**1.) Introduction to Java**

**1.1) Install JDK and set up environment variables.**

1.Download from Oracle: <https://www.oracle.com/java/technologies/javase-downloads.html>

2.Install JDK (Java Development Kit).

3.Set environment variables:

* JAVA\_HOME = path\_to\_jdk
* Add JAVA\_HOME\bin to Path.

**1.2) Write a simple "Hello World" Java program.**

public class HelloWorld {

public static void main(String[] args) {

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

}

}

**1.3) Compile and run the program using command-line tools (javac, java).**

javac HelloWorld.java // Compiles code and creates HelloWorld.class

java HelloWorld // Runs the bytecode

**2.) Data Types, Variables, and Operators.**

**2.1) Write a program to demonstrate the use of different data types.**

int age = 30;

float salary = 45678.90f;

char grade = 'A';

boolean passed = true;

String name = "Alice";

Primitive Types: int, float, char, boolean, double, byte, short, long.

Reference Types: String, Arrays, Objects.

**2.2) Create a calculator using arithmetic and relational operators.**

int a = 10, b = 5;

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

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

Arithmetic: +, -, \*, /, %

Relational: >, <, ==, !=, >=, <=

**2.3) Demonstrate type casting (explicit and implicit).**

int a = 10;

double b = a; // Implicit (widening)

double c = 9.5;

int d = (int) c; // Explicit (narrowing)

**3.) Control Flow Statements.**

**3.1) Write a program to find if a number is even or odd using an if-else statement.**

int n = 4;

if (n % 2 == 0)

{

System.out.println("Even");

}

else{

System.out.println("Odd");

}

**3.2) Implement a simple menu-driven program using a switch-case.**

int choice = 2;

switch (choice) {

case 1: System.out.println("Add"); break;

case 2: System.out.println("Subtract"); break;

default: System.out.println("Invalid");

}

**3.3) Write a program to display the Fibonacci series using a loop.**

int n = 10, a = 0, b = 1;

for (int i = 0; i < n; i++) {

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

int c = a + b;

a = b;

b = c;

}

**4.) Classes and Objects.**

**4.1) Create a class Student with attributes (name, age) and a method to display the details.**

class Student {

String name;

int age;

void display() {

System.out.println(name + " " + age);

}

}

**4.2) Create multiple constructors in a class and demonstrate constructor overloading.**

class Student {

Student() {

System.out.println("Default");

}

Student(String name) {

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

}

**4.3) Implement a simple class with getters and setters for encapsulation.**

class Student {

private int age;

public void setAge(int a) { age = a; }

public int getAge() { return age; }

}

**5.) Methods in Java.**

**5.1) Write a program to find the maximum of three numbers using a method.**

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

return (a > b && a > c) ? a : (b > c ? b : c);

}

**5.2) Implement method overloading by creating methods for different data types.**

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

**5.3) Create a class with static variables and methods to demonstrate their use.**

class Example {

static int count = 0;

static void increment() {

count++;

}

}

**6.) Object-Oriented Programming (OOPs) Concepts.**

**6.1) Write a program demonstrating single inheritance.**

class Animal {

void eat() {

System.out.println("Eat");

}

}

class Dog extends Animal {}

**6.2) Create a class hierarchy and demonstrate multilevel inheritance.**

class A {

void showA() {}

}

class B extends A {

void showB() {}

}

class C extends B {

void showC() {}

}

**6.3) Implement method overriding to show polymorphism in action.**

class Parent {

void message() {

System.out.println("Parent");

}

}

class Child extends Parent {

void message() {

System.out.println("Child");

}

}

**7.) Constructors and Destructors.**

**7.1) Write a program to create and initialize an object using a parameterized constructor.**

class Person {

String name;

