JAVA Eğitimleri

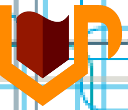
Java ile Tasarım Prensipleri ve Tasarım Örüntüleri

Harezmi Bilişim Çözümleri

JAVA Eğitimleri

İyi Tasarımın Amacı Nedir?

Olası Değişiklikleri Kolay Bir Şekilde Ele Alabilmek...



Peki Neler Değişir?

“Gereksinimler”

Gereksinimlerin Değişmesi

- Gereksinimler **tam değil**dir
- Genellikle **yanlış ve yanıltıcı**dırlar
- **Sürekli** değişirler, sistemle ilgili **yeni olasılıklar** göz önüne gelir
- Geliştiricilerin **sistemi kavrayışı** zaman içerisinde gelişir
- Yazılım sisteminin geliştirildiği **ortam sürekli değişir**

Sonuç: Gereksinimlerin değişmesinden **şikayet etmek anlamsız**dır!

Yapılması gereken:

Değişiklikleri daha efektif biçimde ele alabilen **iyi bir tasarıma** sahip olmalıyız!

İyi Tasarımı Kötü Tasarımdan Nasıl Ayırt Edebiliriz?

Kötü Tasarımın Belirtileri

- 4 temel belirtisi vardır
  - Rigidity (Esnemezlik)
  - Fragility (Kırılganlık)
  - Immobility (Taşınamamamazlık)
  - Viscosity (Akışkanlık)
- Bu belirtiler birbirleri ile bağlantılıdır ve **kötü bir mimarinin işaretleri**dir

Nesne Yönelimli Analiz ve Tasarım Nasıl Olmalı?

Öncelikle probleme bakış açımızı değiştirmeliyiz!

Geleneksel ve Modernist Yaklaşımlar

- Geleneksel yaklaşımda **nesne = veri + metot**
- Modernist yaklaşımda nesne
  - sorumlulukları olan
  - belirli bir davranış sergileyen bir olgudur
- Modernist yaklaşım **nesnenin içerisinde ne olduğu** ile ilgilenmez
- Başka bir deyişle olgulara "**ne yapmaları** gerektiği" söylenmelidir, "**nasıl yapmaları**" değil

Problemden çözüme **nasıl** gidilir?

Ortaklık/Değişkenlik Analizi

- Problem domain'i genelinde **ortak şeyler** (commonality analiz) ve **değişenler** (variability analiz) tespit edilir
- Ortaklık analizi zaman içerisinde **çok sık değişmeyecek kısımları** arar
- Değişkenlik analizi ise **sıklıkla değişecek yapıları** arar
- Ortak kavramlar **soyut sınıflar**la ifade edilecektir
- **Concrete sınıflar** ise varyasyonlardır

Ortaklık/Değişkenlik Analizi

- Mimarisel perspektiften bakılırsa ortaklık analizi mimariye **uzun ömürlülük** katar
- Değişkenlik analizi ise **kullanım kolaylığı** sağlar

Altın Değerinde İki Tasarım Kuralı...

Kural 1: Değişen ne ise bul ve encapsule et

JAVA Eğitimleri

Kural 2: Composition'ı inheritance'a tercih et

Geliştirme Sırasında İzlenecek İki Temel Tasarım Prensibi...

Açıklık Kapalılık Prensibi
Open Closed Principle (OCP)

- Bir modül **genişlemeye açık**, **değişikliğe kapalı** olmalıdır
- Modülleri **genişletilebilir (extend)** biçimde yazmalıyız
- Modüller ileride **değişiklik gerektirmemeli**dir
- **Soyutlama** OCP'de anahtar kelimedir

Tersine Bağımlılık Prensibi
Dependency Inversion Principle (DIP)

