Computer Science 152

Assignment 5

Due: Monday March 19 (moodle)

Total: 50 marks

General Instructions

- *Assignment 5 is a revision of Assignment 1 to incorporate a GUI interface and implements a simulation of a Blackjack card game. The assignment is done in a* ***two-person team****. Submit one assignment for your two-person team.*
- *Your assignment should be printed on (standard) 8.5x11 inch paper and placed in a 9x12 envelope. The envelope should be labeled with this information:*
  - *Your names, and course number*
  - *Assignment number and date submitted*
- *Submit a print out of all Java "source code" (via JCreator)*
- *Combine all your Java files into a single a compressed file. Please use a filename which includes your UPEI username and is of the form:* ***username_ast5.zip****; submit the zip file via moodle no later than March 19. Submit your hardcopy in class on Tuesday, March 20.*
- *Image icons for the playing cards are on moodle.*

The Game

Blackjack (Twenty-one or Vingt-et-un) is a popular casino betting card game dating back to the French casinos in the 1700s. The name "blackjack" was introduced in America for a hand consisting of the Ace of spades and a "black" jack (clubs or spades). Today, a two-card hand consisting of an Ace and a ten-value card is called a 'blackjack" or a "natural" and is an automatic winner unless the dealer also has "blackjack".

Blackjack is usually played with a "shoe" of cards (a collection of four or more decks). Typically, a game of blackjack involves multiple players playing against a dealer. Each playing card in a hand is awarded points based upon its face value. Face cards (king, queen, jack) are worth 10 points each; numeric cards (2 .. 10) are worth their face value; Aces are worth either 1 or 11 points. The goal of the game is to be closer to 21 than the dealer without going over 21; a hand with a higher total than 21 is said to "bust". A player who busts loses the game, even if the dealer busts. If a player and the dealer have the same point value, it is called a "push" and neither player wins. (A rule variation, not in the standard game, declares that a dealer wins tied games.)

Initially, a player and the dealer are dealt two cards. A player can see their own cards and one card of the dealer. A player then has the choice to "hit" and take another card or "stand" ("stick", "stay") and take no more cards for this game. Once the player elects to stand, the dealer then chooses to hit or stay.
The dealer must hit on 16 or less and must stay otherwise.

A hand where an Ace is counted as 11 is called a "soft" hand since it cannot be busted if a player draws another card. An Ace is considered to be 1 point rather than 11 points if it causes a player or the dealer to bust.

Establishing the Requirements

The objective is to allow a user to play a series of games against the dealer (computer). The placing of bets is a requirement. Establish a fixed initial amount to play each game; also, as a player draws a card, you may specify a bet as a fixed amount. Assume the dealer matches a players bet. Keep track of the player betting chips (money won or lost) over a series of games.

Creating a Design

- Consider including classes such as:
  - **PlayingCard**, **DeckofCards**, **BlackjackHand, PlayBlackjack, PlayingCardsPanel** and **CardGameTester**.

- The **PlayingCard** class represents a single playing card. It has instance variables for the card suit (club, diamond, heart, spade) and card value (2, 3, …, 10, J, Q, K, A), a graphical image (ImageIcon), and a boolean "faceUp" (is the face of the card visible?).
  Define methods:
  - **compareTo** – compares this card to a given Card and returns -1, 0, or 1 depending on whether it is less than, equal to, or greater than the given card. Use the card value to compare cards; 2 ... 10 use the value; J (jack) is 10; Q (queen) is 10; K (king) is 10; A (ace) is 1 or 11.
  - **toString** – overwritten to return card information, like "5-club"
  - **getImage** – returns the graphical image of the card
  - **flip** – turns the card over. (i.e. face-down becomes face-up or visible, and vice-versa)
  - **isFaceUp** – returns a Boolean (i.e. is the face of card visible?)

- The **DeckOfCards** class holds 52 playing cards. It should have the following public methods:
  - **create** – initializes an array holding the 52 standard playing cards. Cards are face down. *Hint: Use integer division and remainder to convert the numbers 0..51 into the corresponding card suits and values.*
  - **shuffle** – shuffle the deck.
    Hint: For each card in the deck, exchange it with a card in a randomly chosen position. That is, generate a random number in the range 0..51 and swap the two cards.
  - **dealUp** – returns a card "face-up" and remove it from the deck.
  - **dealDown**– returns a card "face down" from the deck.
  - **cardsLeft** – returns the number of cards left on the deck.

- The **BlackjackHand** class holds up to 5 playing cards. It has public methods to:
  - **addCard** – add a Card to the hand.
  - **value** – returns the value (or total points) of the hand
  - **size** – returns the number of cards in the hand.
  - **reduceHand** – if required reduce value of hand containing Aces.
  - **toString** – returns information on all cards in the hand in sorted order (i.e. 5-club 7-heart 7-club K-diamond).
  - **compareTo** – compares this hand to a given blackjack hand and returns -1, 0, or 1 depending on whether it **evaluates** less than, equal to, or greater than the given hand.

– The **PlayBlackjack** class manages a simulation for one player playing a Blackjack game against a dealer.
It proceeds something like this:
  - Create and shuffle a deck of cards.
    - Deal two cards to a player and show the hand.
    - Give two cards to the dealer (one face-up; one face-down).

  - No more cards are dealt, if someone has blackjack

  - At this point, the dealer will bet 2 chips. The player can choose to "fold" his/her hand; otherwise it costs the player 2 chips to stay in the game.

  - The game continues with the player going first, responding when a player hits or stays, and determining if a player has busted.

  - If necessary, the dealer plays his/her hand, according to the "house" rules as defined in the game. The dealer sticks on 17 through to 21.

  - If neither hand "busted", reveal the value of each hand (and the cards), announcing the winner of a game.

  - At the start of a new game, check the deck size. If there are less than 30 cards in the deck, create and shuffle the deck.

– The **PlayingCardPanel** class displays the game board which includes the card hands and may display the game results.

– The **CardGamePanel** class creates a simulation for one player playing a game of blackjack against a dealer. It proceeds something like this:

Start Create and shuffle a deck of cards.
  - Deal two cards to a player and show the cards in the hand.
  - Give two cards to the dealer (first cardface-down; second card face-up).
  - The game is over, if someone has blackjack

Fold The player decides not to match the dealer's initial bet to draw a card.
  - The player loses.

Hit Deal one (1) card, face-up, to the player
  - The game is over, if the players "busts".
    - The player's turn is over, if they have a 5-card hand valued at 21 or less.

Stick Computer/dealer plays his/her hand, according to the "house" rules as defined in the game.
  - The dealer's face-down card is turned over.

Showdown The dealer plays its hand and a winner is determined.
  - On showdown reveal dealer's hand.
  - On fold, player loses.

As the player continues playing more games, develop a scoring system for displaying the amount of chips the player has for betting. On startup, give the player 100 chips.

Implement the Code

Use an evolutionary development model to design and implement this application. Assign specific roles to team members. As you refine your design for the game, write the corresponding program code, and test this refinement.

Testing the System

Try to discover "bugs" in your application. Allow sufficient time for fixing problems. When you're confident that it is working correctly, "release" your work.

Grading

Program Design / Elegance **(7 marks)**

The degree of sophistication of your application is judged. Is there a "good" graphical user interface. How closely does it match a "real' game of Blackjack?

Program Documentation / Style **(5 marks)**

Programs should be commented, including a block comment at the beginning of each class and each method. Proper use of indentation, adequate white space, good identifiers and appropriate choice of language structures are expected.

Program Code **(30 marks)**

Testing / Program Correctness **(8 marks)**

Make sure you test your programs. The grader will not debug your program.