SP2 — Lab sheet 3

2014

Based upon exercises from Java for Everyone, 2e, Chapter 6.
Some of these you may have encountered during Software and Programming I, although the `ArrayList` questions are probably "new" to you.

---

1. Perform each of the following tasks:
   (a) Create an array `x` of doubles with an initialiser list that contains the following values: 8, 4, 5, 21, 7, 9, 18, 2, and 100.
   (b) Print the number of items in the array by printing the expression `x.length`.
   (c) Print the first array item, `x[0]`.
   (d) Print the last array item. Be careful to choose the right index.
   (e) Print the expression `x[x.length - 1]`.
   (f) Use a standard `for` loop to print all the values in the array without labels.
   (g) Use a standard `for` loop to print all the values in the array with labels to indicate what each element is.
   (h) Use a standard `for` loop to print all the values in the array in reverse order with labels to indicate what each element is.
   (i) Use an *enhanced* `for` loop to print all the values in the array without labels.

2. Write a method that is passed an array, `x`, of doubles and an integer rotation amount, `n`. The method creates a new array with the items of `x` moved forward by `n` positions. Elements that are rotated off the array will appear at the end. For example, suppose x contains the following items in sequence:

   ```
   1 2 3 4 5 6 7
   ```

   After rotating by 3, the elements in the new array will appear in this sequence:

   ```
   4 5 6 7 1 2 3
   ```

   Array `x` should be left unchanged by this method. Use the following code to help you get started. Be sure to test your program with different rotation amounts.

```
public class Arrays2 {
    public static void main(String[] args) {
        double[] x = {8, 4, 5, 21, 7, 9, 18, 2, 100};
        System.out.println("Before rotation:  ==============================");
        for (int i = 0; i < x.length; i++) {
            System.out.println("x[" + i + "]:  " + x[i]);
        }
        x = rotate(x, 3);
        System.out.println("After rotation:  ==============================");
        for (int i = 0; i < x.length; i++) {
            System.out.println("x[" + i + "]:  " + x[i]);
        }
    }

    // Your method goes here.
}
```

3. Create a class `CustomerLister` with a `main` method that instantiates an array of `String` objects called `customerName`. The array should have room for five `String` objects. Use an *initializer list* to put the following names into the array:

   ```
   Cathy
   Ben
   Jorge
   Wanda
   Freddie
   ```

   Print the array of names.

4. We can use the `Math.random` method to generate random integers. For example, `Math.random()` generates a random integer greater than or equal to `0` and less than `1`. The expression `Math.random() * 6` generates random numbers between 0 and 5, simulating the throw of a die. In this lab assignment, you will use an array to test whether the random generator is fair; that is, whether each possible value is generated approximately the same number of times.

   Your program should ask the user:

   - How many random numbers should be generated?
   - What is the number of values for each random draw? (e.g., 6)

   Make an array with one element for each possible outcome. Set each element to 0. Then keep calling the random number generator. If it returns a value `v`, then increment the counter belonging to `v`.

   After all numbers have been generated, print the counters. Here is a typical program run:

```
How many random numbers do you want to generate? 1000
What is the number of values for each random draw? 10.

0    78
1   101
2   118
3    97
4   103
5   102
6   112
7    91
8    94
9   104
```

What is the code for your program?

5. Array lists are objects that, like arrays, provide you the ability to store items sequentially and recall items by index. Working with array lists involves invoking `ArrayList` methods, so we will need to develop some basic skills. Let's start with the code below:

```
import java.util.ArrayList;

public class ArrayListRunner {
   public static void main(String[] args) {
       ArrayList<String> names = new ArrayList<String>();
       System.out.println(names);
   }
}
```

The main method imports `java.util.ArrayList` and creates an `ArrayList` that can hold strings. It also prints out the `ArrayList` and, when it does, we see that the list is empty: `[]`.

Complete the following tasks by adding code to this skeleton program. If you are asked to print a value, provide a suitable label to identify it when it is printed.

(a) Invoke `add()` to enter the following names in sequence: `Alice`, `Bob`, `Connie`, `David`, `Edward`, `Fran`, `Gomez`, `Harry`.
Print the `ArrayList` again.

(b) Use `get()` to retrieve and print the first and last names.

(c) Print the `size()` of the `ArrayList`.

(d) Use `size()` to help you print the last name in the list.

(e) Use `set()` to change `Alice` to `Alice B Toklas`.
Print the `ArrayList` to verify the change.

(f) Use the alternate form of `add()` to insert `Doug` after `David`.
Print the `ArrayList` again.

(g) Use an *enhanced* for loop to print each name in the `ArrayList`.

(h) Create a second `ArrayList` called `names2` that is built by calling the `ArrayList` constructor that accepts another `ArrayList` as an argument. Pass `names` to the constructor to build `names2`. Then print the `ArrayList`.

(i) Call `names.remove(0)` to remove the first element. Print `names` and `names2`. Verify that `Alice B Toklas` was removed from `names`, but not from `names2`.

6. This is a *pen and paper* exercise. Consider the following code fragment.

```
ArrayList list = new ArrayList();
list.add("P");
list.add("Q");
list.add("R");
list.set(2,"s");
list.add(2,"T");
list.add("u");
System.out.println(list);
```

What is printed as a result of executing the code segment?

7. Write a program to:

- read numbers from the user,
- put each number in an `ArrayList`, and
- prints various information about the entered numbers.

Specifically, keep reading and adding numbers to your list until they enter `-1` (that is, a loop that keeps getting numbers until you read -1).

Then output:

(a) number of items entered before -1,

(b) the average of the input numbers,

(c) the standard deviation of the even numbers, and

(d) the sum of the odd numbers.

For example, this should be what happens when you run your program and enter:

```
Enter a number: 1
Enter a number: 2
Enter a number: 75
Enter a number: 26
Enter a number: -1

# of items: 4
Average: 26
Standard Deviation: 16.97
Sum of odd: 76
```

8. A very good sorting algorithm that uses recursion is named *merge sort*. It works as follows:

   1. To start, consider each element of the `ArrayList` as a "section" of size 1 (numbered 0 through n-1).
   2. Merge neighbouring *sections* together so that the resulting section (of double the size of the original section) is sorted.
   3. Repeat the previous step until there is only one section left.

   The above looks well and good in concept but there are a couple snags. The first is how to handle weirdly-sized sections (*leftovers*). Another is how to store the sections while we're working with them.).

   Here are some example runs of merge sort:

```
Input: arr =          { 1 8 4 13 99 23 17 7 25 }
 Initial sections =      {1} {8} {4} {13} {99} {23} {17} {7} {25}
 Merge #1 =           {1 8} {4 13} {23 99} {7 17} {25}
 Merge #2 =           {1 4 8 13} {7 17 23 99} {25}
 Merge #3 =           {1 4 7 8 13 17 23 99} {25}
 Cleanup =            {1 4 7 8 13 17 23 25 99}

 Input: arr =         { 13 99 47 0 23 13 86 }
 Merge #1 =           { 13 99 } { 0 47 } { 13 23 } { 86 }
 Merge #2 =           { 0 13 47 99 } {13 23 86 } //Note odd sized section
 Merge #3 =           { 0 13 13 23 47 86 99 }
```

   You are required to:

   (a) Perform a *merge sort* on the following `ArrayList` (showing all steps as above):

   ```
   { 13 47 200 53 0 100 33 8 31 75 123 47 99 }
   ```

   (b) Write a recursive method, `mergeSort(ArrayList arr)` that sorts the input `ArrayList` using merge sort.
   Hint: You'll need a *helper function*, `merge`, that merges two sections together.