**[](http://en.wikipedia.org/wiki/File:BlackJackGame.jpg)CSC251 Advanced Java  
First Project: Blackjack**

The project is the design and implement of a java program which is capable of playing a series of Blackjack hands. For consistency, everyone will use the rules from en.wikipedia.org/wiki/Blackjack as qualified below.

1. The game will support from 1 to 4 decks (chosen at the beginning of play)
2. The game will support either “hit soft 17” or “stand on all 17’s” (specified at the beginning of play). A “soft 17” is when the dealer has an ace. For example, an ace and a 6 could be 7 or 17 (that’s why it’s called a soft 17)
3. Payoffs are 1:1 or 3:2 for player blackjack
4. Minimum/maximum bet is $2 to $100 in increments of $2
5. Player decisions are: Hit, Stand, Double down
6. Not supported: Double after split, double on 9/10/11 or 10/11, European no-hole-card rule, altered payout, or insurance.

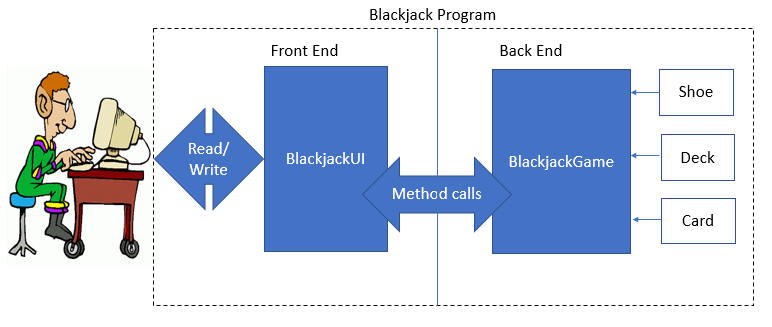
Your program will be the ‘dealer’ and will play against a single player. At the beginning of the game, the player will be credited with $1000 in betting chips. The game ends whenever the player is broke or the player indicates to terminate the game. The game will keep track of the player’s chips and adjust it accordingly based on wins/loses. The user interface should permit the player to interact with the game in a fashion analogous to playing the game in a casino. To wit:

* Cards are dealt as follows: Player, Dealer, Player, Dealer. When the cards are dealt, both player cards are face up. One dealer card is face up the other face down. The player’s hand value is displayed (NOTE: could be two values if one of the cards is an ace). The player’s hand value is updated when he/she is dealt an additional card.
* Until the player stands or goes bust, each hit request causes one more card to be dealt to the player face up. If after the first 2 cards are dealt to the player and the player opts to double down, the initial bet is doubled (provided there’re enough chips in the cache to do so) and the player is dealt only one additional card.
* If the player goes bust, the program indicates that fact. No additional cards are given to the dealer and the face down card remains face down (we won’t ever know its value).
* If the player stands, the dealer’s face down card is revealed, any additional cards dealt to the dealer are shown and a statement as to who is the winner is displayed and the player’s chip count is updated. If the hand is a tie, the player’s bet is maintained but the player has the option to increase the amount of the bet.
* The user interface displays the number of cards remaining in the shoe…this number is updated as each card is dealt.
* A sample program execution is shown later in this document. Yours may vary somewhat, but it should still contain the data shown and the interaction should be very similar.

You’re expected to create a class (BlackjackGame) which contains the processing logic. One other required class is the Shoe. The Shoe holds the shuffled cards and it is from the Shoe the cards are dealt. After each Blackjack hand is played, the cards on the table are collected and placed in a hopper (a holding area). When the last card is dealt from the Shoe, the cards in the hopper are shuffled and transferred to the Shoe and the program continues. While not required, most likely you’ll have Card, Hand and Deck classes, but that’ll be up to you.

Neither BlackjackGame, Shoe, or any of the supporting classes will have any display capability. For that, you are to create a **console** user interface (UI) front-end which is the go-between between the user and BlackjackGame. The class, BlackJackUI is the mechanism by which all interaction/output is accomplished via the interaction pane. In other words, the back-end classes (BlackJackGame, Shoe, et al.) know nothing about the front-end’s (BlackJackUI) implementation or vice versa.

