Java Fundamentals

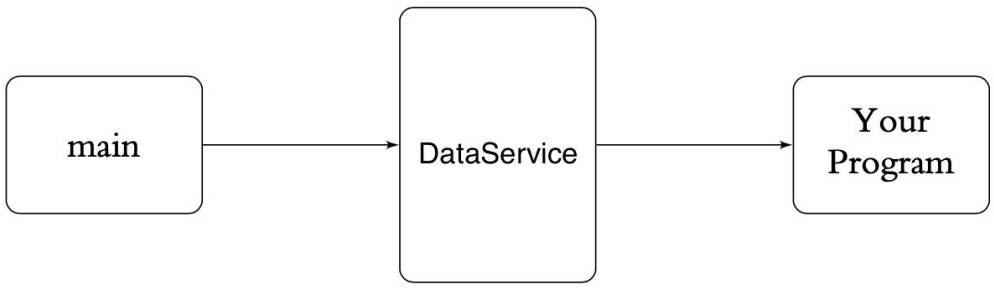
JAVA PROJECT #2
EPITRELLO

Introduction

One of the most important factors for effectively promoting a project is the proper management of tasks in the executive team. In this exercise, we intend to set up a system for managing tasks as well as controlling them. This system is made up of users, lists, and tasks. Each task belongs to a list and can be assigned to a user. In addition, for each task priority and estimated time is intended to help us make better decisions about our assignments and tasks.

Description

The general position of this class is as follows. In the main function, an object is created from the interface class, and then functions of this interface class are called. Note that your implementation details should not be in the interface class. You can add fields and functions if necessary to the interface class. Note that you should not change the signature of the predefined interface class functions at all.



1. DataService class

Introducing each of the interface class functions:

a. Creator

This function is called when we create an object from the data service class.

b. Adding functions

We use these functions to add entity to the system. The functions in this section return the Success string if successful. Possible errors for each function are described in the function description.

i. User

We use this function to add users to the system. Notice the name of each user in the system is a unique field. Returns the “User already exists” if a duplicate name is entered for the user.

ii. List

We use this function to add a list to the system. Notice the name of each list in the system is a unique field. Returns the “List string already exists” if a duplicate name is entered for the list.

iii. Task

We use this function to add a task to a list. The name of each task in the system is a unique field. The approximate time and priority of each are positive. A task with a higher priority number has a higher priority. The task is added to the system when it is not completed. Nor is the newly created task is not assigned to anyone. If there is no list with the name entered in the system, the “List does not exist” and returns “Task already exists” if it is a duplicate task.

c. Deleting functions

We use these functions to delete system entities. The functions in this section return the Success string if successful. Possible errors for each function are described in the function description.

i. List

We use this function to delete the list. If the list is not available in the system, returns the “List does not exist”. Note that deleting the list also eliminates the list tasks.

ii. Task

We use this function to remove the task. Returns the Task does not exist if the task is not present in the system.

d. Functions related to tasks

We use these functions to make changes to tasks. The functions in this section also return the Success string if successful. Also return the Task does not exist string if there are no named tasks. The remaining possible errors of each function are described in the function description.

i. Assign a task

With this function we can delegate a task to a user. In case of user item In the system is not available, the string “User does not exist” returns.

ii. Complete task

Using this function we can get a task done.

iii. Edit task

Using this function we can change what we set for the task at the time of construction.

iv. Move Task

Using this function we can move a task to another list. Returns the "List does not exist" if the destination list is not available in the system.

e. Reporting functions

We use these functions to report to the system. The return value of these functions is a positive integer.

i. Print total estimated time

This function returns the estimated completion time of all tasks. This value is equal to the maximum time required by users to perform their tasks. The time that users need to perform their tasks is equal to the sum of the estimated time of all tasks assigned to that user. Note that this function does not include unassigned tasks.

ii. Print total remaining time

This function returns the estimated time of completion of all tasks performed. This value is equal to the maximum the time required by users to perform their remaining tasks. The time that users need to perform their remaining tasks is equal to the sum of the estimated time of all tasks assigned to that user that has not been completed. Note that this function does not include unassigned tasks.

iii. User workload

This function returns the sum of all user tasks in all lists. If User does not exist in the system, returns 0.

f. Returning functions

The functions in this section return the details of the system entities.

i. Task

This function returns the task details in the following format, Returns the “Task does not exist” if the task is not present in the system.

ii. List

Returns all the tasks in a list in order of creation. In the first line if the item list comment is not available in the system, returns the “List does not exist” string Returns.

iii. All lists

Make all the tasks of all lists in order of the list and in each list the tasks in order, returns the output format of each list is the same as the function of the previous function.

iv. User tasks

Returns all user tasks in the order that they were created. If the user is not in the System, returns the string “User does not exist”.

v. User unfinished tasks

Returns all tasks performed by the user in sequence. Returns the “User does not exist” string if the user is not in the system.

vi. All unassigned tasks

Prioritize all tasks not assigned to the system in the order in which they are equal, returns the desired order.

vii. All unfinished tasks

Prioritize all tasks not assigned to the system in the order in which they are equal, returns the desired order.

viii. Users by workload

This function returns users in ascending order and if equal, return in any order. The amount of work a user does is the estimated time that all of the tasks assigned to that user.

ix. Users by performance

This function renders users in descending order and if equal, return in any order. The performance of the user is equal to the total time estimate of the tasks performed by that user.

Deliverables :

- Private github repo (add **thomasbroussard** and **amirali313** as authorized users on your repo)
- Commented ScreenShots in a separate directory
- UML Class and Activity Diagrams
- Software Architecture diagrams (principal layers and services interactions)

Grading

- Not complete program(some codes written) : 40%
- Correctly using classes and objects : **BONUS** +20%
- Clean code: **BONUS** +10%

- Your ability to design a custom Data Structure to optimize the in-memory representation and access of/to the data. Justify your choice and tell why it is better than the built-in data structures. **BONUS** +30%
- Taken in account :
    - Industrialization (configuration, presence of tests etc.)
    - Code Quality (code comments, javadoc, results of SonarLint plugin)
    - Documentation Quality
    - Find the best choice for your data structure (justify your choice in the documentation).
    - You will be using the same **main** given in the project.
    - Write the sample output into a **file**.
    - You need to register all the users that are added to Epitrello, into a simple **database**.

Notice

- This project is done by pairs of two.
- **NO CHEATING IS ACCEPTED**. If caught, you will get **zero**
- In this exercise, it is important to properly design system entities and their relationships using the concepts learned in the class.
- Deadline is due: 10/02/2020 midnight
- Submit your project via ONLY REPLYING TO THE MAIL THAT I SENT YOU for each homework
- Send me your gitlab repository link.
- Write you names and last names and your group number.