Participating team members: David Nuthakki, Varoon Kodithala, Lily Fischer, Sanjana Surapaneni, Rohan Gandham, and Arnav Patidar

We incorporated several of the design patterns that we learned into our Week 9 assignment in CS2340.

The first of these patterns was the Liskov Design Principle (LSP), which is defined in the SOLID framework. LSP compels programmers to ensure that all subtypes conform with the contracts defined in their supertypes such that each part of code gets the expected result. We implemented LSP by defining a Responsibilities interface where we defined three actions central to both Employees and Managers: inMeeting(), reviewingProject(), and writingCode(). Employees and Managers ended up having radically different implementations of each of these methods (for instance, employees attended meetings while managers led meetings), which made LSP that much more useful to us. Of course, in conformance with LSP, we made sure that our Employee and Manager classes incorporated the methods defined in the Responsibilities interface.

Another design pattern that was at the core of our implementation was the Single Responsibility Principle (SRP), which is also defined in the SOLID framework. We made sure that each of our classes maintained a single responsibility throughout our project. For instance, our Manager class was responsible for instantiating and maintaining an object of its type, whereas the Responsibilities interface was responsible for manipulating the current task assigned to an employee or manager. In an implementation not compliant with SRP, this might look like a Manager class that both instantiates and maintains instances of its type and assigns tasks to its children.

We also adhered to the Interface Segregation Principle (ISP), which is defined in the SOLID framework. We did so by making sure that our interface was as simple as possible, applying it to only the Member and Employee classes and adding to it three methods that we felt were core to the functionality of our system.

We also incorporated a few design principles from the GRASP framework, including the Information Expert/Expert principle. In our case, the Project class would be considered as an Information Expert, being that you can retrieve information about all of the managers/employees assigned to a project from it. Most of the information in our system is passed through the Project class, solidifying it as an Information Expert/Expert.

Our code was highly cohesive, mainly because we made sure that the functions defined in each class/interface were directly related to the functionality that we sought to achieve in them. And lastly, we incorporated the Creator design pattern from the GRASP framework in several instances, namely by defining several of them (Manager and Employee were the principal ones).