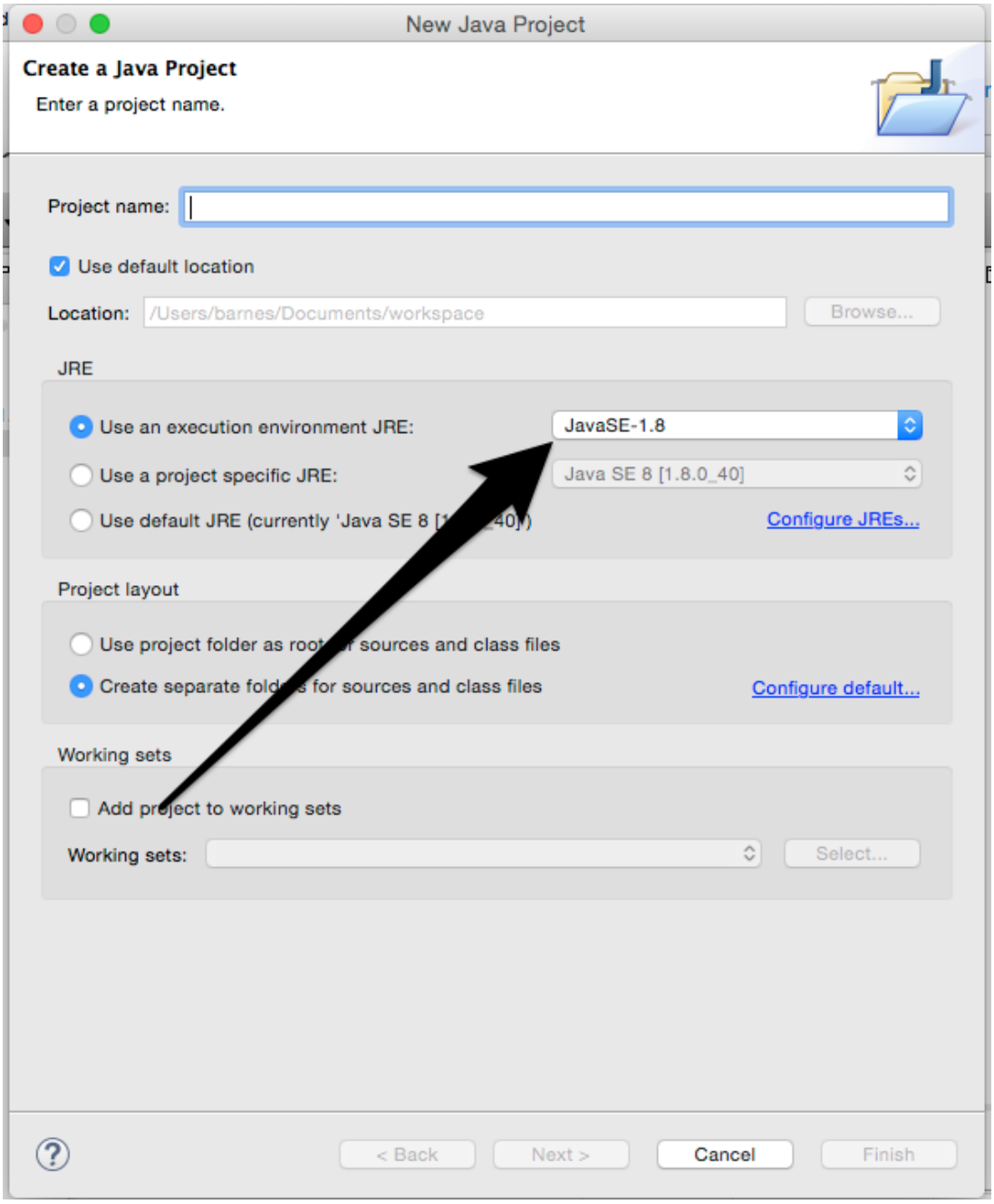
Project 5: Black Jack Game

You will need JDK 1.8 (Java 8) for this project!!!

When you create your BlackJack Project in Eclipse, you should verify that you have JDK 1.8 (Java 8).



