Project:
System Testing from UML

Initial Document

Author: Thomas Werner
Student Number: 740786
Supervisor: Markus Roggenbach
Contents

1. Introduction
2. Case study: TWLlibrary
3. Testing from UML diagrams in literature
4. UML Development Tools
5. TWLlibrary: UML diagrams
6. TWLlibrary: Implementation
7. TWLlibrary: Testing
8. Case study: Summary
9. Project Management
10. References
11. Appendix
1. Introduction

The Unified Modelling Language (UML) is a widely used modelling language for Object-oriented systems, and is considered the standard modelling approach for software development of this type [2] [6]. The UML’s ability to display system information has been reviewed as having significantly positive impacts on software development in many areas, including system testing [7]. One of these areas is the use of UML diagrams for deriving test suites, though the application of this is largely dependent on the preferences of the software developer(s).

Due to the large number of UML diagram types and the inherit complexity in the UML language, it may be difficult for developers to know the most efficient methods of deriving valid test suites from their requirements [1]. This area could therefore benefit from further research into systematic methods of generating test suites from UML diagrams.

There is a vast amount of literature on the methodology of forming test approaches based on individual UML diagrams [3] [8]. However, when testing an entire system, the integration of these diagram types is often not considered. Our project will implement a concrete system in order to evaluate the test cases derived from various diagram types, and how their interaction can lead to a final test suite for the system.

Overall Project aims:

- Design the requirements for a basic software system and accompany these requirements using UML notation in the form of UML diagrams.
- Implement the designed system in the Java programming language.
- Explore various systematic methods for deriving test cases from UML diagram types, such as class and activity diagrams, using existing work as a basis and influence. These methods should be fully implemented as test cases within the system.
- Evaluate the effectiveness of these approaches, using the original specification and the test suite as indicators of success.
2. Case study: TWLibrary

For the purposes of this initial document, I have undertaken a case study of Object-oriented system testing based in the Java programming language. The system used is a basic, self-contained library system known as TWLibrary, for which I designed the specification and provided the total implementation and testing. The specification was partially influenced by the library example used in “A UML Based Approach to System Testing” (L.Briand, Y.Labiche, 2002) [3], but deviates on the requirements of the system and the available operations. The specification document that I produced for the TWLibrary system is present in the appendix of this document.

This case study had a selection of aims that intended to provide a background of research and materials for the project. The completion of the case study would also advance the skills needed for the project, and allow work on the project to begin at a faster rate.

The case study aims were:

- Construct a java implementation of a small scale system (a library).
- Review UML testing methods in literature, and use this as a basis to develop a simplistic test suite.
- For the specified library system, create Class and Activity UML diagrams (some of the most commonly used types [4]).
- Evaluate a selection of the available UML diagram construction tools, and select a preferable option for the case study and project.

This initial document is devoted to presentation and analysis of the case study and its materials, and thus advances the project as a whole.
3. Testing from UML diagrams in literature

In order to devise an appropriate method of forming a test suite for the TWLibrary case study, I undertook a brief analysis of available research material relating to testing from UML diagrams. To keep the testing methodology within the required scope of the study, it was optimal to review testing from the UML diagrams that are sufficiently abstract from the system implementation. One of the common diagrams type for this purpose is the Activity Diagram. An abstracted example of this is shown in Y.Vanderperren et al. (2008) [9]:

![Activity Diagram](image1.png)

Figure 1 – a literature example of an abstract Activity diagram
Source: “UML for electronic systems design: a comprehensive overview” Y.Vanderperren et al. (2008) [9].

Testing with this diagram type is explored in “A UML Based Approach to System Testing” (L.Briand, Y.Labiche, 2002) [3], using a mock library system:

![Activity Diagram](image2.png)

Figure 2 – An Activity Diagram for a library system.
Source: “A UML Based Approach to System Testing” (L.Briand, Y.Labiche, 2002) [3].
For each action in the activity diagram, represented by a labelled node, L.Briand and Y.Labiche (2002) [3] produce an associated Use Case Description. These descriptions expand on the details of each action by providing information that is not displayed in the Activity Diagram, yet would aid generation of test cases.

**Add Item**

*Use case name:* Add Item  
*Participating actor:* Librarian  
*Parameters:* in ISBN: Integer, out ItemID: Integer  

*Entry condition:*
1. The librarian requests the title information from the librarian terminal. The Find Title use case is used. If the title doesn’t exist, the system will ask the librarian to add the title; the Add Title use case is used.

*Flow of events:*
2. The title information is displayed on the librarian terminal.

*Exit condition:*
3. The librarian adds item, an itemID is generated for it.

Figure 3 - The Use Case Description for the Add Item action.  
Source: “A UML Based Approach to System Testing” (L.Briand, Y.Labiche, 2002) [3].

Using the Activity diagram, it is then possible to generate valid sequences of actions that can be performed by a particular actor, which are known as Use Case Sequences. An example of a use case sequence used by L.Briand and Y.Labiche (2002) [3] from the above diagram would be:

AddTitle(isbn).AddItem(isbn,itemId).RemoveItem(itemId).RemoveTitle(isbn)

Figure 4 - A Use Case Sequence derived from the Activity diagram.  
Source: “A UML Based Approach to System Testing” (L.Briand, Y.Labiche, 2002) [3].

As these Use Case Sequences directly correspond to consecutive actions that should be performable by the system, these sequences can be used as the basis for test cases. For the purposes of the TWLibrary case study, I chose to use these sequences directly rather than progress to the additional steps used by L.Briand and Y.Labiche (2002) [3]. The additional steps would exceed the scope of the case study, and would be more suited to proper analysis during the main project.
O. Pilskalns et al. (2007) [8] demonstrates a method of deriving use case sequences from a different UML diagram type, the UML Sequence Diagram. The first stage of their testing methodology involves creating a Directed Graph based on the Sequence Diagram for their system.

![Directed Graph](image)

**Figure 5** - A UML Sequence diagram.  
Source: “Testing UML designs” O. Pilskalns et al. (2007) [8].

As shown in Figure 6, the Use Case Sequences derived from this diagram time have the potential to immediately be closer to the notation of the final test cases by including specific method names. This level of detail is usually not included in UML Activity diagrams, as they are considered more abstract from the actual implementation of the system.

![Directed Graph](image)

**Figure 6** - A Directed Graph derived from a Sequence diagram.  
Source: “Testing UML designs” O. Pilskalns et al. (2007) [8].
In order to use such a method as in O.Pilskalns et al. (2007) [8] for my case study, it would be required to have complete knowledge of the system implementation before the UML Sequence diagram could be constructed, either through an explicit specification or through having the system already implemented. Due to the scope of the case study, I decided to focus instead on a methodology more similar to that used in the first stages of L.Briand and Y.Labiche (2002) [3]. The absence of specific methods from Activity Diagrams allows them to be more abstract from the system implementation, which was beneficial for the circumstances of the study.

4. UML Development Tools

As I will be producing a variety of UML diagrams for both the final project and my TWLibrary case study, I decided to dedicate a chapter of this initial document to analysing some of the available UML development tools that are available. The tools I looked at are:

**yEd Graph Editor** *version 3.13*, produced by yWorks [10]

**Modelio** *version 3.2*, produced by Modeliosoft [11]

**Papyrus UML** *version 1.01*, an Eclipse Modelling project [12]

After review of these programs, I chose to use Papyrus for the case study and the project.

