**ASSIGNMENT-13**

**Question-1**

**Problem Statement:** Create a class Shape having data members length, breadth, height and abstract methods such as volume and surfaceArea. Inherit this class into cube, cylinder and cuboid classes. Redefine the required methods to calculate and display the volume and surface area of each shape.

**Source Code**

//Abstract class shape

public abstract class Shape {

protected double length;

protected double breadth;

protected double height;

public Shape(double length, double breadth, double height) {

this.length = length;

this.breadth = breadth;

this.height = height;

}

// Abstract methods to be implemented by subclasses

public abstract double volume();

public abstract double surfaceArea();

}

//Class Cube

public class Cube extends Shape {

public Cube(double side) {

super(side, side, side);

}

@Override

public double volume() {

return length \* length \* length;

}

@Override

public double surfaceArea() {

return 6 \* length \* length;

}

}

//Class Cuboid

public class Cuboid extends Shape {

public Cuboid(double length, double breadth, double height) {

super(length, breadth, height);

}

@Override

public double volume() {

return length \* breadth \* height;

}

@Override

public double surfaceArea() {

return 2 \* (length \* breadth + breadth \* height + height \* length);

}

}

//Class Cylinder

public class Cylinder extends Shape {

public Cylinder(double radius, double height) {

super(radius, radius, height);

}

@Override

public double volume() {

return Math.PI \* length \* length \* height;

}

@Override

public double surfaceArea() {

return 2 \* Math.PI \* length \* (length + height);

}

}

//Class TestShape(with main method)

public class TestShape {

public static void main(String[] args) {

Cube cube = new Cube(5);

Cylinder cylinder = new Cylinder(3, 7);

Cuboid cuboid = new Cuboid(4, 6, 8);

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

System.out.println("Volume: " + cube.volume());

System.out.println("Surface Area: " + cube.surfaceArea());

System.out.println("\nCylinder:");

System.out.println("Volume: " + cylinder.volume());

System.out.println("Surface Area: " + cylinder.surfaceArea());

System.out.println("\nCuboid:");

System.out.println("Volume: " + cuboid.volume());

System.out.println("Surface Area: " + cuboid.surfaceArea());

}

}

**OUTPUT:**

Cube:

Volume: 125.0

Surface Area: 150.0

Cylinder:

Volume: 197.92033717615698

Surface Area: 188.49555921538757

Cuboid:

Volume: 192.0

Surface Area: 208.0

**Question-2**

**Problem Statement:** Design an abstract class fruit with data members colour, taste and an abstract method display. Inherit this class to other classes such as Apple, Banana, Orange and Strawberry. Redefine the display method to show the color and taste of each fruit along with its class name.

**Source Code**

//Abstract class Fruit

public abstract class Fruit {

protected String color;

protected String taste;

public Fruit(String color, String taste) {

this.color = color;

this.taste = taste;

}

// Abstract method to be implemented by subclasses

public abstract void display();

}

//Class Apple

public class Apple extends Fruit {

public Apple(String color, String taste) {

super(color, taste);

}

@Override

public void display() {

System.out.println("Fruit: Apple");

System.out.println("Color: " + color);

System.out.println("Taste: " + taste);

}

}

//Class Banana

public class Banana extends Fruit {

public Banana(String color, String taste) {

super(color, taste);

}

@Override

public void display() {

System.out.println("Fruit: Banana");

System.out.println("Color: " + color);

System.out.println("Taste: " + taste);

}

}

//Class Orange

public class Orange extends Fruit {

public Orange(String color, String taste) {

super(color, taste);

}

@Override

public void display() {

System.out.println("Fruit: Orange");

System.out.println("Color: " + color);

System.out.println("Taste: " + taste);

}

}

//Class Strawberry

public class Strawberry extends Fruit {

public Strawberry(String color, String taste) {

super(color, taste);

}

@Override

public void display() {

System.out.println("Fruit: Strawberry");

System.out.println("Color: " + color);

System.out.println("Taste: " + taste);

}

}

//Class TestFruit (with main method)

public class TestFruit {

public static void main(String[] args) {

Apple apple = new Apple("Red", "Sweet");

