Task Management and Assignment System - Deliverable 4

Daniel Orejuela Liu

Vanier College

Data Structures and Object Oriented Programming
420-SF2-RE sect. 00001

Presented to Yi Wang

Table of Contents

Project Description	3
Program Features	3
Code Examples of Key Required Functionalities	6
Challenges	10
Learning Outcomes	10
New UML Class Diagram	11

Project Description

- The task management and assignment system is a java application designed for users to receive tasks to do from administrators. Standard users will have a list of tasks attributed to them in their task manager, and admins can assign them tasks to do, adding to that list. The tasks can then be sorted according to their priority, their due date, or a mix of both, and they come in two formats : regular tasks (tasks to repeat on a scheduled basis) and non regular tasks (one-time tasks). It meets the requirements of having :
- Class hierarchies with two layers (Tasks, RegularTask, NonRegularTask and SuperUser, Admin, and User)
- A user defined interface with abstract methods (TaskOperations)
- Runtime polymorphism (getExportData() in Task and SuperUser and their child classes)
- TextIO (importing and exporting users and tasks from and into CSV files, through the TaskManager class)
- Comparable and Comparator (Task extend comparable and Task.TaskComparator is a comparator)
- JUnit testing for all user-defined methods (excluding getters and setters)

Program Features

Class Hierarchies and their Methods

Task (abstract class)

- fromExportData(String) : Task (static) - creates a Task object according to a CSV formatted string of its data
- reschedule(LocalDateTime) : boolean - attempts to reschedule and returns the success of the attempt
- compareTo(Task) implements Comparable<Task> - compares tasks by priority
- getExportData(Task) : String - returns a CSV formatted string of the Task's data

RegularTask (subclass of Task)

- skip(int) : LocalDateTime - reschedules the task as if it has been skipped x times
- refresh() : LocalDateTime - reschedules the task as if it has been completed
- getExportData() : String - gets the CSV formatted string of the RegularTask's data

NonRegularTask (subclass of Task)

- closeTask() : void - sets the task's status as completed, lowering its priority and changing the title to show its completion
- getExportData() : String - gets the CSV formatted string of the NonRegularTask's data

SuperUser (abstract class)

- fromExportData(String) : Task (static) - creates a SuperUser object according to a CSV formatted string of its data
- getExportData(Task) : String - returns a CSV formatted string of the SuperUser's data

Admin (subclass of SuperUser)

- checkUserTasks(User) : List<Task> - checks the tasks a user has
- giveUserTask(User) : boolean - attempts to give the user a task and returns the success of the attempt
- getExportData() : String - gets the CSV formatted string of the Admin's data

User (subclass of SuperUser)

- clearTasks() : void - empties the list of tasks in the user's task manager
- setTaskCompleted(Task) : boolean - sets a NonRegularTask in the user's task manager as being completed, returning the success of the attempt
- getExportData() : String - gets the CSV formatted string of the User's data

Supporting / Other classes

TaskOperations (interface)

- addTask(Task) : boolean - adds a task to a list of tasks
- deleteTask(Task) : boolean - deletes a task from the list of tasks
- sortTasks(TaskComparator) - sorts the list of tasks with the TaskComparator

TaskManager (implements TaskOperations interface)

- getTasksToDo(TaskComparator) : Queue<Task> - returns a queue of the tasks to do sorted in order of urgency according to the TaskComparator
- exportTasks(String) : void - writes a the tasks in the TaskManager into a CSV file with the corresponding file name as the argument
- importTasks(String) : List<Task> - returns a list of tasks according to the CSV file (from the file's name as argument) with stored task data
- searchForTasks(String) : List<Task> - returns a list of tasks (from its own tasks) that have the corresponding keywords in their title (not case sensitive)
- searchForTasks(String, boolean) : List<Task> - returns a list of tasks (from its own tasks) that have the corresponding keywords in their title (not case sensitive) and that have the same completion status as the boolean argument
- exportTasks(Collection<Task>, String) : void (static) - writes a the Tasks into a CSV file with the corresponding file name as the argument

TaskComparator (public static class in the Task class)

- Compare(Task, Task) : int - compares both tasks with respect to the comparator's taskCompareType field.
- minutesBetweenDates(LocalDateTime, LocalDateTime) : int - gives the amount of minutes between the two dates, positive if the first one is earlier and negative if the first one is later.

TaskCompareType (enum used for TaskComparator)

- Priority, DueDate, Default

Code Examples of Key Functionalities

Runtime polymorphism with getExportData() methods in Task and SuperUser. To obtain the CSV format of the child classes of Task and SuperUser, getExportData() is used and overridden in the child classes to export the appropriate data, which can later be read and converted back.

