Inheritance Kinds:

1. Structural Inheritance
   1. Features inherited from the superclass. Provides the structure needed for implementing the subclass.
2. Implementation Inheritance
   1. Operations are implemented using the methods provided by the class.
3. Type Inheritance
4. Repeated inheritance

Limitations of inheritance:

1. Subclassing could result in DEEP hierarchies.
   1. Complexity increases with hierarchy depth.
   2. Subclassing is not always feasible if multiple inheritance is not allowed.
2. Hiding features of the superclass is difficult, or not possible.
   1. Also consider the changes within a superclass.
3. Combining inheritance with genericity may result in complications due to implementation issues with a particular language. I.E the way Java handles generics in the case of a singleton.
4. The derived class’s type may not be a true SUBTYPE of the superclass’s type.
5. Addressing limitations
   1. Inherit from abstract types rather than concrete classes
   2. Favor Composition over inheritance

Liskov’s Substitution principle:

1. Subclasses should be substitutable for their base classes.
2. This is the essence of the ‘is-a’ relationship.

Clonability:

1. Any class has four possibilities:
   1. The class supports cloning
      1. The class implements clonable interface and declares clone to throw no exceptions.
   2. The class conditionally supports cloning
      1. Class implements clonable, but cannot guarantee that the contents can be cloned
   3. Not publicly support clone, but subclasses can clone
      1. Does not implement clonable, but can override default implementation of clone to ensure that it works correctly.
      2. Would enable subclasses to invoke super.clone()
   4. Forbid clone
      1. Not implement Clonable
         1. Clone() method always throws CloneNotSupportedException

Designing Hierarchy:

1. Depend in the direction of stability -> stable dependencies principle
2. Two options when designing hierarchy.
   1. Both classes share a common ancestor
   2. One class inherits from the other
3. Three options for invoking constructors
   1. Redefine method added whenever new types of items are added
   2. Move constructor logic to abstract superclass.
   3. Develop new class that takes care of creating items. -> factory
      1. Factory is employed when we want to make a system independent of how its products are created composed, and represented.
4. Distribute the responsibilities
5. Factor responsibilities across the hierarchy

FSM vs. USE case

1. Use FSM when there is a continual sequence of events and the manner in which these events are processed depends on the state in which system is.

Concrete vs. Abstract:

Pg 318

FSM MODELING:

PROJECT 2:

Comparative evaluation of design choices: