انجمن جاواکاپ تقدیم می‌کند

دوره برنامه‌نویسی جاوا

وراثت
Inheritance

صادق علی‌اکبری

حقوق مؤلف

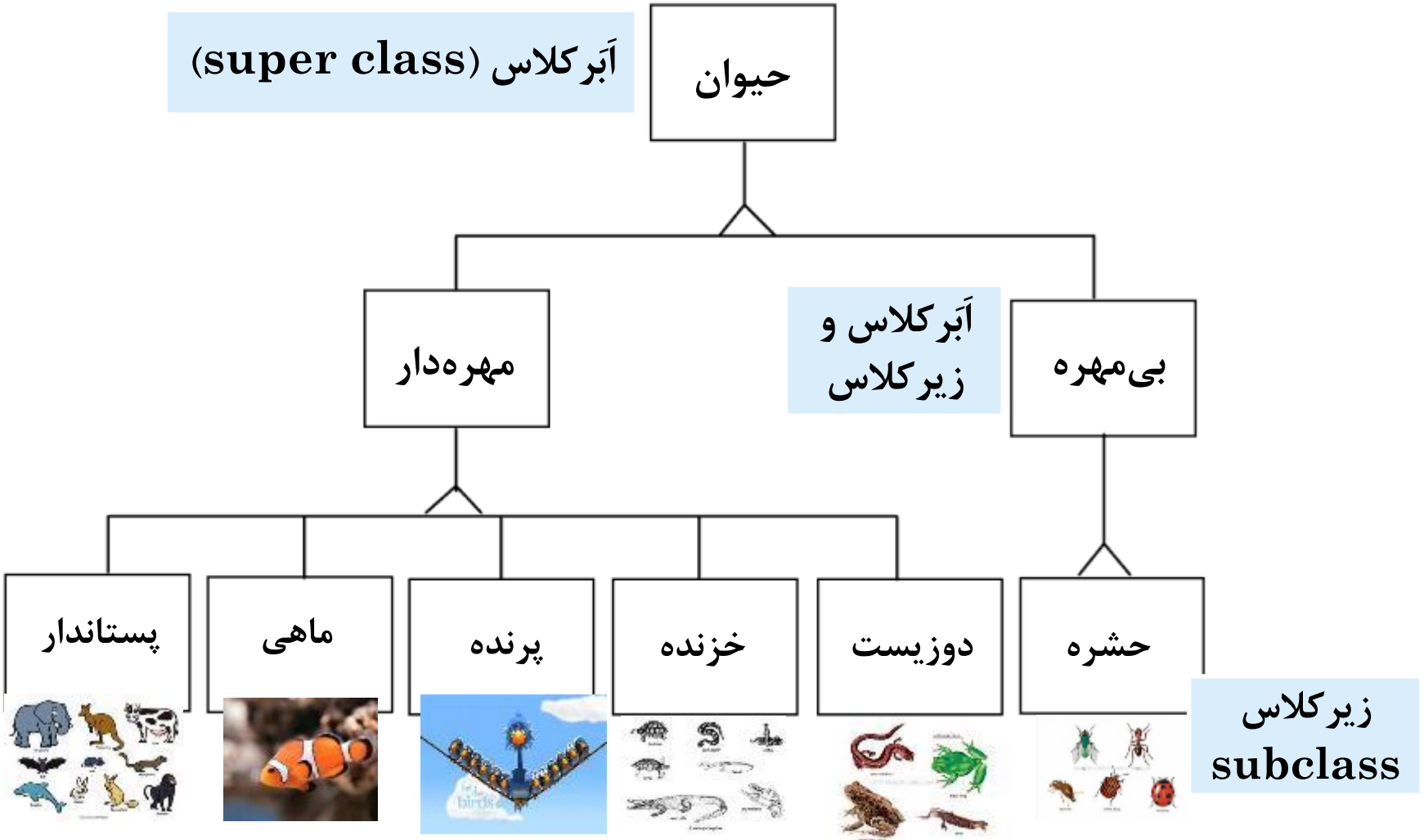
- کلیه حقوق این اثر متعلق به انجمن جاواکاپ است
- بازنشر یا تدریس آن‌چه توسط جاواکاپ و به صورت عمومی منتشر شده است، با ذکر مرجع (جاواکاپ) بلامانع است
- اگر این اثر توسط جاواکاپ به صورت عمومی منتشر نشده است و به صورت اختصاصی در اختیار شما یا شرکت شما قرار گرفته، بازنشر آن مجاز نیست
- تغییر محتوای این اثر بدون اطلاع و تأیید انجمن جاواکاپ مجاز نیست

سرفصل موضوعات

- وراثت (Inheritance)
  - سلسله‌مراتب کلاس‌ها (Class Hierarchies)
  - ارتباط IS-A
- فرایند مقداردهی اولیه در زیرکلاس‌ها
- جایگاه و کاربرد وراثت در طراحی نرم‌افزار
- نمایش وراثت در UML Class Diagram