Person(String n) { name = n; }

}

**7.2) Demonstrate constructor overloading by passing different types of parameters.**

Person() {}

Person(String name) {}

Person(String name, int age) {}

**8.) Arrays and Strings.**

**8.1) Write a program to perform matrix addition and subtraction using 2D arrays.**

import java.util.Scanner;

public class MatrixOperations {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int[][] b = {{5, 6}, {7, 8}};

int[][] sum = new int[2][2];

int[][] diff = new int[2][2];

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

sum[i][j] = a[i][j] + b[i][j];

diff[i][j] = a[i][j] - b[i][j];

}

}

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

printMatrix(sum);

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

printMatrix(diff);

}

static void printMatrix(int[][] m) {

for (int[] row : m) {

for {

(int val : row) System.out.print(val + " ");

System.out.println();

}

}

}

}

**8.2) Create a program to reverse a string and check for palindromes.**

import java.util.Scanner;

public class ReversePalindrome {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

String str = sc.nextLine();

String rev = new StringBuilder(str).reverse().toString();

System.out.println("Reversed: " + rev);

if {

(str.equals(rev))

System.out.println("Palindrome");

}

else{

System.out.println("Not a Palindrome");

}

}

}

**8.3) Implement string comparison using equals() and compareTo() methods.**

public class StringCompare {

public static void main(String[] args) {

String s1 = "Hello", s2 = "World", s3 = "Hello";

System.out.println("s1 equals s2? " + s1.equals(s2)); //false

System.out.println("s1 equals s3? " + s1.equals(s3)); // true

System.out.println("s1 compareTo s2: " + s1.compareTo(s2)); // < 0

System.out.println("s1 compareTo s3: " + s1.compareTo(s3)); // 0

}

}

**9.) Inheritance and Polymorphism.**

**9.1) Write a program that demonstrates inheritance using extends keyword.**

class Parent {

void display() {

System.out.println("This is the Parent class.");

}

}

class Child extends Parent {

void show() {

System.out.println("This is the Child class.");

}

}

public class Main {

public static void main(String[] args) {

Child obj = new Child();

obj.display(); // inherited from Parent

obj.show(); // Child's own method

}

}

**Output:**

This is the Parent class.

This is the Child class.

**9.2) Implement runtime polymorphism by overriding methods in the child class.**

class Animal {

void sound() {

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal {

@Override

void sound() {

System.out.println("Dog barks");

}

}

public class Main {

public static void main(String[] args) {

Animal obj = new Dog(); // Parent reference, Child object

obj.sound(); // Calls Dog's sound() at runtime

}

}

**Output:**

Dog barks

**9.3) Use the super keyword to call the parent class constructor and methods.**

class Parent {

Parent() {

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

}

void greet() {

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

}

}

class Child extends Parent {

Child() {

super(); // Call Parent constructor

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

}

void greet() {

super.greet(); // Call Parent method

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

}

}

public class Main {

public static void main(String[] args) {

Child obj = new Child();

obj.greet();

}

}

**Output:**

Parent constructor

Child constructor

Hello from Parent

Hello from Child

**10.) Interfaces and Abstract Classes.**

**10.1) Create an abstract class and implement its methods in a subclass.**

abstract class Shape {

abstract void draw(); // abstract method

void display() { // concrete method

System.out.println("This is a shape.");

}

}

class Circle extends Shape {

@Override

void draw() {

System.out.println("Drawing a circle.");

}

}

public class Main {

public static void main(String[] args) {

Circle c = new Circle();

c.display();

c.draw();

}

}

**Output:**

This is a shape.

Drawing a circle.

**10.2) Write a program that implements multiple interfaces in a single class.**

interface Printable {

void print();

}

interface Showable {

void show();

}

class Demo implements Printable, Showable {

public void print() {

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

}

public void show() {

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

}

}

public class Main {

public static void main(String[] args) {

Demo d = new Demo();

d.print();

d.show();

}

}

**Output:**

Printing...

Showing...

**10.3) Implement an interface for a real-world example,such as a payment gateway.**

interface PaymentGateway {

void processPayment(double amount);

}

class PayPal implements PaymentGateway {

public void processPayment(double amount) {

System.out.println("Processing $" + amount + " via PayPal.");

}

}

public class Main {

public static void main(String[] args) {

PaymentGateway payment = new PayPal();

payment.processPayment(150.75);

}

}

**Output:**

Processing $150.75 via PayPal.

**11.) Packages and Access Modifiers.**

**11.1) Create a user-defined package and import it into another program.**

**mypack/Message.java**

package mypack;

public class Message {

public void greet() {

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

}

}

**TestPackage.java**

import mypack.Message;

public class TestPackage {

public static void main(String[] args) {

Message msg = new Message();

msg.greet();

}

}

**How to Compile & Run:**

javac -d . mypack/Message.java

javac TestPackage.java

java TestPackage

**Output:**

Hello from mypack!

**11.2) Demonstrate the use of different access modifiers within the same package and across different packages.**

**pack1/AccessDemo.java**

package pack1;

public class AccessDemo {

public String publicVar = "Public Variable";

protected String protectedVar = "Protected Variable";

String defaultVar = "Default Variable";

private String privateVar = "Private Variable";

public void show() {

System.out.println("Inside pack1.AccessDemo");

System.out.println(publicVar);

System.out.println(protectedVar);

System.out.println(defaultVar);

System.out.println(privateVar);

}

}

**pack1/SamePackage.java**

package pack1;

public class SamePackage {

public static void main(String[] args) {

AccessDemo obj = new AccessDemo();

obj.show();

System.out.println(obj.publicVar);

System.out.println(obj.protectedVar);

System.out.println(obj.defaultVar);

// System.out.println(obj.privateVar); // Not accessible

}

}

**pack2/DifferentPackage.java**

package pack2;

**import pack1.AccessDemo;**

public class DifferentPackage {

public static void main(String[] args) {

AccessDemo obj = new AccessDemo();

System.out.println(obj.publicVar); //Accessible

// System.out.println(obj.protectedVar); //Not accessible

// System.out.println(obj.defaultVar); //Not accessible

// System.out.println(obj.privateVar); //Not accessible

}

}

**Compile & Run:**

javac -d . pack1/AccessDemo.java

javac -d . pack1/SamePackage.java

javac -d . pack2/DifferentPackage.java

java pack1.SamePackage

java pack2.DifferentPackage

**Output (SamePackage):**

Inside pack1.AccessDemo

Public Variable

Protected Variable

Default Variable

Private Variable

Public Variable

Protected Variable

Default Variable

**Output (DifferentPackage):**

Public Variable

**12.) Exception Handling.**

**12.1) Write a program to demonstrate exception handling using try-catch-finally.**

public class TryCatchFinally {

public static void main(String[] args) {

try {

int result = 10 / 0; // Division by zero

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

} catch (ArithmeticException e) {

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

} finally {

System.out.println("Finally block always executes.");

}

}

}

**Output:**

Caught an exception: java.lang.ArithmeticException: // by zero

Finally block always executes.

**12.2) Implement multiple catch blocks for different types of exceptions.**

public class MultipleCatch {

public static void main(String[] args) {

try {

String str = null;

System.out.println(str.length()); //NullPointerException

int arr[] = new int[5];

arr[10] = 50; // ArrayIndexOutOfBoundsException

} catch (NullPointerException e) {

System.out.println("Caught NullPointerException: " + e);

} catch (ArrayIndexOutOfBoundsException e) {

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

} catch (Exception e) {

System.out.println("Caught General Exception: " + e);

}

}

}

**Output:**

Caught NullPointerException: java.lang.NullPointerException

**12.3) Create a custom exception class and use it in your program.**

class MyException extends Exception {

public MyException(String message) {

super(message);

}

}

public class CustomExceptionDemo {

static void validate(int age) throws MyException {

if (age < 18){

throw new MyException("Not eligible to vote");

