# Java Programming Assignment

## Section 1: Java Data Types

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

Answer; In java there are eight primitive data types they are

1. int 2.float3.short4.long5.double6.char7.boolean8.byte

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

Answer; 1. Primitive types are predefined types in java.

1. Non-Primitive types are created by the software developer and is not defined by java (except for string).
2. Non-primitive types can be used to call methods to perform certain functions while primitive types cannot.
3. A primitive type has always a value, while non-primitive types can be null.
4. A primitive type starts with lowercase letters (int, char),while non-primitive types starts with an uppercase letter (String and Class).

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

Code; **package** Datatype\_practice;

**public** **class** Employee {

String employeename="RAJU";

**int** employeeid=145678;

String Designation="automation test engineer";

String companyname="wipro";

String Toolsused="selenium and java";

**long** salary=25000;

**short** workinghours=8;

**float** experience=2.6f;

**byte** noofleaves=2;

String location="Hyderabad";

**char** worktype='H'; /\*H=HYBRID R=REMOTE \*/

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

Employee emp=**new** Employee();

System.***out***.println("EmployeeName = "+emp.employeename);

System.***out***.println("EmployeeId = "+emp.employeeid);

System.***out***.println("Employee Designation = "+emp.Designation);

System.***out***.println("Employee companyname = "+emp.companyname);

System.***out***.println("Employee Toolsused = "+emp.Toolsused);

System.***out***.println("Employee Workinghours = "+emp.workinghours);

System.***out***.println("Employee noofleaves = "+emp.noofleaves);

System.***out***.println("Employee salary = "+emp.salary);

System.***out***.println("Employee experience = "+emp.experience);

System.***out***.println("Employee location = "+emp.location);

System.***out***.println("Employee worktype = "+emp.worktype);

}

}

Output; EmployeeName = RAJU

EmployeeId = 145678

Employee Designation = automation test engineer

Employee companyname = wipro

Employee Toolsused = selenium and java

Employee Workinghours = 8

Employee noofleaves = 2

Employee salary = 25000

Employee experience = 2.6

Employee location = Hyderabad

Employee worktype = H

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

Answer; Converting one type of data to another type is called casting.

Implicit casting (Widening casting); It is also done automatically. Implicit casting occurs automatically when converting a smaller data type to a larger one. This process is safe and doesn't lead to data loss.

Explicit casting(Narrowing casting);It is done manually. Explicit casting is required when converting a larger data type to a smaller one. This process can lead to data loss and must be specified manually.

Code; **package** Type\_casting;

**public** **class** Convert\_demo {

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

//widening(implicit casting,Automatically cast)

**int** num=10;

System.***out***.println("Value of num = "+num);

**float** d=num;

System.***out***.println("Value of d = "+d);

//Narrowing(explicit casting ,Manually cast)

**int** a=(**int**)d;

System.***out***.println("Value of a = "+a);

}

}

Output; Value of num = 10

Value of d = 10.0

Value of a = 10

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

Answer; Primitive data types inside methods are declared with some values otherwise compiler throw error. But as they initialized in a class or interface their default values are:

Byte: 0;

Float: 0.0f;

Int: 0;

Short: 0;

Long: 0l;

Double: 0.0d

Char: '\u0000'

Boolean: false

## Section 2: Java Control Statements

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

Answer; Control statements in Java are constructs that dictate the flow of execution—determining which statements run, when, and how many times. They’re fundamental for creating dynamic, logical, and responsive programs. There are **three main categories**:

1. Decision making (conditional ) statements:

=>if-True

=>if-else-True/false

=>if-else-if-Ladder multiple conditions

=>nested-if-condition inside

=>switch-case

Code; **package** Control\_statement\_practice;

**public** **class** if\_else\_if {

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

**float** per=60f;

**if**(per>=75 && per<=100)

{

**if**(per>=75) {

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

}

}

**else** **if**(per>=60 && per<75 )

{

System.***out***.println("First class");

}

**else** **if**(per>=50 && per<60)

{

System.***out***.println("Second class");

}

**else** **if**(per>=40 && per<50)

{

System.***out***.println("pass class");

}

**else** **if**(per>=0 && per<40)

{

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

}

**else** {

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

}

}

}

Output; First class

Code; **package** Control\_statement\_practice;

/\*2. WAP for simple calculator

\* like addition, subtraction , multiplication, division

\*/

**public** **class** calculator {

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

**int** a=3;

**int** b=7;

**char** operation='/';

**switch**(operation){

**case** '+':System.***out***.println("sum of the numbers "+(a+b));

**break**;

**case** '-':System.***out***.println("sub of the numbers "+(a-b));

**break**;

**case** '\*':System.***out***.println("mup of the numbers "+(a\*b));

**break**;

**case** '/':System.***out***.println("div of the numbers "+(a/b));

**break**;

**default**:

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

}

}

}

Output; div of the numbers 0

2. Looping statements:

=>for loop-it is known to run how many times

=>while loop-it is unknown how many time to run using condition it works.

