Module Two Lesson Review

Task 0: Provide a short overview of what you learned in the module.

GitHub

Video Link

ChatGPT Link

Software Design

- Good software design makes use of the **right** pattern for the job. It should follow the SOLID principles.
  - **SOLID**
    - **S**ingle Responsibility Principle
    - **O**pen-Closed Principle
    - **L**iskov Substitution Principle
    - **I**nterface Segregation Principle
    - **D**ependency Inversion Principle
- Poduct Development can be broken down into 5 phases:
  - Analysis -> Design -> Implementation -> Testing -> Deployment
  - This is not a linear process, it is iterative.

UML

- UML is a meta-language for describing software design and the rules governing the relationships between it's componenets.
- **Class Diagrams** - present a static view of the system.
  - **Class** - A blueprint for creating objects
    - **Attributes** - Properties of the class
    - **Methods** - Actions that can be performed by the class
  - **Relationships** - How the classes are related to each other
    - **Association** - Semantically weak relationship between two classes
    - **Aggregation** - Aggregation is a special form of association where the part can exist without the whole
    - **Composition** - Composition is a stronger form of aggregation where the part cannot exist without the whole
    - **Inheritance** - A relationship between two classes where one class inherits the properties and methods of the other
    - **Dependency** - Changes in one class may cause changes in another class
  - **Diagrams** - A visual representation of the classes and their relationships

        * **Use Case** - A diagram that shows the interactions between the system and the actors
        * **Sequence** - A diagram that shows the interactions between the classes
        * **Activity** - A diagram that shows the flow of control between the classes
        * **State** - A diagram that shows the states of the classes and the events that cause them to change

Design Principles

- **Single Responsibility Principle**
    - The single responsiblity principle tells us that a class should have only one reason to change.
    - A class should have only one responsibility.
- **Open-Closed Principle**
    - A class should be open for extension but closed for modification
    - A class should be easily extended without modifying the class itself
    - We achieve this by using abstractions
- **Liskov Substitution Principle**
    - A class should be replaceable by its subclass without affecting the functionality of the program.
- **Dependency Inversion Principle**
    - High-level modules should not depend on low-level modules. Both should depend on abstractions.
    - Abstractions should not depend on details. Details should depend on abstractions.
    - Abstract things change infrequently, concrete things change often.
    - Abstraction provides a hinge point for change.

Aha! Moments

- **Common methods should be pushed up to the parent class**
    - This gave me a better understanding of the open-closed principle
    - This also gave me a better understanding of the Liskov substitution principle
- **Inheritance is a powerful tool for code reuse**
    - Opened my eyes to design patterns
- **Inheritance can be used to create a framework for future extensions**
    - The word framework confused me in the begginning
    - I now understand that it is a set of classes that can be used to create a program
- **I am still using a hammer. Everything is a nail!**
    - I am still using inheritance for everything

    - I need to learn more about design patterns
- **Python doesn't have interfaces**
    - I was initially confused about how to implement the dependency inversion principle without interfaces
    - I now understand that I can use abstract classes. Thanks to other students for pointing this out

---

task 1: re-factor tic-tac-toe code

- GitHub TicTacToe
- UML Outline

---

task 2: check activity 1 to make sure that your use of inheritance is safe

- Example of **inheritence**
    - TicTacToeBoard inherits from Board
    - TicTacToeGame inherits from GameLogic
    - ComputerPlayer inherits from Player
    - MinimaxAlgorithm inherits from Algorithm
- The inheritance is safe because the subclasses can be used in place of the parent class without affecting the functionality of the program.
- For example the TicTacToeBoard can be used in place of the Board class without affecting the functionality of the program.
- Inheritence allows other developers to easily understand the code.
- The readability of the code is improved by using inheritance. It's easy to follow the flow of the program.
- My program is now flexible enough to support different board sizes and different algorithms.
- The code is now easier to maintain. If I want to add a new algorithm, I can simply create a new class that inherits from Algorithm.
- Composition was not a good fit for this program. I would have had to create a new class for each board size and algorithm.
- Potential downside of inheritance is that it can lead to a large class hierarchy. The TicTacToe program is small enough that this is not an issue.

Readings

- The Refactoring Guru
- Clean Code in Python
- Design Patterns