آشنایی با مفهوم وراثت
Introduction to Inheritance

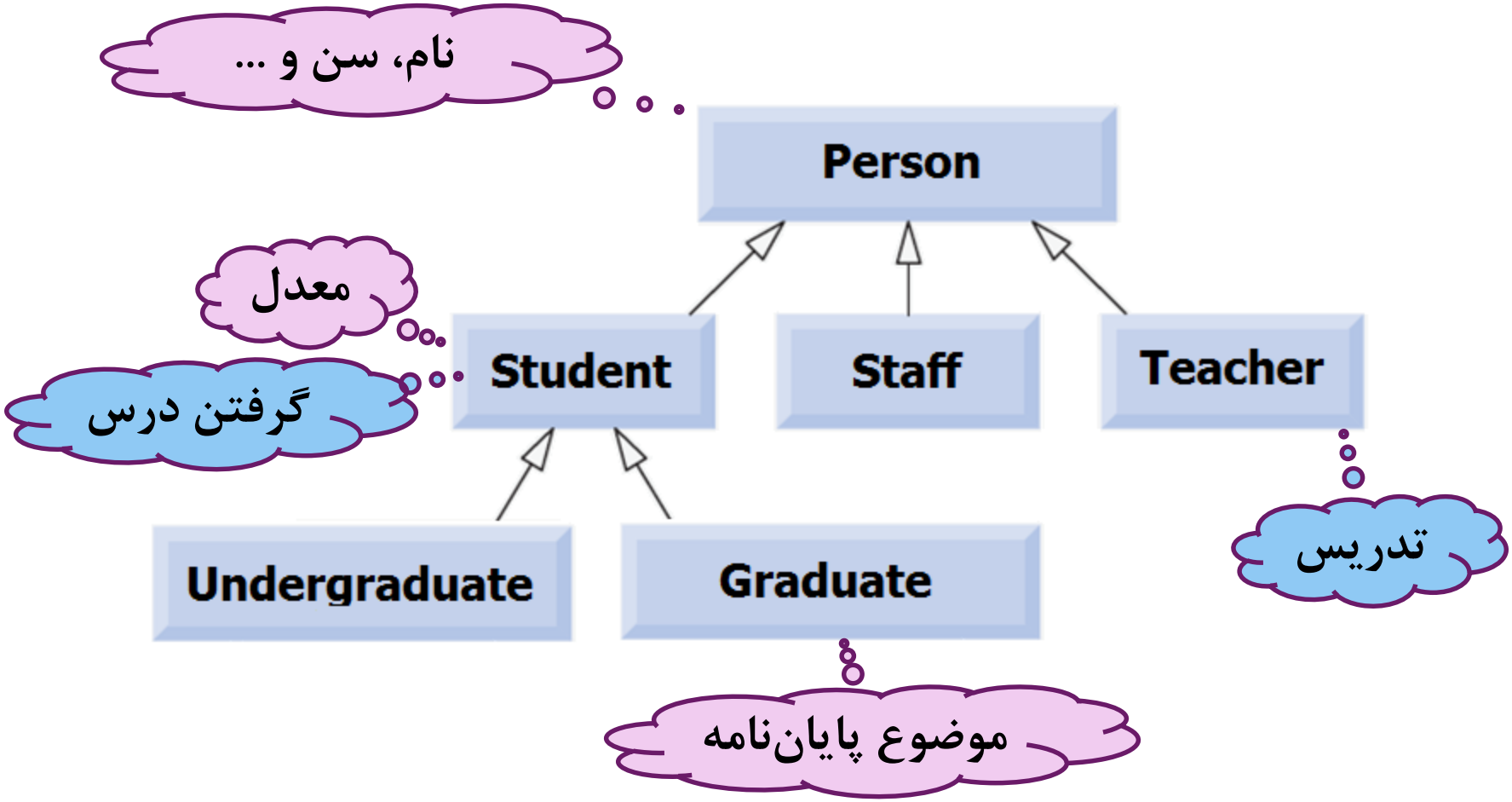
سلسله‌مراتب کلاس‌ها



معنای سلسله‌مراتب انواع کلاس‌ها

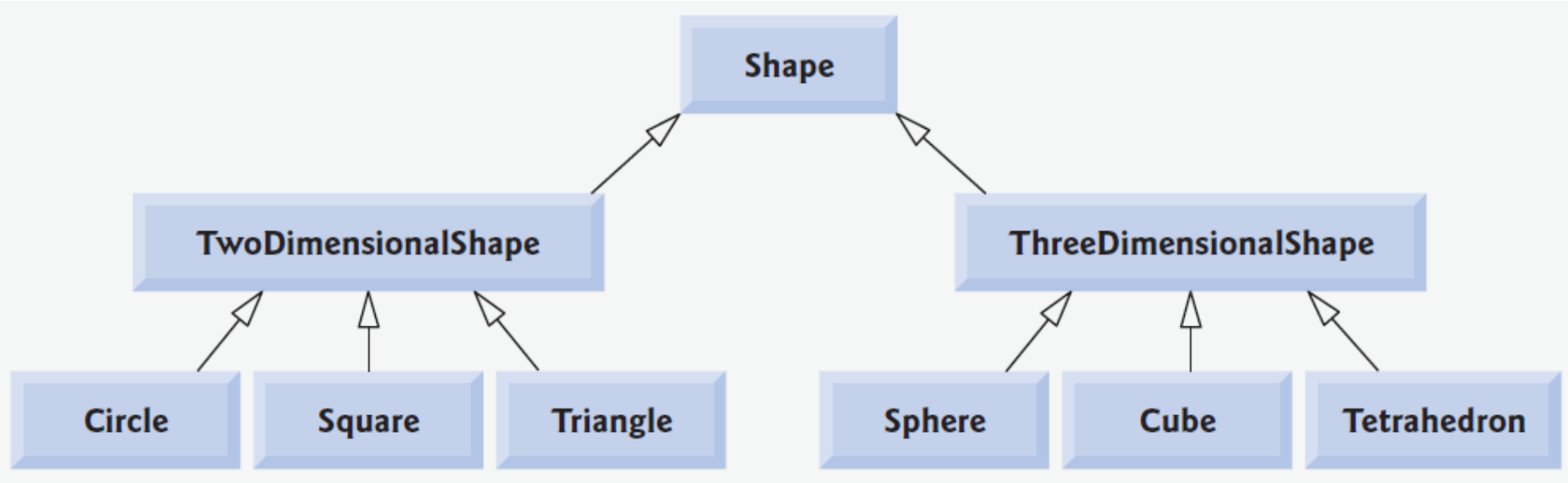
- دایره نمونه‌ها (اشیاء) در زیرکلاس محدودتر می‌شود
- زیرکلاس، **ویژگی‌ها** و **رفتار** اَبَرکلاس را به **ارث می‌برد**
  - اصطلاح **وراثت** و ارث‌بری (**Inheritance**)
- مثال: هر **حیوان**، ویژگی‌هایی مانند «سن» و «وضعیت سلامتی» دارد
  - این ویژگی‌ها به همه زیرکلاس‌ها به ارث می‌رسد
  - همه زیرکلاس‌های مستقیم و غیرمستقیم: مهره‌دار، بی‌مهره، ماهی، حشره و ...
  - یعنی هر شیء از زیرکلاس‌ها هم همین ویژگی‌ها را دارد
  - احتمالاً ویژگی‌های دیگری هم دارد (مثلاً هر ماهی «سرعت شناکردن» دارد)
- مثال: هر **حیوان**، رفتارهایی مانند «غذا خوردن» و «جابجا شدن» دارد
  - پس همه زیرکلاس‌ها هم این رفتارها را دارند (این رفتارها را به ارث می‌برند)

سلسله‌مراتب کلاس‌ها (مثال)

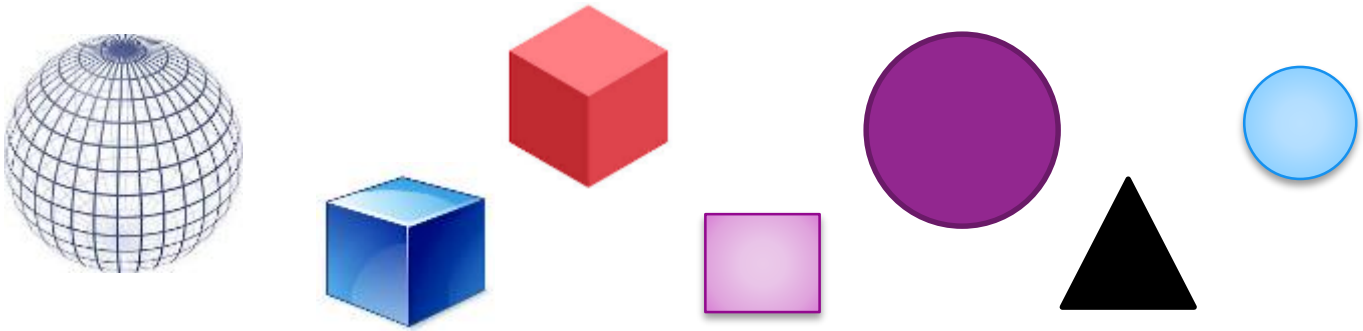


سلسله‌مراتب کلاس‌ها (مثال)

- کلاس‌ها:

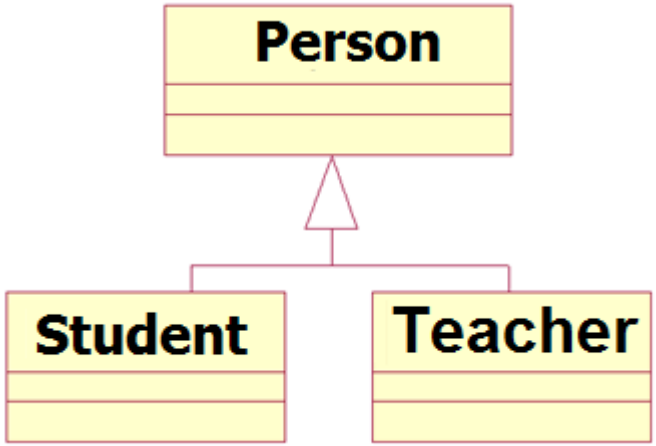


- نمونه‌ها:



انواع عام‌تر و انواع خاص‌تر

- اَبَرکلاس، نوع عام‌تری از زیرکلاس است (more general)
- زیرکلاس، نوع خاص‌تری از اَبَرکلاس است (more specific)
- تأکید: زیرکلاس و ابرکلاس هر دو «کلاس» هستند
- هر شیء از زیرکلاس، شیئی از ابرکلاس هم هست



- دانشجو زیرکلاس انسان است
- دانشجو نوع خاص‌تری از کلاس انسان است
  - (دایره محدودتری از نمونه‌ها را شامل می‌شود)
- همه ویژگی‌ها و رفتارهای انسان در دانشجو هم وجود دارد
  - مثل: نام، سن، غذاخوردن و ... البته دانشجو ویژگی‌ها و رفتارهای دیگر هم دارد
- علی علوی یک دانشجو است (یک نمونه، شیء). پس علی علوی، انسان هم هست

وراثت

- چند سلسله‌مراتب از انواع (کلاس‌ها) نام ببرید
- مثلاً هر یک از موارد زیر چه اَبَرکلاس‌ها و چه زیرکلاس‌هایی دارد؟
  - حساب بانکی، کارمند بانک، وام قرض‌الحسنه
  - خودرو
  - ورزشکار
- برای هر یک از کلاس‌های فوق چند نمونه فرضی نام ببرید
- آیا «علی کریمی» زیرکلاس فوتبالیست است؟
  - خیر. علی کریمی یک نمونه (شیء) است. یک کلاس نیست
  - زیرکلاس و اَبَرکلاس، هر دو «کلاس» هستند

