به نام پروردگار دانایی

طراحی سیستم‌های شی‌گرا

برنامه‌نویسی شی‌گرا

درس پنجم: بیشتر در مورد وراثت (بررسی چندریختی)

سید کاوه احمدی

جلسه گذشته

مفاهیم

- Inheritance
- superclass (parent)
- subclass (child)
- is-a
- inheritance hierarchy
- abstract class
- subtype
- substitution
- polymorphic variable
- polymorphic
- collection
- type loss
- wrapper classes

جاوا

- extends
- super (in constructor)
- cast (revisited)
- Object
- wrapper classes

مفاهیم این جلسه

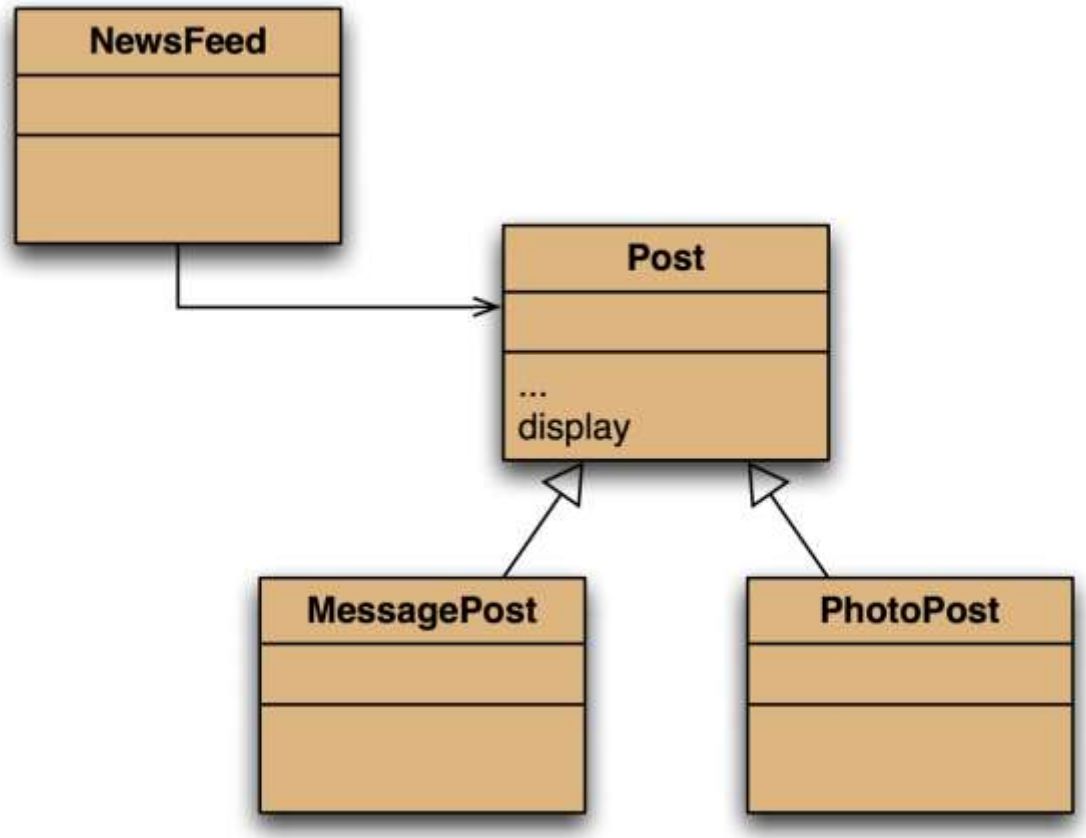
- نوع‌های ایستا و پویا
  - نوع ایستا اجازه بررسی زمان کامپایل را فراهم می‌کنند.
  - نوع پویا اجازه چندریختی متدها هنگام اجرا را فراهم می‌کنند.
- چندریختی متدها
  - رفتارهای متفاوت با فراخوانی متد مشابه با توجه به نوع پویای شی
- بازنویسی (`Override`) کردن متدهای به ارث برده شده.
- جستجوی پویای متد
- دسترسی محافظت شده

کد کلاس NewsFeed

- چاپ اطلاعات با فقط یک حلقه تکرار:

```
/**
 * Show the news feed. Currently: print the
 * news feed details to the terminal.
 * (Later: display in a web browser.)
 */
public void show()
{
    for(Post post : posts) {
        post.display();
        System.out.println(); // Empty line ...
    }
}
```

