Assignment No:-33

Name:-Suryawanshi Sangramsingh Sambhaji

Batch: - Delta - DCA (Java) 2024 Date:-20/6/2024

1) What is a StringBuffer in Java and how is it different from String?

Ans: StringBuffer is a thread-safe, mutable sequence of characters. Unlike String, which is immutable, StringBuffer allows modification of the character sequence without creating new objects.

2) Explain the concept of mutability in the context of StringBuffer.

Ans: Mutability means that an object can be changed after it is created. StringBuffer is mutable, so its contents can be modified through methods like append(), insert(), delete(), and reverse() without creating new instances.

3) Why is StringBuffer considered thread-safe, and how is this achieved?

Ans: StringBuffer is considered thread-safe because its methods are synchronized, meaning only one thread can execute any of its synchronized methods at a time. This ensures safe use by multiple threads.

4) Compare and contrast String and StringBuffer regarding performance and memory usage.

Ans: **String**: Immutable, requires new objects for modifications, potentially leading to higher memory usage and slower performance in scenarios with many modifications.

StringBuffer: Mutable, modifications do not create new objects, generally faster and more memory-efficient for scenarios with frequent modifications. However, StringBuffer is slower than StringBuilder due to synchronization.

5) Can you provide examples of situations where using StringBuffer is preferable over String?

Ans: StringBuffer is preferable in multi-threaded environments where strings are frequently modified. For example, when constructing log messages in a multi-threaded logging system.

6) Discuss the key methods available in the StringBuffer class.

Ans: append(), insert(), delete(), deleteCharAt(), reverse(), setCharAt(), charAt(), capacity(), ensureCapacity(), length(), substring(), toString().

7) How does the append() method work in StringBuffer, and what is its significance?

Ans: The append() method adds the specified data to the end of the StringBuffer. It can accept various types such as String, int, char, boolean, etc. It is significant because it allows efficient concatenation without creating new strings.

8) Explain the role of the reverse() method in the StringBuffer class.

Ans: The reverse() method reverses the sequence of characters in the StringBuffer. It is useful for reversing strings or palindromic operations.

9) What is the capacity of a StringBuffer, and how does it dynamically adjust its size?

Ans: The capacity is the amount of storage available for characters without reallocating. It starts with an initial capacity, and if the internal buffer overflows, it automatically increases, usually by (old capacity * 2) + 2.

10) Describe the synchronization mechanism employed by StringBuffer for thread safety.

Ans: StringBuffer uses synchronized methods to ensure that only one thread can modify the buffer at a time, providing thread safety by preventing concurrent access issues.

11) Can you explain the importance of the ensureCapacity() method in StringBuffer?

Ans: The ensureCapacity(int minimumCapacity) method ensures that the StringBuffer has at least the specified minimum capacity. This can improve performance by reducing the number of reallocations needed when appending large amounts of data.

12) Discuss scenarios where using StringBuffer might result in better performance than using String concatenation.

Ans: StringBuffer provides better performance in scenarios with frequent string concatenations, such as building complex strings in loops or multi-threaded environments where thread safety is required.

13) How does the StringBuffer class handle the deletion of characters from a sequence of characters?

Ans: The delete(int start, int end) method removes the characters from the specified start index to the end index. The deleteCharAt(int index) method removes the character at the specified index.

14) What is the significance of the setCharAt() method in the StringBuffer class?

Ans: The setCharAt(int index, char ch) method sets the character at the specified index to the given character. It is useful for modifying individual characters within the buffer.

15) Compare the performance of StringBuffer with StringBuilder and highlight the differences.

Ans: StringBuffer is synchronized and thread-safe, making it slower than StringBuilder, which is not synchronized and faster. Use StringBuffer for multi-threaded contexts and StringBuilder for single-threaded contexts where performance is critical.

16) Explain the role of the charAt() method in the context of StringBuffer.

Ans: The charAt(int index) method returns the character at the specified index in the StringBuffer. It allows for reading individual characters within the buffer.

17) Discuss the exceptions that can be thrown by the methods in the StringBuffer class.

Ans: Methods in StringBuffer can throw IndexOutOfBoundsException when attempting to access an invalid index, such as in charAt(), delete(), insert(), setCharAt(), etc.

18) How can you convert a StringBuffer to a String in Java?

Ans: Use the toString() method to convert a StringBuffer to a String. Example

```
StringBuffer sb = new StringBuffer("Hello");
String str = sb.toString();
```

19) Discuss the impact of using StringBuffer in a multi-threaded environment.

Ans: StringBuffer ensures thread safety through synchronization, preventing concurrent modification issues. However, this synchronization can lead to performance overhead compared to non-synchronized alternatives like StringBuilder.

20) Explain the concept of "chaining" when it comes to invoking methods on a StringBuffer object.

Ans: Method chaining allows multiple method calls to be chained together in a single statement by returning the StringBuffer object itself from each method. Example:

```
StringBuffer sb = new StringBuffer();
sb.append("Hello").append(" ").append("World");
```

This approach leads to more readable and concise code.