**Day 1**

**Variable , data type and taking input**

**What is a variable in Java?**

* **Answer**: A variable in Java is a container that holds data that can be changed during the execution of a program. It acts as a storage location in memory with a specific data type.

**2. What are the different types of variables in Java?**

* **Answer**: There are three types of variables in Java:
  1. **Local Variables**: Declared inside a method, constructor, or block.
  2. **Instance Variables**: Declared in a class but outside of methods, constructors, or blocks (non-static fields).
  3. **Static Variables**: Declared with the static keyword and shared across all instances of a class.

**3. What are the eight primitive data types in Java?**

* **Answer**:
  1. byte: 8-bit integer (Range: -128 to 127)
  2. short: 16-bit integer (Range: -32,768 to 32,767)
  3. int: 32-bit integer (Range: -2^31 to 2^31-1)
  4. long: 64-bit integer (Range: -2^63 to 2^63-1)
  5. float: 32-bit floating point
  6. double: 64-bit floating point
  7. char: 16-bit Unicode character
  8. boolean: true/false values

**4. What is the default value of a local variable?**

* **Answer**: Local variables do not have a default value in Java. They must be initialized before use, otherwise, the compiler will throw an error.

**5. What is typecasting in Java?**

* **Answer**: Typecasting is converting one data type into another. There are two types:
  1. **Implicit Typecasting (Widening)**: Automatic conversion of a smaller data type to a larger data type (e.g., int to double).
  2. **Explicit Typecasting (Narrowing)**: Manually converting a larger data type to a smaller data type (e.g., double to int).

**6. What is the difference between float and double?**

* **Answer**:
  + float is a single-precision 32-bit floating-point data type, while double is a double-precision 64-bit floating-point data type.
  + double has a greater precision and can store larger decimal values compared to float.

**How do you take input from the user in Java?**

To take input from the user, Java provides the Scanner class, which is part of the java.util package.

Example:

**java**

**Copy code**

**import java.util.Scanner;**

**public class InputExample {**

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

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

**// Taking input for an integer**

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

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

**// Taking input for a string**

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

**String name = sc.next();**

**System.out.println("You entered: " + number + " and " + name);**

**}**

**}**

**7. What is the difference between next() and nextLine() methods of Scanner class?**

* **next()**: Reads input till the first space or whitespace is encountered. It’s used to read a single word.
* **nextLine()**: Reads the entire line of input (including spaces) until the user hits the Enter key.

**8. What is the default value of variables in Java?**

* **Local variables**: No default value. They must be initialized before use.
* **Instance variables**: Default values depend on the data type:
  + int = 0
  + double = 0.0
  + char = '\u0000'
  + boolean = false
  + Object references (non-primitives) = null

**9. Can a variable name start with a number in Java?**

No, variable names in Java cannot start with a number. They must start with a letter (A-Z or a-z), a dollar sign ($), or an underscore (\_).

**10. What is the difference between primitive and reference data types?**

* **Primitive types**: Store the actual values (e.g., int, char, boolean).
* **Reference types**: Store references (addresses) to objects or arrays. Examples are String, arrays, and user-defined objects.

**11 Here are the key differences between float and double in Java:**

**1. Size:**

* **float**: 4 bytes (32 bits)
* **double**: 8 bytes (64 bits)

**2. Precision:**

* **float**: Single precision, which gives approximately 6-7 decimal digits of accuracy.
* **double**: Double precision, which gives approximately 15-16 decimal digits of accuracy.

**3. Range:**

* **float**: Can represent numbers roughly between 1.4×10−451.4 \times 10^{-45}1.4×10−45 to 3.4×10383.4 \times 10^{38}3.4×1038.
* **double**: Can represent numbers roughly between 4.9×10−3244.9 \times 10^{-324}4.9×10−324 to 1.7×103081.7 \times 10^{308}1.7×10308.

**4. Default Type:**

* **double** is the default type for decimal numbers in Java. If you write a number with a decimal point without specifying the type, Java assumes it is a double. To specify a float, you need to add an F or f at the end of the value.

Example:

java

Copy code

float f = 3.14F; // Float literal

double d = 3.14; // Double literal (default)

**5. Use Cases:**

* **float**: Used when memory savings are more critical than precision (e.g., in large arrays of floating-point numbers).
* **double**: Preferred when higher precision is required, such as in scientific computations.

**6. Performance:**

* On some processors, float operations might be slightly faster than double because of the smaller size. However, in most modern systems, double is typically used because of its better precision and the performance difference is often negligible

**12 the import statements**

import java.util.\*;

import java.lang.\*;

import java.io.\*;

In Java, the import statements are used to bring various classes, packages, or entire libraries into your code so that you can use their functionalities without having to specify their full package names. Here's when and why you would use each of the import statements you mentioned:

**1. import java.util.\*;**

* **Purpose**: Imports all the classes from the java.util package.
* **Common Uses**: You would use this when you need to work with utility classes such as:
  + ArrayList, HashMap, HashSet, LinkedList, etc.
  + Date, Calendar, Random, Scanner, etc.
* **Example**:

java

Copy code

import java.util.ArrayList;

import java.util.HashMap;

public class Example {

public static void main(String[] args) {

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

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

}

}

**2. import java.lang.\*;**

* **Purpose**: Imports all the classes from the java.lang package.
* **Common Uses**: The java.lang package is automatically imported by default, so you generally don't need to explicitly import it. It includes fundamental classes like:
  + String, Math, Integer, System, Thread, Exception, etc.
* **Example**:

java

Copy code

public class Example {

public static void main(String[] args) {

String str = "Hello, world!";

int result = Math.max(10, 20);

}

}

**3. import java.io.\*;**

* **Purpose**: Imports all the classes from the java.io package.
* **Common Uses**: You would use this when working with input and output operations, such as:
  + File handling with File, FileReader, FileWriter, BufferedReader, BufferedWriter, etc.
  + Streams like InputStream, OutputStream, ObjectInputStream, ObjectOutputStream, etc.
* **Example**:

java

Copy code

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

public class Example {

public static void main(String[] args) {

File file = new File("example.txt");

try (FileWriter writer = new FileWriter(file)) {

writer.write("Hello, world!");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Summary:**

* **java.util.\***: For utility classes like collections, dates, and random number generation.
* **java.lang.\***: For fundamental classes like String, Math, and System (automatically imported).
* **java.io.\***: For input and output operations, including file handling and stream processing.

**Day 2**

**Operators & Conditionals**

Operators in Java are special symbols that perform operations on variables and values. Java supports several types of operators, categorized into different groups based on the type of operation they perform.

**1. Arithmetic Operators**

These operators perform basic mathematical operations.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| + | Addition | a + b |
| - | Subtraction | a - b |
| \* | Multiplication | a \* b |
| / | Division | a / b |
| % | Modulus (remainder) | a % b |

**Example:**

java

Copy code

int a = 10, b = 3;

System.out.println(a + b); // Output: 13

System.out.println(a % b); // Output: 1

**2. Relational (Comparison) Operators**

These operators compare two values and return a boolean result (true or false).

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| == | Equal to | a == b |
| != | Not equal to | a != b |
| > | Greater than | a > b |
| < | Less than | a < b |
| >= | Greater than or equal to | a >= b |
| <= | Less than or equal to | a <= b |

**Example:**

java

Copy code

int a = 10, b = 20;

System.out.println(a > b); // Output: false

System.out.println(a <= b); // Output: true

**3. Logical Operators**

These operators perform logical operations on boolean expressions.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| && | Logical AND (true if both are true) | a && b |
| ` |  | ` |
| ! | Logical NOT (negates the boolean value) | !a |

**Example:**

java

Copy code

boolean a = true, b = false;

System.out.println(a && b); // Output: false

System.out.println(a || b); // Output: true

System.out.println(!a); // Output: false

**4. Assignment Operators**

These operators are used to assign values to variables.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| = | Assigns value | a = 5 |
| += | Adds and assigns | a += 5 (same as a = a + 5) |
| -= | Subtracts and assigns | a -= 5 |
| \*= | Multiplies and assigns | a \*= 5 |
| /= | Divides and assigns | a /= 5 |
| %= | Modulus and assigns | a %= 5 |

**Example:**

java

Copy code

int a = 10;

a += 5; // Now a is 15

System.out.println(a);

**5. Increment and Decrement Operators**

These operators are used to increment or decrement a value by 1.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| ++ | Increment (adds 1) | a++ or ++a |
| -- | Decrement (subtracts 1) | a-- or --a |

* a++ (Post-increment): Value is used first, then incremented.
* ++a (Pre-increment): Value is incremented first, then used.

