# Java Programming Assignment

## Section 1: Java Data Types

1. What are the different primitive data types available in Java?

**Primitive Data Type:**These are the basic building blocks that store simple values directly in memory. Examples of primitive data types are

* **boolean**
* **char**
* **byte**
* **short**
* **int**
* **long**
* **float**
* **double**

1. Explain the difference between primitive and non-primitive data types in Java.
2. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
3. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Primitive data types are a set of basic data types from which all other data types are constructed.

Non-primitive data types are also called ‘reference variables’ or ‘object references’ as they reference a memory location where data is stored. Non-primitive data types in Java are not predefined.

3. Write a Java program that demonstrates the use of all primitive data types.

public class PrimitiveTypesDemo {

public static void main(String[] args) {

byte b = 100;

short s = 20000;

int i = 100000;

long l = 10000000000L;

float f = 10.5f;

double d = 20.123456789;

char ch = 'A';

boolean flag = true;

System.out.println("byte: " + b);

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

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

System.out.println("long: " + l);

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

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

System.out.println("char: " + ch);

System.out.println("boolean: " + flag);

}

}

Output:

byte: 100

short: 20000

int: 100000

long: 10000000000

float: 10.5

double: 20.123456789

char: A

boolean: true

1. What is type casting? Provide an example of implicit and explicit casting in Java.

**Type Casting** is converting a variable from one data type to another.

**Two Types:**

1. **Implicit Casting (Widening)** – Performed automatically when converting from smaller to larger type.  
   Example:

int num = 10;

double d = num; // int → double

System.out.println(d); // Output: 10.0

**Explicit Casting (Narrowing)** – Done manually when converting from larger to smaller type.  
Example:

double val = 9.78;

int num = (int) val; // double → int

System.out.println(num); // Output: 9

1. What is the default value of each primitive data type in Java?

| **Data Type** | **Default Value** |
| --- | --- |
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0L |
| float | 0.0f |
| double | 0.0d |
| char | '\u0000' (null character) |
| boolean | false |

## Section 2: Java Control Statements

* 1. What are control statements in Java? List the types with examples.

**Control statements** in Java are used to control the flow of execution in a program. They determine which part of the program should execute next based on conditions or loops.

**Types of Control Statements in Java:**

1. **Conditional Statements** – Decision-making
   * if statement
   * if-else statement
   * if-else-if ladder
   * switch statement
2. **Looping Statements** – Iteration
   * for loop
   * while loop
   * do-while loop
   * Enhanced for loop (for-each)
3. **Jump Statements** – Alter normal flow
   * break (exit loop/switch)
   * continue (skip current iteration)
   * return (exit from method)

**example 1:**

if (age >= 18) {

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

} else {

System.out.println("Not eligible to vote");

}

**Eample 2:**

import java.util.Scanner;

public class DecisionDemo {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int age = sc.nextInt();

if (age >= 18) {

System.out.println("You are eligible to vote.");

} else {

System.out.println("You are NOT eligible to vote.");

}

System.out.print("Enter a day number (1-3): ");

int day = sc.nextInt();

switch (day) {

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

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

case 3: System.out.println("Wednesday"); break;

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

}

sc.close();

}

}

* 1. Write a Java program to demonstrate the use of if-else and switch-case statements.

import java.util.Scanner;

public class DecisionDemo {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// if-else example

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

int age = sc.nextInt();

if (age >= 18) {

System.out.println("You are eligible to vote.");

} else {

System.out.println("You are NOT eligible to vote.");

}

// switch-case example

System.out.print("Enter a day number (1-3): ");

int day = sc.nextInt();

switch (day) {

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

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

case 3: System.out.println("Wednesday"); break;

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

}

sc.close();

}

}

Output:

* 1. What is the difference between break and continue statements?

## ****Difference Between**** break ****and**** continue

| **Feature** | **break** | **continue** |
| --- | --- | --- |
| **Purpose** | Terminates the loop or switch entirely. | Skips the current iteration and continues with the next. |
| **Usage** | Used in loops and switch-case. | Used only in loops. |
| **Effect** | Jumps out of the loop/switch. | Skips rest of loop body for current iteration. |

* 1. Write a Java program to print even numbers between 1 to 50 using a for loop.

public class EvenNumbers {

public static void main(String[] args) {

for (int i = 1; i <= 50; i++) {

if (i % 2 == 0) {

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

}

}

}

}

Output:

2 4 6 8 10 ... 50

* 1. Explain the differences between while and do-while loops with examples.

| **Feature** | **while Loop** | **do-while Loop** |
| --- | --- | --- |
| **Condition Check** | Condition is checked **before** execution. | Condition is checked **after** execution. |
| **Execution Guarantee** | May execute **0 times** if condition is false initially. | Executes **at least once**, even if condition is false. |
| Example 1: |  |  |

int i = 5;

while (i < 5) {

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

i++;

}

Example 2:

int j = 5;

do {

System.out.println("Do-while loop");

j++;

} while (j < 5); // Prints once

## Section 3: Java Keywords and Operators

1. What are keywords in Java? List 10 commonly used keywords.

**Keywords** are **reserved words** in Java that have a predefined meaning and purpose in the language.

* They **cannot** be used as identifiers (variable names, method names, class names, etc.).
* Java has **50 keywords** (as of Java 17, including const and goto which are unused).

