**Project Requirements**

Contents

[1 System Requirements 2](#_Toc199939053)

[2 Development Requirements 2](#_Toc199939054)

[2.1 Project Management 2](#_Toc199939055)

[2.2 Testing 2](#_Toc199939056)

[2.3 Coding 3](#_Toc199939057)

[2.4 AI Generated Code 3](#_Toc199939058)

[2.5 Version Control 3](#_Toc199939059)

[2.6 Individual Team Members 3](#_Toc199939060)

[3 User Story Directions 4](#_Toc199939061)

[3.1 User Stories Tab 4](#_Toc199939062)

[3.2 System Tests Tabs 5](#_Toc199939063)

[4 Time Log Instructions 5](#_Toc199939064)

[4.1 *Individual Hours* Tab 5](#_Toc199939065)

[4.2 *Activity Codes* Tab 7](#_Toc199939066)

[4.3 *Time Log* Tab 7](#_Toc199939067)

[5 Getting Started 8](#_Toc199939068)

This document contains the procedures you will adhere to as you develop your software system.

# System Requirements

1. The system is written as a stand-alone Gui application in Java. As described, this is an example of a web or app-based system. You will not write such a system. Instead, you will write a stand-alone, desktop system, a simulator of sorts, that enables all these functions via a single GUI. You can use Java Swing or JavaFX.
2. The system is object-oriented (OO) and should utilize best practices for OO design.
3. The system utilizes text, binary, JSON, or XML files for data persistence and all data is in memory as the program runs, represented as objects. The system DOES NOT utilize a database.

# Development Requirements

## Project Management

1. You will use a GitHub *Project* to manage your development. You will have these columns: *To Do, Doing, Done.* You can add columns if it helps your team.
2. You will break work down into small tasks that 1-2 people will work on (some tasks may involve the whole group). These tasks should be estimated to take between 0.5 hours and 4 hours in length. **ALL tasks must be entered in your GitHub Project. Do not do work on the project without a task.**

Think about this carefully keeping in mind that you must work as a team, and that all team members should contribute to the development tasks. Tasks will be entered as *Issues* in GitHub and added to the *To Do* column. From there, they will be moved to a *Doing* column and finally to the *Done* column.

This will be new activity and it might be a bit of a struggle to do this. Almost all software companies use this approach. Many times, a project manager will develop these, sometimes in conjunction with developers. In some companies, a developer will simply go to the *To Do* list and “pull a ticket.” When complete, they pull another one.

Task/Issue breakdown is iterative. Your group will identify some initial tasks to get started on development. As issues are in progress or completed you will learn more about what you need to do. If a task turns out to be much bigger, or involve things you didn’t initially think of, then simply add them as new tasks, possibly closing out the initial task. Thus, Task/Issue breakdown is ongoing.

Sometimes a task may be to figure out what needs to be done, in other words, a planning session. The conclusion of that task will generally result in concrete tasks that contribute to the development.

Sometimes a task may be a placeholder of sorts, broad, open-ended, just so you don’t forget out it. Then, when ready to address it, it might be broken down into concrete tasks. In this case, I think I would remove the initial task once it has been broken down.

## Testing

1. You are required to have JUnit tests for each class following the conventions discussed in class. You do not need unit tests for GUI classes.
2. You are required to define *system tests* that test each of the user stories. **Each user story should be tested in as many sets of circumstances as needed to verify that your implementation is robust.** Each User Story system test(s) is documented on a separate tab in the *user\_stories\_ps\_team\_X.xlsx* spreadsheet. These serve as specific directions for manual tests that are run through the GUI. You will demonstrate these in project videos.

## Coding

1. Programmed in Java using best practices for object-oriented software development: meaningful class and member names (and proper format-camel case), consistent indentation, naming of collections, naming booleans, encapsulation, class design, no static methods or variables (if you think you need them, submit your reasons to me immediately for my review), etc.

## AI Generated Code

1. Generative AI is allowed; however, **all** prompts must be listed in document 03-Generative AI prompts.
2. All use of AI generated code must follow these conventions:
	1. A comment is placed at the top of **each** generated method along with the engine you used:

// AI Generated – *engine used*

* 1. If you modified the AI generated code moderately or more, add this comment

// AI Generated – *engine used* - MODIFIED

## Version Control

1. Only working, tested code should be in your master branch of GitHub and it should be in a package(s).
2. Each person is required to work in their own branch. Branches are required to be named: *lastname1, lastName2,* etc. If you have more advanced knowledge of GitHub, you can implement your own workflow.
3. Commit early and commit often. When you add a method, commit. When you change something, commit. When you rename some files, commit.
4. The title of all commit messages is required to be prefixed with “ADD”, “FIX”, or “CHANGE”. For example: “Add *short title”*, “FIX *short title”*, “CHANGE  *short title”*. If you find you need another prefix, then use it. Another two may be: REFACTOR *short title*, MERGE *short title*. Titles should meaningfully summarize what was committed.

A commit message should detail why something was done. A commit message is not always necessary.

1. When you have code on a branch ready to merge with master, you should use the rebase workflow and then open a pull request. At least “some” pull requests are required to be reviewed by at least one other member before merging. In practice, all pull requests are reviewed. The more the better for your grade.

## Individual Team Members

1. All time spent on the project is entered, on a timely basis, into a spreadsheet I will share with your group. Directions for the Time Log are in [Section 6](#_Time_Log_Instructions).
2. You should work consistently on the project. In other words, you should not do a minimal amount of work one week, and then double the next week. This ensures that project is not held back.

