CS 465 - Programming Assignment 2

Kryzstof Kudlak - 800294138
March 28, 2022

Status

To my knowledge, everything works

Programming Environment

Versions:

- LOUD 20.04
- JDK 16
- Gradle 7.4

Libraries used:

- JUnit 4 (for testing only)
- Base Java libraries like java.io and java.util
- Gradle “application” and “java” plugin to create the “run” task

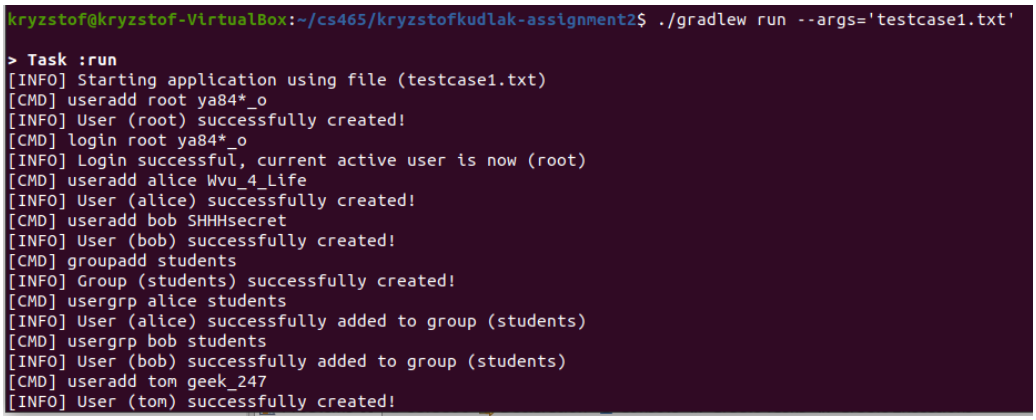
As with the last assignment, I have completed this one as a gradle project written in Java. The directory structure looks like this:

- src/main/java - all the program files & packages
- src/test/java - any test classes I made
- src/main/resources - any classpath resources used by the main application. For this project, I made the test input files classpath resources
- src/test/resources - classpath resources for tests, I did not use this
- build.gradle - file specifying dependencies (I only used JUnit 4 for testing) & run task for command line execution
- outfiles - directory containing any output files generated as a result of executing the program
- static-files - directory containing output files from past executions of the test cases
- gradlew - the gradle wrapper

If you want to compile and run my program from command line, follow these steps:

- cd into the root directory of the project
- Run ./gradlew run --args=’infilename.txt’

Example command line execution:



The specified input file must be located in src/main/resources since it will be loaded in as a classpath resource.

An alternative to command line execution would be to load the project into Eclipse & run it from there. To do so:

- Import as a gradle project. File > Import… > Gradle > Existing Gradle Project > Next > > Browse… > select the root directory of the project > Finish
- Right click on the Access.java file in the main package, mouse over Run As, then click Run Configurations…
- In the popup window, click Arguments and specify the file name in the input box labeled “Program arguments:”
- Click Run

The main method of my program exists in Access.java.

Also here’s the GitHub repo with my code:
https://github.com/kkudlak45/cs465-programming-assignment-2

Program Description

My project has the following packages in src/main/java:

- kryzstof-kudlak.assignment2 - main package, contains Access.java and Driver interface
- ‘’.drivers - contains all the command drivers
- ‘’.logger - contains EasyLogger
- ‘’.objects - contains POJOs for File, User, and Group
- ‘’.services - contains services to manage collections of the POJOs in ‘’.objects
- ‘’.utils - contains constants files, probably should’ve named it constants but whatever

My project structure somewhat emulates the layout of the CS 210 project. I have a driver for every command. Each driver extends the IDriver class and has a parse method which accepts a command string. This method will return true or false depending whether it is able to parse the command using a regular expression. If it can parse the command, the driver calls its execute function which takes care of the necessary logic to fulfill that command.

The main method has a static list of all the drivers. With every line the main method reads in from the input file, it sends that as a command through all of the drivers until one is able to parse it. If no driver can parse the command, it is assumed that this was a malformed command, an error is reported, and the next command is moved on to.

The services package maintains the state of the program. There is a service for each POJO in the objects package (FileService, GroupService, UserService). Each service has an underlying static list of instances of its designated object as well as a bunch of static methods to perform operations on this list. The user service has an extra static field to store its active user (the logged in user). The services are managed by the drivers which cause state changes within each service. The services manage & use the POJOs by storing them.

EasyLogger is a class I created which has a bunch of static methods that log output to both files and stdout at the same time. EasyLogger also has the capacity to append data to arbitrary files. Easy logger stores a map of filenames to output streams. When EasyLogger is told to write to a file, it first checks to see if it already has an output stream for that file. If there’s already an output stream that exists, it uses that, otherwise it creates a new output stream for that file (thus overwriting whatever was already there). EasyLogger is what I used to output pretty much everything.

Properties like who owns a file and what group a user is in are stored within the POJOs. These objects also have fields like name, password, etc which are updated and accessed using getters & setters. Additionally, the File object has methods which take in a user & return whether that user can read/write/execute.

I implemented file permissions such that the level of access which grants the most amount of privilege is what’s given to the user. For example, if a user both owns a file and is in the group associated with the file, but the file’s permissions are - - - - - - rwx, then the user will

still retain rwx access since that’s the global level of access. In other words, permissions are not exclusionary.

Testing

I have 4 test cases in src/main/resources named testcase[1-5].txt. I also made a unit test for every driver with a test for each bullet point in the grading doc.

testcase1 - given to me
testcase2 - given to me
testcase3 - tries to run a first command that isn't useradd root [password]. Should crash the program
testcase4 - I wrote a bunch of random commands to test file access, read/write/execute, changing file permissions, etc. I pretty much just stream of consciousness wrote whatever came to mind as sort of a smoke test/integration test since I've already unit tested everything.
testcase5 - pretty much every command has a requirement that a user must be logged in in order to run it. In this file, I just ran commands with nobody logged in.

Output for each of these test cases can be found in its respective directory within the static-files/ directory in the root of the project.

I will also again note that the majority of my testing efforts were focused on creating unit tests within the src/test/java directory.

Extra

To address some things that are specifically laid out in the document so that I can get points:

accounts.txt format is in the form:

```
username:password:group names separated by spaces
```

I chose spaces as a delimiter between groups because they are disallowed in group names. I chose to separate the types of fields by ":" because that's what unix systems use in their /etc/passwd file.

The underlying data structures that I maintained for groups and files were LinkedLists. I'm sure I could have used a set for this, but lists are more natural to me. These lists were managed by FileService and GroupService respectively.