Computer Science Moderation Scheduling Program

Josef Lazar

April 24^{th} 2022

Introduction

Faced with the logistical challenge of organizing people in a busy world, algorithms offers an elegant solution. This scheduling program allows its users to easily find available times to meet with various people, and to quickly find out who is and isn't available at any given point in time. It works by taking a text document with people's availability as input, and scanning it to find when people's available times overlap.

To do this and to create an environment for users to easily work with the data, it utilizes the following four classes: `Person`, `Button`, `DropDown`, and `Scheduling_Program` (the main class). The program creates an instance of a `Person` class for each person in the text file, and adds that person's available times to the object's attributes. `Button` objects' and `DropDown` objects' main function is to serve as tools in the program window to help the user interact with the data. Within the `Scheduling_Program` class, the `Person` objects are declared and added to an array list that the program can iterate over. This class is also responsible to displaying the `Button`s and `DropDown` menus, and returning useful information to the user.

Data input

The scheduling program gets people's time availability information from a `names.txt` file in the program's data folder. To change people's time availability, and to add or remove people, the user must change and save the `names.txt` file. As of right now, there is no way to do this through the program window.

The program translates `names.txt` into a string array called `lines`, where each line in the text file is a single string in the array. The program iterates over `lines` looking for strings that qualify as names and time frames; ignoring all other elements. A string is considered a name if it meets the following two

conditions: It contains at least one non-space character, and it does not contain any numbers in it. When a name is found, a new `Person` object is created. A string is considered a time frame if it is a number, followed by a space, followed by a larger number. "`30810 31425`" is an example of a valid time frame. When a time frame is found, it is assigned to the most recently added `Person`.

To use this program, the user must understand how to read and write time frames. The first number is considered the time when a person becomes available, and the second number is the time when a person stops being available. The first digit of a number indicates the day in the following way: 1 = Monday, 2 = Tuesday, ..., 7 = Sunday. The second and third digits indicate the hour. Military time is used, so 03 = 3am, and 15 = 3pm. It is important to include the zero in 03, or else all following numbers will be shifted over by one digit - in other words $03 \neq 3$. The fourth and fifth digits indicate the minutes. Taking this into consideration, our original example, 30810 31425 would be used to indicate that someone is available on Wednesday from 8:10am to 2:25pm. Figure 1 depicts an example of a `names.txt` file.

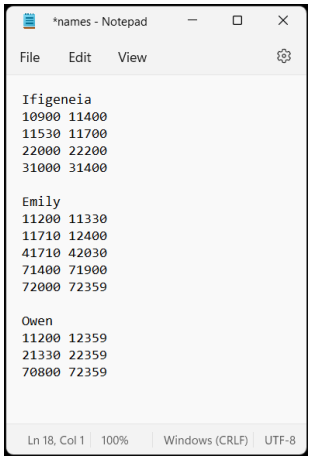

Figure 1: names.txt file example. A Time frame gets assigned to the name listed most recently above it. For example, 41710 42030 gets assigned to Emily.

Program Window

The program window is a graphical interface for the user to interact with the data. Through it, the user is given two tools: Finding overlapping free times in peoples schedules, and seeing who is available at a specific point in time. Respectively, they are displayed on the right and left side of the window.

The overlapping free times feature works by making a `Button` for each `Person` object, with their name on it. Each of these `Button`s works like a switch;

when it is pressed it gets darker, indicating that it has been selected, when the `Button` is pressed again, it returns to its original state. Once the user has selected all the people they want to meet up with, they press the "Find Times" `Button`. All the selected people's overlapping times are then listed.

Figure 2: Scheduling Program Window. This example is showing all the times that Ifigeneia, Emily, Kent, and Maia are all available to meet.

The tool to see who is available at a specific point in time utilizes `DropDown` menus. Using them, the user can select a specific time by choosing among the options listed under: day, hour, minute, am or pm. The program will request that valid options be chosen, until all the parameters are filled in. See Figures 3 and 4 for more for a visual description.

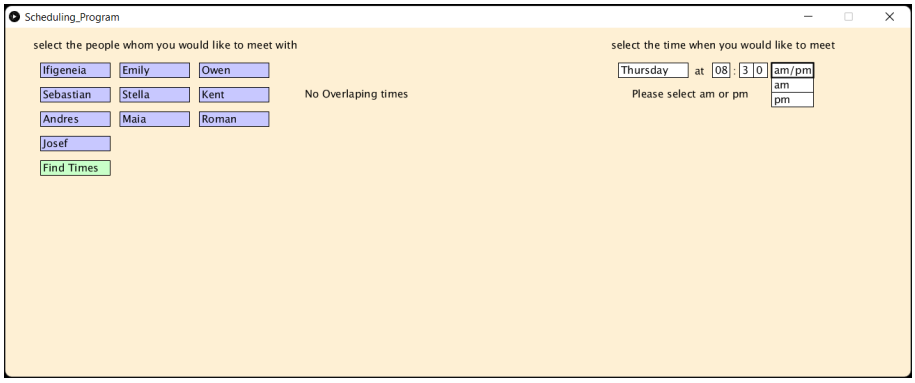

Figure 3: The user has selected a valid day, hour, and minute, and is being prompted to select am or pm.

