Tasarım Örüntüleri
Strategy

Örüntülerin Temel Prensipleri

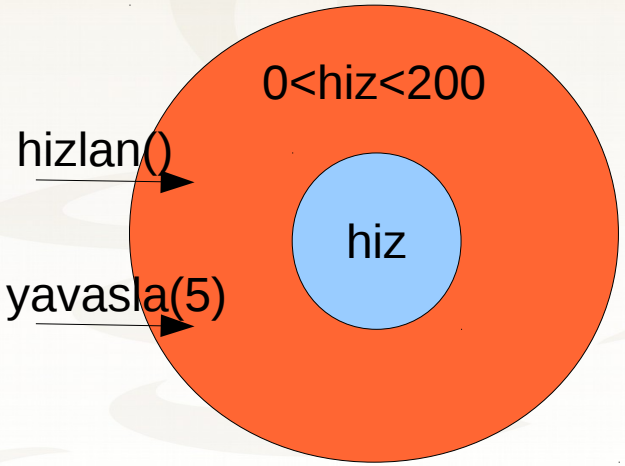
- GoF tasarım örüntülerinin altında yatan temel prensipler
  - Encapsulation
  - Composition
  - Abstract Data Types

Strategy Örüntüsü

- Strategy
  - Sık kullanılan bir örüntü
  - Davranışsal
  - Algoritma encapsulation
- Tasarım örüntüleri eğitiminde ilk anlattığımız örüntü
  - Encapsulation'ın sadece data encapsulation olmadığına iyi bir örnek

Data Encapsulation?

- Encapsulation
  - Verinin gizlenmesi



Data Encapsulation: Doğru Ama Eksik!

- Doğru ama **oldukça eksik** bir tanım
- Encapsulation “**veri gizleme**”den daha fazlası
  - Concrete sınıfların soyut sınıflarla gizlenmesi
  - Nesne yaratım sürecinin gizlenmesi
  - Metot implemantasyonunun gizlenmesi

Encapsulation için Daha Genel Bir Tanım

- Encapsulation **değişen herhangi bir şeyin değişmeyen/sabit başka bir şey arkasında gizlenmesi**, izole edilmesi işlemidir!

Değişen Ne ise Bul ve Encapsule Et!

- Tasarımsal olarak **neler değişebilir? neler sabit kalmalı** veya kalabilir?
- Sistemde **değişen olgu veya konsept** üzerine odaklanmalı
- Bu olgu veya konsept **değişmeyen/sabit başka bir yapı arkasında** gizlenmeli
- Yani **encapsule** edilmeli!

