Lesson 02 Demo 06

Using String Methods

Objective: To explore methods available with string data type

Tools required: Eclipse IDE

Prerequisites: None

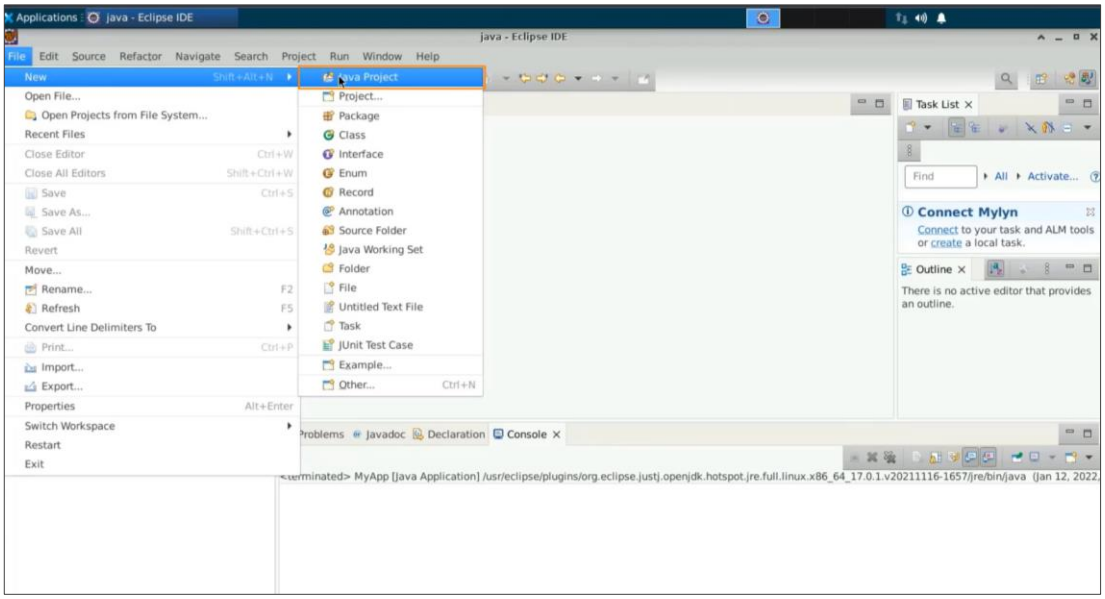
Steps to be followed:

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

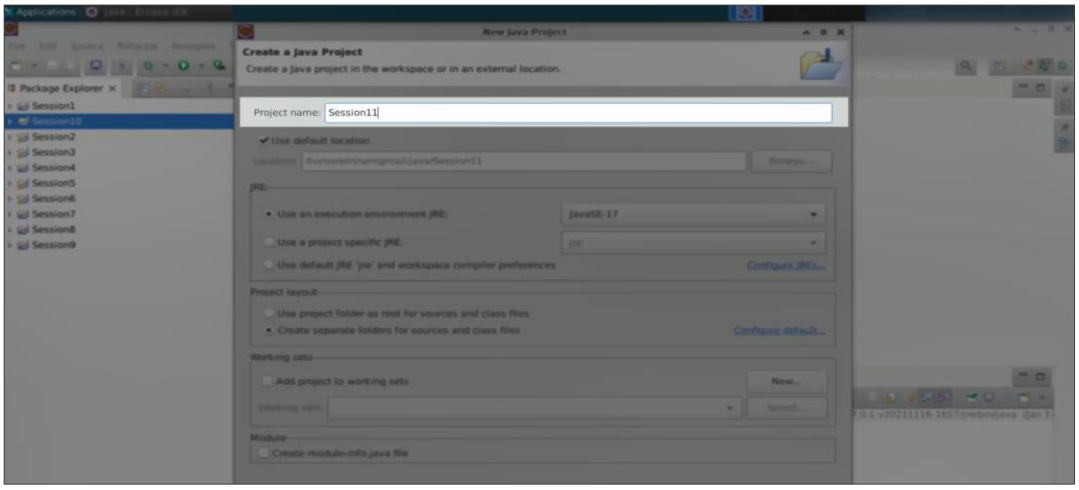
Step 1: Write a string and calculate the number of characters in the string

1.1 Open the **Eclipse IDE**

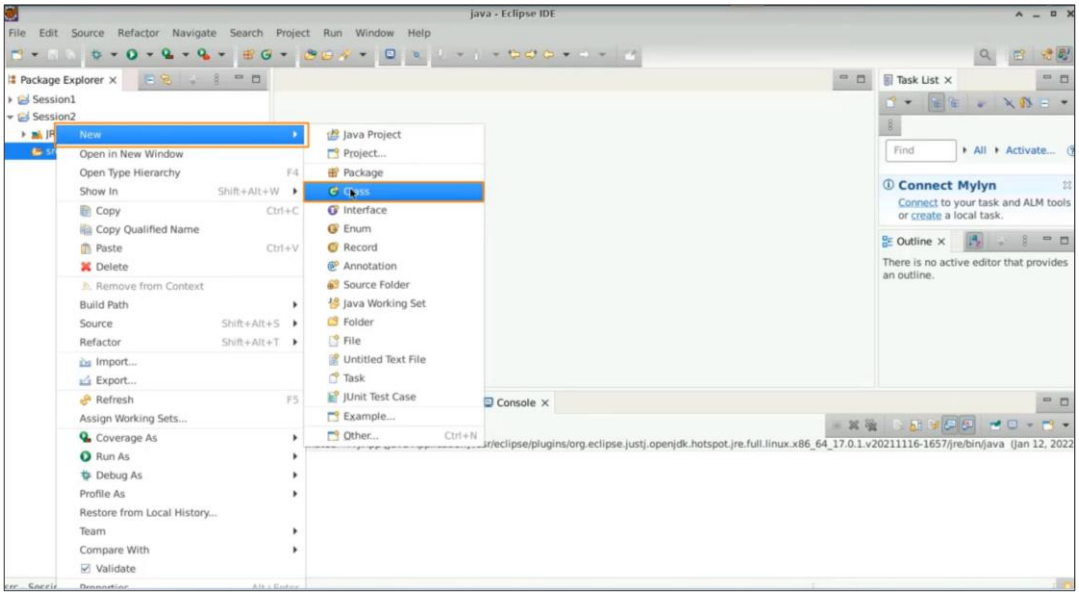
1.2. Select **File**, then **New,** and then **Java project**



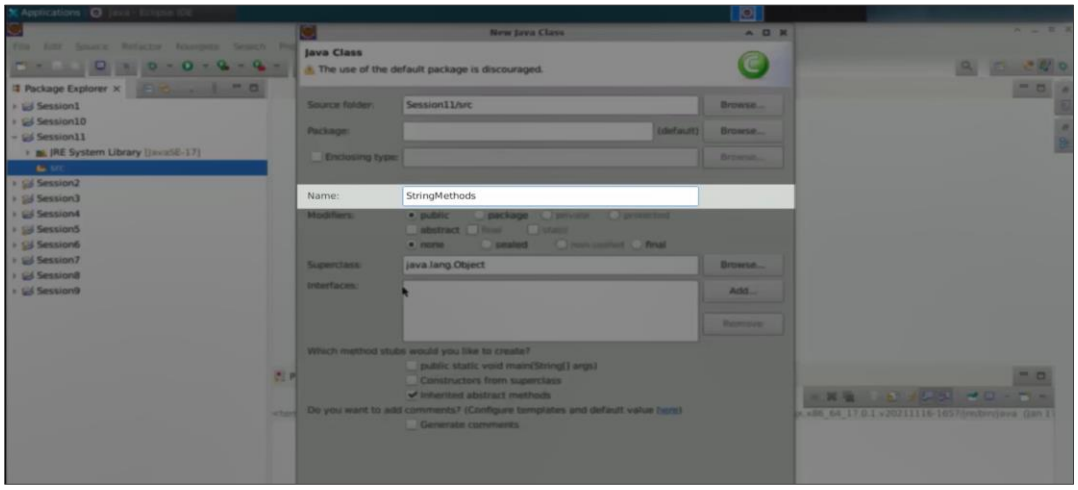
1.3 Name the project **“Session11”,** uncheck **“Create a module info dot Java file”**, and press **Finish**



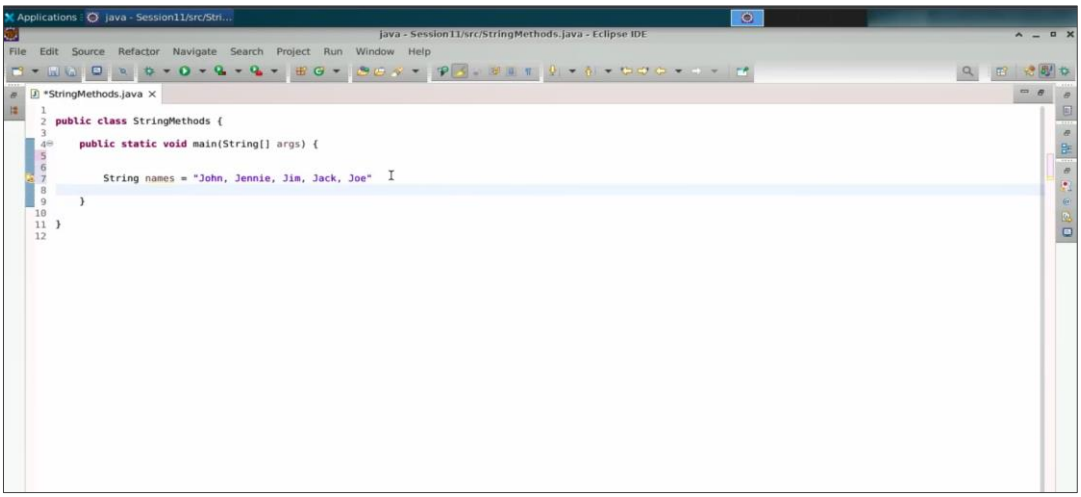
1.4 With a **Session11** on the src, do a right-click and create a **new class**



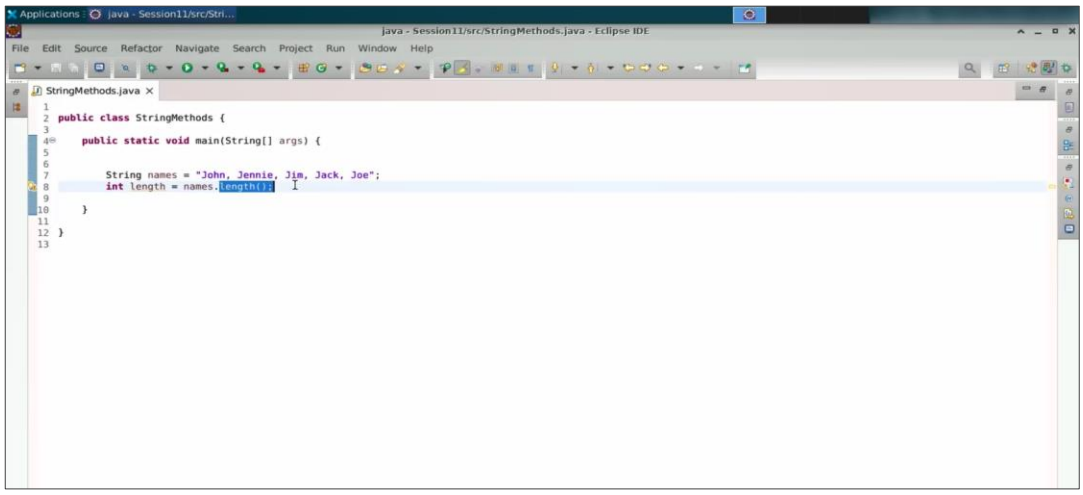
1.5 Name this class as an **StringMethods**, then select the **main method,** and then select **finish**



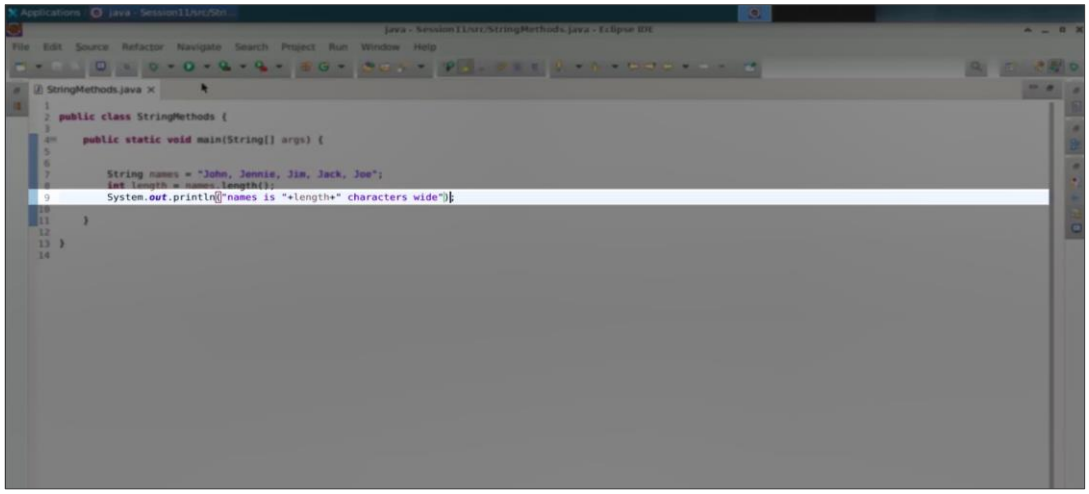
1.6 Let us write one string called names and these names are John, Jenny, Jim, Jack, and Joe. This is altogether a single string



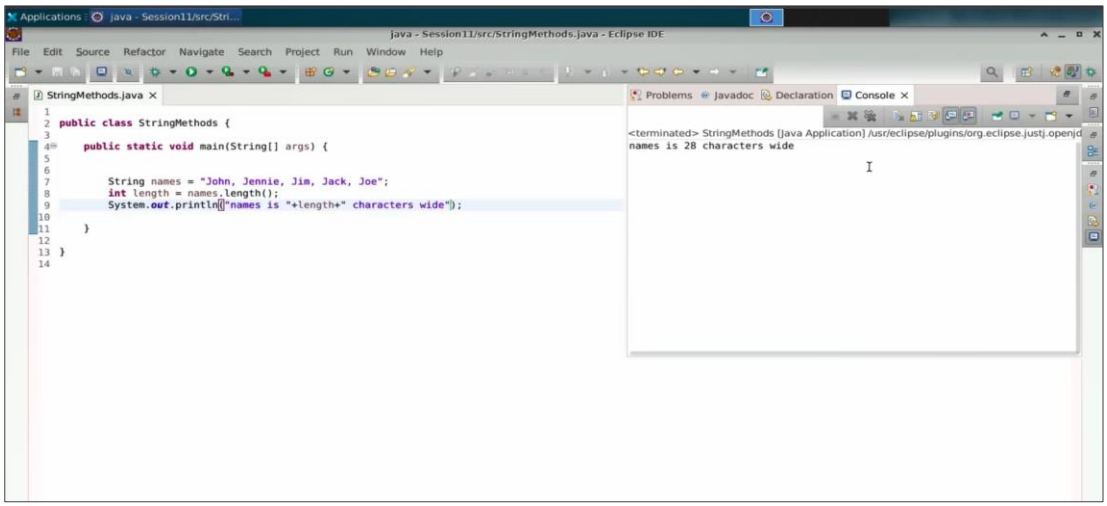
1.7 Add **int length = names.length();** to execute a function called length. The length method is available as a built-in method



