1. class Parent {

void even() {

System.out.println("Even numbers:");

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

if (i % 2 == 0) {

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

}

}

System.out.println();

}

void odd() {

System.out.println("Odd numbers:");

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

if (i % 2 != 0) {

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

}

}

System.out.println();

}

void prime() {

System.out.println("Prime numbers:");

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

boolean isPrime = true;

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 {

void even() {

System.out.println("Even numbers:");

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

if (i % 2 == 0) {

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

}

}

System.out.println();

}

void odd() {

System.out.println("Odd numbers:");

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

if (i % 2 != 0) {

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

}

}

System.out.println();

}

void prime() {

System.out.println("Prime numbers:");

for (int i = 2; i <= 30; i++) {

boolean isPrime = true;

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. class Parent {

int a, b;

Parent() {

this.a = 10;

this.b = 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(); // calling default constructor

obj1.disp();

Parent obj2 = new Parent(20, 25); // calling parameterized constructor

obj2.disp();

}

}

3.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) {

Base baseObj = new Base();

baseObj.area(5, 0); // Output: Area of square: 25

Derived derivedObj = new Derived();

derivedObj.area(5, 10); // Output: Area of rectangle: 50

}

}

4.public class Main {

// a) Overloading by increasing and decreasing the number of parameters

static int sum(int a, int b) {

return a + b;

}

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

return a + b + c;

}

// b) Overloading by changing the data types of parameters

static double sum(double a, double b) {

return a + b;

}

// c) Overloading by interchanging the parameters

static int sum(int b, int a) {

return a + b;

}

public static void main(String[] args) {

System.out.println("Sum of 5 and 10: " + sum(5, 10));

System.out.println("Sum of 5, 10, and 15: " + sum(5, 10, 15));

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

System.out.println("Sum of 100 and 200 (interchanged parameters): " + sum(100, 200));

}

}

5.

import java.util.Arrays;

// Abstract class with 2 abstract methods

abstract class AbstractStatistics {

// Abstract methods

abstract double total(double[] numbers);

abstract double average(double[] numbers);

// Concrete methods

double mean(double[] numbers) {

return total(numbers) / numbers.length;

}

double mode(double[] numbers) {

Arrays.sort(numbers);

double mode = numbers[0];

int maxCount = 1;

int currentCount = 1;

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

if (numbers[i] == numbers[i - 1]) {

currentCount++;

} else {

if (currentCount > maxCount) {

maxCount = currentCount;

mode = numbers[i - 1];

}

currentCount = 1;

}

}

return currentCount > maxCount ? numbers[numbers.length - 1] : mode;

}

double median(double[] numbers) {

Arrays.sort(numbers);

int middle = numbers.length / 2;

if (numbers.length % 2 == 0) {

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

} else {

return numbers[middle];

}

}

}

// Concrete class extending the abstract class

class ConcreteStatistics extends AbstractStatistics {

// Implementation of abstract methods

@Override

double total(double[] numbers) {

double sum = 0;

for (double num : numbers) {

sum += num;

}

return sum;

}

@Override

double average(double[] numbers) {

return total(numbers) / numbers.length;

}

}

public class Main {

public static void main(String[] args) {

// Creating an instance of the concrete class

ConcreteStatistics statistics = new ConcreteStatistics();

// Example usage

double[] data = {10, 20, 30, 40, 50};

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

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

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

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

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

}

}

7.

interface ArithmeticOperations {

double add(double a, double b);

double sub(double a, double b);

double mul(double a, double b);

double div(double a, double b);

}

// Class implementing the interface

class Calculator implements ArithmeticOperations {

@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 ArithmeticException("Cannot divide by zero");

}

return a / b;

}

}

public class Main {

public static void main(String[] args) {

// Creating an instance of the class implementing the interface

Calculator calculator = new Calculator();

// Example usage

double x = 10;

double y = 5;

System.out.println("Addition: " + calculator.add(x, y));

System.out.println("Subtraction: " + calculator.sub(x, y));

System.out.println("Multiplication: " + calculator.mul(x, y));

System.out.println("Division: " + calculator.div(x, y));

}

}

8.

