به نام پروردگار دانایی

طراحی سیستم‌های شی‌گرا

برنامه‌نویسی شی‌گرا

درس چهارم: بهبود ساختارها با استفاده از وراثت

سید کاوه احمدی

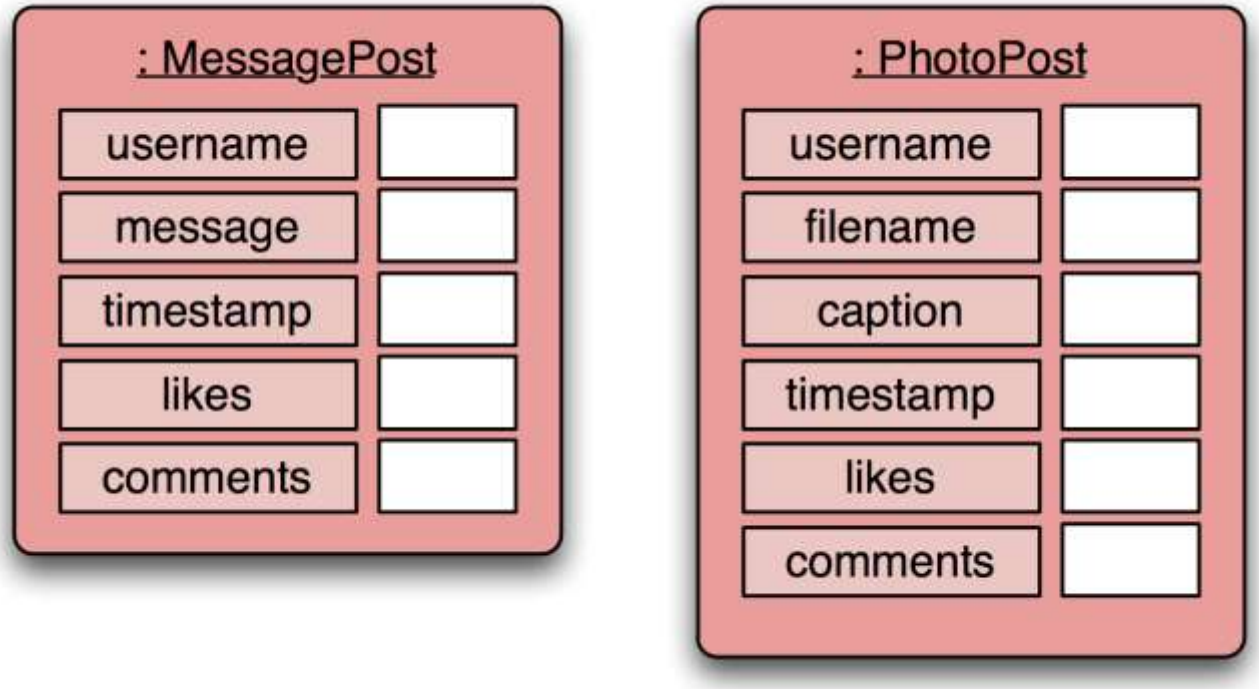
مفاهیم این جلسه

- وراثت (Inheritance)
  - یک زیرکلاس، فیلدها، متدها و نوع ابرکلاس را به ارث می‌برد.
- زیرنوع (Subtyping)
  - یک زیرکلاس یک زیرنوع از ابرکلاس خود است.
  - جایگزینی (substitution)
- متغیرهای چندریخت (Polymorphic variables)
  - متغیرهایی که می‌توانند نوع‌های مختلفی بگیرند (جایگزینی با زیرنوع)
- تغییر نوع، wrapper classes و autoboxing.

