**Name:-kovuru mahesh naidu**

**regno:-192224085**

1.Write java code for creating 3 methods called even, odd and prime in parent class and override them from child class. The parent class should print 1 to 20 range in all method, but child print 1 to 30 array range in all methods. And access all the methods using derived class object

class Parent {

public void even() {

for (int i = 1; i <= 20; i++) {

if (i % 2 == 0) {

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

}

}

System.out.println();

}

public void odd() {

for (int i = 1; i <= 20; i++) {

if (i % 2 != 0) {

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

}

}

System.out.println();

}

public void prime() {

for (int i = 1; i <= 20; i++) {

boolean isPrime = true;

if (i == 1 || i == 0)

continue;

for (int j = 2; j <= i / 2; j++) {

if (i % j == 0) {

isPrime = false;

break;

}

}

if (isPrime)

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

}

System.out.println();

}

}

class Child extends Parent {

@Override

public void even() {

for (int i = 1; i <= 30; i++) {

if (i % 2 == 0) {

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

}

}

System.out.println();

}

@Override

public void odd() {

for (int i = 1; i <= 30; i++) {

if (i % 2 != 0) {

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

}

}

System.out.println();

}

@Override

public void prime() {

for (int i = 1; i <= 30; i++) {

boolean isPrime = true;

if (i == 1 || i == 0)

continue;

for (int j = 2; j <= i / 2; j++) {

if (i % j == 0) {

isPrime = false;

break;

}

}

if (isPrime)

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

}

System.out.println();

}

}

public class Main {

public static void main(String[] args) {

Child child = new Child();

child.even();

child.odd();

child.prime();

}

}

2. Override a default constructor, which contains a=10,b=15 in java using parametrized constructor which contains a=20,b=25. Create a method called disp(). And call both the constructors associating it with disp()

class Parent {

int a, b;

Parent() {

this(10, 15);

}

Parent(int a, int b) {

this.a = a;

this.b = b;

}

void disp() {

System.out.println("a: " + a + ", b: " + b);

}

}

public class Main {

public static void main(String[] args) {

Parent obj1 = new Parent();

obj1.disp();

Parent obj2 = new Parent(20, 25);

obj2.disp();

}

}

3) Java program to create a Base class with a method called ‘void area(int a, int b)’ which print are a of a square. Now override the method from Derived class and make it print area of a rectangle.

class Base {

void area(int a, int b) {

System.out.println("Area of square: " + (a \* a));

}

}

class Derived extends Base {

@Override

void area(int a, int b) {

System.out.println("Area of rectangle: " + (a \* b));

}

}

public class Main {

public static void main(String[] args) {

Derived obj = new Derived();

obj.area(5, 6); // This will print the area of rectangle

}

}

4) Write java code to overload a method called ‘int sum(int a, int b)’ by the 3 ways. a) By increasing and decreasing no. of parameter

**class MyClass {**

**// Method overloading by changing number of parameters**

**int sum(int a, int b) {**

**return a + b;**

**}**

**int sum(int a, int b, int c) {**

**return a + b + c;**

**}**

**// Method overloading by changing data type of parameters**

**double sum(double a, double b) {**

**return a + b;**

**}**

**}**

**public class Main {**

**public static void main(String[] args) {**

**MyClass obj = new MyClass();**

**System.out.println("Sum of two integers: " + obj.sum(5, 10));**

**System.out.println("Sum of three integers: " + obj.sum(5, 10, 15));**

**System.out.println("Sum of two doubles: " + obj.sum(2.5, 3.5));**

**}**

**}**

5. Create an abstract class with 2 abstract methods(total() and average()) and 3 concrete methods(mean(), mode(), median()). Now extend the abstract class from a concrete class and use all the methods in that abstract class

abstract class MyAbstractClass {

abstract int total();

abstract double average();

void mean() {

System.out.println("This is the mean method.");

}

void mode() {

System.out.println("This is the mode method.");

}

void median() {

System.out.println("This is the median method.");

}

}

public class Main extends MyAbstractClass {

@Override

int total() {

return 100;

}

@Override

double average() {

return 50.0;

}

public static void main(String[] args) {

Main obj = new Main();

System.out.println("Total: " + obj.total());

System.out.println("Average: " + obj.average());

obj.mean();

obj.mode();

obj.median();

}

}

6. Create an interface with 4 methods called add(), sub(), mul() and div(). Then give implementation for all in the implementing class

interface Calculator {

int add(int a, int b);

int sub(int a, int b);

int mul(int a, int b);

int div(int a, int b);

