Java Notes Part2:

Comments In Java:

- Comments in Java are used to add explanatory notes to the Java code.
- The compiler does not execute these notes, and therefore, they are not part of the program's functionality. Instead, they serve as notes that help developers understand the code and make it more readable.

There are three types of comments in Java:

1. **Single-line comment:**
2. **Multi-line comment:**
3. **Javadoc comment or Documentation comment:**

Single-Line comment: //

- Single-line comments start with two forward slashes (//) and continue to the end of the line. Anything after the slashes is considered a comment and is ignored by the compiler. Here are a few examples:

```
int age; // Represents the age of the customer

//Represents the maximum login retry attempts allowed
public static final int MAX_ATTEMPT = 3;
```

Multi-line comment: / /

- Multi-line comments can be used to explain a complex code snippet or to comment multiple lines of code at a time.

  Example:

```
/*
This class represents a customer in the Bank app.
It contains information such as firstName,
lastName, DOB, address, and account details.
*/
public class Customer {

/*
This method calculates the area of the rectangle based on
```

```
on given length and width. The logic is going to have
Multiply two given input method arguments and return
the same to the caller
*/
        public double calculateRectangleArea (double length, double width) {
                return length * width;
        }
}
```

The Javadoc comment or documentation comment: / */**

- **Javadoc** comments are a special type of comment used in the Java programming language to document code.
- The JDK provides the **javadoc** command-line tool to extract the documentation comments from the source code and generate documentation in HTML format.

```
/**
* Takes two {@code int} numbers as input * and add them.
*  <p>
*        <b> For example, 2 + 3 =5 </b>
* @param num1 represents the first number
* @param num2 represents the second number
* @return sum value of first and second number </p> */
public int sum (int num1, int num2) {
        return num1 + num1;
}
```

Refer to the Java API documentation:

Java® Platform, Standard Edition & Java Development Kit Version 21 API Specification

Java Statements:

- In Java programming, a statement denotes an operation, such as assigning the sum of variables a and b to c, outputting a message to the standard output, iterating through a list of values, conditionally executing a code block, or writing data to a file, etc.
- These statements are constructed using **keywords**, **operators**, and **expressions**. Depending on the action performed, statements in Java are broadly classified into the following categories.

1. General Purpose Statements

- Declaring variables, methods, and classes
- Creating objects
- Accessing variables and methods
- Using an expression.

Examples:

```
double average; // declaring a variable

int count = 5; // declaring and initializing

count++;  // Incrementing the value

average = 10.55+6; //assigning a variable

System.out.println("Hello");

Student s = new Student();
s.display();
```

2. Conditional (Decision-Making) Statements:

- Used to execute code **based on conditions**.
  - `if`
  - `if-else`
  - `if-else if-else`
  - `nested if`
  - `switch`
  - `switch expression (Java 14+)`

3. Iterative (Looping) Statements

- Used to repeat execution.
  - `for`
  - `while`
  - `do-while`
  - `nested loops`

4. Transfer (Branching) Statements

- Used to **change the flow abruptly**.
    - `break`
    - `continue`
    - `return`

5. Exception Handling Statements

- Used to handle runtime errors.
    - `try`
    - `catch`
    - `finally`
    - `throw`
    - `throws`

6. Synchronized Statements

- Used in **multithreading**.
    - `synchronized` methods
    - `synchronized` blocks

Conditional Statements in Detail:

1. **if Statement:**
    - The simplest form of a conditional statement. It checks a condition and executes the block of code inside it if the condition is **true**.

Syntax

```
if(condition){

        // executes if the condition is true
}
```

Example:

```
//General Statement

int age = 20;

//General Statement
System.out.println("The age is"+age);


if (age >= 18) {
        //conditional statement
         System.out.println("You are eligible to vote.");
}

//General Statement
System.out.println("Thanks");
```

Note: Although it is feasible to omit the curly braces {} of an if statement when there is only a single Java statement within it, it is not advisable since it can lead to decreased code readability.

2. if-else Statement:

- It provides two execution paths: one if the condition is true, and another if the condition is false.

Syntax:

```
if (condition) {
        // Code to execute if the condition is true
} else {
        // Code to execute if the condition is false
}
```

Example:

```
int age = 16;
```

```
if (age >= 18) {
        System.out.println("You are eligible to vote.");
} else {
        System.out.println("You are not eligible to vote.");
}
```

3. if-else if-else Statement: (if-else ladder)

- It allows you to check multiple conditions. If the first condition is false, it checks the next condition, and so on, until a true condition is found or the last else is executed.

