CS107L Assignment: Blackjack

Much of this assignment was designed and written by Julie Zelenski and Nick Parlante.

You're tired of hanging out in Terman and have decided to seek your fortune as a card shark and reap wealth and glory in Reno. The only problem is that you don't really know how to play Blackjack and need to get some airtime in before you run off and squander your savings. You also need to test out various strategies to find out what might be a good one to put into practice.

To this end, you have decided to write a Blackjack game in your new favorite language C++. This system will tirelessly (and cheaply) allow you to hone your game until you are ready to face the real thing.

Due: Sunday, December 16th at 11:59 p.m.

The Rules

Blackjack is played with an ordinary deck of 52 cards. Each card has a rank (Ace through King, or equivalently 1 to 13) and a suit (Hearts, Diamonds, Clubs, Spades). In this game, human players and various other computer players are pitted against a computer dealer. (The players don't play against each other; each player is simultaneously playing a one-on-one game against the dealer.) A round of Blackjack is played according to these rules:

1) The dealer gives each player two cards: one face down and one face up. Players can look at their own face-down card, but not other player's.
2) Each player's goal is to get more points than the dealer, but not to exceed 21 points. A player's points are determined by summing the contributions of their cards:
   - Face cards (rank = 11, 12, and 13) are worth 10 points.
   - An ace (rank = 1) is worth either 1 point or 11 points.
   - All other cards are worth their face value.
3) After dealing out the starting cards, each player gets a turn to accumulate a hand as close as they can to 21 without going over. A player can draw additional cards one by one from the deck until they no longer want any more. If at any point the player's hand exceeds 21, they are "busted" and lose.
4) After all of the players have had their turn and drawn as many cards as they want, the dealer draws his cards.
5) After all cards are drawn, the face-down cards are revealed and the dealer announces the winners. All ties go to the dealer.

Betting

Of course it wouldn't be interesting without a little money riding on the outcome. At the beginning of a round, before any cards are dealt, each player is asked how much they would like to bet on this round. If they win (i.e. beat the dealer), the player will double their money, if they lose, they lost the entire bet. The minimum bet is $1 at our table. Our casino is very generous with credit, so players can bet more money than they have and go way down deep into the hole if they like.

```
Okay, time for betting!
-----------------------
Bob, how much would you like to bet? 15
Bob bets $15
Meek bets $2
Random bets $47

The initial starting cards are:
--------------------------------
Bob's current hand: [??][10D]
Meek's current hand: [??][9H]
Random's current hand: [??][5D]
Dealer's current hand: [??][3C]

Bob's turn:
-------------------
Bob's current hand: [9C][10D] (19 points)
Would you like to draw another card? (Y or N): N
Bob chooses to stay

Meek's turn:
-------------------
Meek's current hand: [6H][9H] (17 points)
Meek chooses to stay

Random's turn:
-------------------
Random's current hand: [8H][5D] (13 points)
Random chooses to draw
Random's current hand: [8H][5D][H4] (17 points)
Random chooses to draw
Random busted at 27!

Dealer's turn:
-------------------
Dealer's current hand: [3S][3C] (6 points)
Dealer chooses to draw
Dealer's current hand: [3S][3C][4C] (10 points)
Dealer chooses to draw
Dealer's current hand: [3S][3C][4C][8D] (18 points)
Dealer chooses to stay
```

```
Let's see how it turned out:
----------------------------
Yowzah! Bob wins $15
Ouch! Meek loses $2
Ouch! Random loses $47

The standings so far:
--------------------
Bob     $115
Meek    $98
Random  $53
Dealer  $10034

Another round? (Y or N):
```

Players

Each player has a name, a starting bankroll, and some strategy (how they decide how much to bet and how to determine when to draw and when to stay). As a player wins and loses money, their bankroll grows and shrinks. Of course, the end goal is to end up with lots of cash. By trying out players with different strategies, you hope to learn which strategies pay off in the long run, thus gathering information for your upcoming trip to Reno.

Here are the types of Blackjack players we have:

Dealer	There is always exactly one dealer player in a game. The dealer always plays by the house rules, which mandates that they draw while their hand is worth 16 or fewer points, and stop when they reach 17 or more. The dealer's "bankroll" is actually the house money. The dealer never bets any of the house money, but it pays out the house money to any player which beats the dealer and keeps the money bet by any player that loses.
Human	A human player is one that makes choices by asking the user questions and getting answers using the standard I/O functions (you should use C++ I/O). For example, the name and starting bankroll of a human player are obtained by asking the user to type them in. At the beginning of the round, the user is prompted for the human's bet. On each turn of a human player the user is asked whether the player chooses to draw or stay. There can be zero, one, or many human players in a game.
Meek	The meek computer player is a rather meek soul. He usually bets just $2 every round. However, he goes a little crazy when he starts winning. Each time he wins 3 rounds in a row, he doubles his current betting amount. So after a string of 3 wins, he would start betting $4 a round. After 3 more wins, he would start betting $8. However, the first time he busts (i.e. his card total goes over 21), he goes back to his $2 bet. His card-choosing strategy is a little bizarre, he draws a card whenever he has an even

number of points, he stays whenever he has an odd amount. However if he has the seven of clubs in his hand, no matter what his total, he draws.

