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

CODE:

class Parent {

void even() {

System.out.println("Even numbers from 1 to 20:");

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

if (i % 2 == 0) {

System.out.println(i);

}

}

}

void odd() {

System.out.println("Odd numbers from 1 to 20:");

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

if (i % 2 != 0) {

System.out.println(i);

}

}

}

void prime() {

System.out.println("Prime numbers from 1 to 20:");

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

if (isPrime(i)) {

System.out.println(i);

}

}

}

boolean isPrime(int n) {

if (n <= 1) {

return false;

}

for (int i = 2; i < n; i++) {

if (n % i == 0) {

return false;

}

}

return true;

}

}

class Child extends Parent {

@Override

void even() {

System.out.println("Even numbers from 1 to 30:");

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

if (i % 2 == 0) {

System.out.println(i);

}

}

}

@Override

void odd() {

System.out.println("Odd numbers from 1 to 30:");

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

if (i % 2 != 0) {

System.out.println(i);

}

}

}

void prime() {

System.out.println("Prime numbers from 1 to 30:");

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

if (isPrime(i)) {

System.out.println(i);

}

}

}

}

public class Main {

public static void main(String[] args) {

Child child = new Child();

child.even();

System.out.println();

child.odd();

System.out.println();

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()

CODE:

class MyClass {

int a;

int b;

MyClass() {

this.a = 10;

this.b = 15;

}

MyClass(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) {

MyClass obj1 = new MyClass();

obj1.disp();

MyClass obj2 = new MyClass(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.

CODE:

class Base {

void area(int a, int b) {

int squareArea = a \* a;

System.out.println("Area of the square: " + squareArea);

}

}

class Derived extends Base {

@Override

void area(int a, int b) {

int rectangleArea = a \* b;

System.out.println("Area of the rectangle: " + rectangleArea);

}

}

public class Main {

public static void main(String[] args) {

Base base = new Base();

base.area(4, 5); // This will call the method in the Base class

Derived derived = new Derived();

derived.area(4, 5); // This will call the overridden method in the Derived

class

}

}

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 parameters

CODE:

public class Main {

int sum(int a, int b) {

return a + b;

}

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

return a + b + c;

}

int sum(int a) {

return a;

}

public static void main(String[] args) {

Main main = new Main();

System.out.println("Sum of 2 and 3: " + main.sum(2, 3));

System.out.println("Sum of 2, 3 and 4: " + main.sum(2, 3, 4));

System.out.println("Sum of 2: " + main.sum(2)); /

}

}

b) By changing the data types of parameters

CODE:

public class Main {

int sum(int a, int b) {

return a + b;

}

double sum(double a, double b) {

return a + b;

}

public static void main(String[] args) {

Main main = new Main();

System.out.println("Sum of 2 and 3: " + main.sum(2, 3));

System.out.println("Sum of 2.5 and 3.5: " + main.sum(2.5, 3.5));

}

}

c) By interchanging the parameters

CODE:

public class Main {

public int sum(int a, int b) {

return a + b;

}

public int sum(double a, double b) {

return (int)(a + b);

}

public static void main(String[] args) {

Main main = new Main();

int sum1 = main.sum(1, 2);

System.out.println("Sum of two integers: " + sum1);

int sum2 = main.sum(1.5, 2.5);

System.out.println("Sum of two doubles: " + sum2);

}

}

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

CODE:

import java.util.Arrays;

abstract class AbstractMath {

abstract int total(int[] numbers);

abstract double average(int[] numbers);

double mean(int[] numbers) {

return total(numbers) / (double) numbers.length;

}

int mode(int[] numbers) {

return numbers[0];

}

double median(int[] numbers) {

Arrays.sort(numbers);

int n = numbers.length;

if (n % 2 == 0) {

return (numbers[n/2 - 1] + numbers[n/2]) / 2.0;

} else {

return numbers[n/2];

}

}

}

class MathOperations extends AbstractMath {

int total(int[] numbers) {

int sum = 0;

for (int num : numbers) {

sum += num;

}

return sum;

}

double average(int[] numbers) {

return total(numbers) / (double) numbers.length;

}

}

public class Java {

public static void main(String[] args) {

int[] numbers = {3, 7, 2, 5, 8};

MathOperations math = new MathOperations();

