Characters and Strings: Data Types for Text

- Java handles text via `char` and `String` with rich built-in methods.

The `char` Data Type

- Represents **single characters**.
- Literals enclosed in **single quotes** ('A').
- Distinct from `String` (double quotes: "A").

```
char letter = 'A';
char numChar = '4';
```

Type	Syntax	Example
char	' '	'A'
String	" "	"A"

Unicode and ASCII

Unicode:

- 16-bit standard (2 bytes).
- Format: `\uXXXX` (hexadecimal).
- Range: `\u0000` to `\uFFFF`.

ASCII subset:

- `'0'–'9'` → `\u0030–\u0039` (48–57).
- `'A'–'Z'` → `\u0041–\u005A` (65–90).
- `'a'–'z'` → `\u0061–\u007A` (97–122).

```
char letter = 'A';          // same as:
char letter = '\u0041';     // Unicode for 'A'
```

char Arithmetic: Increment/Decrement

char supports ++/-- operators

```
char ch = 'a';
System.out.println(++ch);  // Output: 'b'
System.out.println(ch++);  // Output: 'c' (then ch becomes 'd')
```

Escape Sequences

Purpose: Represent special characters

Sequence	Name	Unicode	Decimal
\b	Backspace	\u0008	8
\t	Tab	\u0009	9
\n	New line	\u000A	10
\\	Backslash	\u005C	92
\"	Double quote	\u0022	34

```
System.out.println("He said: \"Let's go!\"");
// Output: He said: "Let's go!"

System.out.println("\\t is a tab");
// Output: \t is a tab
```

Example

```
Hello	World!
This is a backslash: \
She said: "Hello!"
```

```
public class EscapeSequencesDemo {

    public static void main(String[] args) {

        System.out.println("Hello\tWorld!");

        System.out.println("This is a backslash: \\");

        System.out.println("She said: \"Hello!\"");

    }

}
```

Character Comparison

Rules:

- Compared by **Unicode/ASCII values**.
- Case-sensitive: `'a'` (97) > `'A'` (65).

Examples:

```
'a' < 'b';      // true (97 < 98)
'a' < 'A';      // false (97 > 65)
'1' < '8';      // true (49 < 56)
```

Character **Class Methods**

Utility methods (static):

- `isDigit(ch)` → `true` if digit
- `isLetter(ch)` → `true` if letter
- `isLowerCase(ch)` / `isUpperCase(ch)` → `true false`
- `toLowerCase(ch)` / `toUpperCase(ch)` → Returns lowercase/uppercase of ch

```
char ch = 'A';

if (Character.isUpperCase(ch)) {
    System.out.println(ch + " is uppercase");
}
char lowerCh = Character.toLowerCase(ch);  // 'a'
```

All methods take `char` as parameter.

Example

Task: create a Java program that:

- uses Unicode to declare Greek letters α, β, γ as char variables;
- prints each letter with its Unicode code (α : \u03b1, β : \u03b2, γ : \u03b4).

Expected output:

```
Alpha: α (\u03b1)
Beta: β (\u03b2)
Gamma: γ (\u03b4)
```

```java
public class UnicodeDemo {
    public static void main(String[] args) {
        char alpha = '\u03b1';
        char beta = '\u03b2';
        char gamma = '\u03b4';
        System.out.println("Alpha: " + alpha + " (\\u03b1)");
        System.out.println("Beta: " + beta + " (\\u03b2)");
        System.out.println("Gamma: " + gamma + " (\\u03b4)");
    }
}
```

Expected output:

```
Alpha: α (\u03b1)
Beta: β (\u03b2)
Gamma: γ (\u03b4)
```

Example

Task: write a program that:

- starts with char ch = 'c';
- uses ++ch to increment the character three times;
- prints the character after each increment.

```
After 1st increment: d
After 2nd increment: e
After 3rd increment: f
```

```
public class CharIncrementDemo {
    public static void main(String[] args) {
        char ch = 'c';
        System.out.println("After 1st increment: " + ++ch);
        System.out.println("After 2nd increment: " + ++ch);
        System.out.println("After 3rd increment: " + ++ch);
    }
}
```