}else{

System.out.println("Eligible to vote");

}

}

public static void main(String[] args) {

try {

validate(16);

} catch (MyException e) {

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

}

}

}

**Output:**

Caught custom exception: Not eligible to vote

**13.) Multithreading.**

**13.1) Write a program to create and run multiple threads using the Thread class.**

class MyThread extends Thread {

public void run() {

for (int i = 1; i <= 5; i++) {

System.out.println(Thread.currentThread().getName() + " - " + i);

}

}

}

public class MultiThreadDemo {

public static void main(String[] args) {

MyThread t1 = new MyThread();

MyThread t2 = new MyThread();

t1.start();

t2.start();

}

}

**Sample Output:**

Thread-0 - 1

Thread-1 - 1

Thread-0 - 2

Thread-1 - 2

**13.2) Implement thread synchronization using synchronized blocks or methods.**

class Table {

synchronized void printTable(int n) { // synchronized method

for (int i = 1; i <= 5; i++) {

System.out.println(n + " x " + i + " = " + (n \* i));

try {

Thread.sleep(100);

} catch (Exception e) {}

}

}

}

class MyThread1 extends Thread {

Table t;

MyThread1(Table t) { this.t = t; }

public void run() { t.printTable(5); }

}

class MyThread2 extends Thread {

Table t;

MyThread2(Table t) { this.t = t; }

public void run() { t.printTable(10); }

}

public class SyncDemo {

public static void main(String[] args) {

Table obj = new Table();

MyThread1 t1 = new MyThread1(obj);

MyThread2 t2 = new MyThread2(obj);

t1.start();

t2.start();

}

}

**Sample Output (synchronized):**

5 x 1 = 5

5 x 2 = 10

...

10 x 1 = 10

10 x 2 = 20

...

**13.3) Use inter-thread communication methods like wait(), notify(), and notifyAll().**

class Shared {

boolean flag = false;

synchronized void produce() {

System.out.println("Producer started...");

flag = true;

System.out.println("Producer waiting...");

try {

wait();

} catch (InterruptedException e) {}

System.out.println("Producer resumed...");

}

synchronized void consume() {

System.out.println("Consumer started...");

if (!flag) {

try {

Thread.sleep(1000);

} catch (Exception e) {}

}

notify();

System.out.println("Consumer notified producer.");

}

}

public class InterThreadDemo {

public static void main(String[] args) {

Shared s = new Shared();

Thread producer = new Thread(() -> s.produce());

Thread consumer = new Thread(() -> s.consume());

producer.start();

consumer.start();

}

}

**Sample Output:**

Producer started...

Producer waiting...

Consumer started...

Consumer notified producer.

Producer resumed...

**14.) File Handling.**

**14.1) Write a program to read and write content to a file using FileReader and FileWriter.**

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class FileReaderWriterExample {

public static void main(String[] args) {

String fileName = "example.txt";

try {

// Writing to the file

FileWriter writer = new FileWriter(fileName);

writer.write("Hello, this is a test file.\nWelcome to FileReader and FileWriter example.");

writer.close();

System.out.println("File written successfully.");

// Reading from the file

FileReader reader = new FileReader(fileName);

int character;

System.out.println("Reading file content:");

while ((character = reader.read()) != -1) {

System.out.print((char) character);

}

reader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

**14.2) Implement a program that reads a file line by line using BufferedReader.**

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class BufferedReaderExample {

public static void main(String[] args) {

String fileName = "example.txt";

try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {

String line;

System.out.println("Reading file line by line:");

while ((line = br.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**14.3) Create a program that demonstrates object serialization and deserialization.**

import java.io.\*;

class Student implements Serializable {

private static final long serialVersionUID = 1L;

String name;

int age;

Student(String name, int age) {

this.name = name;

this.age = age;

}

void display() {

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

}

}

public class SerializationExample {

public static void main(String[] args) {

String fileName = "student.ser";