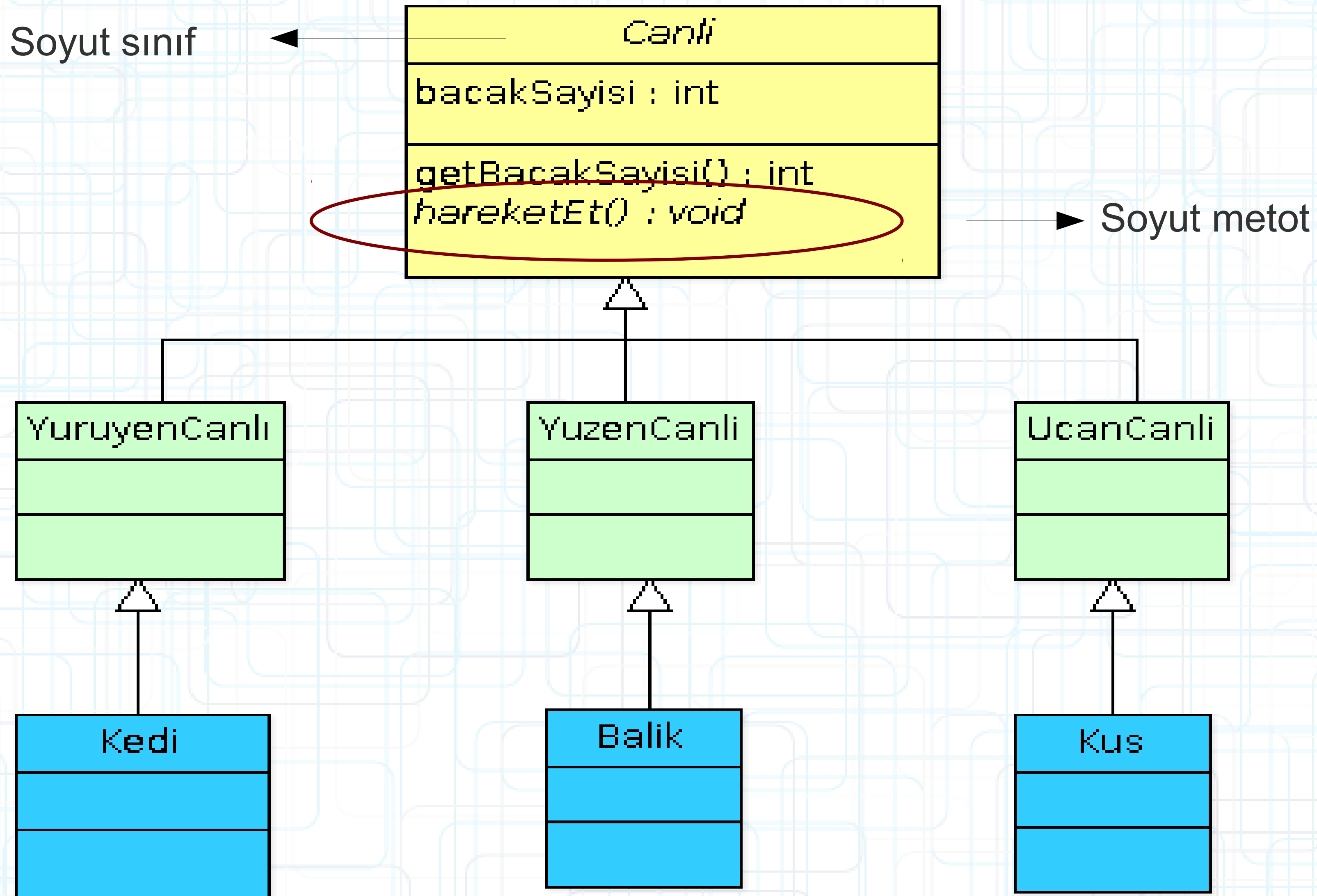
- Sadece **arayüz veya soyut sınıflara bağımlı olunmalı**dır
- Concrete sınıflara bağımlılık olmamalıdır
- COM, CORBA, EJB gibi bileşen teknolojilerinin dayandığı temel prensiptir
- Tasarımdaki **bütün bağımlılıklar soyut olgulara doğru** olmalıdır
- **Soyutlama noktaları** tasarımın genişletilebileceği noktaları oluşturur

Örnek Problem: Simülasyon Programı Faz 1

Doğadaki **canlılar** hareket kabiliyetlerini sahip oldukları **bacakları** vasıtası ile sağlamaktadır. Her türün **farklı sayıda** bacakları olabilir. Canlılar karada **yürüyebilir**, denizde **yüzebilir**, havada ise **uçabilir**ler.

Farklı canlı türlerinin **hareket şekiller**ini modelleyen bir **simülasyon programı** yazılması istenmektedir. Simülasyon programında farklı canlı türlerini temsil etmek için **kedi**, **kuş** ve **balık** türleri kullanılabilir.

