Programming Structures and Abstractions (CS 2334) Project 4

November 22, 2010

Introduction

Graphical users interfaces give a user the opportunity to view and manipulate the state of your program. In this project, you will be creating a graphical user interface for viewing, manipulating and executing Finch Actions. This project brings together your work in project 3 and labs 5 & 6.

Your program will maintain a single *master* Finch Action list that will be manipulated by the user and stored by the **FinchModel** class. At any given time, the user may view the entire master list or just those actions that match a specified "filter" name. The master list will be manipulated by the user in several ways. First, the user may load an action list from a text or binary file. For binary files, the user may specify whether the contents of the file replace the existing master list or are either *unioned* or *intersected* with the existing list. Second, the user may manually add new actions or edit existing ones through a dialog box. Finally, the user may delete actions from the master list.

Objectives

By the time you complete this laboratory, you should be able to:

1. analyze and modify the layout of a graphical user interface,
2. use anonymous classes to attach listener objects various swing components, including JButtons and JTextFields,
3. use the listener objects to manipulate the state of the GUI and the data presented by the GUI, and
4. create pop-up dialog boxes that have short-term interactions with a user.

Milestones

1. Create a **FinchModel** class that extends **FinchModelAbstract.** This class contains the master action list and provides a set of methods that will allow the GUI code to manipulate the list. (15 pts)
2. Create the GUI layout for the main window. This class, **FinchFrame,** will extend the **FinchFrameAbstract** class. (5 pts)
3. Add action listeners to the main window that allow the user to manipulate the action list. These actions include file manipulation, name filtering, execution, and deleting individual actions. (15 pts)
4. Create the layout for the pop-up dialog box that allows the user to view/edit a single Finch Action. This class, **FinchActionDialog,** will extend **FinchActionDialogAbstract.** (10 pts)
5. Add the action listeners to the dialog box that allow the user to edit the Finch Action. In addition, add the necessary functionality that will allow the dialog box to create a new Finch Action object and insert it into the master action list. (15 pts)

Other components:

- Develop and use a proper design (15 pts total):
    1. Draw the layout of the two GUI windows. Include all components contained within the windows (including the non-rendered ones, such as JPanels). For each component, indicate its type and (if known) the reference variable name.
    2. Develop a UML diagram that shows the relationships between the classes. Only provide details for the **FinchModel** and **FinchModelAbstract**.
- Use proper documentation and formatting (javadoc and in-line documentation) (15 pts total)
- Perform a short demonstration of your work (10 pts)

Note: of the design and documentation components, a total of 15 points are available during the design phase (7.5 points are available for each of the instance diagram and code stubs). The remaining 85 points are obtainable for the final submission of the project.

Other notes:

- This project is **much** more complicated than any of the others to date. Start early.
- You must extend the provided abstract classes (**FinchModelAbstract, FinchFrameAbstract** and **FinchActionDialogAbstract**). These abstract classes 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.

This lab is due in two phases:

1. Thursday, November 11th at 5:00pm: design
2. Monday, November 29th at 5:00pm: completed program and short demonstration. The early deadline, for a 5% bonus, is Tuesday, November 23rd at 5:00pm.

More details for what to hand-in and when may be found below.

Resources

Main web page / projects / project4

- project4.pdf: this project description
- project4-slides.pdf: a copy of the lab section slides
- Abstract class definitions: FinchFrameAbstract.java FinchActionDialogAbstract.java and FinchModelAbstract.java
- FinchDriver.java: a sample driver class with some testing materials
- FinchActionType.java: an enumerated data type over all possible concrete **FinchAction**s.
- LoadType.java: an enumerated data type over all possible binary loading possibilities (replace, union, intersect).

Input Files

We will keep the same input text file format as with the previous project.

Milestones

Milestone 1: FinchModel Class

Your program will maintain a single *master* Finch Action list that will be manipulated by the user. Your **FinchModel** class will provide this functionality and must extend the **FinchModelAbstract** class. This class will provide:

- A set of methods for manipulating the master action list. It is through these methods that your **FinchFrame** class will manipulate the action list.
- A set of methods for interacting with the Finch. Note: the other classes **will not** touch the Finch directly.

Notes:

- You may need to introduce private helper methods (in particular, you have already implemented some of these in previous projects and labs).
- You should not have to introduce any other public methods.
- You must provide a **Milestone1.java** class that will test the public methods provided by FinchModel.
- You may find it useful to move some of your project 3 driver class methods (e.g., file reading) over to your **FinchActionList** class. Then, **FinchModel** provides much of its functionality by simply making the appropriate calls to methods in **FinchActionList**.

