Programming Assignment 7: Writing Classes

Total Points (40 pts) - Due Wednesday, March 1st at 11:59 PM

This programming assignment is intended to demonstrate your knowledge of the following:

- declare a new class and instantiate an object
- write instance methods that return a value
- write instance methods that take arguments

A Triple String Class [40 points]

Create a class called **TripleString**. **TripleString** will consist of three private member **Strings** as its basic data. There will be few instance methods to support that data.

Once defined, we will use it to instantiate **TripleString** objects that can be used in our **main()** method. In a later assignment, the **TripleString** class will help us create a more involved application.

TripleString will contain three private member **Strings** as its main data: **string1**, **string2**, and **string3**. We will also add a few public static member such as final int **MAX_LEN** and **MIN_LEN**. These represent the maximum and minimum length that our class will allow any of its strings to be set to. We can use these static members in the **TripleString** method whose job it is to test for valid strings (see below).

Class TripleString Spec

1. **Private Class Instance Members:**

   - **String string1**
   - **String string2**
   - **String string3**

   All legal strings should be between 1 and 50 characters.

   We never want to see a literal in our methods. So, the class should have **static** members to hold values for the limits described above, as well as default values for any field that is **construct**ed using illegal arguments from the client. These are put in the **public static** section.

2. **Public Class Static Constants (declare to be final):**

   - **MIN_LEN = 1**
   - **MAX_LEN = 50**
   - **DEFAULT_STRING = " (undefined) "**

Public Instance Methods

3. **Default Constructor**

   TripleString() -- a default constructor that initializes all members to **DEFAULT_STRING**.

4. **Parameter-Taking Constructor**

TripleString(String str1, String str2, String str3) -- a constructor that initializes all members according to the passed parameters. It has to be sure each **String** satisfies the class requirement for a member **String**. It does this, as in the modules, by calling the mutators and taking possible action if a return value is false. If any passed parameter does not pass the test, a **default String** should be stored in that member.

5. Mutators/Accessor

Mutators in our course are named using the convention as follows: **setString1(...)**, **setString3(...)**, etc. Likewise with accessors: **getString2()**.

We need an accessor and mutator for each String member, so three pairs of methods in this category.

Mutators make use of the private helper method **validString().** When a mutator detects an invalid **String**, no action should be taken. The mutator returns **false**, and the existing **String** stored in that member prior to the call remains in that member, not a new default **String**.

String toString() - a method that returns a **String** which contains all the information (three strings) of the **TripleString** object. This **String** can be in any format as long as it is understandable and clearly formatted.

boolean validString(String str) - a helper function that the mutators can use to determine whether a **String** is legal. This method returns **true** if the string is **not null** and its **length is between MIN_LEN and MAX_LEN**(inclusive). It returns **false**, otherwise.

Where it All Goes

TripleString should be a class distinct from (and not contained within) your **main class** (which we call **TestTripleString**). You can create the **TripleString** class as a non-public class directly in your client **TestTripleString.java** file.

This makes it a sibling class, capable of being used by any other classes in the file (of which there happens to be only one: **TestTripleString**).

You type it directly into that file; do not ask Eclipse to create a new class for you or it will generate a second .java file which we don't want right now. In other words, **TripleString.java** will look like this:

```
import java.util.*;
import java.lang.Math;
public class TestTripleString
{
   // main class stuff ...
}
class TripleString
{
   // TripleString class stuff ...
}
```

As you see, **TripleString** is to be defined **after**, not within, the TripleStringTest class definition. This makes it a sibling class, capable of being used by any other classes in the file (of which there happens to be only one**: TestTripleString**).

The TestTripleString main()

1. **Instantiate** four or more **TripleString** objects, some of them using the default constructor, some using the constructor that takes parameters.
2. Immediately **display** all objects.
3. **Mutate** one or more members of every object.
4. **Display** all objects a second time.
5. Do two explicit **mutator tests.** For each, call a **mutator** in an **if/else** statement which prints one message if the call is successful and a different message if the call fails.
6. Make two **accessor calls** to demonstrate that they work.

More:

- Be sure that all output is descriptive. Use labels to differentiate your output sections and full sentences in your mutator/accessor tests.
- No user input should be done in this program.

Here is a sample run:

```
TripleStrings after instantiation ----------
(undefined), (undefined), (undefined)
(undefined), (undefined), (undefined)
hello, Foothill, World
1, 2, 3

TripleStrings after changes ----------
(undefined), (undefined), (undefined)
string one, another string, the third string
Pickles, Foothill College, World
Cherry, 2, 3

Mutator Tests ----------

 Correctly rejected blank string
Accessor Tests ----------
 TripleString2 String 3: World
 TripleString3 String 2: 2
```

Submission Instructions

- Execute the program and copy/paste the output that is produced by your program into the bottom of the source code file, making it into a comment. I will run the programs myself to see the output.
- Make sure the run "matches" your source. If the run you submit could not have come from the source you submit, it will be graded as if you did not hand in a run.
- Use the Assignment Submission link to submit the source code file.
- Submit the following file:
    - TestTripleString.java
- Do not submit **.class** files.

Standard program header

Each programming assignment should have the following header, with italicized text, appropriately replaced.

```
/*
 * Class: CS1A
 * Description: (Give a brief description of Assignment 7)
 * Due date:
 * Name: (your name)
 * File name: TripleStringTest.java
```