**Day 28 - Sorting More Complex Data**

Consider data from a file that represents students. For each student, the data will have a lastName (UTF), a firstName (UTF), an id (int), a GPA (double), an email (UTF), a street address (UTF), city (UTF), state (UTF), zip code (int), and phone number (UTF).

We will read each student into a Student object, store them in an ArrayList of Students, sort them by id number, and then print the list.

To do this, we need to use one of our sorting algorithms (selection sort) and change it to work with the ArrayList instead of an array. Check out the code below.

public class StuSort {  
 // 'list' is the list to sort ... read in from the  
 // file named in STU\_DATA  
 **private ArrayList<Student> list = new ArrayList<Student>();  
 public static final String STU\_DATA = "StuData.dat";**   
 /\*\* main program ... just instantiate StuSort2 \*/  
 public static void main(String[] args) {  
 new StuSort();  
 }  
   
 /\*\* constructor ... the effective main program \*/  
 public StuSort() {  
 **readData();** **// read the file**  
 int start = 0; // the index of first element that is unsorted  
 int candidate = start; // the index of the candidate for smallest  
 int current = 1; // the index of the element being checked  
   
 // One pass for each element except first  
 while(start < list.**size()**-1) {  
 // One pass ... find smallest element's index  
 current = start + 1; // reset for another pass  
 candidate = start; // reset for next pass  
   
 // The inner while loop == one pass  
 // Looking for smallest value between list.get(start)  
 // and list.get(list.size()-1]  
 while(current < list.**size()**) {  
 if(list**.get(current).getId()** <   
 list**.get(candidate).getId()**) {   
 // current is smaller than the candidate  
 candidate = current;  
 }  
 current++;  
 }  
   
 // If we found a smaller candidate than list.get(start), swap with  
 // list.get(start) so the smallest value is at the start of the list  
 if(start != candidate) {  
 **Student** tmp = list**.get(start);**  
 list**.set(start, list.get(candidate));**  
 list**.set(candidate, tmp);**  
 }  
   
 // For next pass, increment start to start 1 place further on  
 start++;  
 }  
   
 // When out of the outer loop we are all done  
 **dispList(); // print out the entire list** }

In the above, the code in **bold** is code that changed from our earlier selection sort implementation. Much of this is due to using an ArrayList instead of an array, or to reading data in and printing it out. Could we use this code as the basis for sorting an ArrayList of **any** kind of objects? And sorting those objects on **any** field? We can and Java has done this for us. The only place you need to know exactly what the data is (what structure it has), is when you are comparing the two elements under consideration.

To do so, we will use the **Collections** class and the **Comparator** interface. The way this works is that the Collections class has a **sort** method to which you pass a Collection (e.g., ArrayList or Vector) and a way to compare two elements. The way to compare two elements is a class (usually inner) that implements the **Comparator** interface and includes a method called **compare**. Now the above looks like:

public class StuSort2 {  
 // 'list' is the list to sort ... read in from the  
 // file named in STU\_DATA  
 private ArrayList<Student> list = new ArrayList<Student>();  
 public static final String STU\_DATA = "StuData.dat";  
   
 /\*\* main program ... just instantiate StuSort2 \*/  
 public static void main(String[] args) {  
 new StuSort2();  
 }  
   
 /\*\* constructor ... the effective main program \*/  
 public StuSort2() {  
 readData(); // read data in from the file  
   
 // This line does the sorting. Comparisons are done  
 // by calling a SortByName object with two Students  
 Collections.sort(list, new SortByLastName());   
   
 dispList(); // print data out  
 }

The actual sorting code is gone, replaced by a call to Collections.sort. It uses a class, SortByLastName, which is an inner class that looks like:

/\*\*  
 \* SortByLastName ... inner class that implements Comparator<Student>  
 \* with method compare to compare two students any way we wish.  
 \*/  
 class SortByLastName implements Comparator<Student> {  
 public int compare(Student s1, Student s2) {  
 return s1.getLastName().compareTo(s2.getLastName());  
 // return -s1.getLastName().compareTo(s2.getLastName());  
 }  
 }

When **Collections.sort** needs to compare two items in the ArrayList, list, it passes those two items to the **compare** method of the **SortByLastName** class. It compares the Strings using the String class’s **compareTo** method. Look this method up and determine what is going on here.

So, if s1 < s2 (alphabetically), **compareTo** returns a negative number. If s1 > s2, **compareTo** returns a positive number. If s1 and s2 are identical, **compareTo** returns a 0.

What would happen if we comment out the first return statement in the **compare** method, and uncomment the second one?

**In Class Exercise**

Rename StuSort2 to StuSort3. Change it so that it reads from the command line a field name and an optional reverse indicator at the start of the constructor. You will have to pass String[] args from the main to the constructor for this:

String field = "";  
 String reverse = "";  
 if(args.length < 1 || args.length > 2) {  
 System.out.println("Usage: StuSort3 FIELD [reverse]");  
 System.exit(1);  
 }  
 else {  
 field = args[0].toUpperCase();  
 reverse = args.length == 2 ? args[1].toUpperCase() : "";  
 }

Add **SortByFirstName, SortByID,** and **SortByZipcode**. Give each the reverse indicator, above (pass it to the constructor of each of these classes in the **new** statement).

Finally, have the **each** class’ **compare** method do a comparison by LastName, FirstName, ID, or Zipcode (depending on the class), in either the forward (increasing) or reverse (decreasing) mode (depending on the value passed to the constructor). Treat attempts to sort on any other field as an error (print a message and terminate the program).

When done, this should allow you to sort and print the data by either name, ID, or Zipcode in either increasing or decreasing order. For example, to sort by First Name in increasing order, supply the command line argument:  
 FIRSTNAME

To sort by Zipcode in decreasing order, supply the command line arguments:  
 ZIPCODE REVERSE

**Searching**

***Sequential Search***In selection sort, we searched for the smallest value and then swapped it with the 1st element in the collection of elements we were searching.

The selection sort used the simplest form of searching, called a sequential search. Just look at the first element, then the second, and so on until you find the one you are looking form. See SequentialSearch.java in today’s downloads.

***Binary Search***If the data is sorted, we can employ a different searching technique, known as binary search. Suppose we have an array of 10 numbers, sorted in increasing (ascending) order. We wish to find out if a value is in the array and, if so, at what position. Our approach will be to choose the value in the middle. If it is the one for which we are looking, great. If not, then depending on whether the value we just checked is < the value we are looking for, or > the value we are looking for, we can discount one half of the array or the other and reduce the search space by ½. Now, check the value ½ way through this smaller array, and so on.

Search the array for a target value.  
 int binarySearch(int[] array, int target) throws Exception {  
 int start = 0;  
 int stop = array.length-1;  
 while(start < stop) {  
 int location = (start + stop) / 2;  
 if(target == array[location]) // FOUND IT!  
 return location;  
 else if(target < array[location]) // Look in 1st half of array  
 stop = location – 1;  
 else if(target > array[location]) // Look in 2nd half of array  
 start = location + 1;  
 }  
 if(target == array[start])  
 return start; // Found it at the end of the search  
 else  
 throw new Exception("Value: " + target + " not found");   
 }