Example

Task: create a program that:

- declares a char variable and assigns it the value 'F';
- uses Character.isUpperCase(), Character.isDigit(), and Character.toLowerCase() methods;
- prints results with descriptive labels.

```
'F' is uppercase: true
'F' is a digit: false
Lowercase of 'F': f
```

```java
public class CharacterMethodsDemo {
    public static void main(String[] args) {
        char ch = 'F';
        System.out.println("'" + ch + "' is uppercase: " + Character.isUpperCase(ch));
        System.out.println("'" + ch + "' is a digit: " + Character.isDigit(ch));
        char lowerCh = Character.toLowerCase(ch);
        System.out.println("Lowercase of '" + ch + "': " + lowerCh);
    }
}
```

The `String` Class

- `String` = sequence of `chars`. In Java, strings are objects that represent sequences of characters.
- Java provides the String **class** in the `java.lang` package to create and manipulate strings.

Key Characteristics of Strings in Java:

- **Immutable** (cannot be changed after creation).
- **Objects**, not primitive types
- Declared with double quotes.
- String class provides many useful methods for string manipulation
- String literals are stored in a **string pool** for memory efficiency

```
String message = "Welcome";
```

String Declaration

Method 1: Using String Literal (Most Common)

```
String message = "Welcome to Java";
```

How it works:

- Java checks the **string pool** (special memory area)
- If "Welcome to Java" already exists, reference is reused
- If not, a new string is created in the pool

Method 2: Using new Keyword

```
String message = new String("Welcome to Java");
```

How it works:

- Creates a **new object** in heap memory
- Creates a separate object even if the same string exists in pool
- Generally **not recommended** unless you need a distinct object

String Declaration

Method 3: Using Character Array

```
char[] charArray = {'J', 'a', 'v', 'a'};
String message = new String(charArray);  // "Java"
```

```java
// Example demonstrating the difference
public class StringDeclaration {
    public static void main(String[] args) {
        // Using literals (pooled)
        String s1 = "Java";
        String s2 = "Java";

        // Using constructor (heap)
        String s3 = new String("Java");

        System.out.println(s1 == s2);     // true (same object in pool)
        System.out.println(s1 == s3);     // false (different objects)
        System.out.println(s1.equals(s3)); // true (same content)
    }
}
```

Basic String Methods

Core methods:

1. `length()` → number of chars.
2. `charAt(index)` → char at position.
3. `concat(s1)` → join strings.
4. `toUpperCase()`/`toLowerCase()`.
5. `trim()` → remove leading/trailing spaces.

```
String message = "Welcome to Java";

int len = message.length();                          15
char ch = message.charAt(0);                         'W'
String upper = message.toUpperCase();                "WELCOME TO JAVA"
String lower = message.toLowerCase();                "welcome to java"
String trimmed = "  Hello  ".trim();                 "Hello"
String java = "Ja".concat("va");                     "Java"
```

Detailed Method Explanations

1. length() Method

Purpose: Returns the number of characters in the string.

```
String message = "Welcome to Java";
int length = message.length();
System.out.println("Length: " + length);  // Length: 15
```

2. charAt(index) Method

Purpose: Returns the character at the specified index (0-based indexing).

```
String message = "Welcome to Java";
char firstChar = message.charAt(0);    // 'W'
char fourthChar = message.charAt(3);   // 'c'
char lastChar = message.charAt(message.length() - 1); // 'a'

// Common error: IndexOutOfBoundsException
// char invalid = message.charAt(20);  // ERROR! Index 20 out of bounds
```

Visualizing String Indices

String:	W	e	l	c	o	m	e		t	o		J	a	v	a
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

String Concatenation

3. concat(String str) Method

Purpose: Concatenates (joins) the specified string to the end of this string.

+ operator.