سلسله مراتب وراثت



تداخل در خروجی

```
Leonardo da Vinci
Had a great idea this morning.
But now I forgot what it was. Something to do with flying
...
40 seconds ago - 2 people like this.
  No comments.

Alexander Graham Bell
[experiment.jpg]
I think I might call this thing 'telephone'.
12 minutes ago  -  4 people like this.
  No comments.
```

خروجی مورد نظر ما

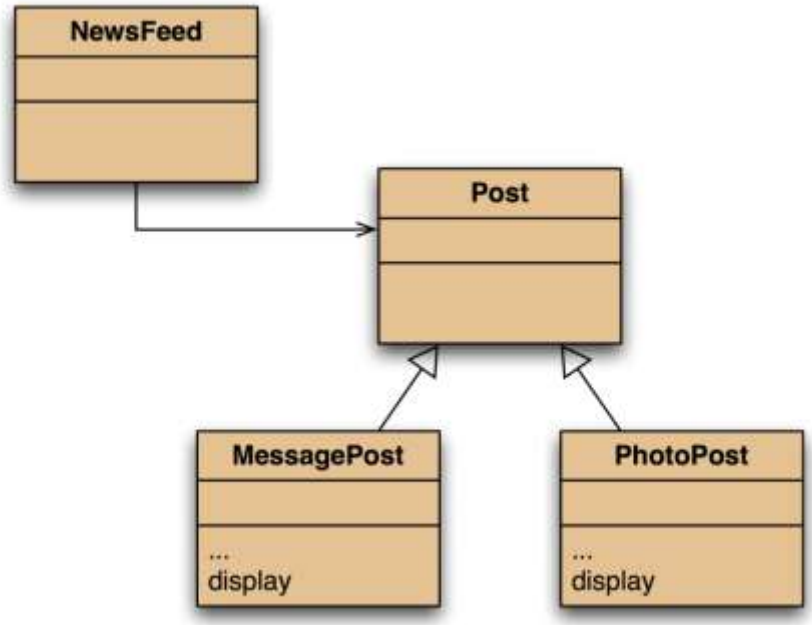
```
 Leonardo da Vinci
 40 seconds ago  -  2 people like this.
   No comments.

 Alexander Graham Bell
 12 minutes ago  -  4 people like this.
   No comments.
```

خروجی‌ای که داریم

مشکل کجاست؟

- متد **`display`** در **`Post`** فقط می‌تواند فیلدهای مشترک را چاپ کند.
- وراثت مثل یک خیابان یکطرفه است:
  - زیرکلاس فیلدهای ابرکلاس را به ارث می‌برد.
  - ابرکلاس چیزی در مورد فیلدهای زیرکلاس‌های خود نمی‌داند.
    - بنابراین نمی‌تواند آنها را چاپ کند.

اقدام برای حل مشکل



- متد **`display`** را در جایی قرار دهید که به اطلاعات مورد نیاز خود دسترسی داشته باشد:
  - هر زیرکلاس نسخه مخصوص به خود را خواهد داشت.
- اما

1. فیلدهای کلاس `Post` از نوع `private` هستند. و زیرکلاس به آنها دسترسی نخواهند داشت.
2. کلاس `NewsFeed` نمی‌تواند یک متد `display` را در `Post` پیدا کند (چرایی در ادامه...).

نوع‌های ایستا و پویا (Static type and dynamic type)

- نوع‌های سلسله مراتبی پیچیده‌تر نیاز به مفاهیم بیشتری برای توصیف دارند.
- چند اصطلاح جدید
  - نوع ایستا (static type)
  - نوع پویا (dynamic type)
  - ارسال/ارجاع به متد (method dispatch/lookup)

نوع‌های ایستا و پویا

```
Car c1 = new Car();
```

c1 از چه نوعی است؟

```
Vehicle v1 = new Car();
```

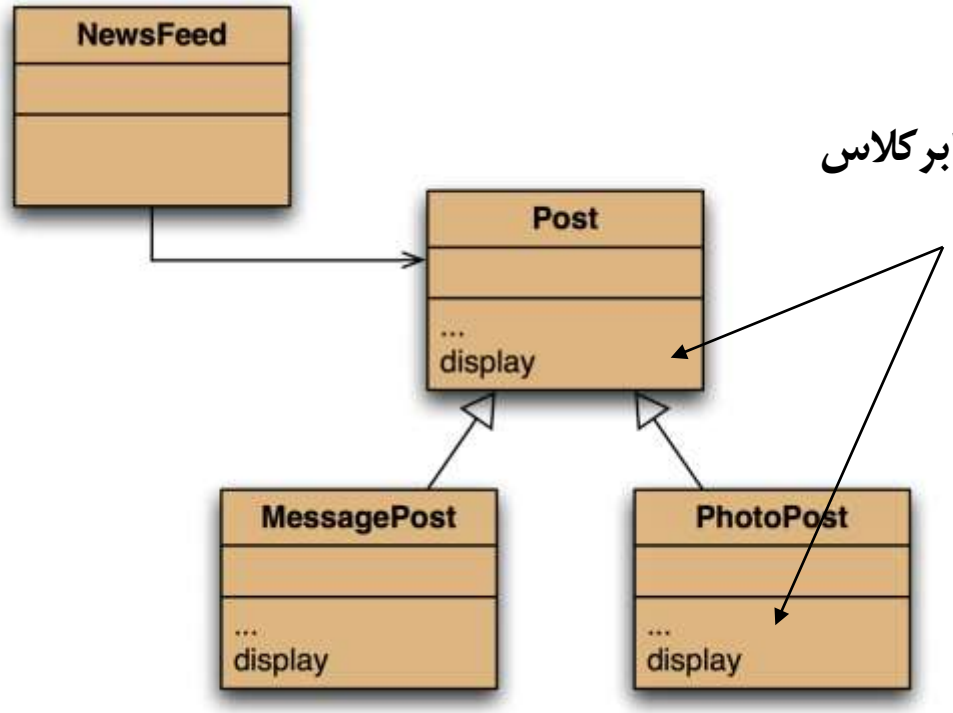
v1 از چه نوعی است؟

نوع‌های ایستا و پویا

- نوع اعلان شده متغیر، نوع ایستای آن است.
- نوع شی‌ای که متغیر به آن اشاره می‌کند، نوع پویای آن است.
- کامپایلر نقض نوع ایستا را بررسی می‌کند.

```
for(Post post : posts) {
   post.display();   // Compile-time error
                     // display() not defined in Post
}
```

راه حل: Overriding



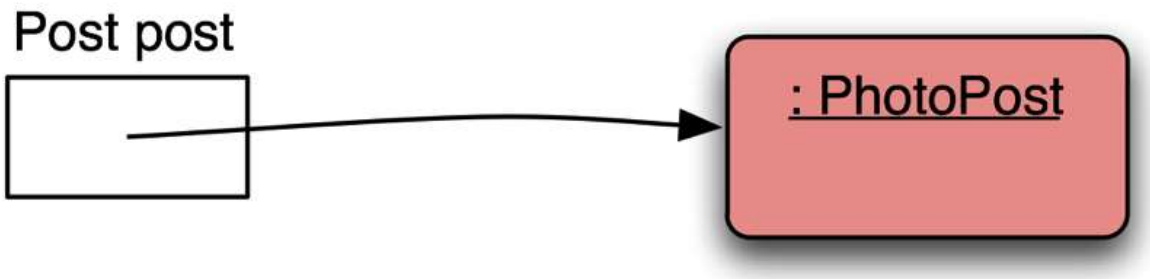
متد print هم در ابرکلاس
و هم در زیرکلاس

هم بررسی نوع ایستا
و هم پویا را برآورده
می‌کند.

