CS3560 Lecture 15: The Project

What's Coming Up in this Class?

- We've covered the basics of Object Oriented Design
- The remaining lectures cover the implementation of a system using the Object Oriented Models that we have written.
- Much of the rest of this class will be focused on your building a task management system, PSS
- Some of our upcoming classes will be lab classes where you can ask questions, get help, show your progress
- You will be working in groups to build PSS
- You are free to form your own groups, but everyone needs to be in a group (3-6 people per group). I may need to assign people to your group.
- You can use any language you want: C++, Java, Python, C#

Details Matter

- One part of working on a large project, or working with a team, is to follow the instructions.
- Assume that if something is written in the instructions, it is important, and should be followed.
- If you have questions understanding what the instructions are saying, ask.
- I *think* these instructions are clear and unambiguous, but you might disagree! If anything is unclear, ask.
- If you are building a project for a company, and you don't follow the specification, the project will not work, and you will fail!

More Details

- This lecture adds some details, features, and requirements to the PSS documentation in Homework 3.
- Follow what this lecture says.
- Ideally, this lecture provides all of the product specification that you will need.

Evaluation

- You will be evaluated in a couple of ways:
  - I will give you some sample test cases for you to use as you are building your project.
  - You will demonstrate your project to the class.
  - There will be a final evaluation with the following protocol:
    - I will provide some new test cases.
    - You will run these test cases through your PSS.
    - You will submit the outputs generated by your PSS.
    - I will run your outputs through an evaluation program that will grade your project.
  - Finally, your group will submit the source code for your PSS. A portion of your grade will reflect how your code follows good programming practices:
    - Organization of the code
    - Comments
    - Names and coding style

PSS Overview

- PSS is a tool that will assist the user in scheduling activities.
- Each of these activities is a 'task'.
- Each task has:
  - A name, which is a user-selected text string.
  - A type, also a string, but from a defined list, which will be provided on the next slide.
  - A start time, rounded to the nearest 15-minutes, expressed as a floating point number of a 24-hour clock. For example, a task that starts at 8:30 am would have a start time of 8.5, and a task that starts at 7:45 pm would have a start time of 19.75.
  - A duration, rounded to the nearest 15 minutes, expressed as a floating point number giving the number of hours. A task that is one hour and 45 minutes would have a duration of 1.75.
  - Dates are represented as integers, of the form YYYYMMDD, so April 14, 2020 would be 20200414.
  - For this project, ignore daylight savings time.

Task Types

- The following strings give the task 'type' values. Note that the spelling and capitalization of these strings must be correct!

Recurring Task	Transient Task	Anti-Task
"Class"	"Visit"	"Cancellation"
"Study"	"Shopping"	
"Sleep"	"Appointment"	
"Exercise"		
"Work"		
"Meal"		

Recurring Tasks

- A recurring task has these attributes:
  - Name
  - Type
  - StartDate
  - StartTime
  - Duration
  - EndDate
  - Frequency: 1 (daily), 7 (weekly)

Transient Tasks

- A transient task has these attributes:
  - Name
  - Type
  - Date
  - StartTime
  - Duration

Anti-Tasks

- A anti-task has these attributes:
  - Name
  - Type
  - Date
  - StartTime
  - Duration
- Note that an anti-task is used to cancel out one repetition of a recurring task. Consequently, there must be a recurring task that is scheduled for the given date, and that recurring task must have a start time that matches the start time of this anti-task. Similarly, the durations of the task and anti-task must be the same.

Note About Anti-Tasks

- If an anti-task is present that eliminates one recurrence of a recurring task, then it would be possible to have one or more transient tasks that would have overlapped or partially overlapped with the original task.

Restrictions and Limitations on Tasks

- The start time for a task must be a positive number from 0 (midnight) to 23.75 (11:45 pm). Again, the numbers are rounded to the nearest 15 minutes (0.25).
- The duration of a task must be a positive number from 0.25 to 23.75, rounded to the nearest 15 minutes.
- Two tasks overlap if there is any amount of time when both tasks are 'active'.
- If one task starts just as another task is ending, these tasks do *not* overlap.
- The names given for tasks must be unique.

Restrictions and Limitations on Dates

- Again, dates are integers of the form YYYYMMDD. The year can be any 4-digit number. The month value must range from 01 to 12, and the day must range from 01 through the number of days in that month (for that year).
- For a recurring task, the End Date cannot be the same or earlier than the Start Date.
- For a recurring task, the End Date does not have to be a date in which the task would recur. For example, if this is a weekly recurring task with a start date of 20200110 (January 10th), it could have an End Date of 20200505 (May 5th), even though the task would not normally be on that date.

Basic PSS Operation

- PSS holds the user's *schedule*, which is a list of all the tasks for that user.
- When the program starts, its schedule is empty.
- The user can perform the following operations:
  - Create a task
  - View a task
  - Delete a task
  - Edit a task
  - Write the schedule to a file
  - Read the schedule to a file
  - View or write the schedule for one day
  - View or write the schedule for one week
  - VIew or write the schedule for one month

Create a Task

- To create a task, the user selects the type of task and gives the attribute values.
- PSS will verify that the name of the task is unique. If it is not, PSS will give an error message, and will not create the task.
- PSS will verify that the task does not overlap any existing task in the system.
  - Be sure to consider tasks that wrap past midnight.
  - Be sure to consider recurring tasks, where some instances might overlap while others don't.
  - Be sure to consider anti-tasks, which might eliminate what would otherwise be an overlap.
  - There can be only one anti-task cancelling out a particular instance of a recurring task.
  - When creating an anti-task, make sure that the anti-task matches up with an instance of a recurring task.
- If the information for the task is correct with no conflicts, the task is created.

