PRACTICAL No. 1: Implement a Program to define a structure of basic JAVA Program.

THEORY-

Structure of Java Program

Java is an object-oriented programming, platform-independent, and secure programming language that makes it popular. Using the Java programming language, we can develop a wide variety of applications.

A typical structure of a Java program contains the following elements:

- Documentation Section
- Package Declaration
- Import Statements
- Interface Section
- Class Definition
- Class Variables and Variables
- Main Method Class
- Methods and Behaviors

Documentation Section

The documentation section is an important section but optional for a Java program. It includes **basic information** about a Java program. The information includes the **author's name, date of creation, version, program name, company name,** and **description** of the program. To write the statements in the documentation section, we use **comments**. The comments may be **single-line, multi-line,** and **documentation** comments.

- **Single-line Comment:** It starts with a pair of forwarding slash **(//)**. For example:

```
//First Java Program
```

- **Multi-line Comment:** It starts with a **/*** and ends with ***/.** We write between these two symbols. For example:

```
/*It is an example of
multiline comment*/
```

- **Documentation Comment:** It starts with the delimiter **(/**)** and ends with ***/**. For example:

```
/**It is an example of documentation comment*/
```

Package Declaration

The package declaration is optional. It is placed just after the documentation section. In this section, we declare the **package name** in which the class is placed. Note that there can be **only**

one package statement in a Java program. It must be defined before any class and interface declaration. It is necessary because a Java class can be placed in different packages and directories based on the module they are used. For all these classes package belongs to a single parent directory. We use the keyword **package** to declare the package name. For example:

```
package javatpoint; //where javatpoint is the package name
package com.javatpoint; //where com is the root directory and javatpoint is the subdirectory
```

Import Statements

The package contains the many predefined classes and interfaces. If we want to use any class of a particular package, we need to import that class. The import statement represents the class stored in the other package. We use the **import** keyword to import the class. It is written before the class declaration and after the package statement. We use the import statement in two ways, either import a specific class or import all classes of a particular package. In a Java program, we can use multiple import statements. For example:

```
import java.util.Scanner; //it imports the Scanner class only
import java.util.*; //it imports all the class of the java.util package
```

Interface Section

It is an optional section. We can create an **interface** in this section if required. We use the **interface** keyword to create an interface. An interface is a slightly different from the class. It contains only **constants** and **method** declarations. Another difference is that it cannot be instantiated. We can use interface in classes by using the **implements** keyword. An interface can also be used with other interfaces by using the **extends** keyword. For example:

```
interface car
{
void start();
void stop();
}
```

Class Definition

In this section, we define the class. It is **vital** part of a Java program. Without the class, we cannot create any Java program. A Java program may conation more than one class definition. We use the **class** keyword to define the class. The class is a blueprint of a Java program. It contains information about user-defined methods, variables, and constants. Every Java program has at least one class that contains the main() method. For example:

```
class Student //class definition
{
}
```

Class Variables and Constants

In this section, we define variables and **constants** that are to be used later in the program. In a Java program, the variables and constants are defined just after the class definition. The variables and constants store values of the parameters. It is used during the execution of the program. We can also decide and define the scope of variables by using the modifiers. It defines the life of the variables. For example:

```
class Student //class definition
{
String sname;  //variable
int id;
double percentage;
}
```

Main Method Class

In this section, we define the **main() method.** It is essential for all Java programs. Because the execution of all Java programs starts from the main() method. In other words, it is an entry point of the class. It must be inside the class. Inside the main method, we create objects and call the methods. We use the following statement to define the main() method:

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

Methods and behavior

In this section, we define the functionality of the program by using the methods. The methods are the set of instructions that we want to perform. These instructions execute at runtime and perform the specified task. For example:

```
public class Demo //class definition
{
public static void main(String args[])
{
void display()
{
System.out.println("Welcome to javatpoint");
}
//statements
}
}
```

PROGRAM:

```
import java.util.Scanner;

/* This is a programm to add two integer numbers.
   This program is created to understand basic struture of Java program.
   The structure is as follows:
   1. Documentation Section
   2. Package statement
   3. Import statement
   4. Interface statement
   5. Class Definition
   6. Main Method class{
         main method definition
     }
*/

class Adder{
        int add(int a, int b){
                return (a+b);
        }
}

public class TestAdder {
        public static void main(String args[]){
                Scanner scan = new Scanner(System.in);
                System.out.println("Enter the first number: ");
                int fn = scan.nextInt();
                System.out.println("Enter the second number: ");
                int sn = scan.nextInt();

                Adder ad = new Adder();
                int sum = ad.add(fn, sn);
                System.out.println("Sum of two numbers is " + sum);
        }
}
```

Output

```
Enter the first number: 5

Enter the second number: 7

Sum of two numbers is 12
```

Result: Thus the basic structure of JAVA program was successfully executed.

