
List of Common String Methods in Java

Here are the most important methods provided by the `String` class:

Method	Description
<code>charAt(int index)</code>	Returns the character at the specified index.
<code>length()</code>	Returns the length of the string.
<code>substring(int beginIndex)</code>	Returns a substring from the given index to the end.
<code>substring(int beginIndex, int endIndex)</code>	Returns a substring between two indices.
<code>equals(Object another)</code>	Compares two strings for exact match.
<code>equalsIgnoreCase(String another)</code>	Compares two strings, ignoring case differences.
<code>compareTo(String another)</code>	Compares two strings lexicographically.
<code>compareToIgnoreCase(String another)</code>	Compares two strings lexicographically, ignoring case.
<code>contains(CharSequence s)</code>	Checks if the string contains the specified sequence.
<code>indexOf(String str)</code>	Returns the index of the first occurrence of the substring.
<code>lastIndexOf(String str)</code>	Returns the index of the last occurrence of the substring.
<code>toUpperCase()</code>	Converts all characters to uppercase.
<code>toLowerCase()</code>	Converts all characters to lowercase.
<code>trim()</code>	Removes leading and trailing whitespace.

<code>replace(char oldChar, char newChar)</code>	Replaces all occurrences of a character with a new character.
<code>replaceAll(String regex, String replacement)</code>	Replaces each substring matching regex with replacement.
<code>startsWith(String prefix)</code>	Checks if the string starts with a specified prefix.
<code>endsWith(String suffix)</code>	Checks if the string ends with a specified suffix.
<code>split(String regex)</code>	Splits the string into an array based on the given regex.
<code>toCharArray()</code>	Converts the string into a character array.
<code>isEmpty()</code>	Checks if the string is empty.
<code>matches(String regex)</code>	Checks if the string matches a given regular expression.
<code>intern()</code>	Returns a canonical representation for the string object.
<code>hashCode()</code>	Returns the hash code of the string.
<code>concat(String str)</code>	Concatenates the specified string to the end.
<code>valueOf(primitive types)</code>	Converts primitive types to string.

Big Example Covering Many String Methods

Here's a full Java program using many String methods together:

```
public class StringMethodsExample {
    public static void main(String[] args) {
        String original = " Hello World! Welcome to Java
Programming. ";

        // 1. length()
        System.out.println("Length: " + original.length());
```

```
// 2. trim()
String trimmed = original.trim();
System.out.println("Trimmed: '" + trimmed + "'");

// 3. toUpperCase() and toLowerCase()
System.out.println("Uppercase: " + trimmed.toUpperCase());
System.out.println("Lowercase: " + trimmed.toLowerCase());

// 4. charAt()
System.out.println("Character at index 6: " +
trimmed.charAt(6));

// 5. substring()
System.out.println("Substring from index 13: " +
trimmed.substring(13));
System.out.println("Substring from index 0 to 11: " +
trimmed.substring(0, 11));

// 6. contains()
System.out.println("Contains 'Java': " +
trimmed.contains("Java"));

// 7. startsWith() and endsWith()
System.out.println("Starts with 'Hello': " +
trimmed.startsWith("Hello"));
System.out.println("Ends with 'Programming.': " +
trimmed.endsWith("Programming.));

// 8. indexOf() and lastIndexOf()
System.out.println("First index of 'o': " +
trimmed.indexOf('o'));
System.out.println("Last index of 'o': " +
trimmed.lastIndexOf('o'));

// 9. replace()
String replaced = trimmed.replace('o', '0');
System.out.println("After replacing 'o' with '0': " +
replaced);

// 10. split()
```

```
String[] words = trimmed.split(" ");
System.out.println("Words:");
for (String word : words) {
    System.out.println(word);
}

// 11. equals() and equalsIgnoreCase()
String compare1 = "hello world! welcome to java
programming.";
System.out.println("Equals: " + trimmed.equals(compare1));
System.out.println("Equals Ignore Case: " +
trimmed.equalsIgnoreCase(compare1));

// 12. compareTo()
System.out.println("Compare To 'Hello': " +
trimmed.compareTo("Hello"));

// 13. toCharArray()
char[] characters = trimmed.toCharArray();
System.out.println("Characters:");
for (char ch : characters) {
    System.out.print(ch + " ");
}
System.out.println();

// 14. isEmpty()
String emptyString = "";
System.out.println("Is Empty String: " +
emptyString.isEmpty());

// 15. matches()
boolean matches = trimmed.matches(".*Java.*");
System.out.println("Matches regex '.*Java.*': " + matches);

// 16. intern()
String interned = trimmed.intern();
System.out.println("Interned String: " + interned);
}
}
```

Output (Shortened and Explained):

Length: 45

Trimmed: 'Hello World! Welcome to Java Programming.'