Milestone 2: FinchFrame Look-and-Feel

The primary GUI window will provide a view of the master action list and a means for the user to manipulate the list. This GUI will include the following components:

- A menu for file manipulation. This menu will allow the user to select text files to read and binary files to read/union/intersect/write.
- A name filter panel that accepts typed text. If nothing is specified, then all actions in the master list will be displayed by the window. If a non-empty string is specified, then only actions whose names that match the string will be displayed.
- Within the above panel, buttons for executing the filtered action list in forward and reverse orders.
- A panel that contains the current (filtered) set of actions (one action per line). Note that we are using a **JList** class to display this list.
- A set of buttons for manipulating individual finch actions (right hand side of the window):
  - creating new actions,
  - editing a selected action, and
  - deleting one or more selected actions.

For this milestone, complete the parts of the constructor that will create all of the graphical components. Complete this before moving on to the next milestone. Note: for testing, you should create a small example FinchActionList to display.

Milestone 3: FinchFrame Behavior

Provide the full implementation of FinchFrame, including:

- Implementations of all of the abstract methods.
- Creation of event listeners that will handle the various events.

Note: you should have this milestone completely working (except for the edit/new action buttons) before moving on to the next milestone.

Milestone 4: FinchActionDialog Class

The **FinchActionDialog** class will extend the **FinchActionDialogAbstract** class and will provide a pop-up dialog box that will allow a user to create a new Finch Action or to edit an existing one.

In particular, the behavior will be as follows:

- If the user clicks on the **New** button in the main window, then a dialog box with a reasonable set of default values will be opened. Once the user completes the specification of the action and clicks on the **Ok** button in the dialog box, then this action will be added to the master action list.
- If the user selects an action in the main window, and then clicks on the **Edit** button, the dialog box will be initialized with values that reflect the selected action. Once the user completes the specification of the action and clicks on the **Ok** button in the dialog box, then this action will be added to the master action list and the old action will be removed.
- If the user clicks on the **Edit** button without selecting an action, then the interface should behave as if the **New** button was pressed.
- If the user clicks on the **Cancel** button in the dialog box, then the dialog will close, but no changes will be made to the master action list.

For this milestone, complete the parts of the constructor that will create all of the graphical components.

Hints:

- The dialog box will contain components for all possible properties for any action. For the purposes of this milestone, simply display all of these components (we will control the visibility of the different components in Milestone 5).

Milestone 5: FinchActionDialog Behavior

Provide the full implementation of **FinchActionDialog,** including:

- Implementations of the abstract methods.
- From within the constructor, creation of event listeners that will handle the various events.

Hints:

- We will create one instance of this pop-up dialog box at the beginning of execution. The "pop-up" will be implemented by turning on the visibility of the frame when the edit/new buttons are pressed and turning off the visibility when either the ok/cancel buttons are pressed.
- Depending on the selected action type, a subset of the components will be displayed. Which components are visible can be configured using the setVisible() method for any graphics component. Note that if a container is set to be not visible, then all of its children are not rendered. As a result, if a **JPanel**'s visibility is turned off, then all of its children will not be visible.
- **FinchActionDialog** *should not* interact with **FinchModel** in any way. Changes to the master action list should be handled only by **FinchFrame**. The helper method **FinchFrameAbstract.openDialogAction()** will handle the process of opening the dialog box and responding to the selected action.
- While the dialog box is open, the property values of the selected action should be stored in the various graphical components. As you are opening the dialog box, the values associated with a specific action will be copied into the graphical components. As the dialog box is closing, a new action will be created (assuming that "OK" is selected).

Hand-In Procedures

Part 1: Design

Deadline: Thursday, November 11th 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, a plan for which group member will implement which classes, and any outside citations.
2. An *layout diagram* that shows the layout of the different graphical components for each of the windows.
3. A UML diagram of all of the involved classes. Only provide details for the **FinchModel** and **FinchModelAbstract** classes.

4. project4_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). **If you are re-using methods from project 3, then it is okay to leave these implementations intact. However, for any new methods, only include dummy return calls or calls to this() or super(), and not any other code.**
    - html description of your project produced by javadoc. Make sure to check that the resulting html files contain all of the correct information.

Part 2: Complete program and short demonstration.

Deadline: Monday, November 29th 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.
2. Turn in an updated version of the layout and UML diagrams.
3. Turn in project4.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).
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. In order to demo, both group members should be present, and you must have the original (or copy) of the cover sheet and your **graded** design document.

General 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).
- You must provide a Milestone1.java file that tests your **FinchModel** class. However, the other milestones should be tested with a user "in the driver's seat."