Tan Chin Khang – Project Portfolio for SpinBox

About the project

My team of 4 software engineering students and I were tasked with enhancing a basic command line interface desktop Duke application for our Software Engineering project. We chose to morph it into a NUS Student companion app called **SpinBox** *(Fig. 1)*. This enhanced application enables NUS students to see their most urgent tasks; have an overview of their task on the calendar; and see their tasks, grades breakdown and files downloaded from LumiNUS for each module.

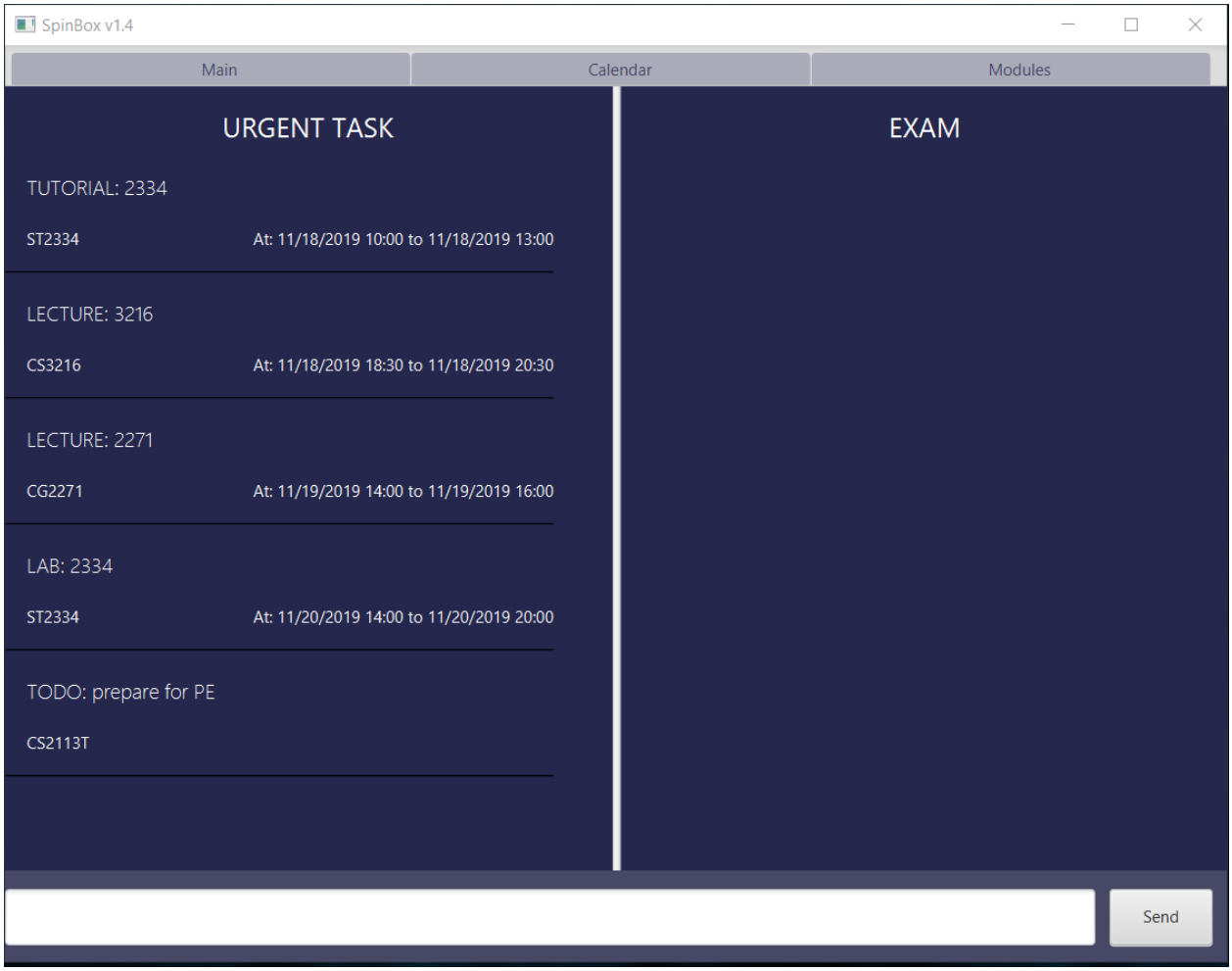

Figure 1: Screenshot of Application

My role was to design and write the codes for the `DateTime` and `Calendar` features as well as the GUI features. The following sections illustrate these enhancements in more detail, as well as the relevant documentation I have added to the user and developer guides in relation to these enhancements.

