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

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

İçerik

- Behavioral Patterns(Davranışsal Kalıplar)
- Strategy Tasarım Kalıbı
- Observer Tasarım Kalıbı
- Command Tasarım Kalıbı
- Iterator Tasarım Kalıbı
- State Tasarım Kalıbı

Tasarım Kalıpları

Behavioral Patterns(Davranışsal Kalıplar)

- Chain of Responsibility Pattern
- Command Pattern
- Interpreter Pattern
- Iterator Pattern
- Mediator Pattern
- Memento Pattern
- Observer Pattern
- Null Object Pattern
- Strategy Pattern
- State Pattern
- Visitor Pattern

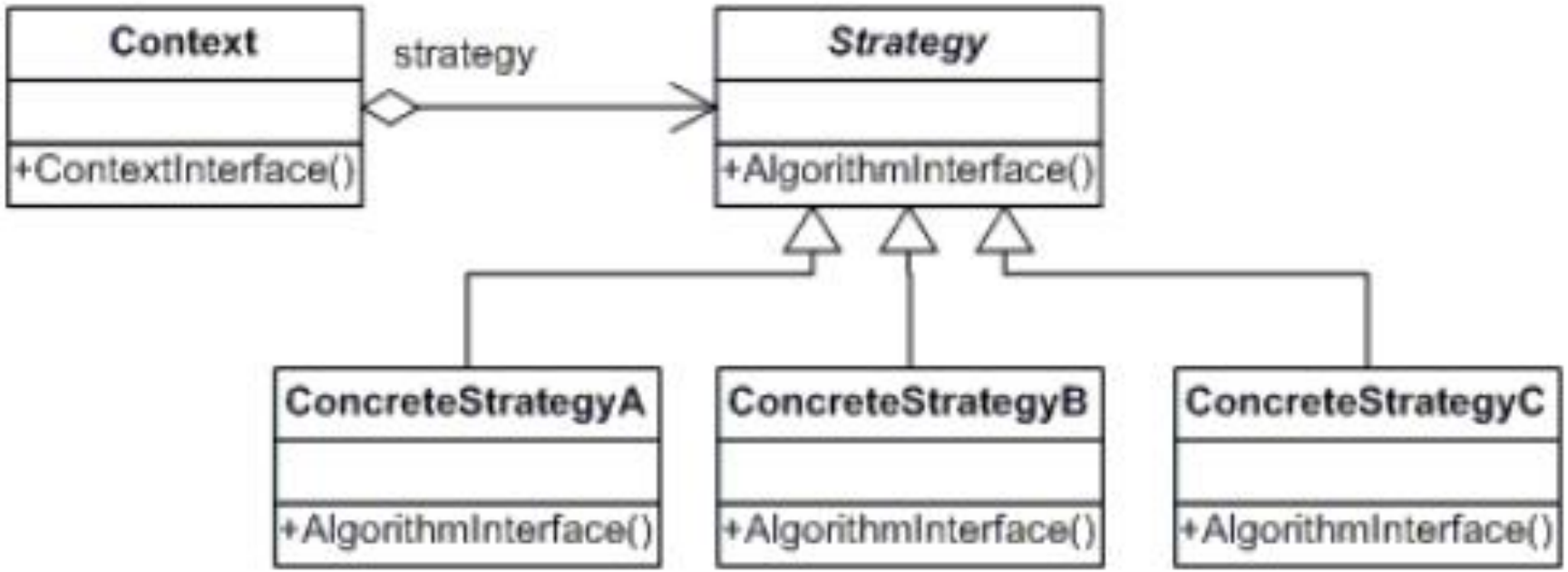
Davranışsal Tasarım Kalıpları

- Belirli görevlerin bu işleri yapmak üzere tasarlanmış nesnelere gönderilmesi ve geri dönen sonuçların işlenmesi davranışsal tasarım kalıpları içerisinde yapılan bir işlemdir.(bidb.itu.edu.tr)

Strategy Tasarım Kalıbı

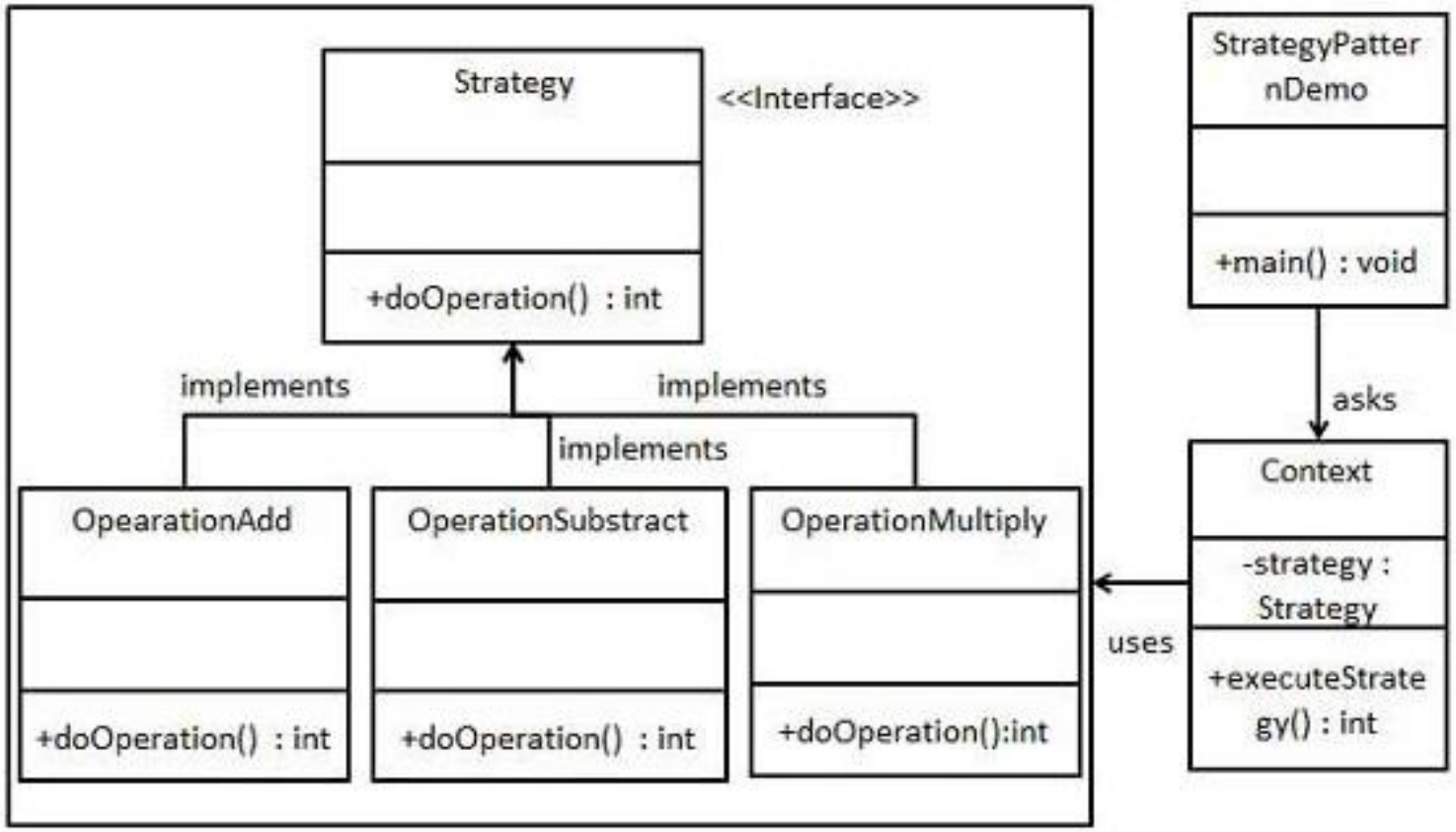
- Bir fonksiyonun birden fazla yapılış şekli olduğu takdirde, bu fonksiyonelliği farklı versiyonlarıyla kullanmak istendiğinde kullanılabilecek bir tasarım kalıbıdır.
- Aynı işi farklı şekillerde yapan sınıfları ortak bir interface üzerinden istemciye sunma yaklaşımıdır.
- Bir işlemin yerine getirilmesi için birden fazla metot kullanılıyor olabilir. İhtiyaçlara göre bir metodun seçilip, uygulaması için strateji tasarım kalıbı kullanılmaktadır. Her metot bir sınıf içinde tanımlanabilir.(bidb.itu.edu.tr)

