Java Fundamentals: Tokens, Identifiers, and More

1. Overview

Java provides a robust set of fundamentals to prepare and compile applications. Some key components include:

- **Tokens**
- **Data Types**
- **Type Casting**
- **Java Statements**
- **Arrays**

This document covers tokens in detail, the rules for identifiers, and best practices when naming elements in Java applications.

---

2. Tokens and Lexemes

2.1 Definitions

- **Lexeme:**
  The individual logical unit in Java programming.
- **Token:**
  A collection of lexemes under a particular group.

2.2 Example

Consider the statement:

```
java
CopyEdit
int a = b + c * d ;
```

- **Lexemes:**
  `int`, `a`, `=`, `b`, `+`, `c`, `*`, `d`, `;` → **9 lexemes**
- **Token Breakdown:**
  - `int` → Data Type
  - `a, b, c, d` → Variables
  - `=, +, *` → Operators
  - `;` → Special Symbol

2.3 Types of Tokens

Java tokens are grouped into four main types:

1. **Identifiers**
2. **Literals**
3. **Keywords / Reserved words**
4. **Operators**

---

3. Identifiers

An **identifier** is a name given to programming elements such as variables, methods, classes, interfaces, etc.

3.1 Rules for Java Identifiers

1. **Starting Character:**
   - Must not start with a number.
   - Valid starting characters include alphabets, underscore _, and dollar sign `$`.
   - **Examples:**
     - `int 9eno = 999;` → **Invalid**
     - `String eid = "E-1111";` → **Valid**
     - `float _esal = 50000.0f;` → **Valid**
     - `float $esal = 25000.0f;` → **Valid**
     - `String emp9No = "E-999";` → **Valid**
     - `String emp_Addr = "Hyd";` → **Valid**
     - `double salIn$s = 10000.0;` → **Valid**
2. **No Spaces Allowed:**
   - Identifiers cannot have spaces.
   - **Examples:**
     - `String emp Addr = "Hyd";` → **Invalid**
     - `String empAddr = "Hyd";` → **Valid**
     - `float annualIncome = 500000000.0f;` → **Valid**
     - `float annual Income = 2500000.0f;` → **Invalid**
3. **No Operators Allowed:**
   - Identifiers must not contain operators.
   - **Examples:**
     - `int empNo = 111;` → **Valid**
     - `int emp+No = 111;` → **Invalid**
     - `String emp-Name = "Durga";` → **Invalid**
     - `float emp*Sal = 25000.0f;` → **Invalid**
4. **No Special Symbols (except _ and $):**
   - Special symbols (other than _ and `$`) are not allowed.
   - **Examples:**
     - `int emp.No = 555;` → **Invalid**
     - `String #eaddr = "Hyd";` → **Invalid**
     - `String emp@Hyd = "Durga";` → **Invalid**
     - `float emp-Sal = 250000.0f;` → **Invalid**
     - `double annual_Income = 50000000D;` → **Valid**
5. **Keywords Are Not Allowed:**
   - Java identifiers cannot be keywords.

- **Examples:**
  - `int eno = 10;` → **Valid**
  - `int for = 20;` → **Invalid**
  - `String str = "abc";` → **Valid**
  - `String break = "xyz";` → **Invalid**
  - `float while = 250000.0f;` → **Invalid**

6. **Primitive Data Types Are Not Allowed as Identifiers:**
   - Identifiers cannot be the names of primitive data types.
   - **Examples:**
     - `int eno = 10;` → **Valid**
     - `int boolean = 20;` → **Invalid**
     - `float int = 33.33f;` → **Invalid**
     - `double byte = 345.456D;` → **Invalid**
     - `String long = "abc";` → **Invalid**
7. **Using Predefined Class Names, Abstract Classes, and Interfaces:**
   - It is possible to use names of predefined classes, abstract classes, and interfaces as identifiers. However, once declared, they lose their original reference within that scope.
   - **Examples:**

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int Exception = 10;
       System.out.println(Exception);
   }
}
```

Output: `10`

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        String String  = "String";
        System.out.println(String);
    }
}
```

Output: `String`

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int System = 10;
        System.out.println(System); // 10
    }
}
```

   - **Note:** Once a predefined class name (e.g., `System`) is used as a variable, it can only be used as that variable in the remaining part of the code. To refer back to the original `System` class, its fully qualified name must be used:

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int System = 10;
        java.lang.System.out.println(System); // 10
        System = System + 10; // System = 10 + 10 = 20
        java.lang.System.out.println(System); // 20
        System = System + 10; // System = 20 + 10 = 30
        java.lang.System.out.println(System); // 30
    }
}
```

8. **No Duplicate Identifiers in the Same Scope:**
    - Identifiers must not be duplicated within the same scope.
    - **Scopes in Java:**
        - **Local Scope:** Variables declared inside a method are local and are not accessible outside that method.
        - **Class Level Scope:** Variables declared outside all methods but within a class. These are accessible across all methods within the class.

Example:

```
java
CopyEdit
class A {
    int x = 100; // Class Level Variable

    void m1() {
        int i = 10; // Local variable
        System.out.println(i);
        System.out.println(j); // Error: j is not declared
        System.out.println(x);
    }

    void m2() {
        System.out.println(i); // Error: i is not accessible here
        int j = 20; // Local variable
        System.out.println(j);
        System.out.println(x);
    }
}
```

9. **Duplicate Declaration Errors:**

```
java
CopyEdit
class A {
    int i = 10;
    long i = 20; // Compilation Error: Duplicate variable in the same
scope
    double b = 234.345d; // No Compilation Error

    void m1() {
        float i = 22.22f; // No Compilation Error: Local scope
        byte b = 10;
        short b = 20; // Compilation Error: Duplicate variable in the
same local scope
```

```
        }
    }
```

---

4. Best Practices for Identifiers

4.1 Naming Suggestions

- **Meaningful Names:**
  - Use identifiers that clearly reflect their purpose.
  - **Example:**
    - `String xxx = "abc123";` → **Not Suggestible**
    - `String accNo = "abc123";` → **Suggestible**
- **Length Considerations:**
  - There are no strict length restrictions, but it is recommended to keep identifiers to around 10 characters where possible.
  - **Example:**
    - `String temporaryemployeeaddress = "Hyd";` → **Not Suggestible**
    - `String tempEmpAddr = "Hyd";` → **Suggestible**
- **Handling Multiple Words:**
  - When an identifier has more than one word, use separators for clarity.
  - **Examples:**
    - `String tempEmpAddr = "Hyd";` → **Not Suggestible**
    - `String temp_Emp_Addr = "Hyd";` → **Suggestible**

4.2 Summary Table: Identifier Rules

Rule	Valid Example	Invalid Example
Starts with a letter, _, or $	`float _esal = 50000.0f;`	`int 9eno = 999;`
No spaces allowed	`String empAddr = "Hyd";`	`String emp Addr = "Hyd";`
No operators in identifier	`int empNo = 111;`	`int emp+No = 111;`
No special symbols (except _ and $)	`double annual_Income = 50000000D;`	`int emp.No = 555;`
No Java keywords	`int eno = 10;`	`int for = 20;`
No primitive data types as identifiers	`int eno = 10;`	`int boolean = 20;`

---

5. Important Note on Variables

A **variable** is a memory block where data is stored. The name given to this memory block is an identifier. Remember:

- Use meaningful names for clarity.
- Follow the naming conventions and rules discussed above to avoid compilation errors.

---

6. Conclusion

These notes cover the fundamentals of tokens and identifiers in Java. Understanding these basics is essential for preparing and writing Java applications. Always remember:

- Follow the rules for identifiers.
- Use meaningful, clear, and concise names.
- Be aware of scope rules to avoid duplicate declarations within the same scope.

Feel free to copy and paste these notes into your DOC file for ready reference when preparing Java applications.

Java Literals

Overview

- **Literal**: A literal is a constant value assigned to a variable.
- Every literal represents a fixed value that cannot change during the program execution.
- In a typical statement like:

```
java
CopyEdit
int a = 10;
```

  - `int` → Data Type
  - `a` → Identifier (variable)
  - `=` → Operator
  - `10` → **Literal (constant)**
  - `;` → Special Symbol, Terminator

Types of Literals

Java provides several kinds of literals. Below are the main types with code examples.

1. Integral / Integer Literals

- **Data Types**: `byte`, `short`, `int`, `long`
- **Examples**: `10, 20, 30, ...`
- **Character Literals**: Represented using single quotes:
  - Examples: `'a'`, `'b'`, `'\n'`, `'\t'`

Example:

```
java
CopyEdit
```

```
public class Test {
    public static void main(String[] args) {
        System.out.println("DurgaSoftwareSolutions");
        System.out.println("Durga\tSoftware\tSolutions");
        System.out.println("Durga\nSoftware\nSolutions");
    }
}
```

Output:

```
nginx
CopyEdit
DurgaSoftwareSolutions
Durga   Software        Solutions
Durga
Software
Solutions
```

Note:

- For `byte`, `short`, and `int` no suffix is required.
- For `long` values, use `l` or `L` as a suffix.

