# Programming Structures and Abstractions (CS 2334) Project 3

October 7, 2009

## Introduction

In this project, you will be applying what you have learned about the Collections framework and about Serialization.

## Milestones

1. Use an **ArrayList** to represent **FinchActionList** (15 pts)
2. Show/execute FinchActions in both natural and reverse order (10 pts)
3. Add user command **write** that writes the FinchActionList out to a binary file (10 pts)
4. Add user commands **read** and **merge** that read a FinchActionList from a binary file (15 pts)
5. Add user command **clean** that removes the duplicate entries from FinchActionList (10 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)
- Use proper constructors: all constructors must check for valid parameters. If an incorrect value is given, then it is reasonable at this time to set the property to a reasonable default. (10 pts)

Note: of the design and documentation components, a total of 20 points are available during the design phase (10 points for each). The remaining 80 points are obtainable for the final submission of the project.

This lab is due in two phases:

1. Thursday, October 15th at 5:00pm: design.
2. Thursday, October 22nd at 5:00pm: completed program and short demonstration.

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

## Resources

Main web page / projects / project3 :

- project3.pdf: this project description
- cs2334_project3.pdf: a copy of the lab section slides
- *.txt: example input files

## Input Files

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

## Milestones

### Milestone 1: ArrayLists to represent FinchActionLists

Change your FinchActionList implementation so that it uses ArrayLists (instead of an array of FinchActions). Specifically, your FinchActionList class should be declared accordingly:

```
public class FinchActionList extends ArrayList<FinchAction> implements Serializable{...}
```

Notes:

- You will need to alter the set of properties that are explicitly contained within the FinchActionList (most of the properties will be handled by the ArrayList class). In addition, FinchActionList will need to provide fewer methods.
- FinchAction will need to implement **Serializable**:

  ```
  public abstract class FinchAction implements Comparable<FinchAction>,
                                               Serializable{...}
  ```

- Very few changes will be required outside of your FinchActionList class. One thing you will notice is that Eclipse will raise a warning about not defining **serialVersionUID**. This is a static variable that is used to check the compatibility between the current class definitions and the files. To make this warning go away, add the following to FinchActionList and all of your concrete child classes of FinchAction:

```
static final long serialVersionUID = 1138L;
```

The value that you use is not particularly important for our purposes...

## Milestone 2: Show/execute FinchActions in reverse order

In your FinchActionList class, alter the implementations of the display() and execute() methods so that they adhere to these prototypes:

```
public void display(String name, boolean reverse) {...}

public void execute(Finch myFinch, String name, boolean reverse) {...}
```

The **reverse** parameter indicates whether the FinchActions should be displayed (or executed) in natural (**false**) or reverse (**true**) order.

Finally, in your driver class, allow the user to indicate whether to show/execute FinchActions in natural or reverse order.

Hint: look at the functionality provided by the **Collections** class.

## Milestone 3: Add a "write" user command

Add the following method to your driver class:

```
public static void WriteList(String fname, FinchActionList list, String action_name);
```

This method will open the specified file as an **ObjectOutputStream**, write the elements of **list** that match **action_name** to the file, and close the file.

Hints:

- Remember that FinchActionList already provides the facilities to create a list that matches a String name.
- See your text file parser for examples of how to catch exceptions.
- You will only have to write a single object to the file.

Give the user the ability to execute a **write** command from your command line interface. The user can optionally specify the name with which to filter. Examples:

- If the user types "write foo.dat" at the command line, then all FinchActions in the existing FinchActionList will be written to the file "foo.dat"
- If the user types "write foo.dat dance" at the command line, then all FinchActions with a name of "dance" in the FinchActionList will be written to the file "foo.dat"

## Milestone 4: Add the “read” and “merge” user commands

Add the following method to your driver class:

```
public static FinchActionList ReadList(String fname);
```

This method will read a FinchActionList from the specified file. If there is an error, the method should return **null**.

Give the user the ability to execute **read** and **merge** commands from your command line interface. In both cases, the user specifies a file name. Examples:

- If the user types “read foo.dat” at the command line, then the newly read FinchActionList will replace the existing one.
- If the user types “merge foo.dat” at the command line, then the newly read FinchActionList will be added to the old one.

Hints:

- See the **List** interface for useful methods.
- As part of the merge process, the merged FinchActionList must be re-sorted.
- Test milestones 3 and 4 together by performing sequences of writes, reads and merges.

## Milestone 5: Add a “clean” command

Add to the FinchActionList class:

```
public void clean();
```

This method removes duplicate entries from the current FinchActionList.

Hints:

- Duplicates can be found using the **compareTo()** method.
- You may assume that the FinchActionList is already sorted. Therefore: duplicates must occur next to each other in the list.
- Test this component by “merging in” the same binary file multiple times (this will give you duplicate entries). An execution of **clean** should then remove these duplicates.

## Hand-In Procedures

### Part 1: Design

Deadline: Thursday, October 15th 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. The diagram must contain all of the relevant classes. However, **only detail the FinchActionList class (class variables and methods)**.
3. project3_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 2, 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: Thursday, October 22nd 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. 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. The requirements are the same as for the design phase.
3. Turn in project3.zip to D2L. This is the zip file produced by Eclipse that contains:
   - Each of the class implementations with documentation. The author(s) of each class should be documented at the top of the java file.
   - 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.

## 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).