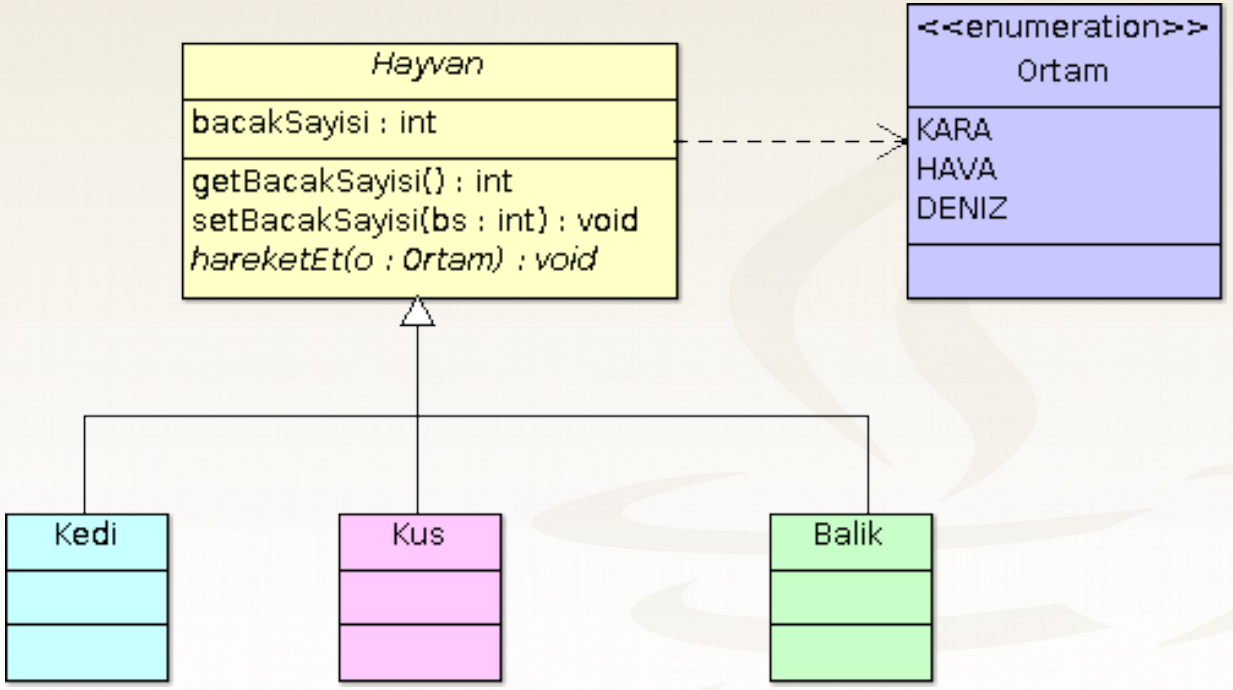
Örnek Problem

- Hayvanlar aleminde **değişik hayvan türleri** bulunmaktadır
- Her bir hayvan türünün **farklı sayıda bacakları** bulunur
- Her bir hayvan türü **farklı hareket şekilleri**ne sahiptir
- Hayvanlar **farklı ortamlarda farklı hareket şekilleri**ne sahiptir

Ortaklık/Değişkenlik

Ortak	Değişken
Hayvan	Kedi, Kuş, Balık,Uçan Balık,...
Back Sayısı	0,2,4,...
Hareket Şekli	Sürünmek, Yürümek, Yüzmek, Uçmak,...
Ortam	Kara, Hava, Deniz,...

İlk Tasarım


Hayvan
bacakSayisi : int
getBacakSayisi() : int
setBacakSayisi(bs : int) : void
hareketEt(o : Ortam) : void
<<enumeration>>
Ortam
KARA
HAVA
DENIZ
Kedi
Kus
Balik

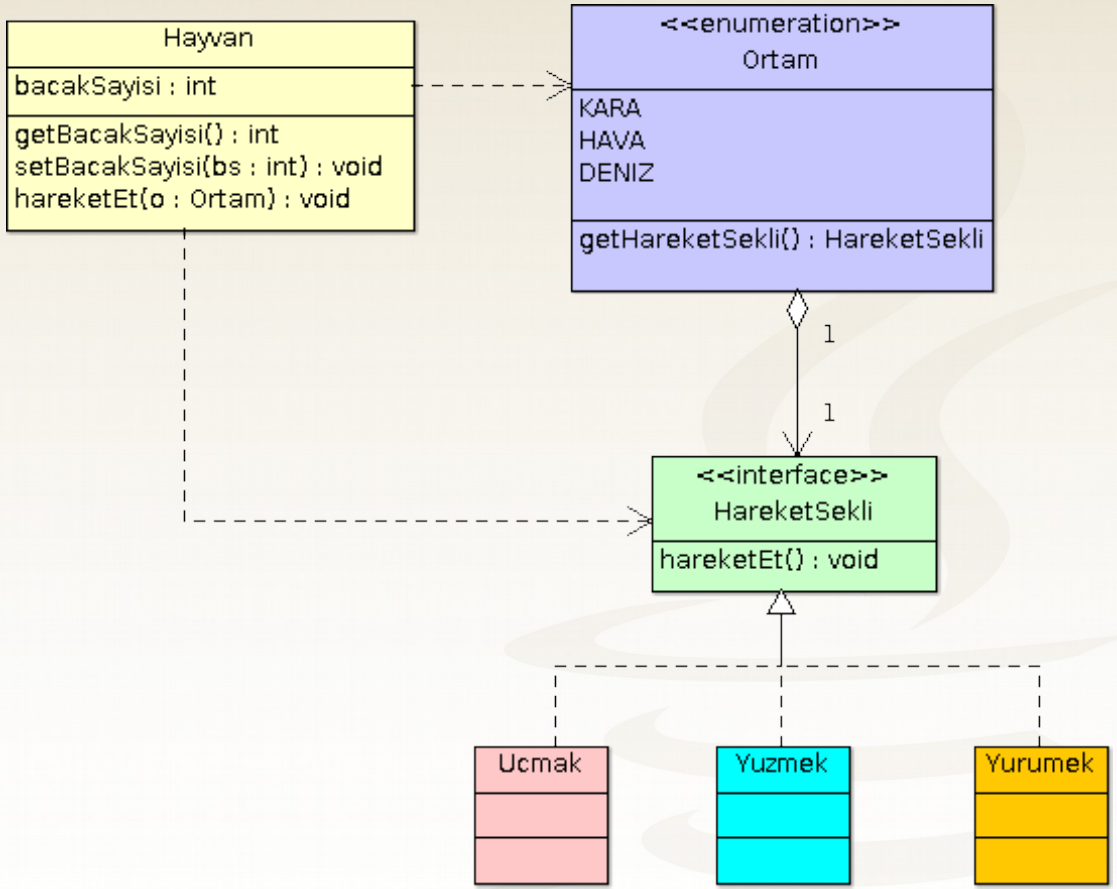
İlk Tasarım ve Gerçekleştirimi

```
public class Balik extends Hayvan {
     @Override
     public void hareketEt(Ortam o) {
          switch (o) {
            case KARA:
               System.out.println("yürüyor");
               break;
            case DENIZ:
               System.out.println("yüzüyor");
               break;
            case HAVA:
               System.out.println("uçuyor");
               break;
          }
     }
}
```

