Java Fundamentals Review

If Statements, Data Types, and Operations

Aadrit Talukdar, Nikola Mazzola, Arjun Maganti, Harish Senthilkumar

Basis Independent Silicon Valley

August 20, 2025

Agenda

1. If Statements
2. Data Types
3. Operations
4. Comprehensive Exercise

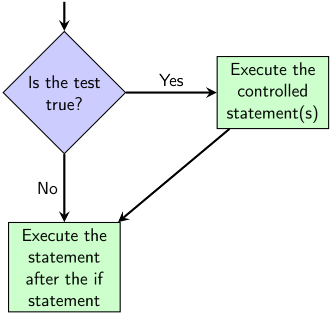
If Statements in Java

- If statements allow programs to make decisions
- They control the flow of execution based on conditions
- Essential for creating dynamic, responsive programs

Basic Syntax:

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

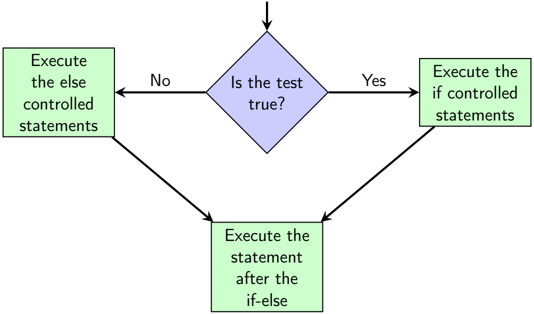
Note: Condition must evaluate to a boolean value (true or false).


Is the test true?
Yes
Execute the controlled statement(s)
No
Execute the statement after the if statement

If-Else Statements

```java
if (condition) {
    // code if true
} else {
    // code if false
}
```

If-else provides an alternative path when the condition is false.



Is the test true?

No

Execute the else controlled statements

Yes

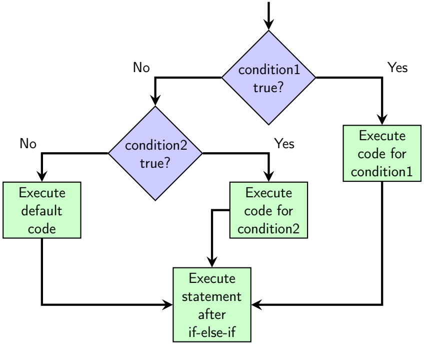
Execute the if controlled statements

Execute the statement after the if-else

Multiple Conditions with Else-If

```
if (condition1) {
    // code for condition1
} else if (condition2) {
    // code for condition2
} else {
    // default code
}
```

Else-if chains allow testing multiple conditions in sequence.


condition1 true?
No
Yes
condition2 true?
No
Yes
Execute code for condition1
Execute default code
Execute code for condition2
Execute statement after if-else-if

Exercise: Grade Calculator

Task: Write a program that takes a numerical grade and outputs the corresponding letter grade.

Grading Scale:

- 90-100: A
- 80-89: B
- 70-79: C
- 60-69: D
- Below 60: F

Solution: Grade Calculator

```java
if (grade <= 100 && grade >= 90) {
    System.out.println("A");
} else if (grade >= 80) {
    System.out.println("B");
} else if (grade >= 70) {
    System.out.println("C");
} else if (grade >= 60) {
    System.out.println("D");
} else if (grade < 60 && grade >= 0) {
    System.out.println("F");
} else {
    System.out.println("Invalid grade");
}
```

Java Data Types

Definition: Data Types

A name for a category of data values that are all related, as in type int in Java, which is used to represent integer values.

Why are Data Types Important?

- Every variable in Java must have a declared data type
- Java is a strongly typed language - data types are enforced at compile time
- Help prevent errors by ensuring operations are performed on compatible data

Primitive Data Types in Java

What are Primitive Data Types?

- Built-in data types provided by Java
- Store simple values directly in memory
- Eight primitive data types in Java

Type	Description	Examples
`byte`	8-bit integer (-128 to 127)	`byte age = 25;`
`short`	16-bit integer (-32,768 to 32,767)	`short year = 2024;`
`int`	32-bit integer (most common)	`int score = 95;`
`long`	64-bit integer (very large numbers)	`long population = 8000000000L;`
`float`	32-bit decimal number	`float price = 19.99f;`
`double`	64-bit decimal (more precise)	`double pi = 3.14159;`
`boolean`	True or false values	`boolean isActive = true;`
`char`	Single character (16-bit Unicode)	`char grade = 'A';`

Working with Numbers

Understanding Numeric Types and Their Uses:

- Choose the right type based on the range of values you need
- Consider memory usage for large datasets
- Be aware of precision differences between float and double

```java
int age = 18;                       // Most common for whole numbers
double price = 29.99;               // Default for decimal numbers
long population = 7800000000L;      // Need 'L' suffix for long literals
float temperature = 98.6f;          // Need 'f' suffix for float literals
```

Working with Numbers Cont.