Example:

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        long mobileNo = 9988776655L;
        System.out.println(mobileNo);
        long balance = 123457778998761;
        System.out.println(balance);
    }
}
```

Output:

```
CopyEdit
9988776655
12345777899876
```

---

2. Floating Point Literals

- **Data Types**: `float`, `double`
- **Examples**:
    - Float: `22.22f`, `345.347F`, ...
    - Double: `4567.2345`, `6789.456d`, `12345.3456D`

Note:

- For `float`, use `f` or `F` as a suffix.
- For `double`, you can use no suffix or `d`/`D`.

Example (float):

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        float smarksAggregate = 7.8f;
        System.out.println("Student Aggregate Marks   : " +
smarksAggregate);
        float temp = 28.5F;
        System.out.println("Temp   : " + temp);
    }
}
```

Output:

```
yaml
CopyEdit
Student Aggregate Marks   : 7.8
Temp   : 28.5
```

Example (double):

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        double annualIncome = 12345678.345678;
        System.out.println(annualIncome);

        double balance = 876543432.5689d;
        System.out.println(balance);

        double latitude = 123.3456788D;
        System.out.println(latitude);
    }
}
```

Output:

```
CopyEdit
1.2345678345678E7
8.765434325689E8
123.3456788
```

Compilation Tip:
The following code causes an error because the literal is not marked as a float:

```
java
CopyEdit
float f = 12345.3456; // Compilation error
```

---

3. Boolean Literals

- **Data Type**: `boolean`

- **Literals**: `true`, `false`

Example:

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        boolean b1 = true;
        boolean b2 = false;

        System.out.println("Today is SUNDAY : " + b1);
        System.out.println("This is JAVA Class  : " + b2);
    }
}
```

Output:

```
vbnet
CopyEdit
Today is SUNDAY : true
This is JAVA Class  : false
```

---

4. String Literals

- **Data Type**: `String`
- **Representation**: Data inside double quotations `"..."`.
- **Note**:
    - Strings in Java can represent characters, digits, and special symbols.
    - Single quotes can appear directly; for double quotes inside a string, escape them using a backslash `\`.

Example:

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        String userName = "Durga";
        String userpassword = "durga123";
        String userEmail = "durga123@durgasoft.com";
        String userMobile = "91-9988776655";

        System.out.println("User Details");
        System.out.println("----------------------");
        System.out.println("User Name         : " + userName);
        System.out.println("User password     : " + userpassword);
        System.out.println("User Email Id     : " + userEmail);
        System.out.println("User Mobile No    : " + userMobile);
    }
}
```

Output:

```
pgsql
```

```
CopyEdit
User Details
----------------------
User Name          : Durga
User password      : durga123
User Email Id      : durga123@durgasoft.com
User Mobile No     : 91-9988776655
```

Additional Examples for Quotations:

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        String str1 = "Durga Software Solutions";
        System.out.println(str1);
        String str2 = "Durga 'Software' Solutions";
        System.out.println(str2);
        String str3 = "Durga \"Software\" Solutions";
        System.out.println(str3);
    }
}
```

Output:

```
nginx
CopyEdit
Durga Software Solutions
Durga 'Software' Solutions
Durga "Software" Solutions
```

Or combining both:

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        String str = "Durga 'Software' \"Solutions\" ";
        System.out.println(str);
    }
}
```

Output:

```
nginx
CopyEdit
Durga 'Software' "Solutions"
```

---

5. Readability in Numeric Literals (Java 7+)

- **Feature:**
  Java 7 allows using underscore (_) characters to improve readability in numeric literals.
- **Example:**

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        double d = 12_34_56_789.34567;
        System.out.println(d);
    }
}
```

Important:

- In Java 6, the underscores in numeric literals are not supported and cause compilation errors.

Compilation in Java 6:

```
cpp
CopyEdit
Test.java:3: ';' expected
        double d = 12_34_56_789.34567;
                     ^
Test.java:3: ';' expected
        double d = 12_34_56_789.34567;
                           ^
```

Compilation in Java 7:

- The same code compiles without error:

```
CopyEdit
1.2345678934567E8
```

Summary Table of Literal Types

Literal Type	Examples	Suffix Requirements	Notes
Integer Literals	`10`, `20`, `30`, `'a'` (for char)	`long` requires `l` or `L`	No suffix needed for `byte`, `short`, `int`
Floating-Point	`22.22f`, `345.347F`, `4567.2345`, `6789.456d`	`float` requires `f` or `F`; `double` no suffix or `d`/`D`	Float must have a suffix
Boolean Literals	`true`, `false`	N/A	Only two boolean literals exist
String Literals	`"abc"`, `"xyz"`, `"Durga 'Software' \"Solutions\""`	Data enclosed in double quotes	Supports all types of characters, digits, and symbols
Numeric Readability	`12_34_56_789.34567`	Supported in Java 7 and later	Improves readability; not allowed in earlier versions

---

Final Notes

- **Literals** are essential constants in Java used to initialize variables.
- Always use appropriate suffixes for `long` and `float` to avoid compilation errors.
- Use escape sequences (`\`) in String literals to include special characters like double quotes.
- Remember the Java version differences when using underscores in numeric literals.

Number Systems in Java

Overview

In all programming languages, numbers must follow a standard representation known as **Number Systems**. In Java, there are four types of number systems:

1. **Binary Number System**
2. **Octal Number System**
3. **Decimal Number System**
4. **Hexa-decimal Number System**

Note: Although all number systems are allowed in Java, the **default** number system is the **Decimal Number System**.

---

1. Binary Number System

- **Base Value:** 2
- **Allowed Digits:** 0, 1
- **Prefix:** `0b` or `0B`
- **Introduced in:** Java 1.7 onwards

Example:

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int a = 10;
        System.out.println(a);  // 10 (decimal)

        int b = 0b10;          // Binary: 0b10 ==> (0*2^0 + 1*2^1) = 2
        System.out.println(b);

        int c = 0b1010;        // Binary: 0b1010 ==> (0*2^0 + 1*2^1 + 0*2^2
+ 1*2^3) = 10
        System.out.println(c);

        int d = 0B1111;        // Binary: 0B1111 ==> (1+2+4+8) = 15
        System.out.println(d);
```

```
    }
}
```

Output:

```
CopyEdit
10
2
10
15
```

Additional Examples & Notes:

- **Valid:**

  ```
  java
  CopyEdit
  int b = 0b1010;
  int c = 0B1111;
  ```

- **Invalid:**

  ```
  java
  CopyEdit
  int d = 0b1020; // '2' is not allowed in binary numbers
  ```

- **Java 1.6 Compatibility:**
  Binary literals are not supported in Java 1.6.
  Example with Java 1.6:

  ```
  java
  CopyEdit
  public class Test {
      public static void main(String[] args) {
          int a = 0b1010;
          System.out.println(a);
      }
  }
  ```

  Compilation in Java 1.6 produces errors.

---

2. Octal Number System

- **Base Value:** 8
- **Allowed Digits:** 0, 1, 2, 3, 4, 5, 6, 7
- **Prefix:** A leading zero (`0`)

Example:

```
java
CopyEdit
public class Test {
```

```
    public static void main(String[] args) {
        int a = 10;
        System.out.println(a); // 10 (decimal)

        int b = 010; // Octal: 010 ==> (0*8^0 + 1*8^1) = 8
        System.out.println(b); // 8

        int c = 0123; // Octal: 0123 ==> (3*8^0 + 2*8^1 + 1*8^2) = 3 + 16 +
64 = 83
        System.out.println(c); // 83
    }
}
```

Output:

```
CopyEdit
10
8
83
```

Additional Examples & Notes:

- **Valid:**

  ```
  java
  CopyEdit
  int b = 012345;
  ```

- **Invalid:**

  ```
  java
  CopyEdit
  int c = O567345; // 'O' is not a valid digit; must be zero
  int d = 045678;  // '8' is not allowed in octal digits
  ```

- **Decimal numbers (e.g., `23`) are not prefixed and are interpreted as decimal.**

---

3. Decimal Number System

- **Base Value:** 10
- **Allowed Digits:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- **Prefix:** *No prefix required*

---

4. Hexa-decimal Number System

- **Base Value:** 16
- **Allowed Digits:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f (or A–F)
- **Prefix:** `0x` or `0X`

Example:

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int a = 10;
        System.out.println(a); // 10 (decimal)

        int b = 0x10; // Hex: 0x10 ==> (0*16^0 + 1*16^1) = 16
        System.out.println(b); // 16

        int c = 0X123; // Hex: 0X123 ==> (3*16^0 + 2*16^1 + 1*16^2) = 3 +
32 + 256 = 291
        System.out.println(c); // 291
    }
}
```

Output:

```
CopyEdit
10
16
291
```

Additional Examples & Notes:

- **Valid Hexa-decimal Examples:**

  ```
  java
  CopyEdit
  int b = 0x123456;
  int c = 0X56789abc;
  int d = 0x10def23abc;
  ```

- **Invalid Example:**

  ```
  java
  CopyEdit
  int d = 0X7854defg; // 'g' is not allowed in hexadecimal numbers
  ```

- **Decimal numbers (e.g., `10`) are not considered hexadecimal.**

Summary of Number Systems in Java

Number System	Base	Allowed Digits	Prefix
Binary	2	0, 1	`0b` or `0B`
Octal	8	0, 1, 2, 3, 4, 5, 6, 7	`0` (zero)
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	*No prefix*
Hexa-decimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a-f	`0x` or `0X`

Additional Example: Arithmetic with Mixed Number Systems

In Java, even if numbers are specified in different number systems, the compiler converts them into decimal values. All arithmetic operations are performed on the decimal equivalents.

