**1. Write a program in Java to perform implicit and explicit type casting**

public class TypeCastingDemo {

public static void main(String[] args) {

int intValue = 10;

double doubleValue = intValue; // Automatically converts int to double

System.out.println("Implicit Type Casting (Widening):");

System.out.println("int to double: " + doubleValue);

double doubleNum = 15.75;

int intNum = (int) doubleNum; // Manually convert double to int (data loss)

System.out.println("\nExplicit Type Casting (Narrowing):");

System.out.println("double to int: " + intNum);

double doubleValue2 = 20.49;

int roundedInt = (int) Math.round(doubleValue2); // Round and convert

System.out.println("\nExplicit Type Casting with Rounding:");

System.out.println("double to int (rounded): " + roundedInt);

}

}

**2.Write a program in Java to verify the working of access modifiers**

public class AccessModifiersDemo {

public int publicVar = 10;

private int privateVar = 20;

protected int protectedVar = 30;

int defaultVar = 40;

public AccessModifiersDemo() {

System.out.println("Inside the AccessModifiersDemo constructor");

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

System.out.println("privateVar: " + privateVar**);**

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

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

}

public void publicMethod() {

System.out.println("Inside the publicMethod");

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

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

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

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

}

private void privateMethod() {

System.out.println("Inside the privateMethod");

}

protected void protectedMethod() {

System.out.println("Inside the protectedMethod");

}

void defaultMethod() {

System.out.println("Inside the defaultMethod");

}

public static void main(String[] args) {

AccessModifiersDemo demo = new AccessModifiersDemo();

System.out.println("\nAccessing members fromoutside the class:");

System.out.println("publicVar from outside: " + demo.publicVar);

System.out.println("\nAccessing methods from outside the class:");

demo.publicMethod();;

}

}

**3.Write a program to demonstrate the while loop**

package javaprograms;

public class Whileloop {

public static void main(String[] args) {

// TODO Auto-generated method stub

int i=1;

while(i<=10)

{

System.out.println(i);

i++;

}

System.out.println("printed values from 1 to 10");

}

}

**4. Write a program to demonstrate the do while loop**

package javaprograms;

public class Dowhileloop {

public static void main(String[] args) {

// TODO Auto-generated method stub

int i=1;

do

{

System.out.println("today is wednesday");

i++;

}while(i<=4);

System.out.println("printed the meswsage");

}

}

**5.Write a program to demonstrate the for loop**

package javaprograms;

public class Forloop {

public static void main(String[] args) {

// TODO Auto-generated method stub

int i;

for(i=100;i>=50;i--)

{

System.out.println(i);

}

System.out.println("out of loop");

}

}

**6. Demonstrate the Classes, Objects, and Constructors**

public class Person {

String name;

int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public void displayInfo() {

System.out.println("Name: " + name);

System.out.println("Age: " + age);

}

public static void main(String[] args) {

Person person1 = new Person("Alice", 30);

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

System.out.println("Person 1:");

person1.displayInfo();

System.out.println("\nPerson 2:");

person2.displayInfo();

}

}

**7. Demonstrate types of inheritance**

Single inheritance:

class Animal {

void eat() {

System.out.println("The animal eats food.");

}

}

class Dog extends Animal {

void bark() {

System.out.println("The dog barks.");

}

}

public class SingleInheritanceDemo {

public static void main(String[] args) {

Dog dog = new Dog();

dog.eat(); // Inherited from Animal class

dog.bark(); // Defined in Dog class

}

}

Multiple inheritance:

interface Swim {

void swim();

}

interface Fly {

void fly();

}

class Bird implements Swim, Fly {

public void swim() {

System.out.println("The bird swims.");

}

public void fly() {

System.out.println("The bird flies.");

}

}

public class MultipleInheritanceDemo {

public static void main(String[] args) {

Bird bird = new Bird();

bird.swim(); bird.fly();

}

}

**Multilevel inheritance:**

class Animal {

void eat() {

System.out.println("The animal eats food.");

}

}

class Dog extends Animal {

void bark() {

System.out.println("The dog barks.");

}

}

class GermanShepherd extends Dog {

void guard() {

System.out.println("The German Shepherd guards.");

}

}

public class MultilevelInheritanceDemo {

public static void main(String[] args) {

GermanShepherd shepherd = new GermanShepherd();

shepherd.eat(); shepherd.bark();

shepherd.guard(); }

}

Hirarcichal inheritance:

class Vehicle {

void start() {

System.out.println("Vehicle starts.");

}

}

class Car extends Vehicle {

void drive() {

System.out.println("Car drives.");

}

}

class Bike extends Vehicle {

void ride() {

System.out.println("Bike rides.");

}

}

public class HierarchicalInheritanceDemo {

public static void main(String[] args) {

Car car = new Car();

Bike bike = new Bike();

car.start();

car.drive();

bike.start(); bike.ride(); }

}

**7.Writing a program in Java to verify implementations of collection**

package collection;

import java.util.ArrayList;

public class Arraylistdemo {

public static void main(String[] args) {

// TODO Auto-generated method stub

ArrayList <String> cities=new ArrayList<>();

cities.add("london");

cities.add("paris");

cities.add(2,"new delhi");

cities.add("mumbai");

System.out.println(cities.size());

for(String t:cities)

{

System.out.println(t);

}

}

}

package collection;

import java.util.Iterator;

import java.util.LinkedList;

public class Linkedlistdemo

{

public static void main(String[] args)

{

// TODO Auto-generated method stub

LinkedList<String> cities = new LinkedList<>();

cities.add("london");

cities.add("paris");

cities.add(2,"new delhi");

cities.add("mumbai");

System.out.println(cities.size());

Iterator itr=cities.iterator();

while(itr.hasNext())

{

System.out.println(itr.next());

}

System.out.println(cities.get(1));

System.out.println(cities.contains("mumbai"));

}

}

**8.Writing a program to perform try-catch block**

public class TryCatchDemo {

public static void main(String[] args) {

try {

int result = divide(10, 0);

System.out.println("Result: " + result); } catch (ArithmeticException e) {

System.out.println("An exception occurred: " + e.getMessage());

}

System.out.println("Program continues after the try-catch block.");

}

public static int divide(int dividend, int divisor) {

return dividend / divisor;

}

}

**9. Writing code for throwand throws keyword**

public class ThrowDemo {

public static void main(String[] args) {

try {

validateAge(15);

} catch (IllegalArgumentException e) {

System.out.println("Caught an exception: " + e.getMessage());

}

}

public static void validateAge(int age) {

if (age < 18) {

throw new IllegalArgumentException("Age must be 18 or older.");

}

System.out.println("Age is valid.");

}

}

public class ThrowsDemo {

public static void main(String[] args) {

try {

callMethod();

} catch (IOException e) {

System.out.println("Caught an IOException: " + e.getMessage());

}

}

public static void callMethod() throws IOException {

throw new IOException("An IOException occurred.");

}

}

**10.Writing code for a try block with parameters**

public class TryBlockWithParametersDemo {

public static void main(String[] args) {

try {

int dividend = 10;

int divisor = 0;

divideAndPrintResult(dividend, divisor);

} catch (ArithmeticException e) {

System.out.println("An exception occurred: " + e.getMessage());

}

}

public static void divideAndPrintResult(int dividend, int divisor) {

if (divisor == 0) {

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

}

int result = dividend / divisor;

System.out.println("Result of division: " + result);

}

}

**11. Writing code for multiple catch blocks**

public class MultipleCatchBlocksDemo {

public static void main(String[] args) {

try {

int[] numbers = {1, 2, 3};

int result = divide(numbers, 0);

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("ArithmeticException: " + e.getMessage());

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("ArrayIndexOutOfBoundsException: " + e.getMessage());

} catch (Exception e) {

System.out.println("Generic Exception: " + e.getMessage());

}

}

public static int divide(int[] numbers, int index) {

try {

return numbers[index] / 0;

} catch (ArithmeticException e) {

throw e; // Re-throw the ArithmeticException

} catch (ArrayIndexOutOfBoundsException e) {

throw e;

}

}

}

**12. Writing code for finally{} block**

import java.io.FileReader;

import java.io.IOException;

public class FinallyBlockDemo {

public static void main(String[] args) {

FileReader reader = null;

try {

reader = new FileReader("example.txt");

System.out.println("File opened and read successfully.");

} catch (IOException e) {

System.out.println("An IOException occurred: " + e.getMessage());

} finally {

try {

if (reader != null) {

reader.close();

System.out.println("File reader closed.");

}

} catch (IOException e) {

System.out.println("Error while closing the file: " + e.getMessage());

}

}

}

}