Example with Task's getExportData():

```
User user1 = new User( name: "joe");
Admin admin1 = new Admin( name: "bob");


Task regularTask = new RegularTask( title: "Wipe the floor", description: "with a mop",
        taskPriority: 1, LocalDateTime.now().plusDays(2), timeIntervalHours: 72);
Task nonRegularTask = new NonRegularTask( title: "Programming homework", description: "use Intellij",
        taskPriority: 3, LocalDateTime.now().plusDays(1));


admin1.giveUserTask(user1, regularTask);
admin1.giveUserTask(user1, nonRegularTask);

for (Task task : admin1.checkUserTasks(user1)) {
    System.out.println(task.getExportData());
}
```

Output:

```
Wipe the floor,with a mop,1,2025-05-13T20:14:13.983736200,72.0,RT
Programming homework,use Intellij,3,2025-05-12T20:14:13.986728200,false,NRT
```

Example with SuperUser's getExportData():

```
User user1 = new User( name: "joe");
Admin admin1 = new Admin( name: "bob");

Task regularTask = new RegularTask( title: "Wipe the floor", description: "with a mop",
        taskPriority: 1, LocalDateTime.now().plusDays(2), timeIntervalHours: 72);
Task nonRegularTask = new NonRegularTask( title: "Programming homework", description: "use Intellij",
        taskPriority: 3, LocalDateTime.now().plusDays(1));

admin1.giveUserTask(user1, regularTask);
admin1.giveUserTask(user1, nonRegularTask);

List<SuperUser> SuperUsers = new LinkedList<SuperUser>();
SuperUsers.add(user1);
SuperUsers.add(admin1);

for (SuperUser superUser : SuperUsers) {
    System.out.println(superUser.getExportData());
}
```

Output:

```
joe,0,Wipe the floor,with a mop,1,2025-05-13T20:17:36.424474700,72.0,RT,Programming homework,use Intellij,3,2025-05-12T20:17:36.428463700,false,NRT
bob,1,Ad
```

TextIO writing and reading in TaskManager**.** Users can write and read CSV files of their tasks, and users can also be written and read from CSV, while conserving their tasks.

Example of writing data through TextIO using the exportTasks method in TaskManager:

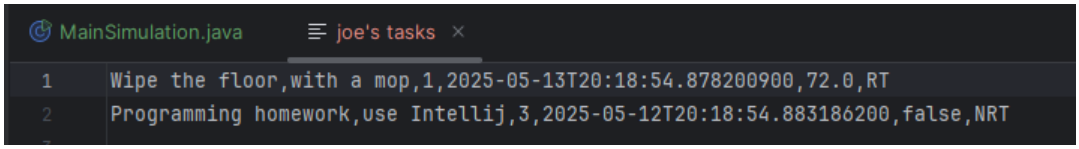
```
User user1 = new User( name: "joe");
Admin admin1 = new Admin( name: "bob");

Task regularTask = new RegularTask( title: "Wipe the floor", description: "with a mop",
        taskPriority: 1, LocalDateTime.now().plusDays(2), timeIntervalHours: 72);
Task nonRegularTask = new NonRegularTask( title: "Programming homework", description: "use Intellij",
        taskPriority: 3, LocalDateTime.now().plusDays(1));

admin1.giveUserTask(user1, regularTask);
admin1.giveUserTask(user1, nonRegularTask);

user1.getTaskManager().exportTasks( fileName: "joe's tasks");
```

Output :



Example of Text IO reading from a CSV file with the static TaskManager's importTask method:

```
List<Task> importedTasks = TaskManager.importTasks( fileName: "joe's tasks");
for (Task task : importedTasks) {
    System.out.println(task);
}
```

Output :

```
RegularTask{timeIntervalHours=72.0}Task{title='Wipe the floor', description='with a mop', taskPriority=1, dueDate=2025-05-13T20:22:44.848794200}
NonRegularTask{completed=false}Task{title='Programming homework', description='use Intellij', taskPriority=3, dueDate=2025-05-12T20:22:44.854778400}
```

Sorting with Comparable. Tasks can be sorted with their implementation of the Comparable<Task> interface, which sorts them in ascending order based on their priority.

