NAME :TALASANIYA NAVDIP R

ROLL NO :35

CLASS:SY D

Lab Assignment -3

Subject : Programming with Java

---

1. Write a program to demonstrate Universal Class.

```
class UniversalClass<T> {
    private T value;

    public UniversalClass(T value) {
        this.value = value;
    }

    public T getValue() {
        return value;
    }

    public void display() {
        System.out.println("Value: " + value + " (Type: " +
value.getClass().getName() + ")");
    }
}

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

        UniversalClass<Integer> integerObject = new UniversalClass<>(10);
        integerObject.display(); // Output: Value: 10 (Type:
java.lang.Integer)

        UniversalClass<String> stringObject = new UniversalClass<>("Hello,
Java!");
        stringObject.display(); // Output: Value: Hello, Java! (Type:
java.lang.String)

        UniversalClass<Double> doubleObject = new UniversalClass<>(3.14);
        doubleObject.display(); // Output: Value: 3.14 (Type:
java.lang.Double)
    }
}

/*
OUTPUT
Value: 10 (Type: java.lang.Integer)
Value: Hello, Java! (Type: java.lang.String)
```

```
Value: 3.14 (Type: java.lang.Double)
*/
```

2. Write a code to demonstrate public, private, protected and default access specifiers.

```
class AccessSpecifiers {

    public int publicVar = 10;
    private int privateVar = 20;
    protected int protectedVar = 30;
    int defaultVar = 40;

    public void displayPublic() {
        System.out.println("Public Variable: " + publicVar);
    }

    private void displayPrivate() {
        System.out.println("Private Variable: " + privateVar);
    }


    protected void displayProtected() {
        System.out.println("Protected Variable: " + protectedVar);
    }

    void displayDefault() {
        System.out.println("Default Variable: " + defaultVar);
    }
}

public class Main {
    public static void main(String[] args) {
        AccessSpecifiers obj = new AccessSpecifiers();

        obj.displayPublic();  // This will work


        // obj.displayPrivate(); // Uncommenting this line will give an error
because it's private

        obj.displayProtected();  // This will work as it's in the same package

        obj.displayDefault();  // This will work because both are in the same
package

        // Accessing private variable directly - This will give an error
        // System.out.println(obj.privateVar);  // Uncommenting this line will
give an error
```

```
    }
}

/*
OUTPUT
Public Variable: 10
Protected Variable: 30
Default Variable: 40
*/
```

3. Write a program to define a Vehicle class with a startEngine() method, then extend it with Car and Truck classes, overriding the startEngine() method in each subclass to provide specific behavior.

```
class Vehicle {

    public void startEngine() {
        System.out.println("Vehicle engine is starting...");
    }
}


class Car extends Vehicle {
    // Overriding the startEngine method for Car class
    @Override
    public void startEngine() {
        System.out.println("Car engine is starting...");
    }
}


class Truck extends Vehicle {
    // Overriding the startEngine method for Truck class
    @Override
    public void startEngine() {
        System.out.println("Truck engine is starting...");
    }
}

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

        Vehicle myCar = new Car();
        Vehicle myTruck = new Truck();
```

```
        myCar.startEngine();  // Car specific behavior
        myTruck.startEngine();  // Truck specific behavior
    }
}
/*
OUTPUT
Car engine is starting...
Truck engine is starting...
*/
```

4. Write a program to demonstrate interface.

```
// Define an interface
interface Vehicle {
    // Abstract method (no body)
    void startEngine();
    void stopEngine();
}

// Implementing the interface in Car class
class Car implements Vehicle {
    // Implementing startEngine method
    @Override
    public void startEngine() {
        System.out.println("Car engine started.");
    }

    // Implementing stopEngine method
    @Override
    public void stopEngine() {
        System.out.println("Car engine stopped.");
    }
}

// Implementing the interface in Truck class
class Truck implements Vehicle {
    // Implementing startEngine method
    @Override
    public void startEngine() {
        System.out.println("Truck engine started.");
    }

    // Implementing stopEngine method
    @Override
    public void stopEngine() {
        System.out.println("Truck engine stopped.");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        // Creating objects of Car and Truck
        Vehicle myCar = new Car();
        Vehicle myTruck = new Truck();

        // Calling methods of Vehicle interface through Car and Truck objects
        myCar.startEngine();  // Car specific startEngine behavior
        myCar.stopEngine();   // Car specific stopEngine behavior

        myTruck.startEngine();  // Truck specific startEngine behavior
        myTruck.stopEngine();   // Truck specific stopEngine behavior
    }
}

/*
OUTPUT
Car engine started.
Car engine stopped.
Truck engine started.
Truck engine stopped.
*/
```

5. Write a program to demonstrate Nested Class.

```
// Outer class
class OuterClass {

    // Instance variable of the outer class
    private String outerMessage = "Hello from Outer Class";

    // Inner class (Non-static nested class)
    class InnerClass {
        public void display() {
            // Inner class can access outer class's instance variables and
methods
            System.out.println("Message from Inner Class: " + outerMessage);
        }
    }

    // Static Nested class
    static class StaticNestedClass {
        public void display() {
            // Static nested class cannot access instance variables directly
            System.out.println("Message from Static Nested Class");
        }
    }
```

```
}

public class Main {
    public static void main(String[] args) {
        // Creating an instance of the outer class
        OuterClass outer = new OuterClass();

        // Creating an instance of the Inner class (non-static nested class)
        OuterClass.InnerClass inner = outer.new InnerClass();
        inner.display();  // Accessing method of inner class

        // Creating an instance of the Static Nested class (static nested
class doesn't require an outer class instance)
        OuterClass.StaticNestedClass staticNested = new
OuterClass.StaticNestedClass();
        staticNested.display();  // Accessing method of static nested class
    }
}

/*
OUTPUT
Message from Inner Class: Hello from Outer Class
Message from Static Nested Class
*/
```

6. Write a program to demonstrate Abstract class.

```
// Abstract class
abstract class Animal {
    // Abstract method (no implementation)
    public abstract void sound();

    // Concrete method (with implementation)
    public void sleep() {
        System.out.println("This animal is sleeping.");
    }
}

// Subclass that extends the abstract class
class Dog extends Animal {
    // Providing implementation for the abstract method sound()
    @Override
    public void sound() {
        System.out.println("The dog barks.");
    }
}

// Subclass that extends the abstract class
class Cat extends Animal {
```

```
    // Providing implementation for the abstract method sound()
    @Override
    public void sound() {
        System.out.println("The cat meows.");
    }
}

public class Main {
    public static void main(String[] args) {
        // Creating objects of Dog and Cat class
        Animal dog = new Dog();
        Animal cat = new Cat();

        // Calling methods
        dog.sound();  // Output: The dog barks.
        dog.sleep();  // Output: This animal is sleeping.

        cat.sound();  // Output: The cat meows.
        cat.sleep();  // Output: This animal is sleeping.
    }
}

/*
OUTPUT
The dog barks.
This animal is sleeping.
The cat meows.
This animal is sleeping.
*/
```

7. Write a program to define Final Class.

```
// Final class declaration
final class FinalClass {
    // Method inside the final class
    public void displayMessage() {
        System.out.println("This is a final class. It cannot be inherited.");
    }
}

// This will cause an error if uncommented, because a final class cannot be
extended
// class ChildClass extends FinalClass {
//     // Compilation error: Cannot inherit from final class 'FinalClass'
// }

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

```
        // Creating an object of the final class
        FinalClass obj = new FinalClass();
        obj.displayMessage();
    }
}

/*
OUTPUT
This is a final class. It cannot be inherited.
*/
```