**yEd Graph Editor** [10]

This software consists of a standalone diagram construction program, with various graphic elements being available for use. One portion of these graphic elements is labelled UML, and contains a selection of UML components. Unfortunately, by default yEd only supports elements for Class and Activity UML diagrams.

These elements are treated as purely graphic elements with the ability to support strings, and so there is no checking by the software for correct UML notation. While this does allow for greater flexibility for adding attributes and operations to classes, it
also means that mistakes in UML notation can be made more easily and could go unrecognised. There is also the added benefit of being able to quickly produce a class diagram from scratch, as attribute types of other classes can immediately be added without the need for any relationship connections. However, this means there is no automatic processing of class relationships, and thus no visual display of relationships between classes unless it is done completely by hand.

Despite a perceived lack of UML functionality, the graphic focus of the UML elements allows for more responsive control of the diagram layout, as elements are often not locked to any predefined position. The software also supports a very useful feature in the form of automatic diagram layouts and grouping algorithms. The automatic layout in particular is an impressive feature, and supports a variety of layout options that reduces the workload of manually setting the layout.

Modelio [11]

In contrast to yEd, Modelio is a standalone UML modelling environment, and so has a much greater degree of intrinsic UML support in all its features. This presents itself immediately in the ability to create specific UML diagram types right from the creation screen, which then present the associated components for addition. All options for editing components are then performed through a properties window, which is strictly controlled according to a correct UML specification.

By default, Modelio supports Class, Activity, State Machine, Use Case, Communication, Development, Object and Sequence diagram types. This is a much greater selection than yEd, but does not cover all UML diagram types.

Modelio uses a grid interface for adding UML components, which aids with diagram clarity in the construction phase and improves the ability to create a desirable layout. The amount of components types for the Class and Activity diagrams was also impressive, and the software was intuitive in the options presented and how classes were constructed.
Papyrus UML [12]

Papyrus is modelling software that exists as a plugin for the Eclipse Integrated Development Environment (IDE). Similar to Modelio, it has intrinsic support for the UML specification and has the ability to produce a number of UML diagram types by default.

The diagram types supported by Papyrus are Class, Activity, Communication, Composite, Deployment, Inner Class, Interaction, Package, Sequence, StateMachine, Timing and UseCase. This covers nearly all possible UML diagram types, and covers any diagram types I would wish to explore during the project.

The method of creating UML diagrams is also similar to that of Modelio, where after a diagram type has been selected its associated components are made available to drag on to a central editing window. Editing of specific attributes for these components is then done through a properties window belonging to each component, which is controlled by proper UML notation. In practice, I found that Papyrus offered a greater amount of options for editing these properties than Modelio, and that the interface better facilitated large systems.

The best example of this is how Papyrus handles display of attributes and operations for Class diagrams, where all added elements are initially hidden and can then be made visible by the user. This allows all attributes and operations to be added to a class, after which private and non-informative elements can be left hidden but still attached for later use. This improves usability when the system is under change, as information can be hidden or revealed as needed to provide a clearer display.

The most important factor in my choice of Papyrus is its integration with Eclipse, which I am using as my Integrated Development Environment for both the java implementation and test cases associated with the TWLibrary case study. The ability to store diagrams as an Eclipse project alongside the implementation, and to view any model alongside the classes it represents is a valuable asset. This functionality also makes moving information between classes and diagrams an easier process.
5. TWLibrary: UML diagrams

For the TWLibrary system, I produced a UML Class diagram for the 7 classes contained in the implementation, and 3 UML Activity diagrams for the 3 actors in my system, a “New User”, a “Standard” user, and a “Staff/Admin” user (which share the same activity).

Although not directly used for testing, creation of this Class Diagram aided with the java implementation of the TWLibrary system. The positive aspects of this process influenced the inclusion of this diagram in my project management section for scheduling the java implementation of the main project.
TWLibrary Activity Diagrams

The “New User” actor, which describes someone who does not have an account created, and thus cannot immediately log in.

![Activity diagram for a New User actor in the TWLibrary system.](image)

The “Standard User” actor, who can use an existing account or create a new one.

![Activity diagram for a Standard User actor in the TWLibrary system.](image)

The “Staff/Admin” actor. These share the same activity and can perform operations on the book list.

![Activity diagram for a Staff/Admin User actor in the TWLibrary system.](image)
These Activity diagrams represent the possible actions in the TWLibrary system for each actor (New User, Standard and Staff). The Admin user coupled with the staff user remains from an original version of the implementation. These two actor types have the exact same set of actions available, making them indistinguishable, but both still have presence in the code and so remain in the diagram for continuity.

Using the methodology described in the ‘Testing from UML Diagrams in literature’ chapter, these Activity diagrams were used to develop the test suite for the TWLibrary system. This process is expanded on in the ‘TWLibrary: Testing’ chapter. The entire test suite is available to view in the appendix of this document.

6. TWLibrary: Implementation

For the implementation of the TWLibrary system, I produced 7 java classes making up the running library system and a testing class using JUnit which then tests these 7 classes. These 7 classes needed to fully incorporate all needed features described in the requirements document (see Appendix). I kept these requirements intentionally basic and broad-scope due to the time constraints of the initial document, and the desire to produce accompanying material such as UML diagrams and test cases.

The full java implementation of the TWLibrary system can be found in the appendix of this document, along with screenshots of the program running in the Eclipse Integrated Development Environment.

After completing the implementation, I would reflect on the chosen requirements of the system as not perfectly ideal for the case study aims. I found the decision to support user logins in the system, and to store book and user lists in the method I chose, disproportionally divided the time I took to complete this implementation. This meant that a large portion of each class was being devoted to performing IO operations, which lead to some classes being bloated and not forming a streamlined system. These operations could also have been better represented in the test cases derived from the Activity diagrams, and could benefit from using different diagram types.
It would have been preferable to handle these operations with a fully incorporated
database system, although implementing such a feature would have been too large
for the scope of this case study. This realisation will be important when designing the
requirements for the software system used for the overall project, as it is likely to
utilise similar operations.

7. TWLibrary: Testing

As mentioned in the introductory chapters, in order to generate test cases for the
TWLibrary system I studied the methodology of forming Use Case Sequences from
an Activity diagram as described in L.Briand and Y.Labiche, (2002) [3]. Rather than
attempt the entire procedure in the article, as this would be suited to the project itself
and a more robust system, I took the first step of identifying sequences from the
Activity diagrams and translating these in to test cases in JUnit. For example, the log
in procedure for a new user is displayed in the New User Activity diagram as:

![Activity Diagram](image)

Figure 11 – A subsection of the New User Activity diagram for the
TWLibrary system.

The following Use Case Sequence was then derived from this:

**Create Account**(username1, password1).**Login**(username1, password1).
This was then developed into the JUnit test:

```java
// User type: New user
// Use case sequence: Create Account(username1, password1).Login(username1, password1).
// Description: A new user attempts to create and account and then log in with that account.
// Expected Result: Actions 'Create Account' and 'Login' both succeed.
@Test
public void NewUserCase1() {
    String testUsername = "username1";
    String testPassword = "password1";

    introWindow.createAccount(testUsername, testPassword);
    assertTrue(activeSession.buildUserList());
    assertEquals("Login attempt", STANDARD_TYPE, activeSession.checkValidUser(testUsername, testPassword));

    try {
        mainWindow.initStartScreen();
        mainWindow.initLobbyScreen(testUsername, testPassword);
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for mainWindow.initLobbyScreen ");
    }
}
```

Figure 12 – A test case in JUnit for the New User actor.

