CMSC 132

Sections 020X/030X

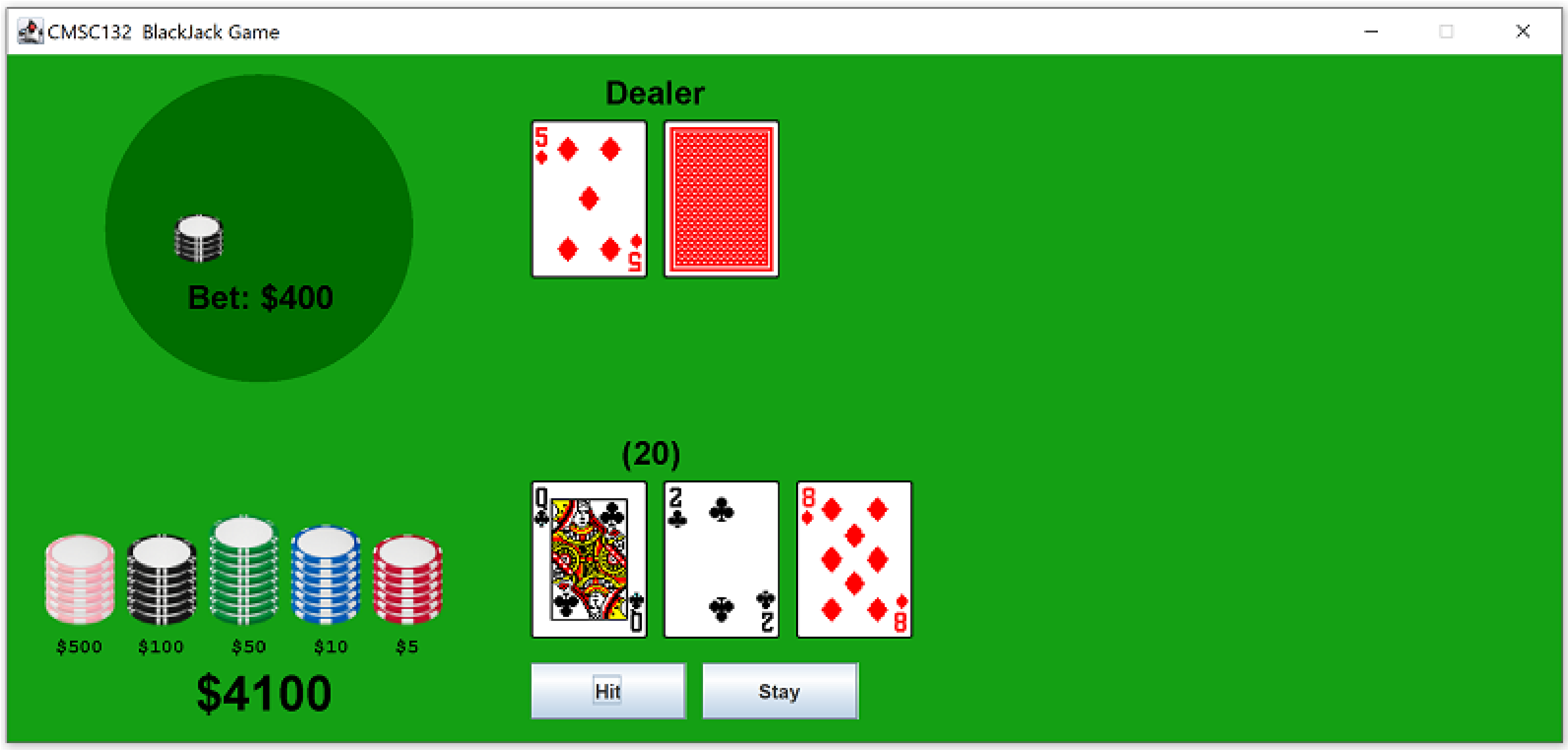
DEPARTMENT OF COMPUTER SCIENCE

Assignment: **Project #1**

Due Date: **Wednesday 02/09, 11:00PM**

Open/Closed policy: **OPEN**

Blackjack Game



Objective

This project will give you a review/warm-up of basic Java and Eclipse, as well as providing some practice with Java's enumerated types, and the ArrayList class.

