Reading:

Chapter 3.1 - 3.6	pages 114-137
Chapter 3.8 - 3.11	pages 141-154

Terminology:

new operator	return	DecimalFormat class
class	alias	wrapper class
object	garbage collection	autoboxing
reference variable	String class	unboxing
instantiation	immutable	*Optional terms*:
null	Java API	component
method	package	container
static method	import declaration	frame, JFrame class
constructor	Random class	panel, JPanel class
dot (**.**) operator	Math class	label, JLabel class
parameter	NumberFormat class	ImageIcon class

COMP 1210 Coding Standard Additions

- When using the import declaration, import classes from the same package individually (i.e. do <u>not</u> use the * notation).

- When making calculations with π, use Math.PI rather than the literal value such as 3.141. You should always use constants when they are available.

- From now on, when using a Scanner object to scan input (e.g., from the keyboard), use nextLine to get numerical input from the user and convert it to a number using methods from the corresponding wrapper class.
  For example to read in an `int` value from Scanner userInput:
  ```
  int myInt = Integer.ParseInt(userInput.nextLine());
  ```
  For example to read in a `double` value from Scanner scan:
  ```
  double myDouble = Double.ParseDouble(userInput.nextLine());
  ```

Due: Wednesday, September 7, 2016 by the end of the lab.

Goals:

By the end of this activity you should be able to do the following:
  ➢ Understand how to instantiate an object using the **new** operator
  ➢ Use the Java standard library (the Java API) to look up a class
  ➢ Use the String class and the Scanner class
  ➢ Construct a viewer canvas and run your program in the canvas

Program Description:

You will create a program that exchanges letters in a String to encode a message.

In-lab Directions:

As you build your program incrementally by entering each of the code segments below, you should compile and run your program after each segment is completed. This will allow you to verify that your program is correct after each step as you progress toward the completed program.

  MessageConverter.java
  • This program will read in a message from the user and then require the user to enter a number. The following will happen based on the number entered by user:
      o If the user enters a 1, the message will be printed as it was entered.
      o If the user enters a 2, the message will be printed in lower case.
      o If the user enters a 3, the message will be printed in upper case.
      o If the user enters a 4, the message will be printed with all vowels replaced with underscores.
      o If the user enters a 5, the message will be printed without the first and last character.
      o Any other number should generate an appropriate message.

  • Create a class called **MessageConverter** that includes a main method. Also add an import statement for the java.util.Scanner class. The import statement should be the first line in your Java file (i.e., it comes before the class declaration as well as comments).

  • In the main method, declare the following variables:
      - userInput: Scanner object for reading the user input from the keyboard.
        ```
        Scanner userInput = new Scanner(System.in);
        ```
      - **message**: String object for the user input.
        ```
        String message = "";
        ```
      - **outputType**: An int representing the type of output the user wants.
      - **result**: String object for the converted message; initialized to an empty String ("").
        ```
        String result = "";
        ```

  • Request input from the user then get user input using the nextLine() method of the Scanner class.
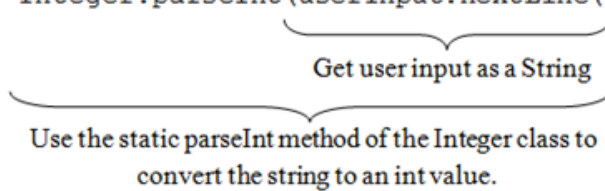   ```
    System.out.print("Type in a message and press enter:\n\t> ");
    message = userInput.nextLine();
    ```

- Now get the user's input for the output type. Make sure that you understand how the code below works.

```
System.out.print("\nOutput types:"
    + "\n\t1: As is"
    + "\n\t2: lower case"
    + "\n\t3: UPPER CASE"
    + "\n\t4: v_w_ls r_pl_c_d"
    + "\n\t5: Without first and last character"
    + "\nEnter your choice: ");
```

```
outputType = Integer.parseInt(userInput.nextLine());
```

Get user input as a String

Use the static parseInt method of the Integer class to convert the string to an int value.

