

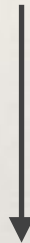
Nesneye Y6nelik Yazılım M6uhendislięi (376)

Yrd. Doę. Dr. Ahmet Arif AYDIN

Object-Oriented Design: Functional Decomposition

- ❖ Karmaşık bir problemi daha anlaşılır ve çözülebilir alt aşamalara ve parçalara ayrıştırıp çözme işlemidir (*process of taking a complex process and breaking it down into its smaller, simpler parts*)

Kütüphane Otomasyonu



Arama (Search)

Ekleme (Insert)

Güncelleme (Update)

Raporlar (Reports)

Ödünç Kitap

Admin (Yönetim)

Veritabanı

Object-Oriented Design: Functional Decomposition

❖ Problemler

1. Bütün detaylar ana program tarafından bilinmesi gerekir.

❖ ana program temelli tasarım yapılır

2. Değişim isteklerine uygun değildir

❖ İyi bir modüler bir yapı bulunmadığından küçük değişiklikler programın tamamına etkisi olabilir

❖ *Many bugs originate with changes to the code*

❖ Problemlerin ortaya çıkmasının başlıca nedeni

- abstraction (*soyutlama*)
- encapsulation (*kapsülleme*)
- information hiding (*bilgi gizleme*)
- polymorphism (*çok biçimlilik*)
- modularity (*modülerlik*)

kavramlarının zayıf olarak kullanılmasıdır!

Object-Oriented Paradigm: Abstraction

- ❖ Bir varlığın belirli amaçları gerçekleştirmek , bir görevi veya problemi çözmek için sağlamış olduğu tanımlamadır.
 - ➔ Assembly dili makine dili için bir *abstraction* dır.
 - ➔ Yüksek seviyeli programlama dilleri assembly dili için bir *abstraction* dır.
 - ➔ Bir sınıf içerisinde tanımlanan değişkenler methodlar
 - ➔ Bir sınıfa erişmek için oluşturulan nesneler

Object-Oriented Paradigm: Abstraction

```
import java.util.List;
import java.util.LinkedList;

public class Student{
    private String isim;
    public int yaş;

    public Student(String isim){
        this.isim = isim;
    }

    public void çıktıal() {
        System.out.print(isim + " says \"");
    }

    public String formatlıcıktı() {
        return String.format("%14s : %s",
                                getClass().getName(),
                                name);
    }
}
```

```
public static void main(String[] args) {
    Student öğrenci1= new Student("iu");
    öğrenci1.cıktıal
}
```


Access Specifiers (Bağlantı Tanımlayıcıları)

❖ **public**

- ➔ sınıfa erişim sağlayabilen nesnelerin kullanımına açık

❖ **private**

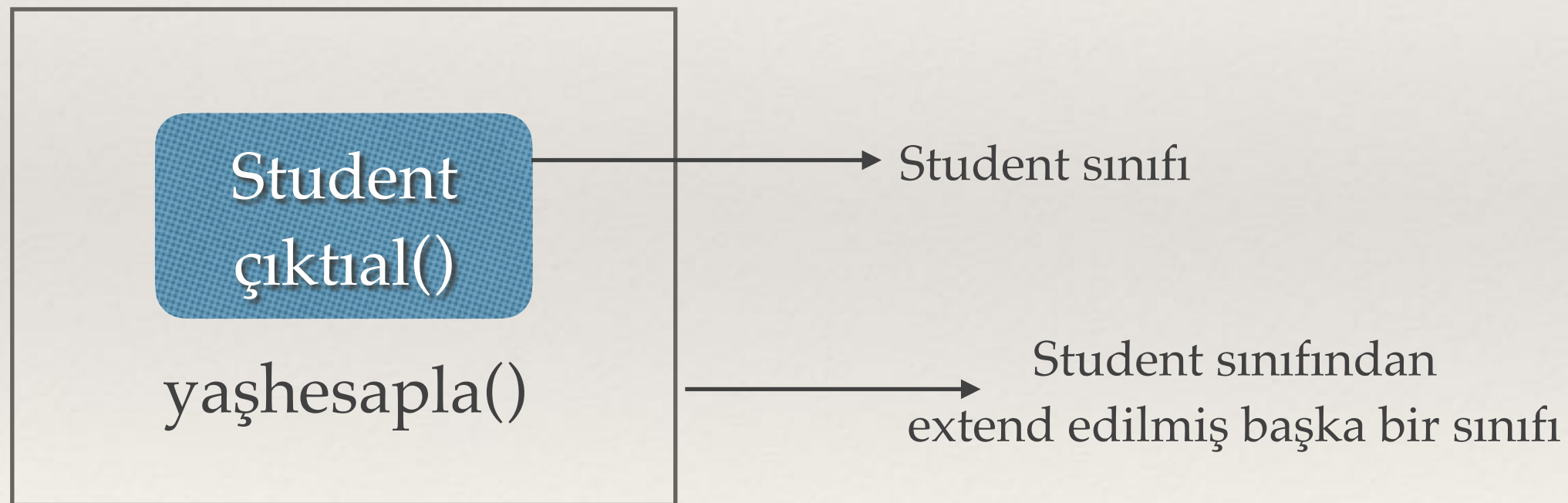
- ➔ sadece tanımlanan sınıfın içindeki methodlar erişebilir.
- ➔ dışarıya kapalı