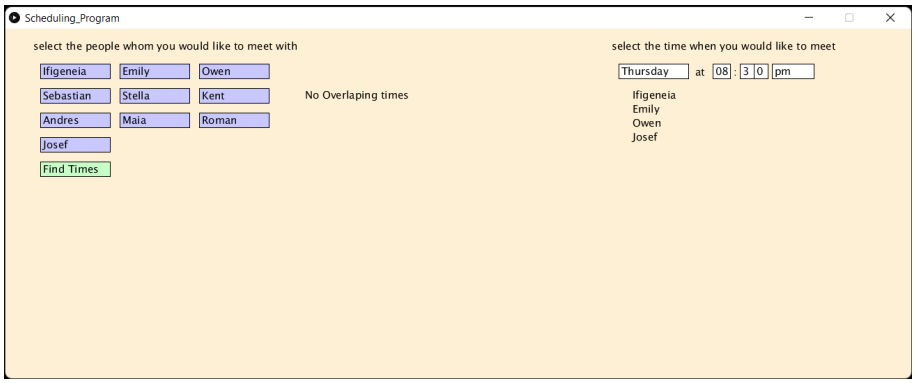

Figure 4: The user has selected a valid time, and all the people who are available to meet at that time are being lisetd.

Classes

The `Button` class is used to make clickable buttons in the program window. Its attributes include the coordinates it should be displayed at, its parameters, its color, and the text that should be displayed on it. Further, it has a `boolean` variable `clicked`, which keeps track of whether the `Button` is selected or not. A `Button` can work like a switch, where the value of `click` will alternate each time it is clicked, as is the case with the `Button`s with people's names on them, or it can work like a regular `Button` whose value is false all the time, except when it is being pressed, as is the case with the "Find Times" `Button`, and the `Button`s used in the `DropDown` class.

The `DropDown` class is used to make drop down menus in the program window. Its attributes include a `Button` that unveils the drop-down menu when clicked, and an array list of `Button`s that appear as options in the drop-down menu, and it includes an integer called `selected`, which keeps track of the index of the chosen option. In its `display` method, `DropDown` highlights the option that the user's mouse cursor in on.

The `Person` class is used to store a person's name and the times that they are available to meet. It has an `addTime` method that is used to assign a new available time to a `Person`. This is important because the algorithm for deducing available times from the imputed time-frame strings is in the `Scheduling_Program` class. The `Person` class also has its own `Button` object with the `Person`s name as the text that should be displayed on it. This is because in this program, all `Person` objects are stored in an array list, so it makes iterating over their respective `Button`s easier.

The `Scheduling_Program` class creates an array list of `Person` objects called `people`. It iterates over the lines in the imputed `names.txt` file, adding

a new `Person` to `people` each time it finds a valid name. When it finds a valid time frame, it adds all the integers between the smaller number and larger number of the time frame to the most recently added `Person`s available times by using its `addTime` method. This means that if "11030 11130" (Monday 10:30am to 11:30am) is imputed, all the times between 11:30 and 11:30 will be in the `Person`s list of available times. As a byproduct, unreal times such as 10:80am will also be added because 11080 is an integer between 11030 and 11130, but as long as the imputed times in `names.txt` are real, this will not come up as an issue when the program is ran.

Next, the `Scheduling_Program` creates the `DropDown` menus needed to select a time, and puts them into an array list called `DropDowns`. Then it displays all the `Button`s and `DropDown`s. When the user selects a `Person`, they get added to a `selectedPeople` array list, which is used as input for the `timeOverlaps` function. The `timeOverlaps` function returns an `IntList` of all the `selectedPeople`s available times. It does this by starting at Monday, 0:00am, and checking if all `Person`s in `selectedPeople` are available at this time. Then it moves to Monday, 0:01am, all the way to Sunday, 11:59pm, storing each time that all members of `selectedPeople` are available.

This `IntList` is used as input for the `timeFrames` function, which returns a `PVector` array list, where every x coordinate is the start of everyone's availability, and it's corresponding y coordinate marks its end. This means that imputing $\{61730, 61731, 61732, 61733, 61750, 61751, 61752\}$ would return $(61730, 61733), (61750, 61752)$ - the time frames of when everyone is available. It may seem counterintuitive to translate time frames to an array of points in time, and then translate it back to time frames, but it is a solution to the issue that arrises when two (or more) overlapping time frames don't have the same start and end point. Once, the overlapping time frames have been found, `numberToDayAndTime` is used to translate the time frame numbers to strings stating the day and time which the number represents. This string is displayed in the program window when the "Find Times" `Button` is clicked, as seen in Figure 2.