MSSE 670

Pet Grooming Reservation System

- Create a pet grooming reservation system.
- Objects include
  - Grooming services available with denoted durations.
  - Breeds with available services.
  - Pets with breed, age, and weight
  - Address of Customers
  - Customers with detailed information inclusive of all pets, address.
  - Hours of Operation to understand availability of time.
  - Reservations of when pets will arrive for service.

Facade

- Facade is a part of the Gang of Four design patterns and it is categorized under Structural design patterns.
- The purpose of the facade pattern is to allow for easier access to large systems by providing a simplified interface, thus hiding complex implementation details from the client
- It shields clients from subsystem components, thereby reducing the number of objects that clients deal with and making the subsystem easier to use. It promotes weak coupling between the subsystem and its clients. Often the components in a subsystem are strongly coupled.
- It's also self preservation from other developers.

Composite

- The composite pattern is meant to allow treating individual objects and compositions of objects, or “composites” in the same way.
- It can be viewed as a tree structure made up of types that inherit a base type, and it can represent a single part or a whole hierarchy of objects.
- We can break the pattern down into:
- component – is the base interface for all the objects in the composition. It should be either an interface or an abstract class with the common methods to manage the child composites.
- leaf – implements the default behavior of the base component. It doesn’t contain a reference to the other objects.
- composite – has leaf elements. It implements the base component methods and defines the child-related operations.
- client – has access to the composition elements by using the base component object.

Factory

- The factory method pattern loosens the coupling code by separating our *Product*‘s construction code from the code that uses this *Product*. This design makes it easy to extract the *Product* construction independently from the rest of the application. Besides, it allows the introduction of new products without breaking existing code.

DAO

- The Data Access Object (DAO) pattern is a structural pattern that allows us to isolate the application/business layer from the persistence layer (usually a relational database but could be any other persistence mechanism) using an abstract API.
- The API hides from the application all the complexity of performing CRUD operations in the underlying storage mechanism. This permits both layers to evolve separately without knowing anything about each other.

Retrospective

- Designed the database a head of time with the objects I was creating
- Looked at what i was building as smaller increments vs building all of the domain classes as I found myself bouncing around in the code for various assignments
- Reviewed use of UUID and determined a better approach of a UUID vs saying defining an a number that auto increments.