- Set up an *if* statement that will print out the appropriate message based on the user's selected output type. Note that you can write code for alternate conditions in the *if* statement using *else if* after the *if* block.

```
if (outputType == 1) { // as is

}
else if (outputType == 2) { // lower case

}
else if (outputType == 3) { // upper case

}
else if (outputType == 4) { // replace vowels

}
else if (outputType == 5) { // without first and last character

}
else { // invalid input}

}
```

- <u>After</u> the *if-else* statement, add the following line of code to print out the result.

```
System.out.println("\n" + result);
```

Fill in the *if* statement in the following steps.

- The first condition just sets the result equal to the original message.

```
if (outputType == 1) { // as is
    result = message;
}
```

Activity 3: Classes and Objects

- Because message references an instance of the String class, we can use the methods in the String class to return a new String that is a modified version of the original String referenced by message (the original String object is not changed). To see the available methods in the String class, go to the Java API at http://docs.oracle.com/javase/8/docs/api/ and find the **String** class in the lower left pane on the page and left-click on it. This will open the String page in large right window of the API as shown below. You should become familiar with the API since it describes all of the classes that come with the JDK.

String Class
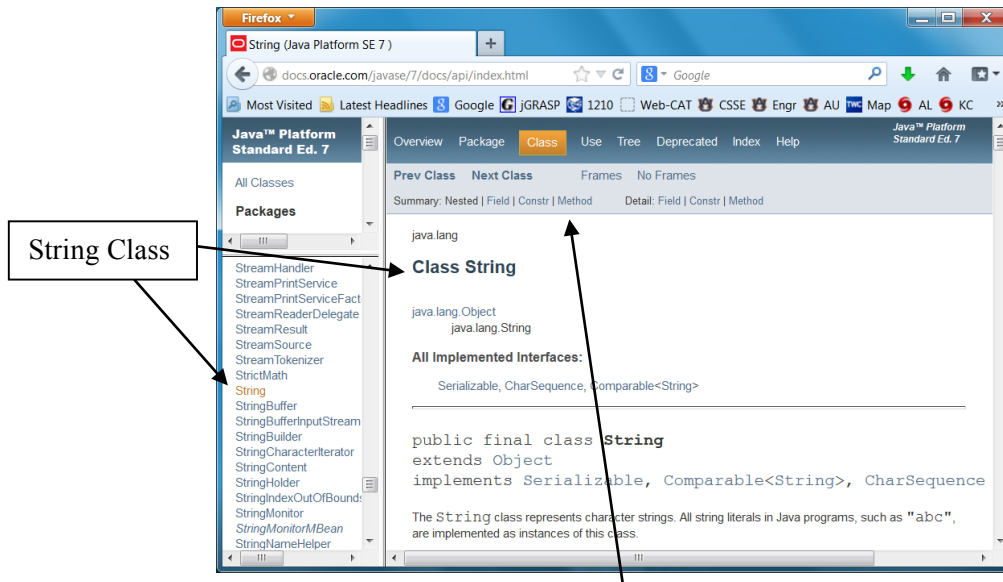
- In the main window on the right side, click method at the top or scroll down to Method Summary to see the methods that our String object **message** can use.

String Methods
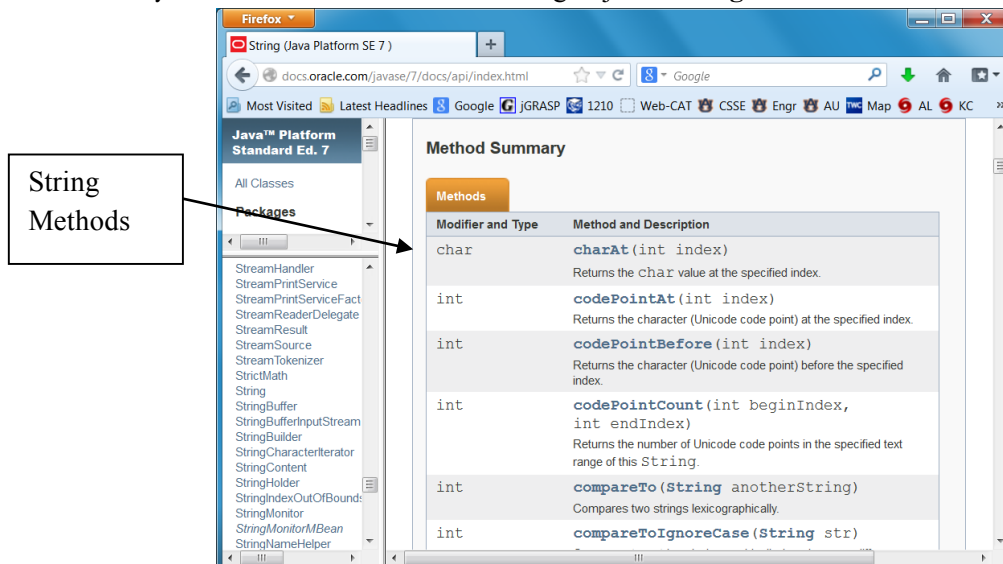
- Find a method called toLowerCase, and take a look at the method summary.
  - What *parameters* do we need to give the method?
  - What does the method do?
  - What does the method *return*?

