## **Introduction to String in Java**

**✅ What is a String?**

* A **sequence of characters**.
* In Java, Strings are **objects** of class java.lang.String.
* Strings are **immutable** (once created, cannot be changed).

✅ String Declaration

String s1 = "Hello"; // String literal

String s2 = new String("World"); // Using new keyword

**🔒 What is Immutability?**

**✅ Definition:**

**An object is immutable** if its **state (i.e., its fields or data) cannot be changed** after it is created.

**✅ In Java:**

* You can still assign a new object to a variable, but the original object’s **contents don't change**.
* **Strings are immutable** in Java

🔍 Example:

String s = "Hello";

s.concat(" World");

System.out.println(s); // Output: Hello  
  
- You might expect "Hello World" — but you get "Hello".

- Why? Because concat() creates a **new string**, and doesn't modify the original.

To see the new result, you'd do:

String s2 = s.concat(" World");

System.out.println(s2); // Output: Hello World

**🧠 Why are Strings Immutable in Java?**

Here are **real and practical reasons** behind this design:

**1. 🔐 Security**

* Strings are used in **sensitive operations** like:
  + File paths, network connections
  + Class loading (Class.forName("com.example.MyClass"))
* If strings were mutable, someone could change "com.example.MyClass" after it’s used — creating major **security issues**.

**2. 💾 String Pooling (SCP)**

* Java maintains a **String pool** to save memory.
* When you write String s = "hello";, it checks if "hello" already exists in the pool.
* If strings were mutable, changing one reference would accidentally affect others — breaking pooling.

**✅ Example:**

String a = "Java";

String b = "Java";

System.out.println(a == b); // true — same object in SCP

If one of them could be changed, both would be affected.

**3. 💻 Thread Safety**

* Immutable objects are naturally **thread-safe**.
* No need for synchronization, locking, or fear of concurrent modification.

**📦 Internally: How is Immutability Achieved?**

public final class String {

private final char[] value; // final array

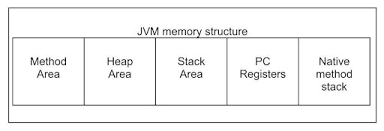
...

}

* String class is final → it **cannot be subclassed**.
* Internal char[] is private and final.
* All modifying methods (like toUpperCase(), concat(), replace()) return **new Strings**.

**☕ What is the JVM Memory Structure?**

The **Java Virtual Machine (JVM)** divides memory into different areas to **efficiently load, run, and manage Java programs**.



**1. 🧠 Method Area (a.k.a. MetaSpace)**

* Stores class metadata:
  + Class name, superclass name
  + Method names and return types
  + Static variables
* One per JVM
* Shared across all threads

📌 In Java 8+, this is called **MetaSpace** and is stored in **native memory** (outside heap).

class A {

static int count = 0;

}  
Class A and static variable count are stored in **Method Area**

**2. 🟩 Heap**

* **Biggest memory area**
* Stores:
  + All **Objects**
  + All **Instance variables**
  + **String Pool** (part of heap)
* Shared by all threads
* Managed by **Garbage Collector**

String s = new String("Hello");

Student student = new Student();  
  
→ s, "Hello", and student are created in **heap**.  
  
**3. 🟦 JVM Stack (Thread Stack)**

* Each **thread** has its own **stack**
* Stores:
  + Method calls (stack frames)
  + Local variables
  + Parameters
* Memory is allocated when a method is called and deallocated when it returns

void add(int a, int b) {

int sum = a + b;

}  
a, b, and sum live in the **stack** (inside a stack frame for add())  
  
**4. 🔁 Program Counter (PC) Register**

* Each thread has its own PC register
* Stores the **address of the current instruction** being executed
* Helps the CPU know **what to execute next**

**5. 🧩 Native Method Stack**

* Used to manage **native (non-Java) code** via JNI (Java Native Interface)
* Stores data for C/C++ methods called from Java