Important Concepts:

- **Default Types:** Integer literals are `int`, decimal literals are `double`
- **Suffix Notation:** Use `L` for long, `f` for float to avoid type errors
- **Overflow:** When a value exceeds the maximum, it wraps around to the minimum
- **Precision:** `float` has 7 decimal digits, `double` has 15 decimal digits

Example of Overflow: `byte b = 127; b++; // b becomes -128`

Boolean and Character Data Types

Boolean Type:

- Only two values: `true` or `false`
- Used for logical conditions and flags
- Essential for if statements and loops

Character Type:

- Stores a single character (16-bit Unicode)
- Use single quotes for character literals
- Supports Unicode escape sequences
- Can represent any character from any language

Common Escape Sequences: ’\n’ (newline), ’\t’ (tab), ’\’’ (single quote), ’\\’ (backslash)

Examples:

```java
// Boolean examples
boolean isStudent = true;
boolean hasLicense = false;
boolean canVote = (age >= 18);

// Character examples
char grade = 'A';
char symbol = '$';
char newline = '\n';        // Escape sequence
char unicode = '\u0041';    // Unicode for 'A'
char digit = '5';           // Character, not number
```

Reference Types: Strings and Arrays

What are Reference Data Types?

- Store references (memory addresses) to objects, not the actual values
- More complex than primitive types - can hold multiple values or complex data
- Can be `null` (pointing to no object)
- Include classes, interfaces, arrays, and enums

Key Difference: Primitive variables store values directly; reference variables store memory addresses pointing to objects.

Reference Types: Strings and Arrays Cont.

Strings:

- Sequence of characters
- Immutable (cannot be changed once created)
- Use double quotes for string literals
- Rich set of built-in methods

Arrays:

- Collection of elements of the same type
- Fixed size once created
- Zero-indexed (first element at index 0)
- Can store primitives or objects

Examples:

```java
// String examples
String name = "John Doe";
String greet = "Hi, " + name;
String empty = "";             // Empty string
String nullStr = null;         // No object
// Array examples
int[] scores = {95, 87, 92, 78};     // Array literal
String[] subjects = new String[4];    // Array of size 4
subjects[0] = "Math"; // Assign value
```

String Operations and Methods

String Immutability:

- Strings cannot be modified after creation
- Operations create new String objects
- Original string remains unchanged
- Important for memory management

Common String Methods:

- `length()` - get string length
- `charAt(index)` - get char at pos
- `substring(start, end)` - get part
- `toUpperCase()/toLowerCase()` - change case
- `equals(other)` - compare strings

Important: Always use `.equals()` to compare strings, not == (which compares references).

Examples:

```java
String orig = "Java Program";
int len = orig.length(); // 16
String part = orig.substring(0, 4); // "Java"
String upper = orig.toUpperCase(); // "JAVA PROGRAM"
// String comparison
String name1 = "Alice";
String name2 = "Alice";
boolean same = name1.equals(name2); // true
// String concatenation
String full = "Hello" + " " + "World"; // "Hello World"
```

Java Operations

What are Operators?

- Symbols that perform operations on variables and values
- Essential for making decisions and calculations in programs
- Return results that can be used in conditions and assignments

Relational Operators:

- == (equal to)
- != (not equal to)
- < (less than)
- > (greater than)
- <= (less than or equal)
- >= (greater than or equal)

Return boolean values (true/false)

Logical Operators:

- && (logical AND)
- || (logical OR)
- ! (logical NOT)

Truth Table (AND):

A	B	A && B
T	T	T
T	F	F
F	T	F
F	F	F

Using Logical Operators

```java
boolean isStudent = true;
int age = 20;
boolean hasLicense = false;
// Combining conditions
boolean canVote = (age >= 18) && isStudent;         // true
boolean eligible = (age >= 21) || hasLicense;       // false
boolean invalid = !(age > 0 && age < 150);          // false
// Short-circuit examples
boolean result1 = false && (10/0 > 1);// false (divide by zero)
boolean result2 = true || (10/0 > 1); // true (divide by zero)
// Complex condition with parentheses
boolean qualified = (age >= 18) && (isStudent || hasLicense);
```

Key Points:

- `&&`: If first condition is false, second isn't evaluated
- `||`: If first condition is true, second isn't evaluated
- Use parentheses to clarify complex expressions

Student Grade Management System

Task: Create a program that calculates final grades and determines honors eligibility.

Given Variables:

- `int examScore` (0-100)
- `int homeworkScore` (0-100)
- `double attendanceRate` (0.0-1.0)
- `boolean extraCredit` (completed extra credit?)
- `String studentName`

Requirements:

- Calculate weighted final score: 70% exam + 30% homework
- Add 5 points if extra credit is completed
- Determine letter grade (A: 90+, B: 80-89, C: 70-79, D: 60-69, F: <60)
- Student qualifies for honors if: final grade >= 85 AND attendance >= 90%
- Display results with student name

Student Grade Management System Solution

```java
// Sample data
String studentName = "Alice Johnson";
int examScore = 87;
int homeworkScore = 92;
double attendanceRate = 0.95;
boolean extraCredit = true;
// Calculate weighted final score
double finalScore = (examScore * 0.7) + (homeworkScore * 0.3);
if (extraCredit) {
    finalScore += 5;  // Add extra credit bonus
}
```

Student Grade Management System Solution Cont.

```java
// Determine letter grade using if-else chain
char letterGrade;
if (finalScore >= 90) {
    letterGrade = ’A’;
} else if (finalScore >= 80) {
    letterGrade = ’B’;
} else if (finalScore >= 70) {
    letterGrade = ’C’;
} else if (finalScore >= 60) {
    letterGrade = ’D’;
} else {
    letterGrade = ’F’;
}
```

Student Grade Management System Solution Cont. Cont.

```java
// Check honors eligibility
boolean honorsEligible = (finalScore >= 85) && (attendanceRate >= 0.9);

// Display results
System.out.println("Student: " + studentName);
System.out.println("Final Score: " + finalScore + " (" + letterGrade + ")");
System.out.println("Honors Eligible: " + honorsEligible);
```

Conclusion

Thank you for listening!