Viva Voice Question:

1. What is Java?

2. What is object-oriented programming?

3. What are the different types of variables in Java?

4. What are access modifiers?

PRACTICAL No. 2: Implement a Program to define the Data types, Variable, Operators, arrays, and Control Structure.

THEORY-

Variables in Java

A variable in Java is a basic storage unit in programming. It's a memory location where you can store data values.

Declaring Variables

Syntax: dataType variableName = value;
Example: int age = 30;

Naming Conventions: Start with a lowercase letter and use camelCase for multi-word names, like totalAmount.

Types of Variables

- Local Variables: Declared inside methods or blocks and accessible only within.
- Instance Variables: Declared inside a class but outside any method, and accessible by any method in the class.
- Class/Static Variables: Declared with the static keyword, they are shared among all instances of the class.

Exploring Data Types

Data types in Java specify the size and type of values stored in variables. There are two main categories:

Primitive Data Types

- Numeric Types: byte, short, int, long, float, double.
- Character Type: char.
- Boolean Type: boolean.

Non-Primitive Data Types

- Classes, Interfaces, Arrays

Choosing the Right Data Type

- Use int for whole numbers, double for fractional numbers.
- char for single characters, and boolean for true/false values.

Operators in Java

Operators in Java are symbols that perform operations on variables and values.

Arithmetic Operators

• Used for mathematical operations like addition (+), subtraction (-), multiplication (*), division (/), and modulo (%).

Relational Operators

• Compare two values and determine their relationship, like equals (==), not equals (!=), greater than (>), less than (<).

Logical Operators

• Used to perform logical "AND" (&&), "OR" (||), and "NOT" (!) operations.

Assignment Operators

• Used to assign values to variables, like = (simple assignment), += (add and assign), -= (subtract and assign).

Arrays in Java

Arrays are fundamental structures in Java that allow us to store multiple values of the same type in a single variable. They are useful for storing and managing collections of data. Arrays in Java are objects, which makes them work differently from arrays in C/C++ in terms of memory management.

For **primitive arrays**, **elements** are stored in a contiguous memory location. For **non-primitive arrays**, **references** are stored at contiguous locations, but the actual objects may be at different locations in memory.

Basics of Arrays in Java
There are some basic operations we can start with as mentioned below:

1. Array Declaration
syntax:

```
type[] arrayName;
```

2. Create an Array
To create an array, you need to allocate memory for it using the new keyword:

```
// Creating an array of 5 integers
int[] numbers = new int[5];
```

This statement initializes the numbers array to hold 5 integers. The default value for each element is 0.

3. Access an Element of an Array
We can access array elements using their index, which starts from 0:

```
// Setting the first element of the array
numbers[0] = 10;

// Accessing the first element
int firstElement = numbers[0];
```

The first line sets the value of the first element to 10. The second line retrieves the value of the first element.

4. Change an Array Element
To change an element, assign a new value to a specific index:

// Changing the first element to 20
numbers[0] = 20;

5. Array Length

We can get the length of an array using the length property:

// Getting the length of the array
int length = numbers.length;

Java Control Statements

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

1. Decision Making statements
    1. if statements
    2. switch statement
2. Loop statements
    1. do while loop
    2. while loop
    3. for loop
    4. for-each loop
3. Jump statements
    1. break statement
    2. continue statement

Decision-Making statements:

As the name suggests, decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in Java, i.e., If statement and switch statement.

1) If Statement:

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

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

Let's understand the if-statements one by one.

1) Simple if statement:

Syntax:

```
if(condition) {
statement 1; //executes when condition is true
}
```

2) if-else statement

Syntax:

```
if(condition) {
statement 1; //executes when condition is true
}
else{
statement 2; //executes when condition is false
}
```

3) if-else-if ladder:

Syntax:

```
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false
}
```

4. Nested if-statement

Syntax:

```
if(condition 1) {
statement 1; //executes when condition 1 is true
if(condition 2) {
statement 2; //executes when condition 2 is true
}
else{
statement 2; //executes when condition 2 is false
}
}
```

Switch Statement:

In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

Syntax:

```
switch (expression){
   case value1:
```

```
 statement1;
 **break**;
.
.
.
case valueN:
statementN;
break;
default:
 default statement;
}
```

Loop Statements

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition. In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

1. for loop
2. while loop
3. do-while loop

Java for loop

The syntax of the while loop is given below.

```
for(initialization, condition, increment/decrement) {
//block of statements
}
```

Java while loop.

Syntax

```
while(condition){
//looping statements
}
```

Java do-while loop

Syntax:

```
do
{
//statements
} while (condition);
```

PROGRAM:

a. Program to implements Data types, variables and operators.

```
public class OperatorsEx {

    public static void main(String[] args) {
        int a = 12, b = 5; //Defining Data type and variable
        int var;
        int result1, result2;
        int februaryDays = 29;
        String result;
        int d = 0b1010;
        int e = 0b1100;
        System.out.println("Types of Operators");
        System.out.println("------------------------------------------------------");
        System.out.println("------------1) Arithmetic Operators----------- ");
        // addition operator
        System.out.println("a + b = " + (a + b));

        // subtraction operator
        System.out.println("a - b = " + (a - b));

        // multiplication operator
        System.out.println("a * b = " + (a * b));

        // division operator
        System.out.println("a / b = " + (a / b));

        // modulo operator
        System.out.println("a % b = " + (a % b));
        System.out.println("------------2) Assignment Operators ---------- ");
        // assign value using =
        var = a;
        System.out.println("var using =: " + var);

        // assign value using =+
        var += a;
        System.out.println("var using +=: " + var);

        // assign value using =*
        var *= a;
        System.out.println("var using *=: " + var);
        System.out.println("------------3) Relational Operators ---------- ");
        // value of a and b
```

```
System.out.println("a is " + a + " and b is " + b);

// == operator
System.out.println(a == b); // false

// != operator
System.out.println(a != b); // true

// > operator
System.out.println(a > b); // true

// < operator
System.out.println(a < b); // false

// >= operator
System.out.println(a >= b); // true

// <= operator
System.out.println(a <= b); // false
System.out.println("------------4) Logical Operators ---------- ");
// && operator
System.out.println((5 > 3) && (8 > 5)); // true
System.out.println((5 > 3) && (8 < 5)); // false

// || operator
System.out.println((5 < 3) || (8 > 5)); // true
System.out.println((5 > 3) || (8 < 5)); // true
System.out.println((5 < 3) || (8 < 5)); // false

// ! operator
System.out.println(!(5 == 3)); // true
System.out.println(!(5 > 3)); // false

System.out.println("------------5) Increment and Decrement        Operators ----------");
// original value
System.out.println("Value of a: " + a);

// increment operator
result1 = ++a;
System.out.println("After increment: " + result1);

System.out.println("Value of b: " + b);

// decrement operator
result2 = --b;
System.out.println("After decrement: " + result2);
```

```
        System.out.println("------------6) Bitwise Operators ---------- ");
        System.out.println("d & e : " + (d & e));
        System.out.println("d | e : " + (d | e));
        System.out.println("d ^ e : " + (d ^ e));
        System.out.println("~d : " + (~d));
        System.out.println("d << 2 : " + (d << 2));
        System.out.println("e >> 1 : " + (e >> 1));
        System.out.println("e >>> 1 : " + (e >>> 1));
        System.out.println("------------7) Ternary Operator ---------- ");
        // ternary operator
        result = (februaryDays == 28) ? "leap year" : "Not a Leap year";
        System.out.println(result);
    }
}
```

Output

```
Types of Operators
-----------------------------------------------------------
------------1) Arithmetic Operators-----------
a + b = 17
a - b = 7
a * b = 60
a / b = 2
a % b = 2
------------2) Assignment Operators-----------
var using =: 12
var using +=: 24
var using *=: 288
------------3) Relational Operators-----------
a is 12 and b is 5
false
true
true
false
true
false
------------4) Logical Operators-----------
true
false
true
true
false
true
false
------------5) Increment and Decrement Operators-----------
```

```
Value of a: 12
After increment: 13
Value of b: 5
After decrement: 4
------------6) Bitwise Operators-----------
d & e : 8
d | e : 14
d ^ e : 6
~d : -11
d << 2 : 40
e >> 1 : 6
e >>> 1 : 6
------------7) Ternary Operator-----------
Not a Leap year
```

Result: Thus the Program to implements Data types, variables and operators was successfully executed.

b. Program to implements arrays and control structures (if else block).

```
import java.util.Scanner;

public class ArrayControlStructureEx {
    public static void main(String[] args) {
        int arr[] = new int[10];
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter 10 elements in array: ");
        for(int i=0; i<arr.length; i++) {
            arr[i] = scan.nextInt();
        }
        for(int i=0; i<arr.length; i++) {
            if(arr[i]%2 == 0) {
                System.out.println(arr[i] + " is an Even Number.");
            } else {
                System.out.println(arr[i] + " is an Odd Number.");
            }
        }
    }
}
```

Output

```
Enter 10 elements in array:

3

8

9

6

7

44

99

33

25

57

3 is an Odd Number.

8 is an Even Number.

9 is an Odd Number.

6 is an Even Number.

7 is an Odd Number.

44 is an Even Number.

99 is an Odd Number.

33 is an Odd Number.

25 is an Odd Number.

57 is an Odd Number.
```

Result: Thus the Program to implements arrays and control structures was successfully executed.

Viva Voice Question:

1. What is the purpose of initializing a variable in Java?
2. What is the ‘this’ keyword in Java, and how is it used to refer to instance variables?
3. What are Relational operators in Java?
4. What is the difference between a string and an array of characters?
5. What is the ‘do-while’ loop in Java, and how is it different from the ‘while’ loop?

