PERIYAR UNIVERSITY

(NAAC 'A++' Grade with CGPA 3.61 (Cycle - 3)

State University - NIRF Rank 56 - State Public University Rank 25

SALEM - 636 011

CENTRE FOR DISTANCE AND ONLINE EDUCATION

(CDOE)

BACHELOR OF COMPUTER SCIENCE

SEMESTER - IV

JAVA PROGRAMMING LAB

(Candidates admitted from 2024 onwards)

PERIYAR UNIVERSITY

CENTRE FOR DISTANCE AND ONLINE EDUCATION (CDOE)

B.Sc. COMPUTER SCIENCE 2024 admission onwards

Core Course VIII

Java Programming Lab

Prepared by:

Centre for Distance and Online Education (CDOE)

Periyar niversityH Salem – 11.

CONTENTS

Program 1

This Java program aims to find and print prime numbers up to a user-provided integer `n`.

Program

```
import java.io.*;
public class prg1
{
        public static void main(String[] args)
        {
                int i,j,n;

                try
                {
                        DataInputStream dis = new DataInputStream(System.in);

                        System.out.println("Enter value: ");
                        n = Integer.parseInt(dis.readLine());

                        boolean[] prime = new boolean[n+1];
                        for (i=1;i<=n;i++)
                                prime[i]=true;


                        for (i=2;i<=n;i++)
                        {
                                for (j=2;j<i;j++)
                                {
                                        if (i%j==0)
                                        {
                                                prime[i]=false;
                                                break;
                                        }
                                }
                        }

                        System.out.println("Prime Number...");
                        for (i=2;i<n;i++)
                                if (prime[i])
                                        System.out.print (i + "\t");

                } catch (Exception ex)
                        {
                                System.out.println(ex);
                        }
        }
}
```

Explanation:

- The program imports `java.io.*` to use `DataInputStream` for reading user input.
- Defines a class named `prg1`.
- Declares variables `i`, `j`, and `n`.
- Uses `DataInputStream` to read user input (`n`).
- Integer.parseInt(dis.readLine())` converts the input string to an integer.
- Creates a boolean array `prime` to track whether numbers from `1` to `n` are prime.
- Initializes all entries in `prime` to `true`.
- Uses nested loops to determine prime numbers:
- Outer loop (`i`) iterates from `2` to `n`.
- Inner loop (`j`) checks divisibility of `i` by all numbers from `2` to `i-1`.
- If `i` is divisible by `j`, sets `prime[i]` to `false` (not prime) and breaks out of the inner loop.
- Prints a message indicating prime numbers are being displayed.
- Iterates through the `prime` array from `2` to `n-1`.
- Prints numbers (`i`) where `prime[i]` is `true` (indicating `i` is prime).

Output:

If the user enters `10` as input (`n = 10`), the output would be:

```
Prime Number...
2   3   5   7
```

This indicates that `2`, `3`, `5`, and `7` are the prime numbers less than `10`.

Overall, the program demonstrates basic input handling, prime number determination, and output in Java.

Program 2

This program aims to multiply two matrices.

Program

```
import java.io.*;
public class matmul
{
        public static void main (String args[])
        {
                int r1,r2,c1,c2,i,j,k;

                try
                {
                        DataInputStream dis = new DataInputStream(System.in);

                        System.out.println("Enter rows of Matrix A.. ");
                        r1=Integer.parseInt(dis.readLine());
                        System.out.println("Enter col of Matrix A.. ");
                        c1=Integer.parseInt(dis.readLine());
                        System.out.println("Enter rows of Matrix B.. ");
                        r2=Integer.parseInt(dis.readLine());
                        System.out.println("Enter col of Matrix B.. ");
                        c2=Integer.parseInt(dis.readLine());

                        if (c1 != r2)
                        {
                                System.out.println("Can't multiply the matrices..");
                        }
                        else
                        {
                                int a[][] = new int [r1][c1];
                                int b[][] = new int [r2][c2];
                                int c[][] = new int [r1][c2];

                                System.out.println ("Enter Matrix A values..");
                                for (i=0;i<r1;i++)
                                        for (j=0;j<c1;j++)
                                                a[i][j] = Integer.parseInt(dis.readLine());

                                System.out.println ("Enter Matrix B values..");
                                for (i=0;i<r2;i++)
                                        for (j=0;j<c2;j++)
                                                b[i][j] = Integer.parseInt(dis.readLine());

                                for (i=0;i<r1;i++)
                                        for (j=0;j<c2;j++)
                                        {
```

```
                        c[i][j]=0;
                        for (k=0;k<c1;k++)
                            c[i][j] = c[i][j] + a[i][k] * b[k][j];
                    }

                System.out.println ("Result...");
                for (i=0;i<r1;i++)
                {
                    System.out.println();
                    for (j=0;j<c2;j++)
                        System.out.print (c[i][j] + "\t");
                }
            }
        } catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

Explanation:

- The program imports `java.io.*` to use `DataInputStream` for reading user input.
- Defines a class named `matmul`.
- Declares variables `r1`, `r2`, `c1`, `c2` for dimensions of matrices A and B, and loop control variables `i`, `j`, `k`.
- Uses `DataInputStream` to read user input for dimensions (`r1`, `c1` for matrix A and `r2`, `c2` for matrix B).
- Converts the input strings to integers using `Integer.parseInt()`.
- Checks if the number of columns of matrix A (`c1`) is equal to the number of rows of matrix B (`r2`). If not, multiplication is not possible.
- Initializes matrices `a`, `b`, and `c` based on user-provided dimensions.
- Reads elements for matrices A and B using nested loops and stores them in arrays `a` and `b`.
- Performs matrix multiplication using nested loops:
- Outer loops iterate through rows (`i`) and columns (`j`) of resulting matrix `c`.
- Innermost loop (`k`) computes each element of `c` as a sum of products of corresponding elements from `a` and `b`.
- Prints the resultant matrix `c` after multiplication in a formatted manner.

Example

Suppose the user inputs the following:
- `r1 = 2`, `c1 = 2` for matrix A with values `1 2` and `3 4`.
- `r2 = 2`, `c2 = 2` for matrix B with values `5 6` and `7 8`.

The output would be:

```
Result...
19    22
43    50
```

This indicates the result of multiplying matrices A and B.

Notes:
The program assumes valid integer inputs for matrix dimensions and values.
It demonstrates basic matrix operations and nested loop usage in Java for matrix multiplication.

Program 3
This program aims to count no. of characters, words and lines in a given string.

Program

```
import java.io.*;

public class clw
{
        public static void main(String args[])
        {
                String inp;
                int w=0, l=0;

                try
                {
                        DataInputStream dis = new DataInputStream(System.in);

                        System.out.println("Enter Text...");
                        inp = dis.readLine();

                        int c = inp.length();
                        w = inp.split("\\s+").length;
                        l = inp.split("\n").length;

                        System.out.println("Given Text...");
                        System.out.println("Characters \t" + c);
                        System.out.println("Lines \t" + l);
                        System.out.println("Words \t" + w);
                } catch(Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

Explanation:

- The program imports `java.io.*` to use `DataInputStream` for reading user input.
- Defines a class named `clw`.
- Declares variables `inp` to store user input text, `w` for word count, `l` for line count.
- Uses `DataInputStream` to read a line of text (`inp`) from the user.
- dis.readLine() reads the input until the end of the line or an error occurs.
- Calculates the number of characters (`c`) in the input string `inp` using `inp.length()`.
- Splits the input string `inp` based on whitespace (`\\s+`) to count words (`w`).

- Splits the input string `inp` based on newline character (`\n`) to count lines (`l`).
- Prints the metrics of the input text:
  - Number of characters (`c`).
  - Number of lines (`l`).
  - Number of words (`w`).

Example
If the user inputs the following text:

```
Hello world
This is a sample text input.
```

The program would output:

```
Given Text...
Characters    40
Lines      2
Words              8
```

This indicates that the input text has 40 characters, 2 lines, and 8 words.

Program 4
This program aims to usage of random number generation.

Program

```
import java.util.Random;

public class randno
{
    public static void main(String[] args) {
        int lowerLimit = 1;
        int upperLimit = 100;

        Random random = new Random();

        for (int i = 0; i < 10; i++) {
            int randomNumber = random.nextInt(upperLimit - lowerLimit + 1) +
lowerLimit;

            if (randomNumber <= 25)
                System.out.println(randomNumber + " is in the range [1, 25]");
           else if (randomNumber <= 50)
                System.out.println(randomNumber + " is in the range (25, 50]");
           else if (randomNumber <= 75)
                System.out.println(randomNumber + " is in the range (50, 75]");
           else
                System.out.println(randomNumber + " is in the range (75, 100]");
        }
    }
}
```

Explanation

- The program imports `Random` class from `java.util` package to generate random numbers.
- Defines a public class named `randno`.
- Declares variables `lowerLimit` and `upperLimit` which define the range for generating random numbers (inclusive).
- Creates an instance of `Random` class to generate random numbers.
- Initializes a `for` loop that iterates 10 times to generate 10 random numbers.
- `random.nextInt(upperLimit - lowerLimit + 1) + lowerLimit` generates a random integer within the range `[lowerLimit, upperLimit]`.
- Checks the value of `randomNumber` against predefined ranges:
- Prints a message indicating which range `[1, 25]`, `(25, 50]`, `(50, 75]`, or `(75, 100]` the random number falls into.
- Uses conditional statements (`if`, `else if`, `else`) to determine the appropriate range based on the value of `randomNumber`.

Output

```
8 is in the range [1, 25]

38 is in the range (25, 50]

65 is in the range (50, 75]

93 is in the range (75, 100]

15 is in the range [1, 25]

47 is in the range (25, 50]

72 is in the range (50, 75]

86 is in the range (75, 100]

3 is in the range [1, 25]

29 is in the range (25, 50]
```

Each line corresponds to one randomly generated number and its range classification based on the conditions provided.

Notes:

- The program demonstrates how to generate random integers within a specified range using `Random.nextInt(int bound)`.

- It also showcases conditional statements to categorize and print results based on the value of the generated random number.

- `Random` class is a part of the Java standard library (`java.util`) and provides convenient methods for generating random numbers of different types.

Program 5
This program aims to learn string manipulation.

Program

```
import java.io.*;

public class strmanp
{
   public static void main(String[] args)
        {
                try
                {

                        DataInputStream dis = new DataInputStream(System.in);
                        System.out.print("Enter the first string: ");
                        String str1 = dis.readLine();

                        System.out.print("Enter the second string: ");
                        String str2 = dis.readLine();

                        int length1 = str1.length();
                        int length2 = str2.length();

                        int position = 2;
                        char charAtPosition = str1.charAt(position);

                        String x = str1.concat(str2);

                        System.out.println("Length of " + str1 + ": " + length1);
                        System.out.println("Length of " + str2 + ": " + length2);
                        System.out.println("Character at position " + position + " in \"" +
str1 + "\": " + charAtPosition);
                        System.out.println("Concatenated string: " + x);
                } catch (Exception e)
                {
                        System.out.println(e.getMessage());
                }
        }
}
```

Explanation:

- The program imports `java.io.*` to use `DataInputStream` for reading user input.
- Defines a public class named `strmanp`.
- Initializes a `DataInputStream` named `dis` to read input from the console.
- Prompts the user to enter two strings (`str1` and `str2`) and reads them using `dis.readLine()`.

