CS2334

- Project 1 Deliverables
- Lab 3 Introduction

Project 1 Demos

- Before demoing your project:
  - Turn in your UML and Cover Page
  - Submit javadocs and Code stubs to D2L (as a single zip file)
- Design submissions must be in before demoing
  - You are expected to complete the design process prior to coding (it is, after all, meant as a planning phase)

Project 1 Demos cont.

- I have a zip drive with different sets of valid instructions
- One will be chosen at random and placed in the proper directory so your code can open and read it
- Your program will be told to run “ALL” as well as a specific instruction set (for example: “dance”)

Project 1 Demos cont.

- Your group has until 5 pm on the 24th to have a correct demo run (some negotiation possible **BEFORE** the deadline)
- If you do not pass a demo the first time, you can come back after figuring out what went wrong
- Each demo will be run with a different input file

Lab 3 Motivation

- Sensor network: distributed set of sensors that collect data at regular intervals from many locations
- Applications include: weather, earthquake and building health monitoring systems
- Want to automatically detect “sensory events” that indicate something important has happened

Lab 3 Objectives

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

1. Analyze the class structure of an existing java program using UML diagrams,
2. Extract and store sensor data from the Finch
3. Employ abstract classes to provide generic programming functionality
4. Search the Finch “data streams” for key values

Sensor Samples

- We will take a sample of data at regular (100 ms) intervals for 10 seconds (100 samples total).
- Each sample is a **tuple** that contains the values from the light, acceleration, obstacle and temperature sensors.

Queries

Goal: find and report the minimum, maximum and median data sample

- One way to do this: sort the samples and then take the first, last and middle samples

- How do we sort the samples?

Queries

How do we sort the samples?

- Use the sort method from the book (pp. 479-481)
- But: we need some way of telling sort() to use a particular class variable (such as the Y component of the accelerometer)

One Solution…

Define a new interface Comparable2

- Requires that the following is provided by the implementing class:
  ```
  public int compareTo2(Object obj, VariableType var);
  ```

- VariableType is an enumerated type that tells the implementation of compareTo2() which class variable to compare

Enumerated Types

- An enumerated type variable is a means of storing one of several values
- Values are typically symbolic:
  - TRUE and FALSE
  - TEMPERATURE, LIGHT_LEFT, etc.
- Values are often non-ordered
  - The "equals" operator is meaningful
  - Greater-than and less-than are not meaningful

Enum Example

- For an example of enumerated types, consider cardinal direction, which one of four possibilities.
- You have to list each possible type of cardinal direction when defining the enum:

```
public  enum CardinalDirection{
   NORTH,  EARTH, SOUTH, WEST;
}
```

Enum Example cont.

Now that we have defined CardinalDirection consider a function that uses it

```
String getDirection(CardinalDirection direction){
   switch(direction){
        case NORTH:
                return “North”;
        case EAST:
                return “East”;
        case West:
                return “West”;
        case South:
                return “South”;
        default:
                throw(new Exception(“bad enum value”);
   }
}
```

Enum Example cont.

- That was relatively painful when all we wanted was a toString for the enum…
- It turns out toString() can show enum info
- In order for this to work you have to create an instance variable to hold enum names
- How is this possible??? Constructors!
- See Lab3's SensorType.java for an example

Provided Classes and Interfaces

- VariableType: a generic interface for specifying which variable (or “key”) to do the comparison on
- SensorType: a specific interface that provides an enumeration of the different sensor channels
- FinchSensor: a class that stores a single sensor sample
- Comparable2: an interface similar to Comparable
- sensorDriver: the top level program

General “To Do”

- Get the Finch talking to your computers
- Download Lab3.zip
- Analyze program (and draw the UML diagram)
- Provide implementation for compareTo2()
- Perform quick experiments

(try to get through all of these by the end of lab and don’t work on project 1 design until lab3 is done)