Joanna Klukowska
joannakl@cs.nyu.edu

Assignment 4

Due date: Feb. 29, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but **all work for all assignments must be entirely your own**. Any sharing or copying of assignments will be considered cheating.

You should not use any features of Java that have not been covered in class. If you have doubt if you are allowed to use certain structures, just ask your instructor.

Using Javadoc Comments

Starting with this assignment, you should be using Javadoc style comments. This allows one to automatically generate HTML documentation for the code and is a standard used by Java programmers. For a detail guide see, for example, *How to Write Doc Comments for the Javadoc Tool* at `http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`. For what is required in your programs, see the following.

- Header comments at the top of each class (we soon will be using more than one class per program). These should contain description of the class (its purpose, general use, etc), the name of the author and (optionally) the version. For example:

  ```
  /**
   * This class represents a magical potion that consists of multiple
   * ingredients (represented by characters).
   *
   * @author  Harry Potter
   * @version 04/11/2015
   */
  ```

  Notice that the opening of this comment `/**` is different than the standard multiline comment in Java that starts with `/*`. This turns it into a Javadoc comment. This comment also uses special tags `@author` and `@version` that are interpreted by the javadoc program.

- Each method must have a block comment right above its signature. It should contain 1) the description of the method itself (what does it do) and state any assumptions that are made for the method to function properly; 2) a list and descriptions of all the parameters with any assumptions that are made about the parameters and the valid ranges (if applicable); 3) the description of the return value (if the method returns something); 4) the list of all thrown exceptions with the description of conditions under which they are thrown. For example, this is the documentation for the `addIngredient` method in the `Potion` class:

  ```
  /**
    * Adds a specified ingredient to this Potion object.
    *
    * @param ingredient ingredient to be added.
    *
    * @return true if the ingredient is a valid one (one of 'a', 'b', 'c', 'd', 'w'),
    * false, otherwise.
    */
  public boolean addIngredient( char ingredient ) {
    //body of the method
  }
  ```

  Notice the special javadoc tags above:

  - `@param parameterName` followed by description of the parameter
  - `@return` followed by description of the return value

In addition to Javadoc comments for methods and the class, all the code must contain inline comments - short descriptions within the code that guide the author and the reader of the code as to what is going on. Do not comment every line of code, though.

Joanna Klukowska
joannakl@cs.nyu.edu

Problem 1 (40 points): Merge two lists

Write a program that merges two lists of sorted whole numbers entered by the user. The program should use a 'merge' method with the following signature

```
public static int [] merge ( int [] list1, int [] list2)
```

The list returned by the method should contain all elements of the sorted `list1` and sorted `list2`. The list returned by the method should be sorted. Your implementation should be as efficient as possible: it should take advantage of the fact that the two lists are already sorted; it should not use any sort methods.

You also need to implement an 'isSorted' method to verify that the user input is actually sorted from smallest to largest and that your merged list is sorted. The signature of this method should be

```
public static boolean isSorted ( int [] list )
```

Your program should ask the user for the length of each list and then read in the content of the two lists. Your program should print to the console the content of the merged list (numbers should be separated by commas and spaces).

Call your file: `SortedMerge.java`.

Note: Do not use Arrays or ArrayList classes!

Problem 2 (40 points): Number Stats

Write a program that computes statistical information about an array: mean (average), mode (most frequent element) and median (the middle element, when tha values are sorted from smallest to largest). Your program should define a method that takes an array of integers as a parameter and returns the appropriate value. There should be one method that computes the mean, one that computes the mode and one that computes the median.

Note that you will need to implement a sort method in order to sort the list of numbers before you can compute the median.

Your program should generate a list of random integers in the range from -100 to 100 (inclusive) and then use that array with each of the methods. The program should use three different sizes of the array: 100, 1,000, 10,000. For each of the (randomly generated) arrays it should print the three statistics. Do not print the actual array to the screen in the final program that you submit.

HINT: It may be useful to start with even smaller array of just a few elements to make sure that your program performs correctly.

Call your file: `NumberStats.java`.

Problem 3 (20 points): Pattern Regognition (Increasing Sequence)

Write a program that finds an prints to the screen all of the increasing sequences of numbers of maximumal length from a given array. The shortest printed sequence should have length 2. For example, if the array contains the following values

```
    [ -7, 3, 0, 99, 120, 50, 13, 0, 1, 5, 56, 77, 3, 1, 5 ]
```

then the output of the program should be

```
 -7 3
 0 99 120
 0 1 5 56 77
 1 5
```

Your program should generate an array of 15 pseudo-random integers, display the array and then print the sequences. The sequences should be printed one per line with a single space separating each pair of numbers.

HINT: You can use the `toString` method from the Arrays class to print the content of the array. If your array is called list, then

```
  System.out.println( Arrays.toString (list) );
```

prints the array as shown in the above example.

NOTE: this program is short, but requires some algorithm developemnt to come up with a proper way of determining the increasing sequences.

Call your file: `IncreasingSequence.java`.

Joanna Klukowska
joannakl@cs.nyu.edu

Grading

Does the program compile? If not, you will loose all the points for that problem.

Is the program properly documented? (worth approximately 20% of each problem)

Proper documentation at this point in the course includes:

- preamble with the name of the author, date of creation and brief description of the program (the description should specify what the program does, not that it is a solution to problem 1 of homework 1);
- appropriately chosen variable names, i.e., descriptive names (a good name for the variable that stores the bonus amount in the last problem is `bonus`, not `x`);
- comments inside the code describing steps needed to be taken to accomplish the goal of the program;
- appropriate formatting, indentation and use of white space to make the code readable.

Remember that the code is read by humans and it should be easy to read for people who were not involved in its development.

Is the program well developed? (worth approximately 40% of each problem) Make sure you create variables of appropriate types, use control statements (conditionals and loops) that are appropriate for the task, accomplish your task in a well designed and simple way (not a convoluted algorithm that happens to produce the correct output for some unknown reason). You should also design a friendly and informative user interface.

Is the program correct? (worth approximately 40% of each problem) Make sure that your program produces valid results that follow the specification of the problem every time it is run. At this point you can assume a ”well behaved user” who enters the type of data that you request. If the program is not completely correct, you get credit proportional to how well it is developed and how close you got it to the completely correct code.

What and how to submit?

You should submit three source code files combined into a single **zip** file to NYU Classes. Do not submit all the files that Eclipse creates, just the source code files that have `.java` extensions. Name your classes as specified in the problems.

If you wish to use your (one and only) freebie for this project (one week extension, no questions asked), then complete the form at http://goo.gl/forms/fpUJrF64b5 **before the due date for the assignment**. All freebies are due seven days after the original due date and should be submitted to NYU Classes.

Questions

Post any questions you have regarding this assignment to Piazza under the ”homeworks” topic.