واژه‌شناسی

(اصطلاحات انگلیسی مهمتر هستند)

- کلاس اصلی:
  - کلاس پایه (Base Class)
  - اَبَرکلاس (Superclass)
  - کلاس والد (Parent Class)
- کلاس وارث:
  - کلاس مشتق (Derived Class)
  - زیرکلاس (Subclass)
  - کلاس فرزند (Child Class)

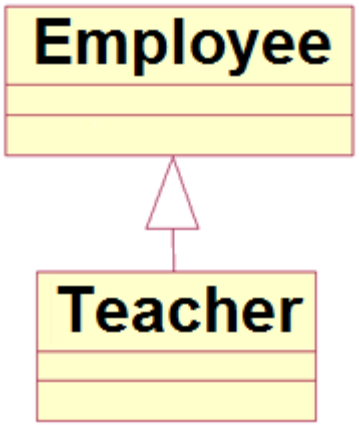
Rectangle **extends** Shape

- Rectangle is inherited/derived from Shape
- Rectangle is subclass/child of Shape
- Shape is the super-class/base-class/parent of Rectangle

نحوه پیاده‌سازی زیرکلاس‌ها

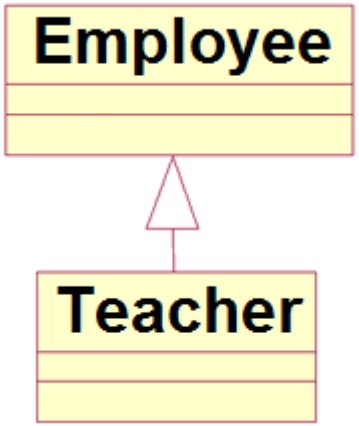
پیاده‌سازی

- یادآوری: زیرکلاس همه مشخصات و رفتارهای اَبَرکلاس را به ارث می‌برد
  - یعنی باید همه ویژگی‌ها و متدهای اَبَرکلاس را داشته باشد
- این وضعیت را چگونه پیاده‌سازی می‌کنید؟
- مثال:

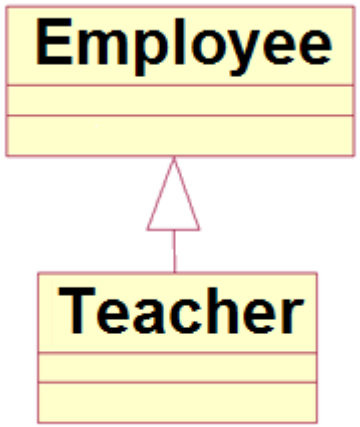


مثال:

```
public class Employee{
    private int salary;
    private Date birthDate;
    public void setName(String name){
        //...
    }
    public void setNationalID(String id){
        //...
    }
    public Date getBirthDate(){
        return birthDate;
    }
    public int getSalary(){
        return salary;
    }
}
```



```
public class Teacher {
    // Employee variables & methods
    private int salary;
    private Date birthDate;
    public void setName(String name){
        //...
    }
                                          ){
    public void setNationalID(String id){
        //...
    }
    public Date getBirthDate(){
        return birthDate;
    }
    public int getSalary(){
        return salary;
    }
    // Teacher variables & methods
    private Course[] courses;
    public Course[] getOfferedCourses(){
        return courses;
    }
}
```



چگونه کلاس Teacher از کلاس Employee ارث‌بری کند؟

اما کپی متن برنامه‌ها روش بسیار بدی است

راه بهتری برای پیاده‌سازی وراثت وجود دارد

وراثت در زبان‌های شیءگرا

- زبان‌های برنامه‌نویسی شیءگرا تعریف **وراثت** را ممکن می‌کنند
  - از جمله جاوا
- از این امکان برای تعریف زیرکلاس‌ها استفاده می‌شود
  - بدون این که نیازی به کپی کد از اَبَرکلاس باشد
- وراثت یکی از راه‌های **استفاده مجدد** از کد است
  - **code reuse**
  - کدی که در اَبَرکلاس نوشته شده، در زیرکلاس بازاستفاده می‌شود
  - دوباره نوشته نمی‌شود
- در جاوا، وراثت با کلیدواژه **extends** معرفی می‌شود

مثال:

```
public class Teacher extends Employee {
    private Course[] courses;
    public Course[] getOfferedCourses(){
        return courses;
    }
```

- کلاس Teacher از کلاس Employee ارث‌بری می‌کند
  - Teacher فرزند یا زیرکلاس Employee است
- ویژگی‌ها و متدهای Employee برای Teacher به ارث می‌رسند
  - Teacher همه این ویژگی‌ها و متدها را دارد
  - بدون این که لازم باشد آن‌ها را دوباره تعریف کنیم
- مثال:

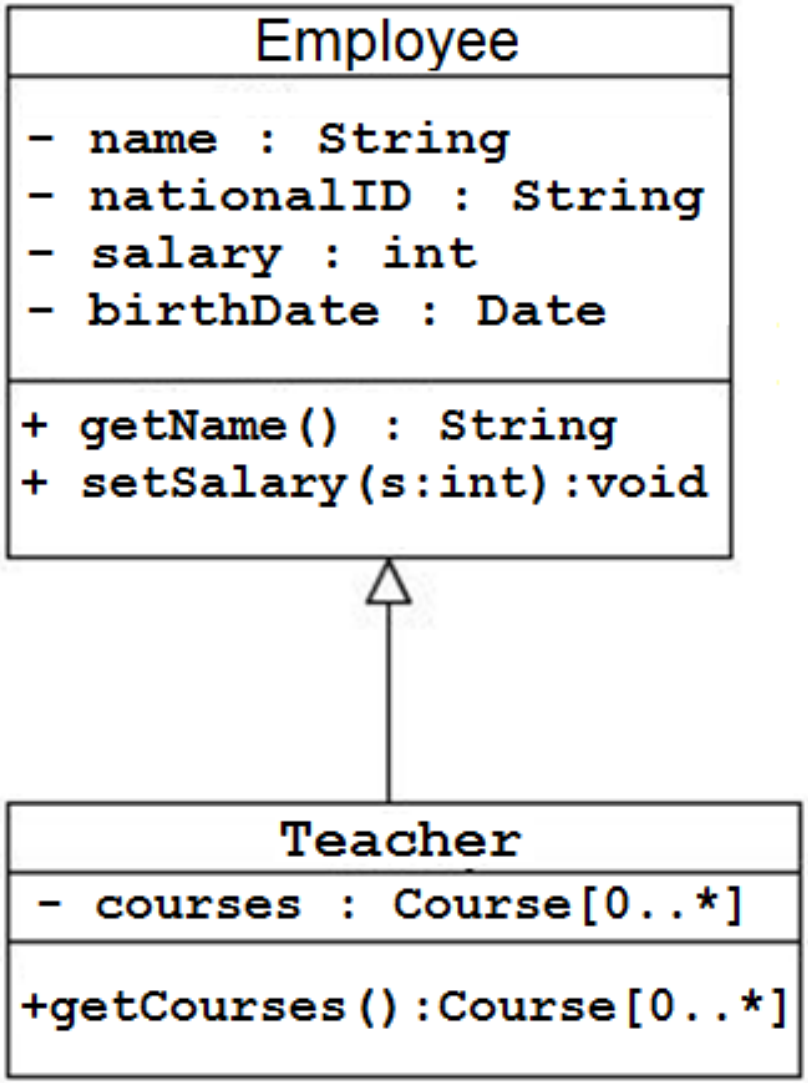
```
Teacher t = new Teacher();
t.setName("Ali");
int s = t.getSalary();
```

مثال‌های دیگری برای پیاده‌سازی وراثت

```
class Shape{
      int color;
      int positionX, positionY;
}
class Circle extends Shape{
      private int radius;
      public double getArea(){
            return 3.14*radius*radius;
      }
}
class Rectangle extends Shape{
      private int width, length;
      public double getArea(){
            return width*length;
      }
}
```

