Term Project: *Task Manager Application*

Design

Table of Contents

1 Introduction

This document intends to outline the description of the Task Manager application, a Java program I shall create as a course requirement. The general goal of this application is to help unique users keep track of personal life tasks, through a simple and straight forward user interface.

1.1 Purpose and Scope

This design document attempts to serve as a comprehensive outline of the features to be included in the finished application, and is important to maximize productive of time spent implementing. This document shall consider the overall design and process of implementation, first in a wide scope, and then in details of components. After outlining important design considerations, this document will explore a complete system overview, to create a high level understanding of the project. Following this, a system architecture analysis will outline the systems which shall compose the application. Details of this architecture will follow is section 5.

1.2 Target Audience

The potential target audience of this application is very broad. It could include any person or professional who is interested in software assistance in keeping track of task progress. This audience may prefer a more simple software solution than the complicated ones available on the market.

1.3 Terms and Definitions

The Task Manager Application (TMA) is the title of the project.
A major part of this project is designing the graphical user interface (GUI).

This project will be built with the help of an IDE (Integrated Development Environment) as well as a JDK (java development kit).

2 Design Considerations

Design considerations are the real and tangible constraints, or development obstacles which I will need to account for. No project exists without certain fundamental requirements and considering here the constraints, dependencies and methodology will make implementation more efficient.

2.1 Constraints and Dependencies

As in most software projects, time is the largest constraint and most important to account for. I balance progress of the project between a job and other coursework so it is important not to make implementation goals too ambitious in order to adhere to deadlines. I am required to implement the application in Java, with a Graphical user interface. I must allow for multiple users on a single system.

2.2 Methodology

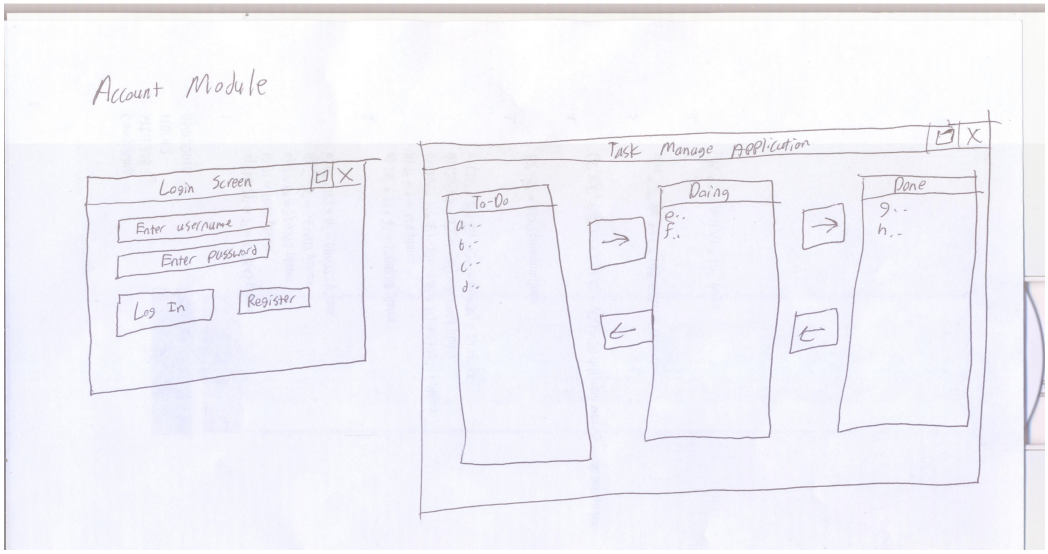
The software engineering methodology I am using for the Task Manager Application is best described as incremental. In accordance with the flow of class progression and time constraints, incremental fits best as courses tend to progress linearly. In general I will focus my attention on one part of the software life-cycle at a time, from requirements, design, analysis, implementation and testing. There will be some overlap of attention between adjacent stages.

3 System Overview

Though I would not consider this system very complicated by relative standards, a complete overview of the entire application shall be a major step in driving the projects focus.

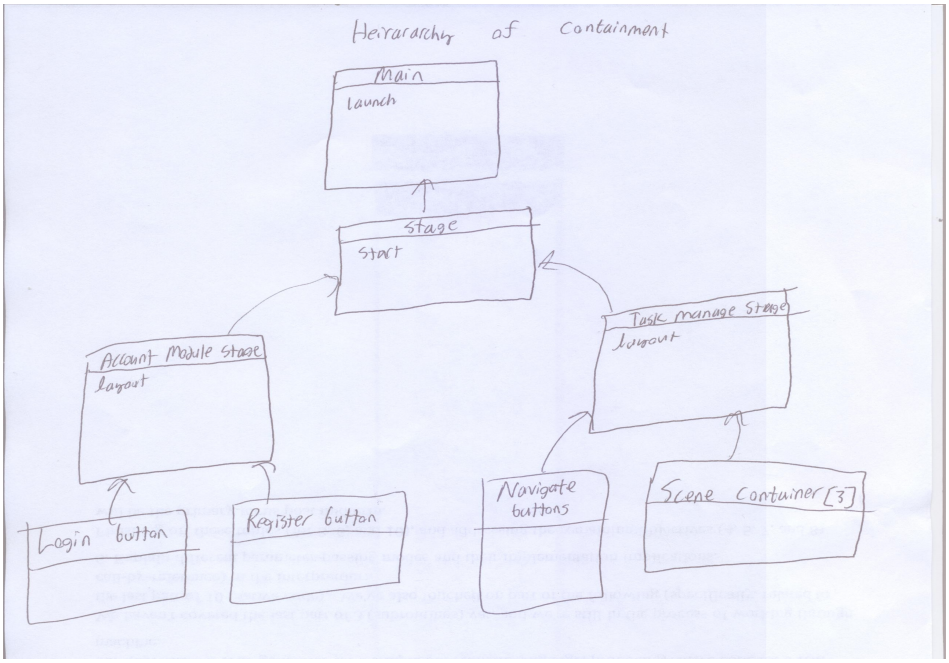
The task manager application as a functional system, will work from two major modules. The first module will be user account management and authentication. Like other applications, users will need to register with a unique name and password, per the system. There are considerations in this module such as simple password encryption, and saving and loading user data. Upon successful login, a user will be transferred to the second module, which shall be the management application.

The management module is the primary goal of the project. It will mainly be divided into 3 aspects, being “to-do”, “in progress” and “completed”. Users will be able to add new tasks as strings, to the “to-do” window. Tasks can be transferred linearly between the 3 stages, and removed once they are “complete” The images captures the intended layouts of the finished modules



4 System Architecture

Both modules, the user authentication as well as the management module will require specific implementations of unique Java architecture. They will require certain libraries and dependencies to be implemented efficiently, as well as powerfully. JavaFX is a Java framework which intents to replace Swing as a cross platform, GUI tool. As such, I will utilize it heavily for all of my projects architecture. The below image represents the classes introduced later and their relations



4.1 The Task Managing Module

The task managing module will exist as an embedded system of frames. The top level container of this system will be a “Stage” in JavaFX. This is essentially the window view. Though not very significant on its own, this stage will house some very important components. Through use of the “stackpane” layout, the stage will contain the primary scene, which is the component to house all other fundamental aspects of this module. The stage will also contain some basic window controllers such as close, maximize and

minimize which are implement by default in JavaFX. The stage will also bear the applications name for user display

4.1.1 The Primary Scene and Layout

The primary scene is the top level container related to all other components of the task management module. A relative layout system here, defined by JavaFX will assist in organizing all of the content of this module.
The main container within this scene will be an “Hbox” pane, used to organize the remaining components in a horizontal manner. This Hbox itself will contain 5 “Vbox”or vertical box panes. Three of these Vbox panes will be specially implementation task status contains. The other two will each contain two transition buttons, vertically. Each transition button, when clicked, will transition the currently highlighted task from the status containers, to the next container.

4.1.2 The Three Task Status Containers

The task status containers themselves will share very similar functionality, so it would be appropriate to extend them from the Vbox class, and force implementation of the necessary features. The top of this unique Vbox will be a text box, to contain a static title, such as To-Do, Doing, or Done. Below this will be another embedded Vbox, which shall have text boxes containing specific tasks within. This inner Vbox will frequently update and be tied to different event listeners.

At the bottom of the task status container/pane there will be a function button, whose purpose will differ depending on which task status container it is found in. In the “To-Do” pane, this button will allow the user to add new tasks. In the “Doing” pane, this button will allow the wording of the task to be edited. In “Done” this button will allow the selected task to be removed.

4.2 The Account Management Module

Upon starting the application, the account management module will be the first thing the user sees. This module will exist as its own stage, or unique window, with a title

and basic exit button. The primary scene, will be defined by a Vbox layout, to contain all of the features. At the top, a text box to welcome the user. Below this, two text field relating to user login, so a username and password entry field. Underneath the text fields, there will be a login button, and finally a registration button. The stage for the account management module will be set to small dimensions than the stage for the task management.

4.2.1 User registration

The very first thing that will concern a new user is the registration subsystem. Upon clicking the registration button in the account management module, a special JavaFX Menu object will be shown. This simple menu will contain 3 text fields and a button. The fields will designate a desired username, desired password, and password confirmation with minimal checks. The button will read “OK” and attempt to validate the registration. Registration will be valid if the username and password meet the minimum criteria, and if the username does not already exist.

