

Lesson 02 Demo 06

Using String Methods

Objective: To explore methods available with string data type

Tools required: Eclipse IDE

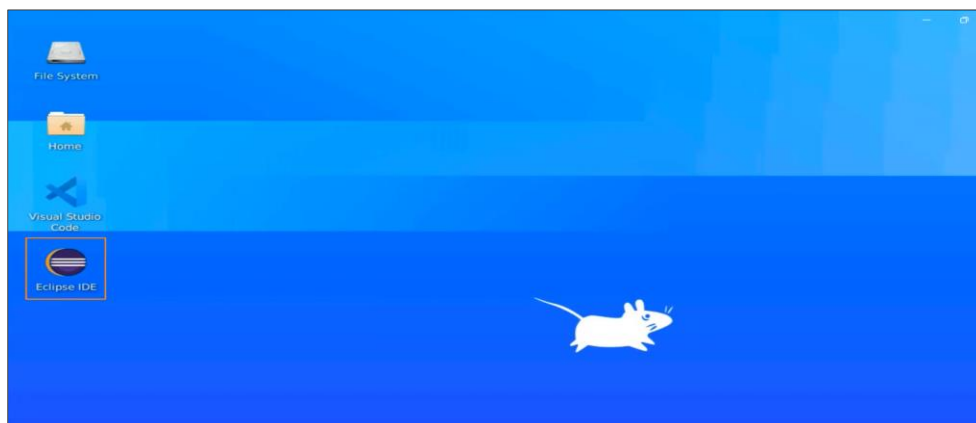
Prerequisites: None

Steps to be followed:

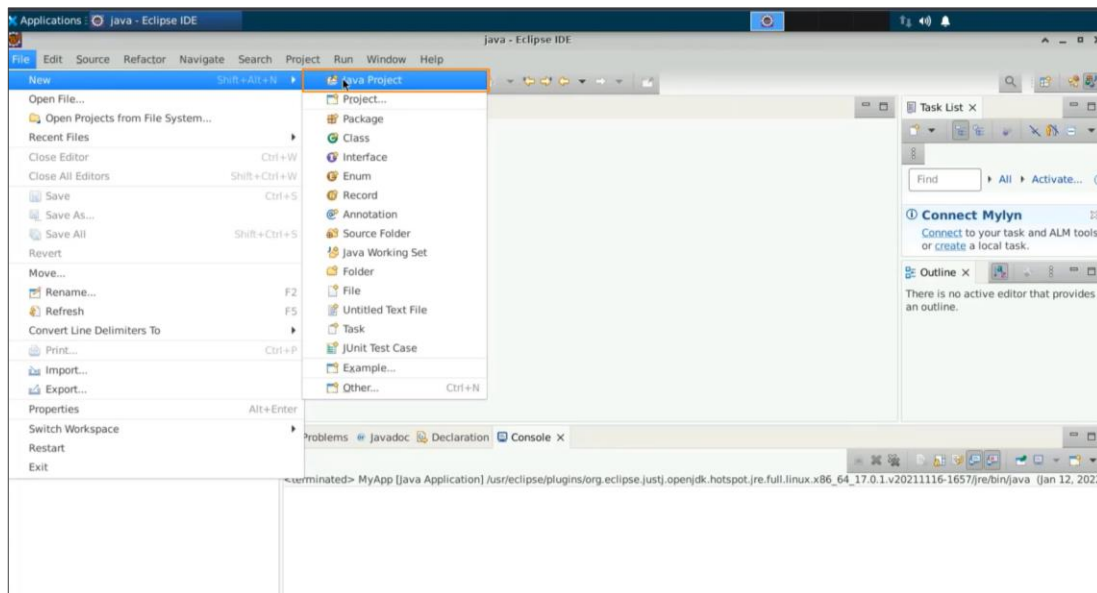
1. Write a string and calculate the number of characters in the string
2. Get the index of characters
3. Use the uppercase method to convert the string to uppercase
4. Implement string slicing and extracting sub-strings
5. Use the trim function on strings
6. Implement the concat method for concatenation of strings
7. Validate the strings using the methods ends, contains ends with, and starts with
8. Implement an algorithm to know the number of times a character appears
9. Implement String comparison and intern strings
10. Use the **method.equals** method

Step 1: Write a string and calculate the number of characters in the string

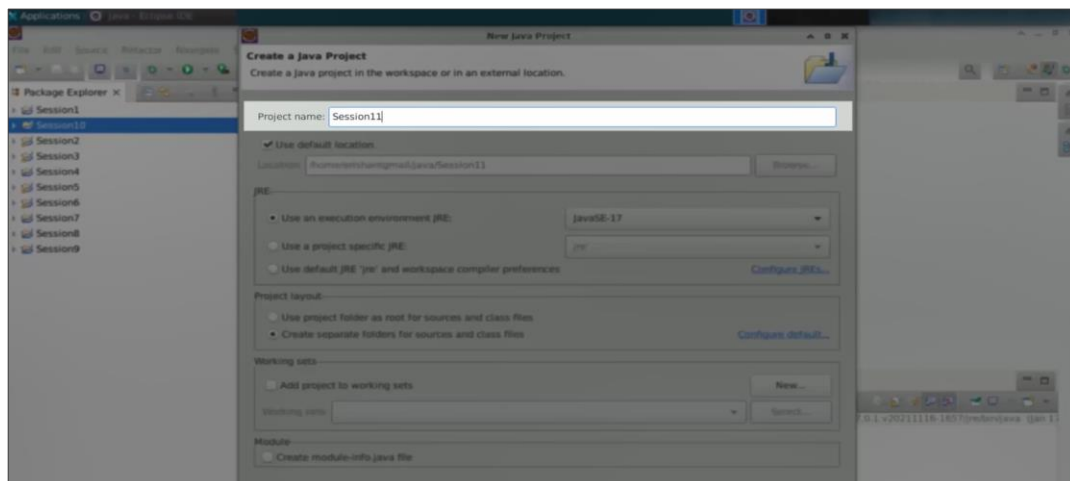
1.1 Open the Eclipse IDE



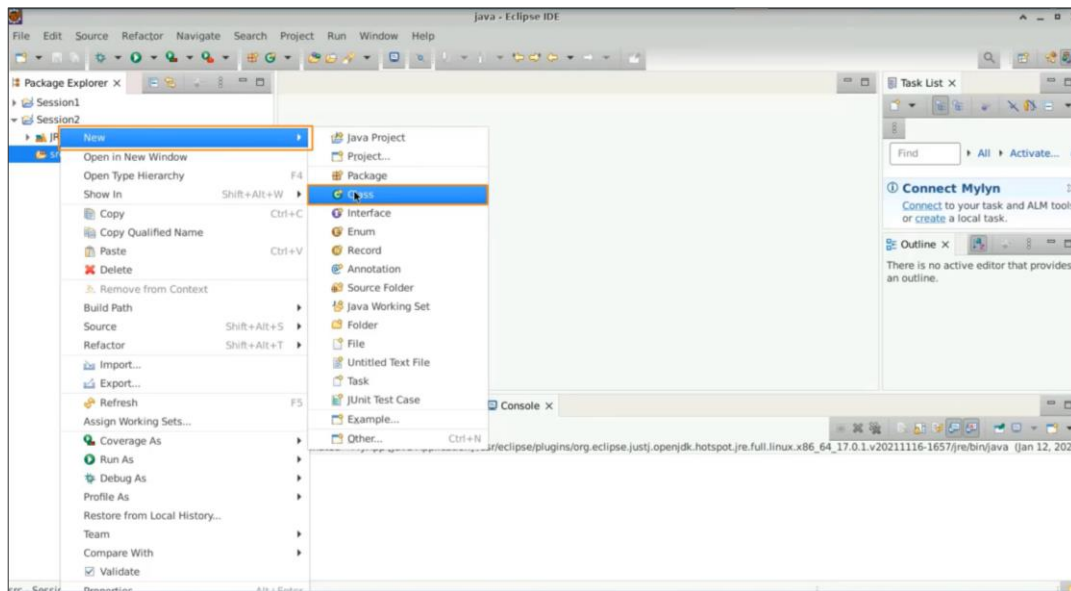
1.2. Select **File**, then **New**, and then **Java project**



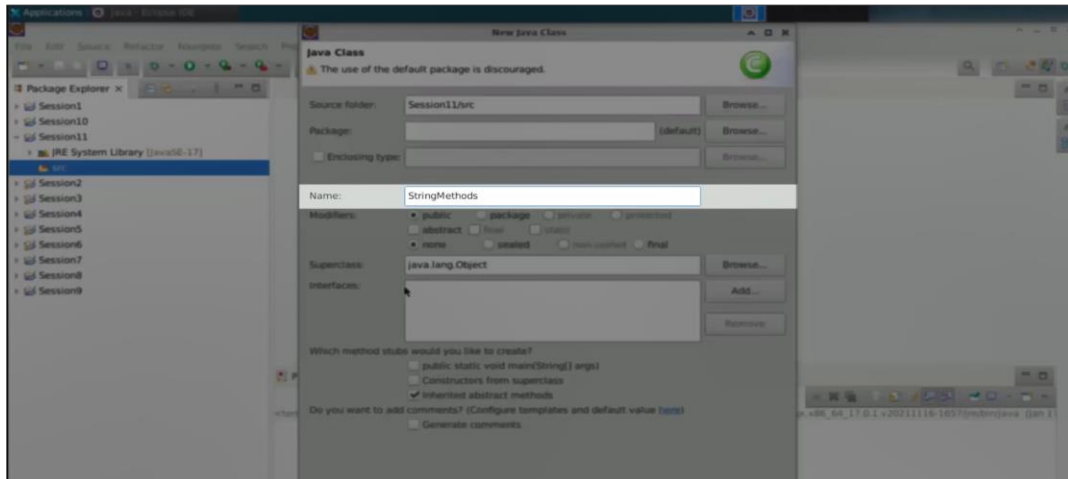
1.3 Name the project **“Session11”**, uncheck **“Create a module info dot Java file”**, and press **Finish**



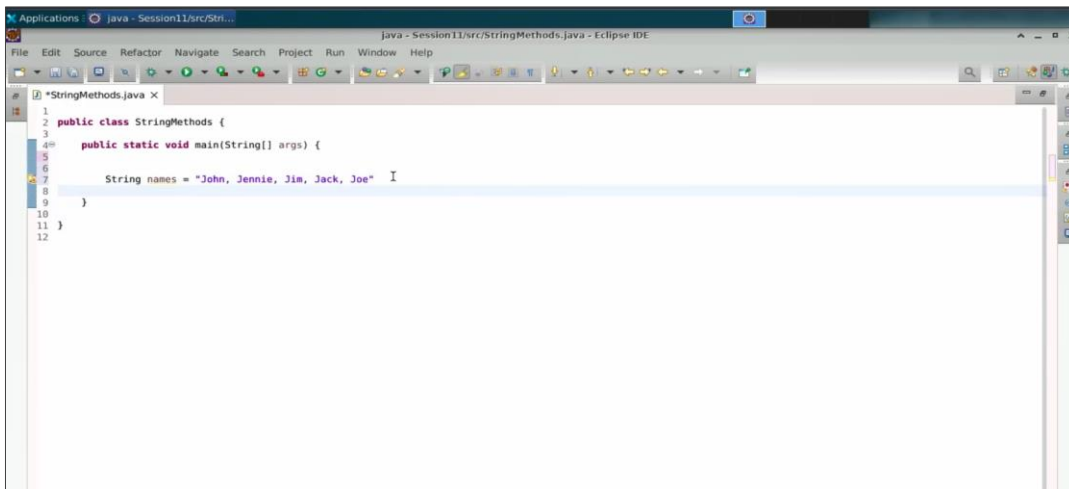
1.4 With a **Session11** on the src, do a right-click and create a **new class**



1.5 Name this class as an **StringMethods**, then select the **main method**, and then select **finish**

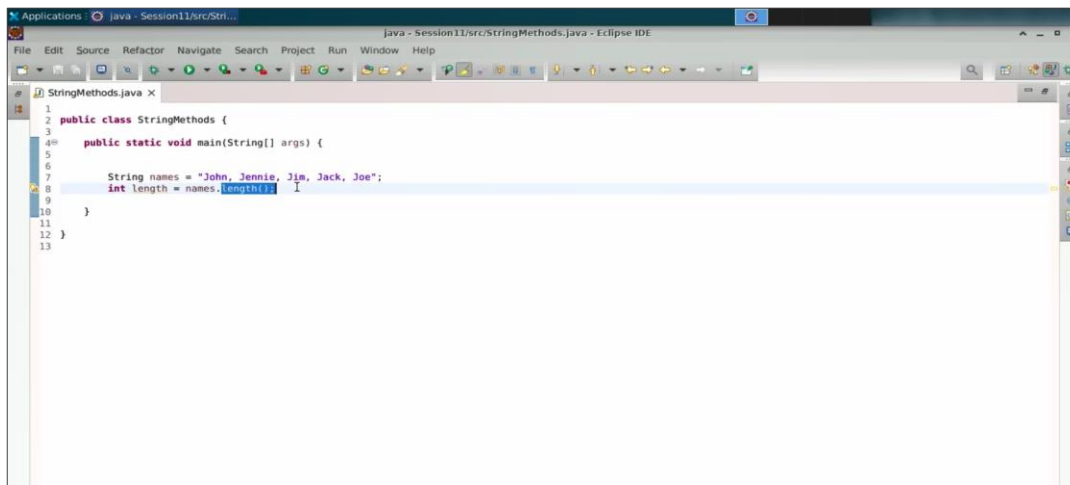


1.6 Let us write one string called names and these names are John, Jenny, Jim, Jack, and Joe. This is altogether a single string



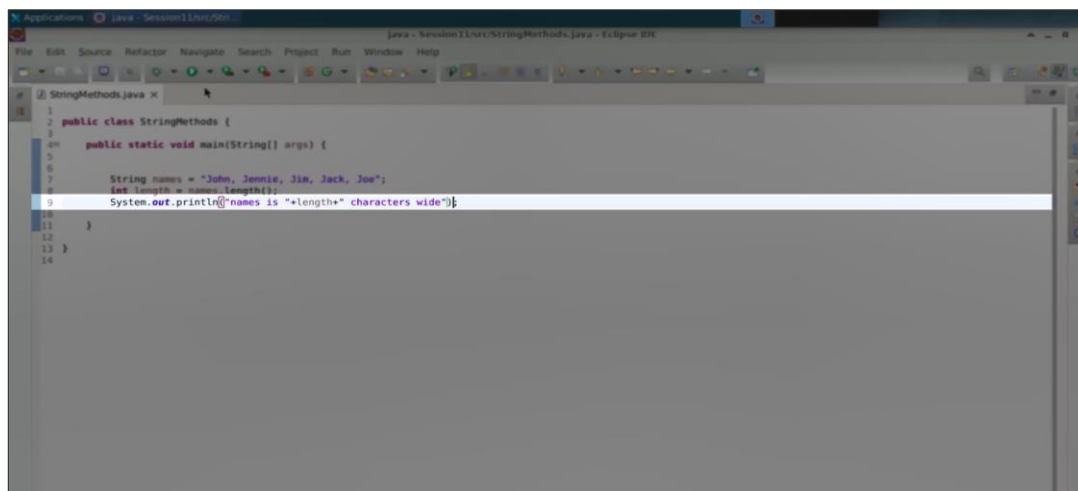
```
1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5         String names = "John, Jenny, Jim, Jack, Joe"
6     }
7 }
```

1.7 Add **int length = names.length();** to execute a function called length. The length method is available as a built-in method



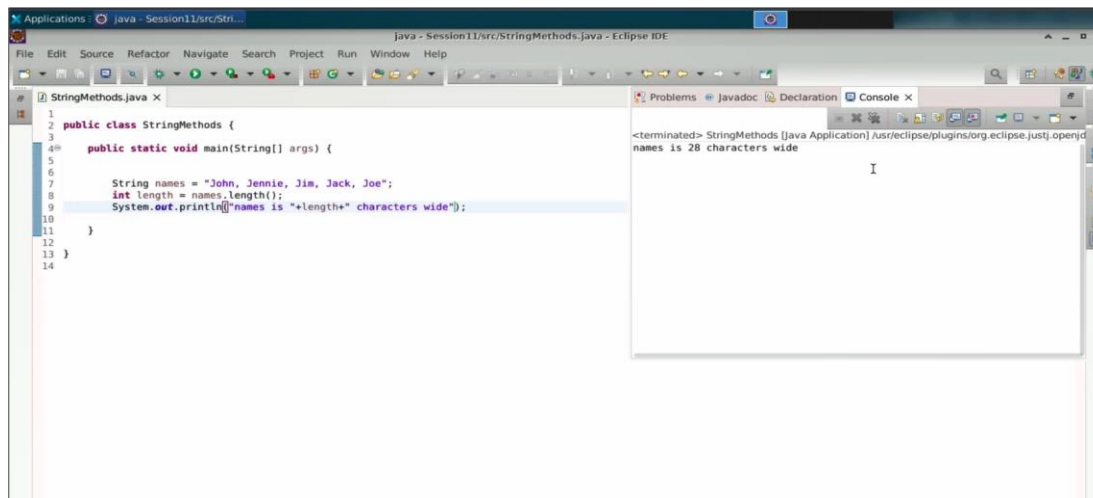
```
1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5         String names = "John, Jenny, Jim, Jack, Joe";
6         int length = names.length();
7     }
8 }
```

1.8 Add `System.out.println("Names are " + length + " characters wide.");`



```
1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5
6         String names = "John, Jennie, Jim, Jack, Joe";
7         int length = names.length();
8         System.out.println("Names are " + length + " characters wide.");
9     }
10 }
11
12
13
14
```

1.9 Run the code. It says that name is 28 characters wide

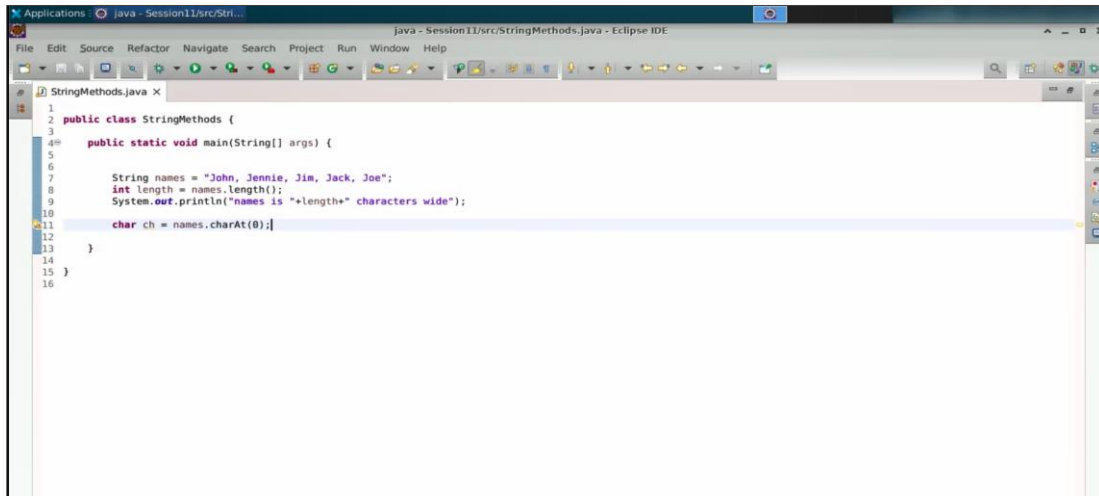


```
1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5
6         String names = "John, Jennie, Jim, Jack, Joe";
7         int length = names.length();
8         System.out.println("Names are " + length + " characters wide.");
9     }
10 }
11
12
13
14
```

<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.just.opened
Names are 28 characters wide

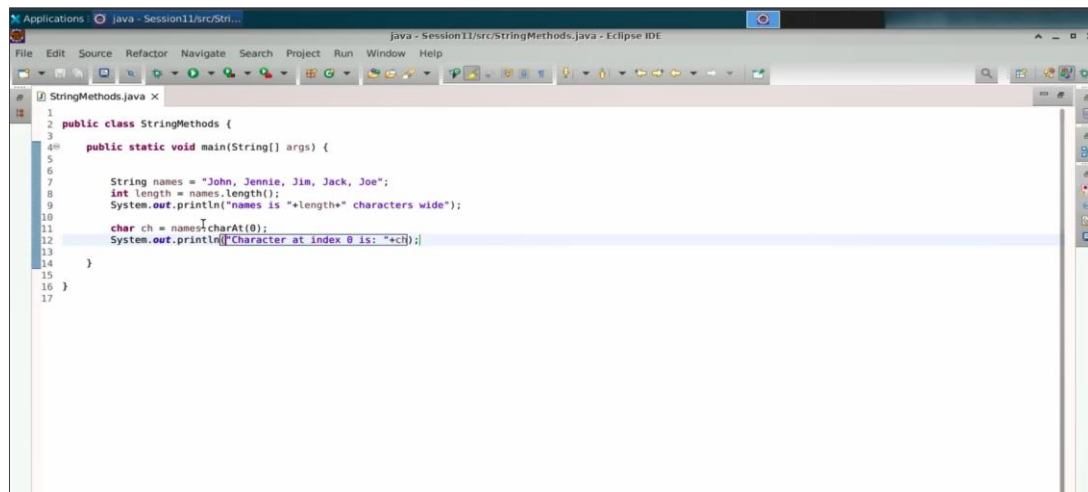
Step 2: Get the index of characters

2.1 A char called Ch is going to be named dot character at the index number 0



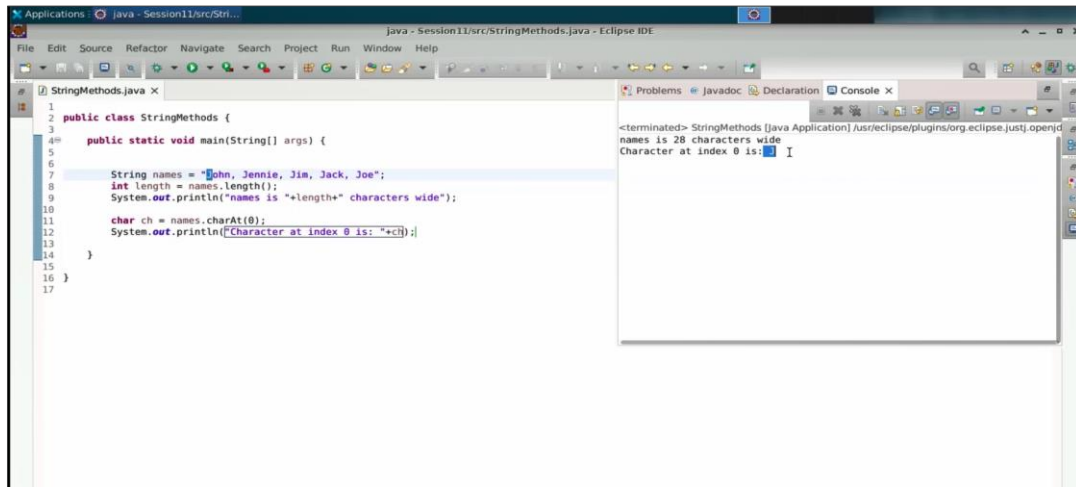
```
1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5
6         String names = "John, Jennie, Jim, Jack, Joe";
7         int length = names.length();
8         System.out.println("names is "+length+" characters wide");
9
10        char ch = names.charAt(0);
11    }
12
13
14
15 }
16 }
```

2.2 Now the character at index number 0, is very much clear, let us say character at Index zero is plus Ch. When you say char, this is a method with the help of which you can get the character from a string



```
1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5
6         String names = "John, Jennie, Jim, Jack, Joe";
7         int length = names.length();
8         System.out.println("names is "+length+" characters wide");
9
10        char ch = names.charAt(0);
11        System.out.println("Character at index 0 is: "+ch);
12    }
13
14
15 }
16 }
17 }
```

2.3 The 0 character is this upper-case J and when you run the program here you get to see it gives the value as J



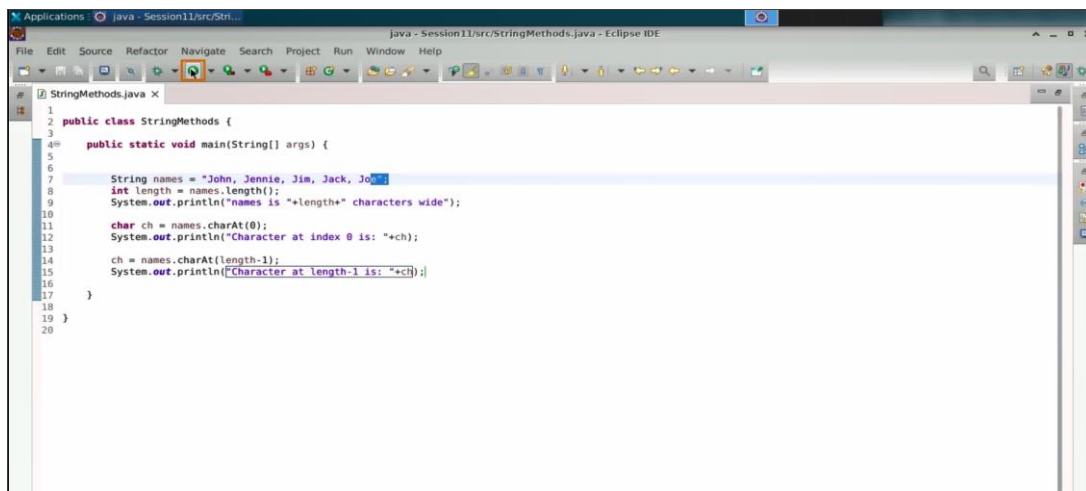
The screenshot shows the Eclipse IDE with a Java project named 'java - Session11/src/Str...'. The editor displays the file 'StringMethods.java' with the following code:

```
1 public class StringMethods {  
2  
3     public static void main(String[] args) {  
4  
5  
6  
7         String names = "John, Jennie, Jim, Jack, Joe";  
8         int length = names.length();  
9         System.out.println("names is "+length+" characters wide");  
10  
11         char ch = names.charAt(0);  
12         System.out.println("Character at index 0 is: "+ch);  
13  
14     }  
15  
16 }  
17
```

The console output on the right shows the following messages:

```
<terminated> StringMethods (Java Application) Aus/eclipse/plugins/org.eclipse.justj.openjdk  
names is 28 characters wide  
Character at index 0 is: J
```

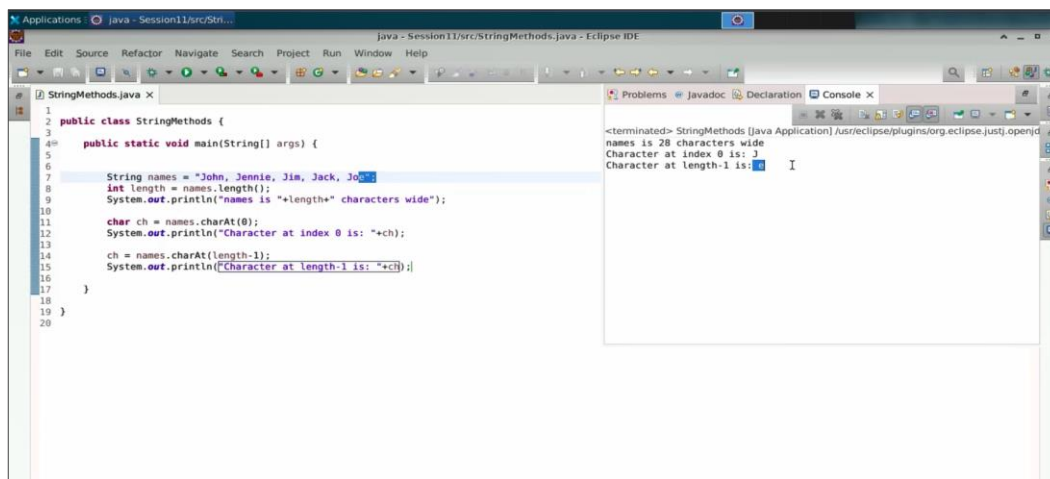
2.4 Let us also say `ch = names.charAt(length - 1)`. Then add `System.out.println("Character at length - 1 is " + ch);`;



The screenshot shows the Eclipse IDE with the same Java project. The editor displays the updated 'StringMethods.java' file with the following code:

```
1 public class StringMethods {  
2  
3     public static void main(String[] args) {  
4  
5  
6  
7         String names = "John, Jennie, Jim, Jack, Joe";  
8         int length = names.length();  
9         System.out.println("names is "+length+" characters wide");  
10  
11         char ch = names.charAt(0);  
12         System.out.println("Character at index 0 is: "+ch);  
13  
14         ch = names.charAt(length-1);  
15         System.out.println("Character at length-1 is: "+ch);  
16  
17     }  
18  
19 }  
20
```

2.5 When you run the code, you will see that the character at **length - 1** is **e**



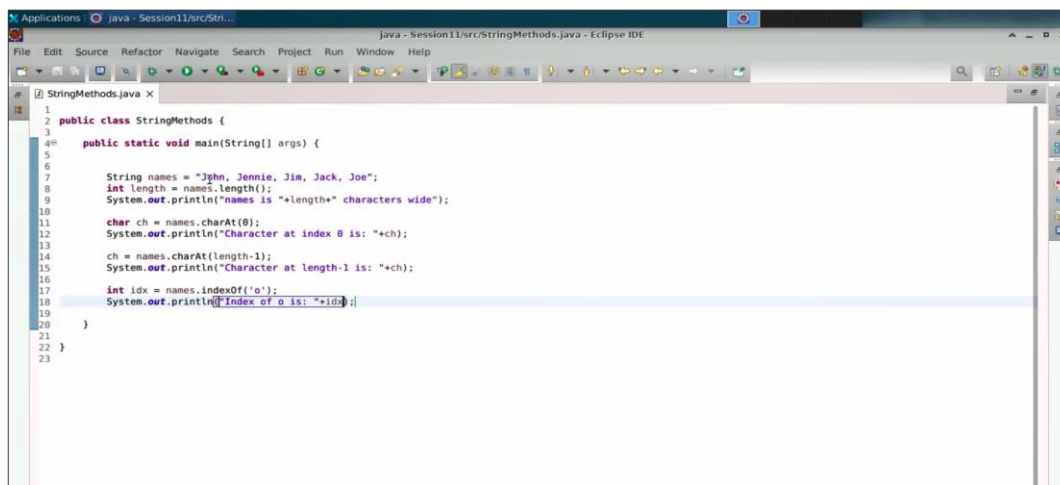
The screenshot shows the Eclipse IDE with a Java file named `StringMethods.java`. The code defines a `StringMethods` class with a `main` method. It initializes a string `names = "John, Jennie, Jim, Jack, Joe"`, calculates its length, and prints it. It then prints the character at index 0 and the character at `length-1`. The console output on the right shows the execution results: `names is 28 characters wide`, `Character at index 0 is: J`, and `Character at length-1 is: e`.

```
1 public class StringMethods {
2
3
4     public static void main(String[] args) {
5
6         String names = "John, Jennie, Jim, Jack, Joe";
7         int length = names.length();
8         System.out.println("names is "+length+" characters wide");
9
10        char ch = names.charAt(0);
11        System.out.println("Character at index 0 is: "+ch);
12
13        ch = names.charAt(length-1);
14        System.out.println("Character at length-1 is: "+ch);
15    }
16 }
17
18
19
20
```

Console Output:

```
<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.openj
names is 28 characters wide
Character at index 0 is: J
Character at length-1 is: e
```

2.6 Type **`idx = name.indexOf('o');`**. Let us see the index of a character called 'o'. The character 'o' is available at the first index as well as the second-to-last index. The **`indexOf`** method will give you the index of the character from the beginning. So, when you type **`indexOf('o')`**, it will give you the value 1



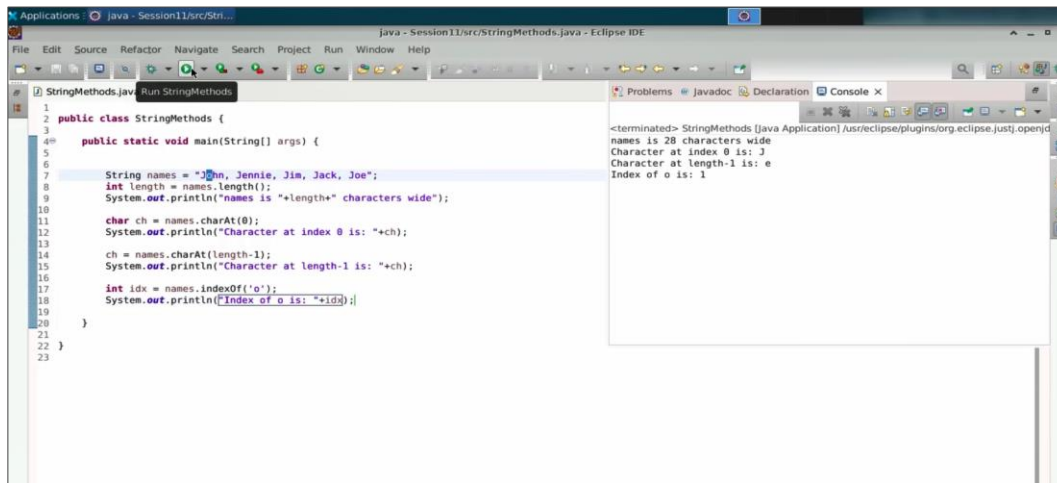
The screenshot shows the Eclipse IDE with the same `StringMethods.java` file. The code is updated to include the `indexOf` method. It initializes the string `names = "John, Jennie, Jim, Jack, Joe"`, calculates its length, and prints it. It then prints the character at index 0 and the character at `length-1`. Finally, it prints the index of the first occurrence of the character 'o' using `indexOf('o')`. The console output on the right shows the execution results: `names is 28 characters wide`, `Character at index 0 is: J`, `Character at length-1 is: e`, and `Index of o is: 1`.

```
1 public class StringMethods {
2
3
4     public static void main(String[] args) {
5
6         String names = "John, Jennie, Jim, Jack, Joe";
7         int length = names.length();
8         System.out.println("names is "+length+" characters wide");
9
10        char ch = names.charAt(0);
11        System.out.println("Character at index 0 is: "+ch);
12
13        ch = names.charAt(length-1);
14        System.out.println("Character at length-1 is: "+ch);
15
16        int idx = names.indexOf('o');
17        System.out.println("Index of o is: "+idx);
18    }
19 }
20
21
22
23
```

Console Output:

```
<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.openj
names is 28 characters wide
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
```


2.7 Run the code and you can see an index of o is 1

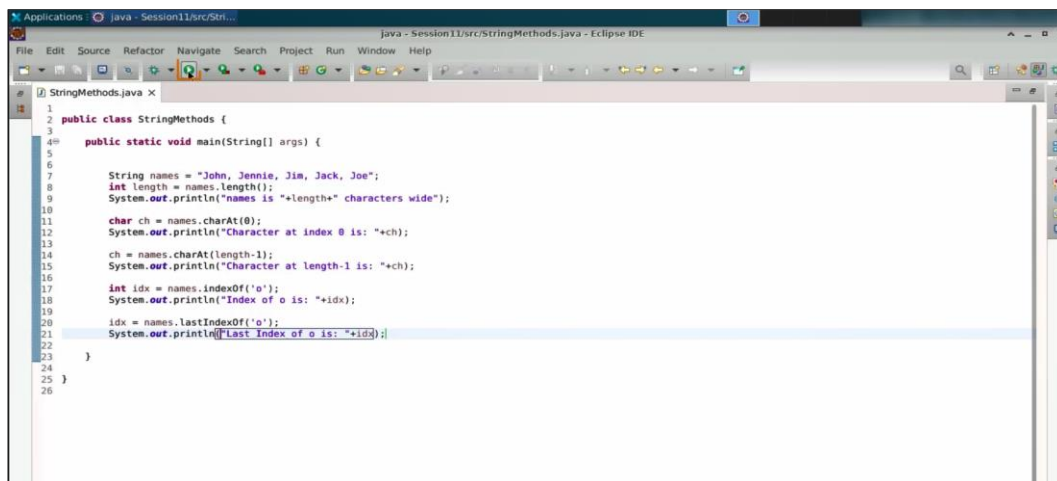


```
1 public class StringMethods {
2
3
4     public static void main(String[] args) {
5
6
7         String names = "John, Jennie, Jim, Jack, Joe";
8         int length = names.length();
9         System.out.println("names is "+length+" characters wide");
10
11         char ch = names.charAt(0);
12         System.out.println("Character at index 0 is: "+ch);
13
14         ch = names.charAt(length-1);
15         System.out.println("Character at length-1 is: "+ch);
16
17         int idx = names.indexOf('o');
18         System.out.println("Index of o is: "+idx);
19
20     }
21 }
22
23
```

Console Output:

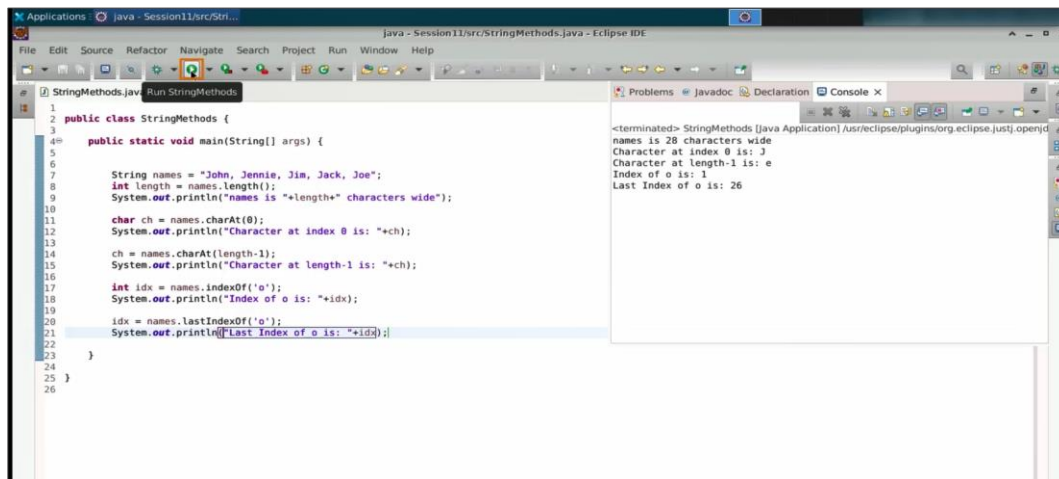
```
<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.openj
names is 28 characters wide
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
```

2.8 You can also add `idx = names.lastIndexOf('o');`. There is another method called `lastIndexOf`, which will give you the index of the character from the end of the string



```
1 public class StringMethods {
2
3
4     public static void main(String[] args) {
5
6
7         String names = "John, Jennie, Jim, Jack, Joe";
8         int length = names.length();
9         System.out.println("names is "+length+" characters wide");
10
11         char ch = names.charAt(0);
12         System.out.println("Character at index 0 is: "+ch);
13
14         ch = names.charAt(length-1);
15         System.out.println("Character at length-1 is: "+ch);
16
17         int idx = names.indexOf('o');
18         System.out.println("Index of o is: "+idx);
19
20         idx = names.lastIndexOf('o');
21         System.out.println("Last index of o is: "+idx);
22
23     }
24 }
25
26
```

2.9 Run this code: **length - 2**. The length is 28, so the index of 'O' is 28 - 2, which is 26



```

1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5         String names = "John, Jennie, Jim, Jack, Joe";
6         int length = names.length();
7         System.out.println("names is "+length+" characters wide");
8
9         char ch = names.charAt(0);
10        System.out.println("Character at index 0 is: "+ch);
11
12        ch = names.charAt(length-1);
13        System.out.println("Character at length-1 is: "+ch);
14
15        int idx = names.indexOf('o');
16        System.out.println("Index of o is: "+idx);
17
18        idx = names.lastIndexOf('o');
19        System.out.println("Last Index of o is: "+idx);
20    }
21
22 }
23
24
25
26

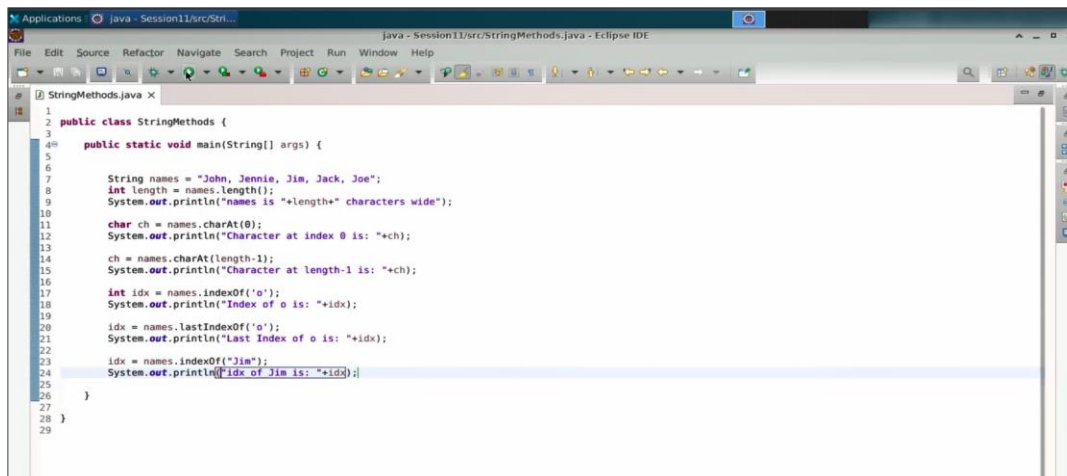
```

```

<terminated> StringMethods [Java Application]
names is 28 characters wide
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26

```

2.10 Use **idx = name.indexOf("Jim")**; instead of a character. You can pass a string, such as "Jim". The index will be where "Jim" starts. You are going to add the index of "Jim" to the existing index



```

1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5         String names = "John, Jennie, Jim, Jack, Joe";
6         int length = names.length();
7         System.out.println("names is "+length+" characters wide");
8
9         char ch = names.charAt(0);
10        System.out.println("Character at index 0 is: "+ch);
11
12        ch = names.charAt(length-1);
13        System.out.println("Character at length-1 is: "+ch);
14
15        int idx = names.indexOf('o');
16        System.out.println("Index of o is: "+idx);
17
18        idx = names.lastIndexOf('o');
19        System.out.println("Last Index of o is: "+idx);
20
21        idx = names.indexOf("Jim");
22        System.out.println("Idx of Jim is: "+idx);
23    }
24
25 }
26
27
28
29

```

2.11 Run the code. Now, this is where Jim's J starts, so that's like 14.

```

1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5         String names = "John, Jennie, Jim, Jack, Joe";
6         int length = names.length();
7         System.out.println("names is "+length+" characters wide");
8
9         char ch = names.charAt(0);
10        System.out.println("Character at index 0 is: "+ch);
11
12        ch = names.charAt(length-1);
13        System.out.println("Character at length-1 is: "+ch);
14
15        int idx = names.indexOf('o');
16        System.out.println("Index of o is: "+idx);
17
18        idx = names.lastIndexOf('o');
19        System.out.println("Last Index of o is: "+idx);
20
21        idx = names.indexOf("Jim");
22        System.out.println("Idx of Jim is: "+idx);
23    }
24 }
25
26
27
28
29

```

```

<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.openj
names is 28 characters wide
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
Idx of Jim is: 14

```

Step 3: Use the uppercase method to convert the string to uppercase

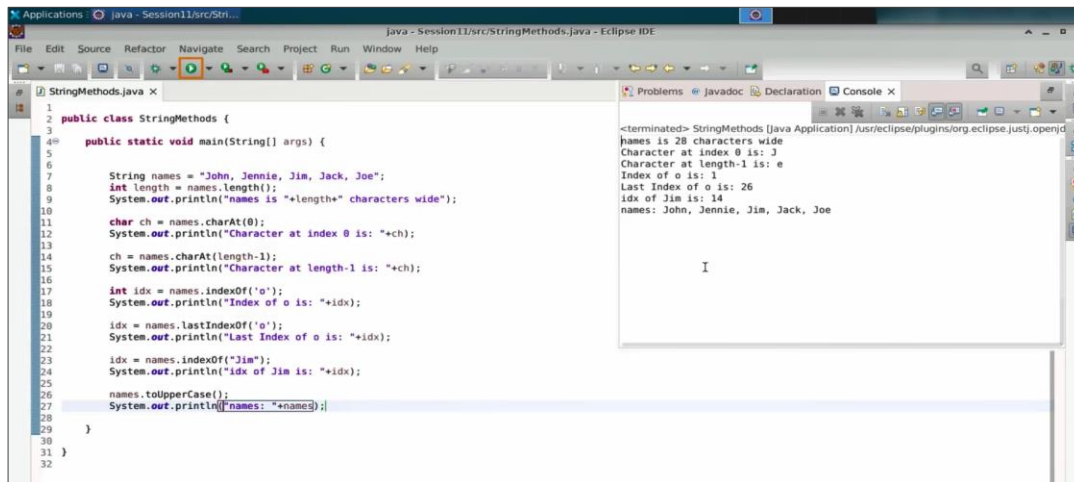
3.1 Type **names.toUpperCase()**. The method **toUpperCase** is very clear in its function; it converts the string to uppercase. Use the print command to print the original names variable followed by the uppercase version

```

1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5         String names = "John, Jennie, Jim, Jack, Joe";
6         int length = names.length();
7         System.out.println("names is "+length+" characters wide");
8
9         char ch = names.charAt(0);
10        System.out.println("Character at index 0 is: "+ch);
11
12        ch = names.charAt(length-1);
13        System.out.println("Character at length-1 is: "+ch);
14
15        int idx = names.indexOf('o');
16        System.out.println("Index of o is: "+idx);
17
18        idx = names.lastIndexOf('o');
19        System.out.println("Last Index of o is: "+idx);
20
21        idx = names.indexOf("Jim");
22        System.out.println("Idx of Jim is: "+idx);
23
24        names.toUpperCase();
25        System.out.println("names: "+names);
26    }
27 }
28
29

```

3.2 Run the program here, you will get to see the names are still the same. There are some methods that are meant to do the manipulation on those strings



```

1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5         String names = "John, Jennie, Jim, Jack, Joe";
6         int length = names.length();
7         System.out.println("names is "+length+" characters wide");
8
9         char ch = names.charAt(0);
10        System.out.println("Character at index 0 is: "+ch);
11
12        ch = names.charAt(length-1);
13        System.out.println("Character at length-1 is: "+ch);
14
15        int idx = names.indexOf('o');
16        System.out.println("Index of o is: "+idx);
17
18        idx = names.lastIndexOf('o');
19        System.out.println("Last Index of o is: "+idx);
20
21        idx = names.indexOf("Jim");
22        System.out.println("idx of Jim is: "+idx);
23
24        names.toUpperCase();
25        System.out.println("names: "+names);
26    }
27 }
28
29
30
31
32

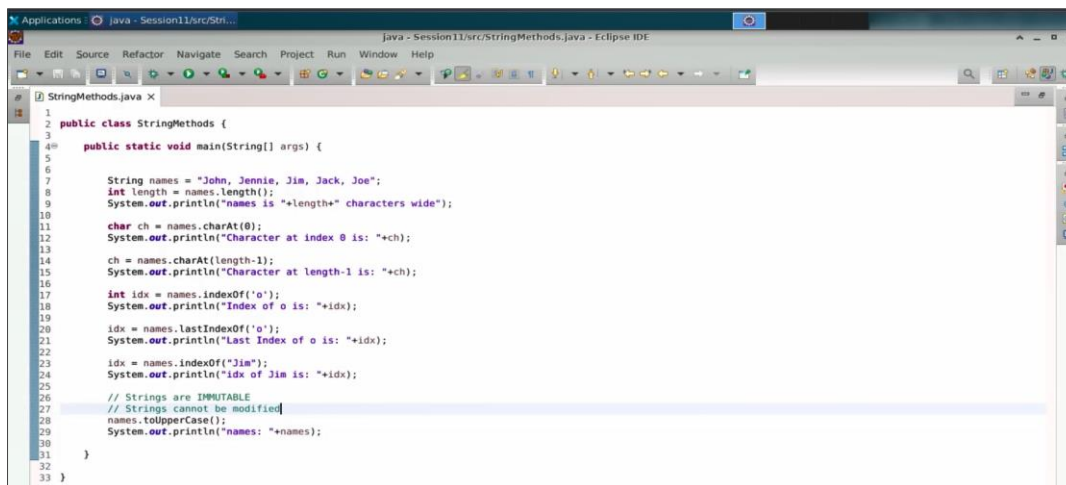
```

```

<terminated> StringMethods [java Application] Jsr/eclipse/plugins/org.eclipse.justi.openid
names is 28 characters wide
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe

```

3.3 Now here you need to understand the fundamental, which is strings are immutable. Immutable means that strings cannot be modified



```

1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5         String names = "John, Jennie, Jim, Jack, Joe";
6         int length = names.length();
7         System.out.println("names is "+length+" characters wide");
8
9         char ch = names.charAt(0);
10        System.out.println("Character at index 0 is: "+ch);
11
12        ch = names.charAt(length-1);
13        System.out.println("Character at length-1 is: "+ch);
14
15        int idx = names.indexOf('o');
16        System.out.println("Index of o is: "+idx);
17
18        idx = names.lastIndexOf('o');
19        System.out.println("Last Index of o is: "+idx);
20
21        idx = names.indexOf("Jim");
22        System.out.println("idx of Jim is: "+idx);
23
24        // Strings are IMMUTABLE
25        // Strings cannot be modified
26        names.toUpperCase();
27        System.out.println("names: "+names);
28    }
29 }
30
31
32
33
34

```

- 3.4 When you type **names.toUpperCase()**, it creates a new string in memory and returns it. Hence, you will have another string called **uppercaseNames** (or any name of your choice). If you print the string called **uppercaseNames**, it will display the converted string

```

1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5         String names = "John, Jennie, Jim, Jack, Joe";
6         int length = names.length();
7         System.out.println("names is "+length+" characters wide");
8
9         char ch = names.charAt(0);
10        System.out.println("Character at index 0 is: "+ch);
11
12        ch = names.charAt(length-1);
13        System.out.println("Character at length-1 is: "+ch);
14
15        int idx = names.indexOf('o');
16        System.out.println("Index of o is: "+idx);
17
18        idx = names.lastIndexOf('o');
19        System.out.println("Last Index of o is: "+idx);
20
21        idx = names.indexOf("Jim");
22        System.out.println("Idx of Jim is: "+idx);
23
24        // Strings are IMMUTABLE
25        // Strings cannot be modified
26        String upperCaseNames = names.toUpperCase();
27        System.out.println("names: "+names);
28        System.out.println("uppercaseNames: "+upperCaseNames);
29    }
30 }

```

- 3.5 Run this code what you see is that the upper-case names are basically containing the characters in the upper case, so this is a return string through to the uppercase method

```

1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5         String names = "John, Jennie, Jim, Jack, Joe";
6         int length = names.length();
7         System.out.println("names is "+length+" characters wide");
8
9         char ch = names.charAt(0);
10        System.out.println("Character at index 0 is: "+ch);
11
12        ch = names.charAt(length-1);
13        System.out.println("Character at length-1 is: "+ch);
14
15        int idx = names.indexOf('o');
16        System.out.println("Index of o is: "+idx);
17
18        idx = names.lastIndexOf('o');
19        System.out.println("Last Index of o is: "+idx);
20
21        idx = names.indexOf("Jim");
22        System.out.println("Idx of Jim is: "+idx);
23
24        // Strings are IMMUTABLE
25        // Strings cannot be modified
26        String upperCaseNames = names.toUpperCase();
27        System.out.println("names: "+names);
28        System.out.println("uppercaseNames: "+upperCaseNames);
29    }
30 }

```

Console Output:

```

names is 28 characters wide
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
Idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
uppercaseNames: JOHN, JENNIE, JIM, JACK, JOE

```

3.6 For a given context, you can even say, spring replaced is going to be named **‘replace’**. Let the old character j be replaced with a new character k. If you print out the names. You will find that names have no change. And when you add, replaced names, it prints down the replace string

```

1 public class StringMethods {
2     public static void main(String[] args) {
3
4         String names = "John, Jennie, Jim, Jack, Joe";
5         int length = names.length();
6         System.out.println("names is "+length+" characters wide");
7
8         char ch = names.charAt(0);
9         System.out.println("Character at index 0 is: "+ch);
10
11         ch = names.charAt(length-1);
12         System.out.println("Character at length-1 is: "+ch);
13
14         int idx = names.indexOf('o');
15         System.out.println("Index of o is: "+idx);
16
17         idx = names.lastIndexOf('o');
18         System.out.println("Last Index of o is: "+idx);
19
20         idx = names.indexOf("Jim");
21         System.out.println("idx of Jim is: "+idx);
22
23         // Strings are IMMUTABLE
24         // Strings cannot be modified
25         String upperCaseNames = names.toUpperCase();
26         System.out.println("names: "+names);
27         System.out.println("upperCaseNames: "+upperCaseNames);
28
29         String replaced = names.replace('J', 'K');
30         System.out.println("names: "+names);
31         System.out.println("replaced names: "+replaced);
32     }
33 }

```

3.7 Run this program, and you get to see the data is manipulated. But the names are not at all changed

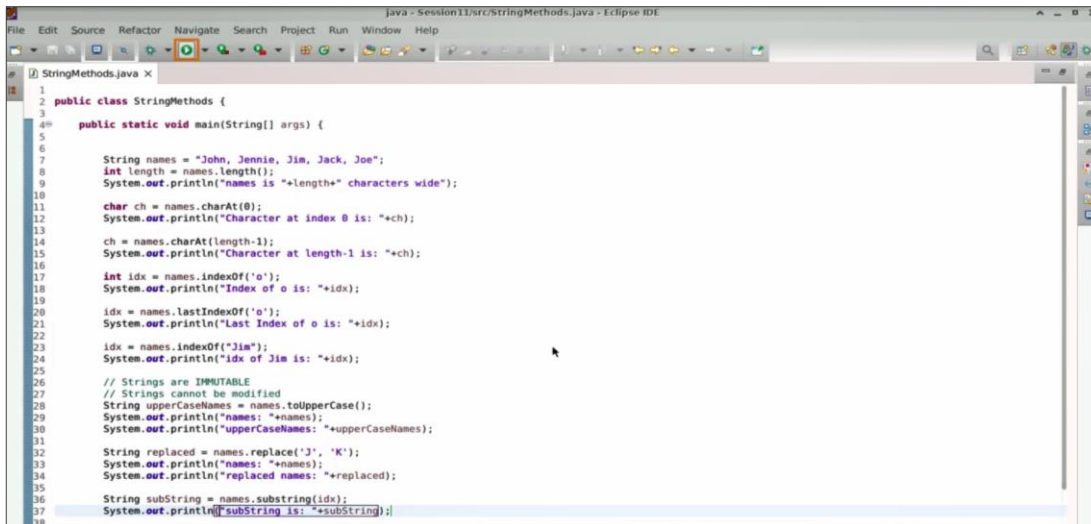
```

<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.justi.openjdk
names is 28 characters wide
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe

```


Step 4: Implement string slicing and extracting sub-strings

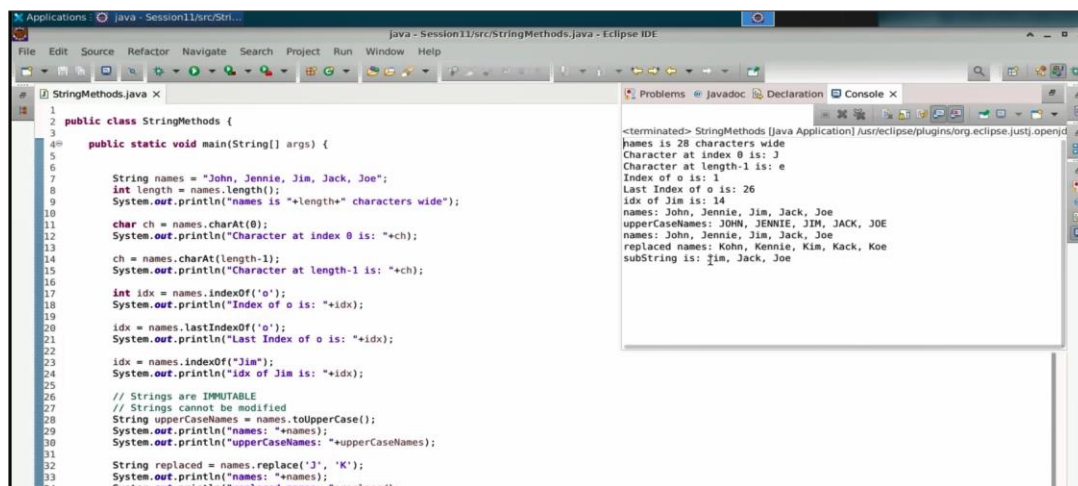
4.1 Now you are going to extract a substring from a string, which is called string slicing. Type **names.substring** and pass the starting index of 'Jim'. This will create a new string in memory. Let us name this substring 'substring' and store it at the specified index. Print out the substring by using **System.out.println("Substring is " + substring);**



```

1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5
6         String names = "John, Jennie, Jim, Jack, Joe";
7         int length = names.length();
8         System.out.println("names is "+length+" characters wide");
9
10        char ch = names.charAt(0);
11        System.out.println("Character at index 0 is: "+ch);
12
13        ch = names.charAt(length-1);
14        System.out.println("Character at length-1 is: "+ch);
15
16        int idx = names.indexOf('o');
17        System.out.println("Index of o is: "+idx);
18
19        idx = names.lastIndexOf('o');
20        System.out.println("Last Index of o is: "+idx);
21
22        idx = names.indexOf("Jim");
23        System.out.println("idx of Jim is: "+idx);
24
25        // Strings are IMMUTABLE
26        // Strings cannot be modified
27        String upperCaseNames = names.toUpperCase();
28        System.out.println("names: "+names);
29        System.out.println("upperCaseNames: "+upperCaseNames);
30
31        String replaced = names.replace('J', 'K');
32        System.out.println("names: "+names);
33        System.out.println("replaced names: "+replaced);
34
35        String subString = names.substring(idx);
36        System.out.println("subString is: "+subString);
37    }
38 }
  
```

4.2 Run the code. A substring with one single input will simply slice your string from that index till the end



```

1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5
6         String names = "John, Jennie, Jim, Jack, Joe";
7         int length = names.length();
8         System.out.println("names is "+length+" characters wide");
9
10        char ch = names.charAt(0);
11        System.out.println("Character at index 0 is: "+ch);
12
13        ch = names.charAt(length-1);
14        System.out.println("Character at length-1 is: "+ch);
15
16        int idx = names.indexOf('o');
17        System.out.println("Index of o is: "+idx);
18
19        idx = names.lastIndexOf('o');
20        System.out.println("Last Index of o is: "+idx);
21
22        idx = names.indexOf("Jim");
23        System.out.println("idx of Jim is: "+idx);
24
25        // Strings are IMMUTABLE
26        // Strings cannot be modified
27        String upperCaseNames = names.toUpperCase();
28        System.out.println("names: "+names);
29        System.out.println("upperCaseNames: "+upperCaseNames);
30
31        String replaced = names.replace('J', 'K');
32        System.out.println("names: "+names);
33        System.out.println("replaced names: "+replaced);
34
35        String subString = names.substring(idx);
36        System.out.println("subString is: "+subString);
37    }
38 }
  
```

```

<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.justi.open...
names is 28 characters wide
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim, Jack, Joe
  
```

4.3 Now try to write one more usage of this substring method, which is an overloaded method, you can pass the index, and then you can pass a number called 3

```

1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5
6         String names = "John, Jennie, Jim, Jack, Joe";
7         int length = names.length();
8         System.out.println("names is "+length+" characters wide");
9
10        char ch = names.charAt(0);
11        System.out.println("Character at index 0 is: "+ch);
12
13        ch = names.charAt(length-1);
14        System.out.println("Character at length-1 is: "+ch);
15
16        int idx = names.indexOf('o');
17        System.out.println("Index of o is: "+idx);
18
19        idx = names.lastIndexOf('o');
20        System.out.println("Last Index of o is: "+idx);
21
22        idx = names.indexOf("Jim");
23        System.out.println("idx of Jim is: "+idx);
24
25        // Strings are IMMUTABLE
26        // Strings cannot be modified
27        String upperCaseNames = names.toUpperCase();
28        System.out.println("names: "+names);
29        System.out.println("upperCaseNames: "+upperCaseNames);
30
31        String replaced = names.replace('J', 'K');
32        System.out.println("names: "+names);
33        System.out.println("replaced names: "+replaced);
34
35        //String subString = names.substring(idx);
36        String subString = names.substring(idx, 3);
37        System.out.println("subString is: "+subString);
38    }
39 }

```

4.4 When you run this code. You get to see that there is an error. Now the index is the beginning point and the second input is the ending point

```

1 public class StringMethods {
2
3     public static void main(String[] args) {
4
5
6         String names = "John, Jennie, Jim, Jack, Joe";
7         int length = names.length();
8         System.out.println("names is "+length+" characters wide");
9
10        char ch = names.charAt(0);
11        System.out.println("Character at index 0 is: "+ch);
12
13        ch = names.charAt(length-1);
14        System.out.println("Character at length-1 is: "+ch);
15
16        int idx = names.indexOf('o');
17        System.out.println("Index of o is: "+idx);
18
19        idx = names.lastIndexOf('o');
20        System.out.println("Last Index of o is: "+idx);
21
22        idx = names.indexOf("Jim");
23        System.out.println("idx of Jim is: "+idx);
24
25        // Strings are IMMUTABLE
26        // Strings cannot be modified
27        String upperCaseNames = names.toUpperCase();
28        System.out.println("names: "+names);
29        System.out.println("upperCaseNames: "+upperCaseNames);
30
31        String replaced = names.replace('J', 'K');
32        System.out.println("names: "+names);
33        System.out.println("replaced names: "+replaced);
34
35        //String subString = names.substring(idx);
36        String subString = names.substring(idx, 3);
37        System.out.println("subString is: "+subString);
38    }
39 }

```

Exception in thread "main" java.lang.StringIndexOutOfBoundsException: b-31
 at java.base/java.lang.String.checkBoundsBeginEnd(String.java:461)
 at java.base/java.lang.String.substring(String.java:2704)
 at StringMethods.main(StringMethods.java:37)

4.5 The index of Jim over gives the output as 14, which means the next index cannot be less than 14. It must be larger than. You're going to add 17

```

17 int idx = names.indexOf('o');
18 System.out.println("Index of o is: "+idx);
19
20 idx = names.lastIndexOf('o');
21 System.out.println("Last Index of o is: "+idx);
22
23 idx = names.indexOf("Jim");
24 System.out.println("idx of Jim is: "+idx);
25
26 // Strings are IMMUTABLE
27 // Strings cannot be modified
28 String upperCaseNames = names.toUpperCase();
29 System.out.println("names: "+names);
30 System.out.println("upperCaseNames: "+upperCaseNames);
31
32 String replaced = names.replace('J', 'K');
33 System.out.println("names: "+names);
34 System.out.println("replaced names: "+replaced);
35
36 //String subString = names.substring(idx);
37 String subString = names.substring(idx, 17);
38 System.out.println("subString is: "+subString);
39
<terminated> StringMethods [Java Application] Aus/clipse/plugins/org.eclipse.justi.openjdk.hotspot.jre.full/linux-x86_64/17.0.1.v20211116-1657/jre/bin/java
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: begin 14, end 3, length 26
    at java.base/java.lang.String.checkBoundsBeginEnd(String.java:4601)
    at java.base/java.lang.String.substring(String.java:1957)
    at StringMethods.main(StringMethods.java:37)
  
```

4.6 Now run the same program and see the entire word called Jim being extracted from the string. It started with the 14, then the 15, and then the 16. 17 is not inclusive. It is less than 17

```

17 int idx = names.indexOf('o');
18 System.out.println("Index of o is: "+idx);
19
20 idx = names.lastIndexOf('o');
21 System.out.println("Last Index of o is: "+idx);
22
23 idx = names.indexOf("Jim");
24 System.out.println("idx of Jim is: "+idx);
25
26 // Strings are IMMUTABLE
27 // Strings cannot be modified
28 String upperCaseNames = names.toUpperCase();
29 System.out.println("names: "+names);
30 System.out.println("upperCaseNames: "+upperCaseNames);
31
32 String replaced = names.replace('J', 'K');
33 System.out.println("names: "+names);
34 System.out.println("replaced names: "+replaced);
35
36 //String subString = names.substring(idx);
37 String subString = names.substring(idx, 17);
38 System.out.println("subString is: "+subString);
39
<terminated> StringMethods [Java Application] Aus/clipse/plugins/org.eclipse.justi.openjdk.hotspot.jre.full/linux-x86_64/17.0.1.v20211116-1657/jre/bin/java
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim
  
```

4.7 You are going to use a comma, which typically means that you will be splitting the string based on the comma. When you execute this split method, you will get an array of strings in return. Type **String[] results = name.split(",");**

```

11 char ch = names.charAt(0);
12 System.out.println("Character at index 0 is: "+ch);
13
14 ch = names.charAt(length-1);
15 System.out.println("Character at length-1 is: "+ch);
16
17 int idx = names.indexOf('o');
18 System.out.println("Index of o is: "+idx);
19
20 idx = names.lastIndexOf('o');
21 System.out.println("Last Index of o is: "+idx);
22
23 idx = names.indexOf("Jim");
24 System.out.println("Idx of Jim is: "+idx);
25
26 // Strings are IMMUTABLE
27 // Strings cannot be modified
28 String upperCaseNames = names.toUpperCase();
29 System.out.println("names: "+names);
30 System.out.println("upperCaseNames: "+upperCaseNames);
31
32 String replaced = names.replace('J', 'K');
33 System.out.println("names: "+names);
34 System.out.println("replaced names: "+replaced);
35
36 //String subString = names.substring(idx);
37 String subString = names.substring(idx, 17); // start from 14 i.e. idx value and less than 17 i.e. till 16
38 System.out.println("subString is: "+subString);
39
40 String[] results = names.split(",");
41 for(String name: results) {
42     System.out.println(name);
43 }

```

4.8 Run the code and it gives the data as John, Jenny, Jim, Jack, and Joe. But you can observe that there is a space coming in the leading of the string. The reason is that whenever you are trying to come up and perform a split that was based on a comma, you do have a space attached to it

```

-terminated- StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.just.opened
names is 28 characters wide
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
Idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim
John
Jennie
Jim
Jack
Joe

