NESNE YÖNELİMLİ PROGRAMLAMA 2(Object Oriented Programming 2/OOP)

Öğr. Gör. Celil ÖZTÜRK
Marmara Üniversitesi
Teknik Bilimler Meslek Yüksekokulu

İçerik

- Bridge
- Composite
- Decorator
- Flyweight
- Proxy

Tasarım Kalıpları

Structural Patterns(Yapısal Kalıplar)

- Adapter Pattern
- Bridge Pattern
- Composite Pattern
- Decorator Pattern
- Facade Pattern
- Flyweight Pattern
- Proxy Pattern

BRIDGE(KÖPRÜ) TASARIM KALIBI

- Bridge tasarım kalıbı temelde implementasyonları abstract yapılardan(soyutlamalardan) ayırabilmek için kullanılır.
- Amaç, soyutlama yöntemi ile mimariler arasında esnek bir yapı sağlamaktır.
  - Soyutlanan nesneler ile işi gerçekleştiren somut nesneler arasında köprü kurar.
  - Soyut sınıfları ve işi yapacak sınıfları birbirinden ayırdığı için iki sınıf tipinde yapılacak bir değişiklik bir birini etkilemez.

BRIDGE(KÖPRÜ) TASARIM KALIBI

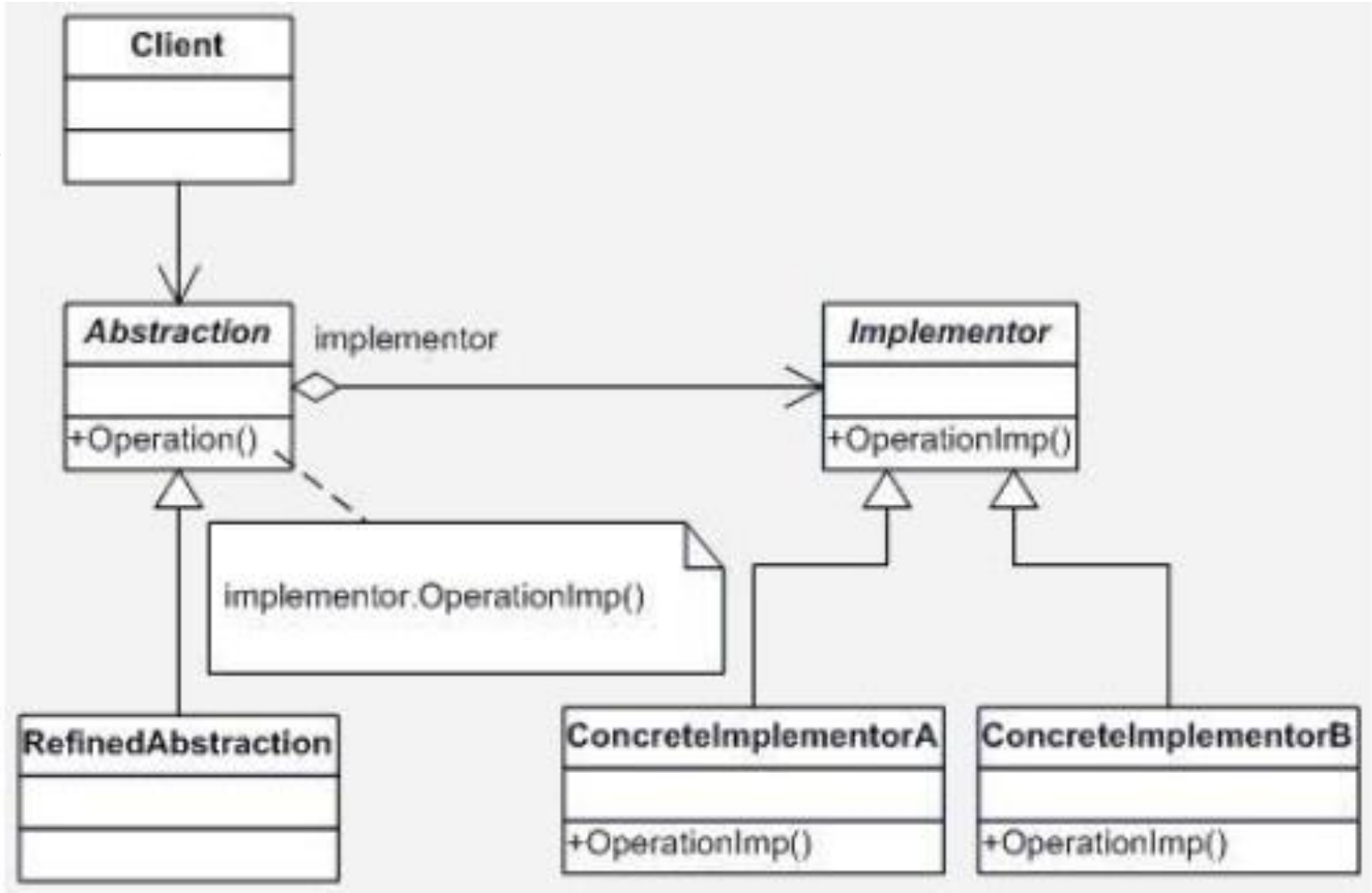
Köprü kalıbı(Bridge Pattern):
soyutlamayı (abstraction) uygulamadan (implementation) ayırarak ikisinin birbirinden bağımsız çalışmasını sağlar.

- Köprü tasarım kalıbı, bir modelleme yapılırken oluşan soyut oluşumlar ve bu oluşumlara ait uygulamaları birbirinden ayırır, bu sayede yazılımcı sınıf hiyerarşilerini daha esnek bir hale getirebilir.
- Sınıf hiyerarşilerinin daha esnek bir hale getirilmesi modellemede bulunan bir üst sınıfın içinde barındırdığı soyut oluşumların bir arayüz sınıfına taşınmasına olanak sağlar.

https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/k%C3%B6pr%C3%BC(bridge)