Summarized, as an illustration:



**Sample program execution:**

Welcome to BlackJack...first a few preliminaries

Enter your name (no spaces)

Doc

How many decks are being used? (1-4)

3

Now finally, does the dealer have to hit on soft 17? (Y or N)

Y

OK...Enter S to start or X to exit

S

Running BlackJack

Shoe now has 156 cards

DOC, you have $1000. Enter your bet or exit (X)?

100

Shoe now has 152 cards

Dealer: 8D XX

DOC : QD 2D (12)

Bet=$100 Cache=$900

Options: Hit (H), Stand (S), Doubledown (D), Exit (X)

H

Shoe now has 151 cards

Dealer: 8D AC

DOC : QD 2D TS

Sorry... you're busted.

DOC, you have $900. Enter your bet or exit (X)? 150

Shoe now has 147 cards

Dealer: JC XX

DOC : 6D 7C (13)

Bet=$150 Cache=$750

Options: Hit (H), Stand (S), Doubledown (D), Exit (X)

H

Shoe now has 146 cards

Dealer: JC XX

DOC : 6D 7C 4H (17)

Bet=$150 Cache=$750

Options: Hit (H), Stand (S), Doubledown (D), Exit (X)

S

Shoe now has 145 cards

Dealer: JC 4S 7S

DOC : 6D 7C 4H

Dealer has 21, you have 17 Sorry... you lose

DOC, you have $750. Enter your bet or exit (X)?

300

Shoe now has 141 cards

Dealer: AC XX

DOC : 7H JH (17)

Bet=$300 Cache=$450

Options: Hit (H), Stand (S), Doubledown (D), Exit (X)

D

Shoe now has 139 cards

Dealer: AC 3D 7D

DOC : 7H JH 4D

Tie game. Your current bet is $600 Do you wish to make an additional bet? Yes (Y) or No (N)

N

Bye

**Submission Requirements:**

* Document each “back end” class by describing its:
  + Data members
  + Constructors
  + public methods (functionality, parameters, return value)
  + private methods (if any, functionality, parameters, return value)
* Junit test files for all “back end” classes. Include a screenshot showing the outcome of running the unit tests.  
  NOTE: for testing purposes provide a compile-time variable (e.g., boolean testing = true/false;) in the Shoe class. If this variable is false, the Shoe deals cards from a set of shuffled decks. If this variable is true, Shoe reads cards from a text file and deals those cards. Ergo, when testing, Shoe reads a “stacked deck” so that when dealt, specific Blackjack hands can be handled.
* Provide the “stacked deck” file and all of the .java files in your program/package
* Sufficient test runs (with screenshots) with testing = true that illustrate requirements 1-5 and specified game progression are implemented.
* Additional test runs (with screenshots) with testing = false that illustrate the randomness of the hands due to using shuffled cards.
* Short feedback as to your experiences, trials, tribulations encountered in the project.

Place all of the above in a zip file named Blackjack\_yourname.zip and submit.

**The project is worth 100 points and represents 12.5% of your overall grade.**

Grading Rubric

|  |  |  |
| --- | --- | --- |
| Item | Evaluation | Value (max) |
| “Back end” classes documentation | Completeness | 10 |
| “Back end” classes implementation | Coding quality, OO design, in line comments | 15 |
| Junit testing | Tests all methods (directly or indirectly). Comprehensiveness. | 15 |
| Incorporation of compile time testing variable and use of a “stacked deck” text file | Works as specified | 12 |
| BlackjackUI implementation | Coding quality, style, OO design, inline comments | 15 |
| Interactive testing of BlackJackUI | Satisfies stated requirements, appropriate user interaction and program response, adequate error checking of user input | 10 |
| Submission package | Single zip file with all required documents | 5 |
| Screenshot document | Contains all requested images, cropped to only show the essential information, annotated with descriptive comments. | 10 |
| Feedback statement |  | 3 |
| Overall quality of project | Subjective evaluation of the submission as a whole | 5 |
| Total |  | 100 |

**Extra credit opportunity**

You can earn up to an additional 10 points by adding the following feature: The user should be able to save and exit Blackjack. Upon restarting Blackjack, the user can “restore” the game to the state of the saved version. This will require that the Shoe, discards, chip cache, etc. are saved and are used to restore the game to that same state. Serialization is one approach to accomplishing this task.

If you opt to take on the extra credit, include in your submission: a description of the game save, restore approach, screenshots showing the use of this feature.