}

class CalculatorImpl implements Calculator {

@Override

public int add(int a, int b) {

return a + b;

}

@Override

public int sub(int a, int b) {

return a - b;

}

@Override

public int mul(int a, int b) {

return a \* b;

}

@Override

public int div(int a, int b) {

if (b == 0) {

throw new IllegalArgumentException("Cannot divide by zero");

}

return a / b;

}

}

public class Main {

public static void main(String[] args) {

Calculator calculator = new CalculatorImpl();

System.out.println("Addition: " + calculator.add(10, 5));

System.out.println("Subtraction: " + calculator.sub(10, 5));

System.out.println("Multiplication: " + calculator.mul(10, 5));

System.out.println("Division: " + calculator.div(10, 5));

}

}

7. Create 3 interfaces with 1 method each sum(), avg(), percentage() respectively. Now implement all the 3 interfaces in your class

interface Sum {

int sum(int[] arr);

}

interface Avg {

double avg(int[] arr);

}

interface Percentage {

double percentage(int total, int obtained);

}

class MyClass implements Sum, Avg, Percentage {

@Override

public int sum(int[] arr) {

int sum = 0;

for (int num : arr) {

sum += num;

}

return sum;

}

@Override

public double avg(int[] arr) {

int sum = sum(arr);

return (double) sum / arr.length;

}

@Override

public double percentage(int total, int obtained) {

return ((double) obtained / total) \* 100;

}

}

public class Main {

public static void main(String[] args) {

int[] numbers = {10, 20, 30, 40, 50};

MyClass obj = new MyClass();

System.out.println("Sum: " + obj.sum(numbers));

System.out.println("Average: " + obj.avg(numbers));

System.out.println("Percentage: " + obj.percentage(500, 350));

}

}

8. Create an interface called Tree and extend 2 classes from it called Branch1 and Branch2. Tree should contain methods fruits(), leaves() and flowers(), these methods contain 2,3,4 parameters respectively.

interface Tree {

void fruits(int num1, int num2);

void leaves(int num1, int num2, int num3);

void flowers(int num1, int num2, int num3, int num4);

}

class Branch1 implements Tree {

@Override

public void fruits(int num1, int num2) {

System.out.println("Branch1 - Fruits: " + (num1 + num2));

}

@Override

public void leaves(int num1, int num2, int num3) {

System.out.println("Branch1 - Leaves: " + (num1 + num2 + num3));

}

@Override

public void flowers(int num1, int num2, int num3, int num4) {

System.out.println("Branch1 - Flowers: " + (num1 + num2 + num3 + num4));

}

}

class Branch2 implements Tree {

@Override

public void fruits(int num1, int num2) {

System.out.println("Branch2 - Fruits: " + (num1 \* num2));

}

@Override

public void leaves(int num1, int num2, int num3) {

System.out.println("Branch2 - Leaves: " + (num1 \* num2 \* num3));

}

@Override

public void flowers(int num1, int num2, int num3, int num4) {

System.out.println("Branch2 - Flowers: " + (num1 \* num2 \* num3 \* num4));

}

}

public class Main {

public static void main(String[] args) {

Branch1 b1 = new Branch1();

b1.fruits(10, 5);

b1.leaves(5, 7, 3);

b1.flowers(2, 3, 4, 5);

Branch2 b2 = new Branch2();

b2.fruits(10, 5);

b2.leaves(5, 7, 3);

b2.flowers(2, 3, 4, 5);

}

}

9. Use static keyword in the following levels a) Static variable

class Example {

static int count = 0;

Example() {

count++;

}

static void display() {

System.out.println("Count: " + count);

}

}

public class Main {

public static void main(String[] args) {

Example e1 = new Example();

Example e2 = new Example();

Example e3 = new Example();

Example.display(); // This will print the count of objects created

}

}

10. Use Final keyword in the following levels a) Final variable

class Example {

final int value = 10;

void display() {

// value = 20; // This will give a compilation error as value is final

System.out.println("Value: " + value);

}

}

public class Main {

public static void main(String[] args) {

Example obj = new Example();

obj.display();

}

}

11. Use ‘this’ keyword in the following purposes a) Referencing instance variable

class Example {

int num;

Example(int num) {

this.num = num; // Using 'this' to refer to instance variable

}

void display() {

System.out.println("Value of num: " + num);

}

}

public class Main {

public static void main(String[] args) {

Example obj = new Example(10);

obj.display();