The rest of the JUnit test cases, along with the implementation code and screenshots of the test suite, can be found in the appendix of this document.
8. Case study: Summary

In review of the TWLibrary case study, I can conclude that the original aims set out in the ‘Case study: TWLibrary’ chapter have been met, namely:

- A small-scale library implementation was produced in Java.
- A review of UML testing literature was used to produce a simplistic test suite for the TWLibrary system, and was implemented in JUnit.
- A Class diagram covering all TWLibrary classes was produced.
- 3 Activity diagrams for each actor in the TWLibrary system (New User, Standard, and Staff) were produced and used as a basis for the test cases.
- A review of a selection of UML diagram construction tools was undertaken, and the Papyrus Eclipse plugin was selected for use with this case study and the project.

All resources associated with this case study are available in the appendix of this initial document. I have used the progress of this case study to determine my time management plan which I have detailed in the ‘Project Management’ chapter.

While the case study can be considered a success, I identified some issues with implementation of the system. These included some classes becoming bloated by IO operations due to inefficient structure, which can be accredited to inexperience and a non-specific set of requirements in written form. This should be less of an issue when the main project beings, as it will include a much more extensive planning phase to develop a more robust requirements document. The experience gained from programming this system will also aid with implementation of the final project.

The test suite for this system was also small in scope, and can be assumed to not guarantee the stability of the TWLibrary implementation. However, this was not a critical goal of the case study and will be expanded upon for the project itself.
9. Project Management

Software development

I intend for the software developed during the main project to consist of 4 parts: the hand-written requirements, the UML system diagrams, the Java implementation, and a set of test suites. Each test suite will be directly related to a particular UML diagram type (or small set of diagram types), from which the test suite is derived.

Due to the modular nature of the UML diagrams and test suites, I have decided to use a Spiral model of software development for this project. The intention is that once the requirements and basic implementation are in place, it would be possible to explore a particular UML diagram type and its derived test suites, and then repeat the process until all types have been covered. In this way, each UML diagram, its associated test suite, and related code and system requirements would form distinct ‘prototypes’ as described in the model, and could each form their own development cycle.

Using this model is advantageous, as once I have completed a single development cycle for a UML diagram and its test suite I will have a good indicator of the time requirement for completing one of these cycles, and can update my time management plan accordingly. Dividing ‘prototypes’ in this way also allows me to be flexible with the scope of the project in terms of number of UML diagram types under review, as it will be simple to either add or remove planned diagrams depending whether I am ahead or behind schedule when I reach my planned milestones.

In order for these diagram-based development cycles to function, I will be required to have the Java implementation in place before the majority of UML diagrams are being created. To facilitate this, the first development cycle will be significantly larger than the others, and will focus on the system requirements, UML class diagrams, and the Java implementation. This first development cycle should be nearing completion around the time of my first milestone in early December, which is detailed in the next section.
In order to sufficiently structure my work on this project, I have devised the following milestones in order to judge my progress at certain intervals. These milestones are:

**Milestone One** – Early December 2014

1. System requirements developed in written language.
2. Written requirements developed into a minimum of a UML class diagram to aid with implementation.
3. The implementation is underway, and is nearing a testable form. This testable form will be decided by the ability to create valid test cases using the code, as derived from the UML class diagrams that should already be produced.

By milestone one, I should have the system requirements in place in both written language and also accompanied by a minimum of a UML class diagram, such that implementation has begun. I have singled out the class diagram specifically as I believe this will be a valuable tool for developing the java implementation. As the implementation progresses, I naturally expect the UML diagrams to evolve along with the implementation as the class structure changes. However, I expect the written requirements of the system to remain concrete unless a larger re-evaluation is required, such as if the scope of the system requirements is too large.
Milestone Two – Early March 2014

4. Implementation should be complete for testing purposes, and should not need expanding in terms of features. If implementation is in any way incomplete by this stage, the original requirements may need to be revised.

5. A significant majority of UML diagrams completed along with their test suites.

6. Valid analysis of produced test suites is underway and nearing completion.

By this stage, I should be looking to being writing the Dissertation itself, and so this milestone focuses on the majority of content required for the project being completed.

Risk management

As this is project is focused on software development, one of the principle risks is that of data loss in the event of hardware failure or accidents. This risk is amplified due to the fact that I am the single user working on the code and other resources, and so am using the same hardware and storage devices over an extended period of time. In order to reduce this risk, I have set up a file storage plan to be followed during the project:

- As I will be working on both my home computer and also those in the university, I will be storing all needed files on a USB storage device dedicated to the project. My Eclipse IDE workspace will then be loaded directly from the USB on all machines.
- I will keep a local copy of all project files on both my home machine and also on my university account. These local copies will then be updated whenever I have finished a work session, with whatever files have been changed or added. I expect one of these copies to be up-to-date at all times, with the other being 1-2 days behind.
- In addition to this, I will be transferring the project files to an external hard drive on a weekly basis as an emergency backup, which will not be in use.
Another potential risk in this project is the potential difficulty in interpreting the UML-derived testing approaches from the literature. During the research I performed leading up to this initial document, I have noticed that the methods employed vary significantly between diagrams. This coupled with the inherent complexity of the UML diagrams and the number of diagrams available has the risk of extending the development cycles beyond what is acceptable for my project timeline, especially for the diagram types I have the least experience with.

While this risk is valid, I am hopeful that the modular nature of the software development model I have chosen will allow me the flexibility to alter the diagram types I am investigating. This re-evaluation should be done when nearing the second milestone that I have set in my time management plan.
10. References


Modelling Tools


11. Appendix

The resources associated with the TWLibrary case study are found below.

**UML Diagrams**

**Class Diagram:**

![Class Diagram](image)

**Activity Diagram (New User):**

![Activity Diagram](image)
Activity Diagram (Standard User):

Activity Diagram (Staff User):
The Login Screen

- The system must support the ability to store account details for users.
- Users should have an ‘access level’ that is tied to their account and is unchangeable. This should support AT LEAST the levels STANDARD and STAFF.
- A login screen with inputs for a username and password must be presented.
- The login screen must facilitate user logins, checking against existing user details.
- The login screen must provide functionality to create a new user account of the default user type STANDARD.
- Creating an account with access level STAFF should not be possible from within the system.
- When valid user details are entered, the login screen should progress to the next window with the user’s details selected.

Session window

- The system should store a list of books representing books held by the library.
- This list should be displayed visually in some form on the session window.
- Functionality to add and remove books from this list should be present, provided the user has an account access level of STAFF (or higher).
- The current user’s name and account access level should be displayed somewhere on the panel.
- The book list must have some way of opening a single view for each book in the system (the ‘Book View’).

