A Blackjack Game Program

Introduction

New Java Skills: Implementing a Game Control Loop and gaining more practice with classes and methods. Before you begin, be sure that you have mastered the practice problems (Basic Looping and Nested Loop)

Assignment Goals: The goal of this assignment is to write a program which plays a simplified version of Blackjack. Blackjack is a two player game between you and the dealer. The computer will be the dealer. You will need to implement several types of loops:

1. A loop to test the correctness of your Card class.
2. A loop to control the user input (see problem 9 in the Basic Looping practice problems).
3. A Game Control Loop which controls the play of the game.

Game Rules (see https://www.pagat.com/banking/blackjack.html#objective)

- *Goal:* The aim of the game is to accumulate a higher point total than the dealer, but without going over 21. You compute your score by adding the values of your individual cards.
- *Card Values:* Cards have suit (spades, hearts, diamonds, clubs) and rank (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K). The cards 2 through 10 have a value equal to their rank. J, Q, and K each have a value of 10 points, and the Ace has a value of either 1 or 11 points (player's choice).
- *Play:* At each round of the blackjack game, the players (you and the computer) receive two cards each. The score is computed to determine who wins.
- *Simplification:* In this assignment we will
    - not allow for additional cards to be dealt (hits).
    - not worry about dealing duplicate cards, i.e. in your game, it will be possible to draw two aces of hearts. You can fix all of this later if you want, once we cover arrays.
    - Assume Aces always have a value of 11.

  If you finish the simplified game early, consider adding more complexity such as allowing the ace to be 1 or 11, or allowing hits.

Software development is always done in stages. Thus, we will use the concept of ***stepwise refinement*** to build our code up a bit at a time, testing carefully at each stage along the way.

Code Structure

We keep the class structure very simple. There will be required 3 classes:

- `BlackJackProject.java`: a class that contains the main method
- `BlackJack`: a class that contains the game code
- `Card`: a helper class that represents a single card
- (Optional) `Player`: stores information about you, the player, such as name and score.

The benefit of object oriented programming is that you can break your program into small bite sized pieces (i.e. classes) which are easy to code. These pieces can be independently coded and tested before being incorporated into the larger program. In this lab, we begin by writing a class that contains all of the functionality of a single card. Once the Card class is written and tested, we need no longer need to think about how a Card is implemented, rather we just make use of the Card class in the game code. This will greatly simplify the implementation of the game code.

Part 1: The Card Class

In this part of the lab you will write and test a Card class. It will give you practice writing classes and using if-statements. You will also use a loop to test your code.

Begin by creating a Netbeans project (e.g. called BlackJackProject) containing a main method. Add a second class called Card. In the main method (in the BlackJackProject class) you can test your Card class by creating a card and calling the various methods in that class

Recall that a class is a set of data (member variables or fields) and actions (methods). Here, we need to store sufficient data to completely identify the card. Since there are 52 possible cards, all we really need to store is a single index (0 to 51) in order to know which card we have. Note, we assume a specific ordering of the cards where cards are ordered by suit (spades, hearts, diamonds, clubs) and within suit, by rank (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K). With this assumed ordering, the Ace of Spades will have index 0, the 2 of Spades will have index 1, etc. Can you figure out the index of the Ace of hearts? King of diamonds? In any case, ***we need exactly one member variable (e.g. called index) of type integer whose value will be anywhere from 0 to 51.*** *You do not want to store the suit or the rank in a member variable* because they are redundant, i.e. they can be trivially computed from the index. If you were to store all three variables (index, suit, rank) you would have to be very careful about maintaining consistency, i.e if you were to change the index, you would need to immediately update the suit and rank otherwise the suit and rank would not represent the same card as the index. This is a potential source of error.

Now, what do you want your class to be able do? This will determine ***the methods***:

- **Constructor**: your constructor should randomly select and set your card's index.
- **Constructor**: your constructor should allow the user to pass in (as a parameter) the index of your card.
- **getRank**: a method that returns an integer (0 to 12) presenting the card's rank
- **getSuit**: a method that returns an integer (0 to 3) presenting the card's suit
- **getSuitName**: a methods that returns a string with name of the card's suit
- **getRankName**: a method that returns a string with name of the card's rank
- **getPoints**: a method that returns worth of the card in the game of BlackJack. For now, assume Ace's are always worth 11.
- **toString**: returns the full name of the card (e.g. 2 of Spades, or Ace of Hearts)

Note, if you want to print out the suit symbol rather than the name, it may work to use Unicode string symbols for card suits, e.g. `"\u2660"` is Unicode for a spade. Do a search to find the codes for the other suits.

That's it! Once you implement the above, you can do unit testing in main to test your class, e.g.

```
Card myCard = new Card();
System.out.println("The card you picked is " + myCard);
System.out.println("The card rank is " + myCard.getRank());
```

Unit Testing Every Card using a Loop

To make sure that every card is correct, you need to create an object for every card. A loop is the best way to do this**. Add a for-loop to main() which loops over the indices 0 to 51.** Inside the loop, create a card object with the current loop index, and print out the name of the card. Since you are looping in the order of 0 to 51, the cards should be in the order of suits (spades, hearts, diamonds, clubs) and within each suit, the order should be Ace, 2, 3,.. King. If this is not what you get, then you need to fix your code. Your output might look something like (shown here in multiple columns to save space)

```
index 0:  A♠
index 1:  1♠
index 2:  2♠
index 3:  3♠
index 4:  4♠
index 5:  5♠
index 6:  6♠
index 7:  7♠
index 8:  8♠
index 9:  9♠
index 10:  J♠
index 11:  Q♠
index 12:  K♠
index 13:  A♡
index 14:  1♡
index 15:  2♡
index 16:  3♡
index 17:  4♡
…
index 47:  8♣
index 48:  9♣
index 49:  J♣
index 50:  Q♣
index 51:  K♣
```

Part 2: The Game Algorithm

Once your Card class is written and tested, you are ready to move to the next step, namely to implement the game. Before you begin coding, be sure to:

A. ***Understand what you are doing:*** Read the above rules. Get a deck of cards and play the game with someone!
B. ***Think about Output:*** What do you want the output to look like? This will determine much of the workings of the code. For example, see the sample output below (modify it if you want, but it probably should have the same basic structure)
C. ***Pseudocode***: Write the *algorithm* in pseudocode, i.e. the sequence of steps required to solve the problem, taking into consideration the sample output. For example (you fill in what is missing):
    1. Print a welcome message to the game.

