**Singleton**

The MVC software pattern makes use of three components: a model representing the data behind the system, a view that can represent that data in a more readable form for the user, and a controller that interprets and transmits the data between the model and the view. As the controller makes use of the same set of commands throughout the system, there is no reason to create multiple controllers for one system, meaning the Controller can be represented as a Singleton.

With a singleton pattern, the user (or multiple users) only needs to see the interactive components they need and can leave all the database stuff to the Controller. This also allows user restriction on actions that are locked behind roles, such as adding more movies and new tickets.

**Ticket Factory**

Our original design did not take into account different types of tickets, such as those given to students, seniors, matinee attendees, and so on. Simply using inheritance would increase the coupling of our classes, making it difficult to add additional types of tickets later on. Instead, using the Factory Method creational pattern allows us to accommodate the creation of different types of tickets while reducing the coupling between classes. In addition, using this pattern helps our design adhere to the Single Responsibility Principle, which says every class should have a single responsibility, by isolating the creation code. This pattern also adheres to the Open/Closed Principle, which says classes should be open for extension but closed for modification, by making it easy to add additional types of tickets without affecting the client code.

**Employee Factory**

In a real-world scenario, companies will have multiple job listings to fill. Oftentimes, the company does not know the sequence of who will be hired and what position they are filling. Whether it be a new manager or an attendant, the hiring system should provide the minimum amount of work for the HR department to do to simplify the transition of future employees into the company.

Our system aims to simulate this scenario by making use of the factory design pattern for our employees. The purpose of the Employee Factory is to create multiple types of employees with different roles and permissions such as Managers and Attendants. For the user who is only concerned with hiring new employees into the team, they don’t need to be concerned about knowing every subclass of the Employee. As the company grows, there may be a need to expand the job expertise into other fields such as onboarding an IT technician, in-house original film maker, and etc. Regardless of job description, they are all still employees of the company which means that these jobs can be added to the Employee Factory. With that said, the same user who only focuses on hiring new employees don’t need to be concerned about these new Employee subclasses so long as they hire the right employee for the job.