}

}

12. Use ‘super’ keyword in the following purposes a) Accessing superclass members

class Parent {

int num = 10;

}

class Child extends Parent {

void display() {

System.out.println("Value of num in parent class: " + super.num); // Accessing superclass member using 'super'

}

}

public class Main {

public static void main(String[] args) {

Child obj = new Child();

obj.display();

}

}

Generics in Java

13. Single type parameter generic class

class Box<T> {

private T value;

public Box(T value) {

this.value = value;

}

public T getValue() {

return value;

}

public void setValue(T value) {

this.value = value;

}

}

public class Main {

public static void main(String[] args) {

Box<Integer> integerBox = new Box<>(10);

Box<String> stringBox = new Box<>("Hello");

System.out.println("Integer Value: " + integerBox.getValue());

System.out.println("String Value: " + stringBox.getValue());

}

}

14. Multiple type parameter generic class

class Pair<K, V> {

private K key;

private V value;

public Pair(K key, V value) {

this.key = key;

this.value = value;

}

public K getKey() {

return key;

}

public void setKey(K key) {

this.key = key;

}

public V getValue() {

return value;

}

public void setValue(V value) {

this.value = value;

}

}

public class Main {

public static void main(String[] args) {

Pair<String, Integer> pair1 = new Pair<>("One", 1);

Pair<String, String> pair2 = new Pair<>("Name", "John");

System.out.println("Pair 1: " + pair1.getKey() + " -> " + pair1.getValue());

System.out.println("Pair 2: " + pair2.getKey() + " -> " + pair2.getValue());

}

}

15. Using generics on methods example

public class Main {

// Generic method to print an array

public static <E> void printArray(E[] inputArray) {

for (E element : inputArray) {

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

}

System.out.println();

}

public static void main(String[] args) {

// Integer array

Integer[] intArray = {1, 2, 3, 4, 5};

// Double array

Double[] doubleArray = {1.1, 2.2, 3.3, 4.4};

// Character array

Character[] charArray = {'H', 'E', 'L', 'L', 'O'};

System.out.println("Array of integers:");

printArray(intArray);

System.out.println("Array of doubles:");

printArray(doubleArray);

System.out.println("Array of characters:");

printArray(charArray);

}

}

16. Restrict use of primitive types using generics

public class Main {

// Generic method that only accepts reference types (not primitive types)

public static <T> void printType(T t) {

System.out.println("Type: " + t.getClass().getName());

}

public static void main(String[] args) {

printType("Hello");

printType(10);

printType(10.5);

printType('A');

}

}

17. Use Scanner to get Char, String, Int, Float and Double input same

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a character: ");

char ch = scanner.next().charAt(0);

System.out.print("Enter a string: ");

String str = scanner.next();

System.out.print("Enter an integer: ");

int numInt = scanner.nextInt();

System.out.print("Enter a float: ");

float numFloat = scanner.nextFloat();

System.out.print("Enter a double: ");

double numDouble = scanner.nextDouble();

System.out.println("Character: " + ch);

System.out.println("String: " + str);

System.out.println("Integer: " + numInt);

System.out.println("Float: " + numFloat);

System.out.println("Double: " + numDouble);

scanner.close();

}

}

18. Find System Date and Time using Date class

import java.util.Date;

public class Main {

public static void main(String[] args) {

Date date = new Date();

System.out.println("Current Date and Time: " + date);

}

}

19. Use UUID to generate a random Universally Unique Identifier

import java.util.UUID;

public class Main {

public static void main(String[] args) {

// Generating a random UUID

UUID uuid = UUID.randomUUID();

System.out.println("Random UUID: " + uuid);

}

}

20. Java toString() and equals() method.

class Student {

private int id;

private String name;

public Student(int id, String name) {

this.id = id;

this.name = name;

}

@Override

public String toString() {

return "Student{" +

"id=" + id +

", name='" + name + '\'' +

'}';

}

@Override

public boolean equals(Object obj) {

if (this == obj) return true;

if (obj == null || getClass() != obj.getClass()) return false;

Student student = (Student) obj;

return id == student.id && name.equals(student.name);

}

}

public class Main {

public static void main(String[] args) {

Student s1 = new Student(1, "Alice");

Student s2 = new Student(2, "Bob");

// toString() example

System.out.println("Student 1: " + s1);

System.out.println("Student 2: " + s2);

// equals() example

Student s3 = new Student(1, "Alice");

System.out.println("Are s1 and s3 equal? " + s1.equals(s3));

}

}