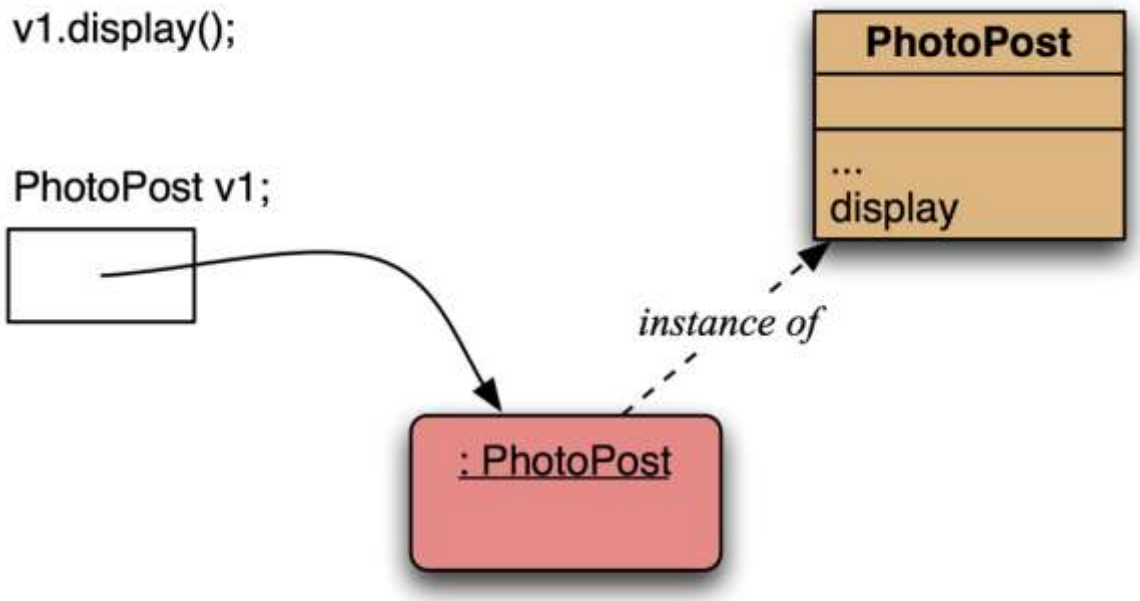
بازنویسی (Override) کردن متدها

- ابرکلاس و زیرکلاس هردو متدهای با امضای مشابه تعریف می‌کنند. (اسم و پارامترها)
- هرکدام به فیلدهای کلاس خود دسترسی دارند.
- ابرکلاس بررسی نوع ایستا را برآورده می‌کند.
- متد زیرکلاس هنگام اجرا فراخوانی می‌شود.
  - نسخه ابرکلاس را بازنویسی می‌کند.
- چه بر سر نسخه موجود در ابرکلاس می‌آید؟