Random The random computer player is an unpredictable fellow. He randomly bets some amount between $1 and half of his bankroll each round. When it comes time for choosing whether to draw or stay, he also makes random decisions. If he has 9 or less points, he always draws. If he has 10-12 points, he draws with random probability 80%, if he has 13-15 points he draws 70% of the time, if he has 16-18 points, he draws with 50% of the time, and if he has 19 points or over, he never draws.

The different players have a lot of behavior in common— the challenge of the assignment is structuring your player class hierarchy so as to avoid duplicate code. You should also be able to add other types of computer players without a lot of modification.

The Game

There can be any number of players in a game. Before starting to play the game, you should ask the user how many players to create and of which kind they are, also each player gets a name and an initial starting bankroll. A dealer must also be created and he starts with the house's money account that always has an initial balance of 10,000.

Once all the players have been set up, your program should go into a loop playing rounds of Blackjack. At the end of each round, you should report the standings and ask the user if they would like to play another round (keeping the same players). You continue playing rounds until the user indicates he no longer wants to continue.

Relationships between objects

You've got card objects, deck objects, player objects, game objects, etc. all roaming about in this "object soup" and you will need to establish lines of communications that allow them to send messages to one another. This sort of situation comes up all the time in OOP— you have several cooperating objects, and they need pointers to each other so they can exchanges messages. If an object only needs to refer to an object for a short time, such as just within a particular method, you might consider passing that object as an argument to that method. If an object needs to message to another object repeatedly, you will probably want to store a pointer to this object as a data member of the object that needs to message it.

Even thought the **`Dealer`** would normally run the game, own the deck, dispense cards, etc., you might find it cleaner to have a **`Game`** object which handles those functions and restrict the **`Dealer`** object to just handling the "player" aspects of the **`Dealer`**, (managing a hand of cards and deciding whether to draw or not). Dividing up the tasks into two separate objects will simplify your design and helps the **`Dealer`** fit more naturally into the **`Player`** design structure. The **`Game`** object will take care of tracking the list of players, deciding whose turn it is, controlling the deck, and handling other game-related functionality.

Since you may have many players in a game and you don't want to run out of cards or have players count cards, your deck should be a "Vegas" deck of cards which is a bunch of normal-sized decks all mixed together. Create your deck with a 52-card pack for every 3 players (e.g. 9 players means a deck of 156 cards). In order to make sure you don't run out of cards in the middle of a round, you should check before the round begins. You can figure that you want enough remaining cards in the deck to cover at least 5 cards per player before you begin a round. If you don't have enough, shuffle the deck and start from the top.

Getting Started

I've provided you with interfaces and implementations for the **`Card`** and the **`Deck`** classes, and we'll be reusing the **`Random`** package from Assignment 2 to get random bets for your random player. As you are programming, you are encouraged to augment any or all of these classes if it makes sense to do so. I've also provided you with an incomplete **`Player`** class specification that you are required to improve upon and extend to build the dealer and non-dealer classes. You'll certainly want to have a class for each type of player, but you might consider the design of intermediate classes to unify code common to two or more players but not common to all of them. As you create your own classes beyond those I give you, you'll need to update the **`Makefile`** in the obvious manner so that **`make`** knows what C++ files to compile and link together.

Programming Tips

1) Keep dynamic memory allocation to a minimum. The use of direct objects in preference to dynamically allocated ones reduces the likelihood that you'll orphan memory. If you succeed removing all dynamic memory allocation from a class definition, you generally free yourself from the need to supply copy constructors, assignment operators, and destructors, because the compiler-generated ones do just fine as is.
2) Use built-in data structures and STL classes (**`string`** and **`vector`** come to mind) in preference to your own. Doing so saves you time and presses the responsibility of issues pertaining to item 1) down into the helper classes.
3) Don't sweat the details. I'm more interested in your ability to design small, easily maintained and intelligently written classes than I am in the details of Blackjack and keeping score.
4) Keep the number of non-OOP functions to a minimum. Prefer classes and method to standalone, yet stateless functions. Package related methods into their own class, and think about whether a method your writing belongs in the **`Player`** class or an other one that you haven't thought of yet.