Example:

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int a = 0b1010; // a = 10 (binary)
       int b = 05;     // b = 5 (octal)

       System.out.println("ADD    : " + (a + b)); // 10 + 5 = 15
       System.out.println("SUB    : " + (a - b)); // 10 - 5 = 5
       System.out.println("MUL    : " + (a * b)); // 10 * 5 = 50
       System.out.println("DIV    : " + (a / b)); // 10 / 5 = 2
   }
}
```

Output:

```
yaml
CopyEdit
ADD    : 15
SUB    : 5
MUL    : 50
DIV    : 2
```

Another Arithmetic Example:

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       System.out.println(0b1100 + 012);   // 12 (binary: 12+ octal: 10) =
22
       System.out.println(0xabc - 2560);    // (Hex: 2748 - 2560) = 188
       System.out.println(010 * 0b10);       // (Octal 10 = 8, Binary 10 =
2) => 8 * 2 = 16
       System.out.println(120 / 0B1010);      // (120 / binary 1010 = 10) =
12
   }
}
```

Output:

```
CopyEdit
22
188
16
12
```

Number System Identification

Question: Find the number systems for the following numbers.

Answer:

1. `1564` — Decimal Number System
2. `0x675dea` — Hexa-decimal Number System
3. `0674` — Octal Number System
4. `0b1010` — Binary Number System
5. `0X12345` — Hexa-decimal Number System
6. `00` — Octal Number System
7. `0` — Decimal Number System

Note:
When a number is provided in any number system, the Java compiler recognizes its number system by its prefix, converts it to its decimal equivalent, and passes the decimal value to the JVM for processing.
Arithmetic operations can be performed over numbers from different number systems.

Final Notes

- **Binary Literals:** Supported from Java 1.7 onwards.
- **Octal Literals:** Use a leading zero; digits must be between 0 and 7.
- **Decimal Literals:** No prefix is required.
- **Hexadecimal Literals:** Use `0x` or `0X` as a prefix; valid digits include 0–9 and a–f.
- The Java compiler converts numbers from their respective systems into decimal form for all arithmetic operations.

Java Keywords, Reserved Words & Operators

---

1. Keywords / Reserved Words

Keywords

- **Definition:**
  A *keyword* is a predefined word in Java that is recognized by the compiler and has a specific internal functionality.
- **Categories of Keywords:**
  1. **Data Types and Return Types:**
     - `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`, `void`.

2. **Access Modifiers:**
    - `public`, `protected`, `private`, `static`, `final`, `abstract`, `native`, `volatile`, `transient`, `synchronized`, `strictfp`, etc.
3. **Flow Controllers:**
    - `if`, `else`, `switch`, `case`, `default`, `for`, `while`, `do`, `break`, `continue`, `return`, etc.
4. **Class / Object Related Keywords:**
    - `class`, `extends`, `interface`, `implements`, `enum`, `new`, `this`, `super`, `package`, `import`, etc.
5. **Exception Handling Keywords:**
    - `throws`, `throw`, `try`, `catch`, `finally`.

Reserved Words

- **Definition:**
  Reserved words are predefined words that are recognized by Java but do not have any internal functionality. These words cannot be used as identifiers in Java.
- **Examples:**
    - `goto`, `const`.

Important:
In Java applications, you can use keywords, but you cannot use reserved words as identifiers. Doing so will cause a compilation error.

---

2. Operators

Overview

- **Definition:**
  An operator is a symbol that performs a specific operation on provided operands.
- **Categories of Operators:**
    1. **Arithmetic Operators:**
        - **Addition:** `+`, `-`, `*`, `/`, `%`
        - **Increment / Decrement:** `++`, `--`
    2. **Assignment Operators:**
        - `=`, `+=`, `-=`, `*=`, `/=`, `%=`
    3. **Comparison Operators:**
        - `==`, `!=`, `<`, `<=`, `>`, `>=`
    4. **Boolean Logical Operators:**
        - Examples: `&`, `|`, `^`
    5. **Bitwise Logical Operators:**
        - Examples: `&`, `|`, `^`, `<<`, `>>`
    6. **Ternary Operator:**
        - Syntax: `Expr1 ? Expr2 : Expr3;`
            - If **Expr1** evaluates to true, **Expr2** is evaluated; if false, **Expr3** is evaluated.
    7. **Short-Circuit Operators:**
        - `&&`, `||`

2.1 Arithmetic Operators Example

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

        System.out.println(a + b); // 13
        System.out.println(a - b); // 7
        System.out.println(a * b); // 30
        System.out.println(a / b); // 3
        System.out.println(a % b); // 1
   }
}
```

Output:

```
CopyEdit
13
7
30
3
1
```

2.2 Increment / Decrement Operators

Increment Operators

- **Pre-increment:**
  Increments the value **before** it is used in an expression.
  Syntax: `++varName`
- **Post-increment:**
  Uses the variable value in the expression **before** incrementing it.
  Syntax: `varName++`

Decrement Operators

- **Pre-decrement:**
  Decrements the value **before** it is used in an expression.
  Syntax: `--varName`
- **Post-decrement:**
  Uses the variable value in the expression **before** decrementing it.
  Syntax: `varName--`

Examples:

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
```

```
        int a = 5;
        System.out.println(++a); // Pre-increment: Output: 6

        int b = 5;
        System.out.println(b++); // Post-increment: Output: 5, then b
becomes 6

        int c = 5;
        System.out.println(--c); // Pre-decrement: Output: 4

        int d = 5;
        System.out.println(d--); // Post-decrement: Output: 5, then d
becomes 4
    }
}
```

Output:

```
CopyEdit
6
5
4
5
```

Another example with multiple operations:

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int a = 10;
        System.out.println(a);      // 10
        System.out.println(++a);    // Pre-increment: 11 (a becomes 11)
        System.out.println(a++);    // Post-increment: 11 (a becomes 12)
        System.out.println(--a);    // Pre-decrement: 11 (a becomes 11)
        System.out.println(a--);    // Post-decrement: 11 (a becomes 10)
        System.out.println(a);      // 10
    }
}
```

Output:

```
CopyEdit
10
11
11
11
11
10
```

More examples:

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
       int a = 8;
       System.out.println(a);      // 8
```

```
        System.out.println(--a);    // 7
        System.out.println(a++);    // 7, then a becomes 8
        System.out.println(a--);    // 8, then a becomes 7
        System.out.println(++a);    // Pre-increment: 8 (a becomes 8)
        System.out.println(a);      // 8
    }
}
```

Output:

```
CopyEdit
8
7
7
8
8
8
```

Complex expressions:

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int a = 7;
        System.out.println(++a - ++a); // Example Output: -1
    }
}
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int a = 8;
        System.out.println(a++ - a++); // Example Output: -1
    }
}
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int a = 7;
        System.out.println(--a + a++ - ++a); // Example Output: 4
    }
}
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int a = 6;
        System.out.println(--a + a-- * ++a - ++a); // Example Output: 24
    }
}
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int a = 5;
        System.out.println(--a * --a + a++ * --a); // Example Output: 21
    }
}
```

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
      int a = 6;
      System.out.println((--a + --a) + (++a - ++a) * (++a - a--) - (--a +
a++)); // Example Output: -1
   }
}
```

2.3 Assignment Operators Example

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
      int a = 10;
      System.out.println(a);  // 10

      a += 2; // a = a + 2 ==> 12
      System.out.println(a);  // 12

      a -= 2; // a = a - 2 ==> 10
      System.out.println(a);  // 10

      a *= 2; // a = a * 2 ==> 20
      System.out.println(a);  // 20

      a /= 2; // a = a / 2 ==> 10
      System.out.println(a);  // 10

      a %= 2; // a = a % 2 ==> 0
      System.out.println(a);  // 0
   }
}
```

Output:

```
CopyEdit
10
12
10
20
10
0
```

2.4 Logical and Bitwise Operators

Boolean Logical Operators (for booleans):

- **Operators:** `&`, `|`, `^`

Truth Table for Boolean Operators:

A	B	A & B	A | B	A ^ B
True	True	True	True	False
True	False	False	True	True
False	True	False	True	True
False	False	False	False	False

Example with booleans:

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
      boolean b1 = true;
      boolean b2 = false;

      System.out.println(b1 & b1); // true
      System.out.println(b1 & b2); // false
      System.out.println(b2 & b1); // false
      System.out.println(b2 & b2); // false

      System.out.println(b1 | b1); // true
      System.out.println(b1 | b2); // true
      System.out.println(b2 | b1); // true
      System.out.println(b2 | b2); // false

      System.out.println(b1 ^ b1); // false
      System.out.println(b1 ^ b2); // true
      System.out.println(b2 ^ b1); // true
      System.out.println(b2 ^ b2); // false
   }
}
```

