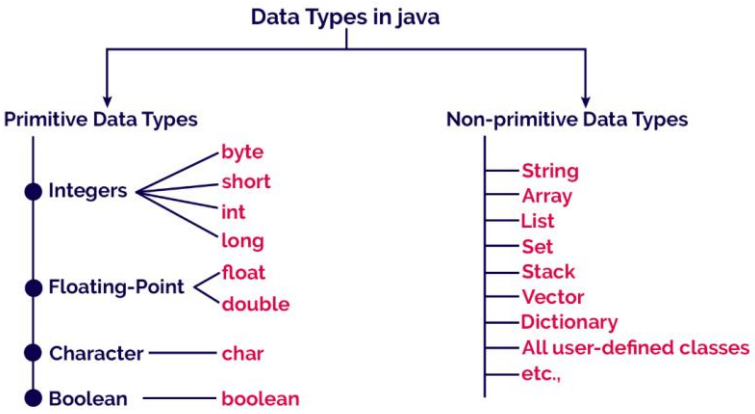
B.Sc. Computer Science

Unit I I

Data Types in Java (जावा में डेटा टाइप्स)



Java एक **Strongly Typed** language है, जिसका मतलब है कि हर variable का type पहले से बताना ज़रूरी है। Data types यह बताते हैं कि एक variable किस तरह की value store करेगा और वह memory में कितनी जगह लेगा।

Java में Data Types को दो मुख्य categories में बाँटा गया है:

1. Primitive Data Types

ये Java में पहले से predefined होते हैं। इनकी संख्या 8 होती है:

Category	Data Type	Size	Default Value	Example
Integer	byte, short, int, long	1, 2, 4, 8 bytes	0	int x = 100;
Floating Point	float, double	4, 8 bytes	0.0	double d = 9.99;
Character	char	2 bytes	'\u0000'	char ch = 'A';
Boolean	boolean	~1 bit	false	boolean isReady = true;

2. Non-Primitive Data Types

इन्हें **Reference Types** भी कहा जाता है क्योंकि ये objects को refer करते हैं।

- **Examples:** String, Array, Class, Interface.

Java Program Example

इस code में हम अलग-अलग data types का इस्तेमाल देखेंगे:

```
public class DataTypeExample {
    public static void main(String[] args) {
        // Integer types
```

```java
        byte age = 25;
        int salary = 50000;
        long population = 7000000000L; // Use 'L' for long

        // Floating point types
        float temperature = 36.6f; // Use 'f' for float
        double price = 199.99;

        // Character and Boolean
        char grade = 'A';
        boolean isJavaFun = true;

        // Non-primitive type
        String message = "Learning Java Data Types";

        // Displaying the values
        System.out.println("Age: " + age);
        System.out.println("Salary: " + salary);
        System.out.println("Temperature: " + temperature);
        System.out.println("Is Java Fun? " + isJavaFun);
        System.out.println("Message: " + message);
    }
}
```

Output

```
Age: 25
Salary: 50000
Temperature: 36.6
Is Java Fun? true
Message: Learning Java Data Types
```

Key Points to Remember (महत्वपूर्ण बातें)

1. **Strictness:** आप int variable में "Hello" store नहीं कर सकते, compiler error देगा।
2. **Size:** Java में char 2 bytes लेता है क्योंकि यह **Unicode** support करता है (C++ में यह 1 byte लेता है)।
3. **Default Values:** अगर आप class-level variable (Instance variable) को value नहीं देते, तो Java उसे default value दे देता है (जैसे int के लिए 0), लेकिन local variables को initialize करना ज़रूरी है।
4. **Suffixes:** long value के पीछे L और float के पीछे f लगाना अच्छी practice मानी जाती है।

Operators in Java (जावा में ऑपरेटर्स)

Operators वे symbols होते हैं जिनका इस्तेमाल variables और values पर operations (जैसे जोड़ना, घटाना, या तुलना करना) करने के लिए किया जाता है।
जैसे 10 + 20 में + एक **Operator** है और 10 और 20 **Operands** हैं।

