Flow Control in Java

Flow control statements determine **the order in which statements are executed** in a Java program.
They are **foundational**, **logic-critical**, and **heavily tested** in Oracle certification exams—especially edge cases involving **conditions, fall-through, scope, and unreachable code**.

Classification of Flow Control Statements

Java flow control statements are grouped into **three major categories**:

1. Selection Statements
2. Iteration Statements
3. Transfer Statements

1️⃣ Selection Statements

Selection statements allow **conditional execution** of code blocks.

1.1 `if` Statement

Syntax

```
if (condition) {
    // statements
}
```

Rules

- `condition` **must be boolean**
- No numeric or object conditions allowed

```
if (true) { }        // valid
if (10) { }          // ❌ CE
if ("true") { }      // ❌ CE
```

Without Braces (Exam Trap)

```
if (true)
    System.out.println("A");
    System.out.println("B");
```

✔ Output:

```
A
B
```

> Only the **immediate next statement** is part of `if`

1.2 `if-else` Statement

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

Dangling `else` Rule (Very Important)

> An `else` always associates with the **nearest unmatched `if`**

```
if (true)
    if (false)
        System.out.println("A");
    else
        System.out.println("B");
```

✔ Output:

```
B
```

1.3 `switch` Statement

General Syntax

```
switch (expression) {
    case value1:
        statements;
        break;
    case value2:
        statements;
        break;
    default:
        statements;
}
```

Allowed Types for `switch` Expression

Java Version	Allowed Types
Java 5	`byte`, `short`, `char`, `int`
Java 7+	+ `String`
Java 5+	+ `enum`

❌ Not allowed:

- `long`
- `float`, `double`
- `boolean`
- Objects (except `String`)

Case Label Rules

- Must be **compile-time constants**
- Must be **unique**
- Must be **assignment-compatible**

```
case 10:        // valid
case 10+20:     // valid
case x:         // ❌ if x is variable
```

`default` Case

- Optional
- Can appear **anywhere** inside switch
- Executes if no case matches

Fall-Through Behavior (High-Weight Topic)

```
int x = 1;
switch (x) {
    case 1:
        System.out.println("One");
    case 2:
        System.out.println("Two");
}
```

✔ Output:

```
One
Two
```

`break` is required to stop fall-through

`switch` with `String`

```
String s = "A";
switch (s) {
    case "A":
        System.out.println("Apple");
        break;
}
```

✔ Comparison uses `.equals()`, not `==`

Modern Note (Java 14+)

Java introduces **switch expressions**, but **Oracle OCA/OCP exams usually test classic switch**.

```
int result = switch (day) {
    case 1,2,3,4,5 -> 1;
    case 6,7 -> 2;
    default -> 0;
};
```

2️⃣ Iteration Statements (Loops)

Used to **execute statements repeatedly**.

2.1 `while` Loop

```
while (condition) {
    statements;
}
```

Rules

- Condition must be boolean
- Executes **zero or more times**

```
while(false) {
    System.out.println("Hi");
}
```

✔ Valid
❌ Unreachable code inside loop causes **compile-time error**

2.2 `do-while` Loop

```
do {
    statements;
} while (condition);
```

Key Difference

✔ Executes **at least once**, even if condition is false

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

✔ Output:

```
Hello
```

2.3 `for` Loop

Traditional `for`

```
for (initialization; condition; increment) {
    statements;
}
```

Rules (Exam Focus)

- All three parts are **optional**

- Semicolons are **mandatory**

```
for(;;) {
    // infinite loop
}
```

Variable Scope

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

✔ Loop variable scope ends with loop

2.4 Enhanced `for` Loop (for-each)

```
for (int x : array) {
    System.out.println(x);
}
```

Characteristics

✔ Used for arrays & collections
❌ No index access
❌ Cannot modify collection structure

Modification Trap

```
for(int x : arr) {
    x = 10;   // does NOT modify array
}
```

3️⃣ Transfer Statements

Transfer control from one part of program to another.

3.1 `break`

Uses

- Terminates loop
- Terminates switch

```
break;
```

Labeled `break`

```
outer:
for(int i=0;i<3;i++) {
    for(int j=0;j<3;j++) {
        break outer;
    }
}
```

✔ Exits **outer loop**

3.2 `continue`

- Skips current iteration
- Moves to next iteration

```
continue;
```

Labeled `continue`

```
continue outer;
```

✔ Skips to next iteration of labeled loop

3.3 `return`

- Transfers control back to caller
- Ends method execution

```
return;
return value;
```

Important Rule

❌ Code after `return` is **unreachable**

```
return;
System.out.println("Hi"); // CE
```

4️⃣ Unreachable Code (Very High Exam Weight)

Java **does not allow unreachable statements**.

Common Causes

- `return`
- `break`
- `continue`
- `throw`
- Infinite loops with no exit

```
while(true) {
    System.out.println("Hi");
}
System.out.println("Bye"); // CE
```

5️⃣ Assertions (Often Overlooked)

```
assert condition;
assert condition : message;
```

✔ Disabled by default
✔ Enabled using `-ea` JVM option
✔ Used for debugging, not production logic

Certification-Critical Comparisons

Statement	Executes At Least Once
`while`	❌
`do-while`	✔

Loop Type	Index Access	Modification
for	✔	✔
for-each	❌	❌

Key Exam Takeaways

- Conditions must be **boolean**
- `else` binds to nearest `if`
- `switch` supports limited types
- Case labels must be constants
- Fall-through occurs without `break`
- Enhanced `for` cannot modify array elements

- Unreachable code causes **compile-time error**
- Labels work only with loops and switch

Modern Relevance

- Modern Java encourages:
    - Streams instead of loops
    - Pattern matching (future exams)
- Oracle exams still emphasize:
    - Classic `if`, `switch`, loops
    - Logical correctness over syntax memorization