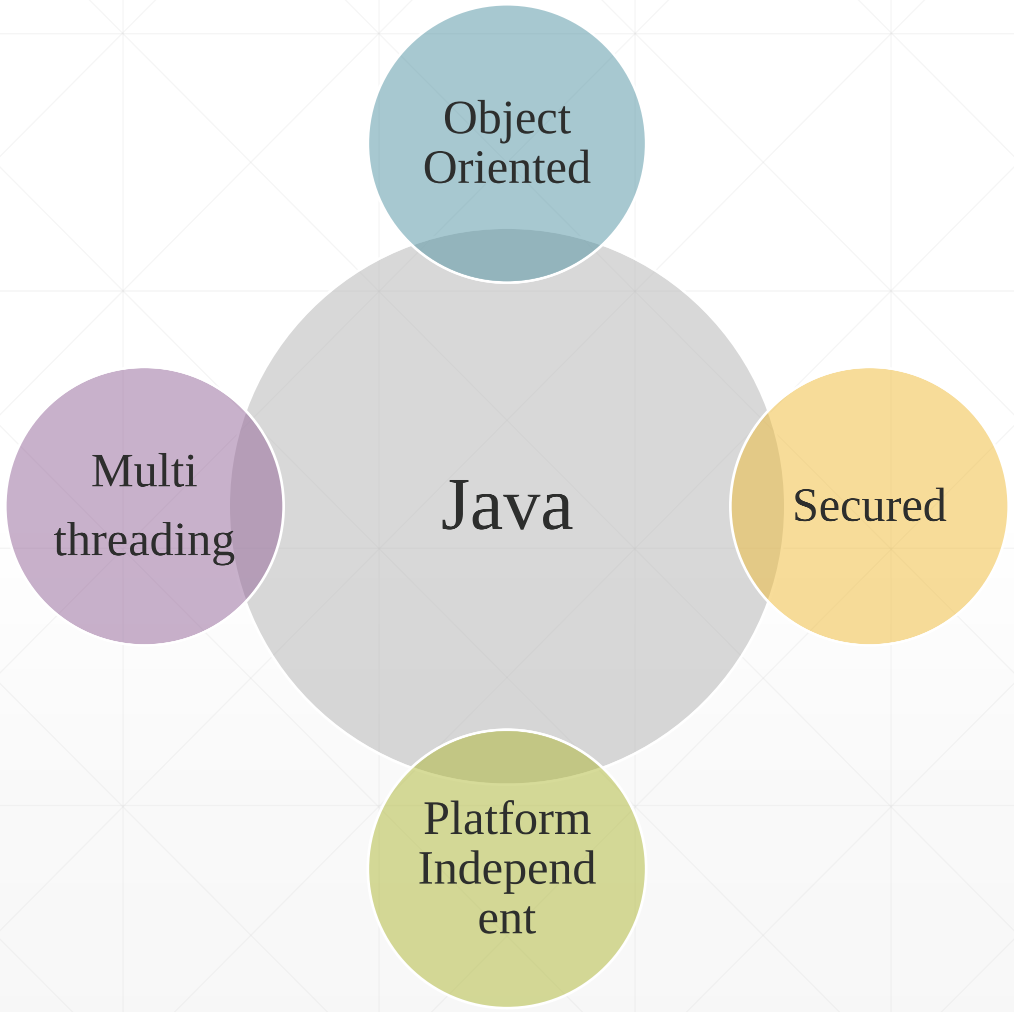
Java Basics

Syntax, and Core Concepts

Presented By:
Mirazul Islam
Java Instructor, EU

Features of Java



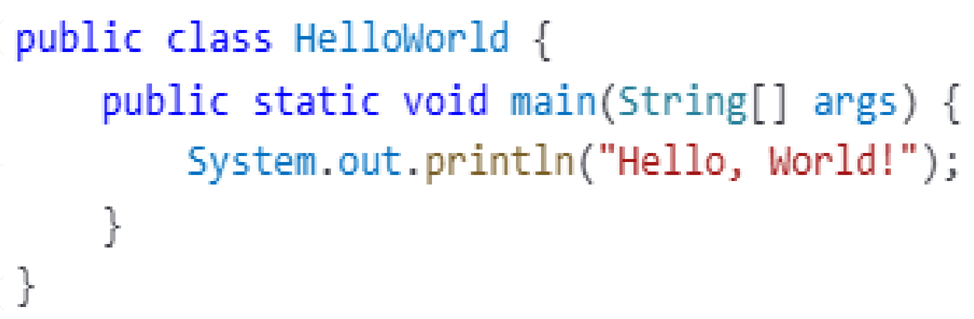
Applications of Java

- Desktop GUI
- Mobile App
- Game Development
- Web Based App
- Enterprise
- Scientific

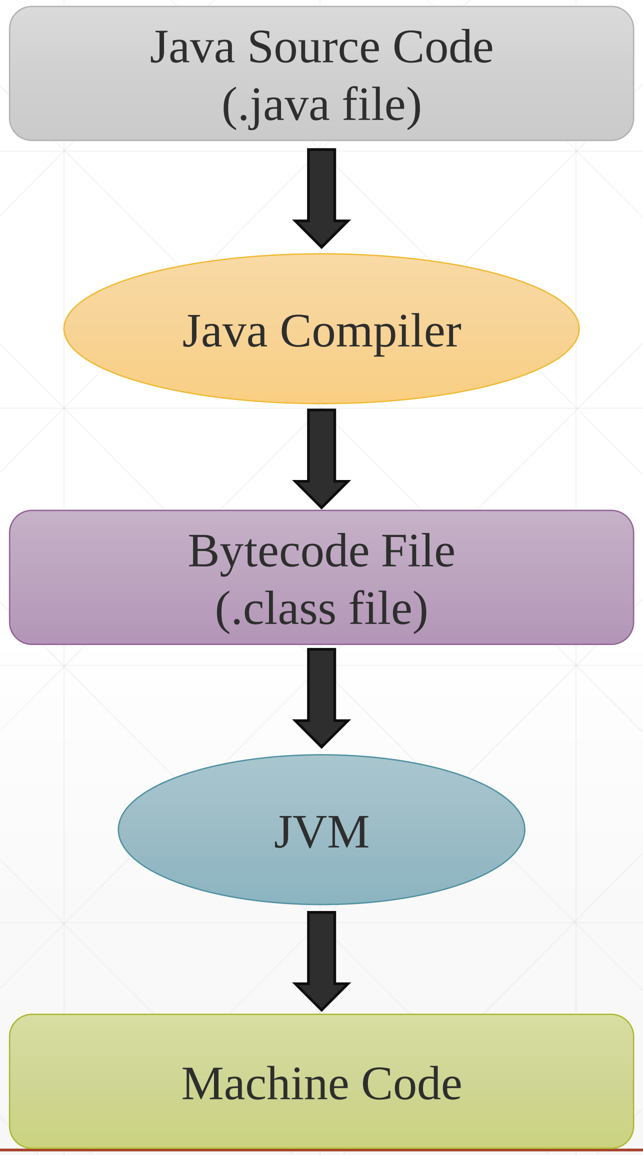
Key Differences Between Java and C++

Feature	**Java**	**C++**
Paradigm	Purely Object-Oriented (except for primitive types)	Supports both Procedural and Object-Oriented Programming
Platform Dependency	Platform-Independent (Runs on JVM)	Platform-Dependent (Compiled to Machine Code)
Memory Management	Automatic Garbage Collection	Manual Memory Management (new/delete)
Multiple Inheritance	Not supported (Uses Interfaces)	Supported
Pointers	Does not support explicit pointers (for security)	Fully supports pointers
Compilation & Execution	Compiled to Bytecode and runs on JVM	Compiled directly to machine code
Speed & Performance	Slightly slower due to JVM overhead	Faster as it compiles to native code

