Lesson 02 Demo 07

Comparing Mutability and Immutability of Strings

Objective: Differentiating and using mutable and immutable strings

Tools required: Eclipse IDE

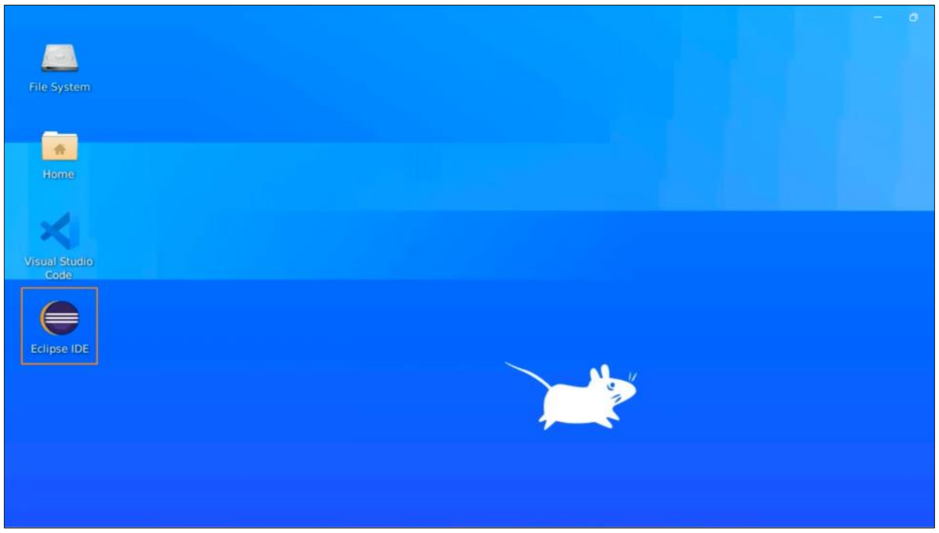
Prerequisites: None

Steps to be followed:

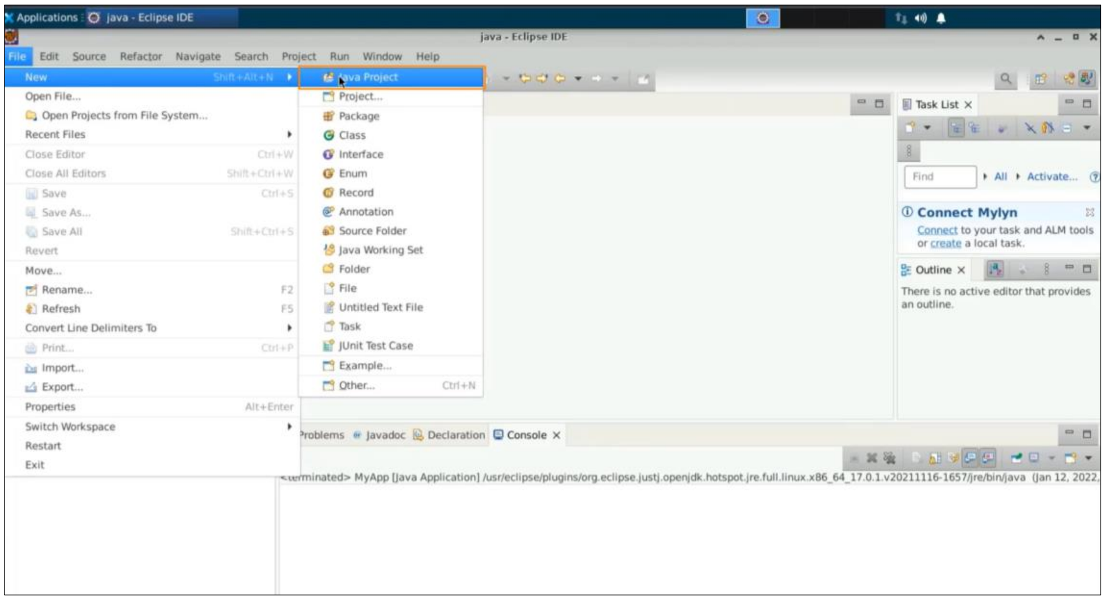
1. Create a class and write the main method
2. Create and concatenate the strings
3. Set Up string buffer and string builder
4. Write a method to accept strings
5. Define classes to implement the char sequence
6. Implement the runtime polymorphic behaviour for the interface
7. Pass a regular string, buffer, and builder
8. Use the common methods with strings

Step 1: Create a class and write the main method

1.1 Open the **Eclipse IDE**

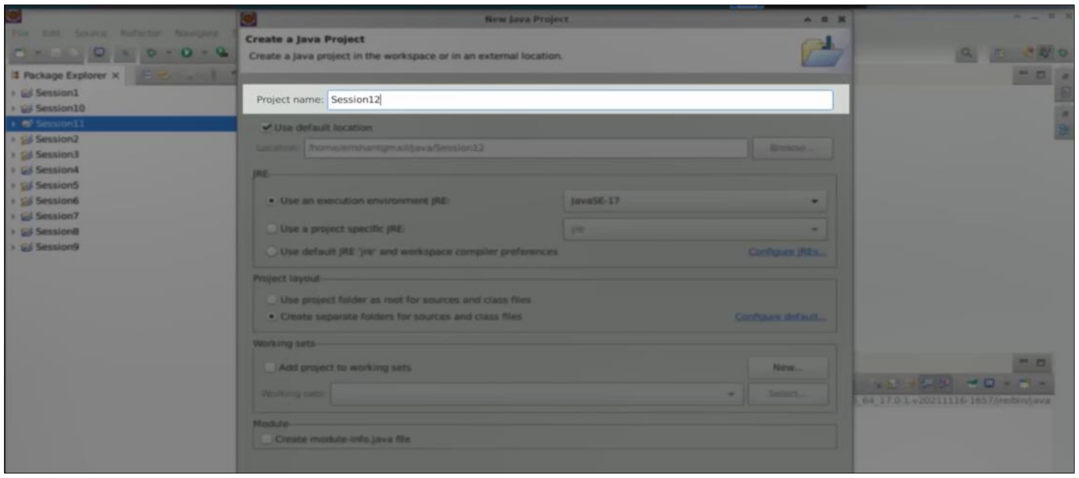


1.2. Select **File**, then **New,** and then **Java project**



```
<terminated> MyApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java (Jan 12, 2022,
```

1.3 Name the project **“Session12”,** uncheck **“Create a module info dot Java file”**, and press **Finish**



Create a Java Project
Create a Java project in the workspace or in an external location.