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

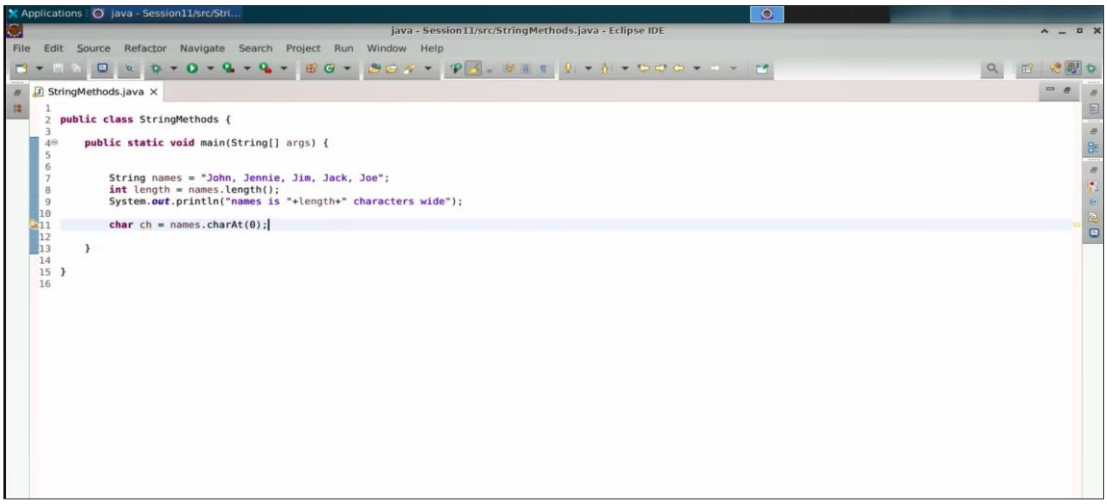


1.9 Run the code. It says that name is 28 characters wide

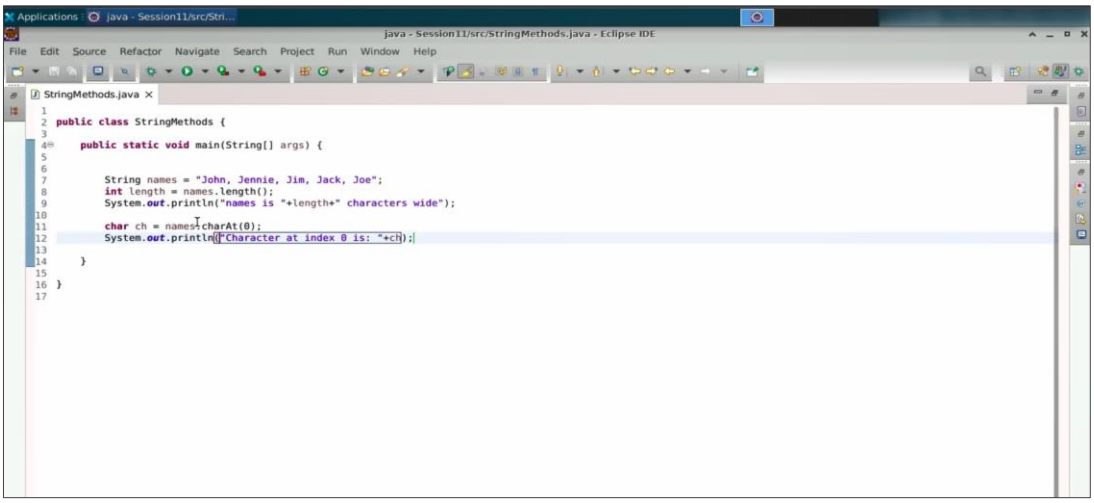


Step 2: Get the index of characters

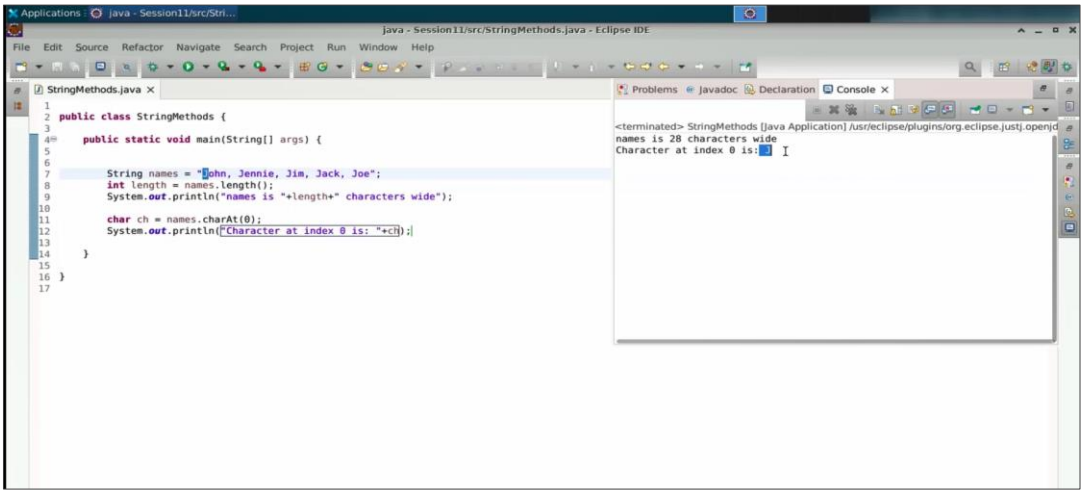
2.1 A char called Ch is going to be named dot character at the index number 0



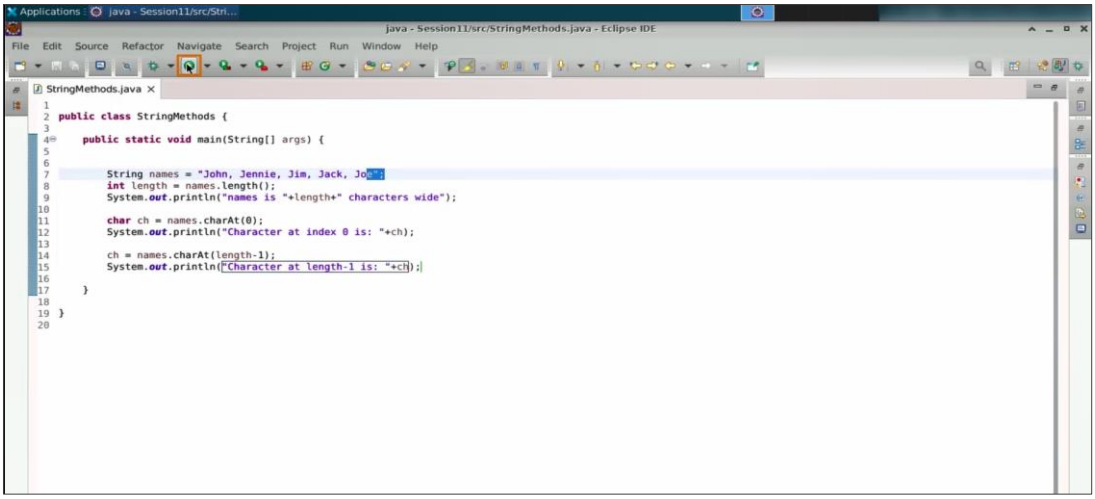
2.2 Now the character at index number 0, is very much clear, let us say character at Index zero is plus Ch. When you say char, this is a method with the help of which you can get the character from a string



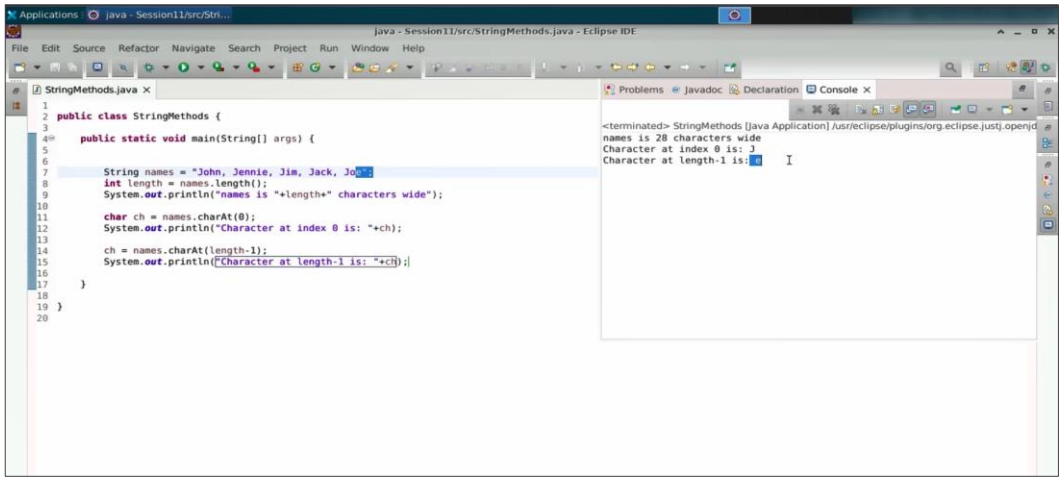
2.3 The 0 character is this upper-case J and when you run the program here you get to see it gives the value as J



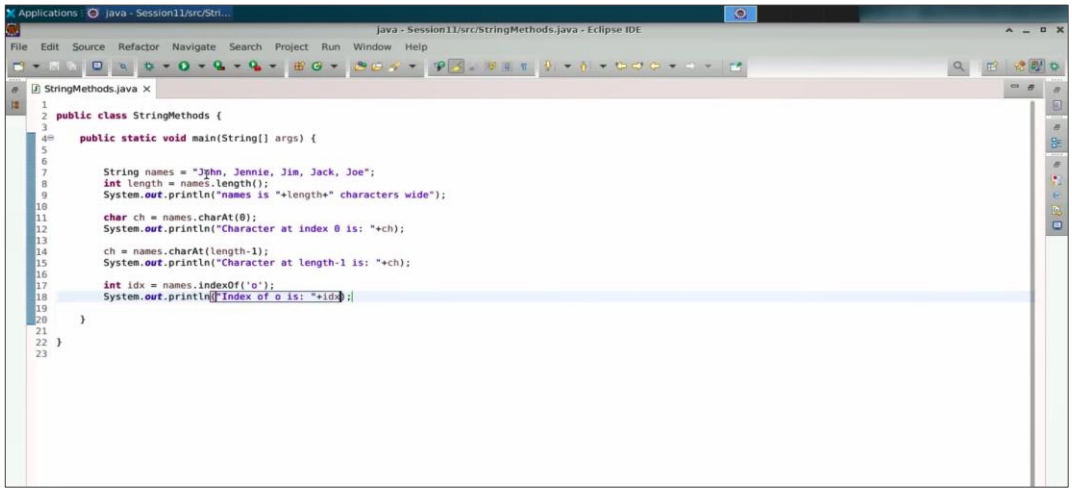
2.4 Let us also say ch is **names.charAt(length - 1).** Then add **System.out.println("Character at length - 1 is " + ch);**



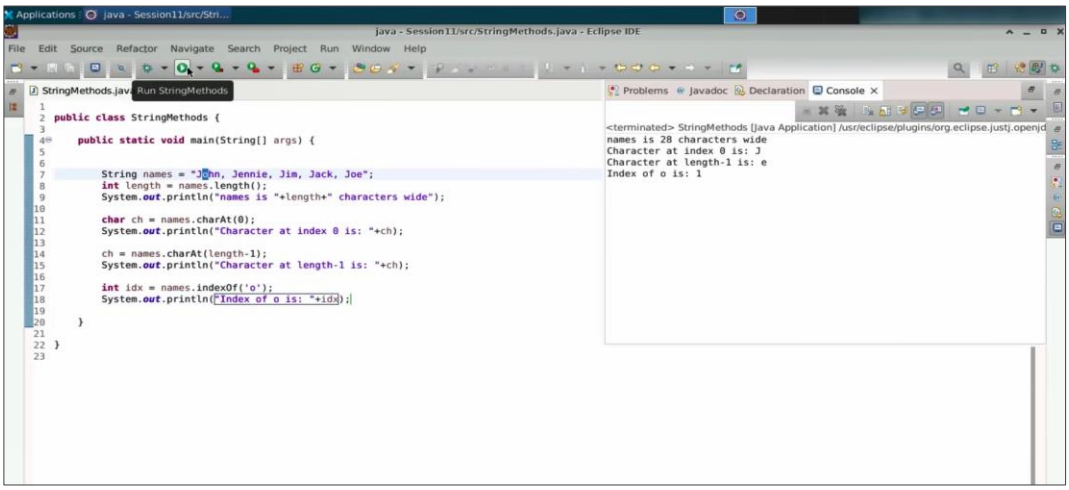
2.5 When you run the code, you will see that the character at **length - 1** is **e**



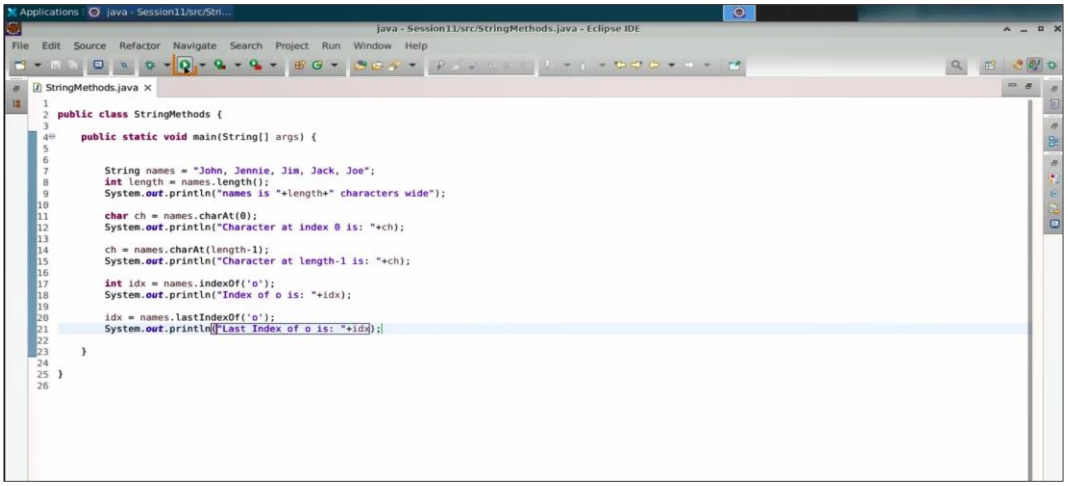
2.6 Type **idx = name.indexOf('o');**. Let us see the index of a character called '**o**'. The character 'o' is available at the first index as well as the second-to-last index. The **indexOf** method will give you the index of the character from the beginning. So, when you type **indexOf('o')**, it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



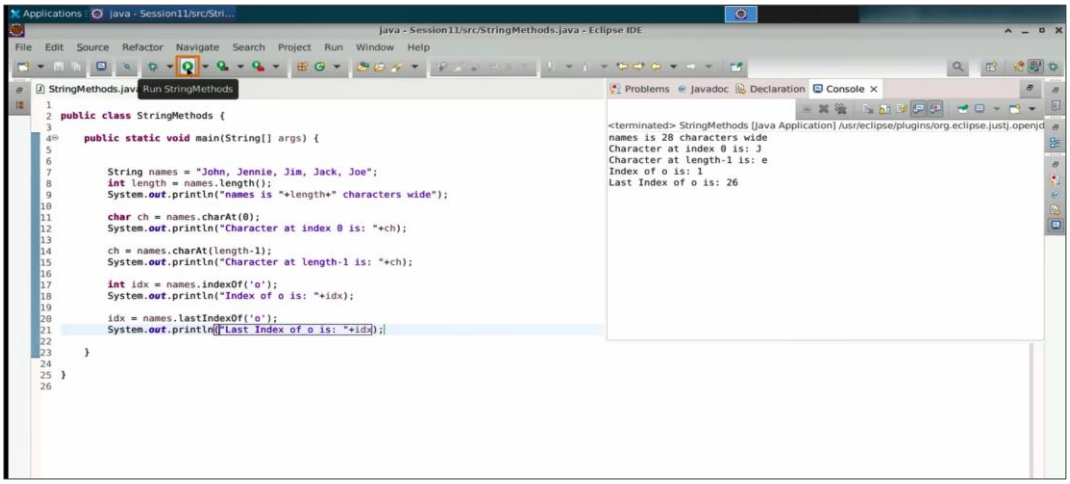
2.7 Run the code and you can see an index of o is 1



