**Summary of Key Topics Related to Strings in Java**

**1. The String Class and Its Methods**

The String class in Java provides a variety of methods to manipulate strings. Some of the most commonly used methods include:

* **length()**: Returns the length of the string.
* **charAt(int index)**: Returns the character at the specified index.
* **substring(int beginIndex)**: Returns a new string starting from the specified index to the end.
* **substring(int beginIndex, int endIndex)**: Returns a new string starting from beginIndex and ending at endIndex - 1.
* **equals(Object obj)**: Compares the string with another string (case-sensitive).
* **equalsIgnoreCase(String anotherString)**: Compares two strings, ignoring case.
* **toUpperCase()**: Converts all characters to uppercase.
* **toLowerCase()**: Converts all characters to lowercase.
* **contains(CharSequence sequence)**: Checks if a string contains the specified sequence of characters.
* **indexOf(String str)**: Returns the index of the first occurrence of a substring.
* **replace(CharSequence old, CharSequence new)**: Replaces occurrences of a specified sequence in the string.

**Example**:

java

CopyEdit

public class Main {

public static void main(String[] args) {

String text = "Hello, World!";

System.out.println("Length: " + text.length()); // 13

System.out.println("Char at index 0: " + text.charAt(0)); // H

System.out.println("Substring: " + text.substring(7)); // World!

System.out.println("Substring: " + text.substring(0, 5)); // Hello

System.out.println("To Upper Case: " + text.toUpperCase()); // HELLO, WORLD!

System.out.println("To Lower Case: " + text.toLowerCase()); // hello, world!

System.out.println("Contains 'World': " + text.contains("World")); // true

System.out.println("Index of 'World': " + text.indexOf("World")); // 7

}

}

**2. String Immutability**

Strings in Java are **immutable**, meaning once a String object is created, its value cannot be changed. If any operation is performed that modifies a string, a new string object is created instead of altering the original string.

**Example**:

java

CopyEdit

public class Main {

public static void main(String[] args) {

String str = "Hello";

str = str + " World"; // Concatenation creates a new String object

System.out.println(str); // Output: Hello World

}

}

**Why is String Immutable?**

* **Security**: Immutable objects are safer when shared across different parts of the program.
* **Hashcode Caching**: Strings are often used as keys in hashmaps, and immutability ensures the hashcode remains constant.
* **Efficiency**: Since strings are immutable, they can be reused (interned) and shared.

**3. StringBuilder and StringBuffer**

Both StringBuilder and StringBuffer are **mutable** alternatives to String that allow for more efficient string manipulation, especially when frequent changes are made.

* **StringBuilder**: Best suited for single-threaded environments. It’s faster because it doesn’t use synchronization.
* **StringBuffer**: Used for thread-safe environments. It’s slower because its methods are synchronized.

Both classes offer the following methods:

* **append()**: Adds the specified string to the end.
* **insert()**: Inserts a string at a given index.
* **delete()**: Removes a substring.
* **reverse()**: Reverses the string.

**Example**:

java

CopyEdit

public class Main {

public static void main(String[] args) {

StringBuilder sb = new StringBuilder("Hello");

sb.append(" World");

sb.insert(6, "Beautiful ");

sb.delete(6, 16);

sb.reverse();

System.out.println(sb); // Output: dlroW olleH

}

}

**4. String Operations**

Common string operations in Java include:

* **Concatenation**: Combining strings together.

java

CopyEdit

String str1 = "Hello";

String str2 = "World";

String result = str1 + " " + str2; // Concatenation

System.out.println(result); // Output: Hello World

* **Comparison**: Comparing strings using methods like equals(), equalsIgnoreCase(), and compareTo().

java

CopyEdit

String str1 = "Hello";

String str2 = "hello";

System.out.println(str1.equals(str2)); // false

System.out.println(str1.equalsIgnoreCase(str2)); // true

* **Splitting**: Splitting a string based on a delimiter.

java

CopyEdit

String str = "apple,banana,orange";

String[] fruits = str.split(",");

for (String fruit : fruits) {

System.out.println(fruit);

}

**Output**:

CopyEdit

apple

banana

orange

**5. Regular Expressions (Pattern and Matcher Classes)**

Regular expressions (regex) are used for searching, matching, and manipulating strings based on patterns.

* **Pattern Class**: Represents a compiled regular expression.
* **Matcher Class**: Performs the actual search and matching.

**Common Regex Operations**:

* **matches()**: Checks if the entire string matches the regex.
* **find()**: Finds the next substring that matches the regex.
* **group()**: Returns the matched substring.

**Example** (Using Regex):

java

CopyEdit

import java.util.regex.\*;

public class Main {

public static void main(String[] args) {

String text = "The quick brown fox";

// Pattern to match words starting with 'b'

Pattern pattern = Pattern.compile("\\b[bB]\\w+");

Matcher matcher = pattern.matcher(text);

while (matcher.find()) {

System.out.println("Found: " + matcher.group());

}

}

}

**Output**:

makefile

CopyEdit

Found: brown

In this example, the regex \\b[bB]\\w+ matches words starting with the letter 'b' (case-insensitive), and find() is used to locate and print matching substrings.

**Summary:**

* **String Class**: Offers methods like length(), substring(), and equals() for efficient string manipulation.
* **String Immutability**: Strings cannot be changed once created, ensuring security and efficiency.
* **StringBuilder and StringBuffer**: Mutable alternatives to String for efficient string modifications, with StringBuilder being faster for single-threaded use.
* **String Operations**: Operations such as concatenation, comparison, and splitting are common for string handling.
* **Regular Expressions**: Use Pattern and Matcher for complex string searching and manipulation tasks.