OOP

Black jack

OOP - 2021

General instructions:

Regarding your file:

1) Submit your code **only once**.
2) Submit **only running** code that you have tested before.
3) Follow the exact same number of classes required from you
4) Submit **only** the created 4 .java files (Card.java, Player.java, Game.java, BlackJack.java) compressed in a **single file.**
   //you can find them after you create your project as required at this path:
   […\NetBeans (or eclipse or visual studio code) \BlackJack\src\blackjack**]**

5) Your compressed file must be named like this: [DepartmentName_SectionNumber_Name_ID]
   For example:
   - General second academic year: [generalSecond_1_Ahmedxxxxxxxx_2020170xxxx]
   - General third academic year: [generalThird_1_Ahmedxxxxxxxx_2020170xxxx]
   - Software Dept: [SW_1_Ahmedxxxxxxxx_2020170xxxx]
   - Bioinformatics Dept: [BIO_1_Ahmedxxxxxxxx_2020170xxxx]
   - AI Dept: [AI_1_Ahmedxxxxxxxx_2020170xxxx]
   - Cyber Security Dept: [CS_1_Ahmedxxxxxxxx_2020170xxxx]
   - Multi-Media Dept: [MM_1_Ahmedxxxxxxxx_2020170xxxx]

6) Submit your code through this Google form link. **[Deadline Thursday 26/11/2021 at 12:00 AM]**

Please note that not following the required naming convention or submitting different file types other than those mentioned in point 4 will cause your submission to be ignored.

This assignment is intended for individual contribution. Sharing ideas or part of answers is considered plagiarism and will not be tolerated. All submissions will be checked for plagiarism automatically.

Blackjack

Your program will be a simplified version of the popular card game Blackjack (without bets). Even if you are familiar with the game, you should still read the instructions thoroughly.

At the beginning of the game, the 3 players and the dealer are dealt 2 cards each. We see both players' cards but see only 1 of the dealer's cards. The aim of the game is to have the sum of the card values (score) in your hand be 21 (**BLACKJACK**), or as close to 21 as possible. However, you lose if you go over 21.

The players go first in turns and have two choices: Hit or Stand. **HIT** means the player wants to pick another card. You make this move to increase your score. **STAND** means that the player is done picking new cards (i.e. Hitting) and their turn ends. You do this move when you are comfortable with your score (e.g. score=19 is very close to 21, picking another card could make you lose the game by going over 21 (Player **BUSTED**).

When the player's turn ends, the next player starts to play in turns. If the maximum score of all players is less than the value of the two cards of the dealer's then the dealer will win the game and doesn't need to hit more cards. Other than that, then the dealer starts hitting until he either scores 21, scores a score bigger than all of the other players or go over 21 (Dealer **BUSTED**).

Finally, if more than one player have the same high score then it's a tie situation (**PUSH**)**,** but if only a single player had the maximum score then that player **WIN**s the game

The strategy is to increase your score by hitting to get close to 21 (but not over), while also taking into consideration what the dealer's score could be.

The game can be summarized with the following steps:

1. Deck creation & initial card draw.
   a. Create deck
   b. Do the initial card draw for the player and the dealer (a total of 4 cards)
   c. Remove drawn cards from the deck

2. Player's turn (repeated 3 times for the 3 players)
   a. Hit or Stand
   b. Update the player's score.

3. Dealer's turn
   a. Hit until he either scores 21 (**BLACKJACK**), a score more than the maximum score of the 3 players or bust.

4. Make decisions (someone **WINS** or it's a **PUSH**) and end the game.

The game is explained more and tested at the recorded lab's video in this link: https://youtu.be/kNRdAsF1CQE

Create a project named with "**BlackJack**" with a **single package** and make sure that the package created is named "**blackjack**" (*the exact name is a must for the GUI*)

In the package create the following classes:

1) **Card** class (*the exact name is a must for the GUI*) with the following attributes and functions:

   a) Integer value called **suit**. *// a must with the same name for the GUI*
   b) Integer value called **rank.** *// a must with the same name for the GUI*
   c) Integer value called **value** *// a must with the same name for the GUI*

   Note that all the attributes should be unchanged once they get their values and should follow the rules of the encapsulation concept.

   d) Parameterized constructor that sets all the attributes
   e) **Copy constructor** *// a must for the GUI*
   f) **Getters for the attributes** *// a must for the GUI "autogenerated not written by hand"*

   More explanation for the class:

   i.e. suit = 0 for Clubs (♣), suit = 1 for Diamonds (♦), suit = 2 for Hearts (♥), suit = 3 for Spades (♠).