Book View

- Should display the following information on the panel: Book ID, Book Title, and copies held by the library.
- Functionality to loan the book should be available, with one copy of the book then being assigned to the current user.
- Loaning a book should report an error if there are no available copies of the book left in the system.
Full TWLibrary implementation in Java

CLASS: TWLibraryMain

```java
public class TWLibraryMain {
    private static MainWindow mainWindow;
    public static void main(String[] args) {
        mainWindow = new MainWindow();
        mainWindow.initStartScreen();
    }
    public static void progressToLobby(String user, String pw) {
        mainWindow.initLobbyScreen(user, pw);
    }
}
```

CLASS: MainWindow.java

```java
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JOptionPane;
public class MainWindow {
    private final int INTRO_WIDTH = 800;
    private final int INTRO_HEIGHT = 500;
    private final String INVALID_TYPE = "invalid";
    private final String STANDARD_TYPE = "standard";
    private final String STAFF_TYPE = "staff";
    private final String ADMIN_TYPE = "admin";
    private IntroWindow intro_window;
    private ActiveSession currentSession;
    public void initStartScreen() {
        intro_window = new IntroWindow();
        intro_window.setSize(INTRO_WIDTH, INTRO_HEIGHT);
        intro_window.setLocationRelativeTo(null);
        intro_window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        intro_window.setVisible(true);
    }
    public void initLobbyScreen(String user, String pw) {
        String returnedType;
        currentSession = new ActiveSession();
        if (currentSession.buildUserList() == false) {
            System.out.println("Error in userList construction, terminating...");
            System.exit(0);
        }
        returnedType = currentSession.checkValidUser(user, pw);
        if (returnedType.equals(INVALID_TYPE)) {
            JOptionPane.showMessageDialog(null, "Error: Invalid login details");
        } else {
            User newUser = new User(returnedType, user, pw);
            boolean startSession = currentSession.beginSession(newUser);
            intro_window.setVisible(false);
            currentSession.setSize(INTRO_WIDTH, INTRO_HEIGHT);
            currentSession.setLocationRelativeTo(null);
            currentSession.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            currentSession.setVisible(true);
            currentSession.populateUserLoans();
            currentSession.populateLoanedNumbers();
        }
    }
}
```

CLASS: IntroWindow.java

```java
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
```
import java.awt.Container;
import java.io.File;
import java.io.FileWriter;
import java.util.ArrayList;
import java.util.Scanner;
import java.io.FileNotFoundException;
import java.io.IOException;

public class IntroWindow extends JFrame {
    private JPanel introPanel1;
    private JPanel introPanel2;
    private JLabel introLabel1;
    private JLabel usernameLabel;
    private JLabel passwordLabel;
    private JButton introButton1;
    private JButton loginButton;
    private JButton createAccButton;
    private JTextField usernameArea;
    private JTextField passwordArea;
    private String userFile = "userList.txt";
    private FileWriter writer;
    private final int USER_X = 20;
    private final int USER_Y = 50;
    private final int PW_X = 20;
    private final int PW_Y = 50;
    private final String STANDARD_TYPE = "standard";

    public IntroWindow() {
        this.buildComponents1();
        this.add(introPanel1);
    }

    private void buildComponents1() {
        introPanel1 = new JPanel();
        introLabel1 = new JLabel("Welcome to TWLibrary version 0.1!");
        introButton1 = new JButton("Enter");
        introButton1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                switchToLogin();
            }
        });
        introPanel1.add(introLabel1);
        introPanel1.add(introButton1);
    }

    private void buildComponents2() {
        usernameLabel = new JLabel("Enter username: ");
        usernameArea = new JTextField("username");
        passwordLabel = new JLabel("Enter password: ");
        passwordArea = new JTextField("password");
        loginButton = new JButton("Login");
        createAccButton = new JButton("Create an Account");
        loginButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String tempUser = usernameArea.getText();
                String tempPW = passwordArea.getText();
                TWLibraryMain.progressToLobby(tempUser, tempPW);
            }
        });
        createAccButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                createAccount();
            }
        });
        introPanel2.add(usernameLabel);
        introPanel2.add(usernameArea);
        introPanel2.add(passwordLabel);
        introPanel2.add(passwordArea);
        introPanel2.add(loginButton);
        introPanel2.add(createAccButton);
    }

    public void switchToLogin() {
        this.getContentPane().removeAll();
        introPanel2 = new JPanel();
        buildComponents2();
        this.getContentPane().add(introPanel2);
        this.getContentPane().revalidate();
    }

    private void createAccount() {
        try {
            String username = JOptionPane.showInputDialog("Enter an account username: ");
            String password = JOptionPane.showInputDialog("Enter an account password: ");
            String accessLevel = STANDARD_TYPE;
            writer = new FileWriter(new File(userFile), true);
            writer.append(System.getProperty("line.separator").
                        + username + " " + password + " " + accessLevel);
            writer.close();
            JOptionPane.showMessageDialog(null, "A new user account has been created with standard access level");
        } catch (IOException e) {
            e.printStackTrace();
            System.out.println("Error: Account creation failed as " + userFile + " writing generated an IOException");
        }
    }
}
A public testing version of the createAccount method, using parameters instead of a JOptionPane prompt for input.

```java
public void createAccount(String username, String password) {
    try {
        String accessLevel = STANDARD_TYPE;
        writer = new FileWriter(new File(userFile), true);
        writer.append(System.getProperty("line.separator"));
        writer.append(username + " " + password + " " + accessLevel);
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
        System.out.println("Error: Account creation failed as " + userFile + " writing generated an IOException");
    }
}
```

CLASS: ActiveSession.java

```java
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JTextField;
import javax.swing.JOptionPane;
import javax.swing.JList;
import javax.swing.JScrollPane;
import java.util.ArrayList;
import java.io.FileWriter;
import java.util.Scanner;
import java.io.File;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
public class ActiveSession extends JFrame {
    private User currentUser;
    private String accessLevel;
    private JPanel homePanel;
    private JLabel userDisplay;
    private JLabel accessDisplay;
    private JButton addBook;
    private JButton deleteBook;
    private static Scanner reader;
    private static Scanner subReader;
    private static Scanner reader2;
    private static PrintWriter writer;
    private static FileWriter writer2;
    private static ArrayList<String> userStrings;
    private static String userFile = "userList.txt";
    private ArrayList<String> bookList;
    private static String bookFile = "bookList.txt";
    private JList bookDisplay;
    private JScrollPane scrollPane;
    private String[][] userLoans;
    private String[][] loanedCopies;
    public BookView bookViewOne;
    private static String loanFile = "loanList.txt";
    private final int MAX_LOAN_NUMBER = 10;
    private final int DOUBLE_CLICK = 2;
    private final String INVALID_TYPE = "invalid";
    private final String STANDARD_TYPE = "standard";
    private final String STAFF_TYPE = "staff";
    private final String ADMIN_TYPE = "admin";
```
public ActiveSession() {
}

public boolean beginSession(User user) {
    boolean success = false;
    currentUser = user;
    accessLevel = user.getUserLevel();
    lobbySetup();
    success = true;
    return success;
}

private void lobbySetup() {
    homePanel = new JPanel();
    userDisplay = new JLabel("User: " + currentUser.getName() + "");
    accessDisplay = new JLabel("Access Level: " + accessLevel);
    populateBookList();
    scrollPane = new JScrollPane(bookDisplay);
    buttonSetup();
    homePanel.add(userDisplay);
    homePanel.add(accessDisplay);
    homePanel.add(scrollPane);
    this.add(homePanel);
}

private void buttonSetup() {
    addBook = new JButton("Add Book");
    deleteBook = new JButton("Delete Book");
    addBook.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            addBook();
        }
    });
    deleteBook.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            deleteBook();
        }
    });
    homePanel.add(addBook);
    homePanel.add(deleteBook);
}