```

Step 5: Use the trim function on strings

5.1 Now there are two remedies to this. You can use a function or method called trim which is used to eliminate the wide spaces from the leading and trailing you can see now there is no whitespace coming in front of the string

```

11 char ch = names.charAt(0);
12 System.out.println("Character at index 0 is: "+ch);
13
14 ch = names.charAt(length-1);
15 System.out.println("Character at length-1 is: "+ch);
16
17 int idx = names.indexOf('o');
18 System.out.println("Index of o is: "+idx);
19
20 idx = names.lastIndexOf('o');
21 System.out.println("Last Index of o is: "+idx);
22
23 idx = names.indexOf("Jim");
24 System.out.println("Idx of Jim is: "+idx);
25
26 // Strings are IMMUTABLE
27 // Strings cannot be modified
28 String upperCaseNames = names.toUpperCase();
29 System.out.println("names: "+names);
30 System.out.println("upperCaseNames: "+upperCaseNames);
31
32 String replaced = names.replace('J', 'K');
33 System.out.println("names: "+names);
34 System.out.println("replaced names: "+replaced);
35
36 //String subString = names.substring(idx);
37 String subString = names.substring(idx, 17); // start from 14 i.e. idx value andl less than 17 i.e. till 16
38 System.out.println("subString is: "+subString);
39
40 String[] results = names.split(",");
41 for(String name: results) {
42     System.out.println(name.trim());
43 }

```

```

<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.justi.openj
names is 28 characters wide
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
Idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim
John
Jennie
Jim
Jack
Joe

```

5.2 The other way is to perform a split right based on a comma with a space. when you do a split as comma space together, it will split based on comma and space. This can even be a regular expression

```

11 char ch = names.charAt(0);
12 System.out.println("Character at index 0 is: "+ch);
13
14 ch = names.charAt(length-1);
15 System.out.println("Character at length-1 is: "+ch);
16
17 int idx = names.indexOf('o');
18 System.out.println("Index of o is: "+idx);
19
20 idx = names.lastIndexOf('o');
21 System.out.println("Last Index of o is: "+idx);
22
23 idx = names.indexOf("Jim");
24 System.out.println("Idx of Jim is: "+idx);
25
26 // Strings are IMMUTABLE
27 // Strings cannot be modified
28 String upperCaseNames = names.toUpperCase();
29 System.out.println("names: "+names);
30 System.out.println("upperCaseNames: "+upperCaseNames);
31
32 String replaced = names.replace('J', 'K');
33 System.out.println("names: "+names);
34 System.out.println("replaced names: "+replaced);
35
36 //String subString = names.substring(idx);
37 String subString = names.substring(idx, 17); // start from 14 i.e. idx value andl less than 17 i.e. till 16
38 System.out.println("subString is: "+subString);
39
40 String[] results = names.split(", ");
41 for(String name: results) {
42     //System.out.println(name.trim());
43     System.out.println(name);
44 }

```

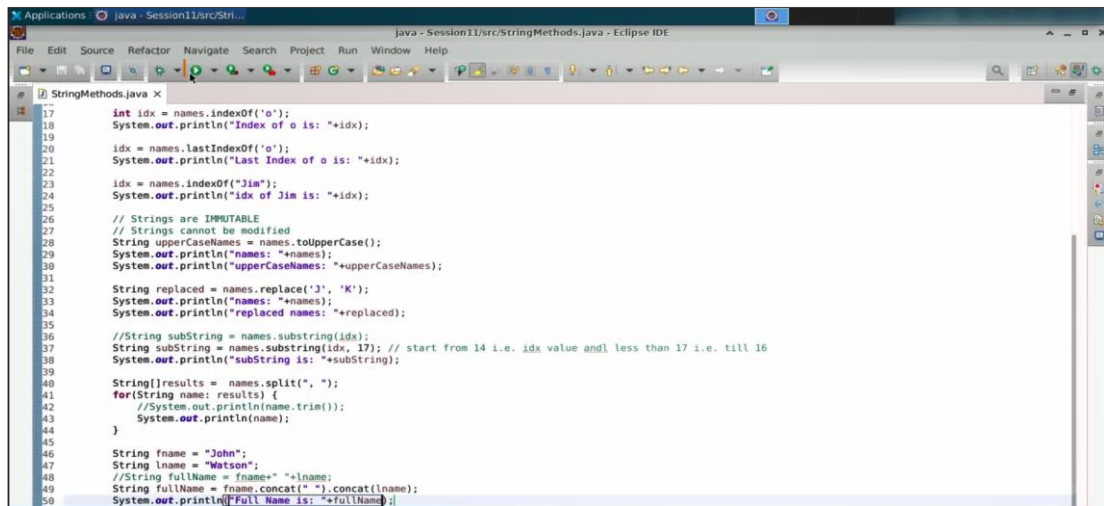
```

<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.justi.openj
names is 28 characters wide
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
Idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim
John
Jennie
Jim
Jack
Joe

```

Step 6: Implement the concat method for concatenation of strings

6.1 There is a string called **firstName**, let's say "John", and there is a string called **lastName**, which will be "Watson". Now you are going to create a string called **fullName**. You will add **firstName**, a space, and then **lastName**. If you wish, you can also use the concat method. You can type `fullName = firstName.concat(" ").concat(lastName);`. Then simply print out **fullName**

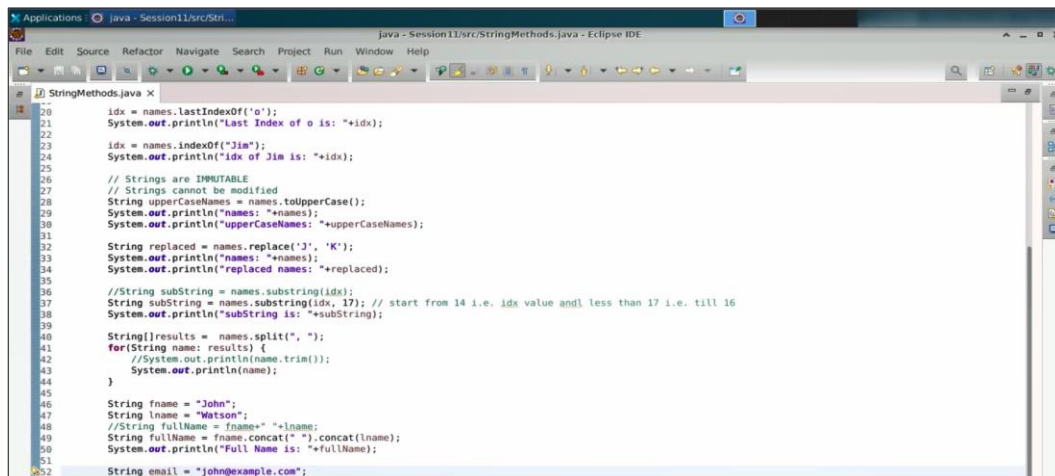


```

17 int idx = names.indexOf('o');
18 System.out.println("Index of o is: "+idx);
19
20 idx = names.lastIndexOf('o');
21 System.out.println("Last Index of o is: "+idx);
22
23 idx = names.indexOf("Jim");
24 System.out.println("Idx of Jim is: "+idx);
25
26 // Strings are IMMUTABLE
27 // Strings cannot be modified
28 String upperCaseNames = names.toUpperCase();
29 System.out.println("names: "+names);
30 System.out.println("upperCaseNames: "+upperCaseNames);
31
32 String replaced = names.replace('J', 'K');
33 System.out.println("names: "+names);
34 System.out.println("replaced names: "+replaced);
35
36 //String subString = names.substring(idx);
37 String subString = names.substring(idx, 17); // start from 14 i.e. idx value and less than 17 i.e. till 16
38 System.out.println("subString is: "+subString);
39
40 String[] results = names.split(", ");
41 for(String name: results) {
42     //System.out.println(name.trim());
43     System.out.println(name);
44 }
45
46 String fName = "John";
47 String lName = "Watson";
48 //String fullName = fName+" "+lName;
49 String fullName = fName.concat(" ").concat(lName);
50 System.out.println("Full Name is: "+fullName);

```

6.2 Now let us say that there is an email. And this email is **john@example.com**. You need to validate that whether this email is correctly formed or not

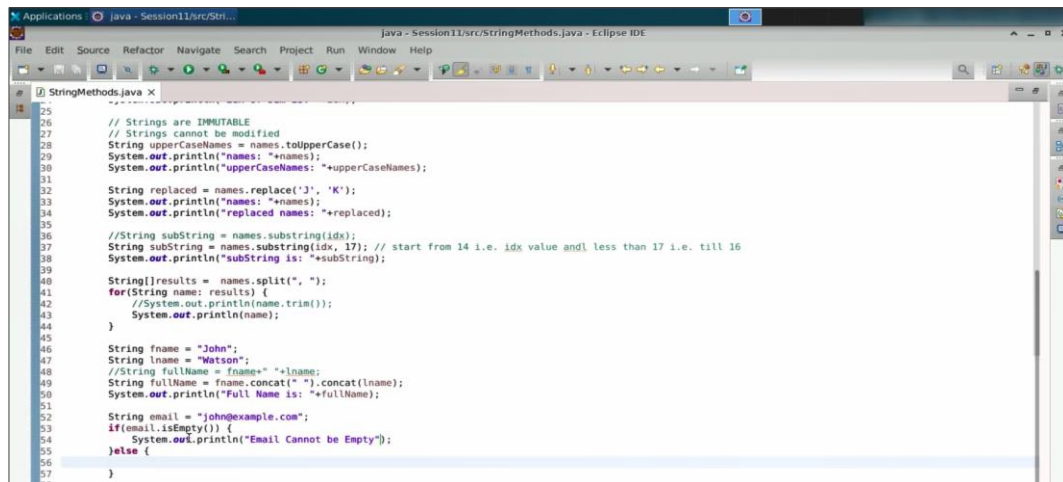


```

20 idx = names.lastIndexOf('o');
21 System.out.println("Last Index of o is: "+idx);
22
23 idx = names.indexOf("Jim");
24 System.out.println("Idx of Jim is: "+idx);
25
26 // Strings are IMMUTABLE
27 // Strings cannot be modified
28 String upperCaseNames = names.toUpperCase();
29 System.out.println("names: "+names);
30 System.out.println("upperCaseNames: "+upperCaseNames);
31
32 String replaced = names.replace('J', 'K');
33 System.out.println("names: "+names);
34 System.out.println("replaced names: "+replaced);
35
36 //String subString = names.substring(idx);
37 String subString = names.substring(idx, 17); // start from 14 i.e. idx value and less than 17 i.e. till 16
38 System.out.println("subString is: "+subString);
39
40 String[] results = names.split(", ");
41 for(String name: results) {
42     //System.out.println(name.trim());
43     System.out.println(name);
44 }
45
46 String fName = "John";
47 String lName = "Watson";
48 //String fullName = fName+" "+lName;
49 String fullName = fName.concat(" ").concat(lName);
50 System.out.println("Full Name is: "+fullName);
51
52 String email = "john@example.com";

```

6.3 You are going to put a condition: **if email.isEmpty()**, print "Email cannot be empty". Otherwise, handle the else case

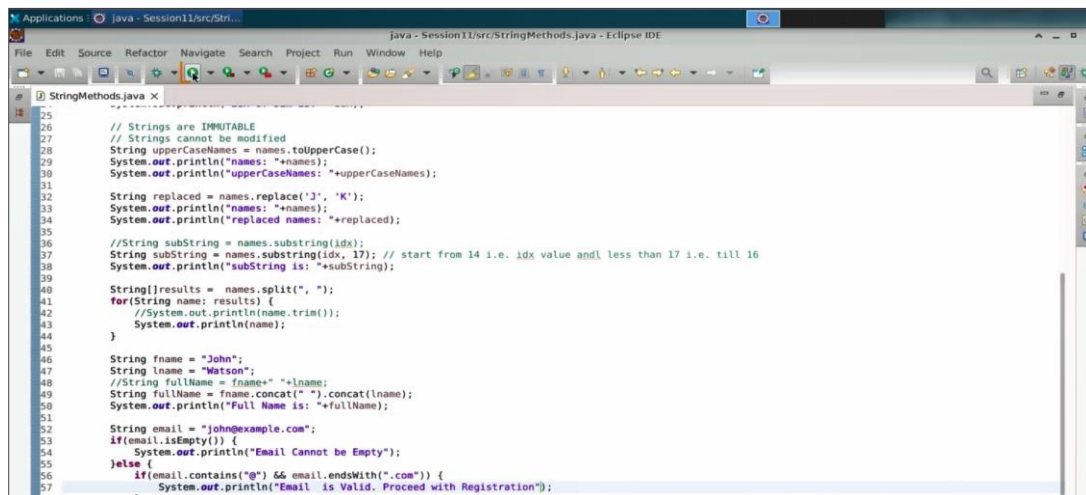


```

25
26 // Strings are IMMUTABLE
27 // Strings cannot be modified
28 String upperCaseNames = names.toUpperCase();
29 System.out.println("names: "+names);
30 System.out.println("upperCaseNames: "+upperCaseNames);
31
32 String replaced = names.replace('J', 'K');
33 System.out.println("names: "+names);
34 System.out.println("replaced names: "+replaced);
35
36 //String subString = names.substring(idx);
37 String subString = names.substring(idx, 17); // start from 14 i.e. idx value andl less than 17 i.e. till 16
38 System.out.println("subString is: "+subString);
39
40 String[]results = names.split(", ");
41 for(String name: results) {
42     //System.out.println(name.trim());
43     System.out.println(name);
44 }
45
46 String fname = "John";
47 String lname = "Watson";
48 //String fullName = fname+" "+lname;
49 String fullName = fname.concat(" ").concat(lname);
50 System.out.println("Full Name is: "+fullName);
51
52 String email = "john@example.com";
53 if(email.isEmpty()) {
54     System.out.println("Email Cannot be Empty");
55 }
56
57
58

```

6.4 If the email is not empty, it will check whether the length is 0 or not, and accordingly, it will return true or false. Now, if the email is not empty and has some data, you are going to check if the email contains the character '@' and ends with '.com'. These are the two conditions you will test. If both conditions are met, you can say the email is valid and proceed with the registration



```

25
26 // Strings are IMMUTABLE
27 // Strings cannot be modified
28 String upperCaseNames = names.toUpperCase();
29 System.out.println("names: "+names);
30 System.out.println("upperCaseNames: "+upperCaseNames);
31
32 String replaced = names.replace('J', 'K');
33 System.out.println("names: "+names);
34 System.out.println("replaced names: "+replaced);
35
36 //String subString = names.substring(idx);
37 String subString = names.substring(idx, 17); // start from 14 i.e. idx value andl less than 17 i.e. till 16
38 System.out.println("subString is: "+subString);
39
40 String[]results = names.split(", ");
41 for(String name: results) {
42     //System.out.println(name.trim());
43     System.out.println(name);
44 }
45
46 String fname = "John";
47 String lname = "Watson";
48 //String fullName = fname+" "+lname;
49 String fullName = fname.concat(" ").concat(lname);
50 System.out.println("Full Name is: "+fullName);
51
52 String email = "john@example.com";
53 if(email.isEmpty()) {
54     System.out.println("Email Cannot be Empty");
55 }
56 else {
57     if(email.contains("@") && email.endsWith(".com")) {
58         System.out.println("Email is Valid. Proceed with Registration");
59     }
60 }
61
62
63

```


6.5 Run this program, now, it says that the email is valid, and proceed with registration.

```

25
26 // Strings are IMMUTABLE
27 // Strings cannot be modified
28 String upperCaseNames = names.toUpperCase();
29 System.out.println("names: "+names);
30 System.out.println("upperCaseNames: "+upperCaseNames);
31
32 String replaced = names.replace('J', 'K');
33 System.out.println("names: "+names);
34 System.out.println("replaced names: "+replaced);
35
36 //String subString = names.substring(idx);
37 String subString = names.substring(idx, 17); // start from 14 i.e. idx value and l
38 System.out.println("subString is: "+subString);
39
40 String[] results = names.split(", ");
41 for(String name: results) {
42     //System.out.println(name.trim());
43     System.out.println(name);
44 }
45
46 String fName = "John";
47 String lName = "Watson";
48 //String fullName = fName+" "+lName;
49 String fullName = fName.concat(" ").concat(lName);
50 System.out.println("Full Name is: "+fullName);
51
52 String email = "john@example.com";
53 if(email.isEmpty()) {
54     System.out.println("Email Cannot be Empty");
55 }else {
56     if(email.contains("@") && email.endsWith(".com")) {
57         System.out.println("Email is Valid. Proceed with Registration");
58     }
59 }

```

```

<terminated> StringMethods [Java Application] hasrecipies@plugins.org.eclipse.justi.openjdk
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim
John
Jennie
Jim
Jack
Joe
Full Name is: John Watson
Email is Valid. Proceed with Registration

