

Name : Sahil Vikas Khude

Batch : B Roll No : 31

Subject In-charge: Prof. Tayyabali Sayyad

Department of Information Technology

**Index**

1. Write a java program to demonstrate to passed by value and pass by reference
2. Write a java program to demonstrate to demonstrate static variables, methods, and blocks
3. Write a java program to demonstrate to demonstrate inner class and outer classes
4. Write a java program to demonstrate accessing private members in the sub class using public methods
5. Write a java program to demonstrate all usage of super keyword
6. Write a java program to demonstrate dynamic method dispatch in the inheritance
7. Write a java program to demonstrate shared constants in interfaces
8. Write a java program to demonstrate default, public, private and protected scope in packages.
9. Write a java program print even and odd numbers using multithreading
10. Write a java program to demonstrate how to read from a text file using FileReader and write to a text file using FileWriter.
11. Write a java program to demonstrate reading a file using FileInputStream and writing to another file using FileOutputStream.

**Program No 1**

**Program :-** Write a java program to demonstrate to passed by value and pass by reference

**Code :-**

public class PassDemo {

    // Method to demonstrate pass-by-value with primitive

    public static void modify(int num) {

        num = 50;  // This change won't affect the original variable

    }

    // Method to demonstrate pass-by-reference-like behavior with an object

    public static void modify(MyObject obj) {

        obj.value = 50;  // Modifies the object's internal value

    }

    public static void main(String[] args) {

        int number = 10;

        modify(number);

        System.out.println("After modifyPrimitive, number: " + number); // Output: 10

        MyObject myObj = new MyObject();

        myObj.value = 10;

        modify(myObj);

        System.out.println("After modifyObject, myObj.value: " + myObj.value); // Output: 50

    }

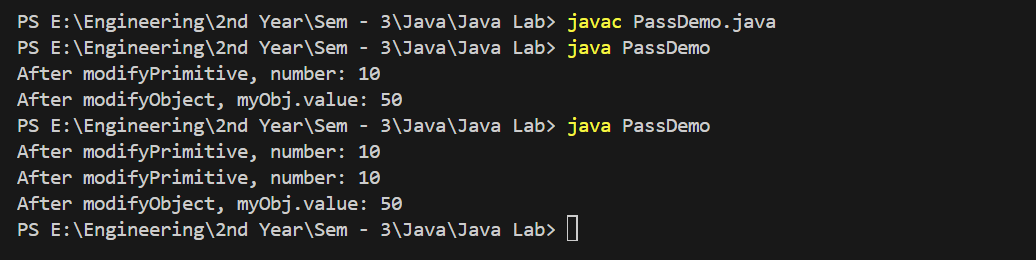
}

class MyObject {

    int value;

}

**Output :**



**Program No : 2**

**Program :** Write a java program to demonstrate to demonstrate static variables, methods, and blocks

**Code:**

class StaticDemo {

// Static variable

static int count = 0;

// Instance variable

int instanceVariable;

// Static block

static {

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

count = 10; // Initializing static variable

}

// Constructor

public StaticDemo() {

instanceVariable = ++count; // Increment static count for each instance

System.out.println("Constructor called, instanceVariable: " + instanceVariable);

}

// Static method

public static void displayCount() {

System.out.println("Current count (static variable): " + count);

}

// Non-static method

public void showInstanceVariable() {

System.out.println("Instance variable for this object: " + instanceVariable);

}

public static void main(String[] args) {

System.out.println("Main method starts.");

// Accessing static method without creating an object

StaticDemo.displayCount();

// Creating instances

StaticDemo obj1 = new StaticDemo();

StaticDemo obj2 = new StaticDemo();

// Accessing static and non-static methods

obj1.showInstanceVariable();

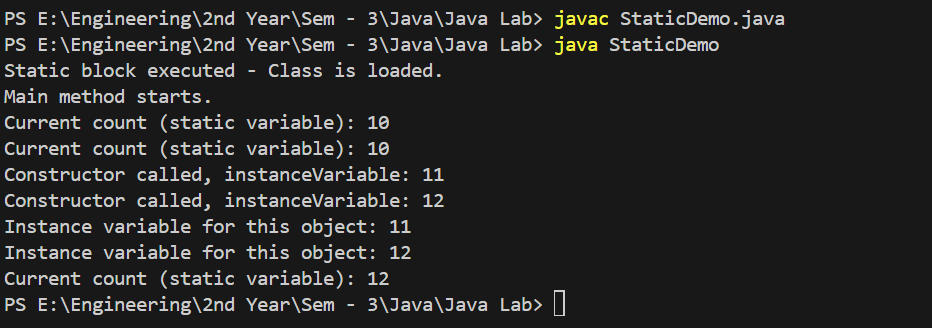
obj2.showInstanceVariable();

