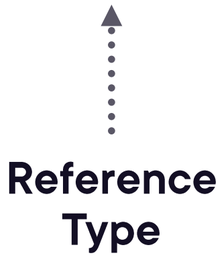
Manipulating Text with String and StringBuilder

Esteban Herrera
Author

@eh3rrera | eherrera.net

String Is a Reference Type

```
String greeting = "Hello";
```



Reference Type

Object

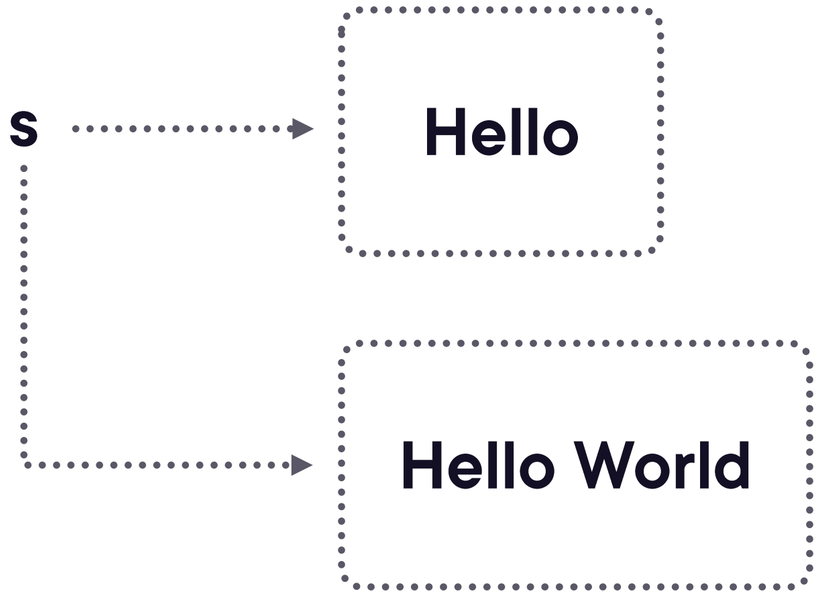
String Is a Reference Type

```
String greeting = "Hello";
```

String Immutability

```
String s = "Hello";
```

```
s = s + " World";
```



Advantages and Trade-offs of String Immutability

No risk of accidental modifications.

The Java Virtual Machine reuses common strings.

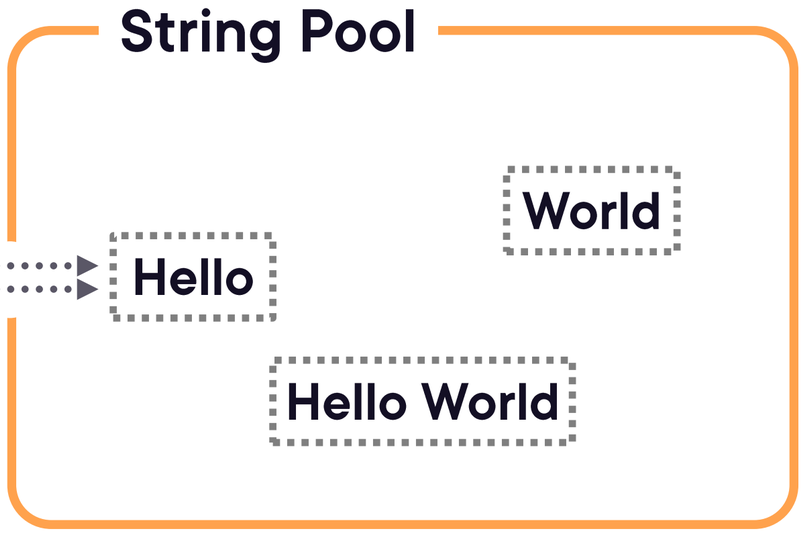
Concatenation creates new strings, reducing efficiency.