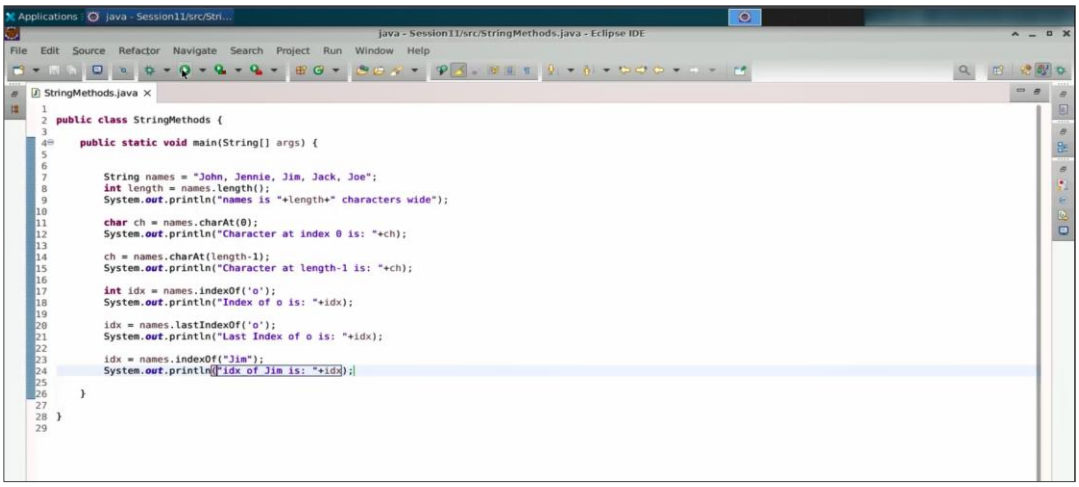
2.8 You can also add **idx = names.lastIndexOf('o');**. There is another method called **lastIndexOf**, which will give you the index of the character from the end of the string



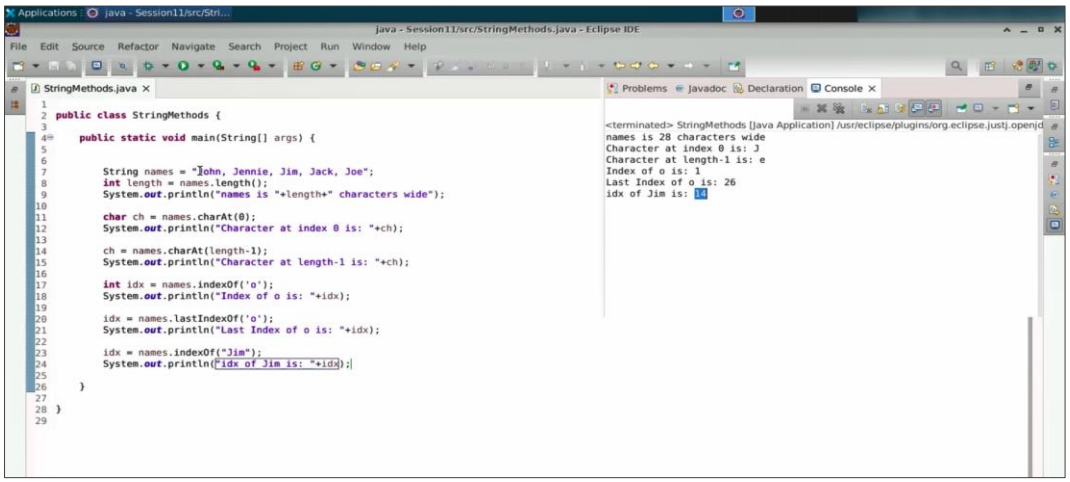
2.9 Run this code: **length - 2**. The length is 28, so the index of '**O**' is **28 - 2**, which is **26**



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

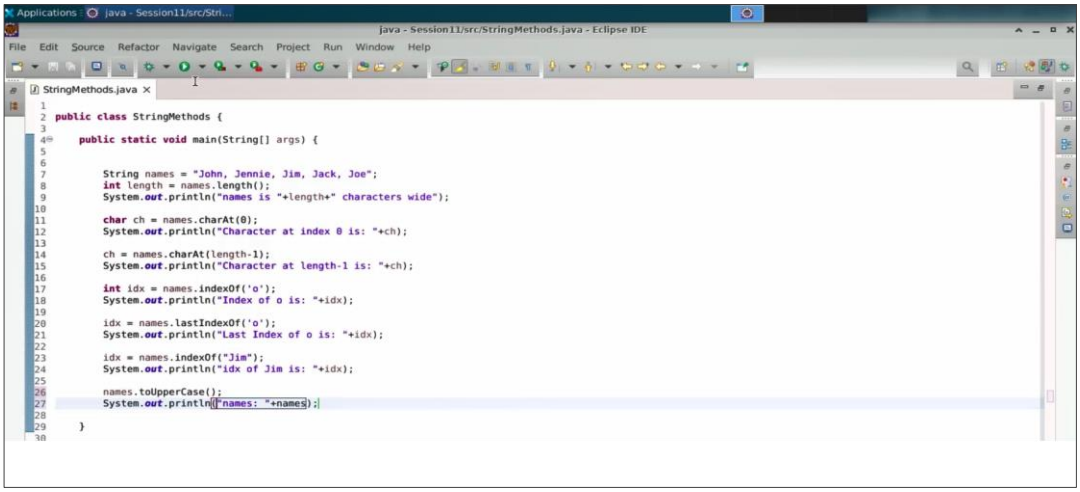


2.11 Run the code. Now, this is where Jim's J starts, so that's like 14.

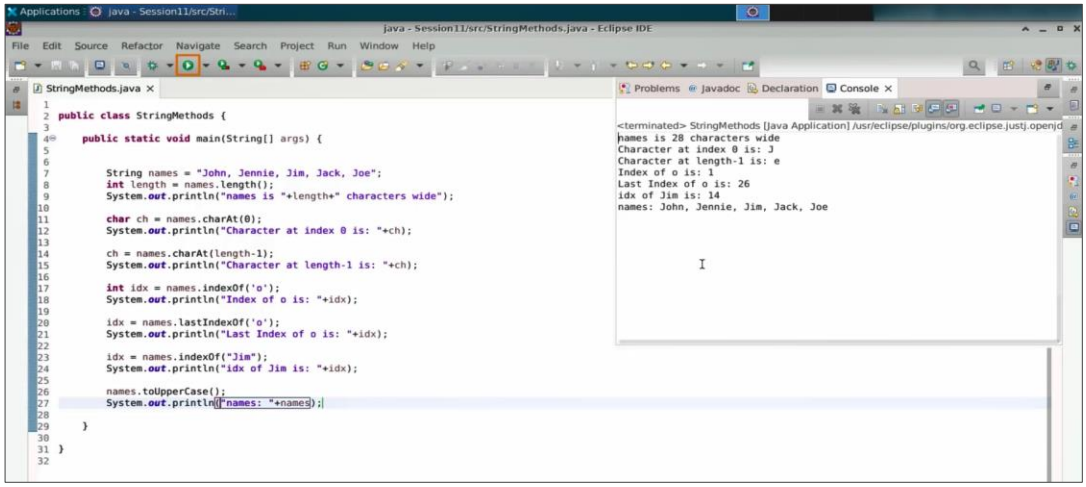


Step 3: Use the uppercase method to convert the string to uppercase

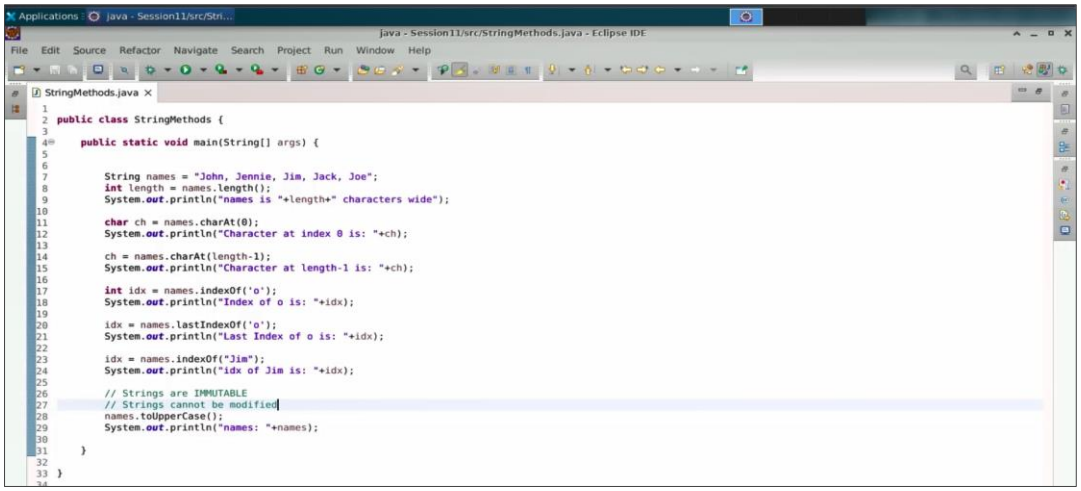
3.1 Type **names.toUpperCase()**. The method **toUpperCase** is very clear in its function; it converts the string to uppercase. Use the print command to print the original names variable followed by the uppercase version



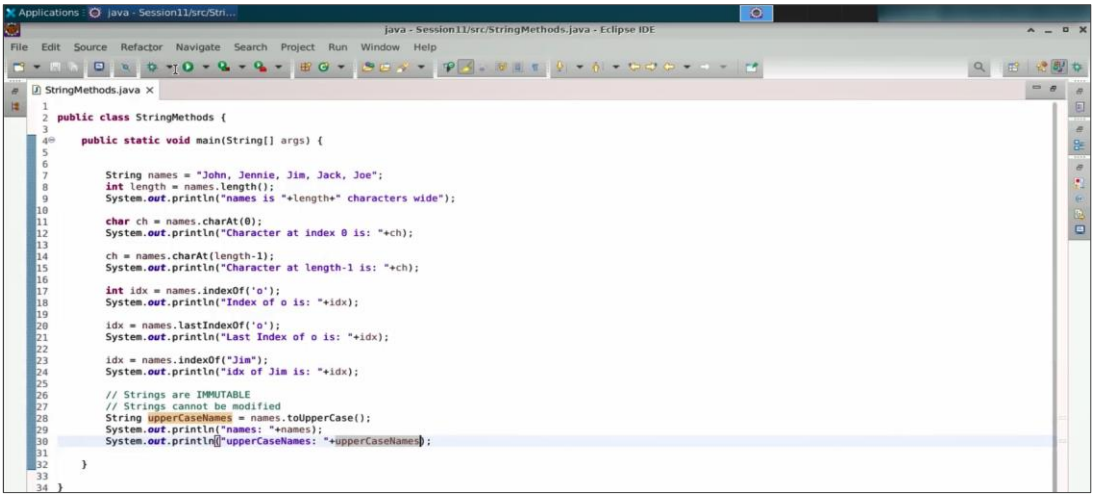
3.2 Run the program here, you will get to see the names are still the same. There are some methods that are meant to do the manipulation on those strings



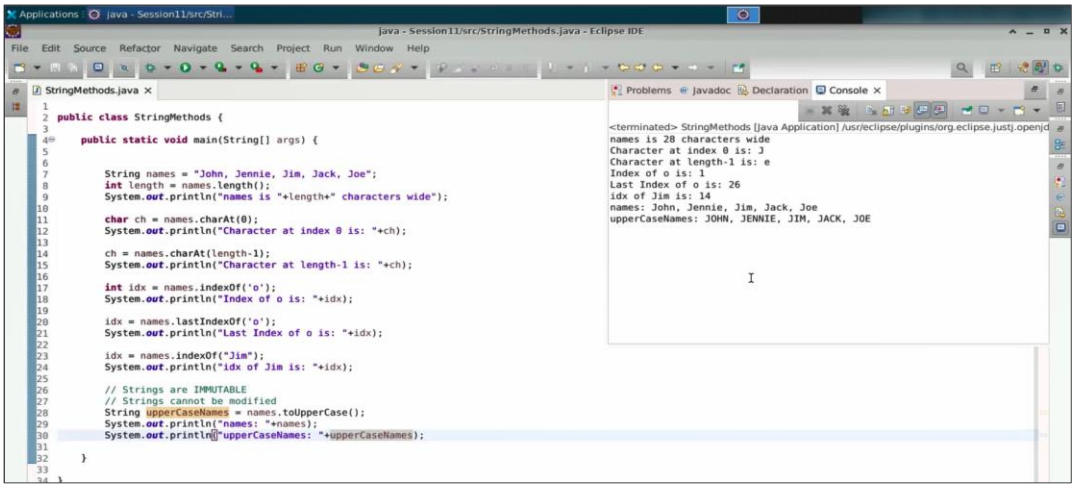
3.3 Now here you need to understand the fundamental, which is strings are immutable. Immutable means that strings cannot be modified



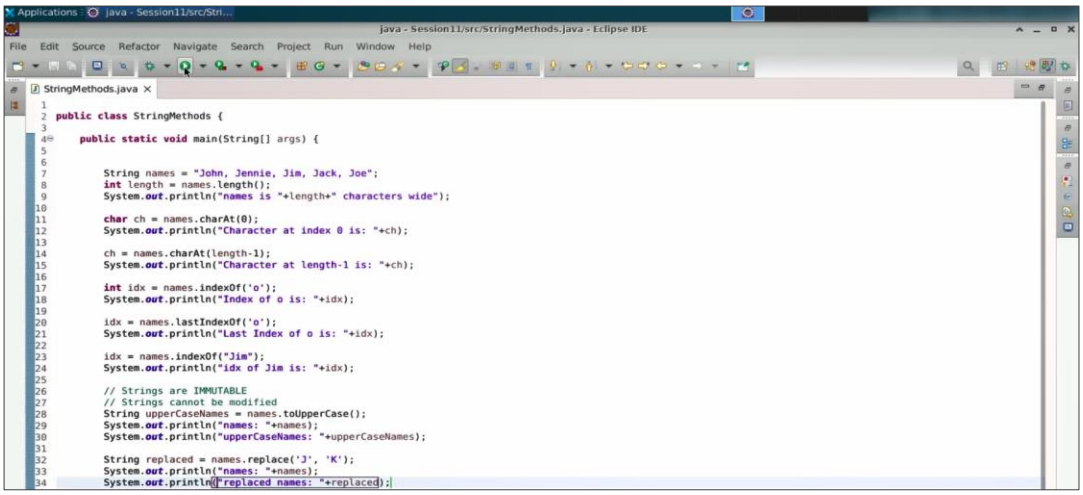
3.4 When you type **names.toUpperCase()**, it creates a new string in memory and returns it. Hence, you will have another string called uppercaseNames (or any name of your choice). If you print the string called **uppercaseNames**, it will display the converted string



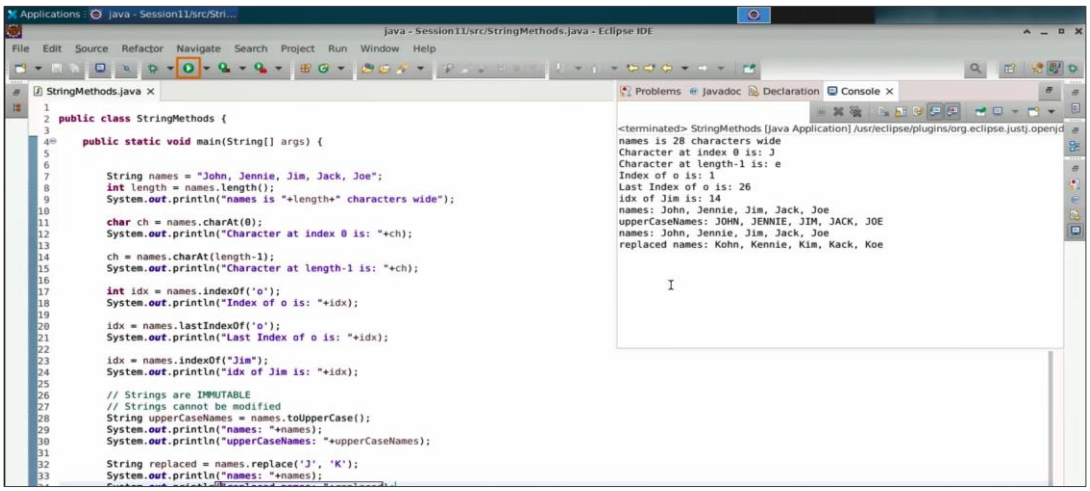
3.5 Run this code what you see is that the upper-case names are basically containing the characters in the upper case, so this is a return string through to the uppercase method