Summary of contributions

This section shows a summary of my coding, documentation, and other helpful contributions to the team project.

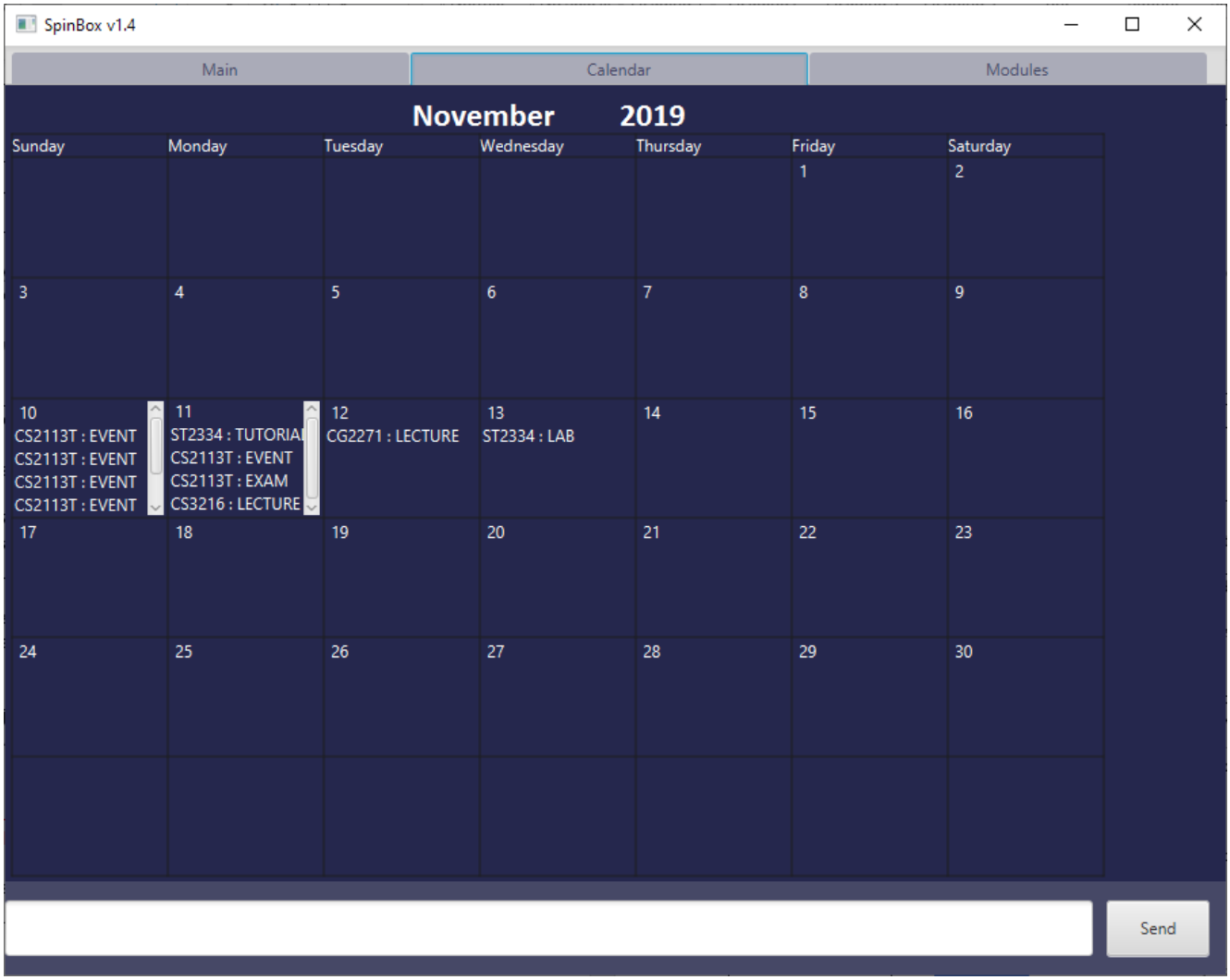

Figure 2:Screenshot of the Calendar tab

Enhancement added: I added the java classes DateTime and Calendar as well as GUI components.