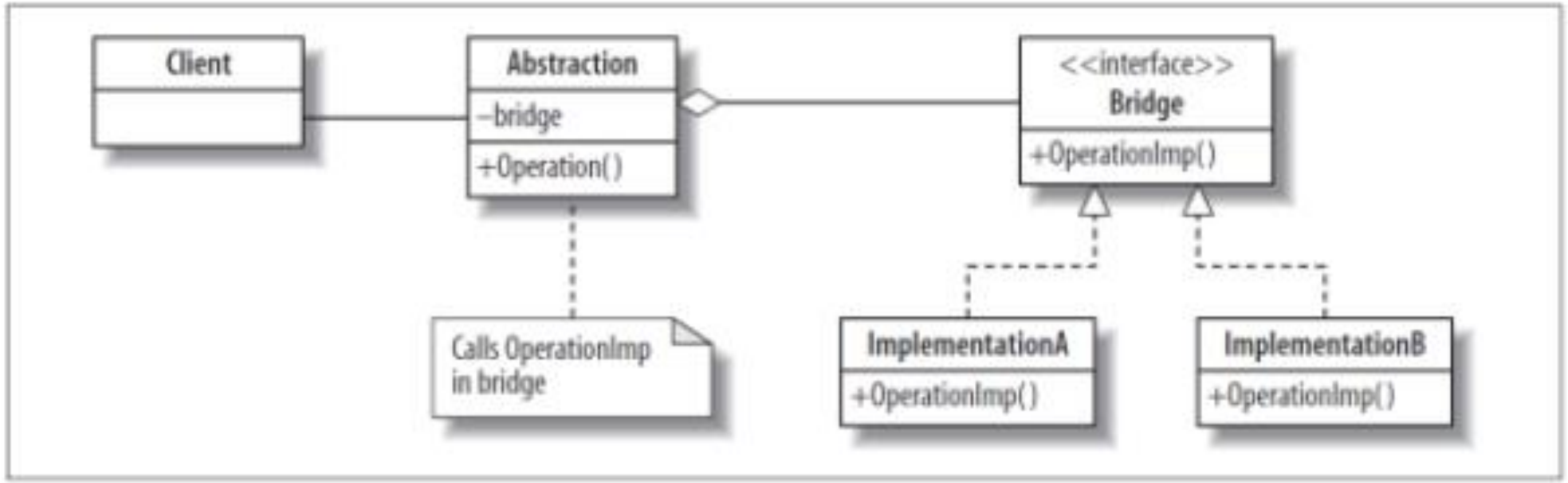
BRIDGE(KÖPRÜ) TASARIM KALIBI

Soyutlama(Abstraction): Soyutlama arayüzünü tanımlar ve uygulamacı tipinde bir referans sağlar.
Uygulamacı(Implementor/Bridge): Uygulama sınıfları için arayüz tanımlar.
SomutUygulamacı(ConcreteImplementor): Uygulamacının arayüzünü uygular ve somut bir uygulama tanımlar.



https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/k%C3%B6pr%C3%BC(bridge)

BRIDGE(KÖPRÜ) TASARIM KALIBI



ImplementationA , ImplementationB : Esas fonksiyonaliteyi içerisinde barındıran classlar.
Bridge : ImplementionA ve ImplementationB classının türediği interface. Bu interface'in görevi abstraction ile Implementation classları arasında köprü görevi görmesi ve onları bağlamasıdır.
Abstraction : Abstraction classı, bridge üzerinden esas classlara ve onların metotlarına ulaşarak bunları clienta ulaştırır. Böylece implementasyon classlarını clienttan soyutlamış olur.

Görsel Kaynağı: http://cagataykiziltan.net/tr/tasarim-kaliplari-design-patterns/3-yapisal-tasarim-desenleri/bridge-tasarim-deseni/

Bridge.java

```
public interface Bridge {

    String operationImp();

}
```

ImplementationA.java

```
public class ImplementationA implements Bridge {

    @Override
    public String operationImp()
    {
       return "Implementation A";
    }

}
```

ImplementationB.java

```
public class ImplementationB implements Bridge {

    @Override
    public String operationImp()
    {
       return "Implementation B";
    }
}
```

Abstraction.java

```
public class Abstraction {

    Bridge bridge;

    public Abstraction(Bridge Implementation)
    {
        bridge = Implementation;
    }

    public String Operation()
    {
      return "Abstraction <> " + bridge.operationImp();
    }
}
```

Program.java(BridgePatterns)

```
public class BridgePatterns {



    public static void main(String[] args) {

        System.out.println(new Abstraction(new ImplementationA()).Operation());
        System.out.println(new Abstraction(new ImplementationB()).Operation());

    }
}
```

Composite(Kompozit/Bileşik) Tasarım Kalıbı

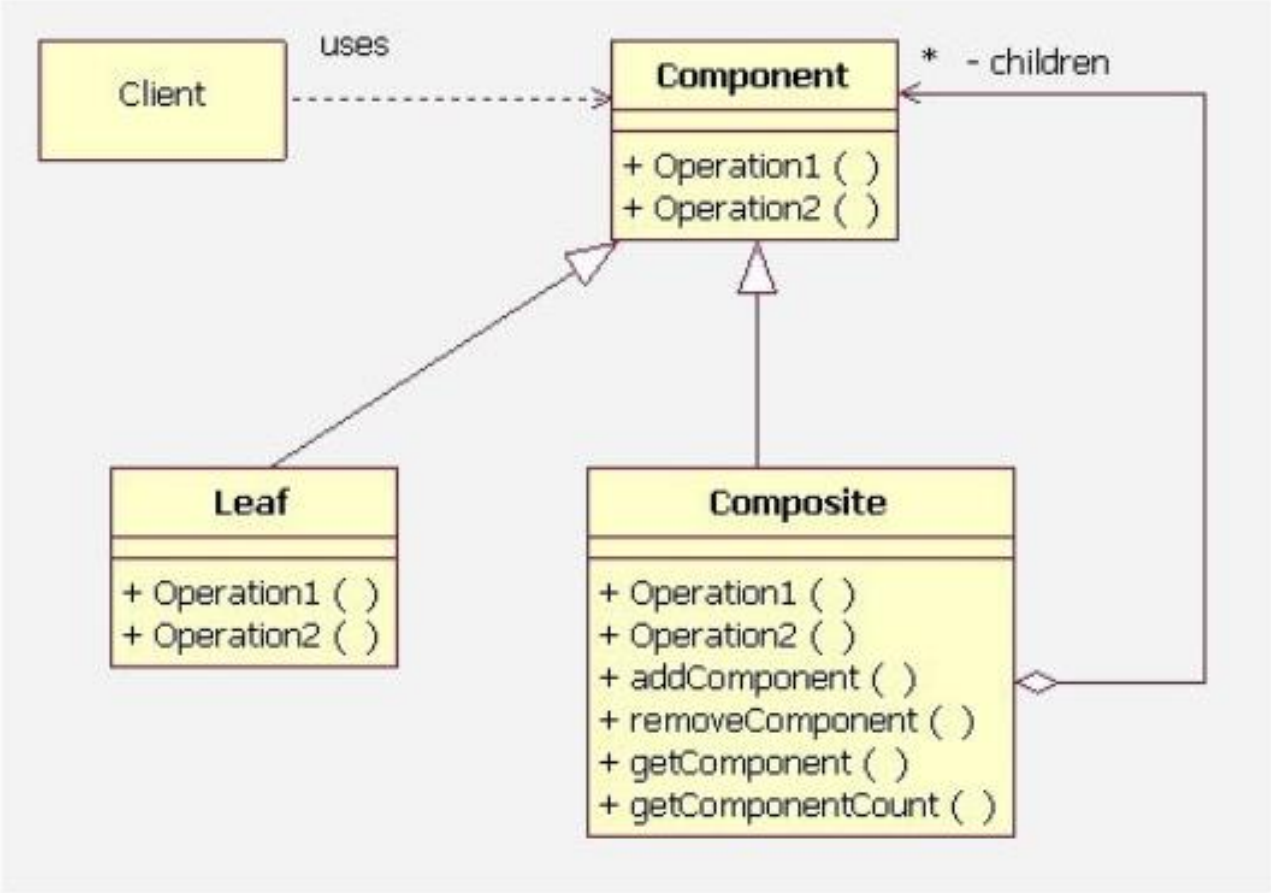
- Kendi içlerinde birbirlerinden farklı olan bir grup nesnenin sanki tek bir bütün nesneymiş gibi kullanılması için kullanılır.
- Composite kalıpların görevi, nesneleri bir ağaç yapısında birleştirip uygulamanın genelindeki parça bütün ilişkisini yeniden düzenleyip şekillendirmektir.
- Bileşik kalıpların görevleri nesneleri bir ağaç yapısında birleştirip uygulamanın genelindeki parça bütün ilişkisini yeniden düzenleyip şekillendirmektir.