Types of Operators (ऑपरेटर्स के प्रकार)

Java में मुख्य रूप से ये ऑपरेटर्स इस्तेमाल होते हैं:

Category	Operators	Description
Arithmetic	+, -, *, /, %	Basic calculations के लिए।
Relational	==, !=, >, <, >=, <=	Comparison (तुलना) करने के लिए।
Logical	&&, `	
Assignment	=, +=, -=, *=, /=	Value assign करने के लिए।
Unary	++, --, +, -, !	Single operand पर काम करने के लिए।

---

1. Arithmetic Operators

इनका उपयोग गणितीय गणना (mathematical calculations) के लिए होता है।

% (Modulus) operator शेषफल (remainder) देता है।

```
public class ArithmeticDemo {
    public static void main(String[] args) {
        int a = 10;
        int b = 3;

        // Basic arithmetic operations
        System.out.println("Addition: " + (a + b));      // 13
        System.out.println("Subtraction: " + (a - b));   // 7
        System.out.println("Multiplication: " + (a * b));// 30
        System.out.println("Division: " + (a / b));      // 3 (Integer division)
        System.out.println("Remainder: " + (a % b));     // 1
    }
}
```

---

2. Relational (Comparison) Operators

ये दो values की तुलना करते हैं और result हमेशा **boolean** (true या false) में देते हैं।

```
public class RelationalDemo {
    public static void main(String[] args) {
        int x = 10, y = 20;

        // Comparing values
        System.out.println(x == y); // false (is x equal to y?)
        System.out.println(x != y); // true  (is x not equal to y?)
        System.out.println(x > y);  // false (is x greater than y?)
        System.out.println(x < y);  // true  (is x less than y?)
    }
}
```

---

3. Logical Operators

इनका उपयोग logic check करने के लिए किया जाता है, खासकर if-else statements में।

- **&& (AND):** दोनों condition सही होनी चाहिए।
- **|| (OR):** कोई एक condition सही होनी चाहिए।
- **! (NOT):** Result को उल्टा कर देता है (True को False)।

---

4. Unary Operators (Increment/Decrement)

ये बहुत important हैं और single value पर काम करते हैं।

- **++ (Increment):** Value को 1 से बढ़ाता है।
- **-- (Decrement):** Value को 1 से घटाता है।

Java

```
public class UnaryDemo {
```

```
    public static void main(String[] args) {
        int count = 10;

        // Post-increment: use value then increment
        System.out.println(count++); // Prints 10, then count becomes 11

        // Pre-increment: increment then use value
        System.out.println(++count); // count becomes 12, then prints 12
    }
}
```

Key Points to Remember (महत्वपूर्ण बातें)

1. **Precedence:** जब एक ही line में कई operators हों, तो Java "Operator Precedence" (जैसे BODMAS) follow करता है। * और / की प्राथमिकता + और - से ज़्यादा होती है।
2. **Assignment vs Equality:** = का मतलब value डालना होता है, जबकि == का मतलब check करना होता है कि दोनों बराबर हैं या नहीं।

Variable Scope in Java (वेरिएबल का दायरा)

Variable Scope का मतलब है कि एक variable प्रोग्राम के किस हिस्से में access (इस्तेमाल) किया जा सकता है। Java में, variable की "Life" और "Visibility" इस बात पर निर्भर करती है कि उसे कहाँ declare किया गया है।
मुख्य रूप से Java में 3 तरह के scopes होते हैं:

Scope Type	Location	Life & Visibility
Instance Variable	Class के अंदर (Methods के बाहर)	जब तक Object memory में है, तब तक रहेगा।
Static Variable	Class के अंदर (static keyword के साथ)	जब तक Program चल रहा है, तब तक रहेगा।
Local Variable	Method या Block { } के अंदर	सिर्फ उस method/block के खत्म होने तक रहता है।