Introduction

In this project, you will implement the classes, **Card.java**, **Hand.java**, **Deck.java**, **Player.java**, **Dealer.java** and **BlackJack.java** that are part of a Java application implementing a simple blackjack game with one dealer and one player. The goal of this project is to gain further practice implementing classes and methods and working with objects.

Several classes are required for this project:

- BlackJack.java
- BlackJackApp.java
- BlackJackTester.java
- Card.java
- Dealer.java
- Deck.java
- Hand.java
- Player.java

The skeletons of the classes Card, Hand, Deck, Player, Dealer, and BlackJack are provided on the labs and projects webpage. Modify these classes according to the specifications below. You will need to download the **classic-cards** folder and add it to your Eclipse project.

Class Card

A **Card** object represents a regular playing card with both a type like Ace, and a Suit like Spades. A Card object cannot be mutated one constructed.

A **Card** object is characterized by the following instance variables:

- **suit**: an enum that represents suit of the card
- **type**: an enum that represents the type of the card

Implement all of the following instance methods in the class **Card**:

1. **public Card(Type type, Suit suit)**: This constructor sets initial values of the instance variables**.**

2. **public boolean equals(Card c)**: Checks if the two cards are equal. Two cards are equal if they have the same type and suit.

3. **public String toString()**: Returns a string representation of the current card (ex. "ACE OF SPADES")
   Hint: Print the enumeration values themselves to make this easier.
   (Note the "S" at the end of the suit. We don't print ACE OF SPADE – We concatenate an "S" to the end. Also note the numbers that are printed.)

The following methods are already completed for you:

1. **public Suit getSuit()**: Returns the suit of the card (ex. SPADES)

2. **public Type getType()**: Returns the type of the card (ex. ACE)

Once you have finished implementing the above methods in the Card class, run BlackJackTester.java to test your class Card. If a method in your Card class fails a test in BlackJackTester, then you'll need to modify that method in a manner that allows it to pass the test. This means you may need to go back and handle various states not aforementioned. You should add some of your own tests to BlackJackTester to ensure all methods in the Card class are working correctly.

Class Hand

The class **Hand** represents the cards in a player or dealers hands during a game of Black Jack *or potentially other card games*. Remember – when we design classes such as Card or Hand, we design them in a general way so that we could reuse this code later when we create a different game.

A **Hand** object is characterized by the following instance variables:

- **cards**: a Card [] that represents the cards in the hands

Implement all of the following instance methods in the class **Hand**:

1. **public Hand()**: This constructor is already done for you. It sets the instance variable **cards** to an array of size 0. You can think of this as an empty hand.

2. **public void addCard(Card c):** This method adds a card to the hand. This involves creating a new array with room for the card (the current length of the cards array +1). You should copy over all of the old elements to the new array, and finally add the new card at the end. Note: Make sure you are saving the deck in the "cards" instance variable.

3. **public int size():** Returns the number of cards in the hand.

4. **public int getCards():** Returns an array of cards contained in the hands. Ensure this method does not return a reference to the instance variable **cards**. Instead it should return a deep copy.

5. **public Card [] emptyHand():** Empties the hand (sets the size of the cards array back to 0 – like we did in the constructor) and returns an array of the discarded cards from that hand.

6. **public String toString()**: Return a string representation of the hand. This includes the order at which the cards were drawn. For example, if the Ace of Spades and then the Queen of Hearts were drawn:
   1. ACE OF SPADES
   2. QUEEN OF HEARTS

   (Note: the numbers are printed.)
   If the hand is empty, this method should return "Empty hand".

Once you have finished implementing the above methods in the Hand class, run BlackJackTester.java to test your class Hand. If a method in your Card class fails a test in BlackJackTester, then you'll need to modify that method in a manner that allows it to pass the test. This means you may need to go back and handle various states not aforementioned. You should add some of your own tests to BlackJackTester to ensure all methods in the Card class are working correctly.

Class Deck

The class **Deck** represents the deck of cards that the Dealer and Player's Hands will come from.

A **Deck** object is characterized by the following instance variables:

- **cards**: an Card [] that represents the cards stored in the Deck. Its size will always be 52 because there are 52 cards in a standard card deck. Conceptually, the 0^{th} card is at the bottom of the deck, and the 51^{st} card is at the top, and will be the first to be drawn.

