Project Report

Group 014
COMP2021 Object-Oriented Programming (Fall 2023)
Author: Raynell Lu Soon Hong
Other group members:
KONG Zirui
LIN Zhanzhi
ZHEN ZHANG Alex

1 Introduction

This document describes Group 014's design and implementation of a command-line-based task management system. The project is part of the course requirement of COMP2021 Object-Oriented Programming at PolyU.

2 My Contribution and Role Distinction

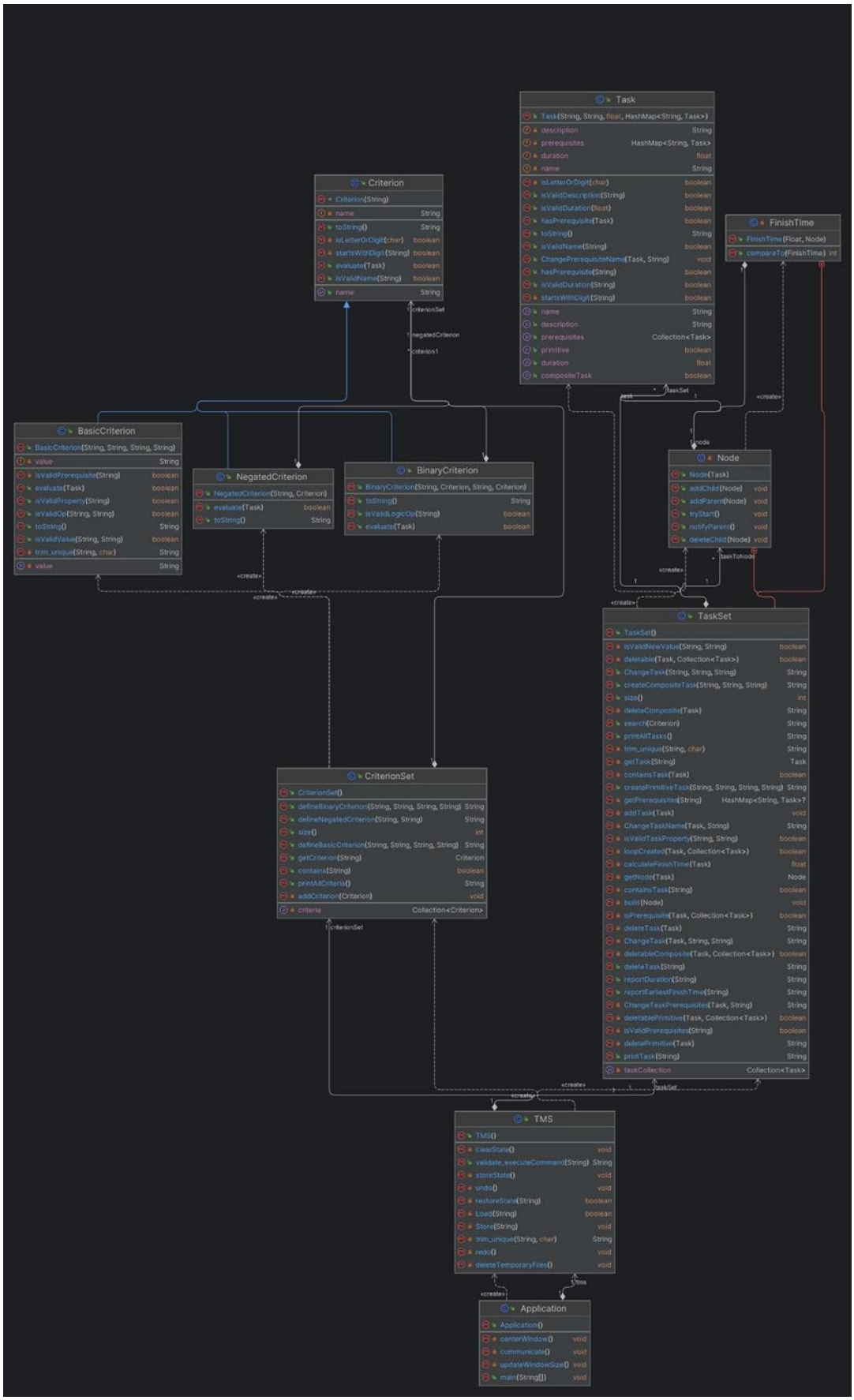
Through the experience of this group project, I have learned a lot from my peers as well as an opportunity to utilize the knowledge taught in class. The contributions I made in this project include co-coding the test files along with Kong Zirui, wrote some of the sections for the user manual, worked on the design of the power point slide, and designed and wrote the report structure along with Kong Zirui and Alex Zhen Zhang. During this project, I do not have a definite role. I am considered to be the flexible member of the group and can be placed to work on any task, I try to fill in gaps and work on other task while my teammates are working on other tasks.

3 A Command-Line Task Management System

3.1 Design

Our System is composed of 11 classes, which have 2 *public* class (*TMS, Application*) and 9 *private* classes (*Task*, *TaskSet*, *Node*, *Criterion*, *BasicCriterion*, *BinaryCriterion*, *NegatedCriterion*, *CriterionSet* and *FinishTime*). The system is designed adhering to the MVC (Model-View-Control) Pattern. The public class TMS serves as the model unit in the system which it gets manipulated by the controller unit which are the other private class in the system, and which this function allows users to perform actions to the model. As for the viewing component of the system which is the GUI, in which this component will display the resulting output of the user input.

Diagram for reference



3.2 Requirements

[REQ1] The system should support creating primitive tasks.

1) The requirement is implemented.

2) Implementation details - *CreatePrimitiveTask* is created under the TaskSet Class, where it takes (String, String, String, String) as parameters. Once the user creates primitive task in the GUI, the user will be informed that the primitive task has been created or not.

3) Error conditions and how they are handled – Errors such as invalid parameters, invalid format, duplicate tasks will be noticed to the user. The methods responsible for checking and informing the user are checking input length, *isValidTaskName, containsTask, isValidDescription, isValidDuration, isValidTaskPrerequisite.*

[REQ2] The system should support creating composite tasks.

1) The requirement is implemented.

2) Implementation details – *CreateCompositeTask* is created under the TaskSet Class, where it takes (String, String, String) as parameters, Once the user creates composite task in the GUI, the user will be informed that the composite task has been created or not.

3) Error conditions and how they are handled – Errors such as invalid parameters, invalid format, duplicating subtask and invalid subtask will be noticed to the user. The methods responsible for handling those errors are checking input length, *isValidTaskName, containsTask, isValidDescription, isValidTaskSubtask.*

[REQ3] It should support deleting tasks.

1) The requirement is implemented.

2) Implementation details – DeleteTask is created under TaskSet Class, where it takes (Task) as parameters, Once the user deletes task in the GUI, the user will be informed that the command has been processed or not

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format, non-existent tasks and non-deletable tasks will be noticed to the user. The methods responsible for handling those errors are checking input length, *isValidTaskName, containsTask.*

[REQ4] It should support changing the properties of an existing task.

1) The requirement is implemented.

2) Implementation details – ChangeTask is created under TaskSet Class, where it takes (Task, String, String) as parameters, Once the user change task in the GUI, the user will be informed that the command has been processed or not

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format, non-existent tasks and invalid task properties will be noticed to the user. The methods responsible for handling those errors are checking input length, *isValidTaskName, containsTask, isValidTaskProperty, isValidNewValue.*

[REQ5] It should support printing the information of a task.

1) The requirement is implemented.

2) Implementation details - PrintTask is created under TaskSet Class, where it takes (String) as parameters, Once the user print task in the GUI, the user will be informed that the command has been processed or not.

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format, non-existent tasks will be noticed to the user. The methods responsible for handling those errors are checking input length, *isValidTaskName, containsTask.*

[REQ6] It should support printing the information of all tasks.

1) The requirement is implemented.

2) Implementation details - PrintAllTask is created under TaskSet Class, where it is a void function, Once the user print all task in the GUI, the user will be informed that the command has been processed or not.

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format, will be noticed to the user. The methods responsible for handling those errors are checking input length.

[REQ7] It should support reporting the duration of a task.

1) The requirement is implemented.

2) Implementation details – ReportDuration is created under TaskSet Class, where it takes (String) as parameter, Once the user type reportduration in the GUI, the user will be informed that the command has been processed or not.

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format will be noticed to the user. The methods responsible for handling those errors are checking input length, *isValidTaskName, containsTask.*

[REQ8] It should support reporting the earliest finish time of a task.