Canlı Hiyerarşisi



JAVA Eğitimleri

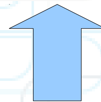
Canli.java

```
public abstract class Canli {

    private int bacakSayisi;

    public int getBacakSayisi() {
        return bacakSayisi;
    }

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

    public abstract void hareketEt();

}
```

JAVA Eğitimleri

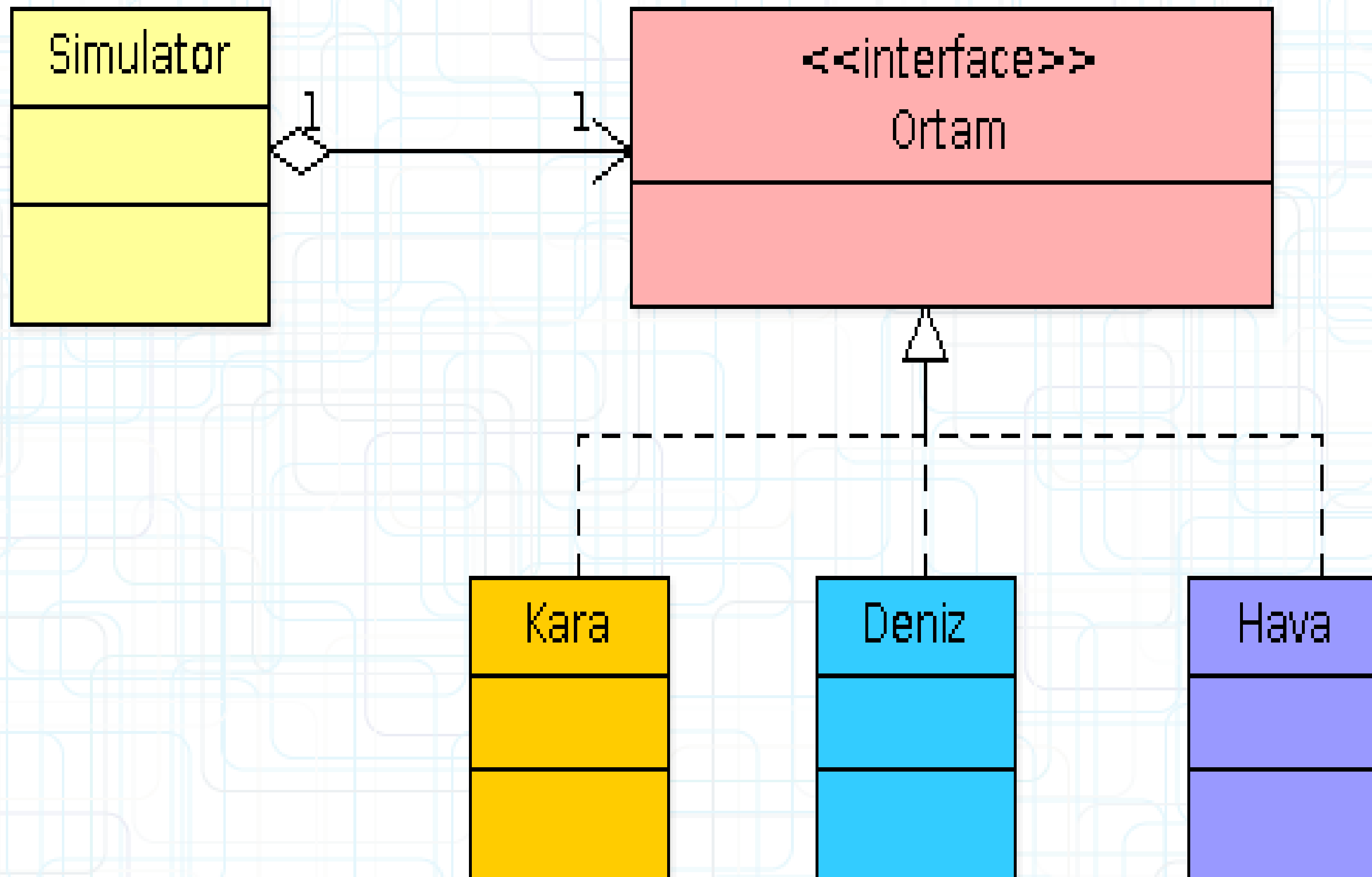
Yüzen Canlılar

```java
public class YuzenCanli extends Canli {

    @Override
    public void hareketEt() {
        System.out.println("yüzüyor...");
    }

}
```