+ with spaces.

```
String s1 = "Hello";
String s2 = "World";
String result = s1.concat(s2);
System.out.println(result);  // "HelloWorld"

// Multiple concatenations
String s3 = s1.concat(" ").concat(s2);
System.out.println(s3);      // "Hello World"
```

```
String s1 = "Hello";
String s2 = "World";
String s3 = s1 + s2;           // "HelloWorld"
String s4 = s1 + " " + s2;     // "Hello World"
```

+ is more readable for complex joins.

Important Notes on Concatenation

Order of operations matters:

```
System.out.println(1 + 2 + "3");      // "33" (1+2=3, then 3+"3"="33")
System.out.println("1" + 2 + 3);      // "123" (left to right)
System.out.println(1 + 2 + "3" + 4); // "334" (1+2=3, then "3"+"3"="33",
then "33"+4)
```

Detailed Method Explanations

4. Case Conversion Methods

```java
String message = "Welcome to Java";

// Convert to uppercase
String upper = message.toUpperCase();
System.out.println(upper);  // "WELCOME TO JAVA"

// Convert to lowercase
String lower = message.toLowerCase();
System.out.println(lower);  // "welcome to java"

// Original string remains unchanged (immutability)
System.out.println(message);  // "Welcome to Java"
```

Detailed Method Explanations

5. trim() Method

Purpose: Removes leading and trailing whitespace.

```
String padded = "  Hello World  ";
String trimmed = padded.trim();
System.out.println("'" + trimmed + "'");  // 'Hello World'

// Whitespace includes: spaces, tabs, newlines
String messy = "\n\t  Java  \n";
String clean = messy.trim();
System.out.println("'" + clean + "'");    // 'Java'
```

The substring() Method

The `substring()` method is one of the most useful methods in the String class. It extracts a portion of a string and returns it as a new string.

`substring(int beginIndex)`	Returns substring from beginIndex to end	`"Hello".substring(2)` → `"llo"`
`substring(int beginIndex, int endIndex)`	Returns substring from beginIndex to endIndex-1	`"Hello".substring(1, 4)` → `"ell"`

Important Rules

- **Begin index is inclusive** - the character at beginIndex is included
- **End index is exclusive** - the character at endIndex is NOT included
- Indices start at **0** (like all Java string methods)
- Returns a **new string** - original string remains unchanged

```
String:  H  e  l  l  o
Index:   0  1  2  3  4

substring(1)    → starts at index 1 → "ello"
substring(2)    → starts at index 2 → "llo"
substring(1,4)  → indices 1,2,3 included → "ell"
substring(0,2)  → indices 0,1 included → "He"
```

Reading Strings from Console

Using Scanner:

1. `next()` → Reading Single Words (until first whitespace).

```java
import java.util.Scanner;

public class ReadWords {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter three words: ");
        String word1 = input.next();  // Reads first word
        String word2 = input.next();  // Reads second word
        String word3 = input.next();  // Reads third word

        System.out.println("Word 1: " + word1);
        System.out.println("Word 2: " + word2);
        System.out.println("Word 3: " + word3);

        input.close();
    }
}
```

output
Enter three words: Java programming is fun
Word 1: Java
Word 2: programming
Word 3: is

Reading Strings from Console

Using Scanner**:**

2. `nextLine()` → reads entire line (reads everything until the Enter key).

```
import java.util.Scanner;

public class ReadLine {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a sentence: ");
        String line = input.nextLine();  // Reads entire line

        System.out.println("You entered: " + line);
        System.out.println("Length: " + line.length());

        input.close();
    }
}
```

output:
Enter a sentence: Java programming is really fun to learn
You entered: Java programming is really fun to learn
Length: 42

Reading a Single Character

Since there's no nextChar() method:

Workaround: Read `String`, then extract first `char`.

```
System.out.print("Enter a character: ");
String s = input.nextLine();
char ch = s.charAt(0);  // Get first char
```

Edge cases:

- Empty input → `StringIndexOutOfBoundsException`.
- Use `if (s.length() > 0)` to validate.

text input → string → char extraction