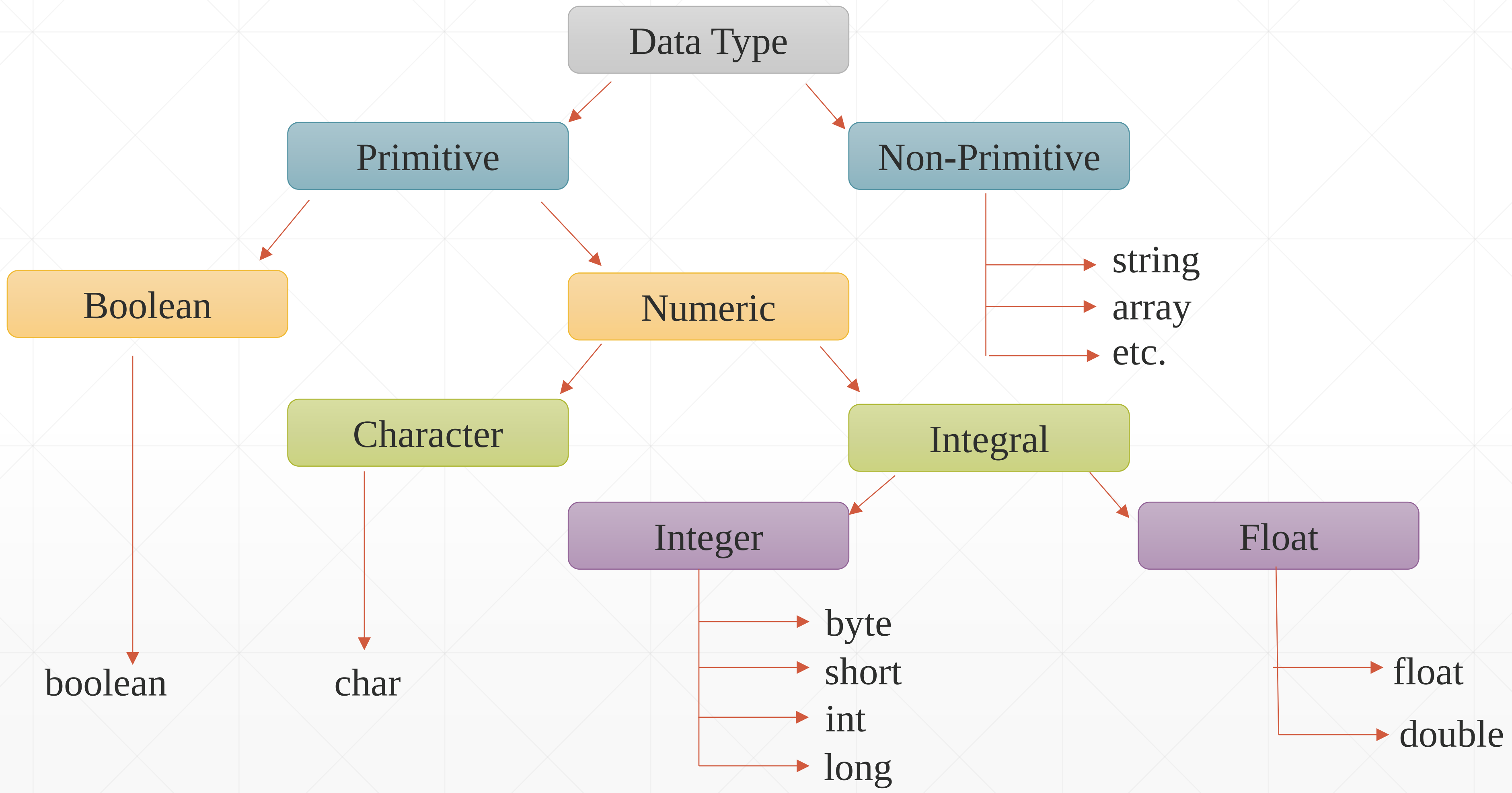
Sample Java Program



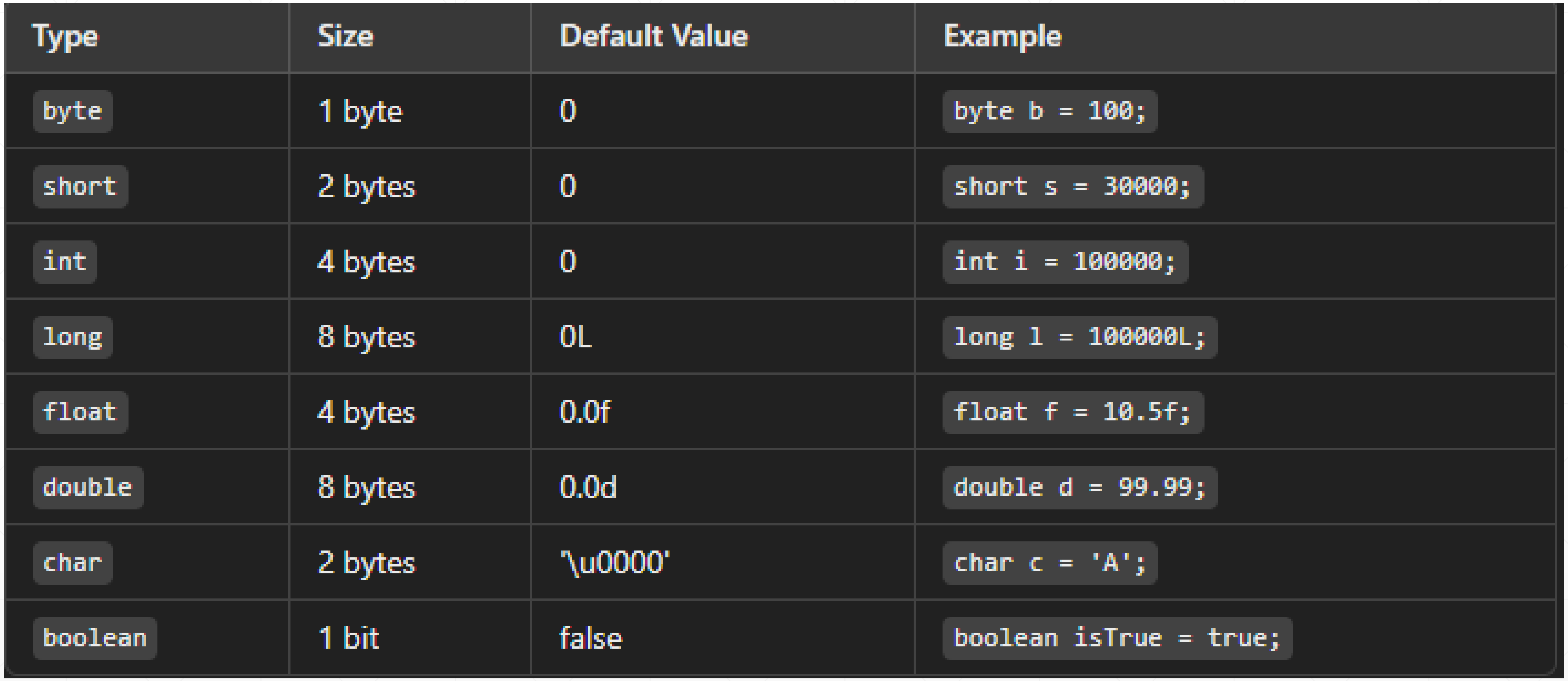
```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```



Data Types in Java



Primitive data types



Type	Size	Default Value	Example
`byte`	1 byte	0	`byte b = 100;`
`short`	2 bytes	0	`short s = 30000;`
`int`	4 bytes	0	`int i = 100000;`
`long`	8 bytes	0L	`long l = 100000L;`
`float`	4 bytes	0.0f	`float f = 10.5f;`
`double`	8 bytes	0.0d	`double d = 99.99;`
`char`	2 bytes	'\u0000'	`char c = 'A';`
`boolean`	1 bit	false	`boolean isTrue = true;`

Type Casting in Java

Converting a variable from one data type to another.

❑ **Implicit Casting (Widening Conversion)**

- Automatically done by Java.
- Converts a smaller type to a larger type.
- Examples:
- int → long → float → double