interface Tree {

void fruits(String fruitType, int quantity);

void leaves(String leafColor, int leafCount, boolean isDeciduous);

void flowers(String flowerColor, int petalCount, String fragrance, boolean hasNectar);

}

// Branch1 class implementing Tree interface

class Branch1 implements Tree {

@Override

public void fruits(String fruitType, int quantity) {

System.out.println("Branch 1 produces " + quantity + " " + fruitType + "(s)");

}

@Override

public void leaves(String leafColor, int leafCount, boolean isDeciduous) {

String type = isDeciduous ? "deciduous" : "evergreen";

System.out.println("Branch 1 has " + leafCount + " " + leafColor + " " + type + " leaves");

}

@Override

public void flowers(String flowerColor, int petalCount, String fragrance, boolean hasNectar) {

String nectar = hasNectar ? "with nectar" : "without nectar";

System.out.println("Branch 1 has flowers with " + petalCount + " " + flowerColor + " petals, fragrance: " + fragrance + ", " + nectar);

}

}

// Branch2 class implementing Tree interface

class Branch2 implements Tree {

@Override

public void fruits(String fruitType, int quantity) {

System.out.println("Branch 2 produces " + quantity + " " + fruitType + "(s)");

}

@Override

public void leaves(String leafColor, int leafCount, boolean isDeciduous) {

String type = isDeciduous ? "deciduous" : "evergreen";

System.out.println("Branch 2 has " + leafCount + " " + leafColor + " " + type + " leaves");

}

@Override

public void flowers(String flowerColor, int petalCount, String fragrance, boolean hasNectar) {

String nectar = hasNectar ? "with nectar" : "without nectar";

System.out.println("Branch 2 has flowers with " + petalCount + " " + flowerColor + " petals, fragrance: " + fragrance + ", " + nectar);

}

}

public class Main {

public static void main(String[] args) {

// Creating instances of Branch1 and Branch2

Branch1 branch1 = new Branch1();

Branch2 branch2 = new Branch2();

// Using methods of Branch1

branch1.fruits("Apple", 10);

branch1.leaves("Green", 100, true);

branch1.flowers("Red", 5, "Sweet", true);

// Using methods of Branch2

branch2.fruits("Orange", 20);

branch2.leaves("Yellow", 80, false);

branch2.flowers("White", 8, "Fragrant", false);

}

}

10.

public class Example {

// a) Final variable

final int finalVariable = 10;

// b) Final method

public final void finalMethod() {

System.out.println("Final method called");

}

// c) Final class

final class FinalClass {

void display() {

System.out.println("Final class method called");

}

}

public static void main(String[] args) {

Example example = new Example();

// Accessing final variable

System.out.println("Final variable: " + example.finalVariable);

// Calling final method

example.finalMethod();

// Creating an instance of the final class

FinalClass finalObj = example.new FinalClass();

finalObj.display();

}

}

9.

public class Example {

// a) Static variable

static int staticVariable = 10;

// b) Static method

public static void staticMethod() {

System.out.println("Static method called");

}

// c) Static block

static {

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

}

// d) Static nested class

static class StaticNestedClass {

void display() {

System.out.println("Static nested class method called");

}

}

public static void main(String[] args) {

// Accessing static variable

System.out.println("Static variable: " + staticVariable);

// Calling static method

staticMethod();

// Accessing static nested class

StaticNestedClass nestedObject = new StaticNestedClass();

nestedObject.display();

}

}

11.

public class Example {

// Instance variable

int number;

// Constructor with one parameter

public Example(int number) {

// a) Referencing instance variable

this.number = number;

}

// Constructor invoking another constructor

public Example() {

// b) Invoking another constructor

this(100); // Calls the constructor with one parameter, passing 100 as the argument

}

// Method to pass current object as a parameter

public void passObject(Example obj) {

System.out.println("Object number: " + obj.number);

}

// Method returning current object

public Example returnObject() {

// c) Returning current object

return this;

}

public static void main(String[] args) {

Example obj1 = new Example(); // Invokes constructor with no parameters

System.out.println("Object 1 number: " + obj1.number); // Prints 100

Example obj2 = new Example(200); // Invokes constructor with one parameter

System.out.println("Object 2 number: " + obj2.number); // Prints 200

obj1.passObject(obj2); // Passes obj2 as a parameter

obj2.returnObject(); // Returns obj2

}

}

12.

class Superclass {

String message = "Hello from superclass";

Superclass() {

System.out.println("Superclass constructor");

}

void display() {

System.out.println("Displaying from superclass");

}

}

class Subclass extends Superclass {

String message = "Hello from subclass";

Subclass() {

// b) Calling superclass constructor

super();

System.out.println("Subclass constructor");

}

void display() {

// c) Invoking superclass method

super.display();

System.out.println("Displaying from subclass");

}

void showMessage() {

// a) Accessing superclass members

System.out.println("Superclass message: " + super.message);

System.out.println("Subclass message: " + message);

}

}

public class Main {

public static void main(String[] args) {

Subclass subclassObj = new Subclass();

System.out.println("--------------------");

subclassObj.display();

System.out.println("--------------------");

subclassObj.showMessage();

}

}

13.

public class Box<T> {

private T data;

public Box(T data) {

this.data = data;

}

public T getData() {

return data;

}

public void setData(T data) {

this.data = data;

}

public static void main(String[] args) {

// Creating a Box for Integer

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

System.out.println("Integer Value: " + intBox.getData());

// Creating a Box for String

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

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

}

}

14.

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) {

// Creating a Pair for Integer and String

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

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

// Creating a Pair for String and Double

Pair<String, Double> pair2 = new Pair<>("PI", 3.14159);

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

}

}

15.

public class Example {

// Generic method to compare two objects

public static <T extends Comparable<T>> int compare(T obj1, T obj2) {

return obj1.compareTo(obj2);

}

// Generic method to print an array

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

for (E element : array) {

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

}

System.out.println();

}

public static void main(String[] args) {

// Using the compare method with different data types

System.out.println("Comparison result (Integer): " + compare(10, 20));

System.out.println("Comparison result (String): " + compare("apple", "banana"));

// Using the printArray method with different data types

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

String[] stringArray = {"apple", "banana", "cherry"};

System.out.print("Integer Array: ");

printArray(intArray);

System.out.print("String Array: ");

printArray(stringArray);

}

}

16.

public class Example<T extends Number> {

private T value;

public Example(T value) {

this.value = value;

}

public T getValue() {

return value;

}

public void setValue(T value) {

this.value = value;

}

public static void main(String[] args) {

Example<Integer> intExample = new Example<>(10);

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

// This will give a compile-time error as String is not a subclass of Number

// Example<String> stringExample = new Example<>("Hello");

}

}

17.

import java.util.Scanner;

import java.util.Calendar;

import java.util.Date;

public class InputExample {

public static void main(String[] args) {

// Get the current time

Calendar calendar = Calendar.getInstance();

Date currentTime = calendar.getTime();

int hour = calendar.get(Calendar.HOUR\_OF\_DAY);

int minute = calendar.get(Calendar.MINUTE);

// Check if the current time is within the specified range (01:00 PM to 02:00 PM)

if (hour == 13 && minute >= 0 && minute <= 59) {

Scanner scanner = new Scanner(System.in);

// Get char input

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

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

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

// Get String input

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

String str = scanner.next();

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

// Get int input

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

int numInt = scanner.nextInt();

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

// Get float input

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

float numFloat = scanner.nextFloat();

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

// Get double input

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

double numDouble = scanner.nextDouble();

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

// Close the scanner

scanner.close();

} else {

System.out.println("Input is only accepted between 01:00 PM to 02:00 PM.");

}

}

}

18.

import java.util.Date;

public class CurrentDateTimeExample {

public static void main(String[] args) {

// Create a new Date object

Date currentDate = new Date();

// Print the current date and time

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

}

}

19.

import java.util.UUID;

public class UUIDExample {

public static void main(String[] args) {

// Generate a random UUID

UUID uuid = UUID.randomUUID();

// Print the generated UUID

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

}

}

20. public class MyClass {

private int id;

private String name;

public MyClass(int id, String name) {

this.id = id;

this.name = name;

}

@Override

public String toString() {

return "MyClass{id=" + id + ", name='" + name + "'}";

}

}