   ---

   rank = 0 for Ace card
   rank = 1 for 2 card
   rank = 2 for 3 card
   rank = 3 for 4 card
   rank = 4 for 5 card
   rank = 5 for 6 card
   rank = 6 for 7 card
   rank = 7 for 8 card
   rank = 8 for 9 card
   rank = 9 for 10 card
   rank = 10 for Jack card
   rank = 11 for Queen card
   rank = 12 for King card

   ---

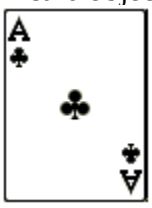
   value = 1 for Ace card
   value = 2 for 2 card
   value = 3 for 3 card
   value = 4 for 4 card
   value = 5 for 5 card
   value = 6 for 6 card
   value = 7 for 7 card
   value = 8 for 8 card
   value = 9 for 9 card
   value = 10 for 10 card
   value = 10 for Jack card
   value = 10 for Queen card
   value = 10 for King card

For example:

A card object with suit =1, rank =10 and value = 10 is:

A card object with suit =2, rank =5 and value = 6 is:

A card object with suit = 0, rank = 0 and value = 1 is:



2) **Player** class with the following attributes and functions:
   a) Name
   b) Score (the values of all the cards in the player's hand at the beginning and after each hit)
   c) Array of 11 Card objects
   d) A boolean value that indicates whether he got a blackjack or not. //optional if you want
   e) A boolean value indicates whether he already lost or not. //optional if you want

Note that all the attributes should follow the rules of the encapsulation concept.

3) **Game** class with the following attributes and functions:
   a) Array of 4 players *(the dealer is the last player object in the array at index [3])*
   b) Array of cards for the card deck (52 cards)
   c) A variable that keeps track of the existing VALID high score of all players *(<= 21)*
   d) A function that generates the card deck array (52 cards with different suits, ranks and values as explained previously)
   e) A function that draws a card randomly from the card deck array by following these steps:
      1) Generate a random index number using Random class that indicates the index of the card to be drawn from the card deck.

```
import java.util.Random; // Similar to what we do with the Scanner
...
...
Random rand = new Random(); // Again, very similar to Scanner
int randomChoice = rand.nextInt(5); // get a random number in range [0,5)
```

Note that rand.nextInt(5) will return a number between ***[0, 5[***

2) Saves that card object in a new object to be returned
3) Make that object equals null in the card deck array
4) Return the new card object created that holds the drawn card from the deck

Make sure that the drawn card from the deck is not equal to null before saving and returning it.

f) A function that sets the information of the players (take names from the user and draw 2 random cards for each player) at the beginning of the game.
g) A function to update the game maximum score of all players after any player draw a card to his hand.

4) **BlackJack** class that contains the main function and a game object declared outside the main function. The main function should follow steps:
   a) Generates the card deck and then sets the information of the players.
   b) Starts the game with the first player whether to hit or stand until he either gets a blackjack, busts, or stands and the same for the second and third player.
   c) The dealer's turn starts if his score doesn't already exceed the maximum game score, and he keeps hitting until he either gets a blackjack, a score more than the maximum score of the 3 players or bust.
   d) Decides whether a certain player won the game or it's a **PUSH**.

   Try to divide your code among different functions not writing all of it in the main function.

Do the following changes to use the GUI

- Put the downloaded .java files (4 .java files) in the same directory as your Java file at […\NetBeans (or eclipse or visual studio code) \BlackJack\src\blackjack]

- Put the downloaded image (4 images) in the same directory as your project. […\NetBeans (or eclipse or visual studio code) \BlackJack]

- Add the following line as the very first line of your main function.
  GUI gui = new GUI();

- After the first time you draw cards for the players and the dealer, put the following line after modifying it:
  gui.runGUI(arg1, arg2, arg3, arg4, arg5);

  Replace the arguments with following:
  - **arg1** with the card deck array generated in game class.
  - **arg2** with the array of cards of the first player (ex: game_object.player[0].cards)
  - **arg3** with the array of cards of the second player
  - **arg4** with the array of cards of the third player
  - **arg5** with the array of cards of the fourth player

This should open an interface similar to the picture below:

○ Then, each time you add a card to the player add this line:
gui.updatePlayerHand(the card object added to the player, player index in the array currently playing [0,1,2]);

○ while for the dealer add this line:
gui.updateDealerHand(the card object added to the dealer, the array of card deck after removing the drawn cards and reinitializing them with null);

This should open an interface similar to the picture below: