Week 6 Notes
============

+ Ticket Vendor Activity (from last week)

1.(a) Why is it a bad idea to let the Show class store an instance of TicketManager?

Show is an entity, and should not depend on a Use Case class like TicketManager

1.(b) Where else can we store TicketManager? How would that move change the functionality of the TicketManager class?

Instead of storing in Show, you can store it in the TicketVenderSystem and it will be responsible foe managing all tickets for all shows.

+ Clean Architecture (from last week)

2. Write the layer of clean architecture that contains each of the classes in ReadWriteEx from the Week 5 Slides and Code page:

(a) Course -- Entity
(b) DemoMain -- Controller
(c) EnrolmentSystem -- Controller
(d) Student -- Entity
(e) StudentManager -- Use Case
(f) StudentPropertiesIterator -- Gateway / Presenter

+ CRC vs. UML

3. What is a UML diagram?

A graphical way to represent how classes of a program interact with each other.

4. List three benefits of using a UML diagram to describe a program.

5. What is a CRC card? (from last week)

A Class Responsibilies Collborators card defines class names, their responsibilities, and what classes collaborate with it.

6. What are CRC cards used for? (from last week)

Planning the design of a program before starting to write it.

7. List 3 differences between CRC cards and boxes in a UML diagram.

UMLs define all the class variables and methods (and their types) while CRC cards are just general responsibilities and interactions
CRC cards are typically made before a program is written, while UMLs are usually made after
UMLs require knowledge of the written code to make

8. Name one technique for describing an object-oriented program that has already been written, without showing the code.

UML diagrams

9. Name one technique for describing an object-oriented program before you write it, so that a team can agree on the overall design before implementing individual classes.

CRC cards

+ Design Patterns

10. What is a design pattern?

A general solution for a common problem using OOP.

11. This semester, we will cover the following design patterns:

Dependency Injection, Iterator, Observer, Strategy, Factory Method, Abstract Factory, Adapter, Builder, Façade,

For each pattern we cover in this lecture, explain

(i) What problem is the design pattern trying to solve?
(ii) How do we implement the design pattern?
(iii) Implementing this design pattern works towards which SOLID principle(s)?

Iterator
i) Need to iterate through elements that may not be stored in a iterable
ii) Implement the Iterable interface to the class you want to iterate. (Implements a hasNext() and next() method)
iii) Single responsibility, offloads the responsibility of iterating through objects from the class storing them to the iterator

Observer
i) Need to maintain consistency of info about a object in another.
ii) Add a seperate observer class and have whenever a change (update) happens in the class you want to observe, you will notify all observers of that data. Uses Observable and Observer in Java
iii) Open/Closed, Single Responsibiliy

Strategy
i) Multiple classes that only differ by some strategy to handle a certain task within itself. (ex using different sorting algos)
ii) Seperate the strategies into their own interface and have the class store a certain type of that strategy.
iii) Single responsibility, Open/Closed