Code Example: Variable Scopes

Java

```
public class ScopeDemo {
    // Instance Variable: Available to all methods in this class
    int instanceVar = 10;

    // Static Variable: Shared among all objects of this class
    static int staticVar = 20;

    public void display() {
        // Local Variable: Can only be used inside this method
        int localVar = 30;

        System.out.println("Instance: " + instanceVar);
        System.out.println("Static: " + staticVar);
        System.out.println("Local: " + localVar);
    }

    public void anotherMethod() {
        // System.out.println(localVar); // ERROR: localVar is not visible here
    }
```

```
}
```

Type Conversion in Java (टाइप कन्वर्जन)

जब हम एक data type की value को दूसरे data type में बदलते हैं, तो उसे **Type Conversion** कहते हैं। Java में यह दो तरह से होता है:

1. Widening (Implicit) Conversion - "Automatic"

जब हम छोटे data type को बड़े data type में डालते हैं, तो Java इसे अपने आप कर देता है क्योंकि इसमें data loss का कोई डर नहीं होता।

- **Order:** byte -> short -> int -> long -> float -> double

2. Narrowing (Explicit) Conversion - "Manual"

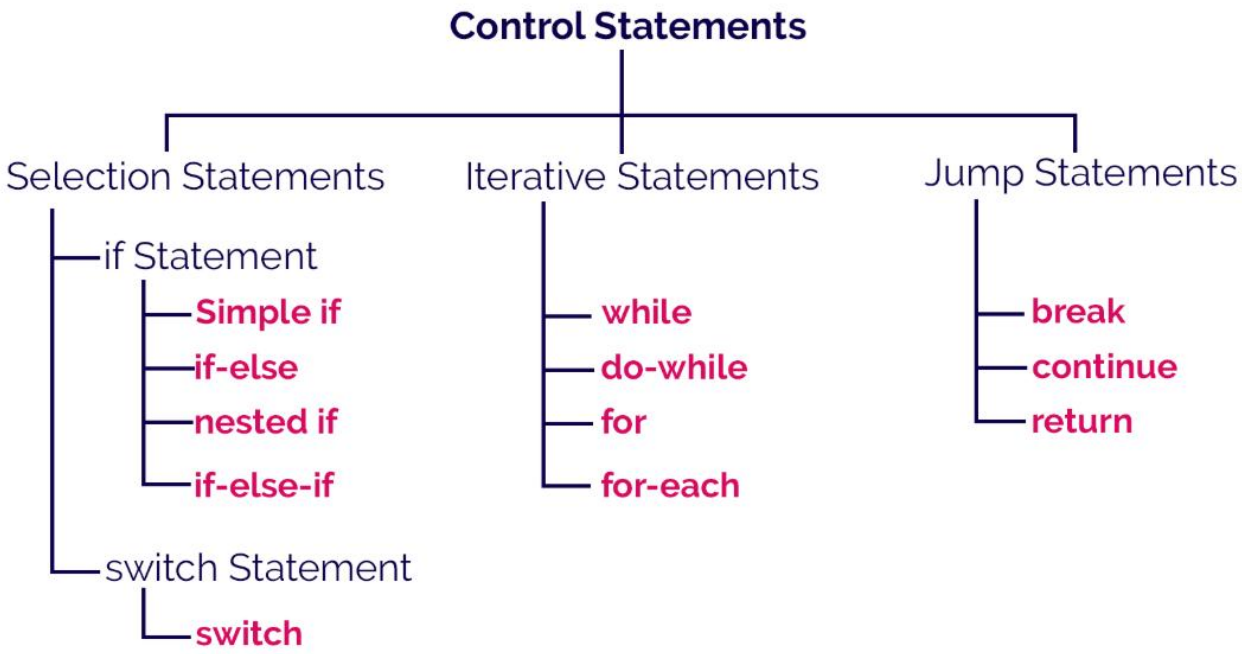
जब हम बड़े data type को छोटे data type में डालते हैं, तो हमें manually बताना पड़ता है (Casting)। इसमें data loss हो सकता है (जैसे 9.99 को int में बदलने पर सिर्फ 9 बचेगा)।

