StringBuilder in Java

In Java, `StringBuilder` is a class used to create and manipulate mutable strings. Unlike `String`, which creates a new object whenever modified, `StringBuilder` allows for efficient in-place modifications, saving memory and processing time.

---

1. Why Use StringBuilder?

The `String` class in Java is immutable, meaning any change to a `String` object creates a new object. This can be inefficient when performing numerous modifications, such as concatenations, in a loop.

`StringBuilder` addresses this inefficiency:

- **Mutable:** Allows changes without creating new objects.
- **Performance:** Faster than `String` for operations involving frequent modifications.

---

2. Creating a StringBuilder

```
StringBuilder sb = new StringBuilder();
```

You can also initialize it with:

- **Empty Constructor**: Creates an empty `StringBuilder` with a capacity of 16.
- **Capacity Constructor**: Specify initial capacity.
- **String Constructor**: Initialize with a specific string.

Examples

```
// Empty StringBuilder
StringBuilder sb1 = new StringBuilder();

// Specified capacity
```

```
StringBuilder sb2 = new StringBuilder(50);

// Initialized with a string
StringBuilder sb3 = new StringBuilder("Hello");
```

3. Common Methods of StringBuilder

Here are the frequently used methods of `StringBuilder`:

Method	Description
`append(String str)`	Adds the specified string to the end of the current string.
`insert(int offset, String str)`	Inserts a string at the specified position.
`replace(int start, int end, String str)`	Replaces characters in a specified range with another string.
`delete(int start, int end)`	Deletes characters in a specified range.
`reverse()`	Reverses the sequence of characters in the `StringBuilder`.
`length()`	Returns the number of characters in the `StringBuilder`.
`capacity()`	Returns the current capacity of the `StringBuilder`.
`charAt(int index)`	Returns the character at the specified index.
`setCharAt(int index, char ch)`	Sets the character at the specified index.
`toString()`	Converts the `StringBuilder` to a `String`.

4. Examples of StringBuilder in Action

Example 1: Append and Insert

```
public class StringBuilderExample {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Hello");
```

```
        // Append
        sb.append(" World");
        System.out.println("After Append: " + sb); // Hello W
orld

        // Insert
        sb.insert(5, ",");
        System.out.println("After Insert: " + sb); // Hello,
World
    }
}
```

Example 2: Replace and Delete

```
public class StringBuilderExample {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Hello, Worl
d!");

        // Replace
        sb.replace(7, 12, "Java");
        System.out.println("After Replace: " + sb); // Hello,
Java!

        // Delete
        sb.delete(5, 7);
        System.out.println("After Delete: " + sb); // HelloJa
va!
    }
}
```

Example 3: Reverse a String

```
public class StringBuilderExample {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Hello");

        // Reverse
        sb.reverse();
        System.out.println("Reversed: " + sb); // olleH
    }
}
```

Example 4: Check Capacity and Length

```
public class StringBuilderExample {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Java");

        // Length
        System.out.println("Length: " + sb.length()); // 4

        // Capacity
        System.out.println("Capacity: " + sb.capacity()); //
16 (default) + 4
    }
}
```

Example 5: Building a String Dynamically

```
public class StringBuilderExample {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();

        // Append numbers from 1 to 10
        for (int i = 1; i <= 10; i++) {
```

```
            sb.append(i).append(" ");
        }

        System.out.println("Generated String: " + sb); // 1 2
3 4 5 6 7 8 9 10
    }
}
```

5. Advantages of StringBuilder

1. **Efficient Memory Usage**: Does not create unnecessary objects for every modification.
2. **Ease of Use**: Provides intuitive methods for string manipulation.
3. **Performance**: Faster than using concatenation (`+`) in loops.

6. When to Use StringBuilder?

- When you need to perform multiple string manipulations (concatenation, insertion, etc.).
- When you are building dynamic strings, such as in loops.

7. Practice Problems

1. Write a program to reverse a string using `StringBuilder`.
2. Use `StringBuilder` to create a comma-separated list of integers from 1 to 50.
3. Implement a method that takes a sentence and replaces all spaces with underscores using `StringBuilder`.

By understanding and practicing `StringBuilder`, you'll be able to perform efficient and flexible string manipulations in your Java programs.