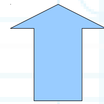
```java
public class Balik extends YuzenCanli {
    public Balik() {
        setBacakSayisi(0);
    }
}
```

Simülator ve Ortam



Ortam Arayüzü ve Alt Sınıfları

```
public interface Ortam {

}
```



```
public class Deniz implements Ortam {

}
```

Simulator.java

```
public class Simulator {
    private Ortam ortam;

    public Ortam getOrtam() {
        return ortam;
    }

    public void setOrtam(Ortam ortam) {
        this.ortam = ortam;
    }

    public void hareketEttir(Canli...canlilar) {
        for(Canli c:canlilar) {
            c.hareketEt();
        }
    }
}
```

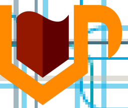
Main.java

```
public class Main {

public static void main(String[] args) {
   Simulator simulator = new Simulator();

   simulator.hareketEttir(new Kedi(),new
Kus(),new Balik());
   }

}
```

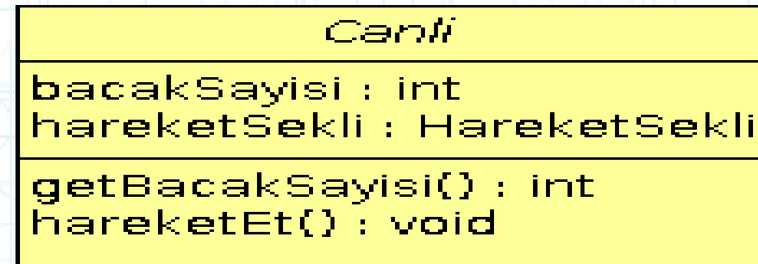
```
yürüyor...
uçuyor...
yüzüyor...
```

Örnek Problem: Simülasyon Programı Faz 2

Bazı canlılar her ortamda tek bir hareket şekline sahip iken, diğer bazıları ise **farklı ortamlarda farklı hareket şekiller**ine sahip olabilirler. Örneğin, kuşlar karada yürüme, havada ise uçma kabiliyetine sahiptirler. Farklı bir balık türü ise denizde yüzebilirken, belirli süre deniz yüzeyinin üzerinden havalanarak uçabilmektedir. Simülasyon programı, canlı türün **hareket şeklinin ortama göre değişiklik gösterme**sini de desteklemelidir.



Uçan Balık



YuruyenCanli

YuzenCanli

UcanCanli

Kedi

Balik

Kus

UcanBalik

JAVA Eğitimleri

UcanBalik.java

```
public class UcanBalik extends Balik {
    private boolean uc = false;

    public boolean isUc() {
        return uc;
    }

    public void setUc(boolean uc) {
        this.uc = uc;
    }

    @Override
    public void hareketEt() {
        if(uc) {
            System.out.println("uçuyor");
        } else {
            super.hareketEt();
        }
    }
}
```