Syntax:

```
if (condition1) {

        // Code to execute if condition1 is true
} else if (condition2) {
        // Code to execute if condition2 is true
} else {
        // Code to execute if none of the conditions are true
}
```

- JVM checks conditions **top-to-bottom** and stops at the **first true condition**.

Example:

```
int score = 85;

 if (score >= 90) {
  System.out.println("Grade A");
} else if (score >= 70) {
    System.out.println("Grade B");
 } else {
    System.out.println("Grade C");
 }
```

4. Nested if-else Statements in Java

- A nested if-else is an if statement inside another if or else block. It helps check multiple conditions one after another.

Syntax:

```
if (condition1) {

        if (condition2) {
            // Code if both condition1 and condition2 are true
        } else {
            // Code if condition1 is true but condition2 is fal
        }
} else {
    // Code if condition1 is false
}
```

Example:

```
int age = 20;

boolean hasTicket = true;

if (age >= 18) {

    if (hasTicket) {
        System.out.println("You can enter the concert.");
    } else {
        System.out.println("You need a ticket to enter.");
    }

} else {
        System.out.println("You are not allowed to enter.");
}
```

Note: We can combine the above nested if-else using logical && operator in generalize way.

```
if(age >=18 && hasTicket){

        System.out.println("You can enter the concert");
}
else{
        System.out.println("You need to enter a ticket or you are not allowed to enter");
}
```

Activity 1: Multiple Ways to Solve Same Problem

```
int marks = 75;

if(marks >= 60){
        System.out.println("You are Pass");
}else{
        System.out.println("You are Fail");
}


//The above program we can write without using the else part also.

String result = "You are Fail";

if(marks >= 60){
        result = "You are Pass";
}

System.out.println(result);

//The Same program we can write without using the if part also

String result = (marks >= 60) ? "You are Pass" : "You are Fail";

System.out.println(result);
```

Note: The ternary operator is an expression, whereas `if-else` is a statement.

Activity2:

- Write a program to print :
  - If the number is divisible by 5, then: **Welcome**
  - If the number is divisible by 3, then: **Hello**
  - If the number is divisible by 5 and 3 both: **Masai**
  - Otherwise: **Invalid number**

Wrong Order (Logical Bug):

```
int num = 10

if(num % 5 == 0){
```

```
    System.out.println("Welcome");
}
else if(num % 3 == 0){
    System.out.println("Hello");
}
else if(num % 5 == 0 && num % 3 == 0){
    System.out.println("Masai");
}
```

❌ Problem with the above approach:

- For num = 15, the first condition (num % 5 == 0) becomes true.
- The program prints "Welcome" and exits the if-else chain.

Correct Order (Most Restrictive First):

```
int num =10;


if(num % 5 == 0 && num % 3 == 0){
    System.out.println("Masai");
}
else if(num % 5 == 0){
    System.out.println("Welcome");
}
else if(num % 3 == 0){
    System.out.println("Hello");
}
else{
    System.out.println("Invalid number");
}
```

Rule: Always check **combined / strict conditions first**.

5. **switch Statement:**

- The switch statement provides a way to test a variable or expression against multiple possible values. It is often more readable than multiple if-else statements when there are many conditions.

  Syntax:

```
switch (expression) {

 case value1:
          // Code to execute if expression equals value1
          break;
 case value2:
          // Code to execute if expression equals value2
          break;
 default:
          // Code to execute if the expression doesn't match any case
}
```

Example:

```
int day = 3;


 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");

}
```

Rules for switch case:

1. The expression value derived in a switch statement must be of type byte, short, char, int, enum, or String.
2. Switch case expression does not allow boolean, long, float, and double data types.
3. The break statement is used to exit the switch statement once the corresponding code segment is executed. otherwise all the following cases would be executed.

4. Two case labels in a switch statement cannot be the same. It will result in a compilation error.

switch with Multiple Case Labels

```
char c = 'x';

switch (c) {
   case 'a':
   case 'e':
   case 'i':
   case 'o':
   case 'u':
      System.out.println("It is a vowel");
      break;
   default:
      System.out.println("It is a consonant");
}
```

switch Expression:

- **Java 14** introduced a more concise form of the switch statement called switch expression.
- It uses an arrow (→) and does not use a break statement, resulting in a less verbose syntax.
- It also supports multiple constants per case, separated by a comma.