PRACTICAL No.3: Implement a Program to define class and constructors. Demonstrate Constructor

THEORY- Constructors in Java

In Java, a Constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling the constructor, memory for the object is allocated in the memory. It is a special type of method that is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called.

Types of Constructors in Java

Now is the correct time to discuss the types of the constructor, so primarily there are three types of constructors in Java are mentioned below:

Default Constructor
Parameterized Constructor
Copy Constructor

1. Default Constructor in Java

A constructor that has no parameters is known as default constructor. A default constructor is invisible. And if we write a constructor with no arguments, the compiler does not create a default constructor.

2. Parameterized Constructor in Java

A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.

3. Copy Constructor in Java

Unlike other constructors copy constructor is passed with another object which copies the data available from the passed object to the newly created object.

PROGRAM:

```
public class Student {
    String StudentName;
    int StudentId;

    Student() {
        System.out.println("Default constructor");
        System.out.println("Name of first student is " + StudentName);
        System.out.println("Id of first student is " + StudentId);
        System.out.println();
    }

    Student(String StudentName, int StudentId) {
        System.out.println("Parameterized constructor");
        this.StudentName= StudentName;
        this.StudentId = StudentId;
    }

    Student(Student student3){
        System.out.println("Copy constructor");
        this.StudentName = student3.StudentName;
        this.StudentId = student3.StudentId;
        System.out.println("Name of third student is " + StudentName +
        " and Id of third student is " + StudentId);
    }

    public static void main(String[] args) {
        Student student1 = new Student();

        Student student2 = new Student("Sarika", 25);
        System.out.println("Name of second student is " + student2.StudentName);
        System.out.println("Id of second student is " + student2.StudentId);
        System.out.println();

        Student student3 = new Student(student2);
    }
}
```

Output

```
Default constructor
Name of first student is null
Id of first student is 0

Parameterized constructor
Name of second student is Sarika
Id of second student is 25

Copy constructor
Name of third student is Sarika and Id of third student is 25
```

Result: Thus the Program to define class and constructors and to demonstrate Constructor was successfully executed.

Viva Voice Question:

1. Define Constructor?
2. Why constructors in Java cannot be static?
3. What are possible access modifiers that can be marked for a constructor?
4. How many types of constructors are in Java?
5. What is the main purpose of default constructor in java?

PRACTICAL No.4: Implement a program to define class, methods and objects. Demonstrate method overloading.

THEORY-

Method Overloading in Java

Method overloading in Java is the feature that enables defining several methods in a class having the same name but with different parameters lists. These algorithms may vary with regard to the number or type of parameters. When a method is called, Java decides which version of it to execute depending on the arguments given. If we have to perform only one operation, having the same name of the methods increases the readability of t**he program.**

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

1.Method Overloading: changing no. of arguments

Method overloading in Java allows defining multiple methods with the same name but different parameter lists. One common form of overloading is changing the number of arguments in the method signature. In this example, we have created two methods, the first add() method performs addition of two numbers, and the second add method performs addition of three numbers.

2.Method Overloading: changing data type of arguments

Method overloading in Java also allows changing the data type of arguments in the method signature. Here's an example demonstrating method overloading based on the data type of arguments: In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

PROGRAM:

```
class Addition {
    void add(int a, int b) {
        System.out.println("Addition of two integers is " + (a+b));
    }

    void add(int a, int b, int c) {
        System.out.println("Addition of three integers is " + (a+b+c));
    }

    void add(float a, float b, float c) {
        System.out.println("Addition of three floating numbers is " + (a+b+c));
    }
}
```

```
public class MethodOverloadingEx {

public static void main(String[] args) {
          Addition ad = new Addition();
          ad.add(6, 4);
          ad.add(7.6f, 3.8f, 8.2f);
          ad.add(3, 5, 4);
     }

}
```

Output

```
Addition of three integers is 66

Addition of two floating point numbers is 7.7

Addition of two integers is 10
```

Result: Thus the Program to define class, methods and objects and to demonstrate method overloading was successfully executed.

Viva Voice Question:

1. What is method overloading in Java?
2. How do we handle method overloading when inheritance is involved?
3. How does Java distinguish between overloaded methods?
4. Can a static method be overloaded?
5. What are the rules for method overloading?

PRACTICAL No.5: Implement a program to define inheritance and show method overriding.

THEORY-

Inheritance in Java

Java, Inheritance is an important pillar of OOP(Object-Oriented Programming). It is the mechanism in Java by which one class is allowed to inherit the features(fields and methods) of another class. In Java, Inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class. In addition, you can add new fields and methods to your current class as well.

Java Inheritance Types
Below are the different types of inheritance which are supported by Java.