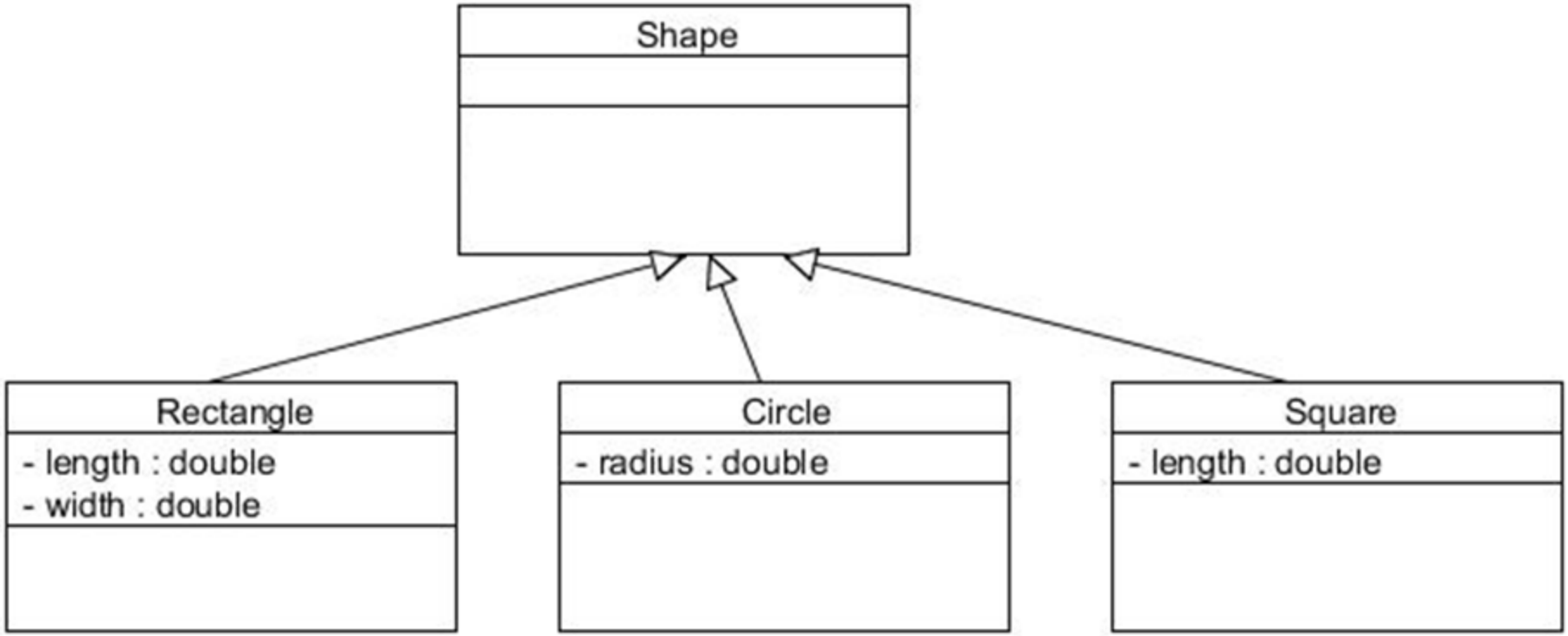
UML Class Diagram

نمودار UML برای کلاس‌ها



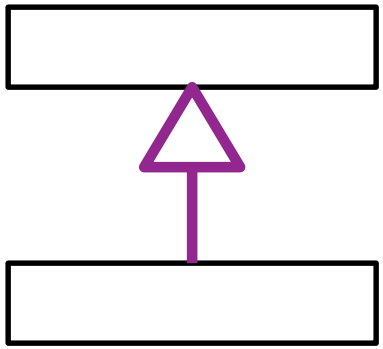
- UML Class Diagram
- نموداری برای توصیف طراحی کلاس‌ها
- کاربردهای مختلفی دارد
  - مثال: تعامل بین طراح و برنامه‌نویس
- نمودار UML قواعد خاصی دارد
  - مخصوص زبان جاوا نیست
- نمودار UML شامل:
  - متدها و ویژگی‌های کلاس‌ها
  - سطوح دسترسی
  - روابط بین کلاس‌ها
  - (وراثت: یکی از انواع رابطه ممکن است)

مثال دیگری برای UML Class Diagram

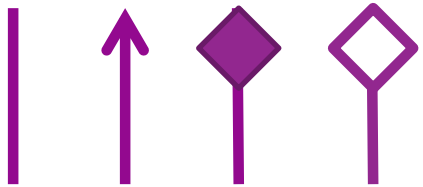


رابطه IS A

- اصطلاحاً: بین زیرکلاس و اَبَرکلاس رابطه IS A برقرار است
- Rectangle **is a** Shape
- A rectangle instance **is a** Shape instance too



- رابطه وراثت (IS A) در UML با یک فلش با سر مثلث توخالی نمایش داده می‌شود:
- شکل‌های دیگر معانی دیگری دارند:



کوییز

کوییز

- کلاس‌های زیر را در نظر بگیرید:
  - حساب بانکی ← شماره حساب، موجودی | واریز، برداشت
  - حساب سپرده کوتاه‌مدت ← سود سپرده | تبدیل به بلندمدت
  - حساب سپرده بلندمدت ← سود سپرده | تبدیل به کوتاه‌مدت
  - حساب قرض‌الحسنه ← امتیاز | شرکت در قرعه‌کشی
- هر کلاس چه ویژگی‌ها و متدهایی داشته باشد؟
- به رابطه وراثت دقت کنید (چه کلاس‌هایی زیرکلاس چه کلاس‌هایی هستند؟)
- نمودار UML Class Diagram را رسم کنید و این کلاس‌ها را پیاده‌سازی کنید

جایگاه وراثت در طراحی نرم‌افزار

استفاده مجدد از برنامه (Software Reuse)

- اصل مهم مهندسی نرم‌افزار: استفاده مجدد از دارایی‌های نرم‌افزاری
  - دارایی نرم‌افزاری: مثل کلاس‌های پیاده‌سازی شده
  - پرهیز از پیاده‌سازی مجدد، پرهیز از کپی متن برنامه‌ها
  - از کپی بخشی از یک برنامه در جای دیگر (Copy/Paste) جداً **بپرهیزید!**
  - کد تکراری (duplicate code) معایب فراوانی دارد
- چرا بازاستفاده اصل مهمی است؟
  - چون سرمایه‌ای که با تلاش و هزینه فراوان ایجاد شده، حفظ می‌شود
  - و به بخش‌های جدید منتقل می‌شود
  - چه سرمایه‌ای؟
    - طراحی، پیاده‌سازی، مستندسازی و تست نرم‌افزار

جایگاه وراثت در طراحی نرم‌افزار

- وراثت: راهی برای ایجاد کلاس‌های جدید با کمک کلاس‌های موجود
  - **استفاده مجدد** از ویژگی‌ها و رفتارهای کلاس اصلی در کلاس جدید
  - ایجاد امکانات جدید در کلاس جدید: زیرکلاس، اَبَرکلاس را توسعه می‌دهد (**extends**)
- راههای دیگری هم وجود دارد
  - مثلاً استفاده از یک کلاس به عنوان نوع یک ویژگی
- زیرکلاس: گروه محدودتری از اشیاء (نمونه‌ها) را در بر می‌گیرد
  - همه این اشیاء رفتار و ویژگی‌های اَبَرکلاس را دارند
  - اما برخی از رفتارها در زیرکلاس تغییر می‌کند
  - ممکن است ویژگی‌ها و رفتارهای جدیدی هم در زیرکلاس تعریف شوند

زیرکلاس ممکن است:

۱- متدها یا ویژگی‌های جدیدی تعریف کند

- ویژگی تعدادگل‌زده و رفتار شوت‌زدن در ورزشکار نیست و در فوتبالیست اضافه می‌شود

۲- از متدها و ویژگی‌های اَبَرکلاس استفاده کند

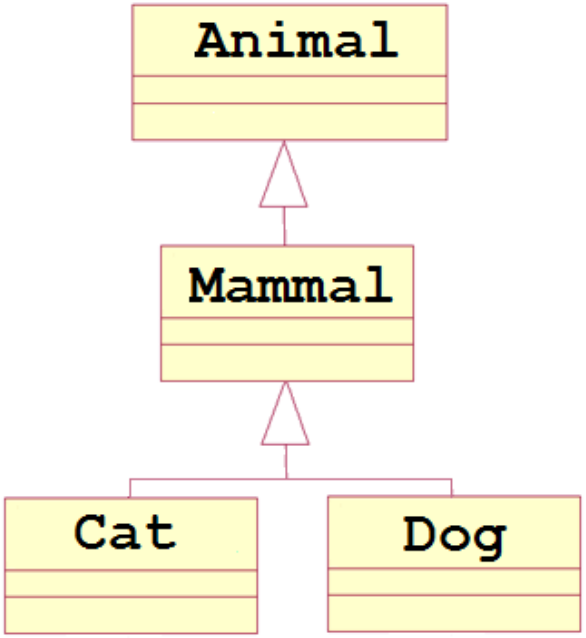
- استفاده از از ویژگی‌هایی که قبلاً در انسان تعریف شده در تعریف متدهای کارمند
- فراخوانی متدهایی که در ورزشکار تعریف شده برای شیئی از نوع فوتبالیست

