a common prefix, which is the address of the home page of the site.

Special Kinds of Web Sites and Pages

Portals

A **portal** is a Web site that wants to be your start page, the page that your browser displays first.

Web Guide

A **Web guide** is Web site with a system of categories and subcategories that organizes links to Web pages.

Search Engines

You give a **search engines** search only the titles of Web pages, while others search every word. Some allow more complicated queries than others. Keywords can be combined with Boolean (logical) operations, such as AND, OR, and NOT, to produce rather complicated queries.

Home Pages

A **home page** is the front door of a Web site.

Personal Home Pages

A **personal home page** is the front door of a Web site that an individual puts on the Web to introduce himself or herself, to share interests with others, and to keep distant friends and acquaintances up-to-date on the course of life.

Business and Organization Home pages

A **business home page** is the front door to a business's Web site.

1.6.2. Web Browsers

A **Web browser** is a program that your computer runs to communicate with Web servers on the Internet, which enables it to download and display the Web pages that you request. Because a Web browser has the ability to interpret or display so many types of files, you often may use a Web browser even when you aren't connected to the Internet. The most *popular browsers* are

- ❖ Netscape Navigator and
- ❖ Microsoft Internet Explorer.

Both Navigator and IE are available over the Internet at *no charge*. Microsoft designed IE for the Windows operating system, but it is now available for Macintosh and some UNIX systems, as well. Navigator is available for Windows, Macintosh, UNIX, and Linux operating systems.

Browser Concepts

The Default Browser

Graphical user interfaces (GUIs) such as Windows, MacOS, and various UNIX desktop applications enable you to open a file by clicking or double-clicking an icon that represents the file. When the icon represents a Web page, a local HTML file, or even an image file in a format such as JPEG that Web browsers display well, the operating system runs a Web browser to display the file.

Browser Home Pages and Start Pages

Your browser's start page is the Web page that the browser loads when you open the browser without requesting a specific page. Internet Explorer automatically starts with the browser home page, but Navigator allows the start page to be different from the browser home page.

Plug-Ins

Plug-ins are *programs* that are independent of your Web browser, but that "plug in" to it in a seamless way, so that you may not even be aware that you are using a different piece of software.

Various plug-ins (such as RealAudio for receiving streaming audio, or QuickTime for downloading video) have become standard accessories for IE or Navigator, and are installed automatically when you install the Web browser. To install other plug0ins, download them from the Internet and then follow the directions that come with the plug-in.

Elements of a Browser Window

Most browser windows have the same basic layout. From top to bottom, you find these *basic elements*:

- ❖ Menu bar
- ❖ Toolbars
- ❖ Address or Location window
- ❖ Viewing window
- ❖ Status bar

Some Web pages are divided into *independent panes*, called **frames**. When such a Web page is viewed, the viewing window is similarly divided into independent panes. You can scroll up or down in a frame or even move from link to link, without disturbing the contents of the other frames.

Viewing Pages with a Browser

The purpose of a Web browser is to display Web pages, which may either arrive over the Internet or already be on your computer system.

Viewing Pages on Your Local Drives

You can use your Web browser to view files of any common Web format (HTML, SRML, JPEG, and so on) that are stored on your hard drive or elsewhere on your system. In Windows, Macintosh, and some UNIX desktops, simply clicking or double-clicking the file icon opens the file in the default Web browser.

Viewing Pages on the Web

- ❖ Enter its URL into the Address or Location box of a Web browser

  You can type in the URL or cut-and-paste it. Both IE and Navigator have an auto-complete feature-the browser tries to guess what URL you are typing and finishes it for you, by guessing similar URLs that you've visited before.

- ❖ Select it from the list that drops down from the Address or Location box

  Both IE and Navigator remember the last 25 URLs that you have typed into the Address or Location box.

- ❖ Link to it from another Web page

  The reason it's called a "Web" is that pages are linked to each other in a tangled, unpredictable way. Click a link (usually an icon or underlined, blue text) to see the Web page it refers to.

- ❖ Link to it from mail message or newsgroup article

  Many e-mail and news reading programs are able to notice when a URL appears in a mail message or newsgroup article.

- ❖ Select it from the Bookmarks list, the Favorites menu (in IE), the History folder, or open an Internet shortcut.

1.6.3. Security And Privacy Issues

Interacting with the Web is a little like *passing note across a classroom*. You can't know ahead of time what computers are going to handle your messages as they pass between **you** and a **Web server**, and you can't be sure that none of those intermediate computers will copy the message, or let someone else read it.

Each new generation Web browsers introduces new features to the Web. Some of these features, such as *firewalls* or *the Secure Socket Layer (SSL) protocol*, make your Web interactions safer. Others, such as cookies or scripting languages, open new opportunities for mischief in addition to their beneficial uses.

Cookies

A cookie is a *small (at most 4K) file that a Web server can store on your machine*. Its purpose is to allow a Web server to personalize a Web page, depending on whether you have been to that Web site before, and what you may have told it during previous sessions.

For example, when you establish an account with an online retailer or subscribe to an online magazine, you may be asked to fill out a form that includes some information about yourself and your preferences.

The Web server may store that information in a cookie on your machine. When you return to that Web site in the future, the retailer's Web server can read its cookie, recall this information, and structure its Web pages accordingly.

Firewalls

A firewall is a *piece of hardware or software that sits between two networks for security purposes.*

Typically, an organization might have its own computers linked in an Internet-like network called an **intranet**. A **firewall** is places between this *intranet* and *the Internet* to prevent *unauthorized users* from gaining access to all the resources of the intranet. If you communicate with the Internet through a firewall, you must configure your Web browse to request Web pages from the firewall's proxy server, the program that filters packets of information between the intranet and the Internet.

Secure Communications and Transactions

Transactions and communications on the Internet involve more than just your computer and the server of the Web site that you are dealing with. Each message back and forth goes through several other computers along the way, and you can't even predict which computers will be involved. The following are the three major risks involved in any Internet transaction:

- **Eavesdropping:**

  Any information that you transmit may be overheard by other computers- your credit card number.

- **Manipulation:**

  The information that you send or receive may be altered by third parties. For **example**, the delivery address for your shipment might be altered.

- **Impersonation:**

  You might not be dealing with the entity that you think you are dealing with. Or, conversely, whoever you are dealing with might gather enough information to impersonate you in another transaction.

Servers and Web browsers use complex encryption techniques to guard against these threats. When a server and your browser are acting in a secure mode, the messages transmitted appear to be gibberish to anyone but the designated receiver, tests are done to detect altered messages, and identities are verified.

How Secure Transactions Work

Secure Web transactions use a protocol called *the Secure Sockets Layer* (SSL). This protocol depends on public key cryptography to establish proofs of identity, called digital certificates.

Public Key Cryptography

Web transactions are encrypted using public key cryptography, a system in which pairs of very large numbers are used to encode and decode messages. One number of the pair, called the public key, is published; and the second number, the private key, is kept secret. When one of the two numbers has been used to encode a message, the other one is needed to decode it.

Digital Certificates

A digital certificate (or certificate or digital ID) is a file that identifies a person or organization.

Secure Sockets Layer

The Secure Sockets Layer (SSL) protocol is a method for secure communications and transactions to take place over the Internet. SSL uses digital certificates, to verify that server is what it claims to be. The server and your browser then send encrypted messages back and forth until your transaction is complete.

Executing a Secure Transaction

You need only keep three things in mind

- ❖ Don’t enter any sensitive information into a form until a secure connection has been established.
- ❖ Pay attention to any warning messages that your browser gives you.
- ❖ Remember your common sense.

Executable Applets and Scripts

Java, JavaScript, VBScript, and ActiveX controls are all languages for Web servers to run applications on your computer. These programming systems have security safeguards, but occasionally, bugs are found either in a programming language or in the way that is implemented by a particular browser or on a particular machine. These bugs involve some security risk.

Netscape and Microsoft have a strong interest in reacting quickly to fix the security holes that people discover in their browsers.

With the authorship of the control verified, you are left with the decision: Do I trust the author or not? If you say yes, the control can run on your computer, and you may not be able to tell what it’s doing. Worse, you are then asked whether to trust code from this author automatically in the future. If you say yes to this, ActiveX controls from that author will install and run without you being made aware that anything has happened.

Viruses

The most risky thing that you can do on the Web is to download an executable file from someone who you don’t know and then run it on your computer. This is no different from letting a stranger put a diskette into your disk drive.

Check regularly for patches, warnings, and new versions at the Web site of the company that wrote your Web browser. Viruses come in through security holes, and many people are working to find and patch those holes before they can be exploited.

Recognize, though, that a small risk of a virus infecting your computer exists, no matter how careful you are. Everyone should have a virus-checking program, keep it up-to-date, and run it regularly.

Privacy Implications of Browser Catches and History

As you browse the Web, your browser may be storing in a cache (temporary storage area) the Web pages that you visit, and may be making a list of the Web sites that you have visited. Browsers do so with the best of *intentions:* cached Web pages can be reloaded quickly when you hit the Back button, and History files enable you easily to find a Web site that you remember looking at last week.

Adjusting Your Security Settings

Navigator and Internet Explorer give you broad latitude about how strict a security policy you want to set. If your policy is too strict, you miss out on some of the cool features of the Web. If it is too loose, you can be defrauded or introduce viruses into your computer system. If you choose too many “ask me at the time” options, your browsing session is constantly interrupted with questions whose implications you may not fully grasp.

Anonymous Web Browsing Through Proxies

When you use a proxy, you send your Web page requests to the proxy, and it makes the request in its own name. To the server whose pages you are requesting, you appear to be at the proxy’s IP address, not at your own.

Two Web sites where you can arrange to work with a proxy are these:

http://www.anonymizer.com

http://www.iproxy.com

Each site offers a no-frills free anonymous browsing service.

1.6.4. Self Assessment Questions

Fill in the blank

1. The popular image formats for displaying graphical information are _____________ and ________.

2. The _____________ protocol is a method for secure communications and transactions to take place over the Internet.

True / False

1. The combination of applets and scripts is to speed up the interactivity of Web pages.

Short Answer

1. List out any two element of web.

----

2. What is the use of digital certificates?

----

1. 8. Summary

Basically in this unit covered by the Internet connection concepts. We have discussed about how to implement internet connection concepts , how to secure the messages in e-mail, how to use major elements of public key cryptography , reasons to usage of chatting and on line conferencing and what is world wide web.

Unit Questions

2. Explain about the internet communication protocol.
3. What are all the types of internet connection? Explain.
4. How Do You Get Your E-Mail? Explain.
5. List out the different formatted e-mail. Explain.
6. How public key cryptography is useful in Internet? Explain.
7. Describe about Internet Service Providers.
8. What is the use of web browsers? Explain with examples.
9. List out the element of web. Explain.
10. How The Chat Work?
11. Explain security and privacy issues in internet.

Answer for Self Assessment Questions

Answer 1.3.5

Fill in the blank

1. host computer or host 2. virus

True / False

1. True 2. True

Multiple Choices

1. b 2. c

Short Answer

1. Internet is the world largest computer network, the network of networks that connects computers all over the world.

2. Computers that provide services for other computers to use are called servers. Programs that ask servers for services are called clients.

3. A firewall is a program that controls what information passes from one network to another.

Answer 1.4.10

Fill in the blank

1. host name 2. Antivirus

True / False

1. True 2. False

Multiple Choice

1. c

Short Answer

1. Mail servers receive and store e-mail messages in mailboxes by using a protocol called Post Office Protocol or Internet Message Access Protocol

2. SMTP is a protocol; it is used to pass the E-mail messages through the Internet

Answer 1.5.4

Fill in the blank

1. digital certificates and PGP

True / False

1. True

Short Answer

1.Cryptography protects your information by encoding it-substituting letters and numbers with different characters, based on a code.

Answer 1.6.4

Fill in the blank

1. Usenet		2. online conferencing

True / False

1. False		2. True

Multiple Choice

1. c

Short Answer

1. Real time chat and asynchronous chat

2. In mailing lists and newsgroups, you are identified by your name and e-mail address.

Answer 1.7.4

Fill in the blank

1. JPEG and GIF		2. Secure Sockets Layer (SSL)

True / False

1. True

Short Answer

1. Clients & servers and Web's Languages & protocol

2. A digital certificate is a file that identifies a person or organization.

NOTES

UNIT – II

2.1 Introduction

This section provides you with *fundamental concepts of object-oriented programming and the elements of the Java programming language* that will be used throughout this chapter. It also deals with process of the various types of *constants, variables and its data types*; basic structure of the *operators, expressions,* decision making with *branching and looping* in java programming. It assures you to understand general programming concepts.

2.2 Objectives

After studying this lesson, you should be able to:

- ❖ Understand about concepts of object-oriented programming.
- ❖ Describe the various features of Java programming language.
- ❖ Describe the various types of constants, variables and data types.
- ❖ Understand about operators and expression concepts and its classifications.
- ❖ Understand about, how to making the decision with branching and looping statement

2.3 Fundamentals Of Object Oriented Programming

2.3.1 Introduction

The invention of the computer, many programming techniques have been tried such as *modular programming, top-down programming, bottom-up programming and structured programming*. The objectives of these techniques to handle the increasing complexity of programs are reliable and maintainable.

Structured programming, like C became very popular and was the paradigm of the 1980s. The structured approach failed to show the desired results in terms of bug-free, easy-to-maintain and reusable programs.

Object-Oriented programming is an approach to program organization and development, which attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several new concepts.

C++ is a *procedural language with object-oriented extension*, but Java, a pure *object oriented language*.

2.3.2 Object Oriented Paradigm

The objective of object-oriented approach is to *eliminate some of the flaw encountered* in the procedural approach. OOP allows us to *decompose a problem* into a number of *entities* called **objects** and then *build data and functions* around these entities. The organization of data and methods in object as shown in the fig. 2.3.1.

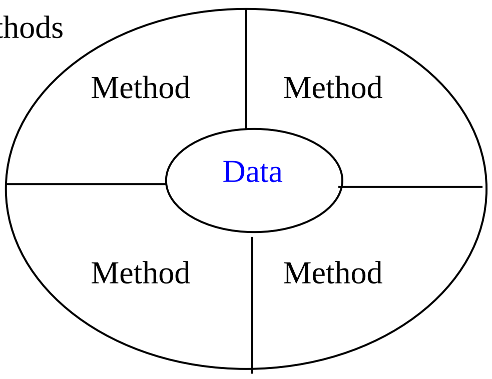


Fig.2.3.1 Object = Data + Methods

The data of an object can be accessed by methods associated with object and method of one object can be access the methods of other object. *Features of object-oriented paradigm* are:

- ❖ Emphasis is on data rather than procedure.
- ❖ Programs are divided into objects.
- ❖ Data structures are designed.
- ❖ Methods that operate on the data of an object are tied together in the data structure.
- ❖ Data hidden and cannot be accessed by external functions.
- ❖ Object may communicate with each other through methods
- ❖ New data and method can
- ❖ Employs bottom-up approach in program design.

Object-oriented programming is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and function that can be used as templates for creating copies of such modules on demand.

2.3.3 Basic Concepts

The basic concepts of OOPS, which form the heart of Java language as follow as:

1.Objects

Objects are basic runtime entities in an object-oriented system. They may represent person, a place, a table of data or any data item that the program must handle.

They may also represent user-defined data types such as vectors and lists. Objects take up space in the memory and have an associated address like a structure in C.

When a program is executed, the object interacts by sending messages to one another. **For example** ‘customer’ and ’account’ are two objects in a banking system, then the customer object send a message to the account object requesting for balance. Each object has data and code to manipulate the data. Fig. 2.3.2 shows a notation to represent an object.

OBJECT: Student
DATA: Name Reg.No. Marks
METHODS: Total Average

Fig.2.3.2 Representation of an object

2. Classes

The entire set of data and code of an object can be made a user defined data type with the help of a **class.** In fact, objects are variables of type class. Once a class has been defined, we can create any number of objects based on that class. A class has collection of objects of similar type**. For example** circle, square, rectangle and ellipse are members of the class shape.

3. Data Abstraction and Encapsulation

The wrapping up of data and methods into single unit is called **encapsulation.** The data is not accessible to the outside world and only those methods are wrapped in the class can access it. These methods provide the interface between object's data and the program. This insulation of the data from direct access by the program is called **data hiding**.

Abstraction refers to representing features without the background details. Classes use the concept of abstraction and are defined as a list of abstract attribute such as size, weight and cost, and methods that operate on these attributes.

4. Inheritance

Inheritance is the process by which objects of one class acquire the properties of objects of another class. **For example**, the **B.Sc., (Computer Science)** is a part of the class **Department Computer science,** which is again a part of the class **College**. Inheritance properties are as shown in Fig.2.3.3. In OOP, the concept of inheritance provides the idea of reusability. We can add additional features to an exiting class without modifying it.

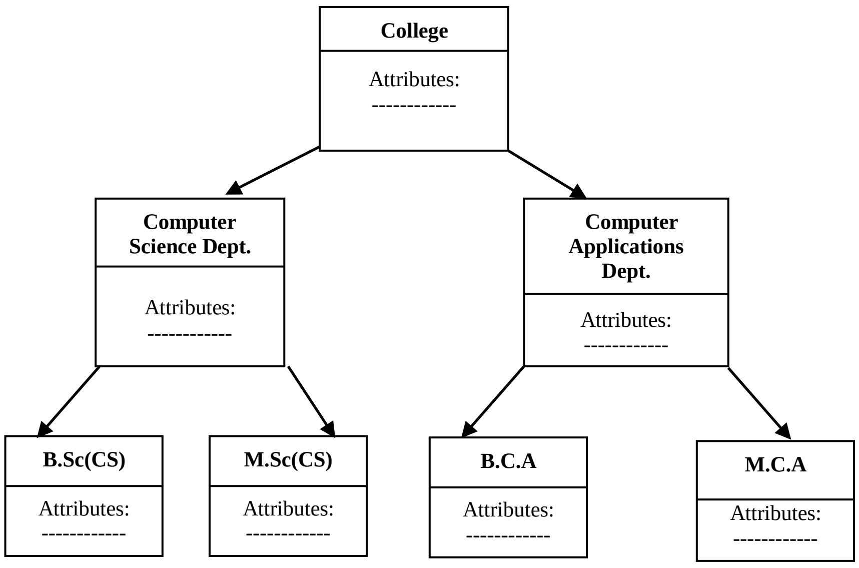


Fig.2.3.3 Property Inheritance

5.Polymorphism

Polymorphism is the ability to take more than one form. An operation may exhibit different behavior in different instances. The behavior depends upon the data types of data used in the operation. **For example**, consider the operation of addition with *two numbers* will generate *summation* and with *two strings* would produce a third string by *concatenation*. Fig. 2.3.4 that shows the single method name can be used to handle different number and different types of arguments.

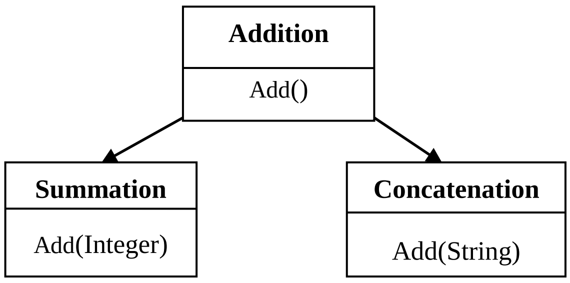


Fig.2.3.4 Polymorphism

6.Dynamic Binding

Binding refers to linking of a procedure call to the code executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at runtime.

7. Message communication

An Object-oriented program consists of set objects that communicate with each other. The process of programming in an object-oriented language, involves the following **basic steps**:

- ❖ Creating classes that define objects and behavior.
- ❖ Creating objects from class definitions.

- Establishing communication among object.

Objects communicate with one another by sending and receiving information as shown in Fig.2.3.5. A message for an object is a request for execution of a procedure, and will invoke a method in the receiving object that generates desired result, as shown in Fig.2.3.6

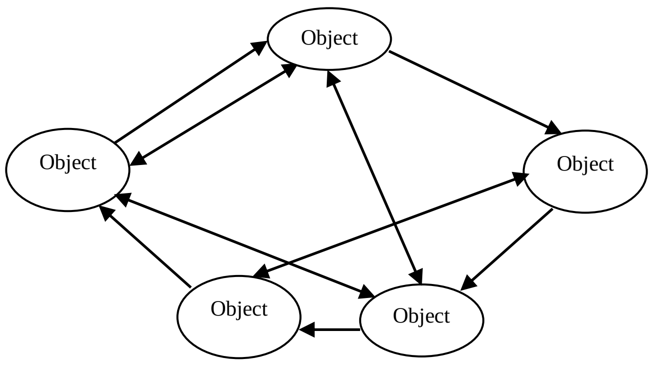


Fig.2.3.5 Network of objects communicating between them

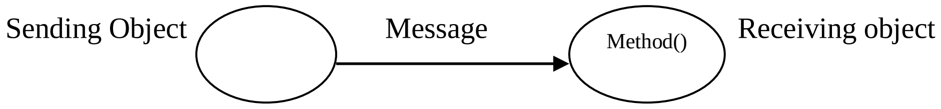


Fig.2.3.6 Message triggers a method

Message passing involves specifying name of the object, the name of the method (message) and information to be sent.

Example:

Student.averageMarks(Reg.No)

2.3.4 Benefits Of OOP

OOP offers several benefits to both the program designer and the user.

- Through inheritance, we can eliminate redundant code and extend the use exiting classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- It is possible to have multiple objects to coexist without any interference.
- It is possible to map objects in the problem domain to those objects in the program
- It is easy to partition the work in a project based on objects.
- The data-centered design approach enables us to capture more details of a model in an implementable form

- ❖ Object-oriented systems can be easily upgraded from small to large systems.
- ❖ Message passing techniques for communication between objects make the interface description with external systems much simpler.
- ❖ Software complexity can be easily managed.

2.3.5 Applications Of OOP

The promising areas for application of OOP includes:

- ❖ Real-time systems
- ❖ Simulation and modeling
- ❖ Object-oriented databases
- ❖ Hypertext, hypermedia and expertext
- ❖ AI and expert systems
- ❖ Neural networks and parallel programming
- ❖ Decision support and office automation systems
- ❖ CIM/CAD/CAM System

2.3.6. Self Assessment Questions

Fill in the Blank

1. The wrapping up of data and methods into a single unit is known as _______________.

2. Programs are divided into what are known as________________.

True / False

1. Objects are not the basic runtime entities in an object-oriented system.

2. Data is hidden and cannot be accessed by external functions.

Multiple Choices

1. Dynamic binding is happened at

a) Compile time	b) Run time
c) Both time	d) none of the above

Short Answer

1. Define inheritance.

____

____

2. List some applications of OOP.

____

____

2.4 Java Evolution

2.4.1 Java History

Java is a general-purpose, object-oriented programming language developed by **Sun Microsystems of USA in 1991**. Originally called **Oak** by

James Gosling. Java was designed for the development of software for consumer electronic devices like TVs, VCR and Other electronic machines.

The goal had a strong impact on the development team to make the language *simple, portable and reliable*. Table 2.4.1 shows important milestones in the development of Java.

Table 2.4.1 Milestones in the development of Java

JAVA MILE STONES	
Year	**Development**
1990	Sun Microsystems decided to develop special software that could be used to manipulate consumer electronic devices. A term of Sun Microsystems programmers headed by James Gostling was formed to undertake this task.
1991	After exploring the possibility of using the most popular object-oriented language C++, the team announced a new language named "Oak".
1992	The team, known as Green Project team by Sun, demonstrated the application of their new language to control a list of home appliances using a hand-held device with a tiny touch-sensitive screen.
1993	The World Wide Web (WWW) appeared on the Internet and transformed the text-based Internet into a graphical-rich environment. The Green Project team came up with the idea of developing Web applets (tiny programs) using the new language that could run on all types of computers connected to Internet.
1994	The team developed a Web browser called "HotJave" to locate and run applet programs on Internet. HotJava demonstrated the power of the new language, thus making it instantly popular among the Internet users.
1995	Oak was renamed "Java", due to some legal snags. Java is just a name and is not an acronym. Many popular companies including Netscape and Microsoft announced their support to Java.
1996	Java established itself not only as leader for Internet programming but also as a general-purpose object-oriented programming language. Sun releases Java Development Kit 1.0
1997	Sun releases Java Development Kit 1.1 (JDK 1.1)
1998	Sun releases Java 2 with version 1.2 of the Software Development Kit (SDK 1.2)
1999	Sun releases Java 2 Platform, Standard Edition (J2SE) and Enterprise Edition (J2EE)

2000	J2SE with SDK 1.3 was released
2002	J2SE with SDK 1.4 was released
2004	J2SE with JDK 5.0 (Instead of JDK 1.5) was released. This is known as J2SE 5.0

2.4.2 Java Features

Sun Microsystems describes Java with the following attributes:

- ❖ Complied and Interpreted
- ❖ Platform-Independent and Portable
- ❖ Object-Oriented
- ❖ Robust and Secure
- ❖ Distributed
- ❖ Familiar, Simple and Small
- ❖ Multithreaded and Interactive
- ❖ High Performance
- ❖ Dynamic and Extensible

These features have made Java the first application language of the World Wide Web.

Complied and Interpreted

A computer language is either **compiled** or **interpreted**. Java combines both these approaches thus making Java a two-stage system. **First**, *Java compiler* translates source code into byte code instruction. Byte codes are not machine instruction and in the **next** stage *Java interpreter* generates machine code that can be directly executed by the machine that is running the Java program.

Platform-Independent and Portable

Java programs can be easily moved from one computer to another, **anywhere and anytime.** Changes and upgrades in operating system, processors and system resources *will not force any changes in Java programs*.

Java ensures portability in *two ways*.

- ❖ Java compiler generates byte code that can be implemented on any machine.
- ❖ The sizes of the primitive data types are machine-independent.

Object-Oriented

Java is a **pure** object-oriented language. All program code and data reside within objects and classes and they are arranged in packages, that we use in our programs by inheritance. The object model in Java is simple and easy to extend.

Robust and Secure

Java provides many safeguards to ensure reliable code. It has strict compile and run time checking for data types. Java also has the concepts of exception handling, which captures errors and eliminates any risk of crashing the system.

Security is an important issue for a language that is used for programming on Internet. The absence of pointer in Java ensures that programs cannot access to memory locations without proper authorization.

Distributed

Java is designed as a distributed language for creating applications on networks. Java applications can open and access objects on Internet as easily as they can do in a local system. So multiple programmers at multiple remote locations to collaborate and work together on single project.

Familiar, Simple and Small

Java does not use pointers, preprocessor header files etc. Java eliminates operator overloading, multiple inheritance. So it is considered as a simple and small language. To make language look familiar to the exiting programmers, it modeled on C and C++ language.

Multithreaded and Interactive

Java handling multiple tasks simultaneously is called **multithreaded**. We need not wait for the application to finish one task before beginning the other. **For example** we can listen music while scrolling a page and at the same time download an applet from a distinct computer.

High Performance

Java performance is impressive for an interpreted language, mainly due to the use of intermediate byte code. Java architecture designed to reduce overheads during runtime and incorporating multithreading enhances the overall execution speed of Java programs.

Dynamic and Extensible

Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe and extensible manner.

2.4.3 How Java Differs From C And C++

Java is a lot like C and C++ but the major difference between Java with C and C++. Java is a pure object-oriented language. Java also adds some new features. C and C++ features that were omitted from Java are:

Java and C

- ❖ Java does not include the C unique statement keywords **sizeof**, and **typeof**
- ❖ Java does not contain the data types **struct** and **union**

- Java does not define the type modifiers keywords **auto, extern, register, signed**, and **unsigned**
- Java does not support and explicit pointer type
- Java does not have a preprocessor and therefore we cannot use **#define, #include**, and **# ifdef** statements
- Java requires that the functions with no arguments must be declared with empty parenthesis and not with the **void** keyword as done in C
- Java adds new operators such as **instanceof** and >>>
- Java adds labeled **break** and **continue** statements
- Java adds many features required for object-oriented programming

Java and C++

- Java does not support operator overloading
- Java does not have template classes as in C++
- Java does not support multiple inheritances of classes. This is accomplished using a new features called "interface"
- Java does not support global variables. Every variables and method is declared within a class and forms part of that class
- Java does not use pointers
- Java has replaced the destructor function with a finalize () function
- There are no header files in Java

2.4.4 Java And Internet

Java is associated with the Internet because of the fact that the first application program written in Java was HotJava, a Web browser to run applets on Internet. Internet users can use Java to create applet programs and run them locally using a "*Java-enabled browser*". Download an applet located on Internet and run it on local computer using Java-enabled browser as shown in Fig.2.4.1.

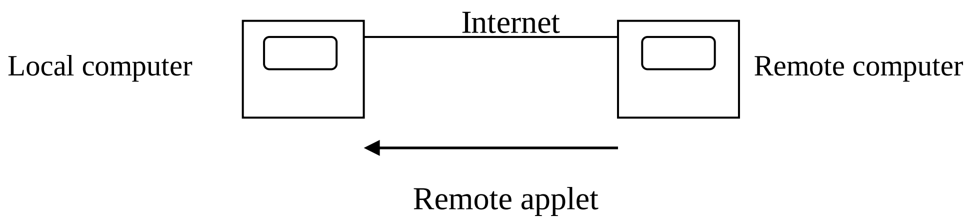


Fig. 2.4.1 Downloading of applets via Internet

2.4.5 Java And World Wide Web

World Wide Web (WWW) is an open-ended information retrieval system designed to be used in the Internet' environment. The system contains Web pages that provide both information and controls. The Web system is open-ended and we can navigate to a new document in any direction as shown in Fig.2.4.2. Web pages contain HTML tags that enable us to find, retrieve, manipulate and display documents worldwide.

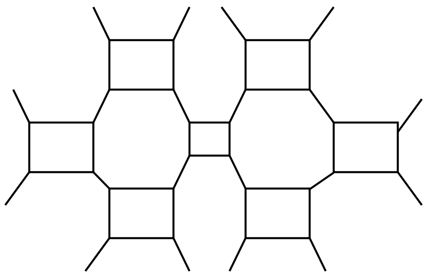


Fig.2.4.2 Web structure of information search

Java communicates with Web pages through a tag called <APPLET>. Fig.2.4.3 shows this process with the following steps:

1. The user sends request for HTML document to remote computer' Web Server. The Web Server is a program that accepts request, processes the request, and sends the required document.

2. The HTML document is returned to the user's browser. The document contains the APPLET tag, which identifies the applet.

3. The applet byte code is transferred to the user's computer.

4. The Java-enabled browser on the user's computer interprets the byte codes and provides output.

5. The user may have further interaction with the applet but with no further downloading from the provider's Web server.

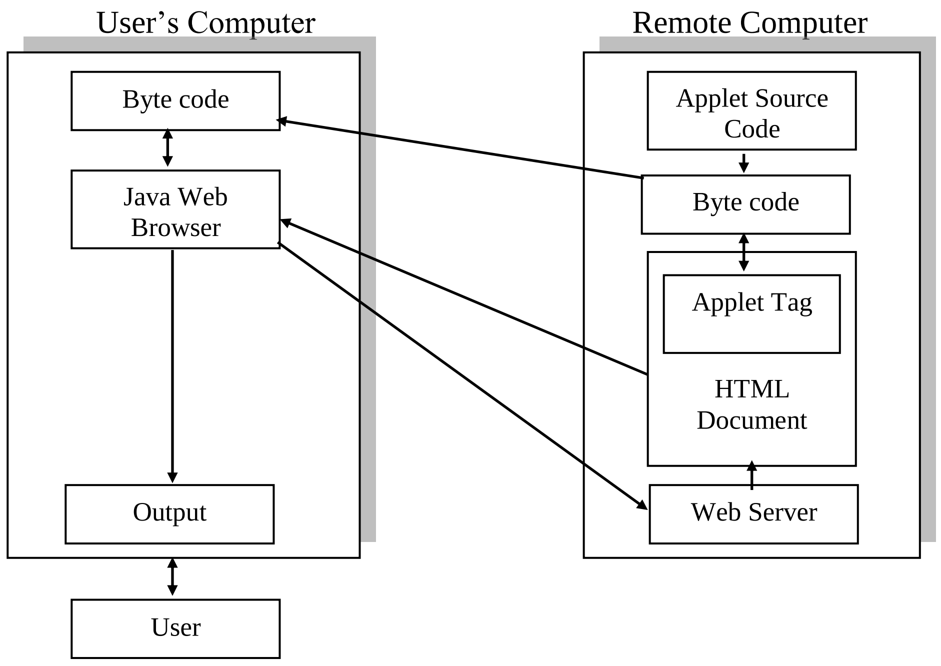


Fig.2.4.3 Java's interaction with the web

2.4.6 Web Browsers

Web browsers are used to navigate through the information found on the net. They allow us to retrieve the information spread across the Internet and display it using the HTML. *Web browsers* are

- HotJava
- Netscape Navigator
- Internet Explorer

2.4.7 Hardware And Software Requirements

Java is currently supported on Windows 95, Windows NT, Windows XP, Sun Solaris, Macintosh and UNIX machines.

The minimum hardware and software requirements for Windows 95 version of Java are

- IBM-compatible 486 system
- Minimum of 8 MB memory
- Windows 95 software
- A Windows-compatible sound card, if necessary
- A hard drive
- A CD-ROM drive
- A Microsoft-compatible mouse

2.4.8 Java Support Systems

Systems to support Java for delivering information on the Internet as shown below.

	Support systems		**Description**
❖	Internet connection	-	Local computer should be connected to the Internet.
❖	Web Server	-	A program that accepts requests and sends the required document.
❖	Web Browser	-	A program that provide access to WWW and runs Java applets
❖	HTML	-	A language for creating hypertext for the Web.
❖	APPLET Tag	-	For placing Java applets in HTML document.
❖	Java Code	-	Java code is used for defining Java applets.
❖	Byte Code	-	Compiled code and transferred to the user computer

2.4.9 Java Environment

Java Environment includes a large number of development tools is known as **Java Development Kit (JDK)** and classes and methods is known as **Java Standard Library (JSL)** or **Application Programming Interface (API)**.

Java Development Kit (JDK)

The Java Development Kit with a collection of tools that are used for developing and running Java programs. Java development tools are:

❖ Appletviewer	– Enable us to run Java applets
❖ javac	– Java compiler, which translates Java source code to byte code files.
❖ java	– ava interpreter, which runs applets and applications by reading and interpreting byte code files.
❖ Javap	– Java disassembler, which enables us to convert byte code files into a program description.
❖ javah	– Produce header files for us with native methods.
❖ javadoc	– Create HTML document from Java source code files.
❖ jdb	– Java debugger, which helps us to find errors in our programs.

To create a Java program, we need to create a source code file using a *text editor*. The source code compiled using the Java compiler **javac** and executed using the Java interpreter **java.** The tools are applied to build and run application programs are shown in Fig. 2.4.4.

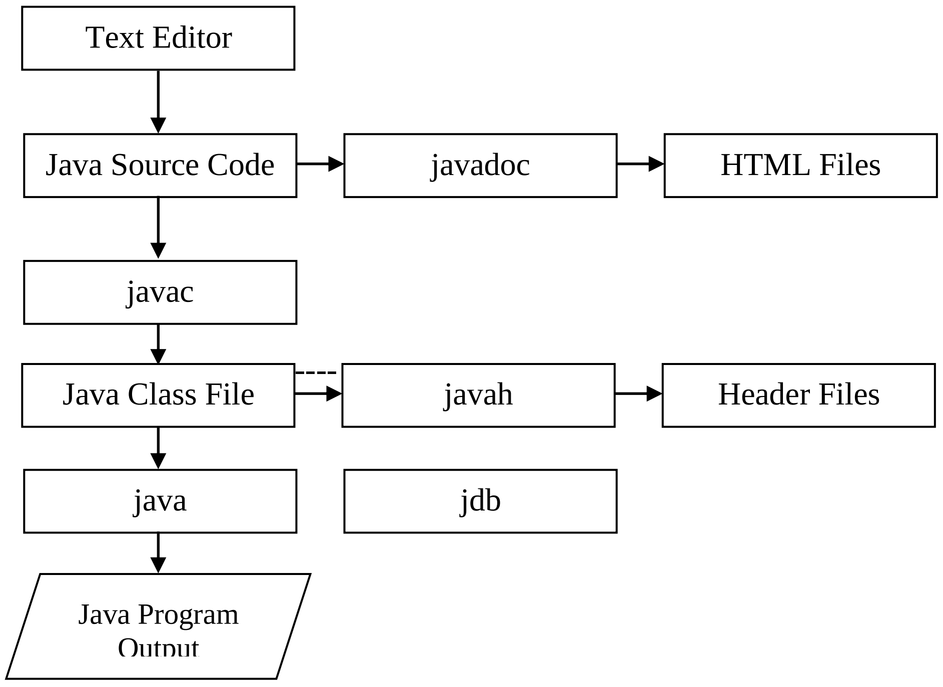


Fig.2.4.4 Process of building and running Java application programs

Application Programming Interface (API)

The Java standard Library includes hundreds of classes and methods grouped into several functional packages. Most common packages are

❖ **Language Support Package**: A collection of classes and methods for implementing basic features of Java.

- **Utilities Package**: A collection of classes to provide utility functions such as date and time manipulation.
- **Input/Output Package**: A collection of classes to provide Input/ Output manipulation.
- **Networking Package**: A collection of classes for communicating with other computers via Internet.
- **AWT Package**: The Abstract Window Tool Kit package contains classes that implement platform independent GUI.
- **Applet Package**: A collection of classes that allow us to create Java applet.

2.4.10 Self Assessment Questions

Fill in the blank

1. Java handling multiple tasks simultaneously is called ______________________.
2. Java ____________translates source code into byte code and Java______________ translates byte code into machine code that can be directly executed by the machine.

True / False

1. Java does support operator overloading.
2. AWT package is available in Java.

Multiple Choices

1.Java is a

a) Object-oriented language b) Pure Object-oriented language

c) Procedure language d) Object modeling language

2. JDK stands for

a) Java document kit b) Java definition Kit

c) Java development Kit d) none of the above

Short Answer

1. List any four features of Java.

____

____

2. What is mean by WWW?

____

____

2.5 Overview Of Java Language

2.5.1 Introduction

Java is a general-purpose object-oriented programming language. We can write *two types* of Java programs:

- Stand-alone applications
- Web applets

Stand-alone applications

Executing a stand-alone Java program need two-steps as shown in Fig.2.5.1.