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance
5. Hybrid Inheritance

Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

PROGRAM:

```
class A {
    int i;
    public A() {
        i=2;
    }
    void display() {
        System.out.println("Value of i is " + i);
    }
}

class B extends A {
    int j;
    public B() {
        j= 5;
    }
    void display() {
        super.display();
        System.out.println("Value of j is " + j);
        System.out.println("Value of i is " + i);
    }
}

public class MethodOverriding {
    public static void main(String[] args) {
        B b = new B();
        b.display();
    }
}
```

Output

```
Value of i is 5
Value if j is 8
Value if i is 5
```

Result: Thus the Program to define inheritance and show method overriding was successfully executed.

Viva Voice Question:

1. What does Java's inheritance mean?
2. Write the syntax for a class's subclass creation.
3. Why do Java programmers use inheritance?
4. What is Method Overriding?
5. What are the rules to be followed while overriding a method?

PRACTICAL No. 6: Implement a Program to demonstrate Packages.

THEORY-

In Java, **packages** are used to group related classes and interfaces together to improve modularity, maintainability, and reusability. Packages help prevent name conflicts and allow for better access control.

Types of Packages:

1. **Built-in Packages** – Provided by Java (e.g., java.util, java.io, javax.swing).
2. **User-defined Packages** – Created by developers to organize custom classes.

Creating and Using a Package:

- Use the package keyword at the start of the file.
- Compile the file with javac -d . ClassName.java to place it inside the package.
- Use import package_name.*; to access the package in another program.

Advantages of Packages:

- Avoids class name conflicts.
- Improves code reusability and maintenance.
- Provides controlled access to classes and interfaces.

PROGRAM:

```
//Example of package that import the packagename.*
//save by A.java
package pack;
public class A{
        public void msg(){
                System.out.println("Hello");
        }
}

//save by B.java
package mypack;
import pack.*;

class B{
        public static void main(String args[]){
                A obj = new A();
                obj.msg();
        }
}
```

Output

```
Hello
```

Result: Thus the Program to demonstrate packages was successfully executed.

Viva Voice Question:

1. What is the purpose of packages in Java?
2. How do you create and use a package?
3. What is the difference between built-in and user-defined packages?
4. How can you access a class from another package?
5. What is the role of the import statement?

PRACTICAL No. 7: Implement a Program to demonstrate Exception Handling.

THEORY:

Exception Handling in Java is used to handle runtime errors and maintain normal program execution. An exception is an unexpected event that disrupts program flow.

Types of Exceptions:

1. **Checked Exceptions** – Caught at compile-time (e.g., IOException).
2. **Unchecked Exceptions** – Occur at runtime (e.g., NullPointerException).
3. **User-defined Exceptions** – Custom exceptions created by extending Exception class.

Exception Handling Mechanisms:

- try – Defines the block of code that may cause an exception.
- catch – Handles the exception.
- finally – Executes code after try-catch, whether an exception occurs or not.
- throw – Manually throws an exception.
- throws – Declares exceptions that a method might throw.

Advantages of Exception Handling:

- Prevents abnormal termination of the program.
- Separates error-handling code from regular code.
- Helps maintain program flow.

PROGRAM:

```
public class ExceptionHandlingExample {

     public static void main(String[] args) {

          try{
             int a[]=new int[5];

             System.out.println(a[10]);
          }
          catch(ArithmeticException e) {
               System.out.println("Arithmetic Exception occurs");
          }
          catch(ArrayIndexOutOfBoundsException e) {
               System.out.println("ArrayIndexOutOfBounds Exception occurs");
          }
          catch(Exception e){
               System.out.println("Parent Exception occurs");
          }
          System.out.println("rest of the code");
     }
}
```

Output:

```
ArrayIndexOutOfBounds Exception occurs
rest of the code
```

Result: Thus the Program to demonstrate exception handling was successfully executed.

Viva Voice Question:

1. What is an exception in Java?
2. Explain the difference between throw and throws.
3. What is the purpose of the finally block?
4. Can a program have multiple catch blocks?
5. How do you create a user-defined exception?

PRACTICAL No. 8: Implement a Program to demonstrate Multithreading/Applet structure and event handling.

THEORY:

Multithreading:
Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Multithreading allows multiple threads to run simultaneously, improving performance and responsiveness.
Ways to Create Threads:

1. **Extending Thread class** – Override run() method.
2. **Implementing Runnable interface** – Pass to Thread object.

