**Phase II – Sortie Logger**

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

**Section 6381**

**Group 4**

**04 October 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 |
| Zachary Hager | 10/3/2022 | Extensive revamp/Added error checking for all sortie fields when creating/saving a new sortie. |
|  |  |  |
|  |  |  |

**User’s Guide**

Package: Sorties

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

1. Download Group4\_Phase3.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 III Milestone**

Our milestone for Phase III of our Sortie Logger was to continue to build upon exception handling, and data storing. In the prior Phase we managed to get the program to catch but not prevent erroneous entries. We have now succeeded in preventing all erroneous entries in which panes pop up illustrating were the error occurred. This was done by setting more parameters to each field which I discuss further in “reevaluations/changes.” Moreover, the program now saves all the input data, however it is still only in the IDE’s console. Nevertheless, we feel confident going into our last Phase as everything we need is now in the program, and is functioning almost exactly as intended.

**Schedule**

Our project is currently on schedule as we have met the criteria for Phase III chiefly of which was having almost all of our exception handling covered. That being said, our goal in storing the data to an external file is still a challenge we will meet going into the final week. As stated before, this has been our main challenge and will be our primary focus going into the Final.

**Reevaluations/changes:**

Our goals have not changed much going into Phase III from Phase II. As discussed in Phase II, we now aim to store the data to an external file within the program as opposed to the initial proposal of building an online data base. We have also redone how we handled dates and times, by having the program treat them as integers exclusively. This gave us better means to handle erroneous entries. We utilized built in options to catch entries that contained characters other than numbers. This also provided us the means to set an exact number of digits the user must enter.

Our biggest hurdle for the next phase will be storing the data. We are now working on methods and constructors to transpose the data in the console to a .txt file than route that file to a sub-directory within the program. We may also revaluate Tail Number entries and time deviation prompts. As of now the user can make notes if the take off and land times are over or under 30 minutes from the scheduled time, however nothing forces them to do so. That being said we may view this as an onus on the user to fill as they need as there is nothing intrinsically necessary for the program to function with this feature. As far as Tail Numbers, we may change that as integers only as it was an effective means to catch errors in other fields and would still meet our programs intent.

**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 III 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. Note too that all the test photos only contain change made with Phase III, but you can view previous results in our Phase I and Phase II discussion. The Final will have a comprehensive photo catalog of all test results though.

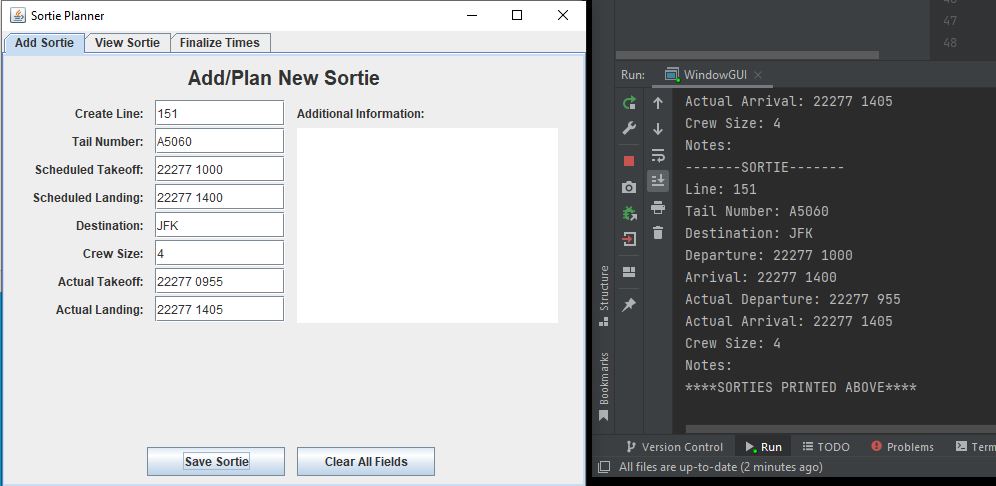
**Test Table:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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. | Prompt appears notifying the user of their error, user retries. | **PASS** |
| 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. | Prompt appears notifying the user of their error, user retries. | **PASS** |
| 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. | Prompt appears notifying the user of their error, user retries without clearing the other fields. | **PASS** |
| 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 cannot retrieve from console | **FAIL** |

