# DAY 1 AND 2

# Java Programming Assignment

## Section 1: Java Data Types

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

Java provides **8 primitive data types** that form the building blocks of data manipulation:

* **byte** – It contain 8-bit signed integer, range **−128 to 127**; often used to save memory in large arrays.
* **short** – It contain 16-bit signed integer, range **−32,768 to 32,767**; suitable for smaller numerical values.
* **int** – It contain 32-bit signed integer, range **−2³¹ to 2³¹−1**; default for integer operations.
* **long** – It contain 64-bit signed integer, range **−2⁶³ to 2⁶³−1**; used for very large numbers.
* **float** – It contain 32-bit single-precision floating-point; approx. 7 digits of precision.
* **double** – It contain 64-bit double-precision floating-point; approx. 15 digits of precision, default for decimal values.
* **char** – It contain 16-bit Unicode character; stores a single character like 'A' or '\u0041'.
* **boolean** –It represents **true** or **false**; used for logical conditions

1. **Explain the difference between primitive and non-primitive data types in Java.**

**Primitive Data Types:**

Definition: The most basic built‑in data types in Java, used to store simple values directly.

* 1. It store the actual value in memory, not a reference.
  2. Not objects; cannot call methods directly on them.
  3. It have a fixed size (determined by the Java specification, except for boolean).
  4. It stores in stack memory for local variables (making access faster).
  5. It has default values like 0, 0.0, false, or '\u0000' when used as class/instance fields.
* Examples: byte, short, int, long, float, double, char, boolean.

int age = 27; // stores integer directly

boolean isReady = true; // stores true/false directly

 Non‑Primitive (Reference) Data Types

* Definition: Data types that refer to objects that may contain multiple values or complex data.
  1. Store a memory address (reference) pointing to data stored in the heap.
  2. It can be null (meaning no object is referenced).
  3. It can have methods and properties.
  4. The size is variable and depends on the object structure.
  5. Even if declared as local variables, the object’s data resides in heap memory, but the reference is on the stack.
* Examples: String, arrays, user‑defined classes, interfaces, enums, wrapper classes.

String name = "Java"; // 'name' holds reference to String object

int[] marks = {85, 90, 95}; // reference to array object

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

package Day2;

public class primitiveDataTypes {

public static void main(String[] args) {

byte byteVal = 100;

short shortVal = 10000;

int intVal = 100000;

long longVal = 10000000000L;

float floatVal = 10.5f;

double doubleVal = 20.99;

char charVal = 'A';

boolean boolVal = true;

System.***out***.println("Value Of byte: " + byteVal);

System.***out***.println("Value Of short: " + shortVal);

System.***out***.println("Value Of int: " + intVal);

System.***out***.println("Value Of long: " + longVal);

System.***out***.println("Value Of float: " + floatVal);

System.***out***.println("Value Of double: " + doubleVal);

System.***out***.println("Value Of char: " + charVal);

System.***out***.println("Value Of boolean: " + boolVal);

}

}

Output:

Value Of byte: 100

Value Of short: 10000

Value Of int: 100000

Value Of long: 10000000000

Value Of float: 10.5

Value Of double: 20.99

Value Of char: A

Value Of boolean: true

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

Type casting in Java is the process of converting a value from one data type to another.  
It is essential when you want to assign a value of one primitive type to a variable of another type, especially when the types differ in size or kind.

**Types of Casting in Java:**

1. Implicit Casting (Widening Conversion):
   * In this Implicit Casting automatically done by the Java compiler when converting a smaller data type to a larger data type size.
   * No data loss occurs because the larger type can comfortably hold the smaller type’s value.
2. Explicit Casting (Narrowing Conversion):
   * It is done by manually by the programmer using a cast operator () when converting a larger data type to a smaller type.
   * May cause data loss or truncation if the value does not fit in the target type.
   * Required because Java does not allow implicit narrowing.