The API entry for each method shows the return type for the method in the left and the method name with formal parameters, if any, on the right along with a brief description of what the method does. So the entry shown below for the *toLowerCase* method indicates that it returns a String and takes no parameters.

Returns a String   Method name – there are no formal parameters for this method

```
String              toLowerCase ()
                    Converts all of the characters in this String to lower case using
                    the rules of the default locale.
```

The **_toLowerCase_** method does not modify the String object upon which it is called. Instead it returns a new String object which is the lower case version of original String.

- Based on this knowledge, you can now invoke the toLowerCase method on our **message** object to return its value in lower case. Place the following code in second block of the if statement:

```
else if(outputType == 2) { // lower case
    result = message.toLowerCase();
}
```

- Look at the API description for the toUpperCase method and place the following line in the second block of the if statement:

```
else if(outputType == 3) { // upper case
    result = message.toUpperCase();
}
```

- Now write code that will replace specified characters in the message with other characters. In order to change a character, we need to use the *replace* method of String to change certain characters of our string. Take a look at the *replace* method in the Java API. We see that it returns a String and that it requires two parameters: the character you want to replace and the new character. Note there are two versions of replace: the first takes parameters of type *char* (e.g., 'a' or 'e').and the second takes parameters of type *CharSequence* which includes String literals (e.g., "a" or "abc"). In the code below, we will use the *char* version.

Returns a String   Method name and two formal parameters

```
String              replace(char oldChar, char newChar)
                    Returns a string resulting from replacing all occurrences of
                    oldChar in this string with newChar.

String              replace(CharSequence target, CharSequence replacement)
                    Replaces each substring of this string that matches the literal
                    target sequence with the specified literal replacement
                    sequence.
```

The **_replace_** method does not modify the String object it. Instead it returns a new String object in which the new character has replaced the old character.

- First, use the *replace* method to change all of the a's to underscores. The following line of code replaces the 'a' characters of the message value and then assigns the new version of message to result. Again, note that message itself remains unchanged.

```java
else if(outputType == 4) { // replace vowels
    result = message.replace('a', '_');
}
```

  Now complete the translation by converting the remaining vowels to underscores for outputType equal to 4 .

```java
else if(outputType == 4) { // replace vowels
    result = message.replace('a', '_');
    result = result.replace('e', '_');
    result = result.replace('i', '_');
    result = result.replace('o', '_');
    result = result.replace('u', '_');
}
```

- Now use the *substring* method and length method to extract a part (or substring) of the message.

Returns a String   Method name and formal parameters

String	substring(int beginIndex) Returns a string that is a substring of this string.
String	substring(int beginIndex, int endIndex) Returns a string that is a substring of this string.

