**ASSIGNMENT 4**

**SET A**

**Q.1 Define an Interface Shape with abstract method area(). Write a java program to calculate an area of Circle and Sphere. (use final keyword)**

interface Shape {

double area();

}

class Circle implements Shape {

final double radius;

Circle(double radius) {

this.radius = radius;

}

public double area() {

return Math.PI \* radius \* radius;

}

}

class Sphere implements Shape {

final double radius;

public Sphere (double radius) {

this.radius = radius;

}

public double area() {

return 4 \* Math.PI \* radius \* radius;

}

}

public class AreaSphere {

public static void main(String[] args) {

Shape circle = new Circle(5);

Shape sphere = new Sphere(5);

System.out.println("Area of Circle: " + circle.area());

System.out.println("Surface Area of Sphere: " + sphere.area());

}

}

Output:

Area of Circle: 78.53981633974483

Surface Area of Sphere: 314.1592653589793

=== Code Execution Successful ===

Q.2 Define an Interface “Integer” with a abstract method check().Write a Java program to check whether a given number is Positive or Negative.

interface NumberChecker

{

void check(int number);

}

class CheckNumber implements NumberChecker

{

public void check(int number)

{

if (number > 0)

{

System.out.println(number + " is positive.");

}

else if (number < 0)

{

System.out.println(number + " is negative.");

} else

{

System.out.println(number + " is zero.");

}

}

}

public class CheckNo

{

public static void main(String[] args)

{

CheckNumber ch = new CheckNumber();

ch.check(10);

ch.check(-5);

ch.check(0);

}

}

Output:

10 is positive.

-5 is negative.

0 is zero.

=== Code Execution Successful ===

Q.3 Write a Java program to create an interface Flyable with a method called fly\_obj(). Create three classes Spacecraft, Airplane, and Helicopter that implement the Flyable interface. Implement the fly\_obj() method for each of the three classes.

interface Flyable

{

void fly\_obj();

}

class Spacecraft implements Flyable

{

public void fly\_obj()

{

System.out.println("Spacecraft is flying through space.");

}

}

class Airplane implements Flyable

{

public void fly\_obj()

{

System.out.println("Airplane is flying in the sky.");

}

}

class Helicopter implements Flyable

{

public void fly\_obj()

{

System.out.println("Helicopter is hovering and flying at low altitudes.");

}

}

public class Fly

{

public static void main(String[] args)

{

Flyable spacecraft = new Spacecraft();

Flyable airplane = new Airplane();

Flyable helicopter = new Helicopter();

spacecraft.fly\_obj();

airplane.fly\_obj();

helicopter.fly\_obj();

}

}

Output:

Spacecraft is flying through space.

Airplane is flying in the sky.

Helicopter is hovering and flying at low altitudes.

=== Code Execution Successful ===

SET C:

1. Create an interface ― CreditCardInterface with method: viewCreditAmount(), useCard(),payCredit() and increaseLimit(). Create a class SilverCardCustomer (name,cardnumber (16digits), creditAmount – initialized to 0, creditLimit - set to 50,000 ) which implements the above interface. Inherit class GoldCardCustomer from SilverCardCustomer having the same methods but creditLimit of 1,00,000. Create an object of each class and perform operations. Display appropriate messages for success or failure of transactions. (Use method overriding)

i. useCard() method increases the creditAmount by a specific amount upto creditLimit

ii. payCredit() reduces the creditAmount by a specific amount.

iii. increaseLimit() increases the creditLimit for GoldCardCustomers (only 3 times, not more than 5000 Rs. each time)

interface CreditCardInterface

{

void viewCreditAmount();

void useCard(double amount);

void payCredit(double amount);

void increaseLimit(double amount);

}

class SilverCardCustomer implements CreditCardInterface

{

String name;

String cardNumber;

double creditAmount;

double creditLimit;

public SilverCardCustomer(String name, String cardNumber)

{

this.name = name;

this.cardNumber = cardNumber;

this.creditAmount = 0;

this.creditLimit = 50000;

}

public void viewCreditAmount()

{

System.out.println("Current Credit Amount: " + creditAmount);

}

public void useCard(double amount)

{

if (creditAmount + amount > creditLimit)

{

System.out.println("Transaction failed: Exceeds credit limit.");

}

else

{

creditAmount += amount;

System.out.println("Transaction successful: Used " + amount);

}

}

public void payCredit(double amount)

{

if (amount > creditAmount)

{

System.out.println("Payment exceeds credit amount.");

}

Else

{

creditAmount-= amount;

System.out.println("Payment successful: Paid " + amount);

}

}

public void increaseLimit(double amount)

{

System.out.println("Cannot increase limit for SilverCardCustomer.");

}

}

class GoldCardCustomer extends SilverCardCustomer

{

int increaseCount = 0;

public GoldCardCustomer(String name, String cardNumber)

{

super(name, cardNumber);

this.creditLimit = 100000;

}

public void increaseLimit(double amount)

{

if (increaseCount < 3 && amount <= 5000)

{

creditLimit += amount;

increaseCount++;

System.out.println("Credit limit increased by " + amount + ". New limit: " + creditLimit);

}

else if (increaseCount >= 3)

{

System.out.println("Limit increase failed: Limit can only be increased 3 times.");

}

else

{

System.out.println("Limit increase failed: Cannot increase more than 5000 at once.");

