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.

import java.util.Arrays;

public class Parent {

void even() {

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

for (int i = 1; i <= 20; i += 2) {

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

}

System.out.println();

}

void odd() {

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

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

if (i % 2 != 0) {

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

}

}

System.out.println();

}

void prime(int n) {

System.out.println("Parent prime numbers from 1 to " + n + ":");

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

if (isPrime(i)) {

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

}

}

System.out.println();

}

private boolean isPrime(int n) {

if (n <= 1) {

return false;

}

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

if (n % i == 0) {

return false;

}

}

return true;

}

}

public class Child extends Parent {

@Override

void even() {

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

int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,29, 30};

for (int number : numbers) {

if (number % 2 == 0) {

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

}

}

System.out.println();

}

@Override

void odd() {

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

int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30};

for (int number : numbers) {

if (number % 2 != 0) {

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

}

}

System.out.println();

}

@Override

void prime(int n) {

System.out.println("Child prime numbers from 1 to " + n + ":");

int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30};

for (int number : numbers) {

if (isPrime(number)) {

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

}

}

System.out.println();

}

}

public class Main {

public static void main(String[] args) {

Child child = new Child();

child.even();

child.odd();

child.prime(30);

}

}

Output:

Child even numbers from 1 to 30:

2 4 6 8 10 12 14 16 18 20 22 24 26 28 30

Child odd numbers from 1 to 30:

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29

Child prime numbers from 1 to 30:

2 3 5 7 11 13 17 19 23 29

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 MyClass {

int a;

int b;

// Default constructor

MyClass() {

a = 10;

b = 15;

disp();

}

// Parameterized constructor

MyClass(int a, int b) {

this.a = a;

this.b = b;

disp();

}

void disp() {

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

}

}

public class Main {

public static void main(String[] args) {

// Creating object using default constructor

System.out.println("Using default constructor:");

MyClass obj1 = new MyClass();

// Creating object using parameterized constructor

System.out.println("\nUsing parameterized constructor:");

MyClass obj2 = new MyClass(20, 25);

}

}

Output:

a = 10, b = 15

a = 20, b = 25

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

public class Shape {

public void area(int side) {

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

}

}

public class Rectangle extends Shape {

@Override

public void area(int length, int width) {

super.area(length); // Call the base class method with the length as the side of a square

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

}

}

public class Main {

public static void main(String[] args) {

Rectangle rectangle = new Rectangle();

rectangle.area(5, 10); // Call the derived class method with the length and width of a rectangle

}

}

Output:

Area of a square: 25

Area of a rectangle: 50

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 b) By changing the data types of parameters c) By interchanging the parameters.

A) By increasing and decreasing no. of parameters

public class Main {

public static int sum(int a, int b) {

return a + b;

}

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

return a + b + c;

}

public static void main(String[] args) {

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

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

}

}

Output:

Sum of 2 numbers: 5

Sum of 3 numbers: 9

B) By changing the data types of parameters:

public class Main {

public static int sum(int a, int b) {

return a + b;

}

public static double sum(double a, double b) {

return a + b;

}

public static void main(String[] args) {

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

System.out.println("Sum of 2 doubles: " + sum(2.5, 3.5));

}

}

Output:

Sum of 2 integers: 5

Sum of 2 doubles: 6.0

C) By interchanging the parameters:

public class Main {

public static int sum(int a, int b) {

return a + b;

}

public static int sum(int b, int a) {

return a + b;

}

public static void main(String[] args) {

System.out.println("Sum of 2 numbers (a + b): " + sum(2, 3));

System.out.println("Sum of 2 numbers (b + a): " + sum(3, 2));

}

}

Output:

Sum of 2 numbers (a + b): 5

Sum of 2 numbers (b + a): 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 with 2 abstract methods and 3 concrete methods

public abstract class AbstractStatistics {

// Abstract method for calculating the total

public abstract double total();

// Abstract method for calculating the average

public abstract double average();

// Concrete method for calculating the mean

public double mean() {

double total = this.total();

double count = this.count();

return total / count;

}

// Concrete method for calculating the mode

public int mode() {

// Assumes that the data is stored in an array called data

int[] counts = new int[100]; // assuming the data values are between 0 and 99

for (int i = 0; i < data.length; i++) {

counts[data[i]]++;

}

int modeValue = 0;

int maxCount = 0;

for (int i = 0; i < counts.length; i++) {

if (counts[i] > maxCount) {

maxCount = counts[i];

modeValue = i;

}

}

return modeValue;

}

// Concrete method for calculating the median

public double median() {

double[] sortedData = this.sortedData();

int n = sortedData.length;

if (n % 2 == 0) {

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

} else {

return sortedData[n/2];

}

}

// Abstract method for getting the sorted data

public abstract double[] sortedData();

// Abstract method for getting the count of data points

public abstract int count();

}

// Concrete class that extends the abstract class

public class ConcreteStatistics extends AbstractStatistics {

private int[] data;public ConcreteStatistics(int[] data) {

this.data = data;

}

// Implementation of the abstract method for calculating the total

@Override

public double total() {

double total = 0;

for (int i = 0; i < data.length; i++) {

total += data[i];

}

return total;

}

// Implementation of the abstract method for calculating the average

@Override

public double average() {

return this.total() / this.count();

}

// Implementation of the abstract method for getting the sorted data

@Override

public double[] sortedData() {

double[] sortedData = new double[data.length];

System.arraycopy(data, 0, sortedData, 0, data.length);

Arrays.sort(sortedData);

return sortedData;

}

// Implementation of the abstract method for getting the count of data points

@Override

public int count() {

return data.length;

}

}

int[] data = {1, 2, 3, 4, 5};

ConcreteStatistics stats = new ConcreteStatistics(data);

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

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

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

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

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

Output:

Total: 15

Average: 3.0

Mean: 2.5

Mode: 1

Median: 3.0

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

// Interface with 4 methods

public interface Calculator {

double add(double a, double b);

double sub(double a, double b);

double mul(double a, double b);

double div(double a, double b);

}

// Implementing class with implementation for the 4 methods

public class BasicCalculator implements Calculator {

@Override

public double add(double a, double b) {

return a + b;

}

@Override

public double sub(double a, double b) {

return a - b;

}

@Override

public double mul(double a, double b) {

return a \* b;

}

@Override

public double div(double a, double b) {

if (b == 0) {

throw new IllegalArgumentException("Division by zero is not allowed.");

}

return a / b;

}

}

// Example usage

public class Main {

public static void main(String[] args) {

Calculator calculator = new BasicCalculator();

System.out.println("Addition: " + calculator.add(2, 3));

System.out.println("Subtraction: " + calculator.sub(2, 3));

System.out.println("Multiplication: " + calculator.mul(2, 3));

System.out.println("Division: " + calculator.div(2, 3));

}

}

Output:

Addition: 5.0

Subtraction: -1.0

Multiplication: 6.0

Division: 0.6666666666666666

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

public interface Summable {

double sum(double[] numbers);

}

// Interface with 1 method for calculating the average

public interface Averageable {

double avg(double[] numbers);

}

public interface Percentageable {

double percentage(double part, double whole);

}

public class BasicStatistics implements Summable, Averageable, Percentageable {

@Override

public double sum(double[] numbers) {

double sum = 0;

for (double number : numbers) {

sum += number;

}

return sum;

}

@Override

public double avg(double[] numbers) {

return sum(numbers) / numbers.length;

}

@Override

public double percentage(double part, double whole) {

return (part / whole) \* 100;

}

}

// Example usage

public class Main {

public static void main(String[]args) {

BasicStatistics statistics = new BasicStatistics();

double[] numbers = {1, 2, 3, 4, 5};

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

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

System.out.println("Percentage: " + statistics.percentage(3, 5));

}

}

Output:

Sum: 15.0

Average: 3.0

Percentage: 60.0

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.

public interface Tree {

int fruits(int apple, int orange);

int leaves(int green, int yellow, int red);

double flowers(double white, double pink, double red, double yellow);

}

// Class extending the Tree interface

public class Branch1 implements Tree {

@Override

public int fruits(int apple, int orange) {

return apple + orange;

}

@Override

public int leaves(int green, int yellow, int red) {

return green + yellow + red;

}

@Override

public double flowers(double white, double pink, double red, double yellow) {

return white + pink + red + yellow;

}

}

// Class extending the Tree interface

public class Branch2 implements Tree {

@Override

public int fruits(int apple, int orange) {

return apple \* 2 + orange \* 3;

}

@Override

public int leaves(int green, int yellow, int red) {

return green \* 2 + yellow \* 3 + red \* 4;

}

@Override

public double flowers(double white, double pink, double red, double yellow) {

return white \* 0.5 + pink \* 0.75 + red \* 1 + yellow \* 1.5;

}

}

// Example usage

public class Main {

public static void main(String[] args) {

Tree branch1 = new Branch1();

System.out.println("Branch1:");

System.out.println("Fruits: " + branch1.fruits(2, 3));

System.out.println("Leaves: " + branch1.leaves(2, 3, 4));

System.out.println("Flowers: " + branch1.flowers(2, 3, 4, 5));

Tree branch2 = new Branch2();

System.out.println("Branch2:");

System.out.println("Fruits: " + branch2.fruits(2, 3));

System.out.println("Leaves: " + branch2.leaves(2, 3, 4));

System.out.println("Flowers: " + branch2.flowers(2, 3, 4, 5));