Ex: package Day2;

public class TypeCasting {

public static void main(String[] args) {

int num = 600;

double d = num;

System.***out***.println("Implicit Casting (int to double): " + d);

double pi = 3.14;

int i = (int) pi;

System.***out***.println("Explicit Casting (double to int): " + i);

}

}

Output:

Implicit Casting (int to double): 600.0

Explicit Casting (double to int):

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

## In Java, when primitive data type variables are declared as class-level fields (static or instance) but not explicitly initialized, the compiler assigns them predefined default values. These are:

## byte → Default value is 0, representing a zero integer in 8‑bit form.

## short → Default value is 0, representing a zero integer in 16‑bit form.

## int → Default value is 0, the standard 32‑bit integer zero.

## long → Default value is 0L, which is the long integer zero literal (64‑bit).

## float → Default value is 0.0f, representing a single‑precision floating‑point zero.

## double → Default value is 0.0d, representing a double‑precision floating‑point zero.

## char → Default value is '\u0000', which is the Unicode null character (non‑printable).

## boolean → Default value is false, representing a logical false condition.

## Section 2: Java Control Statements

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

Control statements are used in Java to control the flow of execution in a program like deciding, repeating, or jumping to a certain part of code based on some conditions.

They are mainly of three types:

**1. Decision-making statements:**

Used to make choices in code:

* if
* if-else
* if-else-if
* switch

**2. Looping statements:**

Used to repeat a block of code:

* for
* while
* do-while

**3. Jumping statements:**

Used to jump out of or skip code:

* break
* continue
* return

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

package Day2;

import java.util.Scanner;

public class controlStatments {

public static void main(String[] args) {

Scanner input = new Scanner(System.***in***);

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

int age = input.nextInt();

if (age >= 18) {

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

} else {

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

}

// Use Of Switch-case With An Example

System.***out***.print("Enter a day number (1-7): ");

int day = input.nextInt();

switch (day) {

case 1:

System.***out***.println("Sunday");

break;

case 2:

System.***out***.println("Monday");

break;

case 3:

System.***out***.println("Tuesday");

break;

case 4:

System.***out***.println("Wednesday");

break;

case 5:

System.***out***.println("Thursday");

break;

case 6:

System.***out***.println("Friday");

break;

case 7:

System.***out***.println("Saturday");

break;

default:

System.***out***.println("Invalid day");

}

input.close();

}

}

Output:

Enter your age: 22

You are eligible to vote.

Enter a day number (1-7): 1

Sunday

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

**Ans)**

|  | **BREAK** | **CONTINUE** |
| --- | --- | --- |
| **Purpose** | It will terminate the loop immediately. | The current iteration will be skipped and moves to the next. |
| **Effect on Loop** | It helps to exit the loop completely. | Loop continues running from the next iteration. |
| **Usage** | Often used when a certain condition is met and no further loop execution is needed. | Used when certain iterations need to be skipped without stopping the loop. |
| **Applicable In** | for, while, do-while, and switch. | for, while, do-while. |
| **Example** | When the data found it stops searching. | Skips processing unwanted data. |

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

package Day2;

public class EvenNumbersBtw1to50 {

public static void main(String[] args) {

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

if (i % 2 == 0) { // checks for the even number condition

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

}

}

}

}

Output

2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50

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

The while and do-while loops are two common looping constructs in Java used for repeating a block of code based on a condition.

1. while loop

* The while loop tests the loop condition before executing the loop body.
* If the condition is false initially, the loop body may never execute.
* It is primarily used when the number of iterations is not known before and you want to ensure the condition is checked before each iteration.

Syntax:

java

**while** (condition) {

// body

}

Example:

**int** i = 0;

**while** (i < 3) {

System.out.println("while loop iteration: " + i);

i++;

}

Output:

while loop iteration: 0

while loop iteration: 1

while loop iteration: 2

2. do-while loop

* The do-while loop executes the loop body at least once before testing the condition.
* The condition is tested after the loop body executes.
* This ensures that the loop body runs at least once irrespective of the condition.
* It is used when you want the code inside the loop to execute before any condition check.

Syntax:

java

**do** {

// body

} **while** (condition);

Example:

java

**int** j = 0;

**do** {

System.out.println("Karthik kavali: " + j);

j++;

} **while** (j < 3);

Output:

Karthik kavali: 0

Karthik kavali: 1

Karthik kavali: 2

## Section 3: Java Keywords and Operators

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

Keywords are predefined, reserved words that have a specific meaning to the Java compiler. Each keyword serves a special purpose in defining the structure, control flow, and behavior of a Java program.

class – Used to declare a class. Ex: class King{}

public – An access modifier Ex: public int age;

static – Used to declare class-level members that belong to the class rather than objects. Ex: static int age = 21;

void – Indicates a method does not return any value. Ex: public void display()

int – Primitive data type ex: int a = 21;

if – Used for decision making. Ex: if(age > 21){ }

else –alternate block for if block. Ex: else{}

for – Loop control statement to iterate a set number of times. for (int i = 0; i < 5; i++) { }

return – Exits from a method and optionally returns a value. Ex: return 0;

new – Used to create new objects. Ex: King obj = new King();

1. **Explain the purpose of the following keywords: static, final, this, super.**

1.**Static:**

* The static keyword is used to declare class-level members (variables or methods) that belong to the class itself rather than to any specific instance (object) of the class.
* Static members are shared among all instances of the class.

**2. Final**

* The final keyword is used to declare constants or variables/methods/classes that are not allowed to be changed or overridden once initialized or defined.

**3. This**

* The this keyword refers to the current instance of the class where it is used.

1. **What are the types of operators in Java?**

**Ans**) Java supports several types of operators:

* Arithmetic Operators:  
  These are used to perform basic mathematical calculations like addition, subtraction, multiplication, division, and modulus.  
  Examples: +, -, \*, /, %
* Unary Operators:  
  Operate on a single operand to perform operations such as increment, decrement, negation, and logical complement.  
  Examples: ++ ,--, (decrement), + (unary plus), - (unary minus), ! (logical not)
* Assignment Operators:  
  Used to assign values to variables. Compound assignment operators combine arithmetic or bitwise operations with assignment.  
  Examples: =, +=, -=, \*=, /=, %= etc.

Ex: a+= 2;

* Relational (Comparison) Operators:  
  Compare two values and return a boolean result (true or false).  
  Examples: ==, !=, >, <, >=, <=
* Logical Operators:  
  Combine multiple boolean expressions to form complex conditions.  
  Examples: && (logical AND), || (logical OR), ! (logical NOT)
* Bitwise Operators:  
  Operate on the individual bits of integer types to perform bit-level manipulations.  
  Examples: & (bitwise AND), | (bitwise OR), ^ (bitwise XOR), ~ (bitwise complement)
* Shift Operators:  
  Shift the bits of a number to the left or right.  
  Examples: << (left shift), >> (right shift with sign extension), >>> (unsigned right shift)
* Ternary Operator:  
  A conditional operator that takes three operands, acting as a shorthand for if-else statements.  
  Syntax: condition ? expressionIfTrue : expressionIfFalse
* Instance of Operator:  
  Tests whether an object is an instance of a specific class or implements an interface.  
  Syntax: object instanceof ClassName

1. **Write a Java program demonstrating the use of arithmetic, relational, and logical operators.**

**package Day2;**

**import java.util.Scanner;**

**public class AllOperators {**

**public static void main(String[] args) {**

**Scanner in = new Scanner(System.in);**

**System.out.println("Enter Value Of A: ");**

**int a = in.nextInt();**

**System.out.println("Enter Value Of B: ");**

**int b = in.nextInt();**

**// Arithmetic Operators**

**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));**

**// Relational Operators**

**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));**

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

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

**// Logical Operators**

**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:

Enter Value Of A:

5

Enter Value Of B:

4

Arithmetic Operators:

a + b = 9

a - b = 1

a \* b = 20

a / b = 1

a % b = 1

Relational Operators:

a == b: false

a != b: true

a > b: true

a < b: false

a >= b: true

a <= b: false

Logical Operators:

x && y: false

x || y: true

!x: false

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

**Ans)** Operator precedence in Java (and most programming languages) refers to the rules that determine the order in which operators are evaluated in an expression when there are multiple operators involved and expressions can contain multiple operators, understanding and following operator precedence ensures that the expression is evaluated correctly and produces the intended result. Ignoring precedence rules or misunderstanding them can lead to unexpected outcomes or logical errors.

Ex: int result = (10 + 20) \* 3;

System.out.println(result);

Output:

90