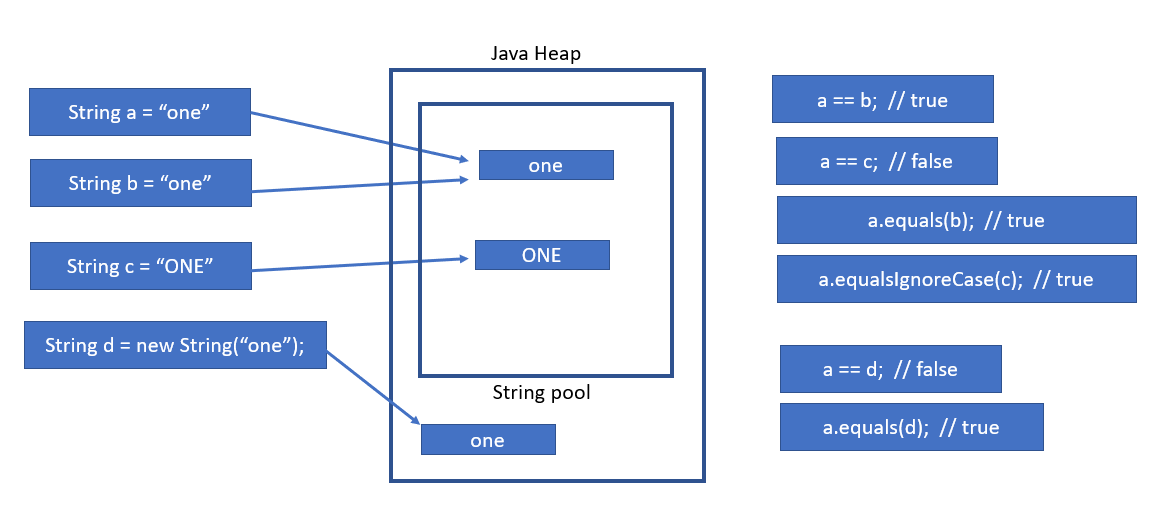
**✅ Why does Java use a String Pool?**

Because Strings are used a lot (e.g., keys in HashMaps, user input, config names), and they are immutable, Java can safely reuse them.

Instead of creating a new object every time you write "Java", it stores only one copy of "Java" in the String Pool and reuses it.

📌 Important Rules:

| **Statement** | **Behavior** |
| --- | --- |
| String s1 = "Hello"; | Stored in the String Pool |
| String s2 = "Hello"; | Reuses same object from pool |
| String s3 = new String("Hello"); | Creates new object in **heap**, **not reused** |
| s3.intern() | Adds "Hello" to pool (if not already) and returns pooled reference |

  
  
🔍 Example with Explanation  
String a = "Java";

String b = "Java";

String c = new String("Java");

System.out.println(a == b); // true → same object from pool

System.out.println(a == c); // false → c is a different object

System.out.println(a == c.intern()); // true → intern returns pooled reference  
  
**🛠️ What is intern()?**

The intern() method tells JVM:

"Please give me the reference from the **pool**, or create and store it if it doesn't exist."

**Common String Methods**

| Method | Description | Example |
| --- | --- | --- |
| length() | Returns length | "Hello".length() → 5 |
| charAt(i) | Char at index | "Java".charAt(2) → 'v' |
| substring(start, end) | Part of string | "Java".substring(1, 3) → "av" |
| equals() / equalsIgnoreCase() | Compare strings | "Hi".equals("hi") → false |
| indexOf(ch) | Index of char | "hello".indexOf('l') → 2 |
| toLowerCase() / toUpperCase() | Case conversion | "Java".toUpperCase() → "JAVA" |
| trim() | Removes spaces | " hi ".trim() → "hi" |
| replace(a, b) | Replace chars | "banana".replace('a', 'o') → "bonono" |
| split() | Split string | "a,b,c".split(",") → [a, b, c] |
| contains() | Check substring | "hello".contains("ell") → true |

**🔷 Why StringBuilder and StringBuffer Came Into Existence**

**In Java:**

* The String class is immutable, meaning once a String object is created, it cannot be changed.
* Any operation like concatenation or replacement on a String creates a new object in memory.
* This leads to performance and memory issues, especially inside loops or repeated string manipulations.

