Computer Science 2 CSCI-142
War Lab 2: In-Lab Activity

1/20/2021

1 In-Lab Activity (10%)

This activity is to be completed and submitted to the MyCourses Assignment dropbox for the In-Lab activity.

2 Implementation

During lab time you will focus on two things:

1. Installing the starter code and setting your project to work with *JUnit*, a popular unit testing framework for the Java programming language.
2. Using the supplied online javadocs, *stub* out each of the required classes so that supplied test code compiles cleanly.

2.1 Starter Code

Download the starter code from the course website and unpack and copy the directories into your root project directory:

`https://www.cs.rit.edu/~csci142/Labs/02/code.zip`

You should have the following directories in your project:

- `sample`, sample runs that you will use for reference later on when writing the main program.
- `src`, the top level source directory which contains two packages:
  `test` - A package that contains all of the supplied test programs.
  - `TestCard.java` - Test unit for the `Card` class.
  - `TestPile.java` - Test unit for the `Pile` class.
  - `TestPlayer.java` - Test unit for the `Player` class.

  `war` - The package that contains all the functional implementation of the game. To begin with, you are provided with:
  - `Rank.java` - A Java enum type for a `Card`'s rank.
  - `Suit.java` - A Java enum type for a `Suit`'s rank.

2.2 Java Documentation

The root directory for the java documentation can be found at:

`https://www.cs.rit.edu/~csci142/Labs/02/doc`

For the in-lab activity, you will stub out the following classes in the `war` package:

- `Card` - Represents a single playing card.
- `Pile` - Represents a pile of cards.
- `Player` - Represents a single player in the game.

Make sure all your classes are defined in this package. The top line of each source code file you create should be:

```
package war;
```

2.3 Activities

1. Create your project and unzip and copy over the code from the starter code.

2. Stub out the classes described for the `war` package above so that the unit tests will compile. When stubbing, you are only concerned with defining each of the classes in the `war` package. There should be no state to begin with.

   Each method (typically `public`) should be stubbed so that it takes the correct number of arguments.
   - For constructors and `void` return methods, there should be no implementation inside.
   - For methods that return numbers, have them return 0. For a `boolean` return method, return `false`.
   - For methods that return object references, e.g. `toString`, return `null`. Take a look at the stubbed out `Card` class for reference.

   Don't concern yourself with commenting for now, Methods that override methods in `Object`, e.g. `equals` and `toString`, should begin their definition with the `@Override` annotation.

3. To begin with, IntelliJ will not recognize that you want to use JUnit for the test classes. Go to the `TestCard` class in the `test` package. Highlight one of the `@Test` annotations and hit the ALT+ENTER keys. This should bring up a menu - select "Add JUnit5.4 to classpath". In the next window to add the library to the project just select the OK button.

4. Now we will make sure you can run the unit test for the `Card` class. In `TestCard`, highlight over the code where the `TestCard` class is declared, right click and select "Run" to create a run configuration. It should run but produce a bunch of errors for tests that have failed.

5. Likewise, you can right click on the `test` package in the project window and run all of the tests. They should all compile but will fail all the tests.

6. From here you should complete the implementation for the `Card` class so it passes all the tests. This time we are giving you the `private` state in the documentation, but that won't always be the case as we move forward in the course. Note that the implementation of the `Card` class is slightly different than what you did in problem

solving. For the rank and suit it uses Java enums. Here is an example of how to use them.

```
Rank ace = Rank.ACE;
System.out.println(ace);        // A
int val = ace.getValue();       // val == 14

Suit heart = Suit.HEART;
System.out.println(heart);      // (heart icon)

Card aceofHearts = new Card(Rank.ACE, Suit.HEART);
```

7. The full implementation of the `Card` class is all you need to complete for the in-lab activity. From there you can continue on with the `Pile` and `Player` classes.