- What it does: Theses java classes allows the developers to create DateTime object without worrying about external libraries since it is nicely encapsulated inside the object. Similarly, the Calendar classes can extract out the start of the month, or week, or a day from a DateTime object.
- Justification: All the developers need to do is to make a new DateTime object with the Date in the form of a string, and the external library natty, which is used in this project, would take care of it. The Calendar class is useful when we implement the digital calendar on the GUI.
- Highlights: This enhancement works well as we can extract out the DateTime with SimpleDateFormat library into a String that is reusable. All needed Date functions are similarly named.
- Credits: *In the DateTime object, it encapsulates java.util.Date, java.util.Calendar java.text.SimpleDateFormat and an external library: com.joestelmach.natty.Parser*

The Screenshot above shows the implementation of DateTime and Calendar in the GUI. *(Figure 2)*

Code contributed: Please click these links to see a sample of my code:
[Functional code for DateTime][Functional code for Calendar][Functional code for GUI]
[Test code For DateTime][Test code For Calendar]

Other contributions:

- Enhancements to existing features:
  - Added framework for GUI (#118)
  - Added more GUI functionality (#119)
  - Add ExamView to Main (#208)
  - Add module code to calendar and main (#206)
  - Improve Calendar GUI and view command (#221)
- Implementation of architecture:
  - Modified Task abstract classes to fit Architecture (#81)
  - Fix GUI bugs and update tests (#196)
- Refactor:
  - Add Exception for Schedulable tasks (#204, #209)
- Tools:
  - Integrated a third-party library (Natty) to the project (#9)

Contributions to the User Guide

We had to update the original duke User Guide with instructions for the enhancements that we had added. The following is an excerpt from our ***Spinbox User Guide,*** showing additions that I have made.

3.3. Viewing the calendar tab : `view / calendar`

Displays the calendar in the calendar tab populated using task data across all modules. The calendar only displays tasks that have a start and end date (a To do or deadline will not show up on the calendar). The current month is displayed by default, unless the full command is used.
Format: `view / calendar or view / calendar [MM/YYYY]`
An expected output is shown below.

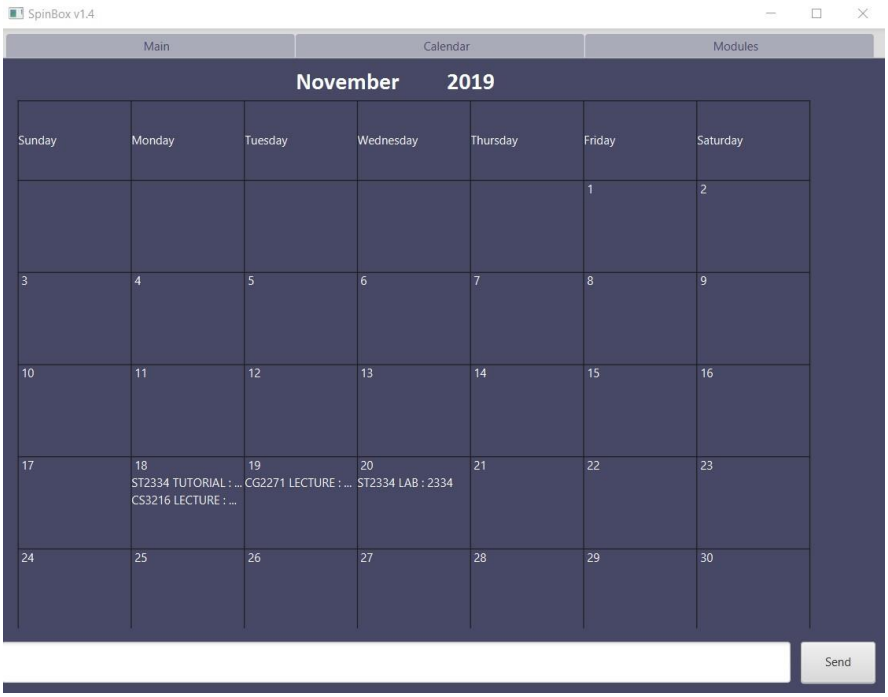

Figure 2: The calendar tab, displaying the schedule for November 2019

Contributions to the Developer Guide

The following section shows my additions to the ***SpinBox Developer Guide*** for the DateTime and Calendar class.