https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/bile%C5%9Fik-tasar%C4%B1m-kal%C4%B1b%C4%B1-(composite-design-pattern)

Composite(Kompozit/Bileşik) Tasarım Kalıbı

Component(Bileşen):

- Bileşikler için temel soyut tanımlamalardır.
- Bileşik işlemi için nesnelerin arayüzünü oluşturur.
- Tüm sınıfların arayüzündeki varsayılan davranışı gerçekleştirir.
- Yavru bileşenlere ulaşmamızı ve onları kontrol etmemizi sağlamak için bir arayüz tanımlar.



https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/bile%C5%9Fik-tasar%C4%B1m-kal%C4%B1b%C4%B1-(composite-design-pattern)

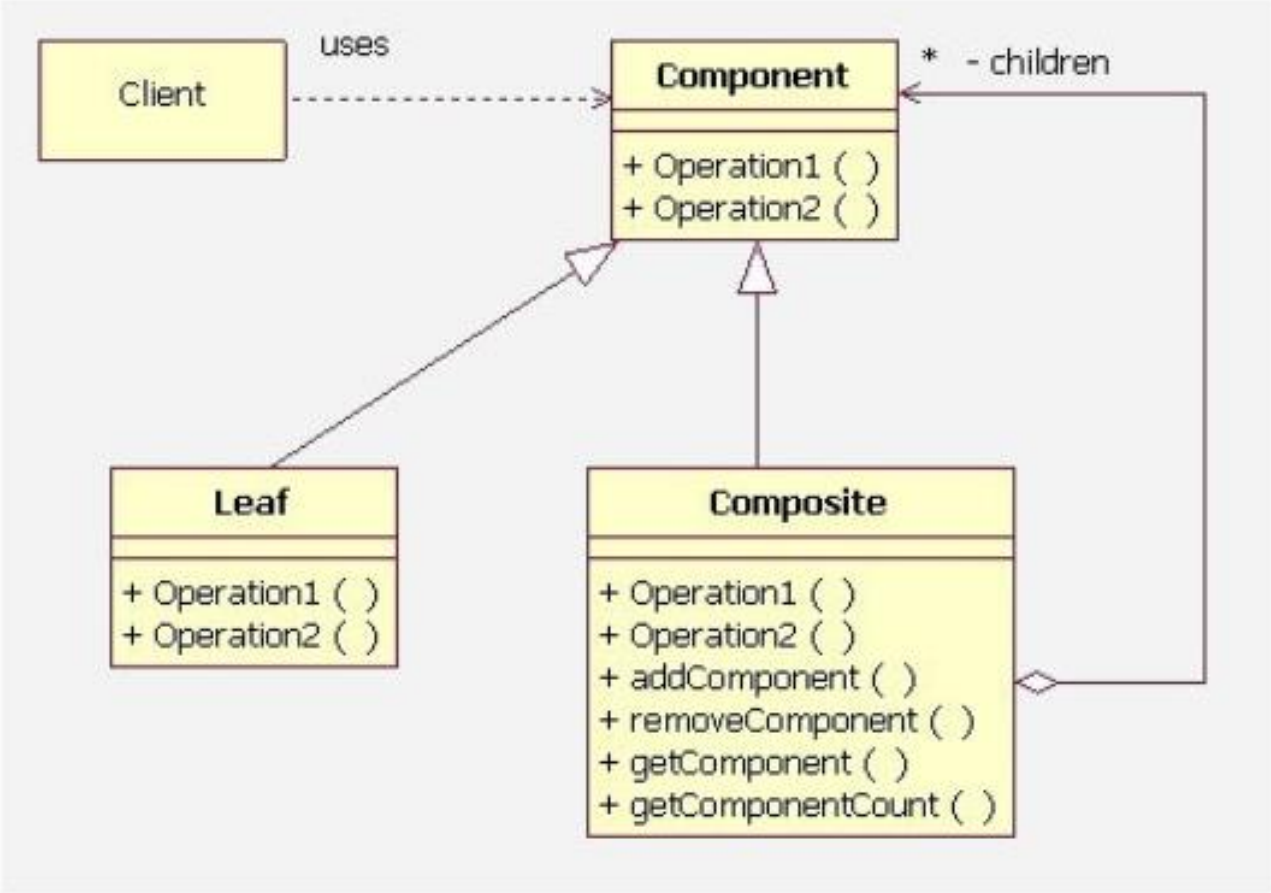
Composite(Kompozit/Bileşik) Tasarım Kalıbı

Yaprak(Leaf):

- Bileşik işleminde yavru nesneleri temsil eder.
- Tüm bileşen metodları yapraklar tarafından tamamlanır.

Composite(Bileşik):

- Yaprakları olan bileşenleri temsil eder.
- Çocuklarını yönlendiren metodları gerçekler.
- Genelde çocuklarını görevlendirerek bileşik metodlarını gerçekler.



https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/bile%C5%9Fik-tasar%C4%B1m-kal%C4%B1b%C4%B1-(composite-design-pattern)

Composite Sınıfı

**Interface yada Abstract olabilir !!!

- Component Sınıfı(Abstract)

```
abstract class Component
{
  protected string name;
  public Component(string name)
  {
    this.name = name;
  }
  public abstract void Add(Component c);
  public abstract void Remove(Component c);
  public abstract void Display(int depth);
}
```

```
class Composite : Component
{
  private List<Component> _children = new List<Component>();
  public Composite(string name) : base(name)
  { }
  public override void Add(Component component)
  {
    _children.Add(component);
  }
  public override void Remove(Component component)
  {
    _children.Remove(component);
  }
  public override void Display(int depth)
  {
    Console.WriteLine(new String('-', depth) + name);
    foreach (Component component in _children)
    {
      component.Display(depth + 2);
    }
  }
}
```

https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/bile%C5%9Fik-tasar%C4%B1m-kal%C4%B1b%C4%B1-(composite-design-pattern)

Leaf Sınıfı

```
class Leaf : Component
{
  public Leaf(string name) : base(name)
  { }
  public override void Add(Component c)
  {
    Console.WriteLine("Cannot add to a leaf");
  }
  public override void Remove(Component c)
  {
    Console.WriteLine("Cannot remove from a leaf");
  }
  public override void Display(int depth)
  {
    Console.WriteLine(new String('-', depth) + name);
  }
}
```

Leaf tipi: Tek bir component tipini belirtmek için kullanılır.