Project name: Session12

✓ Use default location
Location: /home/emshantgmail/java/Session12

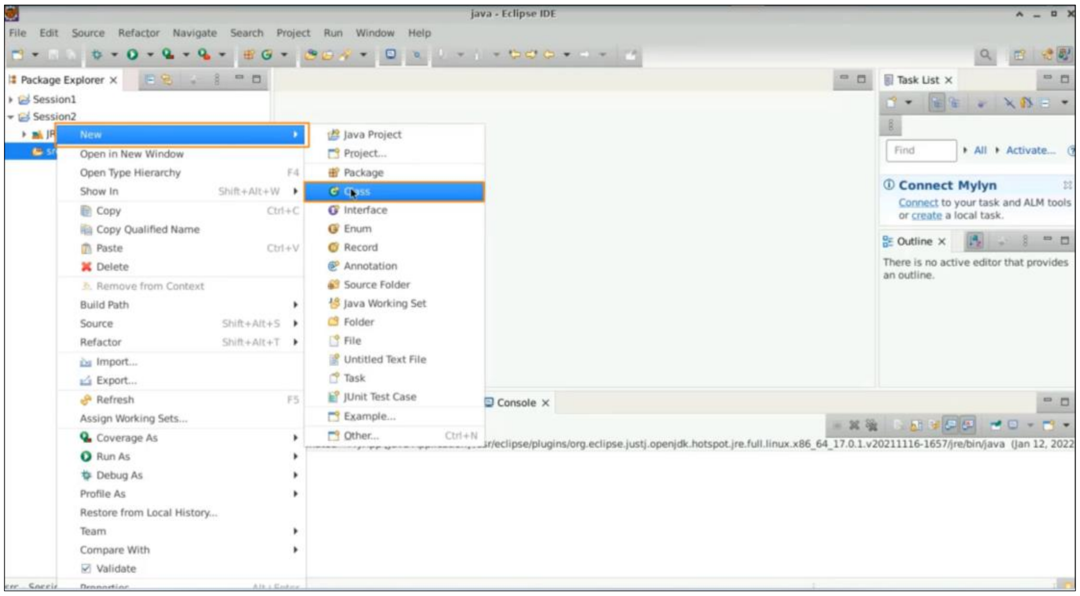
JRE
● Use an execution environment JRE: JavaSE-17
○ Use a project specific JRE: jre
○ Use default JRE 'jre' and workspace compiler preferences

Project layout
○ Use project folder as root for sources and class files
● Create separate folders for sources and class files

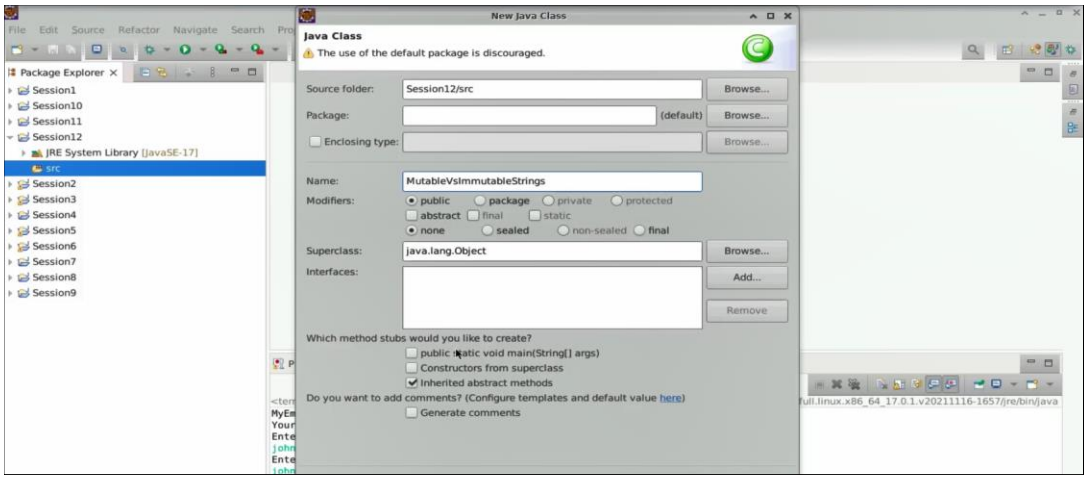
Working sets
☐ Add project to working sets
Working sets:

Module
☐ Create module-info.java file

1.4 With a **Session12** on the src, do a right-click and create a **new class**



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



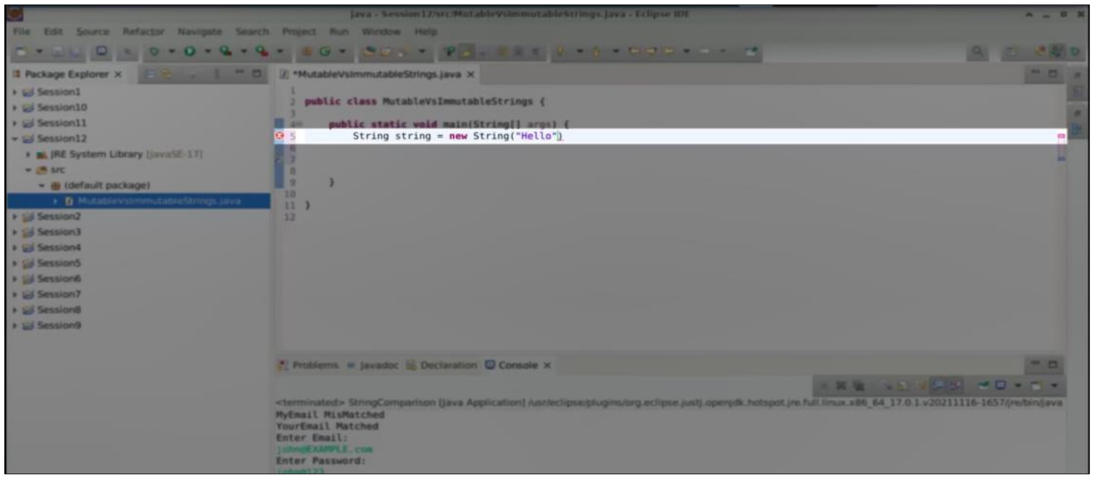
Java Class
The use of the default package is discouraged.

Source folder: Session12/src
Package: (default)
Enclosing type:
Name: MutableVsImmutableStrings
Modifiers: (● public) ○ package ○ private ○ protected
☐ abstract ☐ final ☐ static
(● none) ○ sealed ○ non-sealed ○ final
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

Step 2: Create and concatenate the strings

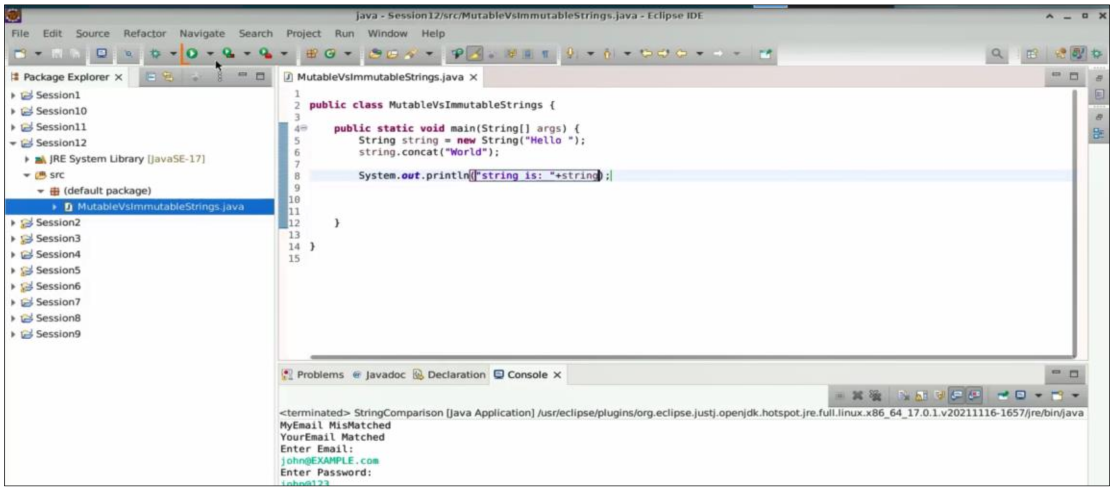
2.1 Create the first string over here as string, Object, and pass the value called hello



```java
public class MutableVsImmutableStrings {

    public static void main(String[] args) {
        String string = new String("Hello")

    }

}
```

```
<terminated> StringComparison [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java
MyEmail MisMatched
YourEmail Matched
Enter Email:
john@EXAMPLE.com
Enter Password:
```

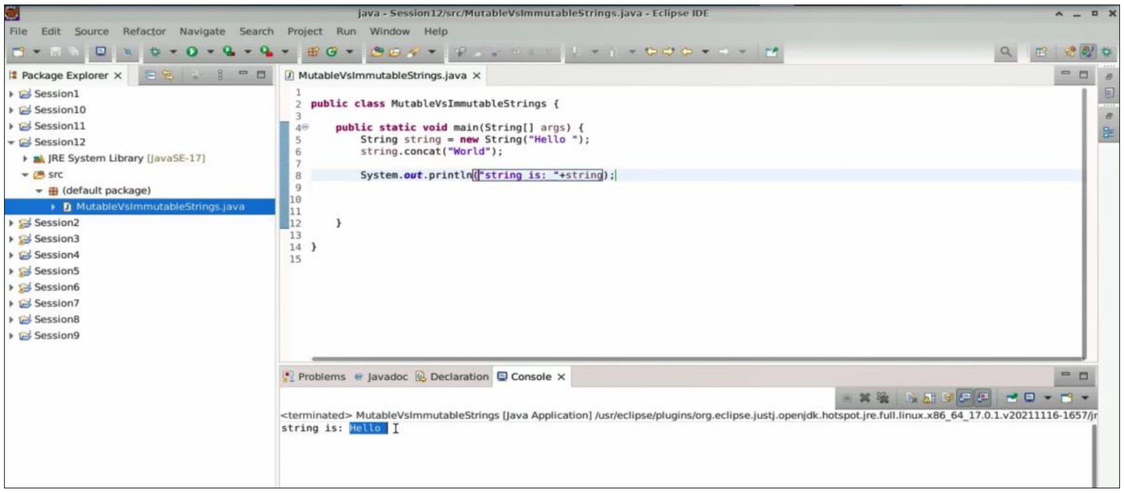
2.2 Now in the same string perform a concatenation and try to add something known as the world. If you print down this string. Let us say the string is and concatenate the string



```java
public class MutableVsImmutableStrings {

    public static void main(String[] args) {
        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);



    }

}
```

```
<terminated> StringComparison [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java
MyEmail MisMatched
YourEmail Matched
Enter Email:
john@EXAMPLE.com
Enter Password:
```

2.3 Run this code. You will observe that you will get to see only hello and not the concatenated word called world along with it. It means the strings are immutable in nature

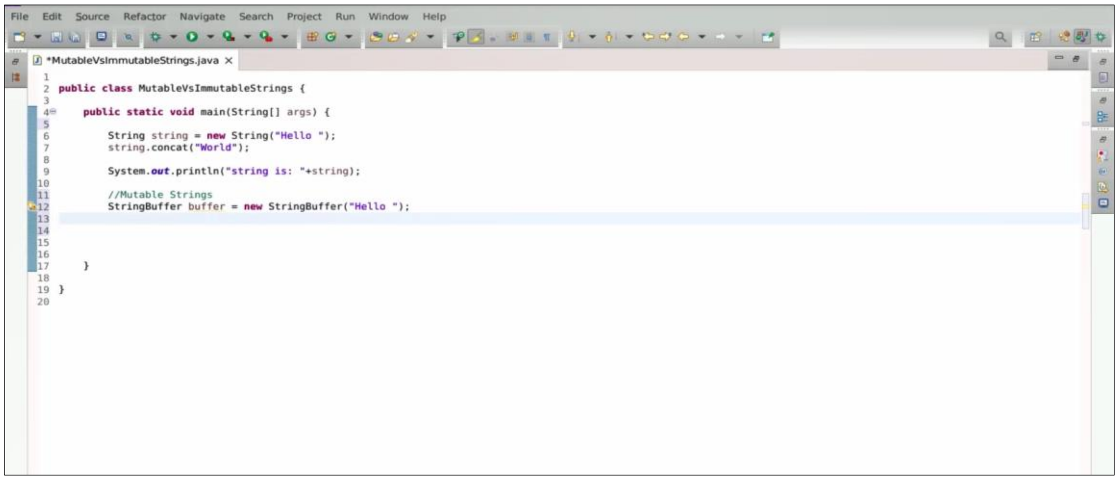


```java
public class MutableVsImmutableStrings {

    public static void main(String[] args) {
        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);



    }

}
```

```
<terminated> MutableVsImmutableStrings [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jr
string is: Hello 
```

