# PRACTICAL 4

**Write a program to demonstrate use of method overloading to calculate area of square, rectangle and triangle.**

**Explanation:**

* **class AreaCal { }:** Defines a class named AreaCal that contains methods for calculating the area of different shapes.

* **int cal\_Area(int side) / int cal\_Area(int length, int breadth) / double cal\_Area(double base, double height):** Defines an overloaded method cal\_Area that calculates the area of a rectangle given its length and breadth.

* **AreaCal cal = new AreaCal();** Creates a new instance of the AreaCal class named cal.

* **Scanner sc = new Scanner(System.in);** Creates a Scanner object sc to read input from the console

* **int Side = sc.nextInt();** Reads the integer input for the side length of the square.

* **System.out.println("Area of the square: " + cal.cal\_Area(Side));** Calls the cal\_Area method to calculate and print the area of the square.

* **System.out.println("Area of the rectangle: " + cal.cal\_Area(rectl, rectb));** Calls the overloaded cal\_Area method to calculate and print the area of the rectangle

* **System.out.println("Area of the triangle: " + cal.cal\_Area(triBase, triHeight));** Calls the overloaded cal\_Area method to calculate and print the area of the triangle.

**Code:**

import java.util.Scanner; class AreaCal { public int cal\_Area(int side) { return side \* side;

}

public int cal\_Area(int length, int breadth) { return length \* breadth;

}

public double cal\_Area(double base, double height) {

return 0.5 \* base \* height;

}

public static void main(String[] args) {

System.out.println("35221402023 Parshv Jain");

System.out.println(" ");

AreaCal cal = new AreaCal();

Scanner sc = new Scanner(System.in);

System.out.print("Enter the side length of the square: "); int Side = sc.nextInt();

System.out.println("Area of the square: " + cal.cal\_Area(Side));

System.out.println(" ");

System.out.print("Enter the length of the rectangle: "); int rectl= sc.nextInt();

System.out.print("Enter the breadth of the rectangle: ");

int rectb = sc.nextInt();

System.out.println("Area of the rectangle: " + cal.cal\_Area(rectl, rectb));

System.out.println(" ");

System.out.print("Enter the base of the triangle: "); double triBase = sc.nextDouble();

System.out.print("Enter the height of the triangle: "); double triHeight = sc.nextDouble();

System.out.println("Area of the triangle: " + cal.cal\_Area(triBase, triHeight));

}

}

# PRACTICAL 5

**Create a class employee which have name, age and address of employee, include methods getdata() and showdata(), getdata() takes the input from the user, showdata() display the data in following format:**

**Name:**

**Age:**

**Address:**

**Explanation:**

* **class Employee {}** Defines a class named Employee that represents an employee with attributes like age, name, and address.

* **int age; String name; String address;:** Declares instance variables age, name, and address to store the employee's details.

* **public void getdata()** {: Defines a method getdata() that prompts the user to enter the employee's name, age, and address.

* **name = s.nextLine();** Reads the user's input for the employee's name.

* **age = s.nextInt();** Reads the user's input for the employee's age.

* **s.nextLine();** Consumes the newline character left in the input buffer after reading the integer for age.

* **Employee obj = new Employee();** Creates a new instance of the Employee class named obj.

* **obj.getdata();**  Calls the getdata() method on the obj object to read the employee's details from the user.

* **obj.showdata();** Calls the showdata() method on the obj object to display the employee's details.

**Code:**

import java.util.Scanner; class Employee { int age;

String name;

String address;

Scanner s=new Scanner(System.in); public void getdata()

{

System.out.println("Enter your name:"); name=s.nextLine();

System.out.println("Enter your age:");

age=s.nextInt();

s.nextLine();

System.out.println("Enter your address:"); address=s.nextLine();

}

public void showdata()

{

System.out.println("Name:"+name);

System.out.println("Age:"+age);

System.out.println("Addess"+address);

}

public static void main(String[] args)

{

System.out.println("35221402023 Parshv Jain"); System.out.println(" ");

Employee obj=new Employee();

obj.getdata();

System.out.println(" ");

obj.showdata();

}

}

# PRACTICAL 6

**Write a program to demonstrate concept of “this”.**

**Explanation:**

* **class temp {}** Defines a class named temp that contains methods and variables related to a person's age and name.