**Results:**

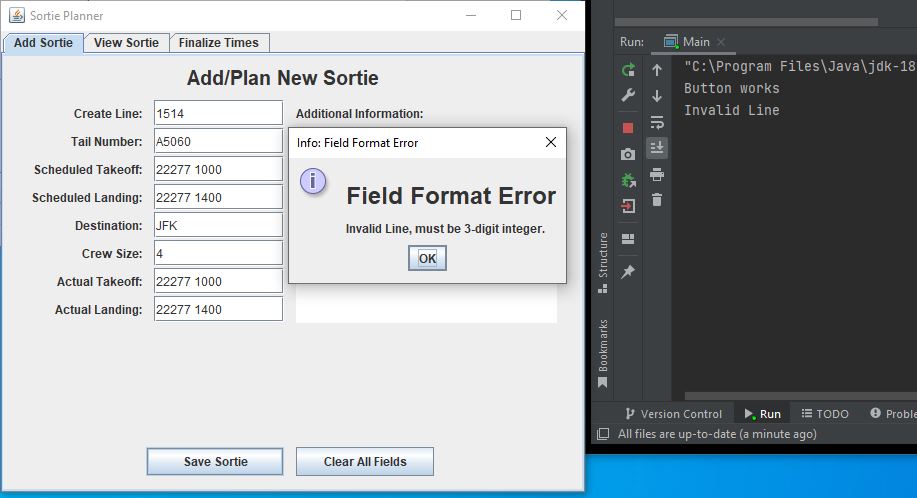
**WindowGUI.java**

* **Add Sortie:**



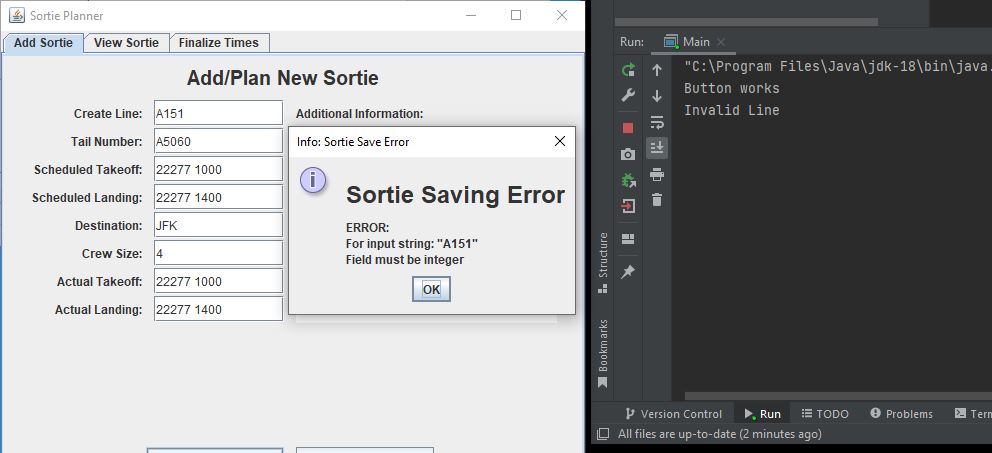
Program opens automatically to the add sortie tab. Save and Clear function operate as expected and further more stores all data entered by the user.

* **Exception Handling Line entry error (too many integers):**



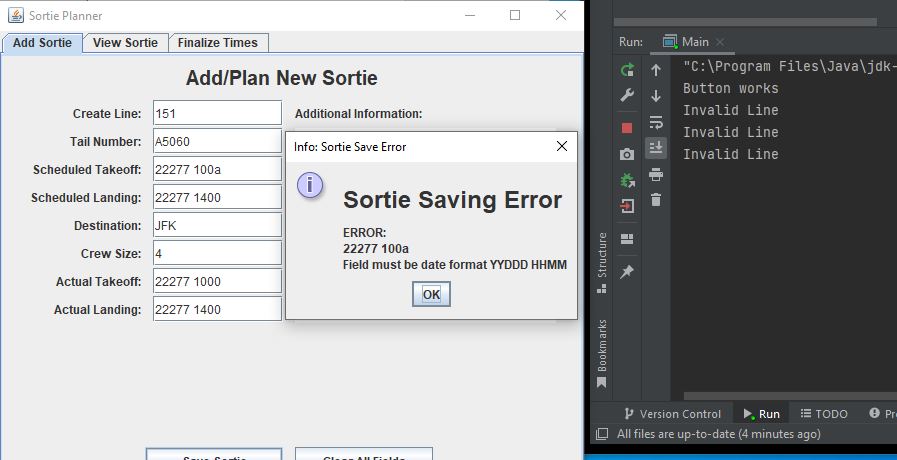
Catches Line entries over 3 digits

* **Exception Handling Line entry error (wrong character):**



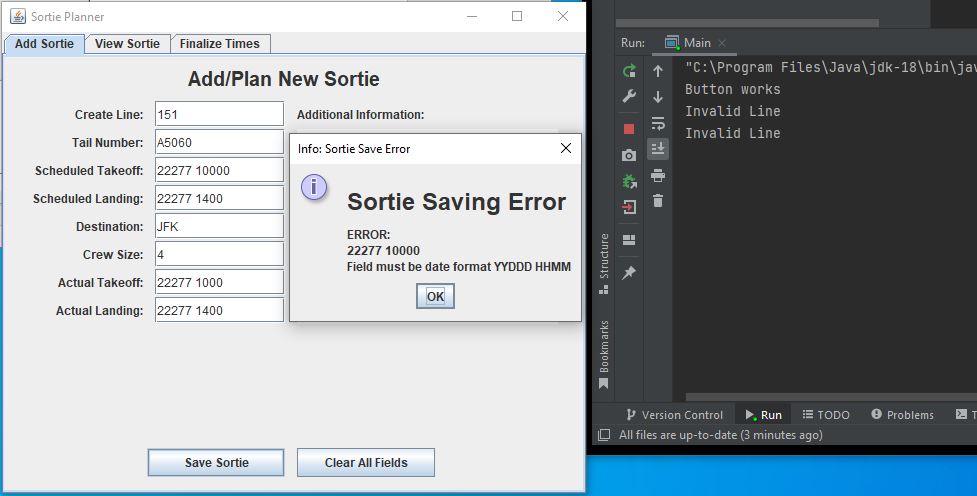
Catches Line if line has any charter other than an integer

* **Exception Handling Time entry error (wrong character):**



Catches time error that has a character other than an integer

**Exception Handling Time entry error (too many digits):**



Catches time error that has too many digits