=>do-while loop-without checking condition first do loop statement is executed first for one time then it checks for while loop condition.

3. Jumping statements;

=>break-to terminate the loop

=>continue-to skip the iteration of particular element and continue the iteration

1. Write a Java program to demonstrate the use of if-else and switch-case statements.
2. Code; **package** Control\_statement\_practice;
3. **public** **class** if\_else\_if {
4. **public** **static** **void** main(String[] args) {
5. **float** per=60f;
6. **if**(per>=75 && per<=100)
7. {
8. **if**(per>=75) {
9. System.***out***.println("Dist");
10. }
11. }
12. **else** **if**(per>=60 && per<75 )
13. {
14. System.***out***.println("First class");
15. }
16. **else** **if**(per>=50 && per<60)
17. {
18. System.***out***.println("Second class");
19. }
20. **else** **if**(per>=40 && per<50)
21. {
22. System.***out***.println("pass class");
23. }
24. **else** **if**(per>=0 && per<40)
25. {
26. System.***out***.println("Fail");
27. }
28. **else** {
29. System.***out***.println("Invalid Input");
30. }
31. }
32. }
33. Output; First class
34. Code; **package** Control\_statement\_practice;
35. /\*2. WAP for simple calculator
36. \* like addition, subtraction , multiplication, division
37. \*/
38. **public** **class** calculator {
39. **public** **static** **void** main(String[] args) {
40. **int** a=3;
41. **int** b=7;
42. **char** operation='/';
43. **switch**(operation){
44. **case** '+':System.***out***.println("sum of the numbers "+(a+b));
45. **break**;
46. **case** '-':System.***out***.println("sub of the numbers "+(a-b));
47. **break**;
48. **case** '\*':System.***out***.println("mup of the numbers "+(a\*b));
49. **break**;
50. **case** '/':System.***out***.println("div of the numbers "+(a/b));
51. **break**;
52. **default**:
53. System.***out***.println("Invalid input");
55. }
56. }
57. }
58. Output; div of the numbers 0
59. What is the difference between break and continue statements?

Answer; **break**  
Immediately exits the current loop or switch, terminating it altogether and transferring control to the first statement after that construct.

**continue**  
Skips the rest of the current iteration in a loop and proceeds to the next iteration. The loop continues after the current iteration ends.

Both statements can be applied to for, while, do-while, and (in the case of break) switch constructs.

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

Code; **package** Control\_statement\_practice;

**public** **class** Even\_numbers\_1\_50 {

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

System.***out***.println("Even numbers from 1 to 50:");

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

**if**(i%2==0) {

System.***out***.println(i);

}

}

}

}

Output; Even numbers from 1 to 50:

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

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

Answer;

While loop (Entry-Controlled Loop)

=>The condition is evaluated **before** entering the loop.

=>If the condition is initially false, the loop body **may never execute**.

do-while **loop (Exit-Controlled Loop)**

=>The loop body executes **first**, and then the condition is checked.

=>This ensures the loop body **executes at least once**, even if the condition is false from the start.

Code; **package** loop\_statements\_practice;

**public** **class** While\_do\_while\_demo {

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

**int** i=10;

//here , i<5 is false intially , so the loop body is not executed

**while**(i<5) {

System.***out***.println("This won't print");

}

//here, first it executes the do body statement first then only it checks for while loop condition

**do** {

System.***out***.println("This prints once even if condition is false");

}**while**(i<5);

}

}

Output; This prints once even if condition is false

## Section 3: Java Keywords and Operators

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

Answer; In Java, **keywords** are **reserved words** with special meanings predefined by the language. You cannot use them as identifiers (like variable, class, or method names). They help the compiler interpret and execute the code correctly.

Their list of 10 are; 1.class 2.public 3.static 4.void 5.int 6.if. 7. for 8.while 9. Do-while 10. break

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

Answer;

|  |  |
| --- | --- |
| static | -Defines class-level members shared across all instances. |
| final | -Makes variables immutable, methods non-overridable, or classes non-extendable. |
| this | -Refers to current object, allows chaining constructors and disambiguation. |
| super | -Accesses superclass members—variables, methods, or constructors. |

1. What are the types of operators in Java?

Answer: In Java, **operators** are special symbols that perform operations on one, two, or three operands and return a result. They're essential for handling data manipulation, logic control, and flow.