Example:

```
package com.masai;

public class Demo {

        public static void main(String[] args) {

                int day = 7;

                switch (day) {
                        case 1 -> System.out.println("Monday");

                        case 2 -> System.out.println("Tuesday");

                        case 3 -> System.out.println("Wednesday");

                        case 4 -> System.out.println("Thursday");
```

```
                case 5 -> System.out.println("Friday");

                case 6, 7 -> System.out.println("Weekend holiday");

                default -> System.out.println("Invalid day");
            }
        }
}
```

If needed, we can use a switch expression to return a value also:

Example:

```
package com.masai;

public class Demo {

    public static void main(String[] args) {

        int day = 7;

        String result= switch (day) {

            case 1 -> "Monday";

            case 2 -> "Tuesday";

            case 3 -> "Wednesday";

            case 4 -> "Thursday";

            case 5 -> "Friday";

            case 6, 7 -> "Weekend holiday";

            default -> "Invalid day";
        };
        System.out.println(result);
    }
}
```

Using `yield` in switch Expression:

- The yield keyword facilitates the termination of a switch expression by providing a value that subsequently becomes the overall value of the switch expression.

  Example:

```java
package com.masai;

public class Demo {

	public static void main(String[] args) {

	int day = 7;

	String result= switch (day) {
		case 1 -> {
			System.out.println("Monday");
			yield "Mon";
		}

		case 2 -> {
			System.out.println("Tuesday");
			yield "Tue";
		}

		case 3 -> {
			System.out.println("Wednesday");
			yield "Wed";
		}

		case 4 -> {
			System.out.println("Thursday");
			yield "Thu";
		}

		case 5 -> {
			System.out.println("Friday");
			yield "Fri";
		}

		case 6, 7 -> {
			System.out.println("Weekend holiday");
			yield "Weekend";
		}

		default -> {
			System.out.println("Invalid day");
			yield "Invalid";
		}
	};

	System.out.println(result);
	}
}
```

Iterative Statement:

- These statements are able to allow the JVM to execute a set of instructions repeatedly on the basis of a particular condition.

1. **while loop:**

    - The while loop is used when the number of iterations is not known in advance. The loop continues as long as the specified condition is true. The condition is checked before executing the loop body.
    - A loop should have a starting point and an ending point(condition), also.
    - A loop that does not have any ending point is known as an infinite loop.

    Syntax:

```
while (condition) {

        // code to be executed
}
```

    Example: Using a while loop printing 1 to 5;

```
int num = 1;

while(num <= 5){
        System.out.println(num);
        num++;
}
//here if we comment out the num++, then it will become an infinite loop.
```

    Example: An infinite while loop:

```
while(true){

        System.out.println("I can print infinitely");
}
```

2. **do-while loop:**

    - The **do-while** loop is similar to the while loop, but it guarantees that the loop body will execute at least once, regardless of whether the condition is true.

- This happens because the condition is checked after executing the loop body.

Syntax:

```
do {
        // code to be executed
} while (condition);
```

Example:

```
int i = 1;
do {
        System.out.println(i);
        i++;
} while (i > 5);
```

3. for loop:

- The for loop is used when the number of iterations is known beforehand. It consists of three parts: initialization, condition, and update in a single place. It is ideal for iterating over a range or a known number of elements.
- It is one of the most commonly used loops in Java.

Syntax:

```
for (initialization; condition; update) {
        // code to be executed
}
```

Example: printing 1 to 5 using a for loop:

```
for(int i=1;i<=5;i++){
        System.out.println(i);
}
```

Example: Display the 5 tables:

```
int num =5;
```

```
for(int i=1; i<=10;i++){
        System.out.println(num+" * "+i+" = "+ (num*i));
}
```

Example: an infinite for loop:

```
for(; ;){

        System.out.println("I can print infinitely");
}
```

Note: all the 3 part (initialization, condition, and update) are optional, and by default the 2nd part is treated as true. And the 1st part will be executed only one time.

Example:

```
int i=1;

for(System.out.println("Hello"); i<=5; System.out.println("Welcome")) {
        i++;
}

//Output:
Hello
Welcome
Welcome
Welcome
Welcome
Welcome
```

Explanation

- "Hello" → printed once (initialization)
- "Welcome" → printed after each iteration (update)

Example: Print the sum of all even numbers between 1 and 50 using a for loop.

```
int sum = 0;

 for (int i = 1; i <= 50; i++) {
        if (i % 2 == 0) {
                sum += i;
        }
```

```
}

 System.out.println("Sum of even numbers: " + sum);
```

4. **Nested loop:**

- A nested loop is a loop inside another loop. It is used when you need to perform iterations inside another set of iterations, like when dealing with multi dimensional arrays or complex patterns or working with matrix(rows and columns).
- We can nest any type of loop inside any other loop.