```

6.6 Let us try to say that the email goes something like a blank

```

25
26 // Strings are IMMUTABLE
27 // Strings cannot be modified
28 String upperCaseNames = names.toUpperCase();
29 System.out.println("names: "+names);
30 System.out.println("upperCaseNames: "+upperCaseNames);
31
32 String replaced = names.replace('J', 'K');
33 System.out.println("names: "+names);
34 System.out.println("replaced names: "+replaced);
35
36 //String subString = names.substring(idx);
37 String subString = names.substring(idx, 17); // start from 14 i.e. idx value and less than 17 i.e. till 16
38 System.out.println("subString is: "+subString);
39
40 String[] results = names.split(", ");
41 for(String name: results) {
42     //System.out.println(name.trim());
43     System.out.println(name);
44 }
45
46 String fName = "John";
47 String lName = "Watson";
48 //String fullName = fName+" "+lName;
49 String fullName = fName.concat(" ").concat(lName);
50 System.out.println("Full Name is: "+fullName);
51
52 String email = "";
53 if(email.isEmpty()) {
54     System.out.println("Email Cannot be Empty");
55 }else {
56     if(email.contains("@") && email.endsWith(".com")) {
57         System.out.println("Email is Valid. Proceed with Registration");
58     }
59 }

```

6.7 Run this program, now, it says Email Cannot Be Empty

```

StringMethods.java
25 // Strings are IMMUTABLE
26 // Strings cannot be modified
27 String upperCaseNames = names.toUpperCase();
28 System.out.println("names: "+names);
29 System.out.println("upperCaseNames: "+upperCaseNames);
30
31 String replaced = names.replace('J', 'K');
32 System.out.println("names: "+names);
33 System.out.println("replaced names: "+replaced);
34
35 //String subString = names.substring(idx);
36 String subString = names.substring(idx, 17); // start from 14 i.e. idx value and less than 17 i.e. till 16
37 System.out.println("subString is: "+subString);
38
39 String[] results = names.split(", ");
40 for(String name: results) {
41     //System.out.println(name.trim());
42     System.out.println(name);
43 }
44
45 String fName = "John";
46 String lName = "Watson";
47 //String fullName = fName+" "+lName;
48 String fullName = fName.concat(" ").concat(lName);
49 System.out.println("Full Name is: "+fullName);
50
51 String email = "john@example.com";
52 if(email.isEmpty()) {
53     System.out.println("Email Cannot be Empty");
54 } else {
55     if(email.contains("@") && email.endsWith(".com")) {
56         System.out.println("Email is Valid. Proceed with Registration");
57     }
58 }

```

```

<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.justi.openjdk
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim
John
Jennie
Jim
Jack
Joe
Full Name is: John Watson
Email Cannot be Empty

```

6.8 Let us simply remove this @. So, contains check anything as a part of your string

```

StringMethods.java
25 // Strings are IMMUTABLE
26 // Strings cannot be modified
27 String upperCaseNames = names.toUpperCase();
28 System.out.println("names: "+names);
29 System.out.println("upperCaseNames: "+upperCaseNames);
30
31 String replaced = names.replace('J', 'K');
32 System.out.println("names: "+names);
33 System.out.println("replaced names: "+replaced);
34
35 //String subString = names.substring(idx);
36 String subString = names.substring(idx, 17); // start from 14 i.e. idx value and less than 17 i.e. till 16
37 System.out.println("subString is: "+subString);
38
39 String[] results = names.split(", ");
40 for(String name: results) {
41     //System.out.println(name.trim());
42     System.out.println(name);
43 }
44
45 String fName = "John";
46 String lName = "Watson";
47 //String fullName = fName+" "+lName;
48 String fullName = fName.concat(" ").concat(lName);
49 System.out.println("Full Name is: "+fullName);
50
51 String email = "johnxample.com";
52 if(email.isEmpty()) {
53     System.out.println("Email Cannot be Empty");
54 } else {
55     if(email.contains("w") && email.endsWith(".com")) {
56         System.out.println("Email is Valid. Proceed with Registration");
57     }
58 }

```

6.9 Run this program. Now, you can see else part is not written

```

StringMethods.java
25 // Strings are IMMUTABLE
26 // Strings cannot be modified
27 String upperCaseNames = names.toUpperCase();
28 System.out.println("names: "+names);
29 System.out.println("upperCaseNames: "+upperCaseNames);
30
31 String replaced = names.replace('J', 'K');
32 System.out.println("names: "+names);
33 System.out.println("replaced names: "+replaced);
34
35 //String subString = names.substring(idx);
36 String subString = names.substring(idx, 17); // start from 14 i.e. idx value and l less than 17 i.e. till 16
37 System.out.println("subString is: "+subString);
38
39 String[] results = names.split(", ");
40 for(String name: results) {
41     //System.out.println(name.trim());
42     System.out.println(name);
43 }
44
45 String fName = "John";
46 String lName = "Watson";
47 //String fullName = fName+" "+lName;
48 String fullName = fName.concat(" ").concat(lName);
49 System.out.println("Full Name is: "+fullName);
50
51 String email = "johnexample.com";
52 if(email.isEmpty()) {
53     System.out.println("Email Cannot be Empty");
54 } else {
55     if(email.contains("@") && email.endsWith(".com")) {
56         System.out.println("Email is Valid. Proceed with Registration");
57     }
58 }

```

```

<terminated> StringMethods (Java Application) /usr/eclipse/plugins/org.eclipse.justj.openjdk
names is 28 characters wide
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim
John
Jennie
Jim
Jack
Joe
Full Name is: John Watson

```

6.10 You can once again come up and say that this is an invalid email. You can even print the variable

```

StringMethods.java
25 // Strings are IMMUTABLE
26 // Strings cannot be modified
27 String upperCaseNames = names.toUpperCase();
28 System.out.println("names: "+names);
29 System.out.println("upperCaseNames: "+upperCaseNames);
30
31 String replaced = names.replace('J', 'K');
32 System.out.println("names: "+names);
33 System.out.println("replaced names: "+replaced);
34
35 //String subString = names.substring(idx);
36 String subString = names.substring(idx, 17); // start from 14 i.e. idx value and l less than 17 i.e. till 16
37 System.out.println("subString is: "+subString);
38
39 String[] results = names.split(", ");
40 for(String name: results) {
41     //System.out.println(name.trim());
42     System.out.println(name);
43 }
44
45 String fName = "John";
46 String lName = "Watson";
47 //String fullName = fName+" "+lName;
48 String fullName = fName.concat(" ").concat(lName);
49 System.out.println("Full Name is: "+fullName);
50
51 String email = "johnexample.com";
52 if(email.isEmpty()) {
53     System.out.println("Email Cannot be Empty");
54 } else {
55     if(email.contains("@") && email.endsWith(".com")) {
56         System.out.println("Email is Valid. Proceed with Registration");
57     } else {
58         System.out.println("Invalid Email: "+email);
59     }
60 }
61

```


6.11 Run this program. Now, it says invalid email **Johnexample.com**

The screenshot shows the Eclipse IDE with a Java file named `StringMethods.java`. The code includes logic for string manipulation and email validation. The console output shows the following:

```
<terminated> StringMethods [Java Application] J:\s\ eclipse\plugins\org.eclipse.justi.openjdk
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim
John
Jennie
Jim
Jack
Joe
Full Name is: John Watson
Invalid Email: johnexample.com
```

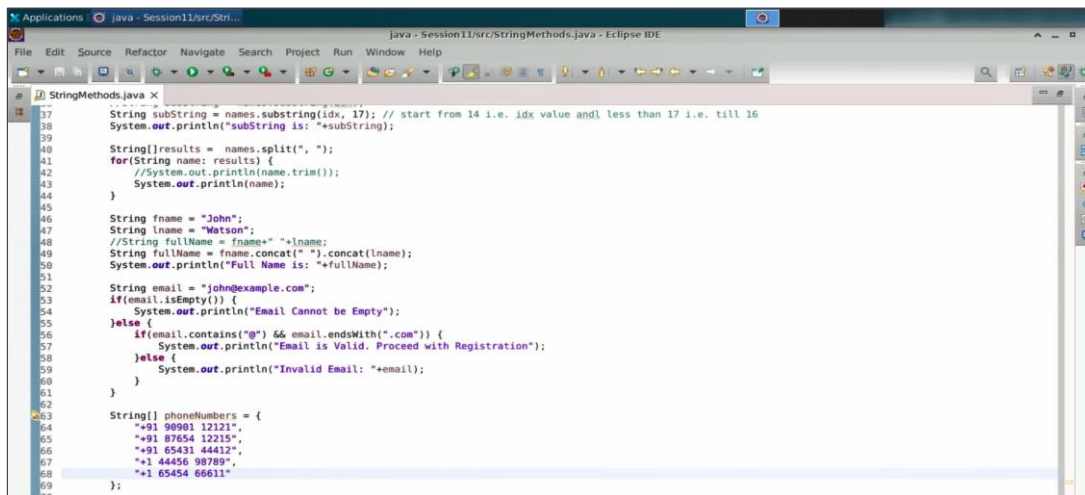
6.12 Let us have '@' at the same location, and you can eliminate the '.com' condition, or you can simply say this is not valid. However, as per the condition, it should end with '.com', so it will say **Invalid email**". This clarifies what is meant by the **endsWith** method

The screenshot shows the Eclipse IDE with the same Java file `StringMethods.java`. The code is identical to the previous one, but the email variable is now `john@example.in`. The console output is as follows:

```
<terminated> StringMethods [Java Application] J:\s\ eclipse\plugins\org.eclipse.justi.openjdk
Character at index 0 is: J
Character at length-1 is: e
Index of o is: 1
Last Index of o is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim
John
Jennie
Jim
Jack
Joe
Full Name is: John Watson
Invalid Email: johnexample.in
```

Step 7: Validate the strings using the methods ends, contains ends with, and starts with

7.1 Let us try to explore something more on this string method called contains ends with and starts with. You are going to create a list of contacts or the phone numbers available in your address book. You have this array of phone numbers and you want to perform a search operation



```
37 String subString = names.substring(idx, 17); // start from 14 i.e. idx value and less than 17 i.e. till 16
38 System.out.println("subString is: "+subString);
39
40 String[] results = names.split(", ");
41 for(String name: results) {
42     //System.out.println(name.trim());
43     System.out.println(name);
44 }
45
46 String fName = "John";
47 String lName = "Watson";
48 //String fullName = fName+" "+lName;
49 String fullName = fName.concat(" ").concat(lName);
50 System.out.println("Full Name is: "+fullName);
51
52 String email = "john@example.com";
53 if(email.isEmpty()) {
54     System.out.println("Email Cannot be Empty");
55 }else {
56     if(email.contains("@") && email.endsWith(".com")) {
57         System.out.println("Email is Valid. Proceed with Registration");
58     }else {
59         System.out.println("Invalid Email: "+email);
60     }
61 }
62
63 String[] phoneNumbers = {
64     "+91 98901 12121",
65     "+91 87654 12215",
66     "+91 65431 44412",
67     "+1 44456 98789",
68     "+1 65454 66611"
69 };
70
```

7.2 Let us say there is a search keyword, and it is +91. You want to filter the phone numbers. Put a loop that starts with the index value 0 and continues while the index is less than **myPhoneNumbers.length**. Use **idx++** to increment the index. Now, if the phone number at the current index starts with your search keyword, you can display this phone number. Add the phone numbers at the index

```

41 for(String name: results) {
42     //System.out.println(name.trim());
43     System.out.println(name);
44 }
45
46 String fname = "John";
47 String lname = "Watson";
48 //String fullName = fname+" "+lname;
49 String fullName = fname.concat(" ").concat(lname);
50 System.out.println("Full Name is: "+fullName);
51
52 String email = "john@example.com";
53 if(email.isEmpty()) {
54     System.out.println("Email Cannot be Empty");
55 }
56 else {
57     if(email.contains("@") && email.endsWith(".com")) {
58         System.out.println("Email is Valid. Proceed with Registration");
59     }
60     else {
61         System.out.println("Invalid Email: "+email);
62     }
63 }
64
65 String[] phoneNumbers = {
66     "+91 98901 12121",
67     "+91 87654 12215",
68     "+91 65431 44412",
69     "+1 44456 98789",
70     "+1 65454 66611"
71 };
72
73 String searchKeyword = "+91";
74
75 for(idx=0;idx<phoneNumbers.length;idx++) {
76     if(phoneNumbers[idx].startsWith(searchKeyword)) {
77         System.out.println(phoneNumbers[idx]);
78     }
79 }

```

7.3 Run this program, and it gives you all the phone numbers that start with +91

```

41 for(String name: results) {
42     //System.out.println(name.trim());
43     System.out.println(name);
44 }
45
46 String fname = "John";
47 String lname = "Watson";
48 //String fullName = fname+" "+lname;
49 String fullName = fname.concat(" ").concat(lname);
50 System.out.println("Full Name is: "+fullName);
51
52 String email = "john@example.com";
53 if(email.isEmpty()) {
54     System.out.println("Email Cannot be Empty");
55 }
56 else {
57     if(email.contains("@") && email.endsWith(".com")) {
58         System.out.println("Email is Valid. Proceed with Registration");
59     }
60     else {
61         System.out.println("Invalid Email: "+email);
62     }
63 }
64
65 String[] phoneNumbers = {
66     "+91 98901 12121",
67     "+91 87654 12215",
68     "+91 65431 44412",
69     "+1 44456 98789",
70     "+1 65454 66611"
71 };
72
73 String searchKeyword = "+91";
74
75 for(idx=0;idx<phoneNumbers.length;idx++) {
76     if(phoneNumbers[idx].startsWith(searchKeyword)) {
77         System.out.println(phoneNumbers[idx]);
78     }
79 }

```

```

<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.justi.openj
Last Index of 0 is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kin, Kack, Koe
subString is: Jim
John
Jennie
Jim
Jack
Joe
Full Name is: John Watson
Email is Valid. Proceed with Registration
+91 98901 12121
+91 87654 12215
+91 65431 44412

```

7.4 If you just remove your search keyword with plus one, it is going to filter all your phone numbers with the +1

```

41 for(String name: results) {
42     //System.out.println(name.trim());
43     System.out.println(name);
44 }
45
46 String fname = "John";
47 String lname = "Watson";
48 //String fullName = fname+" "+lname;
49 String fullName = fname.concat(" ").concat(lname);
50 System.out.println("Full Name is: "+fullName);
51
52 String email = "john@example.com";
53 if(email.isEmpty()) {
54     System.out.println("Email Cannot be Empty");
55 }
56 else {
57     if(email.contains("@") && email.endsWith(".com")) {
58         System.out.println("Email is Valid. Proceed with Registration");
59     }
60     else {
61         System.out.println("Invalid Email: "+email);
62     }
63 }
64
65 String[] phoneNumbers = {
66     "+91 98901 12121",
67     "+91 87654 12215",
68     "+91 65431 44412",
69     "+1 44456 98789",
70     "+1 65454 66611"
71 };
72
73 String searchKeyword = "+1";
74 for(int idx=0; idx<phoneNumbers.length; idx++) {
75     if(phoneNumbers[idx].startsWith(searchKeyword)) {

```

```

<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.justi.openj
Index of o is: 1
Last Index of o is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim
John
Jennie
Jim
Jack
Joe
Full Name is: John Watson
Email is Valid. Proceed with Registration
+1 44456 98789
+1 65454 66611

```

7.5 Let us see a magical thing where the search keyword can be "12". If you implement this with **startsWith**, it will not work. This is where you can use a very useful concept called **contains**. You will check if "12" is contained within the string, and you can see the list of phone numbers with "12" appearing anywhere in them

```

46 String fname = "John";
47 String lname = "Watson";
48 //String fullName = fname+" "+lname;
49 String fullName = fname.concat(" ").concat(lname);
50 System.out.println("Full Name is: "+fullName);
51
52 String email = "john@example.com";
53 if(email.isEmpty()) {
54     System.out.println("Email Cannot be Empty");
55 }
56 else {
57     if(email.contains("@") && email.endsWith(".com")) {
58         System.out.println("Email is Valid. Proceed with Registration");
59     }
60     else {
61         System.out.println("Invalid Email: "+email);
62     }
63 }
64
65 String[] phoneNumbers = {
66     "+91 98901 12121",
67     "+91 87654 12215",
68     "+91 65431 44412",
69     "+1 44456 98789",
70     "+1 65454 66611"
71 };
72
73 //String searchKeyword = "+1";
74 String searchKeyword = "12";
75 for(int idx=0; idx<phoneNumbers.length; idx++) {
76     /*if(phoneNumbers[idx].startsWith(searchKeyword)) {
77         System.out.println(phoneNumbers[idx]);
78     }*/
79     if(phoneNumbers[idx].contains(searchKeyword)) {
80         System.out.println(phoneNumbers[idx]);
81     }
82 }
83

```

```

<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.justi.openj
Last Index of o is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim
John
Jennie
Jim
Jack
Joe
Full Name is: John Watson
Email is Valid. Proceed with Registration
+91 98901 12121
+91 87654 12215
+91 65431 44412