Bir Problem İşareti: Switch ve If-Else Blokları

- Çoğunlukla bir “**switch**” ifadesi veya “**if-else**” blokları, **polymorphic bir davranış ihtiyacının göz ardı edildiğini ve sorumlulukların yanlış dağıtıldığını** gösterir
- Böyle bir durumda yeniden **soyutlama yapılıp, sorumlulukların diğer nesnelere dağıtılması** düşünülebilir

Davranışın Encapsule Edilmesi



Davranışın Encapsule Edilmesi

```
interface HareketSekli {
     void hareketEt();
}
```

```
public class Yuzmek implements HareketSekli {
     public void hareketEt() {
          System.out.println("yüzüyor");
     }
}
```

```
public class Yurumek implements HareketSekli {
     public void hareketEt() {
          System.out.println("yürüyor");
     }
}
```

```
public class Ucmak implements HareketSekli {
     public void hareketEt() {
          System.out.println("uçuyor");
     }
}
```

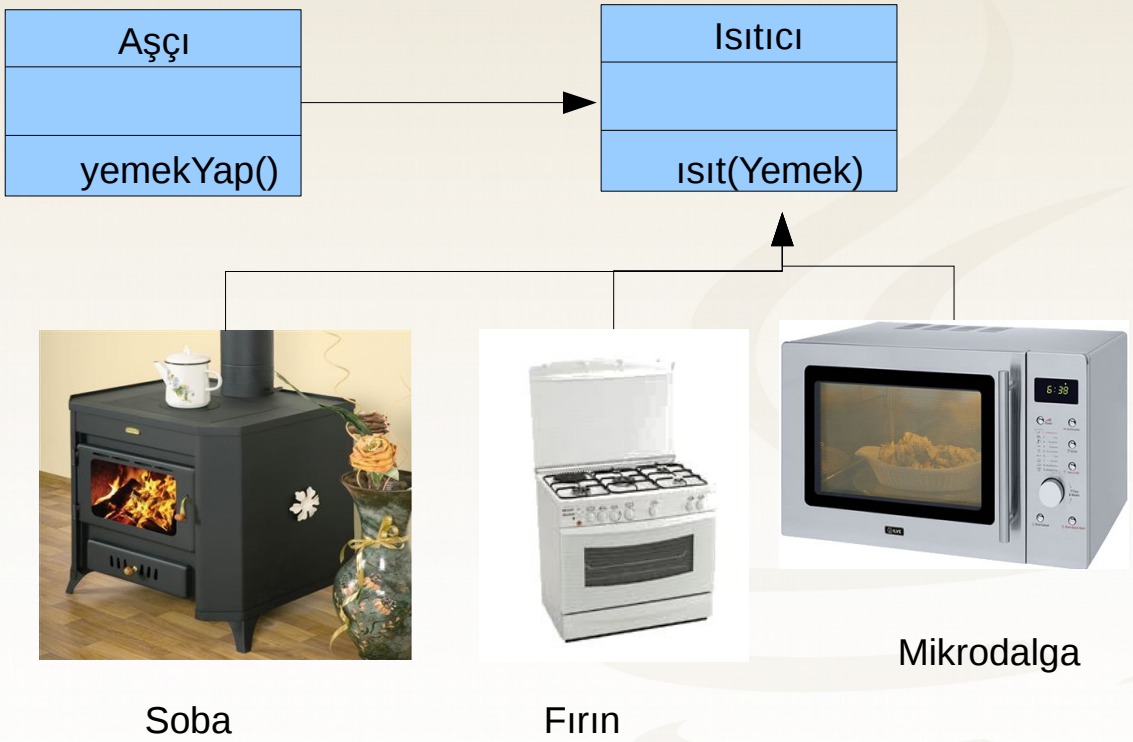
Davranışın Encapsule Edilmesi

```
public enum Ortam {
     KARA(new Yurumek()),
     HAVA(new Ucmak()),
     DENIZ(new Yuzmek());

     private HareketSekli hareketSekli;

     private Ortam(HareketSekli hs) {
          this.hareketSekli = hs;
     }

     public HareketSekli getHareketSekli() {
          return hareketSekli;
     }
}
```

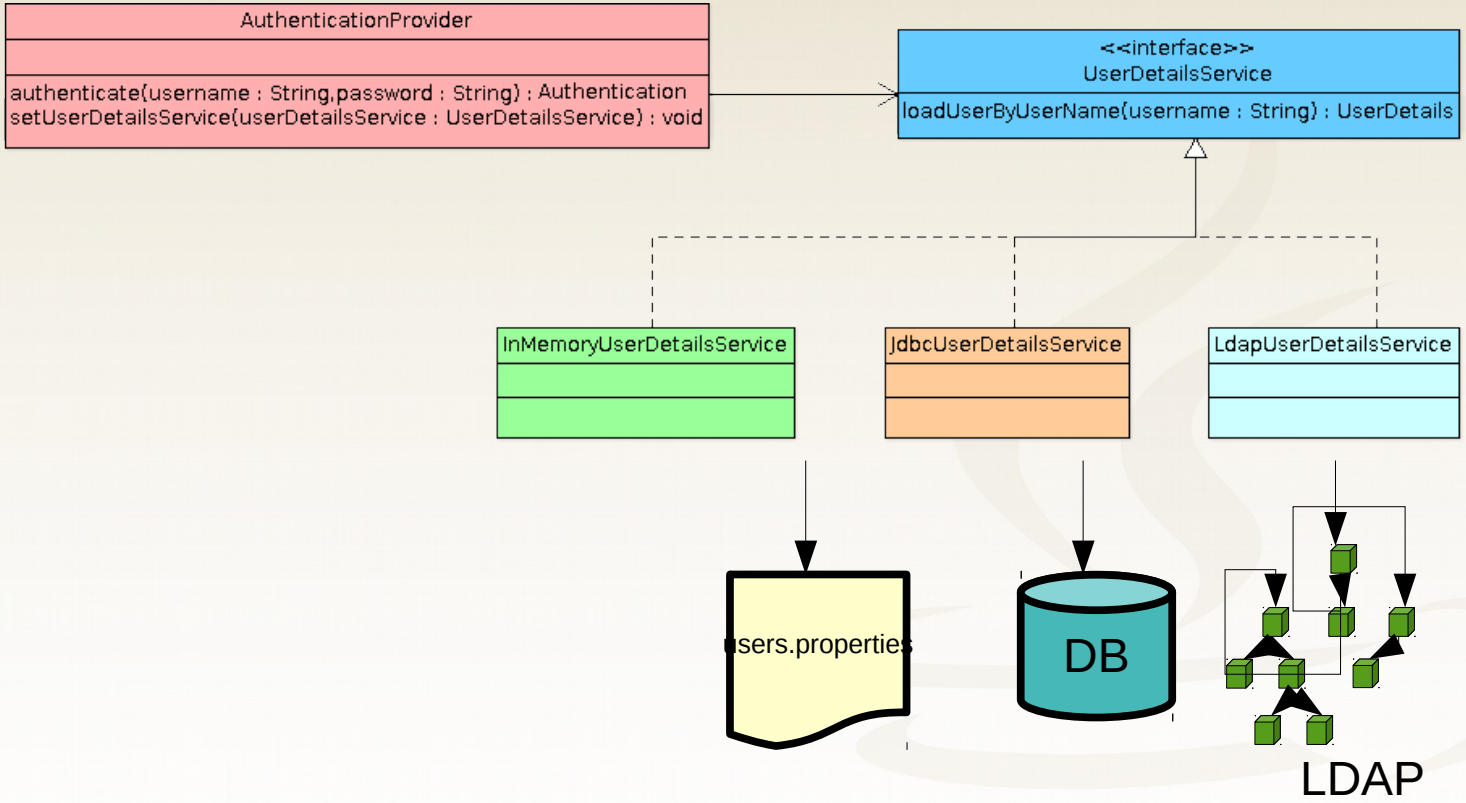
Davranışın Encapsule Edilmesi

```
public class Hayvan {
     private int bacakSayisi;

     public int getBacakSayisi() {
          return bacakSayisi;
     }

     public void setBacakSayisi(int bacakSayisi) {
          this.bacakSayisi = bacakSayisi;
     }

     public void hareketEt(Ortam o) {
       HareketSekli hareketSekli = o.getHareketSekli();
       hareketSekli.hareketEt();
     }
}
```