- **numCardInDeck**: an int that represents the number of cards actually in the physical deck. If drawn from, this is decremented, and if cards are added, it is subsequently incremented.

Implement all of the following instance methods in the class **Deck**:

1. **public Deck()**: The constructor is **completed for you.** It instantiates the **cards** instance variable to an array of size 52 and assigns a unique card to each slot.

2. **public Card draw()**: Returns the card at the top of the deck (0 is at the bottom of the deck), and decrements the **numCardsInDeck** instance variable. If the deck is empty (check the numCardsInDeck variable), return null and print an error message.

3. **private void swap(int a, int b)**: Swaps the cards at the **a**-th and **b**-th indices in the **cards** instance variable. This method can be used as a helper method later when you shuffle(). It will never be called outside of this class, so it is declared to be private.

4. **private void shiftRightOne()**: Shifts the elements in the instance variable **cards** one to the right. The card at index 0 will now be at index 1, the index at 1 would now be at 2, etc. Print an error and don't do the shift if the deck is full (**numCardsInDeck** == 52). This will be used as a helper method to **addToBottom** which explains why it is private – this method will never be called outside of this class.
   HINT: Start at the top of the deck (i.e., numCardsInDeck) & moving towards the bottom(0).
   Optional: You can use the swap method you have already created to help you with this process.

5. **public void addToBottom(Card c)**: Adds the card c to the bottom of the deck if the deck is not full. Do this by shifting the deck right and then assigning the 0^{th} index of the **cards** instance variable to **c**. Make sure you increment the **numCardsInDeck**. If the deck is full, print an error message & do nothing.

6. **public void shuffle(int n)**: Do **n** random swaps on all the cards contained in the deck (that is their index is less than **numCardsInDeck**). This does not call the other shuffle method();

7. **public void shuffle()**: Do 100 random swaps on all the cards contained in the deck (that is their index is less than **numCardsInDeck**).

Once you have finished implementing the above methods in the Deck class, run BlackJackTester.java to test your class Deck. If a method in your Deck class fails a test in BlackJackTester, then you'll need to modify that method in a manner that allows it to pass the test. This means you may need to go back and handle various states not aforementioned. You should add some of your own tests to BlackJackTester to ensure all methods in the Deck class are working correctly.

Class BlackJack

The class **BlackJack** represents the actual game of blackjack being played. You will not be implementing a full simulation however, as that is done by **BlackJackApp**.

A **BlackJack** object is characterized by the following instance variables:

- **deck**: The **Deck** being used to play.
- **dealer**: The **Dealer** of the game.
- **player**: the **Player** of the game. Essentially the human player

Implement all of the following instance methods in the class **BlackJack**:

1. **public BlackJack()**: Constructs a new game of BlackJack. Creates the player, the dealer, and deck objects by calling their default constructors. The player and the dealer both a Hand instance variable. You should use their getHand() methods which return a reference to their Hand. Call the shuffle method to shuffle the deck. Next we will want to give the dealer and the player 2 cards each by drawing the cards from the deck and adding them to the players/dealers hand.
   HINT: 1) Draw 2 cards from the deck, put them in the players hand 2) Draw 2 cards from the deck, put them in the dealer's hand.

2. **public void restart()**: Resets the game in three steps:
   1. The player and the dealer return their cards to the deck. This should **remove** the cards from the player's hand.
   2. The deck is shuffled
   3. The player and dealer both receive two cards from the top of the deck

Implement all of the following static methods of the **BlackJack** class:

1. **public static int getValueOfCard(Card c)**: Returns the value of a card in a standard game of Black Jack. For example the Two of Clubs would return a 2. Jacks, Queens, and Kings all have the value of 10. Aces are a special case as they can have two different values, 1 or 11, but for this method they will return a value of 1.

2. **public static int getValueOfHand(Hand h)**: Returns the maximum value of **h** that does not result in a bust (if avoidable). A bust is a value above the desired maximum of 21. This constraint is what determines if the Ace(s) in the hand are evaluated as 11 or 1.
   HINT: Compute the values of the aces **after** you have computed the total value of the other cards.

