Design Patterns

Facade Pattern*

ebru@hacettepe.edu.tr
ebruakcapinarsezer@gmail.com
http://yunus.hacettepe.edu.tr/~ebru/
@ebru176

Kasım 2017

*modified from http://*sce2.umkc.edu/csee/leeyu/class/CS590L.../Pattern/Facade.pp*

Facade Design Pattern

Provide a unified interface to a set of interfaces in a subsystem. Defines a high level interface that makes the subsystem easier to use.

Facade

- ✓**Problem:**

  Need for a simplified interface to the overall functionality of a complex system.

- ✓**Motivation:**

  Reduce Complexity: Introduce a façade object that provides a single, simplified interface to a more general facilities of a subsystem.

Facade – Why and When?

✓Applicability

- Use the facade pattern when you want to provide a simple interface to a complex system.
- You want to layer your subsystem

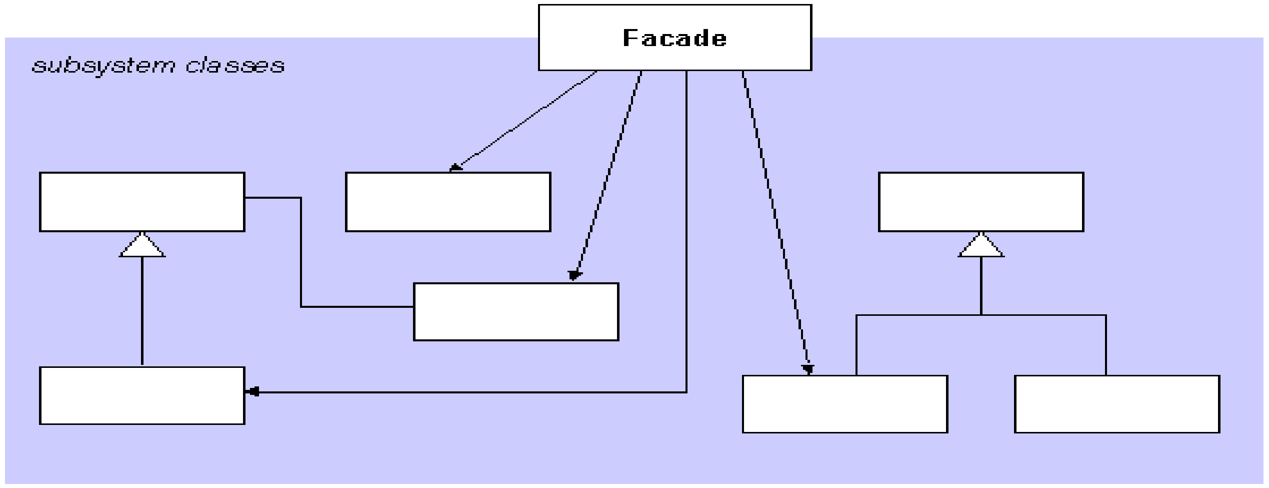
✓Consequences

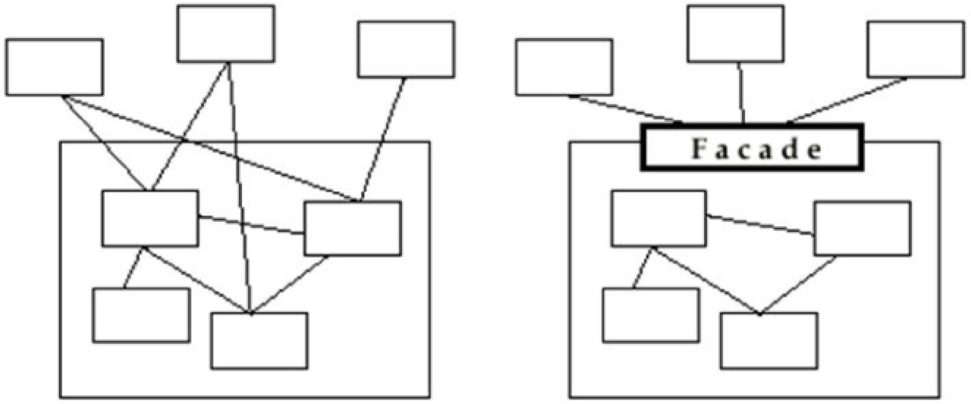
- Shielding clients from subsystem promotes weak coupling b/w subsystem and the clients.
- Simplifies porting systems to other platforms.