Strategy Tasarım Kalıbı



http://cagataykiziltan.net/tr/tasarim-kaliplari-design-patterns/2-davranissal-tasarim-kaliplari/strategy-pattern/

Strategy Tasarım Kalıbı



https://medium.com/gokhanyavas/behavioral-patterns-davran%C4%B1%C5%9Fsal-desenler-11eec419589b

Strategy Tasarım Kalıbı/Örnek

- Crypt interface'inden türeyen CryptA, CryptB ve CryptC sınıfları var.
- İnterface altında encrypt(şifreleme) ve decrypt(şifre çözme) metotları olacak.
- Her Crypt sınıfı farklı algoritmalar kullanarak şifreleme gerçekleştirmektedir.

https://metinalniacik.medium.com/strategy-design-pattern-strateji-tasar%C4%B1m-%C3%B6r%C3%BCnt%C3%BCs%C3%BC-d7a43290969c

Strategy Tasarım Kalıbı/Örnek

```java
public interface Crypt {

    void encrypt(String text);

    void decrypt(String text);
}
```

```java
public class CryptA implements Crypt {

    public void encrypt(String text) {
        System.out.println(text + " is encrypting using CryptA");
    }

    public void decrypt(String text) {
        System.out.println(text + " is decrypting using CryptA");
    }
}
```

```java
public class CryptB implements Crypt {

    public void encrypt(String text) {
        System.out.println(text + " is encrypting using CryptB");
    }

    public void decrypt(String text) {
        System.out.println(text + " is decrypting using CryptB");
    }
}
```

https://metinalniacik.medium.com/strategy-design-pattern-strateji-tasar%C4%B1m-%C3%B6r%C3%BCnt%C3%BCs%C3%BC-d7a43290969c

Strategy Tasarım Kalıbı/Örnek

```
public class Test {

    public static void main(String[] args) {

        Crypt cryptA = new CryptA();
        cryptA.encrypt("Sifrelenecek metin1");
        cryptA.decrypt("Sifrelenecek metin1");

        System.out.println();

        Crypt cryptB = new CryptB();
        cryptB.encrypt("Sifrelenecek metin2");
        cryptB.decrypt("Sifrelenecek metin2");

        System.out.println();

        Crypt cryptC = new CryptC();
        cryptC.encrypt("Sifrelenecek metin3");
        cryptC.decrypt("Sifrelenecek metin3");
    }
}
```

https://metinalniacik.medium.com/strategy-design-pattern-strateji-tasar%C4%B1m-%C3%B6

Strategy Tasarım Kalıbı/Örnek 2

```
public interface DB {

    void select();

    void update();

    void delete();

    void insert();
}
```

```
public class MSSQL implements DB {

    @Override
    public void select() {
        System.out.println("\"Select\" command is running on \"MSSQL\"");
    }

    @Override
    public void update() {
        System.out.println("\"Update\" command is running on \"MSSQL\"");
    }

    @Override
    public void delete() {
        System.out.println("\"Delete\" command is running on \"MSSQL\"");
    }

    @Override
    public void insert() {
        System.out.println("\"Insert\" command is running on \"MSSQL\"");
    }
}
```