StaticDemo.displayCount();

}

}

**Output:**



**Program No 3**

**Program :** Write a java program to demonstrate to demonstrate inner class and outer classes

**Code :**

// Outer class

class OuterClass {

private String outerData = "Outer class data";

// Non-static inner class

class InnerClass {

void display() {

// Accessing outer class private member

System.out.println("Accessing from InnerClass: " + outerData);

}

}

// Static nested class

static class StaticNestedClass {

void show() {

// Directly accessing static members of outer class (if any)

System.out.println("Static Nested Class Method Called");

}

}

// Method demonstrating local inner class

public void localInnerClassDemo() {

class LocalInnerClass {

void print() {

System.out.println("Inside Local Inner Class");

}

}

LocalInnerClass localInner = new LocalInnerClass();

localInner.print();

}

// Method demonstrating anonymous inner class

public void anonymousInnerClassDemo() {

Runnable runnable = new Runnable() {

public void run() {

System.out.println("Anonymous Inner Class Runnable");

}

};

runnable.run();

}

}

public class Main {

public static void main(String[] args) {

// Creating an instance of OuterClass

OuterClass outer = new OuterClass();

// Creating an instance of InnerClass using OuterClass instance

OuterClass.InnerClass inner = outer.new InnerClass();

inner.display(); // Calling method of inner class

// Creating an instance of StaticNestedClass directly

OuterClass.StaticNestedClass staticNested = new OuterClass.StaticNestedClass();

staticNested.show(); // Calling method of static nested class

// Calling method with Local Inner Class

outer.localInnerClassDemo();

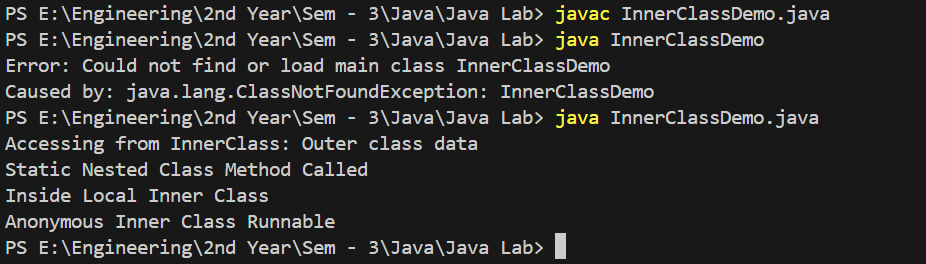
// Calling method with Anonymous Inner Class

outer.anonymousInnerClassDemo();

}

}

**Output :**

****

**Program No 4**

**Program :** Write a java program to demonstrate accessing private members in the sub class using public methods

**Code**:

// Superclass

class Person {

private String name; // Private member

// Constructor

public Person(String name) {

this.name = name;

}

// Public getter for private member

public String getName() {

return name;

}

// Public setter for private member

public void setName(String name) {

this.name = name;

}

}

// Subclass

class Student extends Person {

private int studentId;

// Constructor

public Student(String name, int studentId) {

super(name); // Call superclass constructor

this.studentId = studentId;

}

// Method in subclass accessing private member indirectly

public void displayInfo() {

// Access private 'name' field from superclass using the public getter

System.out.println("Student Name: " + getName());

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

}

}

public class Main {

public static void main(String[] args) {

// Create Student object

Student student = new Student("Alice", 101);

// Display student information

student.displayInfo();

// Modify private 'name' field in superclass using the public setter

student.setName("Bob");