2. Prompt the user to find out if they want to play. Repeat if they enter an answer that is not allowed.
3. If they want to quit, end the program.
4. If they want to play the game:
    i. deal the cards for player1
    ii. etc // you fill in the rest …
5. repeat starting at step 2.

Implementing the Loops

When you are ready to begin coding, add a third class to your Netbeans project (e.g. called BlackJack) which will contain your game code (you now have 3 classes: BlackJackProject, Card, BlackJack). The loop that plays the game, can be in a method called play() in the BlackJack class. Thus, **in the <u>main</u> method (in the BlackJackProject class) you can add code to test your program**, e.g.

```
BlackJack blackjack = new BlackJack();
blackjack.play();
```

Remember, you never want to write the code all at once. Instead, identify small tasks that can be independently implemented and thoroughly tested. Here, we will look at the tasks that require loops.

In the pseudocode, identify the input prompt loop and the game play loop.

A. ***Prompt() Method:*** Create a *private* method called `prompt()`in your BlackJack class that prompts the user for an answer (y/n) and returns the answer `y` or `n`. (This method will be called from the `play()` method). If the user doesn't enter an answer that begins with an upper or lower case y or n, then the user is repeatedly prompted until such an answer is given. The prompting should be done with a loop (see problem 9 in the Basic Looping practice problems). Depending on how you want the interface to take place, a **<u>do-while loop</u>** is probably the best type of loop to use (do you see why?). (*Note, do not use recursion! Use only a loop.*)

   Test the `prompt()` method by calling it from the `play()` method. The output should look something like the following

```
Do you want to play blackjack? (y/n) maybe
Please answer y or n.
Do you want to play blackjack? (y/n) y
```

(the prompt method will then return the answer **y** *to where the method was called, i.e. in* `play`*)*

```
Great, let's play the game.  // this gets printed in play
```

Or, alternatively, the output might look like

```
Do you want to play blackjack? (y/n) n
```

(the prompt method will then return the answer **n** *to where the method was called, i.e. in* `play`*)*

```
Game Over. Good bye.        // this gets printed in play
```

Make sure this works before continuing. *Test every possible sequence of things that could happen.*

B. ***Dealing and Printing a Card:*** This is easy since you already created a Card class!

```
Card myCard = new Card();  // creates a random new card
System.out.println("You have been dealt the card " + myCard);
```

C. ***Game Loop:*** Here, we won't implement the full game yet but rather we just want to get the control loop working. Use a **while-loop** (do you see why?). The program should take the answer from the prompt method that you wrote above and, if the user says yes, deal a card and print the result. Note**, the call to prompt() can be done right in the loop condition**:

```
while (prompt() == 'y')  {
            …
}
```

The output should look something like:

```
WELCOME TO THE BLACKJACK GAME
Do you want to play blackjack? (y/n) maybe
Please answer y or n.
Do you want to play blackjack? (y/n) yes
Great, let's play the game.
You have picked 8♥
Do you want to play blackjack? (y/n) y
Great, let's play the game.
You have picked 5♠
Do you want to play blackjack? (y/n) n
Game over. Good-bye.
```

Make sure this works before continuing. *Test every possible sequence of things that could happen.*

D. **Finishing the Game**: You are now ready to implement the full game. Look at the sample output below and figure out how to extend your code so as to implement the rest of the game.

E. ***Clean-up:*** Clean up your code so it is nicely formatted. Use blank lines to separate sections of your code. As you might have noticed, the code is getting rather long and unwieldy. It is generally a good idea to break your code into short methods. However, you have enough to deal with in this lab so break things into methods only if you want to.

Sample Output:

When you run your code, the output should look something like this:

```
WELCOME TO THE BLACKJACK GAME
Do you want to play blackjack? (y/n) maybe
Please answer y or n.
Do you want to play blackjack? (y/n) y
Great, let's play the game.

Player:  A♣  8♠     Score is 19
Dealer:  8♠  4♥     Score is 12

Congratulations!  You win.

Do you want to play blackjack? (y/n) y
Great, let's play the game.

Player:  K♣  2♦     Score is 12
Dealer:  6♦  A♦     Score is 17

Sorry. The dealer wins.

Do you want to play blackjack? (y/n) n

Game over. Good-bye.
```

If you finish this lab early, challenge yourself by adding more complex features, for example,

- Enable the ace to be worth 11 or 1
- Enable the player or dealer to ask for more cards (hits).
- Add a Player class which can be used to create Player objects for both the computer and the human player. It will keep track of the total accumulated score if the player plays more than one round. At each round, the program will print out both the score for the round and the accumulated score.
- (Hard) Rewrite the code to have a GUI interface. Note this takes quite a bit of restructuring the code.
- Once we learn about arrays, you will be able to create a Deck class which stores an entire deck of cards and can deal and shuffle. The Deck class can prevent the same card being dealt more than once.