Questions

1. What is String in java
    1. Explain the Diff Constructors of string class
    2. also list the diff btw string and string buffer class
2. Explain the difference between equalsto and == ,with examples
3. explain how to check the occurance of a substring in a given string
4. Explain the following methods with respect to a string buffer class
    1. capacity()
    2. delete()
    3. replace()
    4. append()
    5. substring()
    6. reverse()
    7. deletecharAt()
    8. charAt()
    9. insert()
5. Explain any two character handling questions in string class
6. Write a java program to sort names using bubble sort
7. explain the follwing string comparisioin methods, with examples
    1. equals()
    2. regionMatches()
    3. startsWith()
    4. endsWith()
8. How CompareTo() method diffrs from CompareToIgnoreCase(). Write a java program to sort an array of string in decending order by ignoring the case
9. Character extraction methods
    1. charAt()
    2. toCharArr()
10. Explain how to modify a string by using the following methods
    1. substring()
    2. Concat()
    3. Replace()
    4. Trim()
11. Develop a java program to count the occurances of a character in a string
12. Differentiate between String and String buffer
13. Explain reverse() of string buffer with an example

Answers

1. String in Java

String is a class in Java that represents a sequence of characters. Strings are immutable, meaning once a `String` object is created, its value cannot be changed.

1.1 Constructors of String Class

- **String()**: Creates an empty string.

```
String str = new String();
```

- **String(String original)**: Creates a string with the same value as the specified string.

```
String str = new String("Hello");
```

- **String(char[] value)**: Converts a character array into a string.

```
char[] chars = {'H', 'e', 'l', 'l', 'o'};
String str = new String(chars);
```

- **String(char[] value, int offset, int count)**: Converts a subarray of characters into a string.

```
char[] chars = {'H', 'e', 'l', 'l', 'o'};
String str = new String(chars, 1, 3); // "ell"
```

- **String(byte[] bytes)**: Constructs a new string by decoding the specified array of bytes using the platform's default charset.

```
byte[] bytes = {65, 66, 67};
String str = new String(bytes); // "ABC"
```

1.2 Difference between String and StringBuffer

- **Mutability**: `String` is immutable, while `StringBuffer` is mutable.
- **Performance**: `StringBuffer` is faster for concatenation and other modifications because it doesn't create new objects.
- **Thread-Safety**: `StringBuffer` is synchronized, making it thread-safe. `StringBuilder` (another mutable class) is not synchronized and thus not thread-safe but faster.

2. Difference between `equals` and `==`

- `==` **Operator**: Compares references, not values. It checks if both references point to the same object.

```
String a = new String("hello");
String b = new String("hello");
```

```
System.out.println(a == b); // false
```

- `equals` **Method**: Compares values of the objects.

  ```
  System.out.println(a.equals(b)); // true
  ```

3. Check the Occurrence of a Substring

You can use the `contains` method or `indexOf` method to check the occurrence of a substring in a given string.

```
String str = "hello world";
String substr = "world";

// Using contains
boolean contains = str.contains(substr); // true

// Using indexOf
int index = str.indexOf(substr); // 6, -1 if not found
```

4. StringBuffer Methods

4.1 `capacity()`

Returns the current capacity of the buffer.

```
StringBuffer sb = new StringBuffer();
System.out.println(sb.capacity()); // Default 16
```

4.2 `delete(int start, int end)`

Deletes characters from the start index to the end index.

```
StringBuffer sb = new StringBuffer("Hello");
sb.delete(1, 3); // "Hlo"
```

4.3 `replace(int start, int end, String str)`

Replaces characters from the start index to the end index with the specified string.

```java
StringBuffer sb = new StringBuffer("Hello");
sb.replace(1, 3, "ai"); // "Hallo"
```

4.4 `append(String str)`

Appends the specified string to the buffer.

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

4.5 `substring(int start, int end)`

Returns a new string that contains a subsequence of characters currently contained in the buffer.

```java
StringBuffer sb = new StringBuffer("Hello");
String sub = sb.substring(1, 3); // "el"
```

4.6 `reverse()`

Reverses the characters in the buffer.

```java
StringBuffer sb = new StringBuffer("Hello");
sb.reverse(); // "olleH"
```

4.7 `deleteCharAt(int index)`

Deletes the character at the specified index.

```java
StringBuffer sb = new StringBuffer("Hello");
sb.deleteCharAt(1); // "Hllo"
```

4.8 `charAt(int index)`

Returns the character at the specified index.

```java
StringBuffer sb = new StringBuffer("Hello");
char c = sb.charAt(1); // 'e'
```