  Recall the characters in a String are indexed beginning at 0. Since the requirement here is for result to include everything except the first and last character, use the second version of *substring* with 1 as the parameter beginIndex and message.length() - 1 as endIndex.

```java
else if (outputType == 5) { // without first and last character
    result = message.substring(1, message.length() - 1);
}
```

- Finally, if the user enters a number other than 1, 2, 3, or 4, result should be set to an error message.

```java
else { // invalid input
    result = "Error: Invalid choice input.";
}
```

- Test your program under each of the conditions:
  Original ("as is"):

```
Type in a message and press enter:
   > This is a test.

Output types:
   1: As is
   2: lower case
   3: UPPER CASE
   4: v_w_ls r_pl_c_d
   5: Without first and last character
Enter your choice: 1


This is a test.
```

Lower case:

```
Type in a message and press enter:
   > This is a test.

Output types:
   1: As is
   2: lower case
   3: UPPER CASE
   4: v_w_ls r_pl_c_d
   5: Without first and last character
Enter your choice: 2


this is a test.
```

Upper case:

```
Type in a message and press enter:
   > This is a test.

Output types:
   1: As is
   2: lower case
   3: UPPER CASE
   4: v_w_ls r_pl_c_d
   5: Without first and last character
Enter your choice: 3


THIS IS A TEST.
```

Vowels replaced:

```
Type in a message and press enter:
   > This is a test.

Output types:
   1: As is
   2: lower case
   3: UPPER CASE
   4: v_w_ls r_pl_c_d
   5: Without first and last character
Enter your choice: 4


Th_s _s _ t_st.
```

Without first and last character:

```
Type in a message and press enter:
   > This is a test.

Output types:
   1: As is
   2: lower case
   3: UPPER CASE
   4: v_w_ls r_pl_c_d
   5: Without first and last character
Enter your choice: 5


his is a test
```

Invalid input:

```
Type in a message and press enter:
   > This is a test.

Output types:
   1: As is
   2: lower case
   3: UPPER CASE
   4: v_w_ls r_pl_c_d
   5: Without first and last character
Enter your choice: 7


Error: Invalid choice input.
```

- **Try the input "A message" for the message and choose to replace all vowels with underscores. What do you notice about the output? Why do you think that one of the vowels was not replaced? How would you go about correcting the issue?**

Using the Viewer Canvas for your MessageConverter program

Now we want to create a viewer canvas so we can see what is happening in the MessageConverter program step-by-step as it runs. Note that we could have done this step as soon as the variables had been declared above. For your project assignments, you should consider creating a canvas at any point that you need insight into exactly what is happening in your program.