Find a Task

- The user can search for a task by name.
- If a task is found with the given name, PSS displays the information about the task.

Delete a Task

- The user can delete a task by name.
- If the task is found, PSS will delete that task.
- However:
  - If this is a recurring task, then any anti-tasks associated with any occurrences of this task are also deleted.
  - If this is an anti-task, and deleting this anti-task would leave a conflict between two recurring tasks, or between a recurring task and a transient task, the anti-task will not be deleted, and an error message will be generating. Perhaps the message will give the name of the two tasks that would conflict if the anti-task were removed.

Edit a Task

- The user can identify by name a task to be edited.
- If the task is found, PSS brings up an editor for that task.
  - The editor displays the current attribute values for the task.
  - Upon submission of any edits, PSS will verify that all of the new attribute values are acceptable.
  - PSS will also verify that the edit does not cause any conflict with other tasks in the system.
  - If there are any errors, PSS displays an appropriate error message, and makes no changes to the task.

Write the Schedule to a File

- When the user runs this command, PSS asks for a file name.
- If the name is valid, PSS will write the schedule to the file using the JSON format (as described in a couple of slides).

Read the Schedule from a File

- When the user runs this command, PSS asks the user for a file name.
- The program verifies that a file with that name exists.
- This file should hold a valid schedule in JSON format.
- PSS will read each task from the file, adding that task to the schedule.
- However:
  - If there is ANY error in the file, then the read is terminated, and all changes to the schedule are removed. The changes are only accepted if there are no errors.
  - If the syntax of the file is not value JSON, this would be an error.
  - If any of the tasks given in the file have invalid or inconsistent values, this would be an error.
  - If any tasks would overlap after reading the file, this would be an error.

JSON Schedule File Format

- The JSON file for a schedule will be an array of objects, where each object is one task. So the overall syntax of the file would be:

```
[
  <task> ,
  <task> ,
  <task>
]
```

- The array starts with a '[', ends with a ']', and there are commas between each of the tasks. Note that there is *not* a comma after the last task.

JSON Schedule File Format

- Each of the tasks is represented as a JSON object. The following shows a typical transient task:

```
{
  "Name" : "Go to Store for socks",
  "Type" : "Shopping",
  "Date" : 20200415,
  "StartTime" : 10.25,
  "Duration" : 0.75
}
```

JSON Schedule File Format

- Note that the object starts with a '{', ends with a '}', and there are commas between each of the key/value pairs (but not after the last one).
- Each attribute is a key (a string within quotes), followed by a colon, followed by the value (either a string within quotes or a number).
- Note that the key names have to be correct! The checking program will verify that the keys have exactly these strings.
- The key/value pairs can appear in any order.

JSON Schedule File Format

- A typical recurring task:

```
{
  "Name" : "Take a walk",
  "Type" : "Exercise",
  "StartDate" : 20200415,
  "StartTime" : 10.25,
  "Duration" : 0.75,
  "EndDate" : 20200715,
  "Frequency" : 7
}
```

JSON Schedule File Format

- A typical anti-task:

```
{
  "Name" : "I really didn't want to walk",
  "Type" : "Cancellation",
  "Date" : 20200415,
  "StartTime" : 10.25,
  "Duration" : 0.75
}
```

View Schedule for a Day, a Week, or a Month

- The user can ask to see a schedule for a particular day, a week, or a month.
- The user enters the start date of the time period.
- PSS will list all of the tasks, sorted by date and time, in a format chosen by the designer.
- Note that if there is an anti-task, both it and the recurring task instance it cancels will *not* be displayed.

Write Schedule for a Day, a Week, or a Month

- The user can ask for PSS to write a schedule for a particular day, a week, or a month.
- The user enters a file name and the start date of the time period.
- PSS will then write all of the tasks for that time period, in sorted order, to the JSON-formatted output file.
- Note that if there is an anti-task, both it and the recurring task instance it cancels will *not* be included in the list.
- A recurring task will not be displayed as a single entry. Rather, each *instance* of the recurring task that appears in the time period will be displayed.
- The format matches that of the schedule file, except that all of the transient tasks or recurring instances will appear using the transient task format.

Test Cases

- I will provide some test cases for you to use while building the system.
- Each test case will consist of:
  - An input JSON file
  - A script of actions to perform
  - An output JSON file
- The script will say things like "create *this* task" or "delete *that* task".
- Some of the actions will actually be incorrect, such as incorrect values or missing or overlapping tasks. The script will point out that this is an error, you can check to make sure your PSS catches the error.
- After performing the actions, write the schedule to an output JSON file, then compare your output with the expected output.

Conclusion

- This may seem like a lot of work, but:
  - You are building this as a team.
    - Partition the work
    - Use what we learned in class!
  - This will be your homeworks 3 and 4
  - If the class demonstrates that you as a group have a solid understanding of Object Oriented Design and Implementation, then perhaps the Final can be a bit simpler…
- Ask questions!
- Don't put this off...get started on it!