Main Program

```
class MainApp
{
  static void Main()
  {
    Composite root = new Composite("root");
    root.Add(new Leaf("Leaf A"));
    root.Add(new Leaf("Leaf B"));

    Composite comp = new Composite("Composite X");
    comp.Add(new Leaf("Leaf XA"));
    comp.Add(new Leaf("Leaf XB"));

    root.Add(comp);
    root.Add(new Leaf("Leaf C"));

    Leaf leaf = new Leaf("Leaf D");
    root.Add(leaf);
    root.Remove(leaf);

    root.Display(1);

    Console.ReadKey();
  }
}
```

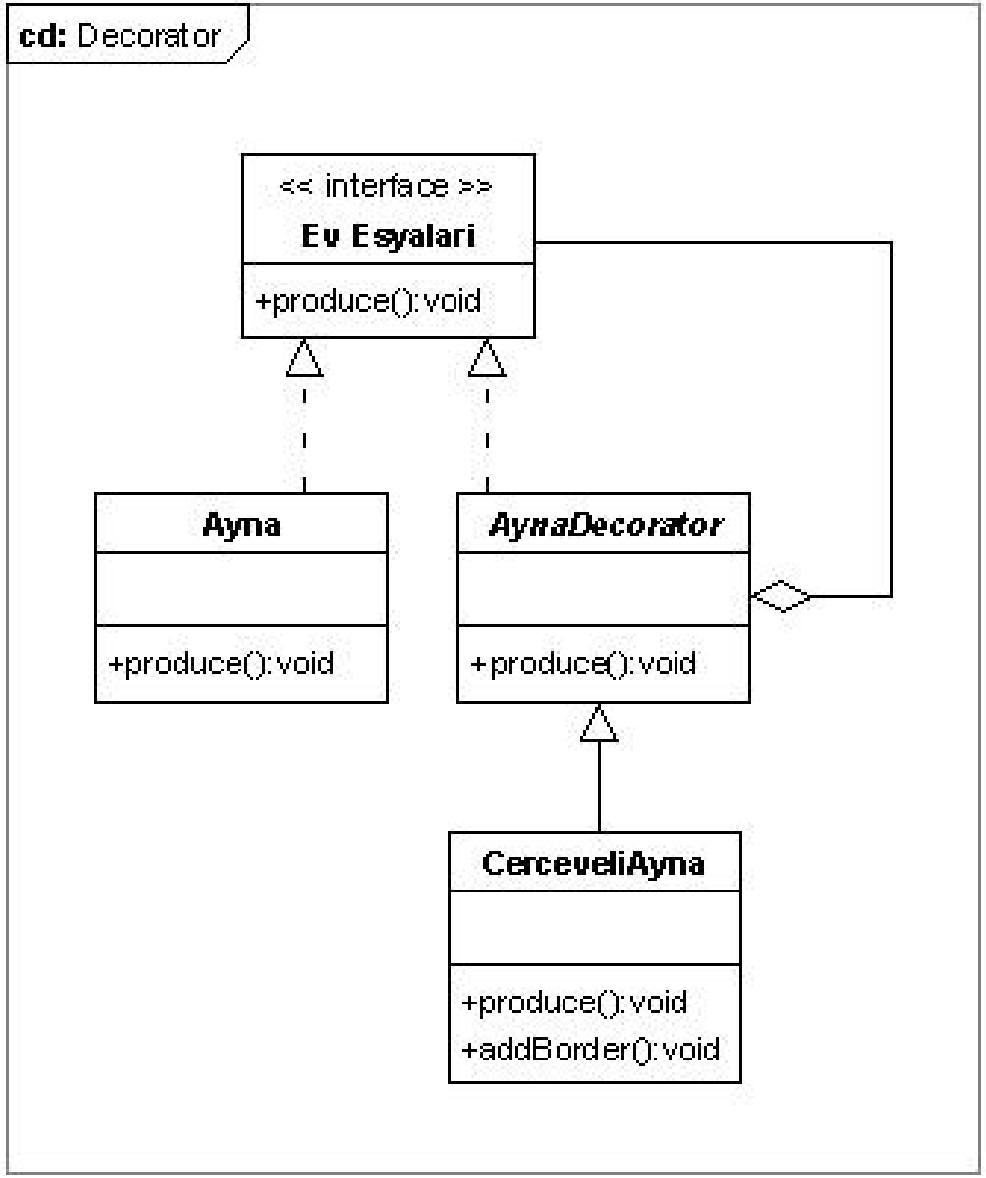
https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/bile%C5%9Fik-tasar%C4%B1m-kal%C4%B1b%C4%B1-(composite-design-pattern)

Decorator(Dekoratör) Tasarım Kalıbı

- Dekoratör tasarım kalıbının temel amacı, var olan nesneye, çalışma zamanında, başka sınıflar oluşturmaya gerek kalmadan yeni durum ve davranışlar eklemektir.
- Dekoratör kalıbında nesne oluşturularak üzerinde birtakım değişiklikler, eklemeler yapılır; orijinal nesne bu değişikliklerden etkilenmez.
- Temel sınıf üzerinde köklü değişiklikler olmaz ve her nesne üzerinde yapılan değişiklikler birbirinden bağımsızdır.
- Nesnelere dinamik olarak özellik eklemek için kullanılırlar.

Decorator Tasarım Kalıbı

- Nesnelere, sahip oldukları sınıfların yapılarının değiştirilmeden yeni özelliklerin eklenmesini sağlsayan Decorator ile, istenilen özelliklerin ekleneceği nesne başka bir nesne içine gömülür.
- Yeni özellik eklenen nesneyi içine alan nesneye/sınıfa decorator denir.



http://www.kurumsaljava.com/2010/07/24/decorator-tasarim-sablonu/

Decorator Tasarım Kalıbı

EvEsyaları İnterface'i

```
public interface EvEsyalari
{
        /**
         * Üretimi gerceklestirmek için
         * kullanilan metod.
         */
        public void produce();
}
```

