Project Report

Group 11

COMP2021 Object-Oriented Programming (Fall 2023)

Author: Bailin ZENG 21107431D

Other group members:

Xin DAI

Ileana Pal

Caryse CHONG Jia Wern

1. Introduction.

This document describes Group 11’s design and implementation of a command-line-based task management system. The project is part of the course COMP2021 Object-Oriented Programming at PolyU.

1. My Contribution and Role Distribution

My primary responsibility was contributing to the system architecture and implementing requirements 1 to 8. I participated in the brainstorming sessions to define the overall structure and design of the command-line task management system (TMS). I created a system architecture that could handle task creation, modification, and deletion (basically the task system). I implemented the functionalities to support the creation of simple and composite tasks, deleting tasks, and modifying task properties. I ensured that the system adhered to the defined requirements and that the code was well-organized and maintainable.

1. A Command-Line Task Management System

Overview:

We have implemented the concept of Model-View-Controller (MVC) in our system.

The TMS class is the View. It is the one that communicates to the users. It takes input from the command lines and outputs the result to the command lines. Inside this class, the input, output, and controller are separated into different methods, which allows it to be tested quickly.

The four operation classes, CheckAvailability, TaskOperation, CriterionOperation, and FileOperation are the Controllers. These files contain multiple static methods that handle the logical control flow of the input and output. These controllers are also responsible for connecting the View and the Model.

图示

描述已自动生成

* 1. Design

1. TMS.java:

TMS is the entrance and exit of the task management system. It can be instantiated. The method run() is the one to start the system. This module can be further extended to support concurrent programming by letting it extend the Thread class.

There is also a test run method for testing multiple command lines input as String arrays.

Inside this module, the operation method is the access to the Controller. It will create a general model of StorageLists and send it to the controller. This is something that we should improve in the future. The View module should not be dealing with any Model part.

1. TaskOperation.java:

TaskOperation.java is part of the Controller. It consists of static methods since a procedure-oriented operation should be static, and should not be able to be instantiated.

It takes input from the View, and outputs raw input data, such as the String variable, into the CheckAvailability. After getting a converted type from the CheckAvailability, it creates or modifies the model.

It will return the result of the operation back to the View in the form of Strings.

1. CriterionOperation.java:

CriterionOperation.java serves the same control function as TaskOperation, except it deals with the operation of criteria.

1. FileOperation.java:

This Java file contains only Store and Load static methods that allow the user to store the StorageLists model in the form of files.

1. CheckAvailability.java:

This is a checker for checking most of the raw inputs to see if they are available or legal. It consists of static methods. It also helps convert the input into the model. For example, it helps to turn a Task name into a Task object by calling the searching method in the model StorageLists.

This class also throws out most exceptions when the input is illegal. The exceptions will be printed out as output in the View.

1. StorageLists.java:

This is the class that packs up all the data. It is considered the central part of the Model. It consists of two ArrayLists, one storing the task, and the other storing the Criteria. These two lists are encapsulated, and can only be accessed through some specific methods. It also overrides the toString method for the implementation of file storage and print functions.

1. Task.java, PrimitiveTask.java and CompositeTask.java:

Task is an abstract class inherited and implemented by PrimitiveTask and CompositeTask. They have implemented the essential operation for a Task, such as storing the name, description, etc. The prerequisite Tasks and subtasks are stored as ArrayList in each task class.

The CheckDuration, toString, etc, are implemented differently in PrimitiveTask and CompositeTask. We considered this a perfect utilization of Polymorphism to enhance the modulization and reduce the difficulty of redevelopment of our project. We also simplified the code structure a lot by utilizing this Polymorphism feature.

These three classes are part of the Model.

1. Criterion.java, BasicCriterion.java, CompositeCriterion.java, BinaryCriterion.java and NegatedCriterion.java:

The general Criterion class only contains the basic name of a criterion. It also contains a Check method which will be performed differently in each Criterion subclasses.

These are the classes for storing the information of Criteria. The information on property and operand has already been converted to the Enum class in the CheckAvailability.java. The corresponding object is stored in the essential criteria.

The BinaryCriterion contains two Criteria, and the NegatedCriterion includes one.

1. Operand.java, Property.java and LogicOp.java:

These are the helper Enum for establishing and maintaining the Model. It helps to convert String input into objects. Meanwhile, the evaluating functions and checking functions for the criteria are implemented in these Enum classes. Again, Polymorphism is applied for different operands and operators, and the system should return a different result.

* 1. Requirement

Overview: All the requirements except the bonus are implemented. The corresponding control methods are separated into FileOperation.java, TaskOperation.java, and CriterionOperation.java.

1. Req 1, 2:

After the inputs are received from View, which is the TMS, the TaskOperation executes the logic control. The CheckAvailability will check the input. A new task will be created if no exceptions are generated.

Exceptions:

1. Illegal task name: The task name has an illegal format.
2. Illegal description: The task description has an illegal format.
3. Task already exists: We use a name to define the uniqueness of a task, so defining two tasks with the same name is not allowed.
4. Duration must be a number.
5. Duration cannot be negative.
6. The task does not exist: We do not allow the user to create a task with a non-recognized prerequisite or subtask input. A prerequisite or subtask must already exist in the system if the user wants to define a new task based on it.
7. Req 3:

The data flow is the same as previous requirements.

Exceptions:

1. Task not found: The required task to be deleted mush exist in the system to be deleted.
2. The task is the prerequisite for another task. Users cannot delete a task that is the prerequisite of another primitive task.
3. The subtask is the prerequisite of another task. When deleting a composite task, it is required to delete the subtasks that belong to it. If one of its subtasks is the prerequisite of another primitive task, the user cannot delete it.
4. Req 4:

The CheckAvailability.java will help the TaskOperation.java to call the search list function in StorageLists.java. If the task is found, the setProperty method will be executed.

Exceptions:

1. Cannot set duration for a composite task. Duration is the feature of a simple task.
2. Cannot set prerequisites for a composite task and vice versa.
3. A task cannot be a prerequisite or subtask of itself.
4. It is illegal to create a task referencing loop. For example, task t1 is the prerequisite of t2, while t2 is the prerequisite of t3. If we want to set a loop by defining t3 as the prerequisite of t1, it's illegal since we cannot decide the duration of these three tasks.
5. Req 5, 6:

The print single task and all tasks are executed in the TaskOperation.java by calling the toString method in each Task class and the printAllTasks in the StorageList class.

Exception: There are no exceptions for the printing process. The system will output no-task notation if there are no tasks created in the system.

However, it is possible that the input single task name cannot be found in the system.

1. Req 7, 8:

The report duration of a composite task and the report of the earliest finish time are the same. The Polymorphism is utilized so that the getDuration method in the Task class performs differently in these two types of Tasks. The getDuration returns the maximum hours of getDuration result for each task object from its prerequisites or subtasks. This is performed recursively until a task object has no subtasks prerequisites.

Exceptions: There are no exceptions for getting the duration. The only exception that may be thrown is that the task name cannot be found in the system.