```
 int a = 10;
double b = a; // Implicit casting
```

Type Casting in Java

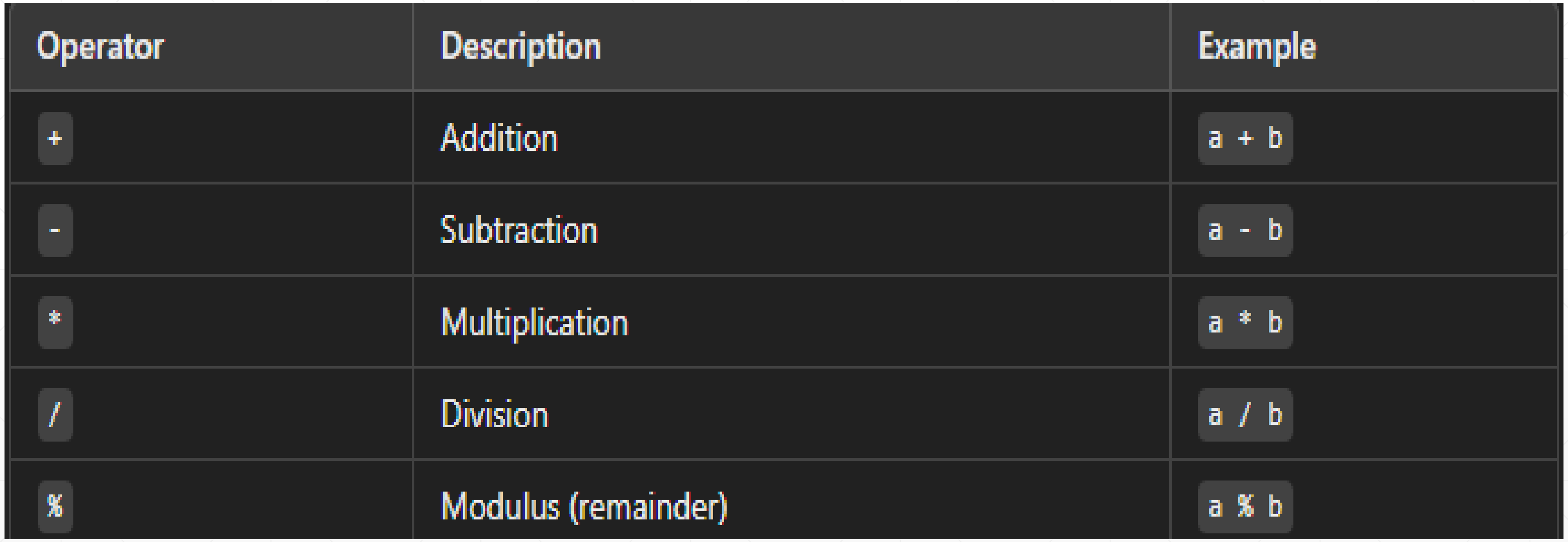
❑ **Explicit Casting (Narrowing Conversion)**

- Must be done manually using a cast operator.
- Converts a larger type to a smaller type.
- Risk of data loss.
- **Example:**

```
double x = 9.8;

int y = (int) x; // Explicit casting
```

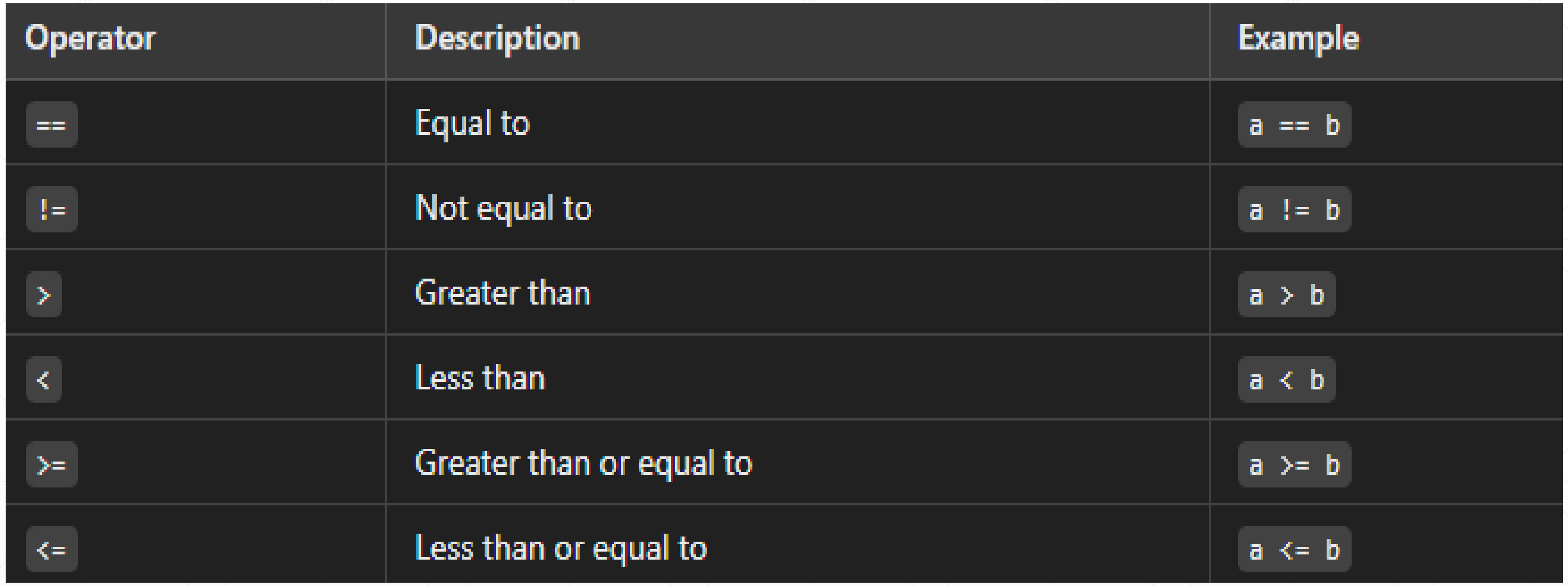
Operators in Java

- Mathematical operations.



Operator	Description	Example
`+`	Addition	`a + b`
`-`	Subtraction	`a - b`
`*`	Multiplication	`a * b`
`/`	Division	`a / b`
`%`	Modulus (remainder)	`a % b`

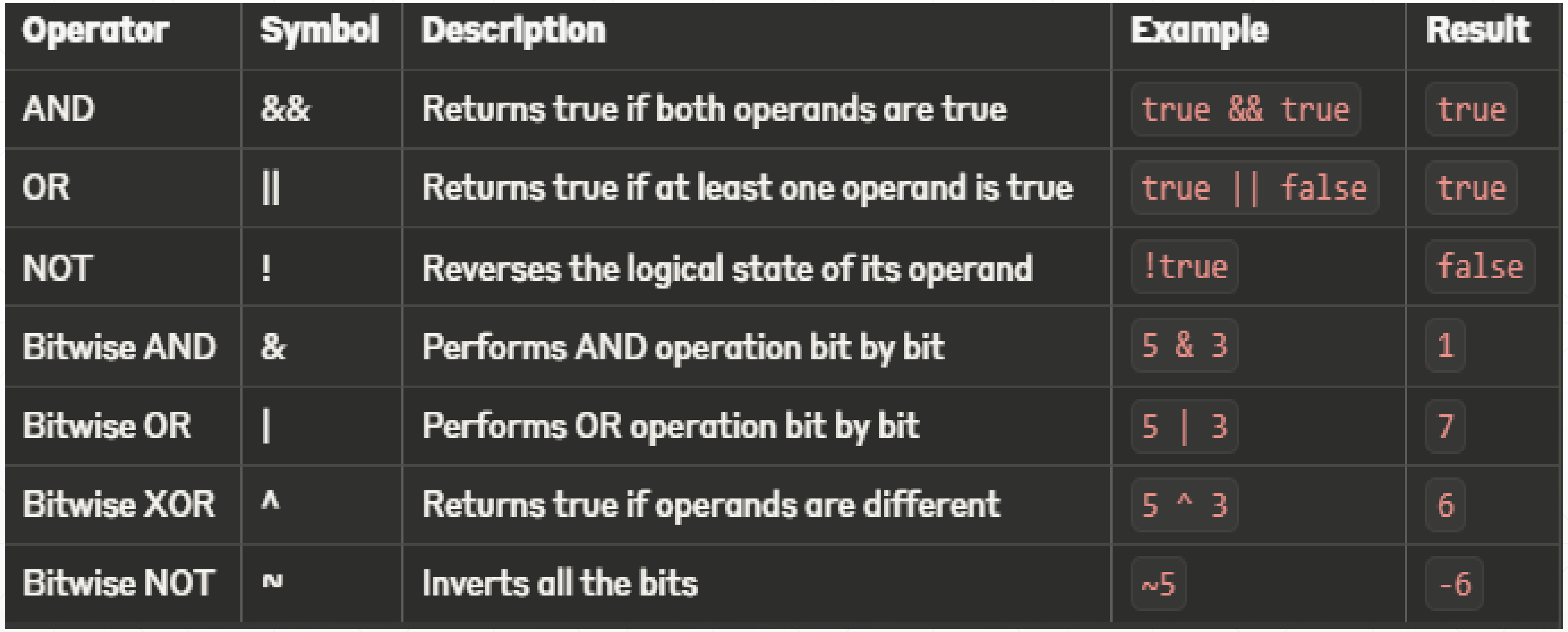
Relational (Comparison) Operators



Operator	Description	Example
`==`	Equal to	`a == b`
`!=`	Not equal to	`a != b`
`>`	Greater than	`a > b`
`<`	Less than	`a < b`
`>=`	Greater than or equal to	`a >= b`
`<=`	Less than or equal to	`a <= b`