Ayna Sınıfı

```
public class Ayna implements EvEsyalari
{

        public void produce()
        {
                System.out.println("Ayna imal edildi.");
        }
}
```

http://www.kurumsaljava.com/2010/07/24/decorator-tasarim-sablonu/

Decorator Tasarım Kalıbı

AynaDecorator Sınıfı

```
public abstract class AynaDecorator implements EvEsyalari
    {
        /*
         * Bünyesinde mevcut bir ayna nesnesi
         * bulundurur ve degisik metodlar kullanarak
         * bu ayna nesnesini dekore eder.
         */
        private EvEsyalari ayna = new Ayna();

        public EvEsyalari getAyna()
        {
            return ayna;
        }

        public void setAyna(EvEsyalari ayna)
        {
            this.ayna = ayna;
        }
    }
```

http://www.kurumsaljava.com/2010/07/24/decorator-tasarim-sablonu/

Decorator Tasarım Kalıbı

CerceveliAyna Sınıfı

```
public class CerceveliAyna extends AynaDecorator
{

        /**
         * Üretim için kullanilan sinif.
         * addBorder metodu ile
         * aynaya cerceve ekler.
         */
        public void produce()
        {
                getAyna().produce();
                addBorder();
        }

          * Cerceve ekleme islemini gerceklestirmek
          * için kullanilan metod.
          *
          */
         public void addBorder()
         {
                System.out.println("Aynaya cerceve eklendi.");
         }
}
```

Önce bir Ayna nesnesi üretiliyor ve sonra AddBorder metodu ile çerçeve ekleniyor.

http://www.kurumsaljava.com/2010/07/24/decorator-tasarim-sablonu/

Decorator Tasarım Kalıbı

- Main Program

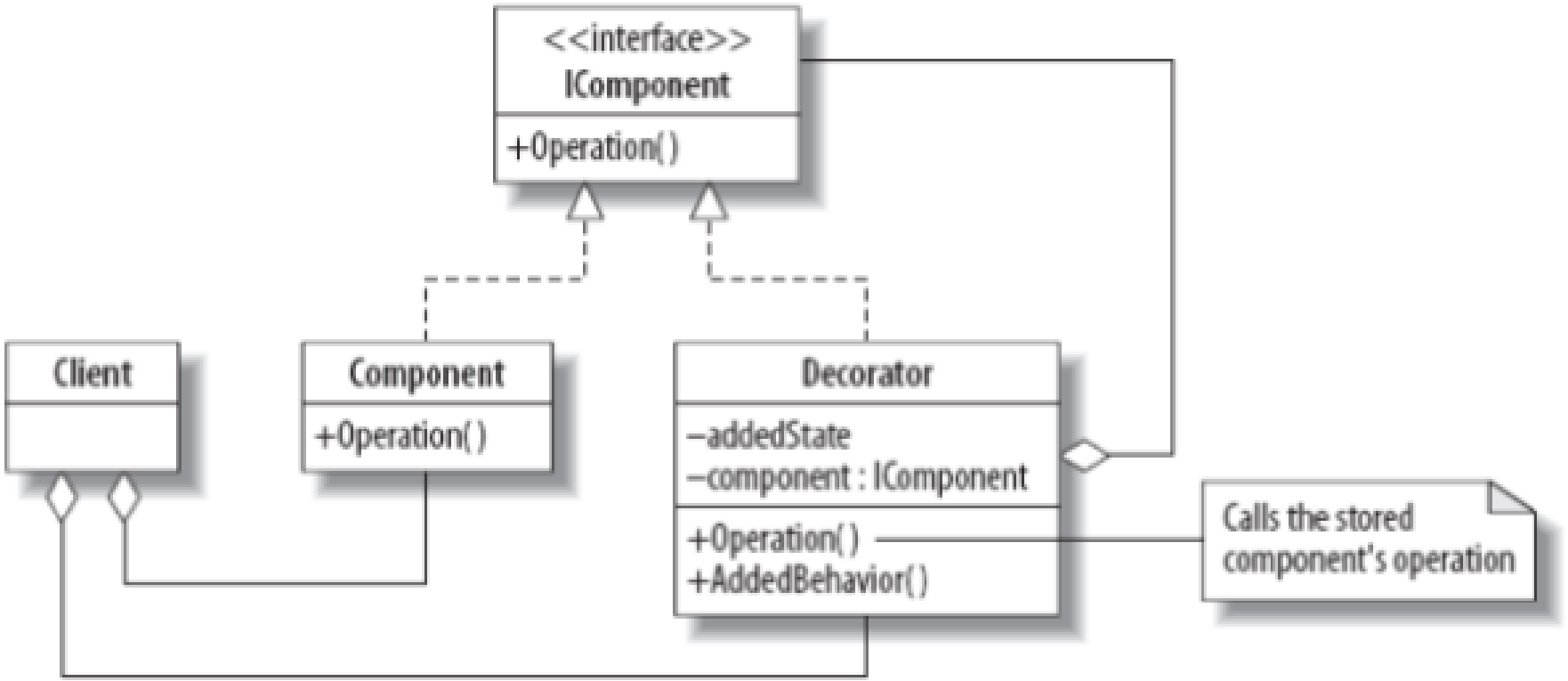
```
public class Test
{

        public static void main(String[] args)
        {
                EvEsyalari ayna = new CerceveliAyna();
                ayna.produce();
        }
}
```

*Decorator üzerinde tanımlanmış olan CerveliAyna sınıfı üzerinden yeni bir özellik ekleyerek ayna sınıfı nesnesi yaratıldı.

http://www.kurumsaljava.com/2010/07/24/decorator-tasarim-sablonu/

Decorator(Dekoratör) Tasarım Kalıbı



http://cagataykiziltan.net/tr/tasarim-kaliplari-design-patterns/3-yapisal-tasarim-desenleri/2448-2/

Decorator(Dekoratör) Tasarım Kalıbı

- Component class : Üzerine dinamik olarak özellik ve davranış eklenecek olan nesnenin classı
- Decorator class : Dinamik olarak özelliğin ve davranışın eklendiği class.
- IComponent : Component ve Decorator classın türediği interface.

http://cagataykiziltan.net/tr/tasarim-kaliplari-design-patterns/3-yapisal-tasarim-desenleri/2448-2/

Decorator Örnek

Icomponent

```
interface IComponent
    {
      String Operation();
    }
```

Component

```
class Component implements IComponent
{
    public String Operation()
   {
      return "I am walking ";
   }
}
```

http://cagataykiziltan.net/tr/tasarim-kaliplari-design-patterns/3-yapisal-tasarim-desenleri/2448-2/

DecoratorA

```
class DecoratorA : IComponent
    {
        IComponent component;

        public DecoratorA(IComponent c)
        {
            component = c;
        }

        public string Operation()
        {
          string s = component.Operation();
          s += " and listening to Classin FM";

          return s;
        }
    }
```

DecoratorB

```
class DecoratorB : IComponent
    {
        IComponent component;
        public string addedState = "past the coffe shop ";

        public DecoratorB(IComponent c)
        {
          component = c;
        }

        public string Operation()
        {
          string s = component.Operation();
          s += " to School";

          return s;
        }

        public string AddedBehavior()
        {
            return " and I bought a cappicuno";
        }
    }
```

http://cagataykiziltan.net/tr/tasarim-kaliplari-design-patterns/3-yapisal-tasarim-desenleri/2448-2/