- Calculates the lengths of `str1` and `str2` using the `length()` method of the `String` class.
- Defines `position` as `2` (indexing starts from 0 in Java).
- Uses `charAt(position)` to extract the character at `position` from `str1` and stores it in `charAtPosition`.
- Concatenates `str1` and `str2` using the `concat()` method of the `String` class and stores the result in `x`.
- Prints out the results:
- Length of `str1` and `str2`.
- Character at the specified `position` in `str1`.
- Concatenated string `x` which combines `str1` and `str2`.

Example

If the user enters:

First string (`str1`): "Hello"

Second string (`str2`): "World"

The program would output:

```
Enter the first string: Hello
Enter the second string: World
Length of Hello: 5
Length of World: 5
Character at position 2 in "Hello": l
Concatenated string: HelloWorld
```

The program demonstrates basic string operations in Java such as length calculation, character extraction, and string concatenation.

Program 6

This program aims to learn about string operations in java.

Program

```
import java.io.*;
public class stropr
{
   public static void main(String[] args)
        {
                try
                {
                        DataInputStream dis = new DataInputStream(System.in);

                        System.out.print("Enter the first string: ");
                        String str1 = dis.readLine();

                        System.out.print("Enter the second string: ");
                        String str2 = dis.readLine();

                        String cString = str1.concat(str2);

                        System.out.print("Enter the substring to search in the first string: 
");
                        String sstr = dis.readLine();
                        boolean sFnd = str1.contains(sstr);

                        System.out.print("Enter the starting index to extract substring: ");
                        int sInd = Integer.parseInt(dis.readLine());
                        System.out.print("Enter the ending index to extract substring: ");
                        int eInd = Integer.parseInt(dis.readLine());
                        String estr = str1.substring(sInd, eInd);

                        System.out.println("Concatenated string: " + cString);
                        System.out.println("Substring '" + sstr + "' found in first string: " + 
                        sFnd);
                        System.out.println("Extracted substring from first string: " + estr);
                } catch(Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

Explanation:

- The program imports `java.io.*` to use `DataInputStream` for reading user input.
- Defines a public class named `stropr`.
- Starts the `main` method where the program execution begins.
- Initializes a `DataInputStream` named `dis` to read input from the console.
- Prompts the user to enter two strings (`str1` and `str2`) and reads them using `dis.readLine()`.
- Concatenates `str1` and `str2` using the `concat()` method of the `String` class and stores the result in `cString`.
- Prompts the user to enter a substring (`sstr`) to search within `str1`.
- Uses the `contains()` method of the `String` class to check if `sstr` exists within `str1` and stores the result in `sFnd` (`true` if found, `false` otherwise).
- Prompts the user to enter starting (`sInd`) and ending (`eInd`) indices to extract a substring (`estr`) from `str1`.
- Uses the `substring(int beginIndex, int endIndex)` method of the `String` class to extract the substring from `sInd` to `eInd` (excluding `eInd`).
- Prints out the results:
- Concatenated string (`cString`).
- Whether the substring `sstr` was found in `str1` (`sFnd`).
- The extracted substring (`estr`) from `str1`.

Example

If the user enters:

- First string (`str1`): "Hello World"

- Second string (`str2`): "!"

- Substring to search (`sstr`): "World"

- Starting index (`sInd`): 6

- Ending index (`eInd`): 11

The program would output:

```
Enter the first string: Hello World

Enter the second string: !

Enter the substring to search in the first string: World

Enter the starting index to extract substring: 6

Enter the ending index to extract substring: 11

Concatenated string: Hello World!

Substring 'World' found in first string: true
```

Program 7

Java program that demonstrates string operations using the `StringBuffer` class:

Program

```
import java.util.Scanner;

public class StringBufferOperations {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a string: ");
        String inputString = scanner.nextLine();

        StringBuffer stringBuffer = new StringBuffer(inputString);

        int lengthOfString = stringBuffer.length();
        System.out.println("Length of the string: " + lengthOfString);

        stringBuffer.reverse();
        String reversedString = stringBuffer.toString();
        System.out.println("Reversed string: " + reversedString);

        System.out.print("Enter the substring to delete: ");
        String substringToDelete = scanner.nextLine();

        int index = stringBuffer.indexOf(substringToDelete);
        if (index != -1) {
            stringBuffer.delete(index, index + substringToDelete.length());
            System.out.println("String after deleting '" + substringToDelete + "': " +
stringBuffer);
        } else {
            System.out.println("Substring '" + substringToDelete + "' not found in the
string.");
        }

        scanner.close();
    }
}
```

Explanation:

1. **Import Statements**: Import necessary classes (`Scanner`) for user input.

2. **Main Method**: Entry point of the program.

3. **User Input**:
   - `Scanner` object (`scanner`) is used to read user input.
   - Prompt the user to enter a string (`inputString`).

4. **StringBuffer Initialization**:
   - Create a `StringBuffer` object (`stringBuffer`) initialized with `inputString`.

5. **a. Length of a String**:
   - Use `stringBuffer.length()` to get the length of the string.

6. **b. Reverse a String**:
   - Use `stringBuffer.reverse()` to reverse the contents of `stringBuffer`.
   - Convert the reversed `StringBuffer` back to a `String` using `stringBuffer.toString()`.

7. **c. Delete a Substring**:
   - Prompt the user to enter a substring (`substringToDelete`).
   - Use `stringBuffer.indexOf(substringToDelete)` to find the index of the substring in `stringBuffer`.
   - If found (`index != -1`), use `stringBuffer.delete(index, index + substringToDelete.length())` to delete the substring.
   - Print the modified string (`stringBuffer`).

8. **Error Handling**:
   - Handle cases where the substring to delete is not found (`index == -1`).

9. **Closing Scanner**:
   - Close the `Scanner` object to release resources.

Example Usage:

```
Enter a string: Hello, world!
Length of the string: 13
Reversed string: !dlrow ,olleH
Enter the substring to delete: world
String after deleting 'world': Hello, !
```

This program showcases basic string operations using the `StringBuffer` class in Java. The `StringBuffer` class is used here due to its mutability, which allows efficient modification of strings, such as appending, deleting, or reversing characters.

Program 8

Java program aims to implement a multi-threaded application with three threads:

Program

```
import java.util.Random;

public class MultiThreadExample {
    public static void main(String[] args) {
        NumberGenerator numberGenerator = new NumberGenerator();
        EvenProcessor evenProcessor = new EvenProcessor(numberGenerator);
        OddProcessor oddProcessor = new OddProcessor(numberGenerator);

        Thread thread1 = new Thread(numberGenerator, "NumberGenerator");
        Thread thread2 = new Thread(evenProcessor, "EvenProcessor");
        Thread thread3 = new Thread(oddProcessor, "OddProcessor");

        thread1.start();
        thread2.start();
        thread3.start();
    }
}

class NumberGenerator implements Runnable {
    private Random random = new Random();

    public void run() {
        while (true) {
            int randomNumber = random.nextInt(100);
            System.out.println("Generated number: " + randomNumber);

            if (randomNumber % 2 == 0) {
                EvenProcessor.process(randomNumber);
            } else {
                OddProcessor.process(randomNumber);
            }

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class EvenProcessor implements Runnable {
    private static NumberGenerator numberGenerator;
    public EvenProcessor(NumberGenerator numberGenerator) {
        this.numberGenerator = numberGenerator;
    }

    public static void process(int number) {
        System.out.println("EvenProcessor: Square of " + number + " is " + (number *
number));
    }

       public void run() {
       }
}

class OddProcessor implements Runnable {
    private static NumberGenerator numberGenerator;

    public OddProcessor(NumberGenerator numberGenerator) {
        this.numberGenerator = numberGenerator;
    }

    public static void process(int number) {
        System.out.println("OddProcessor: Cube of " + number + " is " + (number *
number * number));
    }

       public void run() {
       }
}
```

Explanation:

1. **NumberGenerator (Runnable)**:
   - This class generates random integers every second (`Thread.sleep(1000);`).
   - Depending on whether the number is even or odd, it calls the static `process` methods of `EvenProcessor` or `OddProcessor`.

2. **EvenProcessor (Runnable)** and **OddProcessor (Runnable)**:
   - These classes implement `Runnable` and have a static `process` method that computes either the square (for even numbers) or the cube (for odd numbers) of the provided integer and prints the result.
   - They are initialized with an instance of `NumberGenerator` to access the `process` method.

3. **Main Method (`MultiThreadExample`)**:
   - Creates instances of `NumberGenerator`, `EvenProcessor`, and `OddProcessor`.
   - Creates three separate threads for each of these objects and starts them concurrently.

How It Works:

- `NumberGenerator` continuously generates random integers.
- If the integer is even, it calls `EvenProcessor.process(randomNumber)`, which computes and prints the square of the number.
- If the integer is odd, it calls `OddProcessor.process(randomNumber)`, which computes and prints the cube of the number.
- `EvenProcessor` and `OddProcessor` wait for their respective numbers to be processed by the `NumberGenerator`.

This program demonstrates basic multithreading in Java using `Runnable` interfaces and static methods for processing numbers in separate threads.

Program 9

Java program that uses two threads (`Thread1` and `Thread2`) to asynchronously print numbers 1 to 10 and 90 to 100 respectively:

Program

```
public class ThreadExample {
    public static void main(String[] args) {
        NumberPrinter numberPrinter = new NumberPrinter();

        Thread thread1 = new Thread(() -> {
            numberPrinter.printNumbers(1, 10);
        });

        Thread thread2 = new Thread(() -> {
            numberPrinter.printNumbers(90, 100);
        });

        thread1.start();
        thread2.start();
    }
}

class NumberPrinter {
    public void printNumbers(int start, int end) {
        for (int i = start; i <= end; i++) {
            System.out.println(Thread.currentThread().getName() + ": " + i);
            try {
                Thread.sleep(500); // Sleep for 500 milliseconds between prints
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Explanation:

1. **ThreadExample**:
   - This class contains the `main` method where two threads (`thread1` and `thread2`) are created and started.
   - Each thread uses a lambda expression to define its task, which is to invoke the `printNumbers` method of `NumberPrinter`.

2. **NumberPrinter**:
   - This class has a method `printNumbers` that takes two integers (`start` and `end`).
   - It iterates from `start` to `end` and prints each number along with the name of the current thread (`Thread.currentThread().getName()`).
   - After printing each number, it sleeps for 500 milliseconds (`Thread.sleep(500)`) to simulate some processing time.