Creating a String

```
String literalString = "I am a literal string";
String objectString = new String("I am a String object");
```

The String Pool

```
String s1 = "Hello";

String s2 = "Hello";
```



Common Methods of the String Class

Concatenating Strings

```
String s1 = "Hello";
String s2 = "World";

String s3 = s1 + " " + s2;  // Using the + operator
String s4 = s1.concat(" ").concat(s2);  // Using the concat() method
```

Concatenating Strings

```
String s1 = "Hello";
String s2 = "World";

String s3 = s1 + " " + s2;  // Using the + operator

String s3 = new StringBuilder()
    .append(s1)
    .append(" ")
    .append(s2)
    .toString();

String s4 = s1.concat(" ").concat(s2);  // Using the concat() method
```

```
String s3 = "Hello" + " " + "World"
                  ↓
      String s3 = "Hello World"
```

```
String concat(String str)
```

Concat Method

Concatenating Strings

```
String s1 = "Hello";
String s2 = "World";

String s3 = s1.concat(" ");  // s3 is "Hello "
s3 = s3.concat(s2);          // s3 is now "Hello World"
```

```
int length() // Returns the number of characters in the string
int indexOf(String str) // Returns the index within the string of the first
                        // occurrence of the specified substring
String indent(int n) // Adjusts the indentation of each line of this string
String replace(char oldChar, char newChar) // Returns a new string by replacing
                                           // all occurrences of oldChar with newChar
String substring(int beginIndex, int endIndex) // Returns a new string that is
                                               // a substring of this string
String strip() // Returns a string with all leading and trailing white space removed
String toLowerCase() // Converts all the characters in this string to lower case
String toUpperCase() // Converts all the characters in this string to upper case
```

Common String Methods

```
char charAt(int index) // Returns the character at the specified index
boolean equals(Object anObject) // Compares this string to the specified object
boolean equalsIgnoreCase(String anotherString) // Compares this string to another
                                               // string, ignoring case considerations
boolean startsWith(String prefix) // Tests if this string starts with prefix
boolean endsWith(String suffix)   // Tests if this string ends with suffix
boolean contains(CharSequence s)  // Returns true if and only if this string contains
                                  // the specified sequence of char values
boolean isEmpty() // Returns true if, and only if, length() is 0
boolean isBlank() // Returns true if the string is empty or contains only white space
```

Common String Methods

Methods Returning a String Create a New Object

```
String s1 = "  Hello World   ";
String s2 = s1.strip();

System.out.println(s1);  // Still prints "  Hello World   "
System.out.println(s2);  // Prints "Hello World"
```

Method Chaining

```
String result = "  Hello World".toUpperCase().replace('O', '0');
System.out.println(result);  // Prints "HELL0 W0RLD"
```

Text Blocks

Traditional String vs. Text Block

```
String traditional = "{\n" +
                     "  \"name\": \"John Doe\",\n" +
                     "  \"age\": 30\n" +
                     "}";

String textBlock = """
                   {
                     "name": "John Doe",
                     "age": 30
                   }
                   """;
```

Text Block (SQL Example)

```
String textBlockSQL = """
                      SELECT id, name, age
                      FROM users
                      WHERE age > 21
                      ORDER BY name DESC;
                      """;
```

Text Block (HTML Example)

```
String textBlockHTML = """
                       <!DOCTYPE html>
                       <html>
                         <head>
                           <title>Welcome</title>
                         </head>
                         <body>
                           <h1>Hello, World!</h1>
                         </body>
                       </html>
                       """;
```

Rules for Escaping Special Characters

A single double quote (") or a pair of double quotes ("") within a text block does not need to be escaped.

Triple double quotes (""") within the text block need to be escaped to prevent them from prematurely ending the block.

To include a backslash character (\) in the text block, you need to escape it with another backslash (\\).

The StringBuilder Class

StringBuilder Example

```
StringBuilder sb = new StringBuilder("Hello");

sb.append(" World").insert(0, "Hey, ").delete(4, 9);

System.out.println(sb);  // Prints "Hey, World"
```