Uppercase: HELLO WORLD! WELCOME TO JAVA PROGRAMMING.

Lowercase: hello world! welcome to java programming.

Character at index 6: W

Substring from index 13: Welcome to Java Programming.

Substring from 0 to 11: Hello World

Contains 'Java': true

Starts with 'Hello': true

Ends with 'Programming.': true

First index of 'o': 4

Last index of 'o': 28

After replacing 'o' with '0': Hell0 W0rld! Welc0me t0 Java
Pr0gramming.

Words:

Hello

World!

Welcome

to

Java

Programming.

Equals: false

Equals Ignore Case: true

Compare To 'Hello': 87

Characters:

H e l l o W o r l d ! W e l c o m e t o J a v a P r o g r
a m m i n g .

Is Empty String: true

Matches regex '.*Java.*': true

Interned String: Hello World! Welcome to Java Programming.

Summary:

- Strings are immutable (cannot be changed once created).

- Java provides many useful methods for manipulation, comparison, searching, and splitting strings.
- Practice using multiple methods together makes you very good at handling real-world string problems.

✅ This covered a BIG example with most common methods.

Would you also like me to give some small real-world mini projects (like Email Validator, Password Strength Checker using Strings)? 🚀
(Just tell me!)

4o

You said:
now give for string buffer

ChatGPT said:

Perfect — let's now move to StringBuffer!

About StringBuffer

- **StringBuffer** is a mutable (changeable) sequence of characters.
- Unlike **String**, **StringBuffer** can be modified without creating new objects.
- Thread-safe: It is synchronized, meaning multiple threads can use it safely.

Common Methods of StringBuffer

Method	Description
<code>append(String str)</code>	Appends the specified string to the end.
<code>insert(int offset, String str)</code>	Inserts the string at the specified position.

<code>replace(int start, int end, String str)</code>	Replaces characters from start to end with given string.
<code>delete(int start, int end)</code>	Deletes characters from start to end.
<code>reverse()</code>	Reverses the sequence of characters.
<code>capacity()</code>	Returns the current capacity of the buffer.
<code>ensureCapacity(int minimumCapacity)</code>	Ensures the capacity is at least the given minimum.
<code>length()</code>	Returns the number of characters.
<code>charAt(int index)</code>	Returns the character at the specified index.
<code>setCharAt(int index, char ch)</code>	Changes the character at the specified index.
<code>substring(int start)</code>	Returns a substring starting from given index.
<code>substring(int start, int end)</code>	Returns a substring from start to end.
<code>deleteCharAt(int index)</code>	Deletes a single character at the specified position.
<code>toString()</code>	Converts the buffer to a string.

Big Example with StringBuffer

Here's a full Java program using most StringBuffer methods:

java

CopyEdit

```
public class StringBufferExample {
    public static void main(String[] args) {
        // Create StringBuffer
        StringBuffer sb = new StringBuffer("Hello");

        // 1. append()
        sb.append(" World");
        System.out.println("After append: " + sb);

        // 2. insert()
        sb.insert(5, ",");
```

```
System.out.println("After insert: " + sb);

// 3. replace()
sb.replace(6, 11, " Java");
System.out.println("After replace: " + sb);

// 4. delete()
sb.delete(5, 10);
System.out.println("After delete: " + sb);

// 5. reverse()
sb.reverse();
System.out.println("After reverse: " + sb);

// 6. capacity()
System.out.println("Capacity: " + sb.capacity());

// 7. ensureCapacity()
sb.ensureCapacity(50);
System.out.println("Capacity after ensureCapacity(50): " +
sb.capacity());

// 8. length()
System.out.println("Length: " + sb.length());

// 9. charAt()
System.out.println("Character at index 2: " + sb.charAt(2));

// 10. setCharAt()
sb.setCharAt(0, 'h');
System.out.println("After setCharAt: " + sb);

// 11. substring()
String sub = sb.substring(2, 7);
System.out.println("Substring (2,7): " + sub);

// 12. deleteCharAt()
sb.deleteCharAt(2);
System.out.println("After deleteCharAt: " + sb);

// 13. toString()
```



```
        String normalString = sb.toString();
        System.out.println("Converted to String: " + normalString);
    }
}
```