1) The requirement is implemented.

2) Implementation details – reportearliestfinishtime is created under TaskSet Class, where it takes (String) as parameter, Once the user gets the reportearliestfinishtime in the GUI, the user will be informed that the command has been processed or not.

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format will be noticed to the user. The methods responsible for handling those errors are checking input length, *isValidTaskName, containsTask.*

[REQ9] It should support defining basic task selection criteria.

1) The requirement is implemented.

2) Implementation details – defineBasicCriterion is created under CriterionSet Class, where it takes (String, String, String, String) as parameter, Once the user gets define basic criterion in the GUI, the user will be informed that the command has been processed or not.

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format, non-existent criterion, invalid criterion property will be noticed to the user. The methods responsible for handling those errors are checking input length, *isValidCriterionName, containsCriterion, isValidCriterionProperty, isValidOp, isvalidValue.*

[REQ10] It should support a criterion to check whether a task is primitive.

1) The requirement is implemented.

2) Implementation details – isPrimitive is criterion created under CriterionSet, where it is a method to check whether the task is primitive or not, to check, the criterion simply checks the duration of the task.

3) Error conditions and how they are handled.

[REQ11] It should support defining composite task selection criteria.

1) The requirement is implemented.

2) Implementation details - DefineNegatedCriterion and DefineBinaryCriterion are created under CriterionSet class, where it takes (String, String) and (String, String, String, String) as parameter respectively, Once the user gets define composite task selection criteria in the GUI, the user will be informed that the command has been processed or not.

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format, non-existent criterion, invalid logical operator will be noticed to the user. The methods responsible for handling those errors are checking input length, *isValidCriterionName, containsCriterion, isValidOp.*

[REQ12] The tool should support printing all defined criteria.

1) The requirement is implemented.

2) Implementation details – printAllCriteria is created under CriterionSet class, where it is a void function, Once the user print all defined criteria in the GUI, the user will be informed that the command has been processed or not.

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format, will be noticed to the user. The methods responsible for handling those errors are checking input length.

[REQ13] The tool should support searching for tasks based on an existing criterion.

1) The requirement is implemented.

2) Implementation details - Search is created under TaskSet class, where it takes (Criterion) as parameters, Once the user search task in the GUI, the user will be informed that the command has been processed or not.

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format, invalid criterion name, non-existent criterion will be noticed to the user. The methods responsible for handling those errors are checking input length, *isValidCriterionName, containsCriterion.*

[REQ14] The tool should support storing the defined tasks and criteria from a file.

1) The requirement is implemented.

2) Implementation details - Store is created under TMS class, where it takes (string) as parameters, Once the user store file in the GUI, the user will be informed that the command has been processed or not.

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format, and invalid path will be noticed to the user. The methods responsible for handling those errors are checking input length.

[REQ15] The tool should support loading tasks and criteria from a file.

1) The requirement is implemented.

2) Implementation details - Load is created under TMS class, where it takes (string) as parameters, Once the user load file in the GUI, the user will be informed that the command has been processed or not.

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format, and invalid path will be noticed to the user. The methods responsible for handling those errors are checking input length.

[REQ16] The user should be able to terminate the current execution system.

1) The requirement is implemented.

2) Implementation details - Quit is created under TMS class, where it is void fucntion, Once the user quit, the command will proceed

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format will be noticed to the user. The methods responsible for handling those errors are checking input length.

[BON1] Support for a graphical user interface (GUI) to visually show the tasks created or selected and their relationship.

1) The requirement is implemented.

2) Implementation details – The GUI is created under the Application Class, where it contains 4 different methods. The GUI consist of display panel as well as an input bar.

3) Error conditions and how they are handled.

[BON2] Support for the undo and redo of all the required commands except PrintTask, PrintAllTask, ReportDuration, ReportEarliestFinishTime, PrintAllCriteria, Search, Store, Load and Quit.

1) The requirement is implemented.

2) Implementation details - redo and undo are created under TMS class, where both are void functions, Once the user undo or redo in the GUI, the user will be informed that the command has been processed or not.

3) Error conditions and how they are handled - Errors such as invalid parameters, invalid format will be noticed to the user. The methods responsible for handling those errors are checking input length.