4.9 `insert(int offset, String str)`

Inserts the specified string at the specified position.

```
StringBuffer sb = new StringBuffer("Hello");
sb.insert(1, "ey"); // "Heyllo"
```

5. Character Handling Methods in String Class

`charAt(int index)`: Returns the character at the specified index.

```
String str = "Hello";
char c = str.charAt(1); // 'e'
```

`toCharArray()`: Converts the string into a new character array.

```
String str = "Hello";
char[] chars = str.toCharArray(); // ['H', 'e', 'l', 'l', 'o']
```

6. Java Program to Sort Names Using Bubble Sort

```
public class BubbleSortExample {
    public static void main(String[] args) {
        String[] names = {"John", "Alice", "Bob"};
        bubbleSort(names);

        for (String name : names) {
            System.out.println(name);
        }
    }

    public static void bubbleSort(String[] array) {
        int n = array.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - 1 - i; j++) {
                if (array[j].compareTo(array[j + 1]) > 0) {
                    String temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
    }
}
```

7. String Comparison Methods

7.1 `equals()`

Compares the value of two strings.

```java
String str1 = "Hello";
String str2 = "Hello";
System.out.println(str1.equals(str2)); // true
```

7.2 `regionMatches()`

Compares a specific region of two strings.

```java
String str1 = "Hello";
String str2 = "ell";
boolean match = str1.regionMatches(1, str2, 0, 3); // true
```

7.3 `startsWith()`

Checks if the string starts with the specified prefix.

```java
String str = "Hello";
boolean starts = str.startsWith("He"); // true
```

7.4 `endsWith()`

Checks if the string ends with the specified suffix.

```java
String str = "Hello";
boolean ends = str.endsWith("lo"); // true
```

8. `compareTo()` vs `compareToIgnoreCase()`

- `compareTo()`: Compares two strings lexicographically.
- `compareToIgnoreCase()`: Compares two strings lexicographically, ignoring case differences.

```java
String str1 = "Hello";
String str2 = "hello";

int result = str1.compareTo(str2); // negative
int resultIgnoreCase = str1.compareToIgnoreCase(str2); // 0
```

Program to Sort Strings in Descending Order Ignoring Case

```
import java.util.Arrays;
import java.util.Collections;

public class SortDescending {
    public static void main(String[] args) {
        String[] names = {"John", "alice", "Bob"};
        Arrays.sort(names,
Collections.reverseOrder(String.CASE_INSENSITIVE_ORDER));

        for (String name : names) {
            System.out.println(name);
        }
    }
}
```

9. Character Extraction Methods

9.1 `charAt(int index)`

Returns the character at the specified index.

```
String str = "Hello";
char c = str.charAt(1); // 'e'
```

9.2 `toCharArray()`

Converts the string into a new character array.

```
String str = "Hello";
char[] chars = str.toCharArray(); // ['H', 'e', 'l', 'l', 'o']
```

10. Modify a String

10.1 `substring()`

Extracts a substring from the string.

```
String str = "Hello";
String substr = str.substring(1, 3); // "el"
```

10.2 `concat()`

Concatenates two strings.

```
String str1 = "Hello";
String str2 = " World";
String result = str1.concat(str2); // "Hello World"
```

10.3 `replace()`

Replaces all occurrences of a character or substring.

```
String str = "Hello";
String replaced = str.replace('l', 'p'); // "Heppo"
```

10.4 `trim()`

Removes leading and trailing whitespace.

```
String str = "  Hello  ";
String trimmed = str.trim(); // "Hello"
```

11. Count Occurrences of a Character in a String

```
public class CountOccurrences {
    public static void main(String[] args) {
        String str = "hello";
        char ch = 'l';
        int count = 0;

        for (int i = 0; i < str.length(); i++) {
            if (str.charAt(i) == ch) {
                count++;
            }
        }

        System.out.println("Occurrences of " + ch + ": " + count);
    }
}
```

12. Difference between String and StringBuffer

- **Immutability**: `String` is immutable, while `StringBuffer` is mutable.

- **Thread-Safety**: `StringBuffer` is synchronized and thread-safe. `String` is immutable and inherently thread-safe.
- **Performance**: `StringBuffer` is faster for modifications because it does not create new objects.

13. Reverse() of StringBuffer with Example

```java
public class ReverseExample {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("Hello");
        sb.reverse();
        System.out.println(sb.toString()); // "olleH"
    }
}
```

These explanations and code snippets should provide a comprehensive understanding of the concepts and methods related to Java strings and `StringBuffer`.