Programming Structures and Abstractions (CS 2334)
Project 5

December 8, 2010

Introduction

Recursion is a powerful tool for implementing solutions to some complex problems, from searching for the next best game move to drawing fractals. This project is an extension of project 4 in which we will define a new type of FinchAction, **FinchMeta**, that consists of sequences of FinchActions. This recursive meta action will allow us to draw simple fractal structures with our Finch.

NOTE: This project must be done individually. You may start from your group's implementation of project 4. However, there may be no project-specific interactions between individuals for project 5.

Objectives

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

1. design and implement recursive data structures and programs, and
2. use the recursive nature of our FinchActions to create interesting Finch programs.

Milestones

1. Implement the scaled execution method for all **FinchActions** and for the **FinchActionList** (20 pts)
2. Implement the **FinchMeta** action class that will contain a sequence of FinchActions (20 pts)
3. Update the FinchActionDialog class (20 pts)
4. Write your own FinchAction text files (10 pts)

Other components:

- Describe the details of your FinchMeta class using a detailed UML diagram. Only show the directly related classes (these latter classes may be abbreviated) (5 pts)
- Use proper documentation and formatting (javadoc and in-line documentation) (15 pts)
- A short demonstration (10 pts)

Other notes:

- You must extend the provided abstract class (FinchModelAbstract.java). This class may not be edited. (note: exceptions may be granted by the instructor if appropriate reasons are given)
- You must write your own code for the GUI component layout (no tools may be used to automatically generate this code).
- Implement and test incrementally. Doing the implementation in the order of the milestones will help you do this.

All components of this lab are due on: Thursday, December 9^{th} at 5:00pm. More details for what to hand-in and when may be found below.

Resources

Main web page: see section on project 5

Main web page / projects / project5

- project5.pdf: this project description
- Abstract class definition: FinchModelAbstract.java (replace the project 4 class with this one)
- *.txt: example input files

Input Files

We will keep the same input text file format as with the previous projects. However, the text file format now has two additional options:

```
META <name> <priority> <filterName> <scaleFactor>
SMOVE <name> <priority> <duration> <left distance> <right distance>
```

where:

- *filterName* is a string that will be matched against the *names* of the actions in the master **FinchActionList**.
- *scaleFactor* is a double in the range [0 .. 1] (exclusive) that specifies how much smaller the child actions are than the current meta action.

Milestones

Milestone 1: Scaled Actions

1. Implement for all actions (existing and new):

```
public void execute(Finch myFinch, boolean reverse, double scale)
```

   The default behavior for this method is to simply call: **execute(Finch myFinch)**. Note: think carefully about how to simply implement this default behavior.

2. Create a new class, **FinchMoveScaled**, that extends **FinchMove**. In **FinchMoveScaled**, provide an implementation of:

```
   execute(Finch myFinch, boolean reverse, double scale)
```

   This implementation will be essentially the same as the original **execute(Finch myFinch)** implementation, except: the commanded wheel velocities are equal to **FinchMoveScaled.velocity * scale** (note that the **reverse** parameter is note used).

3. Add **FinchMoveScaled** to your text file parser.

4. In **FinchActionList**, provide an implementation of:

```
   execute(Finch myFinch, String name, boolean reverse, double scale)
```

   As the individual actions are executed, these parameters are passed to each of them.

5. In **FinchModel**, provide an implementation of:

```
   execute(String name, boolean reverse, double scale)
```

   This method should now be called as a result of pressing one of the *Execute* buttons in the GUI.

Milestone 2: FinchMeta

Implement a **FinchMeta** action class that extends **FinchAction**. Unlike our other FinchActions, FinchMeta is a "meta action": executing it results in the execution of a sequence of FinchActions. The FinchActions to be executed are those in the **master FinchActionList** whose names match the **filterName** string.

For this class:

- Property filterName is a String.
- Property scaleFactor is a double.
- Property model is a reference to the **FinchModel** instance.
- Provide one constructor that accepts the name, priority, filterName, scaleFactor and a reference to your **FinchModel** object.
- Define a constant **scaleMin** that specifies the smallest allowable **FinchMeta** object to execute.
- Provide the *execute* methods for this class (the one with *scale* is the most important).
- Add **FinchMeta** to your text file parser.

The specific rules of execution are:

- Display the string "**Entering FinchMeta: Scale = " + scale
- Terminal case: the specified **scale** is less than **minScale**. If this is the case, then no child actions will be executed.
- Recursive case: the specified **scale** is greater than or equal to **minScale**. If this is the case, then the actions of **FinchModel** that match **filterName** are executed, with a scale of **scale * scaleFactor**.
- Display the string "##Leaving FinchMeta: Scale = " + scale

Milestone 3: Add the new FinchAction to ActionDialog

Add the FinchMeta class to your action dialog box. This may require reorganization of your box (I had to go to 2 columns).

Milestone 4: Create Your Own FinchAction Lists

Using meta actions, create (at least) two **interesting recursive** text files. Be ready to demonstrate these.

Hand-In Procedures

Deadline: Thursday, December 9^{th} at 5:00pm

Each individual must do the following:

1. Turn in UML diagram (paper or electronic form).
2. Turn in project5.zip to D2L. This is the zip file produced by Eclipse that contains:
   - Each of the class implementations with documentation.
   - html description of your project produced by javadoc (including private components).
   - Copies of the text files containing your recursive programs.
3. 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. In addition, you may demonstrate during the CS Department's Poster Session on Friday, December 10^{th}, 3:00-5:00 in the Devon Hall first floor atrium.

General Hints and Notes

- Your program must be your own work. Do not discuss or look at the solutions of any other individuals 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.
- Ask for help early. If you are stuck on something, talk to a TA or the instructor sooner than later (this is what we are here for).