Example: for loop inside another for loop:

```
package com.masai;

public class Demo {

	public static void main(String[] args) {

		for(int over=1; over <=20; over++) {

			System.out.println("Starting of Over"+over);

			for(int ball=1;ball <=6; ball++) {
				System.out.println("Bowler bowl"+ball);
			}

			System.out.println("Ending of Over "+over);

		}
	}
}
```

Example: printing matrix: row and columns (5 row and 3 columns)

```
for(int i=1;i<=5;i++) {
	for(int j=1; j<=3;j++) {
		System.out.print(j+" ");
	}
	System.out.println();
}
```

Output:

```
1 2 3
1 2 3
1 2 3
1 2 3
1 2 3
```

```
//Another example to display both i and j values

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

        for (int j = 1; j <= 3; j++) {
                System.out.print(i + "," + j + "  ");
        }

        System.out.println();
}
```

Output:

```
1,1   1,2   1,3
2,1   2,2   2,3
3,1   3,2   3,3
```

```
//In this example, the outer loop runs 3 times, and for each iteration of the outer loop
//loop, the inner loop runs 3 times.
```

Example: Printing a pyramid of asterisks *

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

        for(int j=1; j<=i;j++) {
                System.out.print("* ");
        }
        System.out.println();
}
```

5. break statements:

- The break statement is used to exit the loop prematurely when a certain condition is met.

Example: Using break inside the **while** loop:

```
int i=1;

while(true) {

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

        if(i == 6) {
                break;
        }
}
```

Example: Using break inside the **for** loop:

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

        if (i == 3) {
                break;  // Exit the loop when i equals 3
        }
        System.out.println(i);
 }
```

Example: Using break inside the **nested loop:**

- If we provide a break statement in a nested loop, then that break statement is applicable only to the nested loop; it will not have any effect on the outer loop.

```
package com.masai;

public class Main{

        public static void main(String[] args){

                for(int i=0; i<10; i++)// Outer loop
                {
                        for(int j=0; j<10; j++)// Nested Loop
                        {
                                if(j==5)
                                  break;
                                System.out.println(i+" "+j);
```

```
                    } // end of nested loop
                }//end of outer loop

            }
        }
```

OUTPUT:

```
0 0
0 1
0 2
0 3
0 4
-------
-------
-------
9 0
9 1
9 2
9 3
9 4
```

6. **continue Statement:**

- The continue statement skips the current iteration and moves to the next iteration of the loop.

Example: continue in a for loop:

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

        if (i == 3) {
            continue;  // Skip when i equals 3
        }
        System.out.println(i);
}
```

Output:

```
1
2
4
5
```

Example: Printing all the **odd** numbers using continue:

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

{
        if(i %2 == 0)
                continue;

        System.out.println(i);
}
```

Note: Any statement written after a `break` or `continue` inside the same block is an unreachable statement, causing a compile-time error.

7. **return Statement:**

    - The “return“ statement is used to exit a method and return a value

Example:

```
public String getStudentGrade(int marks) {

        if(marks >= 40) {
                return "Passed";
        }
        else {
                return "Failed";
        }
}
```

Students Task:

Assignment 1 (Beginner)

Write a program to:

- Print numbers from 1 to 100

- Skip multiples of 5
- Stop when the number reaches 70

Assignment 2 (Logic Builder)

Write a program:

- Input marks
- Print:
    - ≥90 → A
    - ≥75 → B
    - ≥60 → C
    - Else → Fail
- Use a **switch expression**

Assignment 3

What will be the output?

```
int x = 5;

if (x++ > 5)
   System.out.println("Hello");
else
   System.out.println("Bye");

System.out.println(x);
```

Assignment 4 (Creative)

Create a **College Day Planner** using switch:

- Monday → Java
- Tuesday → Python

- Wednesday → DBMS
- Thursday → OS
- Friday → Revision
- Weekend → Sleep

Assignment 5:

What will be the output?

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

        System.out.println(i);
          i++;
}
```

Assignment 6:

What will be the output?

```
int a = 10;

if(a = 20){
    System.out.println("Yes");
}
```

Assignment 6:

What will be the output?

```
int a = 10;

if(a = 20){
    System.out.println("Yes");
}
```

Assignment 7:

What will be the output?

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

    if(i==2) continue;
```

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

Assignment 8:

- Print this without using numbers:
    - 1 2 3 4 5

Assignment 9:

- Print the following star pattern without using a nested loop.

```
*
* *
* * *
* * * *
* * * * *
```

Assignment 10:

- Print numbers from 10 to 1 using a `while loop`
- Check if a number is positive, negative, or zero
- Count digits in a number
- Check a palindrome number
- Print the Fibonacci series up to N