Day 8

String Class & Methods

What is a String in Java?

- Strings are used for storing text.
- A String variable contains a collection of characters surrounded by double quotes:

There are two primary ways to declare a string in Java:

1. **Using String Literal:**

```
String s = "Hello, World!";
```

- In this method, the string is stored in a special memory area known as the **String Pool**.
- If a string with the same content already exists in the pool, the new reference points to the existing string.

2. **Using new Keyword:**

```
String s = new String("Hello, World!");
```

- This method explicitly creates a new String object in the **heap memory.**

Methods Available in String Class

1. length()

- **Description**: Returns the length (number of characters) of the string.
- **Example**:

```
String str = "Hello, World!";

int len = str.length(); // 13
```

2. charAt(int index)

- **Description**: Returns the character at the specified index.
- **Example**:

```
String str = "Hello";

char ch = str.charAt(1); // 'e'
```

3. substring(int beginIndex)

- **Description**: Returns a substring starting from the specified index to the end of the string.
- **Example**:

```
String str = "Hello, World!";

String subStr = str.substring(7); // "World!"
```

4. substring(int beginIndex, int endIndex)

- **Description**: Returns a substring starting from beginIndex to endIndex - 1.
- **Example**:

```
String str = "Hello, World!";
String subStr = str.substring(0, 5); // "Hello"
```

5. contains(CharSequence s)

- **Description**: Checks if the string contains the specified sequence of characters.
- **Example**:

```
String str = "Hello, World!";
boolean contains = str.contains("World"); // true
```

6. equals(Object anObject)

- **Description**: Compares the string to the specified object. Returns true if both have the same content.
- **Example**:

```
String str1 = "Hello";
String str2 = "Hello";
boolean isEqual = str1.equals(str2); // true
```

7. equalsIgnoreCase(String anotherString)

- **Description**: Compares the string to another string, ignoring case considerations.
- **Example**:

```
String str1 = "hello";
String str2 = "HELLO";
boolean isEqual = str1.equalsIgnoreCase(str2); // true
```

8. toLowerCase()

- **Description**: Converts all characters in the string to lowercase.
- **Example**:

```
String str = "Hello, World!";
String lowerStr = str.toLowerCase(); // "hello, world!"
```

9. toUpperCase()

- **Description**: Converts all characters in the string to uppercase.
- **Example**:

```
String str = "Hello, World!";

String upperStr = str.toUpperCase(); // "HELLO, WORLD!"
```

10. trim()

- **Description**: Removes leading and trailing white spaces from the string.
- **Example**:

```
String str = "  Hello, World!  ";

String trimmedStr = str.trim(); // "Hello, World!"
```

11. replace(char oldChar, char newChar)

- **Description**: Replaces all occurrences of the specified old character with the new character.
- **Example**:

```
String str = "Hello, World!";

String replacedStr = str.replace('o', 'a'); // "Hella, Warld!"
```

12. replace(CharSequence target, CharSequence replacement)

- **Description**: Replaces each substring of the string that matches the literal target sequence with the specified literal replacement sequence.
- **Example**:

```
String str = "Hello, World!";

String replacedStr = str.replace("World", "Java"); // "Hello, Java!"
```

13. split(String regex)

- **Description**: Splits the string around matches of the given regular expression.
- **Example**:

```
String str = "Hello, World!";

String[] words = str.split(", "); // ["Hello", "World!"]
```

14. indexOf(int ch)

- **Description**: Returns the index of the first occurrence of the specified character.
- **Example**:

```
String str = "Hello, World!";

int index = str.indexOf('o'); // 4
```

15. indexOf(String str)

- **Description**: Returns the index of the first occurrence of the specified substring.
- **Example**:

```
String str = "Hello, World!";
int index = str.indexOf("World"); // 7
```

16. startsWith(String prefix)

- **Description**: Checks if the string starts with the specified prefix.
- **Example**:

```
String str = "Hello, World!";
boolean starts = str.startsWith("Hello"); // true
```

17. endsWith(String suffix)

- **Description**: Checks if the string ends with the specified suffix.
- **Example**:

```
String str = "Hello, World!";
boolean ends = str.endsWith("World!"); // true
```

18. isEmpty()

- **Description**: Checks if the string is empty (length() == 0).
- **Example**:

```
String str = "";
boolean empty = str.isEmpty(); // true
```

19. toCharArray()

- **Description**: Converts the string into a new character array.
- **Example**:

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

20. valueOf(Object obj)

- **Description**: Returns the string representation of the specified object.
- **Example**:

```
int num = 123;
String str = String.valueOf(num); // "123"
```

21. concat(Sting obj)

- **Description**: Concatenates or join two strings.
- **Example**:

```
String str1 =”Welcome”;

String str2=” to java”;

String concatinatesStr = str1.concat(str2); // "Welcome to java"
```

Strings Comparison

In Java, we can declare string in 2 different ways.

1. **String s = "welcome";**
2. **String s = new String("welcome");**

both create a string with the value "welcome", but they do so in different ways:

1. String s = "welcome";

- **String Pool:** This statement creates a string in the **String Pool**, a special area in memory where Java stores unique string literals.
- **Reuse:** If another string with the same value "welcome" already exists in the pool, this statement will not create a new string but will instead point s to the existing one.

2. String s = new String("welcome");

- **Heap Memory:** This statement creates a new String object in the **heap memory** even if "welcome" already exists in the String Pool.
- **No Reuse:** This approach does not reuse the string from the pool; it explicitly creates a new object.

Difference between == and equals ()

- **== checks if the two strings are the same object in memory.**
- **equals () checks if the two strings have the same content.**

In most cases, when you want to compare the content of strings, you should use the equals() method.