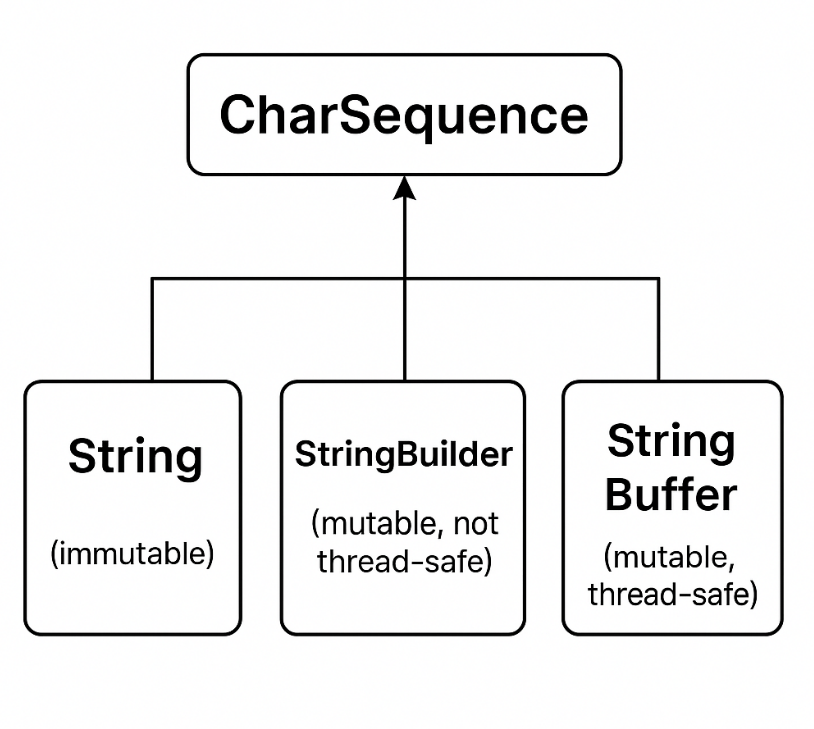
# What is CharSequence?

CharSequence is an **interface** in Java that represents a readable sequence of characters. It is part of the java.lang package.



# ✅ Key Point:

Any class that implements CharSequence can be treated as a sequence of characters — including String, StringBuilder, and StringBuffer.

# Java String, StringBuilder, and StringBuffer - Summary Notes

## 1. String

• Immutable: Once created, the value of a String cannot be changed.  
• Thread-safe: Not applicable, since it's immutable.  
• Performance: Slower when performing multiple modifications (e.g., concatenation).

Example:

public class StringExample {  
 public static void main(String[] args) {  
 String str = "Hello";  
 str = str + " World"; // Creates a new String object  
 System.out.println(str); // Output: Hello World  
 }  
}

## 2. StringBuilder

• Mutable: You can change the contents without creating new objects.  
• Not thread-safe: Should not be used in multi-threaded environments.  
• Performance: Faster than String and StringBuffer for single-threaded operations.

Example:

public class StringBuilderExample {  
 public static void main(String[] args) {  
 StringBuilder sb = new StringBuilder("Hello");  
 sb.append(" World");  
 System.out.println(sb.toString()); // Output: Hello World  
 }  
}

## 3. StringBuffer

• Mutable: Like StringBuilder, but with synchronization.  
• Thread-safe: Safe to use in multi-threaded environments.  
• Performance: Slower than StringBuilder due to synchronization overhead.

Example:

public class StringBufferExample {  
 public static void main(String[] args) {  
 StringBuffer sb = new StringBuffer("Hello");  
 sb.append(" World");  
 System.out.println(sb.toString()); // Output: Hello World  
 }  
}

## Summary Table

| Feature | String | StringBuilder | StringBuffer |  
|----------------|--------------|----------------|--------------|  
| Mutability | Immutable | Mutable | Mutable |  
| Thread Safety | Yes | No | Yes |  
| Performance | Slow | Fast | Slower |  
| Use Case | Fixed text | Fast changes | Safe changes |

## Real-world Analogies

### String – Sealed Letter

Imagine you write a letter, seal it in an envelope, and send it. Once it's sealed, you cannot change the contents. If you want to say something new, you have to write a new letter.  
• Immutable: Just like the sealed letter, once a String is created, it cannot be changed.

### StringBuilder – Whiteboard

Think of a whiteboard where you can write, erase, and rewrite easily. It's perfect for quick changes and drafting ideas.  
• Mutable: You can change the content without creating a new object.  
• Not thread-safe: If multiple people try to write at the same time, it might get messy.

### StringBuffer – Shared Whiteboard with Rules

Imagine a whiteboard in a meeting room where multiple people can write, but only one person at a time. There's a rule: wait your turn.  
• Mutable and thread-safe: Changes are allowed, but synchronized to avoid conflicts.