```
int x = 5, y = 10;
System.out.println(x < y); // Output: true
```

Logical Operators



Operator	Symbol	Description	Example	Result
AND	&&	Returns true if both operands are true	`true && true`	`true`
OR	||	Returns true if at least one operand is true	`true || false`	`true`
NOT	!	Reverses the logical state of its operand	`!true`	`false`
Bitwise AND	&	Performs AND operation bit by bit	`5 & 3`	`1`
Bitwise OR	|	Performs OR operation bit by bit	`5 | 3`	`7`
Bitwise XOR	^	Returns true if operands are different	`5 ^ 3`	`6`
Bitwise NOT	~	Inverts all the bits	`~5`	`-6`

Java Comments

❑ **Single-line Comments**

```
// This is a single-line comment
int age = 25;
```

❑ **Multi-line Comments**

Use /* */ for longer explanations

```
/* This is a multi-line comment
   used to explain logic in detail */
int result = a + b;
```

User Input in Java

import java.util.Scanner;

Scanner input = new Scanner(System.in);

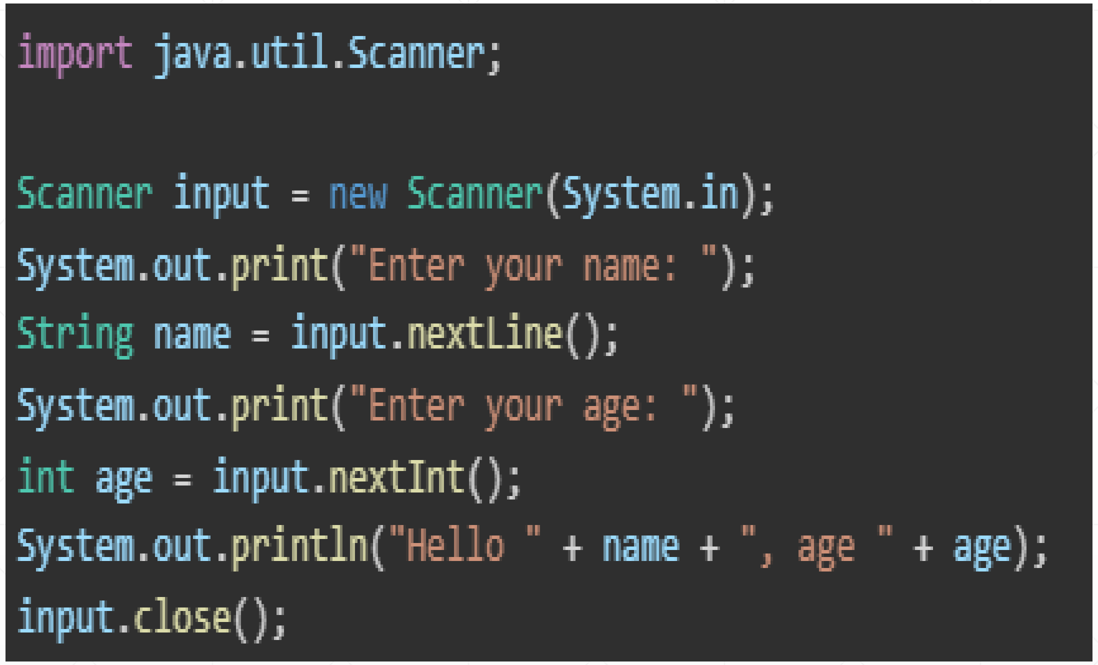
Use methods like:

nextLine() → for strings

nextInt() → for integers

nextDouble() → for decimals

Always close with input.close();



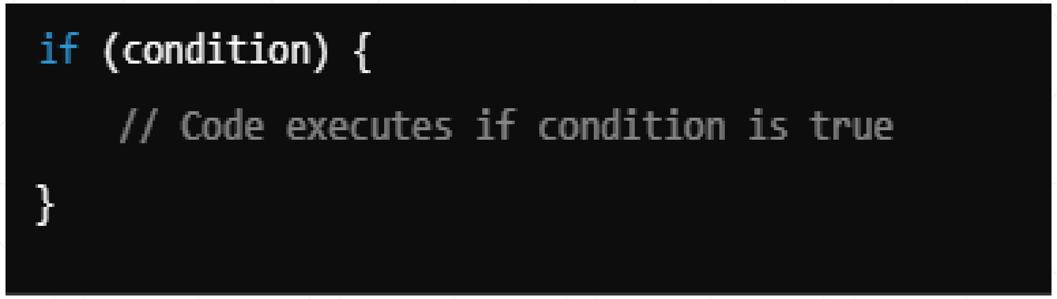
```java
import java.util.Scanner;