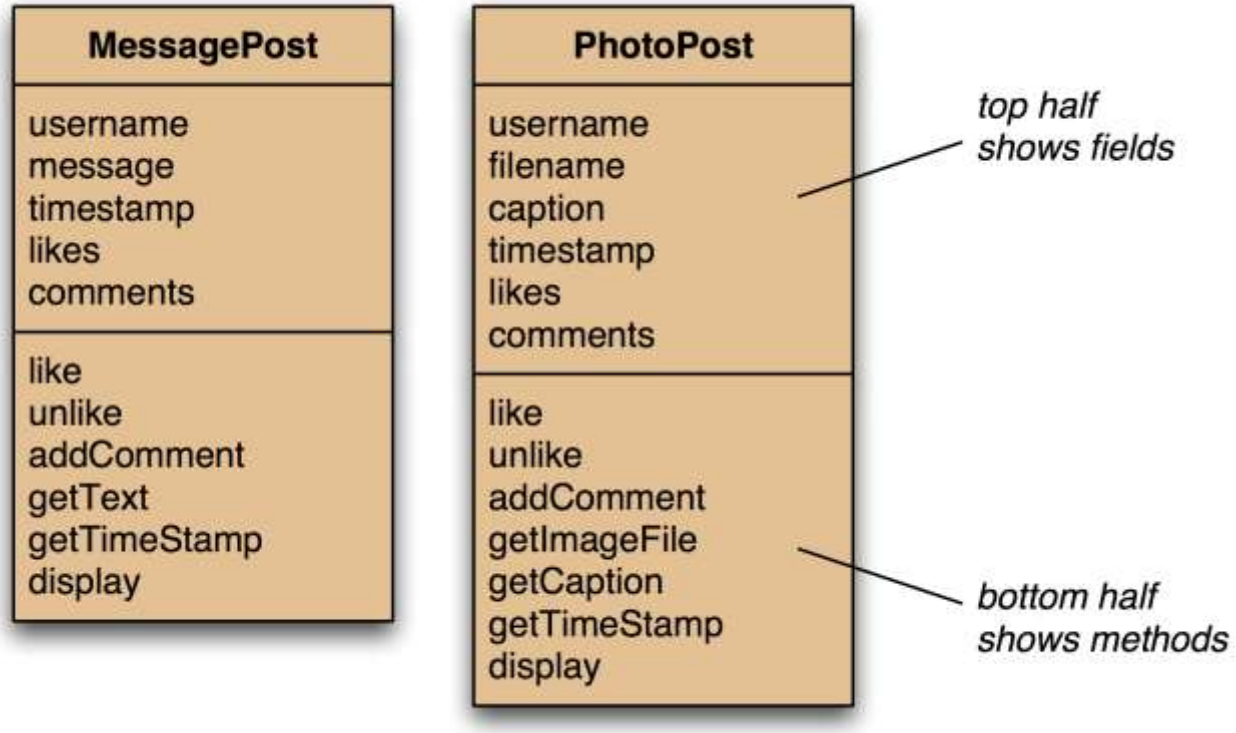
مثال شبکه اجتماعی

- یک نمونه اولیه از یک شبکه اجتماعی کوچک
- مشتمل بر پست‌های متنی و تصویری:
  - `MessagePost`: متن‌های چند خطی
  - `PhotoPost`: یک تصویر و یک عنوان (`caption`) برای آن
- اجازه برخی عملکردها بر پست‌ها
  - همانند جستجو، نمایش و حذف

