Programming Structures and Abstractions (CS 2334)
Project 2

September 22, 2010

Introduction

In project one, you applied your knowledge of class hierarchies to define a set of classes that represent elemental actions that can be taken by the Finch. In this project, you will be applying your skills of modifying these class hierarchies, defining enummerated types and using generic programming techniques. In particular, you will add a notion of *priority* to the Finch actions that will serve in part as a basis for sorting lists of actions. In addition, you will add several new Finch actions that make use of data received from the Finch's sensors.

Objectives

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

1. extend an existing set of classes to meet added requirements,
2. design and implement enumerated data types,
3. employ the *Java generics* facilities to write safe generic code, and
4. employ a generic sorting method through the use of *Comparable* objects.

Milestones

The summary of milestones is as follows:

1. Add an integer **priority** to all FinchActions (5 pts)
2. Create three new subclasses of FinchAction: **FinchMoveGuarded**, **FinchOrientationGuarded** and **FinchObstacleGuarded**. In addition, create two classes that implement enumerated data types: **FinchOrient** and **FinchObstacle** (35 pts)
3. Configure FinchAction to implement **Comparable** (15 pts)
4. Implement **sort()** method for a **FinchActionList** (15 pts)

Other components:

- Develop and use a proper design (UML and class stubs) (15 pts)
- Use proper documentation and formatting (javadoc and in-line documentation) (15 pts)

Note: of these last two components, a total of 15 points are available during the design phase. The remaining 85 points are obtainable for the final submission of the project.

This lab is due in two phases:

1. Thursday, September 30^{th} at 5:00pm: design.
2. Thursday, October 7^{th} at 5:00pm: completed program and short demonstration. If all components are complete by Tuesday, October 5^{th} at 5:00pm, then a bonus of 5% will be added to the group grade (the group grade will be multiplied by 1.05).

More details for what to hand-in and when are given below.

Resources

Main class web page:

- Java JDK 6 Classes
- General Finch documentation
- Finch software installation procedure
- Finch API: how to talk to your Finch
- FinchSoftwarev3_OU.zip: core Finch code and example programs (download and install on your hard disk outside of your project1 folder)

Main web page / projects / general :

- Documentation_Requirements
- Submission Instructions

Main web page / projects / project2 :

- project2.pdf: this project description
- project2_slides.pdf: a copy of the lab section slides
- seek.txt: an example input file

Input Files

The input file specification has changed. In particular, every action now has an integer **priority** in addition to a name. In addition, we have introduced three new actions.

- Move the Finch:

  `MOVE <name> <priority> <duration> <left distance> <right distance>`

- Change the nose color:

  `NOSE <name> <priority> <red> <green> <blue>`

  Color channels are integers in the range of [0 ... 255]

- Generate a sound of a given duration:

  `BUZZ <name> <priority> <duration> <frequency>`

- Guarded move: move the wheels of the Finch until an obstacle is observed:

  `GMOVE <name> <priority> <left velocity> <right velocity>`

  where velocities are specified in cm/sec.

- Guarded orientation: wait until the Finch is in one of six different orientations

  `ORIENT <name> <priority> <orientation>`

  where **orientation** is one of the following strings: "beakup", "beakdown", "upside-down", "level", "leftup" or "rightup"

- Guarded obstacle: wait until the Finch obstacle sensors are in one of seven different configurations:

  `OBSTACLE <name> <priority> <obstacle>`

  where **obstacle** is one of the following strings: "leftblocked", "leftunblocked", "right-blocked", "rightunblocked", "anyblocked", "bothblocked" or "bothunblocked"

Example File

```
NOSE      seek   12      0       0       255
GMOVE     seek   15   30.0    30.0
NOSE      seek    1      255     0       0
MOVE      seek   19   500     -10.0   -10.0
ORIENT    seek    6      level
NOSE      seek    4      0       255     0
OBSTACLE  seek    9      leftblocked
MOVE      seek   27   2000    20.0    -20.0
ORIENT    seek    2      beakup
GMOVE     seek   38   30.0    30.0
OBSTACLE  seek   13      bothunblocked
```

As before, after loading of this file into your data structure, the user will be able to search for a particular name and either display or execute the sequence of FinchActions that match the name. If the user specifies the name as "all", all of the FinchActions are displayed/executed in **name/priority order**.

Milestones

A milestone is a "significant point in development." Milestones serve to guide you in the design and development of your project. Listed below are a set of milestones for this project along with a brief description of each. As you implement each milestone, you must also create a milestoneX class that includes a main() method. This method should test the key components of the milestone. For example, new classes that must be implemented should be tested to ensure that their constructor, accessor and mutator methods perform appropriately.

Milestone 1: Add priority to all FinchActions

Add a class variable called **priority** to all FinchActions. This should be a required parameter for the constructor (as is the name). Provide the appropriate accessor and mutator methods.