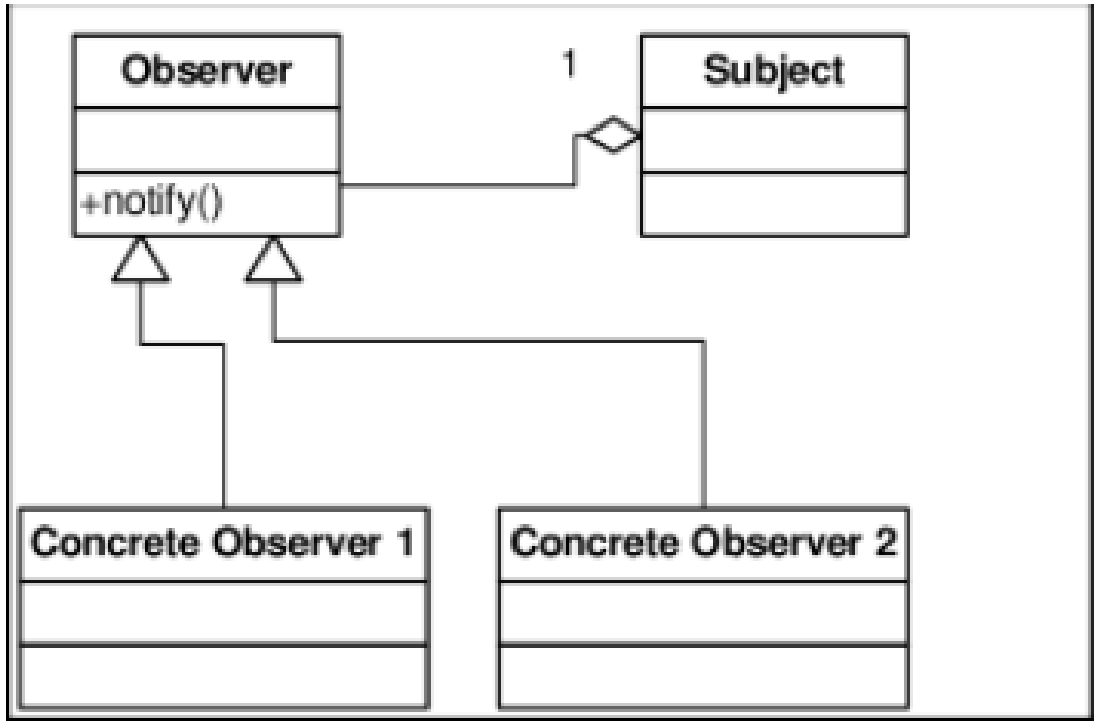
Strategy Tasarım Kalıbı/Örnek 2

```
public class MySQL implements DB {

    @Override
    public void select() {
        System.out.println("\"Select\" command is running on \"MySQL\"");
    }

    @Override
    public void update() {
        System.out.println("\"Update\" command is running on \"MySQL\"");
    }

    @Override
    public void delete() {
        System.out.println("\"Delete\" command is running on \"MySQL\"");
    }

    @Override
    public void insert() {
        System.out.println("\"Insert\" command is running on \"MySQL\"");
    }
}
```

Observer(Gözlemci) Tasarım Kalıbı

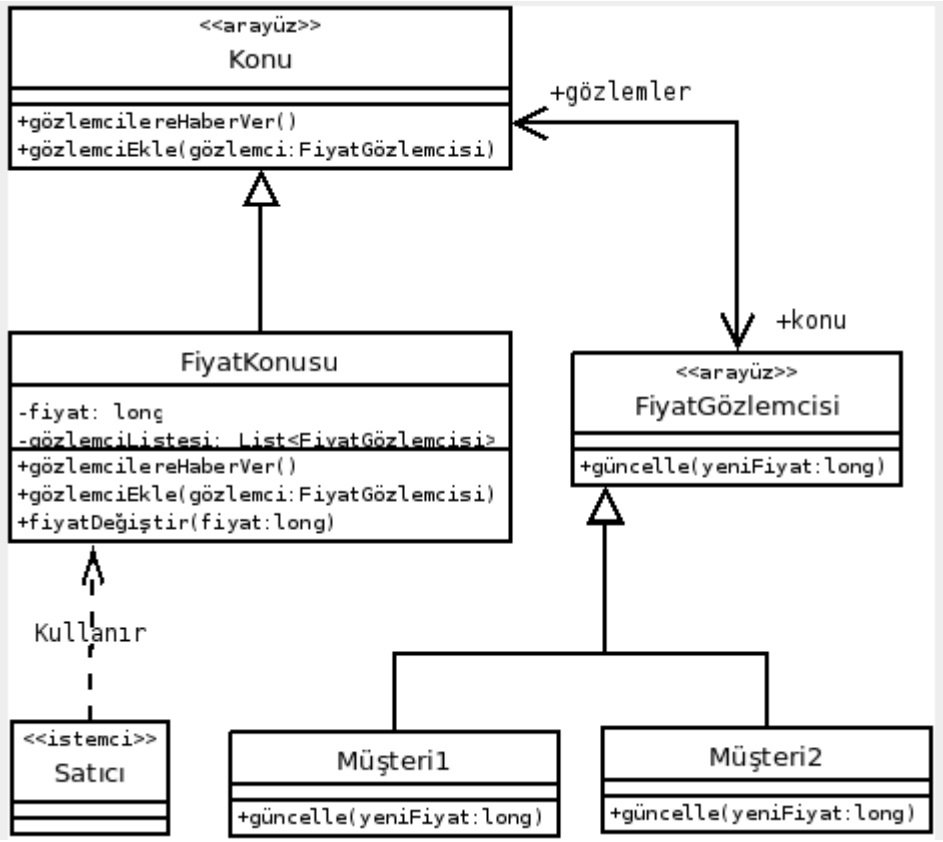
- Uygulama içerisinde bir nesnede meydana gelen değişikliklerden haberdar olup üzerinde belli metotları çalıştırıp değişlikler yapmak isteyen diğer nesneler bulunabilmektedir.
- Bu durumda haberdar olmak isteyen nesneler diğer nesne ile ilişkilendirilerek, ilişkili oldukları nesnede meydana gelen değişikliklerden haberdar edilebilmektedirler.

Observer(Gözlemci) Tasarım Kalıbı



http://cagataykiziltan.net/tr/tasarim-kaliplari-design-patterns/2-davranissal-tasarim-kaliplari/observer-tasarim-kalibi/

