Programming Structures and Abstractions (CS 2334)
Lab 4: Generics and Collections

September 30, 2009

Due: Friday, October 2, 2009, 11:29am

Group members (same as for your project):

Objectives

The objectives of this lab are to:

1. analyze the class structure of an existing java program using UML diagrams,
2. use the ArrayList class to store a set of FinchSensor samples,
3. create several Comparator object classes that enable a sorting of the ArrayList in different ways, and
4. use the ArrayList to compute statistics over individual sensors.

Problem Context

This project is an extension to lab 3 in which we will make use of the Java ArrayList class to organize our set of samples. Specifically, we will:

- replace an array of FinchActions with a ArrayList (this has already been done in the code),
- implement Comparator object classes that will allow us to sort the samples in different ways, and
- compute statistics over the sensor values stored in our data log.

In lab 3, we used an enumerated data type, the **Comparable2** class, and a simple **sort()** method to organize our set of FinchSensor samples. The **ArrayList** and **Collections** classes provide a similar set of capabilities. As before, we are faced with the problem of sorting our samples based on the value of some sensor variable that is determined only by the **main()** method (i.e., the ArrayList or FinchSensor must not know beforehand which sensor variable will be used for sorting). Instead of specifying the variable using an enumerated data type, we will specify it with a set of Comparator-derived classes.

Milestones

Milestone 1: Analyze a Class Structure

From the class web page, go to the **lab4** directory. Download the **Lab4.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.

Using the javadoc documentation and the source files, draw a UML diagram that captures the the relationships between the classes.
In your UML diagram, you should:

- Give the details of **only** the SensorComparator abstract class. Include the variable definitions and method prototypes.
- Detail the relationships between **all** of the the classes and interfaces. In particular, you should express:
  - class/interface inheritance, and
  - the dependency of classes on interfaces and other classes.

Milestone 2

Looking at sensorDriver.java, please answer the following questions:
1. Briefly describe the conditions in which the ArrayList class is explicitly used in the code.

2. The method **mean()** (and several other methods) is intended to operate on the instance of ArrayList that is created in **ReadSensors()**. However, **mean()** does not explicitly accept an ArrayList as a parameter. Briefly explain why this is the case and why this may or may not be a good choice.

3. In method **displayStats()**, briefly explain the function of the call to **sort()** (you may need to look ahead to the next milestone).

Milestone 3

Complete the implementations of the following classes:

- TemperatureComparator
- XaccelComparator
- ZaccelComparator
- LightComparator

For each of these, you must provide the implementations of the following methods:

```
public int compare(FinchSensor obj1, FinchSensor obj2);

public double doubleValue(FinchSensor obj);

public String toString();
```

Note that there are several static helper methods in the super class.

Milestone 4

In the **sensorDriver** class, complete the implementations of the following methods:

```
static public FinchSensor median(List<FinchSensor> log)
```

This method should return the FinchSensor sample that corresponds to the median value of the particular sensor in question. **Assume that log is already sorted from lowest to highest based on this sensor value.** Hint: the solution to this problem is one line of code.

```
static public double mean(List<FinchSensor> log, SensorComparator comp)
```

This method returns the mean (average) of the sensor variable specified by the Sensor-Comparator. Note that the SensorComparator provides a **doubleValue()** method. Hint: use some form of iterator to visit all of the elements in the List. The solution to this problem is a few lines of code.

Milestone 5

With your Finch sitting wheels down, execute your program. While recording, cover the light sensors and slowly turn the Finch all the way over (so its top is facing down). Then: slowly turn the Finch so that is facing up. Uncover the light sensors and allow the program to complete.

After recording is complete, the program will report the raw values in order. In addition, the program will print out the mean value of ACCEL_X and find the samples with the min, max and median ACCEL_X and print out each of these samples. This is repeated for ACCEL_Z, TEMPERATURE and LIGHT values Write down (or print out) these latter numbers (but not the raw values).

Milestone 6

With your Finch sitting wheels down, execute your program. Slowly rotate the Finch so that its beak is facing upwards. Then, cover the eyes for a couple seconds. Finally, uncover the eyes and then return the Finch to the wheels down configuration.

Again, write down (or print out) the average values and the values for the samples of the min, max and median channel values.

What to Hand In

All materials are due: Friday, October 2, 2009, 11:29am

Hand in the following:

- a copy of your UML diagram (hard or soft copy will do),
- a copy of this handout containing the provided answers, and
- an electronic copy of your modified code (to D2L).

NOTE: ONLY HAND IN ONE COPY PER GROUP.