1. Compiling source code into byte code using **javac** compiler.
2. Executing the byte code program using **java** interpreter.

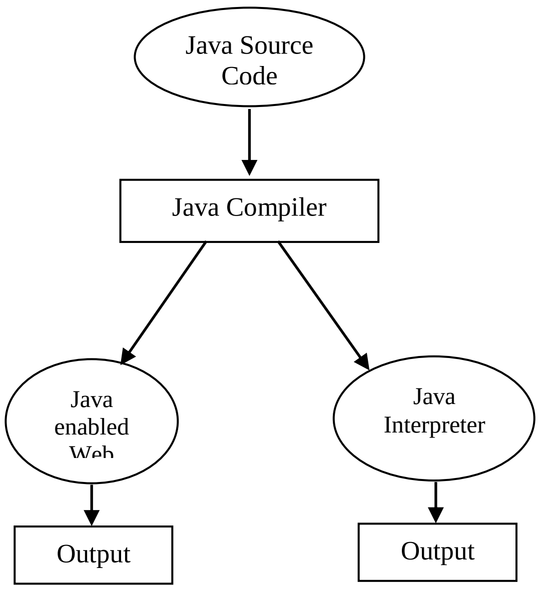


Fig.2.5.1 Ways of using Java

Web applets

Applets are small Java programs developed for Internet applications. Applet located on a distant computer can be downloaded via Internet and executed on a local computer using a java capable browser.

Simple and More Java Program

We begin with a very simple Java program that prints a line of text as output.

Program 2.5.1 A simple Java program

```
/*
*Simple and More Of Java Program
* This code  compute summation of two numbers
*/
class SimpleProgram
{
        public static void  main(String args[])
        {
                int x = 15, y = 25;       // Declaration and initialization
                int z;                    // Simple declaration
                z = x + y;
                System.out.println("Summation of two numbers" + z);
        }
}
```

Program 2.5.1 is the simplest of all Java programs. Let us discuss the program line by line and understand unique features that constitute a Java program.

The first line

```
class SimpleProgram
```

declares a class, Java is pure object-oriented language and everything must be placed inside a class. **class** is a keyword and SimpleProgram is a Java identifier that specifies the name of the class to be defined.

Every class definition in Java begins with opening brace "{" and ends with a matching closing brace "}".

The third line

```
public static void  main(String args[])
```

defines a method name **main**. This is the starting point for the interpreter to begin the execution of the program. A Java application can have any number of classes but only one on them must include a **main** method to initiate the execution. This line contains a **public keyword** is an access modifier, **static keyword**, which declares this method belongs to entire class and not part of any object and **void** states that the main method does not return any value.

String args[] declares a parameter named args, which contains any array of objects of the class type String.

The statement

```
int  x =15 , y = 25;
```

declares variable x and y and initializes it to the value 5 and the statement

```
int z;
```

merely declares a variable z. All of them have been declared as int type variables.

The executable statement in the program is

```
System.out.println("Summation of two numbers: " + z);
```

The **println** method is a member of the **out** object, which is a static data member of **System** class. This line prints the result on the screen as

```
Summation of two numbers: 40
```

Here , the **+** acts as the concatenation operator of two strings. The value of **z** is converted into string before concatenation. The method **println** always appends a new line character to the end of the string. Every Java Program must end with a *semicolon* (**;**).

In Java, the *single-line comments* begin with // and end at the end of the line as shown on the lines of the declaration x, y and z .The multi-line comments by starting with **/*** and end with a ***/** as shown at the beginning of the program.

An Application With Two Classes

A real time application will generally require multiple classes. Program 2.5.2 shows a Java application with two classes.

Program 2.5.2 A program with multiple classes

```
class  Sum
{
         int a, b;
        void getdata(int x, int y)
        (
                 a = x;
                 b = y;
         }
}
class Summation
{
         public static void  main(String args[])
        {
                int c;                    // Simple declaration
                Sum sum1 = new Sum();      // Creates an object  sum1
                sum1.getdata(10, 20);
                c = sum1.a + sum1.b ;
                System.out.println("Summation of two numbers" + c);
        }
}
```

Program 2.5.2 defines two classes **Sum** and **Summation**. The Sum class defines two variables and one method to assign values to these variables. The class **Summation** contains the **main** method that initiates the execution.

The **main** method declares a local variable **'c'** and **Sum** type object **sum1** and then assigns values to the data members of **Sum** class by using the **getdata** Method. Finally, it calculates the summation and prints the results.

2.5.4 Java Program Structure

Java Program Structure as shown in Fig.2.5.2

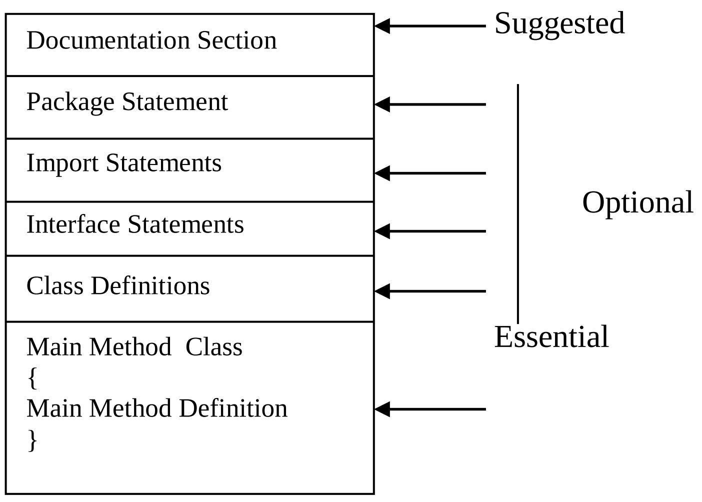


Fig. 2.5.2 General Structure of a Java program

Documentation Section

The documentation section comprises a set of comment lines giving the name of the program, the author and other details. Comments must explain why and what of classes and how of algorithms. In addition to the two styles, Java also uses third style of comment **/**…*/** known as **documentation comment**.

Package Statement

The first statement allowed in a Java file is a package statement. This statement declares a package name and informs the compiler that classes defined here belong to this package. **Example:**

```
package  student;
```

Package statement is optional.

Import statements

Import statement is similar to the #include in C.

Example:

```
import student.Test;
```

This statement instructs the interpreter to load the Test class in the package student.

Interface statement

An interface is like a class but group of *method declaration*. This is also optional section and is used only when we wish to implement the multiple inheritance features in the program.

Class definition

A java program may contain multiple class definitions. Classes are the primary and essential elements of a Java program.

Main Method class

Java stand-alone program requires a main method as its *starting point*; this class is the essential part of a Java program. The main method creates objects of various classes and establishes communications between them. On reaching the end of main, the program terminates and control passes back to the operating system.

2.5.5 Java Tokens

Smallest individual units in a program are known as **tokens**. The smallest units of program are the characters used to write Java tokens. Java language includes *five types* of tokens. They are:

- Reserved Keywords
- Identifiers
- Literals
- Operators
- Separators

Reserved Keywords

Java language has reserved 50 words as keywords. Table 2.5.1 list these keywords. Keywords, combined with operators and separators according to syntax, form definition of the Java language. All keywords are to be written in lower-case letters. Since keywords have specific meaning in Java, we cannot use them as names for variable, classes, methods and so on.

Identifiers

Identifiers are programmer-designed tokens. They are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program.

Rules for forming Java identifier as

1.They can have alphabets, digits, and the underscore and dollar sign characters.

2.They must not begin with a digit.

3.Uppercase and lowercase letters are distinct.

4.They can be of any length.

Table 2.5.1 Java Keywords

abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	interface

implements	import	instanceof	int
long	native	new	package
private	protected	public	return
short	static	stricfp	supe
switch	synchronized	this	throw
throws	transient	try	void
volatile	while		

Literals

Literals in Java are a sequence of characters (digit, letters, and other characters) that represent constant value to be stored in variables. Java language specifies *five types* of literals. They are:

- ❖ Integer literals
- ❖ Floating-point literals
- ❖ Character literals
- ❖ String literals
- ❖ Boolean literals

Operators

An **operator** is symbol that takes one or more arguments and operates on them to produce a result.

Separators

Separators are symbols used to indicate where groups of code are divided and arranged. Table 2.5.2 lists separators and their functions.

Table 2.5.2 Java Separators

Name	Purpose
Parentheses ( )	Used to enclose parameters in method definition and invocation, also used for defining precedence in expression, containing expressions for flow control, and surrounding cast types.
Braces { }	Used to contain the values of automatically initialized arrays and to define a block of code for classes, methods and local scopes
Brackets [ ]	Used to declare array types and for de-referencing array values.
Semicolon ;	Used to separate statements
Comma ,	Used to separate consecutive identifiers in a variable declaration, also used to chain statement inside 'for' statement
Period .	Used to separate package names from sub-packages and classes; also used to separate a variable or method from a reference variable.

2.5.6. Java Statements

The statements in Java are like *sentences in natural language*. Java implements several types of statements in Fig 2.5.3.

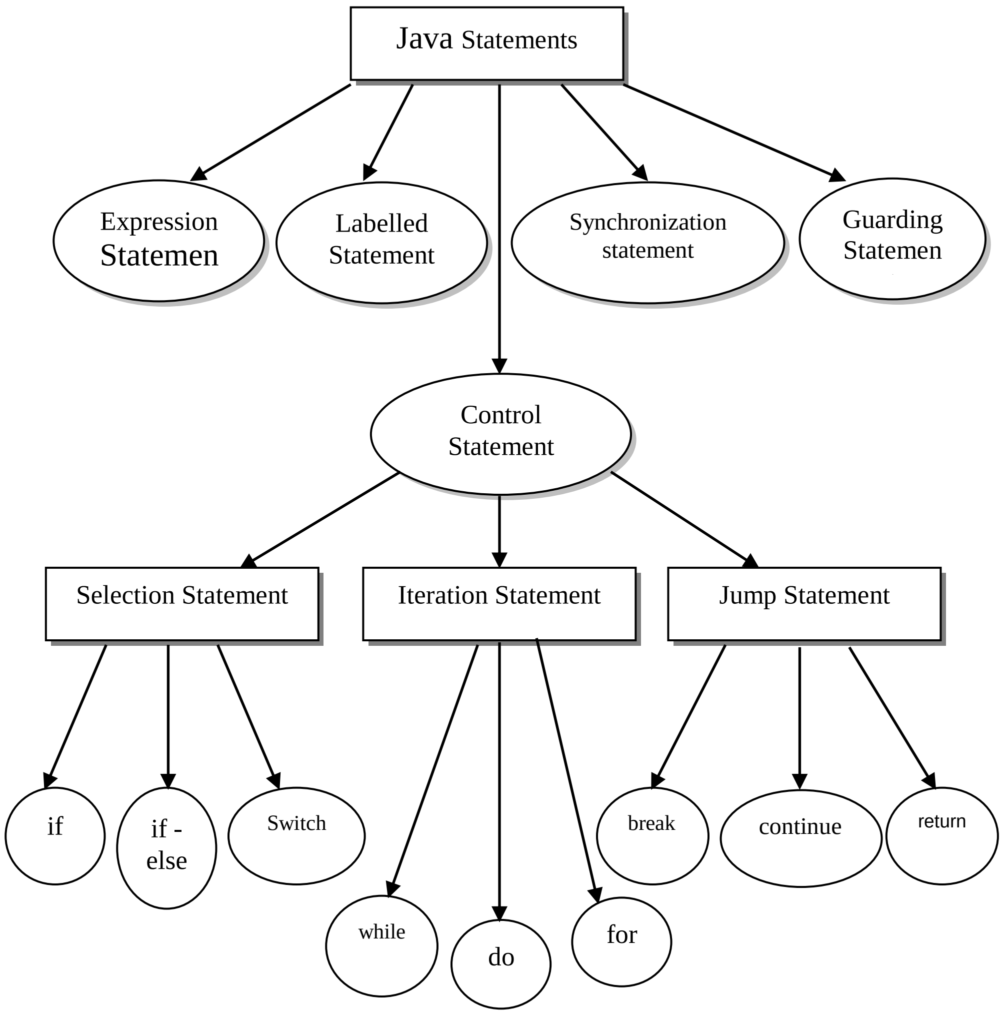


Fig 2.5.3 Classification of Java Statements

2.5.7 Implementing A Java Program

Implementing of a Java application program involves a series of steps. They include:

- Creating the program
- Compiling the program
- Running the program

Creating the program

We can create a program using any *text editor*. Assume that we have entered the following program:

Program 2.5.3 Simple program for testing

```
class SimpleProgramTest
{
        public static void  main(String args[])
        {
                System.out.println("Welcome to Java" );
        }
}
```

We must save this program in a file called **SimpleProgramTest.java** ensuring that the filename contains the *class name properly*. This file is called *source file*, all source files will have the extension .**java**. If a program contains multiple classes, the file name must be the class name of the class containing the **main** method.

Compiling the program

To compile the program, we must run the Java Compiler **javac**, with the name of the source file on the command line as shown below:

```
>javac SimpleProgramTest.java
```

If everything OK, the **javac** compiler creates a file called *SimpleProgramTest.class* containing the byte codes of the program.

Running the program

We need to use the **java** interpreter to run a stand-alone program. At command prompt, type

```
>java SimpleProgramTest
```

Now, the interpreter looks for the main method in the program and begins execution from there. When executed, our program displays the results as:

```
Welcome to Java
```

2.5.8 Java Virtual Machine

Java compiler produces an intermediate code known as byte code for machine that does not exist. This machine is called the **Java Virtual Machine** and it exists only inside the computer memory. Fig. 2.5.4 shows the process of compiling a Java program into byte code, which is also referred to as **virtual machine code**.

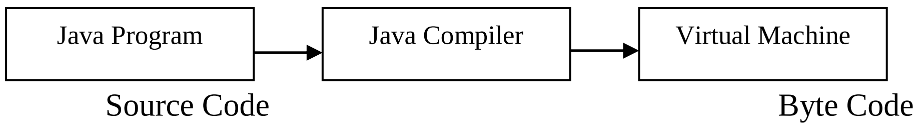


Fig.2.5.4 Process of compilation

The virtual machine code is not machine specific. The machine specific code is generated by the Java interpreter by acting as an intermediary between the virtual machine and real machine as shown in Fig. 2.5.5.

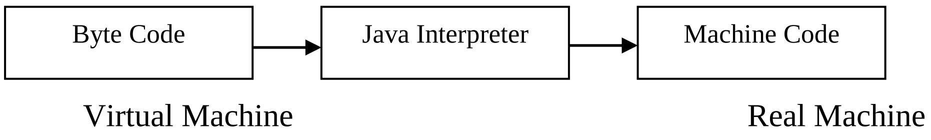


Fig.2.5.5 Process of converting byte code into machine code

2.5.9 Command line Arguments

Command line arguments are *parameters* that are supplied to the application program at time of invoking it for execution. We can write java programs that can receive and use the arguments provided in the command line.

Program 2.5.4 Simple program for Command line argument

```
/*
*  Program for Command line arguments as input
*/
class ComLineTest
{
        public static void  main(String args[])
        {
                int count , i = 0;
                String  string;
                count  = args.length;
                System.out.println("  No.  of  arguments  =  "  +
        count);
                while ( i < count )
                {
                        string = args[ i];
                        i = i +1;
                        System.out.println( i  +"   :  " + string );
                 }
        }
}
```

Program 2.5.4 shows the use of command line arguments. Compile and run the program with the command line as follows.

```
>java ComLineTest   BASIC  FORTRAN  C++  JAVA
```

Upon execution, the command line arguments BASIC FORTRAN C++ JAVA are passed to the program through the array args . that is the element args[0] contains BASIC, args[1] contains FORTRAN, and so on. These elements are accessed using the loop variables i as an index like

```
name = args[i]
```

The index i is incremented using a while loop until all the arguments are accessed. The number arguments is obtained by statement

```
count = args.length;
```

The output of the program as:

```
No. of arguments = 4
1        :  BASIC
2        :  FORTRAN
3        :  C++
4        :  JAVA
```

2.5.10 Programming Style

Java is a freeform language. Java system does not care where on the line we begin typing. For **example**, the statement

```
System.out.println( " Java is Best" );
```

can be written as

```
System.out.println
( " Java is Best" );
```

or, even as

```
System
.out
.println
(
 " Java is Best"
);
```

2.5.11 Self Assessment Question

Fill in the blank

1. Java has two types of program such as____________________ and _______________.
2. Java program is converted into byte code, which is also referred to as ______________.

True / False

1. We can create a program using any *text editor*.
2. main( ) method is the starting point of the interpreter to begin the execution of the program

Multiple Choices

1.Java applet program runs under

a) Hot java	b) Netscape Navigator
c) Internet Explorer	d) All of the above

2.Java program is compiled by

a) java	b) javac
c) javadoc	d) javap

Short Answer

1. Define an applet.

2.What is called Command line arguments?

2.6. Data types, Constants And Variables

2.6.1 Data Types

The size and type of values that can be stored in variable is called as **Data type**. Data types in Java under various *categories* are shown in Fig.2.6.1.

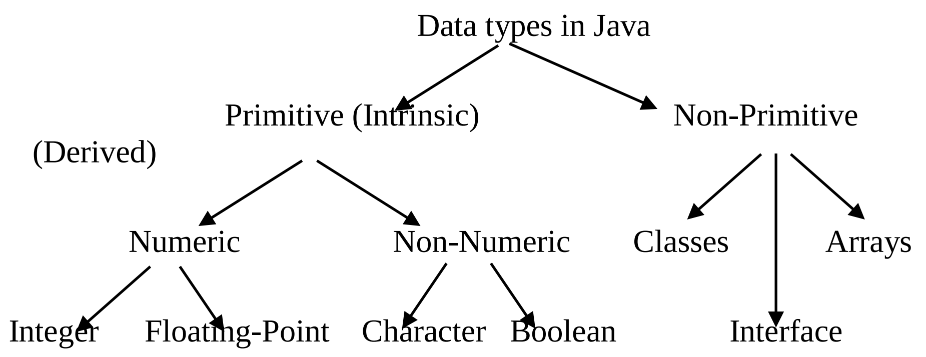


Fig.2.6.1 Data types in Java

Integer Types

Integer types can hold whole numbers such as 456, -26, and 6873. Java supports *four types* of integers are **byte, short, int** and **long** as in Fig.2.6.2. Java *does not* support the concept of *unsigned types* and therefore all Java values are signed meaning they can be positive or negative. Table 2.6.1 shows the memory size of all the **four integer data types**.

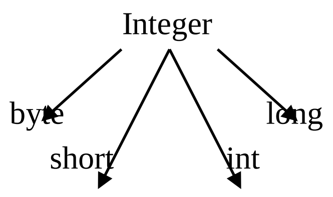


Fig.2.6.2 Integer Data types

We must use a **byte** variable to handle smaller number. This improves the speed of execution of the program. We can make integers **long** by appending the letter *L* or *l* at the end of the number.

Example: 123L or 123l.

Table 2.6.1 Type and size Of Integer Types

Type	Size
byte	One byte
short	Two bytes
int	Four bytes
long	Eight bytes

Floating Point Types

Integer types can hold only whole numbers and therefore we use another type known as **floating point type** to hold numbers containing fractional parts such as 27.59 and -1.375. There are *two kinds* of floating point storage in Java as shown in Fig.2.6.3

$$\begin{tikzcd}
 & \text{Floating Point} \arrow[dl] \arrow[dr] & \\
\text{float} & & \text{double}
\end{tikzcd}$$

Fig.2.6.3 Floating-point data types

The **float** type values are single-precision numbers while the double types represent double-precision numbers. Table 2.6.2 gives the size of these *two types*.

Table 2.6.2 Type And Size Of Floating Point

Type	Size
float	4 bytes
double	8 bytes

Floating point numbers are treated as double-precision quantities. To force them to be in single-precision mode, we must append *f* or *F* to the numbers.

Example:

1.23f 7.56923e5F

Double-precision types are used when we need greater precision in storage of floating point numbers. All mathematical functions such as **sin, cos** and **sqrt** return **double** type values.

Character Type

In order to store character constants in memory, Java provided a character data type called **char**. The char type assumes a size of *2 bytes* but it can hold only a single character.

Boolean Type

Boolean type is used when we want to test a particular condition during the execution of the program. There are only *two values* that a Boolean type can type: **true** or **false**. Boolean type denoted by the keyword **boolean** and uses only one bit of storage. The words **true** and **false** cannot be used as the identifier.

2.6.2 Constants and Symbolic constants

Constants:

Constants in Java refer to *fixed values* that do not change during the execution of a program. Types of constant are as shown in Fig.2.6.4.

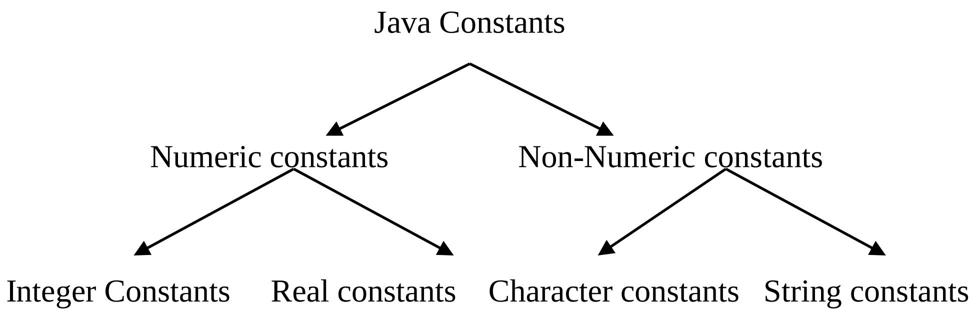


Fig.2.6.4 Java Constants

Numeric Constants

Integer Constants

A *whole number* is called **integer constants**. An integer constant refers to a sequence of digits. There are three types of integers, namely, decimal, octal and hexadecimal integer.

Decimal integers consist of a set of digits, *0 through 9*, preceded by optional minus sign. Valid **examples** are:

123 -231 0 253444

Embedded spaces, commas, and non-digit characters are not permitted between digits. For **example**

36 5689 20.000 $777456 are invalid numbers

An octal integer constant consists of any combination of digit from the set *0 through 7*, with leading 0. Valid **examples** are:

046 123 0 0675

A hexadecimal integer constant consists of any combination of digit from the set *0 through 9* and *A through F* and preceded by 0x or 0X

Valid **examples** are:

0x3Bf7 0X27 0x

Real Constants

Number containing with *decimal point* is called **real constants**. There are *two types* of notation, namely,

- ❖ **Decimal notation**
- ❖ **Exponential (or scientific) notation.**

In **decimal notation** a whole number followed by a decimal point and the fractional part, which is an integer. Valid **examples** are:

0.376 .57 -.46 23.68

The *general form* of an **Exponential notation** is:

mantissa e exponent

The **mantissa** is either a real number expressed in *decimal notation* or *an integer*. The **exponent** is an integer with an optional *plus* or *minus* sign. The letter **e** separating the mantissa and the exponent can be written in either lowercase or uppercase.

Valid **examples** are:

0.47E2 12e-7 1.8e+3 7.3E2 -6.0e-2

Non-Numeric Constants:

Character constants

A single character constant contains a **single character** enclosed within a *pair of single quote marks*. Valid **examples** are:

‘6’ ‘s’ ‘W’ ‘;’ ‘ ’

String constants

A string constant is a **sequence of characters** enclosed between *double quotes*. The characters may be alphabets, digits special characters and blank spaces.

Valid **examples** are:

“Hello” “3566”“%---$” “22-7”

Backslash character constants

Java supports some special backslash character constants that are used in output methods. The characters combinations are known as **escape sequences**.

Table 2.6.3 Backslash character constants

Constants	**Meaning**
‘ \ b’	back space
‘ \ f ’	form feed
‘ \ n’	new line
‘ \r’	carriage return
‘ \ t’	horizontal tab
‘ \ ‘’	single quote
‘ \ “’	double quote

‘ \\’	back slash

Symbolic constants

We often use certain unique constants in a program. These constants may appear repeatedly in a number of places in the program. **For example** of such a constant is 3.142 representing the value of the mathematical constant “pi”. We face two problems in the subsequent use of program. They are:

1. Problem in modification of the program.
2. Problem in understanding the program.

Modifiability

We may like to change the value of “pi” from 3.142 to 3.14159 to improve the accuracy of calculation. In this case, we will have to search throughout the program and explicitly change the value of the constant wherever it has been used. If any value is left unchanged, the program may produce incorrect outputs.

Understandability

Assignment of a symbolic name to numeric constants frees us problems like same value means different things in different places. **For example**, the number 40 may mean the number of students at one place and the “pass marks” at another place of the same program. We may use the name STRENGTH to denote the number of students and PASS_MARK to denote the pass marks required in subject. Constant values are assigned to these names at the beginning of the program.

A constant is declared as follows:

```
final  type symbolic-name = value;
```

Valid **examples** are:

```
final int STRENGTH =  40;
final PASS_MARK = 40;
final float PI = 3.1459;
```

Rules for forming the symbolic constants are:

- ❖ Symbolic names take the same form as variable names. But, they are written in CAPITALS.
- ❖ After declaration of symbolic constants, they should not be assigned any other value within a program.
- ❖ Symbolic constants are declared for types.
- ❖ They cannot be declared inside a method. They should be used only a class data members in the beginning of the class.

2.6.3 Variables

A **variable** is an identifier that denotes a storage location used to store a data value. A variable may take different values at different times during the execution of the program. Variable names may consist of alphabets, digits, the underscore (_) and dollar characters, with following *conditions.*

1. They must not begin with a digit
2. Uppercase and lowercase are distinct. This means that the variable Total is not the same as total or TOTAL.
3. It should not be a keyword
4. White space is not allowed
5. Variable names can be of any length.

2.6.4 Declaration Of Variables

In Java, variables are the names of the storage locations. A variable must be declared before it is used in the program. A variable can be used to store a value of any data type. After designing the variable names, we must declare them to the complier. Declaration does *three things*:

1. It tells the compiler what the variable name is
2. It specifies what type of data the variable will hold.
3. The place of declaration decides the scope of the variables.

The *general form* of declaration of a variable is:

```
type variable1, variable2,……, variableN;
```

Variables are separated by commas. A declaration statement must end with a semicolon. Some **valid declarations** are:

```
int     rollno;
float   average;
double  pi;
byte    b;
char    c1, c2;
```

2.6.5 Giving Values To Variables

A variable must be given a value after it has been declared that before it is used in an expression. This can be achieved in *two ways*:

1. By using an assignment statement
2. By using a read statement

Assignment Statement

A simple method of giving value to a variable is through the assignment statement as

```
variableName = value;
```

For **Example:**

```
rollno = 1;
```

```
c1      = ‘ x ‘;
```

Another method to assign a value to a variable at the time of its declaration as

$$\boxed{\textbf{type variableName = value;}}$$

For **Example**:

```
int      rollno      =      1;
float    average     =      68.66;
```

The process of giving initial values to variables is known as the initialization. The following are **valid Java statements**:

```
float   x,  y,  z;              // declares three float variables
int     m =  3,  n  = 6;// declares and initializes two int variables
```

Read Statement

We may also give values to variables through the keyboard using the readLine() method as shown in Program 2.6.1.

Program 2.6.1 Reading data from keyboard

```
import java.io.DataInputStream;
class Reading
{
public static void main(String args[])
DataInputStream in = new DataInputStream(System.in);
                                          //declare in object
int  IntNumber = 0;
float floatNumber = 0.0f;
try
{
System.out.println(“Enter an integer number:   ”);
intNumber      =  Integer.parseInt(in.readLine());   // read  integer value
                                                        through  keyboard
System.out.println(“Enter a float number:   ”);
floatNumber    = Float.valueOf(in.readLine()).floatValue(); // read float

                value
}
catch (Exception        e)  {     }
System.out.println(“Integer Number  =    ”   +        intNumber);
System.out.println(“Float Number    =    ”   +        floatNumber);
}
}
```

Output:

```
Enter an integer number:
56
Enter a float number:
45.90
Integer Number  =   56
Float Number    =   45.90
```

The **readLine()** method (which is invoked using an object of the class **DataInputStream**) the input from keyboard as a string is then converted to the corresponding data type using data type wrapper classes. We have used the keywords try and catch to handle any errors that might occur during reading process.

2.6.6 Scope Of Variables

Java variables are classified into *three kinds*:

- Instance variables
- Class variables
- Local variables

Instance and class variables are declared *inside a class*. Instance variables are created when the objects are instantiated and therefore they are associated with the objects. They take different values for each object. Class variables are global to a class and belong to the entire set of objects that class creates. Only one memory location is created for each class variable.

Variables declared and used inside methods are called **local variables**. They are not available for use outside the method definition. Local variables can also be declared inside program blocks that are defined between an **opening brace {** and **a closing brace }.** These variables are visible to the program only from the beginning of its program block to the end of the program block. When the program control leaves a block, all the variables in the block will cease to exist. The area of the program where the variable is accessible (ie. Usable) is called its scope.

2.6.7 Getting Values Of Variables

A program is written to manipulate a given set of data and to display or print the results. Java supports *two output methods* that can be used to send the results *to the screen*.

- **print( )** method
- **println( )** method

The **print( )** method sends information into a buffer. This buffer is not flushed until a new line (or end-of-line) character is sent. As a result, the **print( )** method prints output on one line until a new line character is encountered.

For example, the statements,

```
System.out.print(“Hai  ”);
System.out.print(“Java ! “);
```

will display the words **Hai Java !** on one line and waits for displaying further information on the same line. We may display on next line by printing a new line character as follows:

```
System.out.print(“ \n “);
```

For Example:

```
System.out.print(“Hai  ”);
System.out.print(“ \n “);
System.out.print(“Java ! “);
```

will display the output in two lines as

Hai

Java !

The **println( )** method takes the information provided and displays it on a line followed by a line feed . **For Example**, the statements

```
System.out.println(“Hai  ”);
System.out.println(“Java ! “);
```

will produce the following output:

Hai

Java !

2.6.8 Type Casting

We need to store a value of one type into a variable of another type. In such situation, we must cast the value to be stored by proceeding it with the type name in parentheses. The general form is:

type variable1 = (type) variable2;

The process of converting one data type to another is called **casting**.

Examples:

```
int   m   =  50;
byte  n   =  ( byte ) m;
long  c   =  ( long ) m;
```

Four integer types can be cast to any other type except Boolean. Similarly, the float and double can be cast to any other type except Boolean. Casting to smaller type can result in a loss of data. Casting a floating-point value to an integer will result in a loss of the fractional part. Table 2.6.3. lists those casts, which are guaranteed to result in no loss of information

Table 2.6.3 Casts that results in No Loss Information

From	To					
byte	short,	char,	int,	long,	float,	double
short	int,	long,	float,	double		

char	int, long, float, double
int	long, float, double
long	float, double
float	double

Automatic Conversion

For some types, it is possible to assign a value of one type to a variable of a different type without a cast. Java does the conversion of the assigned value automatically. This is known as automatic type conversion. **For example**, int is large enough to hold a byte value. Therefore,

```
byte    b    =    75;
int     a    =    b;
```

are valid statements.

The process of assigning a smaller type to a larger one is known as **widening** or **promotion** and that of assigning a larger type to a smaller one is known as **narrowing**.

2.6.9 Self Assessment Question

Fill in the blank

1. Whole number is called ___________________.

2. The process of assigning a smaller type to a larger one is known as _______________.

True / False

1.Java does support the concept of unsigned types.

2.A variable must be declared before it is used in the program.

Multiple Choices

1. Which of the following is not a valid identifier

   a) averave b) sum_S

   c) Total d) 2ABC

2. Which of the following is correct Java statement

   a) System.Out.println("hello"); b) System.out.println('hello');

   c) System.out.println("hello"); d) system.out.println("hello");

Short Answer

1. Define variables.

----

2. Define local variable.

2.7 Operators and Expressions

2.7.1 Introduction

Java supports a rich set of operators. An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables.

Java operators can be *classified into a number of types* are:

- ❖ Arithmetic operators
- ❖ Relational operators
- ❖ Logical operators
- ❖ Assignment operators
- ❖ Increment and decrement operators
- ❖ Conditional operators
- ❖ Bitwise operators
- ❖ Special operators
- ❖ Special operators

2.7.2 Arithmetic Operators

Java provides all the basic arithmetic operators are listed in Table 2.7.1. The operators **+**, **-**, ***** and **/** all work the same way as they do in other languages. These can operate on any built-in numeric data type of Java. We cannot use these operators on Boolean type. The unary minus operator, in effect, multiplies its single operand by -1. Therefore, a number preceded by a minus sign changes its sign.

Table 2.7.1. Arithmetic Operators

Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division(Remainder)

Arithmetic operators are used as

a - b	a + b
a * b	a / b
a % b	- a + b

Integer Arithmetic

When both the operands in single arithmetic expression such as a + b are integers, the expressions is called an **integer expression**, and the operation is called **integer arithmetic**. Integer arithmetic yields an integer value. In above **examples** if a and b are integers the a =2 and b = 2 we have the following results:

```
a – b   =   0
a + b   =   4
a * b   =   4
a  / b  =   1   (decimal part truncated)
a % b   =   0   (remainder of integer division)
```

For modulo division (%), the sign of the result is always the sign of the first operand.

Real arithmetic

An arithmetic operation involving only *real operands* is called **real arithmetic**. A real operand may assume values either in *decimal* or *exponential notation*. The floating-point modulus operator returns the floating-point equivalent of an integer division. What this means is that the division is carried out with both floating-point operands, but the resulting divisor is treated as an integer, resulting in a floating-point remainder.

Example Program: Program for Arithmetic operator works on Floating values

```
class RealArithmetic
{
        public static void main(String args[ ])
        {
                float a = 10.3f, b= 3.2f;
                System.out.println(" a + b = " + (a + b));
                System.out.println(" a - b = " + (a - b));
                System.out.println(" a * b = " + (a * b));
                System.out.println(" a  / b = " + (a / b));
                System.out.println(" a % b = " + (a % b));
        }
}
```

Output Of Program

```
a + b  = 13.5
a - b  = 7.1000004
a * b  = 32.960003
a / b  = 3.21875
a % b  = 0.70000005
```

Mixed-mode Arithmetic

When one of the operand is real and the other is integer, the expression is called a **mixed-mode arithmetic expression**. If either operand is of the real type, then the other operand is converted to real and the real arithmetic is performed. The result will be a real. Thus

16 / 5   produce the result   3.1

Whereas

16 / 5   produce the result   1

2.7.3 Relational Operators

We often *compare two quantities*, and depending on their relation, take certain decisions. For **example**, we may compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of relational operators. Java supports *six relational operators* as shown in Table 2.7.2.

Table 2.7.2. Arithmetic Operators

Operator	Meaning
<	is less than
<=	is less than equal
>	is greater than
>=	is greater than equal
= =	is equal to
! =	is not equal to

A simple relational expression contains only one relational operator and is of the following *form*:

ae -1   relational operator   ae – 2

When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared. That is, arithmetic operators have a higher priority over relational operators.

Example Program: Program for implementation of Relational Operator

```
class RelationalOp
{
        public static void main(String args[ ])
        {
                float a = 10.6f, b= 3.2f;
                System.out.println(" a  = " + a);
                System.out.println(" b  = " + b);
                System.out.println(" a < b  is " + (a < b));
                System.out.println(" a > b is " + (a > b));
                System.out.println(" a = = b is " + (a ==b));
                System.out.println(" a <= b is " + (a <= b));
                System.out.println(" a >= b is " + (a >= b));
                System.out.println(" a = = b is " + (a !=b));
        }
}
```

Output Of Program

```
a  = 10.6
b  = 3.2
a < b  is false
a > b is true
a = = b is false
a <= b is false
a >= b is true
a = = b is true
```

2.7.4 Logical Operators

Java has *three logical operators* as shown in Table 2.7.3. The logical operators **&&** and **||** are used when we want to form compound conditions by combining two or more relations. **Example:**

```
a  >  b  &&    x    = =  10
```

Table 2.7.3. Logical Operators

Operator	Meaning
&&	is logical AND
||	is logical OR
!	is logical NOT

An expression combines two or more relational expression is called as *logical expression* or *compound relational expression*. Logical expression also yields a value of true or false.

2.7.5 Assignment Operators

Assignment operators are used to assign the value of an expression to a variable. The *form*

v op= exp;

Where **v** is a variable, **exp** is an expression and **op** is a java binary operator. The operator **op=** is known as the *shorthand assignment operator*.

The shorthand assignment operators are illustrated in Table 2.7.4.

Table 2.7.4 Shorthand Assignment Operators

Statement with simple Assignment operator	Statement with Shorthand operator
a = a + 1	a + = 1

```
a  =  a  - 1                a  - = 1
a  =  a * ( n  + 1)         a  * =  n + 1
a  =  a   %  b              a   % =  b
a  =  a / ( n  + 1)         a  / =  n + 1
```

The use of shorthand assignment operators has *three advantages*:

1. What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
2. The statement is more concise and easier to read
3. Use of shorthand operator results in a more efficient code.

2.7.6 Increment And Decrement Operators

The increment and decrement operators:

++ and **- -**

The operator ++ adds 1 to the operand while -- subtracts 1. Both are unary operators and are used in the following form:

```
++m;    or      m++;
--m;    or      m--;
```

We use the increment and decrement operators extensively in for and while loops.

Example:

```
m   =  5;
y   = ++m;
```

In this case, the value of y and m would be 6. Suppose, if we rewrite the above statements

```
m   =  5;
y   = m++;
```

Then, the value of y would be 5 and m would be 6. Prefix operator adds 1 to the operand and then the result is assigned to the variable on left. Postfix operator first assigns the value to the variable on left and then increments the operand.

Similar is the case, when we use ++(or --) in subscripted variables. That is the statement a[i++] = 10 is equivalent to

```
a[ i ]   =   10
i        =    i + 1
```

2.7.7 Conditional Operators

The conditional pair **? :** is a ternary operator available in Java. This operator is used to construct conditional expressions of the *form*

p1 ? exp2 : exp3

Where exp1,exp2, exp3 are expressions.

The operator ? : works as follows : **exp1** is evaluated first. If it is nonzero (true), then the expression **exp2** is evaluated and becomes the value of the conditional expressions. If **exp1** is false, **exp3** is evaluated and is value becomes the value of the conditional expression.

For **Example**:

```
a    =    10;
b    =    15;
x    =    (a > b)  ?  a : b
```

In this example, the value of x is the value of b.

2.7.8 Bit Wise Operators

Java has a special operators is known as **bitwise operators** or manipulation of data at values of bit level. These operators are used for testing the bits, or shifting them to the right or left as shown in Table 2.7.5.

Table 2.7.5. Bitwise Operators

Operator	**Meaning**
&	bitwise AND
!	bitwise OR
^	bitwise Exclusive OR
~	one's Complement
<<	shift left
>>	shift right
>>>	shift right with zero fill

2.7.9 Special Operators

Java supports some special operators of interest such as **instanceof** operator and **member selection** operator (.).

Instanceof Operator

The **instanceof** is an object reference operator and returns true if the object on the left-hand side is an instance of the class given on the right-hand side. This operator allows us to determine whether the object belong to a particular class or not.

Example:

```
person instanceof    student
```

is **true** if object **person** belong to the class **student**; otherwise it is false.

Dot operator

The dot operator (.) is used to access the instance variables and methods of class objects.

Example:

```
person1.age;            // Reference to the variable age
person1.salary( );      // Reference to the method salary( )
```

It is also used to access classes and sub-packages from a package.

2.7.10 Arithmetic Expressions

An arithmetic expression is a combination of variables, constants, and operators. Example of Java expression is shown in Table 2.7.6.

Table 2.7.6. Expressions

Algebraic Expression	Java Expression
a b - c	a * b - c
$\frac{ab}{c}$	a * b / c

2.7.11 Evaluation Of Expressions

Expressions are evaluated using assignment statement of the *form*

variable = expression;

variable is any valid Java variable name. When the statement is encountered, the expression is evaluated first and the result then replaces the previous value of the variable on the left-hand side.

Examples of evaluation statements are

```
 l  =  x*y-z ;
m = y/z*x ;
```

The blank space around an operator is optional.

2.7.12 Precedence Of Arithmetic Operators

An arithmetic expression without any parentheses will be evaluated from left to right using the rules of precedence of operators. There are *two distinct priority levels* in Java:

High priority   *   /   %

Low priority   +   -

The basic evaluation procedure includes two left-to-right passes through the expression. During the **first pass,** the high priority operators are applied and **second pass**, the low priority operators are applied.

Consider the following evaluation statement:

x = a - b / 3

When a = 9 and b = 6, the statement becomes

x = 9 – 6 / 3

and evaluated as

First pass

x = 9 – 2   (6 / 3 evaluated)

Second pass

x = 7 (9 - 2 evaluated)

Introducing parentheses into expression can change the order of evaluation. Parentheses may be nested, and in such cases, the expression will proceed out from innermost set of parentheses. Every opening parenthesis has a matching closing one. Parentheses allow us to change the order of priority.

2.7.13 Type Conversions In Expressions

Automatic Type Conversion

Java permits mixing of constants and variables of different types in an expression, but during evaluation it adheres to very strict rules of type conversion. If the operands are of different types, the 'lower' type is automatically converted to the higher type before the operation proceeds. The result is of the higher type. Table 2.7.7 provides a reference chart for type conversion.

The final result of an expression is converted to the type of the variable on the left of the assignment sigh before assigning the value to it. The following changes are occurs in final assignment.

1. float to int causes truncation of the fractional part.
2. double to float causes rounding of digits.
3. long to int causes dropping of the excess higher order bits.

Table 2.7.7 Automatic Type Conversion Chart

	char	**byte**	**short**	**int**	**long**	**float**	**double**
char	int	int	int	int	long	float	double
byte	int	int	int	int	long	float	double
short	int	int	int	int	long	float	double
int	int	int	int	int	long	float	double
long	long	long	long	long	long	float	double
float	float	float	float	float	float	float	double
double	double	double	double	double	double	double	double

Casting a Value

We need to store a value of one type into a variable of another type. In such situation, we must cast the value to be stored by proceed it with the type name in parentheses. The *general form* of a cast is:

(type_name) expression

Where type_name is one of the standard data types. The expression may be constant, variable or an expression.

Examples of casts and their actions are

X = (int) 7.5	7.5 is converted to integer by truncation
A = (int) 21.3 / (int)4.5	Evaluated as 21/4 and the result would be 5

2.7.14 Operator Precedence And Associativity

Each operator in Java has precedence associated with it. The operators at the higher level of precedence are evaluated first. The operators of the same level of precedence are evaluated from left to right or from right to left, depending on level. This is known as the **associativity property of an operator**. Table 2.7.8 provides a complete lists of operators, their precedence levels, and their rules of association.

Table 2.7.8 Java Operators precedence and associativity

Operator	**Meaning**	**Associativity**	**Rank**
.	Member selection	Left to Right	1
( )	Function call		
[ ]	Array element reference		
-	Unary minus	Right to left	2
++	Increment		
--	Decrement		
!	Logical Negation		
~	One's complement		
(type)	Casting		
*	Multiplication	Left to Right	3
/	Division		
%	Modulus		
+	Addition	Left to Right	4
-	Subtraction		
<<	Left shift	Left to Right	5
>>	Right shift		
>>>	Right shift with zero fill		
<	Less than	Left to Right	6
< =	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
Instanceof	Type comparison		
= =	Equality	Left to Right	7
! =	Inequality		
&	Bitwise AND	Left to Right	8
^	Bitwise XOR	Left to Right	9
|	Bitwise Or	Left to Right	10
&&	Logical AND	Left to Right	11
||	Logical OR	Left to Right	12
?:	Conditional operator	Right to Left	13
=	Assignment operator	Right to Left	14

Op=	Shorthand assignment

2.7.15 Mathematical Functions

Java support basic math function through **Math** class defined in the **java.lang** package. Table 2.7.9 lists the math functions defined in the Math class. These functions should be used as

```
Math.function_name();
```

Example:

```
int   a = Math.max(10,15);
```

Table 2.7.9 Math Function

Function	**Action**
sin(x)	Returns the sine value the angle x in radians
cos(x)	Returns the cosine value the angle x in radians
tan(x)	Returns the tangent value the angle x in radians
asin(y)	Returns the angle whose sine is y
acos(y)	Returns the angle whose cosine is y
atan(y)	Returns the angle whose tangent is y
atan2(x,y)	Returns the angle whose tangent is x/y
pow(x,y)	Returns x raised to y(xy)
exp(x)	Returns e raised to x(ex)
log(x)	Returns the natural logarithm of x
sqrt(x)	Returns the square root of x
ceil(x)	Returns the smallest whole number greater than or equal to x (rounding up)
floor(x)	Returns the largest whole number less than or equal to x (Rounded down)
rint(x)	Returns the truncated value of x.
round(x)	Returns the integer closest to the argument
abs(a)	Returns the absolute value of a.
max(a, b)	Returns the maximum of a and b
min(a, b)	Returns the minimum of a and b

Note: x and y are double type parameters. a and b may int, long, float and double.

2.7.16 Self Assessment Questions

Fill in the blank

1. ______________ are used in programs to manipulate data and variables.

2. Java support basic math function through ________ class defined in the ______________package.

True / False

1. The logical operators does not return true and false value

2. The dot operator (.) is used to access the instance variables and methods of class objects.

Multiple Choices

1. Suppose m=5,y=m++, what is value of m and y?

a) 5 & 6	b) 6 & 5
c) 5 & 5	d) 6 & 6

2. What is value of 5%2?

a) 2	b) 3
c) 0	d) 1

Short Answer

1. Define integer arithmetic.

____

____

2. What is called arithmetic expression?

____

____

2.8.Decision Making with Branching and Looping

2.8.1 Branching

2.8.1.1 Introduction

We have a number of situations, where we may have to change the order of execution of statements based on certain conditions, or repeat a group of statements until certain specified conditions are met. This involves a kind of decision making to see whether a particular condition has occurred or not and then direct the computer to execute certain statements accordingly.

Control or decision making statements are

1. if statements
2. switch statement
3. Conditional operator statement

2.8.1.2 Decision Making With If Statement

The **if statement** is a decision making statement and is used to control the flow of execution of statements. The *general form* is

```
if ( test expression)
```

It allows the computer to evaluate the expression first and then, based on the value of expression is 'true' or 'false', it transfers the control to a particular statement (See Fig 2.8.1.1)

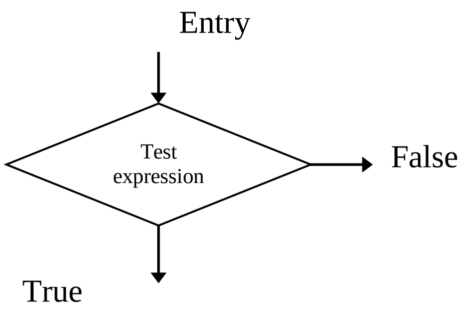


Fig. 2.8.1.1 Two-way branching

The if statement may be implemented in different forms based on the condition to be tested.

1. Simple if statement
2. if …else statement
3. Nested if…else statement
4. else if ladder.

2.8.1.3 Simple If Statement

The *general form* is

```
if ( test expression)
{
statement-block;
}
statement-x;
```

The 'statement-block' may be single or group of statements. If the test expression is true, the statement-block will be executed; otherwise the execution will to the statement-x. (See Fig 2.8.1.2)

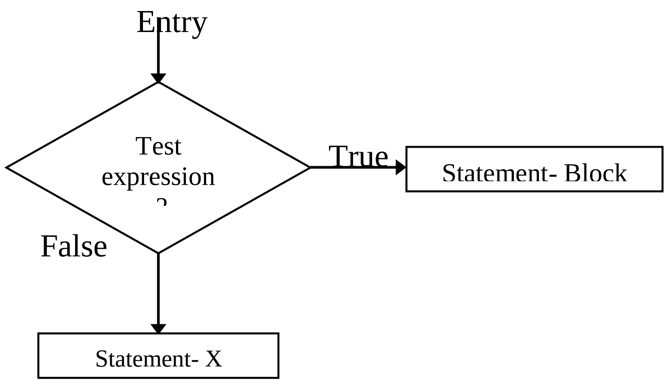


Fig.2.8.1.2 Flowchart of simple if control

Example Program: To find a smallest value among two numbers

```
/*
        Smallest Among Two numbers Using if statement
*/
import java.io.*;
import java.lang.*;
class Smallest
{
         public static void main(String args[])
        {
                int a = 0,b=0;
                int small = 0;
                DataInputStream in = new DataInputStream(System.in);
                System.out.println("\nEnter the two values");
                try
                {
                  a = Integer.parseInt(in.readLine());
                  b = Integer.parseInt(in.readLine());
                }
                catch(Exception e) { }
                small = a;
                if ( small > b )
                  small = b;
           System.out.println("\nSmallest  Among  Two  No.is  :  "  +
small);
        }
}
```

Output Of Program

```
Enter the two values
23
17
Smallest Among Two No.is : 17
```

2.8.1.4 The If…Else Statement

The *general form* is

```
if ( test expression)
{
        statement-block1;
}
else
{
        statement-block2;
}
statement-x;
```

If the test expression is true, the statement-block1 will be executed; otherwise, the statement-block2 will be executed, not both. In both the cases, the control is transferred to the statement-x. (See Fig.2.8.1.3)

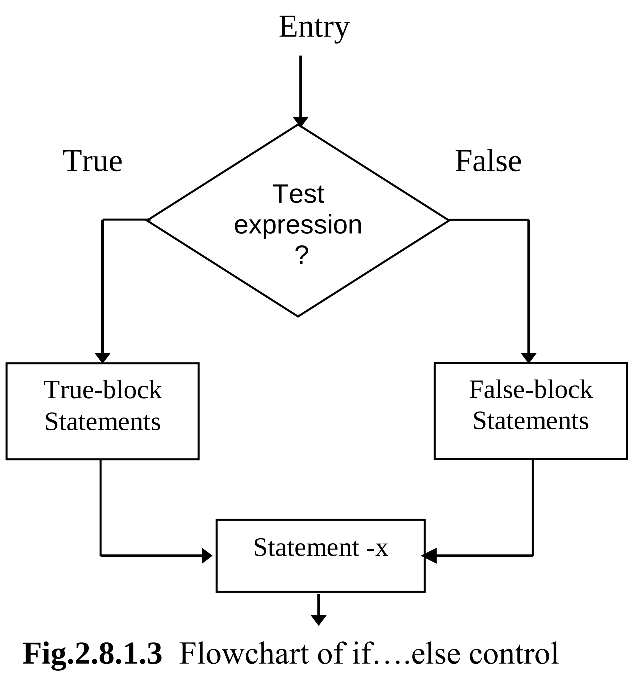


Fig.2.8.1.3 Flowchart of if….else control

Example Program: To Find a smallest value among three numbers.

```
/*
  Smallest Among Three numbers Using if – else statement
*/
import java.io.*;
import java.lang.*;
class Smallest
{
        public static void main(String args[])
         {
                int a = 0,b=0,c=0;
                int small = 0;
                DataInputStream in = new DataInputStream(System.in);
                System.out.println("\nEnter the three values");
                try
                {
                  a = Integer.parseInt(in.readLine());
                  b = Integer.parseInt(in.readLine());
                  c = Integer.parseInt(in.readLine());
                }
                catch(Exception e)
                { }
                small = a;
                if ( small > b )
                     small = b;
                else
                     small = c;
        System.out.println("\nSmallest Among Three No.is : " + small);
        }
}
```

Output Of Program

```
Enter the three values
23
17
56
Smallest Among Three No.is : 17
```

2.8.1.5 Nesting Of If…Else Statement

The *general form* is

```
if ( test expression1)
{
        if ( test expression2)
        {
                statement-block1;
        }
        else
        {
                statement-block2;
        }
}
else
{
        statement-block3;
}
statement-x;
```

If the test expression1 is false, the statement-block3 will be executed; otherwise it continues to perform the second test. If test expression2 is true, the statement-block1 will be executed; other wise the statement-block2 will be executed and then control is transferred to the statement-x. (See Fig.2.8.1.4)

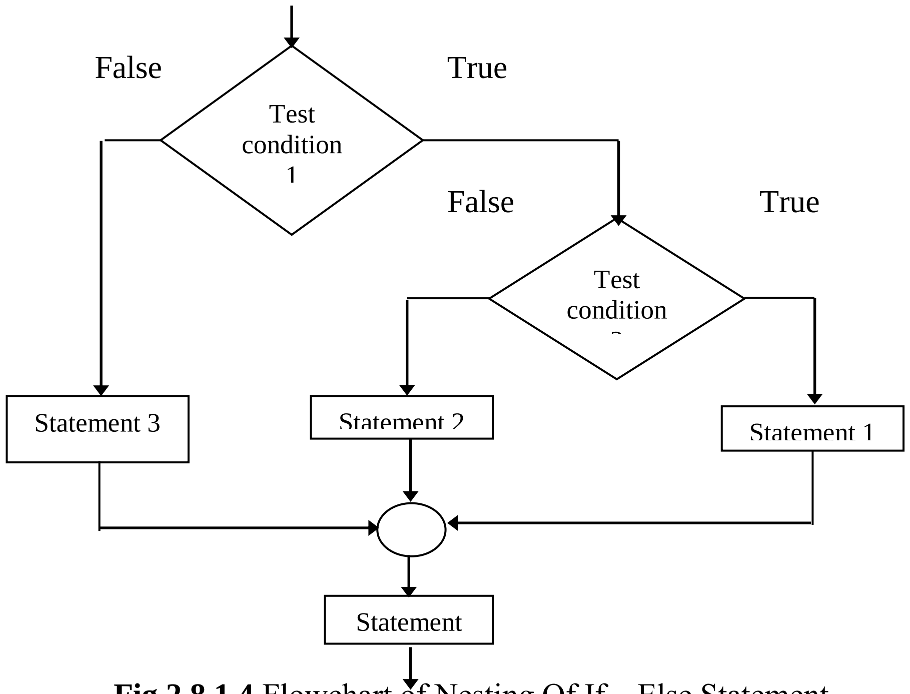


Fig.2.8.1.4 Flowchart of Nesting Of If…Else Statement

Example Program: To Find a largest value among three numbers

```
/*
  Largest Among Three numbers Using nested if-else statement

*/
import java.io.*;
import java.lang.*;
class Largest
{
            public static void main(String args[])
           {
                        int a = 0,b=0,c=0;
                        int large = 0;
                        DataInputStream in = new DataInputStream(System.in);
                        System.out.println("\nEnter the three values");
                        try
                        {
                                    a = Integer.parseInt(in.readLine());
                                    b = Integer.parseInt(in.readLine());
                                   c = Integer.parseInt(in.readLine());
                        }
                        catch(Exception e) { }
                        if ( a > b )
                        {
                                    if (  a > c )
                                    {
                                        large = a;
                                    }
                                    else
                                    {
                                       large = c;
                                    }
                        }
                        else
                        {
                                    if (c > b)
                                    {
                                      large = c;
```

```
                    }
                    else
                    {
                      large = b;
                    }
            }
        System.out.println("\nLargest Among Three Number is : " +
large);
        }
    }
```

Output Of Program

```
Enter the three values
23
17
56

Largest Among Three Number is : 56
```

The If…Else Ladder

The *general form* is

```
if ( test expression1)
        statement-block1;

else if ( test expression2)
        statement-block2;
        ------------------------
        else if (test expression n)
                statement-block-n;

        else
                default -statement;
        statement-x;
```

The below construct is known as the **else if** ladder. The test conditions are evaluated from the top, downwards. As soon as true test-condition is found, the statement associated with it is executed and control is transferred to the statement-x. When all test n conditions become false, then the else containing default-statement will be executed. (See Fig.2.8.1.5)

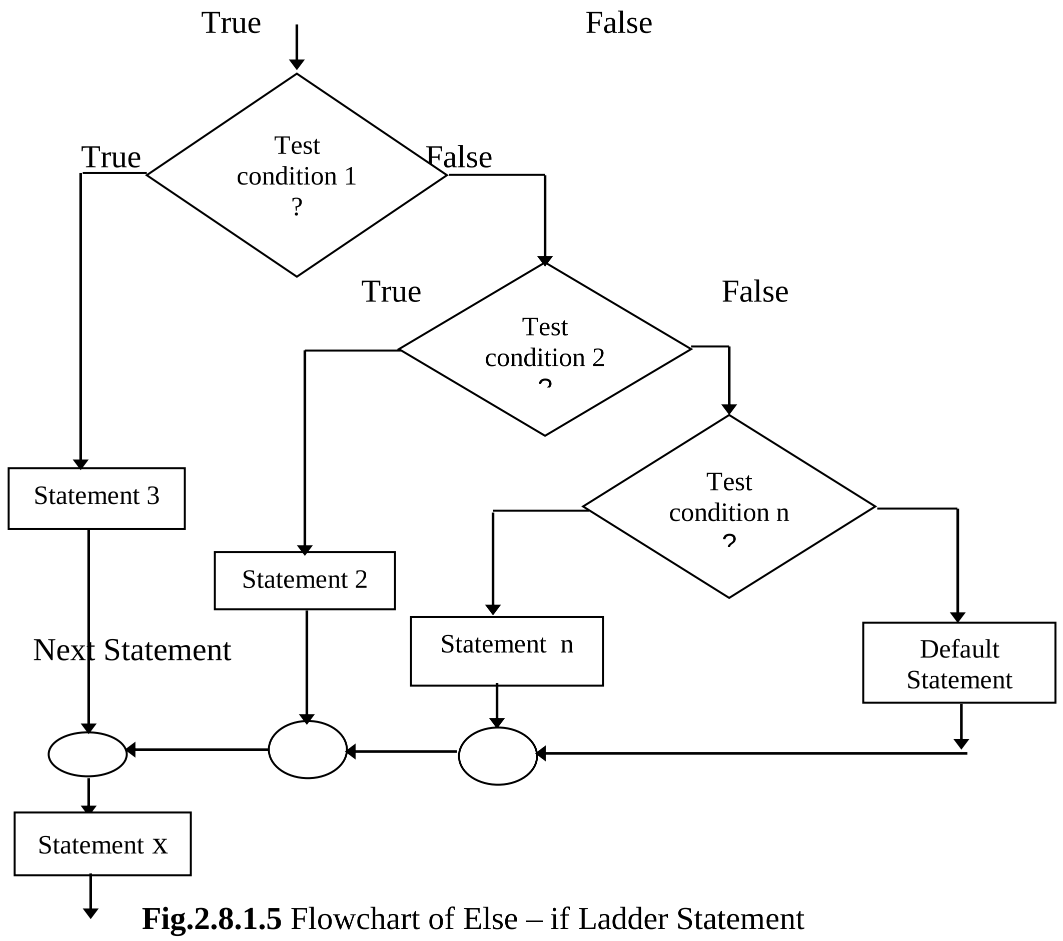


Fig.2.8.1.5 Flowchart of Else – if Ladder Statement

Example Program: To Find a largest value among three numbers

```
/*
Largest Among Three numbers Using else - - if ladder statement

*/
import java.io.*;
import java.lang.*;
class Largest
{
  public static void main(String args[])
  {
        int a = 0,b=0,c=0;
        int large = 0;
        DataInputStream in = new DataInputStream(System.in);
        System.out.println("\nEnter the three values");
        try
```

```
            {
              a = Integer.parseInt(in.readLine());
              b = Integer.parseInt(in.readLine());
             c = Integer.parseInt(in.readLine());
            }
            catch(Exception e) { }
            large = a;
            if ( large < b )
                    large = b;
            else if (large < c)
                    large = c;
            System.out.println("\nLargest Among Three No.is : " + large);
      }
    }
```

Output Of Program

```
Enter the three values
23
17
56
Largest Among Three No.is : 56
```

2.8.1.7 The Switch Statement

The *general form* of the switch statement is

```
switch ( expression)
 {
         case value-1: block-1
                         break;
         case value-2: block-1
                         break;
         ------------
         default:
                         default-block
                         break;
}
statement-x;
```

The expression is an integer expression or characters. value-1, value-2---- are constants or constant expressions and are known as *case labels.*

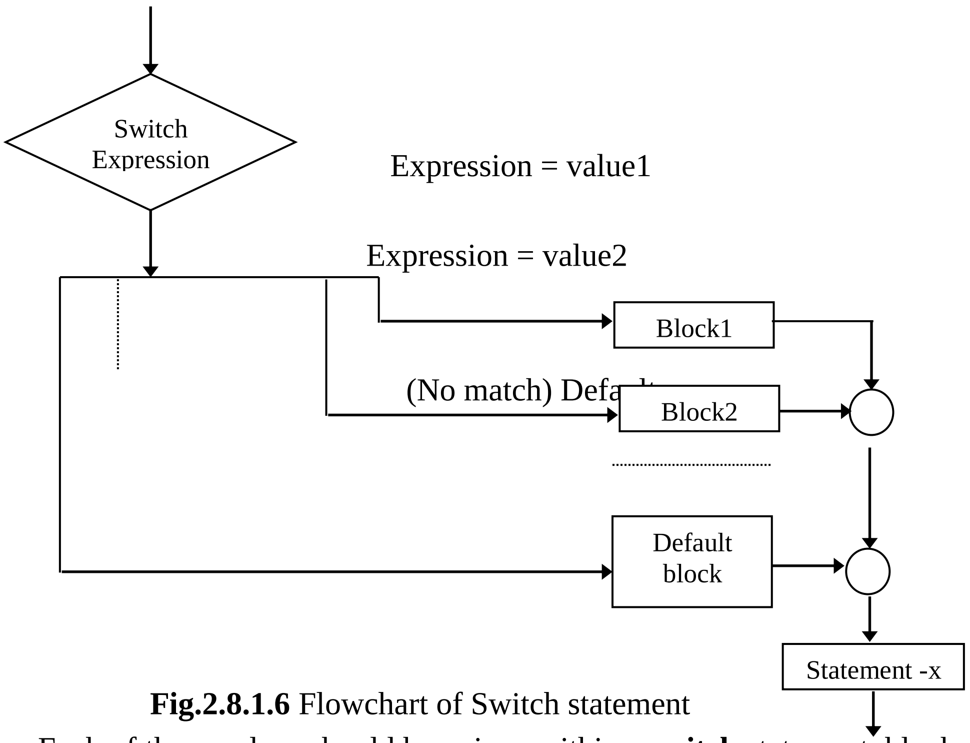


Fig.2.8.1.6 Flowchart of Switch statement

Each of these values should be unique within a **switch** statement. block-1, block-2, ------ are statement lists and may contain zero or more statement. There is no need to put braces around these blocks, case labels end with a colon (:).

When the switch is executed, the value of expression is compared with the values value-1, value-2,………if a case is found then the block of statements that follows the case are executed.

The break statement at the end of each block signals the end of the particular block and control is transferred to the statement-x.

The default is an optional case. When the value of expression not match with any of the case, the default case will be executed. If default statement not present, no action takes place when all matches fail and the control goes to the statement-x. (See Fig.2.8.1.6)

Example Program1: To demonstrate **switch-case** statement

```
/*
    Demonstration of switch-case statement
*/
import java.io.*;
import java.lang.*;
class Switchcase
{
  public static void main(String args[])
  {
        int choice = 0;
        DataInputStream in = new DataInputStream(System.in);
        System.out.println("\nEnter the choice value");
        try
        {
          choice  = Integer.parseInt(in.readLine());
         }
        catch(Exception e) { }
        switch(choice)
        {
        case 1:
                System.out.println("I am in case 1");
                break;
        case 2:
                System.out.println("I am in case 2");
                break;
        case 3:
                System.out.println("I am in case 3");
                break;

default:
                System.out.println("I am in default case");
        }
  }
 }
```

Output Of Program

```
Enter the choice value
2
I am in case 2

Enter the choice value
5
I am in default case
```

Example Program 2: To find whether the given number is even or odd using **switch-** **case** statement

```
/*
```

```
   Program for Even or Odd number using switch-case statement

*/
import java.io.*;
import java.lang.*;
class EvenOdd
{
  public static void main(String args[])
  {
        int n = 0;
        DataInputStream in = new DataInputStream(System.in);
        System.out.println("\nEnter the number n");
        try
        {
          n  = Integer.parseInt(in.readLine());
         }
        catch(Exception e) { }
        switch(n % 2)
        {
        case 0:
                System.out.println("The given number "+n+" is even");
                break;

case 1:
                System.out.println("The given number "+n+" is odd");
                break;
        }
    }
}
```

Output Of Program

```
Enter the number n
35
The given number 35 is odd
```

2.8.2 Looping

2.8.2.1 Introduction

The process of repeatedly executing a block of statements is known as **looping**. The statements in block may be executed any number of times, from zero to infinite number is called an **infinite loop**. The program loop consists of *two segments* are

- ❖ Body of the loop
- ❖ Control statement (tests certain conditions and then directs the repeated execution of the statements in the body of the loop)

A Control structure may be classified either into *two types* are

- ❖ Entry-controlled loop: The control conditions are tested before the start of the loop execution. If conditions are not satisfied, the body of the loop will not be executed. (See Fig.2.8.2.1)
- ❖ Exit-controlled loop: The test is performed at the end of the body of the loop and therefore body is executed unconditionally for the first time. (See Fig.2.8.2.2)

A looping process will follow *four steps*:

1. Setting and initialization of a counter.
2. Execution of the statements in the loop.
3. Test for a specified condition for execution of the loop.
4. Incrementing the counter.

They are three types looping construct are:

1. while construct
2. do construct
3. for construct

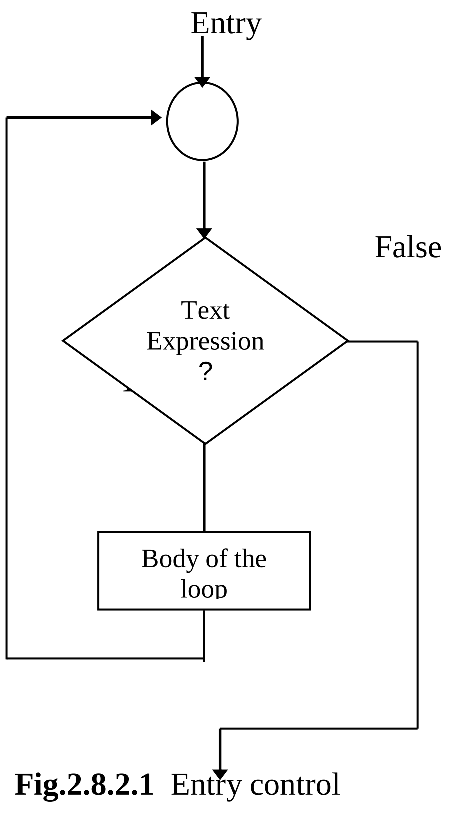


Fig.2.8.2.1 Entry control

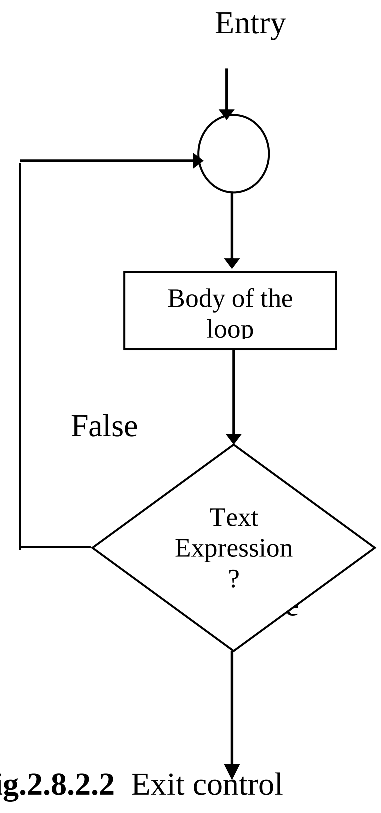


Fig.2.8.2.2 Exit control

2.8.2.2 The While Statement

The *general form* is

```
Initialization;
while ( test condition )
{
        Body of the loop
}
```

The *while* is an entry-controlled loop statement. The test condition is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again. These processes of repeated execution of the body continue until the test condition finally becomes false and the control is transferred out of the loop. On exit, the program continues with the statement immediately after the body of the loop.

Example Program: To reverse the given number using while loop

```
/*
      Reverse of the given number using while loop statement
*/
import java.io.*;
import java.lang.*;
class Reverse
{
   public static void main(String args[])
  {
        int number= 0, digit = 0 , rev =0;
        DataInputStream in = new DataInputStream(System.in);
        System.out.println("\nEnter the number");
        try
        {
              number  = Integer.parseInt(in.readLine());
         }
        catch(Exception e) { }
        while (number != 0)
        {
            digit = number %10;
            rev = rev*10+digit;
            number = number /10;
        }
        System.out.println("\nReverse of the given number is: " + rev);
  }
}
```

Output of Program

```
Enter the number
12345

Reverse of the given number: 54321
```

2.8.2.3 The Do Statement

The *general form* is

```
        Initialization;
        do
        {
                Body of the loop
        }
        while ( test condition )
```

The *do-while* is an exit-controlled loop statement. On *do* statement; the body of the loop will be executed first. At the end of the loop, the test condition in the while statement is evaluated. If condition it true, the program proceed to continues to evaluate the body of the loop once again. This process continues as long as condition is true. When the condition becomes false, the loop will be terminated and control goes to statement after the while statement.

Example Program: To find the summation of 'n' numbers using **do-while** statement

```
/*
  Summation of n numbers using do-while statement
*/
import java.io.*;
import java.lang.*;
class Summation
{
  public static void main(String args[])
  {
        int n = 0, i = 1, sum = 0;;
        DataInputStream in = new DataInputStream(System.in);
        System.out.println("\nEnter the value of  n");
        try
        {
                n  = Integer.parseInt(in.readLine());
```

```
            }
            catch(Exception e) { }
            do
            {
                  sum = sum + i;
                 i = i + 1;
            }
            while (i <= n);
            System.out.println("\nSummation of n numbers is : " +sum);
     }
    }
```

Output Of Program

```
Enter the value of n
11
Summation of n numbers is: 66
```

The For Statement

The execution of the **for** statement is as

1. *Initialization* of the control variables is done using assignment statements.
2. The value of the control variable is tested using the *test condition*.
3. When the body of the loop is executed, the control is transferred back to the **for** statement after evaluating the last statement in the loop. Now, the control variable is incremented using an assignment statement.

The **for** loop is an entry-controlled loop. The *general form* is

```
for (initialization ; test condition ; increment)
{
        Body of the loop
}
```

Example Program: To find the summation of ‘n’ numbers using **for** statement

```
/*
Summation of n numbers using for statement
*/
import java.io.*;
import java.lang.*;
class Summation
{
```

```
                public static void main(String args[])
                {
                        int n = 0, i = 1, sum = 0;;
                        DataInputStream in = new DataInputStream(System.in);
                        System.out.println("\nEnter the value of  n");
                        try
                        {
                            n  = Integer.parseInt(in.readLine());
                         }
                        catch(Exception e) { }
                        for(i = 1 ; i <= n ; i++)
                        {
                                 sum = sum + i;
                        }
                        System.out.println("\nSummation  of  n  numbers  is  :  "
+sum);
                }
        }
```

Output Of Program

```
Enter the value of n
10
Summation of n numbers is: 55
```

Additional features of for loop

The **for** loop has several capabilities that are not found in other loop constructs. **For example** *more than one variable* can be *initialize*d at a time in the **for** statement.

```
p = 1;
for (n = 0 ; n<17; ++n)
```

can be rewritten as

```
for (p =1, n = 0 ; n<17; ++n)
```

Increment section may also have more than one part. **For example**

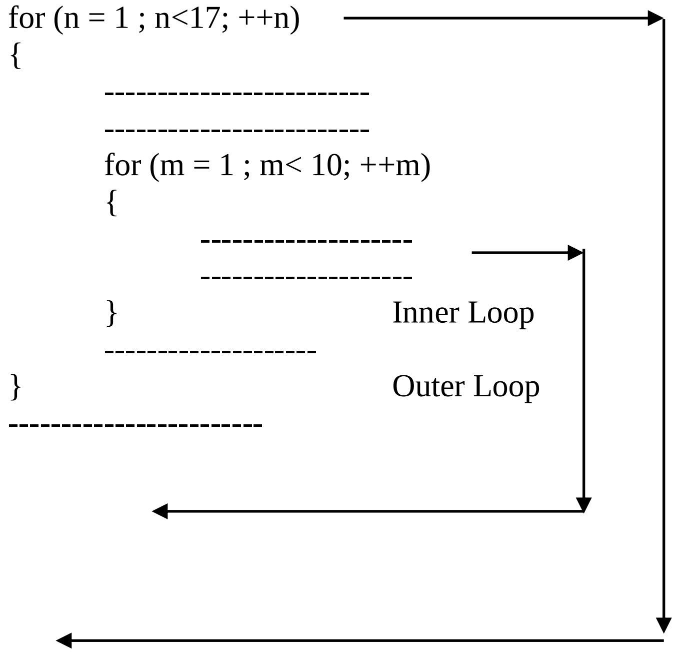
```
for (n = 0, m = 50  ; n<17; ++n, --m)
```

The *test condition* may have any compound relation and testing need not be limited only to the control variable.

Nesting of for loops

Nesting of loops, that is one **for** statement within another **for** statement, is allowed in java. **For example**

```
----------------------
----------------------
```



```
for (n = 1 ; n<17; ++n)
{
        -------------------------
        -------------------------
        for (m = 1 ; m< 10; ++m)
        {
                --------------------
                --------------------
        }                       Inner Loop
        --------------------
}                               Outer Loop
------------------------
```

2.8.2.5 Jump In Loops

Java permits a jump from one statement to the end or beginning of a loop as well as jump out of a loop.

Jumping Out of a Loop

An early exit from a loop can be accomplished by using **break** statement. The *general form* is

```
break     [label] ;
```

The break statement can be used within while, do, for loops. When the break statement is encountered inside a loop, the loop is immediately exited. When the loops are nested, the break would exit from containing it.

Example Program: To find the sum of positive numbers using **break** statement

```
        /*      Program for sum of positive number using break statement
*/
        import java.io.*;
        import java.lang.*;
        class SumPositiveNumber
        {
                public static void main(String args[])
                {
                        int n = 0,sum =0, i;
                        DataInputStream in = new DataInputStream(System.in);
                System.out.println("\nEnter the 5 numbers ");
                for(i=1;i<=5;i++)
                {
                try
                {
                  n  = Integer.parseInt(in.readLine());
                 }
                catch(Exception e) { }
                if (n<0)
                        break;
                else
                        sum = sum + n;
                }
                System.out.println("\nSum    of    positive    numbers    is
        :"+sum);
        }
}
```

Output Of Program

```
Enter the 5 numbers
12
34
56
-78

Sum of positive numbers is :102
```

Skipping a Part of a Loop

During the loop operations, it may be necessary to skip a part of the body of the loop under certain conditions by using **continue** statement. The *general form* is

continue [label];

In **while** and **do** loops, continue causes the control go to directly test condition and then to continue the iteration process. In the case of **for** loop, the *increment* section of the loop is executed before the *test condition* is evaluated.

Example Program: To find the sum of positive numbers using **continue** statement

```
/*
  Program for sum of positive number using continue statement
```

```
*/
import java.io.*;
import java.lang.*;
class SumPositiveNumber
{
        public static void main(String args[])
        {
            int n = 0,sum =0, i;
            DataInputStream in = new DataInputStream(System.in);
            System.out.println("\nEnter the 5 numbers ");
            for(i=1;i<=5;i++)
            {
                 try
                 {
                   n  = Integer.parseInt(in.readLine());
                  }
                 catch(Exception e) { }
                 if (n<0)
                     continue;
                 else
                      sum = sum + n;
            }
            System.out.println("\nSum of positive numbers is
:"+sum);
        }
}
```

Output Of Program

```
Enter the 5 numbers
12
34
56
-78
10
Sum of positive numbers is :112
```

2.8.2.6 Labeled Loops

In Java, we can give a label to a block of statement. A label is any valid Java variable name. To give label to a loop, place it before the loop with colon at the end.

