Lab 4

CS 3793/5233 – Fall 2015
Tom Bylander, Instructor

In Lab 4, you will improve a program that plays Slackjack, a solitaire variation of Blackjack. Your grade on the lab will depend on the performance of your program on 10000 games of Slackjack. The initial program is slackjack.zip, which you can download from the course web site.

Slackjack

You can play Slackjack by running the main method in Dealer.java. Your part of the protocol is to enter hit or stand when asked.

In a hand of Slackjack, you are dealt cards until the sum of the cards is greater than 21, or until you stop the dealing by entering stand instead of hit. Once the hand ends, your reward (number of points) is −10 if the sum of cards is greater than 21 (a bust), 42 if the sum of the cards is equal to 21, or otherwise equal to the sum of the cards.

The sum of the cards is determined as follows, the value of an ace can be 1 or 11, the value of a two through ten is equal to that number, and the value of a face card (jack, queen, or king) is equal to 10. The value of an ace is 1 except if changing the value to 11 keeps the sum less than or equal to 21. If your hand contains two or more aces, at most one ace can be 11.

The current Player.java will stand after being dealt two cards.

Dealer.java will keep dealing from a deck until there are few cards left. Specifically, Dealer.java tests, before a hand starts, whether the sum of the remaining cards is greater than 21. If not, Dealer.java will shuffle all the cards and start dealing from a full deck. Note that you get a line that says shuffle if Dealer.java shuffles the deck.

Your Task

Your task is to improve Player.java. In order of difficulty,

1. Always hit if the sum is less than or equal to 11.

2. Take into account that an ace can be 1 or 11.

3. Take into account how many cards of each value is left. This could improve your results in a couple of ways.

   (a) If there are no more cards with a value of 10, then it is completely safe to hit if the sum of the hand is 12.

   (b) Determine the expected reward based on the cards left to be dealt.

   The last option is the one true path for optimal decision-making. Suppose there are 13 cards left in the deck. Suppose a hit is considered. Getting each of those 13 cards will result
in a certain reward. Summing all 13 possibilities and dividing by 13 is the expected reward of a hit. If this is higher than the reward for a stand, then the player should hit.

To do this processing, the program needs to keep track of the deck. The variable `trackValues` in `Player.java` is available for you to code this.

**Your Grade**

Your grade on this lab will range from 80 for an average reward of 16 to 100 for an average reward of 18. Just like the previous lab, there will be bonuses for the best performers. The five best performances in this lab will receive an additional 100, 80, 60, 40, or 20 points. Any ties will result in proportional allocation. For example, if 10 students all tie for the best performance, they will get an extra 30 points each.