Assignment 3: The Software Architecture

Software applications that are considered successful tend to have a strong and resilient architecture in place. A software architecture defines the structure of the application and lays the foundation for future enhancements. A successful software architecture will allow for:

- The ability to easily plug or unplug features. When I add a new feature to my application, it should feel like it easily fits in the architecture without any hassle or magical hacks to make it work.
- Loose coupling of features – meaning a new action can be plugged in or an action can be unplugged from the application without the need to refactor/rewrite a big portion of the architecture.

In a typical shop, senior level developers or developers in an architectural role define and create the architecture for the application. Those on the team then contribute to the application by adhering to the architecture.

Starting with assignment three, you will be expected to contribute to an existing architecture. Assignment three starts your journey into the LinkedIn command line interface (CLI) application. This application will allow the user to perform actions that can be launched from a menu.

None of the assignments expect you to change any source code in this architecture, however, you do need to understand it enough to know how to contribute to it. This document will explain the architecture and how you will be using it to build your LinkedIn application.

LinkedIn Command Line Interface (CLI)

edu.institution.LinkedInCLI

In lieu of a graphical user interface, we will be interacting with your application from the command line. The *edu.institution.LinkedInCLI* class contains the *main* method that will launch the application. The *main* method initializes the user repository with the file path and file name to the file that will store our LinkedIn users. That file will live in a folder called, *Java2,* located off of your home directory.

```
// The system property, "user.home", returns the file path to
// the home directory on the computer that this is running on.
private static final String PATH =
  System.getProperty("user.home") +
  File.separator + "Java2" + File.separator;

private static final String FILE_NAME = "LinkedInUsers.dat";
```

To launch the LinkedInCLI, right-click on the edu.institution.LinkedInCLI class from within the Project Explorer and choose the *Run As -> Java Application* option.

```java
/**
 * The command line interface for LinkedIn.
 */
public class LinkedInCLI {
    private static String PATH = System.getProperty("user.home") +
      File.separator + "Java2" + File.separator;
    private static String FILE_NAME = "LinkedInUsers.dat";

    public static void main(String[] args) {
        UserRepository userRepository = new SerializedUserRepository();
        userRepository.init(PATH, FILE_NAME);

        ApplicationController controller = new
           ApplicationController(userRepository);
        controller.process();
    }
}
```

All of the application control is maintained within the *ApplicationController* class.

Application Controller

edu.institution.ApplicationController

This class is contained within the *java2-linkedin.architecture.jar* file. This JAR file is already included in your project and has already been added to your class path. It contains one method, called *process*, which processes the actions that your application presents to the user.

```java
public void process() {
    List<MenuAction> actions = createMenuActions();
    try (Scanner scanner = new Scanner(System.in);) {
        int choice = -1;
        do {
            establishLoggedInUser(scanner);
            displayMenu();

            choice = promptForChoice(scanner);
            if (choice < 1 || choice > QUIT) {
                ApplicationHelper.showMessage("Invalid choice. Try again.");
            } else {
                if (choice != QUIT) {
                    MenuAction action = actions.get(choice - 1);
                    if (!action.process(scanner, userRepository, loggedInUser)) {
                        loggedInUser = null;
                    }
                    userRepository.saveAll();
                }
            }
        } while (choice != QUIT);
    }

    ApplicationHelper.showMessage("Goodbye");
}
```

The *process* method does the following:

1. Creates a list of menu actions to present and respond to. The *Action Architecture* section below goes into detail regarding menu actions. The menu action class names are contained within a file called, *actions.txt*, that exists in the *Resources* folder of your project. This file defines the fully qualified class names to each action that your application will present. You will be creating these actions each week as the semester progresses.