public String getAccessLevel() {
    return accessLevel;
}

public boolean buildUserList() {
    try {
        reader = new Scanner(new File(userFile));
        buildSuccess = true;
        String currentLine;
        while (reader.hasNextLine()) {
            currentLine = reader.nextLine();
            userStrings.add(currentLine);
        }
        reader.close();
    } catch (FileNotFoundException e) {
        JOptionPane.showMessageDialog(null, "Error: File " + userFile + " could not be found for reading");
    }
    return buildSuccess;
}

public String checkValidUser(String username, String password) {
    String currentLine;
    String valid = "invalid";
    return valid;
}

public void buttonSetup() {...}
```java
JOptionPane.showMessageDialog(null, "Error: checkValidUser was called before userStrings had been constructed!");

return valid;

for (int i = 0; i < userStrings.size(); i++) {
    currentLine = userStrings.get(i);
    reader = new Scanner(currentLine);
    tempUsername = reader.next();
    tempPassword = reader.next();
    userType = reader.next();
    if (tempUsername.equals(username) && tempPassword.equals(password)) {
        System.out.println("Valid user: "+username + " detected, logging in...");
        System.out.println("Account access level: "+userType);
        valid = userType;
    }
}
return valid;
}

public void populateBookList() {
    Scanner bookScanner;
    bookList = new ArrayList<String>();
    String currentLine;
    try {
        bookScanner = new Scanner(new File(bookFile));
        while (bookScanner.hasNextLine()) {
            currentLine = bookScanner.nextLine();
            bookList.add(currentLine);
        }
        bookDisplay = new JList(bookList.toArray());
        bookDisplay.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent event) {
                if (event.getClickCount() == DOUBLE_CLICK) {
                    populateUserLoans();
                    populateLoanedNumbers();
                    int indexNo = bookDisplay.locationToIndex(event.getPoint());
                    createBookView(indexNo);
                }
            }
        });
    }
    catch (FileNotFoundException e) {
        JOptionPane.showMessageDialog(null, "Error: File " + bookFile + " could not be found for reading");
    }
}

public void populateUserLoans() {
    if (userStrings == null) {
        JOptionPane.showMessageDialog(null, "Error: buildUserList has not been called and cannot be accessed");
        System.exit(0);
    }
    userLoans = new String[userStrings.size()][MAX_LOAN_NUMBER];
    int xIndex = 0;
    int yIndex = 0;
    String currentLine;
    String userWord;
    String currentWord;
    try {
        reader = new Scanner(new File(loanFile));
        while (reader.hasNextLine()) {
            currentLine = reader.nextLine();
            subReader = new Scanner(currentLine);
            userWord = subReader.next();
            userLoans[xIndex][yIndex] = userWord;
            yIndex = 1;
            while (subReader.hasNext() == true) {
                currentWord = subReader.next();
                userLoans[xIndex][yIndex] = currentWord;
                yIndex++;
            }
        }
    }
```
public void populateLoanedNumbers() {
    loanedCopies = new String[bookList.size()][2];
    String currentBookID;
    String currentBookName;
    String comparedBookName;
    String currentLine;
    int currentCopies;
    for(int a=0; a < bookList.size(); a++) {
        currentLine = bookList.get(a);
        reader = new Scanner(currentLine);
        currentBookID = reader.next();
        currentBookName = reader.next();
        currentCopies = 0;
        for(int i = 0; i < userStrings.size(); i++) {
            for(int j = 1; j < MAX_LOAN_NUMBER; j++) {
                if(userLoans[i][j] != null) {
                    comparedBookName = userLoans[i][j];
                    if(comparedBookName.equals(currentBookName)) {
                        currentCopies++;
                    }
                }
            }
        }
        loanedCopies[a][0] = currentBookID;
        loanedCopies[a][1] = Integer.toString(currentCopies);
    }
}

public void createBookView(int listIndex) {
    String indexContents = bookList.get(listIndex);
    System.out.println(indexContents);
    String rawBookID;
    String bookTitle;
    int totalCopies;
    int loanedCopies;
    Book tempBook;
    reader = new Scanner(indexContents);
    rawBookID = reader.next();
    bookTitle = reader.next();
    totalCopies = Integer.parseInt(reader.next());
    tempBook = new Book(rawBookID, bookTitle);
    loanedCopies = findLoanedCopies(rawBookID);
    bookViewOne = new BookView(tempBook, totalCopies, loanedCopies, currentUser);
    bookViewOne.setSize(800, 200);
    bookViewOne.setLocationRelativeTo(null);
    bookViewOne.setVisible(true);
}

private int findLoanedCopies(String bookIDNo) {
    for(int i = 0; i < bookList.size(); i++) {
        if (loanedCopies[i][0].equals(bookIDNo)) {
            return Integer.parseInt(loanedCopies[i][1]);
        }
    }
    return 0; // No copies are found to be loaned
}

private void addBook() {
    String access = this.getAccessLevel();
    if (access.equals(STAFF_TYPE) || access.equals(ADMIN_TYPE)) {
        // Add book logic here
    }
}
String bookID = JOptionPane.showInputDialog("Enter book ID (4 digits) for addition:");
String bookTitle = JOptionPane.showInputDialog("Enter book title:");
String bookCopies = JOptionPane.showInputDialog("Enter the number of book copies:");
try {
    writer2 = new FileWriter(new File(bookFile), true);
    writer2.append(System.getProperty("line.separator"));
    writer2.append(bookID + " " + bookTitle + " " + bookCopies);
    writer2.close();
    this.refreshWindow();
} catch (IOException e) {
    JOptionPane.showMessageDialog(null, "Error: File " + bookFile + " could not be opened for writing");
} else {
    JOptionPane.showMessageDialog(null, "Error: You do not have the access level needed to add or remove books");
}
}
private void deleteBook() {
    String access = this.getAccessLevel();
    if (access.equals(STAFF_TYPE) || access.equals(ADMIN_TYPE)) {
        try {
            String bookID = JOptionPane.showInputDialog("Enter book ID (4 digits) for deletion:");
            ArrayList<String> allText = new ArrayList<String>();
            String currentLine;
            String arrayLine;
            Scanner reader = new Scanner(new File(bookFile));
            while (reader.hasNextLine()) {
                currentLine = reader.nextLine();
                allText.add(currentLine);
            }
            for (int i = 0; i < allText.size(); i++) {
                arrayLine = allText.get(i);
                reader2 = new Scanner(arrayLine);
                if (reader2.next().equals(bookID)) {
                    allText.remove(i);
                }
            }
            reader.close();
            reader2.close();
            writer = new PrintWriter(new File(bookFile));
            currentLine = "";
            for (int i = 0; i < allText.size() - 1; i++) {
                currentLine = allText.get(i);
                writer.println(currentLine);
            }
            currentLine = allText.get(allText.size() - 1);
            writer.println(currentLine);
            writer.close();
            this.refreshWindow();
        } catch (FileNotFoundException e) {
            System.out.println("Error: Book deletion failed as " + bookFile + " could not be found");
        }
    } else {
        JOptionPane.showMessageDialog(null, "Error: You do not have the access level needed to add or remove books");
    }
}
private void refreshWindow() {
    this.getContentPane().removeAll();
    this.lobbySetup();
    this.populateUserLoans();
    this.populateLoanedNumbers();
    this.getContentPane().validate();
    this.getContentPane().repaint();
}
// Testing purposes

