Sql server, .net and c# video tutorial

Free C#, .Net and Sql server video tutorial for beginners and intermediate programmers.

Facade Design Pattern

Suggested Videos
Part 18 - Bridge Design Pattern - Text - Slides
Part 19 - Composite Design Pattern - Text - Slides
Part 20 - Decorator Design Pattern - Text - Slides

In this video we will discuss

- What is Facade Design Pattern?
- Implementation Guidelines of Facade design pattern
- And will take a look at simple example to implement this pattern

Facade Design Pattern : As per the GOF definition, Facade Pattern states that we need to "Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use." This pattern Falls under the category of Structural Design Pattern and is also known as Wrapper.

Implementation Guidelines : We need to use Facade Design Pattern when

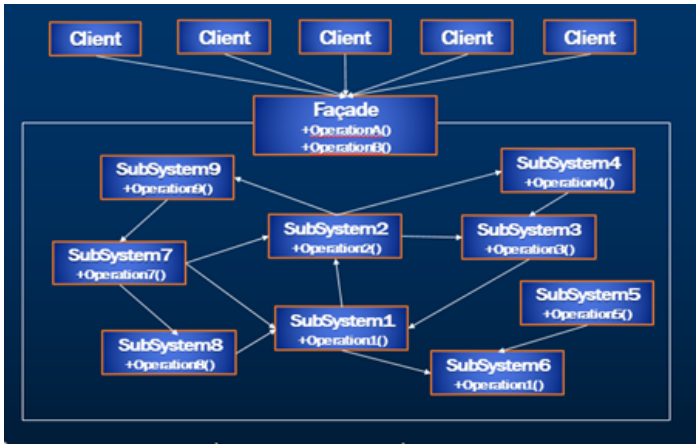
- We want to provide a simple interface to a complex subsystem. Subsystems often get more complex as they evolve.
- There are many dependencies between clients and the implementation classes of an abstraction.
- We want to layer the subsystems. Use a facade to define an entry point to each subsystem level.

Representation Diagram

Complete Tutorials
Angular tutorial for beginners
Angular 5 Tutorial for beginners

Important Videos
The Gift of Education
Web application for your business
How to become .NET developer
Resources available to help you

Dot Net Video Tutorials
ASP.NET Core Tutorial
Angular 6 Tutorial
Angular CRUD Tutorial
Angular CLI Tutorial



Facade

- Knows which subsystem classes are responsible for a request.
- And it delegates client requests to appropriate subsystem objects.

Subsystem classes

- Implement their subsystem functionality to handle work assigned by the Facade object.
- These subsystems have no knowledge of the facade; that is, they keep no references to it.

Facade design pattern implementation

Step 1 : Add Subsystem interfaces IAddress, ICart, ITax, IWallet and IAddress

```
using ShoppingCart.Implementation;
using ShoppingCart.Models;
using System;
using System.Collections.Generic;
using System.Text;

namespace ShoppingCart.Interfaces
{
   public interface IAddress
   {
      Address GetAddressDetails(int userID);
   }
}

using ShoppingCart.Models;
using System;
using System.Collections.Generic;
using System.Text;

namespace ShoppingCart.Interfaces
{
   public interface ICart
   {
      Product GetItemDetails(int itemID);
      bool CheckItemAvailability(Product product);
      bool LockItemInStock(int itemID, int quantity);
      int AddItemToCart(int itemID, int quantity);
      double GetCartPrice(int cartID);
   }
}

using System;
using System.Collections.Generic;
using System.Text;

namespace ShoppingCart.Interfaces
{
   public interface IOrder
   {
      int PlaceOrderDetails(int cartID, int shippingAddressID);
   }
}
```

Angular 2 Tutorial
Design Patterns
SOLID Principles
ASP.NET Web API
Bootstrap
AngularJS Tutorial
jQuery Tutorial
JavaScript with ASP.NET Tutorial
JavaScript Tutorial
Charts Tutorial
LINQ
LINQ to SQL
LINQ to XML
Entity Framework
WCF
ASP.NET Web Services
Dot Net Basics
C#
SQL Server
ADO.NET
ASP.NET
GridView
ASP.NET MVC
Visual Studio Tips and Tricks
Dot Net Interview Questions