Observer(Gözlemci) Tasarım Kalıbı



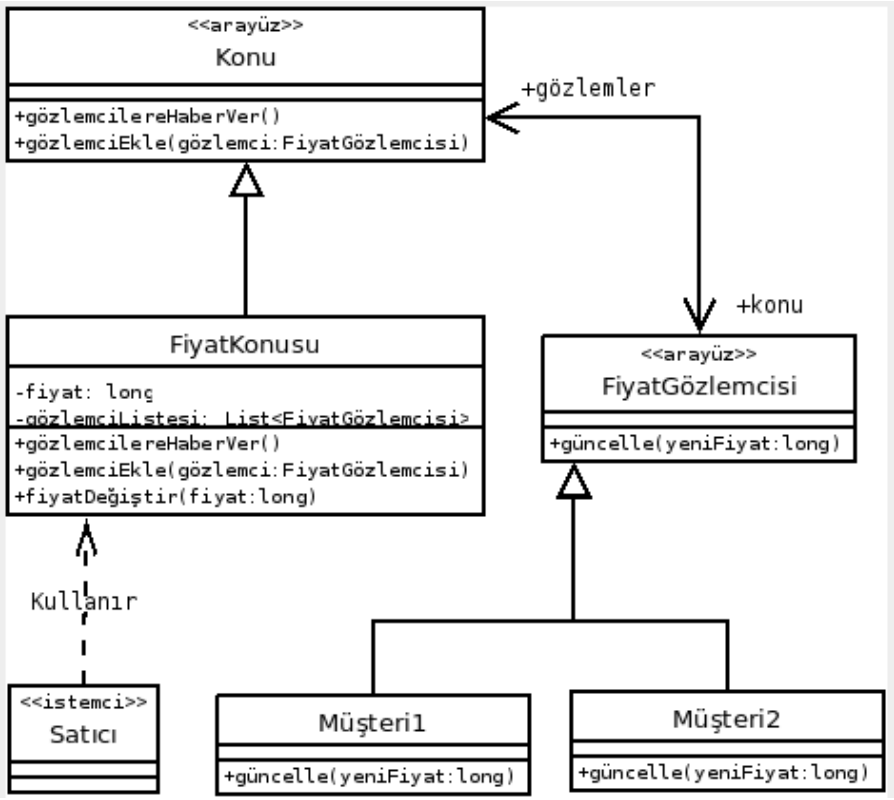

```java
public interface FiyatGözlemcisi {

  void güncelle( long yeniFiyat );
}
```

```java
public class Müşteri1 implements FiyatGözlemcisi {

  @Override
  public void güncelle( final long yeniFiyat ) {
    System.out.println( "Müşteri 1, yeni fiyatı öğrendi: " + yeniFiyat );
  }
}
```

http://www.tasarimdesenleri.com/jsp/tasdesincele/observer.jsp

Observer(Gözlemci) Tasarım Kalıbı

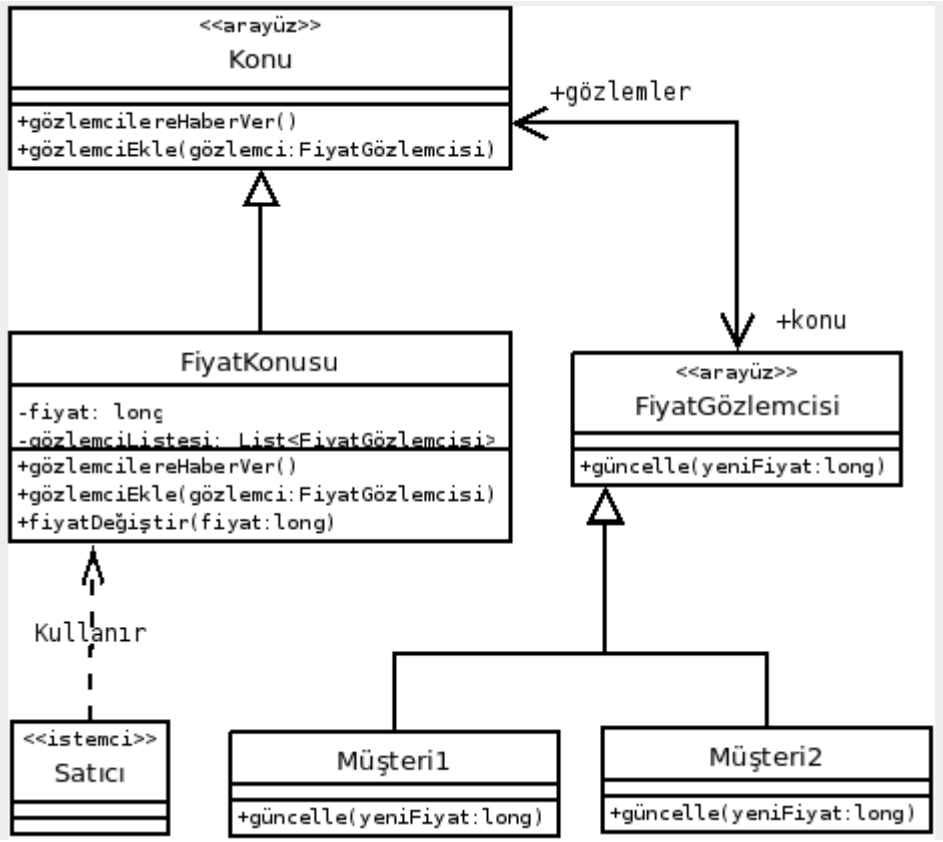
```java
public interface Konu {

  void gözlemcilereHaberVer();

  void gözlemciEkle( FiyatGözlemcisi gözlemci );
}
```




```java
public class FiyatKonusu implements Konu {

  private final List< FiyatGözlemcisi >  gözlemciListesi  = new ArrayList< FiyatGözlemcisi >();
  private long                fiyat;

  @Override
  public void gözlemcilereHaberVer() {
    for ( final FiyatGözlemcisi gözlemci : gözlemciListesi ) {
      gözlemci.güncelle( fiyat );
    }
  }

  @Override
  public void gözlemciEkle( final FiyatGözlemcisi gözlemci ) {
    gözlemciListesi.add( gözlemci );
  }

  public void fiyatDeğiştir( final long fiyat ) {
    this.fiyat = fiyat;
    gözlemcilereHaberVer();
  }
}
```

