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 the user can type 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 basically 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:
    - Richard: improving program structure, brainstorming ideas for phase 1.
    - Zixiu:
    - Krystal:
    - Jiayang:
    - Jingyang:
    - Yixin:

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?