Code Example: Type Casting

```
public class CastingExample {
    public static void main(String[] args) {
        // 1. Widening (Automatic)
        int myInt = 100;
        double myDouble = myInt; // int automatically converted to double

        System.out.println("Integer Value: " + myInt);    // 100
        System.out.println("Double Value: " + myDouble);  // 100.0

        // 2. Narrowing (Manual)
        double pi = 3.14;
        int roundedPi = (int) pi; // Manually casting double to int

        System.out.println("Original Pi: " + pi);         // 3.14
        System.out.println("Converted Pi: " + roundedPi); // 3 (Data lost!)
    }
}
```

Key Points to Remember (महत्वपूर्ण बातें)

1. **Scope Boundary:** Local variables को इस्तेमाल करने से पहले initialize करना ज़रूरी है, जबकि Instance variables को Java default value दे देता है।
2. **Casting Syntax:** Narrowing के समय target type को brackets ( ) में लिखना अनिवार्य है।
3. **Boolean Casting:** Java में boolean को किसी और type (जैसे int) में cast नहीं किया जा सकता।



Selection/Decision Making Statements in Java

Decision Making Statements का इस्तेमाल तब किया जाता है जब हमें प्रोग्राम में किसी खास condition के आधार पर यह तय करना हो कि कौन सा code block execute होगा और कौन सा नहीं।

इसे **Selection Statements** या **Control Flow Statements** भी कहा जाता है।

Types of Decision Making Statements

Java में मुख्य रूप से चार प्रकार के decision-making statements होते हैं:

1. **if statement**
2. **if-else statement**
3. **if-else-if ladder**
4. **switch statement**

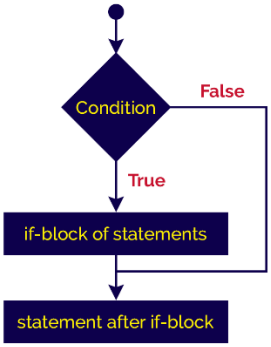
1. if Statement

यह सबसे बेसिक स्टेटमेंट है। अगर ब्रैकेट के अंदर की condition **true** होती है, तो उसके नीचे वाला code block चलता है।

Syntax

```
if(condition){
        if-block of statements;
        ...
}
statement after if-block;
...
```

Flow of execution




```java
public class SimpleIf {
    public static void main(String[] args) {
        int age = 20;

        // If condition is true, code inside braces will run
        if (age >= 18) {
            System.out.println("You are eligible to vote.");
        }
    }
}
```

Output

```
You are eligible to vote.
```

---

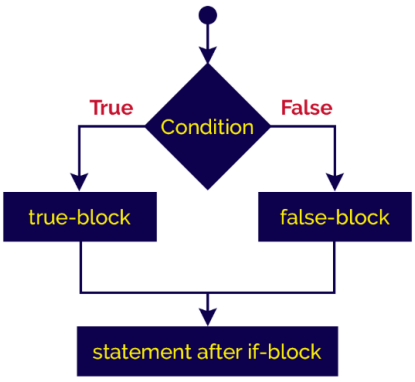
2. if-else Statement

यहाँ हमारे पास दो रास्ते होते हैं। अगर condition **true** है तो if वाला हिस्सा चलेगा, वरना else वाला हिस्सा चलेगा।

Syntax

```
if(condition){
        true-block of statements;
        ...
}
else{
        false-block of statements;
        ...
}
statement after if-block;
...
```

Flow of execution




```java
public class IfElseExample {
    public static void main(String[] args) {
        int number = 13;
```

```
        // Checking if the number is even or odd
        if (number % 2 == 0) {
            System.out.println("This is an Even number.");
        } else {
            System.out.println("This is an Odd number.");
        }
    }
}
```

```
Output
This is an Odd number.
```

---

3. if-else-if Ladder

जब हमारे पास बहुत सारी conditions हों और हमें उनमें से किसी एक को चुनना हो, तब हम ladder का उपयोग करते हैं।

```
public class GradeSystem {
    public static void main(String[] args) {
        int marks = 85;

        // Multiple conditions check
        if (marks >= 90) {
            System.out.println("Grade: A+");
        } else if (marks >= 80) {
            System.out.println("Grade: A");
        } else if (marks >= 70) {
            System.out.println("Grade: B");
        } else {
            System.out.println("Grade: C/Fail");
        }
    }
}
```

```
Output
Grade: A
```