تمایز نوع پویا از ایستا

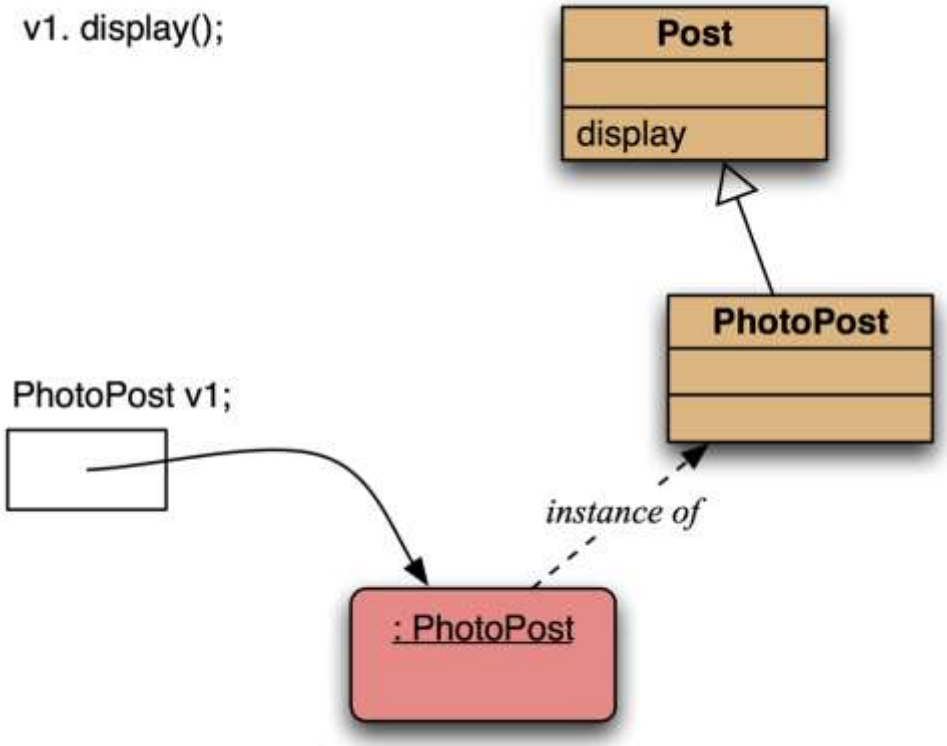


مراجعه به متد



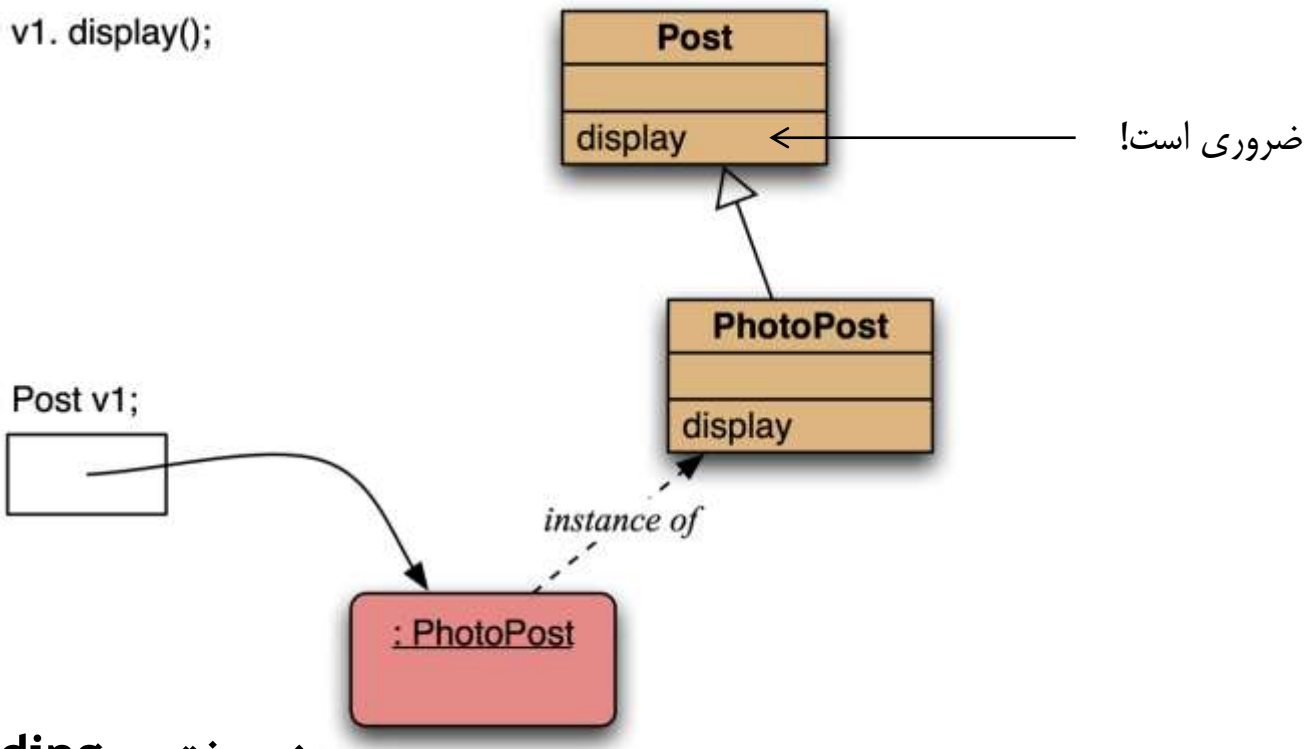
وراثت یا چندریختی وجود ندارد.
متد مشهود انتخاب می‌شود.

مراجعه به متد



وراثت اما بدون overriding.
سلسله مراتب وراثت به طور صعودی برای پیدا کردن متدی که جور شود جستجو می‌شود.

مراجعه به متد



چندریختی و overriding.
اولین نسخه‌ای که پیدا شود اجرا می‌شود.

مراجعه به متد ـ خلاصه

- متغیر دیده می‌شود
- شی ذخیره شده در متغیر پیدا می‌شود
- کلاس شی پیدا می‌شود
- کلاس برای پیدا کردن متد جستجو می‌شود
- اگر متد پیدا نشد، ابرکلاس جستجو می‌شود
- این جستجو تا پیدا کردن متد در ابرکلاس‌ها ادامه می‌یابد
- متدهای بازنویسی شده در اولویت هستند

فراخوانی `super` در متدها

- متدهای بازنویسی شده پنهان هستند.
- ... اما معمولا نیاز به فراخوانی آنها داریم.
- یک متد بازنویسی شده می‌تواند توسط متدی که آنرا بازنویسی کرده فراخوانی شود.
  - `super.method(...)`
    - مقایسه کنید با استفاده از `super` در سازنده‌ها.