۳- برخی رفتارها را تغییر دهد: پیاده‌سازی برخی متدها در زیرکلاس تغییر پیدا کند

- به تغییر معنای یک متد در زیرکلاس، **Override** کردن متد می‌گویند
- مثلاً: خارپشت بی‌دندان نوعی پستاندار است که با تخمگذاری تولیدمثل می‌کند
- متد تولیدمثل که در کلاس اَبَرکلاس (پستاندار) به شکل «بچه‌زایی» پیاده شده در زیرکلاس (خارپشت بی‌دندان) به صورت «تخم‌گذاری» تغییر می‌کند (override)

- نکته: زیرکلاس نمی‌تواند ویژگی یا متد اَبَرکلاس را حذف کند

سلسله مراتب کلاس‌ها



- زیرکلاس مستقیم:
  - کلاس سگ زیرکلاس مستقیم کلاس پستانداران است
- زیرکلاس غیرمستقیم:
  - کلاس گربه زیرکلاس غیرمستقیم کلاس حیوان است
  - همه ویژگی‌ها و رفتارهای حیوان به گربه هم به ارث می‌رسد
  - البته ممکن است کلاس پستانداران برخی از این متدها را تغییر داده باشد (Override)
- بدیهی است که یک کلاس می‌تواند چند زیرکلاس داشته باشد
- وراثت چندگانه (Multiple Inheritance) : ارث‌بری از چند کلاس
  - در برخی زبان‌های برنامه‌نویسی ممکن است
  - در زبان جاوا، یک زیرکلاس نمی‌تواند چند اَبَرکلاس داشته باشد
  - یعنی یک کلاس نمی‌تواند از چند کلاس ارث‌بری کند

البته جاوا هم در شرایط خاصی امکان وراثت چندگانه را فراهم می‌کند

مرور یک مثال از کاربرد وراثت

مثال

```
public class Person {
    private String name;
    private Long nationalID;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Long getNationalID() {
        return nationalID;
    }
    public void setNationalID(Long nationalID) {
        this.nationalID = nationalID;
    }
    public void show() {
        System.out.println("Person: name=" + name
        + ",nationalID=" + nationalID);
    }
}
```

```
class Student extends Person{
  private String studentID;
  public void setStudentID(String studentID) {
      this.studentID = studentID;
  }
  public String getStudentID() {
      return studentID;
  }
  public void takeCourse(Course course){
      ...
  }
  public void show(){
    System.out.println("Student: name=" + getName()
      + ",nationalID=" + getNationalID()
      + ",studentID=" + studentID);
  }
}
```

ویژگی جدید (توسعه مشخصات)

متدهای جدید
(توسعه رفتارها)

استفاده از متدهای اَبَرکلاس

تغییر تعریف یک متد (Override)

اشیاء (نمونه‌های) زیرکلاس

```
Person p1 = new Person();
p1.setName("Ali Alavi");
p1.setNationalID(1498670972L);
p1.show();
```

```
Student st = new Student();
st.setName("Ali Alavi");
st.setNationalID(1498670972L);
st.setStudentID("89072456");
st.show();
```

متدهایی (رفتارهایی) که در Person تعریف شده بودند

رفتارهایی که در Student اضافه شدند

رفتارهایی که در Student تغییر یافتند

کوییز

کوییز

```
class A {
 public int x;
 public int f(){
  return x;
 }
}
```

```
class B extends A {
 public int y;
 public int f(){
  return y;
 }
}
```

```
A a = new A();
a.x = 1;
B b = new B();
b.x = 2;
b.y = 3;
System.out.println(a.f());
System.out.println(b.f());
```

- خروجی این قطعه برنامه چیست؟

پاسخ صحیح:

1

3

تمرین عملی

تمرین عملی

- تمرین کوتاه و ساده:
- کلاس Animal را پیاده‌سازی کنید
  - ویژگی نام و سن + رفتارهای getter و setter
- کلاس Dog را پیاده‌سازی کنید
  - ویژگی سرعت دویدن + getter, setter + رفتار پارس کردن
- اشیاءی از این کلاس‌ها بسازیم و از آن‌ها استفاده کنیم
- طرح مسأله:
  - متد پارس کردن می‌تواند یک متد انتزاعی با عنوان صحبت کردن باشد: talk
  - این متد در Animal معنی دارد ولی قابل پیاده‌سازی نیست
  - جزئیات بیشتر را بعداً می‌بینیم

وراثت و سطوح دسترسی

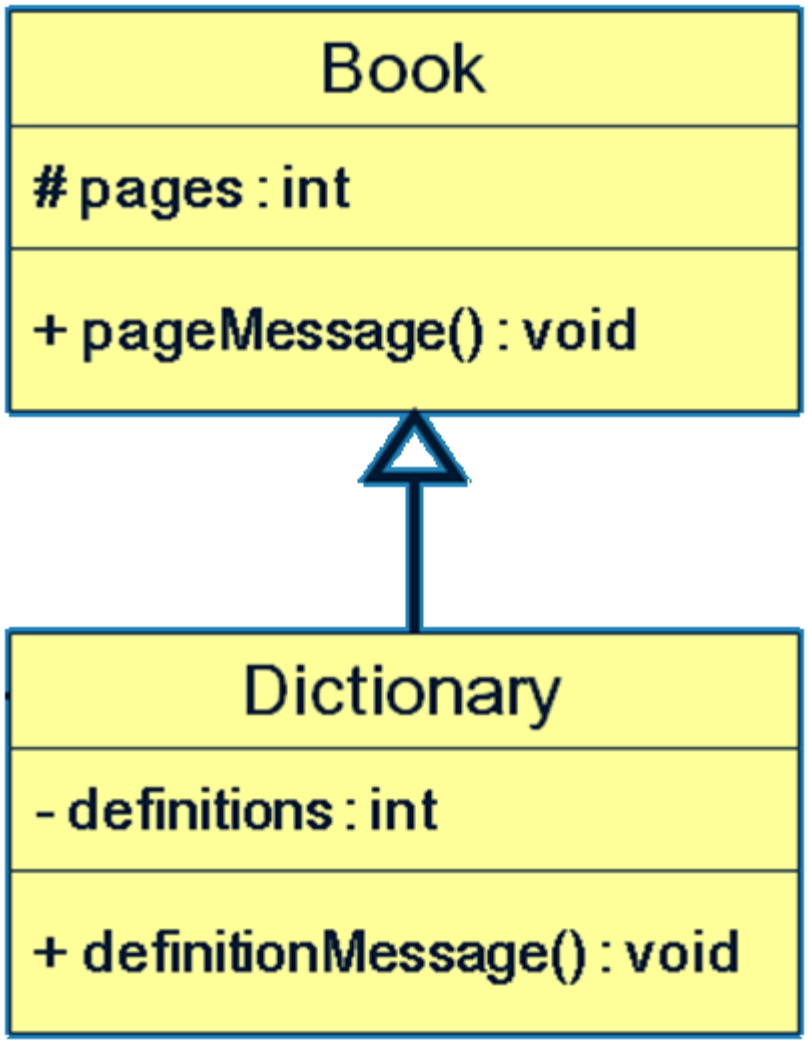
سطح دسترسی protected

- یادآوری: سطوح دسترسی public ، package access و private
- گاهی لازم است یک ویژگی یا متد در زیرکلاس‌ها قابل استفاده باشد
  - ولی از سایر کلاس‌ها مخفی باشد
  - مثال: اگر name در Person ، private باشد:
    کلاس Student نمی‌تواند از این ویژگی استفاده کند
- برای این نیازمندی، سطح دسترسی دیگری ایجاد شده است: **protected**
- اگر عضوی (متد یا متغیر) protected باشد، برای زیرکلاس‌ها در دسترس است
  - نکته: این عضو برای کلاس‌های داخل همان بسته (package) هم قابل استفاده است
  - برای سایر کلاس‌ها مخفی (غیرقابل استفاده) خواهد بود
- یک سطح دسترسی میانی: کمتر از public و بیشتر از package access

