This project was created to show my knowledge of Design Patterns. This project contains practical use and application of the following Design Patterns: Strategy Pattern, Factory Pattern, and Singleton Pattern.

The Factory Method Pattern is applied in this project using five different class. I created an Interface called IDumbbellFactory. This interface was used as my Creator. I then created a DumbbellFactory class and used that as my Concrete Creator class. This is where my dumbbells are created. These creator classes contain methods to create the dumbbells. I created an abstract Dumbbell class that my LightDumbbells and HeavyDumbbells could inherit from. The factory pattern was demonstrated using those classes and was applied for two reasons. In my application there is logic determining which type of dumbbell would be created. I also let the user input the number for the weight of the dumbbell. The factory pattern is a great example of using polymorphism, dependency injection, and inversion of control. IoC is used by letting the user enter the weight of the dumbbell and the program doing the rest of the work. Dependency Injection was used in a few different methods and accessed through properties.

The Strategy Pattern is applied by using different lifting behaviors. I used compound lifting behavior and accessory lifting behavior. I created a superclass called Workout. The Workout class needs a LiftWeightBehavior. There are two different types of behaviors as stated above. The entire work out depends on which workout behavior is chosen. The Workout class contains the algorithms and methods, regardless of which workout behavior is chosen. It can use any of them and will cater to whichever one is chosen. This pattern was applicable because of how interchangeable the behaviors are. This program is easily expandable by adding more behaviors.

The Singleton Pattern was used by the Max class which inherits from the Person class. The Singleton pattern is used to ensure there is only one instance of that class. I made sure my implementation of the Singleton pattern practiced thread safety and lazy initialization. .Net makes doing this easy by having a built in Lazy<T>.

How it works:

When starting up the program the first thing that happens is the creation of the DumbbellFactory. We start creating dumbbells right away and you have the option of changing out the weight as a parameter. This determines what type of dumbbell will be created. But it will be returned as its base class, dumbbell. We then create the two different types of behaviors. Next, Max is instantiated along with a workout object. We put them all together by letting max call his LiftWeights method. That method needs the type of workout and a dumbbell.