Once you have finished implementing the BlackJack class, run BlackJackTester.java. If a method fails a test in BlackJackTester, then you'll need to modify that method in a manner that allows it to pass the test. This means you may need to go back and handle various states not aforementioned. You should add some of your own tests to BlackJackTester to ensure all methods in your BlackJack class are working correctly.

Class Dealer

The class **Dealer** represents the dealer in the Black Jack game. He draws (hits) until he reaches 17 points or 5 cards.

A **Dealer** object is characterized by the following instance variables:

- **hand**: The **Hand** of cards the dealer has.

Implement all of the following instance methods in the class **Dealer**:

1. **public Dealer()**: The constructor instantiates the **hand** instance variable to an empty hand.

2. **public Card playTurn(Deck d)**: Dealer draws a card & adds the card to his hand if his hand is worth less than 17 points and contains less than 5 cards. This method should return the card if one was drawn or null otherwise. (The card returned here is used in the BlackJackApp.java file to display the correct card graphic in the game).
   HINT: Are there methods that we have previously created in other classes to make this task easier?

3. **public boolean busted()**: Returns true if the dealer has busted (value of his hand is **greater than** 21).

4. **public boolean isDone()**: Returns true if the dealer will not draw a card next turn. (the dealer draws if his hand is worth less than 17 points and contains less than 5 cards.)

5. **public boolean returnsCardToDeck(Deck d)**: This method empties the dealer's hand, and returns the discarded cards to the bottom of the deck.

Class Player

The class **Player** class represents the human player in the Black Jack game.

A **Player** object is characterized by the following instance variables:

- **hand**: The **Hand** of cards the player has.

Implement all of the following instance methods in the class **Player**:

1. **public Player()**: The constructor instantiates the **hand** instance variable to an empty hand.

2. **public boolean busted()**: Returns true if the player has busted (value of his hand is **greater than** 21).

3. **public void returnsCardToDeck(Deck d)**: This method empties the player's hand, and returns the discarded cards to the bottom of the deck.

Finally, uncomment the line in BlackJackTester.java that says "launch(args);" This will execute the BlackJack App. Play the game many times to make sure everything is working correctly, and have lots fun!

Additional Instructions

Your program must include a comment at the top of each file you submit. Copy and agree to the comments below, and fill in the file name, your name, the submission date, and the program's purpose for each file you submit.

```
/*
* [Class name here].java
* Author:  [Your name here]
* Submission Date:  [Submission date here]
*
* Purpose: A brief paragraph description of the
* program. What does it do?  How does it do it?
*
* Statement of Academic Honesty:
*
* The following code represents my own work. I have neither
* received nor given inappropriate assistance. I have not copied
* or modified code from any source other than the course webpage
* or the course textbook. I recognize that any unauthorized
* assistance or plagiarism will be handled in accordance with
* the University of Georgia's Academic Honesty Policy and the
* policies of this course. I recognize that my work is based
* on a programming project created by the Department of
* Computer Science at the University of Georgia. Any publishing
* of source code for this project is strictly prohibited without
* written consent from the Department of Computer Science.
*/
```

All instructions must be followed for full credit to be awarded.

Project Submission

After you have thoroughly tested your program with all of the tests provided plus your own tests, you should submit **BlackJack.java**, **Dealer.java**, **Card.java**, **Hand.java**, **Player.java, & Deck.java** to eLC for grading. You should not include any other files.

Project Grading

All projects are graded out of a possible 100 points. Programs can be submitted up to 48 hours late, but late programs will lose 25 points on the first day late and 50 points on the second day late. Programs not submitted within 48 hours after the deadline will receive a grade of zero. Programs that do not compile will also receive a grade of zero. You must make absolutely certain your program compiles before submitting. Also, you must thoroughly test your program by playing the game many times and by running the provided tester classes with your own tests added to it. You must make absolutely certain your project conforms to all specifications and instructions; otherwise, it may not compile or run correctly with our testing program(s), which may result in a failing grade on this project.

This project will be graded for both correctness and style:

Style [20pts]

- 10 points for including the class comment required for all projects in all of your submitted **.java** files

Correctness [80pts]

- 90 points for correct output on various test cases.