```

Step 8: Implement an algorithm to know the number of times a character appears

8.1 Let us also see one more method. Type **names.toCharArray**, and you are going to come up and convert all your characters into the names as an array, you are going to type **CharArray**, the Char names are your **names.toCharArray**. And for, CharArray it is in the char names and you are going to print out the character after the character prints a space

```

StringMethods.java
54 System.out.println("Email Cannot be Empty");
55 }
56 else {
57     if(email.contains("@") && email.endsWith(".com")) {
58         System.out.println("Email is Valid. Proceed with Registration");
59     }
60     else {
61         System.out.println("Invalid Email: "+email);
62     }
63 }
64 String[] phoneNumbers = {
65     "+91 98901 12121",
66     "+91 87654 12215",
67     "+91 65431 44412",
68     "+1 44456 98789",
69     "+1 65454 66611"
70 };
71 //String searchKeyword = "+1";
72 String searchKeyword = "54";
73 for(int idx=0; idx<phoneNumbers.length; idx++) {
74     /*if(phoneNumbers[idx].startsWith(searchKeyword)) {
75         System.out.println(phoneNumbers[idx]);
76     }*/
77     if(phoneNumbers[idx].contains(searchKeyword)) {
78         System.out.println(phoneNumbers[idx]);
79     }
80 }
81
82 I
83 char[] charNames = names.toCharArray();
84 for(char chr: charNames) {
85     System.out.print(chr+" ");
86 }
87 }

```

```

<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.justi.openi
Last Index of o is: 26
idx of Jim is: 14
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim
John
Jennie
Jim
Jack
Joe
Full Name is: John Watson
Email is Valid. Proceed with Registration
+91 87654 12215
+91 65431 44412
+1 65454 66611
J o h n . J e n n i e . J i m , J a c k ,

```

8.2 Now, maintain a count variable initialized to 0. If your character equals 'o', then increment the count. This is a simple algorithm that counts how many times a character appears. Add an empty print line here so that the output appears on the next line. Run the code, and you will see that 'o' appears two times

```

StringMethods.java
61 }
62
63 String[] phoneNumbers = {
64     "+91 98901 12121",
65     "+91 87654 12215",
66     "+91 65431 44412",
67     "+1 44456 98789",
68     "+1 65454 66611"
69 };
70
71 //String searchKeyword = "+1";
72 String searchKeyword = "54";
73 for(int idx=0; idx<phoneNumbers.length; idx++) {
74     /*if(phoneNumbers[idx].startsWith(searchKeyword)) {
75         System.out.println(phoneNumbers[idx]);
76     }*/
77     if(phoneNumbers[idx].contains(searchKeyword)) {
78         System.out.println(phoneNumbers[idx]);
79     }
80 }
81
82
83 int count = 0;
84 char[] charNames = names.toCharArray();
85 for(char chr: charNames) {
86     System.out.print(chr+" ");
87     if(chr == 'o') {
88         count++;
89     }
90 }
91
92 System.out.println();
93 System.out.println("o appears "+count+" times");
94 }

```

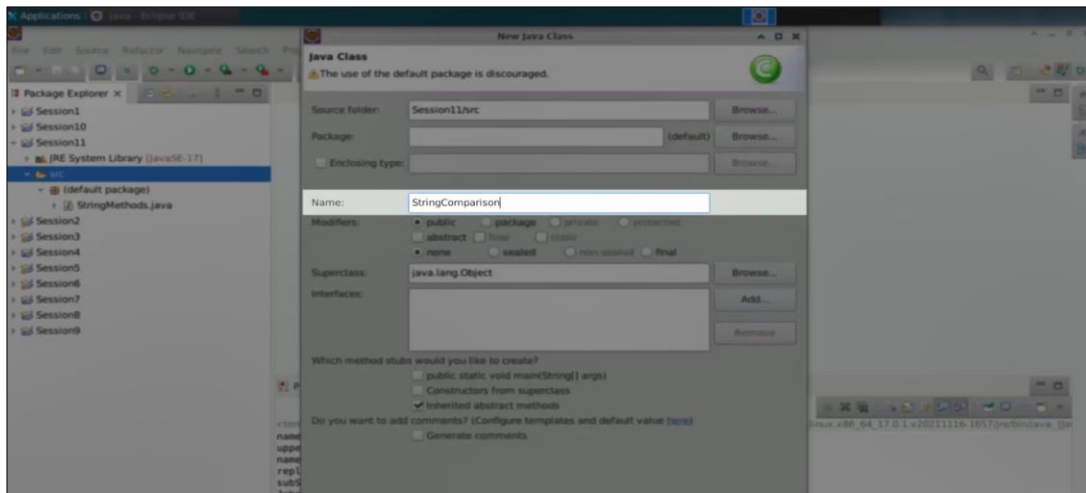
```

<terminated> StringMethods [Java Application] /usr/eclipse/plugins/org.eclipse.justi.openi
names: John, Jennie, Jim, Jack, Joe
upperCaseNames: JOHN, JENNIE, JIM, JACK, JOE
names: John, Jennie, Jim, Jack, Joe
replaced names: Kohn, Kennie, Kim, Kack, Koe
subString is: Jim
John
Jennie
Jim
Jack
Joe
Full Name is: John Watson
Email is Valid. Proceed with Registration
+91 87654 12215
+91 65431 44412
+1 65454 66611
J o h n . J e n n i e . J i m , J a c k ,
o appears 2 times

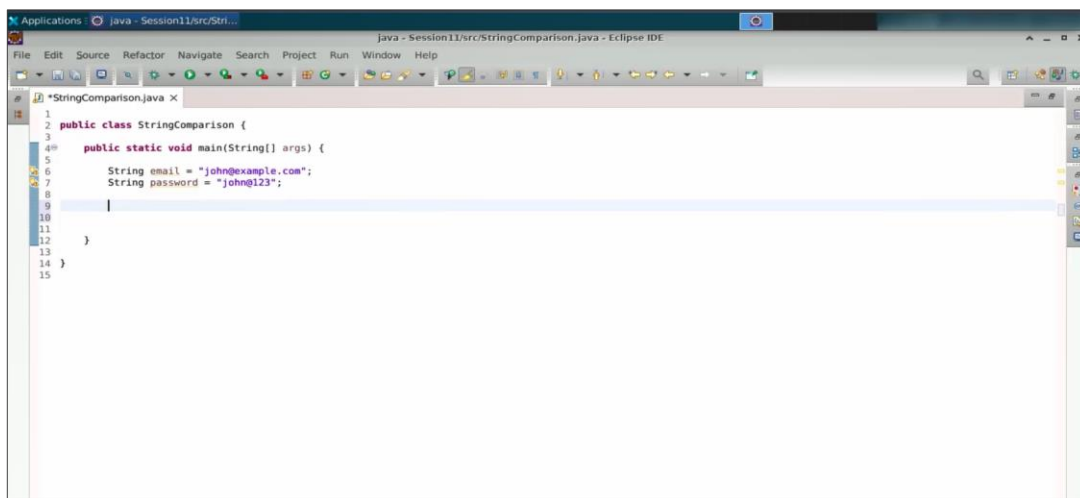
```

Step 9: Implement String comparison and intern strings

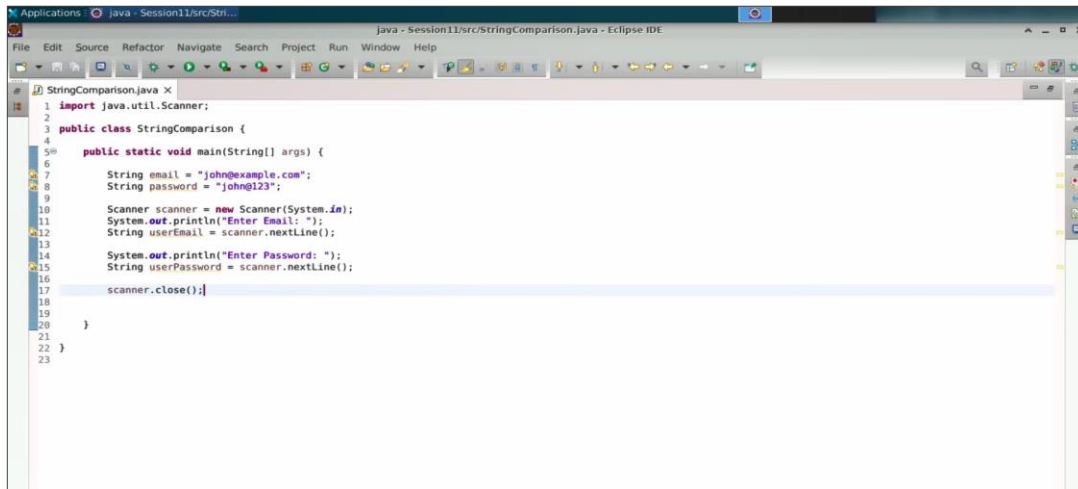
9.1 Now the next thing is string comparison. It is an important concept, and you will create altogether one separate class on it. Let us name it as **StringComparison**



9.2 Let us say there is a string called email which is **john@example.com** and there is a string called password which is **John@123**. Now, these are the credentials, and you need to validate these credentials

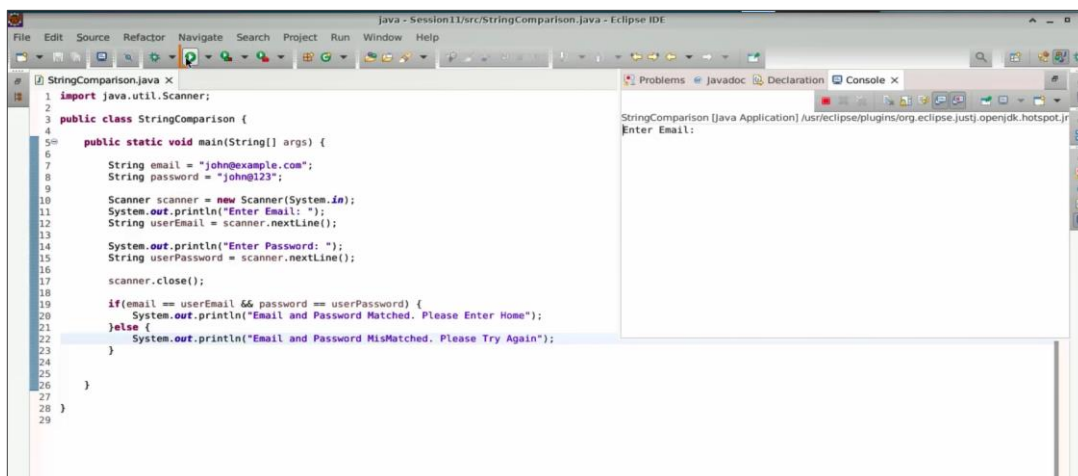


9.3 You are going to create a Scanner class. Type **Scanner scanner = new Scanner(System.in);**. Then prompt the user to enter the email and password. Store the user email with **userEmail = scanner.nextLine();** and the user password with **userPassword = scanner.nextLine();**. Finally, close the scanner



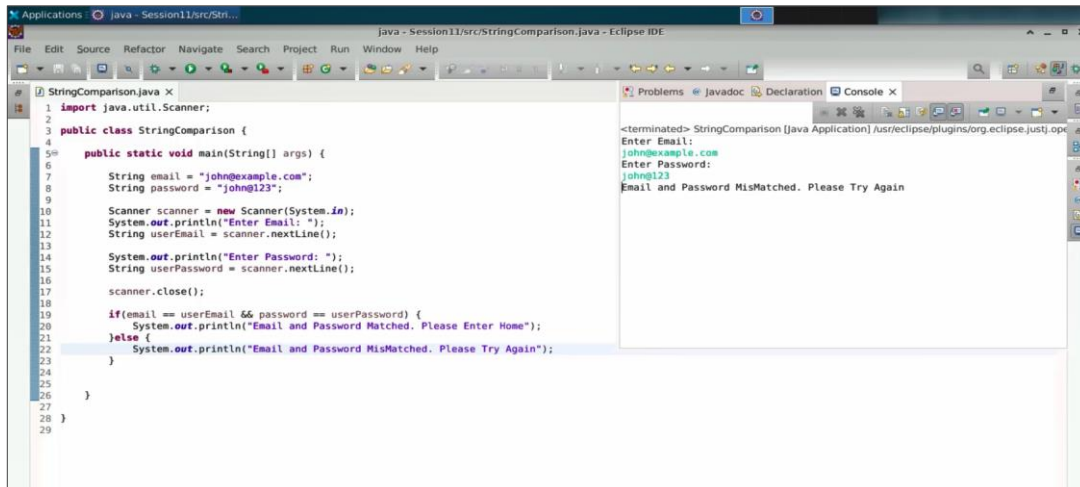
```
1 import java.util.Scanner;
2
3 public class StringComparison {
4
5     public static void main(String[] args) {
6
7         String email = "john@example.com";
8         String password = "john@123";
9
10        Scanner scanner = new Scanner(System.in);
11        System.out.println("Enter Email: ");
12        String userEmail = scanner.nextLine();
13
14        System.out.println("Enter Password: ");
15        String userPassword = scanner.nextLine();
16
17        scanner.close();
18    }
19 }
20
21
22
23
```

9.4 Now, let us do a string comparison. If the email is equal to the user email and the password is equal to the user password, you are going to add a **system.out.println**. If the email and password match, Please Enter Home and if it does not, type Please Try Again



```
1 import java.util.Scanner;
2
3 public class StringComparison {
4
5     public static void main(String[] args) {
6
7         String email = "john@example.com";
8         String password = "john@123";
9
10        Scanner scanner = new Scanner(System.in);
11        System.out.println("Enter Email: ");
12        String userEmail = scanner.nextLine();
13
14        System.out.println("Enter Password: ");
15        String userPassword = scanner.nextLine();
16
17        scanner.close();
18
19        if(email == userEmail && password == userPassword) {
20            System.out.println("Email and Password Matched. Please Enter Home");
21        } else {
22            System.out.println("Email and Password Mismatched. Please Try Again");
23        }
24    }
25 }
26
27
28
29
```

9.5 Run the program and enter the email and password. It indicates a password mismatch even though you have entered the correct credentials. This issue arises because there are two different ways to create strings in Java. The first method involves creating an interned string. The other method creates a string object in memory, which can affect string comparison



```

1 import java.util.Scanner;
2
3 public class StringComparison {
4
5     public static void main(String[] args) {
6         String email = "john@example.com";
7         String password = "john@123";
8
9         Scanner scanner = new Scanner(System.in);
10        System.out.println("Enter Email: ");
11        String userEmail = scanner.nextLine();
12
13        System.out.println("Enter Password: ");
14        String userPassword = scanner.nextLine();
15
16        scanner.close();
17
18        if(email == userEmail && password == userPassword) {
19            System.out.println("Email and Password Matched. Please Enter Home");
20        } else {
21            System.out.println("Email and Password Mismatched. Please Try Again");
22        }
23    }
24 }
25
26 }
27
28 }
29

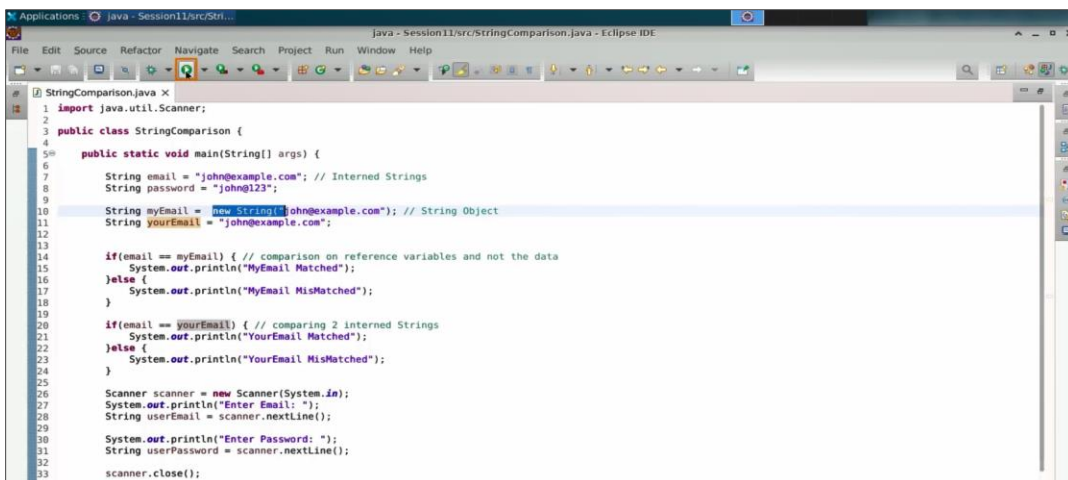
```

```

<terminated> StringComparison [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full/bin/linux64/java
Enter Email:
john@example.com
Enter Password:
john@123
Email and Password Mismatched. Please Try Again