# User Story Directions

This is information about what you need to do to the *user\_stories\_ps\_team\_X.xlsx* spreadsheet as your work progresses.

* Rename: *user\_stories\_ps\_team\_X.xlsx,* where “X” is your team number.
* Store the *user\_stories\_ps\_team \_X.xlsx* spreadsheet in your GitHub repository in a folder named *docs.*

## User Stories Tab

1. Do not add or remove, or renumber any user stories.
2. At the end of the sprint it should be sorted on Priority.
3. For the *Code* column (column *H*), type “C” if the code is complete, “~” if the code is in progress. Leave blank if you have not started the code.
4. For the *System Tests* column (column *I*), type the number of system tests that have been written (in the spreadsheet). Otherwise, leave blank. The *System Tests* will be written in separate tabs, details are in Section 4.2 below.
5. For the *Comments* column (column *J*), write any comments that are needed about the status of the code or tests.

## System Tests Tabs

You will write system tests for each User Story and each test will be on a separate tab. There are two such tabs in the document already, just as samples. Thus, create a tab for each user story. The tabs should be named: US-1, US-2, *etc.* where the number represents the priority of the user story. On each tab, specify the system test(s) in the format below. An example is provided on the US-1 tab (which should be removed before turning in). There should be at least on system test for each user story.

|  |  |
| --- | --- |
| **US Title** | [User story title] |
|  |  |
| **Test Num** | 1 |
| **Description** | [Brief description] |
| **Directions** | [List steps to conduct test so that it is reproducible] |
| **Expected Output** |  |
| **Comments** |  |
|  |  |
| **Test Num** | 2 |
| **Description** |  |
| **Directions** |  |
| **Expected Output** |  |
| **Comments** |  |

# Time Log Instructions

You will record all time that you spend for this User Stories Assignment using the Time Log Google Sheet that was shared with each member of your team. The spreadsheet has three tabs.

## *Individual Hours* Tab

The **second** tab is “Individual Hours” as shown below. The first time you use the spreadsheet, you should type in your initials and name (the first person should replace my entry). Your total hours will be automatically calculated. Do not alter that field.



## *Activity Codes* Tab

The **third** tab is “Activity Codes” as shown below. You will use one of these codes to allocate each increment of time that you contribute to the project.

|  |
| --- |
| **Activity Codes** |
|  | You will record all time (including thinking, meetings, etc) that you spend for this project using the Time Log tab on this spreadsheet. Use the Activity Codes below to indicate how your time was spent. |
|  |  |
| **Code** | **Task** |
| D | Any activity related to the analysis or design |
| C | Any activity related to coding |
| T | Any activity related to testing |
| P | Any activity related to presentations |
| O | Any other activity |

## *Time Log* Tab

The **first** tab is “Time Log” as shown below. Each time you work on the project you will make an entry by typing your initials, begin date and time, end date and time, interrupt time, activity code, and brief description of what you did. The total time and running total time will be automatically calculated. Do not alter those fields.

The date/time field is formatted; however, you don’t have to enter these exactly as shown. For example, for the first entry shown below, typing: “6/5 9 am” will produce the result shown.

Interrupt time is time you were not working. For example, you may have worked from 11 am to 1 pm, and taken 20 minutes for a snack. Thus, for interrupt time you would enter: “0:20”. For this to work properly, you must enter “h:mm”.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Time Log** |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  | Automatically Calculated |  |  |  |
| **Initials** | **Beg Date/Time** | **End Date/Time** | **Int.Time** | **Tot Time** | **Run Total** | **Act Code** | **Description of Activity** |
|  | 06/05 09:00 AM | 06/05 09:00 AM | 0:00 | 0:00 | 0:00 | D |  |
| dg | 06/05 09:00 AM | 06/05 09:06 AM | 0:01 | 0:05 | 0:05 | D | Project overview in class |
|  |  |  |  |  |  |  |  |

# Getting Started

This is what I recommend you do to get started on the project.

1. Individually, or as a group do the following:
2. Read this document carefully.
3. Read the user stories for Sprint 1 in *user\_stories\_ps\_team\_X* carefully.

As you do these, take notes, identify questions, etc. The entire team should meet and discuss any questions you have noted, clarify a mutual understanding of requirements. However, don’t get bogged down into details – there will be time for that later.

1. Important: Setup a standing time to meet. At first it may be every day. Later you may move to every other day. Be specific about each team member’s assignment. Hold each other accountable.
2. For the project, you are following an Agile Software Development Process with two sprints. Pick some number of user stories, maybe the first 6 and determine:
3. Domain classes needed – quick class diagram by hand is recommended.
4. Controller classes needed
5. Design of Gui – quick sketch on paper is recommended.
6. System tests – This should be documented as described [above](#_System_Tests_Tabs).
7. Identify an initial set of tasks and enter them into GitHub Project, and assign them to team members.

Note:

* A user story may be a single task (if estimated to be <= 4 hours), but more usually, it may be several or many tasks.
* Tasks may be to research something needed, e.g. GUI, or how to work with text files
* Tasks may be related to planning and specifying system tests. Ideally, this should be done immediately so that as a developer finishes a user story, it can be tested. And, as stated in the course, this ensures that the developer writes just enough code to pass the test.
1. Start working on tasks, move tasks on Project Board from *To Do* to *Doing* and eventually to *Done.* Return to Step 3.