Major Types of Java Operators

1. **Arithmetic Operators**  
   Perform mathematical operations like addition (+), subtraction (-), multiplication (\*), division (/), and modulus (%).  
   **Example:** a + b, a \* b
2. **Unary Operators**  
   Operate on a single operand. Commonly used for increment (++), decrement (--), logical NOT (!), and unary plus/minus (+, -).  
   **Example:** ++a, !flag
3. **Assignment Operators**  
   Used to assign values to variables. Includes the basic assignment (=) and compound assignments like +=, -=, \*=, /=, %=.  
   **Example:** x += 5 (same as x = x + 5)
4. **Relational (Comparison) Operators**  
   Compare two values and return a boolean. Operators include ==, !=, >, <, >=, <=.  
   **Example:** a > b returns true or false
5. **Logical Operators**  
   Combine boolean expressions. Includes logical AND (&&), OR (||), and NOT (!).  
   **Example:** cond1 && cond2
6. **Bitwise Operators**  
   Perform operations on individual bits. Includes AND (&), OR (|), XOR (^), NOT (~), and shifts: left shift (<<), signed right shift (>>), unsigned right shift (>>>)
7. **Shift Operators**  
   Move bits left or right. Left shift (<<), signed right shift (>>), and unsigned right shift (>>>) are all part of Java's operator set.
8. **Ternary (Conditional) Operator**  
   A shorthand for if-else statements. Syntax: condition ? expr1 : expr2, where one of two expressions is evaluated based on the boolean condition.  
   **Example:** int max = (a > b) ? a : b;
9. **Type Comparison Operator (instanceof)**  
   Checks if an object is an instance of a specified class or interface. Returns a boolean.  
   **Example:** if (obj instanceof String)
10. Write a Java program demonstrating the use of arithmetic, relational, and logical operators.

Code; **package** Control\_statement\_practice;

**public** **class** OperatorDemo {

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

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

**int** a = 15;

**int** b = 4;

System.***out***.println("a + b = " + (a + b)); // 19

System.***out***.println("a - b = " + (a - b)); // 11

System.***out***.println("a \* b = " + (a \* b)); // 60

System.***out***.println("a / b = " + (a / b)); // 3 (integer division)

System.***out***.println("a % b = " + (a % b)); // 3 (remainder)

// Relational Operators: >, <, >=, <=, ==, !=

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

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

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

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

// Logical Operators: && (AND), || (OR), ! (NOT)

**boolean** c = **true**;

**boolean** d = **false**;

System.***out***.println("c && d: " + (c && d)); // false

System.***out***.println("c || d: " + (c || d)); // true

System.***out***.println("!c: " + (!c)); // false

// Combined example: check if 'a' is between 10 and 20 (inclusive)

**boolean** between = (a >= 10) && (a <= 20);

System.***out***.println("Is 'a' between 10 and 20? " + between); // true

}

}

Output; a + b = 19

a - b = 11

a \* b = 60

a / b = 3

a % b = 3

a > b: true

a < b: false

a == b: false

a != b: true

c && d: false

c || d: true

!c: false

Is 'a' between 10 and 20? true

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

Answer; Operator precedence in Java refers to the set of built-in rules that determine the order in which operators in an expression are evaluated. Simply put: **operators with higher precedence are evaluated before operators with lower precedence**

When two operators share the same precedence, Java uses **associativity** to decide how to group them:

**=>Left-to-Right**: Most binary operators (e.g., +, -, \*, /, &&, ||)

**=>Right-to-Left**: Assignment (=), compound assignment (like +=), and ternary (?:)

# Additional Questions

## Java Data Types

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

Answer; In Java, there are eight primitive data types, each with a specific size and range. Here's detailed table;

| Data Type | **Size (Bits)** | **Range** | **Default Value** |
| --- | --- | --- | --- |
| byte | 8 | -128 to 127 | 0 |
| short | 16 | -32,768 to 32,767 | 0 |
| int | 32 | -2,147,483,648 to 2,147,483,647 | 0 |
| long | 64 | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | 0L |
| float | 32 | Approximately ±1.4E-45 to ±3.4E+38 (6-7 decimal digits precision) | 0.0f |
| double | 64 | Approximately ±4.9E-324 to ±1.8E+308 (15-16 decimal digits precision) | 0.0d |
| char | 16 | 0 to 65,535 (Unicode characters '\u0000' to '\uffff') | '\u0000' |
| boolean | 1 (logical) | true or false (size not precisely defined) | false |

