Deliverable-2

a) 6 design patterns to be implemented
b) Composite Diagram: **NO CLUE**
c) Re-design UML Class Diagram
d) Make a report

Design Pattern 1: Singleton Pattern

What is it?: The singleton design pattern ensures that the program only has one instance/object of a certain type and that instance is given global access so that it could be accessed by the rest of the program.

How can it be implemented?: Constructors within the class must be made private so that when called by an outside class, the object cannot be instantiated. A single private static instance of the class must be declared and a public getInstance() method must be defined to allow access to the class which in this case would be the database class.

How can this be used?: Since only one database instance is required in the entire project, the database class can be created with the singleton design pattern in mind with only one instance of the database and is accessible by the entire project outside of that class.

Design Pattern 2: Factory Pattern

What is it?: The factory pattern is used to create a new instance of an object, depending on the argument(s) that is passed into a function. This means that different objects will be created with different arguments.

How can it be implemented?: If-statements are used to check certain arguments. Based on the inputs, different subclasses/objects are created.

How can this be used?: When a user is on the sign up page, it will have to check if the user is creating a new account as a Student, Faculty or Non-Faculty member, or Visitor. We can have a String that sets what type of User is being created, and then create the corresponding User subclass based on that String.

Design Pattern 3: Bridge Pattern

What is it?: The bridge pattern separates an interface from its implementation, so the implementation can be altered without having to change the interface.

How can it be implemented?: An interface is created that is implemented by multiple classes. Each class will implement the interface in their own unique way.

How can this be used?: When we display the physical items, we can use a bridge pattern for displaying the different statuses of the items. For example, if the user is looking at the items that can be checked out, it will display the amount of copies available. However, if the user is looking at the items they have checked out, it should display the number of days they have left to keep it, or if it's overdue.

<u>Design Pattern 4: Composite Pattern</u>

What is it?: The composite pattern separates an object into different components that can be individually edited. This is typically used to create a hierarchy of sub-objects within an object, or break down an object into different parts.

How can it be implemented?:

How can this be used?: This can be used for displaying the newsletter, since that uses a web embed that could be considered separate from the rest of the functions associated with the newsletter.

<u>Design Pattern 5: Chain of Responsibility</u>

What is it?: The chain of responsibility pattern allows requests to be passed sequentially between handlers. Each handler performs a function and decides whether to pass on or terminate the request.

How can it be implemented?: If-statements that decide to pass on or terminate the request

How can this be used?: When logging in the user's email and password must be valid. The checks to validate the email and password can be done sequentially. Therefore, if the email is faulty then, the password doesn't need to be checked.

<u>Design Pattern 6: Observer</u>

What is it?: Defines a "one-to-many" dependency between objects so that when the object changes state, all of its dependents are updated.

How can it be implemented?:

How can this be used?: Classes like *Book* could be subjects. User objects or dedicated notification components could be observers.So that when changes occur in the books, it is automatically updated to the user objects.

Design Patterns Used:

1. Builder design pattern used in the item abstract object.

Code To-Do:

- Make generic class for cart using bridge pattern
    - Three types, Purchase Cart, SubscriptionCart, and RentCart
    - Common functions
        - Add to Cart
        - Remove from Cart
        - Checkout cart
        - getItems
        -