**10 Commonly Used Keywords:**

1. class
2. public
3. static
4. void
5. int
6. if
7. else
8. for
9. return
10. new
11. Explain the purpose of the following keywords: static, final, this, super.

| **Keyword** | **Purpose** | **Example** |
| --- | --- | --- |
| **static** | Used to define class-level variables/methods that belong to the class, not objects. | static int count; |
| **final** | Used to declare constants, prevent method overriding, or inheritance of a class. | final double PI = 3.14; |
| **this** | Refers to the current object of the class. Used to differentiate instance variables from parameters or call another constructor. | this.name = name; |
| **super** | Refers to the immediate parent class. Used to call parent class methods/constructors. | super.display(); |

1. What are the types of operators in Java?

Java operators are classified into:

1. **Arithmetic Operators** – +, -, \*, /, %
2. **Relational Operators** – ==, !=, >, <, >=, <=
3. **Logical Operators** – &&, ||, !
4. **Assignment Operators** – =, +=, -=, \*=, /=, %=
5. **Unary Operators** – ++, --, +, -, !
6. **Bitwise Operators** – &, |, ^, ~, <<, >>, >>>
7. **Ternary Operator** – ?:
8. **Instanceof Operator** – Checks object type (instanceof)
9. Write a Java program demonstrating the use of arithmetic, relational, and logical operators.

public class OperatorDemo {

public static void main(String[] args) {

int a = 10, b = 5;

System.out.println("Arithmetic Operators:");

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

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

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

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

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

System.out.println("\nRelational Operators:");

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

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

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

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

boolean x = true, y = false;

System.out.println("\nLogical Operators:");

System.out.println("x && y: " + (x && y));

System.out.println("x || y: " + (x || y));

System.out.println("!x: " + (!x));

}

}

Output:

Arithmetic Operators:

a + b = 15

a - b = 5

a \* b = 50

a / b = 2

a % b = 0

Relational Operators:

a == b: false

a != b: true

a > b: true

a < b: false

Logical Operators:

x && y: false

x || y: true

!x: false

1. What is operator precedence? How does it affect the outcome of expressions?

**Operator precedence** defines the **order in which operators are evaluated** in an expression.

* Higher precedence operators are evaluated **first**.
* If operators have the **same precedence**, associativity decides the order (left-to-right or right-to-left).

# Additional Questions

## Java Data Types

1. What is the size and range of each primitive data type in Java?

| **Data Type** | **Size (bits)** | **Size (bytes)** | **Range** |
| --- | --- | --- | --- |
| **byte** | **8** | **1** | **-128 to 127** |
| **short** | **16** | **2** | **-32,768 to 32,767** |
| **int** | **32** | **4** | **-2,147,483,648 to 2,147,483,647** |
| **long** | **64** | **8** | **-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807** |
| **float** | **32** | **4** | **~6-7 decimal digits precision** |
| **double** | **64** | **8** | **~15 decimal digits precision** |
| **char** | **16** | **2** | **'\u0000' (0) to '\uffff' (65,535)** |
| **boolean** | **JVM-dependent** | **JVM-dependent** | **true or false** |

1. How does Java handle overflow and underflow with numeric types?
2. **Overflow** happens when a value exceeds the maximum value a data type can hold.
3. **Underflow** happens when a value goes below the minimum value.

Java **does not throw an error** for overflow/underflow in integer arithmetic — instead, it wraps around using **two’s complement representation**.

1. Write a program to convert a double value to an int without data loss.

public class DoubleToInt {

public static void main(String[] args) {

double value = 100.0;

if (value == Math.floor(value)) { // No decimal part

int intValue = (int) value;

System.out.println("Converted value: " + intValue);

} else {

System.out.println("Cannot convert without losing data");

}

}

}

1. What is the difference between char and String in Java?

| **Feature** | **char** | **String** |
| --- | --- | --- |
| **Type** | Primitive data type | Non-primitive (class) |
| **Size** | 2 bytes (stores a single Unicode character) | Varies (stores sequence of characters) |
| **Example** | char c = 'A'; | String s = "Hello"; |
| **Mutability** | Not applicable (single value) | Immutable (cannot be changed once created) |
| **Usage** | Store single character | Store a sequence of characters (words, sentences) |

1. Explain wrapper classes and their use in Java.

**Wrapper classes** are object representations of primitive data types.

Each primitive type has a corresponding wrapper class in java.lang package.