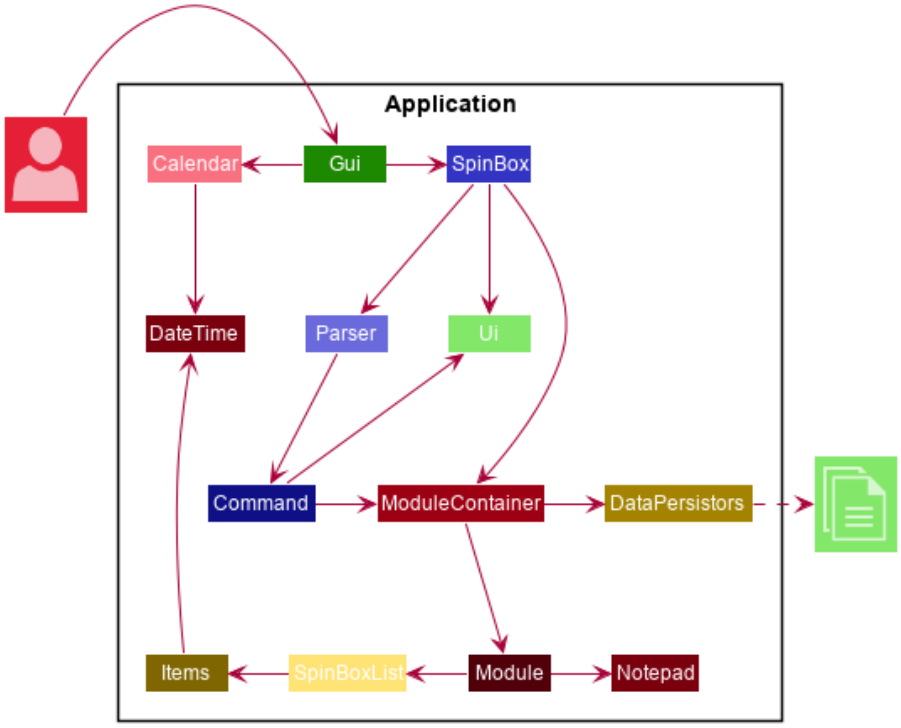

Figure 1.1 Architecture Diagram

This is the ***Architecture Diagram (Figure 1.1)***. It gives the high-level design of the App. Given below is a quick overview of each component.

`Gui` has a `MainWindow` Class which would call `SpinBox`. The `MainWindow` Class is responsible for calling all other `Gui` classes and `Calendar`, for the logic for `Gui` Calendar.

`SpinBox` would call the `Parser` Class to parse input from user. From the input, the `Parser` would be able to return a specific `Command` to `SpinBox`.

It would also interact with the `ModuleContainer` Class and `Ui` Class and pass both of them as parameters into the `Command` Class. This would allow the `Command` class to call functions contained in these two Classes. `ModuleContainers` contains a hashmap of `Module` and also interact with `DataPersistors` and stores the data outside the program.

`Module` contains three `SpinBoxList`, namely `TaskList`, `FileList` and `GradeList`. In addition, each `SpinBoxList` contains `Items` which can be either `GradedComponent` or `File` or a `Task` depending on the type of `SpinBoxList`. If a `Task` is also a `Schedulable`, it would contain two `DateTime` objects for the start and end date and time of the `Schedulable Task`.

`Calendar`, which is one of the deals with logic for `Gui` Calendar, also contain two `DateTime` objects for the start and end date and time of the month it is displaying.

3.1.1 How the architecture components interact with each other

The ***Sequence Diagram (Figure 1.2)*** below shows the sequence diagram for when a user inputs their command into the `Gui`. It shows how SpinBox is able to `setPageTrace()` using `Parser`, before using it to create a `AddCommand` object. The `AddCommand` would be return to `SpinBox`, which would call execute inside `AddCommand` and also pass `ModuleContainer`, and `Ui` objects into the function.

This would then return a response to `SpinBox` which `SpinBox` would be able to update accordingly. The `MainWindow` would also handle the response by updating itself and all `Gui` components to it.

