Queens College
Department of Computer Science

CSCI 212 – Object-Oriented Programming in Java – Fall 2008

Project 1

Assigned: 17 September 2008
Due: 14 October 2008
Cut Off: 20 October 2008

Inventory of Houses

Program Description:
A data file will be provided containing information about houses for sale. Each line of the file will contain the ZIP code for the house, price and number of bed rooms, each separated by a vertical bar ("|"). For example:

```
11367|429000|4
11358|350000|5
11421|27000|2
11530|1700000|8
```

Write a Java application called Project1.java which will read these lines of text from data file and put them into a partially-filled array of strings, and will then print the contents of the array to a series of three **JOptionPane** message dialog boxes with text areas as follows:

- The first message box will display the array, showing the houses in the original order in which they were listed in the file.
- The second message box will display the array after it is sorted by *zipCode*.
- The third message box will display the array after it is sorted by *price*.

The name of the input file should be the one and only command line argument. For example:

```
C> java Project1 inputFile.txt
```

Sorting by zipCode:
Adapt the **selection sort** method as seen in lecture so that it will sort strings rather than shorts, and put it into a static method called **sortByZipCode** (the parameters should be the array you want to sort, and the number of items the array contains.) Strings are compared using the **compareTo** method of class **String**. Since the strings in the file start with the zipCode, this is sufficient to sort the array according to zipCode. Here is an example of the usage of *compareTo*:

```
String s1 = "Hello";
String s2 = "Goodbye";
if (s1.compareTo(s2) < 0)
   System.out.println("s1 is smaller than s2 lexicographically.");
```

Sorting by price:
Sorting the array in ascending order by price is a little more tricky since the number of digits in the price varies. What we need to do is break the line apart, separating the values between the vertical bars ("|"). For example, to compare "11367|270000|4" with "11530|2130000|8" we need to pull out the "270000" and the "2130000". Splitting up the string can be done using a **StringTokenizer**. (See the Project 1 hints on D. Nixon's website.)
Assuming we can extract "270000" and "2130000", the next problem is comparing the two. Since they are strings, "270000" is greater than "2130000"! We will need to convert these strings to integers so they can be compared as integer values. The wrapper class **Integer** provides us with the method *parseInt* which will do this conversion, similar to the *parseShort* method show in lecture.

All of this work should be done in a static method called **comparePrice.** This method should return an integer value with the same interpretation as the **compareTo** method of class *String*: , it should return a number less than zero if the first price is less than the second price, a number greater than zero if the first price is greater than the second price, and zero if the two capacities are equal. Here is an example of its usage:

```
String house1 = "11367|270000|4";
String house2 = "11530|2130000|8";
if (comparePrice(house1, house2) < 0)
   System.out.println("House 1 is smaller than House 2.");
```

If your **sortByZipCode** method works, then make a copy of it and call it **sortByPrice,** replacing the compareTo method of class string with the **comparePrice** method.

Program Requirements:
Your program should contain the following methods:

`public static void main (String args[])`
Obviously the main method for this program, it should call the methods below as appropriate. It should take a command-line argument for the filename of the input data file.

`private static int readFile (String filename, String[] capacities)`
This is essentially the method shown in the PowerPoint on "Arrays and Sorting." It reads from the file whose filename is given as a parameter, and fills array of strings representing capacities.
Error checking:
Each line of the file should be checked for the following errors. Lines of the file that contain errors should not be put in the array (print them to the system console instead).

- The line should have exactly three fields (values separate by "|")
- The price muse be greater than zero and less than 3000000
- The number of rooms must be greater than zero and less than 30

The test for validity should be done by the method **`isValidHouse`** described below. The returned value of the `readFile` method is the number of houses successfully put in the array. (Note: do not use exceptions or assertions for this purpose. A simple if/else within your loop for reading the file should be adequate.)

`private static void sortByZipCode (String[] lines, int length)`
Sorts the array of strings (no tokenizing needed) by zipCode name using selection sort.

`private static void sortByPrice (String[] houses, int length)`
Sorts the array in price order using selection sort. In order to sort the price strings in order by price, using **`comparePrice`** rather than **`compareTo`** method of the **`String`** class.

`private static int comparePrice(String house1, String house2)`
compares two capacities. Returns a negative number if house`1's` is less than house2's, 0 if the price of the houses is the same, or a positive number if house1's price is greater than house2's.

`private static void displayResults(String[] houses, int length)`
Prints the contents of the array to a text area on a **`JOptionPane`**.

`private static boolean isValidHouse (String house)`
Returns true if and only if the string representing a **house** has the proper format (see **Error Checking** above.

Other Project Requirements

You are required to include Javadoc comments in your program, including comments above the class heading and comments above each of the method headings, with appropriate tags (**@author**, including your name, in the class comment, and **@param** and **@return** in the method comments where appropriate). Proper indentation, meaningful variable names and respectable logic are also required.

Developing and Testing Your Project

You may compile and run your project using either the command line or Eclipse. Note however, that your project will be graded **using the command line**. Depending on how you configure Eclipse, some errors will be accepted by Eclipse but **not by the command line**. Therefore, before submitting your project, you should compile it using the command line.

Program Submission:

The name of your file must be **`Project1.java`** with upper/lower case exact.

You must submit ***both*** (1) an electronic copy via Blackboard and (2) a printout, on which the print must be reasonably dark. If you are using Eclipse, see the notes on Blackboard about configuring it. (Note that your project will not be graded unless it is submitted both ways.) Also, to receive any credit at all, your program must at least compile. (Partial credit will be given for projects that compile and run but don't do quite what they are supposed to.)

The printout MUST be received by your lab instructor no later than the next lab session after the cut off date for the project.