* **int age; String name;:** Declares instance variables age and name to store the person's age and name.

* **public void get(int age, String name) {}** Defines a method get that takes parameters for age and name to set the instance variables.

* **this.name = name; this.age = age;** Uses the this keyword to assign the method parameters to the instance variables, distinguishing them from each other.

* **public static void main(String args[]**  Defines the main method, which is the entry point of the program.

* **temp obj = new temp();** Creates a new instance of the temp class named obj.

* **obj.get(18, "Harsh");** Calls the get method on the obj object to set the age to 18 and the name to "Harsh".

* **if (obj.age >= 18)** Checks if the age of the obj object is 18 or older.

* **System.out.println(obj.name + ", You are Eligible to vote.");**  Prints a message indicating that the person is eligible to vote if the age condition is true.

* **else {}** Begins the else block that executes if the age condition is false.

**Code:**

class temp{ int age;

String name; public void get(int age,String name)

{

this.name=name; this.age=age;

} public static void main(String args[])

{

System.out.println("35221402023 Parshv Jain"); System.out.println(" "); temp obj=new temp(); obj.get(18,"Harsh"); if (obj.age >= 18)

{

System.out.println(obj.name+",You are Eligible to vote.");

} else

{

System.out.println(obj.name+",You are not Eligible to vote.");

}}}

Printed

**PRACTICAL 7**

**WAP to read the array dynamically, sort the array and display the sorted array.**

**Explanation:**

This program demonstrates how to read an array dynamically from the user, sort the array, and display the sorted array. It uses the Scanner class to input the number of elements and the array elements, then it uses the Arrays.sort() method to sort the array, and finally, the sorted array is displayed using Arrays.toString().

1. **Array Declaration and User Input:**

* The program first prompts the user to enter the number of elements in the array.
* It creates an array arr with the size n and then uses a loop to fill the array with user input values.

1. **Sorting the Array:**

* The Arrays.sort() method is used to sort the array in ascending order.

1. **Displaying the Sorted Array:**

* Finally, the sorted array is printed using Arrays.toString(arr) to display the contents in a readable format.

**Code:**

import java.util.Scanner;

import java.util.Arrays;

public class SortArray {

public static void main(String args[]){

System.out.println("Name: Parshv Jain\n35221402023");

Scanner scan = new Scanner(System.in);

System.out.println("Enter the no of Elements : ");

int n = scan.nextInt();

int arr[] = new int[n];

for(int i=0;i<n;i++){

System.out.println("Enter " +(i+1)+" Elements : ");

arr[i] = scan.nextInt();

}

Arrays.sort(arr);

System.out.println("Sorted array : "+Arrays.toString(arr));

}

}

**PRACTICAL 8**

**Write a program to implement all string operations.**

**Explanation:**

This program demonstrates various string operations in Java. It takes user input for a string and then performs several operations such as calculating its length, concatenation, character access, substring extraction, comparison, search, replacement, conversion to uppercase and lowercase, trimming, checking if it's empty, and checking if it's a palindrome.

1. **String Operations:** The program handles various string operations like calculating length, concatenating, comparing, checking for substring, replacing characters, changing case, trimming spaces, and checking for palindromes.
2. **User Interaction:** It prompts the user to input strings for each operation and outputs the result accordingly.

**Code:**

import java.util.Scanner;

public class StringOperations {

public static void main(String[] args) {

System.out.println("Name: Parshv \n35221402023");

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a string: ");

String input = scanner.nextLine();

System.out.println("Length of the string: " + input.length());

System.out.print("Enter another string to concatenate: ");

String anotherString = scanner.nextLine();

System.out.println("Concatenated string: " + input + anotherString);

System.out.print("Enter index to access character: ");

int index = scanner.nextInt();

if (index >= 0 && index < input.length()) {

System.out.println("Character at index " + index + ": " + input.charAt(index));

} else {

System.out.println("Invalid index!");

}

System.out.print("Enter start index for substring: ");

int start = scanner.nextInt();

System.out.print("Enter end index for substring: ");

int end = scanner.nextInt();

if (start >= 0 && end <= input.length() && start < end) {

System.out.println("Substring: " + input.substring(start, end));

