Solid Principles:

1. Single responsibility
2. Open closed
3. Liscov substitution
4. Interface segregation
5. Dependency inversion

Single Responsibiliy:

There should never be more than one reason for a class to change.

* Single functionality
* Addresses a specific concern

Eg: Controllers,Services,Entities……

Open Closed principle:

Software entities (Classes, Modules and methods…) should be open foe extension, But closed for modification.

Open for extension: extend existing behavior

Closed for modification: existing code remains unchanged.

Eg: can derive from base class & override methods. Avoid modifying base class.

Inheritance

Liskov Substitute Principle:

We should be able to substitude base class objects with child class objects & this should not alter behaviour/characteristic of a program.

the expected outcome should be need to be there even after changing base class object to child class object.

Eg: Shape base class and Rectangle and square are child classes.

Interface Segregation Principle:

Clients should not be forced to depend upon interfaces that they don’t use.

Interface pollution:

Signs:

Classes have empty method implementations.

Method implementations throw UnsupportedOperationException.

Method implementation return null or default/dummy values.

Eg: in your repository you have method called findByName which is using by one entityService

Now if you have one more entityservice which will use that interface then what if this entity doesn’t have Name. this will create empty method implementation.

Dependency Inversion Principle:

1. High level modules should not depend upon low level modules. Both should depend upon abstractions. (depend upon abstractions (interfaces) not upon concrete classes)
2. Abstractions should not depend upon details. Details should depend upon abstractions.

**Design Patterns**

1. Creational Design Patterns.

Deals with the process of creation of objects of classes.

1. Structural Design Patterns

Deals with how classes and objects are arranged or composed.

1. Behavioral Design Patterns

Describe how classes and objects interact and communicate with each other.