If-else ifadesi bir algoritmik Varyasyon işaretçisidir

Simulator.java

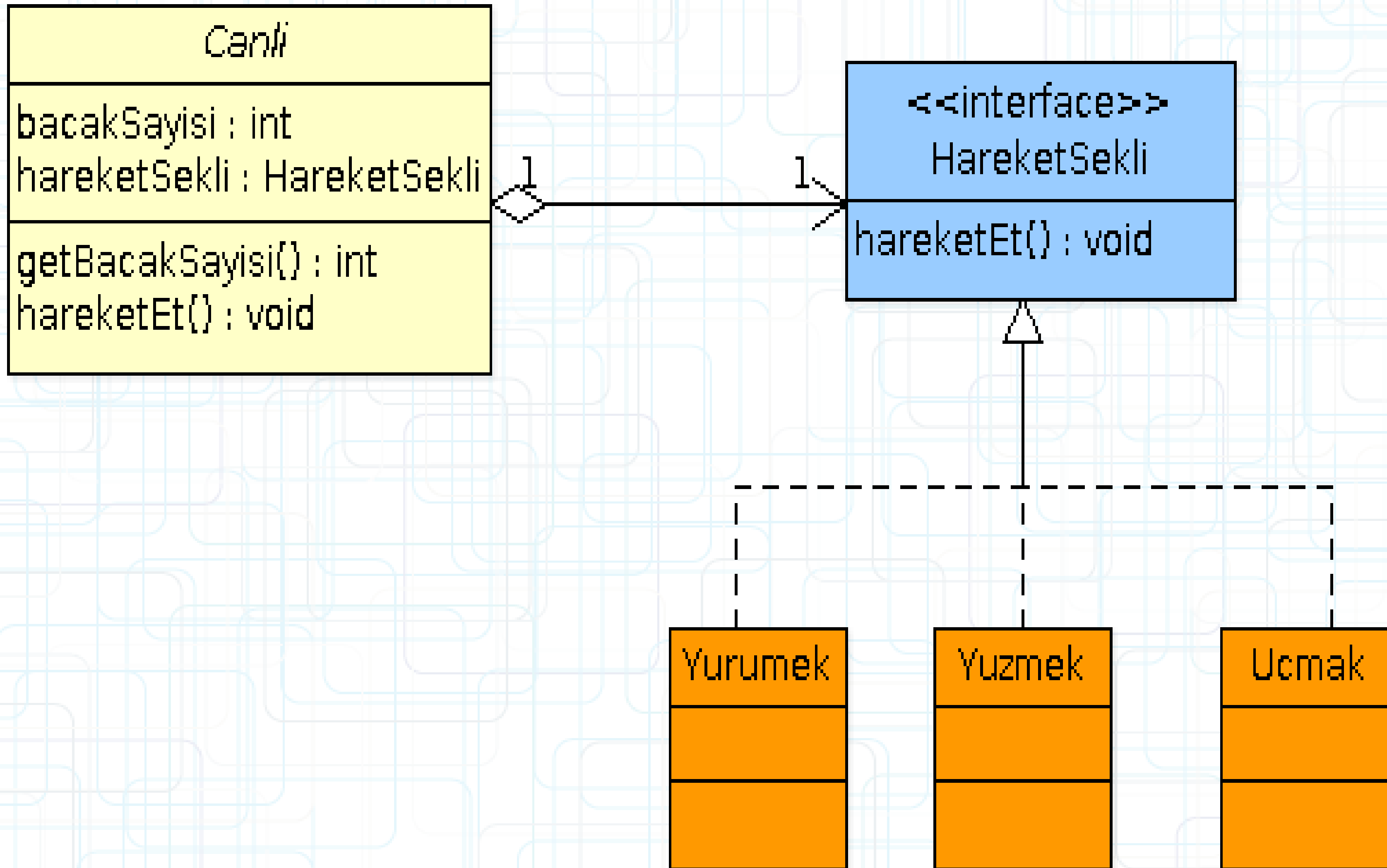
```
public class Simulator {
    private Ortam ortam;

    public Ortam getOrtam() {
        return ortam;
    }

    public void setOrtam(Ortam ortam) {
        this.ortam = ortam;
    }

    public void hareketEttir(Canli...canlilar) {
        for(Canli c:canlilar) {
            if(ortam instanceof Hava && c instanceof
UcanBalik) {
                ((UcanBalik)c).setUc(true);
            }
            c.hareketEt();
        }
    }
}
```

OCP, DIP prensipleri ihlal ediliyor

Kural 1: Değişen ne ise bul ve encapsule et

...farklı ortamlarda farklı hareket şekillerine sahip olabilirler. ...canlı türün **hareket şeklinin ortama göre değişiklik göstermesi**ni de desteklemelidir.

Farklı Hareket Şekilleri



JAVA Eğitimleri

Canli.java

```
public abstract class Canli {

...

private HareketSekli hareketSekli;

    public void hareketEt() {
        hareketSekli.hareketEt();
    }

    public void setHareketSekli(HareketSekli
hareketSekli) {
        this.hareketSekli = hareketSekli;
    }
}
```

Davranışın Encapsule Edilmesi

HareketSekli.java

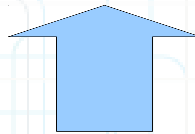
```
public interface HareketSekli {
   public void hareketEt();
}
```

```
public class Ucmak
implements HareketSekli {

   public void hareketEt()
System.out.println("uçuyor..
.");
   }

}
```

```
public class Yuzmek
implements HareketSekli {

   public void hareketEt() {
System.out.println("yüzüyor.
..");
   }

}
```

