COLBY COLLEGE
Computer Science Department

CS151 **Allen B. Downey**
Fall 1999

Assignment 12: Objects of Arrays

Due: Wednesday 8 December

Deck objects

This assignment builds on top of the Card class from Homework 11. If you are not satisfied with your solution to that assignment, feel free to use my version from the class web page.

`http://rocky.colby.edu/cs151/solutions/hw11/cs151.java`

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

In Chapter 12 of the excellent textbook, "How to think like a computer scientist," 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 12 and understand the difference between these two representations before proceeding.

1. Add a second class definition, `Deck`, to the program. Like the `Card` class, the new class cannot be a public class like `cs151`, which contains `main`.

2. Type in the constructors for the `Deck` class as shown in the book.

3. 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 make any changes necessary to get the program to compile and run again.

4. Look over the program and identify every place where an array of `Card`s is being used to represent a deck. Modify the program throughout so that it uses a `Deck` object instead. You can use `printDeck` in the text as an example.

   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 12.5, 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`.