* **What is a design principle?**
  + A Design principle is a basic tool or technique that can be applied to designing or writing code to make that code more maintainable, flexible, or extensible.
* **OO Principle**
  + Encapsulate what varies.
  + Code to an interface rather than to an implementation.
  + Each class in your application should have only one reason to change.
  + Classes are about behavior and functionality.
  + Using proven OO design principles results in more maintainable, flexible, and extensible software.
* **The Open-Closed Principle (OCP)**
  + OCP is all about allowing change but doing it without requiring you to modify existing code.
  + Classes should be open for extension and closed for modification.
  + Close classes by not allowing anyone to touch the working code. And open classes by allowing them to be subclassed and extended.
  + OCP is about flexibility and goes beyond just inheritance. Its certainly true that inheritance is a simple example of the open-closed principle, but there is a lot more to it than just subclassing an overriding a method. Anytime you write working code, you want to do your best to make sure that code stays working and that means not letting other people change that code. But there are going to be times when that code still needs to be changed maybe, for just one or two particular situations. Rather than just diving into your code and making a bunch of changes, the OCP lets you extend your working code without changing that code.
* **The Don’t Repeat Yourself Principle (DRY)**
  + Another principle that looks pretty simple but turns out to be critical in writing code that is easy to maintain and reuse.
  + DRY is about avoiding duplicate code by abstracting out things that are common and placing those things in a single location.
  + DRY is really about one requirement in one place. Abstracting out duplicate code is a good start to using DRY, but there is more to it than just that. When you are trying to avoid duplicate code, you are really trying to make sure that you only implement each feature and requirement in your application one single time.
  + DRY is about having each piece of information and behaviour in your system in a single, sensible place.
* **The Single Responsibility Principle (SRP)**
  + It is all about responsibility, and which objects in your system do what. You want each object that you design to have just one responsibility to focus on and when something about that responsibility changes, you will know exactly where to look to make those changes in your code.
  + Every object in your system should have a single responsibility, and all the object’s services should be focused on carrying out that single responsibility.
  + The SRP is implemented correctly when each of the objects has only one reason to change.
* **The Liskov Substitution Principle (LSP)**
  + Subtypes must be substitutable for their base types.
  + The LSP is all about well-designed inheritance. When you inherit from a base class, you must be able to substitute your subclass for that base class without things going terribly wrong. Otherwise, you have used inheritance incorrectly.
  + It is hard to understand code that misuses inheritance. When you use inheritance, your subclass gets all the methods from its superclass, even if you don't want those methods. And if you have used inheritance badly, then you are going to end up with a lot of methods that you don't want, because they probably don't make sense on your subclass. To avoid this be sure your subclasses can substitute for their base types, which is just following the LSP.
* **Delegation**
  + Is when you hand over the responsibility for a particular task to another class or method. It is an alternative to inheritance.
* **Composition**
  + Allows you to use behaviour from a family of other classes and to change that behaviour at runtime. In composition, the object composed of other behaviours owns those behaviours. When the object is destroyed, so are all of its behaviours. The behaviours in a composition do not exist outside of the composition itself.
* **Aggregation**
  + Is when one class is used as part of another class, but still exist outside of that other class.
* If you favour delegation, composition, and aggregation over inheritance, your software will usually be more flexible, and easier to maintain, extend and reuse.
* Delegation, composition, and aggregation are OO techniques that allow you to reuse behaviour without violating the LSP.