Contents

[Type of Desing Pattern 2](#_Toc143540665)

[CoolShop 2](#_Toc143540666)

[Domain driven development 2](#_Toc143540667)

[Ado Connection Example 3](#_Toc143540668)

[Creational Pattern 3](#_Toc143540669)

[Simple factory 3](#_Toc143540670)

[Factory Method Pattern 3](#_Toc143540671)

[Abstract Factory Pattern 3](#_Toc143540672)

[Builder Pattern 3](#_Toc143540673)

[Prototype 3](#_Toc143540674)

[Singleton Pattern 4](#_Toc143540675)

[Structural Pattern 4](#_Toc143540676)

[Adapter Pattern 4](#_Toc143540677)

[Bridge Pattern 4](#_Toc143540678)

[Composite Pattern 4](#_Toc143540679)

[Mediator Pattern 4](#_Toc143540680)

[Strategy Pattern, Behavioral pattern 4](#_Toc143540681)

[Repository 4](#_Toc143540682)

[Lazy Loading 4](#_Toc143540683)

[Rip 4](#_Toc143540684)

[IOC, Inversion of Control 5](#_Toc143540685)

[Unit of work 5](#_Toc143540686)

[Decorator 5](#_Toc143540687)

[Iterator Pattern 5](#_Toc143540688)

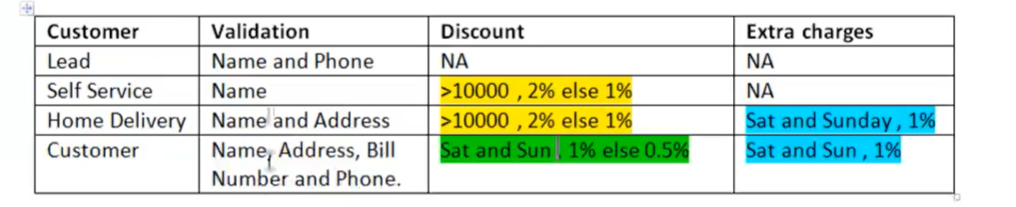
[Memento 5](#_Toc143540689)

# Type of Desing Pattern

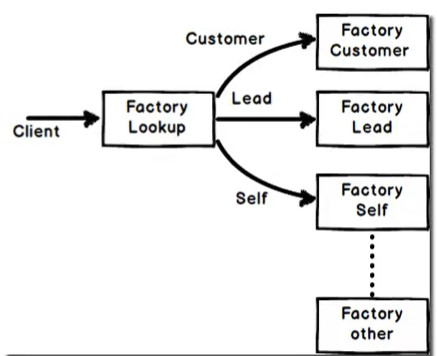
Creational, Structural and Behavioral Patterns

# CoolShop

Customer login



Object creation



# Domain driven development

Follow the vocabulary of you end user (ubiquitous language)

# Ado Connection Example

A computer screen shot of text

Description automatically generated

# Creational Pattern

## Simple factory

Create a new object in a different class.

## Factory Method Pattern

The Factory Method defines an interface for creating objects, but lets subclasses decide which classes to instantiate. The Factory method lets a class defer instantiation to subclass.

## Abstract Factory Pattern

Provide an interface for creating families of related or dependent objects without specifying their concrete class.

## Builder Pattern

Help us to separate the construction of a complex object from its representation. The same construction process can create difference representations. Common steps for creating differences objects.

**Builder** = Is responsible for defining the construction process for individual parts

**Director** = take those individual processes from the builder and defines the sequence to build the product.

**Product =** Is the final object which is produced from the builder and direction coordination.

## Prototype

Create a new copy of the object (Clone an object). By cloning, any changes to the cloned object does not affect the original object.

**Shallow Cloning**

Only the parent is cloning. By using MemberwiseClone

**Deep Cloning**

The parent and child object are cloning.

## Singleton Pattern

Only one instance of the object share for the hole application. Global Data.

No client can create an instance of this object from outside.

Static keywork create only one instance of one object.

# Structural Pattern

## Adapter Pattern

Helps to work in a unified way when we have incompatible interfaces from tree party. For example, we retrieved data in Json and xml format but we mapped this both types into a common object.

## Bridge Pattern

Helps us to decouple the abstraction from implementation.

## Composite Pattern

Treating difference objects in a similar fashion. Is just an interface implementation

# Mediator Pattern

Handle all the communication between difference components.

# Strategy Pattern, Behavioral pattern

This is a behavioral pattern witch help to select algorithms on runtime. It chooses algorithms dynamically.

# Repository

It creates a commons interface, a common base point for creating a differences type of concrete data access methods.

# Lazy Loading

Load static information just ones.

# Rip

Replacing If condition with polymorphism

# IOC, Inversion of Control

Solid Principle, take this unnecessary logic for this class and put it in another place.

# Unit of work

Solve the problems of repository pattern, create a single transaction for difference repositories

# Decorator

Adding behavior by using separate class

# Iterator Pattern

When just need to iterate over a list, do not allow add, remove end other method, just iterate. **Enumerable or Enumerator** encapsulate ADD, REMOVE methods.

# Memento

To reverse back to any old estate of one object