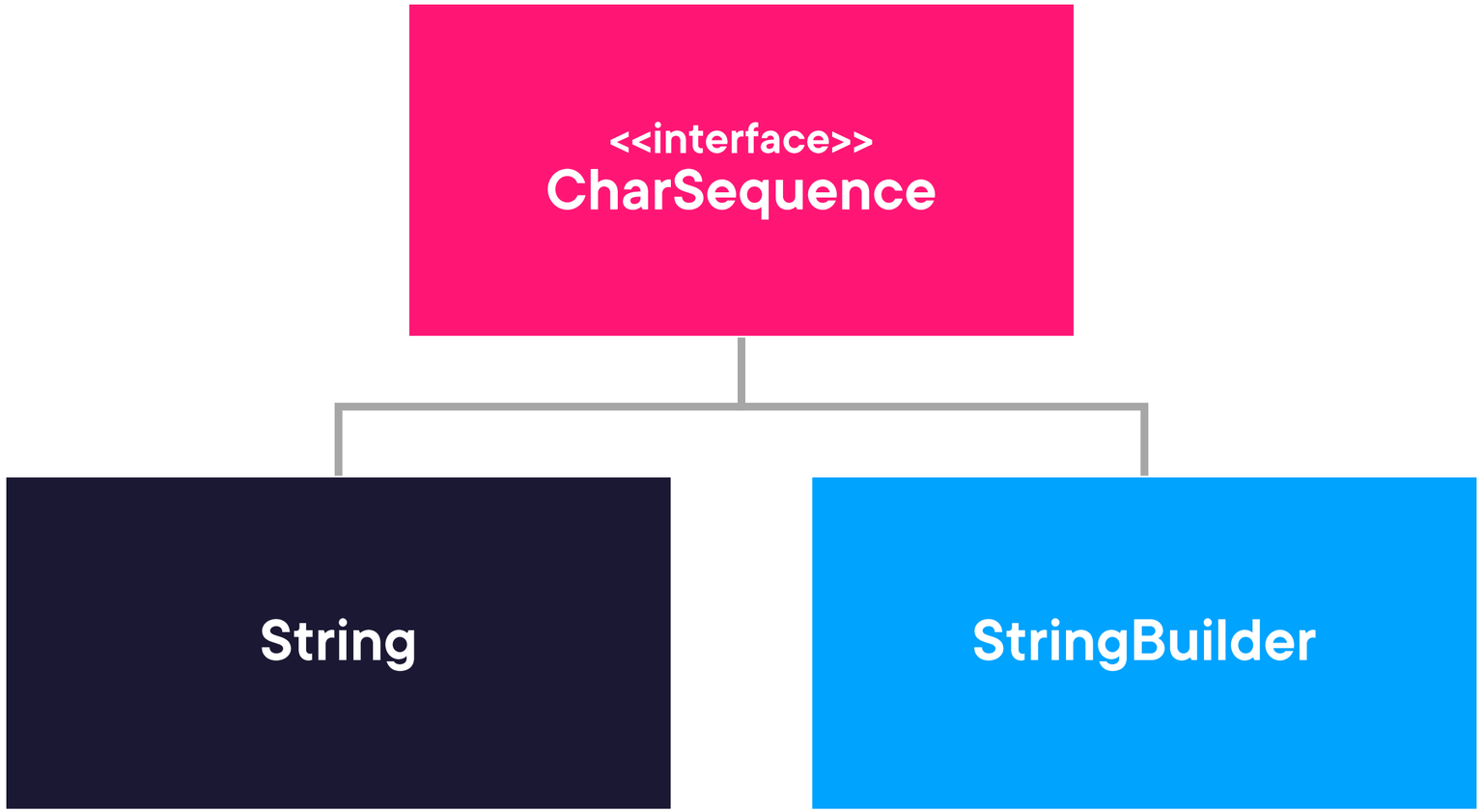
Creating a StringBuilder

```
// Creates an empty StringBuilder
StringBuilder sb1 = new StringBuilder();

// Creates a StringBuilder with initial capacity of 50
StringBuilder sb2 = new StringBuilder(50);

// Creates a StringBuilder initialized with the string "Hello"
StringBuilder sb3 = new StringBuilder("Hello");
```

```
int length() // Returns the number of characters in the StringBuilder
char charAt(int index) // Returns the character at the specified position
int indexOf(String str) // Returns the index within this string of the first occurrence
                        // of the specified substring
String substring(int start) //  Returns a new string that is a substring that begins at
                            // the specified index and extends to the end
String substring(int start, int end) // Returns a new string that is a substring of
                                     // the sequence
String toString() // Returns a string representing the data in the sequence
```

StringBuilder Methods That Mirror String's

CharSequence Interface



```
// Appends the string representation of the argument. There are overloads for all
// primitive types, char arrays, CharSequence, and Object
StringBuilder append(<type> t)

// Inserts the string representation of the second argument into the sequence
// at the position specified by the first argument. There are overloads for all
// primitive types, char arrays, CharSequence, and Object
StringBuilder insert(int offset, <type> t)

StringBuilder delete(int start, int end) // Removes the characters in a substring
StringBuilder deleteCharAt(int index) // Removes the char at the specified position
```

Common StringBuilder Methods

```
// Replaces the characters in a substring with characters in the specified string
StringBuilder replace(int start, int end, String str)

// Sets the character at the specified index to ch
void setCharAt(int index, char ch)

// Reverses the character sequence
StringBuilder reverse()
```

Common StringBuilder Methods

StringBuilder is mutable but not synchronized. The use of proper synchronization is required in multithreaded environments.