❖ **protected**

- ➔ private gibi davranır
- ➔ yalnız kalıtım yoluyla erişilebilir

Object-Oriented Paradigm: Inheritance

- ❖ Bir sınıfın başka bir sınıftan özelliklerini ve metodlarını kalıtsal olarak devralmasıdır (*process of acquiring properties of another class*)
- ❖ **extends** kelimesi kullanılır.
- ❖ Kalıtım yoluyla devralınan özelliklere yenileri eklenip oluşturulan yeni sınıf genişletilebilir.



Object-Oriented Paradigm: Inheritance

```
class Hesaplama {
    int z;
    public void toplama(int x, int y) {
        z = x + y;
        System.out.println("Toplam:"+z);
    }
    public void cikarma(int x, int y) {
        z = x - y;
        System.out.println("Fark:"+z);
    }
}

public class BenimHesap extends Hesaplama
{
    public void carpma(int x, int y) {
        z = x * y;
        System.out.println("Çarpım:"+z);
    }
}

public static void main(String args[]) {
    int a = 20, b = 10;
    BenimHesap demo = new BenimHesap();
    demo.toplama(a, b);
    demo.cikarma(a, b);
    demo.carpma(a, b);
}
```


Object-Oriented Paradigm: Inheritance

```
import java.util.List;
import java.util.LinkedList;

public class Student{
    private String isim;
    public int yaş;

    public Student(String isim){
        this.isim = isim;
    }

    public void çıktıal() {
        System.out.print(isim + "
says \"");
    }

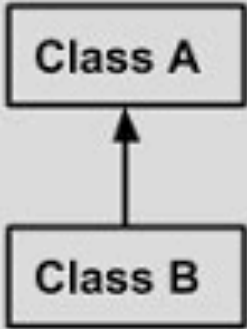
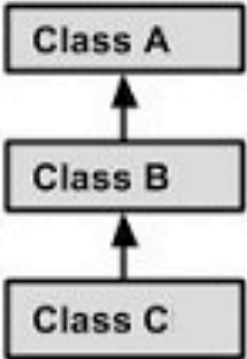
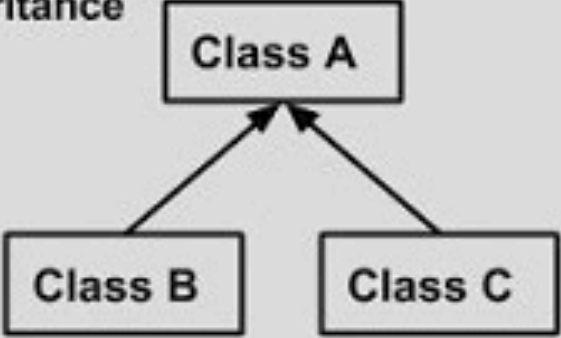
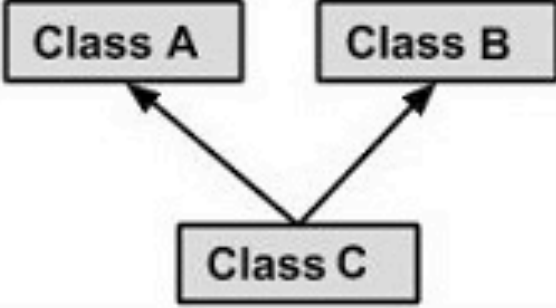
    public String formatlıcıktı() {
        return String.format(
            "%14s : %s",
            getClass().getName(), name);
    }
}
```

```
public class MastersStudent extends Student {

    public MastersStudent(String isim) {
        super(isim);
    }

    public void çıktıal() {
        super.çıktıal();
        System.out.println("*****");
    }
}
```

Object-Oriented Paradigm: Inheritance

Single Inheritance  <pre>graph BT; B[Class B] --> A[Class A]</pre>	<pre>public class A { } public class B extends A { }</pre>
Multi Level Inheritance  <pre>graph BT; C[Class C] --> B[Class B]; B --> A[Class A]</pre>	<pre>public class A {} public class B extends A {.....} public class C extends B {..... }</pre>
Hierarchical Inheritance  <pre>graph BT; B[Class B] --> A[Class A]; C[Class C] --> A</pre>	<pre>public class A {} public class B extends A {.....} public class C extends A {..... }</pre>
Multiple Inheritance  <pre>graph BT; C[Class C] --> A[Class A]; C --> B[Class B]</pre>	<pre>public class A {} public class B {.....} public class C extends A,B { } // Java does not support mutiple Inheritance</pre>

Object-Oriented Paradigm: Inheritance