http://www.tasarimdesenleri.com/jsp/tasdesincele/observer.jsp

Observer(Gözlemci) Tasarım Kalıbı



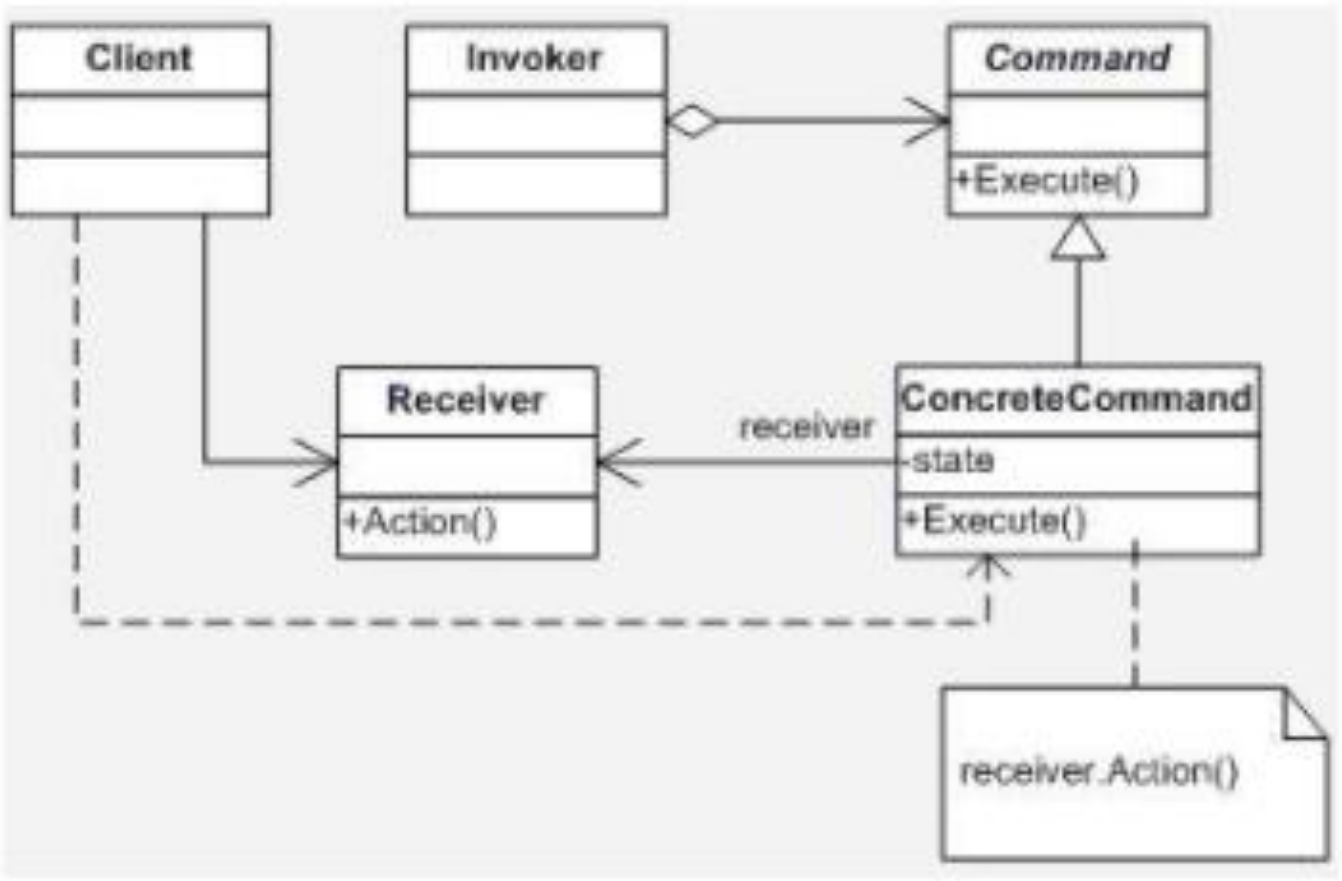

```java
public class Satıcı {

  public static void main( final String[] args ) {

    // Bir gözlem konusu yaratılıyor.
    final FiyatKonusu konu = new FiyatKonusu();

    // Gözlemci 1 yaratılıyor ve gözlem konusuna ekleniyor.
    konu.gözlemciEkle( new Müşteri1() );

    // Gözlemci 2 yaratılıyor ve gözlem konusuna ekleniyor.
    konu.gözlemciEkle( new Müşteri2() );

    // Gözlem konusundaki bir değişiklik, gözlemcilere haber veriliyor.
    konu.fiyatDeğiştir( 50L );
  }
}
```

http://www.tasarimdesenleri.com/jsp/tasdesincele/observer.jsp

Command(Komut) Tasarım Kalıbı

- Bir nesne üzerindeki işlemlerin yapılmasındaki süreci bilmediğimizde ya da kullanılmak istenen nesnenin tanınmadığı durumlarda, komut tasarım kalıbı ile yapılmak istenen işlem bir nesneye dönüştürülerek, alıcı nesne tarafından işlemin yerine getirilmesi amaçlanmaktadır.(bidb.itu.du.tr)
- Nesneye dayalı programlama (Object Oriented Programing) çalışmalarında çoğu zaman farkında olmadan kullanılan bir yapıdır.
- Komut tasarım kalıbı, kullanıcı isteklerini gerçekleştiren kod yapısının sarmalanarak nesneler halinde saklanmasına dayanır.

Command(Komut) Tasarım Kalıbı



https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/komut-tasar%C4%B1m-kal%C4%B1b%C4%B1-(command-design-pattern)

Command(Komut) Tasarım Kalıbı

Komut (Command) : Gerçekleştirilecek işlem için bir ara yüz tanımlar.
Somut Komut (Concrete Command): Alıcı ve gerçekleştirilecek işlemler arasında bir bağ kurar, alıcıda karşılık düşen işlemleri çağırarak çalıştırma eylemini gerçekleştirir.
İstemci (Client): Komut nesnesini oluşturur ve metodun sonraki zamanlarda çağrılabilmesi için gerekli bilgiyi sağlar.
Çağırıcı (Invoker): Metodun ne zaman çağrılacağını belirtir.
Alıcı (Receiver): Kullanıcı isteklerini gerçekleştirecek asıl metod kodlarını içerir.