Scanner input = new Scanner(System.in);
System.out.print("Enter your name: ");
String name = input.nextLine();
System.out.print("Enter your age: ");
int age = input.nextInt();
System.out.println("Hello " + name + ", age " + age);
input.close();
```

Control Flow:

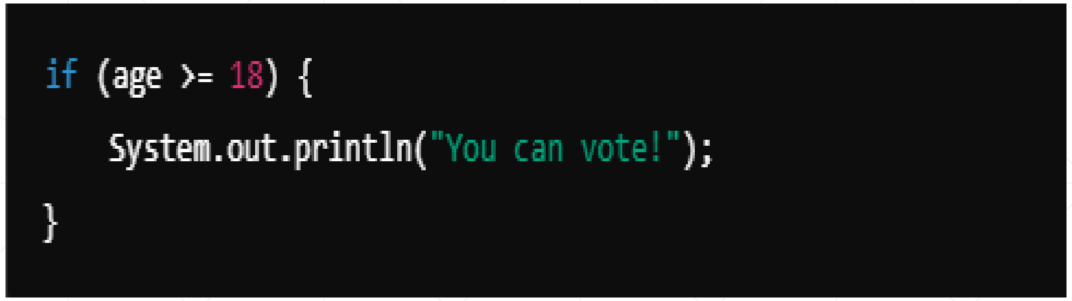
Control flow determines the **order of execution** of statements in Java.
Java provides different types of control flow statements:

- Conditional Statements (if, if-else, switch)
- Looping Statements (for, while, do-while)
- Jump Statements (break, continue, return)

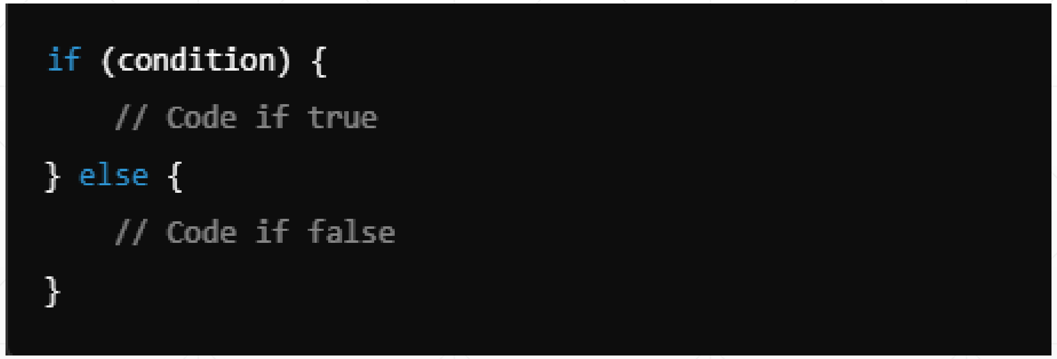
Conditional Statement



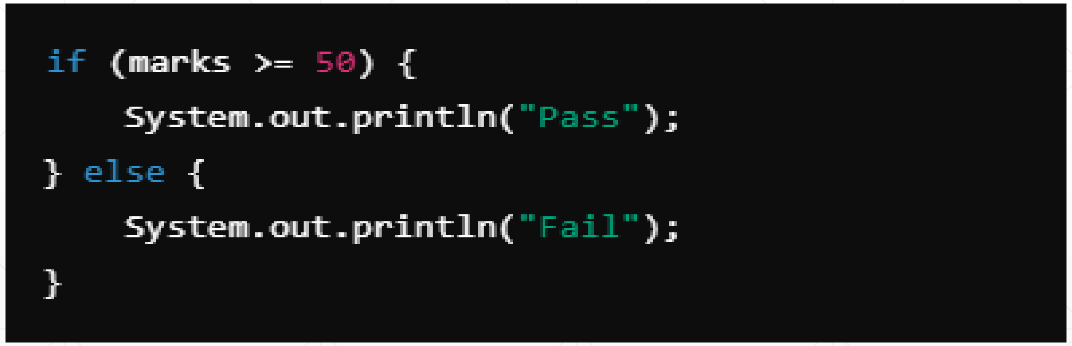
```java
if (condition) {
    // Code executes if condition is true
}
```



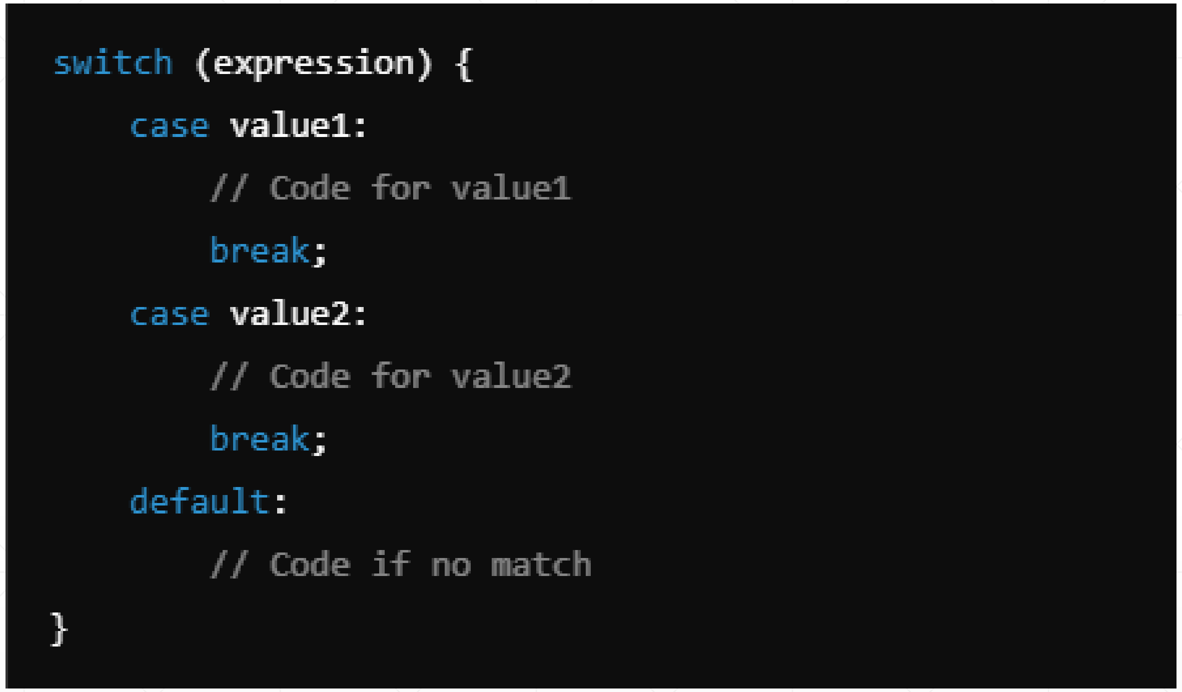
```java
if (age >= 18) {
    System.out.println("You can vote!");
}
```



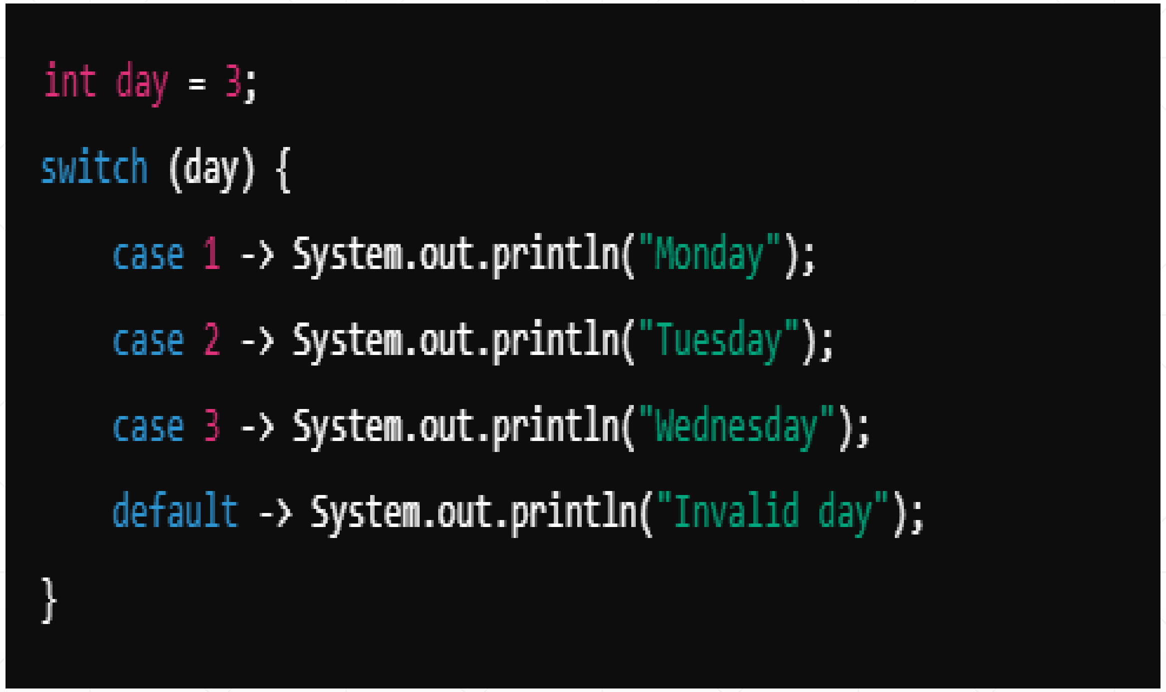
```java
if (condition) {
    // Code if true
} else {
    // Code if false
}
```



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

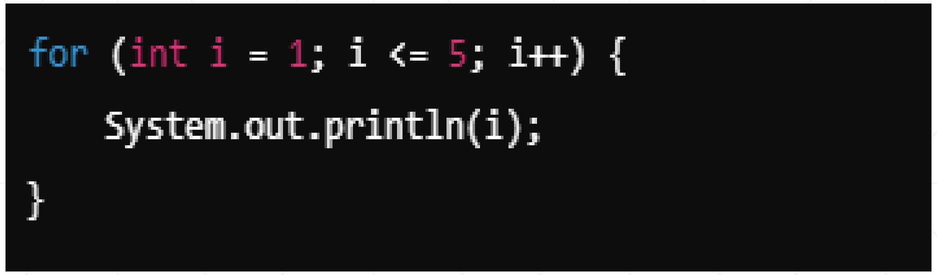