Slides

Entity Framework
WCF
ASP.NET Web Services
Dot Net Basics
C#
SQL Server
ADO.NET
ASP.NET
GridView

```
using ShoppingCart.Models;
using System;
using System.Collections.Generic;
using System.Text;

namespace ShoppingCart.Interfaces
{
    public interface ITax
    {
        double GetTaxByState(string state);
        void ApplyTax(int cartID, double taxPercent);
    }
}

using System;
using System.Collections.Generic;
using System.Text;

namespace ShoppingCart.Interfaces
{
    public interface IWallet
    {
        double GetUserBalance(int userID);
    }
}
```

Step 2 : Add the below models used in the subsystems

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ShoppingCart.Models
{
    public class Address
    {
        public int AddressID { get; set; }
        public string AddressDetails { get; set; }
        public string PinCode { get; set; }
        public string Phone { get; set; }
        public string Country { get; set; }
        public string State { get; set; }
        public string City { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Text;

namespace ShoppingCart.Models
{
    public class Cart
    {
        public int CartID { get; set; }
        public int UserID { get; set; }
        public IEnumerable<CartItem> ShoppingCart { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Text;

namespace ShoppingCart.Models
{
    public class CartItem
    {
        public int ProductID { get; set; }
        public int Quantity { get; set; }
        public double TaxPercentage { get; set; }
        public double Cost { get; set; }
        public double Price { get; set; }
    }
}

using System;
```

```
using System.Collections.Generic;
using System.Text;

namespace ShoppingCart.Models
{
   public class Product
   {
      public int ProductID { get; set; }
      public string Name { get; set; }
      public string Description { get; set; }
      public int Quantity { get; set; }
      public double Cost { get; set; }
      public int LockedQty { get; set; }
   }
}
```

Step 3 : Implement the Subsystem interfaces as shown below Address, Order, Tax, Wallet, ShoppingCartDetails and Address

```
using ShoppingCart.Interfaces;
using System;
using System.Collections.Generic;
using System.Text;

namespace ShoppingCart.Implementation
{
   public class AddressDetails : IAddress
   {
      public  Models.Address  GetAddressDetails(int userID)
      {
         Console.WriteLine("\t SubSystem Address : GetAddressDetails");
         return new Models.Address();
      }

   }
}
```

```
using ShoppingCart.Interfaces;
using System;
using System.Collections.Generic;
using System.Text;

namespace ShoppingCart.Implementation
{
   public class Order : IOrder
   {
      public int PlaceOrderDetails(int cartID, int shippingAddressID)
      {
         Console.WriteLine("\t SubSystem Order : PlaceOrderDetails");
         return 10;
      }
   }
}
```

```
using ShoppingCart.Interfaces;
using System;
using System.Collections.Generic;
using System.Text;
using ShoppingCart.Models;

namespace ShoppingCart.Implementation
{
   public class ShoppingCartDetails : ICart
   {
      public ShoppingCartDetails()
      {
      }
      public int AddItemToCart(int itemID, int Quantity)
      {
         Console.WriteLine("\t SubSystem Cart : AddItemToCart");
         return 15;
      }
      public bool CheckItemAvailability(Product product)
      {
         Console.WriteLine("\t SubSystem Cart : CheckItemAvailability");
         return true;
      }
```

```
        public double GetCartPrice(int cartID)
        {
            Console.WriteLine("\t SubSystem Cart : GetCartPrice");
            return 15;
        }
        public Product GetItemDetails(int itemID)
        {
            Console.WriteLine("\t SubSystem Cart : GetItemDetails");
            return new Product();
        }
        public bool LockItemInStock(int itemID, int quantity)
        {
            Console.WriteLine("\t SubSystem Cart : LockItemInStock");
            return true;
        }
    }
}

using ShoppingCart.Interfaces;
using ShoppingCart.Models;
using System;
using System.Collections.Generic;
using System.Text;

namespace ShoppingCart.Implementation
{
    public class Tax : ITax
    {
        public void ApplyTax(int cartID, double taxPercent)
        {
            Console.WriteLine("\t SubSystem Tax : ApplyTax");
        }
        public double GetTaxByState(string state)
        {
            Console.WriteLine("\t SubSystem Tax : GetTaxByState");
            return 10;
        }
    }
}

using ShoppingCart.Interfaces;
using System;
using System.Collections.Generic;
using System.Text;

namespace ShoppingCart.Implementation
{
    public class Wallet : IWallet
    {
        public double GetUserBalance(int userID)
        {
            Console.WriteLine("\t SubSystem Wallet : GetUserBalance");
            return 100;
        }
    }
}
```

Step 4 : Add Façade layer with interface and implementation as shown below

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ShoppingFacade
{
    public interface IUserOrder
    {
        int AddToCart(int itemId, int qty);
        int PlaceOrder(int cartID, int userID);
    }

}

using ShoppingCart.Implementation;
using ShoppingCart.Interfaces;
using ShoppingCart.Models;
using System;
using System.Collections.Generic;
```

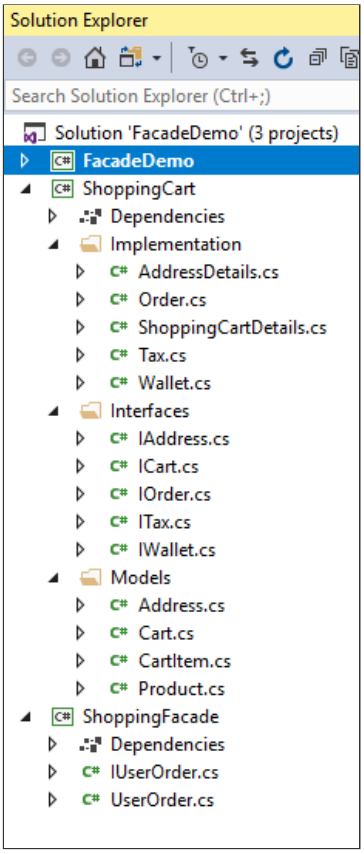
```
using System.Text;

namespace ShoppingFacade
{
   public class UserOrder : IUserOrder
   {
      public int AddToCart(int itemId, int qty )
      {
         Console.WriteLine("Start AddToCart");
         ICart userCart = new ShoppingCartDetails();
         int cartID = 0;
         //Step 1 : GetItem
         Product product = userCart.GetItemDetails(itemId);
         //Step 2 : Check Availability
         if (userCart.CheckItemAvailability(product))
         {
            //Step 3 : Lock Item in the Stock
            userCart.LockItemInStock(itemId, qty);
            //Step 4 : Add Item to the cart
            cartID = userCart.AddItemToCart(itemId, qty);
         }
         Console.WriteLine("End AddToCart");
         return cartID;
      }

      public int PlaceOrder(int cartID, int userID)
      {
         Console.WriteLine("Start PlaceOrderDetails");
         int orderID = -1;
         IWallet wallet = new Wallet();
         ITax tax = new Tax();
         ICart userCart = new ShoppingCartDetails();
         IAddress address = new AddressDetails();
         IOrder order = new Order();
         //Step 1 : Get Tax percentage by State
         double stateTax = tax.GetTaxByState("ABC");
         //Step 2 : Apply Tax on the Cart Items
         tax.ApplyTax(cartID, stateTax);
         //Step 3 : Get user Wallet balance
         double userWalletBalance = wallet.GetUserBalance(userID);
         //Step 4 : Get the cart items price
         double cartPrice = userCart.GetCartPrice(cartID);
         //Step 5 : Compare the balance and price
         if (userWalletBalance > cartPrice)
         {
            //Step 6 : Get user Address and set to cart
            Address userAddress = address.GetAddressDetails(userID);
            //Step 7 : Place the order
            orderID = order.PlaceOrderDetails(cartID, userAddress.AddressID);

         }
         Console.WriteLine("End PlaceOrderDetails");
         return orderID;
      }
   }
}
```

Step 5 : Use the console program and invoke the Façade methods with Console being the client for us

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ShoppingFacade;
namespace FacadeDemo
{
   class Program
   {
      static void Main(string[] args)
      {
         IUserOrder userOrder = new UserOrder();
         Console.WriteLine("Facade : Start");
         Console.WriteLine("************************************");
         int cartID = userOrder.AddToCart(10, 1);
         int userID = 1234;
         Console.WriteLine("************************************");
```

```
            int orderID = userOrder.PlaceOrder(cartID, userID);
            Console.WriteLine("************************************");
            Console.WriteLine("Facade : End CartID = {0}, OrderID = {1}",
               cartID, orderID);
            Console.ReadLine();
        }
    }
}
```

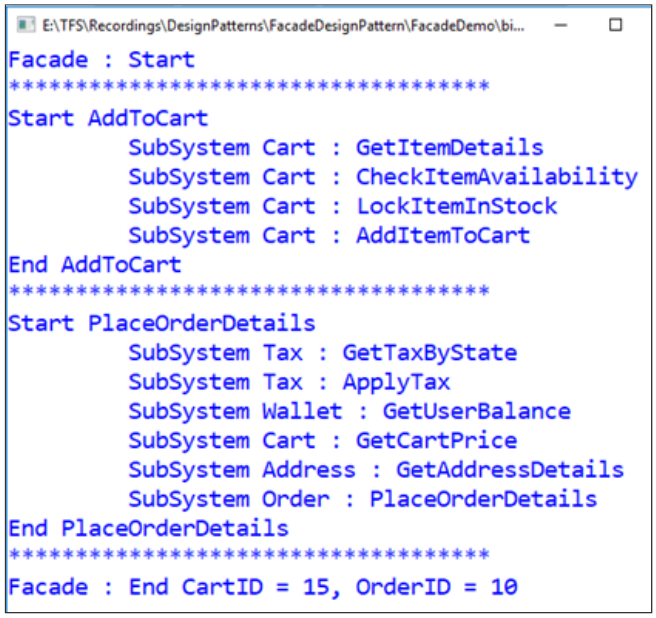
Step 6 : Ensure to maintain the below structure for the code to work



Step 7 : Key notes

- Please note that we are just representing all these methods at high level to show the complexity of the subsystems.
- We have not implemented these methods in detail except that we are printing the details of the methods invoked in each of these subsystems using console.writeline.
- As you all know, Our idea is to just understand the facade implementation and not to focus on the real implementations of these sub systems
- Notice that in the output client invokes the facade methods of add to cart and place order details which Internally calls many subsystem methods to achieve this functionality

Step 8 : Run the application and notice the below output.



3 comments:

jitendra kumar mahapatro April 2, 2018 at 6:53 AM

Nice Explaned. Please upload Dependency Injection and IOC.

Reply

Amit Bhardwaj April 14, 2018 at 4:08 AM

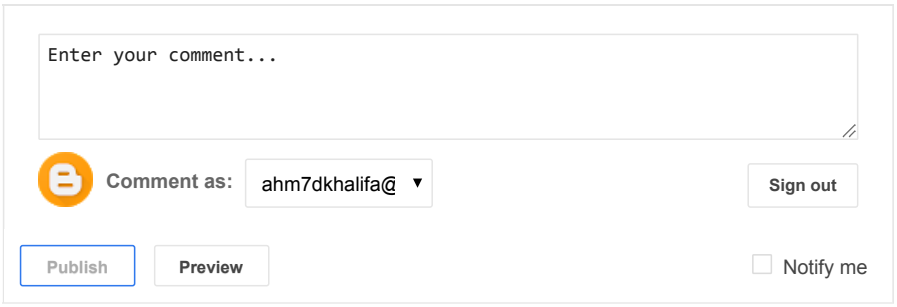
Very nice Explanation and request you please provide short and small example which will help us to understand quickly this is also good example.

Reply

Bilal qureshi June 11, 2018 at 5:44 AM

Dependency Injection and IOC too

Reply



If you like this website, please share with your friends on facebook and Google+ and recommend us on google using the g+1 button on the top right hand corner.

Links to this post

Create a Link

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)