// Serialize object

Student student = new Student("Alice", 20);

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fileName))) {

oos.writeObject(student);

System.out.println("Object serialized successfully.");

} catch (IOException e) {

e.printStackTrace();

}

// Deserialize object

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fileName))) {

Student deserializedStudent = (Student) ois.readObject();

System.out.println("Object deserialized successfully.");

deserializedStudent.display();

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

}

**15.) Collections Framework.**

**15.1) Write a program that demonstrates the use of an ArrayList and LinkedList.**

import java.util.ArrayList;

import java.util.LinkedList;

public class ListExample {

public static void main(String[] args) {

// Using ArrayList

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

arrayList.add("Apple");

arrayList.add("Banana");

arrayList.add("Cherry");

System.out.println("ArrayList: " + arrayList);

// Using LinkedList

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

linkedList.add("Dog");

linkedList.add("Elephant");

linkedList.add("Fox");

System.out.println("LinkedList: " + linkedList);

// Access elements

System.out.println("First element in ArrayList: " + arrayList.get(0));

System.out.println("First element in LinkedList: " + linkedList.get(0));

}

}

**15.2) Implement a program using HashSet to remove duplicate elements from a list.**

import java.util.ArrayList;

import java.util.HashSet;

public class RemoveDuplicates {

public static void main(String[] args) {

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

listWithDuplicates.add("Apple");

listWithDuplicates.add("Banana");

listWithDuplicates.add("Apple");

listWithDuplicates.add("Cherry");

listWithDuplicates.add("Banana");

System.out.println("Original List: " + listWithDuplicates);

// Remove duplicates using HashSet

HashSet<String> uniqueSet = new HashSet<>(listWithDuplicates);

ArrayList<String> listWithoutDuplicates = new ArrayList<>(uniqueSet);

System.out.println("List after removing duplicates: " + listWithoutDuplicates);

}

}

**15.3) Create a HashMap to store and retrieve key-value pairs.**

import java.util.HashMap;

public class HashMapExample {

public static void main(String[] args) {

HashMap<Integer, String> map = new HashMap<>();

// Adding key-value pairs

map.put(1, "Apple");

map.put(2, "Banana");

map.put(3, "Cherry");

System.out.println("HashMap: " + map);

// Retrieve value by key

System.out.println("Value for key 2: " + map.get(2));

// Iterate through HashMap

System.out.println("All key-value pairs:");

for (Integer key : map.keySet()) {

System.out.println("Key: " + key + ", Value: " + map.get(key));

}

}

}

**16.) Java Input/Output (I/O).**

**16.1) Write a program to read input from the console using Scanner.**

import java.util.Scanner;

public class ConsoleInputExample {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter your name: ");

String name = scanner.nextLine();

System.out.print("Enter your age: ");

int age = scanner.nextInt();

System.out.println("Hello, " + name + "! You are " + age + " years old.");

scanner.close();

}

}

**16.2) Implement a file copy program using FileInputStream and FileOutputStream.**

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class FileCopyExample {

public static void main(String[] args) {

String sourceFile = "source.txt";

String destFile = "destination.txt";

try (FileInputStream fis = new FileInputStream(sourceFile);

FileOutputStream fos = new FileOutputStream(destFile)) {

int byteContent;

while ((byteContent = fis.read()) != -1) {

fos.write(byteContent);

}

System.out.println("File copied successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**16.3) Create a program that reads from one file and writes the content to another file.**

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class FileReadWriteExample {

public static void main(String[] args) {

String inputFile = "input.txt";

String outputFile = "output.txt";

try (BufferedReader reader = new BufferedReader(new FileReader(inputFile));

BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile))) {

String line;

while ((line = reader.readLine()) != null) {

writer.write(line);

writer.newLine();

}

System.out.println("Content copied from " + inputFile + " to " + outputFile);

} catch (IOException e) {

e.printStackTrace();

}

}

}