} else {

System.out.println("Invalid indices for substring!");

}

scanner.nextLine();

System.out.print("Enter another string to compare: ");

String compareString = scanner.nextLine();

if (input.equals(compareString)) {

System.out.println("The strings are equal.");

} else {

System.out.println("The strings are not equal.");

}

System.out.print("Enter a character or substring to search: ");

String searchString = scanner.nextLine();

if (input.contains(searchString)) {

System.out.println("Found '" + searchString + "' in the string.");

} else {

System.out.println("Substring not found.");

}

System.out.print("Enter character or substring to replace: ");

String toReplace = scanner.nextLine();

System.out.print("Enter replacement string: ");

String replacement = scanner.nextLine();

System.out.println("String after replacement: " + input.replace(toReplace, replacement));

System.out.println("Uppercase: " + input.toUpperCase());

System.out.println("Lowercase: " + input.toLowerCase());

System.out.println("Trimmed string: '" + input.trim() + "'");

if (input.isEmpty()) {

System.out.println("The string is empty.");

} else {

System.out.println("The string is not empty.");

}

String reversed = new StringBuilder(input).reverse().toString();

if (input.equals(reversed)) {

System.out.println("The string is a palindrome.");

} else {

System.out.println("The string is not a palindrome.");

}

scanner.close();

}

}

**PRACTICAL 9**

**Write a program to demonstrate multi-level and hierarchical inheritance.**

**Explanation:**

This program demonstrates both **multi-level** and **hierarchical inheritance** in Java. In multi-level inheritance, a class is derived from another class, which in turn is derived from another class. In hierarchical inheritance, multiple classes inherit from a single base class. The program shows how a Dog and Cat class inherit from the Animal class, and how the Puppy class extends the Dog class, illustrating both inheritance types.

1. **Multi-level Inheritance:**

* The Puppy class extends the Dog class, which itself extends the Animal class. This creates a chain of inheritance: Puppy → Dog → Animal.
* The Puppy class inherits the eat() method from Animal and bark() from Dog, in addition to its own play() method.

1. **Hierarchical Inheritance:**

The Dog and Cat classes both inherit from the Animal class. This means both classes share the eat() method from the Animal class, but each has its own additional functionality: bark() for Dog and meow() for Cat.

**Code:**

class Animal {

void eat() {

System.out.println("This animal eats food.");

}

}

class Dog extends Animal {

void bark() {

System.out.println("The dog barks.");

}

}

class Puppy extends Dog {

void play() {

System.out.println("The puppy is playing.");

}

}

class Cat extends Animal {

void meow() {

System.out.println("The cat meows.");

}

}

// Main class to demonstrate the inheritance

public class AnimalCreatures {

public static void main(String[] args) {

System.out.println("Name: Parshv\n35221402023");

// Multi-level

Puppy puppy = new Puppy();

puppy.eat();

puppy.bark();

puppy.play();

System.out.println();

// Hierarchical

Dog dog = new Dog();

dog.eat();

dog.bark();

Cat cat = new Cat();

cat.eat();

cat.meow();

}

}

**PRACTICAL 10**

**Write a program to demonstrate run-time polymorphism.**

**Explanation:**

This program demonstrates **run-time polymorphism** (also known as dynamic method dispatch) in Java. Run-time polymorphism allows a method to behave differently based on the object that invokes it, even if the object is referred to by a reference of the base class. In this example, the sound() method is overridden in the Dog1 and Cat1 classes, but the appropriate version of the method is invoked based on the actual object type (either Dog1 or Cat1), not the reference type (Animal1).

1. **Base Class (Animal1):**

* The sound() method is defined in the base class Animal1, and it prints a generic message: "Animal makes a sound."

1. **Overridden Methods:**

* The Dog1 and Cat1 classes extend Animal1 and override the sound() method. In Dog1, it prints "Dog barks", and in Cat1, it prints "Cat meows".

1. **Run-time Polymorphism:**

* In the main() method, objects of Animal1, Dog1, and Cat1 are created, but they are all referred to by the Animal1 reference type.

The actual method that gets called is determined at run-time based on the object's actual type (Dog1 or Cat1), not the reference type (Animal1). This is the core concept of run-time polymorphism.

