Tutorial: Card Games

Programming II : Objects and Collections

Autumn Term 2017

Objective

There are many different card games that are played by cards (some or all) being dealt out and then each person takes turns doing the appropriate moves. This tutorial is about providing the types (objects and enums) needed to implement these sorts of games. The games will be for n players where one is a human player and the other $n-1$ players will be computer players.

What to implement

1. Start with implementing two enumeration types `Rank` and `Suit`. For `Rank` you will need to implement the ranks from Ace low (that is ACE, TWO, etc.). For each constant provide a parameter which is a string (the representation) that holds a letter or number (e.g. "A","10"). This representation should be `private` and `final` (and needs to be set in the constructor). Provide `toString` so that the rank can be printed. Also provide `equals` and `lessThan` to compare two ranks. `Suit` should be similar with the order of the suits being spades, hearts, clubs, and diamonds. You can use unicode symbols for the suits if you so wish. Make sure you write classes `TestRank` and `TestSuit`.

2. Implement is a class for holding cards. This is just a constructor (taking a `Rank` and a `Suit`) and a `toString` method. Implement `TestCard` (preferably before implementing `Card`.

3. Implement a class for holding a deck of cards. Implement it using the following field declarations:

```
private final Card[] cards;
//cards taken off the deck are not actually removed
//rather size points to the first index after the cards
//that are still in the deck
private int size;
private final Random random = new Random();
```

   `Random` is needed for shuffling cards. It needs to be imported at the top of your file with

```
import java.util.Random;
```

   Implement:

   - a constructor which returns a deck with ordered cards
   - a method for shuffling the deck, you will need to make a call such as:

```
int index = random.nextInt(i);
```

     to get an arbitrary position in the array to swap cards with.

- a method that returns the top (or next) card from the deck (removing it from the deck
- either a `toString` method or a `print` method

4. Now reimplement deck using a list (give the class a different name). It needs to provide identical functionality but its fields are:

```
private List<Card> cards = new ArrayList<Card>();
private final Random random = new Random();
```

   When you swap two elements in any collection you do it by making a call such as:

```
Collections.swap(cards, i, index);
```

   Make sure you import `java.util.Collections` at the top of your file.

5. Test your two classes for holding decks by comparing their behaviour. Calling the same methods from the different classes should produce identical results. Think about which implementation you prefer and why.

6. Write a class `Hand` for holding the cards an individual player will hold. You can use a list because you don't know how many or an array since you know there won't be more than 52 cards in a hand. Write the following instance methods:

   - `add` which takes a card and adds it to the hand,
   - `get` which a position and gives back to card (without removing it),
   - `remove` which take a position a removes that card from the hand,
   - `getCardCount`,
   - `sortBySuit`,
   - `sortByHand`.
   - Write a class `Player` which holds the name, score, and hand for a player.
   - Start writing a class `Game` for playing a game. As this is for a large number of games (with different rules) for now just write the constructor to create a game. This requires the players to be set up, the deck created and shuffled and the hands dealt. You will need to have as a parameters to the constructor the number of cards to deal and the number of players. You can assume there is always a human and there are a maximum of four computer players.
   - You can use these classes as a foundation for writing a card game. The easiest game that I can think of is to deal out the cards (either all or some, it doesn't matter) and then higher rank wins (whatever the suit). If two cards have the same rank then the suit order makes the difference. The winner is the player with the most round wins.