Thread Lifecycle:

There are multiple states of the thread in a lifecycle as mentioned below:

1. **New Thread:** When a new thread is created, it is in the new state. The thread has not yet started to run when the thread is in this state. When a thread lies in the new state, its code is yet to be run and hasn’t started to execute.
2. **Runnable State:** A thread that is ready to run is moved to a runnable state. In this state, a thread might actually be running or it might be ready to run at any instant of time. It is the responsibility of the thread scheduler to give the thread, time to run. A multi-threaded program allocates a fixed amount of time to each individual thread. Each and every thread get a small amount of time to run. After running for a while, a thread pauses and gives up the CPU so that other threads can run.
3. **Blocked:** The thread will be in blocked state when it is trying to acquire a lock but currently the lock is acquired by the other thread. The thread will move from the blocked state to runnable state when it acquires the lock.
4. **Waiting state:** The thread will be in waiting state when it calls wait() method or join() method. It will move to the runnable state when other thread will notify or that thread will be terminated.
5. **Timed Waiting:** A thread lies in a timed waiting state when it calls a method with a time-out parameter. A thread lies in this state until the timeout is completed or until a notification is received. For example, when a thread calls sleep or a conditional wait, it is moved to a timed waiting state.
6. **Terminated State:** A thread terminates because of either of the following reasons:

- Because it exits normally. This happens when the code of the thread has been entirely executed by the program.
- Because there occurred some unusual erroneous event, like a segmentation fault or an unhandled exception.

Applet Structure:
An **Applet** is a small Java program that runs inside a web browser or viewer. It uses lifecycle methods:

- init() – Called once when applet starts.
- start() – Called when applet is resumed.
- paint() – Used to draw on the screen.
- stop() – Called when applet is paused.
- destroy() – Final cleanup before closing.

Event Handling:
Java allows handling user actions (e.g., mouse clicks, key presses) using event listeners like ActionListener, MouseListener, KeyListener.

PROGRAM:

a) Program to demonstrate Multithreading

```
package oopsfile;

class A extends Thread {
	public void run(){
		for(int i=1;i<=5;i++){
			System.out.println("From Thread A="+i);
		}
		System.out.println("Exit from A");
	}
}

class B extends Thread{
	public void run(){
		for(int j=1;j<=5;j++){
			System.out.println("From Thread B="+j);
		}
		System.out.println("Exit from B");
	}
}

class C extends Thread{
	public void run(){
		for(int k=1;k<=5;k++){
			System.out.println("From Thread C="+k);
		}
		System.out.println("Exit from C");
	}
}

class MultithreadingEx {
	public static void main(String[] args) {
		new A().start();
		new B().start();
		new C().start();
	}
}
```

OUTPUT:

```
From Thread A=1
From Thread A=2
From Thread A=3
From Thread A=4
From Thread A=5
Exit from A
From Thread B=1
From Thread B=2
From Thread B=3
From Thread B=4
From Thread B=5
Exit from B
From Thread C=1
From Thread C=2
From Thread C=3
From Thread C=4
From Thread C=5
Exit from C
```

b) Program to demonstrate Applet structure

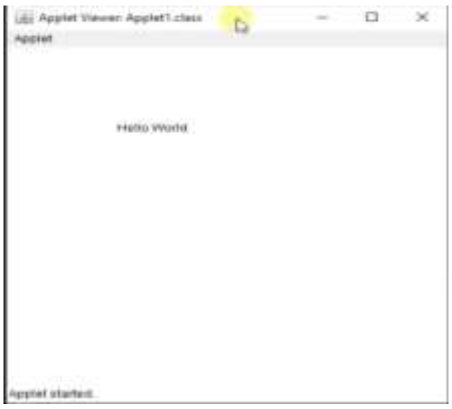
//Applet1 Java file

```
import java.applet.*;
import java.awt.*;

public class Applet1 extends Applet {
      public void paint(Graphics g) {
            g.drawString("Hello World", 100, 100);
    }
}
```

//HTML file A

```
<html>
      <title>Applet Example</title>
       <applet code="Applet1.class" width=400 height=400>
       </applet>
</html>
```

OUTPUT:



```
Hello World
```

```
Applet started.
```

Result: Thus the Program to demonstrate Multithreading/Applet structure and event handling was successfully executed.

Viva Voice Question:

1. What is multithreading in Java?
2. What are the benefits of using threads?
3. How is an applet different from a standalone Java application?
4. What are the major lifecycle methods of an applet?
5. What is event handling in Java?

PRACTICAL No. 9: Implement a Program to demonstrate I/O operations/Network Programming.

THEORY:

I/O Operations in Java:

Java I/O (Input/Output) operations are a crucial part of Java programming. They allow you to read and write data to and from various sources, such as files, the console, or network connections. Java provides the java.io package for handling input and output operations such as reading/writing files.

Basic I/O Concepts:

1. **Streams**

In Java, I/O is based on streams. A stream is a sequence of data elements that can be read or written sequentially. Java provides two types of streams:

- Input Streams: Used for reading data from a source.
- Output Streams: Used for writing data to a destination.

2. **Readers and Writers**

To work with character data, you can use readers and writers, which are specialized streams for text I/O. They are commonly used for reading and writing text files.

3. **Input and Output Streams Hierarchy**

Java I/O classes are organized in hierarchies. The basic hierarchy includes:

- InputStream and OutputStream for binary I/O.
- Reader and Writer for character-based I/O.

Common I/O Classes:

- FileReader / BufferedReader – Reads files.
- FileWriter / BufferedWriter – Writes to files.
- PrintWriter – Used for formatted output.