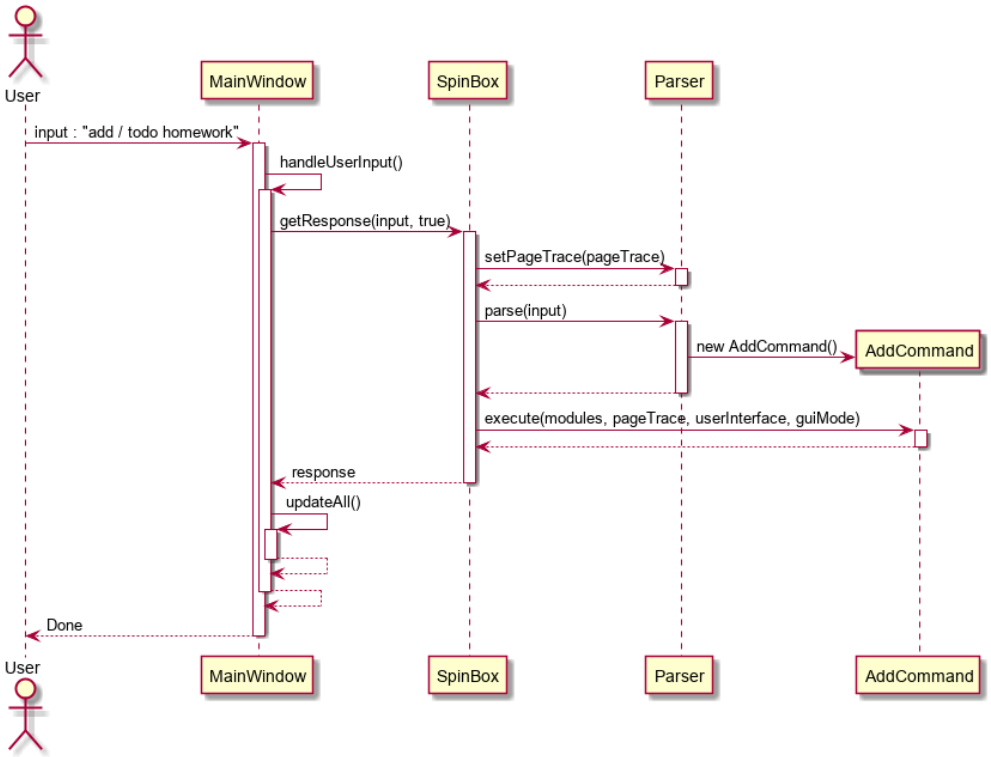

Figure 1.2. Sequential Diagram for Overall Execution

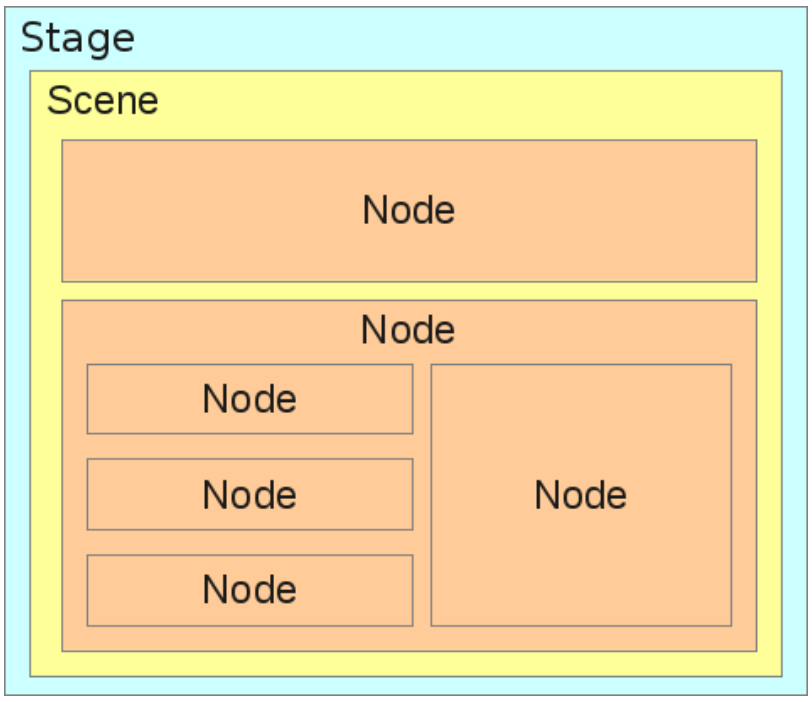

Figure 2.1. Sample JavaFX Node Hierarchy

JavaFX is used to handle the `Gui` component of the program. JavaFX uses node hierarchy to arrange their components ***(Figure 2.1)***. These nodes are contained inside a `Scene` which are contained in a `Stage`.

The `Main` Class in our application sets the `Stage` and `Scene` and sets our `MainWindow` as the root node. All nodes need to be JavaFX components and therefore, `MainWindow` extends a JavaFX component: `GridPane`.