```java
switch (expression) {
    case value1:
        // Code for value1
        break;
    case value2:
        // Code for value2
        break;
    default:
        // Code if no match
}
```

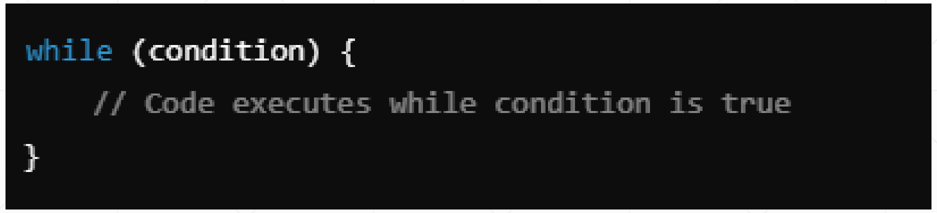


```java
int day = 3;
switch (day) {
    case 1 -> System.out.println("Monday");
    case 2 -> System.out.println("Tuesday");
    case 3 -> System.out.println("Wednesday");
    default -> System.out.println("Invalid day");
}
```

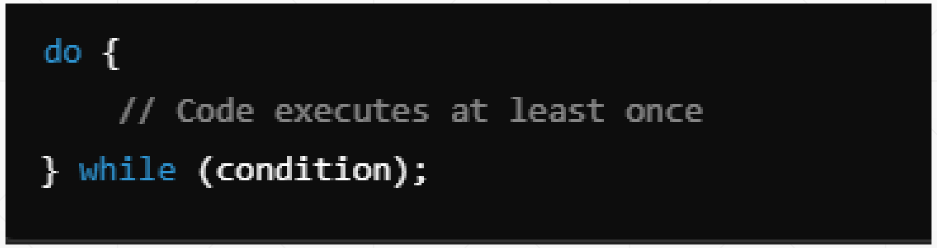
Looping Statement



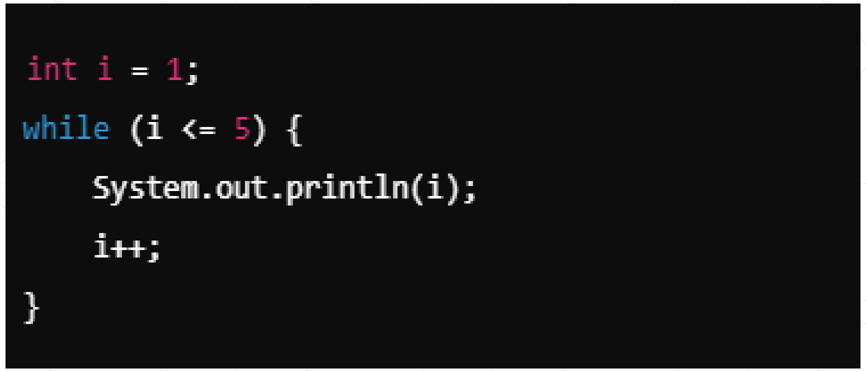
```java
for (int i = 1; i <= 5; i++) {
    System.out.println(i);
}
```



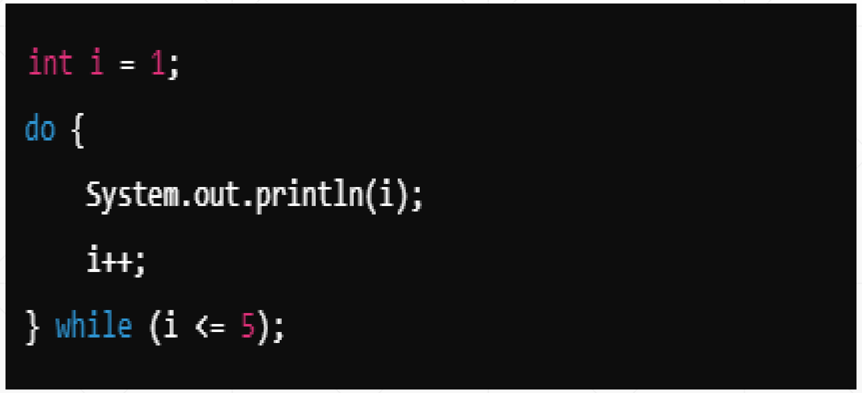
```java
while (condition) {
    // Code executes while condition is true
}
```



```java
do {
    // Code executes at least once
} while (condition);
```



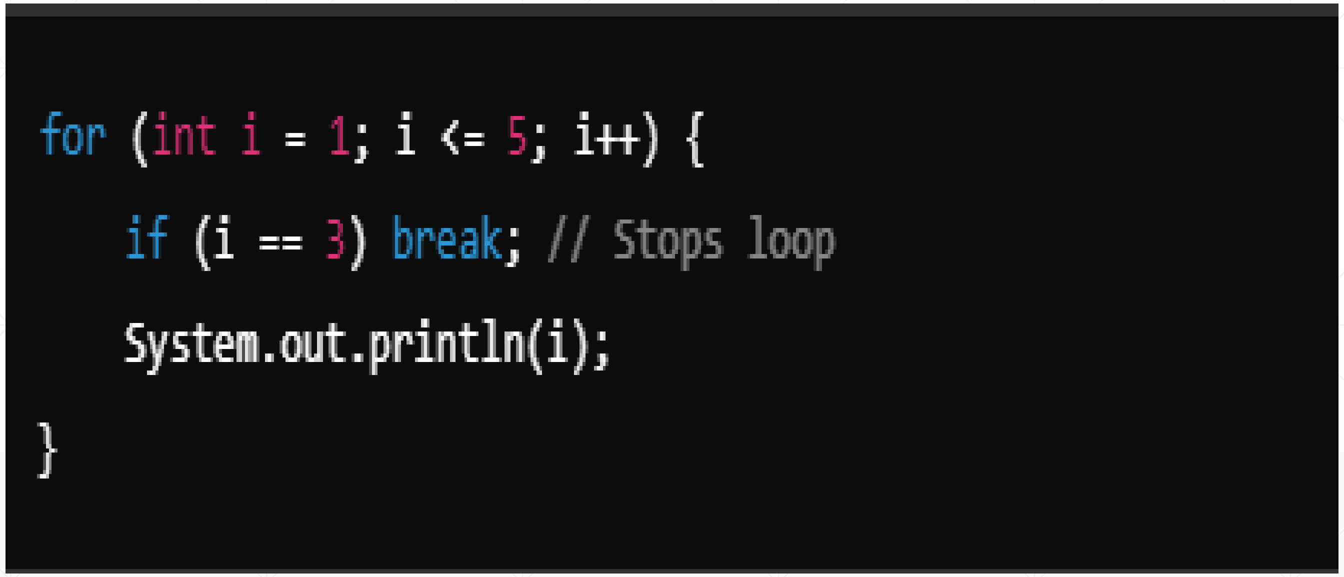
```java
int i = 1;
while (i <= 5) {
    System.out.println(i);
    i++;
}
```



```java
int i = 1;
do {
    System.out.println(i);
    i++;
} while (i <= 5);
```

Jump Statement

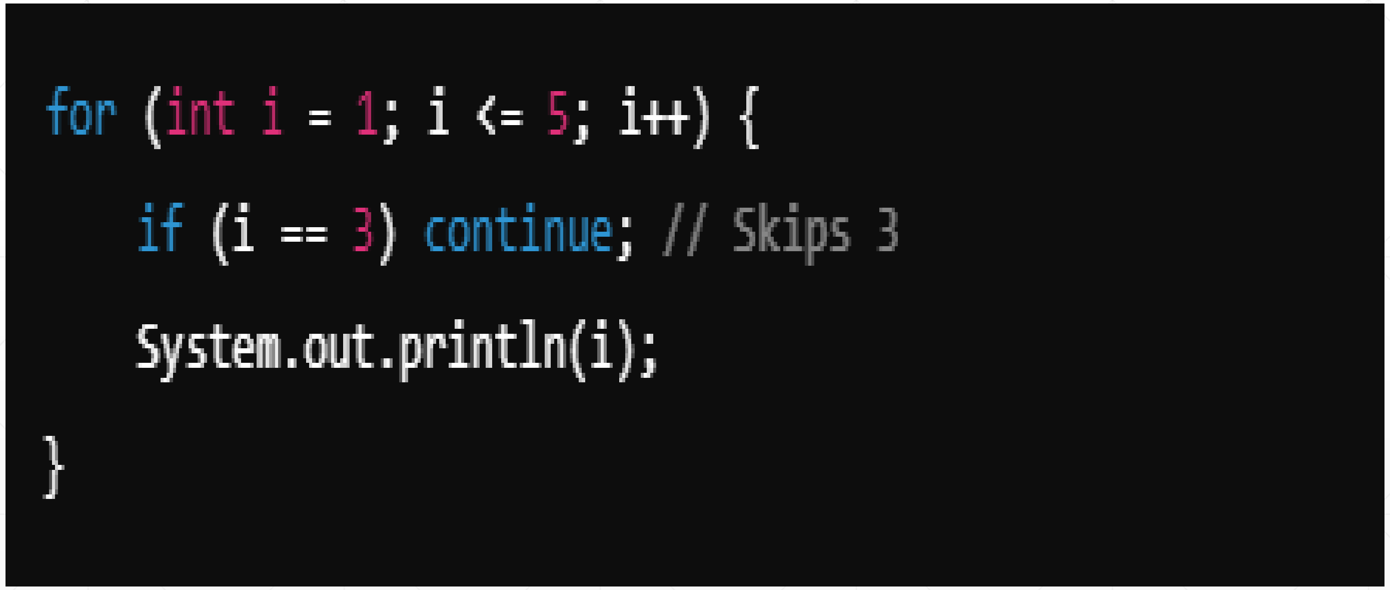
- Break statement (Exit Loop)