Output:

```
arduino
CopyEdit
true
false
false
false
true
true
true
false
false
true
true
false
```

Bitwise Operators with Integers

- **Example:**

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int a = 10; // Binary: 1010
       int b = 2;  // Binary: 0010

       System.out.println(a & b);   // 0010 => 2
       System.out.println(a | b);   // 1010 => 10
       System.out.println(a ^ b);   // 1000 => 8
       System.out.println(a << b);  // 10 << 2:  00001010 -> 00101000 => 40
       System.out.println(a >> b);  // 10 >> 2:  00001010 -> 00000010 => 2
   }
}
```

Output:

```
CopyEdit
2
10
8
40
2
```

Explanation:

- **a & b:** Bitwise AND results in `0010` (binary) which is `2`.
- **a | b:** Bitwise OR results in `1010` (binary) which is `10`.
- **a ^ b:** Bitwise XOR results in `1000` (binary) which is `8`.
- **a << b:** Left shift `a` by 2 positions.
- **a >> b:** Right shift `a` by 2 positions.

2.5 Ternary Operator

- **Syntax:** `Expr1 ? Expr2 : Expr3;`
- **Operation:**
  - If **Expr1** evaluates to true, then **Expr2** is evaluated.
  - If **Expr1** evaluates to false, then **Expr3** is evaluated.

Examples:

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int num = 5;
       String numType = num % 2 == 0 ? "EVEN Number" : "ODD Number";
       System.out.println(num + " Is " + numType);
   }
}
```

Output:

```
javascript
CopyEdit
5 Is ODD Number
```

Another example:

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
        int num1 = 10;
        int num2 = 20;
        int big = num1 > num2 ? num1 : num2;
        int small = num1 < num2 ? num1 : num2;
        System.out.println(num1);
        System.out.println(num2);
        System.out.println("Biggest Num : " + big);
        System.out.println("Smallest Num : " + small);
   }
}
```

Output:

```
yaml
CopyEdit
10
20
Biggest Num : 20
Smallest Num : 10
```

Finding the Biggest Number among Three Numbers

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
        int num1 = 10;
        int num2 = 50;
        int num3 = 30;
        int big = (num1 > num2) ? (num1 > num3 ? num1 : num3)
                                : (num2 > num3 ? num2 : num3);
        int small = (num1 < num2) ? (num1 < num3 ? num1 : num3)
                                  : (num2 < num3 ? num2 : num3);
        System.out.println(num1);
        System.out.println(num2);
        System.out.println(num3);
        System.out.println("Biggest Num   : " + big);
        System.out.println("Smallest Num  : " + small);
   }
}
```

Output:

```
yaml
CopyEdit
10
50
30
Biggest Num   : 50
Smallest Num  : 10
```

2.6 Short-Circuit Operators

- **Operators:** `||` (OR), `&&` (AND)
- **Purpose:**
  Improve performance by avoiding unnecessary evaluations.

Difference Between `|` and `||` Operators

- **`|` (Single OR):**
  Evaluates both operands regardless of the first operand's value.
- **`||` (Double OR):**
  If the first operand is true, the second operand is not evaluated (short-circuit).

Example:

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
        int a = 10;
        int b = 10;

        if(a++ == 10 | b++ == 10) {
            System.out.println(a + "     " + b); // Output: 11  11
        }

        int c = 10;
        int d = 10;

        if(c++ == 10 || d++ == 10) {
            System.out.println(c + "    " + d); // Output: 11   10
        }
   }
}
```

Output:

```
CopyEdit
11     11
11     10
```

Difference Between `&` and `&&` Operators

- **`&` (Single AND):**
  Evaluates both operands regardless of the first operand's value.
- **`&&` (Double AND):**
  If the first operand is false, the second operand is not evaluated (short-circuit).

Example:

```
java
CopyEdit
public class Test {
```

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

        if(a++ != 10 & b++ != 10) {
            // No action needed
        }
        System.out.println(a + "    " + b); // Output: 11    11

        int c = 10;
        int d = 10;

        if(c++ != 10 && d++ != 10) {
            // No action needed
        }
        System.out.println(c + "    " + d); // Output: 11    10
    }
}
```

Output:

```
CopyEdit
11    11
11    10
```

Key Points on Short-Circuiting:

- For **OR (`||`)**:
  - If the first operand is **true**, the overall result is **true** without evaluating the second operand.
- For **AND (`&&`)**:
  - If the first operand is **false**, the overall result is **false** without evaluating the second operand.
- Single operators (`|` and `&`) do **not** short-circuit, so they always evaluate both operands.

Final Notes

- **Keywords** are reserved words with intrinsic functionality, while **reserved words** (like `goto` and `const`) cannot be used as identifiers.
- **Operators** in Java are categorized into arithmetic, assignment, comparison, logical, bitwise, ternary, and short-circuit.
- Use **increment/decrement** operators carefully as their pre and post forms yield different results within expressions.
- Short-circuit operators (`||` and `&&`) help optimize performance by avoiding unnecessary evaluations.

Java Data Types and Type Casting

---

1. Data Types

Overview

- **Typed Programming Languages:**
  Programming languages that represent data as per defined types.
  There are two categories:
  1. **Statically Typed Languages:**
     - The type of data is confirmed **before** the data is represented.
     - **Examples:** C, C++, Java
     - **Example:**

```
java
CopyEdit
a = 10;   // Error: Type not specified
int a = 10;   // Correct: Type is specified
int a;
a = 10;       // Correct
```

  2. **Dynamically Typed Languages:**
     - The type is determined **after** the data is represented.
     - **Example:** Python

```
python
CopyEdit
a = 10
print(type(a))  # Output: <class 'int'> (or int32,
depending on the system)
```

In Java

Since Java is a **statically typed language**, you must specify data types when declaring variables. Java data types are divided into:

1. **Primitive Data Types / Primary Data Types**
2. **User-Defined Data Types / Secondary Data Types**

---

2. Primitive Data Types

A. Numeric Data Types

I. Integral (Integer) Data Types

Data Type	Size (Bytes)	Default Value
byte	1	0
short	2	0

Data Type	Size (Bytes)	Default Value
int	4	0
long	8	0L

II. Non-Integral (Floating-Point) Data Types

Data Type	Size (Bytes)	Default Value
float	4	0.0f
double	8	0.0

B. Non-Numeric Data Types

Data Type	Size (Bytes)	Default Value	Notes
char	2	' ' (space)	Represents a single Unicode character
boolean	1 bit	false	Represents true/false values

---

3. User-Defined Data Types (Secondary Data Types)

- **Examples:**
  Arrays, Classes, Abstract Classes, Interfaces, Enums, etc.
- **Notes:**
  - No fixed memory size is predefined; the memory required is based on the number of members.
  - **Example:**

    ```
    java
    CopyEdit
    String str = "abc";  // Minimum memory = 6 bytes (3 characters
    * 2 bytes each)
    int[] ints = {10, 20, 30};  // Minimum memory = 12 bytes (3
    elements * 4 bytes each)
    ```

  - **Default Value:**
    All user-defined types have a default value of **null**.

    ```
    java
    CopyEdit
    String str; // str == null
    int[] ints; // ints == null
    Thread t;  // t == null
    ```

---

4. Advantages of Data Types in Statically Typed Languages

1. **Memory Calculation:**
   Data types help determine the memory size needed for variables.
2. **Range Determination:**
   It is possible to find the minimum and maximum values for variables based on their data types.

Determining Range for Integral Types

- **Formula:**
  For an *n*-bit data type, the range is from:
  -2^(n-1) to 2^(n-1) - 1
- **Example for byte (8 bits):**
  - n = 8
  - Range: -2^(8-1) to 2^(8-1) - 1 → **-128 to 127**
- **Wrapper Classes:**
  Java provides wrapper classes in the `java.lang` package to access constants for minimum and maximum values.