`MainWindow` layout is set by the matching `MainWindow.fxml` files under `src/main/resources/view` folder. Similarly, other `Gui` Classes would have its fxml files in the same folder. In addition, there is also .css file if there are any styling to be added to the layout.

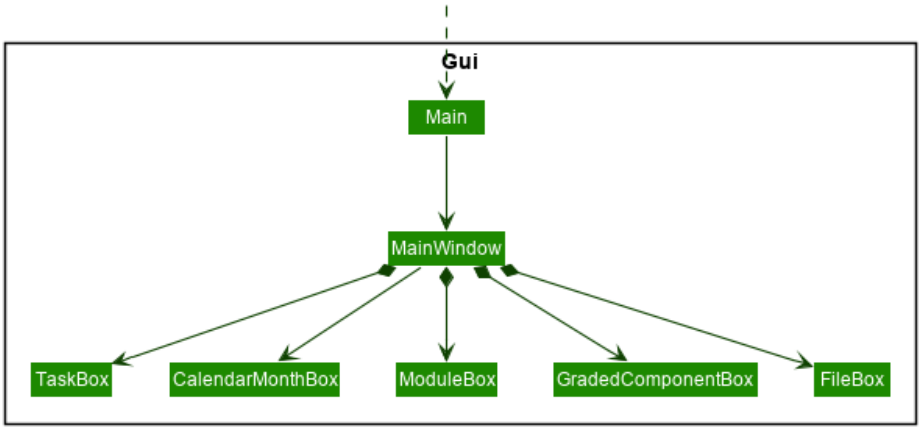

Figure 2.2. Structure of Gui Component

Inside the `MainWindow`, although it contains `Gui` classes ***(Figure 2.2)***, it also contain JavaFX classes which logic, layout and styling are done inside `MainWindow` itself.

This is a sample of the Gui Window with the tab at Main ***(Figure 2.3)***.

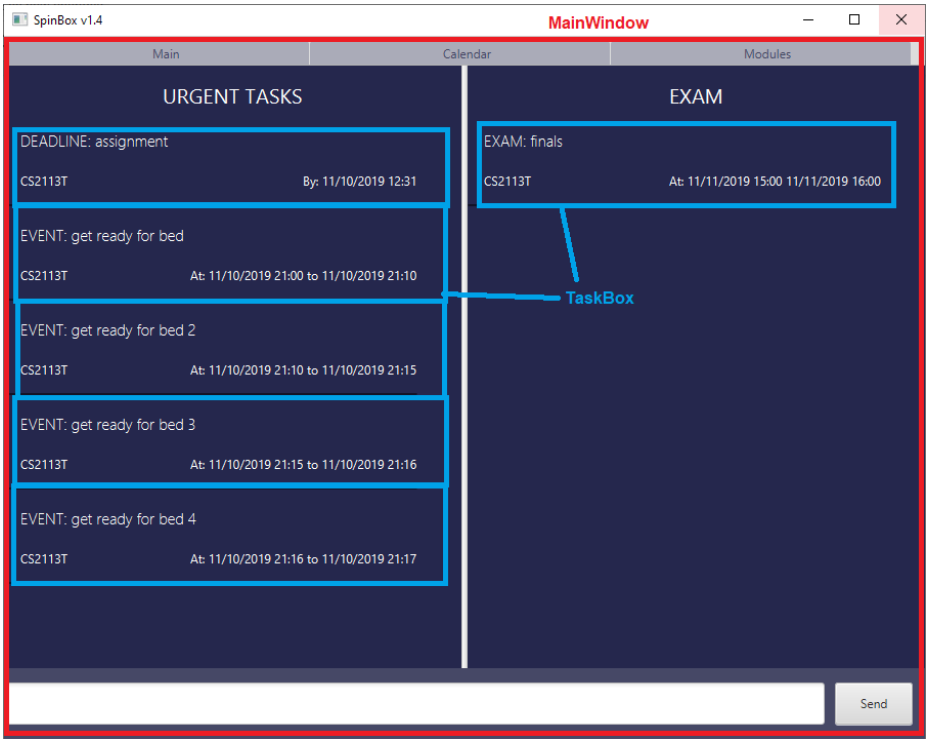

Figure 2.3. Sample of GUI Window

Inside `MainWindow`, examples of its child nodes are JavaFX components are the `TabPane` to swap between tabs, `TextField` for user to write their input and `Button` for the to send the input. `SplitPane` is a child node of `TabPane` and is used to correctly align the two `VBox`. `TaskBox` would populate the `VBox`. Inside, it contains three labels to show the module code, type with description and the date and time.