1. How does Java handle overflow and underflow with numeric types?

Answer;

| **Numeric Type** | **Overflow Behavior** | **Underflow Behavior** | **How to Detect/Handle** |
| --- | --- | --- | --- |
| Integer types | Silently wrap around (modulo) | Same wrap-around when going below min | Use math.addExact() or , BigInteger ,manual checks |
| Floating-point types | Positive/negative infinity (∞) | Becomes 0.0 or -0.0 | Check for Infinity or zero values |

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

Code; **package** Type\_casting;

**public** **class** SafeDoubleToInt {

**public** **static** **int** safeDoubleToInt(**double** value) {

**if** (Double.*isNaN*(value)) {

**throw** **new** IllegalArgumentException("Cannot convert NaN to int");

}

**if** (value > Integer.***MAX\_VALUE*** || value < Integer.***MIN\_VALUE***) {

**throw** **new** ArithmeticException("Value " + value + " is outside int range");

}

**if** (value % 1 != 0) {

**throw** **new** ArithmeticException("Fractional part lost: " + value);

}

**return** (**int**) value;

}

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

**double**[] testValues = {12345.0, 3.0, 1.5, 2\_147\_483\_648.0, Double.***NaN***};

**for** (**double** d : testValues) {

**try** {

System.***out***.println(d + " → " + *safeDoubleToInt*(d));

} **catch** (Exception e) {

System.***out***.println("Error converting " + d + ": " + e.getMessage());

}

}

}

}

Output; 12345.0 → 12345

3.0 → 3

Error converting 1.5: Fractional part lost: 1.5

Error converting 2.147483648E9: Value 2.147483648E9 is outside int range

Error converting NaN: Cannot convert NaN to int

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

Answer; char-is a used to assign a single character.

Whereas string is a collection of such characters

=>char holds exactly **one character**.

=> String can hold **zero, one, or many characters** (even entire sentences)

| **Feature** | **char** | **String** |
| --- | --- | --- |
| Type | Primitive | Reference (class) |
| Can hold | One character | Zero or more characters |
| Literal syntax | Single quotes ('A') | Double quotes ("Hi") |
| Mutability | Immutable (single value) | Immutable, new object for each change |
| Functionality | No built-in methods | Rich API (length(),concat(), etc.) |
| Memory behavior | Low overhead | Higher overhead, string pooling available |
| Operator behavior | + yields integer | + concatenates |

1. Explain wrapper classes and their use in Java.

Answer;

| **Aspect** | **Wrapper Classes** |
| --- | --- |

|  |  |
| --- | --- |
| Type | Final & immutable object versions of primitives |

|  |  |
| --- | --- |
| Use Cases | Collections, generics, nullability, utility methods, reflection |

|  |  |
| --- | --- |
| Conversion | Autoboxing and unboxing automatically handle conversions |

|  |  |
| --- | --- |
| Pros | Increased flexibility, object ability, richer API |

|  |  |
| --- | --- |
| Cons | Performance overhead, memory usage, null risks |

## Java Control Statements

1. Write a Java program using nested if statements.

Code; **package** Control\_statement\_practice;

**import** java.util.Scanner;

**public** **class** Main {

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

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

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

**int** num = sc.nextInt();

**if** (num < 100) {

System.***out***.println("The entered number is less than 100");

**if** (num > 50) {

System.***out***.println("The entered number is greater than 50");

}

} **else** {

System.***out***.println("The entered number is greater than or equal to 100");

}

sc.close();

}

}

Output; Enter a number: 56

The entered number is less than 100

The entered number is greater than 50

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

Code; **package** Control\_statement\_practice;

**import** java.util.Scanner;

**public** **class** Table\_of\_a\_number {

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

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

System.***out***.println("Enter a number:");

**int** n= sc.nextInt();

System.***out***.println("Multiplication table of "+n);

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

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

}

sc.close();

}

}

Output; Enter a number:

5

Multiplication table of 5

5x1=5

5x2=10

5x3=15

5x4=20

5x5=25

5x6=30

5x7=35

5x8=40

5x9=45

5x10=50

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

Answer;

| **Method** | **Breaks Outer Loop?** | **Pros** | **Cons** |
| --- | --- | --- | --- |
| Unlabeled break | No | Simple | Only affects inner loop |
| Labeled break | Yes | Direct and clear | Slightly less common syntax |
| return (in method) | Yes | Clean, encapsulated logic | Must be inside a method |
| Flag variable | Yes (with extra code) | Explicit flow control | More verbose and less intuitive |