Primitive Type	Wrapper Class
byte	`java.lang.Byte`
short	`java.lang.Short`
int	`java.lang.Integer`
long	`java.lang.Long`
float	`java.lang.Float`
double	`java.lang.Double`
char	`java.lang.Character`
boolean	`java.lang.Boolean`

Example:

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
        System.out.println(Byte.MIN_VALUE + "------>" + Byte.MAX_VALUE);
        System.out.println(Short.MIN_VALUE + "------>" + Short.MAX_VALUE);
        System.out.println(Integer.MIN_VALUE + "------>" +
Integer.MAX_VALUE);
        System.out.println(Long.MIN_VALUE + "------>" + Long.MAX_VALUE);
        System.out.println(Float.MIN_VALUE + "------>" + Float.MAX_VALUE);
        System.out.println(Double.MIN_VALUE + "------>" + Double.MAX_VALUE);
        System.out.println(Character.MIN_VALUE + "------>" +
Character.MAX_VALUE);
```

```
       // System.out.println(Boolean.MIN_VALUE + "------>" +
Boolean.MAX_VALUE);  // Error: Boolean has no min/max constants
   }
}
```

Output (example values):

```
rust
CopyEdit
-128------>127
-32768------>32767
-2147483648------>2147483647
-9223372036854775808------>9223372036854775807
1.4E-45------>3.4028235E38
4.9E-324------>1.7976931348623157E308
[ ]------>[ÿ]  (Character range, displayed as non-printable characters)
```

---

5. Type Casting

Definition:
The process of converting data from one data type to another is called **Type Casting**. This provides flexibility to use data across different types.

Categories of Type Casting

1. **Primitive Data Types Type Casting**
2. **User-Defined Data Types Type Casting**

User-Defined Data Types Type Casting:
Converting from one user-defined type to another is only possible if there is an **extends** or **implements** relationship.

5.1 Primitive Data Types Type Casting

There are two types:

1. **Implicit Type Casting (Widening):**
   - Converting data from a lower data type to a higher data type.
   - **Conversion Chart:**

     ```
     csharp
     CopyEdit
     1       2           4         8          4
     byte → short → int → long → float → double
                           ↑
                        char (2 bytes)
     ```

   - **Example:**

     ```
     java
     CopyEdit
     byte b = 10;
     ```

```
int i = b;  // Implicit type casting: no error
System.out.println(b + "    " + i);
```

- **Compiler Behavior:**
  The compiler checks if the right-hand side type is compatible with the left-hand side type. If compatible, no error is raised, and the conversion is done at runtime by the JVM.
- **Error Example:**

```
java
CopyEdit
int i = 10;
byte b = i;  // Error: possible lossy conversion from int to
byte
```

- **Other Examples:**

```
java
CopyEdit
long l = 10;
float f = l;  // Implicit conversion: long to float is allowed
System.out.println(l + "    " + f);
java
CopyEdit
float f = 22.22f;
long l = f;   // Error: cannot implicitly convert float to long
```

- **Special Cases:**
  - **byte to char:**

```
java
CopyEdit
byte b = 65;
char c = b;  // Error: incompatible types (lossy
conversion)
```

  - **char to short:**

```
java
CopyEdit
char c = 'A';
short s = c; // Error: incompatible types
```

2. **Explicit Type Casting (Narrowing):**
   - Converting data from a higher data type to a lower data type.
   - **Syntax Pattern:**

```
java
CopyEdit
P a = (Q) b;
```

   where **b** is of a higher data type, and **(Q)** is the cast operator.

   - **Example:**

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int i = 10;
       byte b = (byte) i;  // Explicitly cast int to byte
       System.out.println(i + "   " + b);
   }
}
```

Output:

```
CopyEdit
10   10
```

- **More Examples:**

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       float f = 22.22f;
       long l = (long) f;  // f is cast to long, value becomes
22
       System.out.println(f + "    " + l);
   }
}
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       byte b = 65;
       char c = (char) b;  // 65 becomes 'A'
       System.out.println(b + "    " + c);
   }
}
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       char c = 'A';
       short s = (byte) c;  // Casting char to byte and then
assigning to short
       System.out.println(c + "    " + s);
   }
}
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int i = 128;
       byte b = (byte) i;  // 128 overflows byte range, results
in -128
       System.out.println(i + "    " + b);
   }
}
```

- **Arithmetic with Mixed Types:**
  When adding two variables, if both are among {byte, short, int}, the result is an **int**.

If either operand is of a higher type (e.g., long, float, double), the result is of that higher type.

Examples:

- `byte + byte = int`
- `byte + long = long`
- `float + long = float`
- `long + double = double`

Example:

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       byte b = 10;
       float f = 22.22f;
       // long l = b + f;  // Error: b+f results in float;
cannot assign to long
   }
}
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       long l = 10;
       double d = 22.222;
       // float f = l + d; // Error: l+d results in double;
cannot assign to float
   }
}
```

- **Complex Casting Example:**

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       double d = 22.345;
       byte b = (byte)(short)(int)(long)(float) d;
       System.out.println(b);
   }
}
```

Output:

```
CopyEdit
22
```

Handling Overflows with Casting

- When a value exceeds the maximum value of a data type, it wraps around.

Example:

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       byte b1 = 60;
       byte b2 = 70;
       byte b = (byte)(b1 + b2);  // 60 + 70 = 130; 130 overflows
byte range, resulting in -126
       System.out.println(b);
   }
}
```

Output:

```
diff
CopyEdit
-126
```

Additional Explanation:

- For a byte, the range is **-128 to 127**.
- When adding two byte values, if the sum exceeds 127, it wraps around by subtracting (MAX_VALUE + 1).

Final Summary Table

Category	Example Data Types	Type Casting	Notes
Primitive Data Types	byte, short, int, long, float, double, char, boolean	Implicit (Widening) and Explicit (Narrowing)	Must declare type in statically typed languages
User-Defined Data Types	Arrays, Classes, Interfaces, Enums	Only allowed with inheritance/implementation relationships	Default value is `null`
Implicit Casting	byte → int, int → long, etc.	No cast operator needed	Compiler ensures compatibility
Explicit Casting	int → byte, float → long, etc.	Use cast operator (e.g., `(byte)`)	May cause data loss if the value is out of range
Wrapper Classes	Byte, Short, Integer, Long, Float, Double, Character, Boolean	–	Provide constants for MIN_VALUE and MAX_VALUE

Java Statements

Overview

A **statement** in Java is a collection of expressions that together perform an action. Java provides different kinds of statements, which include:

1. **General Purpose Statements**
2. **Conditional Statements**
3. **Iterative Statements**
4. **Transfer Statements**
5. **Miscellaneous Statements**

---

1. General Purpose Statements

- **Usage:**
  Common in all Java applications for:
  - Declaring variables, methods, classes, abstract classes, interfaces, etc.
  - Creating objects.
  - Accessing variables and methods.
- (Additional examples and details can be added as needed.)

---

2. Conditional Statements

Conditional statements enable execution of a block of code based on a condition.

2.1 The if Statement

Syntax Variations:

- **Syntax-1:** Single if statement

```
java
CopyEdit
if (Condition) {
    // Instructions
}
```

- **Syntax-2:** if-else statement

```
java
CopyEdit
if (Condition) {
    // Instructions if condition is true
} else {
    // Instructions if condition is false
}
```

- **Syntax-3:** if-else if-else chain

```
java
CopyEdit
if (Condition-1) {
    // Instructions if Condition-1 is true
} else if (Condition-2) {
    // Instructions if Condition-2 is true
} else {
    // Instructions if none of the above conditions are true
}
```

Example 1: Check Even or Odd Number

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int num = 11;
       if (num % 2 == 0) {
           System.out.println(num + " Is EVEN Number");
       } else {
           System.out.println(num + " Is ODD Number");
       }
   }
}
```

Output:

```
javascript
CopyEdit
11 Is ODD Number
```

Example 2: Find the Biggest Number Among Two Numbers

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int num1 = 20;
       int num2 = 20;
       System.out.println(num1);
       System.out.println(num2);
       if (num1 > num2) {
           System.out.println(num1 + " Is Biggest Number");
       } else if (num2 > num1) {
           System.out.println(num2 + " Is Biggest Number");
       } else {
           System.out.println(num1 + "," + num2 + " are equal");
       }
   }
}
```

Example Scenarios:

- **Input:** 10, 20 → **Output:** 20
- **Input:** 20, 10 → **Output:** 20
- **Input:** 10, 10 → **Output:** Both are same

Example 3: Find the Biggest Number Among Three Numbers

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
        int num1 = 20;
        int num2 = 20;
        int num3 = 20;
        System.out.println(num1);
        System.out.println(num2);
        System.out.println(num3);
        if (num1 >= num2 && num1 > num3) {
            System.out.println(num1 + " Is Biggest Number");
        } else if (num2 >= num3 && num2 > num1) {
            System.out.println(num2 + " Is Biggest Number");
        } else if (num3 >= num1 && num3 > num2) {
            System.out.println(num3 + " Is Biggest Number");
        } else {
            System.out.println("All are equal");
        }
   }
}
```

Output Example:

```
sql
CopyEdit
20
20
20
All are equal
```

Example 4: Determine Student Status Based on Marks

Criteria:

1. Marks < 0 or > 100: **Invalid marks**
2. Marks < 35: **FAIL**
3. Marks between 35 and 50: **THIRD CLASS**
4. Marks between 50 and 60: **SECOND CLASS**
5. Marks between 60 and 70: **FIRST CLASS**
6. Marks between 70 and 100: **DISTINCTION**

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
```

```
        String sid = "S-111";
        String sname = "Durga";
        int smarks = 100;
        String saddr = "Hyd";
        String status = "";
        if (smarks < 0 || smarks > 100) {
            status = "INVALID MARKS";
        } else {
            if (smarks < 35) {
                status = "FAIL";
            } else if (smarks < 50) {
                status = "THIRD CLASS";
            } else if (smarks < 60) {
                status = "SECOND CLASS";
            } else if (smarks < 70) {
                status = "FIRST CLASS";
            } else {
                status = "DISTINCTION";
            }
        }
        System.out.println("Student Details");
        System.out.println("--------------------");
        System.out.println("Student Id      : " + sid);
        System.out.println("Student Name    : " + sname);
        System.out.println("Student Address : " + saddr);
        System.out.println("Student marks   : " + smarks);
        System.out.println("Student Status  : " + status);
    }
}
```