This list of urgent tasks and exams is generated inside `MainWindow`.

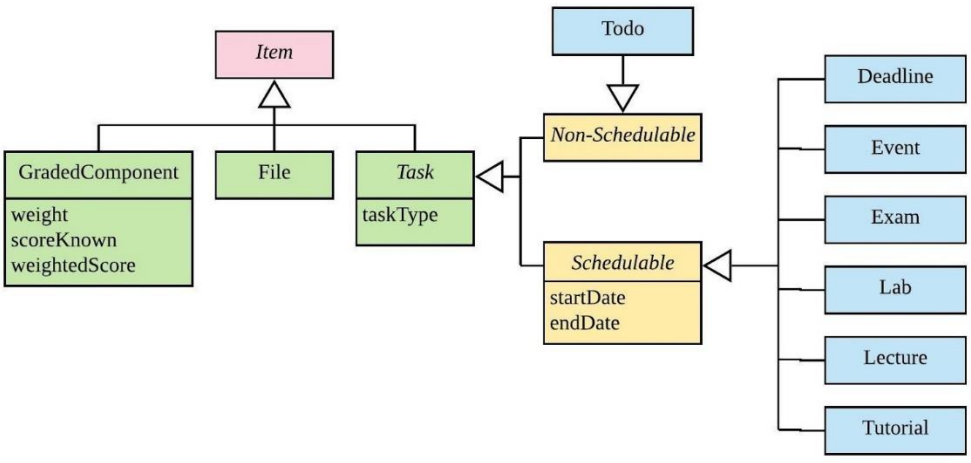

Figure 5. UML Object Diagram for Items

These following classes inherit from the `Item` abstract class:

- `Task`
- `GradedComponent`

- `File`

For `GradedComponents`, there are methods and variables surrounding the weightage of each graded component. These are stored under the `weight`, `scoreKnown` and `weightedScore` variables.

We also create the following subclass to inherit `Task`:
- `Schedulable`
- `NonSchedulable`

Under `NonScheduable`, there is only one subclass which is `Todo`.

For `Schedulable`, there are the following 6 classes that inherits it.
- `Deadline`
- `Event`
- `Exam`
- `Lab`
- `Lecture`
- `Tutorial`

Under `Task`, there is an `enum` called taskType to be able to differentiate between the subclasses. The difference between `Schedulable` class and `Non-Schedulable` class is the two addition DateTime object: startDate and endDate.

3.3. DateTime and Calendar

3.3.1 Implementation

The date and time for each Schedulable Task is being keep tracked by `DateTime`. In this class, it has a private `java.util.Date`. For this class, there are mainly three ways to construct it. The first way with `DateTime(Date)`, the second with `DateTime(String)` and the third with `DateTime(String, Int)`. For construction with `String` instead of `Date` object, which can be passed easily into the private variable, the `String` containing the DateTime would be converted into a `Date` object via third party library called `Natty`.

DateTime implements the following operations on top of trivial functions:

- `DateTime#before(DateTime)/DateTime#after(DateTime)/DateTime#equal(DateTime)` — Return a boolean stating whether this `DateTime` object's date and time is before/after/equal to the `DateTime` object passed
- `DateTime#getStartOfTheWeek()/DateTime#getEndOfTheWeek()` — Return another `DateTime` object with date set as the start/end of the week relative to this `DateTime`.
- `DateTime#getStartOfTheMonth()/DateTime#getEndOfTheMonth()` — Return another `DateTime` object with date set as the start/end of the week relative to this `DateTime`.
- `DateTime#toString()` — Return `DateTime` in `String` version of MM/dd/yyyy HH:mm which can be similarly used to create another `DateTime` object.

The first sets of functions are useful to see if a pair of `DateTime` objects overlaps with another pair of `DateTime` objects. Currently it is utilised by all Schedulable Task to see if it overlaps with any other tasks.

The second and third sets of functions are useful to get the range of days in a week or month which can be paired with the first sets of functions to effectively filter out Schedulable tasks that are in that range. This will be utilised by Calendar to populate the UI to allow the user to see different events and deadlines that are scheduled in that week or month.