HowTodoit

- a simple to-do list app written in java

1.0 Introduction

HowTodoit is a to-do list app written in java. The user will be able to interact with a virtual task manager (via command line) that is able to load data from and save data into local history (using Java's Serializable interface).

2.0 Statement of scope

The app will be built using Gradle to a JAR file and run in JRE 11. It initially only has a CLI (command line interface). We will add a GUI (and more features) in later phases of the project if time allows.

3.0 Architectural and component-level design

Task:
- The user can add tasks with a (unique) name, a due date, and a description.
- Tasks can be modified (name, due date, description, project) and completed.

Project:
- A project is basically a directory that stores tasks; a task must be in exactly one project.
- By default, all tasks created will be added to a project named "Inbox" at first; it's a default project that cannot be modified or deleted.
- The user can create projects with a (unique) name. User-created projects can be modified (name, tasks) and deleted. When deleted, all tasks stored inside this project will be moved back to "Inbox".

Label:
- A label is similar to a project except that a task can be in zero or more labels.
- The user can add labels with a (unique) name. User-created labels can be modified and deleted.
- The user can add tasks to labels. In particular, they can add tasks to a label named "Starred", which cannot be modified or deleted.

4.0 Use cases and user interface design

The user can use the following (case-sensitive) commands to interact with the system. Note that arguments are separated by the character ";".

General commands:
- *upcoming* Show all upcoming tasks in all projects in chronological order.
- *exit* Quit the program and save data.

Commands on projects or labels:

- *newproj;<name>* Create a project called <name>.
- *modproj;<name1>;<name2>* Change a project's name from <name1> to <name2>.
- *delproj;<name>* Delete the project called <name> and move all its tasks to Inbox.
- *viewproj;<name>* View tasks from a project called <name> in chronological order.
- *listproj* Show all projects in alphabetical order but with Inbox at the bottom.

- *newlab;<name>* Create a label called <name>.
- *modlab;<name1>;<name2>* Change a label's name from <name1> to <name2>.
- *dellab;<name>* Delete a label called <name>.
- *viewlab;<name>* View tasks from a label called <name> in chronological order.
- *listlab* Show all labels in alphabetical order but with Starred at the top.

Commands on tasks:
- *newtask;<name>;<time>;<desc>* Create a new task called <name> with due date <time> and description <desc> and add it to Inbox; a valid example of <time> would be "2021 1015 2359".
- *completetask;<name>* Remove a task called <name> from its project.
- *star;<name>* Add a task called <name> to Starred.
- *unstar;<name>* Remove a task called <name> from Starred.
- *rename;<name1>;<name2>* Change the name of a task from <name1> to <name2>.
- *redesc;<name>;<desc>* Change the description of a task called <name> to <desc>.
- *retime;<name>;<time>* Change the due date of a task called <name> to <time>.
- *reproj;<task name>;<proj name>* Move a task called <task name> to a different project called <proj name>.
- *addtasklab;<task name>;<lab name>* Add a task called <task name> to a label called <lab name>.
- *deltasklab;<task name>;<lab name>* Remove a task called <task name> from a label called <lab name>.

CRC Cards

Entities:

Task	
Attributes: ❏ name ❏ due day ❏ description (optional, depends on user) ❏ project ❏ labels Methods: • get name, set name • get due date, set due date • set description • get project, set project • get labels	Project Label

NOT abstract **Folder**	Project, Label
Attributes: ❏ name ❏ a collection of tasks Methods: • get name, set name • get tasks • add a task, delete a task	Task

Project	Folder
	Task

Label	Folder
	Task

TodoSystem	
Attributes (protected): ❏ a collection of all tasks ❏ a collection of all projects ❏ a collection of all labels Methods: • getters for all collections • add a task, delete a task • add a project, delete a project • add a label, delete a label	Task Project Label

Use cases:

Interface **Executable**	
Methods: • execute: take user arguments and DataAccessor, execute the command	

Implements Executable **NewTask**	
Methods: • execute: create a new task and put it into Inbox	Task TodoSystem DataAccessor

Implements Executable **Upcoming**	
Methods: • exercute: return a sorted list of all tasks in chronological order	Task TodoSystem DataAccessor

Implements Executable **CompleteTask**	
Methods: • Exercute: delete an existing task	Task TodoSystem DataAccessor

Implements Executable **Star**	
Methods: • execute: add task to the Starred label	Task Label TodoSystem DataAccessor

Implements Executable **Unstar**	
Methods: • execute: delete task from the Starred label	Task Label TodoSystem DataAccessor

Implements Executable **Rename**	
Methods: • execute: rename a task	Task TodoSystem DataAccessor

Implements Executable **Retime**	
Methods: • execute: modify the due date of a task	Task TodoSystem DataAccessor

Implements Executable **Redesc**	
Methods: • execute: modify the description of a task	Task TodoSystem DataAccessor

Implements Executable **Reproj**	
Methods: • execute: move a task to a different project	Task Project TodoSystem DataAccessor

Implements Executable **AddTaskLab**	
Methods: • execute: add a task to a label	Task Label TodoSystem DataAccessor

Implements Executable **DelTaskLab**	
Methods: • execute: delete a task from a label	Task Label TodoSystem DataAccessor

Implements Executable **NewProj**	
Methods: • execute: create a new project	Project TodoSystem DataAccessor

Implements Executable **ModProj**	
Methods: • execute: modify project name	Project TodoSystem DataAccessor

Implements Executable **DelProj**	
Methods: • execute: delete an existing project	Project TodoSystem DataAccessor

Implements Executable **ViewProj**	
Methods: • execute: return a sorted list of tasks in an existing project	Project TodoSystem DataAccessor

Implements Executable **ListProj**	
Methods: • execute: return a sorted list of existing projects	Project TodoSystem DataAccessor

Implements Executable **NewLab**	
Methods: • execute: create a new label	Label TodoSystem DataAccessor

Implements Executable **ModLab**	
Methods: • execute: modify label name	Label TodoSystem DataAccessor

Implements Executable **DelLab**	
Methods: • execute: delete an existing label	Label TodoSystem DataAccessor

Implements Executable **ViewLab**	
Methods: • execute: return a sorted list of tasks in an existing label	Label TodoSystem DataAccessor

Implements Executable **ListLab**	
Methods: • execute: return a sorted list of existing labels	Label TodoSystem DataAccessor

Controllers:

CommandExecutor	
Methods: • execute an Executable as specified by user command	Executable DataAccessor

Driver & database:

Interface **DataAccessor**	
Methods: • getSystem: return our TodoSystem object	TodoSystem

Implements DataAccessor **DataManager**	
Attributes: ❏ todoSystem Methods: • read data from local files • write data into local files • getSystem: return our TodoSystem object	TodoSystem

Driver	
Methods:: ❏ Main: ❏ initialize the system (read files) ❏ while running, ask user for commands, execute them, and print results returned ❏ exit the system (write files)	CommandExecu ter DataManager

Scenario Walk-Through

As the user starts running the program, we step inside the main loop in Driver. After initializing DataManager, CommandExecutor, and all commands, it attempts to deserialize a local “system.ser” file to populate the system with pre-existing entities. DataManager finds no such file, so we start with a new empty system. Then, inside a while loop, it prompts the user for commands. The user decides to create a new task by typing “newtask;csc207 project phase 0;2021-10-15; due very soon, hurry” into the console. The Driver lets CommandExecutor execute the command, passing in DataManager as a DataAccessor. It first checks whether the user typed a valid command name; then it calls the execute method of the corresponding Executable object (NewTask), passing in DataAccessor and user-specified parameters separated by “;”. After checking whether the arguments are valid, a new Task object gets added to Inbox. A message confirming that the task has been created successfully gets returned back to Driver and printed to the console. Then the user exits the program by typing “exit”. We break out of the while loop in the main method, and DataManager saves data using serialization.

Phase 0 Progress Report

Specification:
Apart from a general introduction to our program and a statement of scope for phase 0, our specification is mainly divided into the following two sections:

- In the first section, we introduce the three main components (or entities) of our program, task, project, and label, as well as their attributes and how they should interact with commands from the user and other entities. Here, we also define certain rules such as "a task must be in exactly one project."
- In the second section, we list all the commands that users can input, along with a description of what they do.

CRC model:

- Entities: We have 5 entity classes: Task, Folder, Project, Label, and TodoSystem. Project and Label store Task objects, and TodoSystem stores all Task, Project, and Label objects. Since the only difference between Project and Label is the number of them a task can or must belong to, we have a Folder parent class for both of them.
- Use cases: Each use case class implements the Executable interface and corresponds to a command that the user types into the console. They interact with entities via the DataAccessor interface which returns our TodoSystem.
- Controllers: There is only one controller, CommandExecuter, which receives user inputs and passes them down to corresponding Executable objects to execute.
- Database and driver: Datamanager implements the DataAccessor interface and uses serialization to translate between a local .ser file and our TodoSystem. HowTodoit is our main driver where the program is run.

Scenario walk-through:
Our scenario walk-through demonstrates a typical scenario our program would encounter: a new user creating their first-ever task and exiting the program right afterwards. A new, empty TodoSystem gets created; one task gets added to the system; and everything gets saved into system.ser before the program terminates.

Skeleton program:

- Our skeleton program is essentially our CRC model translated to Java. We classified the classes according to the Clean Architecture layers except that we placed our controller right next to all the commands.
- To manage our code structure, we created a package called constants which stores all commands and file paths and a package called helpers which stores all helper classes.
- Although we implemented all commands and are able to deal with scenarios way more complicated than our scenario walkthrough, we are leaving out the Exceptions for later stages.
- We have written one unit test for each of the three "create new entity" use cases.

Group member tasks:

- Current tasks:
    - Richard: specification, CRC model, scenario walk-through, coding, progress report, team organization and coordination.
    - Zixiu: specification, difficult parts of the program such as dealing with due dates and sorting chronologically, implementing commands.
    - Krystal: specification, making unit tests for use cases, progress report.
    - Jiayang: specification, making unit tests for use cases, progress report.
    - Jingyang: specification, CRC model, implementing commands, improving program structure.
    - Yixin: specification, CRC model, implementing commands, progress report, keeping track of what teammates are on to.
- Future tasks:
    - Everyone: researching design patterns, coding.
    - Richard: improving program structure, implementing new features.
    - Zixiu: GUI (android studio).
    - Krystal: making unit tests.
    - Jiayang: making unit tests.
    - Jingyang: improving program structure, implementing new features.
    - Yixin: implementing new features, brainstorming design ideas for phase 1.

What has worked well so far:

- All of our use cases can be run smoothly if we ignore Exceptions for now.

Questions:

- Should we implement methods such as addProj and delProj inside TodoSystem or should we put all the code inside their use case classes (NewProj and DelProj)?
    - Those methods are only used once in our program (at least for now).
    - When more features get added, we don't want to have to add methods in Entity classes on top of creating more use cases.
- The Project class and the Label class have no difference. Should we make them an attribute of Folder? Would that limit future development if more differences get introduced to these two classes?