Module Three Lesson Review

- In this lesson review I will be reviewing the learning outcomes from module three.
- I will refer to the **Abstract Factory Pattern**, **Observer Pattern**, and **Singleton Pattern** and **Facade Pattern**.

Task 0: Overview of Learning

- GitHub Repo Link
- ChatGPT Link
- No YouTube link for this module. I spoke to Kiran about my lesson review and she said that I should be fine without it.
- **Design patterns** are a set of solutions to common problems in software design. They are reusable and can be applied to different problems in different contexts.
- **Factory patterns** are a type of design pattern that are used to create objects without exposing the creation logic to the client and refer to newly created objects using a common interface.
- In the TicTacToe code this was acheived by creating a **TicTacToeFactory**. This was responsible for creating the Board, Game Logic, Algorithm, and Player objects.
- In this moduile we used the **abstract factory pattern** to create a framework for multiple games. The framework is designed to be expandable for additional games and algorithms in the future.
- This design pattern provides a **Framwork** for further expansion.
- Real world example of using the **abstract factory pattern** would be a factory that makes cars. The factory is the framework and the various cars are the products.
- **Singletons** are a type of design pattern that restricts the instantiation of a class to one object.
- They are useful for situations where you need to control access to a shared resource.
- This is acheived by the use of a private constructor and a static method to return the instance of the object.
- **Facade patterns** are a type of design pattern that provides a simple interface to a complex system.

- I haven't really come accrross this pattern in my studies. But it makes sense that it would be implented for communicating with complex subsystems.
- I have been actively analysing other peoples code on GitHub and looking to see if I can recognise any design patterns. This will take time and practice to get good at.
- I can forsee that I will use this pattern in when I have a more complex system to communicate with.
- In this course it may turn out to be useful when I'm working on my HD task.
- **Challenges**
- The challenges I faced when refactoring the code was understanding the logic of the code.
- I could conceptualise the code but I couldn't understand the logic.
- I had to go through the code line by line to understand what was happening.
- **Engagement with Peers**
- I have been actively engaging with my peers on the discussion board.
- I helped others understand the Factory design pattern.
- The specific concern was whether or not each part of the game should have its own factory. For example a **BoardFactory** and a **PlayerFactory**.

Aha Moments!

- Understanging the **Observer Patter** came once I understood the notify() method.
- I noticed that the **update()** method was being called in the **notify** method.
- The practical uses of the **Singleton Pattern** were very interesting. Examples such as the **Logger** and **Database Connection** were very useful.
- Realising that **facade pattern** is similar to the **abstract factory pattern** was a great way to understand it.
- They can loosely be used interchangeably. But the **abstract factory pattern** is more complex and can be used to create multiple objects.

Mistake in diagrams

- The abstract Pizza factory diagram in the lecture notes is incorrect. The **Factories** are making the wrong products.
- The **PepperoniFactory** is making **CheesePizza/Burger** and the **CheeseFactory** is making **PepperoniPizza/Burger**.

- If this mistake was to occurr in code the program would create the wrong products.

Task 1: Design Pattern for Multiple Games

- Abstract Factory UML Class Diagram
- Abstract Factory GitHub Repo Link
- The **UML Class Diagram** created for this module is an example of the **abstract factory pattern**.
- An **AbstractGameFactory** class is used to create **gameFactory** objects.
- The concrete **TicTacToeFactory** class is used to create the various game objects.
- Variations of the **AbstractGameFactory** can be added by creating other concrete factory classes.

Task 2: Understanding of Observer Pattern

- I modified the code to demonstrate an understanding of its logic and functions.
- Through the creation of new Observers I was able to see how the code worked.
- One limitation I addressed was to print out whenever an observer was removed.
  - An example of this could be when a user unsubscribes from a newsletter. **Operations of the Observer Pattern**
- The **Subject** object is created
- The **Observer** objects are created and added to the **Subject** object
- the **Subject** object is updated
- The **Observer** objects are notified
- An **Observer** object is removed
- The **Subject** object is updated
- The **Observer** objects are notified
- I'm not exactly sure but I see that the observer pattern could be used to see how differnt algorithms are performing in a system.
- The **Subject** object could be the system and the **Observer** objects could be the algorithms.
- The **Subject** object could be updated with the system data and the **Observer** objects could be notified.

- **Test Cases**
- Create a test case to see if the **Observer** objects were being notified.
- Remove all the **Observer** objects and see if the **Subject** object was updated.
- Update two scores at the same time and see if the **Observer** objects were notified.
- Modify the the subject so it can more parameters. For example a wide and an increase in the score. **Cricket Terms**.

Readings

- Gang of Four Design Patterns - Wikipedia **I own this book and have not really understood until now. Practical application was key.**