CMPSC 112
Introduction to Computer Science II
Spring 2015

Final Project: Automated Analysis of Todo.txt Files

Introduction

Todo.txt is a file format and a suite of tools that supports the management of todo lists and the completion of project tasks. Originally developed by Gina Trapani, Todo.txt currently includes an ecosystem of tools, such as mobile apps and desktop applications, that enable individuals and teams to manage projects. In this final assignment, you and your team members will use Todo.txt to track your progress as you build a tool that can automatically analyze Todo.txt files.

Current Todo.txt tools (e.g., todo.txt-cli, todo.txt-vim, and the Todo.txt Android app) enable a user to input and mark as completed a wide variety of tasks. Yet, right now there is no tool that can automatically analyze an existing Todo.txt file and determine how the time of an individual or a team was spent. You and your team members will implement a complete tool for the automated analysis of Todo.txt files. For instance, your tool should be able to determine how many high-priority tasks you recently completed. In addition, your tool should be able to ascertain how many tasks were completed in the different contexts and projects represented in a certain Todo.txt file.

As you specify, design, implement, test, debug, demonstrate, and release your Todo.txt analysis system, you and your team members will develop your knowledge and skills in computer science as you learn more about advanced command-line argument management, the reading and parsing of an input file, and the analysis of frequencies with the hashtable abstract data type. You will also learn more about the practice of real-world software development as you work in teams and manage the schedule of your project. Finally, you will hone your writing and speaking skills as you document, present, and release your working tool for the analysis of Todo.txt files.

Understanding and Using Todo.txt

Before you start to specify, design, implement, and test your system, you need to learn more about the Todo.txt ecosystem that is described at `http://todotxt.com/`. You and your team members should study and discuss this site so that you understand the philosophy behind the Todo.txt system. To gain more experience with using the Todo.txt system, each of your team members should install and use the todo.txt-vim plugin that is available for download from the `https://github.com/freitass/todo.txt-vim` Web site. If you and your team want to get this plugin fully working, you will need to add a `Bundle` statement to your `.vimrc` file and then run the `BundleInstall` command in a separate instance of Vim. To ensure that Vim always enters into todo-mode when you edit a Todo.txt file, you should also add the following line to your `.vimrc`:

```
autocmd BufNewFile,BufRead [Tt]odo.txt set filetype=todo
```

Once you have properly configured Vim, you are ready to investigate the use of a Todo.txt file. After creating a directory in your account called `todo`, you should edit a file called "`todo.txt`". Before you start to add content to this file, you need to learn more about the Todo.txt format, which is described at `https://github.com/ginatrapani/todo.txt-cli/wiki/The-Todo.txt-Format`.

Students who would like to study an example of an existing Todo.txt file can view the one available at `http://todotxt.com/todo.txt`. Adding a new task to your todo list is as simple as starting to type in the Vim text editor. Moreover, you can mark a task as completed by using the command `<leader>D` and you can sort all of the tasks in your Todo.txt file by typing `<leader>s`. Before you finish this phase of the final project, please make sure that all of your team members understand how to use the priority, context, and project annotations supported by the Todo.txt format.

Advanced Command-Line Argument Handling

In past laboratory assignments, you often handled command-line arguments by writing code that would inspect the values in the `args[]` array that is a parameter to the `main` method. However, the programs that we have used during our class and laboratory sessions handled the command-line arguments in a brittle fashion: they required the command-line arguments to appear in a fixed order and they did not allow for the specification of arguments in a fashion similar to that used by most programs run in the Linux terminal window. Thankfully, there is a third-party Java library, called JCommander, that makes it easier to perform advanced argument management.

Since we want our Todo.txt analysis tool to have a full-featured and easy-to-use command-line argument system, we will use JCommander for this final project. You and your team members should learn how to use JCommander by visiting the following Web sites: `http://jcommander.org/` and `https://github.com/cbeust/jcommander/`. You can find a recent version of the JCommander JAR file by running the "`git pull`" command in the `cs112S2015-share/` repository. After studying JCommander's user documentation, discussing the library with your team members, and resolving any questions by talking with the course instructor, your team should consider trying a simple example with JCommander. Can you display a help menu and recognize the arguments?

