**Ninject**

https://www.c-sharpcorner.com/UploadFile/4d9083/dependency-injection-using-ninject-in-net/

For depenedency injection using ninject perform following steps:

1. add ninject dll in project

2. Configure In NinjectWebCommon class to bind the depnedency

or inherit from ninject module

configure IMailSender and ILogging to resolve to their specified

concrete implementations

kernel.Bind<IMailSender>().To<MockMailSender>();

kernel.Bind<ILogging>().To<MockLogging>();

4. do association in the class where u want to resolve depnedency

public class MockMailSender : IMailSender

{

private readonly ILogging logging;

public MockMailSender(ILogging logging)

{

// when instantiating this class, Ninject will automatically resolve

which implementation of ILogging to use and pass it in

this.logging = logging;

}}

5. Now in program.cs create the bl object which will internally insert the dal dependency.

[Inject] // we can't use constructor injection for webforms, so use property injection

public IMailSender mailSender { get;set; }

-------------------------------url

**Circuit breaker**

Circuit-breakers make sense when calling a somewhat unreliable API.

They use a fail-fast approach when a method has failed several times in a row.

Open state: circiut breaker is open and n more calls will be made

Half open: it will retry

Closed state: The service is working fine and the call will be made

https://andrewlock.net/when-you-use-the-polly-circuit-breaker-make-sure-you-share-your-policy-instances-2/

You can create a circuit-breaker Policy in Polly using the CircuitBreakerSyntax.

**Git**

https://medium.freecodecamp.org/how-to-use-git-efficiently-54320a236369

https://medium.freecodecamp.org/what-is-git-and-how-to-use-it-c341b049ae61

https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

-----------------------------------------------------------fact

**Design Pattern**

Factory Method is a **Design Pattern** which defines an interface for creating an object,

but lets the classes that implement the interface decide which class to instantiate.

Factory Pattern lets a class postpone instantiation to sub-classes.

The factory pattern is used to replace class constructors, abstracting the process of object generation

so that the type of the object instantiated can be determined at run-time.

The Abstract Factory Pattern (AKA Factory of Factories) is a Creational pattern in which

interfaces are defined for creating families of related objects without specifying their actual implementations.

When using this pattern, you create factories which return many kinds of related objects.

Difference:

Factory Method Design Pattern

Here, we define an interface which will expose a method which will create objects for us.

Return type of that method is never a concrete type; rather, it will be some interface (or may be an abstract class).

Creates object through inheritance

Produce only one product

Implements code in the abstract creator that makes use of the concrete type that sub class produces

Abstract Factory Design Pattern

Here, we define an interface which will create families of related or dependent objects.

In simple words, interface will expose multiple methods each of which will create some object.

Again, here method return types will be generic interfaces.

All these objects will together become part of some important functionality.

Creates object through composition

Produce families of products

Concrete factories implements factory method to create product

http://www.c-sharpcorner.com/article/abstract-factory-design-pattern-in-c-sharp/

Unit of Work in the Repository Pattern

 it means that for a specific user action (say registration on a website), all the transactions like insert/update/delete and so on are done in one single transaction, rather then doing multiple database transactions. This means, one unit of work here involves insert/update/delete operations, all in one single transaction.

add a new class called UnitOfWork and this class will receive the instance of the DbContext. The same class will further generate the required repository instances, in other words repository instances for Order and Customer and pass the same DbContext to both the repositories. So changes of a single transaction are either done for both or none