Milestone 2: Create three new FinchAction child classes

The three new action classes are **FinchMoveGuarded**, **FinchOrientationGuarded** and **FinchObstacleGuarded**.

FinchMoveGuarded is similar to class FinchMove, but the velocities of the two wheels are specified. However instead of spinning the wheels for a specified amount of time, the wheels continue to spin until an obstacle is detected by either the left or right obstacle sensors. If an obstacle is detected immediately, then the wheels should not spin.

FinchOrientationGuarded is a class that waits for the Finch to be in one of six different orientations: beakup, beakdown, upsidedown, level, leftup (left wing) or rightup. If "upsidedown" is specified, then the execute method will wait until the Finch is placed on its back before returning.

The six different orientations must be represented using an enummerated data type called **FinchOrient**. Hint: as with FinchSensor of lab 3, you can use the constructor of the enumerated instances to define the string that corresponds to the enumerated value. In addition, the enumerated type can provide a static method that translates a string (obtained from the file) into a reference to the corresponding enumerated value.

See the Finch API documentation for a description of the methods that will tell you when your Finch is in a particular orientation.

FinchObstacleGuarded is similar to FinchOrientationGuarded in that it waits for a specific sensory condition. Specifically, an instance of this class waits for the obstacle sensors to be in one of seven different configurations: left sensor blocked, left unblocked, right blocked, right unblocked, any blocked, both blocked and both unblocked. These different cases must be represented using an enummerated data type called **FinchObstacle**.

Milestone 3: FinchAction implements Comparable

Configure the **FinchAction** class to implement the **Comparable** interface. This includes a concrete implementation of the **compareTo()** method in the FinchAction class:

```
public int compareTo(...) { ...}
```

This method orders actions first by **name** (alphabetical, ignoring case) and then by **priority** (lowest to highest).

Note: you must specify the Comparable implementation such that only FinchActions are compared with other FinchActions using the *Java generics* facilities.

Milestone 4: Implement a sort method for FinchActionList

This milestone is to be performed in three steps:

1. Implement a generic sort. Create a class **myUtil** that provides a static sort() method that is adapted from listing 14.10 (edition 8 of the book; it is listing 11.10 in edition 7). The problem with the book's implementation is that it accepts an array of any Comparable objects. Alter this implementation so that it will only accept an array of Comparable objects of a specified type using Java generics.

2. Provide the following method as part of class **FinchActionList**:

   ```
   public void sort();
   ```

   The implementation of this method must use the generic sort() that you have already implemented.

3. Call FinchActionList.sort() after you read an action list from a file.

Note: if your implementation of **FinchActionList** includes null references in the array, then you will need to account for this in your implementation of **myUtil.sort()**.

Hand-In Procedures

Part 1: Design

Deadline: Thursday, September 30^{th} at 5:00pm

Each group must hand in one copy of each of the following:

1. A printed cover page that lists the group members, work contributed by each, and any outside citations. Turn in the hardcopy to the TA or the lecturer.

2. UML diagram on engineering paper: turn in the hardcopy to the TA or the lecturer.

3. project2_design.zip to D2L. This is the zip file produced by Eclipse that contains:

   - Each of the classes with class variables, method header documentation and method stubs (prototypes). **For any new classes or methods, do not include any code other than returns or calls to this() or super().** Any implementation that you bring from project 1 is fine to include.
   - html description of your project produced by javadoc (export all elements, including private ones). Make sure to check that the resulting html files contain all of the correct information.

Part 2: Complete program and short demonstration.

Deadline: Thursday, October 7^{th} at 5:00pm

Each group must do the following:

1. Turn in a printed cover page that lists the group members, work contributed by each, and any outside citations. This cover page must document which group member implemented which classes and methods. Turn in the hardcopy to the TA or the lecturer.

2. Hand in the modified UML diagram on engineering paper: turn in the hardcopy to the TA or the lecturer.

3. Turn in project2.zip to D2L. This is the zip file produced by Eclipse that contains:

   - Each of the class implementations with documentation. **Do not include the author names in the Java files** (this is to be included on the cover page).
   - html description of your project produced by javadoc.

4. A short demonstration. We will reserve time during your laboratory section for you to demonstrate your working program. You may also attend office hours or make appointments to perform the demonstrations.

Note: If all components are complete by Tuesday, October 5^{th} at 5:00pm, then a bonus of 5% will be added to the group grade (the group grade will be multiplied by 1.05).

Hints and Notes

- Your program must be your own work. Do not discuss or look at the solutions of other groups in the class. However, you may discuss general issues (i.e., not directly related to the project requirements) with your classmates, as well as use the book and the resources available on the net.
- Start your work early. This is not a trivial programming assignment.
- Ask for help early. If you are stuck on something, talk to the TA or the instructor sooner than later (this is what we are here for).
- See the Finch API documentation for a list of methods that will allow you to access the sensors and to produce behavior.
- As the Finches are in short supply, please do not keep them longer than you need them.