**Phase II – Sortie Logger**

**Zachary Hager, Charles Kimmel, and Matthew White**

**Section 6381**

**Group 4**

**27 September 2022**

**Revision History**

|  |  |  |
| --- | --- | --- |
| **Name** | **Date** | **Description** |
| Zachary Hager | 8/22/2022 | Initial GUI with drop downs and text boxes |
| Matthew White | 8/29/2022 | Refinements to critical components. Add data storing methods |
| Zachary Hager | 9/9/2022 | Migrated to tab-based interface & object “Sortie” coordination between 3 tabs (user)/1 class (code) |
| Zachary Hager | 9/25/2022 | Sortie class construction, exception handling in major fields, & save to IDE console |
| Charles Kimmel | 9/27/2022 | Consolidation of revised classes |
|  |  |  |
|  |  |  |
|  |  |  |

**User’s Guide**

Package: Sorties

Files (.java): Main, Window GUI, Sortie, DataHandler

1. Download Group4\_Phase2.zip folder.

2. Extract the Sorties folder.

3. (On Windows OS) right-click the extracted “Sorties” folder.

4. Click “Open Folder as *[Preferred/Installed IDE]* Project”

5. Once IDE opens, compile and run the package

6. Once GUI appears click any of the three tabs to interact with the program.

* The displayed examples were made using IntelliJ IDEA. Eclipse was also used in development. This Guide is not all encompassing. This is merely amongst the easiest ways to run our program across any Windows PC, and IDE.

**Phase II Milestone**

Our milestone for Phase II of our Sortie Logger was to build upon the interactable GUI we have created. Our aim was to store user inputs, and build in more exception handling. To this point, we have met the minimum marks for our milestones of exception handling and storing our data. As of now it is only contained within the console of the IDE. Nevertheless, this still inspires confidence as we now have control of our “Sortie” object and will work further to build a database within the program (as opposed to integrating an online data base).

**Schedule**

Our project is currently on schedule as we have met the criteria for Phase II chiefly of which was having expanding our exception handling, and storing data. That being said, the latter goal we have only been able to meet the absolute minimum on, as it is only retrievable from the IDE’s console. Storing retrievable data has been the biggest challenge to our project thus far and we have had to reassess what that might look like.

**Reevaluations/changes:**

The primary change from Phase I to this week is reconciling the data handling from within the program. We aimed to integrate an online data base from the start, but building a pathway to that transition point seems untenable. As of now we will exclusively be keeping our data stored in the program. We set out multiple means to chart this forward, but have not found any methods that would naturally transition to an online implementation. The time spent on that caused a minor set back in data storage where as of now we are only able to store our information in the console of the IDE running the code.

Our biggest hurdle for the next phase will be further expanding our exception handling and storing data in the program. As of now we have our data and exceptions displaying in the console of the IDE. Going into our next Phase our main focus will be expanding our exception handling. As of now it only annotates when an erroneous entry occurs but does not prevent it. As discussed, before we will also be working to store the information in the program permanently and make it retrievable. Currently, entries can only be viewed in the IDE. Our greatest benefit though is that we already have our Sortie object developed. The current consideration is to route the saved sorties to a subdirectory within the program.

**Current State as compared to Test Plan**

Below is our Test Table for our Program that was created back in Week 2. **Note** That the “PASS, PARTIAL, and FAIL” are not meant to be taken as the efficacy of the program. It is used to illustrate how Phase II lines up with where we want the program to be in the end. In a sense you can see “PASS” as the program currently meeting our objective, “PARTIAL” as meeting some but not all objectives, and “FAIL” as not meeting any objective.

