1. Conditions (decision making)

A **condition** is a boolean expression that decides *whether* code runs.

> The computer does not “think”.
> It evaluates → true or false → acts.

1.1

if

statement (basic gate)

```
if (condition) {
    // runs only if condition is true
}
```

Example:

```
int age = 20;

if (age >= 18) {
    System.out.println("Eligible");
}
```

Execution flow:

- Evaluate age >= 18
- If true → execute block
- If false → skip block

1.2

if–else

(two paths)

```
if (condition) {
    // true path
} else {
    // false path
}
```

Example:

```
if (marks >= 50) {
    System.out.println("Pass");
} else {
    System.out.println("Fail");
}
```

Only **one block** runs. Never both.

1.3

else if

ladder (multiple choices)

```
if (score >= 90) {
    grade = 'A';
} else if (score >= 75) {
    grade = 'B';
} else {
    grade = 'C';
}
```

Key rule:

- Conditions are checked **top to bottom**
- First true wins
- Order matters

Interview bug:

```
if (score >= 50) ...
else if (score >= 90) ...
```

This makes >=90 unreachable.

1.4 Nested conditions (condition inside condition)

```
if (age >= 18) {
    if (hasID) {
        allowEntry();
    }
}
```

Used when **one condition depends on another**.

Rule of thumb:

- Nest only when logically required

- Otherwise use logical operators (&&)

1.5 Switch (value-based decisions)

```
switch(day) {
    case 1: System.out.println("Mon"); break;
    case 2: System.out.println("Tue"); break;
    default: System.out.println("Invalid");
}
```

Use when:

- One variable
- Many fixed values
- Cleaner than long else if

---

2. Loops (repetition engine)

A **loop** repeats code **until a condition breaks it**.

Loops = condition + repetition + state change

If state never changes → **infinite loop** (classic beginner disaster).

2.1

for

loop (count-controlled)

Best when you **know how many times** to run.

```
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

Anatomy:

- initialization → int i = 0
- condition → i < 5

- update → i++

This is the backbone of **arrays and DSA**.

2.2

while

loop (condition-controlled)

Best when repetitions depend on a **condition**.

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
```

Rule:

- Condition checked **before** every iteration
- Might run zero times

Used in:

- binary search
- two pointers
- input-driven logic

2.3

do–while

(run at least once)

```
do {
    System.out.println("Hello");
} while (condition);
```

Runs once **even if condition is false**.

Rare in interviews, but good to know.

---

3. Loop control statements

break

Stops the loop immediately.

```
if (found) break;
```

continue

Skips current iteration, moves to next.

```
if (x < 0) continue;
```

Used heavily in filtering logic.

---

4. Conditions + Loops = DSA core

Example: count even numbers

```
int count = 0;

for (int i = 0; i < arr.length; i++) {
    if (arr[i] % 2 == 0) {
        count++;
    }
}
```

Pattern:

1. Initialize state (count)
2. Loop over data
3. Condition check
4. Update state

Every array/string problem follows this skeleton.

---

5. Infinite loop traps (exam favorite)

```
while (i < 10) {
    System.out.println(i);
}
```

Why infinite?

- i never changes

Always ask:

- What changes the condition?
- When does it stop?

---

6. Interview mental checklist

Before writing any loop:

- What is the **start**?
- What is the **end**?
- What changes every iteration?
- When does it **stop**?

Before writing any condition:

- What exactly becomes true or false?
- Are all cases covered?

---

7. One-screen revision summary

- Conditions decide **whether**
- Loops decide **how many times**
- Every loop needs a changing variable
- Every bug is usually a bad condition or bad update