Step 3: Set Up string buffer and string builder

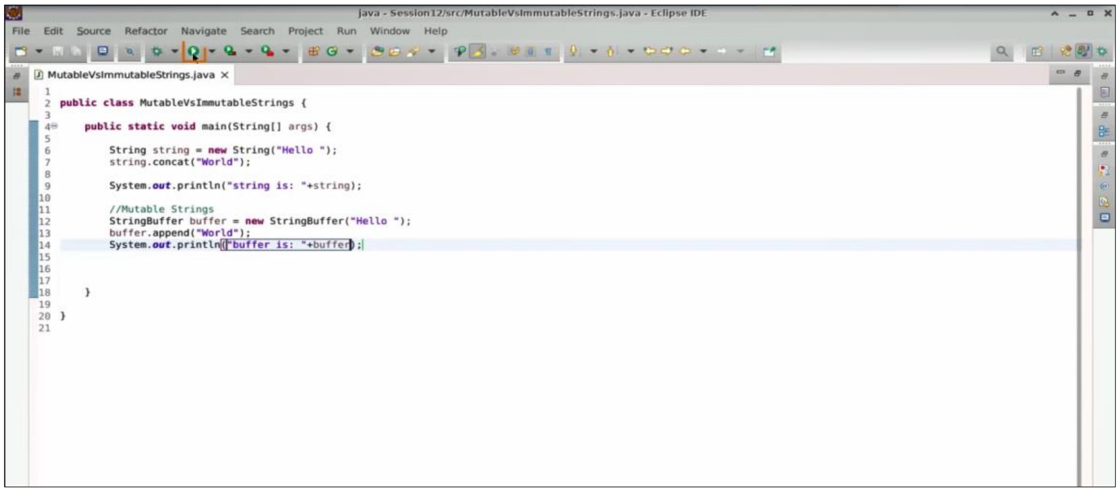
3.1 Mutable strings mean that you can manipulate the data inside the same string object. The first mutable string is known as the string buffer, and you cannot create these mutable strings as intern strings. Let us say buffer as a new string buffer and pass something known as hello



```java
public class MutableVsImmutableStrings {

    public static void main(String[] args) {

        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);

        //Mutable Strings
        StringBuffer buffer = new StringBuffer("Hello ");




    }

}
```

Step 4: Write a method to accept strings

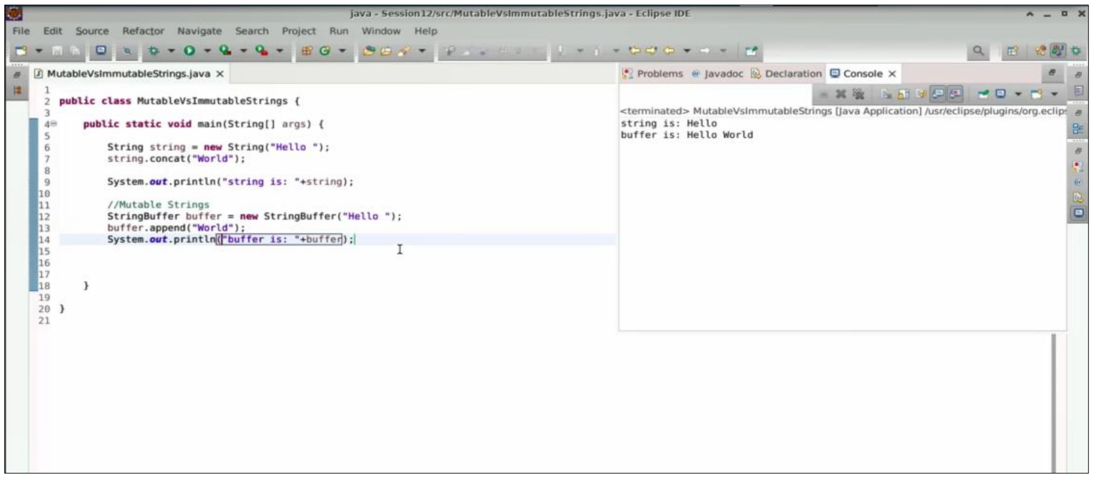
4.1 You have concat method inside this string in the buffer; you can say append method. And Let us say the world. Now print out the buffer, type buffer is the buffer



java - Session12/src/MutableVsImmutableStrings.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
MutableVsImmutableStrings.java

```
public class MutableVsImmutableStrings {

    public static void main(String[] args) {

        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);

        //Mutable Strings
        StringBuffer buffer = new StringBuffer("Hello ");
        buffer.append("World");
        System.out.println("buffer is: "+buffer);



    }

}
```

4.2 Run the code and see that in the same string object, you have got the data concatenated. The append method or the concat method do the same job but the append method inside, you can say string buffer would add a string in the same object string manipulation is required whenever you have a lot of data being processed from the server part

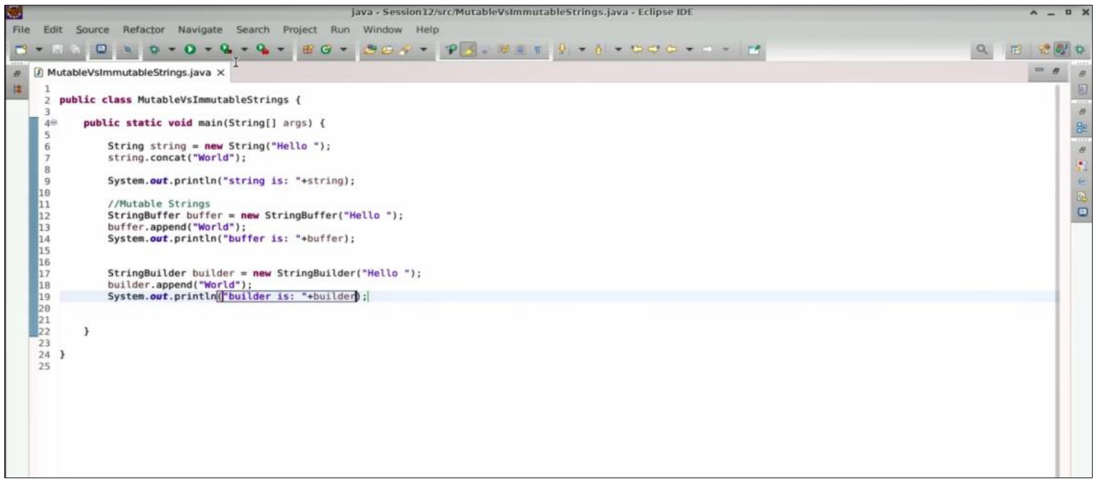


java - Session12/src/MutableVsImmutableStrings.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
MutableVsImmutableStrings.java
Problems
Javadoc
Declaration
Console

```
public class MutableVsImmutableStrings {

    public static void main(String[] args) {

        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);

        //Mutable Strings
        StringBuffer buffer = new StringBuffer("Hello ");
        buffer.append("World");
        System.out.println("buffer is: "+buffer);



    }

}
```

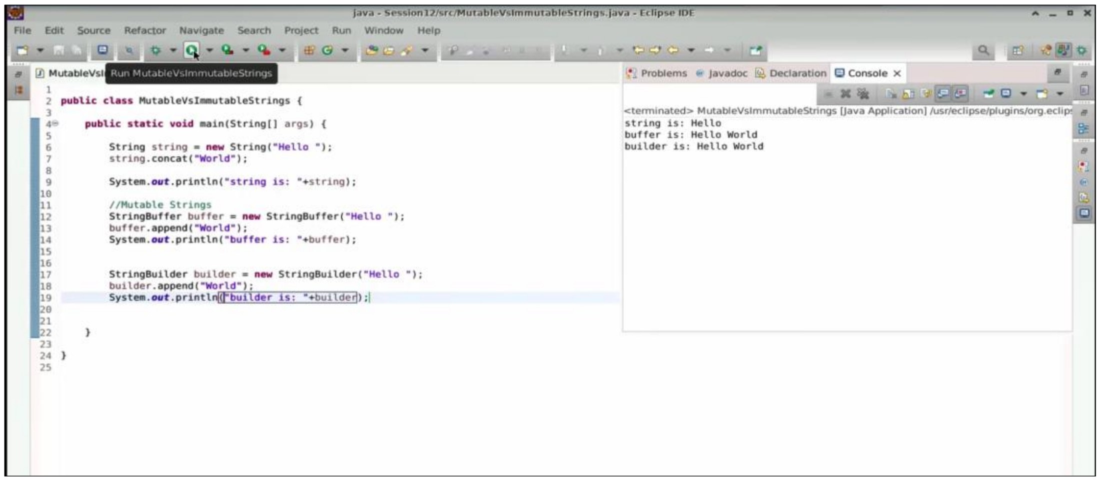
```
<terminated> MutableVsImmutableStrings [Java Application] /usr/eclipse/plugins/org.eclips
string is: Hello
buffer is: Hello World
```

4.3 You have one more string called a mutable string, which is a string builder. Let us type **StringBuilder** as a new StringBuilder. Write **'hello'** with some space, and then you can say **builder.append** with something known as **'world'**. Thereafter, type **builder is: “+builder”**. You will concatenate this string builder


java - Session12/src/MutableVsImmutableStrings.java - Eclipse IDE

```
public class MutableVsImmutableStrings {

    public static void main(String[] args) {

        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);

        //Mutable Strings
        StringBuffer buffer = new StringBuffer("Hello ");
        buffer.append("World");
        System.out.println("buffer is: "+buffer);


        StringBuilder builder = new StringBuilder("Hello ");
        builder.append("World");
        System.out.println("builder is: "+builder);


    }

}
```

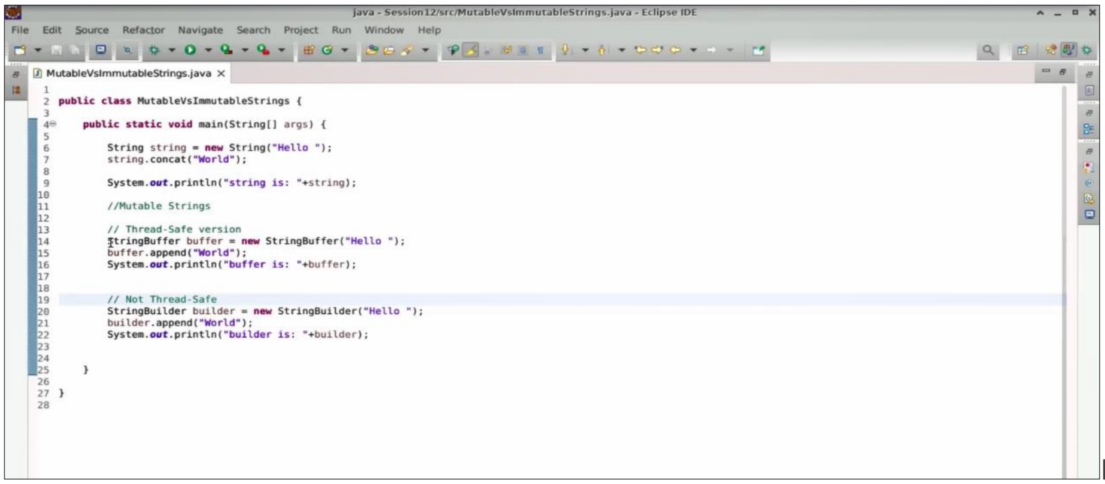
4.4 Run this code. what you see buffer and builder. They are both mutable versions of your strings, and if you want to do the string manipulation, you should either use string buffer or the string builder


java - Session12/src/MutableVsImmutableStrings.java - Eclipse IDE

```
public class MutableVsImmutableStrings {

    public static void main(String[] args) {

        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);

        //Mutable Strings
        StringBuffer buffer = new StringBuffer("Hello ");
        buffer.append("World");
        System.out.println("buffer is: "+buffer);


        StringBuilder builder = new StringBuilder("Hello ");
        builder.append("World");
        System.out.println("builder is: "+builder);


    }

}
```

```
<terminated> MutableVsImmutableStrings [Java Application] /usr/eclipse/plugins/org.eclips
string is: Hello
buffer is: Hello World
builder is: Hello World
```