Example:

```
Loop1: for (…………)
        {
                ------------------
                ------------------
        }
                ------------------
```

A block of statement can be labeled as

```
Block1:         {
                        ----------------
                        ----------------
        Block2:         {
                        ----------------
                        -----------------
                        }
                        -----------------
                }
```

We want to jump outside a nested loops or to continue a loop that is outside the current loop by using labeled break and labeled continue statement.

Self Assessment Questions

Fill in the blank

1. if statements returns either__________ or __________.

2.One loop within a loop is called __________________.

True / False

1. When the break statement is encountered inside a loop, the loop is immediately exited

2. for loop is an exit controlled loop statement.

Multiple Choices

1. Which is one of exit controlled loop?

a) do-while	b) while loop
c) for loop	d) none of the above

2. The segment of a program is

```
x=1; sum=0;
while ( x<=5)
{
        sum=sum+x;
        x++;
}
```

what is the value of sum?

a) 13 b) 14 c) 10 d) 15

Short Answer

1. List out branching control statement.

____

____

2. Define infinite loop

____

____

Summary

In this unit we have learned about the basic concepts of object oriented programming and fundamental concepts of Java programming. We have also discussed about how to use variable, expression , operators and various decision making branching and looping statements.

Unit Questions

1. Explain basic concepts of Object oriented programming.
2. What are the benefits of OOP? List a few areas of applications of OOP technology.
3. What is the difference between C++ and Java.
4. Describe the structure of a typical Java program.
5. What is a Token? List the various types of Tokens supported by Java.
6. What is Separators? Describe the various separators used in Java.
7. Explain Java virtual machine?
8. What are command line arguments? How are they useful?
9. Explain features of Java.
10. What is a constant? List the various types of Constants
11. List the eight basic data types used in Java. Explain with examples.
12. Define operators. Explain operators with examples?
13. Explain different types of expressions with example?
14. Explain decision-making and branching statements with example?
15. Write a Java program to find the largest number among three numbers

2.11 Answer for Self Assessment Questions

Answer 2.3.6

Fill in the blank

1.encapsulation 2. objects

True / False

1. False 2. True

Multiple Choice

1. b

Short Answer

1. Inheritance is the process by which objects of one class acquire the properties of objects of another class.

2. Real-time systems, Simulation and modeling, Object-oriented databases, Hypertext, hypermedia and expertext , AI and expert systems etc.,

Answer 2.4.10

Fill in the blank

1. Multithreaded.   2. Java compiler and Java interpreter

True / False

1. False   2. True

Multiple Choice

1. b   2. c

Short Answer

1. Complied and Interpreted, Platform-Independent and Portable, Object-Oriented, Distributed, Robust and Secure, Familiar, Simple and Small etc.,

2. World Wide Web (WWW) is an open-ended information retrieval system designed to be used in the Internet' environment.

Answer 2.5.11

Fill in the blank

1. Stand alone applications & Web applets   2. Virtual machine code

True / False

1. True 2. True

Multiple Choice

1. d   2. b

Short Answer

1. Applets are small Java programs developed for Internet applications.

2. Command line arguments are parameters that are supplied to the application program at time of invoking it for execution.

Answer 2.6.9

Fill in the blank

1. Integer constant   2. Widening or promotion

True / False

1. False   2. True

Multiple Choice

1. d   2. c

Short Answer

1. A variable is an identifier that denotes a storage location used to store a data value. A variable may take different values at different times during the execution of the program.

2.Variables declared and used inside methods are called local variables.

Answer 2.7.16

Fill in the blank

1. Operators. 2. Math class and java.lang package

True / False

1. False 2. True

Multiple Choice

1. a 2. d

Short Answer

1. When both the operands in single arithmetic expression such as a + b are integers, the expressions is called an integer expression, and the operation is called integer arithmetic.

2. An arithmetic expression is a combination of variables, constants, and operators

Answer 2.8.3

Fill in the blank

1. true or false 2. Nested loop

True / False

1. True 2. False

Multiple Choice

1. a 2. d

Short Answer

1. if statements, switch statement, Conditional operator statement

2. The statements in block may be executed any number of times, from zero to infinite number is called an infinite loop.

NOTES

UNIT – III

3.1 Introduction

Java is a **true object-oriented language** and therefore the underlying structure of all Java programs is classes. Any thing you to represent in a Java program must be encapsulated in a class that defines the state and behaviour of the basic program known as object.

In Java, **classes** provide a method for packing together a group of related data items and functions that work on them. Calling a specific method by using object. It provides a template for an object and behaves like a basic data type such as **int**.

It is important to understand how the **fields** and **methods** are defined in a class and how they are used to build the basic concepts such as encapsulation, inheritance and polymorphism.

An **array** is a group of contiguous or related data items that share a common name. A particular value in array can be identified by using index number or subscript in brackets after the array name.

Example:

salary[10]

represents the salary of the 10th employee. While the complete set of values is referred to as an array.

In this unit we shall discuss in detail how arrays are created and used. We shall also discuss two related concepts, namely **strings a**nd **vectors**.

3.2 Objectives

After studying this lesson, you should be able to:

- ❖ Describe the major elements of a classes and objects.
- ❖ Describe the various features of a classes and objects.
- ❖ Describe about various kind of methods used in classes and its usage.
- ❖ Understand about various types arrays and array related concepts strings and vector.
- ❖ Understand about, how to making the inheritance and interface concept and its usage.

3.3 Classes, Objects And Methods

3.3.1 Introduction

Java is a true object-oriented language and therefore the underlying structure of all java programs is classes. Any thing you to represent in a Java program must be encapsulated in a class that defines the state and behaviour of the basic program known as object.

In Java, classes provide a method for packing together a group of related data items and functions that work on them. Calling a specific method by using object. It provides a template for an object and behaves like a basic data type such as **int**. It is important to understand how the fields and methods are defined in a class and how they are used to build the basic concepts such as encapsulation, inheritance and polymorphism

3.3.2 Defining A Class

The *general form* of a class definition is:

```
class classname [ extends superclassname ]
{
        [  fields declaration; ]
        [ methods declaration; ]
}
```

Inside the square brackets is optional. For **example**

```
class  empty
{    }
```

Because the body is empty, this class does not contain any properties and cannot do anything. **classname** and **superclassname** are any valid identifier. The keyword **extends** indicates that the properties of the **superclassname** class are executed to the **classname** class (is known as inheritance).

3.3.3 Field declaration

By placing data fields inside the body of the class definition are called *instance variables* because they are created whenever an object of the class is instantiated.

Example:

```
class     Sample
{
        int a, b;
}
```

The class **Sample** contains two integer type instance variables (are also called as *member variables*)

3.3.4 Methods Declaration

A class with only data fields has no life. Methods are declared inside the body of the class but immediately after the declaration of instance variables. The *general form* of a method declaration is

```
type methodname (parameter-list)
{
        method-body;
}
```

Method declarations have *four* basic parts:

- ❖ The name of the method (methodname) is a valid identifier.

- ❖ The type of the value the method returns (type). This could be simple data type such as **int** as well as any type. It could even be **void** type, if the method does not return any value.
- ❖ A list of parameters (parameter-list) is always enclosed in parentheses. This list contains variable names and types of all values we want to give to the method as input.

  Example:

  ```
  (int m, float a, float b) // Three parameters
  ( )                       //Empty list
  ```
- ❖ The body of the method actually describes the operations to be performed on the data.

Example:

```
 class Summation
{
        int a , b;  // combined declaration
        void getdata(int x, int y)
        {
                a = x;
                b = y;
        }
        int sum( )      // declaration of another method
        {
                int c = a + b;
        }
}
```

3.3.5 Creating Objects

Creating an object is also referred to as **instantiating an object**. Object in Java created using **new operators**. The new operator creates an object of the specified class and returns a reference to that object.

Example

```
Summation sum1;              // declare the object
sum1 = new Summation( );     // instantiate the object
```

The first statement declares a variable to hold the object reference and the second statement actually assigns the object reference to the variable. The variable sum1 is now an object of the Summation class (See Fig.3.5)

Both lines can be combined into one line as

```
Summation sum1= new Summation( );
```

The Summation( ) is the default constructor of the class. We can create any number of object of Summation class.

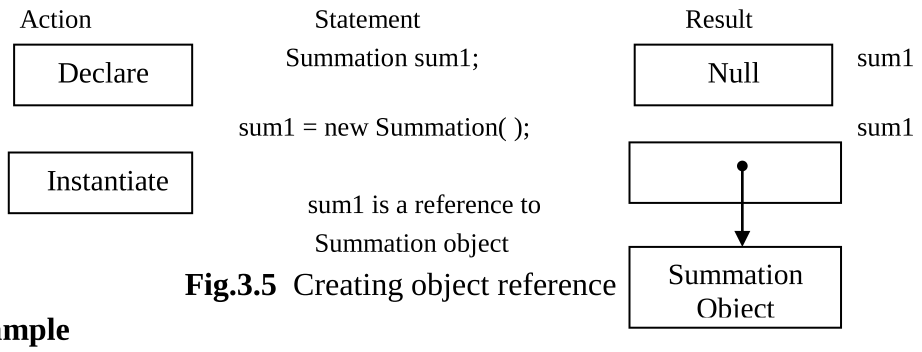


Fig.3.5 Creating object reference

For **Example**

Summation sum1 = new Summation ( );

Summation sum2 = new Summation ( );

and so on.

In this case each object has its own copy of the instance variables of its class. this means that any changes to the variables of one object have no effect on the variables of another.

It can also possible to create two or more references to the same object.

Example:

Summation s1 = new Summation( );

Summation s2 = s1;

Both s1 and s2 refer to the same object.

3.3.6 Accessing Class Members

In an outside the class, we cannot access the instance variables and the methods directly. For this, we must use the concerned object and *dot* operator as

objectname.variablename = value;
objectname.methodname(parameter-list);

where *objectname* - is the name of the object

variablename - that	is the name of the instance variable inside object we wish to access
methodname -	is the method that we wish to call
parameter-list -	is a comma separated list of 'actual values" that must match in type and number with the parameter list of the methodname declared in the class.

For example

First Method:

The instance variables of the Summation class may be accessed and assigned values as

sum1.a = 10;

```
        sum1.b = 20;
        sum2.a = 30;
        sum2.b = 40;
```

Two objects sum1 and sum2 store different values as

sum1.a	10	sum2.a	30
sum1.b	20	sum2.b	40

Second Method:

In this case the method getdata can be used to assign the value to variable a and b.

```
        Summation  sum1 = new Summation( );
        sum1.getdata( 10, 20);//calling the method using the object
```

Now we can find summation of two values using the following ways.

Example

```
        int  s = sum1.a + sum1.b;
        or
        int s = sum1.sum( );
```

Example Program: To illustrate **classes** and **objects**

```
    class Summation
    {
        int a , b;  // combined declaration
        void getdata(int x, int y) // definition of method
        {
            a = x;
            b = y;
        }
        int sum( )        // definition of another method
        {
            int c = a + b;
            return c;
        }
    }
    class SumTwoValues             //class with main method
    {
        public static void main(String args[ ])
        {
            int  s1 , s2;
```

```
                    Summation sum1 = new Summation ( );       //creating
objects
                    Summation sum2 = new Summation ( );
                    sum1.a = 10;                    // accessing variables
                    sum1.b = 20;
                    s1 = sum1.a + sum1.b;
                    sum2.getdata(30, 40); // accessing methods
                    s2 = sum2.sum( );
                    System.out.println(" Sum1  =  " + s1);
                    System.out.println(" Sum2  =  " + s2);
            }
    }
```

Output of Program

```
     Sum1  =  30
     Sum2  =  70
```

3.3.7 Constructors

Java supports a special type of method is called a *constructor* that enables an object to initialize itself when it is created.

Rules for forming the constructors are

- ❖ Constructors have the same name as the class.
- ❖ They do not specify a return type, not even void. This is because they return the instance of the class itself.

Let us consider our Summation class again. We can now replace the **getdata** method by a constructor method as

```
    class Summation
    {
            int a , b;  // combined declaration
            Summation(int x, int y) // definition constructor
            {
                    a = x;
                    b = y;
            }
            int sum( )
            {
                    return (a + b);
            }
    }
```

Example Program: To illustrate **classes** and **objects**

```
        class Summation
        {
                int a , b;  // combined declaration
                Summation(int x, int y) // defining constructor
                {
                        a = x;
                        b = y;
                }
                int sum( )
                {
                        return (a + b);
                }
        }
        class SumTwoValues          //class with main method
        {
            public static void main(String args[ ])
            {
                Summation   sum1   =   new   Summation  (10,  20);  //calling
constructor
                int s1 = sum1.sum( );
                System.out.println(" Sum1  =  " + s1);
            }
        }
```

Output of Program

```
        Sum1  =  30
```

3.3.8 Methods Overloading

In Java, it is possible to create methods that have same name, but different parameter lists and different definitions is called **method overloading**.

When we call a method in an object, java matches up the method name first and then the number and type of parameters to decide which one of the definitions to execute is known as **polymorphism**.

Example

```
        class Summation
        {
                int a , b;  // combined declaration
                Summation(int x, int y)         // constructor1 with two parameter
                {
```

```
                a = x;
                b = y;
        }
        Summation(int x)                // constructor2 with one parameter
        {
                a = b = x;
        }
        int sum( )
        {
                return (a + b);
        }
}
```

Here, we are overloading the constructor method **Summation( )**. An object representing a sum two different values will be created as

```
Summation sum1 = new Summation(10 ,20);          //          using constructor1
```

On the other hand if equal value we want to sum , then we may create the object as

```
Summation sum2 = new Summation(10);              //          using constructor2
```

3.3.9 Static Members

We want to define a member that is common to all the objects accessed without using a particular object. That is, the member belongs to the class as a whole rather than the objects created from the class. Such members can be *defined as*

```
        Static  int count;
        Static int mix(int x, int y);
```

The members are declared as **static** are called as *static members*. The static variables and static methods are often referred to as *class variable* and *class methods*.

Static methods have several restrictions:

- ❖ They can only call other **static** methods
- ❖ They can only access **static** data.
- ❖ They cannot refer to **this** or **super** in any way

Example Program:

```
/*  Program for defining and using static members */
class mathop
{
        static float add(float  x, float  y)
```

```
                {
                        return( x + y);
                }
                 static float sub(float  x, float  y)
                {
                       return(x - y);
                 }
        }
        class mathmain
        {
           public static void main(String args[])
          {
                float  a = mathop.add(4.0f,5.0f);
                float  b = mathop.sub(4.5f,2.2f);
                System.out.println("a = "+a);
                System.out.println("b = "+b);
         }
        }
```

Output of Program

```
        a = 9.0
        b = 2.3
```

3.3.10 Nesting Of Methods

A method can be called by using only its name in another method of the same class is known as *nesting of methods.*

Example Program: To find smallest among two numbers using nesting method

```
        class nesting
        {
                 int a, b;
                nesting()
                {
                        a = 10;  b = 20;
                }
                int small()
                {
                        if(a < b)
                        {
                           return(a);
                }
                else
                {
                   return(b);
                }
            }
            void display()
            {
                int  s = small();                    // calling a method
                System.out.println("The smallest number is "+ s );
            }
        }
        class NestingMethod
        {
            public static void main(String args[])
           {
                   nesting n = new nesting();
                   n.display();
            }
        }
```

Output Of Program

```
        The smallest number is 10
```

The class nesting defines one constructor and two methods, namely **small( )** and **display( )**. The method **display( )** calls the method **small( )** to determine the smallest of the two numbers and then displays the result.

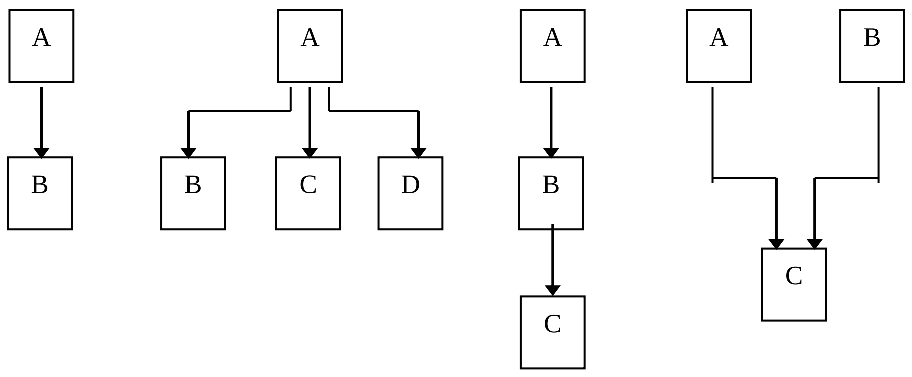
3.3.11 Inheritance :Extending A Class

The mechanism of deriving a new class (*subclass* or *derived class* or *child class*) from an old class (*base* or *super* or *parent class*) is called **inheritance**.

The inheritance allows subclasses to inherit all the variables and methods of their parent classes. Inheritance may take *different forms*:

- ❖ Single inheritance (only one super class)
- ❖ Multiple inheritance (several super classes)
- ❖ Hierarchical inheritance (one super class, many subclasses)
- ❖ Multilevel inheritance (derived from derived class)

These form inheritance are shown in Fig.3.3.1. Java does not directly implement multiple inheritance. However, this concept is implemented using a secondary inheritance path in the form of *interfaces*.



a)Single Inheritance b) Hierarchical Inheritance c) Multilevel Inheritance d) Multiple Inheritance

Fig.3.3.1 Forms of Inheritance

Defining a Subclass

A subclass is defined as

```
class subclassname extends superclassname
{
        variables declaration;
        method declaration;
}
```

The keyword extends specifies that the properties of the *superclassname* are extended to the *subclassname*. The subclass will now contain its own variables and methods as well as the variables and methods of the superclass.

Subclass Constructor

A subclass constructor is used to construct both the subclass and the superclass. The subclass constructor uses the keyword **super** to invoke the constructor method of the superclass.

The keyword **super** is used with the following *condition*s.

- **super** may only be used with in a subclass constructor method.
- The call to superclass constructor must appear as first statement within the subclass constructor
- The parameters in the **super** call must match the order and type of the instance variable declared in the superclass

Multilevel Inheritance

The class **A** serves as base class for derived class **B** which in turn serves as a base class for the derived class **C.** The chain ABC is known as **inheritance path** as shown in Fig.3.3.1(c).

Example:

```
class A
{
-----------------
-----------------
}
class B extends A      //first level
{
----------------
----------------
}
class C extends B      //second level
{
----------------
----------------
}
```

Hierarchical Inheritance

Many Programming problems can be cast into a hierarchy where many others below the level share certain features of one level as shown in Fig.3.3.1 (b)

3.3.12 Overriding Methods

We want an object to respond to the same method but have different behavior when that method is called. That means, we should override the method defined in the superclass. This is possible by defining a method in the subclass that has the same name, same arguments and same return type as a method in the superclass. Then, when that method called, the method defined in the subclass is invoked and executed instead of the one in the superclass. This is known as **overriding.**

3.3.13 Final Variables And Methods

We wish to prevent the subclasses from overriding the members of the superclass, we can declare them as final using the keyword **final** as a modifier.

Examples:

```
Final int SIZE =100;
Final void showstatus ( ) {--------}
```

Making a *method final* ensures that the functionality defined in this method will never be altered in any way. The value of final variable can never be changed. *Final variables*, behave like class variables and they do not take any space on individual objects of the class.

3.3.14 Final Classes

A class that cannot be sub-classed is called a **final class**. Declaring a class final prevents any unwanted extensions to the class.

Examples:

```
Final class Aclass {----------}
Final class Bclass extends Someclass {-----------}
```

Any attempt to inherit these classes will cause an error.

3.3.15 Finalizer Methods

In Java run-time is an automatic garbage collecting system. It automatically free ups the memory resources used by the objects. But object may hold other non-object resources such a file descriptors or window system fonts. The garbage collector cannot free these resources. In order to free these resources we must use a **finalize()** method and it can be added to any class.

3.3.16 Abstract Methods And Classes

Abstract method is a method that must always be redefined in a subclass, thus making overriding compulsory. This is done by using the modifier keyword **abstract** in the method definition.

Example:

```
abstract class shape
{
        --------------
        --------------
        abstract void  draw( );
        --------------
        --------------
}
```

When a class contains one or more abstract methods, it should also declared abstract as in the example.

While using abstract classes, we must satisfy the following conditions:

- ❖ We cannot use abstract classes to instantiate objects directly.
- ❖ The abstract methods of an abstract class must be defined in its subclass.
- ❖ We cannot declare abstract constructors or abstract static methods.

3.3.17 Visibility Control

In inheritance inherit all the members of a class by a subclass using the keyword **extends**. The variables and methods of class are visible everywhere in the program we want to restrict the access to certain variables and methods from outside the class. We can achieve this in Java by *visibility modifiers or access modifiers* to the instance variables and methods. Java provides three types modifiers: **public, private** and **protected.** Table 3.3.1 shows the visibility provided by various modifiers.

Table 3.3.1 Visibility of Field in a Class

Access Modifier → / Access location ↓	Public	Protected	Friendly (default)	Private protected	Private
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other package	Yes	Yes	No	Yes	No
Non-Subclass in other package	Yes	No	No	No	No

3.3.18 Self Assessment Questions

Fill in the blank

1. A class is a ______________ data type.

2. Methods in class can have the same name but different parameter is called ______________________.

3. The static variables and static methods are otherwise called as _________________ and _____________________.

True / False

1. In java, the data items are called fields and the functions are called methods.
2. Constructor name must be class name
3. The value of the final variable can be changed during the execution
4. Abstract classes can create the objects

Multiple Choices

1. In java, instance of class is called as

   a) Object b) fields c) method d) None of the above

2. Static method, which can only access

   a) private data b) pubic data c) protected data d) static data

3. Private variables in a class, which are accessed by

a) Same class b) Subclass in same package

c) Other classes in same package d) Non-subclasses in other packages

Short Answer

1.How can you inherit all the members of a class by a subclass?

____

____

2. What is mean by final class?

____

____

3. What is the purpose of super key word?

____

____

3.4 Arrays, String And Vectors

3.4.1 Creating An Array

An **array** is a group of related data items that share a common name. An array of value can be accessed by index or subscript in brackets after array name. **For example**, a[5] represents the 5th element in an array.

Creation of array has *three steps* are

1. Declaring the array
2. Creating memory locations
3. Putting values into the memory locations.

Declaration of arrays

Arrays in Java may be declared in *two forms*:

```
1.      type arrayname[ ];
2.      type [ ] arrayname;
```

Examples:

```
int a[];
float    b[];
int []   c;
float [ ]  d;
```

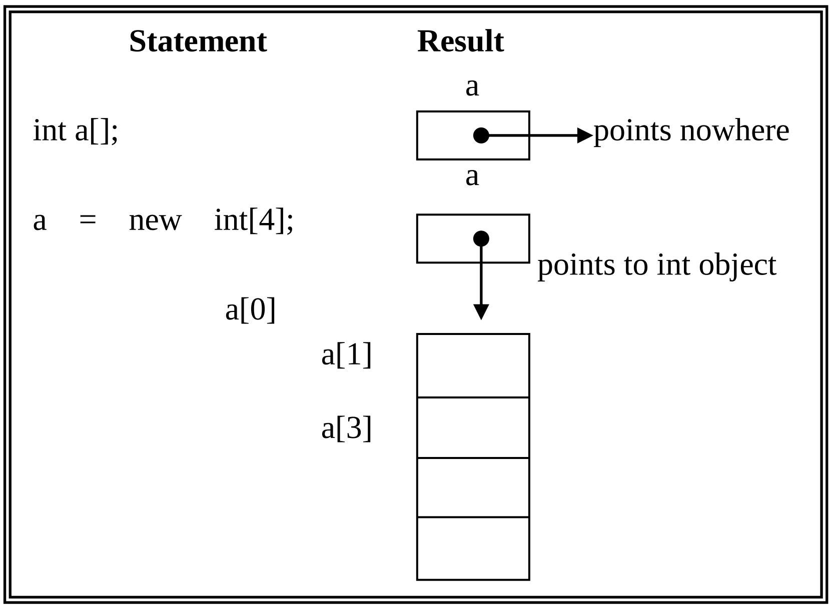


Fig.3.4.1 Creation of an array in memory

Creating memory locations

Java allows us to create arrays using **new** operator as

```
array = new   type[size];
```

Examples:

a = new int[4]; //create *4* memory location for integer array *a*

b = new float[10]; //create *10* memory location for float array *b*

It is also possible to *combine the two steps* - declaration and creation – into *one* as

```
int      a [ ]  = new    int[4];
float    b [ ]  =new    float[10];
```

Fig.3.4.1 above illustrates creation of an array in memory.

Initialization of arrays

To put values into array created is known as **initialization.** This is done using array subscripts as

```
arrayname[subscript]  =  value;
```

Example:

```
a[0]    =       28;
-----------------
------------------
a[3]    =       57;
```

Java create arrays starting with subscript of 0 and ends with a value one less than the *size* specified.

We can also initialize arrays automatically when they are declared, as

```
type arrayname[ ] = { list of values };
```

The array initializer is a list of values separated by commas and surrounded by curly braces .

Example:

```
int a[ ] = { 10, 20,30,40};
```

Loops may be used to initialize large size arrays **Example**:

```
----------------
----------------
for(int   i = 0;    i<10;     i++)
{
        a[i] = i;
}
-----------------
```

Array length

In java, all array store the allocated size in a variable named **length**. We can obtain the length of the array **a** using **a.length**.

Example:

```
int   aSize  =   a.length;
```

The subscript of an array can be integer constants, integer variables like I, or expression that yield integers.

3.4.2 One Dimensional Arrays

A list of items can be given one variable name using only one subscript and such a variable is called a **single-subscripted** variable or **one-dimensional** array. **For example**, if we want to represent a set of four numbers, by array variable **a** then we create the variable **a** as

```
int       a [ ]  = new    int[4];
```

The different way of initialization of value to array variable **a** as shown in section 3.4.1.

Example Program

```
/*  Sorting the n given number in Descending order */
class descending
{
        public static void main(String args[])
        {
                int  a[ ] = {10 , 45 , 100, 37, 25};
```

```
                        int  i, j, t;
                        for (i=0; i<a.length; i++)
                        {
                                for (j=i+1; j<a.length; j++)
                                {
                                        if (a[i] <= a[j] )
                                        {
                                                t = a[i];
                                                a[i] = a[j];
                                                a[j] = t;
                                        }
                                }
                        }
                        System.out.println(" Descending Order");
                        for (i=0;i<a.length;i++)
                        {
                                System.out.println(a[i]);
                        }
                }
        }
```

Output Of Program

```
        Descending Order
        100
        45
        37
        25
        10
```

3.4.3 Two Dimensional Arrays

We may create a two-dimensional array as

```
int    table[ ] [ ];
table    =   new int[2][3];
            or
int   table[ ] [ ] =   new   int[2][3];
```

This creates a table that can store 6 integer values, three across and two down. A two-dimensional array may be initialized by following their declaration with a list of initial values enclosed in braces.

For example

```
    int   table[2] [3]  =  {0,0,0,1,1,1};          //initialize  the  elements  row  by
row
                                   or
    int   table[2] [3]  =   {{0,0,0},{1,1,1}}; //separate the element of each row
by                                                 //braces
                           or
   int   table[2] [3]  =  {
                               {0,0,0},
                               {1,1,1}
                           };
```

Example Program: Program for Addition of Two Matrixes using Two-Dimensional Array

```
/* Matrix Addition Using Two Dimensional Array   */
 import java.io.*;
 import java.lang.*;
 class MatrixAdd
 {
 public static void main(String args[])
 {
        int a[ ][ ] = new int[30][30];
        int b[ ][ ] = new int[30][30];
        int c[ ][ ] = new int[30][30];
        int n = 0, ch = 0, i = 0, j = 0;
        DataInputStream m = new DataInputStream(System.in);
        try
        {
               System.out.println("Enter the order of matrix");
               n = Integer.parseInt(m.readLine());
                System.out.println("\nEnter the A matrix");
                for(i=0; i<n; i++)
               {
                  for(j=0; j<n; j++)
                 {
                     a[i][j] = Integer.parseInt(m.readLine());
                  }
               }
                System.out.println("\nEnter the B matrix");
```

```
            for(i=0; i<n; i++)
           {
               for(j=0; j<n; j++)
              {
                   b[i][j] = Integer.parseInt(m.readLine());
               }
           }
 }
 catch (Exception e)
{    }
System.out.println("\nThe A matrix");
 for(i=0; i<n; i++)
{
    for(j=0; j<n; j++)
   {
         System.out.print(a[i][j]+ "    ");
    }
    System.out.println();
}
System.out.println("\nThe B matrix");
for(i=0; i<n; i++)
{
    for(j=0; j<n; j++)
   {
         System.out.print(b[i][j] + "    ");
    }
    System.out.println();
}
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
   {
          c[i][j] = a[i][j] + b[i][j];
    }
}
System.out.println("\nThe Resultant Matrix is");
for(i=0;i<n;i++)
{
```

```
                for(j=0;j<n;j++)
               {
                    System.out.print(c[i][j]+ "    ");
                }
                System.out.println();
            }
        }
       }
```

Output Of Program

```
Enter the order of matrix
2
Enter the A matrix
2
2
2
2
Enter the B matrix
2
2
2
2
The A matrix
2    2
2    2
The B matrix
2    2
2    2
The Resultant Matrix is
4    4
4    4
```

Variable size arrays

Java treats multidimensional array as "**arrays of arrays**". It is possible to declare a two-dimensional array as

```
int x[ ][ ]  =  new int [3] [ ];
x[0]     =   new int[2];
x[1]     =   new int[4];
x[2]     =   new int[3];
```

These create a two-dimensional array as having different lengths for each row as shown in Fig.3.4.2.

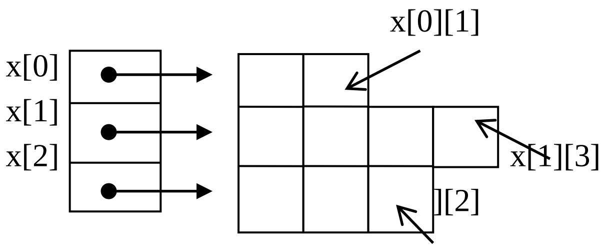


Fig.3.4.2. Variable size arrays

3.4.4 Strings

The string represents a sequence of characters in Java by using a character array. **Example:**

```
char chararray[ ] =  new char[3];
chararray[ ]      = 'H';
chararray[ ]      = 'a';
chararray[ ]      = 'i';
```

In Java strings are class objects and implanted using *two classes*, namely **String** and **StringBuffer**. A Java string is an instantiated object of the **String** class. Strings may be declared and created as

String** stringName**;

*stringName = **new String**("string");*

Example:

```
String firstName;
firstName = new String ("Harshinni");
```

Like arrays, it is possible to get the length of string using the length method of the string class.

```
int m = firstName.length( );
```

Java string can be concatenated using + operator.

Example:

String fullName = name1 + name2; // name1 and name2 containing string constants

```
String city       = "New" + "Delhi";
```

System.out.println(firstName+"Sundar");

String Arrays

We can also create and use arrays that contain strings. The statement

String itemarray[ ] = new String[2]; // create string array with 3 string constants

We can assign the strings to the **itemarray** element by element using 3 different statements or using **for** loop.

String Methods

The **String** class defines a number of methods that allow us to accomplish a variety of string manipulation tasks as shown in Table 3.4.1. **String** class creates strings of *fixed length*.

Table 3.4.1 Commonly Used String Methods

Method Call	*Task performed*
s2 =s1.toLowerCase;	Converts the string s1 to all lowercase
s2 =s1.toUpperCase;	Converts the string s1 to all uppercase
s2 =s1.replace(‘x’ , ‘y’);	Replace all occurrences of x with y
s2 =s1.trim( );	Remove white spaces at the beginning and end of the string s1
s1.equals(s2)	Return ‘true’ if s1 is equal to s2
s1.equalsIgnoreCase(s2)	Return ‘true’ if s1 = s2, ignoring the case of characters
s1.legth( )	Gives the length of s1
s1.CharAt( )	Gives nth character of s1
s1.compareTo(s2)	Return negative if s1<s2, positive if s1>s2 or zero if s1=s2
s1.concat(s2)	Concatenates s1 and s2
s1.subtring(n)	Gives substring starting from nth character
s1.subtring(n, m)	Gives substring starting from nth character up to mth
String.ValueOf(p)	Creates a string object of the parameter p
p.toString( )	Creates a string representation object p
s1.indexOf(‘x’)	Gives the position of the first occurrence of ‘x’ in the string s1
s1.indexOf(‘x’)	Gives the position of ‘x’ that occurs after nth position in string s1
String.ValueOf(Variable)	Convert the parameter value to string representation

Example Program 1: Program to sort given strings in Alphabetical order

```
/*   Sorting the Strings in Alphabetical Order Using Arrays*/
import java.io.*;
import java.lang.*;
class StringOrdering
{
  String name[] = new String[100];
  int n;
  String temp = null;
  DataInputStream in = new DataInputStream(System.in);
  void getString()
  {
   try
```

```
     {
       System.out.println("   Enter the value of n");
       n = Integer.parseInt(in.readLine());
       System.out.println("   Enter the "+ n + " Strings");
       for(int i = 0; i<n; i++)
       {
        name[i] = in.readLine();
       }
     }
     catch(Exception e) { }
    }
    void sort()
    {
     for(int i =0; i <n; i++)
     {
      for(int j= i+1; j<n; j++)
      if(name[j].compareTo(name[i]) < 0)
      {
              // swap the strings
              temp = name[i];
              name[i]= name[j];
              name[j]= temp;
      }
     }
    }
    void putString()
    {
     System.out.println("\n Sorted Order is ");
     for(int i = 0; i<n; i++)
     {
      System.out.println( "    "+name[i]);
     }
    }
  }
 class Sorting
{
       public static void main(String args[])
       {
            StringOrdering so = new StringOrdering();
            so.getString();
            so.sort();
            so.putString();
        }
    }
```

Output Of Program

```
Enter the value of n
5
Enter the 5 Strings
Java
C++
C
Cobol
Basic
Sorted Order is
Basic
C
C++
Cobol
Java
```

Example Program 2: Program for demonstration of string function

```
class strin
{
        public static void main(String args[])
        {
         String s1, s2, s3;
        s1 = "computer";
        s2 = s1.toUpperCase();
        System.out.println("Convert lowercase to uppercase letters"+s2);
         s1 = "COMPUTER";
         s2 = s1.toLowerCase();
        System.out.println("Convert uppercase to lowercase letters"+s2);
        s1= "computer";
        int length = s1.length();
```

```
            System.out.println("Length of string constant in s1 object is" +
    length);
            s1 = "computer";
            s2 = " science";
            s3 = s1.concat(s2);
            System.out.println("Concatenation of two strings is"+s3);
            char c = s2.charAt(2);
            System.out.println("Extract character from s2 object is " +c);
            if (s1.equals(s2))
            {
               System.out.println("Two strings are equals");
            }
            else
            {
               System.out.println("Two strings are not equals");
             }
            }
    }
```

Output Of Program

```
    Convert lowercase to uppercase letters COMPUTER
    Convert uppercase to lowercase letters computer
    Length of string constant in s1 object is 8
    concatenation of two strings iscomputer science
    Extract character from s2 object is c
    Two strings are not equals
```

StringBuffer Class

StringBuffer creates strings of *flexible length* that can be modified both length and content. We can insert characters and substring in the middle of a string, or append another string to the end. Table 3.4.2 lists some of the methods that are used in string manipulations.

Table 3.4.2 Commonly Used StringBuffer Methods

Method Call	*Task performed*
s1.setCharAt(n, 'x')	Modifies the nth character to x
s1.append(s2)	Appends the string s2 to s1 at the end
s1.insert(n, s2)	Inserts the string s2 at the position n of the string s1
s1.setLength(n)	Sets the length of the string s1 to n. If n < s1.length ( ) s1 is truncated.If n>s1.length ( ) zeros are added to s1.

Example Program: Program for Manipulation of Strings

```
class strbuf
{
        public static void main(String args[])
        {
                StringBuffer s1=new StringBuffer("Computer ");
                StringBuffer s2=new StringBuffer(" Department");
                s1.append(s2);
                System.out.println(s1);
                s2.setCharAt(3,'u');
                System.out.println(s2);
                s1.insert(9,"Science ");
                System.out.println(s1);
        }
}
```

Output Of Program

```
Computer Department
Deuartment
Computer Science Department
```

3.4.5 Vectors

For achieving the concepts of *variable arguments to methods* in Java through the use of the **Vector** class contained in the j**ava.util** package. This class can be used to create a generic dynamic array known as *vector* that can hold *objects of any type* and *any number.* Vector are created like arrays as

```
Vector intVect  = new Vector( );     // declaring without size
Vector list     =  new Vector( );    // declaring with size
```

Vectors possess a number of ***advantages*** over arrays.

1. It is convenient to use vectors to store objects.
2. A vector can be used to store a list of objects that may vary in size.
3. We can add and delete objects from the list as and when required.

A major condition in using vectors is that we cannot directly store simple data type in a vector; we can only store objects. Therefore, we need to convert simple type to objects by using **wrapper classes.** The vector class supports a number of methods that can be used to manipulate the vectors created as listed in Table 3.4.3.

Table 3.4.3 Commonly Used Vector Methods

Method Call	*Task performed*
list.addElement(item)	Adds the item specified to the list at the end
list.elementAt(10)	Gives the name of the 10th object
list.size( )	Gives the number of the object present
list.removeElement(item)	Removes the specified item from list
list.removeElementAt(n) of	Removes the item from in the nth position the list
list.removeAllElements( )	Removes all the elements in the list
list.copyInto(array)	Copies all items from list to array
list.insertElementAt(item, n)	Inserts the item at nth position

Example Program: Program for working with vectors and arrays

```
import java.io.*;
import java.util.*;
class vect
{
        public static void main(String args[]) throws IOException
        {
                Vector vec=new Vector();
                vec.addElement("Rose");
                vec.addElement("Lotus");
                vec.addElement("Lily");
                vec.addElement("Jasmine");
                int len=vec.size();
                String str[]=new String[len];
                vec.copyInto(str);
                System.out.println("Result is:");
                for(int i=0;i<len;i++)
                {
                        System.out.println(str[i]);
                }
        }
}
```

Output Of Program

```
Result is:
Rose
Lotus
Lily
```

Jasmine

3.4.6 Wrapper Classes

Vectors cannot handle primitive data type **like int, float, long, char,** and **double**. Primitive data types may be converted into object by using the wrapper classes contained in the **java.lang** package are listed in the following tables.