```

9.6 Let us say my email is defined as a new string with the value **john@example.com**. Now, if you try to compare this email with my email, you will not get the correct result. Let us create another string called your email, which also has the value **john@example.com**. If you check whether 'email' is equal to my email, it will return false. If they match, you will print the Email matched. Otherwise, you will print Email mismatched

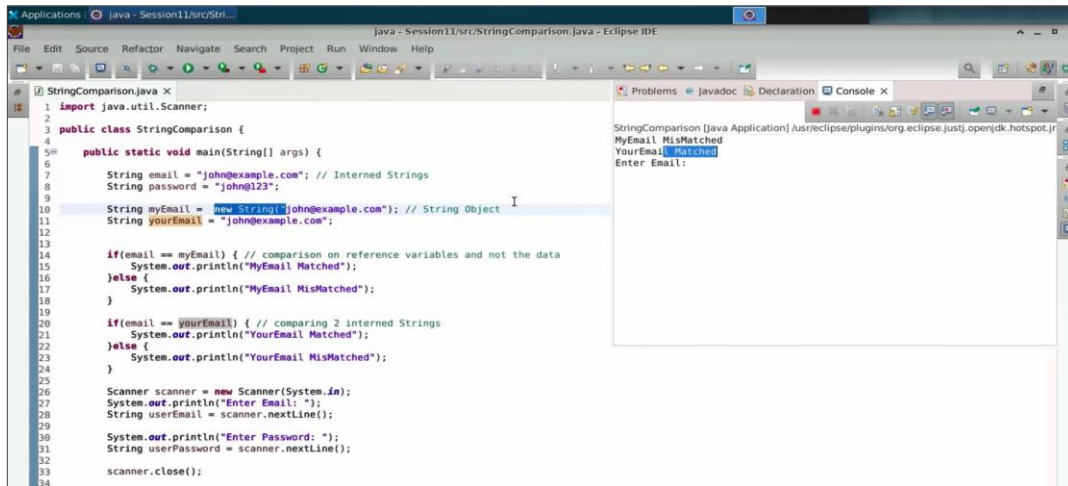


```

1 import java.util.Scanner;
2
3 public class StringComparison {
4
5     public static void main(String[] args) {
6         String email = "john@example.com"; // Interned Strings
7         String password = "john@123";
8
9         String myEmail = new String("john@example.com"); // String Object
10        String yourEmail = "john@example.com";
11
12
13
14        if(email == myEmail) { // comparison on reference variables and not the data
15            System.out.println("MyEmail Matched");
16        } else {
17            System.out.println("MyEmail Mismatched");
18        }
19
20        if(email == yourEmail) { // comparing 2 interned Strings
21            System.out.println("YourEmail Matched");
22        } else {
23            System.out.println("YourEmail Mismatched");
24        }
25
26        Scanner scanner = new Scanner(System.in);
27        System.out.println("Enter Email: ");
28        String userEmail = scanner.nextLine();
29
30        System.out.println("Enter Password: ");
31        String userPassword = scanner.nextLine();
32
33        scanner.close();
34    }
35 }
36
37 }
38
39 }
40

```


9.7 Run the program. You get to see my email mismatched, but your email matched, so comparing the strings that are interned strings can be done through equal to operator. But not for string objects



```

1  import java.util.Scanner;
2
3  public class StringComparison {
4
5      public static void main(String[] args) {
6          String email = "john@example.com"; // Interned Strings
7          String password = "john@123";
8
9          String myEmail = new String("john@example.com"); // String Object
10         String yourEmail = "john@example.com";
11
12
13         if(email == myEmail) { // comparison on reference variables and not the data
14             System.out.println("MyEmail Matched");
15         }
16         else {
17             System.out.println("MyEmail MisMatched");
18         }
19
20         if(email == yourEmail) { // comparing 2 interned Strings
21             System.out.println("YourEmail Matched");
22         }
23         else {
24             System.out.println("YourEmail MisMatched");
25         }
26
27         Scanner scanner = new Scanner(System.in);
28         System.out.println("Enter Email: ");
29         String userEmail = scanner.nextLine();
30
31         System.out.println("Enter Password: ");
32         String userPassword = scanner.nextLine();
33
34         scanner.close();
35     }
36 }

```

Console Output:

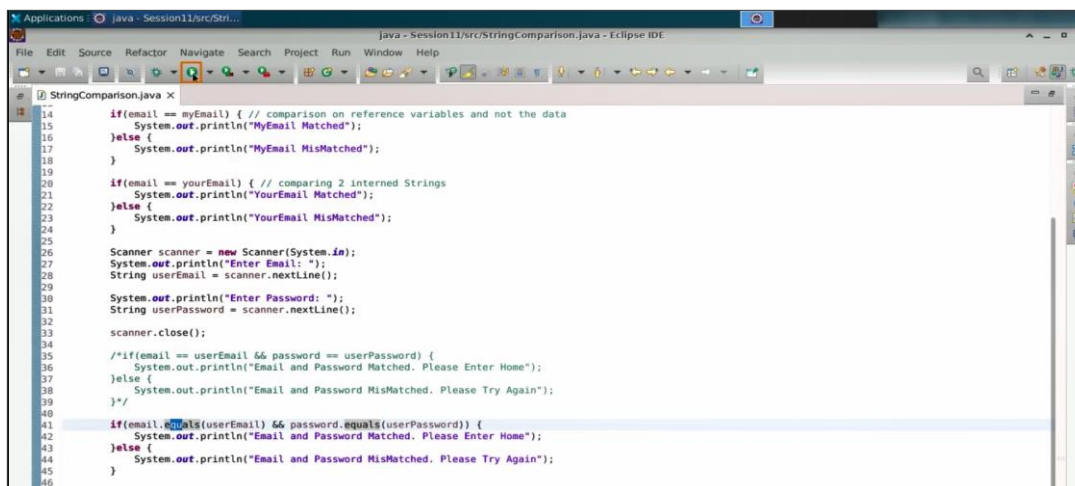
```

StringComparison [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jr
MyEmail MisMatched
YourEmail Matched
Enter Email:

```

Step 10: Use the method.equals method

10.1 If you want to compare the content, the equals operator will not work. You can use a method from the String class called equals. This method equals, will compare the content. After replacing the equals operator with the equals method, the equals method will return true or false

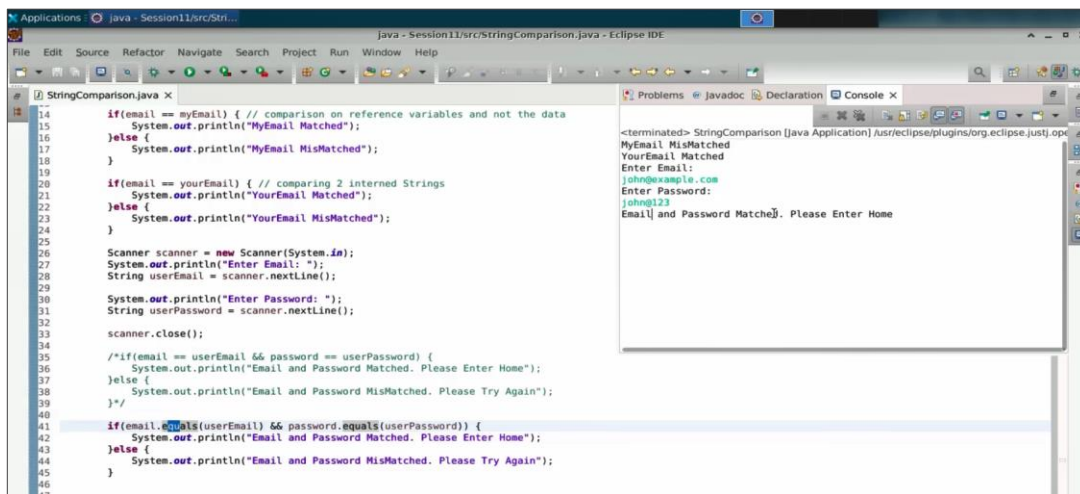


```

14         if(email == myEmail) { // comparison on reference variables and not the data
15             System.out.println("MyEmail Matched");
16         }
17         else {
18             System.out.println("MyEmail MisMatched");
19         }
20
21         if(email == yourEmail) { // comparing 2 interned Strings
22             System.out.println("YourEmail Matched");
23         }
24         else {
25             System.out.println("YourEmail MisMatched");
26         }
27
28         Scanner scanner = new Scanner(System.in);
29         System.out.println("Enter Email: ");
30         String userEmail = scanner.nextLine();
31
32         System.out.println("Enter Password: ");
33         String userPassword = scanner.nextLine();
34
35         scanner.close();
36
37         /*if(email == userEmail && password == userPassword) {
38             System.out.println("Email and Password Matched. Please Enter Home");
39         }
40         else {
41             System.out.println("Email and Password MisMatched. Please Try Again");
42         }
43         */
44
45         if(email.equals(userEmail) && password.equals(userPassword)) {
46             System.out.println("Email and Password Matched. Please Enter Home");
47         }
48         else {
49             System.out.println("Email and Password MisMatched. Please Try Again");
50         }
51     }
52 }

```

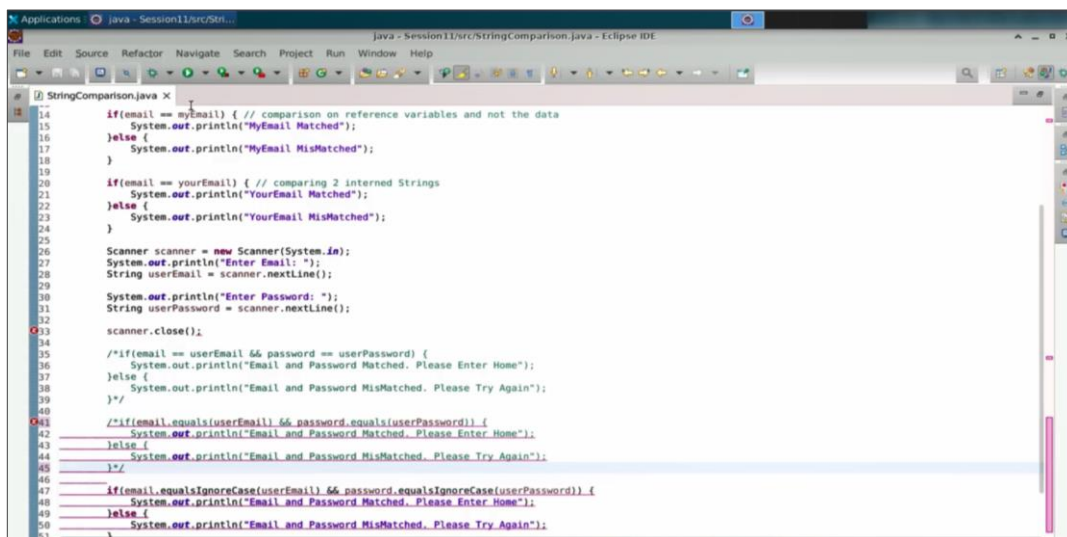
10.2 Run the code and type **john@example.com** as the email and **John@123** as the password. It shows that the email and password matched. Comparing strings is a risky operation. If you have string constants, such as interned strings where the references directly point to the literals, you can use the equals operator. However, when making string comparisons while reading from the console or some UI, you should not use the equals operator. Instead, you should work with the equals method



The screenshot shows the Eclipse IDE with the file `StringComparison.java` open. The code includes comments and logic for comparing email and password strings. The console output on the right shows the execution results:

```
<terminated> StringComparison [Java Application] /usr/eclipse/plugins/org.eclipse.justi.op
MyEmail MisMatched
YourEmail Matched
Enter Email:
john@example.com
Enter Password:
john@123
Email and Password Matched. Please Enter Home
```

10.3 Add equals ignore case, this will even ignore the case sensitivity



The screenshot shows the Eclipse IDE with the file `StringComparison.java` open. The code has been updated to include a comparison using `equalsIgnoreCase` for case-insensitive matching. The console output on the right shows the execution results:

```
<terminated> StringComparison [Java Application] /usr/eclipse/plugins/org.eclipse.justi.op
MyEmail MisMatched
YourEmail Matched
Enter Email:
john@example.com
Enter Password:
john@123
Email and Password Matched. Please Enter Home
```

10.4 Run the code and enter an email which is like **JOHN@example.com**, and the password is **JHON@123**, you can see it says email and password match

The screenshot shows the Eclipse IDE with the file `StringComparison.java` open. The code includes logic for comparing email and password. The console output shows the following sequence of events:

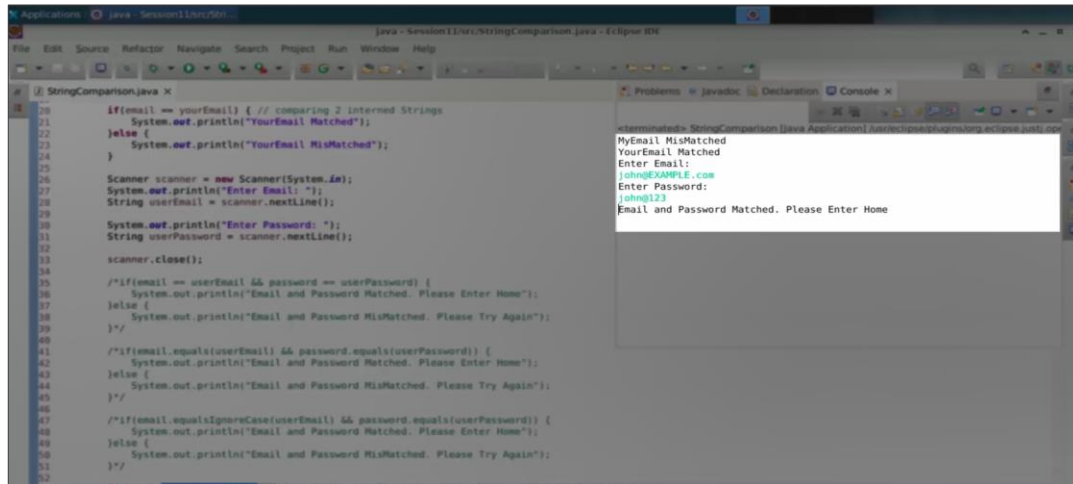
```
<terminated> StringComparison (Java Application) /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full/bin/linux64/java
MyEmail MisMatched
YourEmail Matched
Enter Email:
JOHN@example.com
Enter Password:
JHON@123
Email and Password Matched. Please Enter Home
```

10.5 You have one more method to help with string comparison. This is **`email.compareToIgnoreCase`**, and it should return a value of zero. **`compareTo`** will compare with case sensitivity, whereas **`compareToIgnoreCase`** will ignore case sensitivity

The screenshot shows the Eclipse IDE with the file `StringComparison.java` open. The code is the same as in the previous screenshot, but with an additional method call highlighted in blue:

```
if(email.compareToIgnoreCase(userEmail) == 0 && password.compareTo(userPassword) == 0) {
    System.out.println("Email and Password Matched. Please Enter Home");
} else {
    System.out.println("Email and Password MisMatched. Please Try Again");
}
```

10.6 Run the code, here you are with the email shown as **John@EXAMPLE.com**, where the password is **John@1 2 3**. Password is case sensitive, and email can be the case insensitive



```
StringComparison.java
20 if(email == userEmail) { // comparing 2 interned Strings
21     System.out.println("YourEmail Matched");
22 } else {
23     System.out.println("YourEmail Mismatched");
24 }
25 Scanner scanner = new Scanner(System.in);
26 System.out.println("Enter Email: ");
27 String userEmail = scanner.nextLine();
28 System.out.println("Enter Password: ");
29 String userPassword = scanner.nextLine();
30 scanner.close();
31
32 //if(email == userEmail && password == userPassword) {
33     System.out.println("Email and Password Matched. Please Enter Home");
34 } else {
35     System.out.println("Email and Password Mismatched. Please Try Again");
36 }
37
38 //if(email.equals(userEmail) && password.equals(userPassword)) {
39     System.out.println("Email and Password Matched. Please Enter Home");
40 } else {
41     System.out.println("Email and Password Mismatched. Please Try Again");
42 }
43
44 //if(email.equalsIgnoreCase(userEmail) && password.equals(userPassword)) {
45     System.out.println("Email and Password Matched. Please Enter Home");
46 } else {
47     System.out.println("Email and Password Mismatched. Please Try Again");
48 }
49
50 //if(email.equalsIgnoreCase(userEmail) && password.equalsIgnoreCase(userPassword)) {
51     System.out.println("Email and Password Matched. Please Enter Home");
52 } else {
53     System.out.println("Email and Password Mismatched. Please Try Again");
54 }
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
MyEmail Mismatched
YourEmail Matched
Enter Email:
John@EXAMPLE.com
Enter Password:
John@123
Email and Password Matched. Please Enter Home
```

By following the above steps, you have successfully explored the various methods available with string data type in Java