Output of this Program

pgsql

CopyEdit

```
After append: Hello World
After insert: Hello, World
After replace: Hello Java
After delete: Helloava
After reverse: avaolleH
Capacity: 21
Capacity after ensureCapacity(50): 50
Length: 8
Character at index 2: a
After setCharAt: hvaolleH
Substring (2,7): aolle
After deleteCharAt: hvolleH
Converted to String: hvolleH
```

Important Differences between String and StringBuffer

Feature	String	StringBuffer
Mutability	Immutable (cannot be changed)	Mutable (can be changed)
Performance	Slow for modifications (creates new object)	Fast for modifications
Thread Safety	Not thread-safe	Thread-safe (synchronized)

Usage

When few modifications are needed

When frequent modifications are needed

✅ Now you have a full strong idea of StringBuffer methods and usage!

👉 Do you also want me to show the differences between **StringBuilder** and **StringBuffer**?

(**StringBuilder** is faster but not thread-safe.)

Tell me if yes! 🚀

4o

You said:

now string builder

ChatGPT said:

Awesome! Now let's go into StringBuilder clearly.

About StringBuilder

- **StringBuilder** is almost same as **StringBuffer** – both are mutable.
- Difference:
 - **StringBuilder** is NOT thread-safe (NOT synchronized).
 - It is faster than **StringBuffer** for single-threaded applications.

Use StringBuilder when:

- ✅ You need high performance
 - ✅ You don't need multi-thread safety
-

Common Methods of StringBuilder

Method	Description
<code>append(String str)</code>	Appends the string to the end.
<code>insert(int offset, String str)</code>	Inserts the string at the specified index.
<code>replace(int start, int end, String str)</code>	Replaces characters from start to end with given string.
<code>delete(int start, int end)</code>	Deletes characters from start to end.
<code>reverse()</code>	Reverses the sequence of characters.
<code>capacity()</code>	Returns the current buffer capacity.
<code>ensureCapacity(int minimumCapacity)</code>	Ensures the buffer has minimum specified capacity.
<code>length()</code>	Returns the length (character count).
<code>charAt(int index)</code>	Returns character at given position.
<code>setCharAt(int index, char ch)</code>	Changes character at given position.
<code>substring(int start)</code>	Returns substring from start index.
<code>substring(int start, int end)</code>	Returns substring between two indices.
<code>deleteCharAt(int index)</code>	Deletes character at specified index.
<code>toString()</code>	Converts the builder to a String.

Big Example with StringBuilder

java

CopyEdit

```
public class StringBuilderExample {  
    public static void main(String[] args) {  
        // Create a StringBuilder  
        StringBuilder sb = new StringBuilder("Hello");  
  
        // 1. append()  
        sb.append(" World");  
        System.out.println("After append: " + sb);  
    }  
}
```

```
// 2. insert()
sb.insert(5, ",");
System.out.println("After insert: " + sb);

// 3. replace()
sb.replace(6, 11, " Java");
System.out.println("After replace: " + sb);

// 4. delete()
sb.delete(5, 10);
System.out.println("After delete: " + sb);

// 5. reverse()
sb.reverse();
System.out.println("After reverse: " + sb);

// 6. capacity()
System.out.println("Capacity: " + sb.capacity());

// 7. ensureCapacity()
sb.ensureCapacity(50);
System.out.println("Capacity after ensureCapacity(50): " +
sb.capacity());

// 8. length()
System.out.println("Length: " + sb.length());

// 9. charAt()
System.out.println("Character at index 2: " + sb.charAt(2));

// 10. setCharAt()
sb.setCharAt(0, 'h');
System.out.println("After setCharAt: " + sb);

// 11. substring()
String sub = sb.substring(2, 7);
System.out.println("Substring (2,7): " + sub);

// 12. deleteCharAt()
sb.deleteCharAt(2);
```

```
        System.out.println("After deleteCharAt: " + sb);

        // 13. toString()
        String normalString = sb.toString();
        System.out.println("Converted to String: " + normalString);
    }
}
```

Output

```
pgsql
CopyEdit
After append: Hello World
After insert: Hello, World
After replace: Hello Java
After delete: Helloava
After reverse: avaolleH
Capacity: 21
Capacity after ensureCapacity(50): 50
Length: 8
Character at index 2: a
After setCharAt: hvaolleH
Substring (2,7): aolle
After deleteCharAt: hvolleH
Converted to String: hvolleH
```

Quick Comparison — String vs StringBuffer vs StringBuilder

Feature	String	StringBuffer	StringBuilder
Mutability	Immutable	Mutable	Mutable
Thread-Safety	Not Thread-Safe	Thread-Safe (Synchronized)	Not Thread-Safe

Performance	Slow	Slower (because of synchronization)	Faster
Usage	Few changes, Fixed strings	Many changes, Multithreaded app	Many changes, Single-threaded app

✅ Now you know how StringBuilder works, along with full example and output!