**Code:**

class Animal1 {

// Method to be overridden

public void sound() {

System.out.println("Animal makes a sound");

}

}

class Dog1 extends Animal1 {

@Override

public void sound() {

System.out.println("Dog barks");

}

}

class Cat1 extends Animal1 {

@Override

public void sound() {

System.out.println("Cat meows");

}

}

public class AnimalPloy {

public static void main(String[] args) {

System.out.println("Name: Parshv\n35221402023");

Animal1 myAnimal = new Animal1();

Animal1 myDog = new Dog1();

Animal1 myCat = new Cat1();

myAnimal.sound();

myDog.sound();

myCat.sound();

}}

**PRACTICAL 11**

**Write a program to use super() to invoke base class constructor.**

**Explanation:**

This program demonstrates the use of super() to invoke the constructor of the base class (Person) from the subclass (Student). The super() keyword is used to call the parent class's constructor before executing the subclass's constructor code. This ensures that the parent class is properly initialized before the subclass is initialized.

1. **Base Class (Person):**

* The Person class has a constructor that takes name and age as parameters and initializes the respective fields. It then prints a message containing the name and age.

1. **Subclass (Student):**

* The Student class extends Person and adds a new field studentId. Its constructor takes name, age, and studentId as parameters.
* The super(name, age) call invokes the constructor of the Person class, initializing the fields inherited from Person. After that, the studentId field is initialized, and a message specific to the student is printed.

1. **Using super():**

* The super(name, age) call ensures that the Person class is properly initialized before the Student class adds its own initialization. This is necessary because the Student class inherits from Person, and the parent class must be initialized first.

1. **Output:**

* When a Student object is created in the main method, the program first calls the Person constructor via super(), then the Student constructor.

**Code:**

class Person {

String name;

int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

System.out.println("Person constructor called: " + name + ", Age: " + age);

}

}

class Student extends Person {

String studentId;

public Student(String name, int age, String studentId) {

super(name, age);

this.studentId = studentId;

System.out.println("Student constructor called: Student ID: " + studentId);

}

}

public class College {

public static void main(String[] args) {

System.out.println("Name: Parshv \n35221402023");

Student student = new Student("Alice", 20, "S12345");

}

}

**PRACTICAL 12**

**Write a program to demonstrate the use of static variable, static method and static block.**

**Explanation:**

This program demonstrates the use of static variables, static methods, and static blocks in Java. A static variable is shared among all instances of a class, a static method belongs to the class rather than an instance, and a static block is used for static initialization that runs once when the class is loaded.

1. **Static Variable (staticVariable):**

* staticVariable is a class variable shared by all instances of the class. It is initialized to 0 but modified later by a static block and the constructor.

1. **Static Block:**

* The static block is executed once when the class is first loaded. In this case, it initializes the staticVariable to 10 and prints "Static block is executed."

1. **Static Method (staticMethod()):**

* The staticMethod() is a class method that can be called without creating an object of the class. It prints the value of the staticVariable.

1. **Constructor:**

* The constructor of MyClass increments the staticVariable by 1 each time an object is created. This shows that static variables are shared across instances.

1. **Main Method (bca Class):**

* The main method demonstrates the usage of static elements:
* It prints the value of the static variable before and after creating an object of MyClass.
* It calls the static method before and after object creation.

**Code:**

class MyClass {

static int staticVariable = 0;

static {

System.out.println("Static block is executed.");

staticVariable = 10;

}

public static void staticMethod() {

System.out.println("This is a static method. Static variable value: " + staticVariable);

}

public MyClass() {

System.out.println("Constructor is called.");

staticVariable++;

}

}

public class bca {

public static void main(String[] args) {

System.out.println("Name: Parshv\n35221402023");

System.out.println("Static variable value: " + MyClass.staticVariable);

MyClass.staticMethod();

MyClass obj = new MyClass();

System.out.println("After creating object, static variable value: " + MyClass.staticVariable);

MyClass.staticMethod();

}

}

**PRACTICAL 13**

**Write a program to demonstrate the concept of abstract class**

**Explanation:**

This program demonstrates the concept of **abstract classes** in Java. An abstract class is a class that cannot be instantiated on its own and may contain abstract methods (methods without a body) that must be implemented by subclasses. In this example, we have an abstract Car class with an abstract method startEngine() and a concrete method displayDetails(). The subclasses Sedan and SUV extend the Car class and provide their own implementations of the startEngine() method.

