

به نام پروردگار دانایی

طراحی سیستم‌های شی‌گرا

برنامه‌نویسی شی‌گرا

درس ششم: بیشتر در مورد وراثت (بررسی چندریختی)

سید کاوه احمدی

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

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

مفاهیم این جلسه

■ نوع‌های ایستا و پویا

– نوع ایستا اجازه بررسی زمان کامپایل را فراهم می‌کند.

– نوع پویا اجازه چندریختی متدها هنگام اجرا را فراهم می‌کند.

■ چندریختی متدها

– رفتارهای متفاوت با فراخوانی متد مشابه با توجه به نوع پویای شی

■ بازنویسی (Override) کردن متدهای به ارث برده شده.

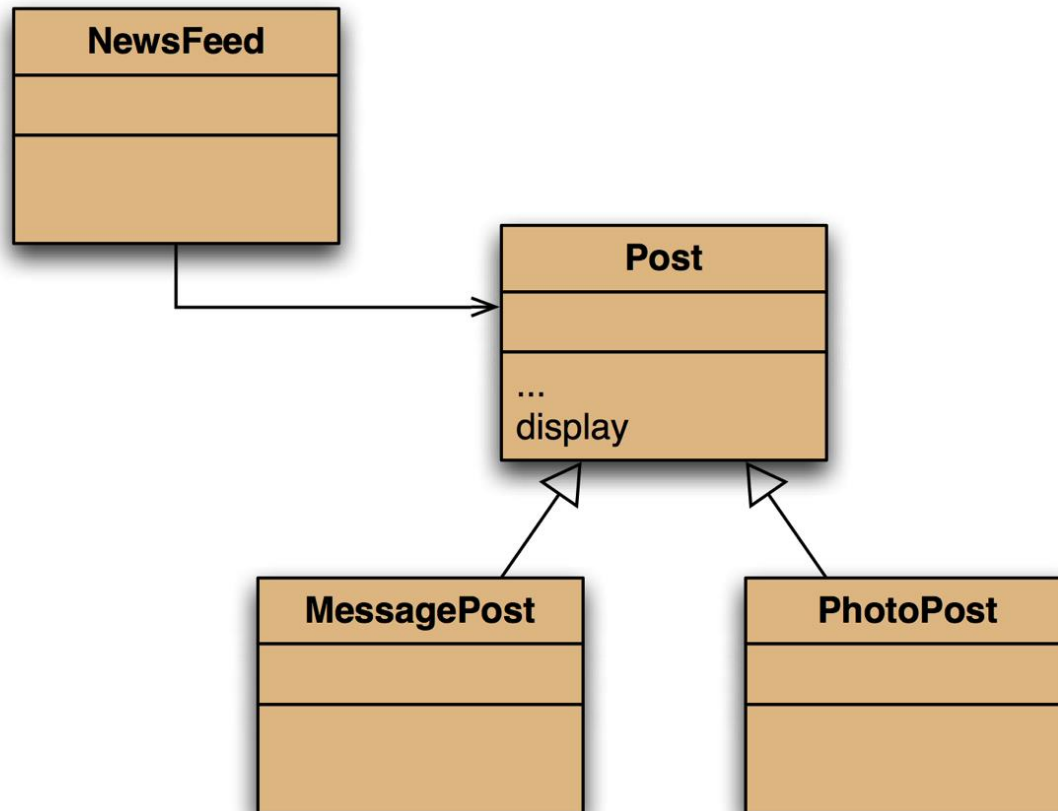
■ جستجوی پویای متد

■ 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 ...