**✅ To solve this:**

**Java introduced:**

* StringBuffer (Java 1.0) – A thread-safe mutable string class.
* StringBuilder (Java 5) – A faster, non-thread-safe alternative when synchronization is not required.

**🔸 StringBuffer – Thread-Safe Mutable Strings**

**📌 Description:**

* Can change string contents without creating new objects.
* All methods are synchronized → safe for multithreaded environments.
* Slightly slower due to synchronization overhead.

**🧠 Rules:**

* Use when multiple threads are modifying the same string object.
* Use in scenarios requiring string manipulation with concurrency.

**✅ Where to Use:**

* In multi-threaded applications (e.g., server logs, thread-safe logging).
* When you want safe string modifications across threads.

**🔄 Common Methods:**

* append(), insert(), replace(), delete(), reverse(), toString()

**🧾 Real-World Examples of StringBuffer**

1. Multi-threaded Logger

StringBuffer log = new StringBuffer();

log.append("Thread1: started\n");

log.append("Thread2: started\n");

Logging from multiple threads using a single shared buffer.

1. Web Server Log Buffering
   * Accumulating logs from different threads handling HTTP requests.
2. Live Chat Application
   * Multiple threads appending messages to a shared message history.

**🔸 StringBuilder – Fast Mutable Strings (Non-Synchronized)**

**📌 Description:**

* Similar to StringBuffer but not thread-safe.
* Faster due to lack of synchronization.

**🧠 Rules:**

* Use when thread safety is not needed.
* Ideal for single-threaded applications doing lots of string modifications.

**✅ Where to Use:**

* Inside loops for building strings.
* When building SQL queries, file content, or dynamic HTML/CSS in Java apps.

**🔄 Common Methods:**

Same as StringBuffer: append(), insert(), replace(), delete(), reverse(), toString()

**🧾 Real-World Examples of StringBuilder**

1. **Generating a CSV File**

**StringBuilder csv = new StringBuilder();**

**csv.append("Name, Age, City\n");**

**csv.append("Alice, 30, NY\n");**

1. **Building SQL Query in JDBC**

**StringBuilder query = new StringBuilder("SELECT \* FROM users WHERE ");**

**query.append("status = 'active'");**

1. **Processing JSON/XML in Memory**
   * Dynamically create XML/JSON strings before sending over API or saving to file.

**🔁 String vs StringBuilder vs StringBuffer – Summary Table**

| Feature | String | StringBuilder | StringBuffer |
| --- | --- | --- | --- |
| Mutable | ❌ No | ✅ Yes | ✅ Yes |
| Thread-Safe | ✅ (Immutable) | ❌ No | ✅ Yes |
| Performance | ❌ Slow (new object each time) | ✅ Fast | ⚠️ Slower (sync) |
| Use In | Constants, keys, few changes | Loops, single-threaded apps | Multi-threaded apps |
| Introduced In | Java 1.0 | Java 5 | Java 1.0 |

**📚 When to Use What**

| **Situation** | **Use** |
| --- | --- |
| **Basic fixed strings** | **String** |
| **Repeated string concatenation (e.g., loops)** | **StringBuilder** |
| **Multi-threaded string manipulations** | **StringBuffer** |

## Quiz on String

1] **Which of the following creates a String object?**  
A. String s = new String("hello");  
B. String s = "hello";  
C. char[] c = {'h','e','l','l','o'}; String s = new String(c);  
D. All of the above

