**Software Design Fundamentals Week 2**

* Never include System as an actor, the blue box is the system
  + Good to have each use case associated with a goal
  + What did the application allow the actors to achieve
  + Use cases are not steps to achieve the goal, they are high level functionalities (goals)
  + Use cases should have more than 2 steps
  + There are no right or wrong answers, as long as you can justify it
* Software Design
  + Class Diagrams
    - A diagram of a function

      Description automatically generated
    - - sign denotes a private attribute or method
    - + sign denotes a public attribute or method
  + Object Oriented Design
    - Encapsulation
      * Internal details are hidden from the user or calling class
      * User or other classes can only interact with the class via a well-defined interface
    - Interface
      * Anything the client object can use
      * Non-private methods
      * Non-private fields
      * The goal is to minimize the interface of the class, the less you expose the class, the less probability something goes wrong
    - Inheritance
      * In java we use inheritance to design classes that avoid redundancy and are easy to maintain. It’s a separation of concerns, where we start with a more general class, called the base class, and derive more specific classes from it (derived class)
      * When we extend the class, the derived class inherits all the non-private code
      * Protected modifier allows a base class to expose code to the derived class without making it public to everyone. Except, any class in the same package has access to them aswell
      * Java refers to base class as superclass, and u access the constructor using super()
    - Abstraction
  + Software Design creates a plan to build the software applications that multiple developers can follow, avoid duplication, and avoid wasted development effort
  + Goals
    - Make sure the software does what the client wants
    - Apply basic OO principles for flexibility
    - Strive for maintainability and reusability
* Software Delivery Phase
  + Analysis What is being built
  + Design how it will be built
* Abstract Classes
  + Have a method with no body
  + All derived classes must implement the methods
  + Cannot instantiate an abstract class
  + Use them to define classes that should only be superclasses to other classes.
* Interfaces
  + Completely abstract
  + Can have many abstract methods which derived classes must use