Output:

```
yaml
CopyEdit
Student Details
--------------------
Student Id      : S-111
Student Name    : Durga
Student Address : Hyd
Student marks   : 100
Student Status  : DISTINCTION
```

Example 5: Calculate Customer Tax Based on Annual Income

Tax Slabs:

1. Income < 5L: 0% tax
2. Income between 5L and 10L: 5% tax
3. Income between 10L and 50L: 10% tax
4. Income between 50L and 1CR: 15% tax
5. Income between 1CR and 10CR: 20% tax
6. Income between 10CR and 50CR: 25% tax
7. Income > 50CR: 30% tax

```
java
CopyEdit
```

```
public class Test {
   public static void main(String[] args) {
        String customerId = "C-111";
        String customerName = "Durga";
        String customerAddress = "Hyd";
        long annualIncome = 500000L;
        int taxPay = 0;

        if (annualIncome < 0) {
            System.out.println("Invalid Annual Income");
        } else {
            if (annualIncome < 500000) {
                taxPay = 0;
            } else if (annualIncome < 1000000) {
                taxPay = (int)(annualIncome * 5 / 100);
            } else if (annualIncome < 5000000) {
                taxPay = (int)(annualIncome * 10 / 100);
            } else if (annualIncome < 10000000) {
                taxPay = (int)(annualIncome * 15 / 100);
            } else if (annualIncome < 100000000) {
                taxPay = (int)(annualIncome * 20 / 100);
            } else if (annualIncome < 50000000) {
                taxPay = (int)(annualIncome * 25 / 100);
            } else {
                taxPay = (int)(annualIncome * 30 / 100);
            }
        }
        System.out.println("Customer Details");
        System.out.println("----------------------");
        System.out.println("Customer Id              : " + customerId);
        System.out.println("Customer Name            : " + customerName);
        System.out.println("Customer Address         : " + customerAddress);
        System.out.println("Customer Annual Income   : " + annualIncome);
        System.out.println("Customer Tax Payment     : " + taxPay);
   }
}
```

Output:

```
yaml
CopyEdit
Customer Details
----------------------
Customer Id              : C-111
Customer Name            : Durga
Customer Address         : Hyd
Customer Annual Income   : 500000
Customer Tax Payment     : 25000
```

---

3. Default Values for Variables

- **Class-Level Variables:**
  Have default values.
  Example: An uninitialized class-level variable of type `int` defaults to 0.
- **Local Variables:**
  Do **not** have default values. They must be explicitly initialized before use.

Example Demonstrating Local Variable Error:

```
java
CopyEdit
class A {
    int i; // Class-level variable; default is 0
    void m1() {
        int j; // Local variable; no default value
        System.out.println(i); // OK: prints 0
        System.out.println(j); // Error: variable j might not have been
initialized
        j = 20;
        System.out.println(j); // OK: prints 20
    }
}
```

---

4. Conditional Expressions in Java

There are two types of conditional expressions in Java:

1. **Constant Conditional Expressions:**
    - Contain only constants (including final variables) and are evaluated by the compiler.
    - **Examples:**
        - `if (10 == 10) { }`
        - `if (true) { }`
        - ```
          java
          CopyEdit
          final int i = 10;
          if (i == 10) { }
          ```
2. **Variable Conditional Expressions:**
    - Contain at least one non-final variable and are evaluated by the JVM.
    - **Examples:**

      ```
      java
      CopyEdit
      int i = 10;
      int j = 10;
      if (i == j) { }
      java
      CopyEdit
      int i = 10;
      if (i == 10) { }
      ```

Local Variable Initialization and Conditional Expressions

- **Example 1:**

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

```
        if (i == 10) {
            j = 20;
        }
        System.out.println(j); // Error: variable j might not have
been initialized
    }
}
```

- **Example 2:**

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int i = 10;
        int j;
        if (i == 10) {
            j = 20;
        } else {
            j = 30;
        }
        System.out.println(j); // OK: prints 20 or 30 depending on
condition
    }
}
```

- **Example 3:** (if-else if without final variable)

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int i = 10;
        int j;
        if (i == 10) {
            j = 20;
        } else if (i == 20) {
            j = 30;
        }
        System.out.println(j); // Error: variable j might not have
been initialized
    }
}
```

- **Example 4:** (Using complete if-else if-else chain)

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int i = 10;
        int j;
        if (i == 10) {
            j = 20;
        } else if (i == 20) {
            j = 30;
        } else {
            j = 40;
        }
```

```
            System.out.println(j); // OK: prints 20
        }
    }
```

- **Example 5:** (Using final variable)

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        final int i = 10;
        int j;
        if (i == 10) {
            j = 20;
        }
        System.out.println(j); // OK: prints 20
    }
}
```

- **Example 6:** (if with constant condition)

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int j;
        if (true) {
            j = 20;
        }
        System.out.println(j); // OK: prints 20
    }
}
```

---

5. The switch Statement

A **switch** statement is used to execute one out of multiple code blocks based on the value of a variable. It is commonly used in menu-driven applications.

General Syntax

```
java
CopyEdit
switch(varName) {
    case 1:
        // Instructions when varName == 1
        break;
    case 2:
        // Instructions when varName == 2
        break;
    // ...
    case n:
        // Instructions when varName == n
        break;
    default:
        // Instructions if none of the cases match
        break;
```

```
}
```

- The JVM compares the switch parameter with each case value sequentially. When a match is found, it executes the instructions in that case until a **break** statement is encountered, which transfers control outside the switch.
- If no case matches, the **default** block (if present) is executed.

Example: Display Weekday Based on Day Number

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
        int dayNo = 5;
        switch (dayNo) {
            case 1:
                System.out.println("MONDAY");
                break;
            case 2:
                System.out.println("TUESDAY");
                break;
            case 3:
                System.out.println("WEDNESDAY");
                break;
            case 4:
                System.out.println("THURSDAY");
                break;
            case 5:
                System.out.println("FRIDAY");
                break;
            case 6:
                System.out.println("SATURDAY");
                break;
            case 7:
                System.out.println("SUNDAY");
                break;
            default:
                System.out.println("Invalid WeekDay, please provide a value
from 1 to 7.");
                break;
        }
   }
}
```

Output:

```
nginx
CopyEdit
FRIDAY
```

Rules and Regulations for switch

1. **Allowed Parameter Types:**
    - Data types such as **byte, short, int,** and **char** are allowed.
    - **Not allowed:** long, float, double, and boolean.

Example (Compilation Error with long):

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        long l = 20;
        switch (l) {  // Error: long is not allowed in switch
            case 5:
                System.out.println("FIVE");
                break;
            //...
        }
    }
}
```

2. **String in switch:**
    - Up to Java 6, switch does not support **String**.
    - From Java 7 onwards, **String** is allowed.

Example:

```
java
CopyEdit
class Test {
    public static void main(String[] args) {
        String data = "BBB";
        switch(data) {
            case "AAA":
                System.out.println("AAA");
                break;
            case "BBB":
                System.out.println("BBB");
                break;
            case "CCC":
                System.out.println("CCC");
                break;
            case "DDD":
                System.out.println("DDD");
                break;
            default:
                System.out.println("Default");
                break;
        }
    }
}
```

    - **Java 6:** Compilation error (String not allowed)
    - **Java 7+:** Compiles and runs correctly.
3. **Cases and Default are Optional:**
    - You can write a switch with only a default block, only cases, or both.

Example (Only Default):

```
java
CopyEdit
public class Test {
```

```
    public static void main(String[] args) {
        int i = 10;
        switch (i) {
            default:
                System.out.println("Default");
                break;
        }
    }
}
```

Example (Only Cases, No Default):

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int i = 10;
        switch (i) {
            case 5:
                System.out.println("FIVE");
                break;
            case 10:
                System.out.println("TEN");
                break;
            case 15:
                System.out.println("FIFTEEN");
                break;
            case 20:
                System.out.println("TWENTY");
                break;
        }
    }
}
```

- In the above, if no case matches, nothing is printed.

4. **Order of Cases:**
   - The order of cases and default is not fixed. They can be arranged in any order.

Example:

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int i = 10;
        switch (i) {
            default:
                System.out.println("Default");
                break;
            case 5:
                System.out.println("FIVE");
                break;
            case 20:
                System.out.println("TWENTY");
                break;
            case 10:
                System.out.println("TEN");
                break;
            case 15:
```

```
                System.out.println("FIFTEEN");
                break;
        }
    }
}
```

Output:

```
nginx
CopyEdit
TEN
```

5. **Duplicate Cases are Not Allowed:**
    - Each case value must be unique.

Example (Compilation Error due to duplicate case):

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int i = 10;
        switch (i) {
            case 5:
                System.out.println("FIVE");
                break;
            case 10:
                System.out.println("TEN");
                break;
            case 15:
                System.out.println("FIFTEEN");
                break;
            case 20:
                System.out.println("TWENTY");
                break;
            case 10:
                System.out.println("TEN-2");
                break;
            default:
                System.out.println("Default");
                break;
        }
    }
}
```

    - Duplicate case label (`case 10:`) will result in an error.
6. **Case Values Must Be Within the Data Type Range:**
    - All case values must be within the range of the data type of the switch parameter.