Overview

You will implement the "model" (logic and data structures) for a simulation of the game Blackjack, pictured above. The GUI (Graphical User Interface) has been provided for you.

In our game, there is just one player (controlled by the user) who plays against the dealer.

To simplify the implementation, we've left out a few features of the game that you may be familiar with if you've ever played Blackjack in a casino: splitting pairs, doubling down, surrender, insurance, and "even money" payoffs are all left out of our simplified version of the game. It's still kind of fun.

Cards

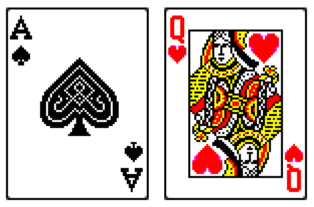
If you are not familiar with the standard deck of 52 playing cards and the terminology that goes along with it ("rank" "suit", "queen", "clubs", etc.), then please take a look at the Playing Card Page.

Blackjack hands

Even if you are an experienced player, you should review the Blackjack Hands Page. This page will define how Blackjack hands are assigned values in our game. Be sure to look over the examples near the bottom, because they describe precisely how some of the trickier Blackjack hands will be assigned values by the code that you will be implementing.

The Special Hand Called "Natural Blackjack"

There is one very special hand called the "Natural Blackjack". It consists of exactly two cards that total exactly 21. In other words: An Ace together with one of the cards with a value of 10. For example, the hand below is a "Natural Blackjack":


Note that a hand that totals 21 but consists of more than 2 cards is not a Natural Blackjack.

If the player has a Natural Blackjack (but the dealer does not have one) then the player wins 1.5 times his/her bet! If both the player and the dealer have a Natural Blackjack then it's a "push", which means the player neither wins nor loses.

Rules of Blackjack

Betting
Before any cards are dealt, the player places a bet, using colorful chips that represent dollar values. In our simulation there are chips representing $5 (red), $10 (blue), $50 (green), $100 (black), and $500 (pink). (See the image above.)

Initial Cards
The dealer initially deals out two cards for the player, and two cards for herself. Note that one of the dealer cards is visible to the player, and the other one is face-down (hidden).

Player's Turn
Now the player takes their turn. The player may choose to either "hit" (take an additional card) or "stay" (ending the turn). The player may hit as many times as they desire as long as the hand's total is under 21. If the total exceeds 21, the player has "busted", which means they automatically lose and their turn is over.

Dealer's Turn
Once the player has decided to "stay" (or has busted), it is the dealer's turn. At this point the dealer will flip over her hidden card so that the player can see it. The dealer in our game must always play according to the following rules:

- As long as the dealer's hand is 16 or less, she must continue taking cards.
- If the dealer's hand reaches 18 or more, she must stop taking cards. This includes cases where an Ace is present and the hand could be assigned a lesser value. In other words, the dealer would stop if she had a hand like {3, 4, Ace} because this hand's value could be considered 18.
- If the dealer's hand includes an Ace, and the hand's value could be either 7 or 17 then she must take another card. In other words, the dealer would take another card if she had a hand like {2, 4, Ace}.
- If the dealer's hand is valued as 17 and could not be valued as 7 then she must stop taking cards. For example, if the dealer had the hand {8, 8, Ace} or the hand {5, King, 2} then she would stop taking cards.

Win, Lose, Push, or Extra Winning (Natural Blackjack)
Once the dealer has stopped taking cards, it's time to determine whether or not the player has won. Here are the cases -- read them carefully:

- If the player has a "Natural Blackjack" but the dealer does not have a "Natural Blackjack" then the player wins extra! They are awarded 1.5 times their bet!
- If both the player and the dealer have a Natural Blackjack then it's a push (the player neither wins nor loses).
- If the player does not have a Natural Blackjack, then the outcome is determined as follows:
    - If the player has busted then they lose their bet no matter what the dealer's hand looks like. ("Busted" means that the total of their cards is more than 21.)
    - If the player has not busted but the dealer has busted then the player wins their bet.
    - If neither the player nor the dealer has busted then the player wins their bet if their hand's value is higher than the dealer's, loses the bet if their hand's value is lower than the dealer's, and "pushes" (neither wins nor loses) if their hand's value is the same as the dealer's. Note that in the case where the dealer has a Natural Blackjack, and the player has 21 (but not a Natural Blackjack) then it is a push (not a loss for the player), since their hands carry the same value.