**Answer:** D

2] **What will System.out.println("abc" + 1 + 2); print?**  
A. abc3  
B. abc12  
C. 3abc  
D. Compilation error

**Answer:** B

3] **Given String s = "Java";, what does s.charAt(2) return?**  
A. 'v'  
B. 'a'  
C. 2  
D. Compilation error

**Answer:** A

4] **Which method compares two strings for content equality?**  
A. ==  
B. equals()  
C. compareTo()  
D. contentEquals()

**Answer:** B

5] **What’s the result of "hello".substring(1, 4)?**  
A. "hel"  
B. "ell"  
C. "ello"  
D. "lo"

**Answer:** B

6] **How do you convert the string "123" to an integer?**  
A. Integer.parseInt("123")  
B. new Integer("123")  
C. String.toInt("123")  
D. (int)"123"

**Answer:** A

7] **Which of these is true about Java String objects?**  
A. They are mutable.  
B. They are stored on the stack.  
C. They are immutable.  
D. Each call to any method always creates a new literal.

**Answer:** C

8] **What does "one,two,three".split(",") return?**  
A. A single String "onetwothree"  
B. A List<String>  
C. A String[] of {"one","two","three"}  
D. A char[]

**Answer:** C

9] **Which is the correct way to check if a string starts with “pre”?**  
A. s.startswith("pre")  
B. s.startsWith("pre")  
C. s.starts\_with("pre")  
D. s.prefix("pre")

**Answer:** B

10] **What does String.format("%s = %d", "age", 30) produce?**  
A. "age = 30"  
B. "30 = age"  
C. "age = %d"  
D. Runtime exception

**Answer:** A

## Quiz on StringBuilder and StringBuffer

**1. Which of the following is true about StringBuilder?**

A. It is thread-safe  
B. It is immutable  
C. It is mutable and not synchronized  
D. It was introduced in Java 1.2

✅ **Answer:** C. It is mutable and not synchronized

**2. What is the output of the following code?**

StringBuilder sb = new StringBuilder("Hello");

sb.append(" World");

System.out.println(sb);

A. Hello  
B. HelloWorld  
C. Hello World  
D. Error

✅ **Answer:** C. Hello World

**3. Which of the following is best for use in a single-threaded environment?**

A. String  
B. StringBuffer  
C. StringBuilder  
D. None of the above

✅ **Answer:** C. StringBuilder

**4. What is the output of the code?**

StringBuffer sb = new StringBuffer("Java");

sb.reverse();

System.out.println(sb);

A. Java  
B. avaJ  
C. Error  
D. No output

✅ **Answer:** B. avaJ

**5. Which class is synchronized?**

A. StringBuilder  
B. StringBuffer  
C. Both  
D. None

✅ **Answer:** B. StringBuffer

**6. Which method is used to remove characters from a StringBuilder?**

A. erase()  
B. delete()  
C. remove()  
D. eliminate()

✅ **Answer:** B. delete()

**7. What is the initial capacity of a StringBuilder created using new StringBuilder()?**

A. 0  
B. 8  
C. 16  
D. 32

✅ **Answer:** C. 16

**8. Which method converts a StringBuilder or StringBuffer to a String?**

A. convert()  
B. toText()  
C. toString()  
D. toCharArray()

✅ **Answer:** C. toString()

**9. What is the result of the following?**

StringBuilder sb = new StringBuilder("abc");

sb.insert(1, "123");

System.out.println(sb);

A. abc123  
B. a123bc  
C. ab123c  
D. 123abc

✅ **Answer:** B. a123bc

**10. Why was StringBuilder introduced when StringBuffer already existed?**

A. To replace String  
B. To support immutability  
C. To provide faster performance in single-threaded scenarios  
D. For backward compatibility

✅ **Answer:** C. To provide faster performance in single-threaded scenarios

# ☕ Introduction to OOPs in Java

**🔰 What is OOP?**

**OOP (Object-Oriented Programming) is a programming paradigm where everything is represented as objects. These objects interact with each other to design applications.**

🎯 Goals of OOP:

* Organize code using classes and objects
* Promote reusability, scalability, and security
* Make code more modular and maintainable

**🔸 Java as an OOP Language**

Java is a **pure object-oriented language** (with some exceptions like primitive data types).  
In Java:

* Everything revolves around **classes** and **objects**.
* Core features of OOP are built directly into the language.

