Factory Build Design Pattern

First design pattern that we utilized during our project is the factory design pattern. Its main purpose is to eliminate complexity and overuse of some cases such as tour creation logics. Instead of directly instantiating objects using new keyword, factory design encapsulates the creation logic and provides a unified interface to utilize when creating different types of tour objects. With the use of it, the client does not have to know the exact class name or how to instantiate it. The factory design pattern is useful in a way that everytime we necessarily need to add some different type of instance we can communicate with the factory instead of creating that new instance separately. This design pattern comes with advantages such as accessing a centralized control unit and easy test process during implementation. Even though it is mainly beneficial in our case there might be disadvantageous scenarios where programmers might end up with struggles during debugging, since it is highly bonded in the factory unit and rest of the subunits might be discarded implicitly.

In our case, we mainly utilized this pattern on the creation logic for “highschool tour” and “individual tour” objects that can be encapsulated using a factory method which levels the project for modular and easy to maintain perspective. If any other tour type will be included in the future all we have to do is to update (extend) the factory without modifying existing codes. Using factory design the type of tour application will be decided based on the user input during creation which will increase flexibility and readability.

Facade Design Pattern

The second design pattern utilized by our system is the facade design pattern. Its purpose is to simplify the interface for the user and hide unnecessary components of the system. It also makes the code more readable for developers, in case the code needs to be updated or handled by another developer. System maintenance and any further development is also made easier by facade design pattern. In our system, we used a facade class called TourApplicationFacade for tour applications. The user only interacts with this class to make a tour application. Thanks to this facade class unnecessary details like the database and the tour applications list is hidden from the user.