1️⃣ WHY CONDITIONS & LOOPS EXIST

Programming is about **decision making** and **repetition**.

- **Conditions** → Decide *what to execute*
- **Loops** → Decide *how many times to execute*

Without them:

- Programs would be **linear only**
- No validation, no automation, no logic

Real-world analogy:

- Condition → *If traffic light is red, stop*
- Loop → *Check attendance for every student*

2️⃣ CONDITIONS IN JAVA

2.1 if Statement

Purpose: Execute code only when condition is true

java
if (condition) {
// code
}

Used when **only one decision** is required

Example:
java
if (age >= 18) {
System.out.println("Eligible to vote");
}

2.2 if – else

Purpose: Choose between two paths

java
if (condition) {

```
// true block
} else {
// false block
}
```

Example:

```
java
if (num % 2 == 0) {
System.out.println("Even");
} else {
System.out.println("Odd");
}
```

2.3 if – else if – else

Purpose: Handle multiple conditions

Execution flow:

- Top to bottom
- First true condition executes
- Remaining conditions are skipped

```
java
if (marks >= 90) {
grade = "A";
} else if (marks >= 75) {
grade = "B";
} else {
grade = "Fail";
}
```

Order matters (INTERVIEW FAVORITE)

2.4 Nested if

Condition inside another condition

Used when:

- One decision depends on another

Example: **Leap year logic**

```
java
if (year % 4 == 0) {
```

```
if (year % 100 != 0 || year % 400 == 0) {
// leap year
}
}
```

2.5 switch Statement

Purpose: Clean alternative to long else-if chains

```
java
switch (value) {
case 1:
// code
break;
default:
// code
}
```

Supports:

- int, char, String, enum

Does NOT support:

- boolean
- ranges (>, <)

Missing break causes **fall-through** (INTERVIEW TRAP)

---

3 LOOPS IN JAVA

3.1 for Loop

Best when number of iterations is known

```
java
for (init; condition; increment) {
// code
}
```

Execution order:

1. Initialization (once)
2. Condition check
3. Loop body

4. Increment

3.2 while Loop

Condition-based repetition

java
while (condition) {
// code
}

If condition never becomes false → **infinite loop**

3.3 do-while Loop

Executes at least once (VERY IMPORTANT)

java
do {
// code
} while (condition);

Used in:

- Menus
- Retry logic

3.4 Enhanced for-each Loop

Used for **arrays & collections**

java
for (int x : arr) {
// read-only access
}

Cannot:

- Modify array index
- Traverse backward

---

4 LOOP CONTROL STATEMENTS

break

- Exits loop immediately

continue

- Skips current iteration

Labeled break

- Exits outer loop (advanced)

java
outer:
for (...) {
for (...) {
break outer;
}
}

---

5 MEMORY & EXECUTION FLOW (INTERVIEW)

- main() → Stack
- Loop variables → Stack
- Objects → Heap
- Condition checks → CPU

Each iteration:

- Condition evaluated
- Stack frame reused

---

6 REAL-WORLD USE CASES

Concept	Example
if	Login validation
switch	Menu selection
for	Attendance system
while	Bank balance check
do-while	ATM menu

7 COMMON MISTAKES (VERY IMPORTANT)

Using = instead of ==
Missing break in switch
Infinite loops
Wrong condition order
Using for-each when index needed

---

8 INTERVIEW TRAPS & QUESTIONS

Trap 1

java
if (false)
System.out.println("Hi");
System.out.println("Bye");

Output: Bye

Trap 2

java
int i = 1;
while (i++ <= 5);
System.out.println(i);

Output: 7

Trap 3

java
switch (2) {
case 1:
System.out.print("A");
case 2:
System.out.print("B");
}

Output: B

9️⃣ WHEN TO USE WHAT (INTERVIEW ANSWER)

- Known iterations → for
- Condition based → while
- Minimum one execution → do-while
- Multiple fixed choices → switch

"Conditions control **decision flow**, loops control **repetition flow**, together they form the **logic backbone of Java programs**."