How It Works:
- `Thread1` is created to print numbers from 1 to 10 using `numberPrinter.printNumbers(1, 10);`.
- `Thread2` is created to print numbers from 90 to 100 using `numberPrinter.printNumbers(90, 100);`.
- Both threads run concurrently, and each invokes the `printNumbers` method with their respective range of numbers.
- The `Thread.sleep(500)` in the `printNumbers` method simulates a delay between printing each number.

Output Example:
The output might not be perfectly sequential due to the asynchronous nature of threads, but it will generally print numbers from 1 to 10 and 90 to 100 concurrently.

```
Thread-0: 1
Thread-1: 90
Thread-0: 2
Thread-1: 91
Thread-0: 3
Thread-1: 92
Thread-0: 4
Thread-1: 93
Thread-0: 5
Thread-1: 94
Thread-0: 6
Thread-1: 95
Thread-0: 7
Thread-1: 96
Thread-0: 8
Thread-1: 97
Thread-0: 9
Thread-1: 98
Thread-0: 10
Thread-1: 99
Thread-1: 100
```

This program demonstrates the use of multithreading in Java to execute tasks concurrently and asynchronously.

Program 10
Java program that demonstrates each of the specified exceptions:

Program

```
public class ExceptionDemo {
    public static void main(String[] args) {
        try {
            int result = 5 / 0;
            System.out.println("Result of division: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception caught: Division by zero");
        }

        try {
            String str = "abc";
            int num = Integer.parseInt(str);
            System.out.println("Parsed number: " + num);
        } catch (NumberFormatException e) {
            System.out.println("Number Format Exception caught: Invalid number
format");
        }

        try {
            int[] arr = {1, 2, 3};
            System.out.println("Accessing element at index 3: " + arr[3]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("ArrayIndexOutOfBoundsException caught: Index out of
bounds");
        }

        try {
            int[] negativeArray = new int[-3];
            System.out.println("Array created successfully with size: " +
negativeArray.length);
        } catch (NegativeArraySizeException e) {
            System.out.println("NegativeArraySizeException caught: Negative array
size");
        }
    }
}
```

Explanation and Output:

1. **Arithmetic Exception**:
   - This occurs when you try to divide by zero (`5 / 0`).
   - Output: `Arithmetic Exception caught: Division by zero`

2. **Number Format Exception**:
   - This occurs when you try to parse a non-numeric string (`Integer.parseInt("abc")`).
   - Output: `Number Format Exception caught: Invalid number format`

3. **ArrayIndexOutOfBoundsException**:
   - This occurs when you try to access an index that is outside the bounds of an array (`arr[3]` where `arr` has length 3).
   - Output: `ArrayIndexOutOfBoundsException caught: Index out of bounds`

4. **NegativeArraySizeException**:
   - This occurs when you try to create an array with a negative size (`new int[-3]`).
   - Output: `NegativeArraySizeException caught: Negative array size`

Each exception is caught using a `try-catch` block specific to the type of exception, and an appropriate error message is printed to indicate the nature of the exception caught.

Note:
- Exception handling in Java allows you to gracefully handle errors that may occur during program execution, preventing abrupt termination and providing feedback or alternative paths as needed.
- It's important to handle exceptions properly in production code to ensure robustness and reliability of your applications.

Program 11

Java program that reads a file name from the user and displays various information about the file:

Program

```
import java.io.File;
import java.util.Scanner;

public class FileInfo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the file name: ");
        String fileName = scanner.nextLine();

        File file = new File(fileName);

        if (file.exists()) {
            System.out.println("File exists: Yes");
            System.out.println("File name: " + file.getName());
            System.out.println("Absolute path: " + file.getAbsolutePath());
            System.out.println("Readable: " + file.canRead());
            System.out.println("Writable: " + file.canWrite());
            System.out.println("File type: " + getFileType(file));
            System.out.println("File length in bytes: " + file.length());
        } else {
            System.out.println("File exists: No");
        }

        scanner.close();
    }

    private static String getFileType(File file) {
        if (file.isDirectory()) {
            return "Directory";
        } else if (file.isFile()) {
            return "File";
        } else {
            return "Unknown";
        }
    }
}
```

Explanation:

1. **Import Statements**:
   - `import java.io.File;`: Imports the `File` class which represents a file or directory pathname in the file system.
   - `import java.util.Scanner;`: Imports `Scanner` class to read input from the user.

2. **Main Method** (`public static void main(String[] args)`) :
   - Creates a `Scanner` object (`scanner`) to read user input from the console.
   - Prompts the user to enter a file name using `System.out.print("Enter the file name: ");` and reads it with `scanner.nextLine();`.
   - Creates a `File` object (`file`) using the entered file name.

3. **File Information**:
   - Checks if the file exists using `file.exists()` and prints "File exists: Yes" if true; otherwise, "File exists: No".
   - If the file exists, prints:
     - File name (`file.getName()`)
     - Absolute path (`file.getAbsolutePath()`)
     - Whether the file is readable (`file.canRead()`)
     - Whether the file is writable (`file.canWrite()`)
     - Type of file (uses `getFileType(file)` method)
     - Length of the file in bytes (`file.length()`)

4. **`getFileType` Method**:
   - This method determines the type of the file based on whether it is a directory (`file.isDirectory()`), a regular file (`file.isFile()`), or unknown if neither.

5. **Closing Scanner**:
   - Closes the `Scanner` object (`scanner`) to free up resources.