**Example:**

java

Copy code

int a = 5;

System.out.println(a++); // Output: 5 (then a becomes 6)

System.out.println(++a); // Output: 7 (a is incremented before print)

**6. Bitwise Operators**

These operators perform operations on bits and are useful for low-level programming.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| & | Bitwise AND | a & b |
| ` | ` | Bitwise OR |
| ^ | Bitwise XOR | a ^ b |
| ~ | Bitwise Complement | ~a |
| << | Left Shift | a << 2 |
| >> | Right Shift | a >> 2 |
| >>> | Unsigned Right Shift | a >>> 2 |

**Example:**

java

Copy code

int a = 5; // Binary: 0101

int b = 3; // Binary: 0011

System.out.println(a & b); // Output: 1 (Binary: 0001)

System.out.println(a | b); // Output: 7 (Binary: 0111)

**7. Ternary (Conditional) Operator**

The ternary operator is a shorthand for if-else conditions. It has the form:  
**condition ? value\_if\_true : value\_if\_false;**

**Example:**

java

Copy code

int a = 10, b = 20;

int max = (a > b) ? a : b;

System.out.println(max); // Output: 20

**8. Instanceof Operator**

This operator is used to test whether an object is an instance of a specific class or subclass.

**Example:**

java

Copy code

String str = "Hello";

boolean result = str instanceof String;

System.out.println(result); // Output: true

**9. Type Casting Operators**

These are used to convert one data type into another.

* **Implicit Casting**: Automatically done when converting a smaller type to a larger type.
* **Explicit Casting**: Manually done when converting a larger type to a smaller type.

**Example:**

java

Copy code

int a = 10;

double d = (double) a; // Explicit casting

1. **What are conditionals in Java?**

* **Answer**: Conditionals in Java refer to statements that allow the program to make decisions and execute different code paths based on certain conditions. These include if, if-else, if-else if, and switch statements.

**2. Explain the if-else statement in Java with an example.**

* **Answer**: The if-else statement evaluates a condition, and if the condition is true, it executes the block of code inside the if statement. If it is false, the code inside the else block is executed.
* **Example**:

java

Copy code

int num = 10;

if (num > 0) {

System.out.println("Positive number");

} else {

System.out.println("Negative number");

}

**3. Can you nest if-else statements in Java?**

* **Answer**: Yes, you can nest if-else statements, meaning you can place an if-else statement inside another if-else statement to check multiple conditions.

**4. What is the difference between if-else and switch statements?**

* **Answer**:
  + if-else is used to evaluate boolean expressions and can work with conditions involving logical operators.
  + switch is more suitable when comparing a single variable against a set of constants, usually for equality checks.
  + if-else can handle complex conditions, while switch is generally more readable when there are multiple possible outcomes for a single variable.

**5. Can a switch statement be used with strings in Java?**

* **Answer**: Yes, starting from Java 7, switch statements can be used with String objects. The switch compares the value of the string against the specified case labels.
* **Example**:

**java**

**Copy code**

**String day = "Monday";**

**switch (day) {**

**case "Monday":**

**System.out.println("Start of the work week");**

**break;**

**case "Friday":**

**System.out.println("End of the work week");**

**break;**

**default:**

**System.out.println("Mid-week");**

**}**

**6. What will happen if there is no break statement in a switch case?**

* **Answer**: If the break statement is omitted in a switch case, Java will continue executing the following cases (fall-through behavior) until it encounters a break or the end of the switch block.

**7. What is the ternary operator in Java? How does it work?**

* **Answer**: The ternary operator is a shorthand for if-else and takes the form: condition ? expression1 : expression2. If the condition is true, expression1 is executed, otherwise, expression2 is executed.
* **Example**:

**java**

**Copy code**

**int a = 10, b = 20;**

**int min = (a < b) ? a : b;**

**System.out.println("Minimum: " + min);**

**8. What is the difference between == and .equals() in Java conditionals?**

* **Answer**: == checks for reference equality (whether two object references point to the same object), while .equals() checks for value equality (whether two objects are meaningfully equal). For primitive types, == checks value equality.

**9. How do logical operators work with conditionals in Java?**

* **Answer**: Logical operators (&&, ||, and !) are used in conjunction with conditionals to combine multiple conditions or to negate conditions.
  + && (AND): Returns true if both conditions are true.
  + || (OR): Returns true if at least one condition is true.
  + ! (NOT): Reverses the result of the condition.

**10. Explain the switch statement limitations in Java.**

* **Answer**: The switch statement has the following limitations:
  + It can only work with byte, short, int, char, String, and enums.
  + It doesn't support floating-point numbers (float or double).
  + It evaluates based on exact matches only and cannot handle complex boolean expressions or ranges

**11. how you can use the Scanner class to take input for different types of variables:**

**Int**

int num = scn.nextInt();

**double**

double d = scn.nextDouble();

**String for single word**

String str = scn.next();

**String for entire line**

String line = scn.nextLine();

**Char for single charater**

char ch = scn.next().charAt(0);

**day 3**

**loop**

.