Review

The String Class

Concept	Description
String as an Object	String is a reference type, not a primitive.
Immutability	Strings cannot be changed after creation; any modification creates a new object.
Benefits of Immutability	Strings are safe for sharing and allow JVM optimizations through reuse.
Performance Consideration	String modifications are less efficient because new objects are created each time.
String Creation (Literal vs. new)	Literals use the string pool for memory efficiency, while new always creates a new object.
String Pool	A special memory area where identical string literals share the same object.
Using intern()	Converts a string object into a pooled string for memory optimization.
Comparing Strings	Use equals() for content comparison, == compares memory references.

Common Methods of the String Class

Method	Description
concat(String str)	Concatenates the specified string to the end of this string and returns a new string.
length()	Returns the number of characters in the string.
indexOf(String str)	Returns the index of the first occurrence of the specified substring.
charAt(int index)	Returns the character at the specified index.
indent(int n)	Adjusts the indentation of each line in the string by n spaces.
replace(char oldChar, char newChar)	Returns a new string with all occurrences of oldChar replaced by newChar.
substring(int beginIndex, int endIndex)	Returns a substring from beginIndex to endIndex - 1
strip()	Removes leading and trailing white space.
toLowerCase()	Converts all characters in the string to lowercase.
toUpperCase()	Converts all characters in the string to uppercase.
equals(Object anObject)	Compares this string to another object for exact equality.
equalsIgnoreCase(String anotherStr)	Compares two strings, ignoring case differences.
startsWith(String prefix)	Checks if the string starts with the specified prefix.
endsWith(String suffix)	Checks if the string ends with the specified suffix.
contains(CharSequence s)	Checks if the string contains the specified character sequence.
isEmpty()	Returns true if the string's length is 0.
isBlank()	Returns true if the string is empty or consists only of whitespace.
toString()	Returns a string representation of the object. Can be overridden for custom output.

Text Blocks

Concept	Description
Definition	A text block is a multi-line string enclosed in triple double-quotes ("""").
Advantages	Eliminates the need for escape sequences (\n, \"), making code cleaner and more readable.
Preserves Formatting	Maintains newlines and indentation exactly as written in the code.
Indentation Handling	The compiler removes the common whitespace prefix based on the least-indented line.
Escaping Rules	- Single and double quotes do not need escaping. - Triple double quotes (""") must be escaped. - Backslashes (\) need double escaping (\\). - Unicode escapes (\uXXXX) are supported.
Concatenation	Text blocks can be concatenated with traditional strings and variables using +.
Use Cases	Ideal for multi-line strings like HTML, JSON, XML, and SQL queries.

String vs. StringBuilder

Feature	String	StringBuilder
Mutability	Immutable	Mutable
Thread Safety	Thread-safe	Not thread-safe
Concatenation	Slower	Faster
Memory Efficiency	Less efficient for many modifications	More efficient for many modifications
When to Use	- Constant strings - Simple concatenation - Thread safety needed	- Building strings - Many modifications - Performance critical string operations

Practice Questions

Test your knowledge.

Question 1

What will be the output of the following program?

```
public class StringTest {
    public static void main(String[] args) {
        String str = " Java Rocks! ";

        String result = str.trim().substring(1, 4).toUpperCase().replace('A', 'X');

        System.out.println(result);
    }
}
```

A) AVX 

B) JXv 

C) JXX 

D) XVX 

```
Java Rocks!

J  a  v  a     R  o  c  k  s  !
0  1  2  3  4  5  6  7  8  9 10

ava

AVA

XVA
```

Question 2

What will be the output of the following program?

```
public class TextBlockTest {
    public static void main(String[] args) {
        String textBlock = """
                Java \
                Rocks!""";

        String normalString = "Java Rocks!";

        System.out.println(textBlock.equals(normalString));
    }
}
```

A) true 

B) false 

C) Compilation error 

D) Runtime exception 

Question 3

What will be the output of the following program?

```
public class StringBuilderTest {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Java");
        sb.append(" Rocks").delete(5, 7).insert(4, "-");

        System.out.println(sb);
    }
}
```

A) Java -ocks ✗

B) Jav-Rocks ✗

C) Java- cks ✓

D) Java-Rocks ✗

E) Compilation error ✗

```
Java

Java Rocks

J  a  v  a     R  o  c  k  s
0  1  2  3  4  5  6  7  8  9

Java cks

J  a  v  a     c  k  s
0  1  2  3  4  5  6  7

Java- cks
```