Command(Komut) Tasarım Kalıbı

Command Interface

```
public interface Command {
    public void execute();
}
```

LightOnCommand Sınıfı

```
/**
* Somut Command sinifimiz
*/
public class LightOnCommand implements Command {
    Light light;

    public LightOnCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        light.on();
    }
}
```

https://www.codesenior.com/tutorial/Komut-Command-Tasarim-Deseni

Command(Komut) Tasarım Kalıbı

LightOfCommand Sınıfı

```
/**
* Somut Command sinifimiz
*/
public class LightOffCommand implements Command {
    Light light;

    public LightOffCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        light.off();
    }
}
```

SimpleRemoteControl Sınıfı

```
/**
* Invoker sinifimiz
*/
public class SimpleRemoteControl {
    Command slot;

    public SimpleRemoteControl() {}

    public void setCommand(Command command) {
        slot = command;
    }

    public void buttonWasPressed() {
        slot.execute();
    }
}
```

https://www.codesenior.com/tutorial/Komut-Command-Tasarim-Deseni

Command(Komut) Tasarım Kalıbı

Light Sınıfı

```
/**
* Receiver sınıfımız
*/
public class Light {

    public Light() {
    }

    public void on() {
        System.out.println('Light is on');
    }

    public void off() {
        System.out.println('Light is off');
    }
}
```

Test Sınıfı

```
/**
* Client sınıfımız
*/
public class RemoteControlTest {
    public static void main(String[] args) {
        SimpleRemoteControl remote = new SimpleRemoteControl();
        Light light = new Light();
        LightOnCommand lightOn = new LightOnCommand(light);
        remote.setCommand(lightOn);
        remote.buttonWasPressed();
        remote.buttonWasPressed();
    }
}
```

https://www.codesenior.com/tutorial/Komut-Command-Tasarim-Deseni

Iterator(Tekrarlayıcı) Tasarım Kalıbı

Tekrarlayıcı tasarım kalıbı; birleşik bir nesnenin bileşenlerine, nesnenin esas ifadesinin gösterilimini açığa çıkarmadan sırayla erişebilmeyi sağlar.

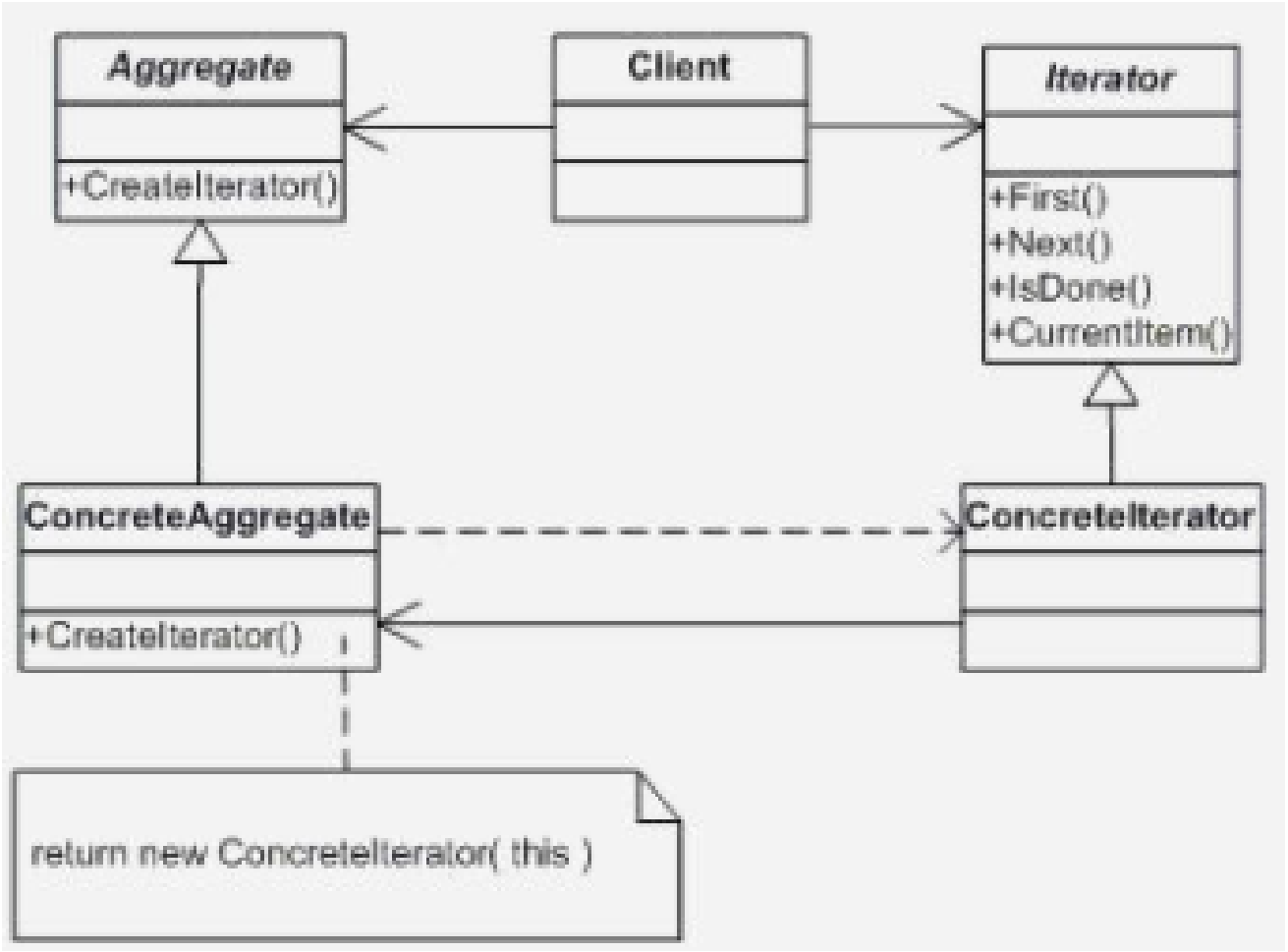
Kullanılan durumlar:

Tekrarlayıcı tasarım kalıbı, bir listenin yapısının ve çalışma tarzının uygulamanın diğer kısımları ile olan bağlantılarını en aza indirmek için; listede yer alan nesnelerin, sırasıyla uygulamadan soyutlanması amacıyla kullanılır.

Iterator(Tekrarlayıcı) Tasarım Kalıbı

- **Tekrarlayıcı (Iterator):** Öğelere ulaşmak ve öğeleri çaprazlamak için bir arabirim tanımlar.
- **Somut Tekrarlayıcı (Concrete Iterator):** Tekrarlayıcı arayüzünü sağlar.
- **Birleşik (Aggregate):** Tekrarlayıcı nesnelerin yaratılması için bir arabirim oluşturur.
- **Somut Birleşik (Concrete Aggregate):** Uygun somut tekrarlayıcı aşamasına dönmek için tekrarlayıcı arabiriminin oluşturulmasını sağlar.

Iterator(Tekrarlayıcı) Tasarım Kalıbı

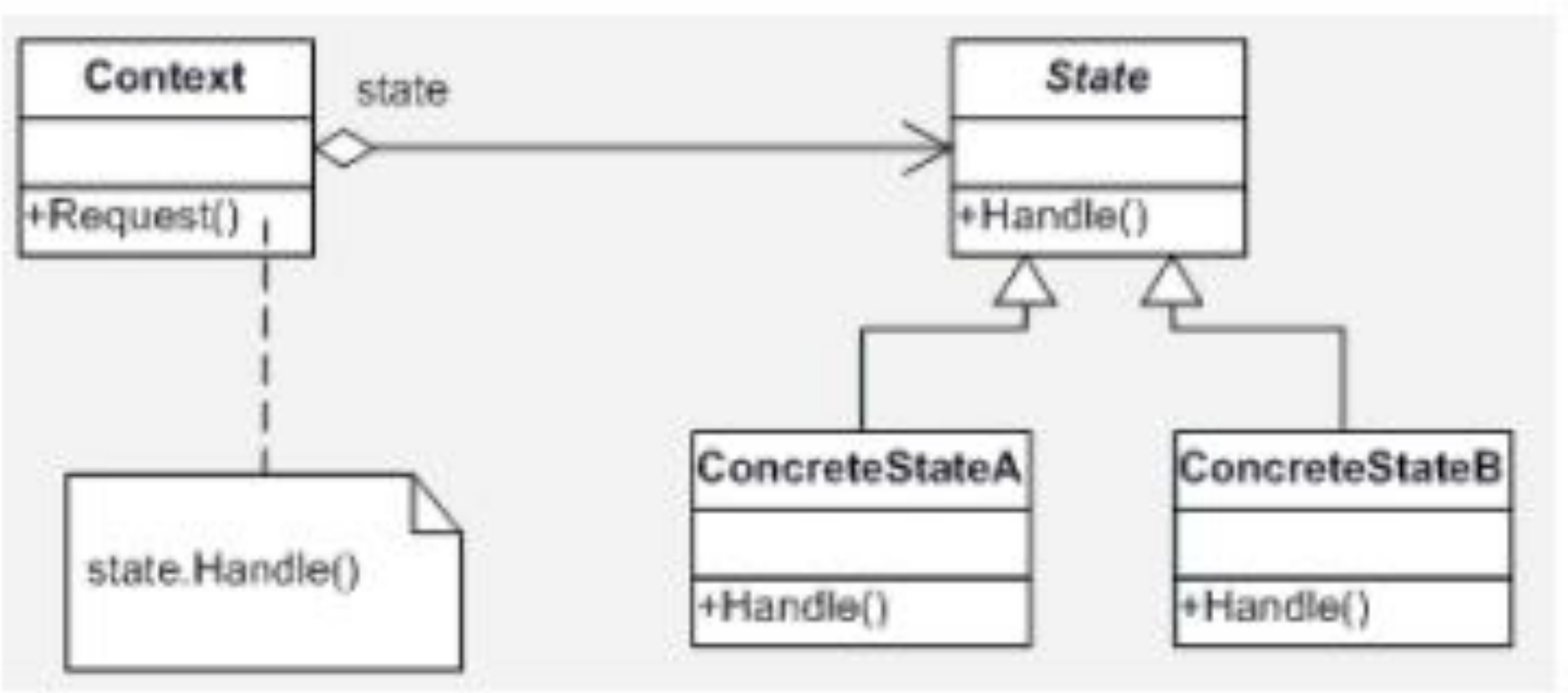


State(Durum) Tasarım Kalıbı

- Durum tasarım kalıbı davranışsal tasarım kalıplarının alt dallarından biri olarak kabul edilir. Bir nesnenin durumu değiştiğinde, davranışı da değişirse, diğer bir ifadeyle nesneler farklı durumlarda farklı davranışlar gösterirse, durum tasarım kalıbı kullanılabilir.
- Kullanıcı açısından, nesne sınıfını değiştiriyormuş izlenimi verir. Uygulamanın gerektirdiği doğrultuda yeni davranışlar eklenip çıkarılmasına olanak sağlar.
- **Durum (State)** tasarım kalıbının kullanılması, nesnelerin durumlarına bağlı değişen davranışlarının karmaşık koşul ve kontrol **(if/else, switch)** ifadeleriyle yönetilmesini önler.

https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/durum-tasar%C4%B1m-kal%C4%B1b%C4%B1-(state-design-pattern)

State(Durum) Tasarım Kalıbı



https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/durum-tasar%C4%B1m-kal%C4%B1b%C4%B1-(state-design-pattern)

State(Durum) Tasarım Kalıbı

Bağlam (Context)

- İstemcilerle ilişkili **Arayüzü (Interface)** tanımlar.
- Anlık durumu tanımlayan **Somut Durum (ConcreteState)** alt sınıfının nesnesini korur.

Durum (State)