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

* **Answer**: Java provides three types of loops:
  + **for loop**: Used when the number of iterations is known.
  + **while loop**: Used when the number of iterations is unknown, and it checks the condition before entering the loop.
  + **do-while loop**: Similar to the while loop but checks the condition after executing the loop at least once.

**2. What is the difference between for loop and while loop?**

* **Answer**:
  + **for loop**: Best used when the number of iterations is known. It initializes, checks the condition, and increments in one line.
  + **while loop**: Best when the number of iterations is unknown. It only checks the condition before entering the loop and iterates as long as the condition is true.

**3. Explain the structure of a for loop in Java.**

* **Answer**:

java

Copy code

for (initialization; condition; update) {

// Code to be executed

}

Example:

java

Copy code

for (int i = 0; i < 5; i++) {

System.out.println(i);

}

**4. Can you write an example of a while loop in Java?**

* **Answer**:

java

Copy code

int i = 0;

while (i < 5) {

System.out.println(i);

i++;

}

**5. What is the difference between while and do-while loops?**

* **Answer**:
  + In a **while loop**, the condition is checked before entering the loop.
  + In a **do-while loop**, the condition is checked after executing the loop body, so the loop will always execute at least once.

**Example of do-while loop**:

java

Copy code

int i = 0;

do {

System.out.println(i);

i++;

} while (i < 5);

**6. How can you exit a loop prematurely in Java?**

* **Answer**: You can use the **break** statement to exit a loop prematurely. It is often used when a certain condition is met inside the loop.

**Example**:

java

Copy code

for (int i = 0; i < 10; i++) {

if (i == 5) {

break; // Loop terminates when i equals 5

}

System.out.println(i);

}

**7. What is the continue statement in Java loops?**

* **Answer**: The **continue** statement skips the current iteration of a loop and proceeds with the next iteration.

**Example**:

java

Copy code

for (int i = 0; i < 5; i++) {

if (i == 2) {

continue; // Skip the current iteration when i equals 2

}

System.out.println(i);

}

**8. How can you implement an infinite loop in Java?**

* **Answer**: An infinite loop can be created by omitting the loop condition or making the condition always true. Be careful with infinite loops as they can make the program unresponsive.

**Example**:

java

Copy code

while (true) {

// Infinite loop

}

**9. How do nested loops work in Java?**

* **Answer**: A **nested loop** is a loop inside another loop. The inner loop will run completely for each iteration of the outer loop.

**Example**:

java

Copy code

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

for (int j = 1; j <= 2; j++) {

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

}

}

**10. What is the time complexity of nested loops?**

* **Answer**: The time complexity of a nested loop is the product of the complexities of the individual loops. For example, if both loops run n times, the time complexity is **O(n²)**.

**11. How do you loop over an array using a for-each loop in Java?**

* **Answer**: The for-each loop is used to iterate over arrays or collections.

**Example**:

java

Copy code

int[] numbers = {1, 2, 3, 4, 5};

for (int num : numbers) {

System.out.println(num);

}

**12. What happens if you omit the initialization, condition, or update expressions in a for loop?**

* **Answer**:
  + Omitting **initialization** or **update** makes the loop still valid; you just need to handle initialization and updates inside the loop.
  + Omitting the **condition** makes the loop infinite (since the condition defaults to true).

**Example** (infinite loop by omitting the condition):

java

Copy code

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

System.out.println(i);

}

**13. How can you iterate over a collection in Java?**

* **Answer**: You can use a for-each loop or an Iterator to iterate over a collection.

**Example using for-each**:

java

Copy code

List<String> list = Arrays.asList("A", "B", "C");

for (String item : list) {

System.out.println(item);

}

**14. Can you use a for loop with multiple variables in Java?**

* **Answer**: Yes, a for loop can use multiple variables by separating them with commas in the initialization and update expressions.

**Example**:

java

Copy code

for (int i = 0, j = 10; i < j; i++, j--) {

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

}

**15. How do you reverse a for loop in Java?**

* **Answer**: You can reverse a for loop by adjusting the initialization, condition, and update expressions.

**Example**:

java

Copy code

for (int i = 5; i >= 0; i--) {

System.out.println(i);

}