Example (Error with out-of-range value for byte):

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        byte b = 126;
        switch (b) {
```

```
            case 125:
                System.out.println("125");
                break;
            case 126:
                System.out.println("126");
                break;
            case 127:
                System.out.println("127");
                break;
            case 128:  // Error: 128 is outside the byte range (-128
to 127)
                System.out.println("128");
                break;
            default:
                System.out.println("Default");
                break;
        }
    }
}
```

7. **Case Values Must Be Constant Expressions:**
    - All case labels must be constants (or final variables), not normal variables.

Example (Error with non-final variables):

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int i = 5, j = 10, k = 15, l = 20;
        switch (10) {
            case i:
                System.out.println("FIVE");
                break;
            case j:
                System.out.println("TEN");
                break;
            case k:
                System.out.println("FIFTEEN");
                break;
            case l:
                System.out.println("TWENTY");
                break;
            default:
                System.out.println("Default");
                break;
        }
    }
}
```

    - To fix this, declare the case values as **final**:

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        final int i = 5, j = 10, k = 15, l = 20;
        switch (10) {
            case i:
```

```
                System.out.println("FIVE");
                break;
            case j:
                System.out.println("TEN");
                break;
            case k:
                System.out.println("FIFTEEN");
                break;
            case l:
                System.out.println("TWENTY");
                break;
            default:
                System.out.println("Default");
                break;
        }
    }
}
```

Output:

```
nginx
CopyEdit
TEN
```

Java Iterative Statements

Overview

Iterative statements allow the JVM to execute a set of instructions repeatedly based on a given conditional expression. The most common iterative statements in Java are:

- **for loop**
- **while loop**
- **do-while loop**

---

1. The for Loop

Syntax

```
java
CopyEdit
for (Expr1; Expr2; Expr3) {
    // Loop Body
}
```

- **Expr1:** Initialization (executed once at the beginning).
- **Expr2:** Condition (evaluated before each iteration; loop continues if true).
- **Expr3:** Update expression (executed at the end of each iteration).
- **Loop Body:** The statements that are executed as long as the condition is true.

Q) Loop Execution Analysis

Consider the following code:

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.println(i);
        }
   }
}
```

- **Expr1:** `int i = 0;` – executed **1 time**.
- **Expr2:** `i < 10;` – evaluated **11 times** (once before each iteration, including one extra time when the condition becomes false).
- **Expr3:** `i++` – executed **10 times**.
- **Loop Body:** `System.out.println(i);` – executed **10 times**.

---

2. Examples Using the for Loop

Example 1: Display Even and Odd Numbers Up to 100

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
        for (int num = 1; num <= 100; num++) {
            if (num % 2 == 0) {
                System.out.println(num + " Is EVEN Number");
            } else {
                System.out.println(num + " Is ODD Number");
            }
        }
   }
}
```

Output:
Each number from 1 to 100 is printed along with a message indicating whether it is even or odd.

Example 2: Factorial of a Given Number

Factorial of 5: 1 × 2 × 3 × 4 × 5 = 120

Using Incrementing Loop:

```
java
CopyEdit
public class Test {
```

```
   public static void main(String[] args) {
        int num = 5;
        int result = 1;
        for (int i = 1; i <= num; i++) {
            result = result * i;
        }
        System.out.println("Factorial of " + num + " Is " + result);
   }
}
```

Output:

```
vbnet
CopyEdit
Factorial of 5 Is 120
```

Using Decrementing Loop:

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
        int num = 5;
        int result = 1;
        for (int i = num; i >= 1; i--) {
            result = result * i;
        }
        System.out.println("Factorial of " + num + " Is " + result);
   }
}
```

Output:

```
vbnet
CopyEdit
Factorial of 5 Is 120
```

Example 3: Check if a Number is Prime

A prime number is only divisible by 1 and itself.

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
        int num = 7;
        int count = 0;
        for (int i = 1; i <= num; i++) {
            if (num % i == 0) {
                count = count + 1;
            }
        }
        if (count == 2) {
            System.out.println(num + " Is a Prime Number");
        } else {
            System.out.println(num + " Is not a Prime Number");
        }
   }
}
```

Output:

```
mathematica
CopyEdit
7 Is a Prime Number
```

Example 4: Display All Prime Numbers Less Than 100

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       for (int num = 1; num <= 100; num++) {
           int count = 0;
           for (int i = 1; i <= num; i++) {
               if (num % i == 0) {
                   count = count + 1;
               }
           }
           if (count == 2) {
               System.out.println(num + " Is a Prime Number");
           } else {
               System.out.println(num + " Is not a Prime Number");
           }
       }
   }
}
```

Output:
The program prints for each number from 1 to 100 whether it is prime or not.

Example 5: Display Fibonacci Series Up to 100

Fibonacci series: 0 1 1 2 3 5 8 13 21 34 55 89

Version 1: Using a for Loop

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int num1 = 0;
       int num2 = 1;
       System.out.print(num1 + " " + num2 + " ");
       int nextVal = 0;
       for (int x = 1; x <= 100; x++) {
           nextVal = num1 + num2;
           if (nextVal < 100) {
               System.out.print(nextVal + " ");
               num1 = num2;
               num2 = nextVal;
           }
       }
   }
}
```

Output:

```
CopyEdit
0 1 1 2 3 5 8 13 21 34 55 89
```

Optimized Version 1:

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int num1 = 0;
       int num2 = 1;
       System.out.print(num1 + " " + num2 + " ");
       int nextVal = num1 + num2;
       for (; nextVal <= 100; nextVal = num1 + num2) {
           System.out.print(nextVal + " ");
           num1 = num2;
           num2 = nextVal;
       }
   }
}
```

Optimized Version 2 (Using multiple declarations in for header):

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       System.out.print(0 + " " + 1 + " ");
       for (int num1 = 0, num2 = 1, nextVal = num1 + num2; nextVal <= 100;
nextVal = num1 + num2) {
           System.out.print(nextVal + " ");
           num1 = num2;
           num2 = nextVal;
       }
   }
}
```

Output (for both optimized versions):

```
CopyEdit
0 1 1 2 3 5 8 13 21 34 55 89
```

---

3. Optional Expressions in the for Loop

- **Expr1 (Initialization):**
  - Optional. You can omit it (e.g., declare loop variable before the loop).
  - Example without initialization in the header:

    ```
    java
    CopyEdit
    int i = 0;
    for (; i < 10; i++) {
        System.out.println(i);
    }
    ```

  - You can also include any statement (e.g., `System.out.println("Hello");`), but it's best to declare loop variables here.

- **Expr2 (Condition):**
  - Optional. If omitted, it defaults to true, potentially causing an infinite loop.
  - Must be a boolean expression. Non-boolean expressions (like `System.out.println()`) are not allowed.
- **Expr3 (Update):**
  - Optional. You can update the variable inside the loop body if not declared here.
  - Example without update in header:

```
java
CopyEdit
for (int i = 0; i < 10;) {
    System.out.println(i);
    i++;
}
```

- **Body:**
  - If a single statement is used, curly braces (`{}`) are optional.
  - If no statement is provided, you must use either `{}` or a semicolon (`;`) to indicate an empty body.

---

4. Unreachable Statements in the for Loop

- **Unreachable Statement:**
  A statement that the JVM can never execute.

  Example 1:
  If a loop is recognized by the compiler as infinite, any statement immediately following it will be flagged as unreachable.

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       for (;;) { // infinite loop
           System.out.println("Inside Loop");
       }
       // Compiler error: unreachable statement
       // System.out.println("After loop");
   }
}
```

  Example 2:
  When the condition is a constant false, the loop body is unreachable.

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       for (int i = 0; false; i++) {
           System.out.println("Inside Loop"); // Unreachable
       }
   }
```

```
    }
```

Note:
When the condition is a variable (non-constant), the compiler may not flag the loop as infinite—even if it logically is.

---

5. Working with Arrays and the for Loop

Traditional for Loop Example

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int[] ints = {10, 20, 30, 40, 50};
       System.out.println("Array Size    : " + ints.length);
       for (int index = 0; index < ints.length; index++) {
           System.out.println(ints[index]);
       }
   }
}
```

Output:

```
javascript
CopyEdit
Array Size    : 5
10
20
30
40
50
```

Enhanced for-each Loop (Available Since Java 1.5)

- **Syntax:**

  ```
  java
  CopyEdit
  for (ArrayDataType var : ArrayReferenceVariable) {
      // Loop Body
  }
  ```

- **Example:**

  ```
  java
  CopyEdit
  public class Test {
     public static void main(String[] args) {
         int[] ints = {10, 20, 30, 40, 50};
         System.out.println("Array Size    : " + ints.length);
         for (int index = 0; index < ints.length; index++) {
             System.out.println(ints[index]);
  ```

```
            }
            System.out.println();
            for (int element : ints) {
                System.out.println(element);
            }
        }
    }
```

Output:
Both loops produce the same list of array elements. The enhanced loop reduces the risk of errors like index out of bounds and eliminates the need for a loop variable.