Example Output:

If you run this program and enter a valid file name that exists on your system, the output might look like this:

```
Enter the file name: test.txt
File exists: Yes
File name: test.txt
Absolute path: /Users/username/Documents/test.txt
Readable: true
Writable: true
File type: File
File length in bytes: 1024
```

This program provides basic file handling functionalities and demonstrates how to use `File` class methods to retrieve file information in Java.

Program 12

Java Swing program that accepts text input and allows changing its size and font style (bold, italic), we'll use `JFrame`, `JTextArea`, `JComboBox`, and `JCheckBox` components.

Program

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TextEditor extends JFrame implements ActionListener {
    private JTextArea textArea;
    private JComboBox<String> fontSizeCombo;
    private JCheckBox boldCheckBox;
    private JCheckBox italicCheckBox;

    public TextEditor() {
        setTitle("Text Editor");
        setSize(600, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        textArea = new JTextArea();
        textArea.setFont(new Font("Arial", Font.PLAIN, 14));
        JScrollPane scrollPane = new JScrollPane(textArea);
        add(scrollPane, BorderLayout.CENTER);

        JPanel controlPanel = new JPanel();
        controlPanel.setLayout(new FlowLayout());

        fontSizeCombo = new JComboBox<>();
        fontSizeCombo.addItem("12");
        fontSizeCombo.addItem("14");
        fontSizeCombo.addItem("16");
        fontSizeCombo.addItem("18");
        fontSizeCombo.addItem("20");
        fontSizeCombo.setSelectedIndex(1);
        fontSizeCombo.addActionListener(this);
        controlPanel.add(new JLabel("Font Size:"));
        controlPanel.add(fontSizeCombo);

        boldCheckBox = new JCheckBox("Bold");
        boldCheckBox.addActionListener(this);
        controlPanel.add(boldCheckBox);

        italicCheckBox = new JCheckBox("Italic");
        italicCheckBox.addActionListener(this);
        controlPanel.add(italicCheckBox);

        add(controlPanel, BorderLayout.NORTH);

        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == fontSizeCombo) {
            int fontSize = Integer.parseInt((String) fontSizeCombo.getSelectedItem());
            Font currentFont = textArea.getFont();
            Font newFont = new Font(currentFont.getFontName(), currentFont.getStyle(), fontSize);
            textArea.setFont(newFont);
        }

        int fontStyle = Font.PLAIN;
        if (boldCheckBox.isSelected()) {
            fontStyle = fontStyle | Font.BOLD;
        }
        if (italicCheckBox.isSelected()) {
            fontStyle = fontStyle | Font.ITALIC;
        }
        Font currentFont = textArea.getFont();
        Font newFont = new Font(currentFont.getFontName(), fontStyle, currentFont.getSize());
        textArea.setFont(newFont);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new TextEditor());
    }
}
```

Explanation:

1. **JFrame and Components**:
   - `TextEditor` class extends `JFrame` and implements `ActionListener` for handling events.
   - Components include a `JTextArea` (`textArea`) for displaying and editing text, a `JComboBox` (`fontSizeCombo`) for selecting font size, and two `JCheckBox` (`boldCheckBox` and `italicCheckBox`) for selecting bold and italic styles.

2. **Initialization in Constructor**:
   - Sets up the main `JFrame` with a title, size, and default close operation.
   - Uses `BorderLayout` for arranging components: `JTextArea` in the center and controls (`JPanel` with `FlowLayout`) at the top.
   - Initializes the `JTextArea` with a default font (`Arial`, plain style, size 14).
   - Adds a `JScrollPane` around `textArea` to enable scrolling.

3. **Controls and Event Handling**:
- `fontSizeCombo` allows selecting from predefined font sizes. Changes in the combo box update the font size of `textArea`.
- `boldCheckBox` and `italicCheckBox` toggle the bold and italic styles of `textArea`. The `actionPerformed` method handles these checkbox events by modifying the font style accordingly and updating `textArea`.

4. **Main Method**:
- Invokes `SwingUtilities.invokeLater()` to ensure the GUI components are created and modified on the Event Dispatch Thread (EDT), which is recommended for Swing applications.

Usage:

- Compile and run the `TextEditor` class.
- Enter text in the `JTextArea`.
- Use the controls (`Font Size` combo box, `Bold`, `Italic` checkboxes) to change the appearance of the text dynamically.

This program demonstrates basic Swing GUI components and event handling in Java, allowing users to interactively adjust text size and style.

Program 13

Mouse events

In Java, handling mouse events can be achieved using adapter classes provided by the `java.awt.event` package. These adapter classes allow you to override only the methods you're interested in, making event handling more convenient. Here's a Java program that demonstrates handling all mouse events and displays the event name at the center of the window:

Program

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class MouseEventDemo extends JFrame {
    private JLabel eventLabel;

    public MouseEventDemo() {
        setTitle("Mouse Event Demo");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        eventLabel = new JLabel("Move mouse over the window", JLabel.CENTER);
        eventLabel.setFont(new Font("Arial", Font.BOLD, 20));
        add(eventLabel, BorderLayout.CENTER);

        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                displayEventName("Mouse Clicked");
            }

            public void mousePressed(MouseEvent e) {
                displayEventName("Mouse Pressed");
            }

            public void mouseReleased(MouseEvent e) {
                displayEventName("Mouse Released");
            }

            public void mouseEntered(MouseEvent e) {
                displayEventName("Mouse Entered");
            }

            public void mouseExited(MouseEvent e) {
                displayEventName("Mouse Exited");
            }
        });

        setVisible(true);
    }
    private void displayEventName(String eventName) {
        eventLabel.setText(eventName);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new MouseEventDemo());
    }
}
```

Explanation:

1. **JFrame Initialization**:
   - `MouseEventDemo` class extends `JFrame` and sets up the main frame for the application.
   - Sets the title, size, and default close operation for the frame.