Getting Started

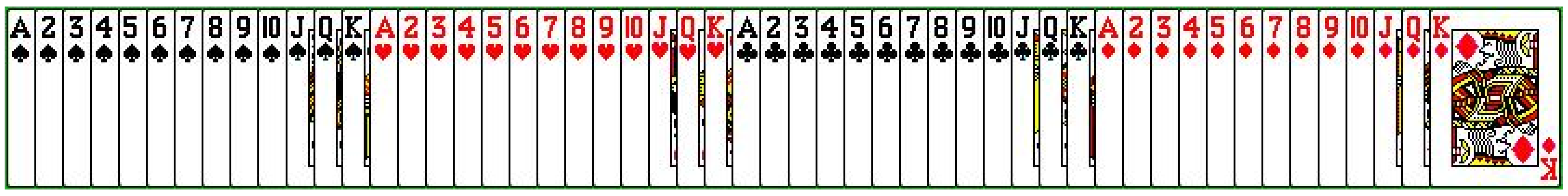
Begin by downloading the project files here. You'll need to import them into Eclipse, as described in the Importing Projects section of the Eclipse Tutorial.

What You Must Implement

Package deckOfCards
Begin by looking in the package called "deckOfCards". Take a look at the enumerations called "Rank" and "Suit", and look over the Card class. These have all been provided for you.

Your first task will be to provide an implementation of the Deck class, which represents a standard deck of 52 cards. Your implementation must include the following members:

- A private instance variable representing an ArrayList of Cards.

- A constructor that takes no parameters. The constructor should instantiate the list of Cards, and populate it with the usual 52 cards found in a deck. The order of the cards matters: The first card in the list should be the Ace of Spades; the last card in the list should be the King of Diamonds. Hint: Do not write 52 statements, one for each card. You should be doing this with loops! Your deck of cards should be initialized so that it looks like this:


- public void shuffle(Random randomNumberGenerator)
This method's implementation will consist of just one statement, which is a call to the version of Collections.shuffle that takes two arguments. The first argument should be your list of cards. The second argument should be the parameter of this method, randomNumberGenerator. This will do a good job of shuffling the deck, randomly. We have included a random number generator as a parameter so that while testing the project we can reliably generate the exact same shuffles over and over.

- public Card dealOneCard()
This method will remove one card from the front of the list (index 0) and return it.

Package blackjack
There are two Java enumerations here that you should look at first: HandAssessment and GameResult.

Your main task will be to implement the class called BlackjackModel. Include the following members:

- Private instance variables:
  - dealerCards -- an ArrayList of Cards that will be used to store the dealer's cards.
  - playerCards -- an ArrayList of Cards that will be used to store the player's cards.
  - deck -- a Deck variable, representing the Deck of cards for the game.

- Getters and setters for both dealerCards and playerCards with the prototypes below. Be sure not to introduce privacy leaks! (When someone calls the getter, it should not give them the ability to modify the cards that the player/dealer is holding. And after someone calls the setter, changes they make to their local list of cards should not affect the list of cards that the player/dealer has been assigned.)
public ArrayList<Card> getDealerCards()
public ArrayList<Card> getPlayerCards()
public void setDealerCards(ArrayList<Card> cards)
public void setPlayerCards(ArrayList<Card> cards)

- public void createAndShuffleDeck(Random random)
This method will assign a new instance of the Deck class to the deck variable, and will then shuffle the deck, passing the parameter (random) along to the deck's shuffle method.

- Methods to deal the initial two cards to the player or to the dealer. Each of these methods will instantiate the respective list of cards (playerCards or dealerCards) and then will deal two cards from the deck and add them to that list. You may assume that the createAndShuffleDeck method has been called before the framework calls either of these methods:
public void initialDealerCards()
public void initialPlayerCards()

- Methods to deal just one card to either the player or the dealer. Each of these methods will simply deal a card from the deck and add it to either the playerCards list or the dealerCards list. You can asume that the initialPlayerCards and initialDealerCards methods have been called before the framework calls either of these:
public void playerTakeCard()
public void dealerTakeCard()

- public static ArrayList<Integer> possibleHandValues(ArrayList<Card> hand)
This method will evaluate the hand in question and return a very short ArrayList that contains either one or two Integers, representing the value(s) that could be assigned to that hand. You can assume that the hand consists of at least 2 cards. In most cases, the value of the hand can be represented by a single integer. For example, if the hand looks like {2, 3, 4}, then the return value should be a list containing just the value 9. When Aces are involved, there could be two values assigned (but not always). For example, for the hand {Ace, 2, 3}, the return value should be a list containing the numbers 6 and 16, but for the hand {A, 6, 5} the return value should be a list containing just the value 12, since 22 is too large (it's over 21). One other case to

consider: The hand {A, 8, 4, Q} should return a list containing just the value 23 (do not include 33). The size of the return value should always be either one or two. In cases where there are two values, list the smaller one first. Read over the information in the Blackjack Hands page very carefully to know exactly what values (and how many) should be returned.

- `public static HandAssessment assessHand(ArrayList<Card> hand)`
  This method will assess the hand and return one of the four HandAssessment constants, as follows:
  - INSUFFICIENT_CARDS -- if the hand is null or contains fewer than 2 cards
  - NATURAL_BLACKJACK -- if the hand represents a "Natural Blackjack"
  - BUST -- if the hand's value is over 21
  - NORMAL -- if none of the categories above apply

- `public GameResult gameAssessment()`
  This method will look at the playerCards and the dealerCards and determine the outcome of the game, returning one of the GameResult constants (PLAYER_WON, PLAYER_LOST, PUSH, or NATURAL_BLACKJACK). NATURAL_BLACKJACK should only be returned in the case where the player wins 1.5 times their bet. Review the section above, titled "Win, Lose, Push, or Extra Winning (Natural Blackjack)", which defines the logic that you must implement for this method. You may assume that this method will only be called after the game has ended (so both the player and the dealer will have taken turns, and each will have at least 2 cards).

- `public boolean dealerShouldTakeCard()`
  This method will look at the dealerCards to determine whether or not the dealer should take another card during her turn, returning true if the dealer should take another card and false otherwise. The method has no side effects. (So don't actually give the dealer another card.) Review the section above, titled "Dealer's Turn", which defines the logic you should implement for this method.

JUnit Tests

We are providing you with a set of JUnit tests (tests/PublicTests.java) that you can run yourself on your code to test its basic functionality.

The submission server will also include "release tests", which will give you limited feedback. You can't see the release test code, and you can only see the results of these tests a limited number of times (roughly 3 times per day).

The submission server will also include one "secret test". You will not be able to see whether or not you are passing this test until after the due date has passed. It is very important for you to learn to test your own code!

Requirements

- Be sure to use proper Java generic notation, when necessary to avoid warnings in Eclipse.
- As with all programming assignments:
  - You must use meaningful variable names and good indentation.
  - You must avoid code duplication, by calling appropriate methods (rather than cutting and pasting code). You may define your own private utility methods to perform often repeated tasks.
  - Because we are using an automatic testing system, it is important that you adhere closely to the specifications.
- You may not use any of the Java collections classes other than the ArrayList class.

Running the Game

When you're done implementing both the Deck class and the BlackjackModel class, you can have some fun by running the GUI. Run the main method in the class GUI/BlackjackGUI. The GUI is not designed as a diagnostic tool -- you should be testing your code using JUnit tests, including the public tests that have been provided, tests you write yourself, and the release tests that are available on the submit server (see the section below).

Submitting the Project

Submit your project from Eclipse (within Java perspective) by right-clicking the project folder and selecting "submit" . **If you do not see the submit option, then you are missing our Eclipse "course management plugin". You can install this plugin by following the instructions in the** Eclipse Tutorial. (Scroll to the bottom of that page for instructions on installing the plugin.) You may submit as many times as you want -- we will always grade the submission that we receive last. After you have submitted your project, you should visit the submit server. There you can obtain limited feedback about how well your project is performing. The number of times you can run our tests on your project (before the due date) is limited. **The earlier you begin working on the project, the more opportunities you will have to see how your project performs on our tests before the due date!**

Grading

Your grade will be computed as follows:

- Public Tests: 15%
- Release Tests: 60%
- Secret Test: 15%
- Style: 10%. (See our Style Guide for a detailed description of what we are looking for with respect to coding style.)