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

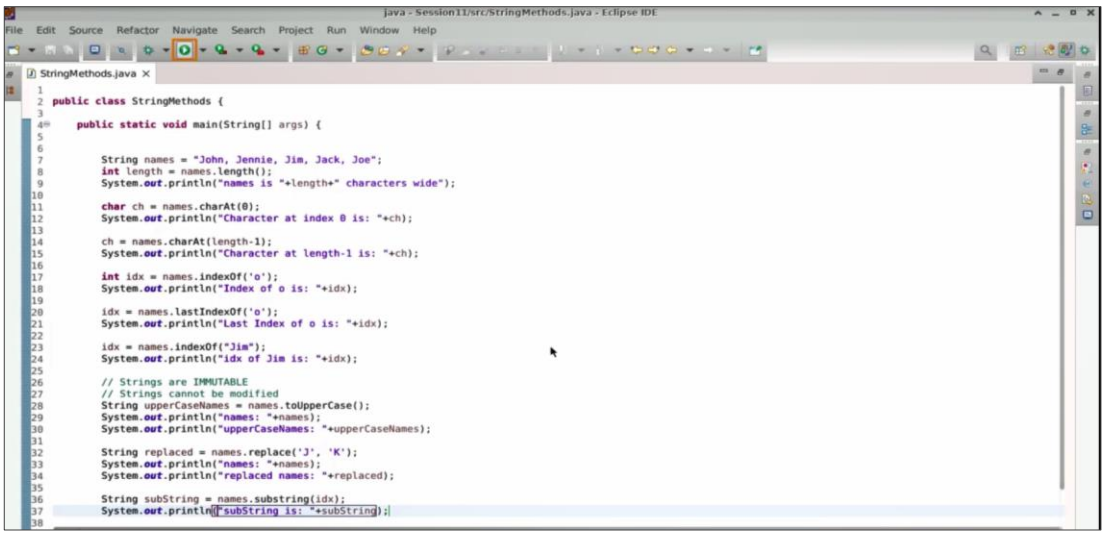


3.7 Run this program, and you get to see the data is manipulated. But the names are not at all changed

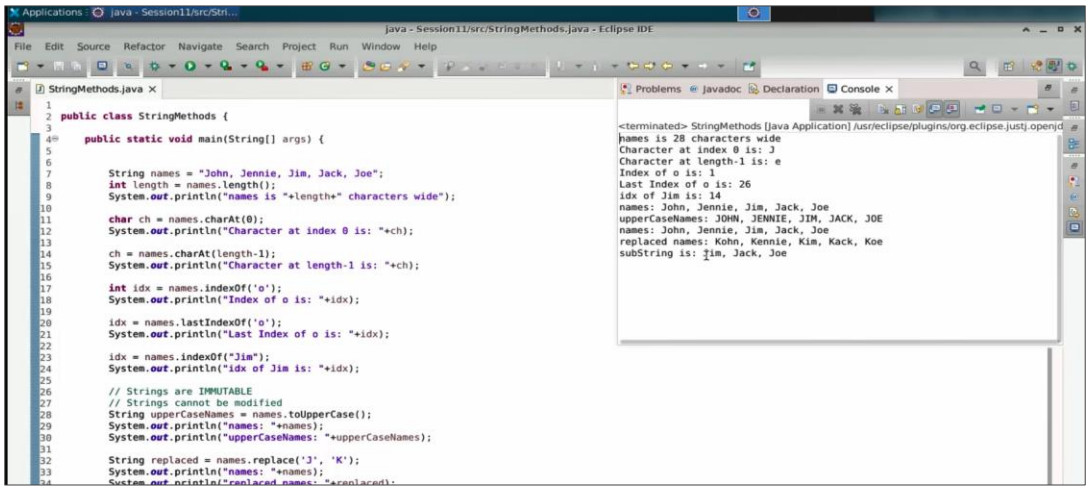


Step 4: Implement string slicing and extracting sub-strings

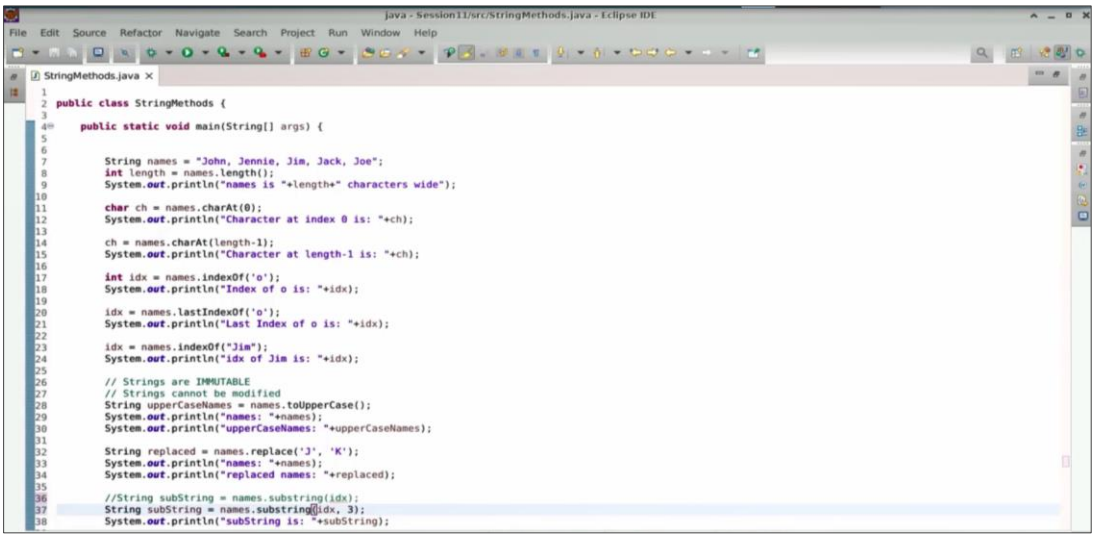
4.1 Now you are going to extract a substring from a string, which is called string slicing. Type **names.substring** and pass the starting index of '**Jim**'. This will create a new string in memory. Let us name this substring '**substring**' and store it at the specified index. Print out the substring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



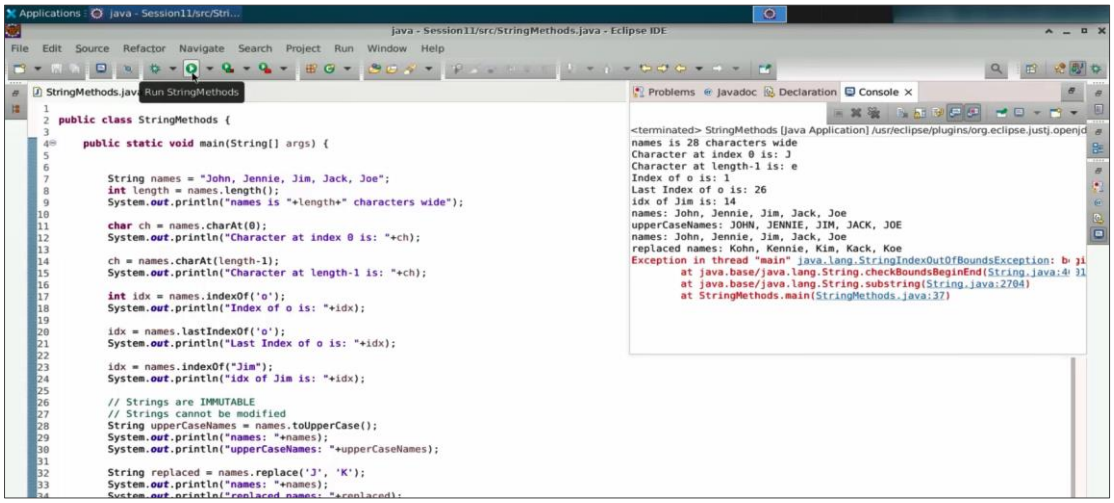
4.2 Run the code. A substring with one single input will simply slice your string from that index till the end



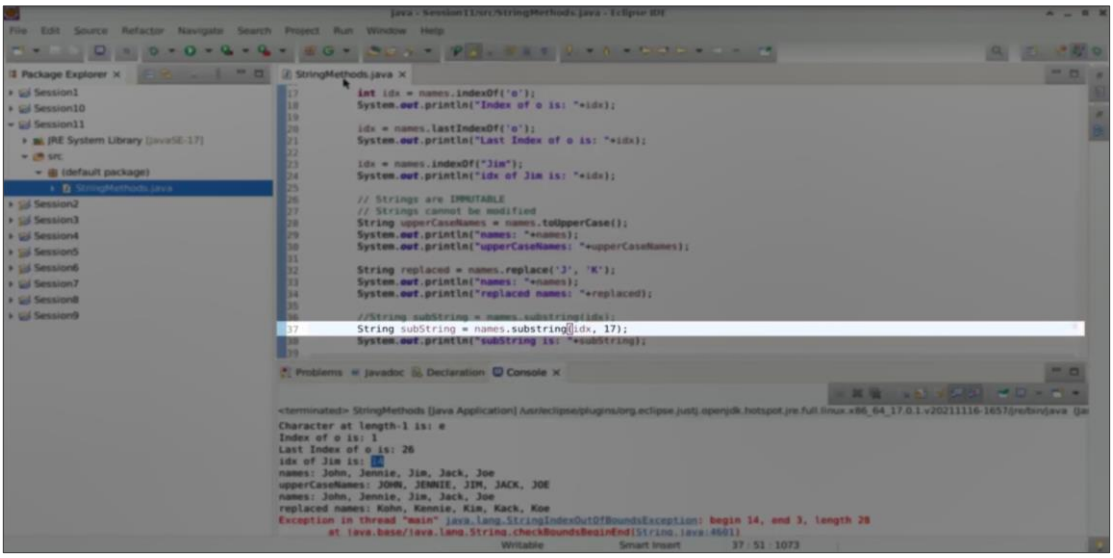
4.3 Now try to write one more usage of this substring method, which is an overloaded method, you can pass the index, and then you can pass a number called 3



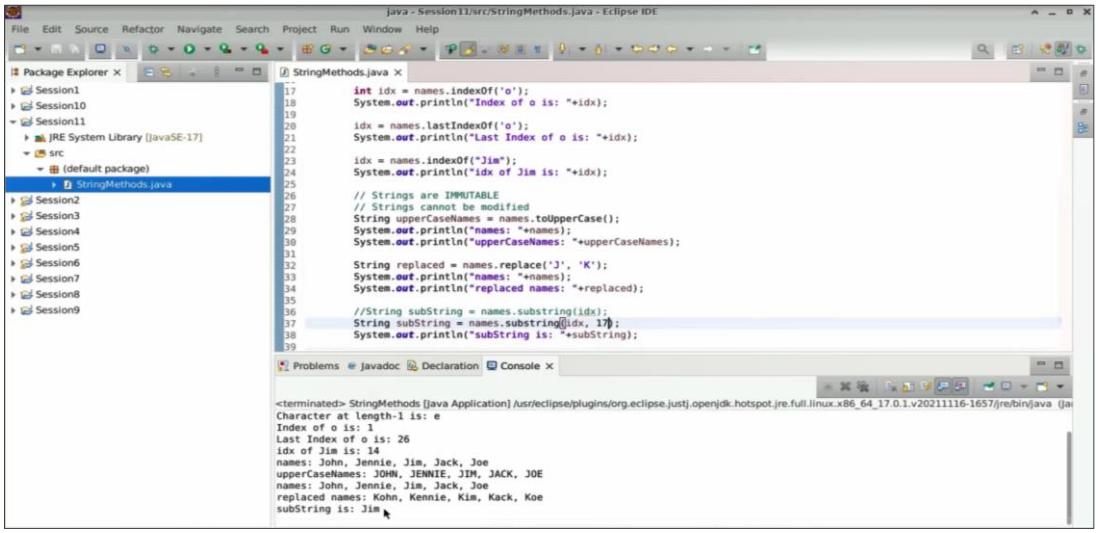
4.4 When you run this code. You get to see that there is an error. Now the index is the beginning point and the second input is the ending point



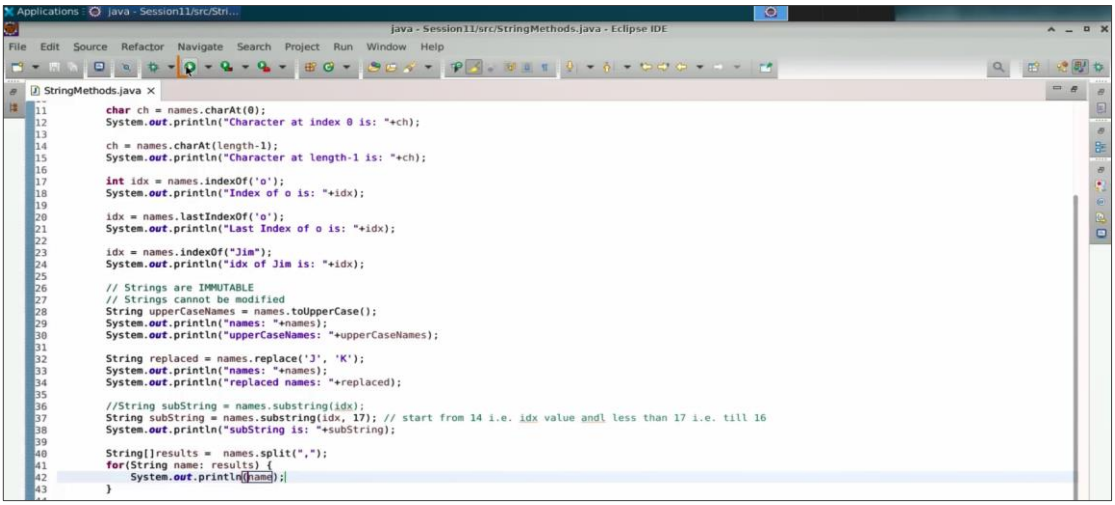
4.5 The index of Jim over gives the output as 14, which means the next index cannot be less than 14. It must be larger than. You're going to add 17



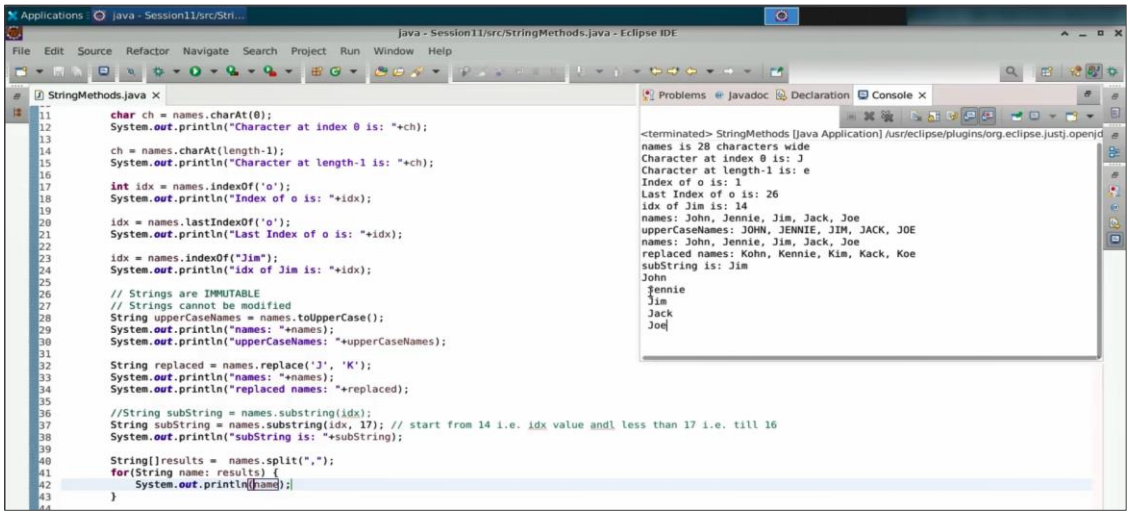
4.6 Now run the same program and see the entire word called Jim being extracted from the string. It started with the 14, then the 15, and then the 16. 17 is not inclusive. It is less than 17



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(",");**

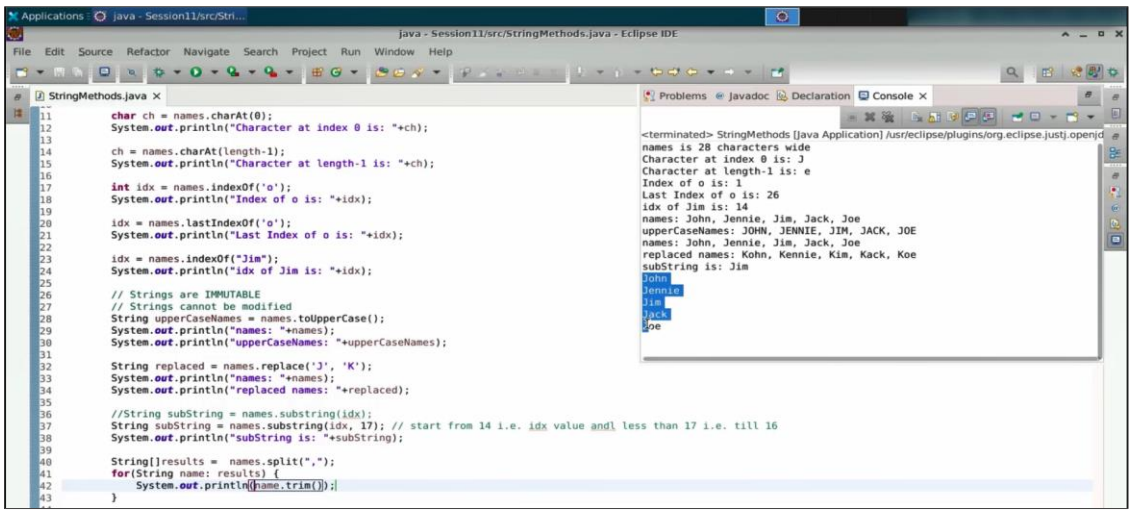


4.8 Run the code and it gives the data as John, Jenny, Jim, Jack, and Joe. But you can observe that there is a space coming in the leading of the string. The reason is that whenever you are trying to come up and perform a split that was based on a comma, you do have a space attached to it

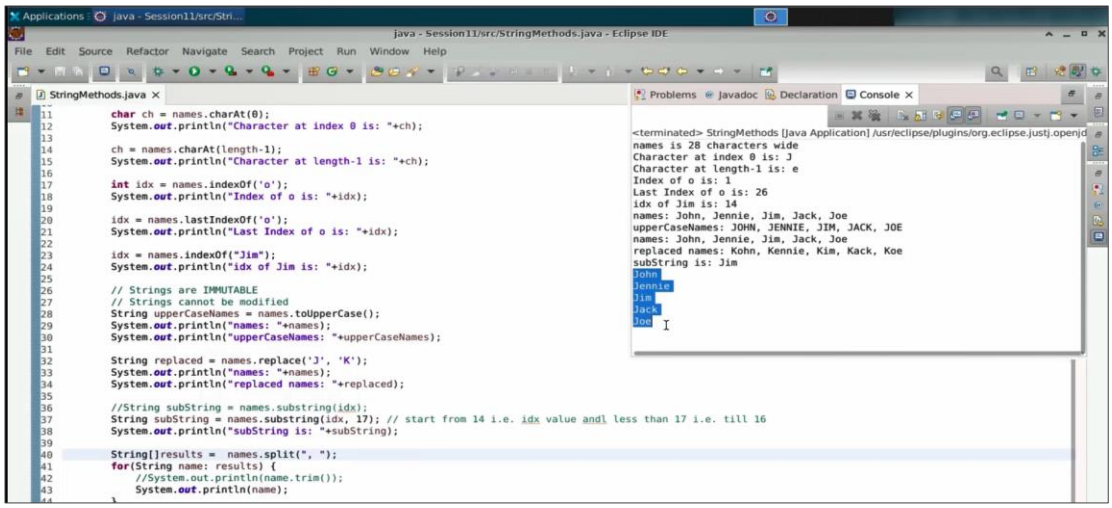


Step 5: Use the trim function on strings

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

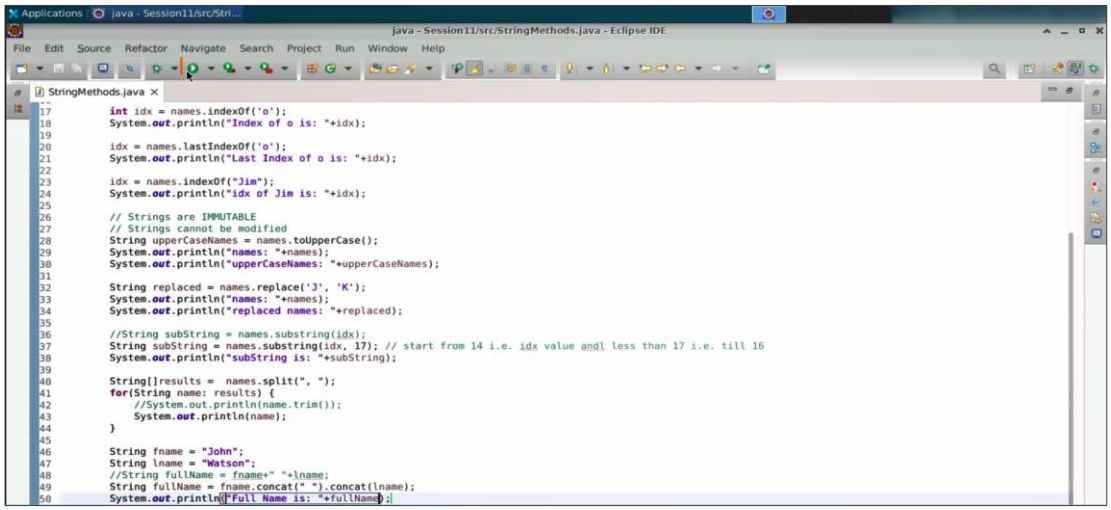


5.2 The other way is to perform a split right based on a comma with a space. when you do a split as comma space together, it will split based on comma and space. This can even be a regular expression



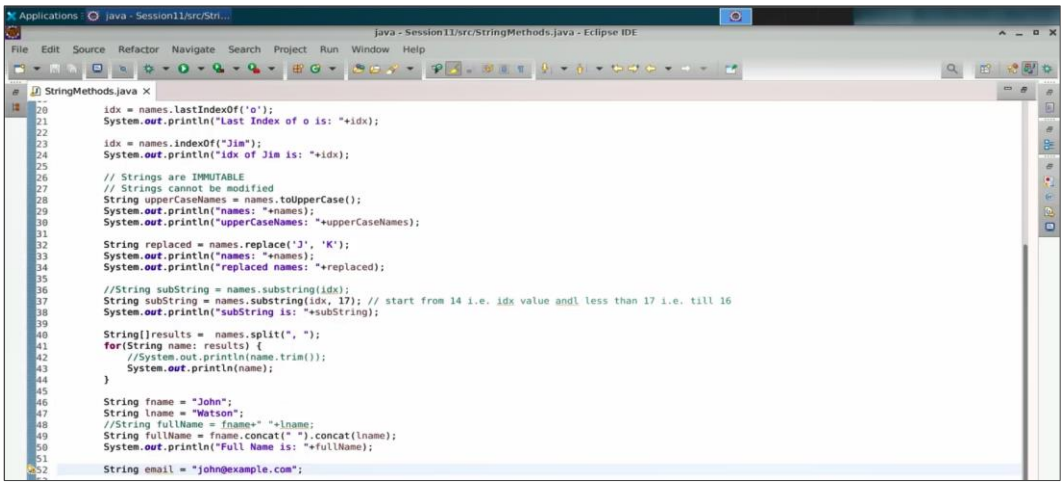
Step 6: Implement the concat method for concatenation of strings

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**

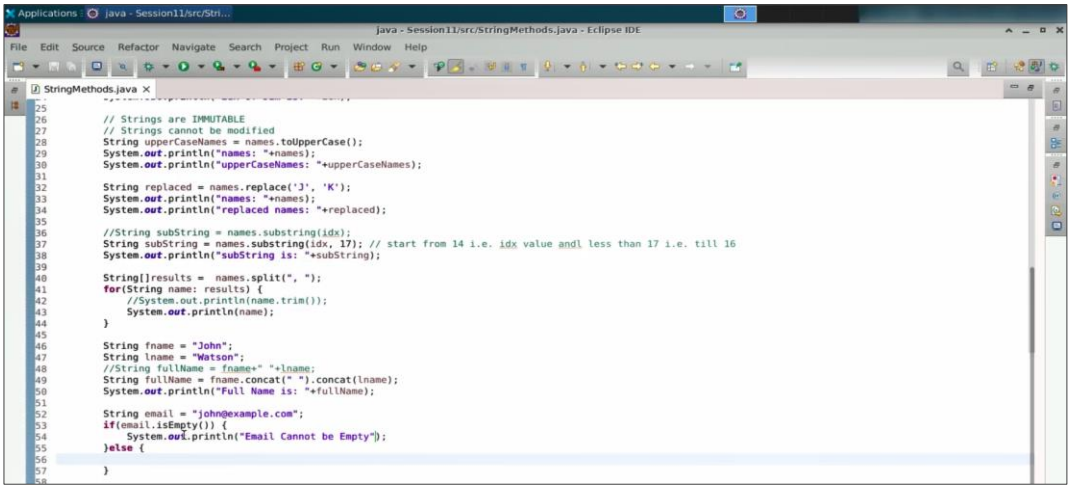


6.2 Now let us say that there is an email. And this email is **john@example.com**. You need to validate that whether this email is correctly formed or not

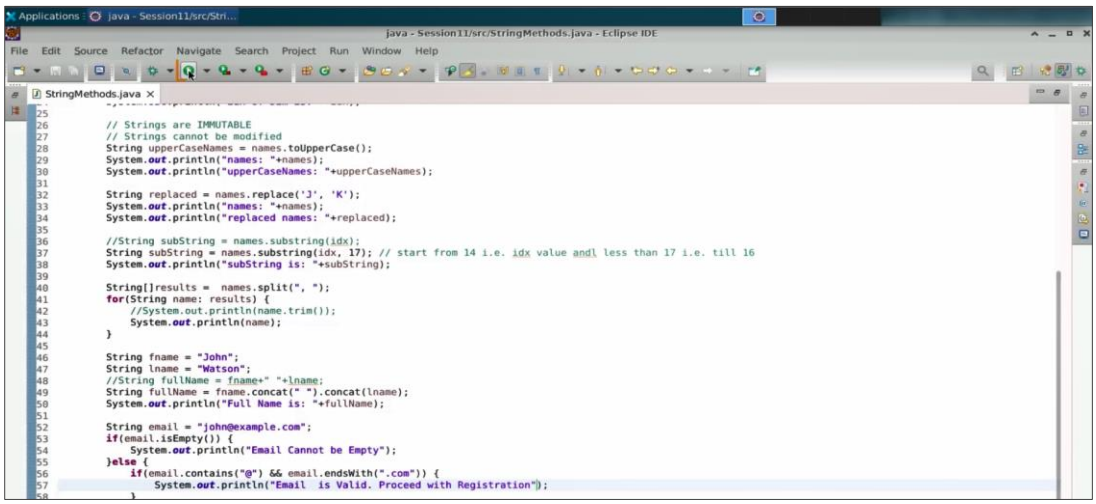
.



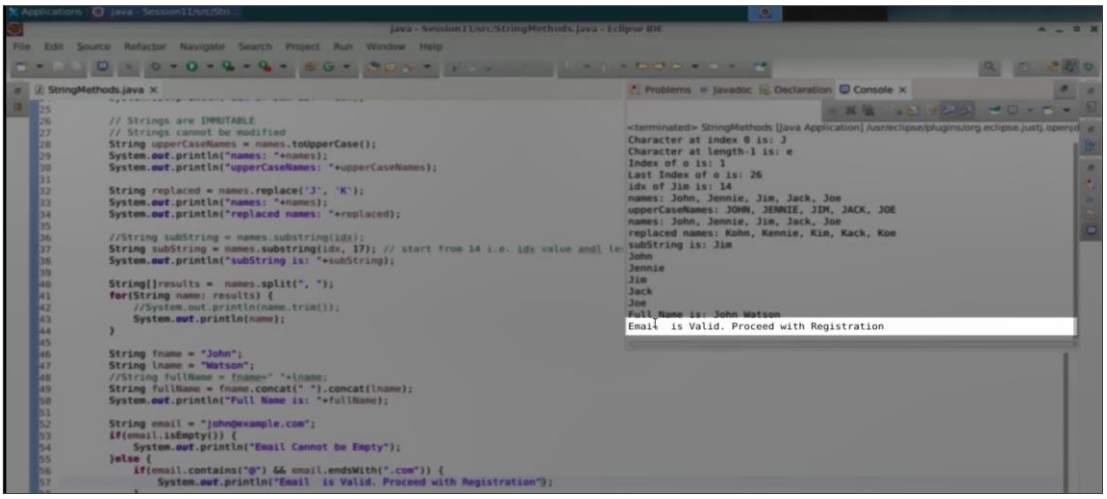
6.3 You are going to put a condition: **if email.isEmpty()**, print "**Email cannot be empty**". Otherwise, handle the else case



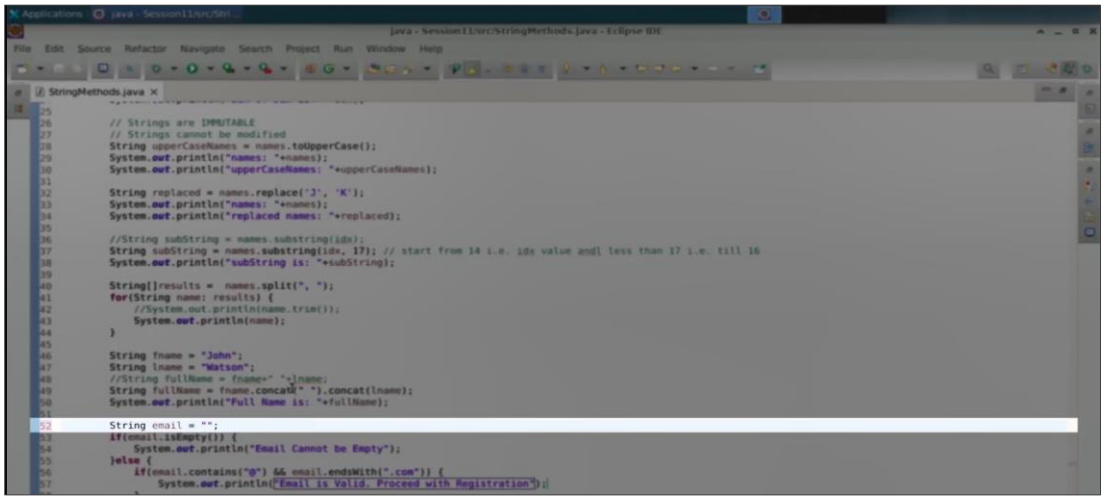
6.4 If the email is not empty, it will check whether the length is 0 or not, and accordingly, it will return true or false. Now, if the email is not empty and has some data, you are going to check if the email contains the character '**@**' and ends with '**.com**'. These are the two conditions you will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



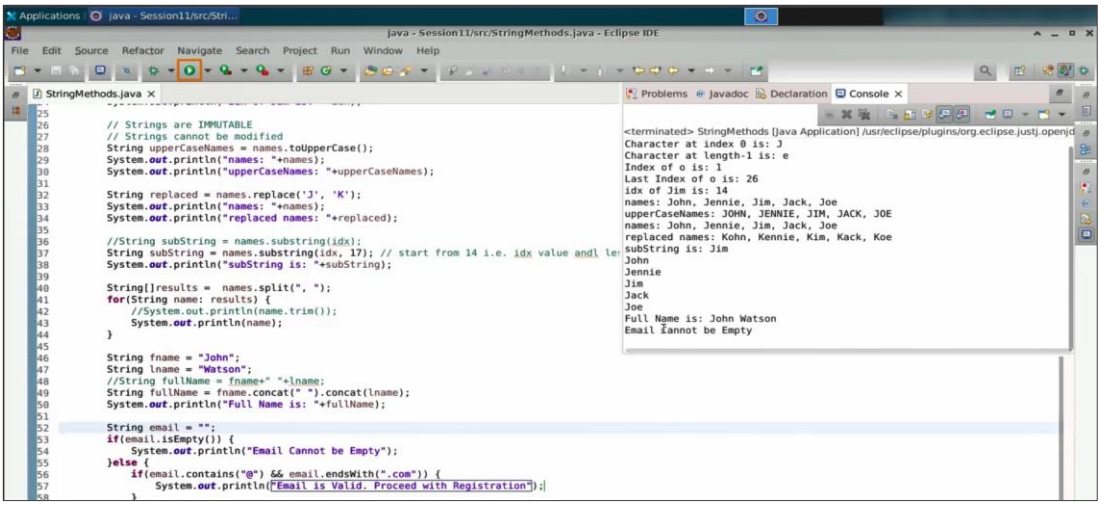
6.5 Run this program, now, it says that the email is valid, and proceed with registration.