}

}

Output:

Branch1:

Fruits: 5

Leaves: 9

Flowers: 11.5

Branch2:

Fruits: 10

Leaves: 25

Flowers: 13.5

9. use static keyword in the following levels

a) Static variable

b) Static method

c) Static block

d) Static nested classes

a)

public class Main {

public static int count = 0;

public Main() {

count++;

}

public static void main(String[] args) {

Main obj1 = new Main();

Main obj2 = new Main();

Main obj3 = new Main();

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

}

}

Output:

Number of objects created: 3

b)

public class Main {

public static int sum(int a, int b) {

return a + b;

}

public static void main(String[] args) {

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

}

}

Output:

Sum of 2 numbers: 5

c)

public class Main {

static {

System.out.println("Static block executed");

}

public static void main(String[] args) {

System.out.println("Main method executed");

}

}

Output:

Static block executed

Main method executed

d)

public class Main {

public static class NestedClass {

public void printMessage() {

System.out.println("Hello, World!");

}

}

public static void main(String[] args) {

Main.NestedClass obj = new Main.NestedClass();

obj.printMessage();

}

}

Output:

Hello, World!

10. Use Final keyword in the following levels

a) Final variable

b) Final method

c) Final classes

a) final variable

public class FinalVariableExample {

public final int MAX\_SIZE = 100;

public static void main(String[] args) {

}

}

Output:

(No output)

b) final method

public class FinalMethodExample {

public final void printMessage() {

System.out.println("Hello, World!");

}

public static void main(String[] args) {

FinalMethodExample example = new FinalMethodExample();

example.printMessage();

}

}

Output:

Hello, World!

c) final classes

public final class FinalClassExample {

public void printMessage() {

System.out.println("Hello, World!");

}

public static void main(String[] args) {

FinalClassExample example = new FinalClassExample();

example.printMessage();

}

}

Output:

Hello, World!

11. Use ‘this’ keyword in the following purposes

a) Referencing instance variable

b) Invoking another constructor

c) Passing current object as a parameter

d) Returning current object

a) Referencing instance variable

public class Rectangle {

private int width;

private int height;

public Rectangle(int width, int height) {

this.width = width;

this.height = height;

}

public int getArea() {

return width \* height;

}

public static void main(String[] args) {

Rectangle rect = new Rectangle(5, 10);

System.out.println("Area: " + rect.getArea());

}

}

Output:

Area: 50

b)

public class Rectangle {

private int width;

private int height;

public Rectangle(int width, int height) {

this.width = width;

this.height = height;

}

public Rectangle(int size) {

this(size, size);

}

public int getArea() {

return width \* height;

}

public static void main(String[] args) {

Rectangle rect1 = new Rectangle(5, 10);

Rectangle rect2 = new Rectangle(7);

System.out.println("Area of rect1: " + rect1.getArea());

System.out.println("Area of rect2: " + rect2.getArea());

}

}

Output:

Area of rect1: 50

Area of rect2: 49

c)

public class Rectangle {

private int width;

private int height;

public Rectangle(int width, int height) {

this.width = width;

this.height = height;

}

public int getArea() {

return width \* height;

}

public static void printArea(Rectangle rect) {

System.out.println("Area: " + rect.getArea());

}

public static void main(String[] args) {

Rectangle rect = new Rectangle(5, 10);

printArea(rect);

}

}

Output:

Area: 50

d)

public class Rectangle {

private int width;

private int height;

public Rectangle(int width, int height) {

this.width = width;

this.height = height;

}

public Rectangle setWidth(int width) {

this.width = width;

return this;

}

public Rectangle setHeight(int height) {

this.height = height;

return this;

}

public int getArea() {

return width \* height;

}

public static void main(String[] args) {

Rectangle rect = new Rectangle(5, 10);

rect.setWidth(10).setHeight(20);

System.out.println("Area: " + rect.getArea());

}

}

Output:

Area: 200

12. Use ‘super’ keyword in the following purposes

a) Accessing superclass members

b) Calling superclass constructor

c) Invoking superclass methods

public class Animal {

protected String name;

public Animal(String name) {

this.name = name;

}

public String getName() {

return name;

}

}

public class Dog extends Animal {

private int age;

public Dog(String name, int age) {

super(name);

this.age = age;

}

public int getAge() {

return age;

}

public void bark() {

System.out.println("Woof!");

}

public String toString() {

return "Dog [name=" + super.getName() + ", age=" + age + "]";

}

public static void main(String[] args) {

Dog dog = new Dog("Fido", 3);

System.out.println(dog);

}

}

Output:

Dog [name=Fido, age=3]

B)

public class Animal {

protected String name;

public Animal(String name) {

this.name = name;

}

public String getName() {

return name;