Network Programming in Java:

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

Java provides the java.net package to facilitate communication between computers over a network.

Key Networking Classes:

- Socket – Used for client-side connections.
- ServerSocket – Used for server-side connections.

- InetAddress – Handles IP address resolution.

Socket Programming

Java Socket programming is practiced for communication between the applications working on different JRE. Sockets implement the communication tool between two computers using TCP. Java Socket programming can either be connection-oriented or connection-less. In Socket Programming, Socket and ServerSocket classes are managed for connection-oriented socket programming. However, DatagramSocket and DatagramPacket classes are utilized for connection-less socket programming.
A client application generates a socket on its end of the communication and strives to combine that socket with a server. When the connection is established, the server generates an object of socket class on its communication end. The client and the server can now communicate by writing to and reading from the socket.

The **java.net.Socket** class describes a socket, and the **java.net.ServerSocket** class implements a tool for the server program to host clients and build connections with them.

Steps to establishing a TCP connection between two computing devices using Socket Programming
The following are the steps that occur on establishing a TCP connection between two computers using socket programming are given as follows:

Step 1 – The server instantiates a ServerSocket object, indicating at which port number communication will occur.
Step 2 – After instantiating the ServerSocket object, the server requests the accept() method of the ServerSocket class. This program pauses until a client connects to the server on the given port.
Step 3 – After the server is idling, a client instantiates an object of Socket class, defining the server name and the port number to connect to.
Step 4 – After the above step, the constructor of the Socket class strives to connect the client to the designated server and the port number. If communication is authenticated, the client forthwith has a Socket object proficient in interacting with the server.
Step 5 – On the server-side, the accept() method returns a reference to a new socket on the server connected to the client's socket.
After the connections are stabilized, communication can happen using I/O streams. Each object of a socket class has both an OutputStream and an InputStream. The client's OutputStream is correlated to the server's InputStream, and the client's InputStream is combined with the server's OutputStream. Transmission Control Protocol (TCP) is a two-way communication protocol. Hence information can be transmitted over both streams at the corresponding time.

Socket Class

The **Socket class** is used to create socket objects that help the users in implementing all fundamental socket operations. The users can implement various networking actions such as sending, reading data, and closing connections. Each Socket object created using **java.net.Socket** class has been correlated specifically with 1 remote host. If a user wants to connect to another host, then he must build a new socket object.

ServerSocket Class

The **ServerSocket class** is used for providing system-independent implementation of the server-side of a client/server Socket Connection. The constructor for ServerSocket class throws an exception if it can't listen on the specified port. For example – it will throw an exception if the port is already being used.

PROGRAM:

a) Program to demonstrate I/O operations.

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class FileWriteReadEx {
	public static void main(String[] args) throws IOException {
		System.out.println("Writing text into the given file:");
		System.out.println("Please enter a string:");
		Scanner sc1 = new Scanner(System.in);
		String str = sc1.nextLine();

		FileWriter fw = new FileWriter("C:\\PCE\\OOPJ - 4th Sem\\Test.txt");

		fw.write(str);

		fw.close();
		System.out.println("Text has been successfully entered in the file");
		System.out.println();
		System.out.println();
		System.out.println("Reading text from the given file:");
		System.out.println("The text entered in the given file is:");
		try {
			FileInputStream fis  = new FileInputStream("C:\\PCE\\OOPJ - 4th Sem\\Test.txt");
			Scanner sc2 = new Scanner(fis);
			while(sc2.hasNext()) {
				System.out.println(sc2.nextLine());
			}
		} catch (FileNotFoundException e) {

			e.printStackTrace();
		}
	}

}
```

OUTPUT:

```
Writing text into the given file:
Please enter a string:
Hi. This is an example for I/O Operations. Here we are reading and writing into the given file.
Text has been successfully entered in the file


Reading text from the given file:
The text entered in the given file is:
Hi. This is an example for I/O Operations. Here we are reading and writing into the given file.
```

b) Program to demonstrate Network Programming.

```
// MyServer Java file
import java.io.*;
import java.net.*;

public class MyServer {
	public static void main(String[] args){
		try{
			ServerSocket ss=new ServerSocket(6666);
			Socket s=ss.accept();//establishes connection
```

```
                DataInputStream dis=new DataInputStream(s.getInputStream());
                String  str=(String)dis.readUTF();
                System.out.println("message= "+str);
                ss.close();
            }catch(Exception e){
                System.out.println(e);
            }
        }
}

// MyClient Java file
import java.io.*;
import java.net.*;

public class MyClient {
        public static void main(String[] args) {
            try{
                Socket s=new Socket("localhost",6666);
                DataOutputStream dout=new DataOutputStream(s.getOutputStream());
                dout.writeUTF("Hello Server");
                dout.flush();
                dout.close();
                s.close();
            }catch(Exception e){
                System.out.println(e);
            }
        }
}
```

OUTPUT:

```
message= Hello Server
```

Result: Thus the Program to demonstrate I/O operations/Network Programming was successfully executed.

Viva Voice Question:

1. What are the different types of streams in Java I/O?
2. What is the difference between FileReader and BufferedReader?
3. What is socket programming in Java?
4. How does a client communicate with a server in Java?
5. What is the difference between TCP and UDP?

PRACTICAL No. 10: Implement a Program to demonstrate event handling using GUI.

THEORY:

Event Handling in Java is a mechanism that allows the program to respond to user interactions such as button clicks, mouse movements, and key presses.

Java GUI Components:

- **Swing (javax.swing)** – Modern GUI framework (e.g., JButton, JLabel, JTextField).
- **AWT (java.awt)** – Older GUI framework (e.g., Frame, Button, Label).

Event Handling Mechanism:

1. **Event Source** – The GUI component that generates an event (e.g., a button).
2. **Event Listener** – An interface that defines the event response.
3. **Event Object** – Encapsulates event details.
4. **Event Handler** – A method that processes the event.

Common Event Listeners:

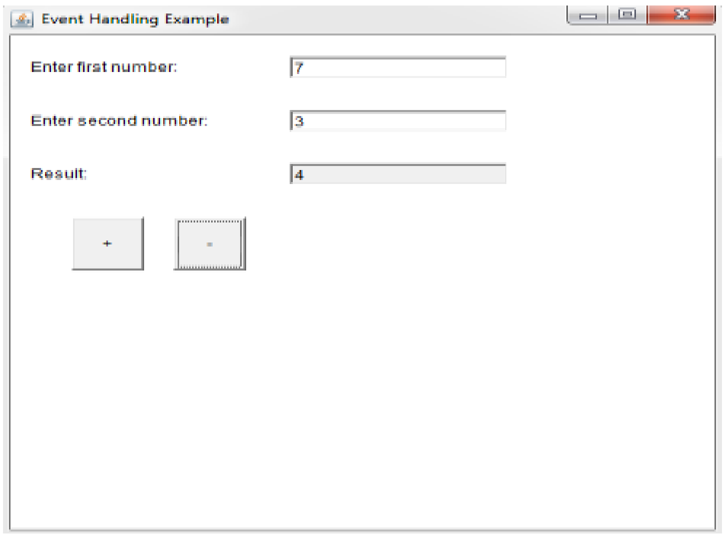
- ActionListener – Handles button clicks.
- MouseListener – Detects mouse events.
- KeyListener – Captures keyboard input.

PROGRAM:

```
package com.demo.eventhandling;

import java.awt.*;
import java.awt.event.*;

public class TextFieldExample extends Frame implements ActionListener {
        TextField tf1, tf2, tf3;
        Button b1, b2;
        Label l1, l2, l3;

        TextFieldExample() {
                Frame f= new Frame("Event Handling Example");
                l1 = new Label("Enter first number:");
                l1.setBounds(20, 50, 150, 20);
                tf1 = new TextField();
                tf1.setBounds(200, 50, 150, 20);

                l2 = new Label("Enter second number:");
                l2.setBounds(20, 100, 150, 20);
                tf2 = new TextField();
                tf2.setBounds(200, 100, 150, 20);

                l3 = new Label("Result:");
                l3.setBounds(20, 150, 150, 20);
                tf3 = new TextField();
                tf3.setBounds(200, 150, 150, 20);
                tf3.setEditable(false);

                b1 = new Button("+");
```

```
		b1.setBounds(50, 200, 50, 50);
		b2 = new Button("-");
		b2.setBounds(120, 200, 50, 50);
		b1.addActionListener(this);
		b2.addActionListener(this);
		f.add(l1);
		f.add(l2);
		f.add(l3);
		f.add(tf1);
		f.add(tf2);
		f.add(tf3);
		f.add(b1);
		f.add(b2);
		f.setSize(500, 500);
		f.setLayout(null);
		f.setVisible(true);
	}

	public void actionPerformed(ActionEvent e) {
		String s1 = tf1.getText();
		String s2 = tf2.getText();
		int a = Integer.parseInt(s1);
		int b = Integer.parseInt(s2);
		int c = 0;
		if (e.getSource() == b1) {
			c = a + b;
		} else if (e.getSource() == b2) {
			c = a - b;
		}
		String result = String.valueOf(c);
		tf3.setText(result);
	}

	public static void main(String[] args) {
		new TextFieldExample();
	}
}
```

OUTPUT:



```
Enter first number:     7
Enter second number:    3
Result:                 4
```

Result: Thus the Program to demonstrate event handling using GUI was successfully executed.

Viva Voice Question:

1. What is event handling in Java?
2. What is the difference between AWT and Swing?
3. What is the purpose of the ActionListener interface?
4. What is an event source and event object?
5. How can multiple components share the same event handler?