Assignment 1.5: Program Developement, Interitance, and Interfaces

Due: 8amm, Wednesday, October 22nd, 2008

Directions

The problems for this assignment are described below. You will submit this assignment by uploading it to the class website as a Zip file. The file will be named "Q1A5-LastName.zip" (Quarter 1, Assignment 5), where [LastName] is obviously your last name. All of your submitted assignments will be named in this fashion. The Zip file will contain a folder of the same name (so, Q1A5-LastName), which will contain a folder for each of the subsequent problems. Each of these subfolders will have the name of its particular problem. Each of these problem subfolders will contain all of the necessary files for that particular problem. It's very important that you follow all of these naming conventions exactly as specified.

Problems

1) Blackjack

In this problem, you will use the `Card` and `Deck` classes you made in Assignment 1.4 to create a fully functional Blackjack game. You will create another class, called `Player`, which will represent each of the players in the game, notably the dealer and you, the user. Although the dealer, which is controlled by the computer, and the user are different players, the things needed of them are almost exactly the same.

First, let's consider what data we need our `Player` class to have:

- It should have some way of storing the cards in the player's hand. This should probably be a `Card` array, right? What should the length of this array be?
- Since the length of the array is fixed but the number of cards in the player's hand can change, we should also have a variable representing the *actual* number of cards in the hand, with the other empty slots filled with `null` values
- It should also have a simple way of storing the player's name, whether it be "dealer" or "Wulsin." What type should this variable?

We must also consider what methods we would like our `Player` class to have:

- It should be able to print out the cards in the player's hand. Note, since sometimes (in the case of the dealer) we don't want to print out the first card in the hand, this method should have a `boolean` argument so the caller can specify whether to print the first card or something like "[Hidden]."
- It should be able to accept another card into the hand. This method might be called something like `addCard` and take a `Card` object as an argument that is then added to the hand of cards.
- It should be able to return the current sum of the hand. Remember that in blackjack, Aces can be either 1 or 11 and all face cards count as 10.

After creating the `Player` class, you will need to make another class, perhaps called `GameRunner`, that manages the play of the game. In this class you can put all of the code in the `public static void main(String args[])` that actually runs the game:

- Creating the two player objects
- Dealing the first two cards to each of the two players
- Prompting the user whether to hit or stay and then performing the desired action
- Controlling the dealer's actions, hit or stay (the dealer always hits if his score is less than 17)
- Determining whether either of the players has bust (gotten a score over 21) at the end of each round
- Continuing the cycle until both players choose to stay
- Determining the winner of the game

Make sure to test your program extensively. Be attentive to irregularities and mistakes in logic of the gameplay.

Extra Credit Introduce some sort of betting aspect into the game as you like.

2) Triangles

In this problem, you will modify an already existing Class, `Triangle`, which does simple calculations of the area and perimeter depending on what type of triangle (equilateral or right) it is. The existing `Triangle` class, in order to tell specifically what type of triangle it is has a `String` field called `type` that has either "right" or "equilateral" as its value.

Examine the `Triangle` class in `Triangle.java` and understand how its data and methods work.

You will restructure this program in Object Oriented Style (OOP) Make the `Triangle` class abstract. Keep the `side` field but remove the `type` field. Make the `getArea` and `getPerimeter` methods abstract, so

```
public abstract double getArea();
public abstract double getPerimeter();
```

You will need to derive two new classes, `EquilateralTriangle` and `RightTriangle` from `Triangle`. You should provide an appropriate constructor for each of the two derived classes and make them call the superclass's constructor. Redefine the abstract methods appropriately in the derived classes. Put `main` in a separate test class and modify it appropriately. (taken from pg. 319 of *Java Methods*)