## Analogy Summary Table

| Concept | Analogy | Key Trait |  
|----------------|--------------------------------|-------------------|  
| String | Sealed letter | Immutable |  
| StringBuilder | Personal whiteboard | Fast, not thread-safe |  
| StringBuffer | Shared whiteboard with rules | Safe, but slower |

# String Pool in Java

In Java, the String Pool is a special memory region where string literals are stored. When a string literal is created, Java checks the pool to see if an identical string already exists. If it does, the reference to the existing string is returned instead of creating a new object. This helps in saving memory and improving performance.

## See the source imagess

## Real-Life Analogy

Imagine a library where multiple students need the same textbook. Instead of each student buying their own copy, they all use the same copy available in the library. This avoids duplication and saves resources. Similarly, the String Pool avoids creating multiple identical string objects.

## Code Example

public class StringPoolExample {  
 public static void main(String[] args) {  
 String s1 = "Hello";  
 String s2 = "Hello";  
 String s3 = new String("Hello");  
  
 System.out.println(s1 == s2); // true, both refer to the same object in the pool  
 System.out.println(s1 == s3); // false, s3 refers to a new object  
 }  
}

Java String Methods

# length()

Purpose: Returns the number of characters in the string.

Use Case: Validating input length (e.g., password must be at least 8 characters).

Example:

String password = "secret123";  
if (password.length() < 8) {  
 System.out.println("Password too short!");  
}

# charAt(int index)

Purpose: Returns the character at the specified index.

Use Case: Parsing characters (e.g., checking if the first character is uppercase).

Example:

String name = "Alice";  
if (Character.isUpperCase(name.charAt(0))) {  
 System.out.println("Name starts with a capital letter.");  
}

# substring(int beginIndex), substring(int beginIndex, int endIndex)

Purpose: Extracts a portion of the string.

Use Case: Extracting domain from an email.

Example:

String email = "user@example.com";  
String domain = email.substring(email.indexOf("@") + 1);  
System.out.println(domain); // example.com

# equals(String anotherString)

Purpose: Compares two strings for exact match (case-sensitive).

Use Case: Checking login credentials.

Example:

if (inputPassword.equals(storedPassword)) {  
 System.out.println("Login successful.");  
}

# equalsIgnoreCase(String anotherString)

Purpose: Compares two strings ignoring case.

Use Case: Comparing usernames or emails.

Example:

if (inputEmail.equalsIgnoreCase(storedEmail)) {  
 System.out.println("Email matched.");  
}

# toLowerCase(), toUpperCase()

Purpose: Converts all characters to lower or upper case.

Use Case: Case-insensitive search or standardizing data.

Example:

String keyword = "Laptop";  
System.out.println(keyword.toLowerCase()); // laptop

# trim()

Purpose: Removes leading and trailing whitespace.

Use Case: Cleaning user input.

Example:

String username = " admin ";  
System.out.println(username.trim()); // "admin"

# contains(CharSequence s)

Purpose: Checks if the string contains a sequence.

Use Case: Searching for keywords in product descriptions.

Example:

String description = "This phone has a great camera.";  
if (description.contains("camera")) {  
 System.out.println("Camera feature found.");  
}

# startsWith(String prefix), endsWith(String suffix)

Purpose: Checks if a string starts or ends with a specific substring.

Use Case: Validating file types or log entries.

Example:

String fileName = "report.pdf";  
if (fileName.endsWith(".pdf")) {  
 System.out.println("PDF file detected.");  
}

# indexOf(String str), lastIndexOf(String str)

Purpose: Returns the index of the first/last occurrence of a substring.

Use Case: Parsing structured strings like URLs or logs.

Example:

String url = "https://example.com/page";  
int index = url.indexOf("example");  
System.out.println(index); // 8

# replace(char oldChar, char newChar), replaceAll(String regex, String replacement)

Purpose: Replaces characters or substrings.

Use Case: Sanitizing input or formatting strings.

Example:

String text = "Hello World";  
System.out.println(text.replace(" ", "\_")); // Hello\_World

# split(String regex)

Purpose: Splits the string into an array based on a delimiter.

Use Case: Parsing CSV data or user input.

Example:

String csv = "apple,banana,grape";  
String[] fruits = csv.split(",");

# isEmpty(), isBlank()

Purpose: Checks if the string is empty or only whitespace.

Use Case: Validating form fields.

Example:

String input = " ";  
System.out.println(input.isBlank()); // true

# valueOf()

Purpose: Converts other data types to string.

Use Case: Logging or displaying numbers.

Example:

int age = 25;  
String ageStr = String.valueOf(age);

# format(String format, Object... args)

Purpose: Formats strings like printf.

Use Case: Invoice numbers, reports, logs.

Example:

String invoice = String.format("INV-%05d", 123);  
System.out.println(invoice); // INV-00123