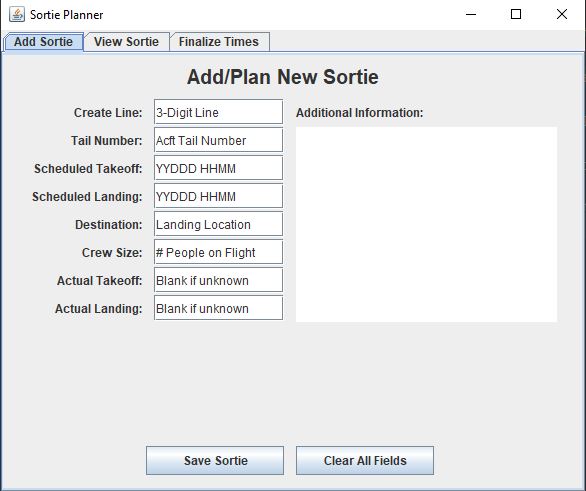
**Test Table:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case # | Test Name | Test Description | Expected Result | Actual Result | Pass/Fail |
| 1 | Start Page | Program opens with two options to proceed to adding a sortie, or retrieving a record. | Test page is present with 2 options | Test page is present with 2 options | **PASS** |
| 2 | Add Sortie | After selecting the ‘Add’ button a page to fill the details of a sortie appear. | Page to add a sortie appears. | Page to add a sortie appears. | **PASS** |
| 3 | Create Line | Acting as the sortie’s axiom, the user gives the sortie a 3-digit designation | Program accepts 3-digit line number. | Program exclusively accepts 3-digit string. | **PASS** |
| 4 | Catching Line error | Makes the user reenter a line number if a character other than a number is entered | Prompt appears notifying the user of their error, user retries. | Catches error, but does not stop it. | **PARTIAL** |
| 5 | Tail Number | Drop down appears requesting the user select a preloaded aircraft tail number. | User selects tail number, and program proceeds. | Shifted to manual entry/ Program accepts Tail Number | **PASS** |
| 6 | Tail number error | Instructs user to select a tail number (aircraft) if the try to proceed without doing so. | Prompt appears notifying the user of their error, user retries. | Does not catch any entry error by the user. | **FAIL** |
| 7 | Scheduled take-off | User enters scheduled take-off date and time in Julian date notation and military time. (YYDDD HHMM) | Program accepts users’ entry. | Program accepts users’ entry. | **PASS** |
| 8 | Scheduled land time | User enters scheduled land date and time in Julian date notation and military time. (YYDDD HHMM) | Program accepts users’ entry. | Program accepts users’ entry. | **PASS** |
| 9 | Actual take-off time | User enters actual take-off date and time in Julian date notation and military time. (YYDDD HHMM) | Program accepts users’ entry. | Program accepts users’ entry. | **PASS** |
| 10 | Actual land time | User enters actual land date and time in Julian date notation and military time. (YYDDD HHMM) | Program accepts users’ entry. | Program accepts users’ entry. | **PASS** |
| 11 | Time deviation over 30 minutes | Prompts user to enter a short explanation on why the aircraft took off late. | User enter string, program accepts string | User enter string, program accepts string/user not prompted | **PARTIAL** |
| 12 | Time deviation under 30 minutes | Prompts user to enter a short explanation on why the aircraft took off early. | User enter string, program accepts string | User enter string, program accepts string/ user not prompted | **PARTIAL** |
| 13 | Date or time annotation error | User enters the date or time in the wrong format, or with wrong characters. | Prompt appears notifying the user of their error, user retries. | Prompts do not appear yet annotating errors | **FAIL** |
| 14 | Save added sortie | User saves all previous information entered. | Sortie data is saved to its own file in the program/data base. | Stores to IDE’s console | **PARTIAL** |
| 15 | Data check before saving | User erroneously attempts to save data while one of the boxes is not filled. | Prompt appears notifying the user of their error, user retries without clearing the other fields. | Prompts do not appear yet annotating errors | **FAIL** |
| 16 | Retrieve Data | After selecting the ‘View Sortie’ button a page appears to retrieve sortie records. | Page appears to fill information to retrieve records. | Page appears to fill information to retrieve records. | **PASS** |
| 17 | Open archived data | User opens archived files. | User view’s record of previously added files. | Program can not retrieve from console | **FAIL** |

