Chapter 8 - Object Oriented Programming in C#

תרגיל 7:

A. Define a class that represents a playing card and a class that holds a deck. You should define for each class the data structure and the collection of the relevant methods. For example, for a deck you may define a constructor for full deck, empty deck, shuffling, taking out, putting in etc.

- Override in both classes the methods **ToString** and **Equals** that inherit from **Object**.

B. Write an application that demonstrates the Twenty-One card game between two players, where one player is the user and the other is the computer. You should use the classes defined in the previous section. The game's objective is to collect cards that their total sum is as closer as possible to 21, without busting (- exceeding 21). Card's values are calculated in the following way:

- The spot cards count 2 to 9.
- The cards: 10, Jack, Queen and King are count as 10 each.
- An Ace card may be either 1 or 11, as needed.
- The symbol and color of the card does not matter at all in the game.

<u>The game is conducted in the following way:</u>

- The cards are shuffled and two cards are dealt to each player.
- Each player in his/her turn takes hits (- asks for additional cards from the deck). Hits are done successively, so that the player will be able to decide whether he/she wants to take another hit or to stand (- stop asking for cards).
- After all the players finished making their decisions, all the cards are revealed and a winner is declared.
- The player whose cards total sum is the closest to 21 – but not over 21 – is the winner.

<u>Design a GUI that includes:</u>

- Revealing player's cards – you can display pictures or draw the card's details by simple graphic.
- A button for hitting (asking an additional card).
- A button for standing (stop asking for cards).
- Displaying game's results (including displaying the computer's cards and declaring the winner).
- After finishing the game there should be a possibility to start a new game.

תרגיל 8:

Write a simple application that simulates a hot-drinks vending machine.
The machine should offer different types of hot drinks, (e.g. tea, coffee, chocolate drink etc.), each with its own name and preparation method.
Clients choose their preferred drink and the machine should prepare it.
New drinks are added from time to time. Build a general mechanism to support these adaptations.
Think of resources relevant for all types of drinks (cups, sugar, hot water, etc.). Who's responsible for managing those?
What happens in extreme circumstances (e.g. no cups or sugar left?)
Create a UML diagram for your application, showing the different classes (methods and data members) and relationships between them.
Implement the project based on your design.

Bonus!
Add a payment mechanism to the machine:
Each drink has its price.
The machine collects should get the coins and only then does the drink preparation continue.
Sometimes change should be given back…