Decorator Örnek

Client

```
static void Main(string[] args)
{
    IComponent component = new Component();

    Console.WriteLine("Basic Component " + component.Operation());
    Console.WriteLine("Decorator A " + new DecoratorA(component).Operation());
    Console.WriteLine("Decorator B " + new DecoratorB(component).Operation());
    Console.WriteLine("Decorator B - A" + new DecoratorB(new DecoratorA(component)).Operation());

    DecoratorB b = new DecoratorB(new Component());

    Console.WriteLine("b.addedState " + b.AddedBehavior());

    Console.ReadKey();

}
```

http://cagataykiziltan.net/tr/tasarim-kaliplari-design-patterns/3-yapisal-tasarim-desenleri/2448-2/

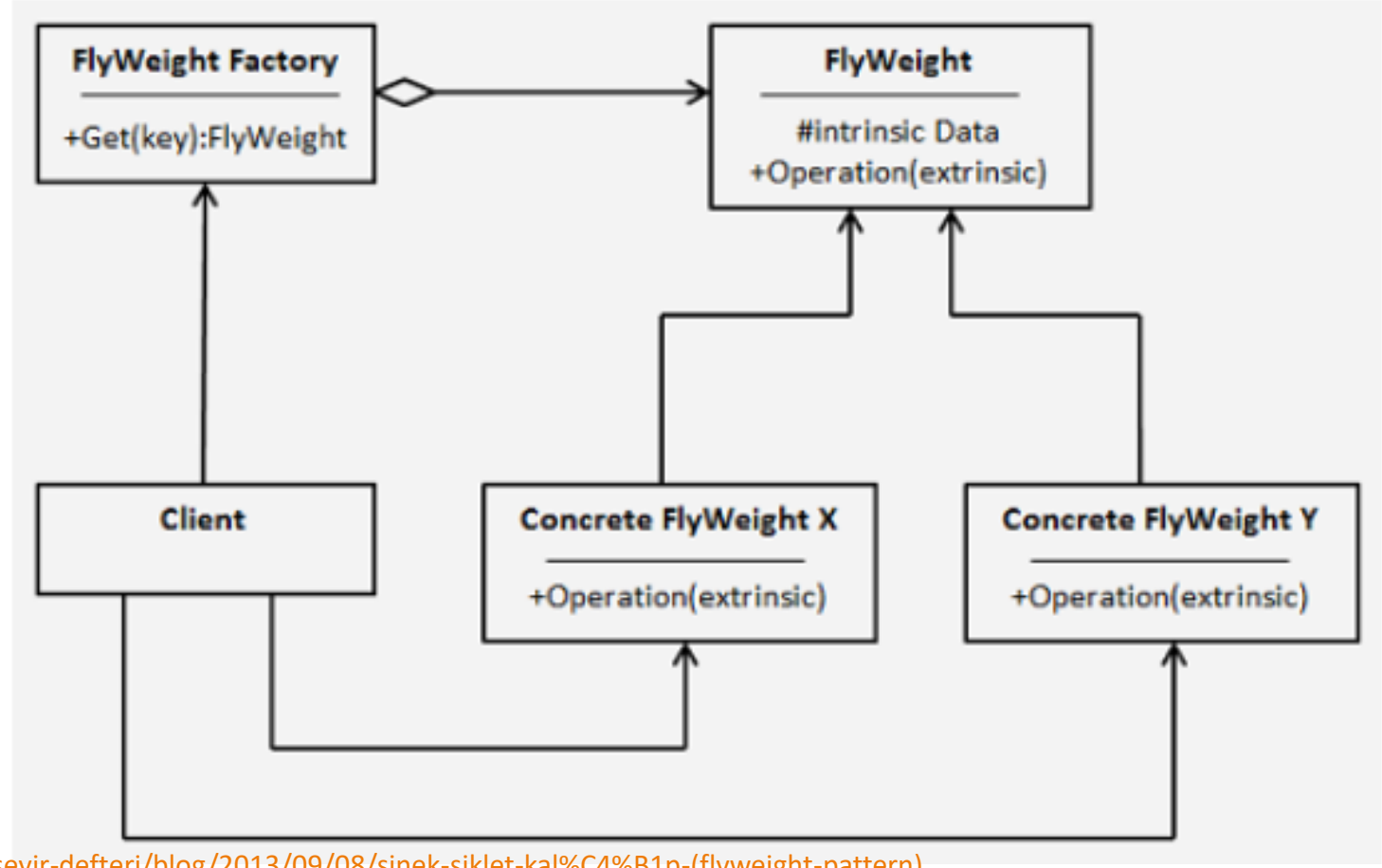
Flyweight(Sinek Siklet) Kalıp

- Flyweight kalıbında amaç, yapıca aynı nesneleri bellekte çokça oluşturmak yerine her bir nesnenin bir kopyasını oluşturmak ve oluşturulan nesneleri ortak bir noktada tutup paylaştırma işlemini yerine getirmektir.
- Tekrar eden aynı nesneleri gruplayarak hafızada çok fazla yer kaplamaması için ,hafıza kullanımını minumuma indirmektir.

Flyweight(Sinek Siklet) Kalıp

- Her bir Flyweight nesnesi temel olarak iki veri kümesinden oluşur.
- **Dahili (intrinsic)** olarak geçen **durum-bağımsız (state-independent)** kısım parçalardan biridir.
- Bu kısımda, çalışma zamanındaki tüm Flyweight nesneleri tarafından saklanan paylaşılmış alanlar yer almaktadır.
- Diğer parça ise **durum-bağımlı (state-dependent**) olarak bilinen ve **dışsal (extrinsic)** olarak belirtilen kısımdır.
- Bu kümedeki veriler ise istemci tarafından saklanır, hesap edilir ve Flyweight nesne örneğine, yine Flyweight'in bir operasyonu yardımıyla aktarılırlar.

Flyweight(Sinek Siklet) Kalıp



https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/sinek-siklet-kal%C4%B1p-(flyweight-pattern)

Flyweight(Sinek Siklet) Kalıp

- Her bir Flyweight nesnesi temel olarak iki veri kümesinden oluşur.
- **FlyWeight**: Nesnenin ortak özelliklerini tutan arabirim (interface) veya soyut (abstract) sınıf (class)tır..
- **ConcreteFlyWeight**: Flyweight şablonunu uygulayan farklı nesneleri içeren sınıflardır.
- **FlyWeightFactory**: Nesneleri ortak bir noktada tutan ve paylaşımını sağlayan sınıftır.
- **Client**: İstemci uygulamadır.