/* Public version of addBook with parameter input rather than JOptionPane */
public boolean addBook(String bookID, String bookTitle, String bookCopies) {
    String access = this.getAccessLevel();
    boolean success = false;

    if (access.equals(STAFF_TYPE) || access.equals(ADMIN_TYPE)) {
        try {
            writer2 = new FileWriter(new File(bookFile), true);
            writer2.append(System.getProperty("line.separator"));
            writer2.append(bookID + " " + bookTitle + " " + bookCopies);
            writer2.close();
            this.refreshWindow();
            success = true;
            return success;
        } catch (IOException e) {
            // JOptionPane Error Message would be here:
            System.out.println("Error: File " + bookFile + " could not be opened for writing");
            return success;
        }
    } else { // JOptionPane Error Message would be here:
        System.out.println("Error: You do not have the access level needed to add or remove books");
        return success;
    }
}

/* Public version of deleteBook with parameter input rather than JOptionPane. */
public boolean deleteBook(String bookID) {
    String access = this.getAccessLevel();
    boolean success = false;

    if (access.equals(STAFF_TYPE) || access.equals(ADMIN_TYPE)) {
        try {
            ArrayList<String> allText = new ArrayList<String>();
            String currentLine;
            String arrayLine;
            Scanner reader = new Scanner(new File(bookFile));
            while (reader.hasNextLine() == true) {
                currentLine = reader.nextLine();
                allText.add(currentLine);
            }
            for (int i = 0; i < allText.size(); i++) {
                arrayLine = allText.get(i);
                reader2 = new Scanner(arrayLine);
                if (reader2.next().equals(bookID)) {
                    allText.remove(i);
                    success = true;
                }
            }
            reader.close();
            reader2.close();
            writer = new PrintWriter(new File(bookFile));
            currentLine = "";
            for (int i = 0; i < allText.size() - 1; i++) {
                currentLine = allText.get(i);
                writer.println(currentLine);
            }
            currentLine = allText.get(allText.size() - 1);
            writer.println(currentLine);
            writer.close();
            this.refreshWindow();
            return success;
        } catch (FileNotFoundException e) {
            System.out.println("Error: Book deletion failed as " + bookFile + " could not be found");
            return success;
        }
    } else {
        // JOptionPane Error Message would be here:
        System.out.println("Error: You do not have the access level needed to add or remove books");
        return success;
    }
}
public class BookView extends JFrame {
    private User currentUser;
    private Book currentBook;
    private String bookID;
    private String bookTitle;
    private String bookCopiesTotal;
    private String bookCopiesLoaned;
    private JPanel viewingPanel;
    private JLabel IDLabel;
    private JLabel titleLabel;
    private JLabel totalLabel;
    private JLabel loanedLabel;
    private JButton loanButton;
    private JButton closeButton;
    private String loanFile = "loanList.txt";
    private PrintWriter writer;
    private FileWriter writer2;
    private Scanner reader;
    private Scanner reader2;
    private boolean testing = false;

    public BookView(Book inputBook, int totalCopies, int loanedCopies, User user) {
        currentUser = user;
        currentBook = inputBook;
        bookID = inputBook.getID();
        bookTitle = inputBook.getTitle();
        bookCopiesTotal = Integer.toString(totalCopies);
        bookCopiesLoaned = Integer.toString(loanedCopies);
        setupPanel();
    }

    private void setupPanel() {
        viewingPanel = new JPanel();
        IDLabel = new JLabel("Book ID: " + bookID);
        titleLabel = new JLabel("Book Title: " + bookTitle);
        totalLabel = new JLabel("Total copies of this book: " + bookCopiesTotal);
        loanedLabel = new JLabel("Copies of this book currently on loan (unavailable): " + bookCopiesLoaned);
        loanButton = new JButton("Loan this book");
        closeButton = new JButton("Close book viewing window");

        loanButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                loanBook();
            }
        });
    }

    public JMenuItem getBookDisplay() {
        return bookDisplay;
    }
}

CLASS: BookView.java
closeButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        setVisible(false);
        dispose();
    }
});
viewingPanel.add(IDLabel);
viewingPanel.add(titleLabel);
viewingPanel.add(totalLabel);
viewingPanel.add(loanedLabel);
viewingPanel.add(loanButton);
viewingPanel.add(closeButton);
this.add(viewingPanel);
}
private void loanBook() {
    if(Integer.parseInt(bookCopiesLoaned) >= Integer.parseInt(bookCopiesTotal)) {
        if (!testing) {
            JOptionPane.showMessageDialog(null, "Error: There are no copies of this book remaining!");
        }
        return;
    }
    try {
        ArrayList<String> allText = new ArrayList<String>();
        String currentLine;
        String arrayLine;
        String username = currentUser.getName();
        reader = new Scanner(new File(loanFile));
        while (reader.hasNextLine() == true) {
            currentLine = reader.nextLine();
            allText.add(currentLine);
        }
        boolean userFound = false;
        for(int i = 0; i < allText.size(); i++) {
            arrayLine = allText.get(i);
            reader2 = new Scanner(arrayLine);
            if(reader2.next().equals(username)) {
                String newString = allText.get(i) + " " + bookTitle;
                allText.set(i, newString);
                userFound = true;
            }
        }
        if(userFound == false) {
            writer2 = new FileWriter(new File(loanFile), true);
            writer2.append(System.getProperty("line.separator"));
            writer2.append(currentUser.getName() + " " + bookTitle);
            writer2.close();
            incrementBookCopies();
            return;
        }
        System.out.println(allText);
        reader.close();
        reader2.close();
        writer = new PrintWriter(new File(loanFile));
        currentLine = "";
        for(int i = 0; i < allText.size()-1; i++) {
            currentLine = allText.get(i);
            writer.println(currentLine);
        }
        currentLine = allText.get(allText.size()-1);
        writer.println(currentLine);
        writer.close();
        incrementBookCopies();
        JOptionPane.showMessageDialog(null, "Book: " + bookTitle + " is now allocated for loan to User: " + currentUser.getName() + " and ready for collection");
    } catch (IOException e) {
        e.printStackTrace();
        System.out.println("Error: Loan creation failed as " + loanFile + " writing generated an IOException");
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("FOUND IT!!!!");
    }
}
private void incrementBookCopies() {
    int temp = Integer.parseInt(bookCopiesLoaned);
    temp = temp + 1;
    bookCopiesLoaned = Integer.toString(temp);
}

// Testing purposes
public JButton getCloseButton() {
    return closeButton;
}
public JButton getLoanButton() {
    return loanButton;
}
public String getBookCopiesLoaned() {
    return bookCopiesLoaned;
}
public String getBookCopiesTotal() {
    return bookCopiesTotal;
}
public void setTesting(boolean b) {
    testing = b;
}

CLASS: Book.java

public class Book {
    private String id_no;
    private String title;
    private String genre;
    private String desc;
    private int copies;
    public Book(String id, String name) {
        id_no = id;
        title = name;
    }
    public String getID() {
        return id_no;
    }
    public String getTitle() {
        return title;
    }
    public void setGenre(String input) {
        genre = input;
    }
    public String getGenre() {
        return genre;
    }
    public void setDesc(String input) {
        desc = input;
    }
    public String getDesc() {
        return desc;
    }
    public void setCopies(int input) {
        copies = input;
    }
    public int getCopies() {
}
return copies;
}
}

CLASS: User.java

import java.util.ArrayList;
public class User {
    private String userLevel;
    private String username;
    private String password;
    private ArrayList<String> allTitles;
    // private ArrayList threeDayTitles;
    // private ArrayList oneWeekTitles;
    public User(String level, String name, String pw) {
        userLevel = level;
        username = name;
        password = pw;
        allTitles = new ArrayList<String>();
    }
    public String getUserLevel() {
        return userLevel;
    }
    public String getName() {
        return username;
    }
    public ArrayList<String> getTitles() {
        return allTitles;
    }
}

