

PG3302 – Exercises for session 03 & 04

NOTE: These exercises will span two sessions: Both #3 and #4.

Task 1 – Deck of cards

Create a deck of cards using object-oriented C# programming. Here are some suggestions for the card deck. (But the task is open for interpretation, and you can make some adjustments/additions if you like.

- a) Create a class `PlayingCard` that represents a normal playing card and contains:
 - Suit/type: Can have the values clubs (♣), diamonds (♦), hearts (♥), or spades (♠)
 - Value: Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, or King
- b) A class `CardDeck` that represents one deck containing 52 cards of type `PlayingCard`. (Or, as an option, possibly several decks mashed together.) The `CardDeck` should contain:
 - At least one data structure to store all `PlayingCard` objects. E.g. `List<PlayingCard>`.
 - A constructor. In the constructor one could set such things as whether drawn cards are automatically added to the discard pile or not (defaults to true). One could also add the option to automatically reshuffle the discard pile when the `PlayingCard` data structure is empty. And, if using several decks mashed together, the number of card decks to use (defaults to 1).
 - A method `Draw` that retrieves the top card from the `PlayingCard` data structure. This card should be returned from the method and possibly also be put in a “discard pile” data structure (depends on constructor settings). If the deck is empty, either return null or shuffle the discard pile back in and then draw.
 - A method `Discard` that places a card in the discard pile.
 - A method `shuffle` that shuffles the discard pile back into the `PlayingCard` data structure.

Task 2a – Small card game

Note: If you want to get to the larger task 2b (BlackJack) as soon as possible, it's ok to skip this task. But this one (2a) is considered an easier start. If you do this first, its ok not to also finish 2b.

Create a game class `PlayingCardGame` that uses at least one instance of `CardDeck` to play a card game. `PlayingCardGame` should create functionality to:

- Create a mixed deck that the game can use.
- Be able to play as one player or more (last option only if the game you create can be played by several players).
- Decide if one wins/loses/draws. In other words - you must be able to end a card game.
- Contain a menu that gives the user the following options:
 - Play a game
 - Show game rules
 - End the program

Possible game: “High or Low?”

Two face-down cards are taken from the deck: one card is given to you, and the other is placed face up so that you can see it. You must guess whether your hidden card is higher or lower than the open card. If you guess right you win, guess wrong you lose. If the cards have the same value, you can compare the type: spades (♠) > hearts (♥) > diamonds (♦) > clubs (♣).

After a game, both cards must be placed at the back/bottom of the deck, and you may be asked about one wants to play again or go back to the main menu.

Task 2b – Blackjack

Note: This is a *large* task. You’re not expected to finish this during the exercises for the current week.

Blackjack is a card game where the player’s goal is to:

- Get Blackjack (two cards that add up to 21).
- Or a higher sum than the “dealer”, without exceeding a sum of 21.

You get dealt 2 cards each. Every picture card counts as 10 (as do 10’s). Aces counts as 11, but if you exceed a sum of 21 with an ace, it counts as 1 instead. (Can be repeated for each Ace you have.)

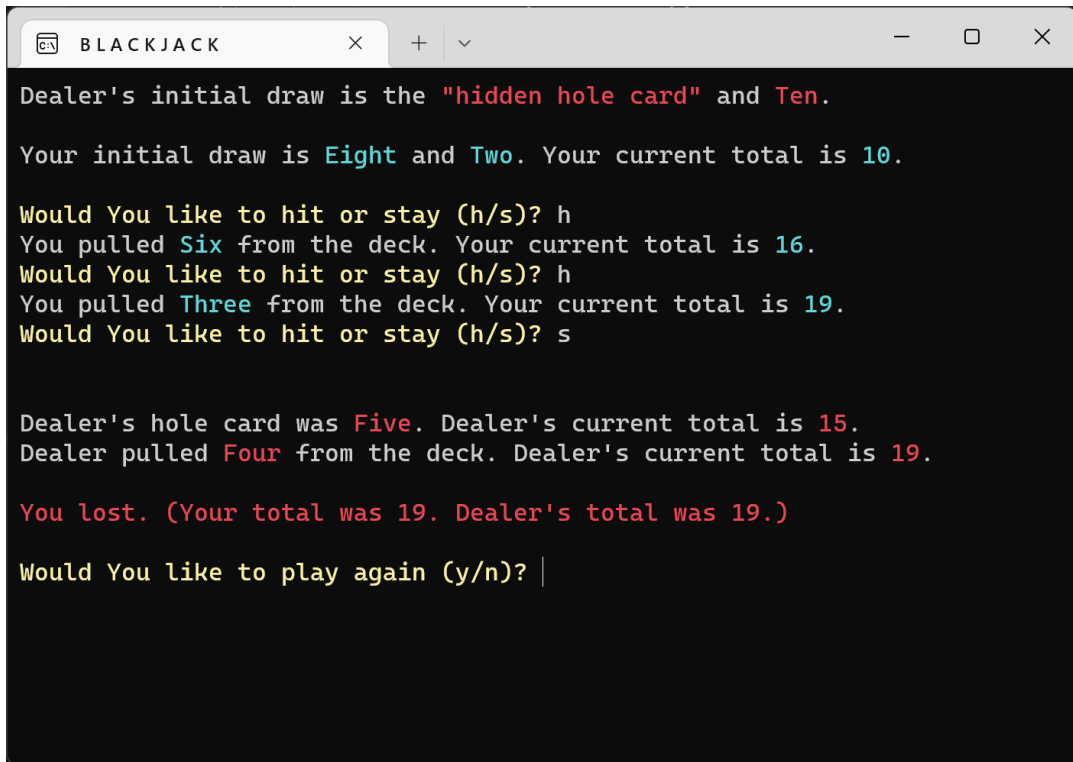
The player only sees one of the dealer’s two cards. The player can ask for a new card (“hit”) as many times as the player wishes, with each card added to the total sum. If the sum exceeds 21, the player loses. If the sum is exactly 21 on the first two cards, the player has “Blackjack” (if the sum is 21 on more than two cards, the player has “21”). As long as the sum is less than 21, the player can ask for more cards. When the player is satisfied with the number of cards (and still has a sum of 21 or less), the players choose to “stay”.

After the player has finished (unless the player lost, in which case the dealer wins), the dealer must draw cards until their sum is 17 or more (then the dealer must stop). The dealer loses if their sum exceeds 21, just like the player. If the sum is equal for the player and dealer, the dealer wins.

What you need to make:

- a) Think object-oriented. Come up with an overview of which objects (classes) you think should be included in Blackjack, based on how it is described above. Try to think about what kind of functionality should be in each class. Create a model before you start coding:
 - Draw a UML Implementation Class Diagram with the classes you plan on using.
 - Write methods, properties and variables.
 - Note what is public and what is private.
- b) Implement the game. Here are a few things to consider:
 - Classes
 - Properties
 - Public vs. private accessors.
 - Using the Random class to draw cards.
 - Placing the right method in the right place.
 - Other stuff?

The result of the game *could* look something like this:



```
BLACKJACK
Dealer's initial draw is the "hidden hole card" and Ten.
Your initial draw is Eight and Two. Your current total is 10.
Would You like to hit or stay (h/s)? h
You pulled Six from the deck. Your current total is 16.
Would You like to hit or stay (h/s)? h
You pulled Three from the deck. Your current total is 19.
Would You like to hit or stay (h/s)? s

Dealer's hole card was Five. Dealer's current total is 15.
Dealer pulled Four from the deck. Dealer's current total is 19.

You lost. (Your total was 19. Dealer's total was 19.)

Would You like to play again (y/n)? |
```

c) << For lesson 4 (not 3) >>

Did you consider layering? This assignment doesn't necessarily have a data layer, but we do have an UI layer (console read and write). If you didn't do so during the design, now refactor the console calls so they instead are called through methods in a separate UI class (please ask if you want help understanding how to approach/think about this).

d) After you have made your game, compare the Class Diagram from task a) with the actual code. How similar did the two end up being (feel free to make a new diagram that shows the changes if you want to)?

e) EXTRA/BONUS: Implement one or more of the following:

- Points (you can bet and then win or lose points based on this).
- Storage of your points to file (so it is kept from one session to the next).
- Multiplayer (the players all play against the dealer, not against each other).