Proxy Tasarım Kalıbı(Vekil Kalıp)

- Var olan bir nesneye ulaşılmak istendiğinde vekil kalıp oluşturulur.
- Nesneyle istemci arasına yeni bir katman koyarak nesnenin kontrollü bir şekilde paylaşılması sağlanır.
- Böylece istemci, işlem yapan sınıfla doğrudan temasa geçmemiş olur.
- Bu durum sayesinde işlemin yapılma performansında bir düşüklük olmaması sağlanır.
- Bu yüzden vekil kalıp fazla yük getiren işlemlerde kullanılır.

https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/vekil-kal%C4%B1p-(proxy-pattern)

Proxy Tasarım Kalıbı(Vekil Kalıp)

- Örnek; bir film sitesinden film izlenirken, filmin indirilmesi beklenmez. Arka tarafta vekil tasarım kalıbı oluşturularak parça parça işlem yapılır ve zaman kaybı önlenmiş olur. Burada film **gerçek nesne (realsubject),** izlenen ise **vekil nesne (proxy)** olur.

https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/vekil-kal%C4%B1p-(proxy-pattern)

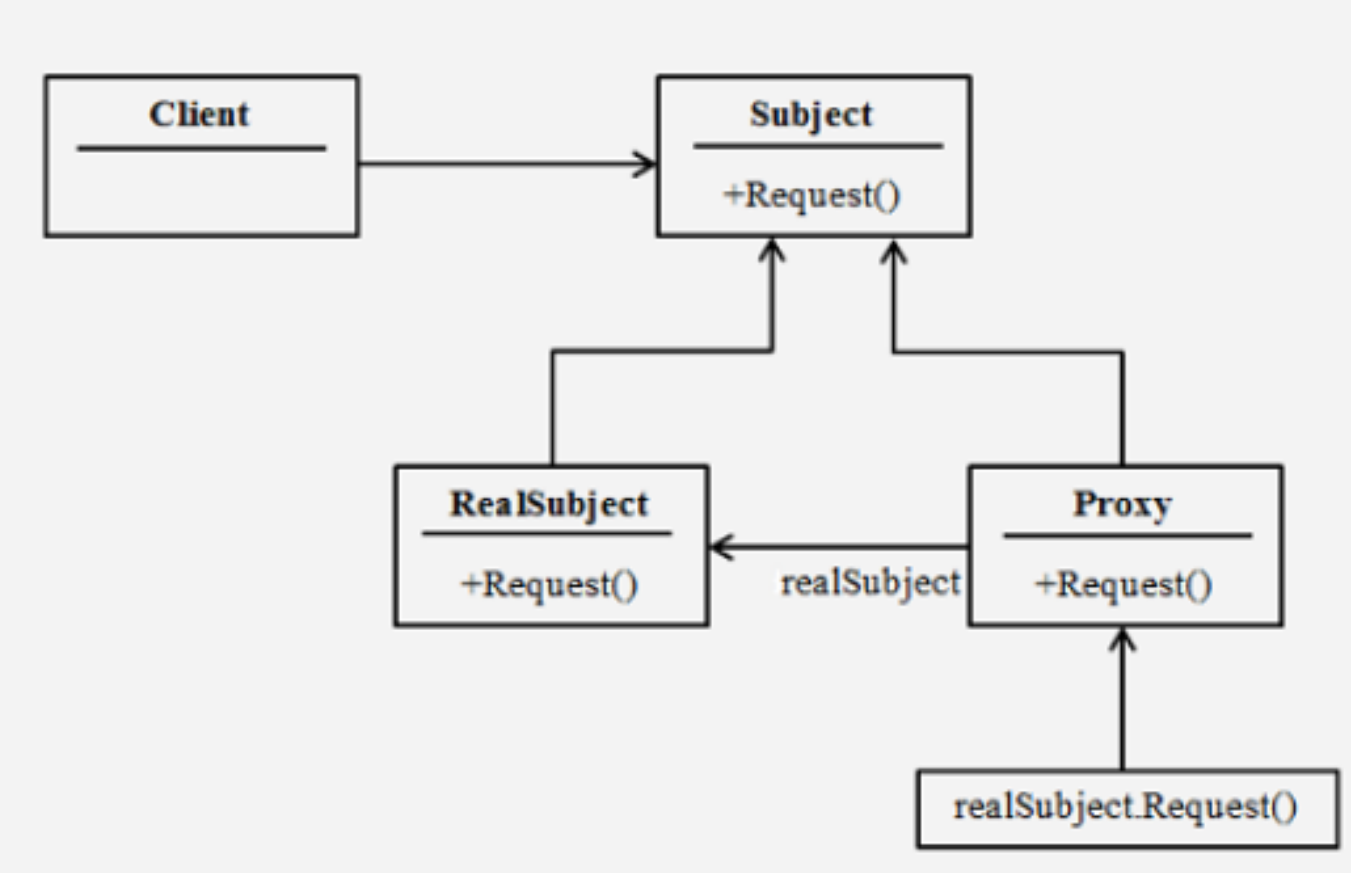
Proxy Tasarım Kalıbı(Vekil Kalıp)

Vekil tasarım deseni;

- Oluşturulması zaman alan bir nesne yaratılması gerektiğinde
- Uzaktan erişilerek bir nesne yaratılması gerektiğinde
- Nesneye erişmeden önce bazı kontroller yapılması gerektiğinde
- Nesneye erişimin kısıtlı olduğunda yararlı olabilir.

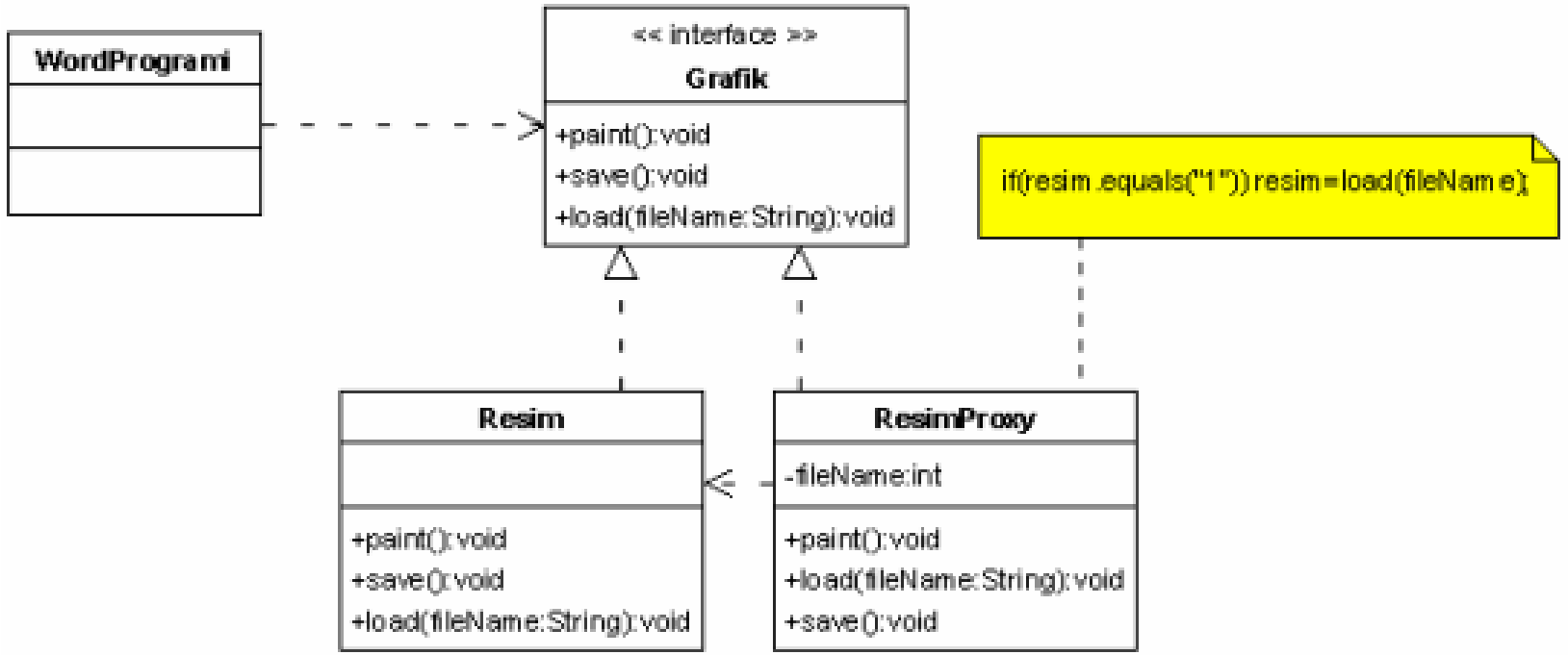
https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/vekil-kal%C4%B1p-(proxy-pattern)

