SWEN 262

Engineering of Software Subsystems

Design Principles
# Object Oriented Principles (Review)

- **Classes**
  - A class is a blueprint that defines the state (data) and behavior (methods) that belong to some class of thing.
  - It is like a recipe; a set of instructions for building objects.

- **Objects (Identity)**
  - An object is a unique, identifiable instance of a specific class.
  - Each object has its own copy of the state and behavior defined by the class.

- **Encapsulation**
  - An object encapsulates its state and behavior, holding it together in one place.
  - It protects access to data and methods using access modifiers.

- **Inheritance**
  - One class (the child) inherits the accessible state and behavior from another class (the parent).
  - Objects of the child class include the state and behavior defined by the parent class.

- **Polymorphism**
  - An instance of a child class can be treated as though it is an instance of the parent class.
  - What appears to be a single method at runtime may be one of many implementations (overriding).
The SOLID principles were first described by Bob Martin.
The GRASP principles were first described by Craig Larman because acronyms are cool.
The Waiter

Consider the design of this system. Where is the unintended coupling?

Are there any other negative consequences?

```java
public class Waiter {
    private double payments = 0;

    public void serve(Diner diner) {
        diner.eat();

        Wallet wallet = diner.getWallet();

        payments += wallet.deduct(100.0);
    }
}
```
One possible solution is to add a new method to the diner class. Coupling is reduced. What are the trade offs?
The Law of Demeter

- The Law of Demeter addresses unintended coupling within a software system.
- Limit the range of classes that a class talks to.
  - Each class only talks to its friends; don’t talk to strangers.
  - Each class only talks to its immediate friends; don’t talk to friends of friends.
  - Chained access exposes each interface (i.e. the Wallet is exposed to the Waiter)!
- If a class needs to talk to something “far away”, do not chain method calls together.
  - Get support from your friends, e.g. getPayment()
  - Get a new friend; establish a direct relationship.
The Liskov Substitution Principle states that, if polymorphism is leveraged to substitute a child class for its parent, there should not be any unintended consequences or side effects.

- The child must not narrow the range of accepted parameters.
  - This, by the way, does not refer to the number of parameters, but the valid values of those parameters.
- The child must not broaden the range of possible outputs.

While Liskov is closely related to polymorphism, novice designers often confuse the two.

- Polymorphism is a core feature of Object Oriented Programming that you get “for free.”
- If there is a parent-child relationship between two classes, polymorphism allows the child to be substituted for the parent.

Liskov is about using polymorphism well.

- To put it simply: substituting a child for its parent should not break the system.
There are some key object-oriented "first principles" that will be stressed in SWEN 262:

- Increase cohesion where possible
- Decrease coupling where possible
- Behaviors follow data (Information Expert)
- Prefer type (interface) inheritance over class (implementation) inheritance.
  - Program to the interface, not the implementation
- Prefer composition to inheritance
  - "has-a" relationships rather than "is-a" relationships
- Use delegation to "simulate" runtime inheritance
- Law of Demeter: "Only talk to your friends."

Software design rarely starts with first principles, but the designer should be able to explain the strengths/weaknesses of a design using them.
Design Principles

There are many more object-oriented design concepts:

- **Abstraction**
  - Provide well-defined, conceptual boundaries that focus on the outside view of an object and so serves to separate an object’s essential behavior from its implementation.

- **Principle of Least Commitment**
  - The interface of an object provides its essential behavior, and nothing more.

- **Principle of Least Astonishment**
  - An abstraction captures the entire behavior of an object and offers no surprises or side effects that go beyond the scope of the abstraction.

- **Open-Closed Principle (OCP)**
  - Software entities (classes, modules, etc.) should be **open** for extension, but **closed** for modification.
  - We should design modules that never need to change.
  - To extend the behavior of a system, we add new code. We do not modify old code.

These are examples of the principles that you should mention throughout your design documentation, but certainly not an exhaustive list!