4.2.2 User Login/Authentication

User authentication is a subsystem related to the components of the account management module. These components, being the username text field, password text field, and login button, must all work together and communicate properly to function. Conveniently, JavaFx has some predefined UI controls to utilize. The text field for the username will use a simple prompt, to give its purpose context. The password field will also have a prompt, and will automatically be able to hide the characters typed. Upon clicking the login button, and event listener will capture the current username and password state, and reject empty states. These can be compared to a lightly encrypted CSV file of login/password pairs. If a match is found, the user proceeds to the next stage.

5 Detailed System Design

5.1 The Task Managing Module

JavaFX is included with all recent java runtime libraries, but to be utilized it must be imported and a base class must extend “Application”. The Task Managing module will itself be implemented at a class hierarchy, and run from a Main/controller class.
The Main class will create and manage two scenes. The scene relevant to this is the task managing scene. This scene will only be bounded to the stage, if the is a successfully login. Once the scene is bounded, the stage will show to the user. This pseudocode shows the insertion point of the program utilizing the stage.

```
Class Main extends Application
{

        main method() {launch();} //Applcation function call
        @override
        public void start(Stage primaryStage){
                if(the user logs in successfully) { //switch stage to this scene.
                        new Layout layout;
                        new Task_Scene task_scene;
                        primaryStage.set_scene(task_scene);
                        primaryStage.show();

}
Class Task_Scene extends Scene{ ….....}
```

5.1.1 The Primary Scene and Layout

The task scene class makes itself responsible for handling and delegating all of the needs of the task management system. By implementing an Hbox layout, it can organize all

elements in a left to right fashion. Using a hbox.getChildren.addall() command, the layout can add everything needed to the scene. The following pseudocode shows this.

```
Class Task_Scene extends Scene
{
        Hbox layout;
        Constructure{ assemble(); this.setlayout(layout);}
        assemble(){
                //Create and initialize status containers
                //Each status container differes only by title, so set the titles here
                layout = new Hbox
                layout.getChildren.AddAll(   Status Container 1,
                                             ~* A Vbox of 2 buttons*~,
                                             Status Container 2
                                             ~*Another Vbox of 2 buttons *~,
                                             Status Continer 3);


}
```

Here shows the generally methodology to create this unique scene. As will shown below, the Status Containers are themselves implementations of a class which extends Vbox. A base class is used due to the similarities to be see of the Status containers. The other Vbox's, each with two buttons must also be implemented here. As stated in the higher level description, these buttons are used move items between status containers. For example:

```
Button 1to2 = new Button();
1to2.setImage(a_right_arrow.png);
1to2.setListener(New Listener(
        Check if a task is highlighted in container 1. If not, do nothing
        If a task is highlighted, Make a copy of it in container 2. Remove it from
        container 1.
Vbox.add(button).
```

5.1.2 The Three Task Status Containers

As introduced above, there will exist three task status container, to hold all tasks that the user wishes. They will differ only by their title, which shall be set by the scene as one of {“to do” “doing” or “done}. As Vboxs, these container will add the elements necessary to a set of tasks

```
Class Task_Status_Container extends Vbox
{
        Create a “title” text box with setter and getter
        Create an ArrayList of task objects.
        Functions to add or remove tasks (text boxes) acting as setters for the arraylist
        Constructor(){ add title and arraylist to “this” Vbox}
}
```

5.2 The Account Management Module

The account management module shall be the first thing to run on program launch. The success or failure of user authentication will determine if the task managing module even appears at all. The account module will exist entirely within a uniquely implemented scene.

```
Class Account_management_module extends Scene{
        Vbox layout = new layout;
        New plaintext textfield, for username with prompt;
        New password textfield for password with prompt;

        Arraylist for all users;
        Constructur(){ go to external file and populate arraylist}
        Login button{clickListener(){
                //see 5.2.2
        Register button(click Listener(){
        //see 5.2.1
        layout.add all children listed above.}
```

5.2.1 User registration

Upon clicking the registration button, a specific algorithm will be followed.

Check the text of the username box. If it is empty, reject

If the text exists, and it matches an entry in the array list, reject.

If the password field is blank, reject.

If none of the above, add a the username and password to the respective array lists. Save both lists out to file.

5.2.2 User Login/Authentication

Authenticating the user is the last step before dismissing the account management module and showing the task managing module. Upon clicking the “login” button.

If the username field is blank, reject.

Compare the username to the username array list. If no match, reject.

Compare the password to the respective element in the password array list. If no match, reject.

If Match, return status code to the calling function so the stage can set the next scene.