---

4. switch Statement

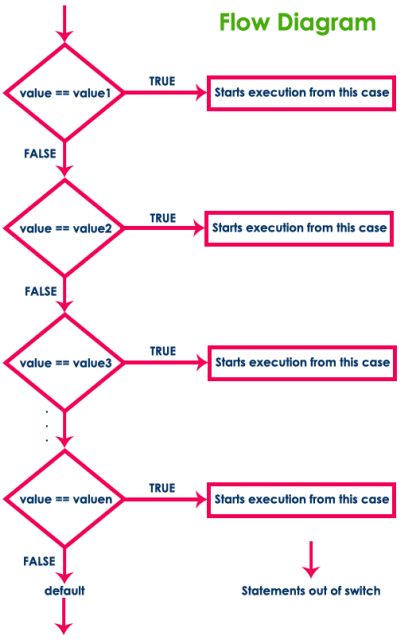
यह if-else-if ladder का एक बेहतर विकल्प है जब हमें किसी एक variable की अलग-अलग **constant values** को चेक करना हो।

- **case**: अलग-अलग values को represent करता है।
- **break**: यह current case से बाहर निकलने के लिए ज़रूरी है, वरना उसके नीचे के सारे cases भी चल जाएंगे (Fall-through)।

- **default**: अगर कोई भी case मैच नहीं होता, तो यह चलता है।

Syntax

```
switch ( expression or value )
{
      case  value1:  set of statements;
                     ....
      case  value2:  set of statements;
                     ....
      case  value3:  set of statements;
                     ....
      case  value4:  set of statements;
                     ....
      case  value5:  set of statements;
                     ....
      .
      .
      default: set of statements;
}
```




```java
public class DayCheck {
    public static void main(String[] args) {
        int day = 3;

        // Selecting a case based on variable value
        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");
        }
    }
}
```

Output

```
Wednesday
```

Key Points to Remember (महत्वपूर्ण बातें)

1. **Boolean Result**: if के अंदर की condition हमेशा boolean (true/false) होनी चाहिए।
2. **Braces { }**: अगर if या else के बाद सिर्फ एक ही line का code है, तो { } लगाना optional है, लेकिन अच्छी practice के लिए लगाना चाहिए।

3. **Switch limitations**: Switch के अंदर हम range (जैसे marks > 80) चेक नहीं कर सकते, हमें exact value देनी होती है।
4. **Floating point in Switch**: Java में switch के अंदर float या double इस्तेमाल नहीं किया जा सकता।

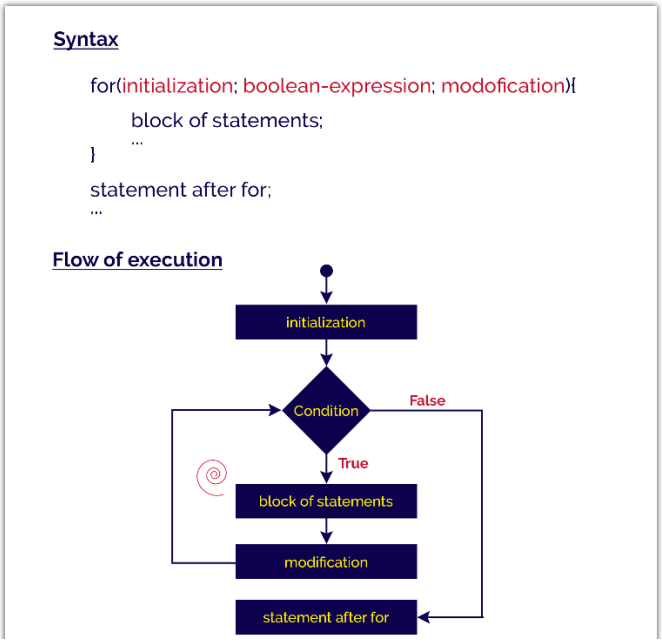
Iterative Statements

(जिन्हें हम **Loops** भी कहते हैं) का इस्तेमाल तब किया जाता है जब हमें एक ही code block को बार-बार execute करना हो। Java में मुख्य रूप से तीन तरह के loops होते हैं।

यहाँ उनका विस्तार से विवरण दिया गया है:

1. The for Loop

यह तब सबसे अच्छा होता है जब आपको पहले से पता हो कि loop कितनी बार चलाना है (Fixed number of iterations)। इसमें initialization, condition, और increment/decrement एक ही line में होते हैं।




```java
public class ForExample {
    public static void main(String[] args) {
        // Loop runs from 1 to 5
        for (int i = 1; i <= 5; i++) {
            System.out.println("Iteration: " + i);
        }
    }
}
```

Output

```
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
```

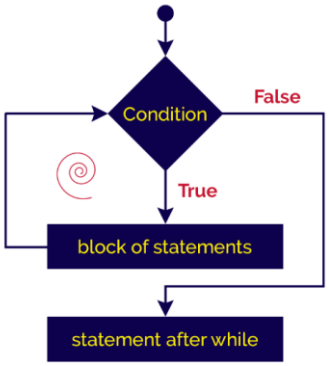
2. The while Loop

इसे **Entry-Controlled Loop** कहा जाता है क्योंकि यह code execute करने से पहले condition चेक करता है। अगर condition शुरू में ही false हो, तो loop एक बार भी नहीं चलेगा।

Syntax

```
while(boolean-expression){
      block of statements;
      ...
}
statement after while;
...
```

Flow of execution




```java
public class WhileExample {
    public static void main(String[] args) {
        int count = 1;

        // Checks condition before entering the loop
        while (count <= 5) {
            System.out.println("Count is: " + count);
            count++; // Incrementing the counter
        }
    }
}
```

Output

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
```

---

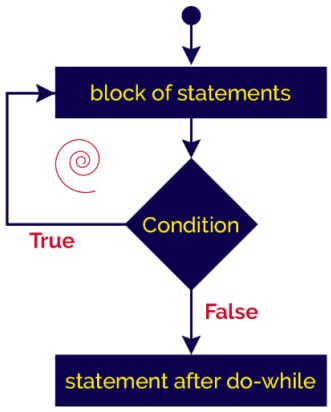
3. The do-while Loop

इसे **Exit-Controlled Loop** कहा जाता है। इसमें condition अंत में चेक होती है, जिसका मतलब है कि loop का code कम से कम **एक बार (once) ज़रूर चलेगा**, चाहे condition पहली बार में ही false क्यों न हो।

Syntax

```
do{
    block of statements;
    ...
}while(boolean-expression);
statement after do-while;
...
```

Flow of execution




```java
public class DoWhileExample {
    public static void main(String[] args) {
        int i = 10;

        do {
            // This prints at least once even if condition is false
            System.out.println("Value of i: " + i);
            i++;
        } while (i < 5);
    }
}
```

```
Output

Value of i: 10
```

Comparison Table: Which loop to use?

Feature	for Loop	while Loop	do-while Loop
Best Used	When you know the exact number of steps.	When the number of steps depends on a condition.	When the code must run at least once (e.g., Menu).
Condition Check	At the beginning (Entry).	At the beginning (Entry).	At the end (Exit).
Minimum Runs	0	0	**1**

Jump Statements inside Loops

कभी-कभी हमें loop को बीच में ही रोकना पड़ता है या किसी step को skip करना पड़ता है:

1. **break**: यह loop को तुरंत पूरी तरह से बंद कर देता है।
2. **continue**: यह current step को छोड़ देता है और सीधे अगली iteration (अगले चक्कर) पर चला जाता है।

```
public class ControlStatements {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            if (i == 3) {
                continue; // Skips printing 3
            }
            if (i == 5) {
                break; // Stops the loop entirely
            }
            System.out.println("i = " + i);
        }
    }
}
```

Output

```
i = 1
i = 2
i = 4
```