✓Liabilities

Does not prevent clients from accessing the underlying classes.

Facade - Structure

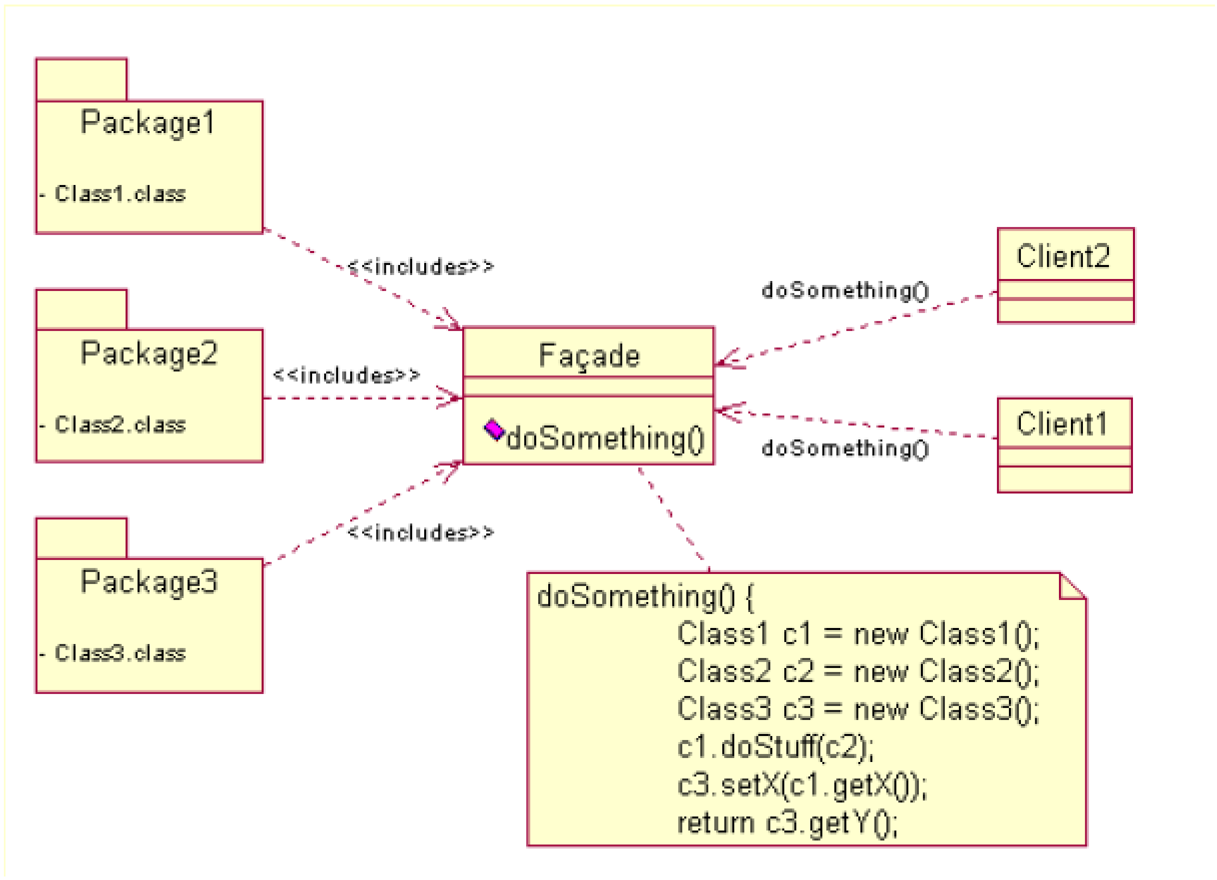




Facade - Participants

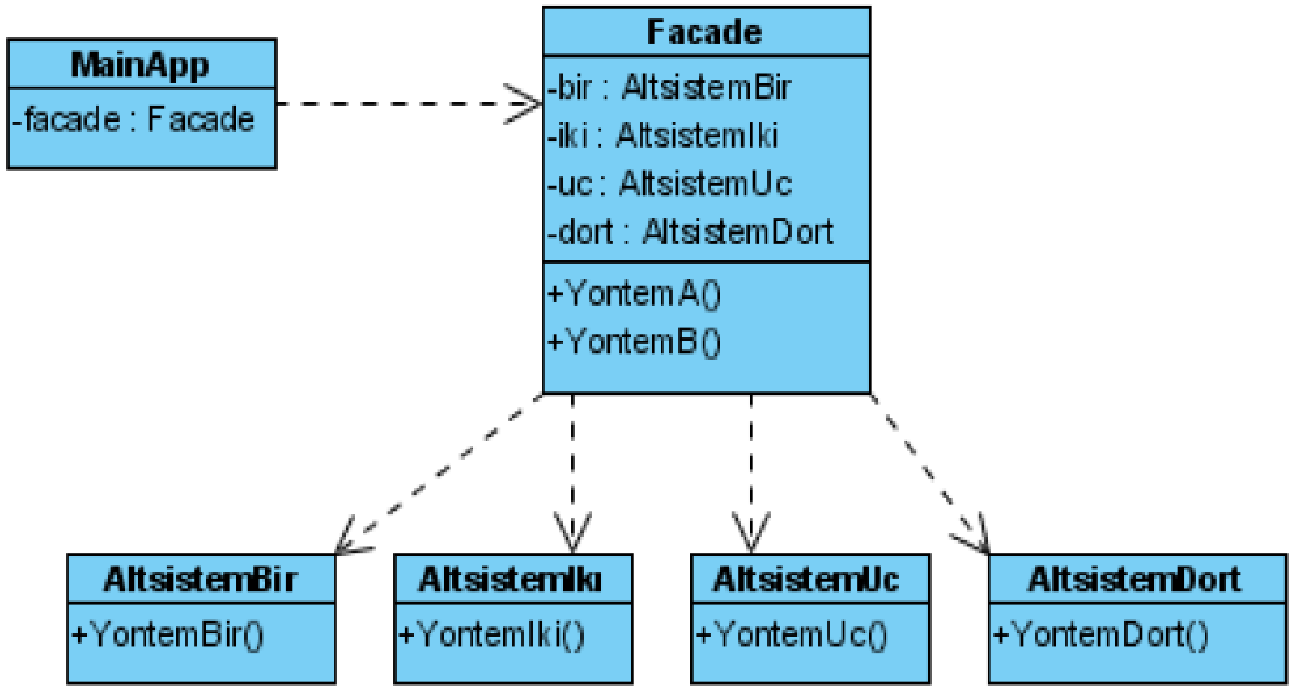
✓Façade

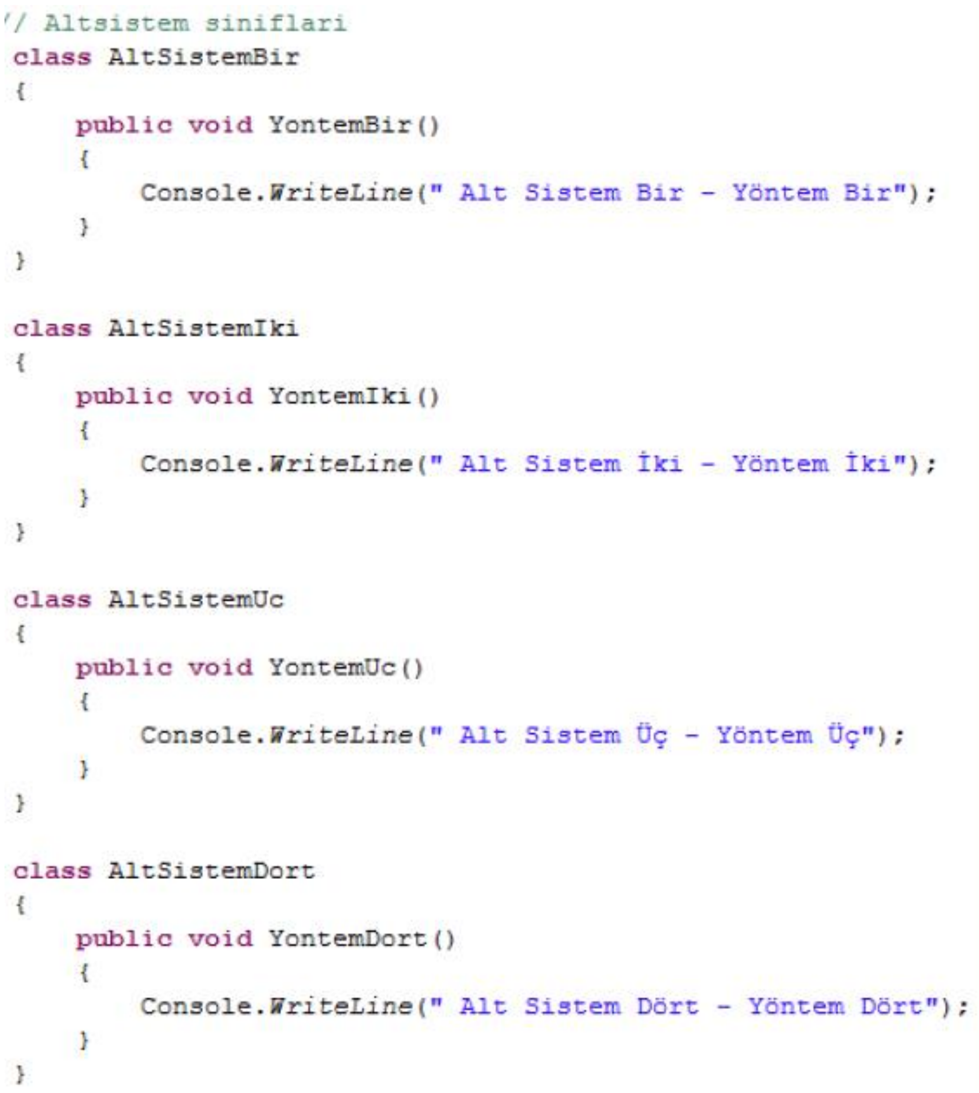
- Implements subsystem functionality
- Knows which subsystem classes are responsible for a request.
- Delegates client requests to appropriate subsystem objects
- Handles work assigned by the façade object
- Has know knowledge of the façade



```
doSomething() {
                Class1 c1 = new Class1();
                Class2 c2 = new Class2();
                Class3 c3 = new Class3();
                c1.doStuff(c2);
                c3.setX(c1.getX());
                return c3.getY();
```

Example


MainApp
-facade : Facade
Facade
-bir : AltsistemBir
-iki : AltsistemIki
-uc : AltsistemUc
-dort : AltsistemDort
+YontemA()
+YontemB()
AltsistemBir
+YontemBir()
AltsistemIki
+YontemIki()
AltsistemUc
+YontemUc()
AltsistemDort
+YontemDort()