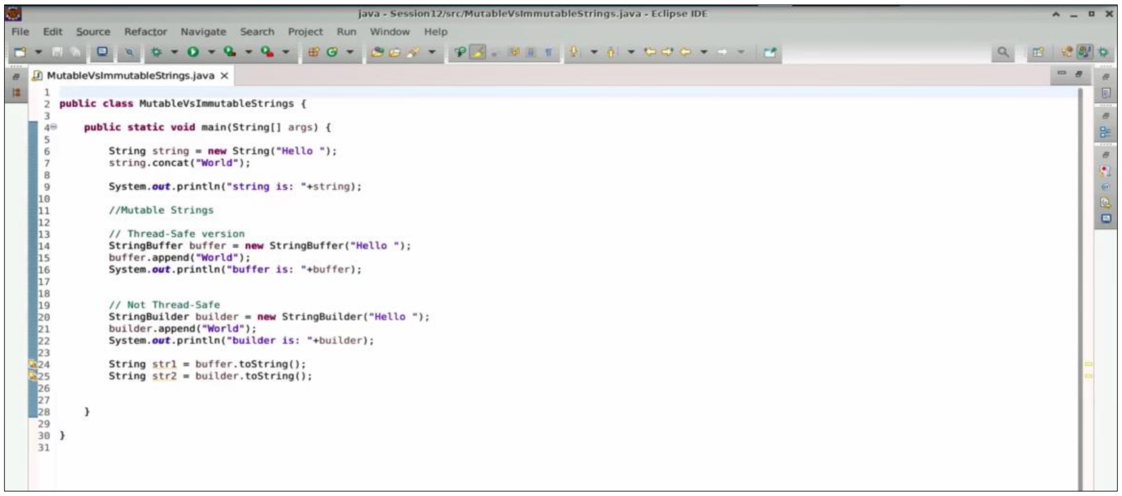
4.5 What is the difference between the string buffer and string builder? Now string buffer is a thread-safe version, so this is a thread-safe version, whereas you got string builder as not thread-safe



java - Session12/src/MutableVsImmutableStrings.java - Eclipse IDE

```java
public class MutableVsImmutableStrings {

    public static void main(String[] args) {

        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);

        //Mutable Strings

        // Thread-Safe version
        StringBuffer buffer = new StringBuffer("Hello ");
        buffer.append("World");
        System.out.println("buffer is: "+buffer);


        // Not Thread-Safe
        StringBuilder builder = new StringBuilder("Hello ");
        builder.append("World");
        System.out.println("builder is: "+builder);


    }

}
```

4.6 Type string **str1** is a **buffer.tostrings()**.When you want the string from your buffer, you can capture it, and the same way, you can say **str2** is **builder.tostring()**

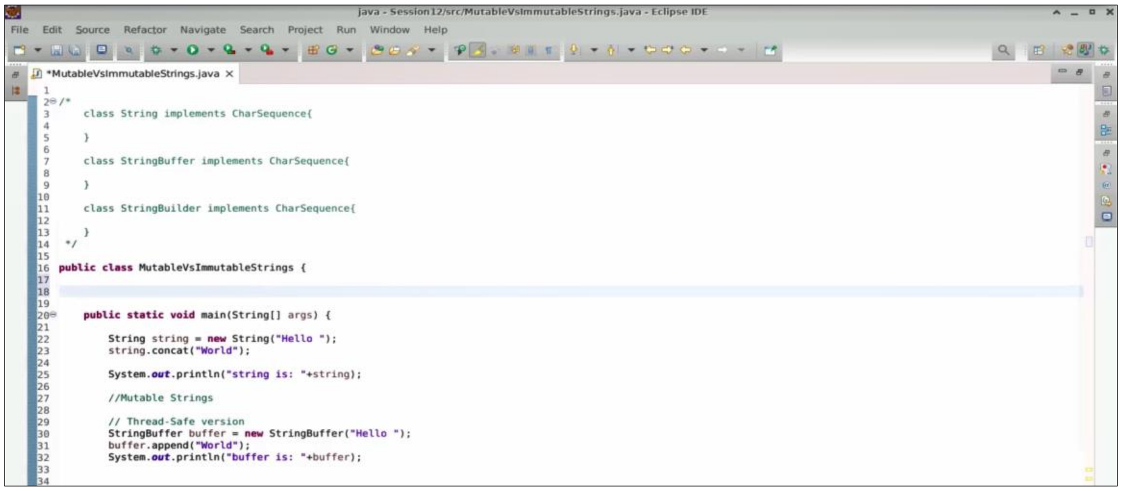


java - Session12/src/MutableVsImmutableStrings.java - Eclipse IDE

```java
public class MutableVsImmutableStrings {

    public static void main(String[] args) {

        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);

        //Mutable Strings

        // Thread-Safe version
        StringBuffer buffer = new StringBuffer("Hello ");
        buffer.append("World");
        System.out.println("buffer is: "+buffer);


        // Not Thread-Safe
        StringBuilder builder = new StringBuilder("Hello ");
        builder.append("World");
        System.out.println("builder is: "+builder);

        String str1 = buffer.toString();
        String str2 = builder.toString();


    }

}
```

Step 5: Define classes to implement the char sequence

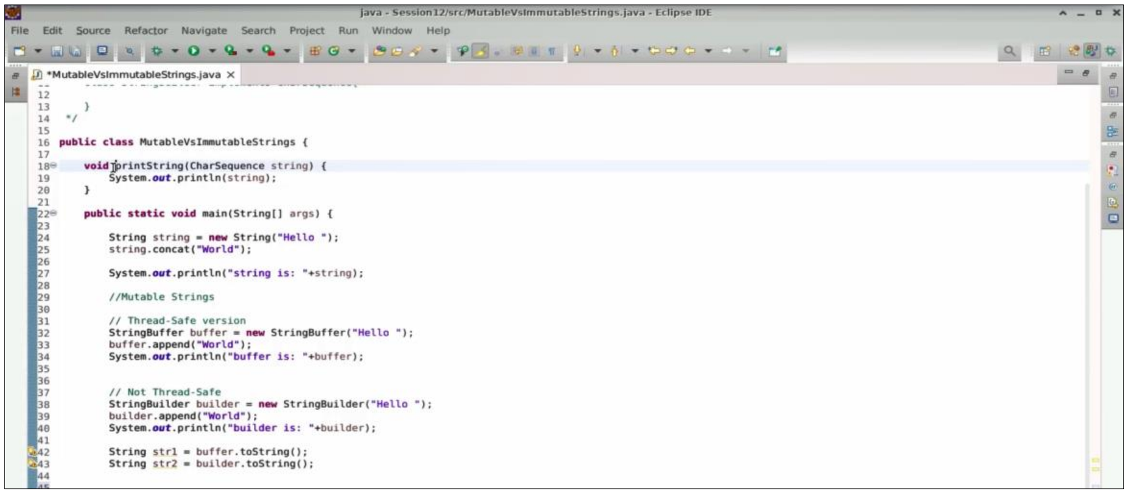
5.1 If you have three different variants of the strings, how can you come up with and define a method to accept these different strings? There is a class called string. This class will implement a char sequence; it is a built-in interface. Also, the buffer and builder implements the char sequence


java - Session12/src/MutableVsImmutableStrings.java - Eclipse IDE

```
/*
    class String implements CharSequence{

    }

    class StringBuffer implements CharSequence{

    }

    class StringBuilder implements CharSequence{

    }
 */

public class MutableVsImmutableStrings {



    public static void main(String[] args) {

        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);

        //Mutable Strings

        // Thread-Safe version
        StringBuffer buffer = new StringBuffer("Hello ");
        buffer.append("World");
        System.out.println("buffer is: "+buffer);
```

