Name: _____

Java Strings

What are Strings?

A **String** is a sequence of characters (letters, numbers, symbols, spaces) grouped together. In Java, Strings are used to store and work with text. Unlike primitive data types, String is actually a class, which means it comes with many helpful methods (functions) to manipulate text.

Simple Analogy: Strings are Like Beaded Necklaces

Imagine a necklace made of letter beads:

- **Each bead is a character:** 'H', 'e', 'l', 'l', 'o'
- **The whole necklace is the String:** "Hello"
- **You can:**
  - Count the beads (length)
  - Look at specific beads (charAt)
  - Cut pieces (substring)
  - Connect necklaces (concatenation)
  - Change all beads to the same color (toUpperCase)

String Positions (Index):
"H e l l o"
0 1 2 3 4
Remember: We start counting from 0, not 1!

How to Create Strings

Different Ways to Make Strings:

```
String name1 = "Alice"; // Most common way
String name2 = new String("Bob"); // Less common way
String empty = ""; // Empty string
String greeting = "Hello World!"; // With spaces and punctuation
```

String Methods (What You Can Do With Strings)

Method	What It Does	Example	Result
length()	Counts characters in string	"Hello".length()	5
charAt(index)	Gets character at position	"Hello".charAt(1)	'e'
substring(start)	Gets part of string from position	"Hello".substring(2)	"llo"
substring(start, end)	Gets part between two positions	"Hello".substring(1, 4)	"ell"
toUpperCase()	Makes all letters capital	"Hello".toUpperCase()	"HELLO"
toLowerCase()	Makes all letters small	"Hello".toLowerCase()	"hello"

equals(other)	Checks if two strings are same	"Hello".equals("hello")	false
contains(part)	Checks if string contains text	"Hello".contains("ell")	true

Example 1: Basic String Operations

```
String message = "Programming is Fun!";
System.out.println("Length: " + message.length()); // 19
System.out.println("First char: " + message.charAt(0)); // 'P'
System.out.println("Uppercase: " + message.toUpperCase()); // "PROGRAMMING IS FUN!"
System.out.println("Contains 'Fun': " + message.contains("Fun")); // true
```

Example 2: String Concatenation (Joining Strings)

```
String firstName = "John";
String lastName = "Doe";
int age = 25;

// Method 1: Using + operator
String fullName = firstName + " " + lastName;
System.out.println(fullName); // "John Doe"

// Method 2: Mixing strings and numbers
String info = firstName + " is " + age + " years old";
System.out.println(info); // "John is 25 years old"

// Method 3: Using concat() method
String greeting = "Hello ".concat(firstName);
System.out.println(greeting); // "Hello John"
```

⚠ Important: Comparing Strings

NEVER use == to compare strings! Always use .equals()

```
// WRONG WAY - Don't do this!
if (name == "Alice") { ... }
// CORRECT WAY - Always do this!
if (name.equals("Alice")) { ... }
// Even safer (prevents errors if name is null):
if ("Alice".equals(name)) { ... }
```

Why? The == operator checks if two strings are the exact same object in memory, but .equals() checks if they have the same text content.

Example 3: Working with User Input

```
Scanner input = new Scanner(System.in);

System.out.print("What's your name? ");
String name = input.nextLine();
```

```
// Clean up the input (remove extra spaces)
name = name.trim();
// Make first letter capital
if (name.length() > 0) {
  name = name.substring(0, 1).toUpperCase() + name.substring(1).toLowerCase();
}
System.out.println("Nice to meet you, " + name + "!");
```

Example 4: String Analysis

```
String sentence = "The quick brown fox jumps over the lazy dog";

// Count words (split by spaces)
String[] words = sentence.split(" ");
System.out.println("Number of words: " + words.length);
// Find and replace
String newSentence = sentence.replace("fox", "cat");
System.out.println("New sentence: " + newSentence);
// Check if it starts/ends with something
System.out.println("Starts with 'The': " + sentence.startsWith("The"));
System.out.println("Ends with 'dog': " + sentence.endsWith("dog"));
```

More Useful String Methods

Method	What It Does	Example
trim()	Removes spaces from beginning and end	" hello ".trim() → "hello"
replace(old, new)	Replaces all occurrences of text	"hello".replace("l", "x") → "hexxo"
split(delimiter)	Breaks string into array of parts	"a,b,c".split(",") → ["a", "b", "c"]
startsWith(text)	Checks if string begins with text	"hello".startsWith("he") → true
endsWith(text)	Checks if string ends with text	"hello".endsWith("lo") → true
indexOf(text)	Finds position of text (or -1 if not found)	"hello".indexOf("ll") → 2

Real-Life Example: Email Validator

```
String email = "user@example.com";
// Basic email validation
boolean hasAt = email.contains("@");
boolean hasDot = email.contains(".");
boolean longEnough = email.length() >= 5;
boolean noSpaces = !email.contains(" ");

if (hasAt && hasDot && longEnough && noSpaces) {
  System.out.println("Email looks valid!");
} else {
  System.out.println("Email format is incorrect.");
}
```

String Building for Performance

```
// Slow way (creates many temporary strings)
String result = "";
for (int i = 1; i <= 5; i++) {
  result = result + i + " "; // Creates new string each time
}
// Fast way (more efficient for lots of concatenation)
StringBuilder sb = new StringBuilder();
for (int i = 1; i <= 5; i++) {
  sb.append(i).append(" ");
}
String fastResult = sb.toString();
```

Use StringBuilder when: You're building a string in a loop or doing many concatenations.

Important Rules to Remember

- **Strings are immutable** - you can't change them, only create new ones
- **Always use .equals() to compare strings**, never ==
- **String indexes start at 0**, not 1
- **Use double quotes** for strings ("hello"), single quotes for characters ('h')
- **Be careful with null strings** - they can cause errors
- **substring(start, end)** includes start but excludes end
- **Many string methods return new strings** - they don't change the original

Your Turn: Write Your Own Definition

What is a String? How would you explain it to a friend?

Write your definition in your own words:

Give three examples of when you might use Strings in a real program:

Example: "Storing someone's name in a contact app"

1. _____

2. _____

3. _____

Why do you think we use .equals() instead of == to compare strings?