The contents of the *actions.txt* file is as follows:

```
edu.institution.actions.asn3.ListUserAction
edu.institution.actions.asn3.AddUserAction
edu.institution.actions.asn3.DeleteUserAction
edu.institution.actions.asn3.SignoffAction

edu.institution.actions.asn4.ListConnectionAction
edu.institution.actions.asn4.AddConnectionAction
edu.institution.actions.asn4.RemoveConnectionAction
edu.institution.actions.asn4.DegreeOfSeparationAction

edu.institution.actions.asn6.ListUserAlphabeticallyAction
edu.institution.actions.asn6.ListUserByConnectionAction
edu.institution.actions.asn6.ListUserByTypeAction

edu.institution.actions.asn7.AddSkillsetAction
edu.institution.actions.asn7.RemoveSkillsetAction
edu.institution.actions.asn7.ListSkillsetAction

edu.institution.actions.asn10.UndoAction
```

2. The *process* method reads the *actions.txt* file and tries to create an instance of each of the above classes.
3. If any of the above classes do not exist, the *process* method will default to the *NotImplementedAction* implementation. This implementation simply prints a message stating that the action has not been completed.
4. Creates a *Scanner* using the system input (the keyboard) as the input stream.
5. Loops continuously until the user enters the *Quit* option displayed from the menu.
6. Within the loop, an initial root user and password is collected from the user. Subsequent executions will prompt the user to log in with an existing user.
7. The menu is displayed on the console and the user is prompted to enter their menu choice (a number between 1 and 16). Every menu action that we will create throughout this course is displayed on this menu.
8. When the user enters their menu option, the appropriate action is pulled out of the list (created in step 1) and the *process* method on the action is called.

9. If the action returns *false*, then it is telling the Application Controller to log out the current user.
10. The *saveAll* method on the user repository is called after the action returns to save any data modified in the action.
11. Steps 4 – 9 are repeated until the user chooses the Quit option.

Action Architecture

edu.institution.MenuAction

The application was architected in a manor where *actions* can be plugged in with ease. An action is any class that implements the *MenuAction* interface. This interface defines a single method, called *process*, which performs the specific logic of the action. This interface is also contained within the java2-linkedin.architecture.jar file. It looks like this:

```
/**
 * Processes a menu action.
 *
 * @param scanner        the scanner accepting user input.
 * @param userRepository the user repository.
 * @param loggedInUser   the logged in user.
 * @return returns true if the logged in user should remain logged in;
 *         false if they should be logged out.
 */
boolean process(
 Scanner scanner, UserRepository userRepository, LinkedInUser loggedInUser);
```

The architecture passes the Scanner instance to be used if the action needs to collect user input. An implementation of the UserRepository interface is supplied if the action needs to retrieve or modify the user data that is stored on the file system. The UserRepository interface is provided for you, however, you have to implement this interface in assignment three. See the UserRepository section within assignment three for the details on this repository. The signed in user is also supplied in the event that the action needs to act against the current user.

The architecture will attempt to create an instance of each MenuAction implementation contained within the *actions.txt* file (see above). The architecture provides one default implementation of the *MenuAction* interface, called *NotImplementedAction*, which is used if any of the actions defined in the *actions.txt* file is not found.

```
/**
 * A place holder menu action displaying that the selection action
 * has not been implemented.
 */
public class NotImplementedAction implements MenuAction {

	@Override
	public boolean process(
	 Scanner scanner, UserRepository userRepository,
	 LinkedInUser loggedInUser) {
		ApplicationHelper.showMessage(
	      "This action has not been implemented.  Check back soon!");
```

```
                return true;
        }
}
```

Each week you will create menu actions. They must implement the MenuAction interface and they must be placed in the defined package and be named exactly as they are defined in the *actions.txt* file.

Application Helper

edu.institution.ApplicationHelper

And finally, there is an ApplicationHelper class that contains common static helper methods with the intent to aid with the command line interface as well as any action that you write. The ApplicationHelper class initially contains one method called, *showMessage*, which prints the supplied message to the console. Feel free to add additional commonly used methods to this class as you see fit.