Table 3.4.4 Wrapper Classes for Converting Simple Types

Simple type	*Wrapper Class*
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long

Table 3.4.5 Converting Primitive Numbers to Object Numbers Using Constructor Methods

Constructor Calling	*Conversion Action*
Integer IntVal = new Integer (i);	Primitive integer to Integer object
Float FloatVal = new Float(f);	Primitive float to Float object
Double DoubleVal = new Double(d);	Primitive double to Double object
Long LongVal = new Long(l);	Primitive long to Long object

Note: I, f, d and l are primitive data values denoting int, float, double and long data types. They may constant or variables.

Table 3.4.5 Converting Object Numbers to Primitive Numbers Using typeValue( ) Methods

Method Calling	*Conversion Action*
int i = IntVal.intValue( )	Object to primitive integer
float f = floatVal.floatValue( )	Object to primitive float
double d = doubleVal.doubleValue( )	Object to primitive double
long i = longVal.longValue( )	Object to primitive long

Table 3.4.6 Converting Numbers to String Using String( ) Methods

Method Calling	*Conversion Action*
Str = Integer.toString(i)	Primitive integer to string
Str = Float.toString(f)	Primitive float to string
Str = Double.toString(d)	Primitive double to string
Str = Long.toString(l)	Primitive long to string

Table 3.4.6 Converting String Object to Numeric objects
Using the Static Method ValueOf()

Method Calling	*Conversion Action*
IntValue = Integer.ValueOf(str)	Converts string to integer object
FloatVal= Float..ValueOf(str)	Converts string to float object
DoubleVal = Double.ValueOf(str)	Converts string to double object
LongVal = Long.ValueOf(str)	Converts string to long object

Table 3.4.6 Converting Numeric Strings to Primitive Numbers
Using Parsing Methods

Method Calling	*Conversion Action*
int i = Integer.parseInt(str)	Converts string to primitive integer
long l = Long.parseLong(str)	Converts string to primitive long

Note: parseInt() and parseLong() methods throws a NumberFormatException if the value of the str does not represent an integer.

3.4.7 Self Assessment Questions

Fill in the blank

1. String class which supports __________ length in their object.

2. Primitive data types may be converted into object by using the ____________ classes

True / False

1. Index value of arrays can begin with the number 1.

2. Arrays in Java may be declared in data type arrayname[], (or) datatype[] arrayname;

3. trim() function in string class can not remove the spaces of both sides in a string constant.

Multiple Choice

1.Vector class is in package

a) java.long.*; b) java.awt.*;
c) java.io.*; d) java.util.*;

2. StringBuffer class creates strings of

a) fixed length b) flexible length
c) constant length d) none of the above

Short Answer

1. Define arrays.

2.fine strings.

3.5 Interface: Multiple Inheritance

3.5.1 Introduction

Classes in Java cannot have more than one superclass. For instance a definition like

```
Class A extends B extends C
{
        --------------
        --------------
}
```

is not permitted in Java. Java provides alternate approaches known as **interfaces** to support the concept of multiple inheritance. Although a Java class cannot be a subclass of more than one superclass, it can **implement** more than one interface.

3.5.2 Defining Interfaces

The *general form* of an interface definition is:

```
interface interfaceName
{
variable declaration;
methods declaration;
}
```

Where interface – is the keyword

interfaceName – is any valid Java variable

Variables are declared as

```
Static final type VariableName = Value;
```

Methods declaration will contain only a list of methods without any body statements.

Example:

```
return-type methodName1( parameter_list);
```

3.5.3 Extending Interfaces

Like classes, interface can also extend. The new interface will inherit all the members of the superinterface. The *general form* of an interface is

```
interface name2 extends name1
{
body of name2
}
```

3.5.4 Implementing Interfaces

Interfaces are used as “superclasses whose properties are inherited by classes. It is therefore necessary to create a class that inherits the given interface. The *general form* is

```
class classname implements interfacename
{
body of classname
}
```

Here the class *classname* “implements” the interface *interfacename*. The more *general form* is

```
class classname extends superclass
implements interfacename1,interfacename1, ………
{
        body of classname
}
```

Here the class can *extend* another class while *implementing* interfaces. When more than one interface, they are separated by a comma. The implementation of interfaces can take various forms as shown in Fig. 3.5.1.

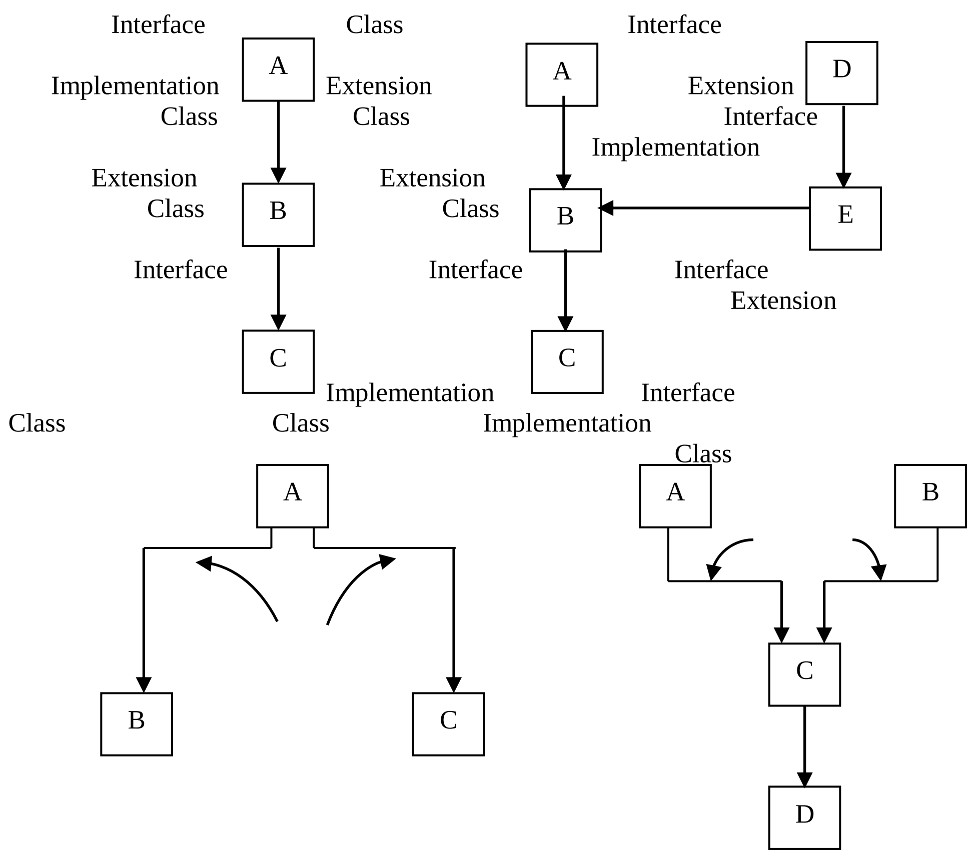


Fig. 3.5.1 Various forms of interface implementation

3.5.5 Accessing Interface Variables

Interfaces can be used to declare a set of constants that can be used in different classes. Interfaces do not contains methods; there is no need to worry about implementing any methods. The constant values will be available to any class that implements the interface.

Example Program: Program for Student Information Using Multiple Inheritance

```
    /*  Student Information Using Multiple Inheritance */
import java.io.*;
import java.lang.*;
class Student
{
    String name = new String(); String rno = new String();
    String course = new String();
    int m1 =0,m2=0;
    DataInputStream in = new DataInputStream(System.in);
    void getdata()
    {
       try
       {
        System.out.println("Enter a Register No.");
        rno = in.readLine();
        System.out.println("Enter a Name ");
        name = in.readLine();
        System.out.println("Enter a Course");
        course = in.readLine();
        System.out.println("Enter a Mark1");
        m1 = Integer.parseInt(in.readLine());
        System.out.println("Enter a Mark2");
        m2 = Integer.parseInt(in.readLine());
         }
      catch(Exception e) { }
      }
      void disdata()
      {
        System.out.println("Students Information");
        System.out.println("~~~~~~~~~~~~~~~~~~~~");
        System.out.println("Register Number :"+ rno);
        System.out.println("          Name :"+ name);
        System.out.println("         Class :"+ course);
        System.out.println("         Mark1 :"+ m1);
        System.out.println("         Mark2 :"+ m2);
        }
 }
 interface Sports
{
       void getsportwt();
       void dissportwt();
}
class Result1 extends Student implements Sports
{
      DataInputStream m = new DataInputStream(System.in);
      int sportwt = 0;
      float percentage = 0.0f;
      int total =0;
      String result = new String();
```

```
String grade = new String();
public void getsportwt()
{
 try
 {
  System.out.println("Enter the Sports mark");
  sportwt = Integer.parseInt(m.readLine());
 }
 catch(Exception e) { }
}
public void dissportwt()
{
  System.out.println("    Sports Mark :"+sportwt);
}
void calculate()
{
 total =m1+m2+sportwt;
 percentage = (float) total/3;
 if(m1>=50 && m2>=50 && sportwt>=50)
 {
  result ="Pass";
  if(percentage >=80)
    grade ="Distinction";
  else if (percentage>=60 && percentage<80)
    grade ="First Class";
  else if (percentage>=50 && percentage<60)
    grade = "Second Class";
 }
 else
 {
  result ="Fail";
  grade ="Nil";
 }
}
void display()
{
 calculate();
 disdata();
```

```
         dissportwt();
         System.out.println("         Total :"+total);
         System.out.println("       Average :"+percentage);
         System.out.println("        Result :"+result);
         System.out.println("         Grade :"+grade);
        }
  }
  class StuDetails
 {
        public static void main(String args[])
        {
         Result1 r = new Result1();
         System.out.println(" Student Information Using Multiple Inheritance");
        r.getdata();
         r.getsportwt();
         r.display();
        }
}
```

Output Of Program

```
Student Information Using Multiple Inheritance
Enter a Register No.
102
Enter a Name
 Harshinni
 Enter a Course
MCA
Enter a Mark1
70
Enter a Mark2
 80
Enter a Mark3
78
Enter the Sports mark
80
Students Information
~~~~~~~~~~~~~~~~~~~~
Register Number :102
            Name :Harshinni
```

```
          Class :MCA
          Mark1 :70
          Mark2 :80
    Sports Mark :80
            Total :230
          Average :76.6
           Result :Pass
           Grade :First Class
```

3.5.6 Self Assessment Questions

Fill in the Blank

1. The mechanism of deriving a new class from an old class is called as _____________.
2. ______________ is used to implement multiple inheritance.
3. __________________ keyword is used to inherits properties from interface.

True / False

1. Interface concept is not used in developing multiple inheritances.
2. Interface cannot extends the other interface.

Multiple Choices

1. One derived class which access properties from one base class is called as

a) Multiple inheritance	b) Multilevel inheritance
c) Hierarchical inheritance	d) Single inheritance

2. Interfaces are used as

a) super classes of other classes	b) sub classes of other classes
c) super and sub classes	c) none of the above

Short Answer

1. Define multiple inheritance.

3.6 Summary

Classes, object and methods are basic elements used in Java programming. In this we have discussed about how to define a class , how to create objects, how to add methods to classes, how to extend class and related application programs.

Three important data structures namely arrays strings and vectors. In this we have discussed about what is an array, how they are used, how to handle strings, how to use String and StringBuffer classes, what is vector and its usage and how are wrapper classes useful.

Java does not support multiple inheritance. Java provide alternate way of implementing this concepts using interface. In interface we have discussed about how to design an interface, to extend one interface by other, to inherit an interface and implement the multiple inheritance using interface.

3.7 Unit Questions

1. What is class? How are objects created from a class?
2. What is a constructor? What are its special properties?
3. Compare method overloading and method overloading
4. Discuss the different levels of access protection available in Java
5. What is an array? How is it different from vector?
6. What are the methods of string & string buffer classes?
7. Write a program, which will read a string and write it in the alphabetical order. For example the word STRING should be written as GNIRTS
8. Write a program to implement multiple inheritance using interface

3.8 Answer for Self Assessment Questions

Answer 3.3.18

Fill in the blank

1. User-defined data type 2. method overloading

3. class variables and class methods

True / False

1. True 2. True 3. False 4. False

Multiple Choice

1. a 2. d 3. a

Short Answer

1.In inheritance inherit all the members of a class by a subclass using extends keyword

2. A class that cannot be sub-classed is called a final class.

3. The subclass constructor uses the keyword **super** to invoke the constructor method of the superclass.

Answer 3.4.7

Fill in the blank

1. fixed 2. wrapper classes

True / False

1. False 2. True 3. False

Multiple Choice

1. a 2. b

Short Answer

1.An **array** is a group of related data items that share a common name. An array of value can be accessed by index or subscript in brackets after array name.

2. The string represents a sequence of characters

Answer 3.5.6

Fill in the blank

1. Inheritance	2. Interface	3. implements

True / False

1. False	2. False	3.True

Multiple Choice

1. d	2. a

Short Answer

1.One derived class which access properties from more than one base class is called as Multiple inheritance.

NOTES

UNIT - IV

4.1 Introduction

To use the classes and or interfaces from other program without physically copying them into the program is done by Java package. Actually packages are the group of classes and interfaces. The grouping is done according to functionality. Packages are containers for the classes.

Java package can be classified into *two types* are

- ❖ System packages or API (Application Program Interface) packages.
- ❖ User-Defined packages

We shall consider both the types packages in this unit and illustrate how to use them in your program.

A thread is small unit of program that is used to perform a particular task. Thus a process can contain multiple threads. Each thread is used to perform a specific task, which is executed simultaneously with other threads. In this unit we shall see how threading concept works in the Java program.

In this unit also describe about how to handling the various types error occurs in a Java program.

4.2 Objectives

After studying this lesson, you should be able to:

- ❖ Describe the major elements of a package.
- ❖ Describe the various features of packages.
- ❖ Describe about various kind of thread and thread methods used in Java programs.
- ❖ Understand about various types errors and related error handling concepts.

4.3 Packages: Putting Classes Together

4.3.1 Introduction

To use the classes and or interfaces from other program without physically copying them into the program is done by Java package. Actually packages are the group of classes and interfaces. The grouping is done according to functionality. Packages are containers for the classes.

Java package can be classified into *two types* are

- ❖ System packages or API (Application Program Interface) packages.
- ❖ User-Defined packages.

Benefits of packages:

- ❖ Packages are used to organize to classes into smaller units and make it easy to locate and use the appropriate file.

- The classes contained in the packages of other programs can be easily reused.
- It is possible to create classes with the same name in different packages. Thus it avoids naming conflicts.
- Packages hide their classes from other programs and other packages.
- Packages are used to protect the classes, data and methods in a larger way than on a class to class basis

4.3.2 Java API Packages

Java API contains a large number of classes grouped into different packages according to their functionality. The various API packages and their description are shown in the Table 4.3.1.

Table 4.3.1 Java System Packages

Package name	*Description*
java.lang	It contains the classes that support language. This package is the default package and they are automatically imported. It includes classes of primitive data types, strings, threads and exceptions, main functions.
java.io	This package consists of classes that are used for input and output operations.
java.util	This package consists of utility classes such as vectors, random numbers, date etc.,
java.awt	This package is useful to create GUI (Graphical User Interface) applications.
java.applet	This package consists of classes for creating, implementing and executing applets.
java.net	This package contains classes for networking.

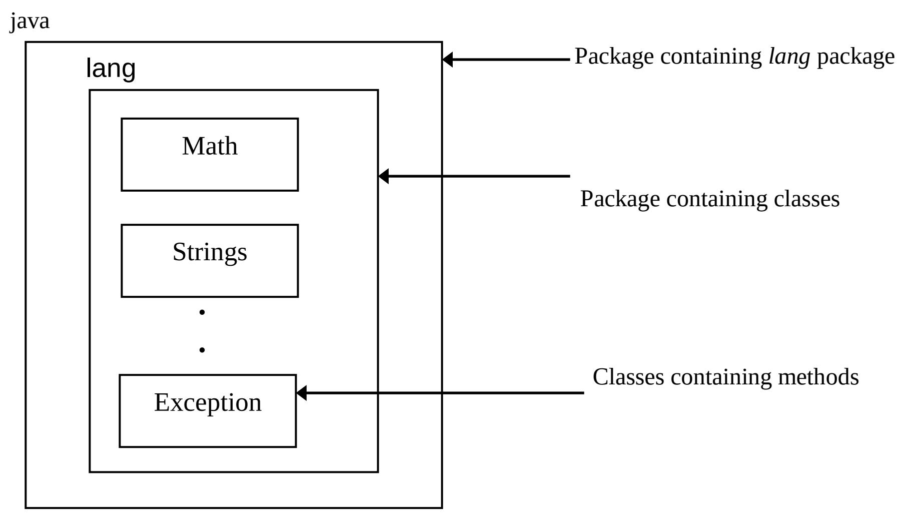
4.3.3 Using System Packages

The System packages are organized in a hierarchical manner as shown in Fig.4.3.1. The main package in the Java is “**java**”. This “**java** “ packages contains several packages. Again it in turn contains packages and classes.

Example:

java.lang.;*

Fig.4.3.1. Hierarchical Structure of *java.lang.**.


java
lang
Math
Strings
Exception
Package containing *lang* package
Package containing classes
Classes containing methods

Classes are stored in the package can be accessed in *two* ways.

- ❖ By specifying the **full path** of the required class.

  This is done by using the package name containing the class and then appending the class name to it using dot (**.**) operator.

  Examples:

  java.lang.Math

  java.applet.Applet

- ❖ By using **import** keyword.

  This statement must appear at the top of the program, before any class declaration. This is used to use a same class in number of places in the program or to use number of classes contained in the package. The *general form* is:

  import packagename.classname;

  import packagename.*;

  Examples:

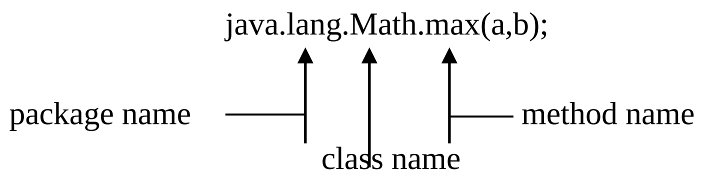
```
import java.lang.Math;
import java.awt.*;
```

4.3.4 Naming Conventions

Packages can be named using the standard java naming rules. The rules may be followed (not a compulsory) while naming the packages are

- ❖ The first letter of the packages name should be a lowercase letters. It is used to distinguish a class name from the package name. Usually, class name begins with uppercase letter.

  Example:



- ❖ Every package name should be unique. In some case, giving the same name to different packages is unavoidable. In this, case domain name is added with the package name.

  Example:

  slm.periyar.exampackage

  cbe.bharati.exampackage

4.3.5 Creating Packages

The *general form* for creating user-defined package is

```
package packagename;        // package declaration
public class Classname      // class definition
{
-----------------           // body of the class
-----------------
}
```

where

package , public & class - are the keywords

packagename & classname - are the any valid user-defined package name &class name

The *steps are used in the creation of a package* are:

- ❖ *Declar*e the package at the *beginning of a program* by using the form

```
package packagename;
```

- ❖ Define the class that is to be put in the package and declare it public.
- ❖ Create a subdirectory under the main directory by using the DOS command as

```
maindirectory > md subdirectory
```

Where

md – is a *make directory* (DOS command) used to create directory *subdirectory* – is the name of the directory, which is same as packagename.

- ❖ Save the program as the *classname.java* file in the subdirectory created.

- Compile the program by using **javac**. This creates *classname.class* file in the subdirectory.

4.3.6 Accessing A Packages

The *import statement* is used to access a particular class in a package or all classes in a package. The *general forms* is

```
import  package1 [.package2][.package3].Classname ;
        or
import  packagename.*;
```

Where

import – is the keyword is used to import classes from package.

package1, package2, package3 & Classname - are the any valid user-defined package name & class name.

“ * “ - All the classes contained in a particular package can be accessed by using *** “** in the **import** statement .

4.3.7 Using A Packages

Let us now consider some simple programs that will use classes from other packages. The below program shows a package name **package1** containing a single class **add.**

```
package package1;
public class add
{
    public void sum( )
    {
         int a = 10; int b = 20;
         int c = a  + b;
          System.out.println("Sum of two number. is" + c);
     }
}
```

This source file should be named **add.java** and stored in the *subdirectory* ***package1***. Now compile this java file. The resultant **add.class** will be stored in the same subdirectory.

```
import package1.add;
class PackageTest1
{
         public static void main(String args[ ])
         {
                add  a = new add( );
                a.sum( );
        }
}
```

The above program shows that imports the class **add** from **package1**. The source file should be saved as PackageTest1.java and then compiled. The source file and the compiled file would be saved in the directory of which package1 was a subdirectory. Now we can run the program and obtain the results.

4.3.8 Adding Class To A Package

It is very simple and easy to add a new class to an existing package. The following steps are used for adding a new class to an existing package.

- ❖ Place the package statement before the class definition like

  package existingpackagename;

- ❖ Define the new class and make it as public.
- ❖ Save the above program as the *classname.java* file in the subdirectory created.
- ❖ Compile the program by using **javac**. This creates *classname.class* file in the subdirectory.

Now, the package will contain new class also. The statement **import** *packagename.** will import all the classes.

For example

The package **p1** contains one public class **ClassA**.

```
package p1;
public ClassA
{
        // body of ClassA
}
```

Suppose we want to add another class **ClassB** to this package. This can be done as

1. Define the class and make it public.
2. Place the package statement

   package p1;

   before the class definition as

   ```
   package p1;
   public ClassB
   {
       // body of ClassB
   }
   ```

3. Store this as **ClassB.java** file under the directory **p1**.
4. Compile **ClassB.java** file. This will create a **ClassB.class** file and place it in the directory p1.

Now, the package p1 will contain the both the classes **ClassA** and **ClassB**. A statement like

```
import p1.*;
```

will import both of them.

4.3.9 Hiding Classes

It is possible to hide some of classes from outside of the packages by declaring the classes as "*not public*". The classes declaring as "*not public*" can be used only in the same package, not possible to access the outside of the packages is called as **Hiding classes**.

For Example

```
package p1;
public class ClassA             // public class, available outside
{
        //body of ClassA
}
class ClassB            // not public class, hidden
{
        // body of ClassB
}
```

Here the class **ClassB**, which is not declared **public** is hidden from outside of the package **p1.** This class can be seen and used only by other classes in the same package.

4.3.10 Self Assessment Questions

Fill in the Blank

1. Java package has two packages such as ______________ and ______________.
2. The ____________statement is used to access a particular class in a package or all classes in a package.

True / False

1. Package contains classes and interfaces.
2. In Java, we cannot add a new class to an existing package

Multiple Choices

1. Private variables in a class which are accessed by

   a) Same class b) Subclass in same package

   c) Other classes in same package c) Non-subclasses in other packages

2. The following character is used to import all the classes & interfaces from the package

   a) + b) * c) / d) **

Short Answer

1. What is the use of Java packages?

____

____

2. What is meant by hiding classes?

____

____

Multithreaded Programming

4.3.11 Introduction

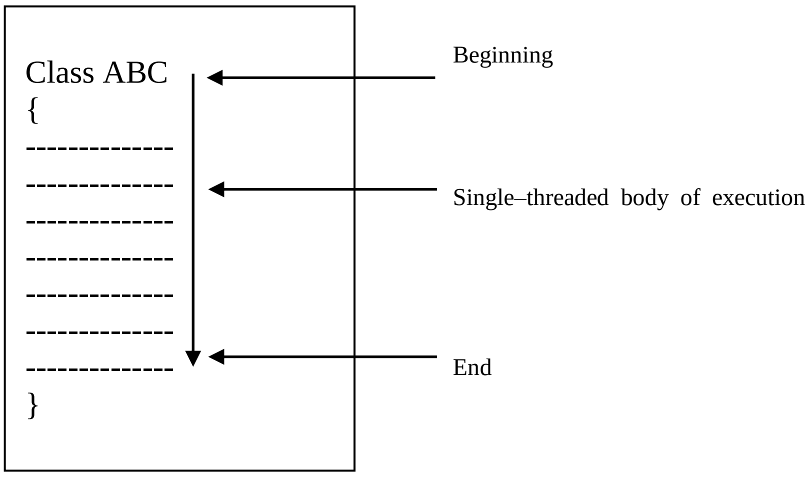


Fig. 4.4.1 Single-threaded program

A thread is small unit of program that is used to perform a particular task. Thus a process can contain multiple threads. Each thread is used to perform a specific task, which is executed simultaneously with other threads.

Every program has at least one thread. If a program contains *only one* thread, it is called **single threaded** program as shown in Fig. 4.4.1. If a program has *more than one* thread, it is called **multithreaded** program as shown in Fig. 4.4.2.

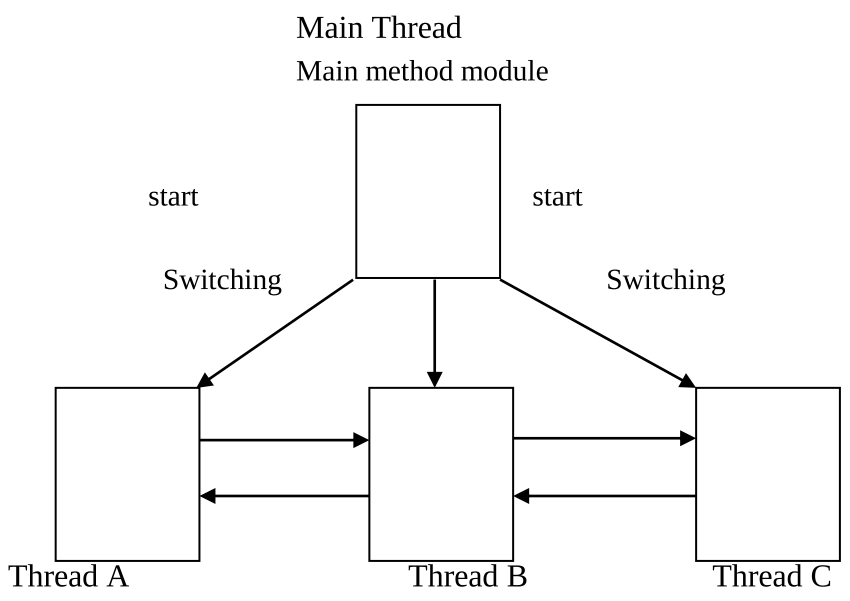


Fig. 4.4.2 A Multithreaded program

Advantages of Multithreading:

- Multithreading is used to write very efficient programs.
- Maximum use of CPU time ie., the idle time CPU is reduced.
- The time required to perform a context switch from one thread to another is less.
- Multithreading require less overheads.
- Multithreading reduces the complexity of large program.
- The resources required for a threads are less than the resources required by a process.

Some Java platform supports the concept of "time slicing". In time slicing, every thread receives a small portion of CPU time, which is called a "quantum". After the time period is over, even if the thread has not finished its execution, the thread is given no more time to continue and next thread of equal priority takes the charge of the CPU time. This is the work of Java scheduler.

4.3.12 Creating Threads

Creating threads in Java is simple. Threads are implemented in the form of objects that contain a method called **run ( )**. The *general form* of **run ( )** method is

```
public void run ( )
{
--------------------
--------------------(statements for implementing thread)
--------------------
}
```

The run ( ) method contains the entire body of the thread an it will be invoked by an object of the concerned thread.

A new thread can be created in **two ways**.

- ❖ By creating a thread class.

  Create a class that extends **Thread** class and override its **run ( )** method with the code required by the thread.

- ❖ By converting a class to a thread.

  Define a class that implements **Runnable** interface, that has only a **run ( )** method .

4.3.13 Extending The Thread Class

Extending the Thread class, it needs the following steps.

- ❖ Declare the class as extending the **Thread** class.
- ❖ Implement the **run ( )** method that is responsible for executing the sequence of code that the thread will execute.
- ❖ Create a thread object and call the **start ( )** method to initiate the thread execution.

Declaring the class

The thread class can be declared as

```
class ThreadName extends Thread
{
        -------------------
        -------------------
}
```

Where

class, extends and Thread - are keywords.

ThreadName - is the name of the thread.

Implement the run ( ) method

The **run ( )** method has been inherited by the new class. The **run ( )** method should be overridden to implement the code to be executed by the thread. The implementation of **run ( )** method is

```
public void run ( )
{
        -----------------   // Thread code here
        -----------------
}
```

Starting New Thread

The newly created thread can be started by using **start ( )** method. The *general form* is

```
ThreadName   ThreadObjectName = new ThreadName( );
ThreadObjectName.start ( );                  // invokes run ( ) method
```

The first statement just creates the object. The thread that will run this object is not yet running. The thread is in a *newborn* state. The second statement calls **start ( )** method and thread will move into the *ruunable* state. Then the Java runtime will schedule the thread to run by invoking its **run ( )** method and thread is to be in the *running* state.

4.3.14 Stopping And Blocking A Thread

Stopping a Thread

The method **stop ( )** is used to stop a thread in running state. The *general form* is

```
ThreadObjectName.stop();
```

Where ThreadObjectName - is the name of the thread object to be stopped.

stop ( ) - is the method causes the thread to move to the dead state.

The **stop ( )** method may be used when the *premature* death of a thread is desired. A thread will also move to the *dead state automatically* when it reaches the end of its method.

Blocking a Thread

A thread can also be temporarily suspended or blocked from entering into the runnable and running state by using either of the following thread methods.

- ❖ sleep ( ):

  The thread will return to the runnable state when the specified time is elapsed. For example, the statement **sleep(1000)**, is block a thread for 1000 milliseconds.

- ❖ suspend ( ):

  The thread will be blocked until further order. The blocked thread can be resumed by **resume( )** method.

- ❖ wait ( ):

  The thread will be blocked until certain condition occurs. The **notify ( )** method is called in the case of **wait ( )**.

4.3.15 Life Cycle Of A Thread

The following states can be occurs during the life cycle of a thread. It can move from one state to another state through a variety of methods as shown in Fig. 4.4.3.

- ❖ Newborn state
- ❖ Runnable state

- ❖ Running state
- ❖ Blocked state
- ❖ Dead state

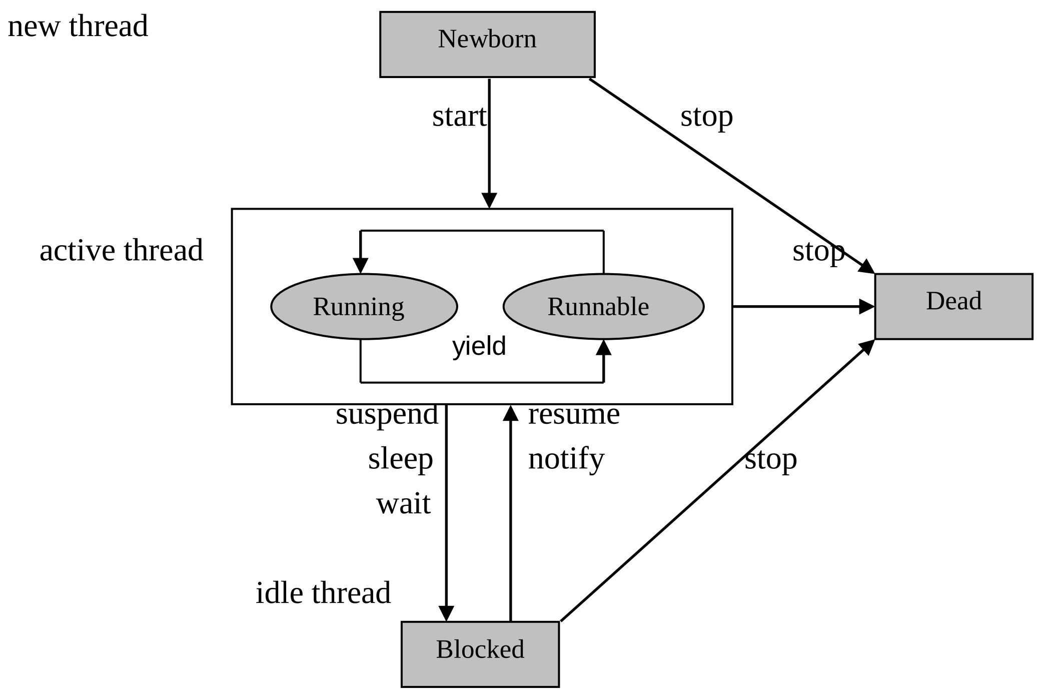


Fig. 4.4.3 State transition of a thread

Newborn state

A thread is in newborn state immediately after we create a thread object. At this state, we can do only one of the following things with it. Scheduling of newborn state as shown in Fig.4.4.4.

- ❖ To move the thread into running state using **start ( )** method.
- ❖ To kill the thread using the **stop( )** method.

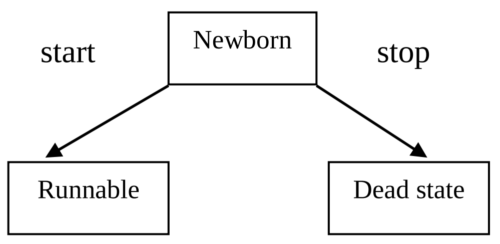


Fig. 4.4.4 Scheduling a newborn thread

Runnable state

The *runnable* state means that a thread is ready to run and is a waiting for the control of the processor. That is, the thread has joined the queue of threads that are waiting for execution. If one of the thread wants to relinquish control to another thread of equal priority, then **yield ( )** method is used.(see Fig 4.4.5)

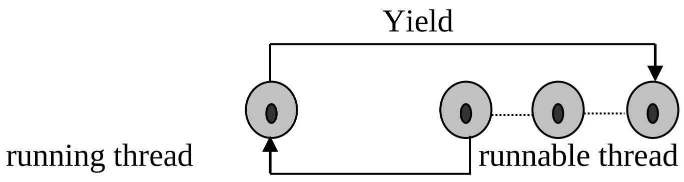


Fig. 4.4.5 Relinquishing control using yield ( ) method

Running state

Running means the processor has given its time to the thread for its execution. A running thread may relinquish its control on its own or other higher priority thread in the one of the following ways.

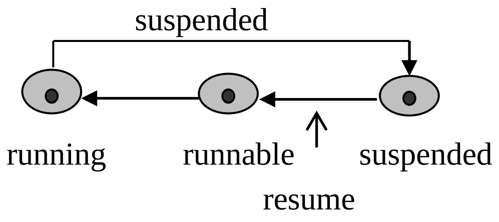


Fig. 4.4.6 Relinquishing control using suspend ( ) method

The thread will be blocked until further order by using **suspend( )** method. The blocked thread can be resumed by **resume( )** method. This is useful when we want to suspend a thread for some time due to certain reason, but do not want to kill it. (See Fig 4.4.6)

It has been made to sleep. We can put a thread to sleep for a specified time period using the method **sleep (time)** where time is in milliseconds. The thread will return to the runnable state when the specified time is elapsed. For example, the statement **sleep (1000)**, is block a thread for 1000 milliseconds. (See Fig 4.4.7)

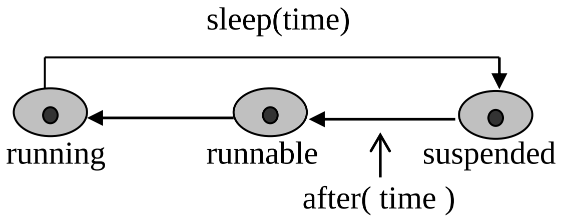


Fig. 4.4.7 Relinquishing control using sleep ( ) method

The thread will be blocked until certain condition occurs by suing **wait ( )** method. The **notify ( )** method is used to schedule thread to run again. (See Fig.4.4.8)

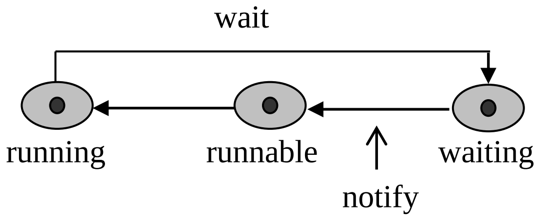


Fig. 4.4.8 Relinquishing control using wait ( ) method

Blocked state

A thread is in blocked state, if it is being prevented from the runnable and running state. This happened when thread is suspended, sleeping, or waiting in order to satisfy certain requirements. While a thread is in the blocked state, the scheduler will simply skip over it and no CPU time is allotted, until a thread re-enters the runnable and running state it will not perform any operation. A blocked thread is considered as "not runnable" but not dead.

Dead state

A thread is dead when it finishes its execution (natural death) or is stopped (killed) by another thread (premature death). A thread can be killed as soon as it born, or while it is running, or even when it is blocked state.

4.3.16 Using Thread Methods

In this section illustrate about how **Thread class** methods can be used to control the behaviour of a thread**.**

Example Program: Program for Multiplication Tables Using Threads and Multithreads

```
class five extends Thread
{
  public void run()
  {
        int mul =0;
        int n =5;
        for(int i=1 ;i<=4;i++)
        {
          mul = i*n;
          System.out.println(i + " * " + n + " = " + mul);
        }
  }
}
class six extends Thread
{
   public void run()
   {
        int mul =0;
        int n =6;
        for(int i=1;i<=4;i++)
        {
          mul = i*n;
          System.out.println(i + " * " + n + " = " + mul);
```

```
             if (i==3)
             **yield();**
           }
      }
}
class seven extends Thread
{
   public void run()
  {
          int mul =0;
          int n =7;
          for(int i=1;i<=4;i++)
          {
          mul = i*n;
           System.out.println(i + " * " + n + " = " + mul);
           if (i==2)
           {
                   try
                   {
                            sleep(1000);
                   }
                   catch(Exception e) { }
             }
          }
    }
}
class eight extends Thread
{
  public void run()
  {
          int mul =0;
          int n =8;
          for(int i=1;i<=4;i++)
          {
             mul = i*n;
             System.out.println(i + " * " + n + " = " + mul);
             if (i==3)
             stop();
```