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

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

System.out.println("Mean: " + math.mean(numbers));

System.out.println("Mode: " + math.mode(numbers));

System.out.println("Median: " + math.median(numbers));

}

}

6. Create an interface with 4 methods called add(), sub(), mul() and div().

Then give implementation for all in the implementing class

CODE:

interface Calculator {

int add(int a, int b);

int sub(int a, int b);

int mul(int a, int b);

double div(int a, int b);

}

class BasicCalculator 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 double div(int a, int b) {

if (b == 0) {

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

}

return (double) a / b;

}

}

public class Java {

public static void main(String[] args) {

BasicCalculator calculator = new BasicCalculator();

int sum = calculator.add(5, 3);

int difference = calculator.sub(10, 4);

int product = calculator.mul(6, 2);

double quotient = calculator.div(15, 3);

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

System.out.println("Difference: " + difference);

System.out.println("Product: " + product);

System.out.println("Quotient: " + quotient);

}

}

7. Create 3 interfaces with 1 method each sum(), avg(), percentage()

respectively. Now implement all the 3 interfaces in your class

CODE:

interface Summable {

int sum(int[] numbers);

}

interface Averagable {

double avg(int[] numbers);

}

interface PercentageCalculable {

double percentage(double obtained, double total);

}

class Calculation implements Summable, Averagable, PercentageCalculable {

@Override

public int sum(int[] numbers) {

int total = 0;

for (int num : numbers) {

total += num;

}

return total;

}

@Override

public double avg(int[] numbers) {

return (double) sum(numbers) / numbers.length;

}

@Override

public double percentage(double obtained, double total) {

return (obtained / total) \* 100.0;

}

}

public class Java {

public static void main(String[] args) {

int[] numbers = {80, 75, 90, 85, 95};

Calculation calculation = new Calculation();

int sum = calculation.sum(numbers);

double avg = calculation.avg(numbers);

double percentage = calculation.percentage(350, 500);

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

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

System.out.println("Percentage: " + percentage + "%");

}

}

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.

CODE:

interface Tree {

void fruits(String fruit1, String fruit2);

void leaves(String leaf1, String leaf2, String leaf3);

void flowers(String flower1, String flower2, String flower3, String flower4);

}

class Branch1 implements Tree {

@Override

public void fruits(String fruit1, String fruit2) {

System.out.println("Fruits on Branch1: " + fruit1 + ", " + fruit2);

}

@Override

public void leaves(String leaf1, String leaf2, String leaf3) {

System.out.println("Leaves on Branch1: " + leaf1 + ", " + leaf2 + ", " + leaf3);

}

@Override

public void flowers(String flower1, String flower2, String flower3, String flower4) {

System.out.println("Flowers on Branch1: " + flower1 + ", " + flower2 + ", " + flower3

+ ", " + flower4);

}

}

class Branch2 implements Tree {

@Override

public void fruits(String fruit1, String fruit2) {

System.out.println("Fruits on Branch2: " + fruit1 + ", " + fruit2);

}

@Override

public void leaves(String leaf1, String leaf2, String leaf3) {

System.out.println("Leaves on Branch2: " + leaf1 + ", " + leaf2 + ", " + leaf3);

}

@Override

public void flowers(String flower1, String flower2, String flower3, String flower4) {

System.out.println("Flowers on Branch2: " + flower1 + ", " + flower2 + ", " + flower3

+ ", " + flower4);

}

}

public class Java {

public static void main(String[] args) {

Branch1 branch1 = new Branch1();

Branch2 branch2 = new Branch2();

branch1.fruits("Apple", "Orange");

branch1.leaves("Green", "Yellow", "Red");

branch1.flowers("Rose", "Lily", "Daisy", "Sunflower");

branch2.fruits("Mango", "Banana");

branch2.leaves("Brown", "Orange", "Purple");

branch2.flowers("Tulip", "Daffodil", "Hibiscus", "Cherry Blossom");

}

}

9. Use static keyword in the following levels

a) Static variable

CODE:

class MyClass {

static int count = 0;

MyClass() {

count++;

}

void displayCount() {

System.out.println("Number of instances created: " + count);

}

}

public class Main {

public static void main(String[] args) {

MyClass obj1 = new MyClass();

obj1.displayCount();

MyClass obj2 = new MyClass();

obj2.displayCount();