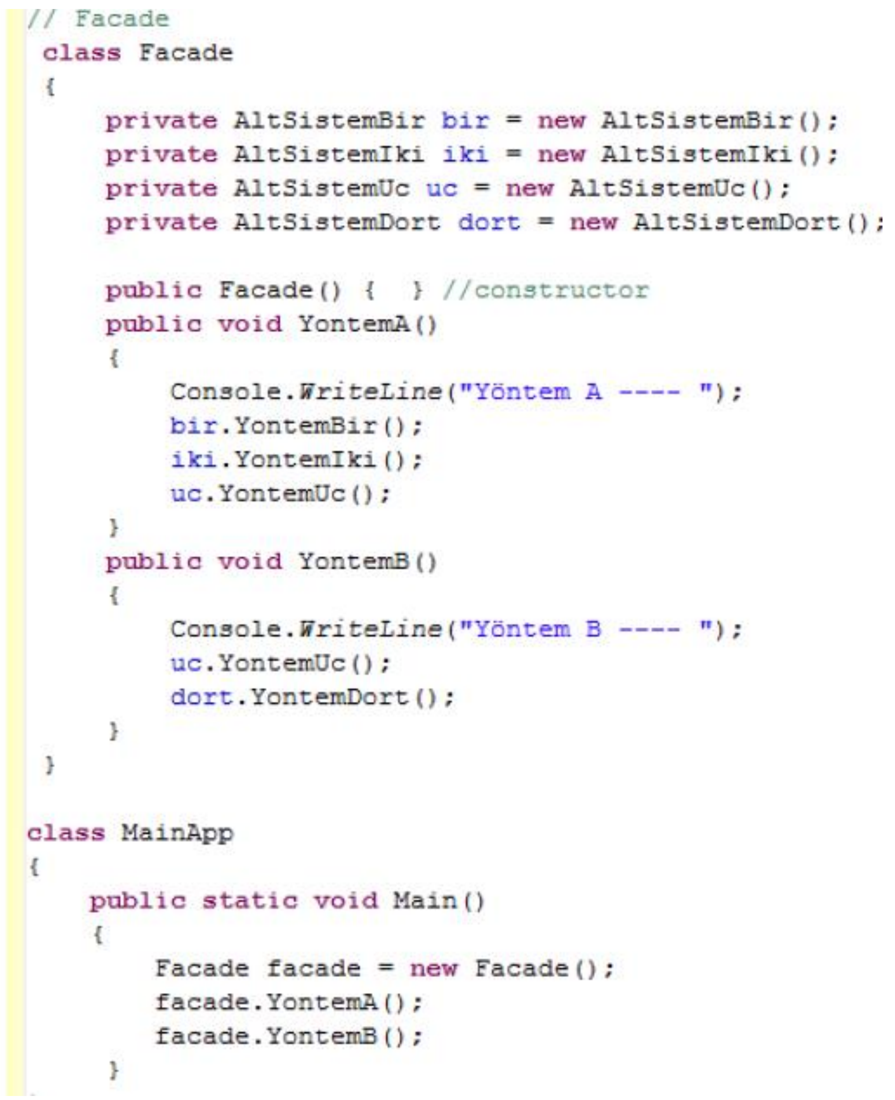


```
// Altsistem siniflari
class AltSistemBir
{
    public void YontemBir()
    {
        Console.WriteLine(" Alt Sistem Bir - Yöntem Bir");
    }
}

class AltSistemIki
{
    public void YontemIki()
    {
        Console.WriteLine(" Alt Sistem İki - Yöntem İki");
    }
}

class AltSistemUc
{
    public void YontemUc()
    {
        Console.WriteLine(" Alt Sistem Üç - Yöntem Üç");
    }
}

class AltSistemDort
{
    public void YontemDort()
    {
        Console.WriteLine(" Alt Sistem Dört - Yöntem Dört");
    }
}
```



```
// Facade
 class Facade
 {
     private AltSistemBir bir = new AltSistemBir();
     private AltSistemIki iki = new AltSistemIki();
     private AltSistemUc uc = new AltSistemUc();
     private AltSistemDort dort = new AltSistemDort();

     public Facade() {  } //constructor
     public void YontemA()
     {
         Console.WriteLine("Yöntem A ---- ");
         bir.YontemBir();
         iki.YontemIki();
         uc.YontemUc();
     }
     public void YontemB()
     {
         Console.WriteLine("Yöntem B ---- ");
         uc.YontemUc();
         dort.YontemDort();
     }
 }

class MainApp
{
    public static void Main()
    {
        Facade facade = new Facade();
        facade.YontemA();
        facade.YontemB();
    }
```

Related Patterns

- **Abstract Factory** can be used with Façade to provide an interface for creating subsystem objects. It can also be used as an alternative to hide platform specific classes.
- **Mediator** is almost similar to façade since it abstracts functionality of existing classes.

  But there is a difference b/w Mediator and Façade.

  Façade merely abstracts the interface to subsystem objects to make them easier to use. It does not define new functionality and subsystem classes know about it.

  Mediator's purpose is to abstract arbitrary communication between colleague objects. It provides a centralized functionality that does not belong to any of the objects , but these objects are aware of the mediator and communicate with it.

  Façade objects are often singletons.