The JUnit test suite for TWLibrary

CLASS: TWLibraryTesting

import static org.junit.Assert.*;
import org.junit.Test;
import org.junit.Before;
import org.junit.After;
import java.util.ArrayList;
import java.util.Scanner;
import java.io.PrintWriter;
import java.io.File;
import java.io.FileNotFoundException;
public class TWLibraryTesting {
    private TWLibraryMain mainClass;
    private MainWindow mainWindow;
    private IntroWindow introWindow;
    private ActiveSession activeSession;
    private BookView bookView;
    private ArrayList<String> userListText;
    private static String userFile = "userList.txt";
    private ArrayList<String> loanListText;
    private static String loanFile = "loanList.txt";
    private Scanner reader;
    private PrintWriter writer;
    private final String INVALID_TYPE = "invalid";
    private final String STANDARD_TYPE = "standard";
    private final String STAFF_TYPE = "staff";
    private final String ADMIN_TYPE = "admin";
    @Before
    public void testingSetup() {
        mainClass = new TWLibraryMain();
        mainWindow = new MainWindow();
        introWindow = new IntroWindow();
    }
}
activeSession = new ActiveSession();
storeUserFileContents();
storeLoanFileContents();
}

@After
public void testingDestroy() {
    rewriteOriginalUserFile();
    rewriteOriginalLoanFile();
    mainClass = null;
    mainWindow = null;
    introWindow = null;
    activeSession = null;
}

// Positive Tests / Valid Use Sequences

// User type: New user
// Use case sequence: Create Account(username1, password1).Login(username1, password1).
// Description: A new user attempts to create and account and then log in with that account.
// Expected Result: Actions 'Create Account' and 'Login' both succeed.
@Test
public void NewUserCase1() {
    String testUsername = "username1";
    String testPassword = "password1";
    introWindow.createAccount(testUsername, testPassword);
    assertTrue(activeSession.buildUserList());
    assertEquals("Login attempt", STANDARD_TYPE, activeSession.checkValidUser(testUsername, testPassword));
    try {
        mainWindow.initStartScreen();
        mainWindow.initLobbyScreen(testUsername, testPassword);
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for mainWindow.initLobbyScreen");
    }
}

// User type: New user
// Use case sequence: Create Account(username1, password1).Login(username1, password1).Open Book View(index 0).Close Book View.
// Description: New user attempts to create an account and login, and then open and close the book view.
// Expected result: All operations succeed.
@Test
public void NewUserCase2() {
    String testUsername = "username1";
    String testPassword = "password1";
    User testUser = new User(STANDARD_TYPE, "name", "password");
    int testIndex = 0;
    introWindow.createAccount(testUsername, testPassword);
    assertTrue(activeSession.buildUserList());
    assertEquals("Login attempt", STANDARD_TYPE, activeSession.checkValidUser(testUsername, testPassword));
    try {
        mainWindow.initStartScreen();
        mainWindow.initLobbyScreen(testUsername, testPassword);
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for mainWindow.initLobbyScreen");
    }
    assertTrue(activeSession.beginSession(testUser));
    try {
        activeSession.populateUserLoans();
        activeSession.populateLoanedNumbers();
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for activeSession.populateUserLoans");
    }
    try {
        activeSession.bookViewOne.getCloseButton().doClick();
        assertFalse(activeSession.bookViewOne.isVisible());
    } catch (Exception e) {
        fail("Exception was thrown for activeSession.bookViewOne.getCloseButton()");
    }
}

// User type: Standard (existing user with details: testuser, testpassword)
// Use case sequence: Login(testuser, testpassword).
// Description: An existing standard user attempts a login with known details
// Expected Result: Login succeeds.
@Test
public void StandardUserCase1() {
    String testUsername = "testuser";
    String testPassword = "testpassword";
    assertTrue(activeSession.buildUserList());
    assertEquals("Login attempt", STANDARD_TYPE, activeSession.checkValidUser(testUsername, testPassword));
    try {
        mainWindow.initStartScreen();
        mainWindow.initLobbyScreen(testUsername, testPassword);
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for mainWindow.initLobbyScreen ");
    }
}

// User type: Standard
// Book Index: 0, no copies loaned
// Use case sequence: Login(testuser, testPassword).Open Book View(index 0).Loan Book.
// Description: An existing standard user attempts to log in, open the book view and loan a book.
// Expected Result: All operations succeed.

public void StandardUserCase2() {
    String testUsername = "testuser";
    String testPassword = "testpassword";
    User testUser = new User(STANDARD_TYPE, "name", "password");
    int testIndex = 0;
    assertTrue(activeSession.buildUserList());
    assertEquals("Login attempt", STANDARD_TYPE, activeSession.checkValidUser(testUsername, testPassword));
    try {
        mainWindow.initStartScreen();
        mainWindow.initLobbyScreen(testUsername, testPassword);
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for mainWindow.initLobbyScreen ");
    }
    assertTrue(activeSession.beginSession(testUser));
    try {
        activeSession.populateUserLoans();
        activeSession.populateLoanedNumbers();
        activeSession.createBookView(testIndex);
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for activeSession.createBookView");
    }
    try {
        String initialCopies = activeSession.bookViewOne.getBookCopiesLoaned();
        activeSession.bookViewOne.getLoanButton().doClick();
        String newCopies = activeSession.bookViewOne.getBookCopiesLoaned();
        assertTrue(activeSession.bookViewOne.isVisible());
        System.out.println("(case4) Old copies: " + initialCopies);
        System.out.println("(case4) New copies: " + newCopies);
        assertFalse(initialCopies.equals(newCopies));
    } catch (Exception e) {
        fail("Exception was thrown for activeSession.bookViewOne.getLoanButton().doClick()");
    }
}

// User type: Staff (existing user with details: staffuser, staffpass)
// Book Index: 0, no copies loaned
// Use case sequence: Login(staffuser, staffpass).
// Description: An existing staff user attempts to log in with known details.
// Expected Result: Login succeeds.

public void StaffUserCase1() {
    String testUsername = "staffuser";
    String testPassword = "staffpass";
    assertTrue(activeSession.buildUserList());
    assertEquals("Login attempt", STAFF_TYPE, activeSession.checkValidUser(testUsername, testPassword));
    try {
        mainWindow.initStartScreen();
        mainWindow.initLobbyScreen(testUsername, testPassword);
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for mainWindow.initLobbyScreen ");
    }
}
public void StaffUserCase2() {

    String testUsername = "staffuser";
    String testPassword = "staffpass";
    User testUser = new User(STAFF_TYPE, "staffuser", "staffpass");
    int testIndex = 0;
    assertTrue(activeSession.buildUserList());
    assertEquals("Login attempt", STAFF_TYPE, activeSession.checkValidUser(testUsername, testPassword));
    try {
        mainWindow.initStartScreen();
        mainWindow.initLobbyScreen(testUsername, testPassword);
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for mainWindow.initLobbyScreen");
    }

    assertTrue(activeSession.beginSession(testUser));
    try {
        activeSession.populateUserLoans();
        activeSession.populateLoanedNumbers();
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for activeSession.createBookView");
    }

    try {
        activeSession.bookViewOne.getCloseButton().doClick();
        assertFalse(activeSession.bookViewOne.isVisible());
    } catch (Exception e) {
        fail("Exception was thrown for activeSession.bookViewOne.getCloseButton().doClick()" );
    }