}

}

b) Static method

CODE:

class MyClass {

static void displayMessage() {

System.out.println("Hello from the static method!");

}

}

public class Main {

public static void main(String[] args) {

MyClass.displayMessage();

}

}

c) Static block

CODE:

class MyClass {

static {

num = 10;

System.out.println("Static block executed. num: " + num);

}

}

public class Main {

public static void main(String[] args) {

MyClass obj = new MyClass();

}

}

d) Static nested classes

CODE:

class OuterClass {

static int outerNum = 10;

static class StaticNestedClass {

void display() {

System.out.println("outerNum from OuterClass: " + outerNum);

}

}

}

public class Main {

public static void main(String[] args) {

OuterClass.StaticNestedClass nestedObj = new OuterClass.StaticNestedClass();

nestedObj.display(); // Output: outerNum from OuterClass: 10

}

}

10. Use Final keyword in the following levels

a) Final variable

CODE:

class MyClass {

final int num = 10;

void display() {

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

}

}

public class Main {

public static void main(String[] args) {

MyClass obj = new MyClass();

obj.display();

}

}

b) Final method

CODE:

class Parent {

final void display() {

System.out.println("Hello from Parent!");

}

}

class Child extends Parent {

void sayHello() {

System.out.println("Hello from Child!");

}

}

public class Main {

public static void main(String[] args) {

Child obj = new Child();

obj.display();

obj.sayHello();

}

}

c) Final classes

CODE:

final class FinalClass {

void display() {

System.out.println("Hello from FinalClass!");

}

}

}

public class Main {

public static void main(String[] args) {

FinalClass obj = new FinalClass();

obj.display(); }

}

11. Use ‘this’ keyword in the following purposes

a) Referencing instance variable

CODE:

public class ReferencingInstanceVariable {

private int number;

public void setNumber(int number) {

this.number = number;

}

public int getNumber() {

return this.number;

}

public static void main(String[] args) {

ReferencingInstanceVariable obj = new ReferencingInstanceVariable();

obj.setNumber(42);

System.out.println("Number: " + obj.getNumber());

}

}

b) Invoking another constructor

CODE:

public class InvokingAnotherConstructor {

private int number;

private String text;

public InvokingAnotherConstructor(int number, String text) {

this.number = number;

this.text = text;

}

public InvokingAnotherConstructor(int number) {

this(number, "Default Text");

}

public void displayValues() {

System.out.println("Number: " + number);

System.out.println("Text: " + text);

}

public static void main(String[] args) {

InvokingAnotherConstructor obj1 = new InvokingAnotherConstructor(42, "Hello,

World!");

obj1.displayValues();

InvokingAnotherConstructor obj2 = new InvokingAnotherConstructor(17);

obj2.displayValues();

}

}

c) Passing current object as a parameter

CODE:

public class PassingCurrentObjectAsParameter {

private int number;

public PassingCurrentObjectAsParameter(int number) {

this.number = number;

}

public void processData(PassingCurrentObjectAsParameter anotherObject) {

int sum = this.number + anotherObject.number;

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

}

public static void main(String[] args) {

PassingCurrentObjectAsParameter obj1 = new

PassingCurrentObjectAsParameter(30);

PassingCurrentObjectAsParameter obj2 = new

PassingCurrentObjectAsParameter(15);

obj1.processData(obj2);

}

}

d) Returning current object

CODE:

public class ReturningCurrentObject {

private int number;

public ReturningCurrentObject setNumberAndReturn(int number) {

this.number = number;

return this;

}

public void displayValue() {

System.out.println("Number: " + number);

}

public static void main(String[] args) {

ReturningCurrentObject obj = new ReturningCurrentObject();

obj.setNumberAndReturn(10).displayValue();

}

}

12. Use ‘super’ keyword in the following purposes

a) Accessing superclass members

CODE:

public class Animal1 {

String sound = "Animal Sound";

}

class Dog1 extends Animal1 {

void displaySound() {

System.out.println("Dog Sound: " + super.sound);

}

}

public class SuperKeywordExample {

public static void main(String[] args) {

Dog1 myDog = new Dog1();

myDog.displaySound();

}

}

b) Calling superclass constructor

CODE:

class Animal {

String sound;

Animal(String sound) {

this.sound = sound;