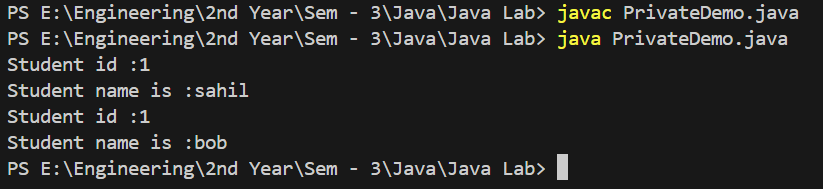
// Display updated student information

student.displayInfo();

}

}

**Output:**

****

**Program No 5**

**Program :** Write a java program to demonstrate all usage of super keyword

**Code:**

// Superclass

class Animal {

String name = "Animal";

// Superclass constructor

public Animal() {

System.out.println("Animal constructor called");

}

// Parameterized constructor

public Animal(String name) {

this.name = name;

System.out.println("Animal parameterized constructor called with name: " + name);

}

// Superclass method

public void makeSound() {

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

}

}

// Subclass

class Dog extends Animal {

String name = "Dog"; // Subclass variable with same name as superclass

// Subclass constructor

public Dog() {

// Use of super() to call superclass constructor

super();

System.out.println("Dog constructor called");

}

// Subclass constructor with parameter

public Dog(String name) {

// Use of super(name) to call superclass parameterized constructor

super(name);

this.name = name;

System.out.println("Dog parameterized constructor called with name: " + name);

}

// Subclass method overriding the superclass method

@Override

public void makeSound() {

super.makeSound(); // Calling superclass method using super

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

}

// Method to show variable access using super

public void displayNames() {

System.out.println("Name in superclass: " + super.name); // Accessing superclass variable

System.out.println("Name in subclass: " + this.name); // Accessing subclass variable

}

}

public class Main {

public static void main(String[] args) {

System.out.println("Creating Dog with no-arg constructor:");

Dog dog1 = new Dog(); // Calls superclass and subclass constructors

System.out.println("\nCreating Dog with parameterized constructor:");

Dog dog2 = new Dog("Buddy"); // Calls parameterized constructor of both superclass and subclass

System.out.println("\nDemonstrating method overriding:");

dog2.makeSound(); // Calls overridden method

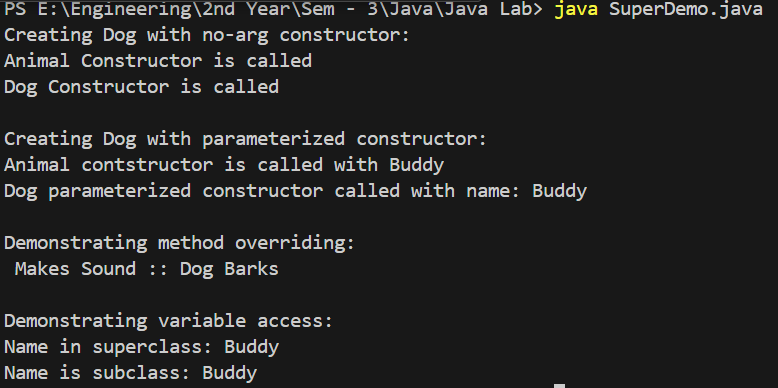
System.out.println("\nDemonstrating variable access:");

dog2.displayNames(); // Shows use of super to access superclass variable

}

}

**Output:**

****

**Program No 6**

**Program :** Write a java program to demonstrate dynamic method dispatch in the inheritance

**Code:**

// Superclass

class Animal {

// Method that will be overridden in the subclass

public void sound() {

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

}

}

// Subclass Dog extending Animal

class Dog extends Animal {

// Overriding the sound method

@Override

public void sound() {

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

}

}

// Subclass Cat extending Animal

class Cat extends Animal {

// Overriding the sound method

@Override

public void sound() {

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

}

}

public class Main {

public static void main(String[] args) {

// Creating superclass reference but subclass object

Animal animal;

// Assigning Dog object to animal reference

animal = new Dog();

animal.sound(); // Calls Dog's sound() method (Dynamic method dispatch)

// Assigning Cat object to animal reference

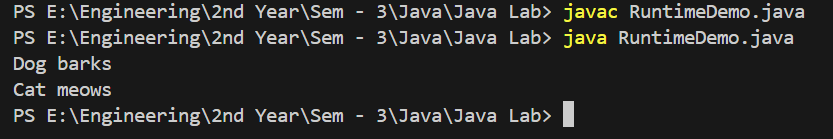
animal = new Cat();

animal.sound(); // Calls Cat's sound() method (Dynamic method dispatch)

