**SOLID Five Principles of Object Oriented Design**

**Single Responsibility Principle:** A class or a function has only one reason to change means that it can perform only one job.

For example if a class function is responsible for calculating the area of the shapes, it takes the length or radius and calculate the area of the shape. Suppose if we want to convert the output area into JSON format rather than HTML and we add this functionality in AreaCaluclator function then this will violate single responsibility principle. For this purpose we can create two new functions for JSON and HTML format and we can call then in AreaCalculator output without violating Single responsibility principle.

**Open-Closed Principle:** Object and entities of a class are open for extension but closed for modification.

For Example if a class function is calculating the areas of some specific shapes like square, rectangle, triangle and circle and we want to calculate areas of some additional shapes like hexagon, pentagon, octagon and we add their area calculation formulas in AreaCalculator function then it will violate the Open-Closed Principle. For this purpose we have to create new functions for all the shapes to calculate their areas and call the function in AreaCalculator function without violating Open-Closed Principle.

**Liskov Substitution Principle:** The principle defines that objects of a superclass shall be replaceable with objects of its subclasses without breaking the application. That requires the objects of your subclasses to behave in the same way as the objects of your superclass.

**Interface Segregation Principle:** Segregation means separating the interfaces. This principle states that client should not be forced to implement the interface functions that they do not need.

For example if a Parkinglot class has functions like parkcar, unparkcar, calculatetotalcars, calculateparkingfees and dopayment. Now we have to implement a base class of Freeparking of Parkinglot. Parking lot class was composed of parking related knowledge functions and payment related functions. In base class FreeParking, payment related functions imposed are irrelevant as there is no parking fees in FreeParking class. We can segregate the FreeParking class and now this model is more extendable and flexible

**Dependency Inversion Principle:** [Shifting from traditional, data-centric N-Tier architecture to a more domain-centric N-Tier architecture](http://www.pluralsight.com/courses/n-tier-apps-part1) and potentially to the full application of [Domain-Driven Design](http://bit.ly/PS-DDD) can yield great maintainability benefits for projects. The end result is a system that is loosely coupled, modular, and easily tested