1. **Abstract Method:** startEngine() is declared abstract in the Car class and must be implemented in the subclasses Sedan and SUV.
2. **Abstract Class:** The Car class cannot be instantiated directly, but its subclasses (Sedan and SUV) can be.
3. **Polymorphism:** The startEngine() method is polymorphic, as it behaves differently depending on the type of car (either Sedan or SUV).

**Code:**

abstract class Car {

String brand;

int year;

public Car(String brand, int year) {

this.brand = brand;

this.year = year;

}

public abstract void startEngine();

public void displayDetails() {

System.out.println("Brand: " + brand);

System.out.println("Year: " + year);

}

}

class Sedan extends Car {

public Sedan(String brand, int year) {

super(brand, year);

}

public void startEngine() {

System.out.println("The " + brand + " sedan's engine starts with a smooth hum.");

}

}

class SUV extends Car {

public SUV(String brand, int year) {

super(brand, year);

}

public void startEngine() {

System.out.println("The " + brand + " SUV's engine roars to life.");

}

}

public class MetalBox {

public static void main(String[] args) {

System.out.println("Name: Parshv\n35221402023");

Car sedan = new Sedan("Toyota", 2023);

Car suv = new SUV("Ford", 2022);

sedan.displayDetails();

sedan.startEngine();

System.out.println();

suv.displayDetails();

suv.startEngine();

}

}

**PRACTICAL 14**

**Write a program to demonstrate the concept of interface**

**Explanation:**

This program demonstrates the concept of **interfaces** in Java. Interfaces define a contract of methods that a class must implement. In this example, we have three interfaces: Drivable, Fuelable, and ElectricVehicle. The Car1 class implements Drivable and Fuelable, while the ElectricScooter class implements Drivable and ElectricVehicle. This illustrates how multiple interfaces can be implemented by a single class.

1. **Interfaces (Drivable, Fuelable, ElectricVehicle):**

* Drivable: Defines the method drive() that any class implementing this interface must provide.
* Fuelable: Defines the method refuel() for refueling a fuel-powered vehicle.
* ElectricVehicle: Defines the method charge() for charging an electric vehicle.

1. **Classes Implementing Interfaces:**

* Car1: Implements both the Drivable and Fuelable interfaces. It provides the implementation for the drive() method (indicating that a car drives) and the refuel() method (indicating that a car refuels with gasoline).
* ElectricScooter: Implements the Drivable and ElectricVehicle interfaces. It provides the implementation for the drive() method (indicating that an electric scooter rides) and the charge() method (indicating that an electric scooter charges).

1. **Main Method (Fuel Class):**

* In the main() method, objects of Car1 and ElectricScooter are created and the appropriate methods are called for each object:
* car.drive(): Drives the car.
* scooter.drive(): Rides the electric scooter.
* car.refuel(): Refuels the car.
* scooter.charge(): Charges the electric scooter.

**Code:**

interface Drivable {

void drive();

}

interface Fuelable {

void refuel();

}

interface ElectricVehicle {

void charge();

}

class Car1 implements Drivable, Fuelable {

String brand;

public Car1(String brand) {

this.brand = brand;

}

@Override

public void drive() {

System.out.println(brand + " car is driving on the road.");

}

@Override

public void refuel() {

System.out.println(brand + " car is refueling with gasoline.");

}

}

class ElectricScooter implements Drivable, ElectricVehicle {

String brand;

public ElectricScooter(String brand) {

this.brand = brand;

}

@Override

public void drive() {

System.out.println(brand + " electric scooter is riding on the road.");

}

@Override

public void charge() {

System.out.println(brand + " electric scooter is charging.");

}

}

public class Fuel {

public static void main(String[] args) {

System.out.println("Name: Parshv\n35221402023");

Car1 car = new Car1("Toyota");

ElectricScooter scooter = new ElectricScooter("Xiaomi");

car.drive();

scooter.drive();

car.refuel();

scooter.charge();

}

}

Printed

# PRACTICAL 15

**Write a program to demonstrate checked exception and unchecked exception.**

**Explanation:**

**readFile() Method**

* Tries to open a file using FileReader.
* If the file doesn't exist, it throws FileNotFoundException, which is a **checked exception**.
* The compiler **forces you** to handle it using try-catch or declare with throws.

