OO design principles

SRP: The Single Responsibility Principle

- Your classes should have one single responsibility and no more.

will not be able to reuse it in a different context. Having validation logic separated into a distinct class would let you reuse it in multiple places and have only a single implementation.

OCP: The Open-Closed Principle

- Create code that does not have to be modified when requirements change or when new use cases arise. "Open for extension but closed for modification"
    - Requires you to break the problem into a set of smaller problems. Each of these tasks can then vary independently without affecting the reusability of remaining components.

    - MVC frameworks. You have the ability to extend the MVC components by adding new routes, intercepting requests, returning different responses, and overriding default behaviors.

LSP: The Liskov Substitution Principle

DIP: The Dependency-Inversion Principle

- Dependency injection provides references to objects that the class depends on instead of allowing the class to gather the dependencies itself. In practice, dependency injection can be summarized as not using the "new" keyword in your classes and demanding instances of your dependencies to be provided to your class by its clients.
- Dependency injection is an important principle and a subclass of a broader principle called inversion of control. Dependency injection is limited to object creation and assembly of its dependencies. Inversion of control, on the other hand, is a more generic idea and can be applied to different problems on different levels of abstraction.
    - IOC is heavily used by several frameworks such as Spring, Rails and even Java EE containers. Instead of you being in control of creating instances of your objects and invoking methods, you become the creator of plugins or extensions to the framework. The IOC framework will look at the web request and figure out which classes should be instantiated and which components should be delegated to. This means your classes do not have to know when their instances are created, who is using them, or how their dependencies are put together.

ISP: The Interface-Segregation Principle

DRY: Don't repeat yourself

- There are a number of reasons developers repeated waste time:
    - Following an inefficient process
    - Lack of automation
    - Reinventing the wheel

- Copy/Paste programming