UcanBalik.java

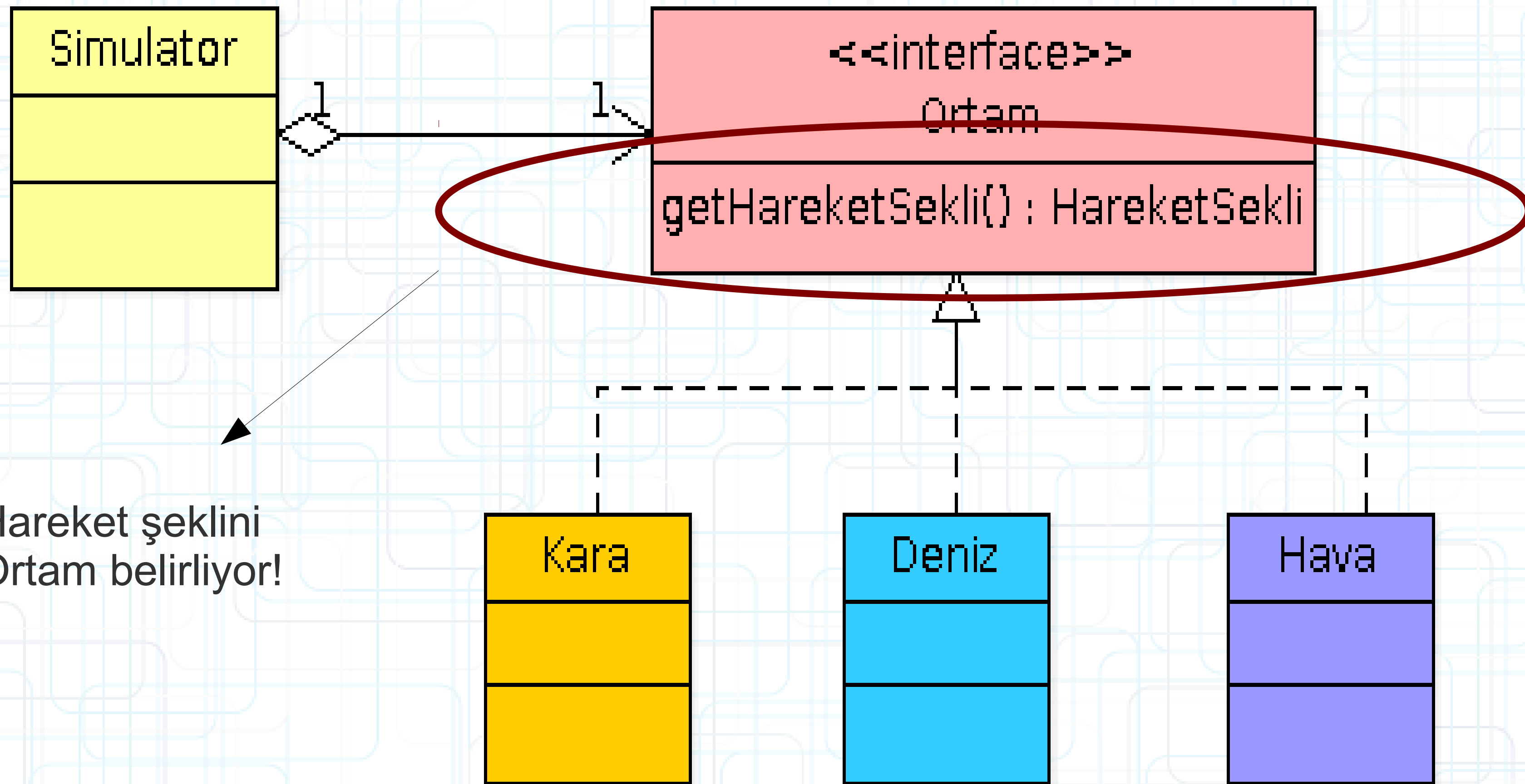
```
public class YuzenCanli extends Canli {
    public YuzenCanli() {
        setHareketSekli(new Yuzmek());
    }
}
```



```
public class UcanBalik extends Balik {
...
    @Override
    public void hareketEt() {
        if(uc) {
            new Ucmak().hareketEt();
        } else {
            super.hareketEt();
        }
    }
}
```

OCP ve DIP problemleri
Devam ediyor!

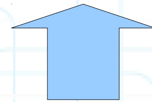
Hareketin Ortama Göre Değişmesi



Hareket şeklini
Ortam belirliyor!

