Lab 03 - Blackjack

Dr. Mark R. Floryan

September 16, 2018

1 PRE-LAB

This week, you will be writing some code to play the game blackjack. You will write one class, and also a blackjack playing bot that tries to win as many chips as possible. Your summary:

1. Download the starter code and import the project into Eclipse.
2. Implement the DeckStack.java class.
3. Implement the MyBlackjackPlayer.java class.
4. **FILES TO DOWNLOAD:** blackjack.zip
5. **FILES TO SUBMIT:** DeckStack.java, MyBlackjackPlayer.java

1.1 DECKSTACK.JAVA

Your first task is to implement the DeckStack class. Casinos will sometimes use multiple decks to make cheating harder, and to increase the house odds. For our simulation, we want the dealer to be able to deal cards from a shuffled stack of multiple decks. To simulate this, you will use the Deck.java class from lecture, but write another class that uses it. This class will be called DeckStack.java and it will contain the following fields / methods:

```
//An array of decks of cards that comprise this multi-deck.
private Deck[] decks;

//Constructor: instantiates the number of decks specified and
//adds them to the list of decks
DeckStack(int numDecks);

//Deals the top card from the stack of decks and returns that Card.
//You can implement this several ways, as long as it correctly
//deals a card from on of the decks.
public Card dealTopCard();

//Reshuffles all of the decks.
protected void restoreDecks();

//returns the number of cards left to be dealt in the
//entire stack of cards.
public int cardsLeft();
```

Once you implement this class, the rest of the blackjack simulator has been written for you and should work. You can run the project and see if the simple player we provided plays blackjack (*More detail on the simple player below*).

1.2 Implement a Blackjack Player

Your second task is to implement a basic blackjack player. This involves writing two required (and one optional) methods:

```
/* Returns the number of chips you'd like to bet this hand */
public int getBet();

/* Returns the Move this player would like to do right now */
/* Make move is called until the player returns Move.STAY */
public Move getMove();

/* The dealer will call this method to show you their entire */
/* set of cards once the hand is completely over. You may use */
/* this information if you'd like. */
public void handOver(Card[] dealerHand);

/* The Move enum looks like this */
public enum Move{
```

```
        STAY, HIT, DOUBLE;
16  }
```

Your job is thus to implement getBet() and getMove() methods such that you maximize your final chip count after 1000 games (the simulator automatically plays 1000 hands). Your player will begin with 1000 chips. In order to reason about your best move, you can access the following variables from within your methods:

```
/* Returns the number of chips the player has */
2  public int getChips();

4  /* A list the players cards */
   /* Access using this.cards */
6  protected ArrarList<Card> cards;

8  /* Example of how to see the dealer's face-up card */
   this.dealer.getVisibleCard(); //returns a Card object
```

Your goal, as stated earlier, is to maximize your profit after 1000 games. Good luck!

2 In-Lab

The goal of this in-lab is to practice doing object oriented code design. The TA will lead you through these exercises.

1. Get into small groups of two

2. The TAs will present you with some programming challenges. These will not be graded, but you are required to attend and to participate.

3. You may think about the challenges before lab below if you'd like, but we highly recommend that you not solve them ahead of time.

4. The TAs will give you time to solve each problem and lead you in sharing solutions with one another.

2.1 Object Oriented Code Design

One skill that employers are interested in is object oriented code design. That is, given a set of requirements, how might you structure the classes, fields, and methods within your code to model that situation. For the following scenarios, list out the classes that you would likely write. In addition, for each class write out the fields and methods contained within. Describe how each class relates to one another as well.

You will do this exercise for the following scenarios.

1. You are designing a system that handles flight bookings for customers. Design a set of objects / classes / methods / fields to model this situation. Some of the requirements include:

    a) A flight has a starting airport, a destination airport, a capacity, a price and possibly other features.

    b) The system needs to handle searching for flights, purchasing flights, viewing booked upcoming flights, etc.

    c) The system needs to support user accounts, logins, purchasing information, etc.

    d) Feel free to make other assumptions about useful features as long as you can justify your decisions.

2. Design some object oriented code that supports a ride sharing service such as Uber or Lyft. Some requirements:

    a) There are driver accounts and rider accounts (some folks might be both)

    b) A user can enter payment info

    c) A user can request rides, a driver can accept rides (or can they? Should they have a choice? You design it.)

    d) A user's fare needs to be calculated depending on various factors.

    e) A driver can deliver food from local restaurants for people if they'd like to.

    f) Include all features you can think of from using these services yourself.

    g) Feel free to make other assumptions about useful features as long as you can justify your decisions.

3. Design an object oriented codebase for SIS. Don't forget the following:

    a) You need to account for different user types (e.g., student, professor, admin, etc.)

    b) Students should not be able to accomplish professor tasks and vice versa.

    c) Remember that SIS also handles things such as grade entry, advisor relationships / advisor holds, etc.

    d) As before, feel free to make other assumptions as long as you can justify those decisions.

3 POST-LAB

The goal of this post-lab is to write a report comparing two distinct strategies for your blackjack player. The requirements for this post-lab are VERY simular to last week.

1. Construct at least two distinct strategies for player.
2. Run an experiment comparing how well each strategy plays blackjack.
3. Write a report summarizing and analyzing your findings.
4. **FILES TO DOWNLOAD:** None
5. **FILES TO SUBMIT:** PostLabThree.pdf

3.1 PERFORMING AN EXPERIMENT

Your first task is to write (if you haven't already) **two unique** blackjack players that incorporate different strategies. You should be able to argue that the two strategies are significantly different AND that it is not obvious to you which is better. You should be geniuinely curious which approach will succeed.

You should then do the following:

- Run each of your two players for 10000 hands in the simulator. Run this simulation 10 times, and calculate the average and standard deviation of the results.
- Which bot does better? Why do you think this is? Does one have a higher average but also higher standard deviation? If so, why?

3.2 REPORT

Summarize your experiment and your findings in a report. Make sure to adhere to these general guidelines:

- Your submission MUST BE a pdf document. You will receive a zero if it is not.

- Your document MUST be presented as if submitted to a professional publication outlet. You can use the template posted in the course repository or follow Springer's guidelines for conference proceedings.

- You should write your report as if it is original novel research.

- The grammar / spelling / professionalism of this document should be sound.

- When possible, do not use the first person. Instead of "I ran the code 60 times", use "The code was executed 60 times...".

In addition to the general guidelines above, please follow the following rough outline for your paper:

- **Abstract**: Summarize the entire document in a single paragraph

- **Introduction**: Present the problem, and provide details regarding the two strategies you implemented.

- **Methods**: Describe your methodology for collecting data. How many hands, how many executions, how you averaged things, etc.

- **Results**: Describe your results from your execution runs.

- **Conclusion**: Interpret your results. Which strategy was better? Why was it better? Were you surprised? Was one strategy better in some situations and not in others? Why do you think that is? Notice that I'm not looking for a particular answer here. Show me that you can interpret what happened when you ran your code.

Lastly, your paper MUST contain the following things:

- A table (methods section) summarizing the different experimental groups and how many execution runs were done in each group.

- A table (results section) summarizing each experimental group and the averages / std. dev. for each (as well as any other data you decided to collect).

- Some kind of graph visualizing the results of the table from the previous bullet.