```
for (int i = 1; i <= 5; i++) {
    if (i == 3) break; // Stops loop
    System.out.println(i);
}
```

Jump Statement(Cont..)

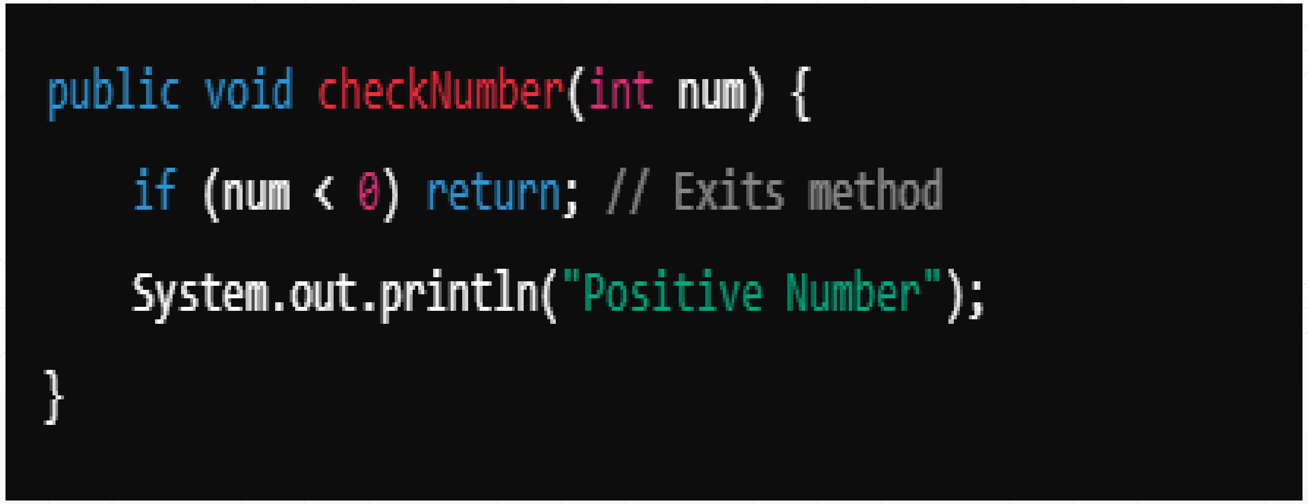
- Continue Statement(Skip Iteration)



```
for (int i = 1; i <= 5; i++) {
    if (i == 3) continue; // Skips 3
    System.out.println(i);
}
```

Jump Statement(cont..)

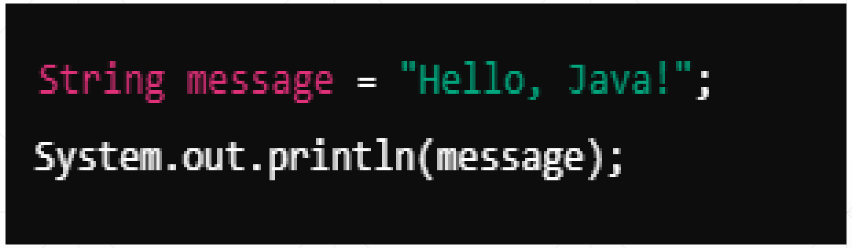
- Return Statement(Exit Method)



```
public void checkNumber(int num) {
    if (num < 0) return; // Exits method
    System.out.println("Positive Number");
}
```

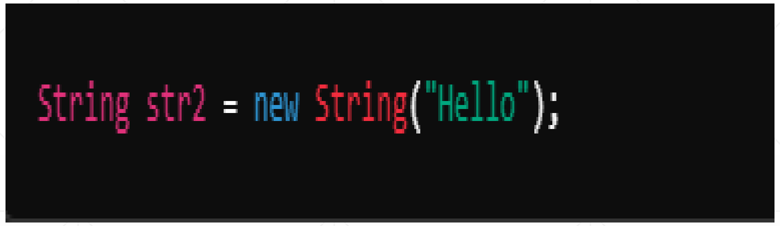
Strings in Java

•A **String** in Java is a sequence of characters.
•**Strings are immutable** (once created, they cannot be changed).
•Defined using the String class in Java



```
String message = "Hello, Java!";
System.out.println(message);
```

Stored in String Pool

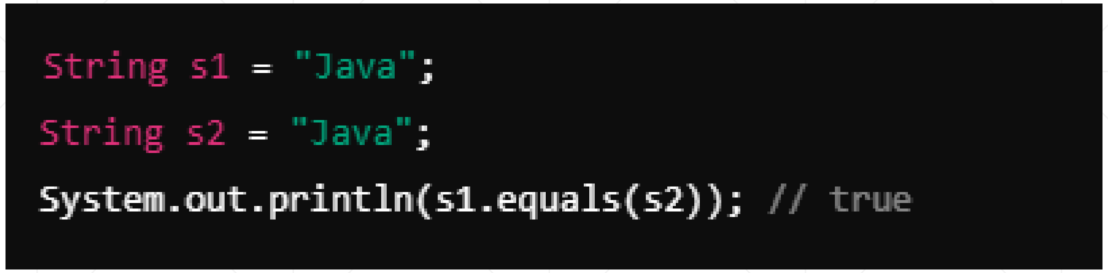


```
String str2 = new String("Hello");
```

Stored in Heap Memory

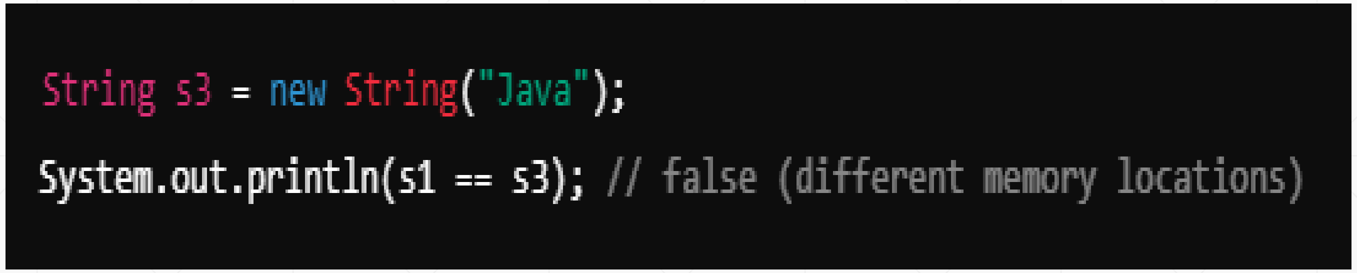
String Comparison

- Using equals() (Checks content)