فراخوانی متد بازنویسی شده

```
public void display()
{
    super.display();
    System.out.println(" [" +
                       filename +
                       "]");
    System.out.println(" " + caption);
}
```

چند ریختی متد

- در مورد مراجعه به متدهای چندریخت توضیح داده‌ایم.
- یک متغیر چند ریخت می‌تواند اشیایی از نوع‌های مختلف را ذخیره کند.
- فراخوانی متدها چندریخت است.
  - متدی که واقعا فراخوانی می‌شود بستگی به نوع پویای متد دارد.

اپراتور `instanceof`

- برای تشخیص نوع پویا مورد استفاده قرار می‌گیرد.
- اطلاعات نوع گمشده را بازیابی می‌کند.
- معمولا قبل از واگذاری (Casting) به نوع پویا مورد استفاده قرار می‌گیرد.

```
if(post instanceof MessagePost) {
    MessagePost msg =
        (MessagePost) post;
    ... access MessagePost methods via msg ...
}
```

متدهای کلاس `Object`

- متدهای درون `Object` توسط تمام کلاس‌ها به ارث برده می‌شود.
- هرکدام از آنها می‌توانند بازنویسی شوند.
- متد `toString` معمولا بازنویسی می‌شود.

– `public String toString()`

  – یک نمایش رشته‌ای از شی را باز می‌گرداند.

بازنویسی متد toString برای Post

```
public String toString()
{
    String text = username + "\n" +
                  timeString(timestamp);
    if(likes > 0) {
        text += " - " + likes + " people like this.\n";
    }
    else {
        text += "\n";
    }
    if(comments.isEmpty()) {
        return text + " No comments.\n";
    }
    else {
        return text + " " + comments.size() +
               " comment(s). Click here to view.\n";
    }
}
```

بازنویسی متد `toString`

- فراخوانی صریح برای چاپ می‌تواند حذف شود.

```
System.out.println(post.toString());
```

- فراخوانی **`println`** با فقط یک شی، به طور خودکار منجر به فراخوانی **`toString`** می‌شود.

```
System.out.println(post);
```

StringBuilder

- استفاده از StringBuilder برای الحاق را در نظر بگیرید:

```
StringBuilder builder = new StringBuilder();
builder.append(username);
builder.append('\n');
builder.append(timeString(timestamp));
…
return builder.toString();
```

برابری اشیا

- برابر بودن دو شی به چه معنا است؟
  - برابر بودن مرجع
  - برابر بودن محتوا
- مورد استفاده == و `equals()` در رشته‌ها را مقایسه کنید.

بازنویسی `equals`

```
public boolean equals(Object obj)
{
    if(this == obj) {
        return true;
    }
    if(!(obj instanceof ThisType)) {
        return false;
    }
    ThisType other = (ThisType) obj;
    … compare fields of this and other
}
```

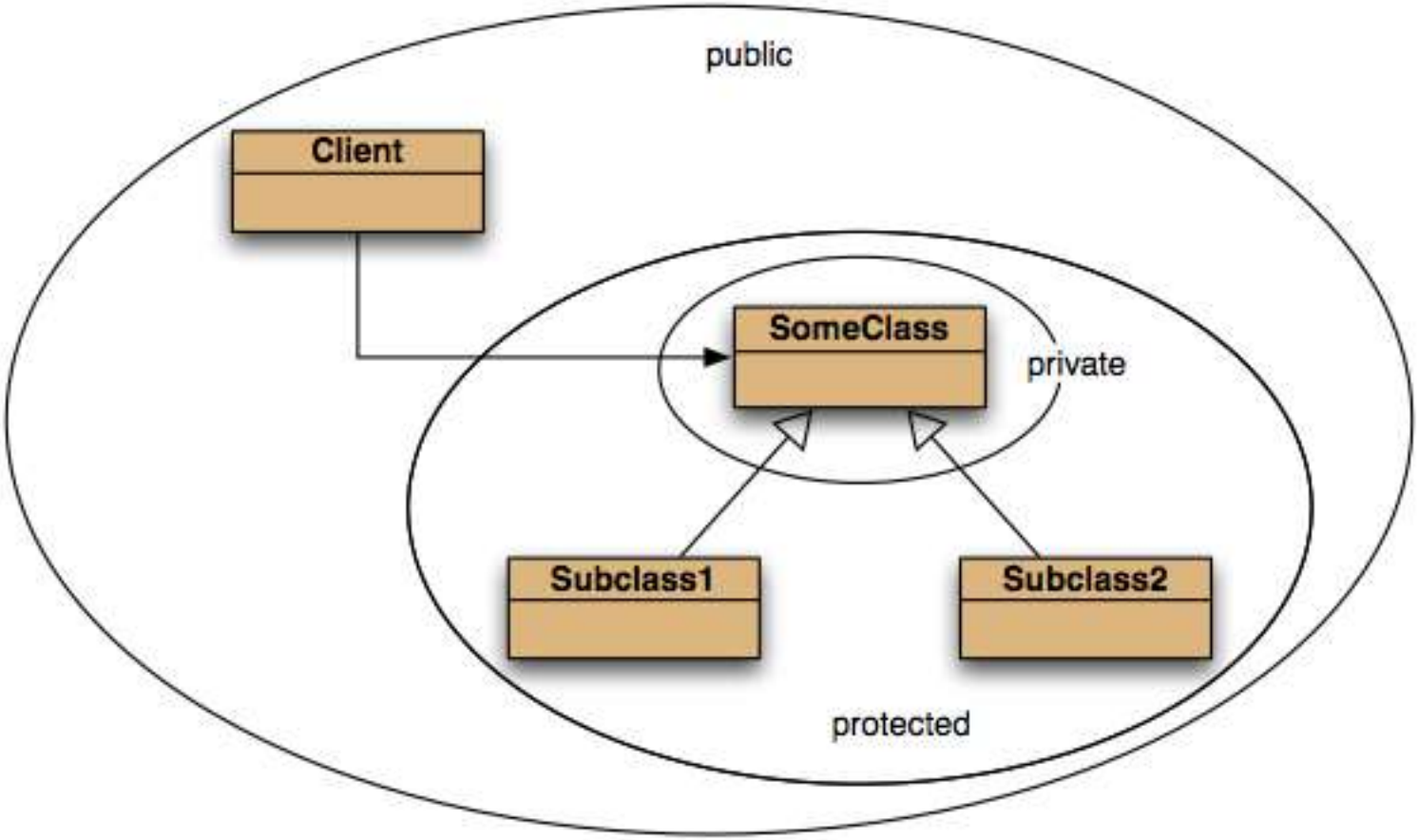
بازنویسی `equals` در `Students`

```
public boolean equals(Object obj)
{
    if(this == obj) {
        return true;
    }
    if(!(obj instanceof Student)) {
        return false;
    }
    Student other = (Student) obj;
    return name.equals(other.name) && id.equals(other.id);
}
```

دسترسی محافظت شده (Protected)

- دسترسی `private` در ابرکلاس ممکن است برای زیرکلاس خیلی محدود کننده باشد.
- یک رابطه وراثت نزدیک‌تر به وسلیه دسترسی محافظت شده (`protected`) پشتیبانی می‌شود.
  - به خودش و زیرکلاس‌هایش اجازه دسترسی می‌دهد.
  - محدودتر از دسترسی عمومی (`public`).
  - یک سطح میانی در پنهان‌سازی داده‌ها.
- همچنان پیشنهاد می‌کنیم فیلدها را `private` نگه داریم.
  - از متدهای محافظت شده `get` و `set` برای استفاده زیرکلاس‌ها استفاده کنید.

سطوح دسترسی


public
Client
SomeClass
private
Subclass1
Subclass2
protected

مرور

- نوع اعلان شده برای یک متغیر، نوع ایستای آن است.
  - کامپایلر نوع ایستا را بررسی می‌کند.
- نوعی که شی است، نوع پویای آن است.
  - نوع پویا موقع اجرا استفاده می‌شود.
- متدها ممکن است در زیرکلاس‌ها بازنویسی شوند.
- جستجو برای مراجعه به متد با نوع پویای آن شروع می‌شود.
- دسترسی محافظت شده در وراثت پشتیبانی می‌شود.

این جلسه

مفاهیم

- Method polymorphism
- Static and dynamic type
- Overriding
- Dynamic method lookup
- Protected visibility modifier

جاوا

- super (in method)
- protected
- toString
- equals