👉 If you want next, I can show some mini projects using StringBuilder (like Palindrome Checker, URL Shortener, etc).

Want me to show you one? 🚀 (Just say "yes" if interested!)

4o

You said:

now give for Character and how it got used in real questions with strings

ChatGPT said:

Excellent — now let's dive into Character in Java.

I'll explain it properly + give real examples (which are asked in coding interviews related to Strings!).

Character Class in Java

- **Character** is a wrapper class for **char** (primitive type).
 - It is found in the package **java.lang**.
 - It provides many static utility methods to work with characters.
-

Important Methods of Character Class

Method	Description
isLetter(char ch)	Checks if the character is a letter (a-z or A-Z).

<code>isDigit(char ch)</code>	Checks if the character is a digit (0-9).
<code>isWhitespace(char ch)</code>	Checks if the character is a whitespace.
<code>isUpperCase(char ch)</code>	Checks if the character is uppercase.
<code>isLowerCase(char ch)</code>	Checks if the character is lowercase.
<code>toUpperCase(char ch)</code>	Converts a character to uppercase.
<code>toLowerCase(char ch)</code>	Converts a character to lowercase.
<code>isAlphabetic(int codePoint)</code>	Checks if the character is an alphabet letter.
<code>isDefined(char ch)</code>	Checks if the character has a defined Unicode value.

Real-World Example 1: Count Letters, Digits, Spaces, and Others

(🔥 Very common interview question!)

java

CopyEdit

```
public class CharacterExample1 {
    public static void main(String[] args) {
        String input = "Hello123 World! @2024";

        int letters = 0, digits = 0, spaces = 0, others = 0;

        for (int i = 0; i < input.length(); i++) {
            char ch = input.charAt(i);
            if (Character.isLetter(ch)) {
                letters++;
            } else if (Character.isDigit(ch)) {
                digits++;
            } else if (Character.isWhitespace(ch)) {
                spaces++;
            } else {
                others++;
            }
        }
    }
}
```

```
        }
    }

    System.out.println("Letters: " + letters);
    System.out.println("Digits: " + digits);
    System.out.println("Spaces: " + spaces);
    System.out.println("Other characters: " + others);
}
}
```

Output

yaml

CopyEdit

Letters: 10

Digits: 7

Spaces: 3

Other characters: 2

Real-World Example 2: Check if String is Only Letters

java

CopyEdit

```
public class CharacterExample2 {
    public static void main(String[] args) {
        String str = "HelloWorld";

        boolean onlyLetters = true;
        for (char ch : str.toCharArray()) {
            if (!Character.isLetter(ch)) {
                onlyLetters = false;
                break;
            }
        }

        if (onlyLetters) {
```



```
        System.out.println("The string contains only letters.");
    } else {
        System.out.println("The string contains non-letter
characters.");
    }
}
}
```

Output

sql

CopyEdit

The string contains only letters.

Real-World Example 3: Convert lowercase to uppercase manually

(Without using `str.toUpperCase()` directly.)

java

CopyEdit

```
public class CharacterExample3 {
    public static void main(String[] args) {
        String input = "java programming";
        StringBuilder result = new StringBuilder();

        for (char ch : input.toCharArray()) {
            if (Character.isLowerCase(ch)) {
                result.append(Character.toUpperCase(ch));
            } else {
                result.append(ch);
            }
        }

        System.out.println("Converted String: " + result);
    }
}
```

Output

Converted String: JAVA PROGRAMMING

Summary

✓ **Character** class is used mainly for:

- Checking character types (letter, digit, whitespace, etc.).
- Changing case (lowercase → uppercase).
- Validating and filtering strings.

✓ It combines with Strings in interview questions like:

- Counting characters types
 - Validation (only letters, only numbers)
 - String formatting without built-in methods
-

Quick Tip

In almost every string-based coding interview, you will somehow use:

- `Character.isLetter()`
- `Character.isDigit()`
- `Character.toUpperCase()`
🔥 Super important!

Would you also like me to show harder real-world problems (like "Check if two strings are anagrams", "Strong Password Checker") where Character is deeply used? 🚀
(If yes, tell me!)

40