Note: The enhanced for-each loop is ideal for iterating over arrays and collections when you don't need the index.

Summary Table: for Loop Components

Component	Description	Optional?	Notes
Expr1	Initialization (e.g., `int i = 0;`)	Yes (can be omitted)	Best used for variable declaration
Expr2	Condition (e.g., `i < 10;`)	Yes (defaults to `true` if omitted)	Must evaluate to a boolean
Expr3	Update (e.g., `i++`)	Yes (can be done in the body)	Commonly used for increment/decrement
Loop Body	Statements executed in each iteration	Required (even if empty: `;`)	Curly braces optional if only one statement

Java Iterative & Transfer Statements

---

1. While Loop

Overview

- The **while** loop is an iterative statement that repeatedly executes a block of instructions as long as a specified condition is true.
- It is generally used when the number of iterations is not known in advance.

Syntax

```
java
```

```
CopyEdit
while (condition) {
    // Loop Body: instructions to execute
}
```

- The **condition** is evaluated before each iteration.
    - If **true**, the loop body executes.
    - If **false**, the loop is terminated and control moves outside the loop.

Example

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int i = 0;
       while (i < 10) {
           System.out.println(i);
           i = i + 1;
       }
   }
}
```

Output:

```
CopyEdit
0
1
2
3
4
5
6
7
8
9
```

Key Points

- **Conditional Expression is Mandatory:**
  Writing a while loop without a condition (e.g., `while ()`) produces a compilation error.
- **Infinite Loop Possibility:**
  If the condition always evaluates to true (for example, `while (true)`), the loop becomes infinite.
    - When a constant condition (like `true`) is used, the compiler recognizes unreachable statements following the loop.

Example of Infinite Loop (with Unreachable Statement)

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int i = 0;
       System.out.println("Before loop");
       while (true) {
           System.out.println("Inside Loop");
```

```
        }
        // This line is unreachable and causes a compilation error.
        System.out.println("After Loop");
    }
}
```

Example of a Loop with a Constant False Condition

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        System.out.println("Before Loop");
        while (false) {
            System.out.println("Inside Loop");
        }
        // Compiler error: The loop body is unreachable because the
condition is false.
        System.out.println("After Loop");
    }
}
```

---

2. Displaying Digits and Sum Using a While Loop

Example: Display Each Digit of a Number Individually

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        int num = 12345;
        System.out.println("Original Number   : " + num);
        String forwardNumber = "";
        String backwardNumber = "";
        int digit = 0;
        while (num != 0) {
            digit = num % 10;
            num = num / 10;
            forwardNumber = digit + "  " + forwardNumber;
            backwardNumber = backwardNumber + "  " + digit;
        }
        System.out.println("Forward Number   : " + forwardNumber);
        System.out.println("Backward number: " + backwardNumber);
    }
}
```

Output:

```
mathematica
CopyEdit
Original Number   : 12345
Forward Number   : 1  2  3  4  5
Backward number:  5  4  3  2  1
```

Example: Sum and Multiply Digits

```
java
CopyEdit
```

```
public class Test {
   public static void main(String[] args) {
       int num = 12345;
       System.out.println("Original Number   : " + num);
       int sumResult = 0;
       int mulResult = 1;
       int digit = 0;
       while (num != 0) {
           digit = num % 10;
           num = num / 10;
           sumResult = sumResult + digit;
           mulResult = mulResult * digit;
       }
       System.out.println("SUM   : " + sumResult);
       System.out.println("MUL   : " + mulResult);
   }
}
```

Output:

```
yaml
CopyEdit
Original Number   : 12345
SUM   : 15
MUL   : 120
```

Example: Check for Palindrome Number

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       int num = 12121;
       int originalNum = num;
       System.out.println("Original Number   : " + num);
       int reverseNum = 0;
       int digit = 0;
       while (num != 0) {
           digit = num % 10;
           num = num / 10;
           reverseNum = reverseNum * 10 + digit;
       }
       if (originalNum == reverseNum) {
           System.out.println(originalNum + " Is Palindrome Number");
       } else {
           System.out.println(originalNum + " Is not a Palindrome Number");
       }
   }
}
```

Output:

```
javascript
CopyEdit
Original Number   : 12121
12121 Is Palindrome Number
```

---

3. Do-while Loop

Overview

- The **do-while** loop is similar to the while loop, but it guarantees that the loop body is executed at least once because the condition is checked after the body.

Syntax

```
java
CopyEdit
do {
    // Loop Body: instructions to execute
} while (condition);
```

- The **condition** is evaluated after executing the loop body.
    - If **true**, the loop continues.
    - If **false**, the loop terminates and execution proceeds after the loop.

Key Differences Between While and Do-while

1. **Order of Execution:**
    - **While loop:** Evaluates the condition first; the loop body may not execute if the condition is false initially.
    - **Do-while loop:** Executes the loop body first, then evaluates the condition.
2. **Guaranteed Execution:**
    - The do-while loop executes its body **at least once**.
3. **Usage:**
    - Use a while loop when the number of iterations is uncertain and you might not want any execution if the condition is false.
    - Use a do-while loop when you require the loop body to execute at least once.

Example: Basic Do-while Loop

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
        int i = 0;
        do {
            System.out.println(i);
            i = i + 1;
        } while (i < 10);
   }
}
```

Output:

```
CopyEdit
0
1
2
3
```

```
4
5
6
7
8
9
```

Examples Demonstrating Unreachable Statements

- **Example with Infinite Loop (Constant true condition):**

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       System.out.println("Before Loop");
       do {
           System.out.println("Inside Loop");
       } while (true);
       // Compiler error: unreachable statement below
       System.out.println("After Loop");
   }
}
```

- **Example with Constant False Condition:**

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       System.out.println("Before Loop");
       do {
           System.out.println("Inside Loop");
       } while (false);
       System.out.println("After Loop");
   }
}
```

Output:

```
pgsql
CopyEdit
Before Loop
Inside Loop
After Loop
```

---

4. Transfer Statements

Transfer statements allow you to alter the normal flow of execution. The primary transfer statements in Java are:

- **break**
- **continue**

4.1 Break Statement

- **Purpose:**
  Exits the current loop or switch block immediately.
- **Usage:**
  Placed inside loops or switch statements.

Example: Using break in a for Loop

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       for (int i = 0; i < 10; i++) {
           if (i == 5) {
               break;
           }
           System.out.println(i);
       }
   }
}
```

Output:

```
CopyEdit
0
1
2
3
4
```

- **Unreachable Statement Example:**

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       for (int i = 0; i < 10; i++) {
           if (i == 5) {
               break;
               System.out.println("After break"); // Unreachable
           }
           System.out.println(i);
       }
   }
}
```

- **Nested Loop Example:**
  - Without a label, a break only exits the innermost loop.

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       for (int i = 0; i < 10; i++) {
           for (int j = 0; j < 10; j++) {
               if (j == 5) {
                   break;
```

```
                }
                System.out.println(i + "    " + j);
            }
        }
    }
}
```

- **Labeled break:** To break out of an outer loop.

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        l1: for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                if (j == 5) {
                    break l1;
                }
                System.out.println(i + "    " + j);
            }
        }
    }
}
```

Output for labeled break (only first few iterations):

```
CopyEdit
0    0
0    1
0    2
0    3
0    4
```

4.2 Continue Statement

- **Purpose:**
  Skips the remainder of the current loop iteration and continues with the next iteration of the loop.
- **Usage:**
  Placed inside loops.

Example: Using continue in a for Loop

```
java
CopyEdit
public class Test {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            if (i == 5) {
                continue;
            }
            System.out.println(i);
        }
    }
}
```

Output:

```
CopyEdit
0
1
2
3
4
6
7
8
9
```

- **Unreachable Statement Example:**

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       for (int i = 0; i < 10; i++) {
           if (i == 5) {
               continue;
               System.out.println("After Continue"); // Unreachable
           }
           System.out.println(i);
       }
   }
}
```

- **Nested Loop Example with continue:**
  - Without a label, continue only affects the innermost loop.

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       for (int i = 0; i < 10; i++) {
           for (int j = 0; j < 10; j++) {
               if (j == 5) {
                   continue;
               }
               System.out.println(i + "    " + j);
           }
       }
   }
}
```

  - **Labeled continue:** To continue the next iteration of an outer loop.

```
java
CopyEdit
public class Test {
   public static void main(String[] args) {
       l1: for (int i = 0; i < 10; i++) {
           for (int j = 0; j < 10; j++) {
               if (j == 5) {
                   continue l1;
               }
               System.out.println(i + "    " + j);
           }
       }
```

```
        }
    }
```

Output for labeled continue (outer loop iterations affected):
The output will show iterations of the outer loop until the inner loop reaches `j == 5`, at which point the outer loop moves to the next iteration.

Summary Table: Transfer Statements

Statement	Function	Scope Affected	Notes
break	Exits the current loop or switch entirely	Innermost loop or switch	Unreachable statements after break in same block are errors.
continue	Skips the current iteration and continues next	Innermost loop	Can use a labeled continue to affect an outer loop.