Step 6: Implement the runtime polymorphic behavior for the interface

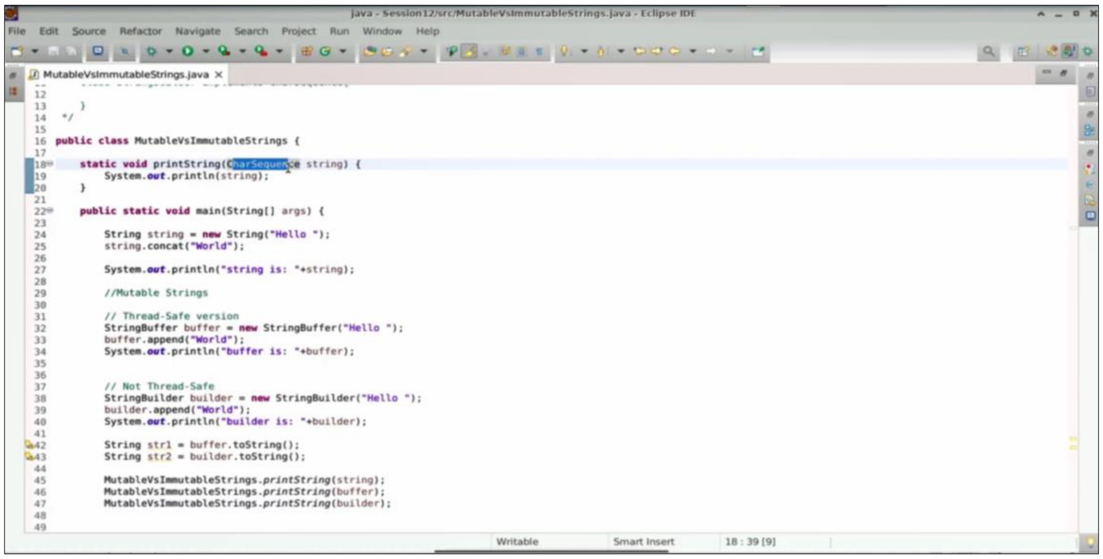
6.1 You can take the char sequence as input, let us see a string, and then simply print out the string


java - Session12/src/MutableVsImmutableStrings.java - Eclipse IDE

```

    }
 */

public class MutableVsImmutableStrings {

    void printString(CharSequence string) {
        System.out.println(string);
    }

    public static void main(String[] args) {

        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);

        //Mutable Strings

        // Thread-Safe version
        StringBuffer buffer = new StringBuffer("Hello ");
        buffer.append("World");
        System.out.println("buffer is: "+buffer);


        // Not Thread-Safe
        StringBuilder builder = new StringBuilder("Hello ");
        builder.append("World");
        System.out.println("builder is: "+builder);

        String str1 = buffer.toString();
        String str2 = builder.toString();
```

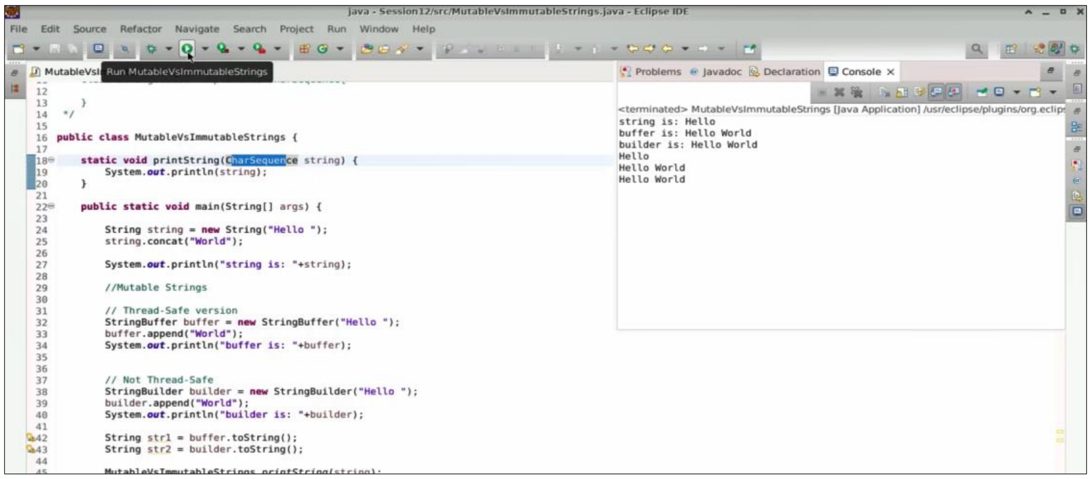
Step 7: Pass a regular string, buffer, and builder

7.1 Now, you are going to execute this method called printing; for the sake of simplicity, you are going to make this method static so that you can access it with the class name. Now you will add mutable versus immutable strings dot print the string. You can pass a regular string. Then pass the buffer as well as the builder


java - Session12/src/MutableVsImmutableStrings.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
MutableVsImmutableStrings.java
Writable
Smart Insert
18 : 39 [9]

```java
    }
 */

public class MutableVsImmutableStrings {

    static void printString(CharSequence string) {
        System.out.println(string);
    }

    public static void main(String[] args) {

        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);

        //Mutable Strings

        // Thread-Safe version
        StringBuffer buffer = new StringBuffer("Hello ");
        buffer.append("World");
        System.out.println("buffer is: "+buffer);


        // Not Thread-Safe
        StringBuilder builder = new StringBuilder("Hello ");
        builder.append("World");
        System.out.println("builder is: "+builder);

        String str1 = buffer.toString();
        String str2 = builder.toString();

        MutableVsImmutableStrings.printString(string);
        MutableVsImmutableStrings.printString(buffer);
        MutableVsImmutableStrings.printString(builder);
```