**Results:**

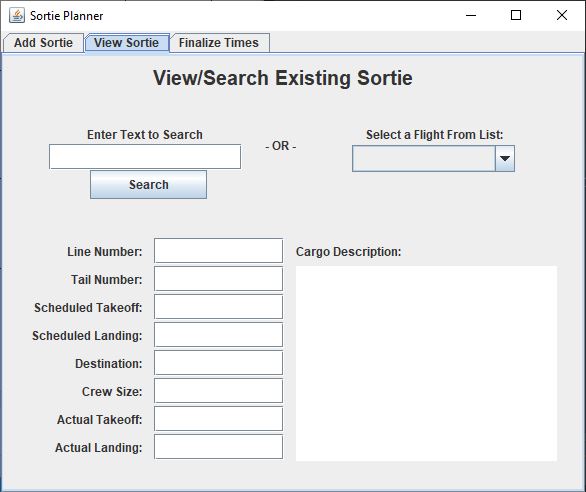
**WindowGUI.java**

* **Add Sortie:**

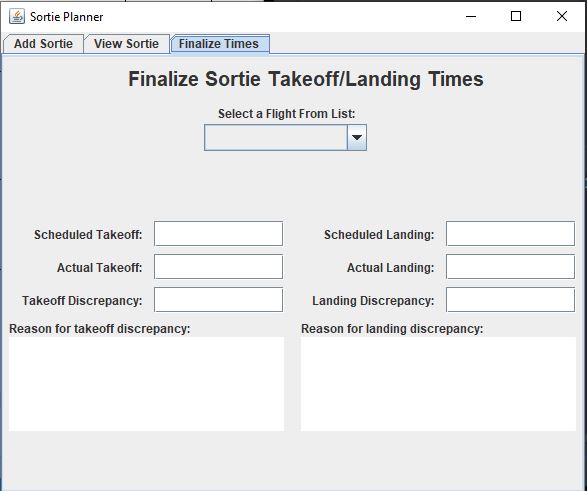


Program opens automatically to the add sortie tab. Save and Clear function operate as expected.

* **View Sortie:**



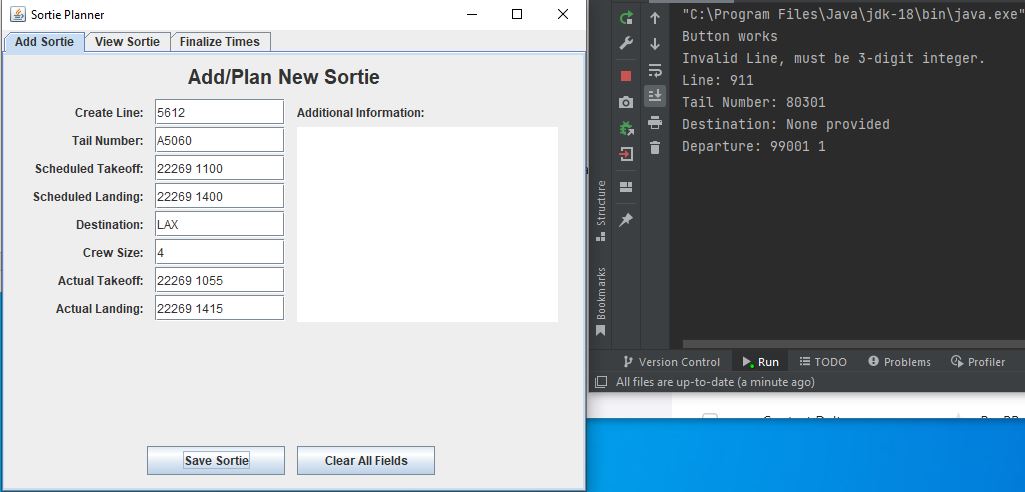
* **Finalize Times**



Experimental tab to better handle time deviations. If this falls short the “Cargo Description” in the other two tabs will be changed to “Deviations” and the user can enter the reason for the deviation in there.

**Sortie.java**

* **Erroneous Line entry**



Catches erroneous entry, but does not prevent it. Additionally, stored data limited to console.

* **Store Sortie/Line entry**