    }
}
```



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.

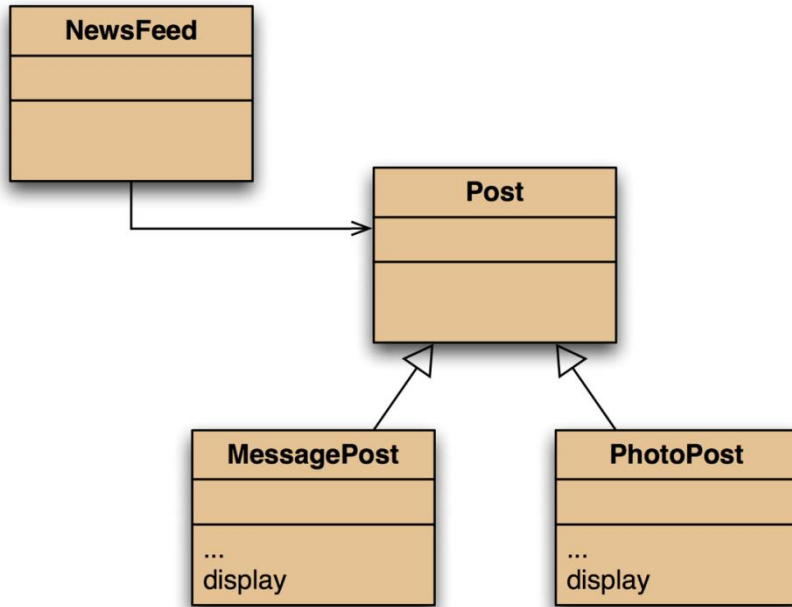
Alexander Graham Bell
12 minutes ago - 4 people like this.
No comments.

خروجی ای 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 زیر کلاس نسخه مخصوص به خود را خواهد داشت.

- اما

1. فیلدهای کلاس **Post** از نوع **private** هستند. و زیر کلاس به آنها دسترسی نخواهند داشت.

2. کلاس **NewsFeed** نمی تواند 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؟



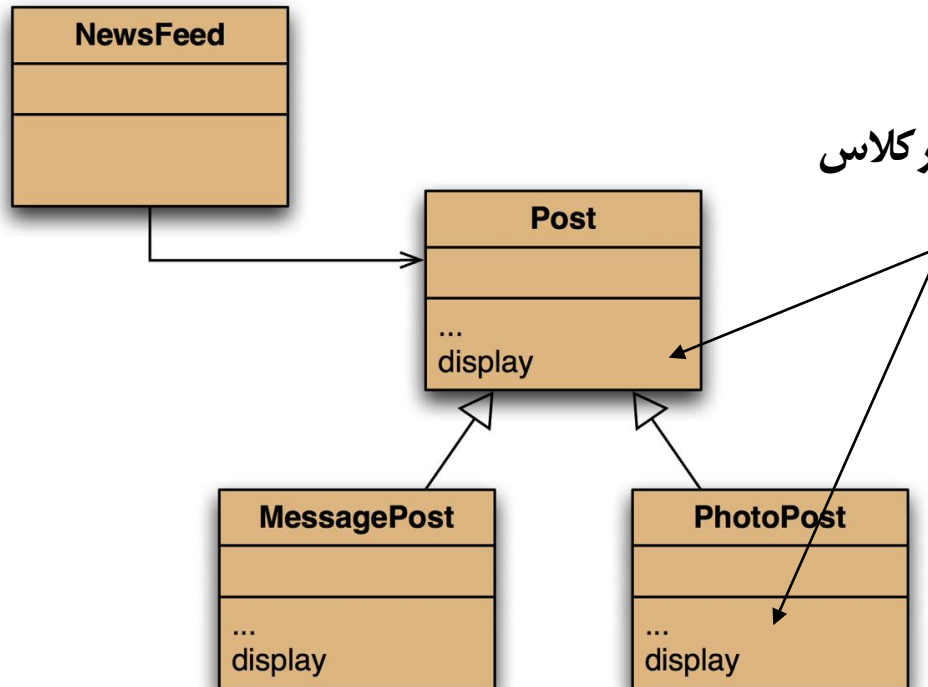
نوع متغیر **Vehicle** است. نوع شی ذخیره شده در متغیر **Car** است.

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 ابر کلاس
و هم در زیر کلاس

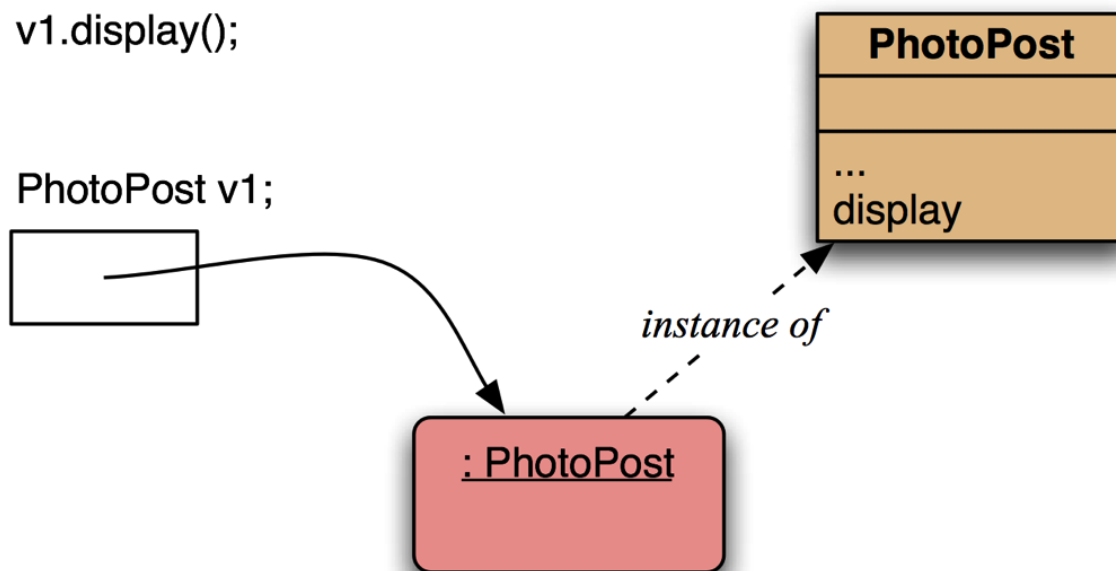
هم بررسی نوع ایستا
و هم پویا را برآورده
می کند.

بازنویسی (Override) کردن متدها

- ابرکلاس و زیرکلاس هر دو متدهای با امضای مشابه تعریف می کنند. (اسم و پارامترها)
- هرکدام به فیلهای کلاس خود دسترسی دارند.
- ابرکلاس بررسی نوع ایستا را برآورده می کند.
- متد زیرکلاس هنگام اجرا فراخوانی می شود.
 - نسخه ابرکلاس را بازنویسی می کند.
- چه بر سر نسخه موجود در ابرکلاس می آید؟

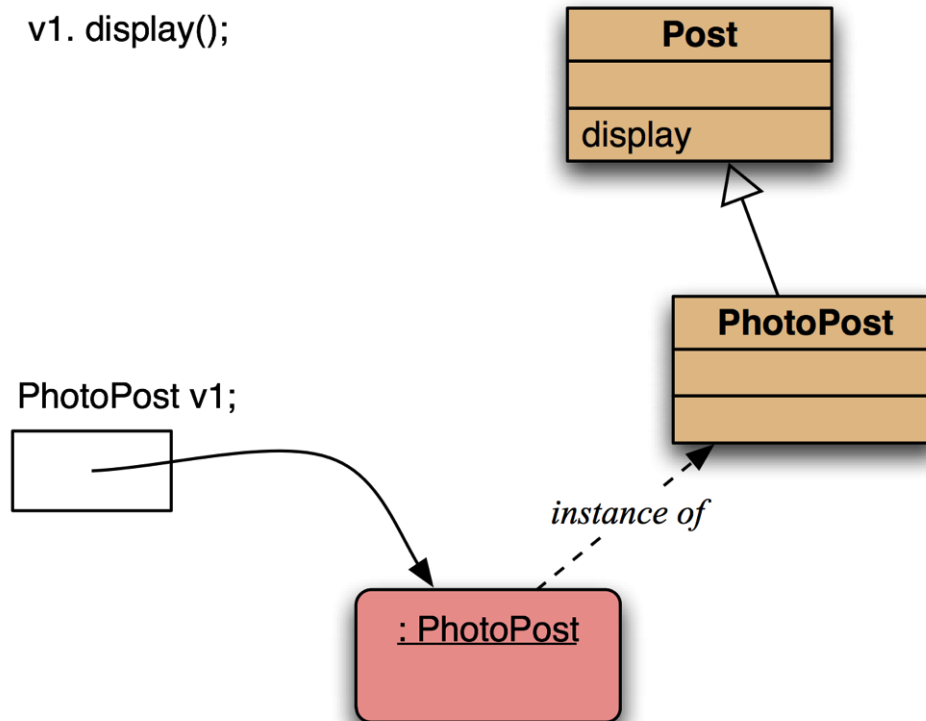
تمایز نوع پویا از ایستا





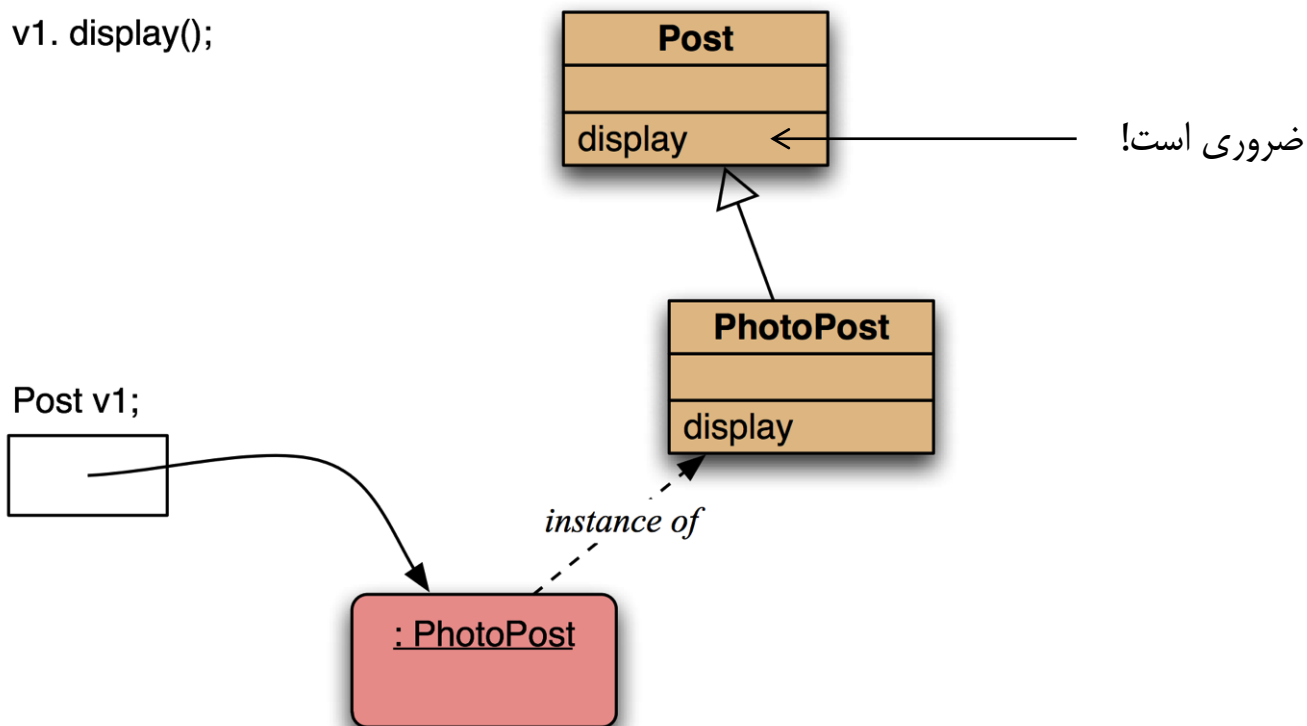
وراثت یا چندریختی وجود ندارد.
متد مشهود انتخاب می شود.

v1. display();



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

v1. display();



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

مراجعه به متد - خلاصه

- متغیر دیده می شود
- شی ذخیره شده در متغیر پیدا می شود
- کلاس شی پیدا می شود
- کلاس برای پیدا کردن متد جستجو می شود
- اگر متد پیدا نشد، ابر کلاس جستجو می شود
- این جستجو تا پیدا کردن متد در ابر کلاس ها ادامه می یابد
- 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 می تواند 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.
- معمولاً 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 معمولاً بازنویسی می‌شود.

– `public String toString()`

– یک نمایش رشته‌ای از شی را باز می‌گرداند.

بازنویسی متد toString

