Composition over Inheritance Principle

Composition

- composition is referred to as a HAS-A relationship between classes in Object Oriented design
  - an object contains (owns) another object as a member variable of its class

- composition implies a relationship where the child cannot exist independent of the parent
  - something is a part of another thing (wheels on an airplane)
  - rooms in a house - each house has a room or many rooms, rooms do not exist separate to a house
  - cells in a body - when the body object is destroyed, the cells get destroyed with it

- when you put two classes together like this you are using composition
  - instead of inheriting their behavior, the houses get their behavior by being composed with the right behavior object

- composition is used in many design patterns

Aggregation

- aggregation is a HAS-A relationship between objects and is closely related to composition

- aggregation implies a relationship where the child can exist independently of the parent
  - a collection of things that are not part of it
  - airplanes at an airport
  - students in a class - get rid of the class and the students still exist
  - tires on a car - the tires can be taken off of the car and installed on a different one

- aggregation and composition are almost completely identical except that composition is used when the life of the child is completely controlled by the parent
  - the distinction loses much of its importance in languages that have garbage collection
    - you do not have to concern yourself with the life of the object

Composition over Inheritance

- favoring object composition over class inheritance helps you keep each class encapsulated and focused on one task
  - your classes and class hierarchies will remain small and will be less likely to grow into unmanageable monsters

- inheritance breaks encapsulation because sub classes are dependent upon the base class behavior
  - inheritance is tightly coupled whereas composition is loosely coupled
  - when behavior of super class changes, functionality in sub class may get broken, without any change on its part

- java does not support multiple inheritance
  - composition can be a "work-around" to this

- most design patterns favor Composition over Inheritance
  - Strategy
  - Decorator
  - If design patterns use composition then that means it has been tried and tested

- composition offers better test-ability of a class than when using Inheritance
  - you can easily provide a mock implementation of the classes that you are using
  - when designing using Inheritance it is harder to test because you need to mock both the base and subclasses
  - unit testing is one of the most important things to consider during software development
    - test driven development

Software Reuse

- here is an excerpt from “Head First Design Patterns, Eric Freeman; Elisabeth Robson; Bert Bates; Kathy Sierra” concerning inheritance and reuse

Master: Grasshopper, tell me what you have learned of the Object-Oriented ways.

Student: Master, I have learned that the promise of the object-oriented way is reuse.

Master: Grasshopper, continue...

Student: Master, through inheritance all good things may be reused and so we come to drastically cut development time like we swiftly cut bamboo in the woods.

Master: Grasshopper, is more time spent on code before or after development is complete?

Student: The answer is after, Master. We always spend more time maintaining and changing software than on initial development.

Master: So Grasshopper, should effort go into reuse above maintainability and extensibility?

Student: Master, I believe that there is truth in this.

Master: I can see that you still have much to learn. I would like for you to go and meditate on inheritance further. As you’ve seen, inheritance has its problems, and there are other ways of achieving reuse.

Summary

- creating systems using composition gives you a lot of flexibility
  - lets you encapsulate a family of algorithms into their own set of classes
  - lets you change behavior at runtime as long as the object you are composing with implements the correct behavior interface

- a design based on object composition will have more objects (if fewer classes)
  - the system’s behavior will depend on their interrelationships instead of being defined in one class

- there are many more reasons to favor Composition over inheritance, which you will start discovering once you start using design patterns