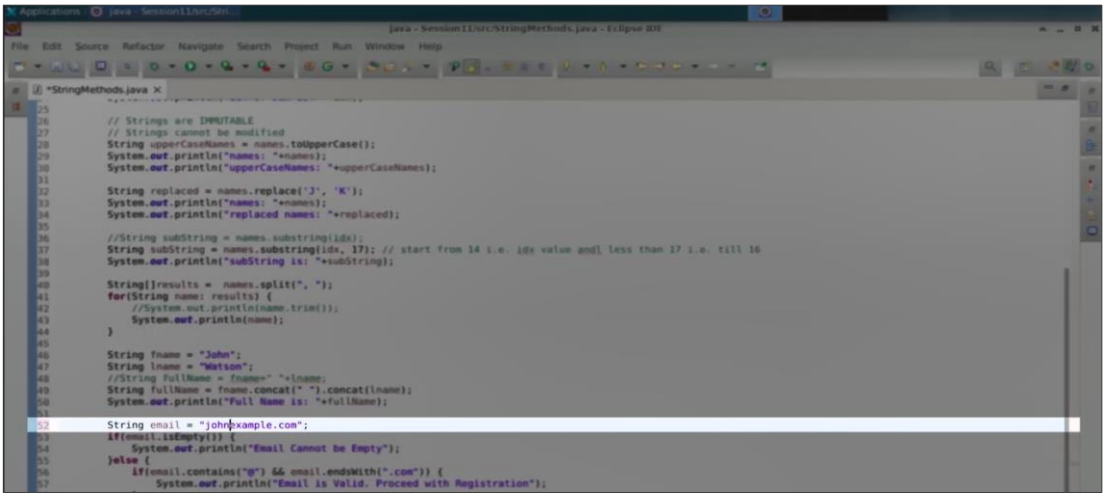
6.6 Let us try to say that the email goes something like a blank



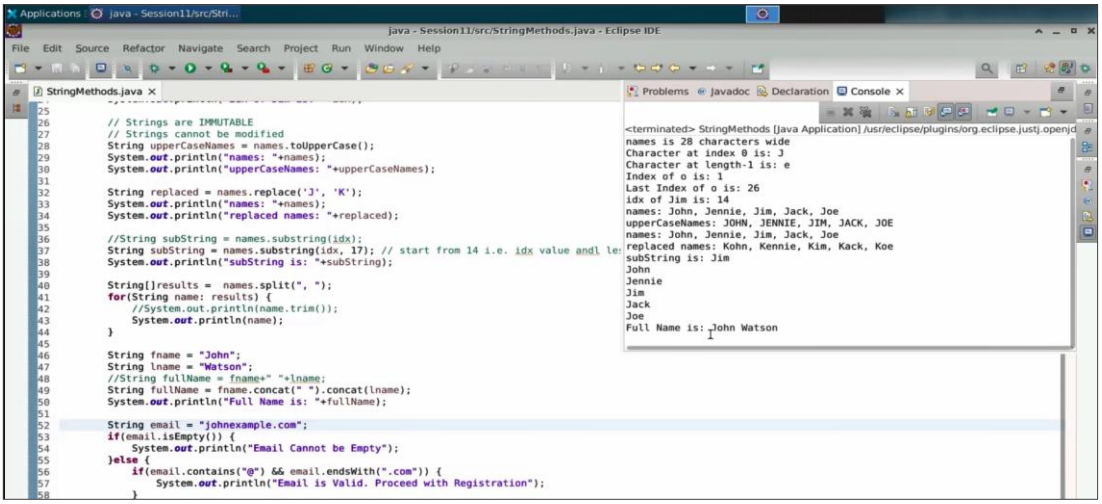
6.7 Run this program, now, it says Email Cannot Be Empty



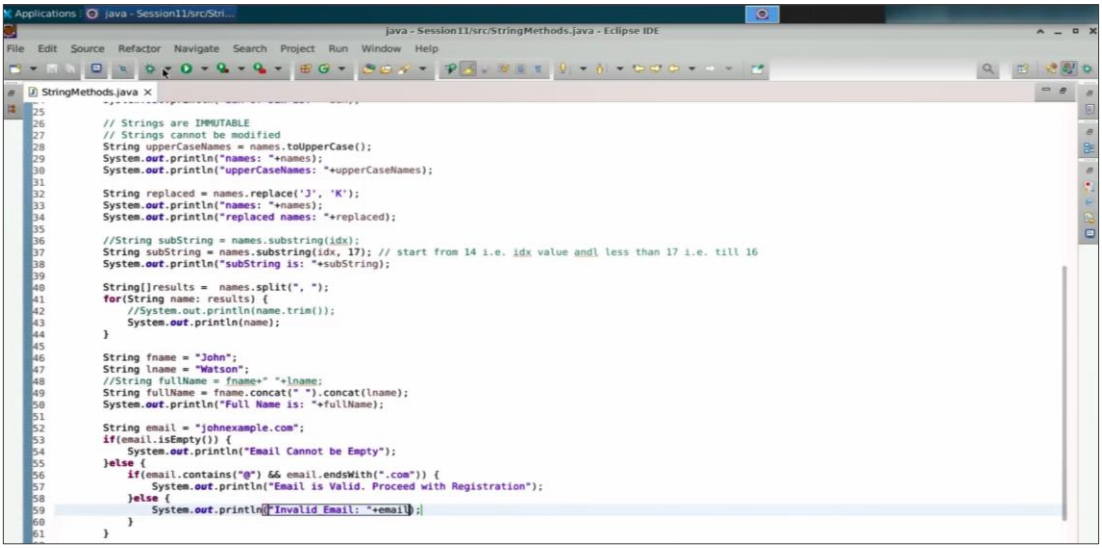
6.8 Let us simply remove this **@**. So, contains check anything as a part of your string



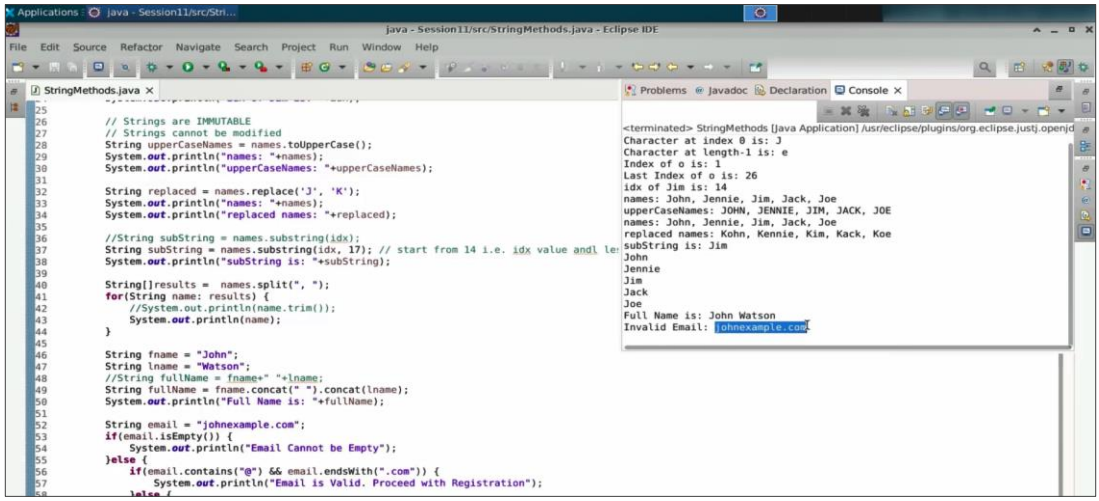
6.9 Run this program. Now, you can see else part is not written



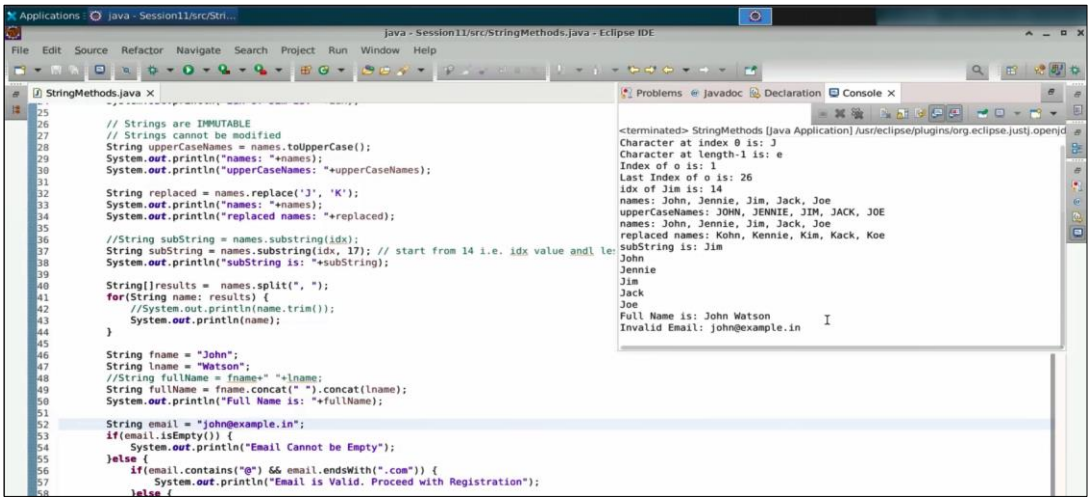
6.10 You can once again come up and say that this is an invalid email. You can even print the variable



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

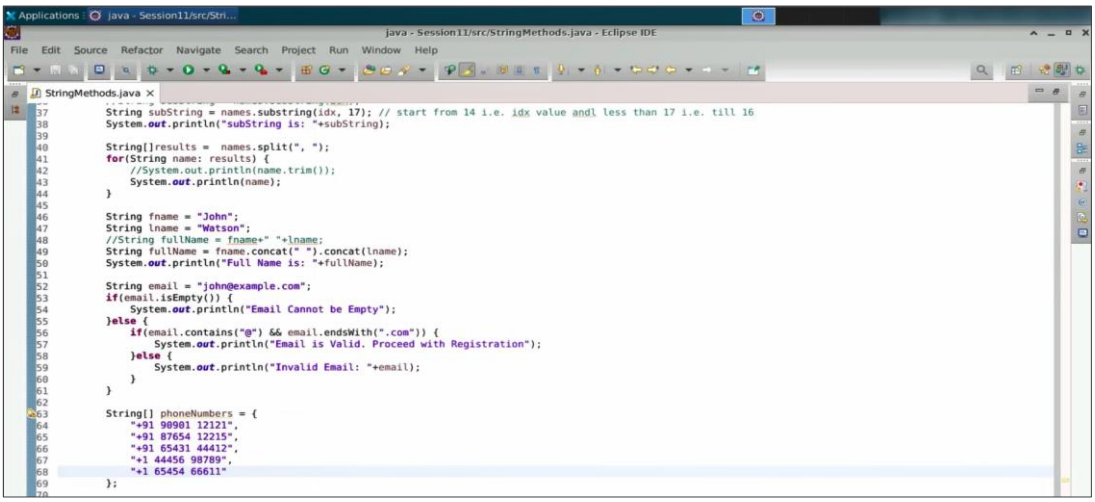


6.12 Let us have '**@**' at the same location, and you can eliminate the '**.com**' condition, or you can simply say this is not valid. However, as per the condition, it should end with '**.com**', so it will say "**Invalid email**". This clarifies what is meant by the **endsWith** method

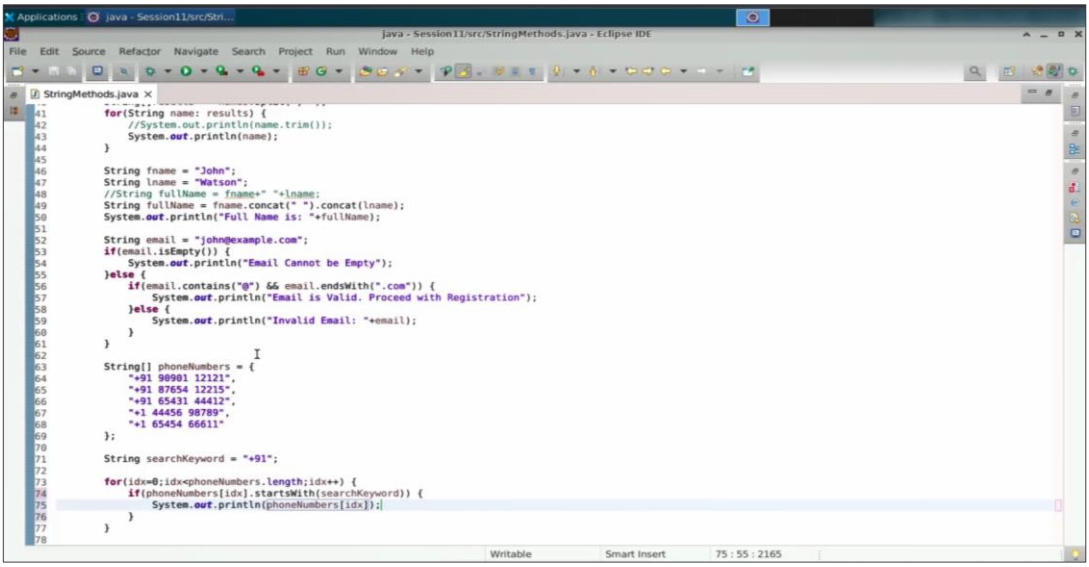


Step 7: Validate the strings using the methods ends, contains ends with, and starts with

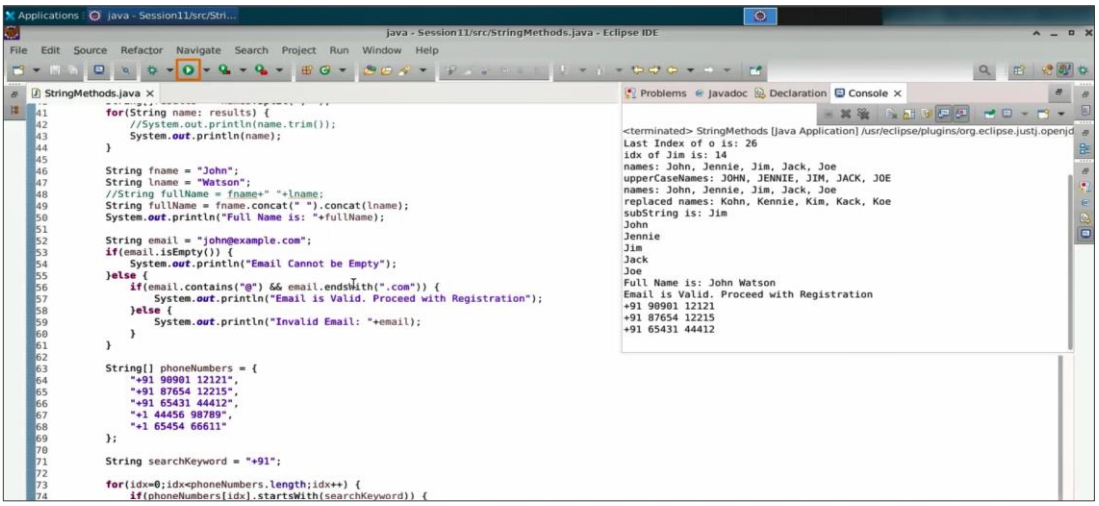
7.1 Let us try to explore something more on this string method called contains ends with and starts with. You are going to create a list of contacts or the phone numbers available in your address book. You have this array of phone numbers and you want to perform a search operation