```
String s1 = "Java";
String s2 = "Java";
System.out.println(s1.equals(s2)); // true
```

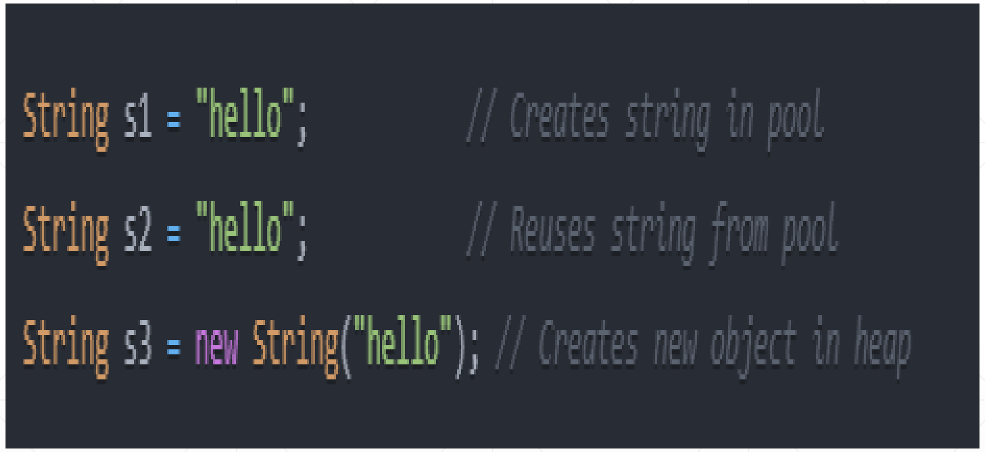
- Using == Operator (Checks reference)



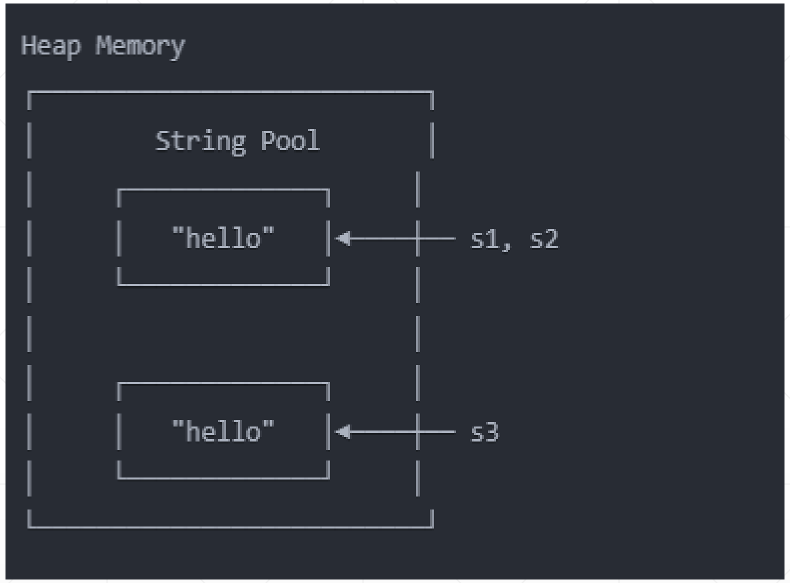
```
String s3 = new String("Java");
System.out.println(s1 == s3); // false (different memory locations)
```

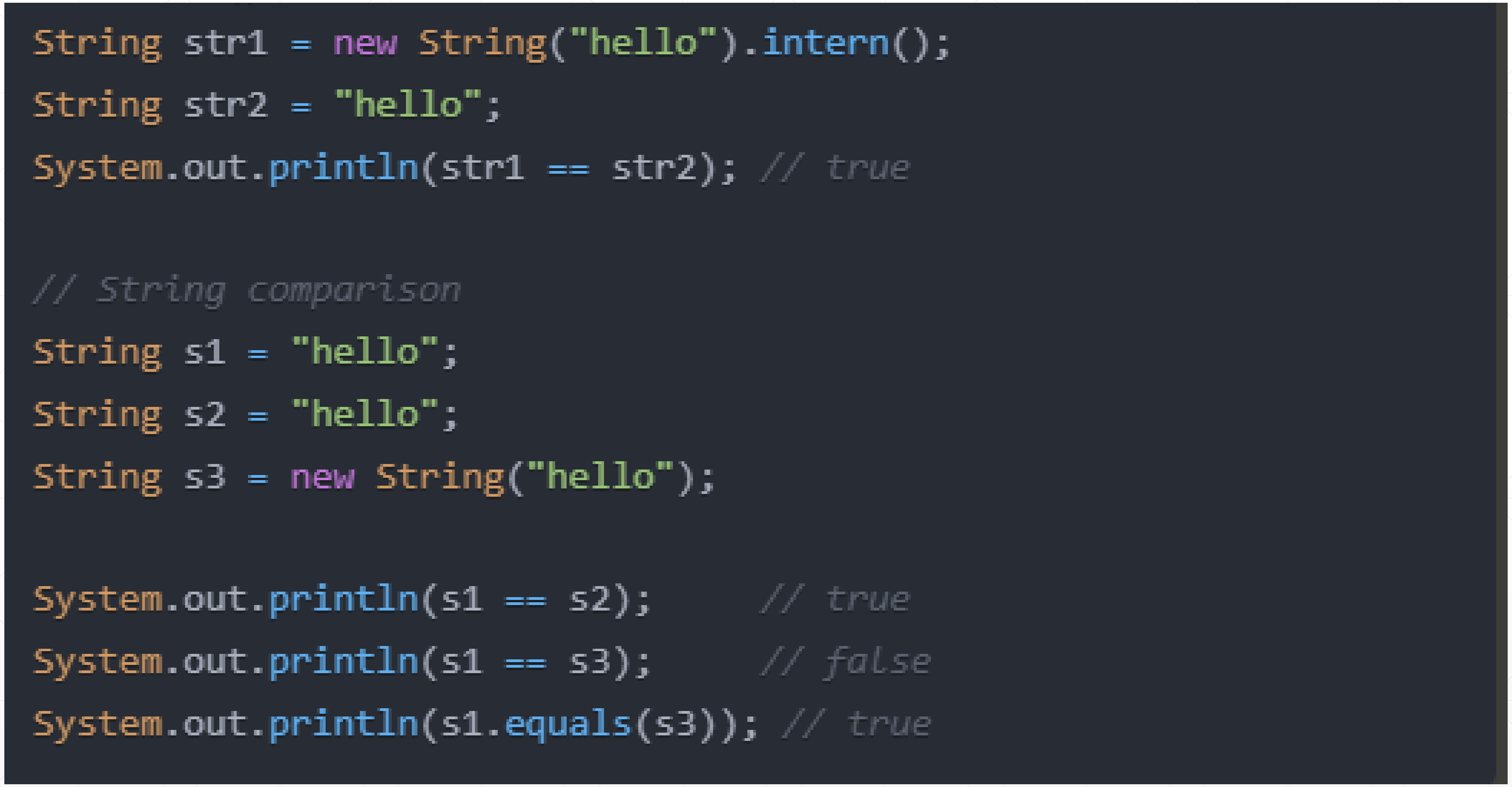
What is the String Pool?

- String Pool (also known as String Intern Pool) is a special memory area in Java's Heap memory to optimize memory usage.
- It stores unique string literals to save memory and improve performance.
- Implements the concept of "string interning"



```java
String s1 = "hello";           // Creates string in pool
String s2 = "hello";           // Reuses string from pool
String s3 = new String("hello"); // Creates new object in heap
```





```java
String str1 = new String("hello").intern();
String str2 = "hello";
System.out.println(str1 == str2); // true

// String comparison
String s1 = "hello";
String s2 = "hello";
String s3 = new String("hello");

System.out.println(s1 == s2);      // true
System.out.println(s1 == s3);      // false
System.out.println(s1.equals(s3)); // true
```

- **Problem Statement 1:**
- You are tasked with implementing a simple calculator that can perform basic arithmetic operations. The calculator should be able to perform addition, subtraction, multiplication, and division on two numbers provided by the user. After performing each operation, the program should allow the user to continue performing calculations until they choose to exit.

  Input:

  ```
  Enter first number: 7

  Enter second number: 3

  Enter operation (+, -, *, /): *

  Do you want to perform another calculation? (yes/no): no
  ```

  Output:

  ```
  Result: 21.0

  Thank you for using the calculator!
  ```

- **Problem Statement 2:**

You are given a string s consisting of lowercase English letters and a character c. Your task is to determine how many times the character c appears in the string s.

Input:
programming

g

Output: 2