Example of sorting with comparable for ascending order of priority :

```
User user1 = new User( name: "joe");
Admin admin1 = new Admin( name: "bob");

Task regularTask = new RegularTask( title: "Wipe the floor", description: "with a mop",
        taskPriority: 1, LocalDateTime.now().plusDays(2), timeIntervalHours: 72);
Task nonRegularTask = new NonRegularTask( title: "Programming homework", description: "use Intellij",
        taskPriority: 3, LocalDateTime.now().plusDays(1));

Task regularTask2 = new RegularTask( title: "Organize the room", description: "make it clean",
        taskPriority: 2, LocalDateTime.now().plusDays(5), timeIntervalHours: 24 * 7);
admin1.giveUserTask(user1, regularTask);
admin1.giveUserTask(user1, nonRegularTask);
admin1.giveUserTask(user1, regularTask2);

Collections.sort(user1.getTaskManager().getTasks());
for (Task task : user1.getTaskManager().getTasks()) {
    System.out.println(task);
}
```

Output :

```
RegularTask{timeIntervalHours=72.0}Task{title='Wipe the floor', description='with a mop', taskPriority=1, dueDate=2025-05-13T20:27:59.389590300}
RegularTask{timeIntervalHours=168.0}Task{title='Organize the room', description='make it clean', taskPriority=2, dueDate=2025-05-16T20:27:59.392582700}
NonRegularTask{completed=false}Task{title='Programming homework', description='use Intellij', taskPriority=3, dueDate=2025-05-12T20:27:59.392582700}
```

Sorting with Comparator. Tasks can be sorted with TaskComparator, which implements Comparator<Task> to be able to compare and sort tasks in order of their urgency (which ones to complete first) with respect to their due dates, their priority, or a mix of both.

Example sorting with TaskComparator with respect to priority:

```
User user1 = new User( name: "joe");
Admin admin1 = new Admin( name: "bob");

Task regularTask = new RegularTask( title: "Wipe the floor", description: "with a mop",
        taskPriority: 1, LocalDateTime.now().plusDays(2), timeIntervalHours: 72);
Task nonRegularTask = new NonRegularTask( title: "Programming homework", description: "use Intellij",
        taskPriority: 3, LocalDateTime.now().plusDays(1));

Task regularTask2 = new RegularTask( title: "Organize the room", description: "make it clean",
        taskPriority: 2, LocalDateTime.now().plusDays(5), timeIntervalHours: 24 * 7);
admin1.giveUserTask(user1, regularTask);
admin1.giveUserTask(user1, nonRegularTask);
admin1.giveUserTask(user1, regularTask2);

user1.getTaskManager().sortTasks(new Task.TaskComparator(Task.TaskComparator.TaskCompareType.Priority));

user1.getTaskManager().getTasks().forEach(System.out::println);
```

The method called for sorting, which uses the sort(Comparator) method of a list.

```
@Override  3 usages  DevManDan1178
public void sortTasks(Task.TaskComparator sortBy) {
    tasks.sort(sortBy);
}
```

Output :

```
NonRegularTask{completed=false}Task{title='Programming homework', description='use Intellij', taskPriority=3, dueDate=2025-05-12T20:31:22.514119100}
RegularTask{timeIntervalHours=168.0}Task{title='Organize the room', description='make it clean', taskPriority=2, dueDate=2025-05-16T20:31:22.514119100}
RegularTask{timeIntervalHours=72.0}Task{title='Wipe the floor', description='with a mop', taskPriority=1, dueDate=2025-05-13T20:31:22.511128600}
```

Challenges

My main challenge was coming up with ideas for the project and planning it. My deliverable 1's plan was missing one hierarchy, so I decided to add modifications to the project with a new file called Modifications.txt, where I would log my modifications. I decided to add a SuperUser class with child classes Admin and User, but this addition meant I had to change my project's goal (what it was made for), so instead of being a task management app for individuals, it became a task management and assignment app that could be used by a company for example.

I was also unfamiliar with constantly using JUnit testing, since I would usually test my code in a main file, where I would read the results through printing in the system's output.

I succeeded in implementing the features I have planned for. However, in retrospect, my project would have been better with more features, like having a system to store the users and the admins, which could then export users and admins. Although I currently have getExportData() for SuperUser objects, there are no methods that explicitly use it. I also feel like a system that stores the admins and the users would allow for more functionality, like allowing the search for specific users and admins (after adding a name field for SuperUser) and having a global task manager from where admins could assign users tasks.

Learning Outcomes

While working on the project, I learned a few useful tricks, such as the existence of LocalDateTime's .plusHours, plusDays, plusYears, … methods, which helped me create different LocalDateTime objects. I also learned during testing that LocalDateTime.new() declared in different lines are not equal even though they were created at virtually the same time (during the same block of code execution). It was also my first time trying to convert data into CSV format where a part of the data to convert was already in CSV format and had variable size. In the end, I had to replace the comma character by an ASCII square character, which let me separate each CSV data in the new CSV so I could read it back again.

This project helped me learn about using Github properly and familiarized me with using Github for version control and committing code onto it with comments to log the progression of the project and the added modifications.

This project was also my first time creating UML class diagrams and helped me learn more about planning and developing projects, debugging code, and using JUnit testing while doing so.

New UML Class Diagram