CSC 171 PROJECT 1

DUE MARCH 7TH AT 11:59PM

Purpose

The overall goal of CSC 171 is to equip students with the ability to bring computational tools to bear on questions of interest. Labs are designed to provide students with an opportunity to practice the tools and techniques learned in the lectures, workshops, and from the readings with small exercises that can be completed in minimal time. The purpose of projects, on the other hand, is to leverage the design and problem solving skills necessary to write an entire application on a much larger scale.

Many topics of education are built on the paradigm of linear thinking: there is one right answer to every question, and it can be found in the back of the book. Computer science presents a significant cognitive shift for students that are used to linear thinking: there are many ways to approach any given problem and create a solution. There is not one “right” answer; there are many acceptable solutions. It is an important and valuable skill to draw on your existing knowledge base and combine that with the ability to independently and collaboratively research new tools and unknown areas to assemble a cohesive whole. You are encouraged to collaborate with your fellow students, but everything you hand in must be your work and your work only.

Problem Domain: Blackjack

In CSC 171 Lab 04 you are tasked with writing a simplified version of the game “21,” which is also commonly called “Blackjack.” In the simplified version, the program drew cards with a value between 2 and 11 until one of three conditions was satisfied: 1.) the total value of the cards drawn was exactly 21 (win), 2.) the total value of the cards drawn exceeded 21 (bust), or 3.) 5 cards were drawn without the total value exceeding 21 (Charlie). In this simplified version, the computer played by itself and simply drew cards until it won or lost. Other players were not allowed to play, and the artificial intelligence did not make smart decisions about whether to continue drawing cards or to stop.

In this project you will incrementally enhance the “21” algorithm that you created in CSC 171 Lab 04 to implement a complete Blackjack game that allows a human player to compete against the computer to try and get the score closest to 21 without going over. Below is a list of the minimum enhancements required to complete the project.

1. A human player will join the game and play against the computer. The computer will play as the dealer. The player should be prompted to enter his or her name, and then be referred to by that name throughout the game.

2. A deck of cards contains only 4 cards of each value, so the cards drawn must be tracked such that no more than four cards of the same value can be drawn. For example, it should not be possible for a total of more than four cards with a value of "2" to be drawn between the dealer and the player. If the player draws three "2s" the dealer should be able to draw at most one "2" (for a total of four).
    a. Keep in mind that drawing a "10" represents either a "10" or a face card (Jack, Queen, or King). There are a total of 16 cards with a value of 10 in the deck, so it should be possible to draw that many.

3. Cards will be randomly generated with values between 1 and 10 (not 2 and 11 as was done in the lab). Keep in mind that there are 4 times as many cards with a value of 10, and so it should be approximately 4 times more likely to draw a card with a value of 10 than any other card.

4. When calculating the score, cards with a value of "1" (Aces) are special. They are treated as being worth 11 points unless this will cause a bust (a score exceeding 21), in which case they are treated as being worth only 1 point. For example, if the player draws three cards with a value of 1, 5, and 8:
    a. Player draws 1, score is 11. (Ace used as 11)
    b. Player draws 5, score is 16. (5 + Ace as 11).
    c. Player draws 8, score is 14. (8 + 5 + Ace as 1)

5. Before the first turn, one card will be randomly drawn for each player (the human and the dealer). The value of these initial cards must be printed before play begins. For example:
    ```
    The player draws 9.
    The dealer draws 7.
    ```

6. At the beginning of each round, the human player must be given the option to either "Hit" (draw another card) or "Stay" (stop drawing). The player will indicate a choice by typing *either* the entire word, *or* just the first character (it does not matter if the letters are capitalized or not). For example:

```
Player, please choose to "Hit" or "Stay": h
Player, please choose to "Hit" or "Stay": STAY
```

7. If the player chooses to **hit**, another card is drawn. If the player chooses to **stay**, they will not take any more turns for the rest of the game.

8. The dealer will be controlled by a simple AI algorithm and will continue to draw as long as the dealer's score is less than the player's, or when one of these conditions occurs:
   a. The dealer's score is 21.
   b. The dealer busts (score exceeds 21).
   c. The player busts (score exceeds 21).
   d. The dealer reaches a score of 17 or higher when the player has a score less than the dealer's score.
   e. The player stays and the dealer has an equal or higher score.

9. Play continues until the player or the dealer busts, or both stop drawing. The winner is the player that did not bust with the score closest to 21. If both the player and the dealer have the same score, it is a draw.
   a. If the human player draws 5 cards without busting, this is called a "Charlie" and is an automatic win. If this occurs, play should stop immediately and the player is declared the winner. This does *not* apply to the dealer.

10. At the end of each game, the player should be able to choose whether to play another game or to quit.

Additional Enhancements (Extra Credit Opportunities)
Get creative. You may earn up to 20% in extra credit depending on how clever your enhancements are.

Grading Criteria

You will be graded according to the following criteria.

Class Definitions (20%)

While you will need a class with a main method that starts and orchestrates the game, your project should *not* be implemented as a single class with one large main method with dozens or hundreds of lines of code. Appropriate classes should be defined for the different parts of a Blackjack game. Every class should have appropriate state and behavior, accessors and mutators, and constructors. Think about a Blackjack game when designing your classes. What is involved? A deck (or decks) of cards, a player, a dealer, hands of cards that have

been drawn or dealt from the deck, etc. Consider making classes that represent these parts and that work together to make the game work.
Good design is about high cohesion and low coupling. High cohesion means that the purpose of a class is very specific: it knows about a small number of very specific, very related things. Conversely, state and behavior that *should* exist together are not spread across multiple classes.

Low coupling means that the number of dependencies, that is objects that *need* each other to function properly, is minimized, and those dependencies make sense. If class A depends on class B, then changes in class B will often break class A. This kind of thing should be minimized. Circular dependencies (that is to say that class A depends on class B, and class B depends on class A) are even worse, and should be avoided unless there is a very good reason not to.

Blackjack Implementation (40%)

Each of the enhancements listed in the problem domain above are worth 4% of your total grade, for a total of 40%. Remember that it is *always* better to turn in a working program that includes solutions for only some of the required features than it is to turn in a broken program. Implement the features one at a time in the order that makes sense to you. It's always a good idea to keep backups of your code so that you can roll back if you break something that was working before.

You may earn up to an additional 20% for implementing additional enhancements. In order to receive credit for enhancements, they must be *your idea*, and you must fully document the enhancements in your project documentation. Finally, you must specifically ask your TA to consider your enhancements for extra credit.

Testing (20%)

You should write tests for each of your classes. It is strongly encouraged that the unit testing capability integrated into Eclipse is used to implement your tests (jUnit). You can find lots of documentation describing jUnit in Eclipse online. You may also simply write a separate test for each of your classes that includes a main method to test the class. For example, if you implement a "Card.java" class you would create a "CardTest.java" class that uses a main method to create and test a "Card." You will **lose credit** if your test methods are implemented in your classes (as opposed to a separate class).

Code Style (20%)

- Modularity of design - Objects that make sense. High cohesion, low coupling.
- Comments - Where appropriate!
- Names - Variables, Methods, Classes, and Parameters

- Indentation and White Space

Documentation (20%)

In addition to submitting the Java code (.java files), you are required to submit a README text document which specifies the following:

1. A one paragraph description of your class designs.
2. A one paragraph description of your Blackjack algorithm implementation.
3. A one paragraph description of your testing strategy.
4. Detailed instructions on how to compile and run your game and your tests. This should include command line arguments (if any), and appropriate responses to any prompts.
5. Full documentation for any enhancements that you wish to be considered for extra credit. YOU must explain what your enhancements are, and how the TA should enable them or otherwise detect their presence.

Additionally, provide a file called “SampleOutput” that includes the output of at least three full games with one player against the computer controlled dealer.

HAND IN

Projects will be submitted as ZIP archives via Blackboard. You will be given at least 2.5 weeks to complete the project, and there will not be a lab due the same week that the project is due. This is more than enough time to complete the project. You will also have unlimited attempts to submit your project (only the last submission will be graded), which means that you will be able to upload your solution as soon as you have something working, and try for improvements or extra credit afterwards. Because of this, late submissions **will not** be accepted.

You will be expected to submit:

- Java source files and compiled Java class files. You must also submit any additional files required to run the programs.
- Documentation described in the section above. If you submitted any additional files, make sure to explain their function in your documentation.

Grading

Each TA will compile and run your code using his or her own test cases to determine correctness based on the above criteria. If your program produces only partial functionality, providing your own test cases will give you partial credit for the parts that operate correctly. Each TA will also conduct a code review to evaluate the correctness of your code and the code design.