- Team package
    - Person
        - The original plan was to develop an abstract Person class with extended classes for a Manager and a Programmer. The abstract class was made, and while working on the first of the extended classes, it was decided that a simple boolean flag, like isManager, would be used in a single class instead of having 3 separate classes. Part of this was because we didn't want to take any functionality away from the Manager, only add permissions, which meant that the majority of the functionality would need to be in the abstract class, making the Programmer class empty. We could've made the Person class concrete and made it what was used for programmers, with the Manager class still extending from it, but the differences were so minor that it was decided to use the boolean flag instead.
        - When developing an *equals* method for the Person class, it became apparent that 2 people with the same name would be flagged as equal to each other, necessitating some other differentiating factor. A static Person ID was developed that would assign a unique ID to each person on creation. This is not a perfect method, as a Person loaded from a file may possess the same ID as a Person already existing. Instead of checking for such collisions, each ID is a random long integer, substantially decreasing the possibilities of a duplication. Also, since the intention is to discard the old set of Person instances as soon as another is loaded, the low possibility of a duplication is further helped by the small amount of time of possible coexistence to ensure that any duplications do not impact the program substantially.
        - It was determined to make the Person and the Team co-dependent, though they were maintained in separate files. While the Team needs to maintain a list of all Person instances on the team, the Person instances needed to maintain a reference back to the Team for registering the hours they had completed. The decision was made early on that it would be the individual Persons submitting the hours instead of the Team on behalf of the Persons.
    - Team
        - Most of the changes to the Person class corresponded to changes in the Team class design since the two were so closely inter-related. The most notable Person-related change was in respect to the Manager implementation. The original plan was to store only a single instance of a Manager, separate from the list of Programmers. When the boolean flag was instead used, the decision was made to do occasional audits (like when adding a new Person, or promoting/demoting a Person) to ensure that no more than one Person was flagged as a manager at a time. Finally, the decision was made to do no such checking in the team

package, but to do it in the GUI instead. This decision was based on the goal of making the team package earlier to work with. If, for some reason, one wanted to make a Team with multiple managers, we didn't want the Team to stop that at a more fundamental level. Specifically, this was made so that future updates or changes in design could be more readily and easily implemented. Therefor, the Team can hold as many Persons flagged as managers as one would wish, but the GUI pane which assigns managers ensures that only one of them is flagged at a time.

    - Hours package
- Risks package
- Requirements package
    - Requirement
        - Originally, the Requirement ID values were to be automatically assigned. However, to allow for customizable and unique IDs, they were converted to be assignable and changeable at runtime.
- Project
    - One of the last decisions was to make a separate Project class that held the Team, Risks, and Requirements. In a real-world implementation, the Team would better off holding the Project, as a Team may be working on multiple Projects at once, but this is what worked best for this situation. Since the Project was made the top-level class, the Load and Save methods were transferred from the Team to the Project.
- GUI package
    - Prototype
        - Mark originally began working on a prototype to get a good feel of the organization and structure of the program. He started developing a paper prototype in Adobe Photoshop, but determined it would be easier to develop it directly with JavaFX instead, with working navigational buttons, but non-working functionality buttons. While doing so, he used some classes he had developed during his free time before this class. The prototype was intended to be discarded once the team began actual development work on the project. However, a decision was made to build on top of the prototype instead of discarding it, and Mark's personally-developed classes were retained for use in the development build.
    - PersonButton/PersonButtonScrollPane
        - After making a few panes where the Person instances of the Team are all displayed, it was determined that it would be much easier to make a PersonButton class as well as a PersonButtonScrollPane class, which inherited from the JavaFX Button and ScrollPane classes respectively. This was done purely out of convenience. The PersonButton is a simple extension of the regular Button that takes in a Person in the constructor and displays their name in the Button's label. It also holds the Person,

which is accessible using the getPerson() and setPerson(Person) methods. The PersonButtonScrollPane simply displays the PersonButtons and ensures that one and only one PersonButton exists for each Person on the Team.

- TeamPresenter
    - This is another convenience change that was made from the original design. With multiple panes that displayed the members of the Team, we wanted a way for all of them to be stored and notified when the Team was changed. The TeamPresenter interface was the answer to that. Whenever the members of the Team changed, the Team would go through its “distro” of TeamPresenters and notify each of them to the change. The presenting panes add themselves to the distro when the Team is set by that class.
- ProjectPane
    - Similar to the TeamPresenter interface, the ProjectPane was meant to provide an easy way to notify any presenters of the Project class that the project had changed. The first time the load feature of the GUI was used, nothing changed on the screen, though using the debugger we could verify that the load had executed correctly. We realized this was because the panes are event-driven, and there was no event that had notified them to update the contents they displayed. So each pane that presented or stored information about the current Project was made to implement a new ProjectPane interface, whose sole method was loadNewProject(Project). This was only done when a new Project was created or loaded from a file.
- Config
-