**Division by Zero**

* In main(), a / b where b = 0 causes ArithmeticException, an **unchecked exception**.
* This is caught using a try-catch block, but the compiler **does not force** you to catch it.

**Code:**

import java.io.\*;

public class ExceptionDemo {

public static void readFile(String filename) throws IOException {

FileReader file = new FileReader(filename);

BufferedReader fileInput = new BufferedReader(file);

System.out.println("Reading file:");

System.out.println(fileInput.readLine());

fileInput.close();

}

public static void main(String[] args) {

try {

readFile("example.txt");

} catch (IOException e) {

System.out.println("Name: Parshv\n35221402023");

System.out.println("Checked Exception Caught: " + e.getMessage());

}

try {

int result = 10 / 0;

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("Unchecked Exception Caught: " + e.getMessage());

}

}

}

# **PRACTICAL 16**

**Write a program declaring a Java class called SavingsAccount with members**

**``accountNumber`` and ``Balance``. Provide member functions as**

**``depositAmount ()`` and ``withdrawAmount ()``. If user tries to withdraw an amount greater than their balance then throw a user-defined exception.**

**Explanation:**

**🔹 InsufficientBalanceException**

* Custom class extending Exception.
* Allows you to define your own message when balance is not enough.

**SavingsAccount**

Encapsulates account number and balance.

depositAmount() adds to the balance.

withdrawAmount() checks if the amount exceeds balance:

If yes, throws InsufficientBalanceException.

**BankApp (Main class)**

* Creates an account.
* Performs deposit and withdraw operations.
* Catches custom exception if withdrawal fails.

**Code:**

class InsufficientBalanceException extends Exception {

public InsufficientBalanceException(String message) {

super(message);

}

}

class SavingsAccount {

private String accountNumber;

private double balance;

public SavingsAccount(String accountNumber, double initialBalance) {

this.accountNumber = accountNumber;

this.balance = initialBalance;

}

public void depositAmount(double amount) {

if (amount > 0) {

balance += amount;

System.out.println("Amount deposited: $" + amount);

} else {

System.out.println("Deposit amount must be positive.");

}

}

public void withdrawAmount(double amount) throws InsufficientBalanceException {

if (amount > balance) {

throw new InsufficientBalanceException("Withdrawal failed: Insufficient balance.");

} else if (amount <= 0) {

System.out.println("Withdrawal amount must be positive.");

} else {

balance -= amount;

System.out.println("Amount withdrawn: $" + amount);

}

}

public void displayBalance() {

System.out.println("Current Balance: $" + balance);

}

}

public class BankApp {

public static void main(String[] args) {

System.out.println("Name: Parshv\n35221402023");

SavingsAccount acc = new SavingsAccount("ACC12345", 500.0);

acc.displayBalance();

acc.depositAmount(200.0);

acc.displayBalance();

try {

acc.withdrawAmount(800.0);

} catch (InsufficientBalanceException e) {

System.out.println(e.getMessage());

}

acc.displayBalance();

}

}

# **PRACTICAL 17**

**Write a program to demonstrate creation of multiple child threads.**

**Explanation:**

**What is a Thread?**

A **thread** is a lightweight process. In Java, you can create a thread in two ways:

1. By **extending Thread** class.
2. By **implementing Runnable** interface.

**In This Program:**

* We extend the Thread class to create MyThread.
* Each thread prints numbers from 1 to 5 with a delay.
* The main thread also prints its own count.
* All threads run **concurrently**, so outputs from different threads may **interleave**.

