

## Design Notes

Design Smells:

- 1) Code Duplication
- 2) Too many public classes, members and methods
- 3) Classes that are too large

Program to an interface:

- 1) The declared type of the variables should be a supertype, usually an abstract class or interface.
- 2) The objects assigned to those variables can be of any concrete implementation of the supertype
- 3) `Animal animal = new Dog()`
- 4) Even better → `animal = getAnimal()`

Composition over Inheritance:

- 1) Use a has-a relationship instead of an is-a relationship. An object contains another object as a member variable of its class.
- 2) Keeps each class encapsulated and focused on one task (cohesion)
- 3) Inheritance is tightly coupled whereas composition is loosely coupled. In inheritance, subclasses are dependent upon the base class behavior and this breaks encapsulation.

Delegation principles:

- 1) Example: `equals()` method in a class. The caller asks the class to do a task itself rather than having a client do it.

Liskov Substitution Principle:

- 1) Objects of a superclass can be replaceable with objects of its subclasses without breaking the application.
- 2) Requires the objects of your subclasses to behave in the same way as the objects of your superclass.

Open-Closed Principle:

- 1) Classes should be Open for Extension and Closed for Modification.

Interface Segregation Principle:

- 1) A client should not implement an interface if it does not use a method in that interface
- 2) Happens mostly when one interface contains more than one functionality and the client only needs one functionality and not the other

#### Dependency Inversion:

- 1) Entities must depend on abstractions and not on concretions
- 2) High level classes must not depend on the low level classes
- 3) Both high level classes and low level classes should depend upon abstractions
- 4) The lower-level class implementation is accessible to the higher-level class via an abstract interface
- 5) Actual implementation of lower level class can then vary
- 6) No variable should hold a reference to a concrete class. Use factory pattern instead.
- 7) No class should subclass from a concrete class. Always subclass from an abstraction.
- 8) No method should override an implemented method of any of its base classes

#### Dependency Injection:

- 1) Makes a class independent of its dependencies.
- 2) Can replace a class without changing the class that uses them.