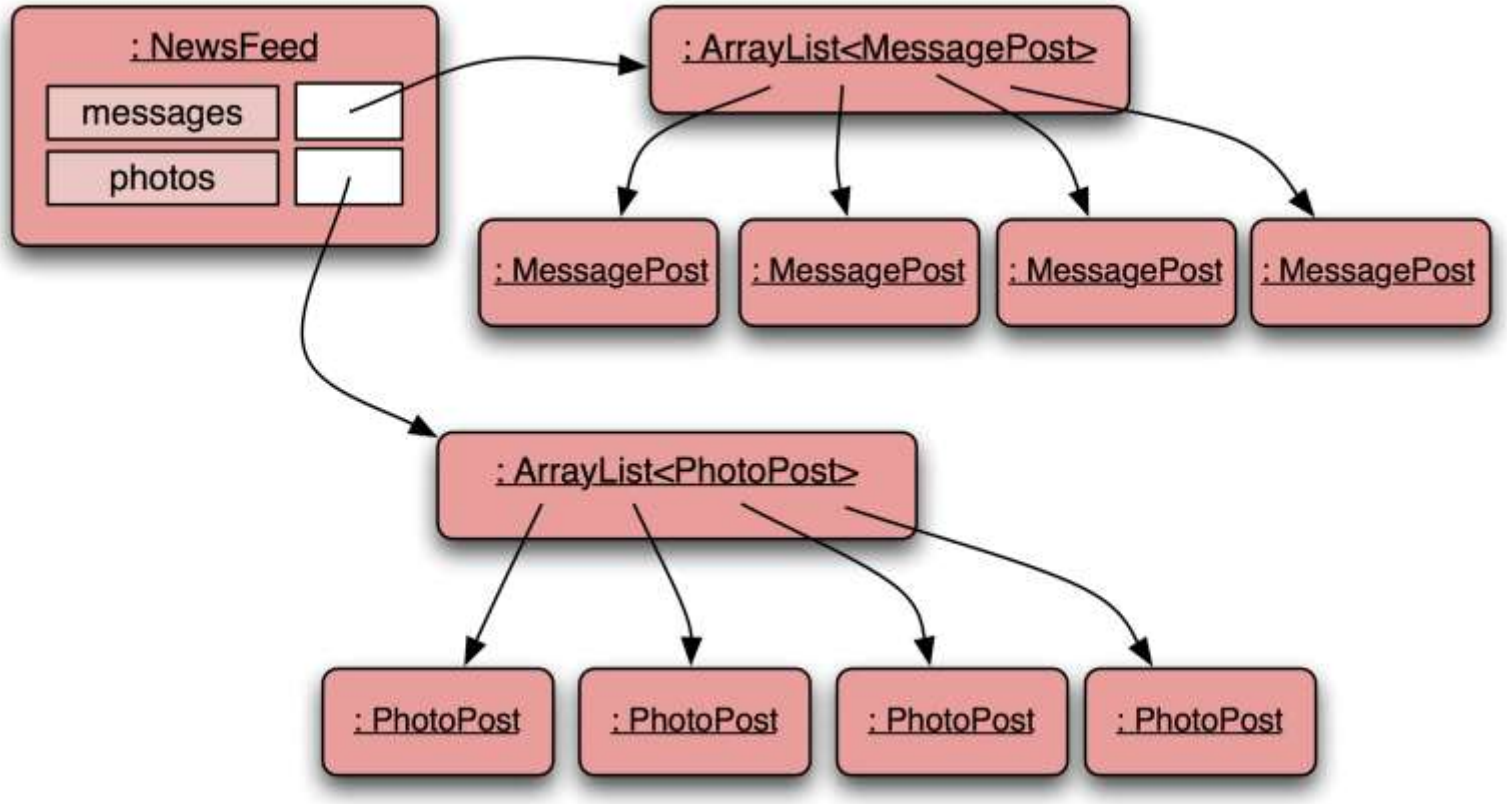
اشیای شبکه


: MessagePost
username
message
timestamp
likes
comments
: PhotoPost
username
filename
caption
timestamp
likes
comments

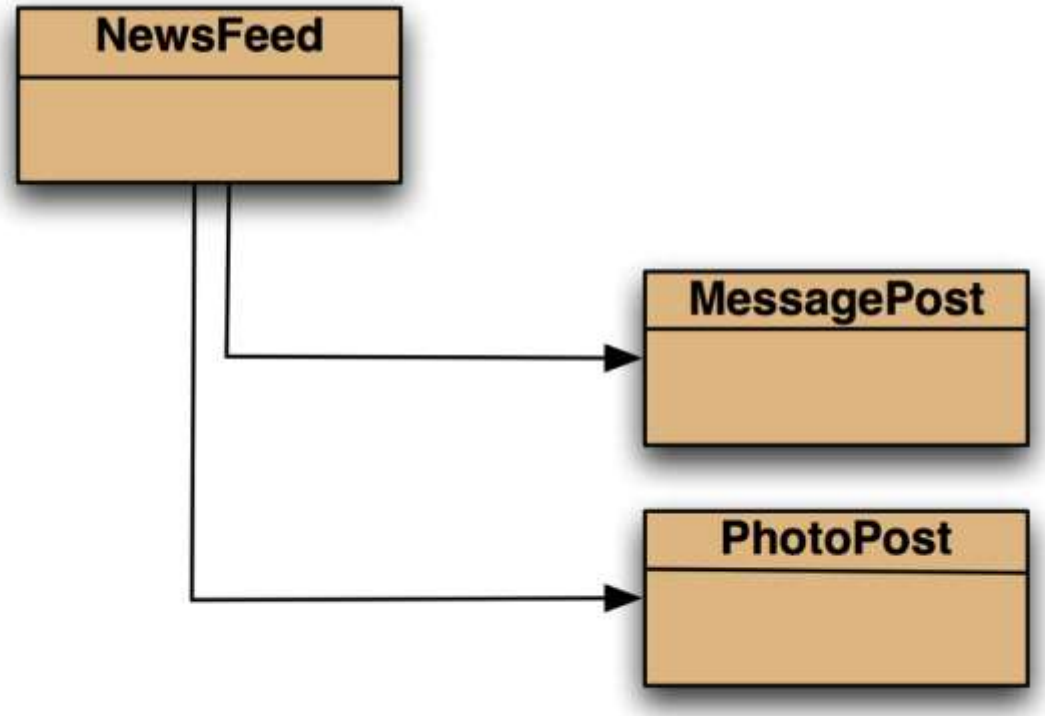
کلاس‌های شبکه



مدل اشیای شبکه



دیاگرام کلاس



به فقدان جزئیات توجه داشته باشید: کتابخانه‌های استاندارد (کلاس‌های کتابخانه‌ای) نمایش داده نشده‌اند.

بخشی از کد کلاس MessagePost

```
public class MessagePost
{
   private String username;
   private String message;
   private long timestamp;
   private int likes;
   private ArrayList<String> comments;

   public MessagePost(String author, String text)
   {
         username = author;
         message = text;
         timestamp = System.currentTimeMillis();
         likes = 0;
         comments = new ArrayList<String>();
   }

   public void addComment(String text) ...

   public void like() ...

   public void display() ...

   ...
}
```

بخشی از کد کلاس PhotoPost

```
public class PhotoPost
{
   private String username;
   private String filename;
   private String caption;
   private long timestamp;
   private int likes;
   private ArrayList<String> comments;

   public PhotoPost(String author, String filename,
                    String caption)
   {
         username = author;
         this.filename = filename;
         this.caption = caption;
         timestamp = System.currentTimeMillis();
         likes = 0;
         comments = new ArrayList<String>();
   }

   public void addComment(String text) ...
   public void like() …
   public void display() …
   ...
}
```

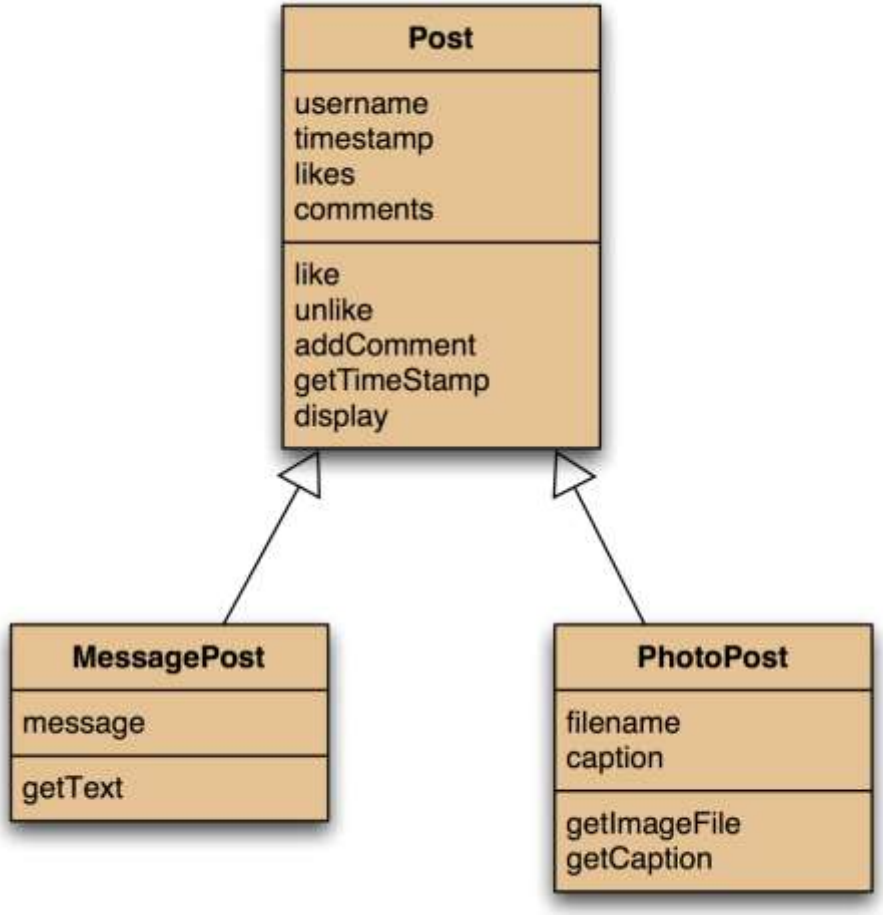
NewsFeed کلاس

```
public class NewsFeed
{
   private ArrayList<MessagePost> messages;
   private ArrayList<PhotoPost> photos;
   ...
   public void show()
   {
       for(MessagePost message : messages) {
          message.display();
          System.out.println();  // empty line between posts
       }

       for(PhotoPost photo : photos) {
          photo.display();
          System.out.println();  // empty line between posts
       }
   }
}
```

انتقاد به Network

- تکرار کد
  - کلاس‌های MessagePost و PhotoPost بسیار مشابه هستند. (بخش اعظمی از پیاده‌سازی آنها یکسان است).
  - اصلاح را سخت‌تر می‌کند / کار بیشتر
  - اصلاح نادرست می‌تواند منجر به ایجاد باگ‌هایی در سیستم شود.
- تکرار کد حتی در کلاس NewsFeed نیز وجود دارد.
  - تکرارها مشابه هم هستند و فقط روی فیلدهای متفاوتی اعمال می‌شوند

استفاده از وراثت

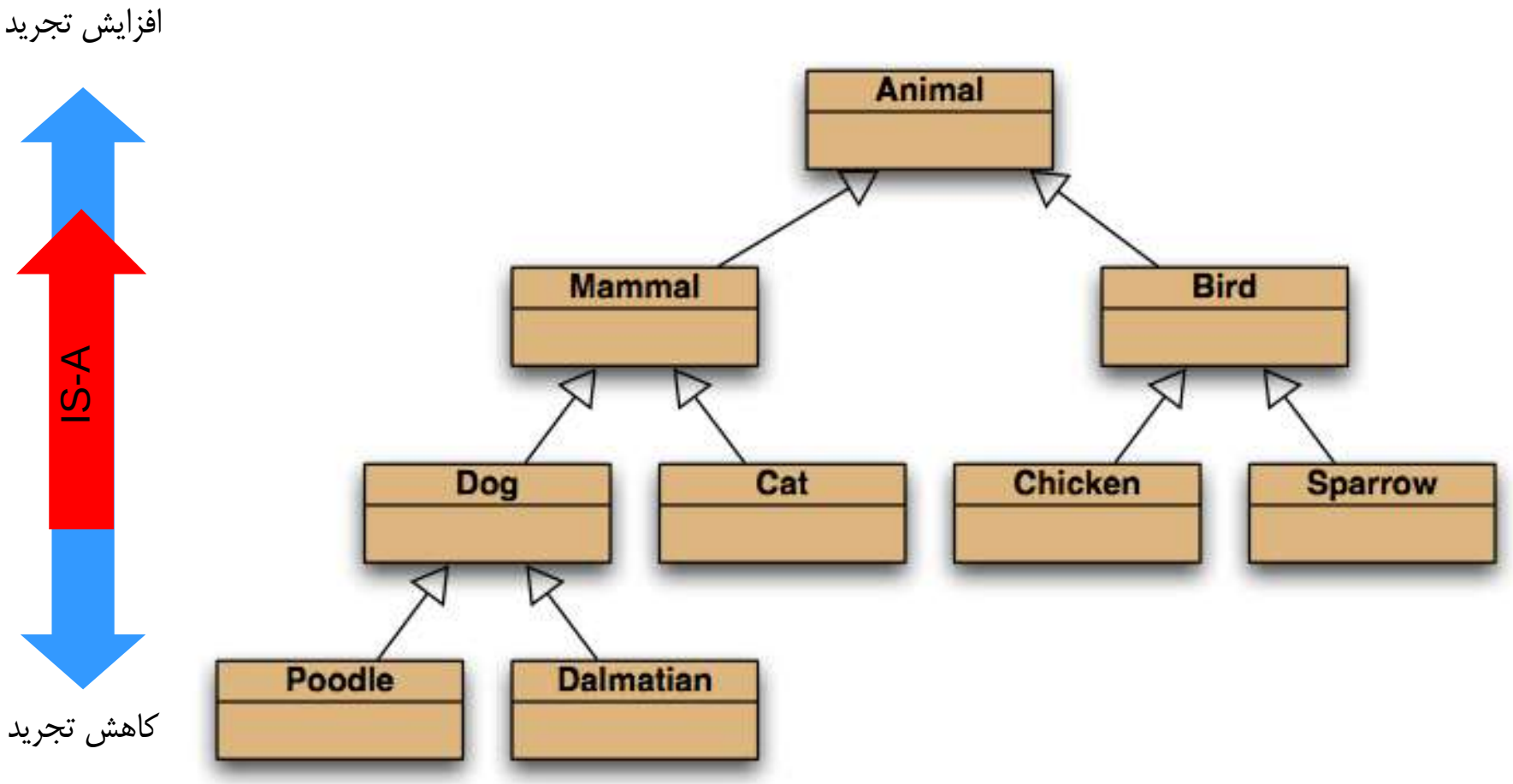


استفاده از وراثت

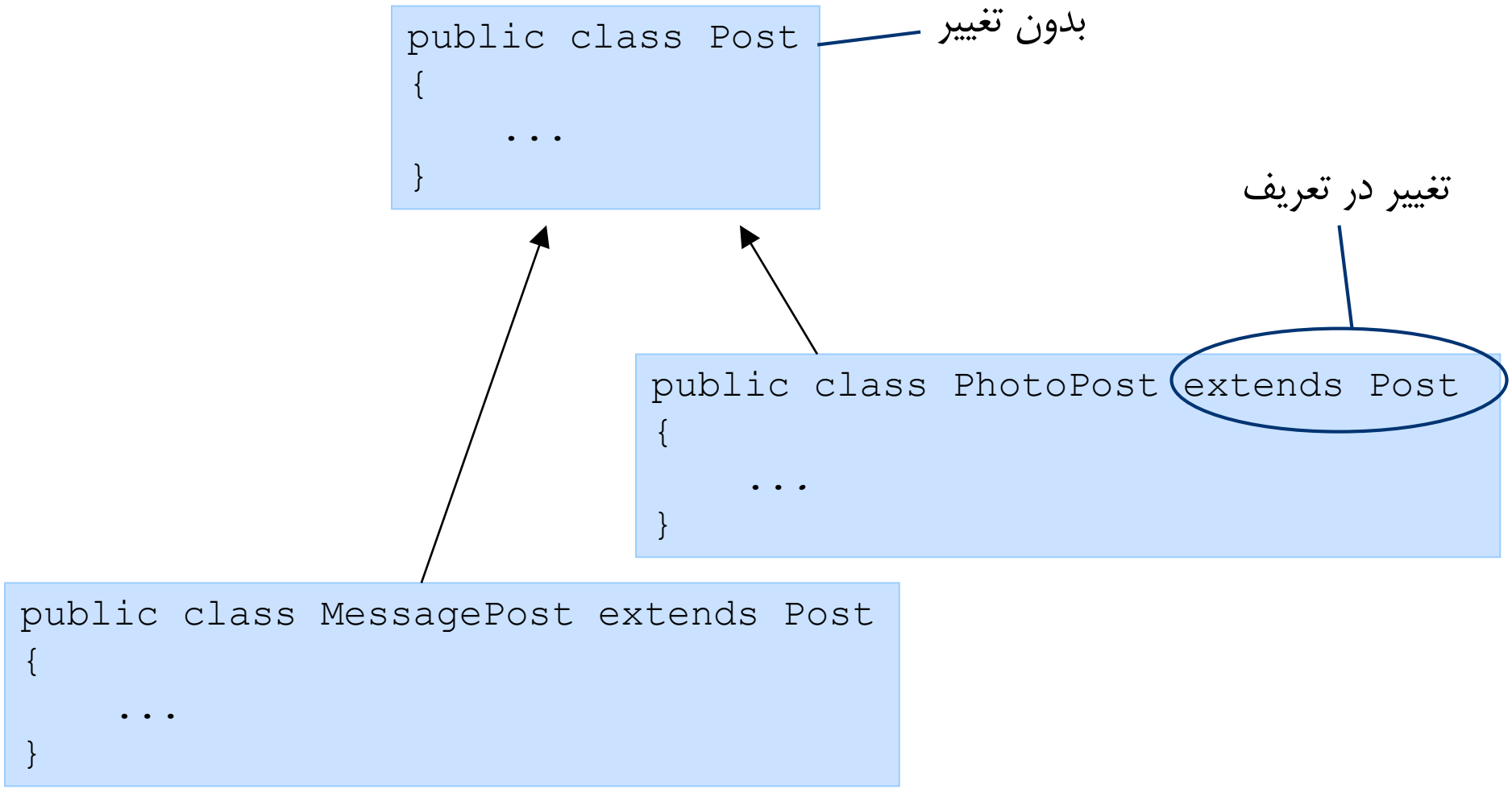
- تعریف یک ابرکلاس (superclass): **Post**
- تعریف زیرکلاس‌هایی (subclasses) برای MessagePost و PhotoPost
- ابر کلاس مشخصات مشترک را تعریف می‌کند (توسط فیلدها).
- زیرکلاس‌ها، خصوصیات ابرکلاس را به ارث می‌برند.
- زیرکلاس‌ها خصوصیات دیگری را اضافه می‌کنند.

- به ابرکلاس، کلاس پایه یا فوق کلاس هم گفته می‌شود.
- به زیرکلاس، کلاس مشتق شده هم گفته می‌شود.

سلسله مراتب وراثت



وراثت در جاوا



ابر کلاس

```
public class Post
{
    private String username;
    private long timestamp;
    private int likes;
    private ArrayList<String> comments;

    // constructor and methods omitted.
}
```

هیچ چیز در فایل `Post.java` به شما در مورد وجود زیرکلاس‌ها توضیحی نمی‌دهد.