مرور سطوح دسترسی

- Public (عمومی)
  - از همه کلاس‌ها قابل استفاده است
- Protected (محفوظ)
  - از زیرکلاس‌ها قابل استفاده است
  - از سایر کلاس‌های همان بسته هم قابل استفاده است
- Package access (دسترسی در بسته)
  - از سایر کلاس‌های همان بسته هم قابل استفاده است
- Private (خصوصی)
  - فقط در همان کلاس قابل استفاده است (از دید هر کلاس دیگری مخفی است)

نمایش در UML



- در UML Class Diagram
  - اعضای protected : با #
  - اعضای خصوصی : با –
  - اعضای عمومی : با +

نکته

- هرگاه یک زیرکلاس می‌سازیم و متدی را Override می‌کنیم:

  حق نداریم سطح دسترسی به این متد را کاهش دهیم

  وگرنه: خطای کامپایل

- مثلاً نمی‌توانیم متدی که در اَبَرکلاس public بوده را در زیرکلاس، private تعریف کنیم (override کنیم)
- چرا؟ زیرا این کار قانون IS-A را نقض می‌کند
  - هر شیئی از زیرکلاس، شیئی از جنس اَبَرکلاس هم هست
  - هر رفتاری که در اَبَرکلاس هست، باید برای اشیاء زیرکلاس هم قابل فراخوانی باشد
  - مثلاً اگر «غذاخوردن» یک متد public در کلاس «حیوان» است:
    - یعنی هر حیوانی این رفتار را دارد
    - مثلاً کلاس سگ نمی‌تواند این متد را مخفی (غیرقابل فراخوانی) کند
- پس سطح دسترسی به متدها در زیرکلاس‌ها قابل کاهش نیست

کلیدواژه super

کلیدواژه *super*

- گاهی در زیرکلاس می‌خواهیم از عضوی استفاده کنیم که در اَبَرکلاس تعریف شده
  - مثلاً یک متد یا ویژگی (متغیر) از اَبَرکلاس
- فرض کنید عضوی دقیقاً با همان نام در زیرکلاس هم وجود داشته باشد
  - مثلاً متد موردنظر را در زیرکلاس override کرده باشیم
- در این شرایط با کمک نام این عضو، عضوی از همین کلاس فراخوانی می‌شود
  - نه از اَبَرکلاس
- راه حل: کلیدواژه **super**
  - با super می‌توانیم از اعضایی که در اَبَرکلاس تعریف شده‌اند استفاده کنیم

کلیدواژه super (ادامه)

- مثال: `super.f();`
  - با این کار متد f که در اَبَرکلاس تعریف شده فراخوانی می‌شود
  - به‌ویژه اگر متد f() در همین کلاس وجود داشته باشید
- مثال: `s = super.name;`
- با کمک super تصریح می‌کنیم که یک عضو از اَبَرکلاس موردنظر است
- با کمک this تصریح می‌کنیم که یک عضو از همین کلاس موردنظر است
- مثال: `super(name, id);`
  - با این کار سازنده اَبَرکلاس فراخوانی می‌شود

مثال: کاربرد super

- استفاده مجدد از تعریف متدی که در اَبَرکلاس هست و در حال Override کردن آن هستیم:

```
class Student extends Person{
  public void show(){
   super.show();
   System.out.println(",studentID="+studentID);
  }
}
```

کاربرد super در فراخوانی سازنده اَبَر کلاس

```
class Person{
 private String name;
 private String nationalID;
 public Person(String name, String nationalID) {
   this.name = name;
   this.nationalID = nationalID;
 }
}
```

```
class Student extends Person{
 private long studentID;
 public Student(String name, String id, long studentID) {
  super(name, id);
   this.studentID = studentID;
}
}
```

کوییز

خروجی این قطعه برنامه چیست؟

```
class Parent{
 protected void f(){
    System.out.println("f() in Parent");
 }
}
public class Child extends Parent{
 public void f(){
   System.out.println("f() in Child");
 }
 public void a(){ f(); }
 public void b(){ this.f(); }
 public void c(){ super.f(); }
}
```

```
Parent parent = new Parent();
Child child = new Child();
parent.f();
child.f();
child.a();
child.b();
child.c();
```

پاسخ صحیح:

```
f() in Parent
f() in Child
f() in Child
f() in Child
f() in Parent
```

خروجی این برنامه چیست؟

```
class A {
    public int a;
}

public class B extends A{
    private int a;
    public void f(){
        int a ;
        this.a = 5;
        super.a = 6;
        a=4;

        System.out.println(a);
        System.out.println(this.a);
        System.out.println(super.a);
    }
    public static void main(String[] args) {
        new B().f();
    }
}
```

نکته:

- مفهوم Override برای متدها معنی دارد
- برای متغیرها معنی ندارد
- تعریف ویژگی‌هایی در زیرکلاس که هم‌نام ویژگی‌های اَبَرکلاس هستند، کار رایجی نیست

کلاس Object

کلاس Object در جاوا

- کلاسی در جاوا با نام **Object** وجود دارد
- هر کلاسی در جاوا، زیرکلاس (فرزند) Object است
  - همه کلاس‌ها از Object ارث‌بری می‌کنند
- کلاس‌هایی که هنگام تعریف از کلیدواژه extends استفاده نمی‌کنند:
  - به صورت ضمنی از Object ارث‌بری می‌کنند
  - مثلاً در تعریف `class Parent{...}` انگار که نوشته‌ایم:

```
class Parent extends Object{...}
```

- کلاس‌هایی که هنگام تعریف از کلیدواژه extends استفاده می‌کنند:
  - به صورت غیرمستقیم از Object ارث‌بری می‌کنند

کلاس Object

- کلاس Object متدهای آشنایی دارد:
  - toString , finalize , equals , hashCode , ...
- هر کلاس جدیدی که ایجاد می‌کنیم:
  - امکانات Object را گسترش می‌دهد
  - و برخی از رفتارهای Object را تغییر می‌دهد
  - مثلاً برای equals یا toString تعریف جدیدی ایجاد می‌کنیم
    - تعریف این متدها در کلاس Object (که معمولاً ناکارامد هستند) را تغییر می‌دهیم
    - یعنی Override می‌کنیم

وراثت و فرایند مقداردهی اولیه اشیاء

فرایند مقداردهی اولیه (Initialization)

- وقتی از وراثت استفاده می‌کنیم و زیرکلاس را می‌سازیم:
  - بخشی از کلاس در اَبَرکلاس تعریف می‌شود
  - بخشی از ویژگی‌ها و متغیرهای شیء در اَبَرکلاس قرار دارند
- ویژگی‌هایی که در اَبَرکلاس تعریف شده‌اند هم باید مقداردهی اولیه شوند
  - فرایند مقداردهی اولیه این متغیرها در اَبَرکلاس تعریف می‌شود
  - چون اَبَرکلاس می‌داند که چه ویژگی‌هایی دارد و چگونه این ویژگی‌ها باید آماده شوند
  - این کار به خصوص توسط سازنده (constructor) در اَبَرکلاس انجام می‌شود
- روال مقداردهی اولیه ویژگی‌های شیئی از نوع زیرکلاس چگونه است؟

مقداردهی اولیه شیئی از نوع زیرکلاس

- وقتی یک زیرکلاس تعریف می‌کنیم:
  - باید سازنده (constructor) مشخصی از اَبَرکلاس در سازنده‌ی زیرکلاس فراخوانی شود
    - این کار با کلیدواژه super انجام می‌شود
    - فراخوانی سازنده‌ی اَبَرکلاس، باید اولین دستور از سازنده زیرکلاس باشد
  - وگرنه سازنده‌ای بدون پارامتر از اَبَرکلاس به صورت ضمنی فراخوانی می‌شود
    - اگر چنین سازنده‌ای در اَبَرکلاس نباشد، خطای کامپایل ایجاد می‌شود
- نکته: سازنده‌ها به ارث نمی‌رسند
  - مثلاً اگر سازنده‌ای در اَبَرکلاس باشد که یک پارامتر int می‌گیرد
  - این سازنده به زیرکلاس به ارث نمی‌رسد
  - اگر زیرکلاس به چنین سازنده‌ای نیاز دارد، باید آن را صراحتاً تعریف کند