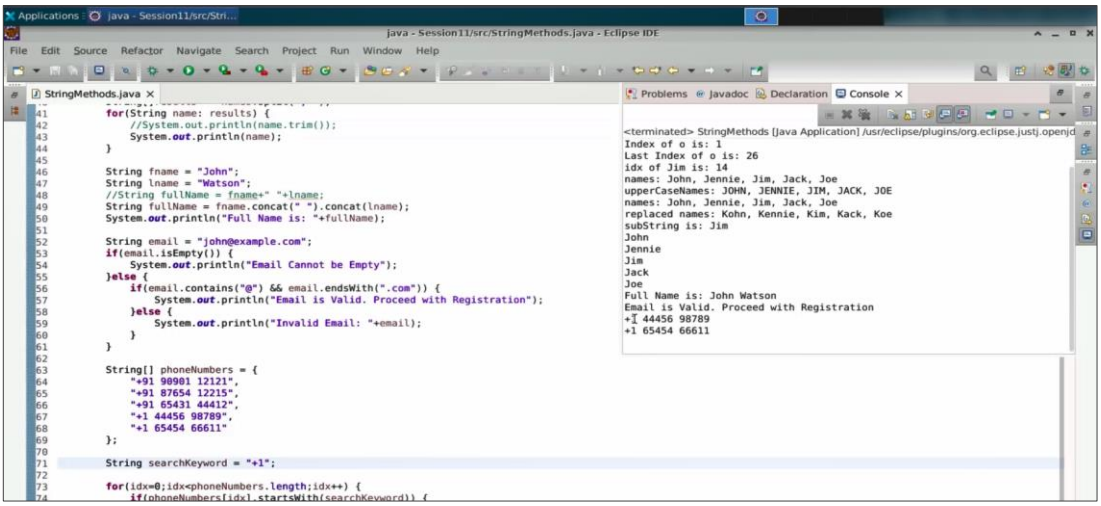
7.2 Let us say there is a search keyword, and it is +91. You want to filter the phone numbers. Put a loop that starts with the index value 0 and continues while the index is less than **myPhoneNumbers.length**. Use **idx++** to increment the index. Now, if the phone number at the current index starts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



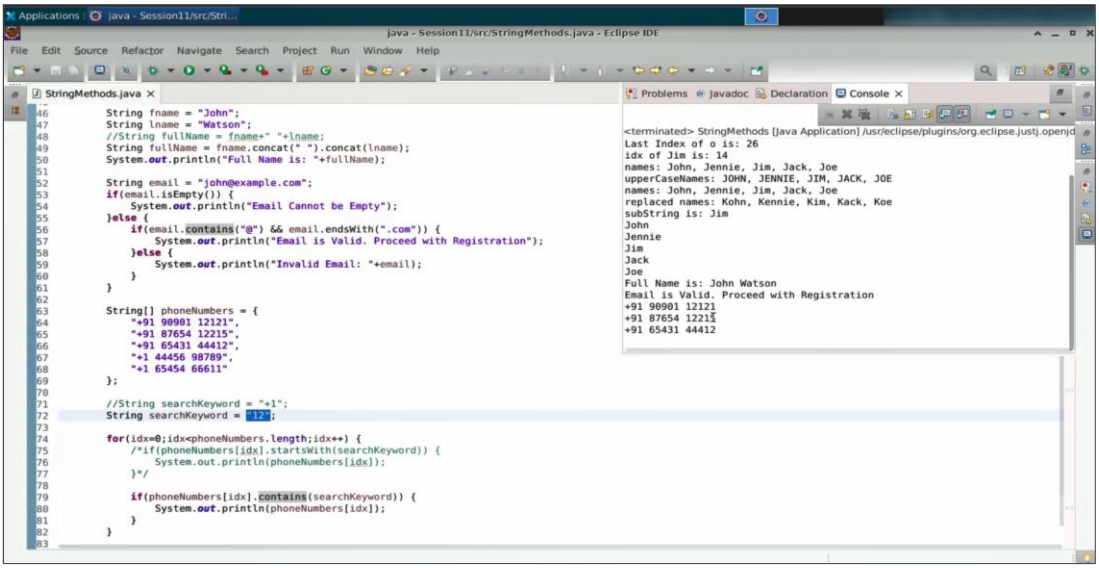
7.3 Run this program, and it gives you all the phone numbers that start with +91



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

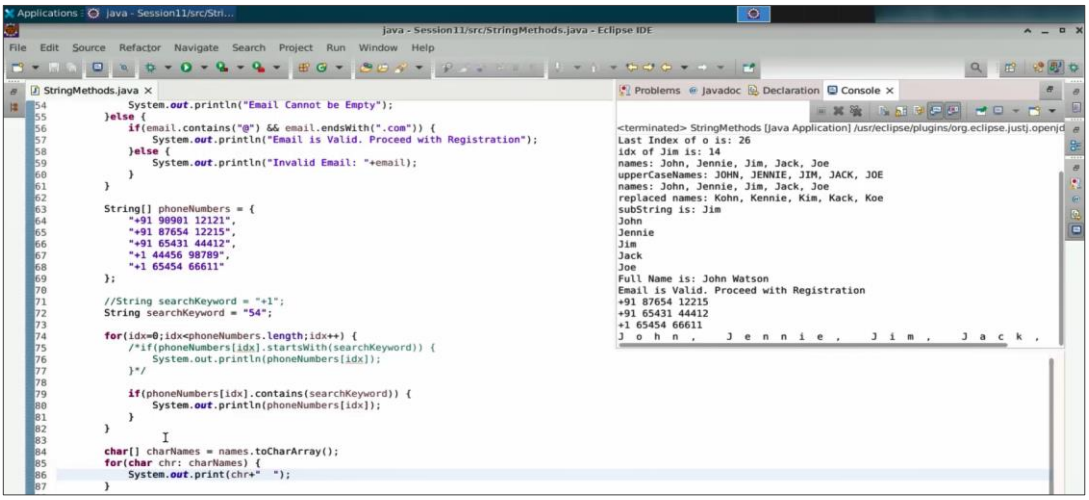


7.5 Let us see a magical thing where the search keyword can be "**12**". If you implement this with **startsWith**, it will not work. This is where you can use a very useful concept called contains. You will check if "**12**" is contained within the string, and you can see the list of phone numbers with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

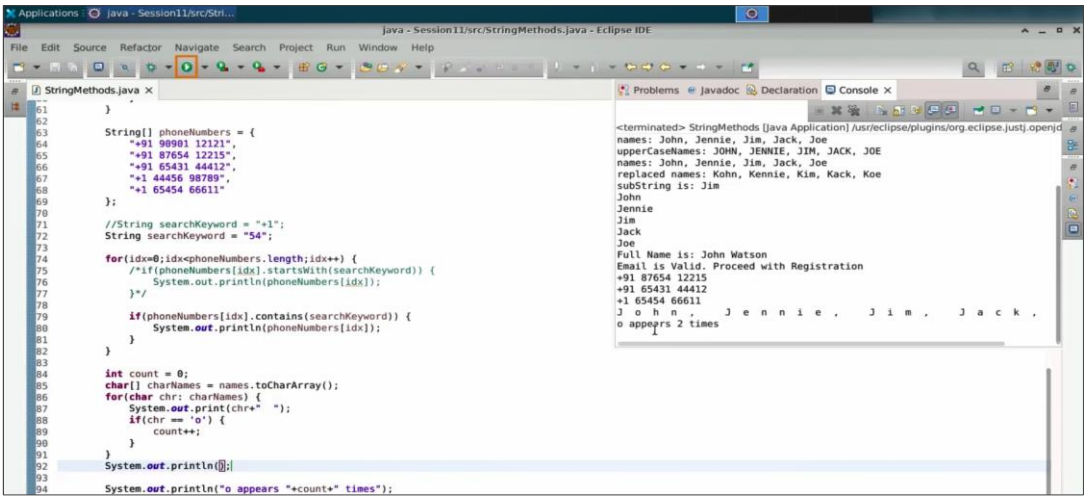


Step 8: Implement an algorithm to know the number of times a character appears

8.1 Let us also see one more method. Type **names.toaCharArray**, 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

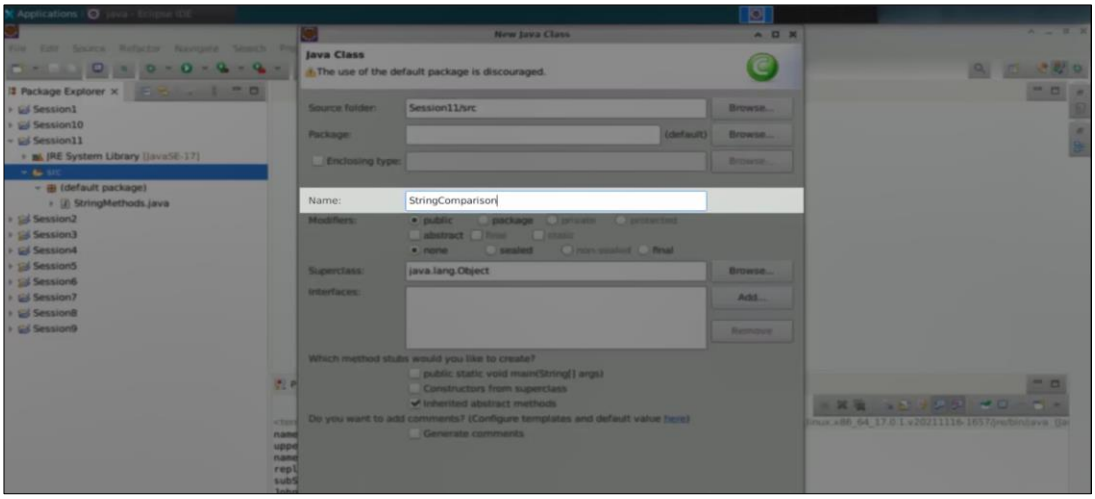


8.2 Now, maintain a count variable initialized to 0. If your character equals 'o', then increment the count. This is a simple algorithm that counts how many times a character appears. Add an empty print line here so that the output appears on the next line. Run the code, and you will see that 'o' appears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

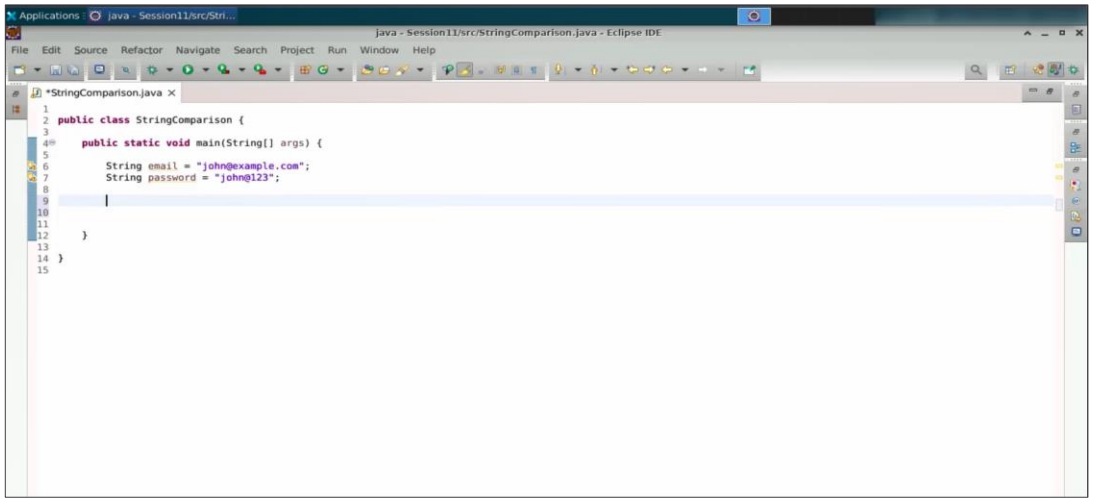


Step 9: Implement String comparison and intern strings

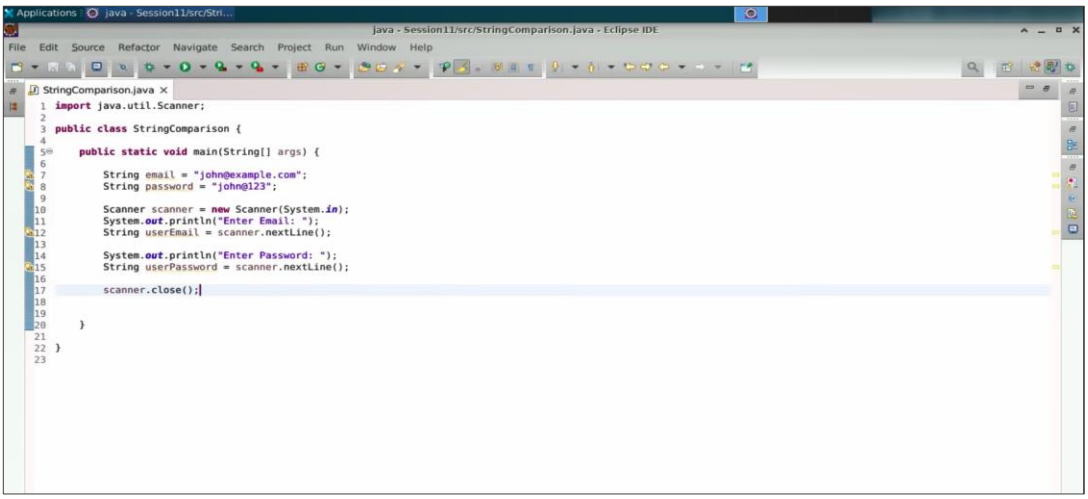
9.1 Now the next thing is string comparison. It is an important concept, and you will create altogether one separate class on it. Let us name it as **StringComparison**



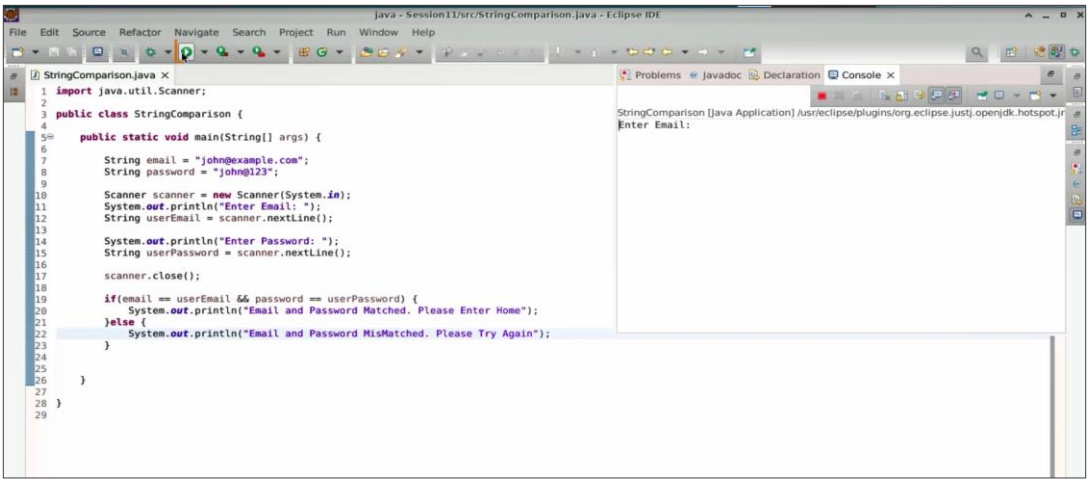
9.2 Let us say there is a string called email which is **john@example.com** and there is a string called password which is **John@123**. Now, these are the credentials, and you need to validate these credentials



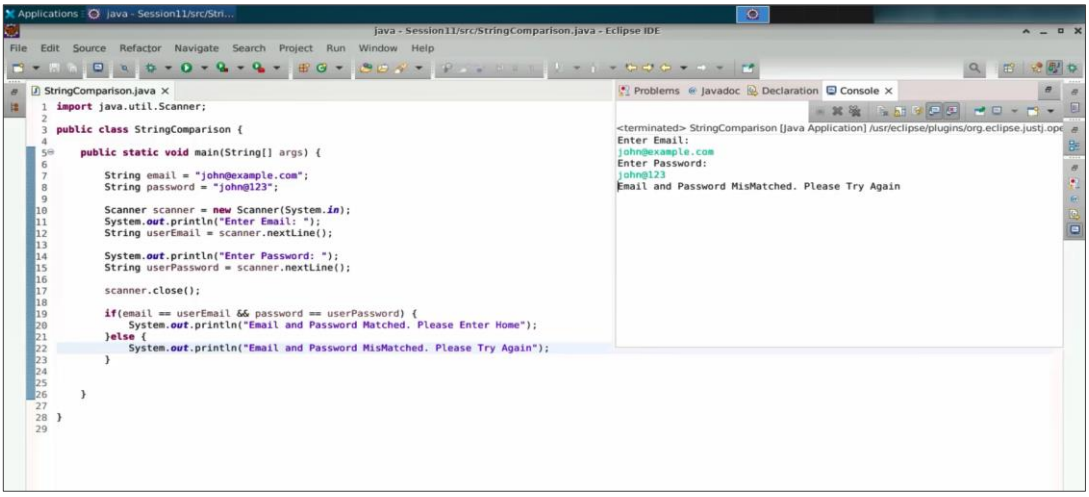
9.3 You are going to create a Scanner class. Type **Scanner scanner = new Scanner(System.in);**. Then prompt the user to enter the email and password. Store the user email with **userEmail = scanner.nextLine();** and the user password with **userPassword = scanner.nextLine();**. Finally, close the scanner