}

}

}

public class Card

{

public static void main(String[] args) {

SilverCardCustomer sc = new SilverCardCustomer("Rohan", "1234567812345678");

sc.viewCreditAmount();

sc.useCard(20000);

sc.viewCreditAmount();

sc.payCredit(5000);

sc.viewCreditAmount();

sc.increaseLimit(3000);

System.out.println();

GoldCardCustomer gc = new GoldCardCustomer("Arpita", "8765432187654321");

gc.viewCreditAmount();

gc.useCard(60000);

gc.viewCreditAmount();

gc.payCredit(10000);

gc.viewCreditAmount();

gc.increaseLimit(3000);

gc.increaseLimit(5000);

gc.increaseLimit(2000);

}

}

Output:

Current Credit Amount: 0.0

Transaction successful: Used 20000.0

Current Credit Amount: 20000.0

Payment successful: Paid 5000.0

Current Credit Amount: 15000.0

Cannot increase limit for SilverCardCustomer.

Current Credit Amount: 0.0

Transaction successful: Used 60000.0

Current Credit Amount: 60000.0

Payment successful: Paid 10000.0

Current Credit Amount: 50000.0

Credit limit increased by 3000.0. New limit: 103000.0

Credit limit increased by 5000.0. New limit: 108000.0

Credit limit increased by 2000.0. New limit: 110000.0

=== Code Execution Successful ===

Q.2 Write a Java program to create an interface Resizable with methods resizeWidth(int width) and resizeHeight(int height) that allow an object to be resized. Create a class Rectangle that implements the Resizable interface and implements the resize methods.

interface Resizable

{

void resizeWidth(int width);

void resizeHeight(int height);

}

class Rectangle implements Resizable

{

int width;

int height;

Rectangle(int width, int height)

{

this.width = width;

this.height = height;

}

public void resizeWidth(int width)

{

this.width = width;

System.out.println("Rectangle width resized to: " + this.width);

}

public void resizeHeight(int height)

{

this.height = height;

System.out.println("Rectangle height resized to: " + this.height);

}

public void displayDimensions()

{

System.out.println("Current dimensions - Width: " + width + ", Height: " + height);

}

}

public class ResizeRect

{

public static void main(String[] args)

{

Rectangle r = new Rectangle(10, 5);

r.displayDimensions();

r.resizeWidth(20);

r.resizeHeight(15);

r.displayDimensions();

}

}

Output:

Current dimensions - Width: 10, Height: 5

Rectangle width resized to: 20

Rectangle height resized to: 15

Current dimensions - Width: 20, Height: 15

=== Code Execution Successful ===

Q.3 Write a Java program to create an interface Sortable with a method sort (int[] array) that sorts an array of integers in descending order. Create two classes QuickSort and MergeSort that implement the Sortable interface and provide their own implementations of the sort() method.

interface Sortable

{

void sort(int[] array);

}

class QuickSort implements Sortable

{

public void sort(int[] array)

{

quickSort(array, 0, array.length - 1);

}

private void quickSort(int[] array, int low, int high)

{

if (low < high)

{

int pivotIndex = partition(array, low, high);

quickSort(array, low, pivotIndex - 1);

quickSort(array, pivotIndex + 1, high);

}

}

private int partition(int[] array, int low, int high)

{

int pivot = array[high];

int i = low - 1;

for (int j = low; j < high; j++)

{

if (array[j] > pivot)

{

i++;

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

int temp = array[i + 1];

array[i + 1] = array[high];

array[high] = temp;

return i + 1;

}

}

class MergeSort implements Sortable

{

public void sort(int[] array)

{

mergeSort(array, 0, array.length - 1);

}

private void mergeSort(int[] array, int left, int right)

{

if (left < right)

{

int mid = (left + right) / 2;

mergeSort(array, left, mid);

mergeSort(array, mid + 1, right);

merge(array, left, mid, right);

}

}

private void merge(int[] array, int left, int mid, int right)

{

int n1 = mid - left + 1;

int n2 = right - mid;

int[] leftArray = new int[n1];

int[] rightArray = new int[n2];

System.arraycopy(array, left, leftArray, 0, n1);

System.arraycopy(array, mid + 1, rightArray, 0, n2);

int i = 0, j = 0, k = left;

while (i < n1 && j < n2)

{

if (leftArray[i] > rightArray[j])

{

array[k] = leftArray[i];

i++;

} else

{

array[k] = rightArray[j];

j++;

}

k++;

}

while (i < n1)

{

array[k] = leftArray[i];

i++;

k++;

}

while (j < n2)

{

array[k] = rightArray[j];

j++;

k++;

}

}

}

public class Main

{

public static void main(String[] args)

{

int[] array = {12, 4, 5, 7, 19, 10, 15};

Sortable quickSort = new QuickSort();

System.out.println("QuickSort:");

quickSort.sort(array);

printArray(array);

array = new int[]{12, 4, 5, 7, 19, 10, 15};

Sortable mergeSort = new MergeSort();

System.out.println("MergeSort:");

mergeSort.sort(array);

printArray(array);

}

public static void printArray(int[] array)

{

for (int num : array)

{

System.out.print(num + " ");

}

System.out.println();

}

}

Output:

QuickSort:

19 15 12 10 7 5 4

MergeSort:

19 15 12 10 7 5 4

=== Code Execution Successful ===