}

public void StaffUserCase3() {

    String testUsername = "staffuser";
    String testPassword = "staffpass";
    User testUser = new User(STAFF_TYPE, "staffuser", "staffpass");
    int testIndex = 0;
    String bookID = "0025";
    String bookTitle = "testTitle";
    String bookCopies = "1";
    assertTrue(activeSession.buildUserList());
    assertEquals("Login attempt", STAFF_TYPE, activeSession.checkValidUser(testUsername, testPassword));
    try {
        mainWindow.initStartScreen();
        mainWindow.initLobbyScreen(testUsername, testPassword);
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for mainWindow.initLobbyScreen");
    }

    assertTrue(activeSession.beginSession(testUser));
    try {
        activeSession.populateUserLoans();
        activeSession.populateLoanedNumbers();
        assertTrue(activeSession.addBook(bookID, bookTitle, bookCopies));
        assertTrue(activeSession.deleteBook(bookID));
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for activeSession.addBook / activeSession.deleteBook" );
    }

}

// Negative Tests / Invalid Use Sequences

// User type: Standard User
// Book index: New book, (0025, testTitle, totalcopies:1)
// Use case sequence: Login(testuser, testpass).Add Book(0025).Delete Book(0025).
// Description: A standard user attempts to log in, then adds and removes the same bookID.
// Expected result: Add book operation fails due to the user type, ending the test case.
@Test
public void StandardUserCase3() {
    String testUsername = "testuser";
    String testPassword = "testpassword";
    User testUser = new User(STANDARD_TYPE, "staffuser", "staffpass");
    int testIndex = 0;
    String bookID = "0025";
    String bookTitle = "testTitle";
    String bookCopies = "1";
    assertTrue(activeSession.buildUserList());
    assertEquals("Login attempt", STANDARD_TYPE, activeSession.checkValidUser(testUsername, testPassword));
    try {
        mainWindow.initStartScreen();
        mainWindow.initLobbyScreen(testUsername, testPassword);
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for mainWindow.initLobbyScreen ");
    }
    assertTrue(activeSession.beginSession(testUser));
    try {
        activeSession.populateUserLoans();
        activeSession.populateLoanedNumbers();
        assertFalse(activeSession.addBook(bookID, bookTitle, bookCopies));
        assertFalse(activeSession.deleteBook(bookID));
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// User type: Staff User
// Book index: bookID 0078
// Use case sequence: Login(staffuser, staffpass).Delete Book(0078).
// Description: Attempted deletion of a bookID not currently stored.
// Expected result: Delete Book(0078) fails.
@Test
public void StaffUserCase4() {
    String testUsername = "staffuser";
    String testPassword = "staffpass";
    User testUser = new User(STAFF_TYPE, "staffuser", "staffpass");
    String bookID = "0078";
    assertTrue(activeSession.buildUserList());
    assertEquals("Login attempt", STAFF_TYPE, activeSession.checkValidUser(testUsername, testPassword));
    try {
        mainWindow.initStartScreen();
        mainWindow.initLobbyScreen(testUsername, testPassword);
    } catch (Exception e) {
        e.printStackTrace();
        fail("Exception was thrown for mainWindow.initLobbyScreen ");
    }
    assertTrue(activeSession.beginSession(testUser));
    try {
        activeSession.populateUserLoans();
        activeSession.populateLoanedNumbers();
        assertFalse(activeSession.deleteBook(bookID));
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// User type: Staff User
// Description: Attempt at loaning a book past its total number of copies.
// Expected result: 2nd Loan Book operation fails.
@Test
public void StaffUserCase5() {
    String testUsername = "staffuser";
    String testPassword = "staffpass";
    User testUser = new User(STAFF_TYPE, "staffuser", "staffpass");
    int testIndex;
    int noError = 2; // Stops JOptionPane error display
    String bookID = "0083";
    ...
String bookTitle = "failBook";
String bookCopies = "1";
assertTrue(activeSession.buildUserList());
assertEqual("Login attempt", STAFF_TYPE, activeSession.checkValidUser(testUsername, testPassword));
try {
    mainWindow.initStartScreen();
    mainWindow.initLobbyScreen(testUsername, testPassword);
} catch (Exception e) {
    e.printStackTrace();
    fail("Exception was thrown for mainWindow.initLobbyScreen ");
} assertTrue(activeSession.beginSession(testUser));
try {
    activeSession.populateUserLoans();
    activeSession.populateLoanedNumbers();
    activeSession.addBook(bookID, bookTitle, bookCopies);
    activeSession.populateUserLoans();
    activeSession.populateLoanedNumbers();
    testIndex = activeSession.getBookDisplay().getModel().getSize() - 1;
    activeSession.createBookView(testIndex);
} catch (Exception e) {
    e.printStackTrace();
} try {
    activeSession.bookViewOne.setTesting(true);
    activeSession.bookViewOne.getLoanButton().doClick();
    activeSession.bookViewOne.getLoanButton().doClick();
    String finalCopies = activeSession.bookViewOne.getBookCopiesLoaned();
    String totalCopies = activeSession.bookViewOne.getBookCopiesTotal();
    assertTrue(finalCopies.equals(totalCopies));
} catch (Exception e) {
    e.printStackTrace();
    System.out.println("Exception in activeSession.bookViewOne.getLoanButton().doClick()");
}

private void storeUserFileContents() {
    try {
        userListText = new ArrayList<String>();
        String currentLine;
        reader = new Scanner(new File(userFile));
        while (reader.hasNextLine() == true) {
            currentLine = reader.nextLine();
            userListText.add(currentLine);
        }
        reader.close();
        reader = null;
    } catch (FileNotFoundException e) {
        System.out.println("Error initialising test conditions: " + userFile + " could not be found for reading");
    }
}

private void rewriteOriginalUserFile() {
    try {
        // System.out.println("writing...");
        // System.out.println(userListText.toString());
        writer = new PrintWriter(new File(userFile));
        String currentLine = "";
        for(int i = 0; i < userListText.size()-1; i++) {
            currentLine = userListText.get(i);
            writer.println(currentLine);
        }
    }
private void storeLoanFileContents() {
    try {
        loanListText = new ArrayList<String>();
        String currentLine;
        reader = new Scanner(new File(loanFile));
        while (reader.hasNextLine() == true) {
            currentLine = reader.nextLine();
            loanListText.add(currentLine);
        }
        reader.close();
        reader = null;
    } catch (FileNotFoundException e) {
        System.out.println("Error initialising test conditions: " + loanFile + " could not be found for reading");
    }
}

private void rewriteOriginalLoanFile() {
    try {
        // System.out.println("writing...");
        // System.out.println(loanListText.toString());
        writer = new PrintWriter(new File(loanFile));
        String currentLine = "";
        for(int i = 0; i < loanListText.size()-1; i++) {
            currentLine = loanListText.get(i);
            writer.println(currentLine);
        }
        currentLine = loanListText.get(loanListText.size()-1);
        writer.print(currentLine);
        writer.close();
        writer = null;
        loanListText = null;
    } catch (FileNotFoundException e) {
        System.out.println("Error initialising test conditions: " + loanFile + " could not be found for reading");
    }
}
The JUnit test suite running in Eclipse

Examples of the TWLibrary application running in Eclipse