```
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.

equals بازنویسی

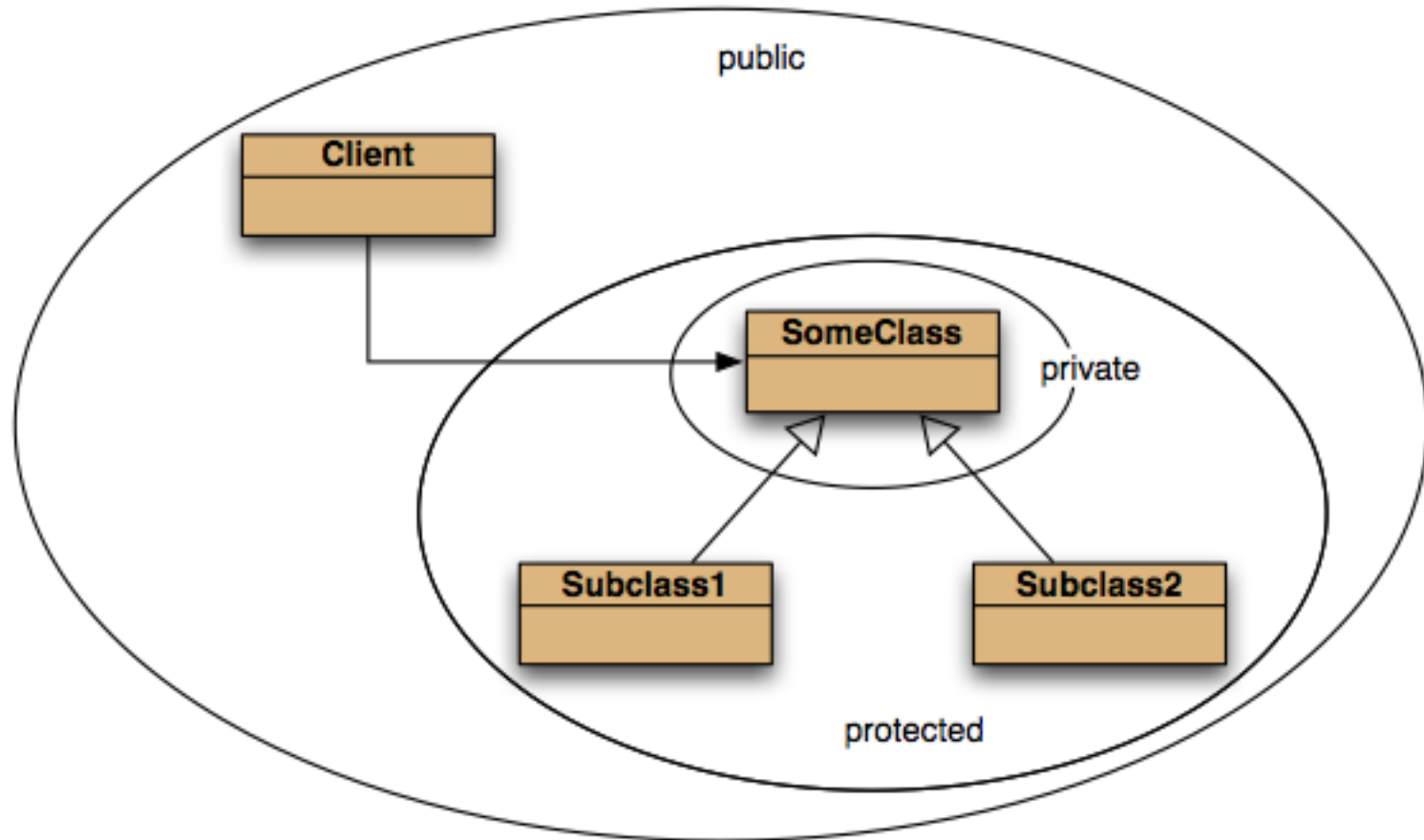
```
public boolean equals(Object obj)
{
    if(this == obj) {
        return true;
    }
    if(!(obj instanceof ThisType)) {
        return false;
    }
    ThisType other = (ThisType) obj;
    ... compare fields of this and other
}
```

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 ابر کلاس ممکن است برای زیر کلاس خیلی محدود کننده باشد.
- یک رابطه وراثت نزدیک تر به وسیله دسترسی محافظت شده (`protected`) پشتیبانی می شود.
- به خودش و زیر کلاس هایش اجازه دسترسی می دهد.
- محدودتر از دسترسی عمومی (`public`).
- یک سطح میانی در پنهان سازی داده ها.
- همچنان پیشنهاد می کنیم فیلدها را `private` نگه داریم.
- از متدهای محافظت شده `get` و `set` برای استفاده زیر کلاس ها استفاده کنید.



- نوع اعلان شده برای یک متغیر، نوع ایستای آن است.

- کامپایلر نوع ایستا را بررسی می کند.

- نوعی که شی است، نوع پویای آن است.

- نوع پویا موقع اجرا استفاده می شود.

- متدها ممکن است در زیر کلاس ها بازنویسی شوند.

- جستجو برای مراجعه به متد با نوع پویای آن شروع می شود.

- دسترسی محافظت شده در وراثت پشتیبانی می شود.

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

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