**💡 OOPs Concepts in Java**

Java supports the following **4 main pillars** of OOP:

| **Pillar** | **Description** |
| --- | --- |
| **1. Encapsulation** | Wrapping data & methods into one unit (class) |
| **2. Inheritance** | Acquiring properties from parent class |
| **3. Polymorphism** | One action, many forms |
| **4. Abstraction** | Hiding internal details and showing only functionality |

**1]Encapsulation – Data Hiding**

**Definition:** Keeping fields (private) and exposing methods (public) to access them.

class Student {

private int age; // hidden data

public void setAge(int a) {

if (a > 0) age = a;

}

public int getAge() {

return age;

}

}

✅ **Benefits:**

* Protects data from outside misuse
* Easy to change internal logic without affecting external code

**2] Inheritance – Code Reuse**

**Definition:** One class inherits properties of another using extends.

class Animal {

void sound() {

System.out.println("Animal makes sound");

}

}

class Dog extends Animal {

void bark() {

System.out.println("Dog barks");

}

}

Dog d = new Dog();

d.sound(); // Inherited from Animal

d.bark(); // Dog's own method

**✅ Benefits:**

* **Promotes code reuse**
* **Helps in hierarchical classification**

**3] Polymorphism – One Name, Many Behaviors**

Types of Polymorphism:

* Compile-time (Method Overloading)
* Run-time (Method Overriding)

**Method Overloading (Compile-time)**

class Calculator {

int add(int a, int b) { return a + b; }

double add(double a, double b) { return a + b; }

}

**b. Method Overriding (Run-time)**

class Parent {

void greet() { System.out.println("Hello from Parent"); }

}

class Child extends Parent {

void greet() { System.out.println("Hello from Child"); }

}

Parent obj = new Child();

obj.greet(); // Hello from Child

✅ **Benefits:**

* Makes code **dynamic and flexible**
* Implements **runtime behavior**

4] **Abstraction – Hiding Implementation Details**

**Definition:** Showing only essential features and hiding internal code using:

* abstract class
* interface

abstract class Vehicle {

abstract void drive();

void fuel() {

System.out.println("Fueling...");

}

}

class Car extends Vehicle {

void drive() {

System.out.println("Car is driving");

}

}

**📦 Other OOP Features in Java**

| **Feature** | **Description** |
| --- | --- |
| **Class** | Blueprint of object |
| **Object** | Instance of class |
| **Constructor** | Initializes object |
| **this keyword** | Refers to current object |
| **super keyword** | Refers to parent class |
| **Interface** | Pure abstraction |

🧪 Example: Full OOP Example

abstract class Animal {

abstract void makeSound();

}

class Dog extends Animal {

void makeSound() {

System.out.println("Bark!");

}

}

public class Test {

public static void main(String[] args) {

Animal a = new Dog(); // Polymorphism

a.makeSound(); // Bark!

