

## Homework 2: The Deck Class

cs230  
Spring 2002

Allen B. Downey  
Computer Science Department

Due: February 19

This homework has two parts. The first part is mostly a syntax exercise in which you will rewrite a monolithic `Card` class as two classes, one for `Cards` and one for `Decks`. As on the previous assignment, all methods are class methods. In the second part you will write an implementation of mergesort, using a development plan for divide-conquer-and-glue programs.

The reading for this assignment is Chapters 10, 11 and 12.

### The Deck class

On the class web page, at the following URL

<http://rocky.wellesley.edu/cs230/code/hw02/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 12 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 this chapter and understand the difference between these two representations before proceeding.

1. Download `Card.java` into a directory named `hw02` 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. Write a class definition for a `Deck` object in a file named `Deck.java`. Copy the instance variables and constructors from pages 121 and 122, along with `printDeck` from page 122.
3. Inside the `Deck` definition, write another `main` method. Put some code in `main` to test what you have done so far and run it.

When you run `java`, you specify which class you want to use as the startup class, and thereby which version of `main` should run. So it’s ok to have different versions of `main` in different files.

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 (see the emacs documentation).

## 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 `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 use `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`.