2. **Event Label (`JLabel`)**:
   - Creates a `JLabel` (`eventLabel`) to display the name of the mouse event.
   - Configures the label to be centered (`JLabel.CENTER`) and uses a bold font (`Arial`, bold, size 20).
   - Adds the label to the center of the frame (`BorderLayout.CENTER`).

3. **Mouse Event Handling**:
   - Uses adapter classes (`MouseAdapter`) to handle mouse events without implementing all methods of `MouseListener`.
   - Overrides methods (`mouseClicked`, `mousePressed`, `mouseReleased`, `mouseEntered`, `mouseExited`) to update the `eventLabel` with the corresponding event name when each event occurs.

4. **Display Event Name**:
   - `displayEventName` method updates the text of `eventLabel` to show the current mouse event.

5. **Main Method**:
   - Invokes `SwingUtilities.invokeLater()` to ensure the GUI components are created and modified on the Event Dispatch Thread (EDT), which is recommended for Swing applications.
   - Creates an instance of `MouseEventDemo`, making the GUI visible and ready for user interaction.

Usage:

- Compile and run the `MouseEventDemo` class.
- Move the mouse over the window and observe the event name (`Mouse Entered`).
- Perform various mouse actions (click, press, release) and see the corresponding event names updated dynamically at the center of the window.

This program demonstrates handling mouse events (`MouseClicked`, `MousePressed`, `MouseReleased`, `MouseEntered`, `MouseExited`) using adapter classes in Java Swing, providing interactive feedback on mouse actions within the application window.

Program 14

Java program that implements a simple calculator using `GridLayout` to arrange buttons for digits and operations (`+`, `-`, `*`, `%`). It includes a `JTextField` to display the result and handles exceptions such as divide by zero:

Program

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Calculator extends JFrame implements ActionListener {
   private JTextField displayField;

   public Calculator() {
      setTitle("Simple Calculator");
      setSize(300, 400);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      displayField = new JTextField();
      displayField.setEditable(false); // Make it read-only
      displayField.setHorizontalAlignment(JTextField.RIGHT);
      add(displayField, BorderLayout.NORTH);

      JPanel buttonPanel = new JPanel(new GridLayout(4, 4, 5, 5));

      addButton(buttonPanel, "7");
      addButton(buttonPanel, "8");
      addButton(buttonPanel, "9");
      addButton(buttonPanel, "/");

      addButton(buttonPanel, "4");
      addButton(buttonPanel, "5");
      addButton(buttonPanel, "6");
      addButton(buttonPanel, "*");

      addButton(buttonPanel, "1");
      addButton(buttonPanel, "2");
      addButton(buttonPanel, "3");
      addButton(buttonPanel, "-");

      addButton(buttonPanel, "0");
      addButton(buttonPanel, ".");
      addButton(buttonPanel, "=");
      addButton(buttonPanel, "+");
```

```
        add(buttonPanel, BorderLayout.CENTER);

        setVisible(true);
    }

    private void addButton(Container container, String text) {
        JButton button = new JButton(text);
        button.addActionListener(this);
        container.add(button);
    }

    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

        if ("0123456789.".contains(command)) {
            displayField.setText(displayField.getText() + command);
        } else if (command.equals("+") || command.equals("-") || command.equals("*") ||
command.equals("/")) {
            displayField.setText(displayField.getText() + " " + command + " ");
        } else if (command.equals("=")) {
            evaluateExpression();
        }
    }

    private void evaluateExpression() {
        String expression = displayField.getText();
        String[] parts = expression.split(" ");

        if (parts.length != 3) {
            displayField.setText("Invalid expression");
            return;
        }

        try {
            double operand1 = Double.parseDouble(parts[0]);
            String operator = parts[1];
            double operand2 = Double.parseDouble(parts[2]);
            double result = 0;

            switch (operator) {
                case "+":
                    result = operand1 + operand2;
                    break;
                case "-":
                    result = operand1 - operand2;
                    break;
                case "*":
                    result = operand1 * operand2;
                    break;
                case "/":
                    if (operand2 == 0) {
                        displayField.setText("Error: Divide by zero");
                        return;
                    }
                    result = operand1 / operand2;
                    break;
                default:
                    displayField.setText("Invalid operator");
                    return;
            }