7.2 Run the code; it will print hello, **Hello World**, and Hello World for all three strings

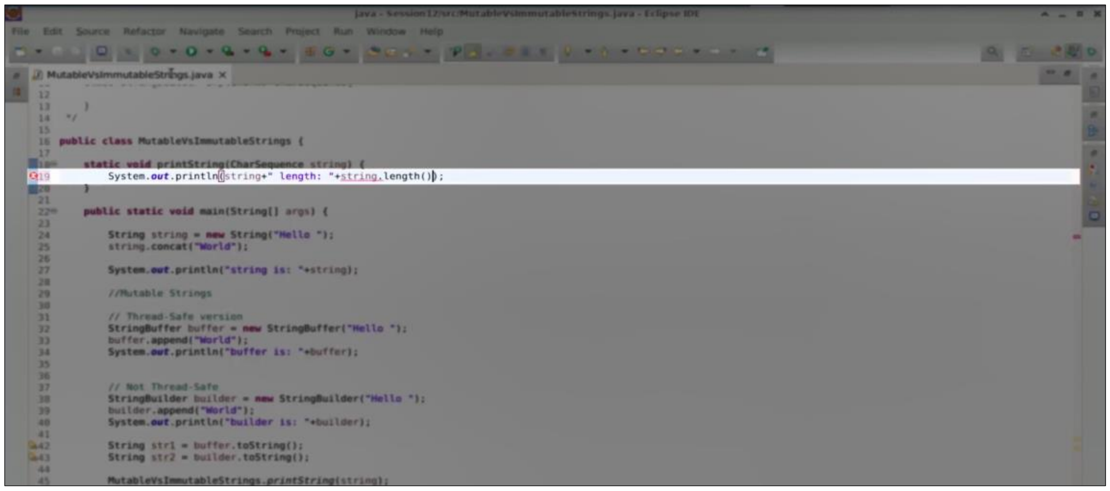

java - Session12/src/MutableVsImmutableStrings.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Run MutableVsImmutableStrings
Problems
Javadoc
Declaration
Console

```java
    }
 */

public class MutableVsImmutableStrings {

    static void printString(CharSequence string) {
        System.out.println(string);
    }

    public static void main(String[] args) {

        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);

        //Mutable Strings

        // Thread-Safe version
        StringBuffer buffer = new StringBuffer("Hello ");
        buffer.append("World");
        System.out.println("buffer is: "+buffer);


        // Not Thread-Safe
        StringBuilder builder = new StringBuilder("Hello ");
        builder.append("World");
        System.out.println("builder is: "+builder);

        String str1 = buffer.toString();
        String str2 = builder.toString();

        MutableVsImmutableStrings.printString(string);
```

```
<terminated> MutableVsImmutableStrings [Java Application] /usr/eclipse/plugins/org.eclip
string is: Hello
buffer is: Hello World
builder is: Hello World
Hello
Hello World
Hello World
```

Step 8: Use the common methods with strings

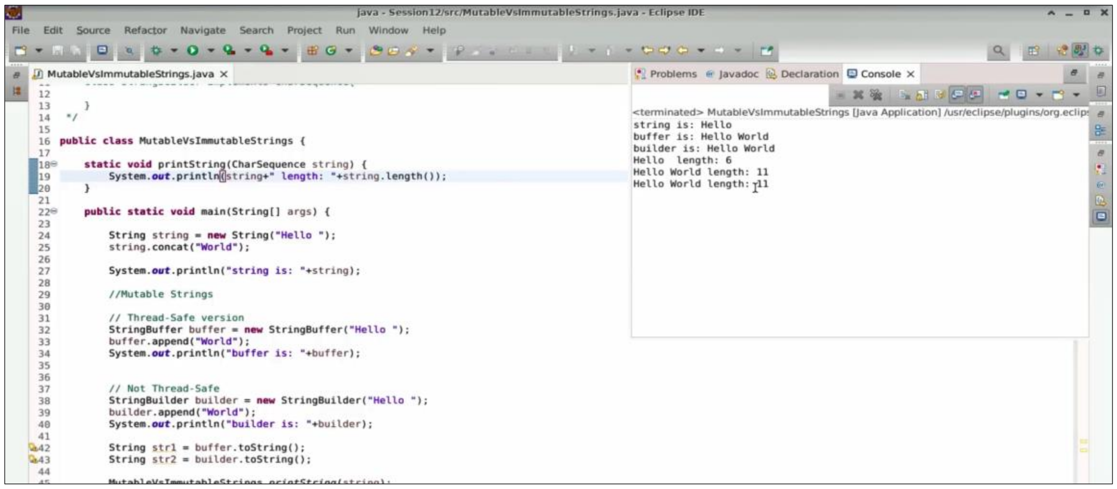
8.1 You can add String and a space followed by length. The common attributes or methods can be executed here, or you can simply print the length



```
java - Session12/src/MutableVsImmutableStrings.java - Eclipse IDE
```

```java
    }
 */

public class MutableVsImmutableStrings {

    static void printString(CharSequence string) {
        System.out.println(string+" length: "+string.length();
    }

    public static void main(String[] args) {

        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);

        //Mutable Strings

        // Thread-Safe version
        StringBuffer buffer = new StringBuffer("Hello ");
        buffer.append("World");
        System.out.println("buffer is: "+buffer);


        // Not Thread-Safe
        StringBuilder builder = new StringBuilder("Hello ");
        builder.append("World");
        System.out.println("builder is: "+builder);

        String str1 = buffer.toString();
        String str2 = builder.toString();

        MutableVsImmutableStrings.printString(string);
```

8.2 Run the code. All these common methods are available, and you can process them. If you want to save your memory and optimize your string manipulation operation, you need to choose the string buffer or the string builder



```
java - Session12/src/MutableVsImmutableStrings.java - Eclipse IDE
```

```java
    }
 */

public class MutableVsImmutableStrings {

    static void printString(CharSequence string) {
        System.out.println(string+" length: "+string.length());
    }

    public static void main(String[] args) {

        String string = new String("Hello ");
        string.concat("World");

        System.out.println("string is: "+string);

        //Mutable Strings

        // Thread-Safe version
        StringBuffer buffer = new StringBuffer("Hello ");
        buffer.append("World");
        System.out.println("buffer is: "+buffer);


        // Not Thread-Safe
        StringBuilder builder = new StringBuilder("Hello ");
        builder.append("World");
        System.out.println("builder is: "+builder);

        String str1 = buffer.toString();
        String str2 = builder.toString();
```

```
<terminated> MutableVsImmutableStrings [Java Application] /usr/eclipse/plugins/org.eclips
string is: Hello
buffer is: Hello World
builder is: Hello World
Hello  length: 6
Hello World length: 11
Hello World length: 11
```

By following the above steps, you have successfully compared the mutability and immutability of strings.