}

}

class Dog extends Animal {

Dog(String sound) {

super(sound);

}

void displaySound() {

System.out.println("Dog Sound: " + super.sound);

}

}

public class SuperKeywordExampleB {

public static void main(String[] args) {

Dog myDog = new Dog("Woof!");

myDog.displaySound();

}

}

c) Invoking superclass method

CODE:

class Animal {

void eat() {

System.out.println("Animal is eating");

}

}

class Dog extends Animal {

void eat() {

super.eat();

System.out.println("Dog is eating");

}

}

public class SuperKeywordExampleC {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.eat();

}

}

13. Single type parameter generic class

CODE:

public 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 static void main(String[] args) {

Box<Integer> integerBox = new Box<>(42);

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

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

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

integerBox.setValue(99);

stringBox.setValue("Updated Value");

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

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

}

}

14. Multiple type parameter generic class

CODE:

public 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 static void main(String[] args) {

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

Pair<String, Double> pair2 = new Pair<>("PI", 3.14);

System.out.println("Pair 1: Key=" + pair1.getKey() + ", Value=" + pair1.getValue());

System.out.println("Pair 2: Key=" + pair2.getKey() + ", Value=" + pair2.getValue());

pair1.setKey(42);

pair2.setValue(2.71);

System.out.println("Updated Pair 1: Key=" + pair1.getKey() + ", Value=" +

pair1.getValue());

System.out.println("Updated Pair 2: Key=" + pair2.getKey() + ", Value=" +

pair2.getValue());

}

}

15. Using generics on methods example

CODE:

public class GenericMethodExample {

public static <T extends Comparable<T>> boolean isEqual(T value1, T value2) {

return value1.compareTo(value2) == 0;

}

public static <T extends Comparable<T>> T findMax(T[] array) {

if (array == null || array.length == 0) {

return null;

}

T max = array[0];

for (int i = 1; i < array.length; i++) {

if (array[i].compareTo(max) > 0) {

max = array[i];

}

}

return max;

}

public static void main(String[] args) {

int int1 = 5;

int int2 = 5;

System.out.println("Are integers equal? " + isEqual(int1, int2));

String str1 = "Hello";

String str2 = "World";

System.out.println("Are strings equal? " + isEqual(str1, str2));

Double[] doubleArray = { 3.14, 2.71, 1.618 };

System.out.println("Maximum double value: " +findMax(doubleArray));

}

}

16. Restrict use of primitive types using generics

CODE:

public class GenericRestrictionExample<T extends Number> {

private T value;

public GenericRestrictionExample(T value) {

this.value = value;

}

public T getValue() {

return value;

}

public static void main(String[] args) {

GenericRestrictionExample<Integer> integerExample = new

GenericRestrictionExample<>(42);

System.out.println("Integer value: " + integerExample.getValue());

GenericRestrictionExample<Double> doubleExample = new

GenericRestrictionExample<>(3.14);

System.out.println("Double value: " + doubleExample.getValue());

}

}

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

moment

CODE:

import java.util.Scanner;

public class InputExample {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

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

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

String stringInput = scanner.next();

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

int intInput = scanner.nextInt();

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

float floatInput = scanner.nextFloat();

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

double doubleInput = scanner.nextDouble();

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

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

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

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

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

scanner.close();

}

}

18. Find System Date and Time using Date class

CODE:

import java.util.Date;

import java.time.LocalDateTime;

public class SystemDateTimeExample {

public static void main(String[] args) {

Date currentDate = new Date();

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

LocalDateTime currentDateTime = LocalDateTime.now();

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

}

}

19. Use UUID to generate a random Universally Unique Identifier

CODE:

import java.util.UUID;

public class UUIDExample {

public static void main(String[] args) {

UUID uuid = UUID.randomUUID();

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

}

}

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

CODE:

public class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public String toString() {

return "Person [name=" + name + ", age=" + age + "]";

}

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj == null || getClass() != obj.getClass()) {

return false;

}

Person person = (Person) obj;

return age == person.age && name.equals(person.name);

}

public static void main(String[] args) {

Person person1 = new Person("John", 25);

Person person2 = new Person("John", 25);

System.out.println("person1: " + person1.toString());

System.out.println("person2: " + person2.toString());

System.out.println("Are persons equal? " + person1.equals(person2));

}

}