```
         }
   }
}
class nine extends Thread
{
     public void run()
     {
           int mul =0;
           int n =9;
           for(int i=1;i<=4;i++)
           {
                mul = i*n;
                System.out.println(i + " * " + n + " = " + mul);
           }
      }
}
class MulTable
{
              public static void main(String args[])
             {
                    five f = new five();
                    six  s = new six();
                    seven sn = new seven();
                    eight e = new eight();
                    nine ni = new nine();
                    ni.setPriority(Thread.MAX_PRIORITY);
                    e.setPriority(ni.getPriority() - 1);
                    f.setPriority(Thread.MIN_PRIORITY);
                    s.setPriority(f.getPriority() + 1);
                    f.start();
                    s.start();
                    sn.start();
                    e.start();
                    ni.start();
              }
}
```

Output Of Program

```
1 * 7 = 7
2 * 7 = 14
1 * 8 = 8
2 * 8 = 16
3 * 8 = 24
1 * 9 = 9
2 * 9 = 18
3 * 9 = 27
4 * 9 = 36
1 * 6 = 6
1 * 5 = 5
2 * 6 = 12
2 * 5 = 10
3 * 6 = 18
3 * 5 = 15
4 * 6 = 24
4 * 5 = 20
3 * 7 = 21
4 * 7 = 28
```

4.3.17 Thread Exceptions

The call to sleep ( ) method is enclosed in a try block and followed by the a **catch** block. This is necessary because the sleep ( ) method throws an exception, which should be caught. If we fail to catch the exception, program will not compile.

Thread exception are caused if the active thread calls method, which is not related to its state.

For example,

- ❖ A sleeping thread cannot deal with the resume ( ) method because a sleeping thread can not receive any instruction.
- ❖ If a dead state thread calls suspend ( ) or sleep ( ) method, then thread exception can occur.
- ❖ If a blocked thread calls suspend ( ) method, than thread exception can occur.

The different forms of the catch statements are

```
catch( ThreadDeath e)
{
        ---------------
        ---------------
        }
catch (InterruptedException e)
{
        ---------------
        ---------------
}
catch (IllegalArgumentException e)
{
        ---------------
        ---------------
}
catch (Exception e)
{
        ---------------
        ---------------
}
```

The exception handler must be specified in catch statement whenever calling a thread method that throws an exception. Some of the exceptions caused by threads are

❖ ThreadDeath	- killed thread
❖ InterruptedException	- cannot handle it in the current state
❖ IllegalArgumentException	- by Illegal method argument passing
❖ Exception	- any kind of exception

4.3.18 Thread Priority

Each thread created has a priority attached to it. The scheduler allocates time according to these priorities. The thread scheduler to decide when each thread should be allowed to run uses thread priorities. A higher priority can preempt the lower priority thread, thus taking the processor's time. The priority of the thread can be set by the method **setPriority ( )**. The *general form* is

```
ThreadObjectName.setPriority(intNumber);
```

Where

ThreadObjectName - is the name of the thread object.

IntNumber - is an integer value (from 1 to 10) to which the thread's priority is set.

The Thread class defines several priority constants are

MIN_PRIORITY	=	1
NORM_PRIORITY	=	5
MAX_PRIORITY	=	10

The default setting of a thread priority value is NORM_PRIORITY. Refer the section 4.4.6 for Thread priority example program .

4.3.19 Synchronization

All the threads in a program share the same memory space. So it is possible for two threads to access the same variable and methods in an object. Problems may occur when two or more threads accessing the same data concurrently. The Java enables us to overcome this problem using a technique is called as **synchronization**.

The keyword **synchronized** is used in the code to enable synchronization. The word '**synchronized**' can be used along with a *method* or within a *block.*

```
synchronized void update ( )
{
        ------------- // code here is synchronized
        --------------
}
```

When declaring a method as synchronized, Java creates a "monitor" and hands it over to the thread that calls the method first time. As long as the thread contains the monitor, no other thread can enter the synchronized section of code.

After the work is over, the thread will hand over the monitor to the next thread that is ready to use the same resource.

To mark block of code as synchronized as shown below:

```
synchronized  (lock object)
{
        ------------- // code here is synchronized
        --------------
}
```

When two or more threads are waiting to gain control of a resources, due to some reasons, the condition on which waiting threads to gain control not happened. This situation is known as **deadlock**.

For example, assume that the thread X must access MethodA before it can release MethodB, but the thread Y cannot release MethodA until it gets hold of MethodB. This is the problem to arises the dead lock.

Thread X

```
synchronized   MethodB ( )
{
        synchronized   MethodA ( )
        {
                ------------- // code here is synchronized
                -------------
        }
}
```

Thread Y

```
synchronized  MethodA
{
        synchronized   MethodB ( )
        {
                ------------- // code here is synchronized
                -------------
        }
}
```

4.3.20 Implementing The Runnable Interface

The Runable interface declares the **run ( )** method that is required for implementing threads in our program. To do this, we must perform the following steps:

- ❖ Declare the class as implementing Runnable interface.
- ❖ Implement the **run ( )** method.
- ❖ Create a thread by defining an object.
- ❖ Call the thread's **start ( )** method to run thread.

Example Program: Program for implementing Runnable Interface

```
class xrun implements Runnable
{
        public void run()
        {
                for(int i=1;i<=6; i++)
                {
                    System.out.println("ThreadX:"+i);
                }
                System.out.println("End of Threadx");
        }
}
class runnabletest
{
        public static void main(String args[])
         {
                 xrun runobject=new xrun();
                Thread threadx=new Thread(runobject);
                threadx.start();
                System.out.println("End of main Thread");
         }
}
```

Output Of Program

```
        End of main Thread
                ThreadX:1
                ThreadX:2
                ThreadX:3
                ThreadX:4
                ThreadX:5
                ThreadX:6
        End of Threadx
```

4.3.21 Self Assessment Questions

Fill in the blank

1.Thread class and __________ interface are used to implement multithread program.

2. The default setting of a thread priority value is _________________.

True / False

1. Multithread programming supports parallel processing.

2. sleep() method in Thread class requires milliseconds as arguments.

Multiple choices

1. Which of the following method is supported for dead state in thread

a) sleep()	b) wait()
c) stop()	d) star()

3. Setpriority() method of Thread class has MAX-PRIORITY constant taken as

a) 2	b) 3
c) 4	d) 10

3. A suspended thread can be received by using the

a) stop() method	b) yield() method
c) wait() method	d) resume() method

4. The following interface is used to implement multithread program

a) Runnable	b) DataInput
c) DataOuput	d) None of the above

Short Answer

1. When the process or thread goes to deadlock state?

____

____

2. List out any two kind of thread exception.

____

____

4.4 Managing Errors And Exceptions

4.4.1 Introduction

A mistake may lead to causing an error to program to produce unexpected results. Errors are the wrong that can make program go wrong. An error may produce an incorrect output or may stop the program execution.

4.4.2 Types Of Errors

Errors may be classified into *two* types.

- ❖ Compile-time errors
- ❖ Run-time errors

Compile-Time Errors:

All syntax errors will be detected and displayed by Java compiler is called as compile-time errors. Most of the compile-time errors are due to typing mistakes. Sometimes, a single error may be the source of multiple errors. For example, use of an undeclared variable in a number of places will cause a series of errors of type *" undefined variable"*. The some common compile-time errors are

- ❖ Missing semicolons.
- ❖ Missing (or mismatch) of the brackets in classes and methods.
- ❖ Misspelling of identifiers and keywords.
- ❖ Missing double quotes in strings.
- ❖ Use of undeclared variables
- ❖ Incompatible types in assignment / initialization
- ❖ Bad references to objects

- ❖ Use of = in place of = =operator and etc.,

Whenever the compiler displays, this kinds of error, it will not create the **.class** file. It is therefore compulsory to correct all the errors before we can successfully compile and run the program.

Run-Time Errors:

A program may compile successfully creating the **.class** file but may not run properly. Such programs may produce wrong results due to wrong logic or stop the program execution due to errors like

- ❖ Dividing an integer by zero.
- ❖ Accessing an element that is out of the bounds of an array.
- ❖ Trying to store a value into array of an incompatible class or type.
- ❖ Trying to cast an instance of a class to one of its subclasses.
- ❖ Passing a parameter that is not valid range or value for a method.
- ❖ Trying to illegally change the state of a thread.
- ❖ Converting invalid string to a number.
- ❖ Accessing a character that is out of bounds of a string And etc.,

4.4.3 Exceptions

An **exception** is a condition that is caused by a run-time error in the program. When the java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws it. If the exception object is not caught and handled properly, the interpreter will display an error message and will stop the program execution. If you want to the program to continue with execution of the remaining code then we try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This task is known as **exception handling**. Some common exceptions listed out in the Table 4.5.1. The following *tasks* are used to handling the errors.

1. Find the problem (***Hit*** the exception)
2. Inform that an error has occurred (***Throw*** the exception)
3. Receive the error information (***Catch*** the exception)
4. Take corrective action (***Handle*** the exception)

The error handling code basically consists of *two* segments.

- ❖ To detect errors and to throw exceptions
- ❖ To catch exception and to take appropriate actions.

Table 4.5.1 Some common exceptions

Exception Type	*Cause of Exception*
ArithmeticException	Caused by math errors such as division by zero
ArrayIndexOutOfBoundsException	Caused by bad array indexes

ArrayStoreException	Caused when a program tries to store the wrong type of data in as array
FileNotFoundException	Caused by an attempt to access a nonexistent file
IOException	Caused by general IO failures
NullPointerException	Caused by referencing a null object
NumberFormatException	Caused when a conversion between strings and number fails.
OutOfMemoryException	Caused when there's not enough memory to allocate a new object
SecurityException	Caused when an applet tries to perform an action not allowed by the browser's security setting
StackOverFlowException	Caused when the system runs out of stack space
StringIndexOutOfBoundsException	Caused when a program attempts to access a nonexistent character position in a string

4.4.4 Syntax Of Exception Handling Code

The basic concepts of exception handling are throwing an exception and catching it as shown in Fig.4.5.1. Java uses a keyword **try** to preface a block of code that is likely to cause an error condition and " *throw* "an exception. A catch block defined by the keyword **catch** "*catches*" the exception "*thrown*" by the try block and handles it appropriately. The catch block is added immediately after the try block. The *general form* is

```
----------------
----------------
try
{
        statement;      // generates an exception
}
catch ( Exception-type e)
{
        statement;      // processes the exception
}
---------------
```

The try block can have one or more statements that could generate an exception. If any one statement generates an exception, the remaining statements in the block are skipped and execution jumps to the catch block that is placed next to the try block. The catch block can have one or more statements that are necessary to process the exception.

4.4.5 Multiple Catch Statements

A single try block can have many catch blocks. This is necessary when the try block has statements that may raise different types of exceptions. The *general form* is

```
----------------
----------------
try
{
             statement;      // generates an exception
}
catch ( Exception-type1 e)
{
            statement;      // processes the exception type 1
}
catch ( Exception-type2 e)
{
            statement;      // processes the exception type 2
}
.
.
catch ( Exception-type-N e)
{
            statement;      // processes the exception type N
}
---------------
---------------
```

Example Program: Program for Handling of Various Exception

```
import java.lang.Exception;
import java.io.DataInputStream;
class except
{
              public static void main(String args[])
            {
                        DataInputStream in =new DataInputStream(System.in);
```

```
                int i=0, b=0,c=0,n=0,sum=0;
                try
                {
                        System.out.print("Enter the value for n :");
                        n=Integer.parseInt(in.readLine());
                        int a[] = new int[n];
                        System.out.println("Enter the value for a:");
                        for(i=0;i<=n-1;i++)
                        {
                                a[i]=Integer.parseInt(in.readLine());
                                sum = sum+a[i];
                        }
                        System.out.println("Enter the value for b:");
                        b=Integer.parseInt(in.readLine());
                        c=sum/b;
                        System.out.println("The value for c is :" + c);
                        a[n+2] = c;
                }
                catch(ArithmeticException e)
                {
                         System.out.println("Divisible by zero\n"+ e);
                }
                catch(ArrayIndexOutOfBoundsException e1)
                {
                         System.out.println("Bad Array Index\n"+ e1);
                }
                catch(NumberFormatException e2)
                {
                   System.out.println("Input must be an Integer\n"+ e2);
                }
                catch(Exception e)
                {   }
        }
}
```

Output Of Program

```
Enter the value for n :4
Enter the value for a:
3
```

```
3
3
3
Enter the value for b:
3
The value for c is :4
Bad Array Index
java.lang.ArrayIndexOutOfBoundsException
```

Java does not require any processing of the exception at all. We can simply have a catch statement with empty block to avoid program abortion.

Example:

catch (Exception e);   // Does nothing

This statement catches the exception and then ignores it.

4.4.6 Using Finally Statement

Finally statement that can be used to handle any exception generated within a try block. It may be added immediately after the try block or after the catch block. The *general form* is

```
try                                     try

{                                       {
}                                        }
finally                                 catch
{                                       {

}                                       }
                                        finally
                                        {

                                        }
```

When a finally block is defined, this is guaranteed to execute, regardless of whether or not in exception is thrown.

4.4.7 Throwing Our Own Exceptions For Debugging

To throw our own exception by using the keyword **throw**. The general form is

```
Throw new Throwable_subclass;
```

Examples :

```
throw new ArithmeticException();
throw new NumberformatException();
```

Example Program: Program for User-Defined Exception Handling

```
import java.lang.Exception;
import java.io.*;
class MyException extends Exception
{
        MyException(String message)
        {
                super(message);
        }
}
class TestMyException
{
        public static void main(String args[])
        {
                int x = 0,  y =0;
                DataInputStream in = new DataInputStream(System.in);
                try
                {
                        System.out.println("Enter the  x & y values");
                        x = Integer.parseInt(in.readLine());
                        y = Integer.parseInt(in.readLine());
                }
                catch(Exception e){ }
                try
                {

                        float z = (float)x / (float) y;
                        if (z <0.01)
                        {
```

```
                    throw new MyException("Number is too
                small");
                    }
                }
                catch(MyException e)
                {
                        System.out.println("Caught My Exception");
                        System.out.println(e.getMessage());
                }
                finally
                {
                        System.out.println("I am always here");
                }
        }
}
```

Output Of Program

```
Enter the  x & y values
10
10
I am always here

Enter the  x & y values
1
1000
Caught My Exception
Number is too small
I am always here
```

4.4.8 Self Assessment Questions

Fill in the blank

1. Misspelling of identification & keywords are examples of ___________ errors.
2. To throw our own exception by using the keyword ______________.

True or False

1. Dividing an integer by zero error is not occurred at run time
2. Number Format exception is generated when a conversion between string and number fails
3. Finally block can be added in last of the try catch block

Multiple Choices

1. Which of the following is occurred at runtime error

a) missing;	b) missing double quotes in strings
c) use of undeclared variables	d) out of bounds in Array

2. Try block, which is doing that

a) Find the error and inform the error	b) Receive the error information
c) Take corrective action	d) none of the above

3. Which of the following is keyword

a) Total b) Sum c) Catches d) throw

Short Answer

1. Define exception.

---

2. What is the purpose of Finally statement?

Summary

Packages, interfaces and multithreaded are the important features of java programming.

In packages we saw the building blocks of coding in Java. We have also discussed about how to create a package, add more classes to package and access the contents of a package. Also seen about how to use Java system packages.

A thread is a single line of execution within the program. Multi threads can run concurrently in a single program. A thread is created either by sub classing the Thread class or implementing Runnable interface. We have also discussed about the how to synchronized threads ,how to set priorities for threads and how to control the execution of threads.

Exception handling is another important features of Java. In exception handling we have discussed about what exceptions are how to throw system & user defined exception and, how to handle different types of exceptions.

4.5 Unit Questions

1. What is a package? How do we add a class or an interface to a package?
2. Describe the complete life cycle of a thread
3. How do we set priorities for threads?
4. What is synchronization? When do we use it?
5. Develop a simple program to illustrate the use of multithreads
6. What is difference between multiprocessing and multithreading?
7. Explain the following terms

   a) Final method & final class b)A bstract method & abstract class

8.What is an exception? Explain types of exceptions with examples

9.Explain user defined exception with an example

4.6 Answer for Self Assessment Questions

Answer 4.3.10

Fill in the blank

1. user defined package and Built-in package 2. import statement

True / False

1.True 2. False

Multiple Choice

1. a 2. b

Short Answer

1. To use the classes and or interfaces from other program without physically copying them into the program is done by Java package.
2. The classes declaring as "*not public*" can be used only in the same package, not possible to access the outside of the packages is called as hiding classes.

Answer 4.4.11

Fill in the blank

1. Runnable interface 2. NORM_PRIORITY

True / False

1. True 2. True

Multiple Choice

1. c 2. d 3.d 4. a

Short Answer

1. When two or more threads are waiting to gain control of a resources, due to some reasons, the condition on which waiting threads to gain control not happened. This situation is known as **deadlock**.

2. ThreadDeath, InterruptedException

Answer 4.5.8

Fill in the blank

1. compile time 2. throw

True / False

1. False 2. True 3. True

Multiple Choice

1. d 2. a 3. d

Short Answer

1. An **exception** is a condition that is caused by a run-time error in the program.
2. Finally statement that can be used to handle any exception generated within a try block.

NOTES

UNIT - V

5.1 Introduction

Applet are *small Java programs* that are primarily used in Internet computing. They can be transported over the Internet from one computer to another computer and run using **Applet Viewer** or **any Web Browser**. It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation, and play interactive games. Java applet when run, it can produce graphics, sound and moving images.

Java Applet has its own area of the screen known as **canvas**, where it creates its display. The size of an applet's space is decided by the attributes of the **<APPLET…> tag.** We can write Java applets that *draw lines, figures, images, and text in different fonts and styles*. A Java applet draws graphical image inside its space using the coordinate system.

The variables and arrays are used for storing data inside the programs. This approach yields the following problems.

- ❖ The data is lost either when variable goes out of scope or when the program is terminated.
- ❖ It is difficult to handle large amount of data using variables and arrays.

We can overcome these problems by storing data on *secondary storage devices* such as floppy disks or hard disks. The data is stored in these devices using the concept of *files*.

In this unit we shall describe in details *about how applet programs works, how to draw picture using the graphics object and its methods, and file concept in Java.*

5.2 Objectives

After studying this lesson, you should be able to:

- ❖ Describe the major elements of applet program
- ❖ Describe about how applet program used in Internet pages.
- ❖ Describe about graphics object and its methods.
- ❖ Understand about how to draw the pictures using graphics methods.
- ❖ Describe the classification of files and its related stream classes.

5.3 Applet Programming

5.3.1 Introduction

Applet are small Java programs that are primarily used in Internet computing. They can be transported over the Internet from one computer to another computer and run suing **Applet Viewer** or **any Web Browser**. It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation, and play interactive games. Java applet when run, it can produce graphics, sound and moving images.

Local applets

An applet developed locally and stored in a local system is known as **local applet**. The local system does not require the Internet connection. It simply searches the directories in the local system and locates and loads the specified applet as shown in Fig.5.3.1.

Remote applets

Remote applet is developed by some one else and stored on a remote computer connected to the Internet. We can download the remote applet onto our system via the Internet and run it as shown in Fig.5.3.2.

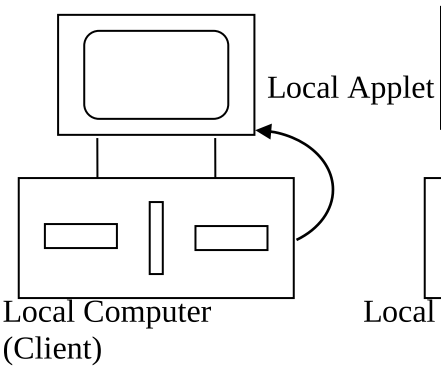


Fig.5.3.1 Loading Local Applet

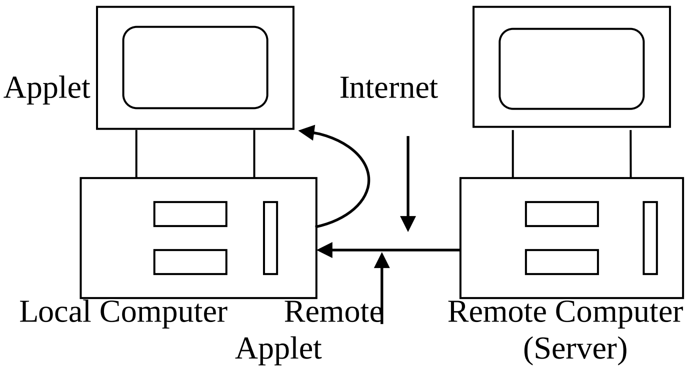


Fig.5.3.2 Loading a Remote Applet

Uniform Resource Locator (URL)

URL is used to locate and load the applet address on the web and must be specified in the applet's HTML document as the value of the CODEBASE attribute.

Example:

CODEBASE =http : // www.netserver.com / applets

In the case of local applets CODEBASE may be absent or may specify a local directory.

5.3.2 How Applets Differ From Applications

The applets and stand-alone applications, both are java program, but applets have some difference from stand-alone application as given below.

- Applets do not use main( ) method for initiating the execution of the program. Applets, when loaded, automatically call certain methods of applet class to start and execute the applet program.
- Unlike stand-alone applications, applets cannot be run independently. They are run from a web page using HTML tag.
- Applets cannot read from or write to the files in the local computer.
- Applets cannot communicate with other servers on the network.
- Applets cannot run any program the local computer.
- Applets are restricted from using libraries from other language such as C or C++.

5.3.3 Preparing To Write Applets

Before, preparing to write applets program we will need to now

- ❖ When to use applet
- ❖ How an applets works
- ❖ What features an applet has and
- ❖ Where to start, when we first create our own applets.

Let us consider the situations when we need to use applets

1. When we need dynamic display of a web page. For example shows daily changes of share prices of various companies.
2. When we require Flash outputs. For example, applets that produce sounds, animations etc.,
3. When we want to create program and use it on the Internet for us by others on their computers.

The following steps are used to develop and test the applets.

- ❖ Building an applet code (.java file)
- ❖ Creating an executable applet (.class file)
- ❖ Designing a Web page using HTML tags
- ❖ Preparing <APPLET> tag
- ❖ Incorporating <APPLET> tag into the Web page.
- ❖ Creating HTML file.
- ❖ Testing the applet code.
- ❖

5.3.4 Building Applet Code

Applet program uses the services of *two* classes, namely, **Applet** and **Graphics** class from the Java class library. The services of two classes are

❖ **Applet** class and

The **Applet** class, which is contained in the **java.applet** package provides life and behavior to the applet through its methods such as **init( ), start( )** and **paint( ).** The **Applet** class maintains the *lifecycle* of an applet.

❖ **Graphics** class

The **paint( )** method of the applet class, when it is called actually display the results of the applet program on the screen. The output may be text, graphics, or sound. The **paint( )** method require graphics object as an argument is declared by the **Graphics** class. The *general form* of paint( ) method is

public void paint (**Graphics** g)

This requires that the applet program import the **java.awt** that contains the **Graphics** class.

The general form of building applet program is

```
import java.awt.*;
import java.applet.*;
---------------------
---------------------
public class appletclassname extends Applet
{
        --------------------
        --------------------
        public void paint ( Graphics g)
        {
                ------------------ // Applet operations code
                ------------------
        }
        ------------------
        ------------------
}
```

The **appletclassname** is the main class for the applet. When the applet is loaded, Java creates an instance of this class, and then a series of Applet class methods are called on that instance to execute the program.

5.3.5 Applet Life Cycle

Every Java applet inherits a set of default behavior s from the **Applet** class. The **Applet** class maintains the *lifecycle* of an applet as shown in Fig.5.3.3. The applet states are

- ❖ Born or Initialization state
- ❖ Running state
- ❖ Idle state
- ❖ Dead or Destroyed state

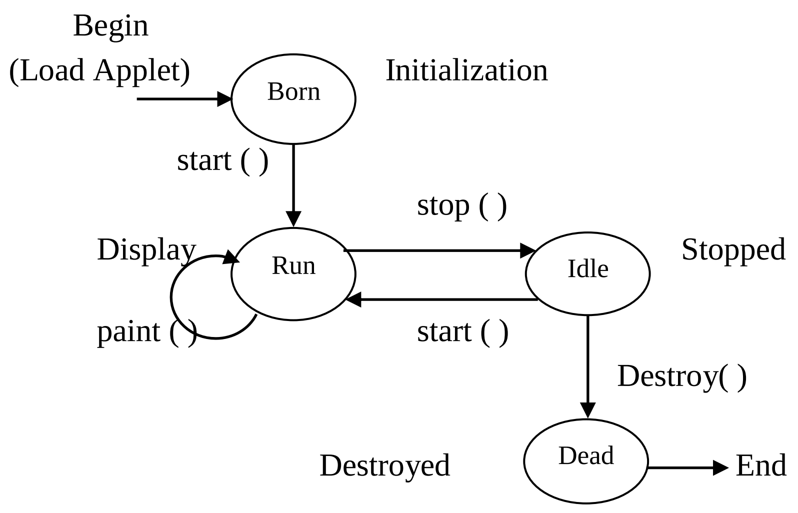


Fig. 5.3.3 Applet Life Cycle

Born or Initialization State

Applet enters the **initialization** state when it is first loaded. This is achieved by calling the **init()** method of **Applet** class. The applet is born. At this stage, we may do the following, if required

- ❖ Create objects needed by the applet
- ❖ Set up initial values
- ❖ Load images or fonts
- ❖ Set up colors

The initialization occurs *only once* in the applet's life cycle. To provide any of the behaviors mentioned above, we must override the **init( )** method.

```
public void init( )
{
        ----------------- (Action)
        -----------------
}
```

Running state

Applet enters the **running** state when the system calls the **start( )** method of applet class. This occurs automatically after the applet is initialized. Starting can also occur if the applet is already in 'stopped' (idle) state.

```
public void start( )
{
        ----------------- (Action)
        -----------------
}
```

For **example** we may leave the web page temporarily to another page and return back to the page. This again starts the applet running. The **start( )** method may be called *more than once*. We may override the **start( )** method to create a thread to control the applet.

Idle or Stopped state

An applet becomes **idle** when it is stopped from running. Stopping occurs automatically when we leave the page containing the currently running applet. We can also do so by calling the **stop()** method explicitly. If we use a thread to run the applet, then we must use **stop()** method to terminate the thread. We can achieve this by overriding the **stop()** method.

```
public void stop( )
{
        ----------------- (Action)
        -----------------
}
```

Dead or Destroyed state

An applet is said to be **dead** when it is removed from memory. This occurs automatically by invoking the **destroy()** method when we quit the browser. Destroying stage occurs *only once* in the applet's lifecycle. If the applet has created any resources, like threads, we may override the **destroy()** method to clean up these resources.

```
public void destroy( )
{
        ---------------- (Action)
        ----------------
}
```

Display State

Applet moves to the **display** state whenever it has to perform some output operations on the screen. This happened immediately after the applet enters into the running state. The **paint( )** method is called to display the output. We must override **paint( )** method if we want to be displayed on the screen.

```
public void paint ( Graphics g)
{
        ---------------- (Display statements)
        ----------------
}
```

5.3.6 Creating An Executable Applet

Executable applet is means that **.class** file of the applet, which is generated by compiling the source program of the applet. Compiling applet program is same as compiling stand- alone application. We use Java compiler (**javac**) to compile the applet program. The following steps are required for compiling the applet program.

- ❖ Move to the directory containing the source code and type the following command:

  > **javac** *appletfilename.java*

- ❖ The compiled output file called *appletfilename.class* is placed in the same directory as the source.
- ❖ If any error message is received, then we must check and correct the errors, and compile the applet program again

5.3.7 Designing A Web Page

A Web page or HTML page or HTML document is made up of text and HTML tags that can be interpreted (run) by a Web browser or applet viewer.

Web pages are stored using file extension **.html**, and it should be stored in the same directory as compiled code of the applets.

A Web page is marked by an opening HTML tag <HTML> and a closing HTML tag </HTML> and is divided into **three sections** are

- ❖ Comment section (optional)
- ❖ Head section (optional)
- ❖ Body section

Comment Section

Comment section contains comments about Web page. A comment line **begins with <!** And **end with a >**. Web browser will ignore the text enclosed between them. Comments are optional and can be included anywhere in the Web page.

Head Section

The head section is defined with a starting <HEAD> tag and a closing </HEAD> tag. Head section usually contains a title for web page. The text enclosed in the tags <TITLE> and </TITLE> will appear in the title bar of the Web browser when it displays the page. The head section is also optional. A slash (/) in a tag indicates the end of that tag section.

Body section

Body section contains the entire information about the web page and its behavior.

5.3.8 Applet Tag

We have included a pair of <APPLET….> and </APPLET> tags in the body section of HTML tags. The <APPLET….> tag supplies the name of the applet to be loaded and tells the browser how much space applet requires. The ellipsis in the tag<APPLET..> indicates that it contains certain attributes that must be specified. The minimum requirement of <APPLET….> tag specifies *three* things.

- ❖ Name of the applet
- ❖ Width of the applet (in pixels)
- ❖ Height of the applet (in pixels)

The *general form of* APPLET tag is

```
<APPLET
        CODE = appletfilename.class
        WIDTH        = (in pixels)
        HEIGHT       = (in pixels)
        ----------------------
        ----------------------    >
</APPLET>
```

5.3.9 Adding Applet To Html File

Adding applet to a HTML document, we should follow the following steps:

- ❖ Insert an <APPLET>tag at an appropriate place in the web page.
- ❖ Specify the name of the applet's **.class** file.
- ❖ If the .class file not in the current directory, use the codebase parameter to specify
    - o The relative path if file on the local system, or
    - o The URL of the directory containing the file, if it is on a remote system.
- ❖ Specify the space required for display of the applet in terms of width and height in pixels.
- ❖ Add any user-defined parameters using <PARAM>tags.
- ❖ Add alternate HTML text to be displayed when a non-Java browser is used.
- ❖ Close the applet declaration with the <APPLET> tag.

5.3.10 Running The Applet

To run an applet, it requires the following *tools*.

- ❖ Java-Enabled Web Browser (such as Internet Explorer or Hot Java)
- ❖ Java appletviewer

Java-Enabled Web Browser

If we use a *java-enabled Web browser* for running the applet program, we will be able to see the entire Web page containing the applet.

Java appletviewer

If we use a *Java appletviewer* for running the applet program, we will only see the applet output. The *appletviewer* is available as a part of the *Java Development Kit.* We can use it to run our applet as follows:

```
>appletviewer appletfilename.html
```

The argument of the **appletviewer** is not the **.java** file or the **.class** file, but rather **.html** file

5.3.11 More About Applet Tag

The *general form of* <APPLET> tag is

```
<APPLET
        [CODEBASE  =codebaseURL]
        CODE       =  appletfilename.class
        [ALT       = alternateText]
        [NAME      = applet_instance_name]
        WIDTH      = (in pixels)
        HEIGHT     = (in pixels)
        [ALIGN     =alignment]
        [VSPACE    =pixels]
        [HSPACE    =pixel]
 >
[ <PARAM NAME   =   name1    VALUE   =   value1 >]
[ <PARAM NAME   =   name2    VALUE   =   value2 >]
-----------------
-----------------
[Text to be displayed in the absence of Java]
</APPLET>
```

The attributes shown inside [ ] indicate optional.

5.3.12 Passing Parameters To Applets

We can supply user-defined parameters to an applet using <PARAM…> tags. Each <PARAM…> tag has a **name** attribute such as **color**, and **value** attribute such as **red**. Inside the applet code, the applet can refer to that parameter by name to find its value. **For example**, we can change the color of the text displayed to red by an applet using <PARAM…> tag as follows:

```
<APPLET ….>
<PARAM NAME = “color” VALUE = “red”>
</APPLET>
```

To set up and handle parameters we need to do two things:

- ❖ Include appropriate <PARAM…>tags in the HTML document.
- ❖ Provide code in the applet to parse these parameters

The parameters are passed to an applet when it is loaded. We can define the **init ( )** method in the applet to get hold of the parameters defined in the <PARAM> tags. This is done by using the **getParameter ( )** method, which takes one string argument containing the value of that parameter.

Example Program: Program for Passing parameters from HTML to Applets

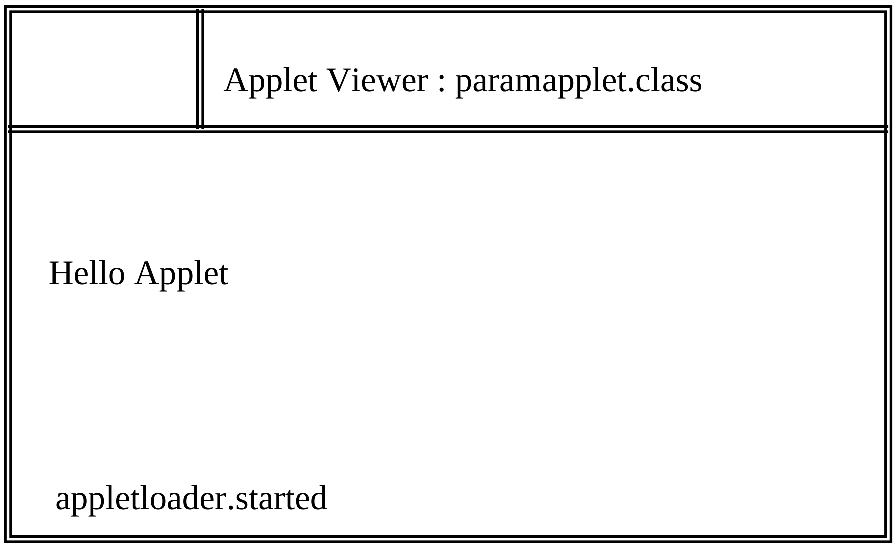
```
import java.awt.*;
import java.applet.*;
```

```
public class paramapplet extends Applet
{       String s;
         public void init()
        {
                s = getParameter("str");
                  if(s = = null)
                        s = "Java";
                  s = "Hello"+s;
         }
        public void paint(Graphics g)
         {
                  g.drawString(s,20,200);
        }
 }
```

When run applet program using the following HTML file displays the output as given below.

```
<html>
        <applet
                code = paramapplet.class
                width =300
                height=200
        >
        <param name="str" value="Applet">
        </applet>
</html>
```

Output Of Program



```
Applet Viewer : paramapplet.class

Hello Applet

appletloader.started
```

5.3.13 Aligning The Display

We can align the output of the applet using the **ALIGN** attribute. This attribute can have one of the ***nine*** values:

LEFT, RIGHT, TOP, TEXT TOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM, ABSBOTTOM

For Example

ALIGN = RIGHT - will display the output at the right margin of the page

5.3.14 More About HTMLTags

The number of HTML tags that can be used to control the style and format of the display in the web pages as shown in a Table 5.3.1.

Table 5.3.1. HTML tags and its Function

Tag	Function
<HTML> ------ </HTML>	To indicate the beginning and end of the HTML File.
<HEAD> ------ </HEAD>	To include the details about the web page and also contains <TITLE> tag
<TITLE> ------ </TITLE>	To display the text in the title bar.
<BODY> ------ </BODY>	To include the main text of the web page.
<CENTER> ---- </CENTER>	To place the text in the center of the page.
<APPLET …> ---- </APPLET>	To declare the applet details as its attributes and also it is used to declare user-defined Parameters.
<PARAM ….>	To supply the user-defined parameter.
<! …….>	To add comments. It is ignored by the Web-browser. It may be placed any where in a web page.
<B> -----</B>	To display text as BOLD type.
<BR>	To skip a line.
<P>	This tag move to the next line and start a paragraph of text.
<FONT ….> ----</FONT>	To change the color and size of the text.
<HR>	To draw horizontal line.
<H1>----</H1>	To display the headings. The header tag <H1> will have largest size and tag <H6> will have smallest size.
<H6>-----<H6>	

5.3.15 Displaying Numerical Values

In applet, we can display numerical values by converting then into strings and then using the **drawString( )** method of Graphics class. We can do this easily by using **valueOf()** method of **String** class . Program illustrate how an applet handles numerical values.

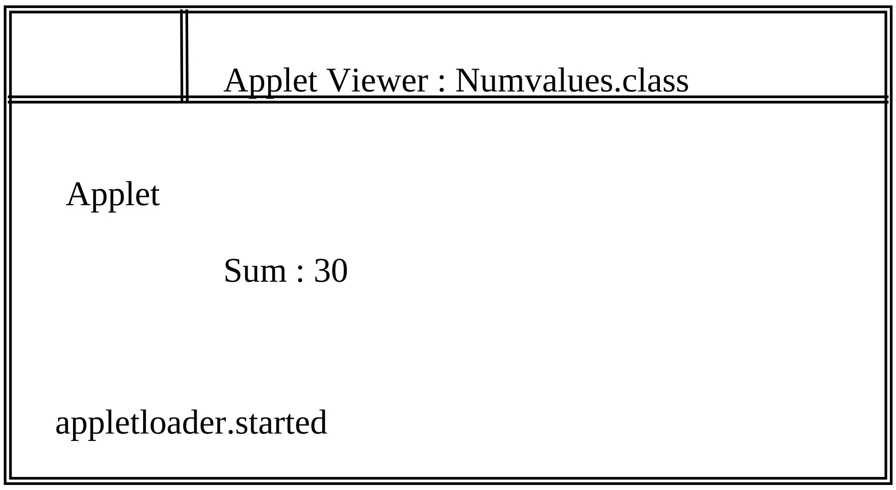
Example Program : Program for displaying numerical values

```
        import java.awt.*;
        import java.applet.*;
        public class Numvalues extends Applet
        {
                public void paint(Graphics g)
                {
                        int a =10;
                        int b=20;
                        int s = a + b;
                        String s1 = "sum:" + string.valueOf(sum);
                        g.drawString(s, 100, 100);
                }
        }
```

When run applet program using the following HTML file displays the output as given below.