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

Answer;

| **Loop Type** | **Condition Check Time** | **Executes if Condition is Initially False?** | **Best Use Case** |
| --- | --- | --- | --- |
| for | Before each iteration | No (skips body entirely) | Known iteration counts, indexed access |
| while | Before each iteration | No | Condition-based, unknown iteration count |
| Do-while | After each iteration | Yes (executes at least once) | At-least-once logic, menus, input-validation |

1. Write a program that uses a switch-case to simulate a basic calculator.
2. **package** Control\_statement\_practice;
3. /\*2. WAP for simple calculator
4. \* like addition, subtraction , multiplication, division
5. \*/
6. **public** **class** calculator {
7. **public** **static** **void** main(String[] args) {
8. **int** a=3;
9. **int** b=7;
10. **char** operation='/';
11. **switch**(operation){
12. **case** '+':System.***out***.println("sum of the numbers "+(a+b));
13. **break**;
14. **case** '-':System.***out***.println("sub of the numbers "+(a-b));
15. **break**;
16. **case** '\*':System.***out***.println("mup of the numbers "+(a\*b));
17. **break**;
18. **case** '/':System.***out***.println("div of the numbers "+(a/b));
19. **break**;
20. **default**:
21. System.***out***.println("Invalid input");
23. }
24. }
25. }

Output; div of the numbers 0

## Java Keywords and Operators

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

Answer; The instanceof() keyword in Java is a binary operator used for **runtime type checking**—it evaluates whether an object is an instance of a specified **class**, a **subclass**, or an **interface**. The result is a **boolean** (true or false)

| **Scenario** | **Instanceof()Result** | **Notes** |
| --- | --- | --- |
| Object is of the same class | true | Checks direct type match |
| Object is subclass or superclass | true | Respects inheritance hierarchy |
| Object implements an interface | true | Validates interface implementation |
| Reference is null | false | Protects against null dereference |
| No relation between types | Compile-time error | Some incompatible comparisons won't compile |

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

Answer;

| **Use Case** | **==** | **. equals()** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| Primitives | Compares values | Not applicable directly |

|  |  |  |
| --- | --- | --- |
| Object references | Compares memory addresses | Checks content if overridden, else same as == |

|  |  |  |
| --- | --- | --- |
| Built-in classes (String, etc.) | May yield unexpected behavior | Correctly compares values (content) |

|  |  |  |
| --- | --- | --- |
| Custom types | Only checks same instance | Customize behavior by overriding |

1. Write a program using the ternary operator.

Code; **package** Control\_statement\_practice;

**import** java.util.Scanner;

**public** **class** EvenOddChecker {

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

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

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

**int** number = scanner.nextInt();

// Ternary operator checks if number is even or odd

String result = (number % 2 == 0) ? "Even" : "Odd";

System.***out***.println("The number " + number + " is " + result + ".");

scanner.close();

}

}

Output; Enter an integer: 56

The number 56 is Even.

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

Answer; In Java, the keywords this and super serve distinct and important roles within the context of **method overriding**, helping manage class hierarchies and clear up ambiguities in method invocation and variable access.

| **Keyword** | **Refers To** | **Method Overriding Behavior** |
| --- | --- | --- |
| this | Current class instance | Invokes subclass’s method |
| super | Parent class (superclass) | Invokes overridden method in superclass |

1. Explain bitwise operators with examples.

Answer; The practical breakdown of **bitwise operators in Java**, which allow you to manipulate data at the individual bit level—perfect for optimizations, flags, cryptography, graphics, and more.

| **Operator** | **Purpose** | **Example (5 = 0101)** | | | **Result** |
| --- | --- | --- | --- | --- | --- |
| & (AND) | Bitwise AND | 5 & 3 → 0101 & 0011 | | | 0001 = 1 |
| `|(OR) | Bitwise inclusive OR | | 5 | 3 → `0101 |
| ^ (XOR) | Bitwise exclusive OR | 5 ^ 3 → 0101 ^ 0011 | | | 0110 = 6 |
| ~ (NOT) | Bitwise complement | ~5 → …11111010 | | | -6 (2’s complement) |
| << (LShift) | Left shift (×2 per shift) | 5 << 2 → 0101 << 2 = 010100 | | | 20 |
| >> (RShift) | Signed right shift | Preserves sign bit | | | Varies |
| >>> UNSHR | Unsigned right shift | Always fills with 0 | | | Varies |