Proxy Tasarım Kalıbı(Vekil Kalıp)



https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/vekil-kal%C4%B1p-(proxy-pattern)

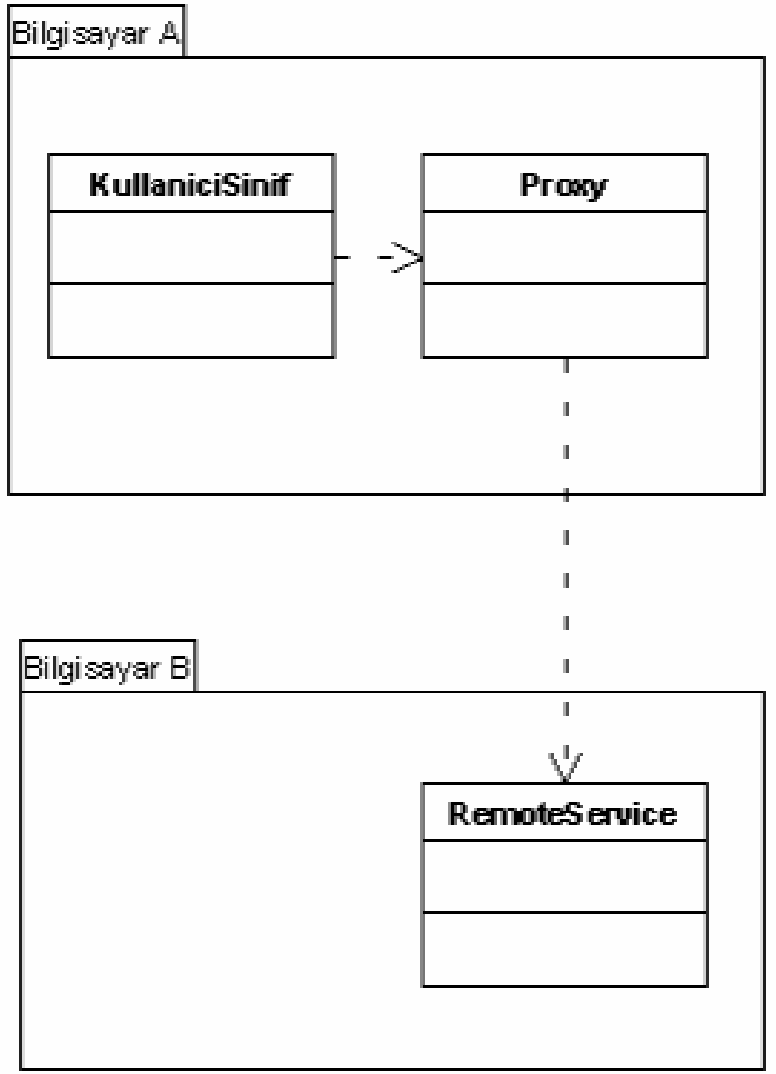
Virtual Proxy Pattern UML Örnek



http://www.kurumsaljava.com/download/48/

Remote Proxy Pattern UML Örnek

- **Remote Method Invocation(RMI):**RMI farklı sunucularda olan metodların başka bir sunucudan çağırılarak kullanılmasında kullanılmaktadır.
- Java dilinde, başka bir bilgisayarın hafızasında bulunan bir nesnenin sunduğu servise ulaşabilmek için RMI teknolojisi kullanılır.
- RMI, bir bilgisayardan diğer bir bilgisayara TCP/IP2 Protokolü ile baglantı kurup, bir nesnenin sahip olduğu metodları, o nesnenin, aynı adres alanı içinde bulunuyormuşcasına kullanımını sağlayan bir protokoldür.



http://www.kurumsaljava.com/download/48/

Kaynaklar

- Java ve Java Teknolojileri, *Tevfik KIZILÖREN* – Kodlab Yayınları
- Dr Öğr. Üyesi Zehra Aysun ALTIKARDEŞ Nesne Yönelimli Programlama 2 Ders notları
- Yazılım Mühendisliği CBU-Dr. Öğr. Üyesi Deniz Kılınç Yazılım Mimarisi ve Tasarımı Ders Notları
- http://cagataykiziltan.net/tr/tasarim-kaliplari-design-patterns/3-yapisal-tasarim-desenleri/bridge-tasarim-deseni/
- http://www.farukbozan.com/2015/06/bridge-design-pattern/
- https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/k%C3%B6pr%C3%BC(bridge)
- https://medium.com/gokhanyavas/structural-patterns-yap%C4%B1sal-desenler-7c84f174b7ae
- https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/bile%C5%9Fik-tasar%C4%B1m-kal%C4%B1b%C4%B1-(composite-design-pattern)
- https://www.buraksenyurt.com/post/Tasarc4b1m-Desenleri-Composite
- https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/bile%C5%9Fik-tasar%C4%B1m-kal%C4%B1b%C4%B1-(composite-design-pattern)
- https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/dekorat%C3%B6r-(decorator)
- http://www.kurumsaljava.com/2010/07/24/decorator-tasarim-sablonu/
- http://cagataykiziltan.net/tr/tasarim-kaliplari-design-patterns/3-yapisal-tasarim-desenleri/2448-2/
- https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/sinek-siklet-kal%C4%B1p-(flyweight-pattern)
- https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/vekil-kal%C4%B1p-(proxy-pattern)
- https://blog.burakkutbay.com/java-remote-method-invocation-nedir.html/