INF100 H17 Assignment 4

Due on Friday, October 6, 2017

Albin Severinson, Dag Haugland

Abstract

This week's assignment will be dedicated to shuffling a deck of cards. Specifically, we will implement the so-called Knuth shuffle and use it to generate an array with elements *Hearts1*, ..., *Hearts13* (we only consider the Hearts) in a random order. This is how you shuffle a deck of cards at an online casino, for example. Interestingly, this is something that casinos sometimes get wrong! See here and here for examples on how shuffling a deck of cards can go wrong. We will write one or two methods in each of the problems and the solution to the final problem will depend on all methods written previously. It is therefore especially important to carefully verify that each method works as intended before moving on! We will only grade (pass/fail) the program for the final problem of the assignment. If you have questions specific to this assignment you should send them to me at `albin.severinson@uib.no`.

Submission

1. Add your program file to a `.zip` archive named `ex4_firstname_lastname.zip`. You do this by right clicking on the file ending with `.java` that contains your program code and choosing compress or add to archive (exactly what it is called varies by operating system and version). You should only add your final program, i.e., the one for problem 4. Remember that the file name must match the class name and that you therefore can not rename the `.java` file! Ask your group leader if you are having issues with this. Note that handing in the wrong file may cause you to fail the assignment!
2. If you complete the voluntary getting ahead part of the assignment, it should be handed in separately. If you complete this part you thus need to hand in two files (the `.java` file for problem 4 and the `.java` file for the getting ahead part). Put both files into the same `.zip` archive.
3. Log in to `https://mitt.uib.no` and press the box marked with "INF100" to enter the page for this course.
4. Press "Oppgåver" on the left. Next, select "Assignment 4".
5. Press "Lever oppgåve" and upload the `.zip` file you created in step 1.

General Guidelines

These are general guidelines that will (hopefully) help you maintain your sanity through your programming endeavours.

- Have your computer indent your code for you. You do this in DrJava by selecting all of your code, right clicking, and selecting `indent lines`.
- You should make a copy of all your code after completing each problem so that you can roll back to a previous version when you inadvertently break something. Even the developers at Google sometimes have to roll back.
- Set aside time to clean up your code. The number that gets bandied around on the internet is that the best programmers spend roughly half their time cleaning up and rewriting their code, i.e., you are halfway done when you get it working.
- You save time by making sure you properly understand your current program before moving on.
- You should probably be spending a significant part of your time on stackoverflow.
- Google is your friend when you want to figure out how a standard class or method works (`Scanner`, say). For example, searching for "java Scanner" gives you its Java docs page as the first result.
- When in doubt, look at the Google Java style guide to figure out how to write something.

Problem 1

Consider a deck of cards consisting of the 13 cards marked with hearts. Write a method that returns an array of length 13 with elements `[Hearts1, ..., Hearts13]` (aces have value 1) in order. You should use a `for` loop to assign values to the elements of the array. Specifically, the method should have the following signature:

```
public static String[] sortedDeck() {

}
```

Call this method from your `main` method and print the returned array.

Printing Arrays

If you print an array `String[] array = new String[13];` with `System.out.println(array);`, you get something like `[I@15db9742`. That is because Java defaults to printing the memory address of the array. Not what we wanted and not interesting to us humans! Instead, you need to import `java.util.Arrays` and print the array with `System.out.println(Arrays.toString(array));`. This will print `[null, null, null, null, null, null, null, null, null, null, null, null, null]`. It prints `null` because we have not yet assigned values to the elements of the array.

Example

```
[Hearts1, Hearts2, Hearts3, Hearts4, Hearts5, Hearts6, Hearts7, Hearts8,
Hearts9, Hearts10, Hearts11, Hearts12, Hearts13]
```

Problem 2

Write a method that swaps two elements of an array. It should have the following signature

```
public static void swap(String[] deck, int i, int j) {

}
```

Example

Swapping elements `i=1` and `j=2` of the array from the previous problem.

```
[Hearts1, Hearts3, Hearts2, Hearts4, Hearts5, Hearts6, Hearts7, Hearts8,
Hearts9, Hearts10, Hearts11, Hearts12, Hearts13]
```

Swapping elements `i=5` and `j=9`.

```
[Hearts1, Hearts2, Hearts3, Hearts4, Hearts5, Hearts10, Hearts7, Hearts8,
Hearts9, Hearts6, Hearts11, Hearts12, Hearts13]
```

Problem 3

For this problem we will write two methods. First, write a method that returns a random integer in $0, \ldots, \text{max}$.

```
public static int randInt(int max) {

}
```

Next, we will implement the actual shuffling. We first give the method signature and then explain how the shuffling should be implemented.

```
public static void shuffle(String[] deck) {

}
```

The shuffling should work as follows: For each `i=0, ..., deck.length-1`, swap the `i`-th element of the deck with a randomly selected element with index in the range `0, ..., i`. Note that you are always swapping the i-th element with an element whose index is at most i. For example, swapping elements `i=3` and `j=2` is fine. Swapping `i=7` and `j=7` is also fine. However, swapping elements `i=4` and `j=6` is forbidden. If you are curious, see this link for an explanation of why this is important: `https://blog.codinghorror.com/the-danger-of-naivete/`

You should call both `randInt` and `swap` from this method. Use this method to shuffle the array returned from your `sortedDeck` method and print the result. This is the Knuth shuffle and it guarantees that all possible orderings of the elements are equally likely. This is very important if you use it to shuffle a deck of cards at an online casino!

Example

```
[Hearts10, Hearts8, Hearts6, Hearts12, Hearts3, Hearts1, Hearts5, Hearts9,
Hearts4, Hearts11, Hearts2, Hearts13, Hearts7]
```

```
[Hearts4, Hearts9, Hearts11, Hearts13, Hearts12, Hearts2, Hearts6, Hearts3,
Hearts1, Hearts8, Hearts7, Hearts10, Hearts5]
```

```
[Hearts5, Hearts7, Hearts11, Hearts9, Hearts6, Hearts8, Hearts3, Hearts10,
Hearts13, Hearts12, Hearts4, Hearts2, Hearts1]
```

Problem 4

Finally, write a method that returns a shuffled deck. Call the `sortedDeck` and `shuffle` methods from this method. Print the resulting array from your main method. The output should be the same as for the previous problem.

```
public static String[] shuffledDeck() {

}
```

Getting Ahead

You now know more about shuffling digital decks of cards than some casino employees! However, we can make an interesting extension. Did you know that arrays can have more than one dimension? Here is a primer: `https://www.youtube.com/watch?v=ctab5xPv-Vk`. Now, write a new method that returns a two-dimensional 4 by 13 array containing all 52 cards in order. Specifically, `deck[0]` should be an array with all 13 hearts in order, `deck[1]` an array with all 13 spades in order and so on.

Next, write a new shuffle method that shuffles this two-dimensional array. It should shuffle between suits (clubs, diamonds, hearts, spades) so that there is an equal probability of assigning any card to any element. You will need to put some thought into how to shuffle this array. Remember that the internet people will take your money if they notice that some orderings are more common than others.

```
public static String[][] shuffledDeck2D() {

}
```

This part is meant as preparation for the coming course material and will not be graded. However, we will still look at it and comment it! Furthermore, you never have to catch up if you are always getting ahead. This part should be handed in separately. If you complete this part you thus need to hand in two files (the `.java` file for problem 4 and the `.java` file for this part). Put both files into the same `.zip` archive.