Assignment 1: The Deck Class

CS230
Fall 2000

Allen B. Downey
Computer Science Deparment

Due: Tuesday 19 September
The reading for this assignment is Chapters 3, 4 and 5.

The Deck class

On the class web page, at the following URL

```
http://rocky.wellesley.edu/cs230/code/hw01/Card.java
```

you will find a program named `Card.java` that creates a deck of playing cards and tests a few simple algorithms, like shuffling the deck, sorting the deck, and searching for a card in the deck (using a bisection search).

In this program, a deck is implemented as an array of `Cards`. For example, when we pass a "deck" as a parameter, the actual type of the parameter is `Card[]`.

In Chapter 5 of the textbook, I present an alternative representation for a deck, an object type named `Deck` that contains an array of cards as an instance variable.

You should read Chapter 5 and understand the difference between these two representations before proceeding.

1. Move `Card.java` into your development environment and compile and run it. It should print all the cards in the deck and then print a "trace" of two bisection searches, one successful, one not.

2. Add a second class definition, `Deck` in a file named `Deck.java`. Copy the instance variables and constructors from page 45, along with `printDeck` from page 46.

3. Create a third class definition, `Test` that will contain `main`.

4. Of the methods currently in the `Card` class, decide which ones would be more appropriate as members of the new `Deck` class. Move them there, and modify them so that they work again.

5. Look over the program and identify every place where an array of `Cards` is being used to represent a deck. Modify the program throughout so that it uses a `Deck` object instead.

   It is probably a good idea to make this transformation one method at a time, and test the program after each change. On the other hand, if you are confident you know what you are doing, you can make most of the changes with search-and-replace commands.

Merge

1. Using the pseudocode in Section 5.5, write the method called `merge`. Be sure to test it before trying to use it as part of a `mergeSort`.

2. Write the simple version of `mergeSort`, the one that divides the deck in half and then uses `sortDeck` to sort the two halves. Then it uses merge to create a new, fully-sorted deck.

   Remember that `sortDeck` is a modifier and `mergeSort` is a function, which means that they get invoked differently:

   ```
   sortDeck (deck);              // modifies existing deck
   deck = mergeSort (deck);      // replaces old deck with new
   ```

3. Write the fully recursive version of `mergeSort`.