| **Primitive** | **Wrapper Class** |
| --- | --- |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| char | Character |
| boolean | Boolean |

## Java Control Statements

* 1. Write a Java program using nested if statements.

public class NestedIfExample {

public static void main(String[] args) {

int age = 25;

boolean hasVoterID = true;

if (age >= 18) {

if (hasVoterID) {

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

} else {

System.out.println("You need a voter ID to vote.");

}

} else {

System.out.println("Not eligible to vote.");

}

}

}

* 1. Write a Java program to display the multiplication table of a number using a loop.

import java.util.Scanner;

public class MultiplicationTable {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int num = sc.nextInt();

for (int i = 1; i <= 10; i++) {

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

}

sc.close();

}

}

* 1. How do you exit from nested loops in Java?

public class ExitNestedLoops {

public static void main(String[] args) {

outerLoop:

for (int i = 1; i <= 3; i++) {

for (int j = 1; j <= 3; j++) {

if (i == 2 && j == 2) {

break outerLoop; // exits both loops

}

System.out.println(i + " " + j);

}

}

}

}

* 1. Compare and contrast for, while, and do-while loops.

| **Feature** | **for Loop** | **while Loop** | **do-while Loop** |
| --- | --- | --- | --- |
| **When to Use** | When the number of iterations is **known**. | When iterations depend on a **condition**. | When you need the loop to run **at least once**. |
| **Condition Check** | Before each iteration. | Before each iteration. | After each iteration. |
| **Syntax** | for (init; condition; update) { } | while (condition) { } | do { } while (condition); |
| **Executes if Condition is False Initially?** | No | No | Yes (once) |
| **Example** | for (int i=0; i<5; i++) | while (i<5) | do { } while (i<5); |

* 1. Write a program that uses a switch-case to simulate a basic calculator.

import java.util.Scanner;

public class Calculator {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter first number: ");

double num1 = sc.nextDouble();

System.out.print("Enter second number: ");

double num2 = sc.nextDouble();

System.out.print("Enter operation (+, -, \*, /): ");

char op = sc.next().charAt(0);

switch (op) {

case '+':

System.out.println("Result: " + (num1 + num2));

break;

case '-':

System.out.println("Result: " + (num1 - num2));

break;

case '\*':

System.out.println("Result: " + (num1 \* num2));

break;

case '/':

if (num2 != 0)

System.out.println("Result: " + (num1 / num2));

else

System.out.println("Error: Division by zero!");

break;

default:

System.out.println("Invalid operator!");

}

sc.close();

}

}

## Java Keywords and Operators

1. What is the use of the `instanceof` keyword in Java?

instanceof is used to **check if an object is an instance of a specific class or subclass**.

It **returns true or false**.

class Animal {}

class Dog extends Animal {}

public class InstanceOfExample {

public static void main(String[] args) {

Dog d = new Dog();

System.out.println(d instanceof Dog); // true

System.out.println(d instanceof Animal); // true

System.out.println(d instanceof Object); // true

}

}

Prevents **ClassCastException** before type casting.

1. Explain the difference between `==` and `.equals()` in Java.

String s1 = new String("Hello");

String s2 = new String("Hello");

System.out.println(s1 == s2); // false (different objects)

System.out.println(s1.equals(s2)); // true (same content)

1. Write a program using the ternary operator.

public class TernaryExample {

public static void main(String[] args) {

int a = 10, b = 20;

String result = (a > b) ? "a is greater" : "b is greater";

System.out.println(result);

}

}

1. What is the use of `this` and `super` in method overriding?

class Parent {

void display() {

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

}

}

class Child extends Parent {

void display() {

super.display(); // Call parent version

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

}

void show() {

this.display(); // Call current class version

}

}

public class ThisSuperExample {

public static void main(String[] args) {

Child c = new Child();

c.show();

}

}

1. Explain bitwise operators with examples.

public class BitwiseExample {

public static void main(String[] args) {

int x = 5; // binary: 0101

int y = 3; // binary: 0011

// Bitwise AND

System.out.println("x & y = " + (x & y)); // 0101 & 0011 = 0001 → 1

// Bitwise OR

System.out.println("x | y = " + (x | y)); // 0101 | 0011 = 0111 → 7

// Bitwise XOR (exclusive OR)

System.out.println("x ^ y = " + (x ^ y)); // 0101 ^ 0011 = 0110 → 6

// Bitwise NOT (flips bits)

System.out.println("~x = " + (~x)); // ~0101 → 2’s complement → -6

// Left shift (multiply by 2 for each shift)

System.out.println("x << 1 = " + (x << 1)); // 0101 → 1010 → 10

// Right shift (divide by 2 for each shift)

System.out.println("x >> 1 = " + (x >> 1)); // 0101 → 0010 → 2

}

}