}

}

📝 Summary Table

| **Concept** | **Keyword/Example** | **Description** |
| --- | --- | --- |
| Encapsulation | private, get/set | Data hiding |
| Inheritance | extends | Reuse of code |
| Polymorphism | @Override, overloading | Many forms |
| Abstraction | abstract, interface | Hide details |

**✅ Guess the Output**

✅ Example 1: Immutability Check

public class Main {

public static void main(String[] args) {

String s = "Hello";

s.concat(" World");

System.out.println(s);

}

}

✅ Example 2: String Equality

public class Main {

public static void main(String[] args) {

String s1 = "Java";

String s2 = new String("Java");

System.out.println(s1 == s2);

System.out.println(s1.equals(s2));

}

}

✅ Example 3: String Tricky Index

public class Main {

public static void main(String[] args) {

String str = "Programming";

System.out.println(str.substring(3, 6));

}

}

✅ Example 4: Append and Capacity

public class Main {

public static void main(String[] args) {

StringBuilder sb = new StringBuilder();

sb.append("Java");

sb.append(" Rocks");

System.out.println(sb);

}

}

✅ Example 5: Reversal

public class Main {

public static void main(String[] args) {

StringBuilder sb = new StringBuilder("Hello");

sb.reverse();

System.out.println(sb);

}

}

✅ Example 6: Insert and Delete

public class Main {

public static void main(String[] args) {

StringBuilder sb = new StringBuilder("World");

sb.insert(0, "Hello ");

sb.delete(5, 7);

System.out.println(sb);

}

}

✅ Example 7: Capacity vs. Length  
public class Main {

public static void main(String[] args) {

StringBuilder sb = new StringBuilder();

sb.append("Hi");

System.out.println("Length: " + sb.length());

System.out.println("Capacity: " + sb.capacity());

}

}

## **Assignments for this week:**

🧾 **Assignment 1: Email Validator and Extractor (String)**

**Objective:** Practice String methods like contains(), indexOf(), substring(), split(), endsWith().

**Problem:** Write a Java program that:

* Accepts a comma-separated list of email addresses.
* Validates each one:
  + Must contain @ and .
  + Must end with .com or .org
* Extracts and prints the username and domain separately.

**Example Input:**

[john.doe@gmail.com,invalidEmail,test@company.org](mailto:john.doe@gmail.com,invalidEmail,test@company.org)

**Example Output:**

Valid: john.doe@gmail.com | Username: john.doe | Domain: gmail.com

Invalid: invalidEmail

Valid: test@company.org | Username: test | Domain: company.org

🧾 **Assignment 2: Custom Text Formatter (StringBuilder)**

**Objective:** Practice StringBuilder operations like append(), insert(), delete(), replace().

**Problem:** Create a simple CLI-based formatter:

* Takes a sentence as input.
* Formats it by:
  + Replacing all spaces with underscores.
  + Inserting the word "START\_" at the beginning.
  + Appending the word "\_END" at the end.
  + Removing all punctuation marks.

**Example Input:**

Hello, this is a test!

**Example Output:**

START\_Hello\_this\_is\_a\_test\_END

🧾 **Assignment 3: Password Strength Checker (String & StringBuilder)**

**Objective:** Use both String and StringBuilder to analyze and transform input.

**Problem:** Create a password checker that:

* Validates password based on:
  + At least 8 characters
  + At least one digit
  + At least one uppercase letter
  + At least one special character (!@#$%^&\*)
* If valid, print "Strong Password"
* If not, suggest a stronger password by:
  + Appending missing character types using StringBuilder

**Example Input:**

welcome1  
  
**Example Output:**

Weak Password. Suggested: welcome1A!  
  
**🧾 Assignment 4: Palindrome Sentence Checker (StringBuilder)**

**Objective:** Use StringBuilder.reverse() and string cleaning.

**Problem:** Write a Java program that checks if a given sentence is a **palindrome**, ignoring spaces, punctuation, and case.

**Example Input:**

A man, a plan, a canal, Panama

**Example Output:**

It's a Palindrome

🧾 **Assignment 5: SMS Abbreviation Expander (String)**

**Objective:** Replace short forms with full words using replace().

**Problem:** Create a dictionary of common SMS abbreviations:

* u → you
* r → are
* idk → I don't know
* btw → by the way

Take a message and replace all abbreviations with full words.

**Example Input:**

idk what u r doing btw

**Example Output:**  
I don't know what you are doing by the way