            displayField.setText(Double.toString(result));
        } catch (NumberFormatException ex) {
            displayField.setText("Invalid expression");
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new Calculator());
    }
}
```

Explanation:

1. **JFrame and Components**:
   - `Calculator` class extends `JFrame` and sets up the main frame for the calculator application.
   - Sets the title, size, and default close operation for the frame.

2. **Display Field (`JTextField`)**:
   - `displayField` is a `JTextField` used to display the input and output of the calculator.
   - Configured to be read-only (`setEditable(false)`) and right-aligned (`setHorizontalAlignment(JTextField.RIGHT)`).
   - Added to the top (`BorderLayout.NORTH`) of the frame.

3. **Button Panel (`JPanel` with `GridLayout`)**:
   - `buttonPanel` is a `JPanel` configured with `GridLayout` (4 rows, 4 columns) to arrange buttons for digits (0-9), decimal point (`.`), and operators (`+`, `-`, `*`, `/`).
   - Buttons are added using `addButton` method which creates `JButton` instances and attaches `ActionListener` (`this`).

4. **Button Handling (`ActionListener` Implementation)**:
   - `actionPerformed` method handles button clicks:
     - Numeric buttons (`0-9`, `.`) append their respective text to `displayField`.
     - Operator buttons (`+`, `-`, `*`, `/`) append a space before and after the operator to `displayField`.
     - Equal button (`=`) triggers `evaluateExpression` method to evaluate and display the result.

5. **Expression Evaluation (`evaluateExpression`)**:
   - `evaluateExpression` method parses the expression from `displayField`, splits it into parts (`operand1 operator operand2`), and evaluates using a `switch` statement.
   - Handles exceptions such as divide by zero (`ArithmeticException`) and invalid expressions (`NumberFormatException`).

6. **Main Method**:
   - Invokes `SwingUtilities.invokeLater()` to ensure the GUI components are created and modified on the Event Dispatch Thread (EDT), which is recommended for Swing applications.
   - Creates an instance of `Calculator`, making the GUI visible and ready for user interaction.

Usage:

- Compile and run the `Calculator` class.
- Use the numeric buttons (`0-9`, `.`), operator buttons (`+`, `-`, `*`, `/`), and equal button (`=`) to perform calculations.
- The result of the calculation or any errors (such as divide by zero) will be displayed in the `JTextField` (`displayField`).

This program demonstrates a basic calculator GUI in Java Swing with support for handling various arithmetic operations and input validation.

Program 15

Java program that simulates a traffic light using radio buttons to select between red, yellow, and green lights. When a radio button is selected, an appropriate message ("Stop", "Ready", "Go") is displayed above the radio buttons in the corresponding color. Initially, no message is shown.

Program

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TrafficLightSimulator extends JFrame implements ActionListener {
   private JLabel messageLabel;
   private JRadioButton redButton, yellowButton, greenButton;

   public TrafficLightSimulator() {
      setTitle("Traffic Light Simulator");
      setSize(300, 200);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setLayout(new BorderLayout());

      JPanel radioPanel = new JPanel(new FlowLayout());

      redButton = new JRadioButton("Red");
      yellowButton = new JRadioButton("Yellow");
      greenButton = new JRadioButton("Green");

      ButtonGroup buttonGroup = new ButtonGroup();
      buttonGroup.add(redButton);
      buttonGroup.add(yellowButton);
      buttonGroup.add(greenButton);

      redButton.addActionListener(this);
      yellowButton.addActionListener(this);
      greenButton.addActionListener(this);

      radioPanel.add(redButton);
      radioPanel.add(yellowButton);
      radioPanel.add(greenButton);

      JPanel messagePanel = new JPanel();
      messagePanel.setLayout(new FlowLayout());
      messageLabel = new JLabel("No message", JLabel.CENTER);
      messageLabel.setFont(new Font("Arial", Font.BOLD, 20));
      messagePanel.add(messageLabel);

      add(radioPanel, BorderLayout.CENTER);
      add(messagePanel, BorderLayout.NORTH);
```

```
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == redButton) {
            messageLabel.setText("Stop");
            messageLabel.setForeground(Color.RED);
        } else if (e.getSource() == yellowButton) {
            messageLabel.setText("Ready");
            messageLabel.setForeground(Color.YELLOW.darker()); // Darker yellow
        } else if (e.getSource() == greenButton) {
            messageLabel.setText("Go");
            messageLabel.setForeground(Color.GREEN.darker()); // Darker green
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new TrafficLightSimulator());
    }
}
```

Explanation:

1. **JFrame and Components**:
   - `TrafficLightSimulator` class extends `JFrame` and sets up the main frame for the traffic light simulator.
   - Sets the title, size, and default close operation for the frame.

2. **Radio Buttons (`JRadioButton`)**:
   - `redButton`, `yellowButton`, and `greenButton` are radio buttons representing the traffic light colors "Red", "Yellow", and "Green".
   - Added to a `ButtonGroup` (`buttonGroup`) to ensure only one button can be selected at a time.
   - Each button is configured with an `ActionListener` (`this`) to handle selection events.

3. **Action Handling (`actionPerformed`)**:
   - `actionPerformed` method handles events when a radio button is selected:
     - Updates `messageLabel` with appropriate text ("Stop", "Ready", "Go") based on the selected button.
     - Sets the foreground color of `messageLabel` to correspond with the traffic light color using `Color` constants (`Color.RED`, `Color.YELLOW`, `Color.GREEN`), with darker shades (`Color.darker()`) for better visibility.

4. **Message Label (`JLabel`)**:
   - `messageLabel` is a `JLabel` initially displaying "No message" above the radio buttons.
   - Configured to use a bold font (`Arial`, bold, size 20) and centered alignment (`JLabel.CENTER`).

5. **Main Method**:
- Invokes `SwingUtilities.invokeLater()` to ensure the GUI components are created and modified on the Event Dispatch Thread (EDT), which is recommended for Swing applications.
- Creates an instance of `TrafficLightSimulator`, making the GUI visible and ready for user interaction.

Usage:

- Compile and run the `TrafficLightSimulator` class.
- Select one of the radio buttons (`Red`, `Yellow`, `Green`) to simulate the traffic light changing.
- Observe the message ("Stop", "Ready", "Go") displayed in the corresponding color above the radio buttons.

This program demonstrates a basic GUI application in Java Swing that simulates a traffic light using radio buttons and displays messages based on the selected light color.