CMPSC 111
Introduction to Computer Science I
Fall 2014

Lab 9 for Sections 03 and 04
6 November 2014
Due Thursday, 13 November by 2:30pm

Objectives

To further enhance your experience with designing, implementing, and enhancing Java methods, including the completion of tasks such as creating and calling methods that use boolean expressions, complex conditional logic, and iteration constructs. Additionally, to practice using Java methods that read input from a text file with the `java.util.Scanner`.

General Guidelines for Labs

- **Work on the Alden Hall computers.** If you want to work on a different machine, be sure to transfer your programs to the Alden machines and re-run them before submitting.
- **Update your repository often!** You should `add`, `commit`, and `push` your updated files each time you work on them. I will not grade your programs until the due date has passed.
- **Review the Honor Code policy.** You may discuss programs with others, but programs that are nearly identical to others' will be taken as evidence of violating the Honor Code.

Reading Assignment

To continue to learn more about "`if/else if/else`" statements and boolean expressions, please again review Sections 5.1–5.3. Since this assignment will also require you to continue to use Java classes and methods, you should once again review Sections 4.1–4.5. To best prepare for the new content in this laboratory assignment, you should also study Sections 5.4–5.6, paying particularly close attention to the material about `while` loops, `break` and `continue` statements, text file input, iterators, and the `java.util.ArrayList` class. Students who are not familiar with text-based todo list management tools are encouraged to review, as an example, `http://todotxt.com/`.

Create a New Directory and Starting the Project

After changing into the "`cs111F2014-share/`" directory, which contains our course's version control repository, you should type the command "`git pull`" to download the source code for this laboratory assignment. In your own "`cs111F2014-<your user name>`" repository inside the "`labs/`" directory, create a directory called "`lab9`". Using the method described in a previous laboratory assignment, please copy the source code from the share repository to your own repository. Now, change into the "`labs/lab9/`" directory, in your own Git repository, and use "`gvim`" to study the source code of the provided files. What methods do these classes provide? How do they work?

The Java classes that you have downloaded provide the preliminary features for a complete todo list manager program. As part of this assignment, you will document all of the provided source code

and then add new methods that furnish new features. Then, you will run this program multiple times to demonstrate the correctness of the finished system. To start this assignment, it is a good idea to draw diagrams like the ones in Figures 4.4 and 4.7 of the textbook to understand the classes involved in the program and the flow of control between method invocations. Next, to understand the structure of the `TodoList` class, you may want to use the Lightweight Java Visualizer (LJV), as explained in a previous assignment, to create a diagram of its structure in the computer's memory.

Understanding the Todo List Manager

The current implementation of the todo list manager reads from a file called "`todo.txt`", an example of which is included below this paragraph. An individual line in the "`todo.txt`" file always adheres to the format "Priority, Category, Task" where "`A`" is the most important priority (with "`B`" being the next level, and so on), "`Understand`" being an example of a category, and "`Use the LJV to see the TodoList`" is a task. Following this format for a task, you should consider adding in all of items that you must complete in order to successfully finish this laboratory assignment. That is, you can actually use your todo list manager to assist you as you complete both this project, later practical and laboratory assignments, and the upcoming final project!

```
A,Understand,Draw diagram(s) to explain classes
A,Understand,Use the LJV to see TodoList
B,Explain,Add comments to all of the Todo classes
```

Implementing New Features

The final version of your todo list manager should provide features to read the todo list, search for specific tasks according to both priority and category, mark a task as done, and list all of the current tasks. Currently, the system does not include the source code to implement the priority-search and category-search features. Yet, you can see from the following output that the todo list manager can already read the todo list from the file, mark a task as done, list the existing tasks, and stop running the program. Can you also compile and run this program and produce this output?

```
Welcome to the Todo List Manager!
What operation would you like to perform?
Available options: read, priority-search, category-search, done, list, quit
read
list
0, A,  Understand,  Draw diagram(s) to explain classes, done? false
1, A,  Understand,  Use the LJV to see TodoList, done? false
2, B,  Explain,  Add comments to all of the Todo classes , done? false
done
What is the id of the task?
1
list
0, A,  Understand,  Draw diagram(s) to explain classes, done? false
1, A,  Understand,  Use the LJV to see TodoList, done? true
2, B,  Explain,  Add comments to all of the Todo classes , done? false
quit
```

To complete this assignment, you are responsible for adding all of the source code that is needed to implement the priority-search and category-search features. This means that you will first have to add code that can determine when the user has input the word "`priority-search`" or "`category-search`"—what file should contain this code? Please notice that you will need to finish implementing the methods that perform these operations! Both of these operations will involve you using a `java.util.Iterator` to iterate through all of the instances of the `TodoItem` class.

When you are performing a priority-search, you will need to collect and return all of the `TodoItem`s that match the provided priority level. For instance, using the example todo list on the previous page, a request for the "`A`" priority tasks would return those with `id` values of zero and one. Similarly, the use of the category-search operation will require you to iterate through all of the `TodoItem`s managed by a `TodoList` as you find those that match the requested category. Again using the aforementioned example todo list, this operation would also return the first two tasks. The `markTaskAsDone` method, as shown below, gives a concrete example of how to iterate through the `todoItems` and use conditional logic to check if a specific `todoItem` has the requested `toMarkId`. You can use this method as an inspiration for those methods that you must implement.

```
public void markTaskAsDone(int toMarkId) {
    Iterator iterator = todoItems.iterator();
    while(iterator.hasNext()) {
        TodoItem todoItem = (TodoItem)iterator.next();
        if(todoItem.getId() == toMarkId) {
            todoItem.markDone();
        }
    }
}
```

Required Deliverables

For this assignment you are invited to submit versions of the following deliverables through both the Bitbucket repository and in a printed and signed format.

1. Completed, fully commented, and properly formatted versions of the three source code files.
2. An output file, called `output`, containing outputs from five runs of `TodoListMain`.
3. The final version of the `todo.txt` file that you used to both test and complete this lab.
4. A written reflection, saved in `reflection`, about the challenges you faced during this lab.

In addition to turning in signed and printed copies of your code and output, share your source code and other required files with me through your Git repository by correctly using "`git add`", "`git commit`", and "`git push`" commands. When you are done, please ensure that the Bitbucket Web site has a "`lab9/`" directory in your repository with the three Java files in the list of deliverables and the other files. Please see the instructor if you have questions about assignment submission.

In adherence to the Honor Code, students should complete this assignment on an individual basis. While it is appropriate for students in this class to have high-level conversations about the assignment, it is necessary to distinguish carefully between the student who discusses the principles underlying a problem with others and the student who produces assignments that are identical to, or merely variations on, someone else's work. Deliverables that are nearly identical to the work of others will be taken as evidence of violating Allegheny College's Honor Code.