Strategy – Algoritmanın Encapsule Edilmesi



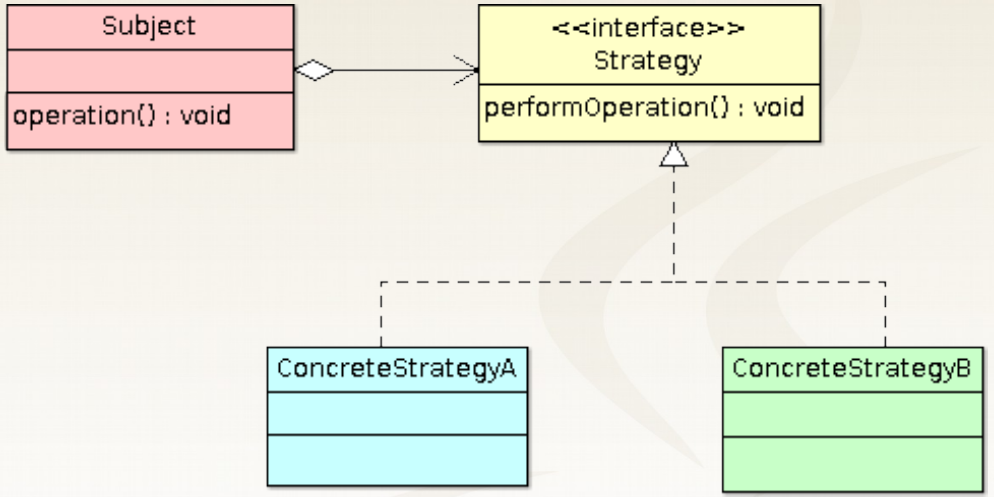
Strategy – Algoritmanın Encapsule Edilmesi



Strategy – Algoritmanın Encapsule Edilmesi

```
public Authentication authenticate(String username, String password) {

  UserDetails user = userDetailsService.loadUserByUsername(username);

  if (user == null || !      user.getPassword().equals(password))
    throw new AuthenticationException();

  return new Authentication(user);
}
```

Strategy Sınıf Diagramı



Strategy Örüntüsünün Faydaları

- Şartlı ifadeleri azaltması veya ortadan kaldırması algoritmayı veya **iş mantığı**nı daha **kolay anlaşılır** kılabilir
- Çalışma zamanında **algoritmanın dinamik olarak değiştirilmesi** sağlanabilir

İletişim

JAVA Eğitimleri

Kurumsal Java Eğitimleri

www.java-egitimleri.com

info@java-egitimleri.com

@javaegitimleri

youtube.com/c/
KurumsalJavaEğitimleri