}

}

**Output:**

****

**Program No 7**

**Program :** Write a java program to demonstrate shared constants in interfaces

**Code:**

// Interface with shared constants

interface Constants {

// Defining constants

int MAX\_SPEED = 120;

int MIN\_SPEED = 0;

String VEHICLE\_TYPE = "Car";

}

// Class implementing the Constants interface

class Car implements Constants {

private int speed;

// Constructor to initialize speed

public Car(int speed) {

if (speed > MAX\_SPEED) {

this.speed = MAX\_SPEED; // Using MAX\_SPEED from the interface

} else if (speed < MIN\_SPEED) {

this.speed = MIN\_SPEED; // Using MIN\_SPEED from the interface

} else {

this.speed = speed;

}

}

// Display information about the car

public void displayInfo() {

System.out.println("Vehicle Type: " + VEHICLE\_TYPE); // Accessing VEHICLE\_TYPE

System.out.println("Speed: " + speed + " km/h");

}

}

// Main class to run the program

public class Main {

public static void main(String[] args) {

Car car1 = new Car(150); // Speed exceeds MAX\_SPEED, should be capped

Car car2 = new Car(50); // Within the allowed range

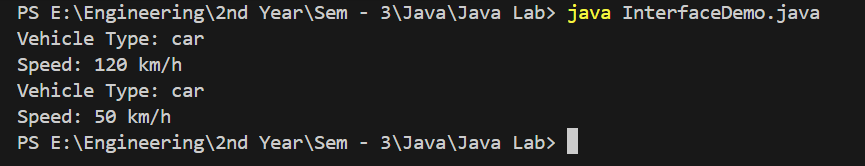
car1.displayInfo();

car2.displayInfo();

}

}

**Output:**

****

**Program No 8**

**Program :** Write a java program to demonstrate default, public, private and protected scope in packages

**Code:**

// Package declaration

package packageA;

// Parent class with different access levels

public class ParentClass {

    public String publicVar = "Public Variable";

    protected String protectedVar = "Protected Variable";

    String defaultVar = "Default Variable"; // default access

    private String privateVar = "Private Variable";

    public void display() {

        System.out.println("Inside ParentClass (packageA):");

        System.out.println("Public Variable: " + publicVar);

        System.out.println("Protected Variable: " + protectedVar);

        System.out.println("Default Variable: " + defaultVar);

        System.out.println("Private Variable: " + privateVar);

    }

}

// Package declaration

package packageB;

// Importing ParentClass from packageA

import packageA.ParentClass;

public class ChildClass extends ParentClass {

    public void displayAccess() {

        System.out.println("Inside ChildClass (packageB):");

        // Accessing variables from ParentClass

        System.out.println("Public Variable: " + publicVar);         // Accessible

        System.out.println("Protected Variable: " + protectedVar);   // Accessible because ChildClass extends ParentClass

        // Default and private variables are not accessible in different packages

        // System.out.println("Default Variable: " + defaultVar);    // Error: Not accessible

        // System.out.println("Private Variable: " + privateVar);    // Error: Not accessible

    }

    public static void main(String[] args) {

        ParentClass parent = new ParentClass();

        parent.display();

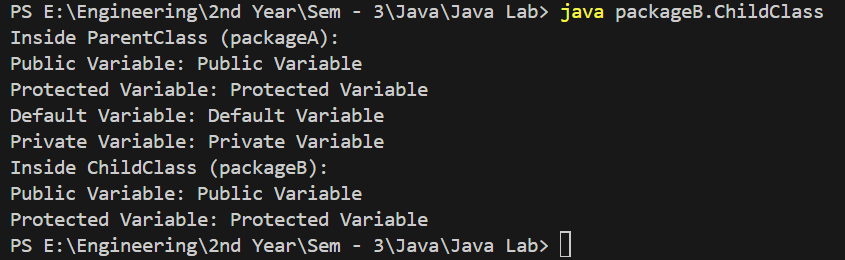
        ChildClass child = new ChildClass();

        child.displayAccess();

    }

}

**Output:**

****

**Program No 9**

**Program :** Write a java program print even and odd numbers using multithreading

**Code:**

class even implements Runnable{

    public void run(){

        for (int i = 0; i < 10; i++) {

            if (i%2 == 0) {

                System.out.println("Even : " + i);