Banana banana = new Banana("Yellow", "Sweet");

Orange orange = new Orange("Orange", "Sweet and Tangy");

Strawberry strawberry = new Strawberry("Red", "Sweet and Tart");

apple.display();

banana.display();

orange.display();

strawberry.display();

}

}

**OUTPUT:**

Fruit: Apple

Color: Red

Taste: Sweet

Fruit: Banana

Color: Yellow

Taste: Sweet

Fruit: Orange

Color: Orange

Taste: Sweet and Tangy

Fruit: Strawberry

Color: Red

Taste: Sweet and Tart

**Question-3**

**Problem Statement:** Define an interface “IntOperations” with methods to check whether a number is positive/ negative, even/odd, prime, palindrome and operations like factorial and sum of digits. Define a class MyNumber having one private data member of type int. Write a default constructor to initialize it to 0 and another constructor to initialize it to a value (Use this). Implement the above interface. Create an object in main method. Input a number and write a menu driven program to check different properties of the number using above methods.

**Source Code**

//Interface IntOperations

public interface IntOperations {

boolean isPositive(int num);

boolean isNegative(int num);

boolean isEven(int num);

boolean isOdd(int num);

boolean isPrime(int num);

boolean isPalindrome(int num);

int factorial(int num);

int sumOfDigits(int num);

}

//Class MyNumber

class MyNumber implements IntOperations {

private int number;

// Default constructor

public MyNumber() {

this.number = 0;

}

// Constructor with parameter

public MyNumber(int number) {

this.number = number;

}

@Override

public boolean isPositive(int num) {

return num > 0;

}

@Override

public boolean isNegative(int num) {

return num < 0;

}

@Override

public boolean isEven(int num) {

return num % 2 == 0;

}

@Override

public boolean isOdd(int num) {

return num % 2 != 0;

}

@Override

public boolean isPrime(int num) {

if (num <= 1) return false;

for (int i = 2; i <= Math.sqrt(num); i++) {

if (num % i == 0) return false;

}

return true;

}

@Override

public boolean isPalindrome(int num) {

int reverse = 0;

int temp = num;

while (temp != 0) {

int remainder = temp % 10;

reverse = reverse \* 10 + remainder;

temp /= 10;

}

return num == reverse;

}

@Override

public int factorial(int num) {

if (num == 0 || num == 1)

return 1;

else

return num \* factorial(num - 1);

}

@Override

public int sumOfDigits(int num) {

int sum = 0;

while (num != 0) {

sum += num % 10;

num /= 10;

}

return sum;

}

public int getNumber() {

return number;

}

}

//Class TestInt (with main method)

import java.util.Scanner;

public class TestInt {

public static void main(String[] args) {

// Create object of MyNumber class

Scanner sc=new Scanner(System.in);

// Input a number

System.out.println("Enter a no:");

int number = sc.nextInt();

MyNumber myNumber = new MyNumber(number);

// Menu driven program

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

System.out.println("1. Check if the number is positive");

System.out.println("2. Check if the number is negative");

System.out.println("3. Check if the number is even");

System.out.println("4. Check if the number is odd");

System.out.println("5. Check if the number is prime");

System.out.println("6. Check if the number is palindrome");

System.out.println("7. Calculate factorial of the number");

System.out.println("8. Calculate sum of digits of the number");

// Assuming user's choice for demonstration

int choice = sc.nextInt();

switch (choice) {

case 1:

System.out.println("Is the number positive? " + myNumber.isPositive(number));

break;

case 2:

System.out.println("Is the number negative? " + myNumber.isNegative(number));

break;

case 3:

System.out.println("Is the number even? " + myNumber.isEven(number));

break;

case 4:

System.out.println("Is the number odd? " + myNumber.isOdd(number));

break;

case 5:

System.out.println("Is the number prime? " + myNumber.isPrime(number));

break;

case 6:

System.out.println("Is the number palindrome? " + myNumber.isPalindrome(number));

break;

case 7:

System.out.println("Factorial of the number: " + myNumber.factorial(number));

break;

case 8:

System.out.println("Sum of digits of the number: " + myNumber.sumOfDigits(number));

break;

default:

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

}

}

}

**OUTPUT:**

//case-1

Enter a no:

11

Menu:

1. Check if the number is positive

2. Check if the number is negative

3. Check if the number is even

4. Check if the number is odd

5. Check if the number is prime

6. Check if the number is palindrome

7. Calculate factorial of the number

8. Calculate sum of digits of the number

Enter your choice: 1

Is the number positive? true

//case-2

Enter a no:

1547

Menu:

1. Check if the number is positive

2. Check if the number is negative

3. Check if the number is even

4. Check if the number is odd

5. Check if the number is prime

6. Check if the number is palindrome

7. Calculate factorial of the number

8. Calculate sum of digits of the number

Enter your choice: 8

Sum of digits of the number: 17

**Question-4**

**Problem Statement:** Define an interface “StackOperations” which declares methods for a static stack. Define a class “MyStack” which contains an array and top as data members and implements the above interface. Initialize the stack using a constructor. Write a menu driven program to perform all operations(Push, POP, Peak) on a MyStack object.

**Source Code**

//Interface StackOperations

interface StackOperations {

void push(int element);

int pop();

int peek();

}

//Class MyStack

class MyStack implements StackOperations {

private int[] stackArray;

private int top;

private int maxSize;

public MyStack(int size) {

maxSize = size;

stackArray = new int[maxSize];

top = -1;

}

@Override

public void push(int element) {

if (top == maxSize - 1) {

System.out.println("Stack Overflow! Cannot push element " + element);

} else {

stackArray[++top] = element;

System.out.println("Pushed element: " + element);

}

}

@Override

public int pop() {

if (top == -1) {

System.out.println("Stack Underflow! Cannot pop element.");

return -1;

} else {

int poppedElement = stackArray[top--];

System.out.println("Popped element: " + poppedElement);

return poppedElement;

}

}

@Override

public int peek() {

if (top == -1) {

System.out.println("Stack is empty! Cannot peek.");

return -1;

} else {

int topElement = stackArray[top];

System.out.println("Top element: " + topElement);

return topElement;

}

}

}

//Class TestStack (with main method)

import java.util.Scanner;

public class TestStack {

public static void main(String[] args) {

MyStack stack = new MyStack(5); // Initialize stack with size 5

Scanner sc=new Scanner(System.in);

while(true) {

// Menu driven program

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

System.out.println("1. Push an element into the stack.");

System.out.println("2. Pop an element from the stack.");

System.out.println("3. Peek inside the stack.");

System.out.println("4. Exit.");

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

int choice = sc.nextInt();

switch (choice) {

case 1:

System.out.println("Enter an element to push into the stack:");

int element=sc.nextInt();

stack.push(element);

break;

case 2:

stack.pop();

break;

case 3:

stack.peek();

break;

case 4:

System.exit(0);

default:

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

}

}

}

}

**OUTPUT:**

Menu:

1. Push an element into the stack.

2. Pop an element from the stack.

3. Peek inside the stack.

4. Exit.

Enter your choice:

1

Enter an element to push into the stack:

1

Pushed element: 1

Menu:

1. Push an element into the stack.

2. Pop an element from the stack.

3. Peek inside the stack.

4. Exit.

Enter your choice:

1

Enter an element to push into the stack:

2

Pushed element: 2

Menu:

1. Push an element into the stack.

2. Pop an element from the stack.

3. Peek inside the stack.

4. Exit.

Enter your choice:

1

Enter an element to push into the stack:

5

Pushed element: 5

Menu:

1. Push an element into the stack.

2. Pop an element from the stack.

3. Peek inside the stack.

4. Exit.

Enter your choice:

2

Popped element: 5

Menu:

1. Push an element into the stack.

2. Pop an element from the stack.

3. Peek inside the stack.

4. Exit.

Enter your choice:

3

Top element: 2

Menu:

1. Push an element into the stack.

2. Pop an element from the stack.

3. Peek inside the stack.

4. Exit.

Enter your choice:

4