**JAVA LAB SHEET 05**

**Exercise 01:**

**Declare an interface called “MyFirstInterface”. Decalre integer type variable called “x”. Declare an abstract method called “display()”.**

1. **Try to declare the variable with/without public static final keywords. Is there any difference between these two approaches? Why?**
2. **Declare the abstract method with/without abstract keyword. Is there any difference between these two approaches? Why?**
3. **Implement this into a class called “IntefaceImplemented” . Override all the abstract methods. Try to change the value of x inside this method and print the value of x. Is it possible for you to change x? why?**

public interface MyFirstInterface {

int x = 10; // Variable is implicitly public, static, and final.

void display(); // Method is implicitly abstract.

}

// Implicitly abstract method in the interface

void display();

// Explicitly declared abstract method in the interface (redundant but valid)

abstract void display();

public class InterfaceImplemented implements MyFirstInterface {

// Implementing the abstract method from the interface

@Override

public void display() {

// Try to change the value of x inside this method

x = 20; // This will result in a compile-time error

// Print the value of x

System.out.println("Value of x inside display(): " + x);

}

public static void main(String[] args) {

InterfaceImplemented obj = new InterfaceImplemented();

obj.display();

}

}

error: cannot assign a value to final variable x

**Exercise 02:**

**Develop a code base for the following scenario. Recall what we have done at the lecture…**

Create a class called BankAccount:

public class BankAccount {

private int accountNumber;

private String accountHolderName;

private double balance;

public BankAccount(int accountNumber, String accountHolderName, double balance) {

this.accountNumber = accountNumber;

this.accountHolderName = accountHolderName;

this.balance = balance;

}

public int getAccountNumber() {

return accountNumber;

}

public String getAccountHolderName() {

return accountHolderName;

}

public double getBalance() {

return balance;

}

public void deposit(double amount) {

balance += amount;

}

public void withdraw(double amount) {

if (amount <= balance) {

balance -= amount;

} else {

System.out.println("Insufficient balance.");

}

}

}

Create a test class to demonstrate the banking application:

public class TestBankAccount {

public static void main(String[] args) {

// Create a BankAccount object for the customer "John Doe"

BankAccount account1 = new BankAccount(123456789, "John Doe", 1000.0);

// Deposit and withdraw money

account1.deposit(500.0);

account1.withdraw(200.0);

// Get account information

int accountNumber1 = account1.getAccountNumber();

String accountHolderName1 = account1.getAccountHolderName();

double balance1 = account1.getBalance();

// Print account information

System.out.println("Account Number: " + accountNumber1);

System.out.println("Account Holder Name: " + accountHolderName1);

System.out.println("Balance: " + balance1);

}

}

Output:

Account Number: 123456789

Account Holder Name: John Doe

Balance: 1300.0

**Exercise 03:**

**Try following code. What is the outcome? Why?**

**Class 01: Class 02:**

**final class Student { class Undergraduate extends Student{}**

**final int marks = 100;**

**final void display();**

**}**

error: cannot inherit from final Student

abstract class Student {

final int marks = 100;

abstract void display();

}

class Undergraduate extends Student {

@Override

void display() {

System.out.println("Undergraduate student's display method.");

}

}

**Exercise 04:**

**Develop a code base for the following scenario. Shape class contains an abstract method called “calculateArea” and non-abstract method called “display”. Try to pass required values at the instantiation. Recall what we have done at the lecture…**

abstract class Shape {

abstract double calculateArea();

void display() {

System.out.println("This is a shape.");

}

}

class Rectangle extends Shape {

private double length;

private double width;

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

@Override

double calculateArea() {

return length \* width;

}

@Override

void display() {

System.out.println("This is a rectangle.");

System.out.println("Length: " + length);

System.out.println("Width: " + width);

System.out.println("Area: " + calculateArea());

}

}

class Circle extends Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

@Override

double calculateArea() {

return Math.PI \* radius \* radius;

}

@Override

void display() {

System.out.println("This is a circle.");

System.out.println("Radius: " + radius);

System.out.println("Area: " + calculateArea());

}

}

public class TestShape {

public static void main(String[] args) {

Rectangle rectangle = new Rectangle(5.0, 3.0);

rectangle.display();

System.out.println();

Circle circle = new Circle(4.0);

circle.display();

}

}

Output-

This is a rectangle.

Length: 5.0

Width: 3.0

Area: 15.0

This is a circle.

Radius: 4.0

Area: 50.26548245743669