            }

            try {

                Thread.sleep(1000);

            }catch(Exception e){

                return;

            }

        }

    }

}

class odd implements Runnable{

    public void run(){

        for (int i = 0; i < 10; i++) {

            if (i%2 == 1) {

                System.out.println("Odd " + i);

            } try {

                Thread.sleep(1000);

            } catch (Exception exception){

                return;

            }

        }

    }

}

public class EvenOddMultiThreading {

    public static void main(String[] args) {

        even e = new even();

        odd o = new odd();

        Thread t1 = new Thread(e);

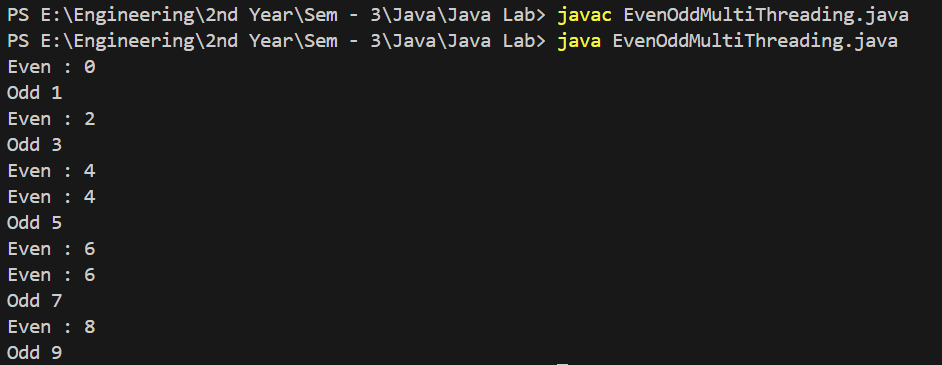
        Thread t2 = new Thread(o);

        t1.start();

        t2.start();

    } }

**Output:**

****

**Program No 10**

**Program :** Write a java program to demonstrate how to read from a text file using FileReader and write to a text file using FileWriter.

**Code:**

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class FileReadWriteExample {

public static void main(String[] args) {

// File paths for input and output

String inputFile = "input.txt";

String outputFile = "output.txt";

// Initialize FileReader and FileWriter in try-with-resources to ensure closure

try (FileReader reader = new FileReader(inputFile); FileWriter writer = new FileWriter(outputFile)) {

int character;

// Read each character from input file and write it to output file

while ((character = reader.read()) != -1) {

writer.write(character);

}

System.out.println("File has been successfully copied to output.txt");

} catch (IOException e) {

System.out.println("An error occurred during file reading/writing.");

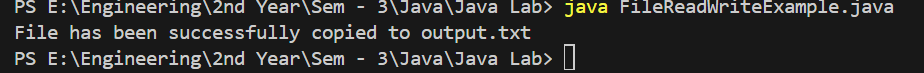
e.printStackTrace();

}

}

}

**Output:**

****

**Input.txt =** hello there from input txt

**Output.txt =** hello there from input txt

**Program No 11**

**Program :** Write a java program to demonstrate reading a file using FileInputStream and writing to another file using FileOutputStream**.**

**Code:**

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class FileStreamExample {

    public static void main(String[] args) {

        String inputFile = "input.txt";     // Specify input file path

        String outputFile = "output.txt";   // Specify output file path

        try (FileInputStream fis = new FileInputStream(inputFile);  // Initialize FileInputStream

             FileOutputStream fos = new FileOutputStream(outputFile)) { // Initialize FileOutputStream

            int byteData;

            // Read each byte from inputFile and write it to outputFile

            while ((byteData = fis.read()) != -1) {

                fos.write(byteData); // Write byte to output file

            }

            System.out.println("File has been successfully copied to outputFile.txt");

        } catch (IOException e) {

            System.out.println("An error occurred during file reading/writing.");

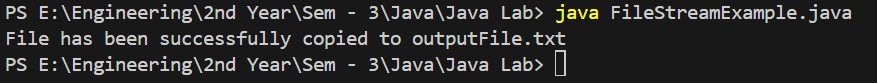
            e.printStackTrace();

        }

    }

}

**Output:**

****