Reading and Parsing Todo.txt Entries

As described in Section 1.6 of your textbook, the `java.util.Scanner` class provides methods, like `hasNextLine` and `nextLine`, to read in lines from a text file. Since the Todo.txt file is stored in plain text, you and your team members can use an instance of `java.util.Scanner` to input all of its lines into your program. Then, your program must break each line of text into its constituent parts. For instance, suppose that the program has just read in the following line from the file:

(A) @Teaching +112 Write the final project assignment sheet

The string tokenization process would separately return each part of any string that adheres to the Todo.txt format, identifying "(A)", "Teaching", "+112", and the other remaining words. According to Todo.txt terminology, which one of these is the priority? Which is the context? And, finally, which of these is the project? To automatically answer these questions, you and your team members may decide to investigate the use of a regular expressions and methods, like the one called `findInLine`, that are also provided by the `java.util.Scanner`. In addition, you may want to look into the use of the Java class called `java.util.StringTokenizer`. Finally, students who want to learn more about regular expressions can consult online references such as the following:

- `http://docs.oracle.com/javase/tutorial/essential/regex/`
- `http://www.vogella.com/tutorials/JavaRegularExpressions/article.html`

Frequency Analysis with Hashtables

Your Todo.txt analysis system should be able to answer a wide variety of questions. For instance, it should be able to determine, for each of the priorities inside of the Todo.txt file, how many tasks are assigned a specific priority. In addition, the tool should be able to automatically determine, for each of the contexts found in the Todo.txt file, how many tasks are associated with a certain context. Finally, the analyzer should determine, for all of the projects given in the Todo.txt file, how many of the tasks are connected to each of the projects. Since many other analyses are also possible, you and your team member should brainstorm ideas and ensure that your final system implements at least five that will enable users to determine how their time is spent.

Of course, it is important to determine what data structure your program will use to store the specific priorities, contexts, and projects. One candidate that your team should consider is the hashtable, as described in Chapter 10 of your textbook and implemented in Java by the `java.util.Hashtable` and `java.util.HashMap` classes. After you and your team review the content in this chapter and study online tutorials you will see that the hashtable enables you to store key-value pairs. In the context of this final project, a key may represent, for instance, a task priority, and the value would correspond to the number of tasks assigned to that priority. Students who wish to learn more about how hashtables can support the counting of frequencies should examine Section 10.1.2 of the textbook and discuss the matter further with their team members.

Summary of the Required Deliverables

To complete this final project, your team should evenly divide the work between its members. In order to support collaboration with your team, please create a version control repository that is named according to the convention `cs112S2015-<your team number>` and share this repository with each other and the course instructor. On the last day of class, April 27, 2015 your team should give a five to eight minute lightning talk that quickly describes your system and gives a brief public demonstration. Finally, you must finish and release your Todo.txt analyzer and submit signed and printed versions of the following deliverables no later than 5 pm on May 5, 2015.

1. The Todo.txt file(s) that your team members used to manage the completion of this project
2. Using examples, a complete description of the features provided by your Todo.txt analysis
3. Comprehensive user documentation that explains how to run the Todo.txt analyzer
4. A written explanation of how your program uses scanning, parsing, and hashtables
5. The output from five separate runs of your final system, demonstrating it's correctness
6. A justified statement of the worst-case time complexity for at least two methods
7. A privately submitted review of each member of your team, documenting all contributions
8. A personal commentary on the challenges that you faced when completing this assignment

Please note that each team is responsible for completing and submitting one version of this assignment. While it is acceptable for members of different teams to have high-level conversations, they should not share source code or full command lines with each other. Deliverables from one team that are nearly identical to the work of another team will be taken as evidence of violating the Honor Code. Please see the course instructor if you have questions about this policy.