فراخوانی سازنده اَبَرکلاس

```
class Person{
 private String name;
 private String nationalID;
 public Person(String name, String nationalID) {
   this.name = name;
   this.nationalID = nationalID;
 }
}
```

```
class Student extends Person{
 private long studentID;
 public Student(String name, String id, long studentID) {
  super(name, id);
   this.studentID = studentID;
}
}
```

```
Person p = new Person("Ali Alavi", "1290562352");
Student s = new Student("Ali Alavi", "1290562352", 94072456);
```

روند مقداردهی اولیه:

- یک بار برای همیشه: کلاس (زیرکلاس) بارگذاری می‌شود
  - قبل از زیرکلاس، اَبَرکلاس بارگذاری (Load) می‌شود
  - اگر اَبَرکلاس هم از کلاس دیگری ارث‌بری کرده، آن کلاس هم قبلاً بارگذاری شده است

(یادآوری: هنگام بارگذاری کلاس، ویژگی‌های استاتیک مقداردهی اولیه می‌شوند)

- هر بار که یک شیء از نوع زیرکلاس ایجاد می‌شود:
  ۱- ابتدا بخشی از شیء که در اَبَرکلاس تعریف شده، مقداردهی اولیه می‌شود
  ۲- سپس سایر ویژگی‌های شیء که در زیرکلاس تعریف شده، آماده می‌شود

خلاصه روند مقداردهی اولیه

- یک بار برای هر کلاس
  - مقداردهی درخط متغیرهای استاتیک در اَبَرکلاس
  - بلوک استاتیک در اَبَرکلاس
  - مقداردهی درخط متغیرهای استاتیک در زیرکلاس
  - بلوک استاتیک در زیرکلاس
- یک بار به ازای ایجاد هر شیء
  - مقداردهی درخط ویژگی‌های اَبَرکلاس
  - بلوک مقداردهی اولیه در اَبَرکلاس
  - سازنده‌ی اَبَرکلاس
  - مقداردهی درخط ویژگی‌های زیرکلاس
  - بلوک مقداردهی اولیه در زیرکلاس
  - سازنده‌ی زیرکلاس

کوییز

```
public class Parent {
   static int a = A();
   static{
     a=B();
   }
   int b = E();
   {
     b = F();
   }
   public Parent(){
     b = G();
   }
}
```

```
class Child
 extends Parent{

   static int c = C();
   static{
      c=D();
   }
   int b = H();
   {
      b = I();
   }
   public Child(){
      b = J();
   }
}
```

اگر **دوبار** دستور `new Child();` فراخوانی شود، به ترتیب چه متدهایی اجرا می‌شوند؟

سؤال

```
class Person{
 private String name;
 private String nationalID;
 public Person(String name, String nationalID) {
   this.name = name;
   this.nationalID = nationalID;
 }
}
```

Implicit super constructor Person() is undefined for default constructor. Must define an explicit constructor

```
class Student extends Person{
 private long studentID;
}
```

```
Person p = new Person("Ali Alavi", "1290562352");
Student s = new Student();
```

- خطای کامپایل کجاست؟

سؤال

- در هنگام تعریف سازنده‌ی زیرکلاس،
  در صورت فراخوانی سازنده‌ی اَبَرکلاس (با کمک super)،
  نمی‌توانیم ویژگی‌های زیرکلاس را پاس کنیم
  - مثال: ~~super(this.name)~~ (دچار خطای کامپایل می‌شود)
- چرا؟
  - زیرا ویژگی‌های زیرکلاس، بعد از ویژگی‌های اَبَرکلاس آماده می‌شوند
  - مقداردهی اولیه آن‌ها بعد از فراخوانی سازنده‌ی اَبَرکلاس انجام می‌شود

ترکیب یا وراثت؟

ترکیب (Composition)

```
class Human{
private Heart heart;
private Hand leftHand;
private Hand rightHand;
}
```

- گاهی در یک شیء، ارجاع به اشیاء دیگری وجود دارد
  - به این رابطه، **ترکیب** (Composition) می‌گویند
  - رابطه **has a** (متفاوت با رابطه **is a**)

```
class Professor extends Human{
 private University university;
 private Course[] courses;
}
```

- در مثال‌های مقابل روابط is a و روابط has a را پیدا کنید
- **ترکیب**: روش دیگری برای **استفاده مجدد** (code reuse)
- کدام یک بهتر است؟ ترکیب یا وراثت؟
  - گاهی ترکیب لازم است و گاهی وراثت
  - بین دو کلاس، رابطه is a برقرار است یا has a ؟

```
class Car extends Object{
 private Engine engine;
 private Tyre[] tyres;
}
```

- اگر بین دو شیء رابطه is a برقرار نیست، از وراثت استفاده **نکنید**

مثال

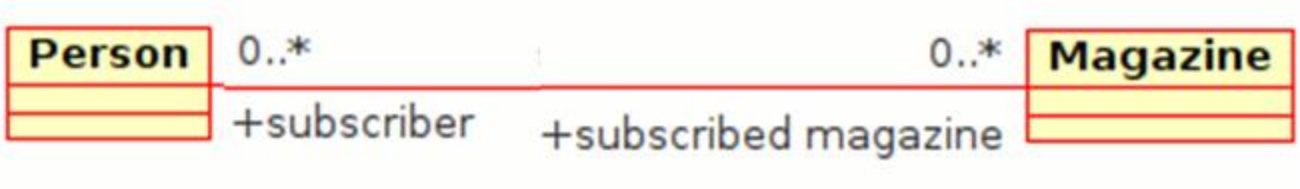
```
public class Sorting {
        public List sort(List list) {
                // sort implementation
                return list;
        }
}
class DynamicDataSet extends Sorting {
        // DynamicDataSet implementation
}
```

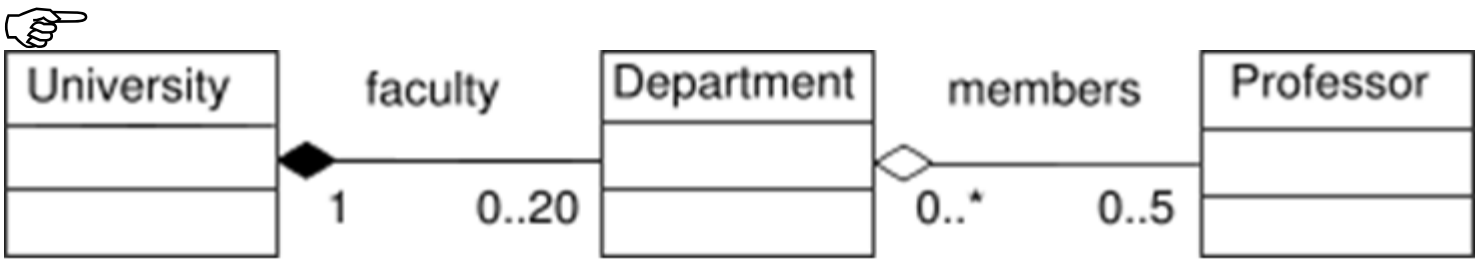
- در برنامه فوق:
  - طراح، کلاس DynamicDataSet را فرزند (زیرکلاس) Sorting قرار داده
  - تا بتواند از امکانات مرتب‌سازی در کلاس DynamicDataSet استفاده کند
- اشتباه طراحی؟
  - بین دو کلاس رابطه is a برقرار نیست. هر DynamicDatSet یک Sorting نیست
- روش بهتر: ترکیب
  - شیئی از جنس Sorting در کلاس DynamicDataSet قرار گیرد

```
class DynamicDataSet{
  private Sorting sorting = new Sorting();
  //...
}
```

درباره «ترکیب» (Composition)

- در مواقعی که شک دارید، Composition را ترجیح دهید
  - یک کلاس فقط از یک کلاس می‌تواند ارث‌بری کند
  - ولی تعداد زیادی کلاس را می‌تواند در ترکیب به‌کار گیرد
- ترکیب، انواع مختلفی دارد: Association, Composition, Aggregation
- نحوه نمایش در UML





چند نکته تکمیلی

چند نکته

- دقت کنید: مفهوم **Overload** و مفهوم **Override** متفاوت هستند
- **Overload** (سربار کردن) :
  - چند متد هم‌نام با امضاهای مختلف (مجموعه پارامترهای متفاوت)
  - همه آن‌ها همزمان معتبر هستند و هریک قابل استفاده هستند

```
int f(){return 2;}
int f(int a){return a;}
```

- **Override** (لغو کردن) :
  - امضای متد (مجموعه پارامترها) در زیرکلاس دقیقاً مثل امضای آن در اَبَرکلاس است
  - معنای متدی که در اَبَرکلاس وجود داشته، تغییر می‌کند: معنی قبلی لغو می‌شود
  - نمونه‌های (اشیاء) زیرکلاس، رفتار تغییریافته را استفاده می‌کنند
- در یک کلاس می‌توانیم متدهای همان کلاس را Overload کنیم
- در یک زیرکلاس می‌توانیم متدهای اَبَرکلاس را هم Overload کنیم
- ولی Override مربوط به وراثت است (در زیرکلاس رخ می‌دهد)

حاشیه‌نگاری @Override

- مفهوم حاشیه‌نگاری (Annotation)
  - توضیحاتی که با @ شروع می‌شوند
  - شکلی از فراداده (Metadata)
  - توضیحی درباره یک متد یا کلاس یا ... می‌دهد
  - بر نحوه کامپایل یا اجرای آن تأثیر می‌گذارد
- مثال: @Override
  - قبل از تعریف یک متد می‌آید
  - تصریح می‌کند که این متد، همین متد از اَبَرکلاس را Override می‌کند
  - اگر به‌درستی متد مورد نظر را Override نکنیم: خطای کامپایل رخ می‌دهد
  - این تصریح، می‌تواند اشتباه‌های ناخواسته برنامه‌نویس را کمتر کند

مثال برای @Override

```
class Animal {
  public void talk();
}
class Dog extends Animal{
  private String name;
  @Override
  public String toString() {
    return name;
  }
  @Override
  public void talk() {
    System.out.println("Hop!");
  }
}
```

- چه می‌شد اگر به اشتباه، هنگام تعریف کلاس Dog:
- toSrting را tostring و یا talk را Talk تایپ می‌کردیم؟
- یا پارامترهایی برای talk در نظر می‌گرفتیم؟
- حاشیه‌نگاری @Override کمک می‌کرد تا این اشتباه را کشف کنیم:
- یک خطای کامپایل ایجاد می‌کرد

کوییز

کوییز

- کدام یک از متدهای کلاس Superclass در متد g از Subclass قابل فراخوانی هستند؟ (چند مورد)

```
public class Superclass{
 private void a(){}
 void b(int a){}
 protected void c(String a){}
 public void d(int a, int b){}
}
```

```
public class Subclass {
public void g(){

}
}
```

- پاسخ صحیح:
  - متدهای c و d : قطعاً بله
  - متد a : قطعاً خیر
  - متد b ممکن است:
    - به شرطی که هر دو کلاس در یک بسته (package) باشند

```java
class Superclass{
 private void f(){ System.out.println("1"); }
 void f(int a){ System.out.println("2"); }
 protected void f(String a){ System.out.println("3"); }
 public void f(int a, int b){ System.out.println("4"); }
}
```

```java
public class Subclass extends Superclass{
public void f(){ System.out.println("5"); }
protected void f(String a){ System.out.println("6"); }
public void f(int a, int b){ System.out.println("7"); }
```

خروجی هر یک از قطعه برنامه‌های زیر چیست؟

```java
Subclass s = new Subclass();
s.f();     5
s.f(1);    2
s.f("1");  6
s.f(1,2);  7
```

```java
Superclass t = new Superclass();
t.f(1);    2
t.f(1,2);  4
t.f("1");  3
```

تمرین عملی

تمرین عملی

- تمرین سطوح دسترسی
- نگاهی به متن کلاس Object
- استفاده از super
  - در مقابل this
  - برای سازنده
- تمرین فرایند مقداردهی اولیه در زیرکلاس
- حاشیه‌نگاری @Override
- عدم امکان کاهش دسترسی به متدها در زیرکلاس

جمع‌بندی

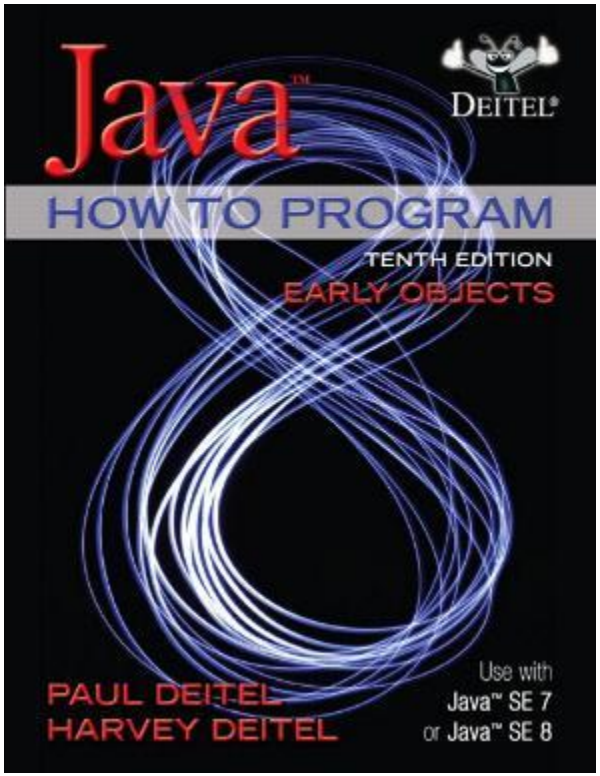
جمع‌بندی

- وراثت یا ارث‌بری (Inheritance)
  - ارتباط IS-A
- نمایش وراثت در نمودارهای UML
- سطح دسترسی protected
- فرایند مقداردهی اولیه در زیرکلاس‌ها
- کلیدواژه super

مطالعه کنید

- فصل ۹ کتاب دایتل

Java How to Program (Deitel & Deitel)



9- Object-Oriented Programming:
Inheritance

- تمرین‌های همین فصل از کتاب دایتل

تمرین

- برای کلاس‌های زیر، ابتدا نمودار UML Class Diagram طراحی کنید
- سپس این کلاس‌ها (البته به جز Object) را پیاده‌سازی کنید
  - جزئیات پیاده‌سازی را تکمیل کنید: سازنده، متدها و ...

- Object
- Person (name, phoneNumber)
- Student (average, entranceYear)
- GraduateStudent (thesisTitle, supervisor)
- Instructor (rank, supervisedStudents)

جستجو کنید و بخوانید

- کپی کد (copy/paste) و کد تکراری (duplicate code) چه معایبی دارد؟
- وراثت چندگانه (Multiple Inheritance)
  - چه مشکلاتی ایجاد می‌کند؟
  - چه زبان‌هایی از آن پشتیبانی می‌کنند؟
  - زبان جاوا چه شکل‌هایی از وراثت چندگانه را پشتیبانی می‌کند؟
  - چه شکلی از وراثت چندگانه در جاوا ۸ ممکن شده است؟
- متدهای کلاس Object

جستجو کنید و بخوانید (ادامه)

- درباره حاشیه‌نگاری (Annotation) بیشتر بخوانید
  - چه کاربردهایی دارد؟
  - چه حاشیه‌نگاری‌های مهم دیگری (به جز @Override) در جاوا هست؟
- UML Class Diagram
- روش‌های ترکیب و تفاوت‌های آن‌ها:
  - Association, Aggregation, Composition
    - تفاوت آن‌ها در نمودار UML
- «ترکیب را به وراثت ترجیح دهید»:
- Favor (prefer) **composition over inheritance**

پایان

سایر مطالب

تاریخچه تغییرات

نسخه	تاریخ	توضیح
۱.۰.۰	۱۳۹۴/۳/۲۴	نسخه اولیه ارائه آماده شد
۱.۱.۰	۱۳۹۴/۳/۲۶	موضوع ترکیب (رابطه is a) اضافه شد
۱.۲.۰	۱۳۹۴/۳/۲۶	بخش متدها و کلاس‌های انتزاعی به موضوع چندریختی منتقل شده