**Code:**

class MyThread extends Thread {

private String threadName;

public MyThread(String name) {

this.threadName = name;

}

public void run() {

for (int i = 1; i <= 5; i++) {

System.out.println(threadName + " - Count: " + i);

try {

Thread.sleep(500);

} catch (InterruptedException e) {

System.out.println(threadName + " interrupted.");

}

}

System.out.println(threadName + " finished.");

}

}

public class MultiThreadDemo {

public static void main(String[] args) {

System.out.println("Name: Parshv\n35221402023");

MyThread t1 = new MyThread("Child Thread 1");

MyThread t2 = new MyThread("Child Thread 2");

MyThread t3 = new MyThread("Child Thread 3");

t1.start();

t2.start();

t3.start();

for (int i = 1; i <= 3; i++) {

System.out.println("Main Thread - Count: " + i);

try {

Thread.sleep(700);

} catch (InterruptedException e) {

System.out.println("Main thread interrupted.");

}

}

System.out.println("Main thread finished.");

} }

# **PRACTICAL 18**

**Write a program creating threads using Runnable interface. Print your name in ``run ()`` method of class.**

**Explanation:**

**Runnable Interface**

* A functional interface with one method: public void run().
* You implement it when you don't want to extend the Thread class (e.g., your class already extends another class).

**How It Works:**

1. MyRunnable implements Runnable and overrides the run() method.
2. In main(), we create a Thread and pass our MyRunnable object to it.
3. Calling start() begins the thread, which runs the run() method and prints the name.

**Code:**

class MyRunnable implements Runnable {

public void run() {

for (int i = 1; i <= 5; i++) {

System.out.println("My name is John Doe - Count: " + i);

try {

Thread.sleep(500); // Pause for 500ms

} catch (InterruptedException e) {

System.out.println("Thread interrupted.");

}

}

System.out.println("Thread finished.");

}

}

public class RunnableThreadDemo {

public static void main(String[] args) {

System.out.println("Name: Parshv\n35221402023");

MyRunnable myRunnable = new MyRunnable();

Thread thread = new Thread(myRunnable);

thread.start();

System.out.println("Main thread is running...");

}

}

# **PRACTICAL 19**

**Write a swing application that uses at least 5 swing controls.**

**Explanation:**

| **Swing Control** | **Purpose** |
| --- | --- |
| JLabel- | Displays static text like a label |
| JTextField- | Lets the user enter text input |
| JCheckBox- | Allows the user to make a yes/no choice |
| JComboBox- | Provides a drop-down list of options |
| JButton- | Performs an action when clicked |

**Code:**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class SwingExample extends JFrame {

public SwingExample() {

System.out.println("Name: Parshv\n35221402023");

setTitle("Swing Controls Demo");

setLayout(new FlowLayout());

JLabel nameLabel = new JLabel("Enter your name:");

JTextField nameField = new JTextField(15);

JCheckBox agreeCheck = new JCheckBox("I agree to terms");

String[] countries = {"India", "USA", "UK", "Canada", "Australia"};

JComboBox<String> countryBox = new JComboBox<>(countries);

JButton submitButton = new JButton("Submit");

submitButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String name = nameField.getText();

String country = (String) countryBox.getSelectedItem();

boolean isAgreed = agreeCheck.isSelected();

if (name.isEmpty()) {

JOptionPane.showMessageDialog(null, "Please enter your name.");

} else if (!isAgreed) {

JOptionPane.showMessageDialog(null, "Please agree to the terms.");

} else {

JOptionPane.showMessageDialog(null, "Hello " + name + " from " + country + "!");

}

}

});

add(nameLabel);

add(nameField);

add(agreeCheck);

add(countryBox);

add(submitButton);

setSize(300, 200);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) {

new SwingExample();

}

}

# **PRACTICAL 20**

**Write a program to demonstrate any event handling.**

**Explanation:**

**What is Event Handling?**

Event handling in Java means writing code that **responds to user actions** — like clicks, key presses, or mouse movements.

**How It Works Here:**

* JButton is a Swing component that can trigger events.
* ActionListener is an interface that listens for action events (like a button click).
* addActionListener() is used to register a listener.
* When the button is clicked, the actionPerformed() method is called, and a message box appears.

**Code:**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class EventHandlingExample extends JFrame {

public EventHandlingExample() {

System.out.println("Name: Parshv\n35221402023");

setTitle("Event Handling Example");

setLayout(new FlowLayout());

JButton clickButton = new JButton("Click Me");

clickButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

JOptionPane.showMessageDialog(null, "Button clicked!");

}

});

add(clickButton);

setSize(250, 150);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) {

new EventHandlingExample(); // Launch the GUI

}

}