زیرکلاس‌ها

```
public class MessagePost extends Post
{
    private String message;

    // constructor and methods omitted.
}
```

```
public class PhotoPost extends Post
{
    private String filename;
    private String caption;

    // constructor and methods omitted.
}
```

وراثت و سازنده‌ها

```
public class Post
{
    private String username;
    private long timestamp;
    private int likes;
    private ArrayList<String> comments;

    /**
     * Initialise the fields of the post.
     */
    public Post(String author)
    {
        username = author;
        timestamp = System.currentTimeMillis();
        likes = 0;
        comments = new ArrayList<String>();
    }

    // methods omitted
}
```

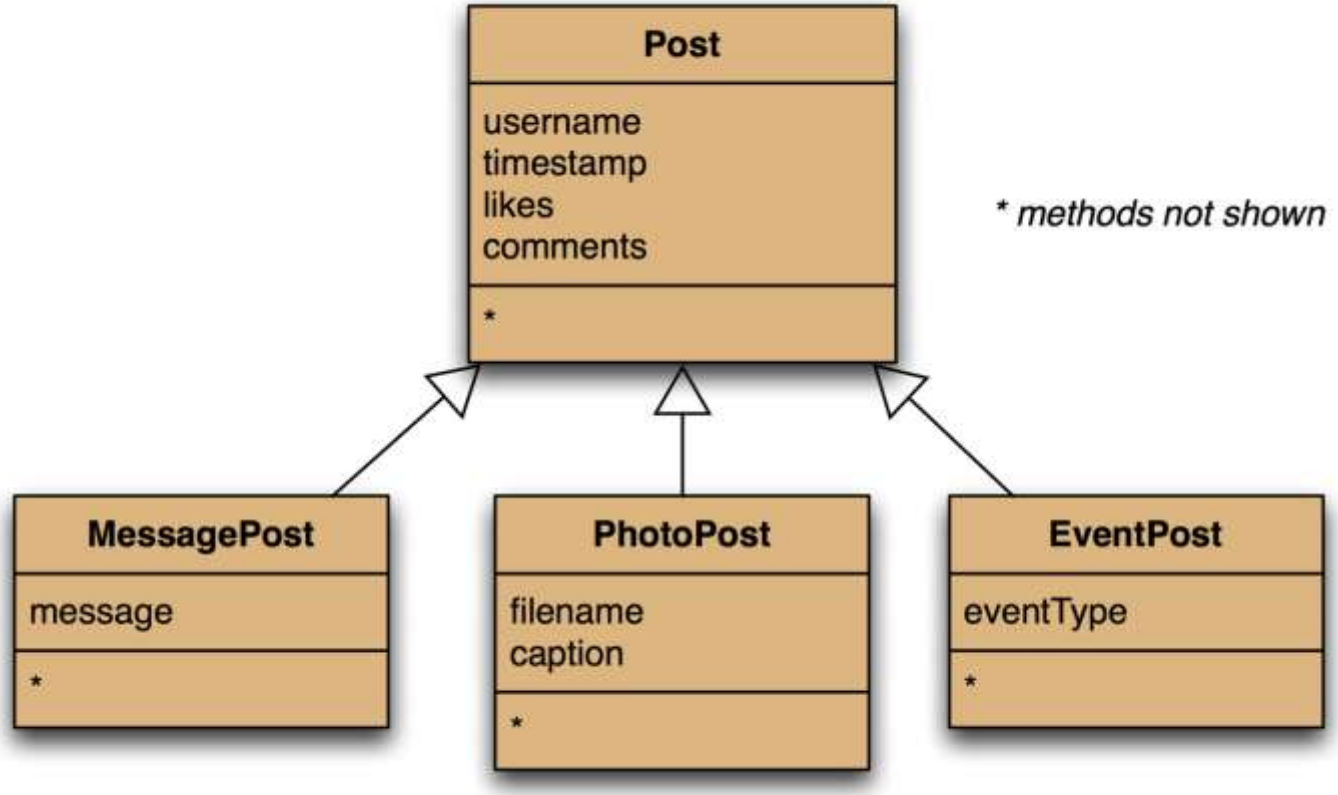
وراثت و سازنده‌ها

```
public class MessagePost extends Post
{
    private String message;

    /**
     * Constructor for objects of class MessagePost
     */
    public MessagePost(String author, String text)
    {
        super(author);
        message = text;
    }

    // methods omitted
}
```

