**Test link:  
<https://codecheck.it/files/20100416242eudkddhq7pvmjjasysknmw28>**

**PART 1:**

**Message.java**

**This class will model an email message.  It will have a sender, a recipient, and message text.  *Note: The message text includes the greeting, the body of the message, and the salutation.***

Example:

From: Mark Fowler

To: Samantha Jones

Dear Samantha,

It is my great pleasure to

offer you a position in the company.

Sincerely, Mark

**UML Diagram**

Timeline

Description automatically generated

**Instructions for Part 1:**

1. Please find the Code Check page for this project: [**Chapter 8 Project**](http://codecheck.it/files/20100416242eudkddhq7pvmjjasysknmw28)
2. Using your IDE (jGrasp, Eclipse, other), create a new **project folder** called Message
   1. Inside this project folder, create a class file named Message.java
3. Write the Class comment describing the class and @author and @version tags
4. Declare and initialize **3 instance variables** for:
   1. The recipient as type String
   2. The sender as type String
   3. The messageText as type String  
      * Remember to declare them "**private**" instead of "public".  Also, please make sure to use the "this" keyword anytime you use your instance variables after declaring them.
5. Create a **Constructor** that takes a recipient and sender as parameters of type String.
   1. Inside the constructor, initialize the recipient and sender instance variables with the parameters.  The messageText instance variable can be initialized as an empty String "".
6. Write a mutator method, append, that takes a String line of text as a parameter
   1. Inside this method, *append* (i.e. add onto) the String line of text to the existing messageText separated by a newline, \n.  (*This is a bit like total = total + "\n" + value* *except with Strings*)
7. Write a toString method that returns the entire message in the form:
   1. From: Mark Fowler   //sender instance variable

To: Samantha Jones  //recipient instance variable  
  
Dear Samantha,  //All of this is the message text

It is my great pleasure to

offer you a position in the company.

Sincerely, Mark  //Part of message text

* 1. You can do all of this in one line of code using your instance variables, the words "To", "From", and newline escape sequence, "\n" (or in multiple lines of code).  Just remember that you are returning **one String object** from the method - you are not printing the String.

1. Write a method, print, that prints the output of a call to the toString method.
2. Once you have written all methods of the Message Class, please **test** your methods by downloading and saving the following tester file (you can right-click and save as if it does not automatically download): [**MessageTester.java**](https://gastate.view.usg.edu/d2l/common/dialogs/quickLink/quickLink.d2l?ou=2490275&type=coursefile&fileId=Programming+Project+java+files%2fMessageTester.java)   **This file needs to be in the same project folder** as Message.java in order to run correctly.  Run MessageTester and check for errors in your Message class.
3. We will copy our code over to Code Check for testing and submission *after* Part 2.

**PART 2:**

**Mailbox.java**

**This class will model an email mailbox that stores email messages.  We will be using an ArrayList to store the messages created in the Message class.**

**UML Diagram**

A picture containing timeline

Description automatically generated

**Instructions for Part 2:**

1. Again, we will be using the Code Check site:  [**Chapter 8 Project**](http://codecheck.it/files/20100416242eudkddhq7pvmjjasysknmw28)
2. In the same project folder from Part I, create a new class file named Mailbox.java
3. Write the Class comment and @author and @version tags
4. Create one **instance variable** of type ArrayList<Message> to hold the messages.
5. Write a **Constructor** that takes no parameters
   * Inside the constructor, initialize the messages instance variable as a new, empty ArrayList of type Message.
6. Write a mutator method, addMessage, that takes a parameter of type **Message** (object) to represent a new message.  (Hint: This is just like String message, except the object type is Message instead of String)  \*Remember, we wrote the Message class to represent an email message, just as Java developers wrote the String class for us to use.
   * Inside this method, add the message parameter to the instance variable ArrayList.
7. Write an accessor method, getMessage, that takes a parameter of type int to represent the index of any given message in the ArrayList.
   * Inside this method, get and return the message located at the given index of the ArrayList instance variable (there is an ArrayList method for this!).
8. Write a mutator method, removeMessage that takes a parameter of type int to represent the index of any given message in the ArrayList.
   * Inside this method, remove the message located at the given index from the ArrayList instance variable (there is an ArrayList method for this!).
9. Once you have written all methods of the Mailbox Class, please **test** your classes by **adding**the lines in this text file, [mailbox.txt](https://gastate.view.usg.edu/d2l/common/dialogs/quickLink/quickLink.d2l?ou=2490275&type=coursefile&fileId=Programming+Project+java+files%2fmailbox.txt) (right-click and "save as"), to your existing **MessageTester.java** class (you can right-click and select save as and then cut and paste).  You can add these lines to the end of the statements already in the class.  Do not delete any lines.  Run MessageTester.java and check for errors in your Mailbox class.
10. Now that you have written both classes, go to the Code Check website and copy your code into the ... inside the code.  I have included the **opening and closing { }** for each class - please do not duplicate.  When you submit your code, it will be tested with the given data, and you will have a chance to error check and correct your code.

**PART 3:**

**Testing and Submitting**

Once you have thoroughly tested your program, please upload the Code Check **zipped file** **only** to this Assignment folder.

**You will be graded on:**

1. Neatness of code and use of proper indentation of 4 spaces, 8 spaces, etc.  This is the **Checkstyle**part of your**Report.**
2. Commenting of code - including class comment, @author, @version, and code comments. This is the **Checkstyle**part of your**Report.**
3. Good use of constant and variable names - they should be *descriptive* names and not individual characters or abbreviations.  Ex.  FINAL\_COST or firstName. This is the **Checkstyle**part of your**Report.**
4. Output is correct and matches the Sample Output for all tests that executed on your code in Code Check.  This is the **Test 1, Test 2, etc.**part of your**Report.**They must say "Pass".
5. Correct calculations using variables and constants (i.e. the math is correct).  This is the **Test 1, Test 2, etc.**part of your**Report.**
6. My visual inspection of your code.  I will be checking for good variable and constant names and well as good indenting of code.  Instructor reserves the right to make adjustments to the Code Check score based on inspection.

ALL DONE!!  Great job!