Ortam.java

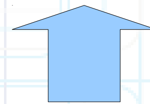
```java
public interface Ortam {
    public HareketSekli getHareketSekli();
}
```



```java
public class Deniz implements Ortam {

    public HareketSekli getHareketSekli() {
        return new Yuzmek();
    }

}
```

Canli.java

```
public abstract class Canli {
...
   private HareketSekli hareketSekli;

   public void hareketEt(Ortam ortam) {
      hareketSekli.hareketEt();
   }
}
```



Hareket şeklini ortama göre değiştirme imkanı sağlanıyor

```
public class UcanBalik extends Balik {

   @Override
   public void hareketEt(Ortam ortam) {
      ortam.getHareketSekli().hareketEt();
   }
}
```

JAVA Eğitimleri

Simulator.java

```
public class Simulator {
    private Ortam ortam;

    public Ortam getOrtam() {
        return ortam;
    }

    public void setOrtam(Ortam ortam) {
        this.ortam = ortam;
    }

    public void hareketEttir(Canli...canlilar) {
        for(Canli c:canlilar) {
            c.hareketEt(getOrtam());
        }
    }
}
```

O anki ortam, hareketEt metoduna input argüman olarak veriliyor

Main.java

```java
public class Main {

    public static void main(String[] args) {
        Simulator simulator = new Simulator();

        simulator.setOrtam(new Deniz());
        simulator.hareketEttir(new UcanBalik());

        simulator.setOrtam(new Hava());
        simulator.hareketEttir(new UcanBalik());
    }
}
```

```
yüzüyor...
uçuyor...
```

JAVA Eğitimleri

Nesne Yönelimli Yazılım Geliştirmede Tasarım Örüntülerinin Rolü Nedir?

Tasarım Örüntülerinin Faydaları

- Tasarım ve nesne yönelimli modelleme işlemine **üst perspektiften bakma**yı sağlar
- Bu sayede daha ilk aşamada **gereksiz detay ve ayrıntılar** içinde boğulmanın önüne geçilebilir
- Bu örüntüler zaman içerisinde **evrilmiş ve olgunlaşmış** çözümlerdir
- Bu nedenle üzerlerinde **değişiklik yapmak daha kolay ve hızlı**dır

Tasarım Örüntülerinin Faydaları

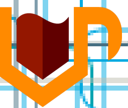
- Çözümlerin **yeniden kullanılmasını** sağlar
- Hazır çözümler probleme sıfırdan başlamayı, ve **olası hatalara düşmeyi önler**
- Diğerlerinin **deneyimlerinden faydalanma**yı sağlar
- Ekip içinde **ortak bir terminoloji** oluşmasını sağlar, ortak **bir bakış açısı** getirir

JAVA Eğitimleri

Örnek Bir Örüntü:Strategy

Örnek Problem: Robot davranışları

Robot davranışları ile ilgili bir **simülasyon** programı geliştirilecektir. Robotların davranışları **agresif**, **savunmacı** ve **sakin** olarak değişmektedir. Her bir davranış tipine göre robot **farklı farklı haraket** etmektedir. Robotların davranışları **dinamik olarak değişebilmekte**dir.



Örnek Bir Örüntü:Strategy

Robot

hareketEt() : void

1

1

<<interface>>
Davranis

hareketEt() : void

Agresif

Savunmaci

Sakin

JAVA Eğitimleri

Örnek Bir Örüntü:Strategy

Problem

- Kullanılacak **algoritma istemciye veya eldeki veriye göre değişiklik** gösterebilir
- İstemcinin **algoritmanın detayını** bilmesine gerek yoktur

Çözüm

- **Algoritmanın seçimi ile implementasyonu birbirinden ayrı** tutulur
- Algoritma seçimi **context**'e göre **dinamik** yapılabilir

Örnek Bir Örüntü:Strategy

Sonuçları

- **Switch ve şartlı ifadeler** ortadan kaldırılır
- Algoritma değişiklikleri için **alt sınıf oluşturmaya bir alternatif**tir
- Bütün **algoritmalar** aynı biçimde invoke edilmelidir
- **Strategy ile context arasında etkileşim** gerekebilir

Soru / Cevap...

İletişim

- Harezmi Bilişim Çözümleri Ltd.
- Kurumsal Java Eğitimleri
- http://www.harezmi.com.tr
- http://www.java-egitimleri.com
- info@java-egitimleri.com