CISC 3120

Exercise 3

War and Peace

(Based on materials from Jason Hallstrom's CPSC 215: Software Development Foundations, http://people.cs.clemson.edu/~jasonoh/courses/cpsc_215_fall_2012)

Overview

In this lab:

- you will be introduced to Java `enum` types and gain further experience with Java arrays,
- you will gain experience developing Java classes,
- you will gain experience using Java objects to implement an object-oriented application, and
- you will gain experience using implementation inheritance in Java.

Requirements

You are required to implement a Java program that plays 52 hands of the classic simple card game *War*. In this two-player game, each player gets one deck of 52 playing cards. One turn of the game proceeds as follows: the players randomly select a card from their decks (usually by shuffling the deck before play begins) and turn it over. The player with the higher card showing wins the turn. Here, "higher" means that the first card has a higher value, or, in the case of a tie in value, has a higher suit alphabetically. (So "3 of Clubs" beats "2 of Clubs", and "5 of Spades" beats "5 of any-suit".) The winner of the game is the player who wins the most turns. (If you've played War before, you know that these rules are somewhat simplified.)

Your program must consist of three classes satisfying the following requirements:

- `Card`. This class will be used to model a single playing card. The class must define private fields to represent a card's **value** (e.g., 2, … 10, Jack, Queen, King, Ace) and **suit** (e.g., club, diamond, heart, spade). I strongly suggest you implement these using Java's `enum` types—see pp 671–673 in the book, or the brief tutorial at https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html. You might also want to consult the API documentation for `Enum`, which is the base class for all `enum` declarations. In particular, look at the documentation for `compareTo()`.

  In addition to any public constructors that you decide to include, the class must define two public methods: The first, `toString()`, will return a reasonably formatted `String` object corresponding to the card's value and suit (e.g., "Ace of Spades"). The second, `winner()`, will accept a `Card` object as argument, and return a boolean value indicating whether the first card (i.e., the target of the method call) is higher than the second card (i.e., the argument to the method call).

- `Deck`. This class will be used to model a standard deck of playing cards. The class must define private fields to represent 52 playing cards: 13 different values, 4 suits each. (Probably, you'll want to implement this as a simple array.) In addition to any public constructors that you

decide to include, the class must define one public method: The `draw()` method will return a random `Card` object from the deck. The card returned by the method will also be removed from the deck. (That is, a card returned by the `draw()` method must never be returned by any subsequent call to `draw()`.)

- `MainDriver`. This class will provide the `main()` entry point to your Java application. It will construct two `Decks` of `Cards` before entering the 52-hand loop. In each iteration of the loop, you will draw one `Card` from each `Deck`, display the `Cards` drawn, and declare a winner for that hand. Upon termination of the loop, you will display the total number of hands won by each player and declare a winner for the game.

Hints

Be sure to explore ways Eclipse can help you—the little 'c' in a green circle is a useful button, near the left edge of the menu.

Don't be afraid to declare “helper methods” in your classes (though be sure these are declared `private`).

Your code will need to select a random card. Here is a hint: The `java.util` package includes a variety of useful classes. Remember that the class `Random` is contained within his package; it provides a public default constructor (i.e., a public constructor that accepts no arguments). Once you’ve created a `new` instance of `Random`, you may invoke `nextDouble()` on the object to return a random `double`. Similarly, you may invoke `nextInt()` to return a random `int`. There may be other techniques available in the API, as well.

You can get skeleton code as an Eclipse project from

https://github.com/BC-CISC3120-F16/class11-code