}

}

public class Dog extends Animal {

private int age;

public Dog(String name, int age) {

super(name);

this.age = age;

}

public int getAge() {

return age;

}

public static void main(String[] args) {

Dog dog = new Dog("Fido", 3);

}

}

Output:

(No output)

c)

public class Animal {

protected String name;

public Animal(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void eat() {

System.out.println("The animal is eating.");

}

}

public class Dog extends Animal {

private int age;

public Dog(String name, int age) {

super(name);

this.age = age;

}

public int getAge() {

return age;

}

@Override

public void eat() {

super.eat();

System.out.println("The dog is eating dog food.");

}

public static void main(String[] args) {

Dog dog = new Dog("Fido", 3);

dog.eat();

}

}

Output:

The animal is eating.

The dog is eating dog food.

13. Single type parameter generic class

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> intBox = new Box<>(42);

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

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

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

}

}

Output:

Integer value: 42

String value: Hello, World!

14. Multiple type parameter generic class

public class Pair<T, U> {

private T first;

private U second;

public Pair(T first, U second) {

this.first = first;

this.second = second;

}

public T getFirst() {

return first;

}

public U getSecond() {

return second;

}

public static void main(String[] args) {

Pair<Integer, String> intStringPair = new Pair<>(42, "Hello, World!");

Pair<String, Double> stringDoublePair = new Pair<>("foo", 3.14);

System.out.println("First: " + intStringPair.getFirst());

System.out.println("Second: " + intStringPair.getSecond());

System.out.println("First: " + stringDoublePair.getFirst());

System.out.println("Second: " + stringDoublePair.getSecond());

}

}

Output:

First: 42

Second: Hello, World!

First: foo

Second: 3.14

15. Using generics on methods example

public class Box<T> {

private T value;

public void setValue(T value) {

this.value = value;

}

public T getValue() {

return value;

}

public <U> void printValues(U value1, U value2) {

System.out.println("Value 1: " + value1);

System.out.println("Value 2: " + value2);

}

public static void main(String[] args) {

Box<Integer> intBox = new Box<>();

intBox.setValue(42);

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

stringBox.setValue("Hello, World!");

intBox.printValues("foo", "bar");

stringBox.printValues(42, 42.0);

}

}

Output:

Value 1: foo

Value 2: bar

Value 1: 42

Value 2: 42.0

16. Restrict use of primitive types using generics.

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> intBox = new Box<>(42);

Box<Double> doubleBox = new Box<>(3.14);

// The following line will give a compile-time error

// Box<int> intBox2 = new Box<>(42);

// The following line will give a compile-time error

// Box<float> floatBox = new Box<>(3.14f);

}

}

Output:

Box.java:21: error: generic array creation

Box<int> intBox2 = new Box<>(42);

^

where T is a type-variable:

T extends Object declared in class Box

Box.java:22: error: generic array creation

Box<float> floatBox = new Box<>(3.14f);

^

where T is a type-variable:

T extends Object declared in class Box

2 errors

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

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 c = scanner.next().charAt(0);

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

String s = scanner.nextLine();

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

int i = scanner.nextInt();

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

float f = scanner.nextFloat();

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

double d = scanner.nextDouble();

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

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

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

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

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

}

}

Output:

Enter a character: a

Enter a string: Hello, World!

Enter an integer: 42

Enter a float: 3.14

Enter a double: 2.71828

Character: a

String: Hello, World!

Integer: 42

Float: 3.14

Double: 2.71828

18. Find System Date and Time using Date classimport java.util.Date;

public class Main {

public static void main(String[] args) {

Date date = new Date();

System.out.println("Current date and time: " + date.toString());

}

}

Output:

Current date and time: Fri Feb 17 14:25:12 IST 2023

19. Use UUID to generate a random Universally Unique Identifier

import java.util.UUID;

public class Main {

public static void main(String[] args) {

UUID uuid = UUID.randomUUID();

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

}

}

Output:

Random UUID: 35b93b7d-86f0-4e1a-a75e-f631c868e1f5

20. Java to String() and equals() method

public class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

@Override

public String toString() {

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

}

public static void main(String[] args) {

Person p1 = new Person("John Doe", 30);

System.out.println(p1); // prints "Person [name=John Doe, age=30]"

}

}

Output:

Person [name=John Doe, age=30]

b)

public class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

@Override

public boolean equals(Object obj) {

if (obj == null) {

return false;

}

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

return false;

}

Person other = (Person) obj;

if (age != other.age) {

return false;

}

if (!name.equals(other.name)) {

return false;

}

return true;

}

public static void main(String[] args) {

Person p1 = new Person("John Doe", 30);

Person p2 =new Person("John Doe", 30);

System.out.println(p1.equals(p2)); // prints "true"

}

}

Output:

True