

Strings

Java Reference

String methods

A `String` is text. Useful methods: `.length()`, `.charAt(i)`, `.substring(a, b)`, `.indexOf(x)`, `.toUpperCase()`, `.equals(...)`. Strings are immutable — each method returns a **new** `String`.

```
public class Main {
    public static void main(String[] args) {
        String s = "Python";
        System.out.println(s.length());           // 6
        System.out.println(s.charAt(0));           // P
        System.out.println(s.substring(0, 3));      // Pyt
        System.out.println(s.toUpperCase());        // PYTHON
    }
}
```

The methods you will use every day:

Method	Meaning	Example → result
<code>.length()</code>	how many characters	<code>"Hi".length()</code> → 2
<code>.charAt(i)</code>	one character	<code>"Hi".charAt(0)</code> → H
<code>.substring(a, b)</code>	part, stops before b	<code>"Python".substring(0, 3)</code> → Pyt
<code>.substring(a)</code>	from a to the end	<code>"Python".substring(3)</code> → hon
<code>.indexOf(x)</code>	first position, -1 if absent	<code>"banana".indexOf("na")</code> → 2
<code>.equals(s)</code>	same text?	<code>"hi".equals("hi")</code> → true
<code>.compareTo(s)</code>	order: negative / 0 / positive	<code>"apple".compareTo("banana")</code> → negative

`compareTo` puts Strings in dictionary order — the AP exam uses it for sorting questions:

```
public class Main {
    public static void main(String[] args) {
        String a = "apple", b = "banana";
        System.out.println(a.compareTo(b) < 0);    // true (apple comes
↪ first)
        System.out.println(a.compareTo("apple"));  // 0 (equal)
    }
}
```

length() how many chars	charAt(i) char at index i	substring slice a part	equals same text? (not ==)
--------------------------------------	--	-------------------------------------	---

Exam tip: Strings are objects — use equals for text, not ==; indices start at 0

Key String methods: length, charAt, substring, equals

Building & traversing strings

Join strings with + (concatenation). Visit each character with a loop and `.charAt(i)`.

```
public class Main {
    public static void main(String[] args) {
        String word = "banana";
        int count = 0;
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) == 'a') count++;
        }
        System.out.println(count);    // 3
    }
}
```

When you build a long String in a loop, `StringBuilder` is much faster: append the pieces, then call `.toString()` once.

```
public class Main {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        for (int i = 1; i <= 5; i++) {
            sb.append(i).append(" ");
        }
        System.out.println(sb.toString().trim());    // 1 2 3 4 5
    }
}
```

Common mistakes

- Strings are immutable: `s.toUpperCase()` returns a new String, so store the result.
- Read a character with `s.charAt(i)`; the length is `s.length()` (a method, with `()`).
- Building a String with `+=` in a big loop is slow; use a `StringBuilder`.