- With MessageConverter.java in the CSD edit window, click the Run in Canvas button 🖼 on the toolbar. In the CSD window, you should see that the program is stopped at the first excutable statement, and you should see an empty viewer canvas window as shown below.
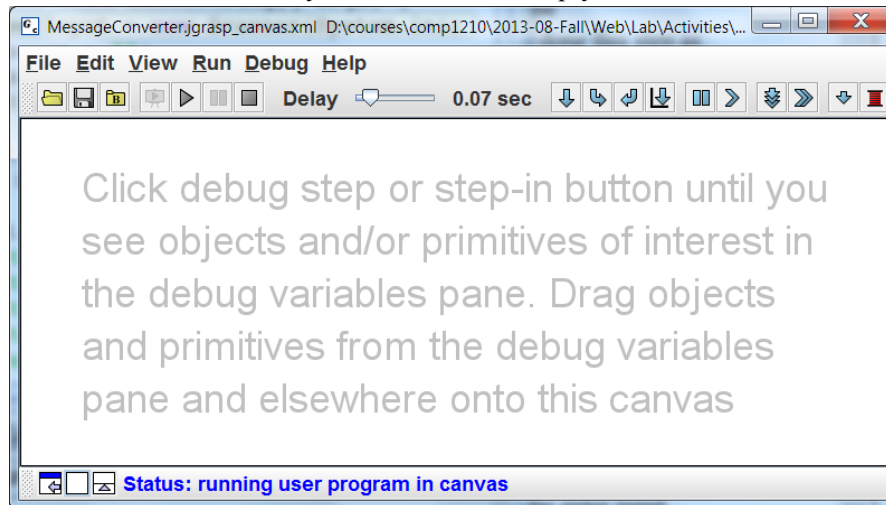
- As indicated by the message in the canvas, click the debug step button ⬇ to execute the statement to create a Scanner object on System.in which is the keyboard. You should now see the variable name `userInput` in the Debug tab. Now click the step button ⬇ three more times so that you see `message`, `result`, and `outputType` in the Debug tab. Now drag each of these variables into the canvas window and arrange them as shown in the figure below. Click the Save button 🖫 on the canvas toolbar.
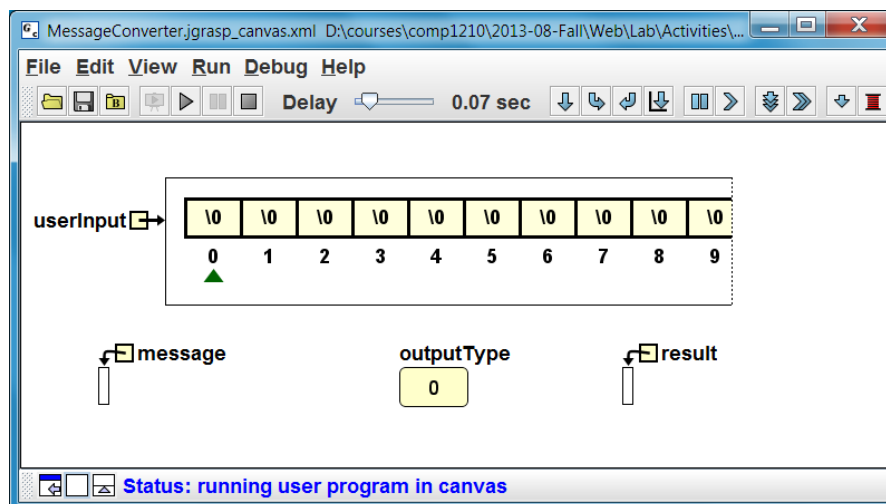
- Now click the play button ▶, and your program will begin auto-stepping until is pauses waiting for input. After you enter the requested input each time, the viewers on canvas will be updated. The *run in canvas* will continue until your program terminates at which time the viewers on the canvas will display their last values as shown in the example below for *message* "This is a test" and *outputType* = 4.
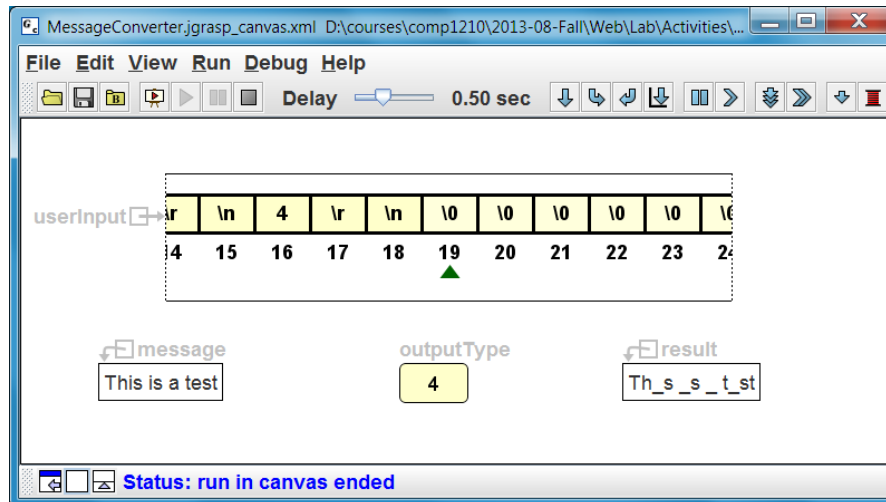
Although we created the canvas for the MessageConverter after the program was completed, it would have been even more appropriate to create the canvas as the program was being coded, especially if you were encountering logical errors in your program. For example, if you sense that an error has occurred at particular place in the program, you should set a breakpoint on the statement where you want to examine the behavior. [To set a breakpoint, move the mouse over the margin to the left of the statement until you see the red octagon (stop sign symbol) and then left-click the mouse.] Then instead of clicking the Run in Canvas button 🖼 on the toolbar, click the Debug button 🐞. The program will run to the breakpoint and stop. You can single-step ⬇ (or perhaps step-in ↳) and observe the variables in the canvas and debug tab to help determine the nature of the error so that you can correct it.