* Strive for loosely coupled designs between objects that interact
* Encapsulate what varies
* Favor composition over inheritance
* Program to interfaces not implementation
* Classes should be open for extension but closed for modification
* Depend on abstractions. Do not depend on concrete classes.
* Classes should be open for extension, but closed for modification.
* Depend upon abstractions. Do not depend upon concrete classes. (Dependency Inversion Principle). Low level and high-level classes depends on abstractions.
* Principle of Least knowledge: talk only to your immediate friends.
* Don’t call us we will call you (The Hollywood Principle)
* A class should have only one reason to change(Single Responsibility Principle)

***Hollywood Principle:***

The Hollywood Principle is a technique for building frameworks or components so that lower-level components can be hooked into the computation, but without creating dependencies between the lower-level components and the higher-level layers. It gives us a technique for creating designs that allow low-level structures to interoperate while preventing other classes from becoming too dependent on them. It guides us to put the decision making in high-level modules that can decide how and when to call low-level modules. It gives us a way to prevent “dependency rot”. Following are the patterns which uses this principle,

* Template Method Pattern
* Observer Pattern
* Factory Method Pattern

***Dependency Inversion Principle:***

The Dependency Inversion Principle teaches us to avoid the use of concrete classes and instead work as much as possible with abstractions. It makes a much stronger and general statement about how to avoid dependencies in design.

***Single Responsibility Principle:***

Every responsibility of a class is an area of potential change. More than one responsibility means more than one area of change. This principle guides us to keep each class to a single responsibility.