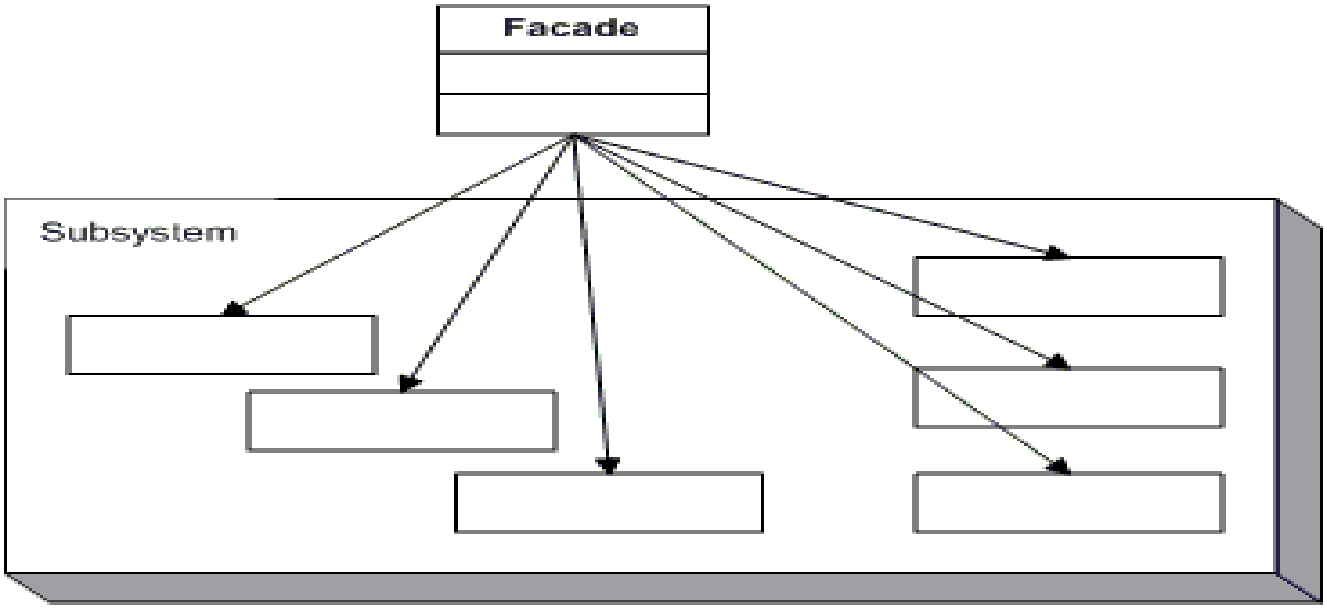
# Design Pattern : Façade

**Team 10 –** *Mitta,Sushma; Yarlagadda,Dig Vijay Kumar;Shekhar Pooja*

**Intent**

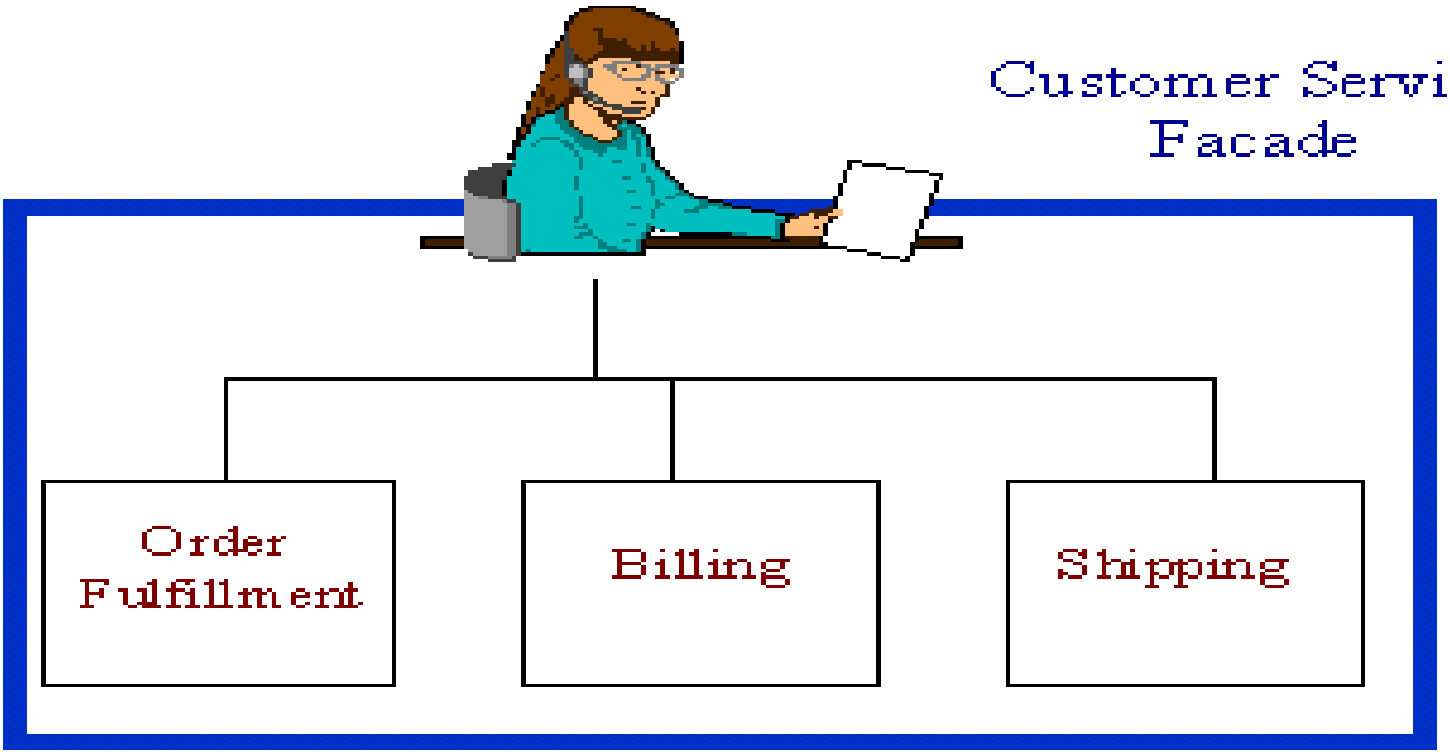
- Provides a unified interface to set of all interfaces in a system. Facade defines a higher level interface that makes the system easy to use.
- Wraps the complication of a system and provides easy to use interface.

**Structure**




**Real World Example**

Operating System is an excellent example of façade pattern which provides easy to use interface encapsulating all complexities. User connects to the façade and façade establishes connection to other sub-systems and gets task done.




The person sitting at the reception desk serves as a façade in the office. She handles all the critical sub tasks which we users want to get initiated.

**Check List**

- Identify a simpler unified interface to the sub-system
- Design a façade/wrapper class that delegates tasks to appropriate methods
- The client uses interacts with the system only through the façade

**Rules of Thumb**

- Façade defines new interface whereas Adapter uses one of the old interfaces
- Whereas Flyweight shows how to make many small objects, façade shows how to make a single object
- Abstract Factory can be used as substitute of Façade
- Adapter and facade both are wrapper design patterns but their intent are different- façade produces a simpler new interface whereas Adapter rebuilds an existing interface.

## Use Case –

Let’s take Online Order Processing The Client places an order without knowing about the internal complexities and sub classes of the system. First inventory is checked if the ordered product is available in the inventory ,if available then amount is deducted at Payment window and order gets successfully placed.

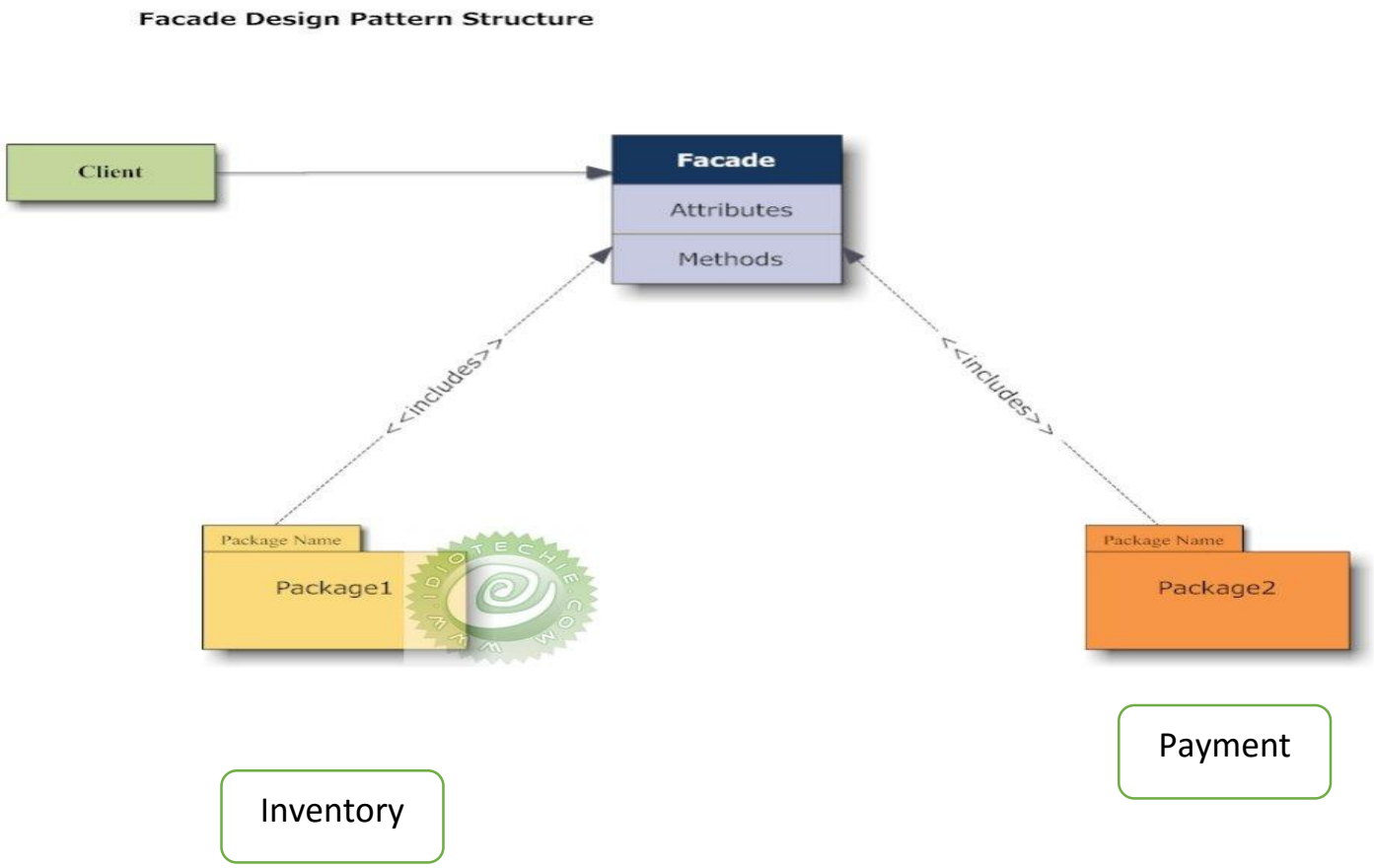
## Classes

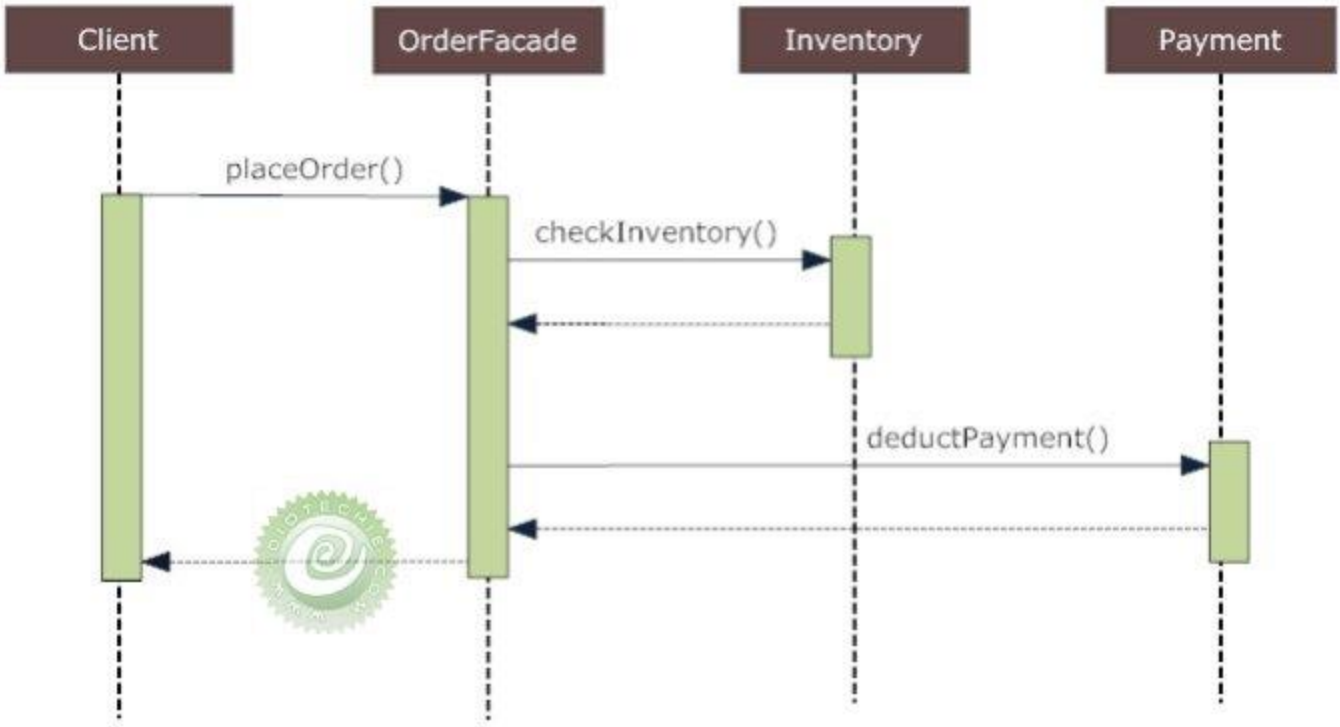
Client.java

OrderFaçade.java

Inventory.java

Payment.java






Sequence Diagram

## Client.java

Client.java | OrderFacade.java | Payment.java | Inventory.java

```
1
2  public class Client {
3
4      public static void main(String args[])
5      {
6
7      OrderFacade orderFacade=new OrderFacade();
8      orderFacade.placeOrder("N12345");
9      System.out.println("Order processing item!");
10
11     }
12
13 }
14
```

## OrderFacade.java

Client.java | OrderFacade.java | Payment.java | Inventory.java

```
1
2  public class OrderFacade {
3
4      private Payment payment=new Payment();
5      private Inventory inventory=new Inventory();
6      public void placeOrder(String orderId)
7      {
8          String step1=inventory.checkInventory(orderId);
9          String step2=payment.getPayemnt(orderId);
10         System.out.println("Following steps completed"+ step1 +"&" +step2);
11
12     }
13
14 }
15
```

## Inventory.java

Client.java | OrderFacade.java | Payment.java | Inventory.java

FacadeesignPattern/src/Client.java

```
1
2  public class Inventory {
3      public String checkInventory(String OrderId)
4      {
5          return "Inventory is available";
6      }
7
8  }
9
```

## Payment.java

Client.java | OrderFacade.java | Payment.java | Inventory.java

```
1
2 public class Payment {
3     public String getPayemnt(String paymentamt )
4     {
5         return "Payment deducted from account";
6     }
7
8 }
9
```

## References

https://www.javacodegeeks.com/2012/11/facade-design-pattern-design-standpoint.html