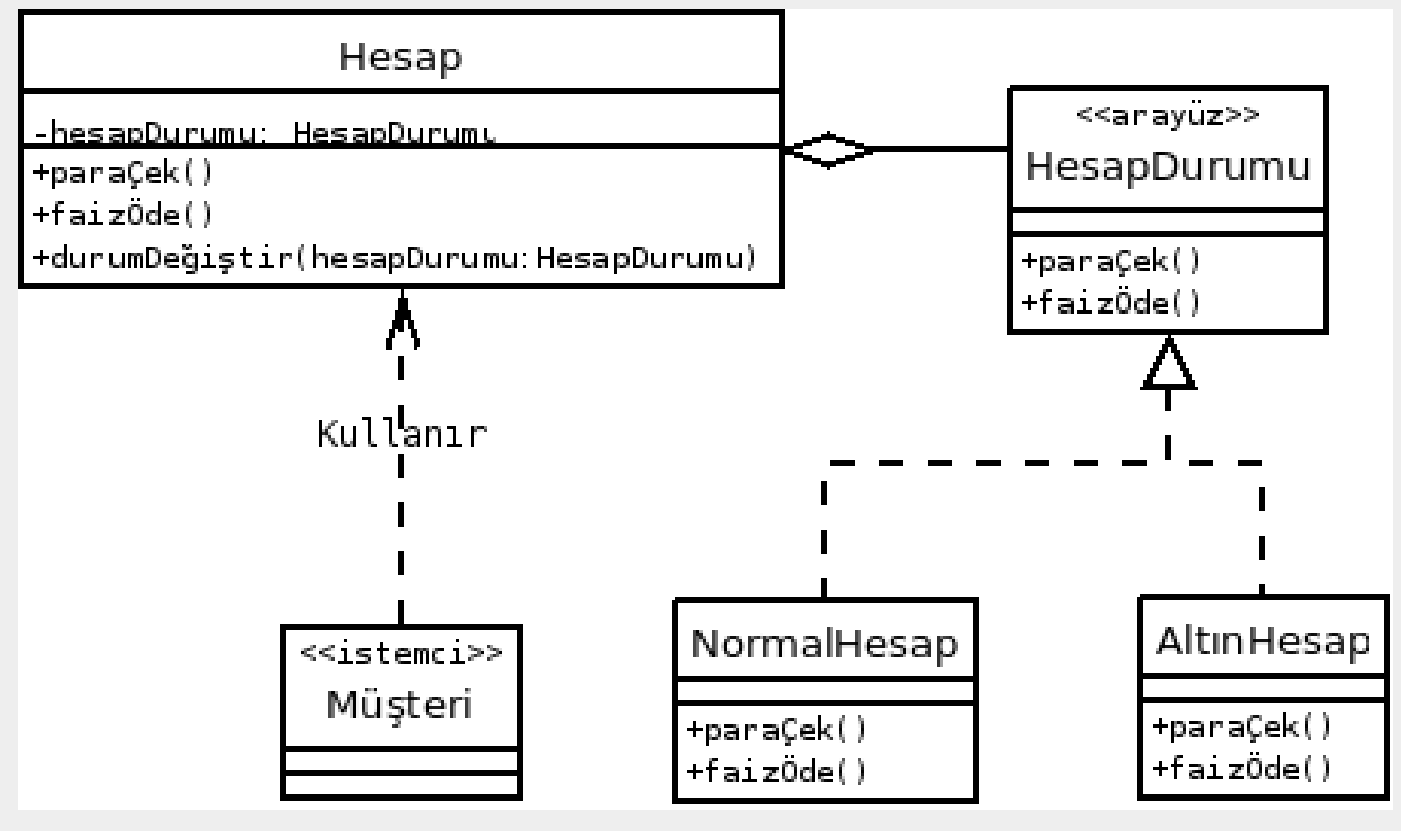
- **Bağlam (Context)** ın özel durumuyla ilgili bir davranışı, kısa ve öz biçimde açıklamak için arayüz tanımlar.

Somut Durum (Concrete State)

- Her alt sınıf **Bağlam (Context)** sınıfının durumuyla ilişkili bir davranışı uygular.

https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/durum-tasar%C4%B1m-kal%C4%B1b%C4%B1-(state-design-pattern)

State(Durum) Tasarım Kalıbı



http://www.tasarimdesenleri.com/jsp/tasdesincele/state.jsp

State(Durum) Tasarım Kalıbı

HesapDurumu.java

```
public interface HesapDurumu {

  void paraÇek();

  void faizÖde();
}
```

NormalHesap.java

```
public class NormalHesap implements HesapDurumu {

  @Override
  public void faizÖde() {
    System.out.println( "Normal hesaptan faiz ödendi." );
  }

  @Override
  public void paraÇek() {
    System.out.println( "Normal hesaptan para çekildi." );
  }
}
```

http://www.tasarimdesenleri.com/jsp/tasdesincele/state.jsp

State(Durum) Tasarım Kalıbı

AltınHesap.java

```
public class AltınHesap implements HesapDurumu {

  @Override
  public void faizÖde() {
    System.out.println( "Altın hesaptan faiz ödendi." );
  }

  @Override
  public void paraÇek() {
    System.out.println( "Altın hesaptan para çekildi." );
  }
}
```

Hesap.java

```
public class Hesap {

  private HesapDurumu  hesapDurumu;

  public Hesap() {
    hesapDurumu = new NormalHesap();
  }

  public void faizÖde() {
    hesapDurumu.faizÖde();
  }

  public void paraÇek() {
    hesapDurumu.paraÇek();
  }

  public void durumDeğiştir( final HesapDurumu hesapDurumu ) {
    this.hesapDurumu = hesapDurumu;
  }
}
```

http://www.tasarimdesenleri.com/jsp/tasdesincele/state.jsp

State(Durum) Tasarım Kalıbı

Musteri.java

```
public class Müşteri {

  public static void main( final String[] args ) {
    // Yaratılan hesap varsayılan olarak 'normal' durumda.
    final Hesap hesap = new Hesap();
    hesap.faizÖde();
    hesap.paraÇek();

    // Çalışma zamanında hesap durum değiştiriliyor.
    // Müşteri altın hesaba geçiriliyor.
    hesap.durumDeğiştir( new AltınHesap() );

    System.out.println( "Müşteri altın hesaba geçiriliyor." );

    // Durum değiştiği için davranış da değişecek.
    hesap.faizÖde();
    hesap.paraÇek();
  }
}
```

Ekran Çıktısı

```
Normal hesaptan faiz ödendi.
Normal hesaptan para çekildi.
Müşteri altın hesaba geçiriliyor.
Altın hesaptan faiz ödendi.
Altın hesaptan para çekildi.
```

http://www.tasarimdesenleri.com/jsp/tasdesincele/state.jsp

Kaynaklar

- https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/davran%C4%B1%C5%9Fsal-kal%C4%B1plar-(behavioral-patterns)
- https://medium.com/gokhanyavas/behavioral-patterns-davran%C4%B1%C5%9Fsal-desenler-11eec419589b
- http://cagataykiziltan.net/tr/tasarim-kaliplari-design-patterns/2-davranissal-tasarim-kaliplari/strategy-pattern/
- https://metinalniacik.medium.com/strategy-design-pattern-strateji-tasar%C4%B1m-%C3%B6r%C3%BCnt%C3%BCs%C3%BC-d7a43290969c
- http://www.tasarimdesenleri.com/jsp/tasdesincele/observer.jsp
- http://cagataykiziltan.net/tr/tasarim-kaliplari-design-patterns/2-davranissal-tasarim-kaliplari/observer-tasarim-kalibi/
- https://www.codesenior.com/tutorial/Komut-Command-Tasarim-Deseni
- https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/komut-tasar%C4%B1m-kal%C4%B1b%C4%B1-(command-design-pattern)
- https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/tekrarlay%C4%B1c%C4%B1-tasar%C4%B1m-kal%C4%B1b%C4%B1-(iterator-pattern)