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.print.ln**. If the email and password match, Please Enter Home and if it does not, type Please Try Again

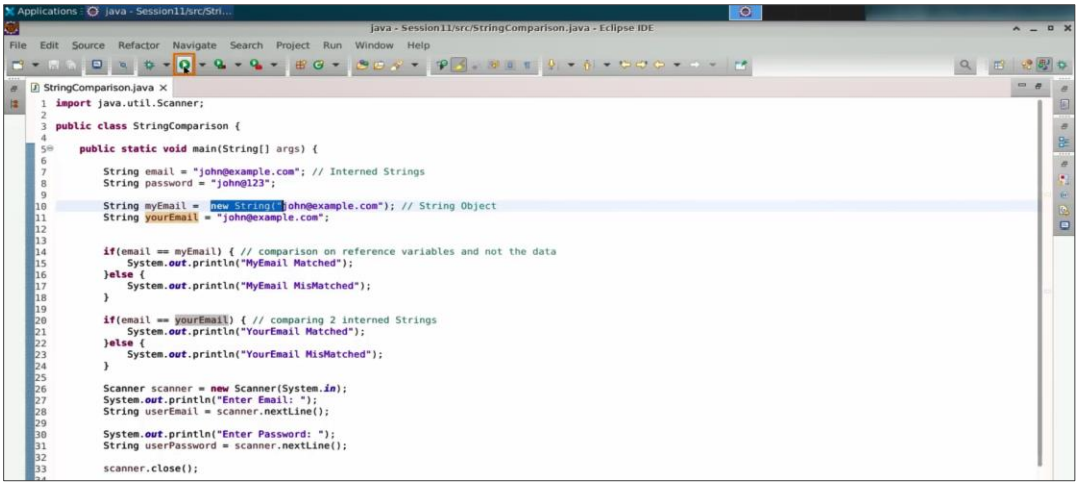


9.5 Run the program and enter the email and password. It indicates a password mismatch even though you have entered the correct credentials. This issue arises because there are two different ways to create strings in Java. The first method involves creating an interned string. The other method creates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



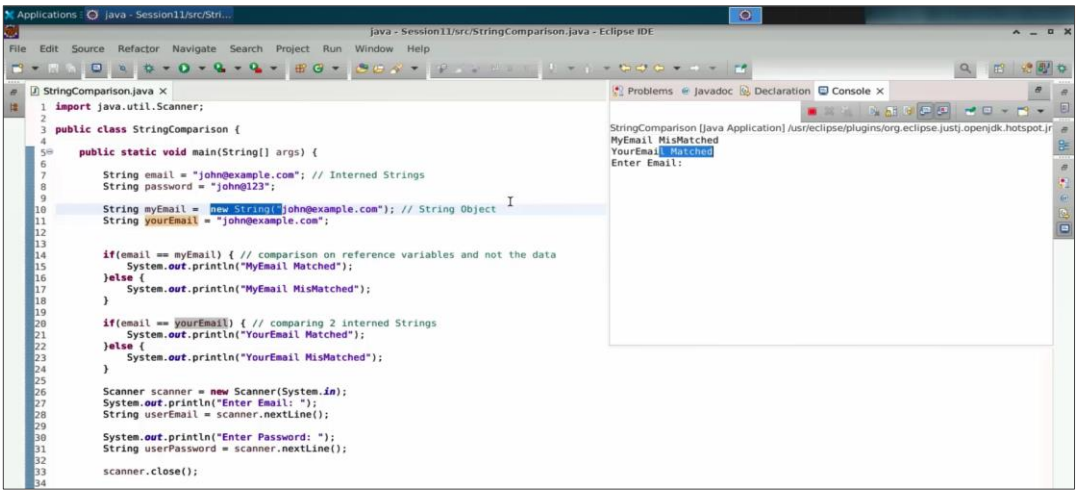

```java
import java.util.Scanner;

public class StringComparison {

    public static void main(String[] args) {

        String email = "john@example.com";
        String password = "john@123";

        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Email: ");
        String userEmail = scanner.nextLine();

        System.out.println("Enter Password: ");
        String userPassword = scanner.nextLine();

        scanner.close();

        if(email == userEmail && password == userPassword) {
            System.out.println("Email and Password Matched. Please Enter Home");
        }else {
            System.out.println("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



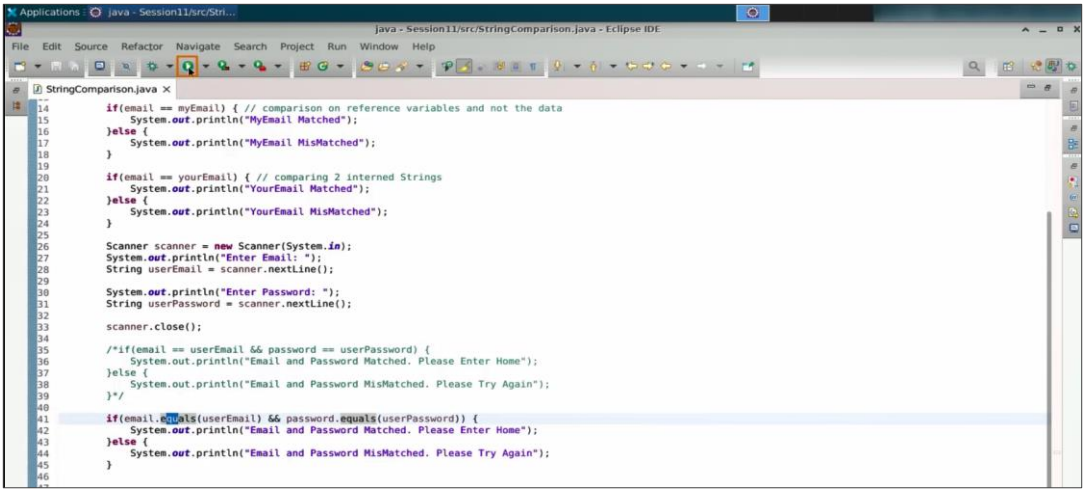

```java
import java.util.Scanner;

public class StringComparison {

    public static void main(String[] args) {

        String email = "john@example.com"; // Interned Strings
        String password = "john@123";

        String myEmail = new String("john@example.com"); // String Object
        String yourEmail = "john@example.com";


        if(email == myEmail) { // comparison on reference variables and not the data
            System.out.println("MyEmail Matched");
        }else {
            System.out.println("MyEmail MisMatched");
        }

        if(email == yourEmail) { // comparing 2 interned Strings
            System.out.println("YourEmail Matched");
        }else {
            System.out.println("YourEmail MisMatched");
        }

        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Email: ");
        String userEmail = scanner.nextLine();

        System.out.println("Enter Password: ");
        String userPassword = scanner.nextLine();

        scanner.close();
```

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

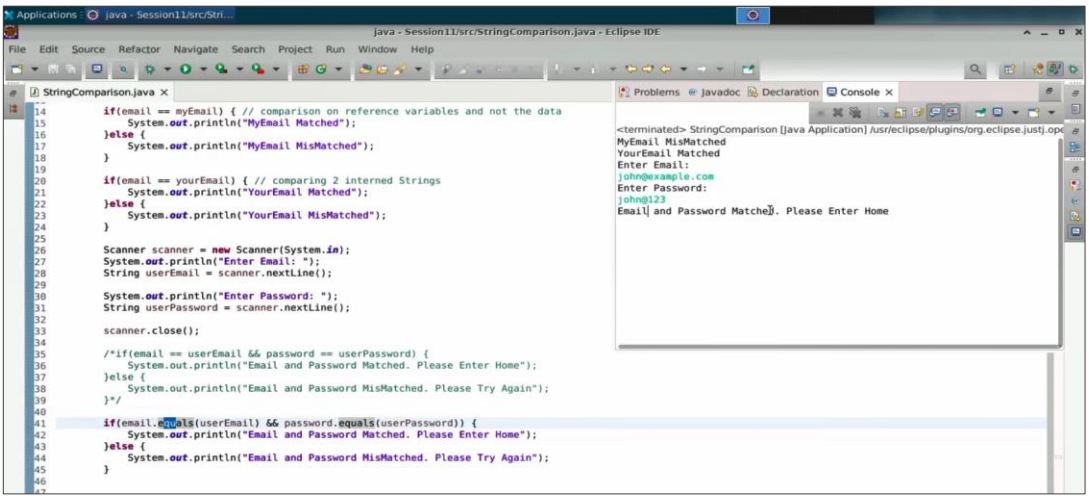


Step 10: Use the method.equals method

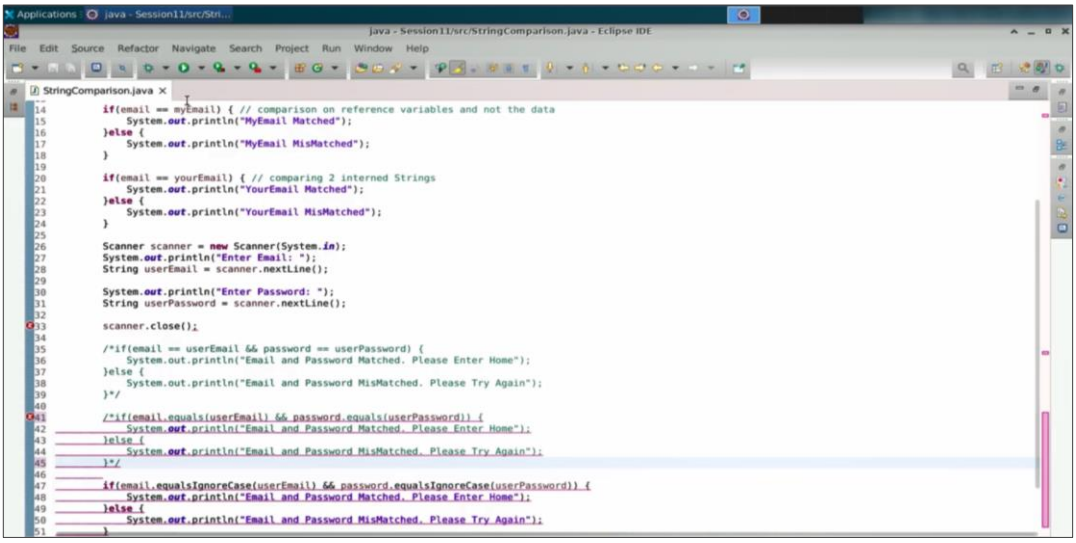
10.1 If you want to compare the content, the equals operator will not work. You can use a method from the String class called equals. This method equals, will compare the content. After replacing the equals operator with the equals method, the equals method will return true or false



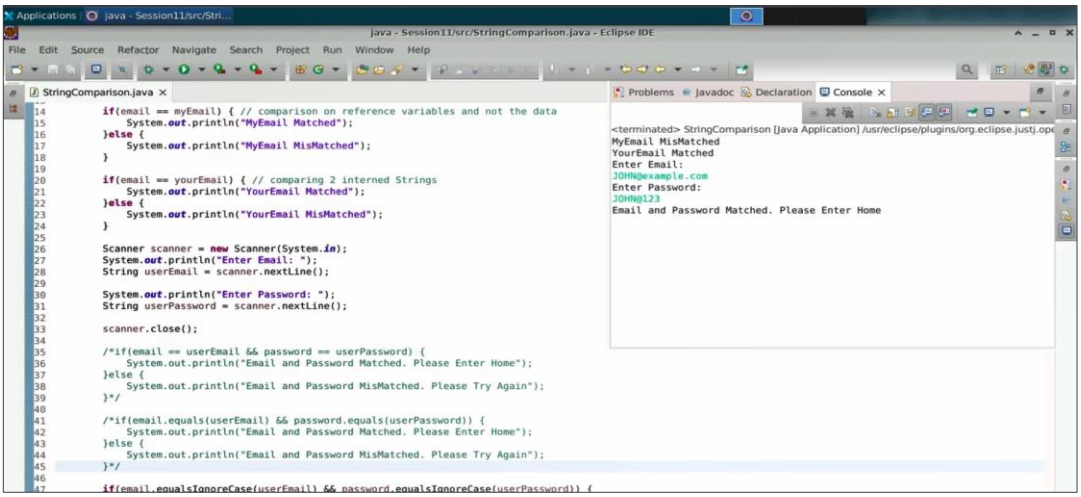
10.2 Run the code and type **john@example.com** as the email and **John@123** as the password. It shows that the email and password matched. Comparing strings is a risky operation. If you have string constants, such as interned strings where the references directly point to the literals, you can use 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



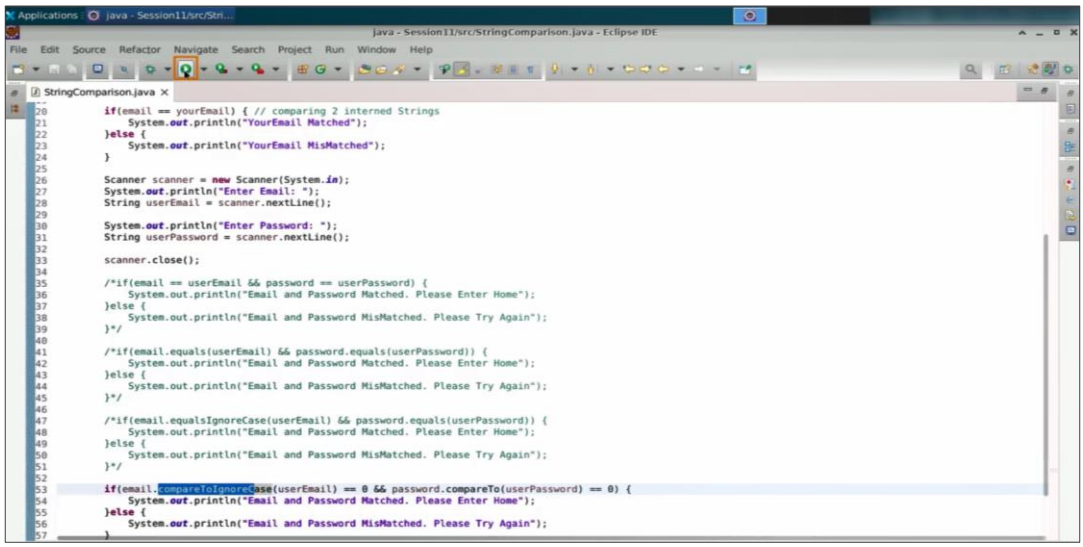
10.3 Add equals ignore case, this will even ignore the case sensitivity



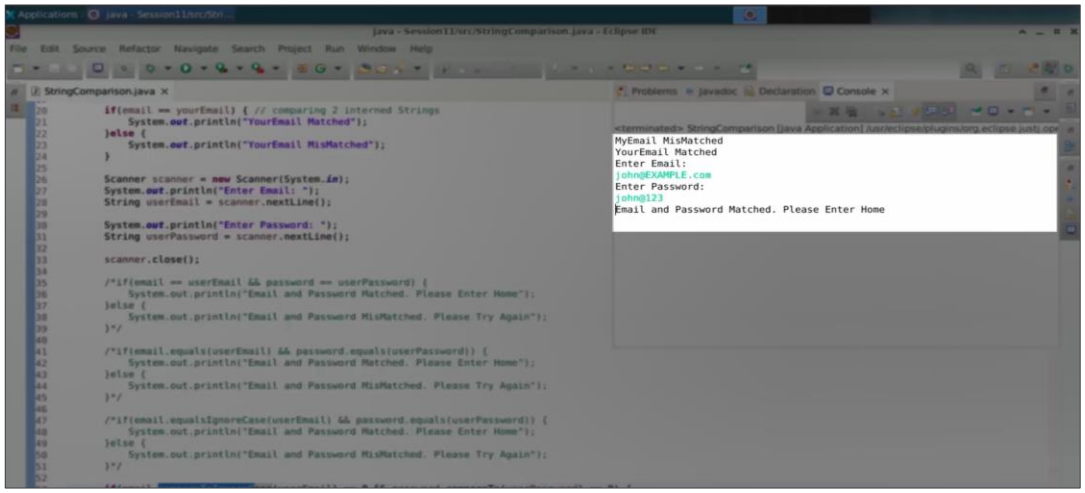
10.4 Run the code and enter an email which is like **JOHN@example.com**, and the password is **JHON@123**, you can see it says email and password match



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



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



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