Programming Structures and Abstractions (CS 2334)
Lab 5: Maps, Lists and Sets (DRAFT)

October 7, 2010

Due: Monday, October 11, 2010, 5:00pm

Group members (same as for your project):

Objectives

By the end of this laboratory exercise, you should be able to:

1. use Maps to store and retrieve data,
2. choose an appropriate Map implementation given your computational and performance requirements,
3. combine Maps and ArrayLists to store multiple Map values with the same key, and
4. compute statistics of data stored within a Map.

Problem Context

As we start to construct databases that contain terabytes or petabytes of data, critical issues that must be addressed include: how much computation time is required to access and/or modify these data? and how much storage space do these data (and associated data structures) require? Depending on the nature of the data, the types of queries of the data that we will be performing, and how often data elements are updated or added, we will make different data structure and algorithmic choices in order to address the computational time and memory measures of performance.

One of the key concepts in the organization of data for efficient access is the *Map*. Here, a collection of data elements (or *values*) is stored as a group of object instances. Access to this collection is through a set of *keys*. The Map data structure is such that it is very efficient to 1. determine if a key exists within the map (much more efficient than searching linearly through a set of keys), and 2. gain access to a reference to the corresponding value that is stored with the key.

For example, imagine a collection of *Person* objects that each contain information about the person's name, date of birth, physical/virtual address, and job title, type and location. We can store this collection of objects within a Map and use a set of unique ID numbers as keys (e.g., social security numbers). This map makes it computationally efficient to look up a given person if we have their ID number.

In most databases, we don't have just one type of key, but many different ones. For example, we may want to look up the Person objects by family name, date of birth or job type. In order to do these efficiently, we could create Maps over the same collection of Person objects, but keyed using these different variables. When we do this, however, we must be aware of the fact that a proper Map will only store one value for each unique key. There are a variety of ways to address this issue.

For this laboratory exercise, we will sample a set of *FinchSensor* objects from our Finches and store them in a Map that is keyed using the time that the sample was taken (hence, there is exactly one FinchSensor object for each key). We will then examine the organization of our data set based on our choice of Map implementation, and then examine the performance of accessing elements within the Maps. Finally, we will construct a new map in which the same set of samples is keyed using the value of the left light sensor value.

Milestones

Milestone 1

From the class web page, go to the **lab5** directory. Download the **Lab5.zip** file and import it into Eclipse as a new project. This project file includes several java class and interface implementations, as well as the associated javadoc.

Examine the **main()** method through the "END Milestone 1", and any methods that it calls. Explain briefly in the space below what the function is of the line following the one marked "MILESTONE 1 QUESTION" (note that you will likely need to refer to your book and to the Java API):

With a Finch connected to your computer, compile and execute the FinchDriver. You will see two reports containing the same set of ten samples: one for HashMap and the other for TreeMap. Each line corresponds to one FinchSensor sample that was taken at the beginning of **main()**. List below the meaning of each of the numbers/words in each sample (you will need to examine the code to determine this):

What is the fundamental difference between the two reports? Explain the source of this differences:

Milestone 2

In the **main()** method, remove the line following "TODO: Milestone 2: remove next line". Change the static constant **numSamples** from 10 to 500. Briefly explain the function of the methods called **queryTest()**:

Compile SensorDriver and execute it five different times. For each execution, the execution times for both the HashMap and TreeMap are reported. In addition, the execution time for the "List" is reported. List all 15 numbers:

What do you notice about the relative performance of the query for the HashMap, TreeMap and List? Explain this difference:

Milestone 3

Both of the maps that we have used are keyed using the time that the FinchSensor sample was taken. When this data set becomes large, these maps make it computationally inexpensive (relative to simple Lists) to retrieve the FinchSensor sample for a given key value. However, we may wish to organize our map so that it is keyed instead by some other variable.

For this milestone, we will create a new map that is keyed by the value of the left light sensor. The problem that we face is that for a given left light sensor value, multiple

FinchSensor samples may share that same value. In a proper map, such duplicate key entries are not allowed. In fact, in any Java map, if a key/value entry is added to a map that already contains the same key, the old key/value entry will be replaced by the new one. Here, we would like to preserve all of the FinchSensor values with the same left light sensor key. Our solution for this milestone is to map from left light sensor values to an ArrayList of FinchSensor objects.

In the **main()** method, remove the line following “TODO: Milestone 3: remove next line”.

Complete the implementation of the method **reKey()**. This method creates a new TreeMap that maps from Integers (light sensor values) to ArrayLists of FinchSensor objects. When we first create the map, it contains no key/value pairs. Your implementation must iterate through the list of key/value pairs in the original map and insert each of these values into the new TreeMap. If a key does not exist in the new TreeMap, then you must first create a new ArrayList object for that key before adding the FinchSensor object to the ArrayList.

Execute SensorDriver while slowly spinning your Finch about its major axis (i.e., its nose). Your program will generate three new reports. For each report, write down the mean value for both **Acceleration magnitude** and **Acceleration Z**:

What is the meaning of each of these reports?

What interesting correlations do you see in the data?

What to Hand In

All components are due: Monday, October 11, 2010, 5:00pm

Hand in the following:

- a copy of this handout containing your answers answers, and
- an electronic copy of your modified code contained within a file called Lab5.zip (to D2L).

Demonstration:

- Perform a short demonstration to the instructor or one of the TAs.

NOTE: ONLY HAND IN ONE ELECTRONIC/PAPER COPY PER GROUP.