1. **Repository Pattern:**

It helps us to centralized data access layer.

DAL: it will deal how to open connection, how to get the data from database tables etc.

Repository pattern: it abstracts the features of DAL

Generic Repository Pattern?

we can create repository in Generic and use it through out application.

1. **Unit of Work:**

Repository pattern not completed without Unit of Work; it helps to implement / abstract transactions in repository.

It helps to begin, commit and rollback DB transactions in Repository pattern.

1. **CQRS Pattern:** command query responsibility and segregation

Separate interfaces created for CRUD, as few fields are not necessary while insert, few fields are not necessary while select, few of them are not necessary while update or delete, but we are using single interfaces for all of them and lot of fields are unused in most of the cases. So, CQRS says that we should create separate interfaces for all CRUD objects.

MediatR: It is a mediator between command and handler

1. **Singleton Pattern:**

Singleton = Static + Encapsulate + Lazy loading + double null check + Thread Safety

Encapsulate – Global objects should be encapsulated, otherwise anyone can overwrite it.

Ex: private static List<Country> countries = null;

Through DI, we update countries in constructor, but there is a chance to update with null in anywhere in code, so it should be better to encapsulated using VIRTUAL.

Lazy Loading – on demand loading data.

Ex:

if(countries == null) // Lazy loading

{

Countries = new List<Country>();

}

Thread Safety – locking while accessing the db.

Ex:

If( countries ==null)

{

Lock(countries)

{

Countries = new List<Country>();

}

}

1. **Factory Pattern:**

Factory pattern helps to centralize the object creation. In general, it’s nothing but Generic implementation, if we made a change in one place and reflected the change in entire application.

Using NEW keyword, we are tightly coupling the object creation, using factory pattern we can delegate object creation.

1. **Iterator Pattern:**

Foreach loop runs based on Iterator pattern.

1. **Dependency Injection:**

DI helps to centralize the object creation by passing objects as a parameterized construction to the class, and store them in many ways like Scoped, transient, singleton etc.

1. **IOC (Inversion of Control):**

It inverses unnecessary work of us to someone else, like delegate.

Ex: we inverse our kitchen works to maid.

Ex:

Public class HomeController:Controller

{

Private IRepository<Customer> \_iCust=null; // IOC – we are inverse to perform object creation to DI

Private HomeController(IRepostiory<Customer> cust) // DI

{

\_iCust= cust; // DI

}

}