```
        <html>
                <applet
                        code = GetUserIn.class
                        width =300
                        height=200
                >
                </applet>
        </html>
```

Output:



```
Applet Viewer : Numvalues.class

Applet

        Sum : 30

appletloader.started
```

5.3.16 Getting Input From The User

Applets work in a graphical environment. Therefore, applets treat inputs as text. We must create area of the screen in which user can type and edit input items. we can do by suing the **TextField** class of the applet package. Once text fields are created for receiving input, we can type values in the field and edit them if need.

Receive the items from field for calculation. They need to convert to the corresponding data type. The results are then converted back to string for display. Program shows this implementation.

Example Program: Program for interactive input to an applet

```
import java.awt.*;
import java.applet.*;
public class GetUserIn extends Applet
{
        TextField t1, t2;
        public void init()
        {
                t1 = new TextField(10);
                t2 = new TextField(10);
                add(t1);
                add(t2);
                t1.setText("o");
                t2.setText("o");
        }
        public void paint(Graphics g)
        {
                int x=0, y=0,z=0;
                String s1,s2,s;
                g.drawString("Input a number in each box", 10, 50);
                try
                {
                        s1 = t1.getText();
                        x = Integer.parseInt(s1);
                        s2 = t2.getText();
                        y = Integer.parseInt(s2);
                }
                catch(Exception e)
                {
                }
                z = x+y;
                s= String.valueOf(z);
                g.drawString("The Sum is : ",10, 75);
                g.drawString(s,100,75);
            }
            public boolean action(Event ev , Object ob)
            {
                repaint();
                return true;
            }
        }
```

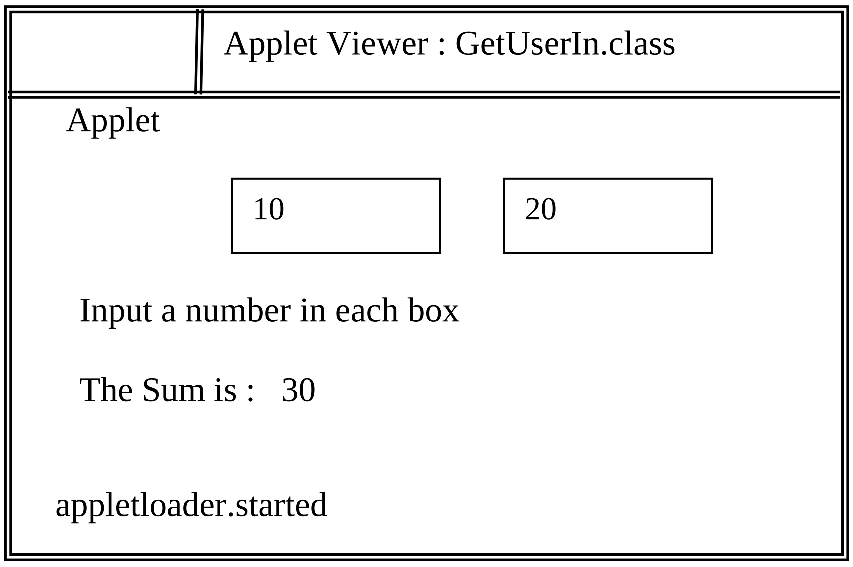
Run the applet GetUserIn using the following steps:

- ❖ Type and save the program (.java file)
- ❖ Compile the applet using java compiler (javac) it generate **.class** file
- ❖ Write a HTML document (.html file)

```
    <html>
        <applet
            code = GetUserIn.class
            width =300
            height=200
        >
        </applet>
    </html>
```

- ❖ Use the appletviewer to display the results

Output:



```
Applet Viewer : GetUserIn.class
Applet

        10          20

Input a number in each box

The Sum is :   30


appletloader.started
```

5.3.17 Self Assessment Questions

Fill in the blank

1. The ____________method of the Applet class which displays result of the applet code on the screen.
2. HTML stands for __________________________.
3. __________class in Java.awt.package, which allows you to type the information from the keyword

True or false

1. Remote applet, which is imported from the local system.
2. Applet does not use the main( ) method for initiating the execution of the code.
3. Internet explores does not support for executing applet program.

Multiple choices

1. What method is used to initiative the execution of

   a) stop() b) print()

   c) stop() d) init()

2. Which tag is used to execute the applet program

   a) <body> b) <applet>

   c) <head> d) <center>

Short Answer

1. Define local applet.

5.4 Graphics Programming

5.4.1 Introduction

Java Applet has its own area of the screen known as canvas, where it creates its display. The size of an applet's space is decided by the attributes of the <APPLET…> tag. We can write Java applets that draw lines, figures, images, and text in different fonts and styles.

A Java applet draws graphical image inside its space using the coordinate system as shown in Fig.5.4.1. Java, coordinate system has the origin (0,0) in the upper-left corner. The positive x values are to the right, and positive y values are to the bottom. The values of coordinates x and y are in pixels.

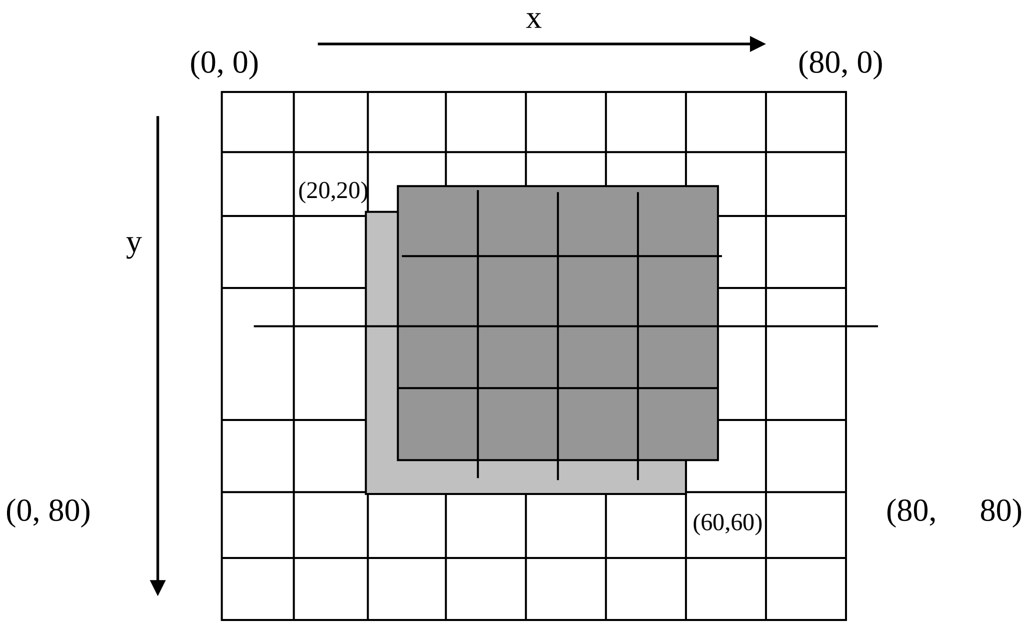


Fig.5.4.1 Coordinate system

5.4.2 The Graphics Class

Java's **Graphics** class includes methods for drawing many different types of shapes, from simple lines to polygons to text in a variety of fonts. To draw a shape on the screen, we may call one of the methods available in the Graphics class. Table 5.4.1 shows the commonly used drawing methods available in the Graphics class. All the drawing methods have arguments representing end points, corners, or starting locations of a shape as values in the applet's coordinate system. To draw a shape, we only need to use the appropriate method with the required arguments.

Table 5.4.1 Drawing methods of the Graphics class

Method	Description
clearRect ( )	Erases a rectangular area of the canvas.
copyArea ( )	Copies a rectangular area of the canvas to another area.
drawArc ( )	Draws a hollow arc.
drawLine ( )	Draws a straight line.
drawOval ( )	Draws a hollow oval.
drawPolygon ( )	Draws a hollow polygon
drawRect ( )	Draws a hollow rectangle.
drawRoundRect ( )	Draws a hollow rectangle with rounded corner.
drawstring ( )	Displays a text string.
fillArc ( )	Draws a filled arc.
fillOval ( )	Draws a filled oval.
fillPolygon ( )	Draws a filled polygon

fillRect ( )	Draws a filled rectangle.
fillRoundRect ( )	Draws a filled rectangle with rounded corners.
getColor ( )	Retrieves the current drawing color.
getFont ( )	Retrieves the currently used font.
getFontMetrics ( )	Retrieves information about the current font
setColor ( )	Set the drawing color.
setFont ( )	Set the font.

5.4.3 Lines And Rectangles

The **drawLine ( )** method is used to draw a line, it takes two pair of coordinates, (x1, y1) and (x2, y2) as arguments and draws a line between them. The *general form* is

g.drawLine(x1, y1, x2, y2);

Example:

g.drawLine(10, 10, 50, 50);

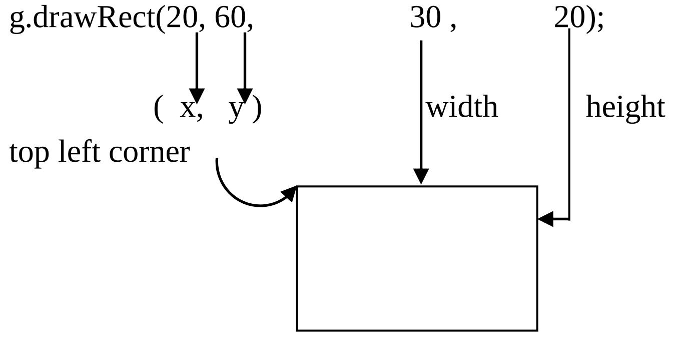
The **g** is the **Graphics** Object passed to **paint ( )** method.

The **drawRect ( )** method is used to draw a rectangle, it takes four arguments, the first two represent the x and y coordinates of the top left corner of the rectangle, and remaining two represent width and height of the rectangle. The *general form* is

g.drawRect(x, y, width, height);

Example:

g.drawRect(20, 60, 30 , 20);



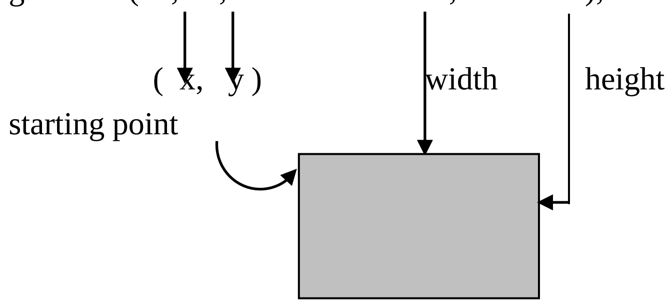
Rectangle

The **drawRect ( )** method draws only the outline of a box. We can draw a solid box using the method **fillRect ( )** method. This also takes four parameters, the first two represent the x and y coordinates of the top left corner of the rectangle, and remaining two represent width and height of the rectangle. The *general form* is

g.fillRect(x, y, width, height);

Example:

g.fillRect(20, 60, 30 , 20);



Filled Rectangle

We can also draw rounded edges rectangles, using the methods **drawRoundRect ( )** and **fillRoundRect ( )**. These two methods are same as drawRect ( ) and fillRect ( ) methods except that they take *extra two arguments* representing the width and height of the angle of corner.The *general forms* are

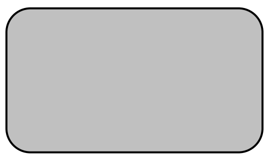
g.drawRoundRect(x, y, width, height , width of angle of corner, height of angle of corner);
g.fillRoundRect(x, y, width, height, width of angle of corner, height of angle of corner);

Example:

g.drawRoundRect(20, 60, 30 , 20, 10, 10);

Rounded Rectangle

g.fillRoundRect(20, 60, 30 , 20, 10, 10);



Example Program: Program for Hut Drawing Using Lines, Rectangles and FillRectangle

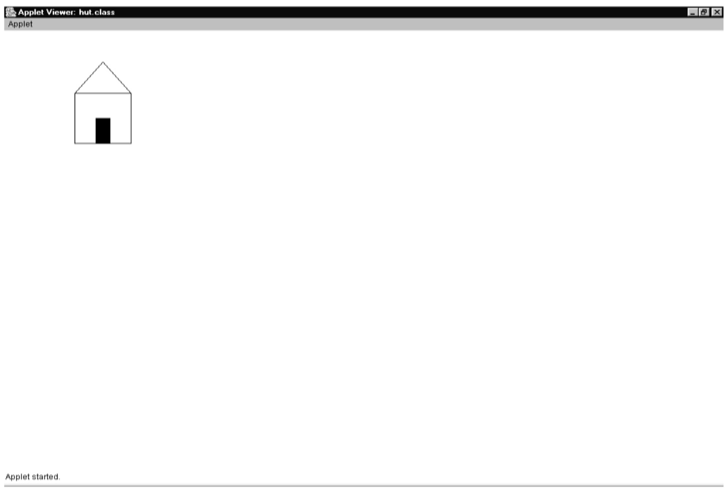
```
import java.awt.*;
import java.applet.*;
public class hut extends Applet
{
        public void paint(Graphics g)
```

```
                {
                        g.drawRect(100,100,80,80);
                        g.drawLine(100,100, 140,50);
                        g.drawLine(140,50 ,180,100);
                        g.drawRect(130,140, 20,40);
                        g.fillRect(130,140,20,40);
                }
        }
```

When run applet program using the following HTML file display the output as given below.

```
        <html>
                <applet
                        code = hut.class
                        width =300
                        height=200
                >
                </applet>
        </html>
```

Output



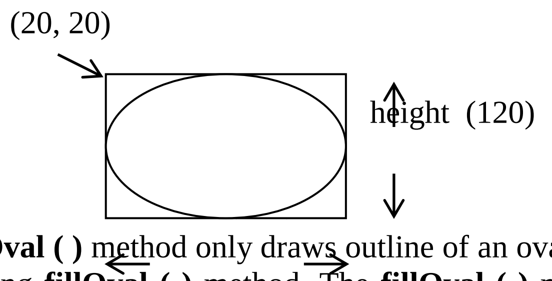
5.4.4 Circles And Ellipses

The **drawOval()** method can be used to draw a circle or an ellipse. The **drawOval ( )** method takes *four arguments*, The first two represent the top left corner of the imaginary rectangle and other two represent the width and height of the oval itself. If the width and height are same, the oval becomes a circle. The *general form* is

drawOval(x, y, width, height);

Example:

```
drawOval(20, 20, 160, 120);
```



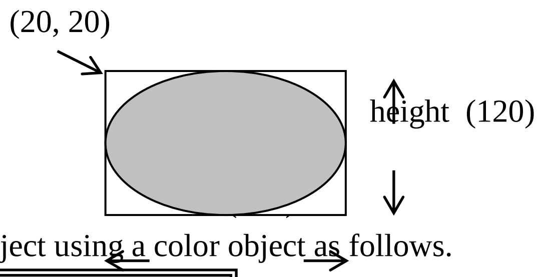
The **drawOval ( )** method only draws outline of an oval. We can draws a solid oval by using **fillOval ( )** method. The **fillOval ( )** method takes *four arguments*. The first two represent the top left corner of the imaginary rectangle and other two represent the width and height of the filled oval itself. If the width and height are same, the filled oval becomes a filled circle.

The *general form* is

```
fillOval(x, y, width, height);
```

Example:

fillOval(20, 20, 160, 120);



We can draw an object using a color object as follows.

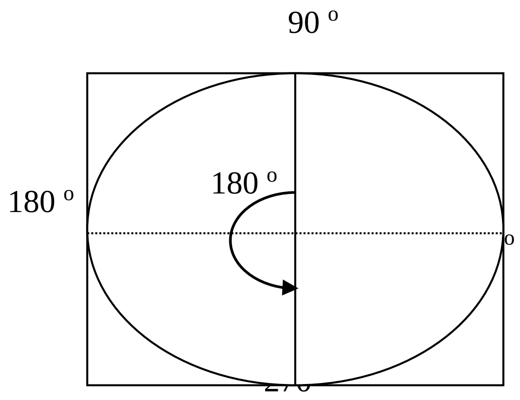
```
g.setColor(Color.green);
```

After setting the color, all drawing operations will occur in that color.

5.4.5 Drawing Arcs

The **drawArc ( )** method is used to draw arcs. The **drawArc ( )** method takes *six arguments*, the first four are the same as the arguments of **drawOval ( )** method and last two represent the *starting angle* of the arc and the *number of degrees* around the arc.

In drawing arcs, java actually formulates the arc as an oval and then draws only a part of it as dictated by last two arguments. Java consider the 3 O' clock position as zero degree position and degree increase in anti-clockwise direction as shown in Fig.5.4.2. So, to draw an arc from 12.00 O' clock position to 6.00 O' clock position, the starting angle would be 90^o, and the sweep angle would be 180^o.



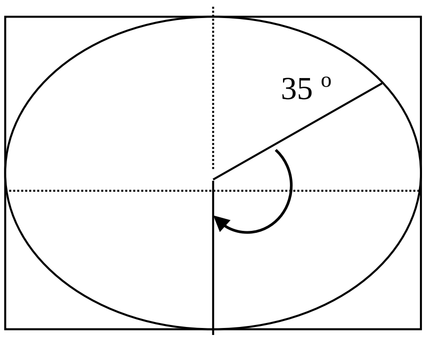


Fig.5.4.2. Arc as a part of an oval **Fig.5.4.3. Drawing an arc in clockwise**

We can also draw an arc in backward direction by specifying the sweep angle as negative. For example, the last angle is –135^o and the starting angle is 35^o, then the arc is drawn as shown in Fig.5.4.3. We can use **fillArc ( )** method to fill the arc.

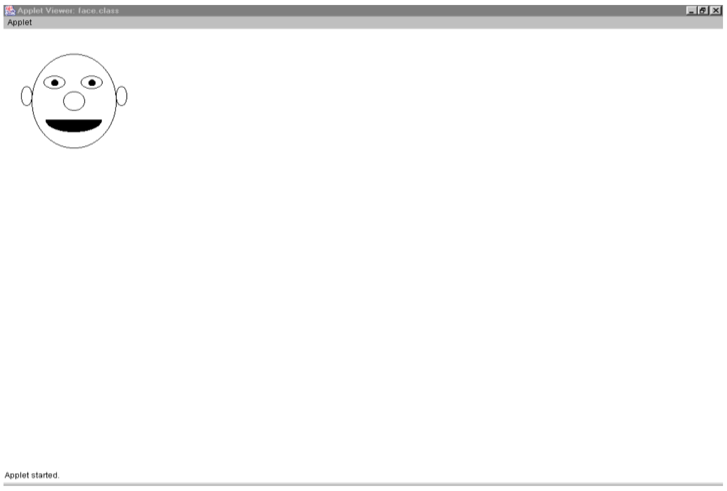
Example Program: Program for Human Face Drawing

```
import java.awt.*;
import java.applet.*;
public class face extends Applet
{
  public void paint(Graphics g)
  {
        g.drawOval(40,40,120,150);
        g.drawOval(57,75,30,20);
        g.drawOval(110,75,30,20);
        g.fillOval(68,81,10,10);
        g.fillOval(121,81,10,10);
        g.drawOval(85,100,30,30);
        g.fillArc(60,125,80,40,180,180);
        g.drawOval(25,92,15,30);
        g.drawOval(160,92,15,30);
   }
}
```

When run applet program using the following HTML file display the output as given below.

```
<html>
        <applet
                code = face.class
                width =300
                height=200
        >
        </applet>
    </html>
```

Output



5.4.6 Drawing Polygons

Polygons are shapes with *many sides*. The **drawPolygon ( )** method is used to draw a polygon. This method takes *three* arguments:

- ❖ An array of integers containing x coordinates.
- ❖ An array of integers containing y coordinates.
- ❖ An integer for the total number of points.

It is obvious that x and y arrays should be of the *equal size* and we must repeat the first point at the end of the array for closing the polygon.

We can also draw a filled polygon by using the **fillPolygon ( )** method.

Second, way of calling the methods **drawPolygon ( )** and **fillPolygon ( )** is to use a **Polygon** object. The **Polygon** class enables us treat the polygon as an object. This approach involves the following steps.

- ❖ Defining x coordinate values as an array.
- ❖ Defining y coordinate values as an array.
- ❖ Defining the number of points n.
- ❖ Creating a **Polygon** object and initializing it with the above x, y and n values.
- ❖ Calling the method **drawPolygon ( )** or **fillPolygon (**) with the **Polygon** object as argument.

The **Polygon** class is useful to add points to the Polygon.

We first create an empty polygon and then add points to it one another. Finally call **drawPolygon ( )** method using the **poly** object as an argument to complete the process of drawing the polygon.

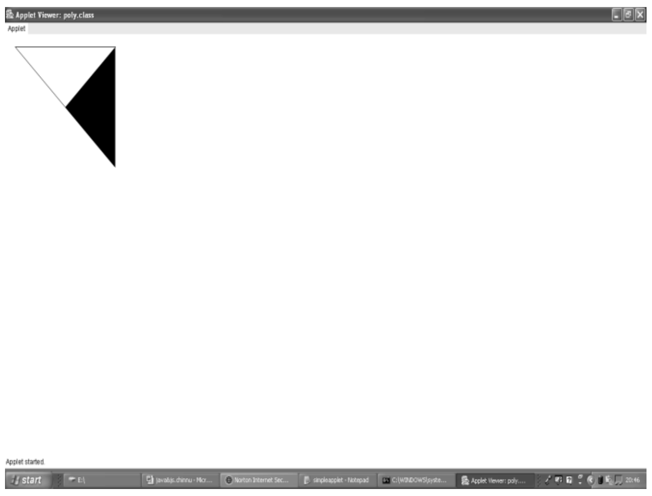
Example Program: Program for drawing polygon

```
import java.awt.*;
import java.applet.*;
public class poly extends Applet
{
        int x1[]={20,120,220,20};
         int y1[]={20,120,20,20};
        int n1=4;
        int x2[]={140,220,220,140};
        int y2[]={140,20,220,140};
        int n2=4;
         public void paint(Graphics g)
         {
                g.drawPolygon(x1,y1,n1);
                 g.fillPolygon(x2,y2,n2);
         }
}
```

When run applet program using the following HTML file display the output as given below.

```
<html>
        <applet
                code = poly.class
                width =300
                height=200
        >
        </applet>
</html>
```

Output



5.4.7 Line Graphs

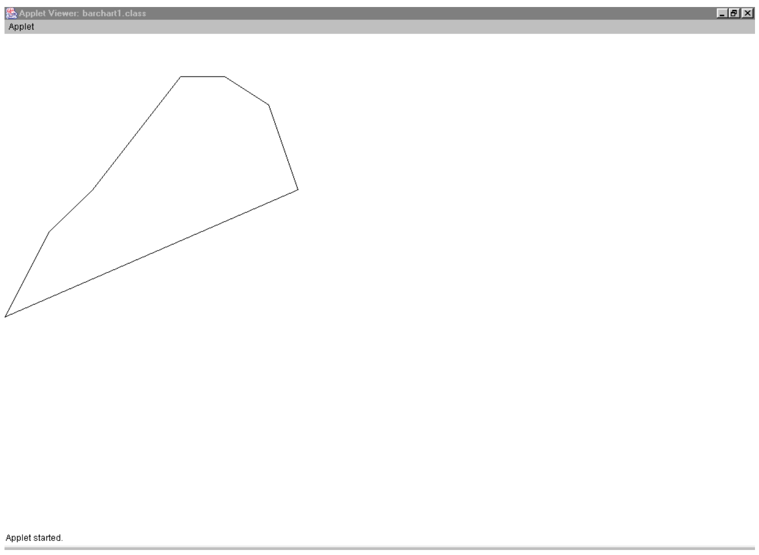
We can design applets to draw line graphs to illustrate graphically the relationship between two variables.

```
import java.awt.*;
import java.applet.*;
public class linegraph extends Applet
{
        int x1[]={0,60,120,180,240,300,360, 400};
        int y1[]={400, 280,220,140,60, 60,100,220};
        int n1=x1.length;
        public void paint(Graphics g)
         {
                g.drawPolygon(x1,y1,n1);
         }
}
```

When run applet program using the following HTML file display the output as given below.

```
        <html>
                <applet
                        code = linegraph.class
                        width =300
                        height=200
                >
                </applet>
        </html>
```

Output



5.4.8 Using Control Loops In Applet

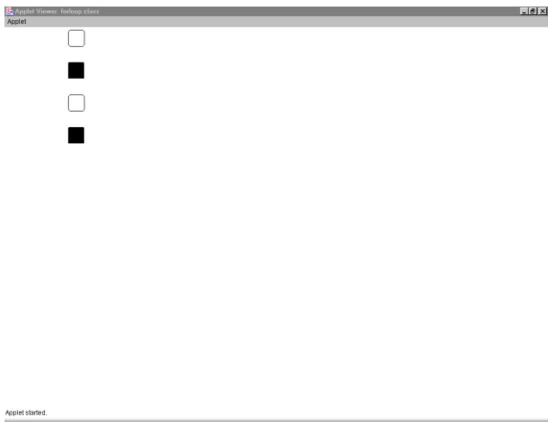
We can use all control structures in an applet.

```
import java.awt.*;
import java.applet.*;
public class forloop extends Applet
{
     public void paint(Graphics g)
    {
        for(int i = 0; i <= 3; i++)
        {
            if((i % 2) == 0 )
                g.drawRoundRect(120,    i*60+5,    30,
30,10,10);
            else
            g.fillRoundRect(120, i*60+5, 30, 30, 5, 5);
        }
    }
}
```

When run applet program using the following HTML file display the output as given below.

```
<html>
    <applet
        code = linegraph.class
        width =300
        height=200
    >
    </applet>
</html>
```

Output



5.4.9 Drawing Bar Charts

Applets can be designed to display bar charts, which are commonly used in comparative of data.

The method **getParameter( )** is used to fetch the data values from the HTML file. The **getParameter( )** returns only string values and therefore we use the wrapper class method **parseInt** to convert strings to integer values.

Example Program: Program to draw a barchart

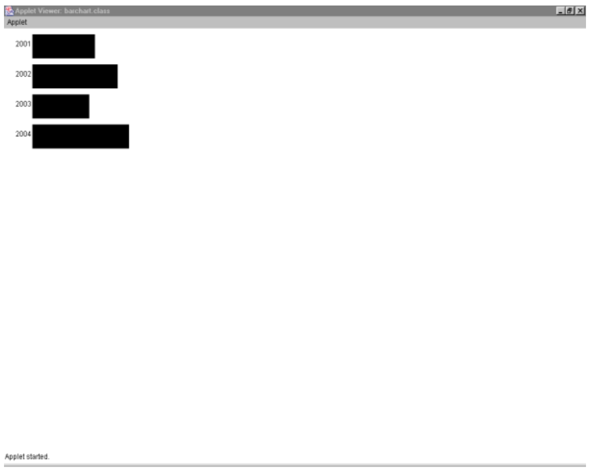
```
import java.awt.*;
import java.applet.*;
public class barchart extends Applet
{
        int n=0;
        String label[];
        int value[];
        public void init()
       {
           try
          {
                  n = Integer.parseInt(getParameter("columns"));
                 label = new String[n];
                 value = new int[n];
                 label[0] = getParameter("label1");
                 label[1] = getParameter("label2");
                 label[2] = getParameter("label3");
                 label[3] = g etParameter("label4");
                 value[0] = Integer.parseInt(getParameter("c1"));
                 value[1] = Integer.parseInt(getParameter("c2"));
                 value[2] = Integer.parseInt(getParameter("c3"));
                 value[3] = Integer.parseInt(getParameter("c4"));
           }
           catch(NumberFormatException e)
           {  }
       }
       public void paint(Graphics g)
       {
            for(int i=0;i<n;i++)
            {
                 g.drawString(label[i],20,i*50+30);
```

```
                    g.fillRect(50,i*50+10,value[i],40);
                }
            }
    }
```

When run applet program using the following HTML file display the output as given below.

```
<html>
        <applet code=barchart.class  width=500 height=500>
        <param name="columns" value="4">
        <param name="c1" value="110">
        <param name="c2" value="150">
        <param name="c3" value="100">
        <param name="c4" value="170">
        <param name="label1" value="2001">
        <param name="label2" value="2002">
        <param name="label3" value="2003">
        <param name="label4" value="2004">
        </applet>
</html>
```

Output



5.4.10 Self Assessment Questions

Fill in the blank

1.___________________ method in Graphic class which is used to draw both square and rectangle pictures.

2. Polygons are shapes with ___________*sides*.

True or False

1. drawPolygon( ) method is one of the method of Graphics class.

2. Method getParameter( ) return all the kinds of values.

Multiple choices

1. Which method is used to draw circle as well as ovals

a) drawLine( ) b) fillrect( )

c) drawOval( ) d) drawstring()

Short Answer

1. How many pair of coordinates used in drawLine( ) method?

5.5 Managing Input / Output Files

5.5.1 Introduction

The variables and arrays are used for storing data inside the programs. This approach yields the following problems.

- ❖ The data is lost either when variable goes out of scope or when the program is terminated.
- ❖ It is difficult to handle large amount of data using variables and arrays.

We can overcome these problems by storing data on *secondary storage devices* such as floppy disks or hard disks. The data is stored in these devices using the concept of *files*.

A *file* is a collection of related records. A *record* is composed of fields and a *field* is a group of characters as shown in Fig.5.5.1. Storing and managing data using files is called as *file processing* which includes tasks such as creating files, updating files and manipulation of data.

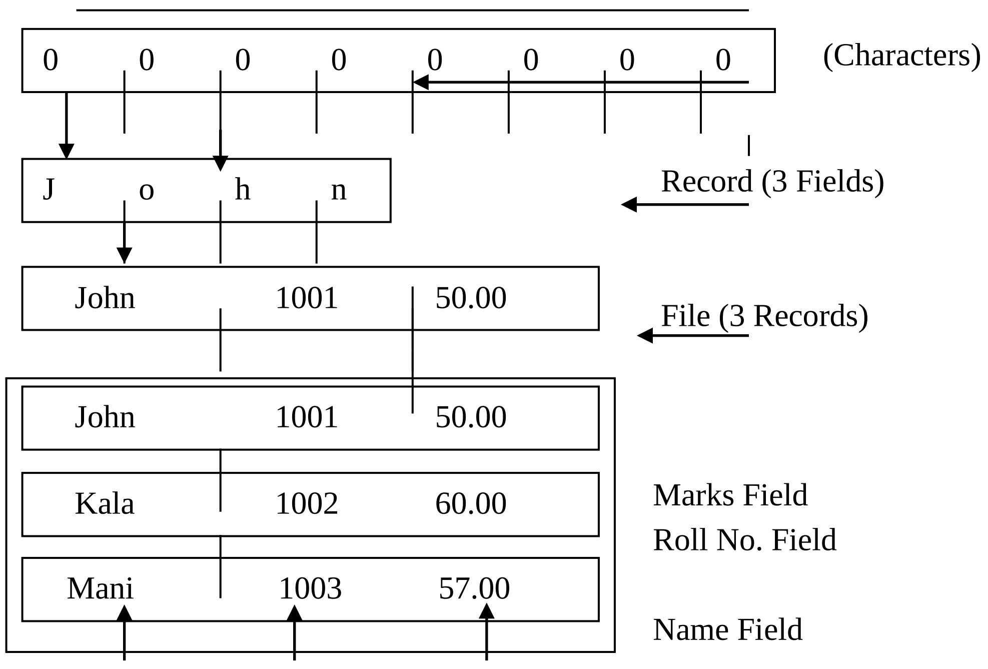


Fig. 5.5.1 Data representation in Java files

Java supports many features for managing input and output of data using files. Reading and writing of data in a file can be done at the level of *bytes* or *characters* or *fields*. Java also provides capabilities to read and write *class object* directly. The process of reading and writing objects is called *object serialization*.

5.5.2 Concepts Of Streams

In file processing, input refers to the flow of data into a program and output means the flow of data out of a program. Input to a program may come from key board, the mouse, the memory, the disk or another program and output from a program may go to the screen, the printer, memory, the disk, or another program (See Fig 5.5.2).

Java uses the concept of streams to represent the ordered sequence of data. A stream presents a uniform, easy-to-use, object-oriented interface between the program and the input/output devices.

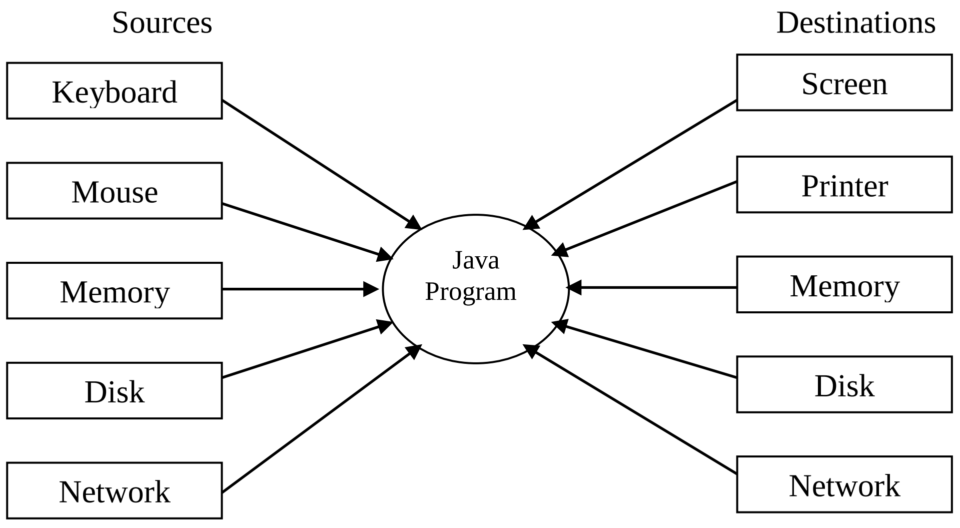
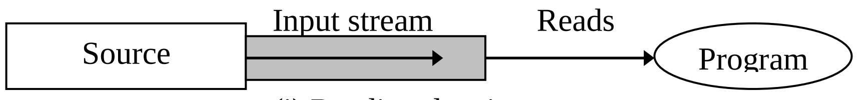


Fig. 5.5.2 Relationship of Java program with I/O devices

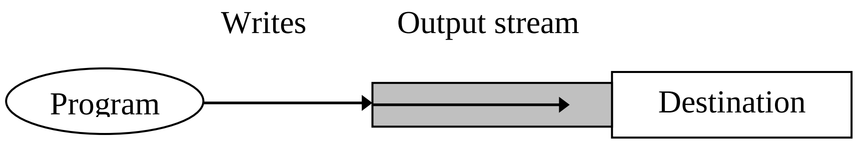
Java streams are classified into *two basic types*.

- ❖ *Input stream* extracts (i.e. reads) data from the source (file) and sends it to the program.
- ❖ *Output stream* takes data from the program and sends (i.e. writes) it to destination (file).

Fig. 5.5.3 illustrates the use of input and output streams. In both the cases, the program does not know the details of end points (i.e. source and destination).



(i) Reading data into a program



(ii) Writing data to a destination

Fig. 5.5.3 Using input and output streams

5.5.3 Stream Classes

The **java.io** package contains a large number of stream classes that provide capabilities for processing all types of data. These classes may be categorized into two groups based on the data type on which type operate.

- ❖ Byte stream classes that provide support for handling I/O operations on bytes.
- ❖ Character stream classes that provide support for managing I/O operations on characters.

Fig. 5.5.4 shows how stream classes are grouped based on their functions.

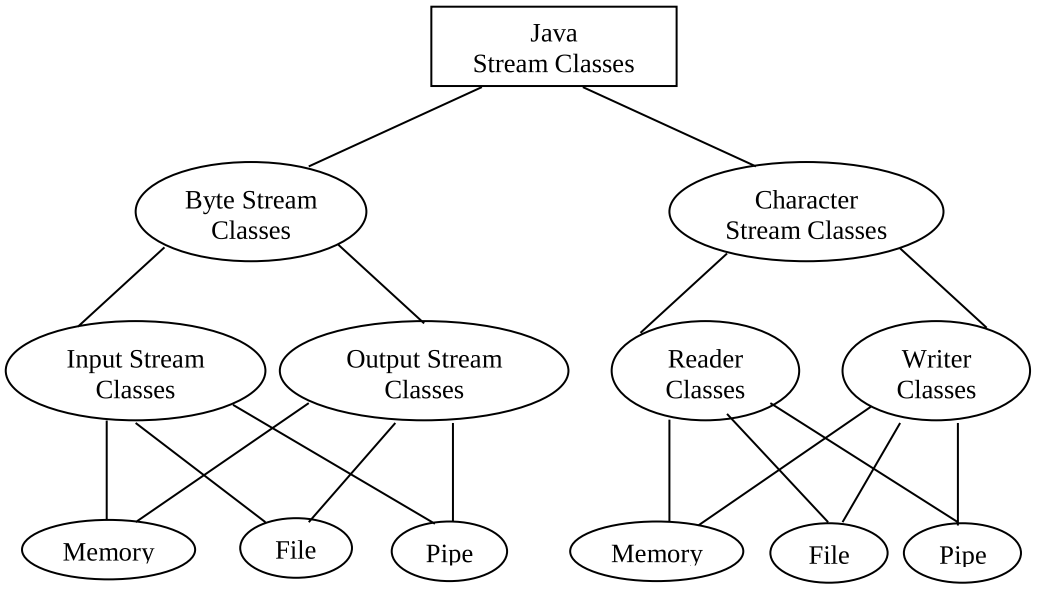


Fig.5.5.4 Classification of Java Stream Classes

5.5.4 Byte Stream Classes

Java provides two kinds of byte stream classes: I*nput stream classes* and O*utput stream classes*.

Input Stream Classes

Input stream classes that are used to read 8-bit bytes include super class known as **InputStream** and a number of subclasses for supporting various input-related functions. Fig 5.5.5 shows the class hierarchy of input stream classes. The **InputStream** class defines methods for performing input functions (See Table 5.5.1) such as

- ❖ Reading bytes
- ❖ Closing streams
- ❖ Marking positions in streams
- ❖ Skipping ahead in a stream
- ❖ Finding the number of bytes in a stream

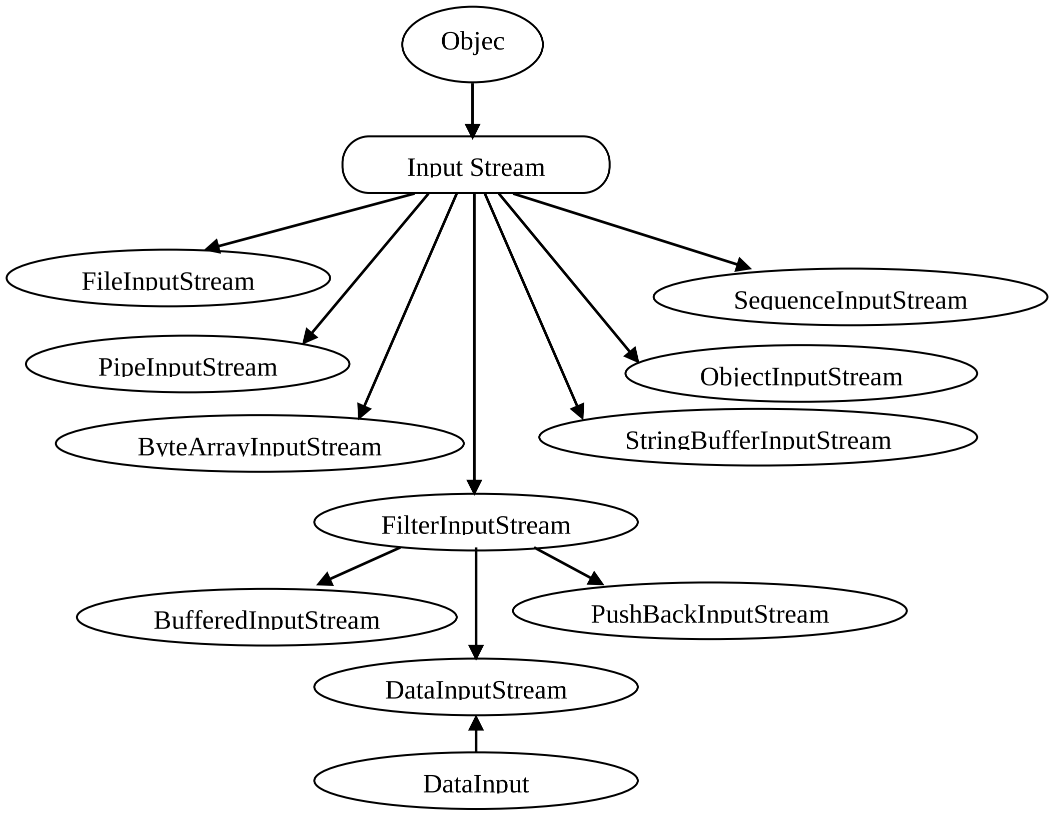


Fig. 5.5.5 Hierarchy of input stream classes

The class **DataInputStream** extends **FilterInputStream** and implements the interface **DataInput** . Therefore, the **DataInputStream** class implements the methods described in **DataInput** in addition to using the methods of **InputStream** class. The **DataInput** interface contains the following methods:

- ❖ readShort( )
- ❖ readInt( )
- ❖ readLong( )

- readFloat( )
- readUTF( )
- readDouble( )
- readLine( )
- readChar( )
- readBoolean( )

Table 5.5.1 InputStream Methods

Method	Description
read( )	Reads a byte from the input stream
read(byte b[ ])	Reads an array of bytes into b
read(byte b[ ], int n, int m)	Reads m bytes into b starting from nth byte
available( )	Gives number of bytes available in the input
skip(n)	Skips over n bytes from the input stream
reset( )	Goes back to the beginning of the stream
close( )	Closes the input stream

Output Stream Classes

Output stream classes are derived from the base class **OutputStream** as shown in Fig 5.5.6. The **OutputStream** is an abstract class and therefore we cannot instantiate it. The several subclasses of the **Outputstream** can be used for performing the output operations. Table 5.5.2 gives a description of all the methods defined by the **OutputStream** class. The **OutputStream** includes methods that are designed to perform the following tasks:

- Writing bytes
- Closing streams
- Flushing streams

The class **DataOutputStream**, counterpart of **DataInputStream**, implements the interface **DataOutput**. Therefore, the **DataOutputStream** class implements the methods described in **DataOutput** in addition to using the methods of **OutputStream** class. The **DataOutput** interface contains the following methods:

- writeShort( )
- writeInt( )
- writeLong( )
- writeFloat( )
- writeUTF( )
- writeDouble( )
- writeLine( )
- writeChar( )
- writeBoolean( ) object

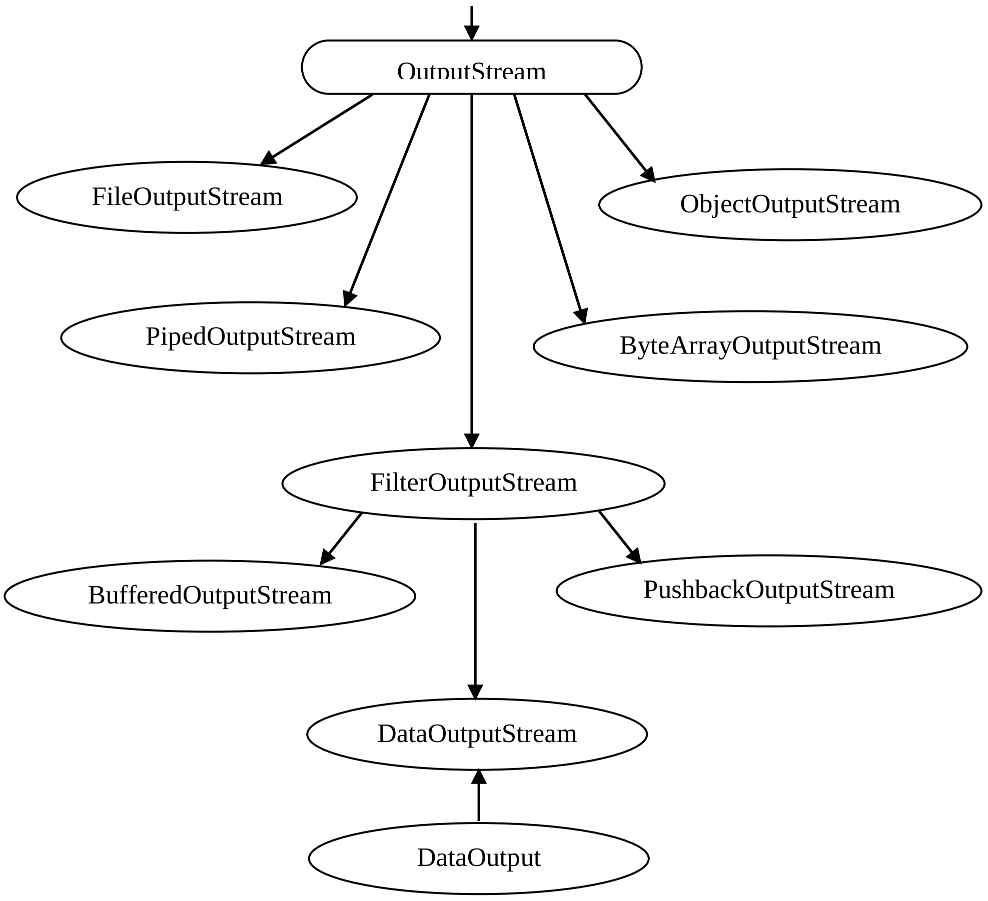


Fig.5.5.6 Hierarchy of output stream classes

Table 5.5.2 OutputStream Methods

Method	Description
write( )	Writes a byte to the output stream
write(byte b[ ])	Writes all bytes in the array b to the output stream
write(byte b[ ], int n, int m)	Writes m bytes from array b starting from nth byte
close( )	Closes the output stream
flush( )	Flushes the output stream

5.5.5 Character Stream Classes

Character streams can be used to read and write 16-bit Unicode characters. There are two kinds of character stream classes: *Reader stream classes* and W*riter stream classes.*

Reader Stream Classes

Reader stream classes are designed to read character from the files. The **Reader** class is the base class for all other classes in this group as shown in Fig. 5.5.7. The **Reader** class contains methods that are identical to those available in the **InputStream** class except is designed to handle characters (See Table 5.5.1).

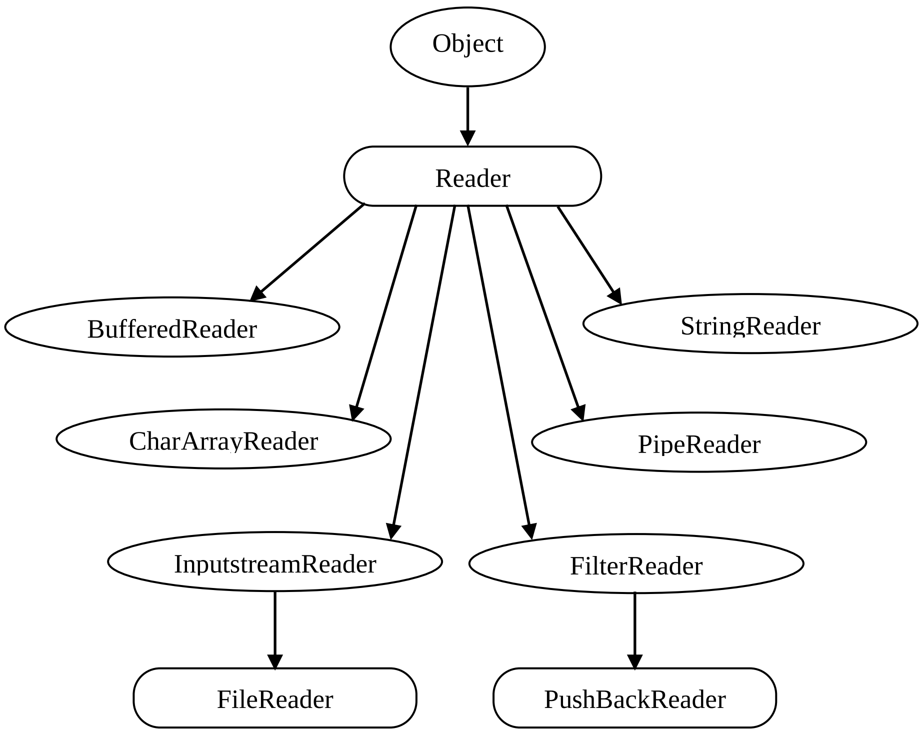


Fig. 5.5.7 Hierarchy of reader stream classes

Writer Stream Classes

The writer stream classes are designed to perform all output operations on files. The **Writer stream classes** are designed to *write characters*. The Write class is an abstract class, which acts as a base class for all the other writer stream classes as shown in Fig 5.5.8. This base class contains methods that are identical to those available in the **OutputStream** class except is designed to handle characters (See Table 5.5.2).

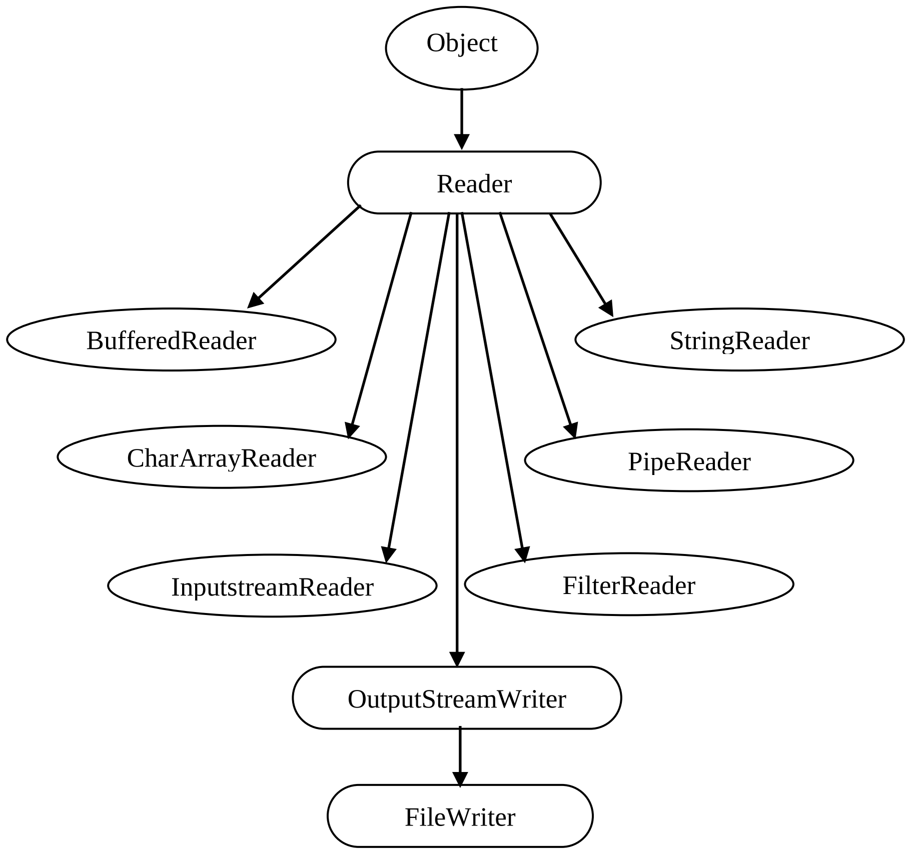


Fig. 5.5.8 Hierarchy of writer stream classes

5.5.6 Using Stream

All the classes are known as I/O classes, not all of them are used for reading and writing operations only. Some perform operations such as buffering, filtering, data conversion, counting and concatenation while carrying out I/O tasks.

5.5.7 Other Useful I/O Classes

The **java.io** package supports many other classes for performing certain specialized functions. They include among others:

- ❖ Random Access File
- ❖ Stream Tokenizer

The **RandomAccessFile** enables us to read and write bytes, text and Java data types to any location in a file (when used with appropriate access permissions). This class extends object class and implements **DataInput** and **DataOutput** interfaces as shown in Fig.5.5.9. This forces the **RandomAccessFile** to implement the methods described in both these interfaces.

The class **StreamTokenizer**, a subclass of object can be used for breaking up a stream of text from an input text file into meaningful pieces called *tokens*. The behaviour of the **StreamTokenizer** class is similar to that of the **StringTokenizer** (class of **java.util** package) that breaks a string into its component tokens.

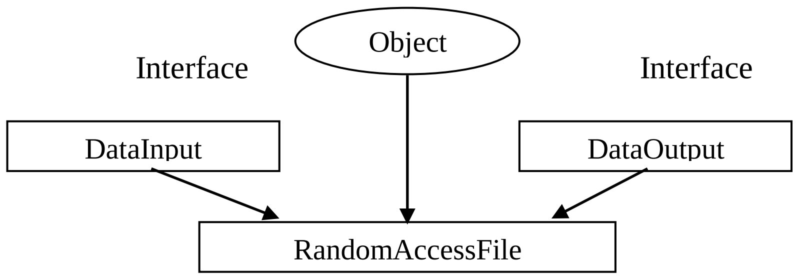


Fig. 5.5.9 Implementation of the RandomAccessFile

5.5.8 Using The File Classes

The **java.io** package includes a class known as the **File class** that provides support for creating files and directories. The class includes several constructors for instantiating the **File** objects. This class also contains *several methods* for supporting the operations such as

- ❖ Creating a file
- ❖ Opening a file
- ❖ Closing a file
- ❖ Deleting a file
- ❖ Getting the name of a file
- ❖ Getting the size of a file
- ❖ Checking the existence of a file
- ❖ Renaming a file
- ❖ Checking whether the file is writable
- ❖ Checking whether the file is readable

5.5.9 Input / Output Exceptions

When creating files and performing I/O operations on them, the system may generate I/O related exceptions. The basic I/O related exception classes and their functions are given in Table.5.5.4.

Each I/O stream statement or group of I/O statements must have an exception handler around it as shown below or the method must declare that it throws an IOException.

```
try
{
        ---------------
        ---------------   // I/O statements
        ---------------
}
catch( IOException   e)
{
```

```
            -------------- // Message output statement
        }
```

Table 5.5.4 Important I/O Exception Classes and their Functions

I/O Exception class	**Function**
EOFException	Signals that end of the file or end of streamhas been reached unexpectedly during input
FileNotFoundException	Informs that a file could not be found
InterruptedIOException	Warns that an I/O operations has been interrupted
IOException	Signals that an I/O exception of some sort has occurred

5.5.10 Creation Of Files

To create and use a disk file, it is necessary to decide the following about the file and its intended purpose:

- ❖ Suitable name for the file
- ❖ Data type to be stored
- ❖ Purpose (reading, writing, or updating)
- ❖ Method of creating the file

A filename is unique string of characters that helps identify a file on the disk. The length of a filename and characters allowed are dependent on the OS on which the Java program is executed. A filename may contain two parts, a primary name and an optional period with extension.

Examples:

Input.data	salary
Test.doc	student.txt
Inventory	rand.dat

Data type is important to decide the type of file stream classes to be used for handling the data. We should decide whether the data to be handled is in the form of characters, bytes or primitive type.

The purpose of using a file must also be decided before using it. For example, we should know whether the file is created for reading only, or writing only, or both the operations.

For using a file, it must be opened first. This is done by creating a file stream and then linking it to the filename. A file stream can be defined using the classes of **Reader/Input Stream** for reading data and **Writer/Output Stream** for writing data. The common stream classes used for various I/O operations are given in Table. 5.5.5.

Table 5.5.5 Common Stream Classes used for I/O Operations

Source or Destination	Character		Bytes	
	Read	Write	Read	Write
Memory	CharArrayReader	CharArrayWriter	ByteArrayInputStream	ByteArrayOutputStream
File	FileReader	FileWriter	FileInputStream	FileOutputStream
Pipe	PipedReader	PipedWriter	PipedInputStream	PipedOutputStream

5.5.11 Reading / Writing Characters

Subclasses of **Reader** and **Writer** implement streams that can handle *characters*. The two subclasses used for handling characters in files are

- ❖ **FileReader** (for reading characters) and
- ❖ **FileWriter** (for writing characters).

Example Program : Program for Reading and Writing a Character

```
import java.io.*;
class WriteReadCharacter
{
    public static void main(String args[])
    {
            //write character
            try
            {
            // creates file stream fw and opens "city.dat"file
                    FileWriter fw = new FileWriter("city.dat");
                    int  ch;
                    // Read character through the keyboard
                    while((ch = System.in.read( )) != -1)
                    {
                        fw.write(ch);
                        // write character to the file "city.dat"
                    }
                    fw.close( );            // close file
             }
             catch(IOException e)
             {
                   System.out.println(e);
                   System.exit(-1);
            }
            // Read character
             int b;
             try
            {
                    //create file stream fr and  opens "city.dat" file
                    FileReader fr=new FileReader("city.dat");
                    // read character from the file "city.dat"
                    while((b = fr.read( )) != -1)
                    {
                        System.out.print((char)b);
                              // write character on monitor
                    }
                    fr.close();        // close file
            }
            catch(IOException e)
            {
                     System.out.println(e);  System.exit(-1);
            }
        }
    }
```

Output of program

```
S
A
L
E
M
Z
S
A
L
E
M
```

5.5.12 Reading / Writing Bytes

Two commonly used classes for handling bytes are

- ❖ **FileInputStream** Classes.
- ❖ **FileOutputStream** Classes.

How File Input Stream class is used for reading bytes from a file. The program reads an existing file and displays its bytes on the screen. The following program uses both **FileInputStream** and **FileOutputStream** classes to copy files. We need to provide a source filename for reading and a target filename for writing.

Example Program : Program for Writing and Reading bytes

```
import java.io.*;
class WriteReadByte
{
  public static void main(String args[])
{
  byte cities[]={'S' , 'A' , 'L' , 'E' , 'M' , '\n' , 'M' , 'A', 'D' , 'R' , 'A' , 'S' ,
'\n'};
        // Write data to the file
         try
         {
              FileOutputStream fos = new
FileOutputStream("city.dat");
              fos.write(cities);
              fos.close();
         }
         catch(IOException e)
         {
               System.out.println(e);
               System.exit(-1);
         }
        // Read data from a file
         int b;
         try
        {
              FileInputStream fis=new FileInputStream("city.dat");
              while(( b = fis.read( )) ! = -1)
               {
                    System.out.print((char)b);
              }
              fis.close();
        }
        catch(IOException e)
        {
```

```
                        System.out.println(e);
                       System.exit(-1);
                }
        }
        }
```
Output of Program

```
        SALEM
        MADRAS
```

5.5.13 Handling Primitive Data Types

If we want to read/write the primitive data types such as integers and doubles, we can use filter classes as wrappers on existing input and output streams to filter data in the original stream. The *two filter classes* used for creating “data streams” for handling primitive types are

- **DataInputStream** Classes
- **DataOutputStream** Classes

The hierarchy of data stream classes as shown in Fig.5.5

A data stream for input can be created as

```
FileInputStream fis = new FileInputStream(infile);
DataInputStream dis = new DataInputStream(fis);
```

This creates the *input file stream* **fis** and then creates the *input data stream* **dis**.

A data stream for ouput can be created as

```
FileOutputStream fos = new FileOutputStream(outfile);
DataOutputStream dos = new DataOutputStream(fos);
```

This create the *output file stream* **fos** and then create the *output data stream* **dos**.

Note that the file objects **infile** and **outfile** must be initialized with appropriate file names before they are used. We may also use file names directly in place of file objects.

In the below program first creates “prim.dat” file and then writes a primitive data type values into it using data output stream. At the end of writing , the streams are closed.

The program also creates a data input stream, and connects it to “prim.dat” file. It then reads data from the file and displays them on the screen. Finally, it closes the streams.

Example Program: Program for Writing and reading primitive data

```
        import java.io.*;
        class writereadprimitive
        {
```

```
        public static void main(String args[]) throws IOException
        {
                File f=new File("prim.dat");
                 // write primitive data to the “prim.dat” file
                FileOutputStream fos=new FileOutputStream(f);
                DataOutputStream dos=new DataOutputStream(fos);
                dos.writeInt(2008);
                dos.writeDouble(443.56);
                dos.writeBoolean(false);
                dos.writeChar('P');
                dos.close( );
                fos.close( );
                 // read primitive data from the “prim.dat” file
                FileInputStream fis=new FileInputStream(f);
                DataInputStream dis=new DataInputStream(fis);
                System.out.println(dis.readInt());
                System.out.println(dis.readDouble());
                System.out.println(dis.readBoolean());
                System.out.println(dis.readChar());
                dis.close();
                fis.close();
        }
}
```

Output of Program

```
2008
443.56
false
P
```

5.5.14 Concatenation And Buffering Files

To combine two or more input streams (files) into a single input stream (file). This process is known as *concatenation of files* and is achieved using the **SequenceInputStream** class. One of the constructors of this class takes two **InputStream** objects as arguments and combines them to construct a single input stream.

Java also supports creation of buffers to store temporarily data that is read from or written to a stream. The process is known as *buffered I/O* operation. A buffer sits between the program and the source (or destination) and functions like a filter. Buffers can be created using the **BufferedInputStream** and **BufferedoutputStream** classes.

Example Program: Program for concatenation and buffering

```
import java.util.*;
import java.io.*;
class conbuf
{
  public static void main(String args[ ] ) throws IOException
  {
            // open the files to be concatenated
            FileInputStream  f1  = new FileInputStream ( "text1.dat");
            FileInputStream  f2  = new FileInputStream ( "text2.dat");
            // concatenate f1 and f2 into f3
            SequenceInputStream f3 = new SequenceInputStream( f1, f2);
            // create buffered input and output streams
            BufferedInputStream inbuf = new BufferedInputStream(f3);
             BufferedOutputStream outbuf =
                                    new BufferedOutputStream(System.out);
            // read and write till the end of buffers
            int c;
            while( ( c= inbuf.read( ) ) != -1)
            {
                        outbuf.write((char) c);
            }
            inbuf.close( );
            outbuf.close( );
            f1.close( );
            f2.close( );
   }
}
```

The entire process of concatenation, buffering and displaying the contents of two files is illustrated in Fig.5.5.10.

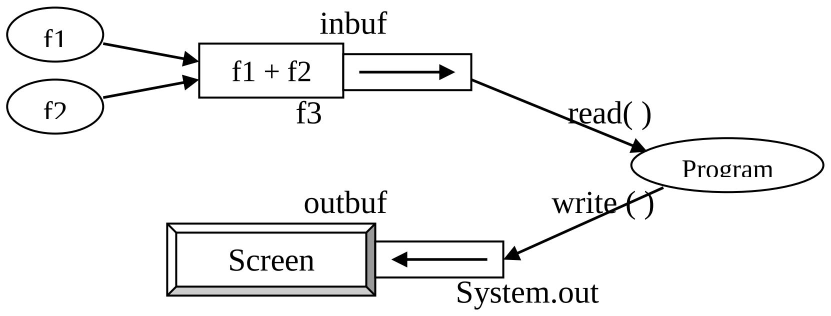


Fig. 5.5.10 Illustration of concatenation and buffering

Given the content of “text1.dat” and “text2.dat” as

Contents of “text1.dat:

    Java development kit

Contents of “text1.dat:

    welcome

Then, the Output Of Program

```
 Java development kit
Welcome
```

5.5.15 Random Access Files

The **RandomAccessFile** class supported by the **java.io** package allows us to create files that can be used for reading and writing data with random access. That is, we can “jump around” in the file while using the file. Such files are known as *random access files.*

A file can be created and opened for random access by giving a mode string as parameter to the constructor when we open the file. We can use one of the following *two mode* strings:

- ‘r’ for reading only
- ‘rw’ for both reading and writing

An existing file can be updated using the ‘rw’ mode. Random access files support a pointer known as *file pointer* that can be moved to arbitrary positions in the file prior to reading or writing. The file pointer is moved using the method **seek()** in the **RandomAccessFile** class.

Example Program: Program for writing and reading from RandomAccessFile

```
import java.io.*;
class random
{
     public static void main(String args[])
    {
         try
         {
             RandomAccessFileraf =
                      new RandomAccessFile("ran.dat","rw");
             raf.writeInt(2000);
             raf.writeDouble(43.56);
             raf.writeBoolean(false);
             raf.writeChar('p');
             raf.seek(0);
             System.out.println(raf.readInt());
             System.out.println(raf.readDouble());
```

```
                        System.out.println(raf.readBoolean());
                        System.out.println(raf.readChar());
                    }
                    catch(IOException e)
                    {   }
                }
        }
```

Output Of Program

```
        2000
        43.56
        false
        p
```

5.5.16 Interactive Input And Output

The process of reading data from the keyboard and displaying output on the screen is known as interactive I/O. There are two types of interactive I/O. First one is referred to as simple interactive I/O, which involves simple input from the keyboard and simple output in a pure text form. The second type is referred to as graphical interactive I/O, which involves input from various input devices and output to graphical environment on frames and applets.

In this section we shall consider how to use interactive I/O while handling files.

Simple Input and Output

The **System** class contains three i/o objects, namely **System.in, System.out,** and **System.err** where **in, out** and **err** are static variables. The variable **in** is of **InputStream** type and the other two are of **PrintStream** type. We use this objects to input from keyboard, output to the screen and display error messages.

To perform keyboard input for primitive data types, we need to use the objects of **DataInputStream** and **StringTokenizer** classes.

Example Program: Program for Creating files interactively from keyboard input

```
import java.util.*;
import java.io.*;
class inventory
{
        static DataInputStream din = new DataInputStream ( System.in);
        static StringTokenizer st;
        public static void main(String args[ ] ) throws IOException
        {
                DataOutputStream dos = new DataOutputStream
                                        ( new  FileOutputStream(" invent.dat"));
                System.out.println("Enter code number");
                st = new StringTokenizer( din.readLine( ) );
                int code = Integer.parseInt( st.nextToken( ));
                System.out.println("Enter number of items");
                st = new StringTokenizer( din.readLine( ) );
                int items = Integer.parseInt( st.nextToken( ));
                System.out.println("Enter cost");
                st = new StringTokenizer( din.readLine( ) );
                double cost = new Double( st.nextToken( )). doubleValue( );
                dos.writeInt(code);
                dos.writeInt(items);
                dos.writeDouble(cost);
                dos.close( );
                DataInputStream dis = new DataInputStream
                                ( new  FileInputStream(" invent.dat"));
                int codeno = dis.readInt( );
                int totitems = dis.readInt( );
                double itemcost = dis.readDouble( );
                double totcost = totitems * itemcost;
                dis.close( );
                System.out.println("Code Number:" + codeno);
                System.out.println("Item cost:" + itemcost);
                System.out.println("Total items:" + totitems);
                System.out.println("Total cost  :" + totcost);
        }
}
```

Output Of Program

```
        Enter code number
        100
        Enter number of items
        300
        Enter cost
        200
        Code Number:100
        Item cost:200.0
        Total items:300
```

```
Total cost  :60000.0
```

Graphical Input and Output

The program uses the TextField classes to receive information from user at keyboard and then write information to a file (tele.data) and also read information from file.

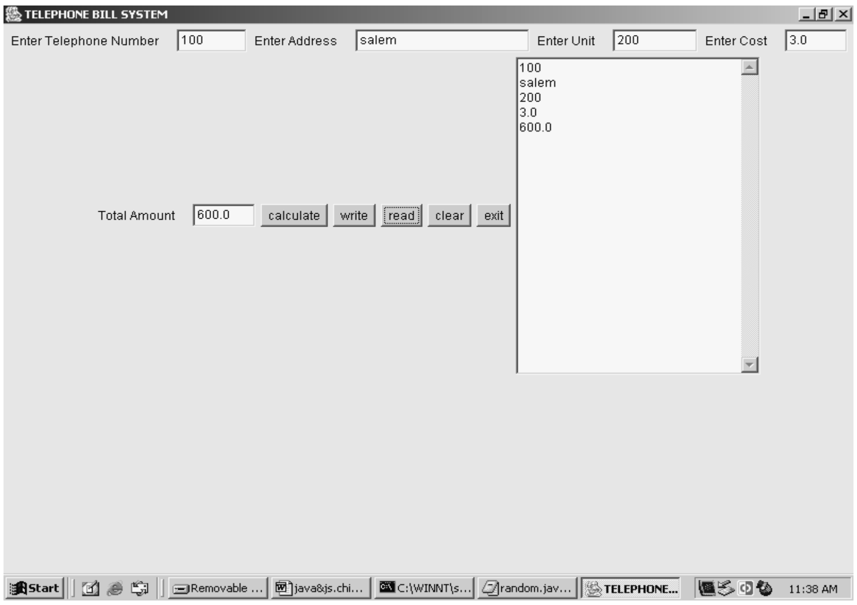
Example Program: Program for file operation using I/O stream

```
import java.io.*;
import java.awt.event.*;
import java.awt.*;
import java.applet.*;
public class tele extends Frame implements ActionListener
{
  TextField t1,t2,t3,t4,t5;
  Label l1,l2,l3,l4,l5;
  Button b1,b2,b3,b4,b5;
  TextArea ta;
 tele()
  {
      setLayout(new FlowLayout());
      setTitle("TELEPHONE BILL SYSTEM");
      setBackground(Color.yellow);
      l1 = new Label("Enter Telephone Number");
      t1 = new TextField(6);
     l2 = new Label("Enter Address");
      t2 = new TextField(20);
      l3 = new Label("Enter Unit");
      t3 = new TextField(8);
     l4 = new Label("Enter Cost");
     t4 = new TextField(5);
     l5 = new Label("Total Amount");
     t5 = new TextField(5);
     ta = new TextArea(20,30);
    b1 = new Button("calculate");
    b2 = new Button("write");
    b3 = new Button("read");
    b4 = new Button("clear");
    b5 = new Button("exit");
   add(l1); add(t1);
```

```
          add(l2); add(t2);
          add(l3); add(t3);
          add(l4); add(t4);
          add(l5); add(t5);
          add(b1); add(b2);
          add(b3); add(b4);
          add(b5); add(ta);
          b1.addActionListener(this);
          b2.addActionListener(this);
          b3.addActionListener(this);
          b4.addActionListener(this);
         b5.addActionListener(this);
       }
       public void actionPerformed(ActionEvent ae)
      {
        String s=ae.getActionCommand();
        try
        {
          if(s.equals("calculate"))
         {
                 float
f=Integer.parseInt(t3.getText())*Float.parseFloat(t4.getText());
                  t5.setText(String.valueOf(f));
        }
         if(s.equals("write"))
        {
                 FileOutputStream fos= new FileOutputStream("hai.dat");
                 DataOutputStream dos= new DataOutputStream(fos);
                 dos.writeInt(Integer.parseInt(t1.getText()));
                 dos.writeUTF(t2.getText());
                 dos.writeInt(Integer.parseInt(t3.getText()));
                 dos.writeFloat(Float.parseFloat(t4.getText()));
                 dos.writeFloat(Float.parseFloat(t5.getText()));
                 dos.close();     fos.close();
         }
        if(s.equals("read"))
        {
                  FileInputStream fis= new FileInputStream("tele.dat");
```

```
                    DataInputStream dis= new DataInputStream(fis);
                    ta.append(dis.readInt()+"\n");
                    ta.append(dis.readUTF()+"\n");
                    ta.append(dis.readInt()+"\n");
                    ta.append(dis.readFloat()+"\n");
                   ta.append(dis.readFloat()+"\n");
                   dis.close();      fis.close();
            }
           if(s.equals("clear"))
           {
                    t1.setText(" ");
                    t2.setText(" ");
                    t3.setText(" ");
                    t4.setText(" ");
                    t5.setText(" ");
                    ta.setText(" ");
            }
           if(s.equals("exit"))
           {
                   System.exit(0);
           }
     }
     catch(Exception e)
    {
           System.out.println(e);
     }
  }
  public static void main(String args[])
 {
   tele t=new tele();
   t.setSize(500,500);
   t.show();
 }
}
```

Output Of Program



```
Enter Telephone Number  100    Enter Address  salem    Enter Unit  200    Enter Cost  3.0

100
salem
200
3.0
600.0

Total Amount  600.0
```

5.5.17 Other Stream Classes

Object Stream

It is also possible to perform input and output operations on objects using the object streams. The object streams are created using the **ObjectInputStream** and **ObjectOutputStream** classes. In this case, we may declare records as objects and use the object classes to write and read these objects from files. This process is known as *object serialization.*

Piped Stream

Piped stream provide functionality for threads to communicate and exchange data between them. Figure shoes how two threads use *pipes* for communication. The write thread sends data to the read thread through a pipeline that connects an object of **PipedInputStream** to an Object of **PipedOutputStream**. The objects **InputPipe** and **OutputPipe** are connected using the **connect( )** method.

Pushback Streams

The pushback streams created by the classes **PushbackInputStream** and **PushbackReader** can be used to push a single byte or a character (that was previously read) back into the input stream so that it can be reread. This is commonly used with parsers. When a character indicating a new input token is read, it is pushed back into the input stream.

Filtered Streams

Java supports *two* abstract classes, namely **FilterInputStream** and **FilterOutputStream** that provide the basic capability to create input and output streams for filtering input/output in a number of ways. These streams, known as *filters*, sit between an input stream and an output stream and perform some optional processing on the data they transfer. We can combine filters to perform a series of filtering operations as shown in Fig 5.5.11. Note that we used **DataInputStream** and **DataOutputStream** as filters in the program for handling primitive type data.

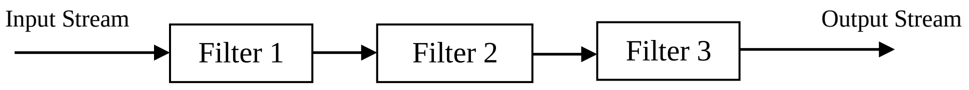


Fig. 5.5.11. The concept of using filters

5.5.18 Self Assessment Questions

Fill in the blank

1. ___________concept in Java is used to store the data or information permanently.
2. ______________ and _______________classes are used to store the characters and read the character in a file

True / False

1. DataOutputStream and DataInputStream classes are supported for handling primitive data values
2. Random access file class can not do write and read operations simultaneously

Multiple Choice

1. The process of reading and writing objects is called as

   a) object-serialization b) object-deserialization c) Object-serialization & object deserialization d) None of the above
2. FileInputStream and FileOutputStream classes are used to handles only

   a) 16 bit bytes b) 8 bit bytes

   c) 32 bit bytes d) None of the above
3. The following are classes of awt package

   a) TextField( ) b) Bulton( )

   c) Label( ) d) All of the above

Short Answer

1. List out types of a stream?

____

____

2. What is random access file?

____

____

Summary

Applets are Java programs developed for use on the Internet. They provides dynamic and interactive applications over the world wide web.

Java Graphics class supports many methods that enables us to draw many types of shapes. We can also use these methods to enhance the appearance of outputs of applets.

Managing Input/Output files means for storing and retrieving data. In this lesson we have discussed about various Input/Output stream classes and their related programs.

5.6 Unit Questions

1. Discuss the steps involved in developing and running a local applet
2. Describe the different stages in the life cycle of an applet
3. Develop an applet that receives three numeric values as input from the user and then displays the largest of three on the screen. Write a HTML page and test the applet.
4. Explain methods of Graphic Class with examples.
5. What are input and output streams? Explain with illustrations?
6. State the steps involved in creating a disk file.
7. Write a program that will count the number of characters. The number of words and lines in the file.
8. Write a program to crate a sequential file that could store details about 5 products. Details include product code, cost and number of items available and are provided through the keyboard
9. Write statements to create data streams for the following operators
   a) Reading primitive data from a file
   b) Writing primitive data to a file
10. Describe the most commonly used classes for handling i/o related exceptions?

5. 8 Answer For Self Assessment Questions

Answer 5.3.16

Fill in the blank

1. paint( ) 2. HyperText Markup Language 3. Textfields class

True / False

1. False 2. True 3. False

Multiple Choice

1. d 2. b

Short Answer

1. An applet developed locally and stored in a local system is known as local applet.

Answer 5.4.10

Fill in the blank

1. drawRect() 2. many sides.

True / False

1. True 2. False

Multiple Choice

1. c

Short Answer

1. The **drawLine ( )** method is used to draw a line, it takes two pair of coordinates, (x1, y1) and (x2, y2) as arguments and draws a line between them

Answer 5.5.17

Fill in the blank

1. File 2. FileWriter and FileReader

True / False

1. True 2. False

Multiple Choice

1. c 2. b 3. d

Short Answer

1. *Input stream* extracts (i.e. reads) data from the source (file) and sends it to the program.*Output stream* takes data from the program and sends (i.e. writes) it to destination (file).

2. We can "jump around" in the file while using the file. Such files are known as *random access files.*

NOTES

NOTES