فراخوانی سازنده سوپرکلاس

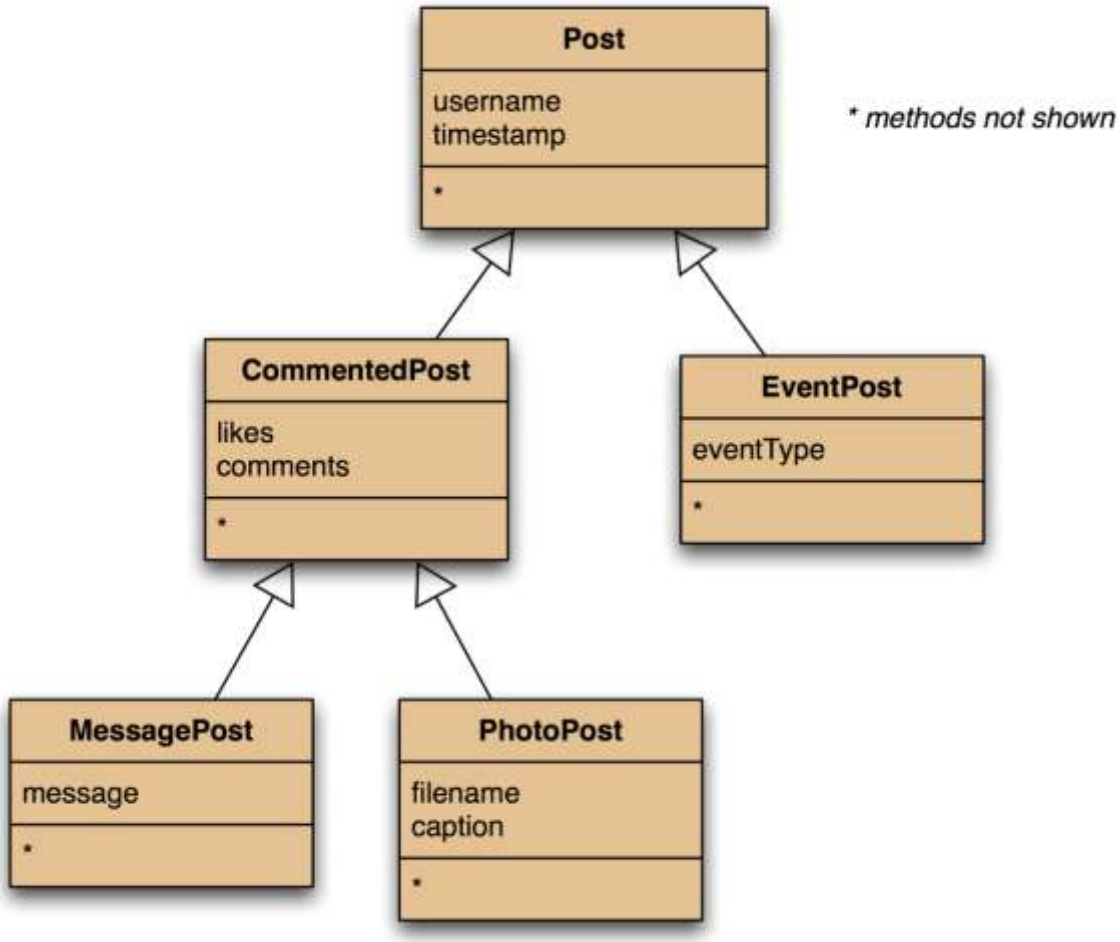
- سازنده‌های زیرنوع‌ها باید حتما شامل یک فراخوانی `'super'` باشند.
  - باید اولین دستور در سازنده زیرکلاس باشد.
- اگر نوشته نشوند، کامپایلر یکی اضافه می‌کند (بدون پارامتر).
  - فقط در صورتی کار می‌کند که ابرکلاس یک سازنده بدون پارامتر داشته باشد.

اضافه کردن موارد جدیدتر



Easy: inheritance lets us reuse existing code

سلسله مراتب با عمق بیشتر



با تغییر ساختار می‌توان فیلدها و متدهای به ارث برده شده توسط زیرکلاس‌ها را کنترل کرد.

مرور

- وراثت (تا کنون) کمک می‌کند تا:
  - جلوگیری از تکرار کدها
  - استفاده مجدد از کدها
  - اصلاح ساده‌تر
  - قابلیت گسترش

کد جدید کلاس NewsFeed

```
public class NewsFeed
{
    private ArrayList<Post> posts;

    /**
     * Construct an empty news feed.
     */
    public NewsFeed()
    {
        posts = new ArrayList<Post>();
    }

    /**
     * Add a post to the news feed.
     */
    public void addPost(Post post)
    {
        posts.add(post);
    }
    ...
}
```

- فقط یک ArrayList و یک متد addPost خواهیم داشت.
- به جای addMessage و addPhoto.
- وراثت از تکرار کدها در کلاس NewsFeed نیز جلوگیری می‌کند. (مقایسه با اسلاید ۱۰)

کد جدید کلاس NewsFeed

- با نسخه اولیه متد show مقایسه کنید.
- اینجا فقط یک حلقه تکرار داریم.

```
/**
 * 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 ...
    }
}
```

زیرنمونه‌ها

- در ابتدا داشتیم:

```
public void addMessagePost(MessagePost message)
public void addPhotoPost(PhotoPost photo)
```

- حالا داریم:

```
public void addPost(Post post)
```

- این متد را به صورت زیر فراخوانی می‌کنیم:

```
PhotoPost myPhoto = new PhotoPost(...);
feed.addPost(myPhoto);
```

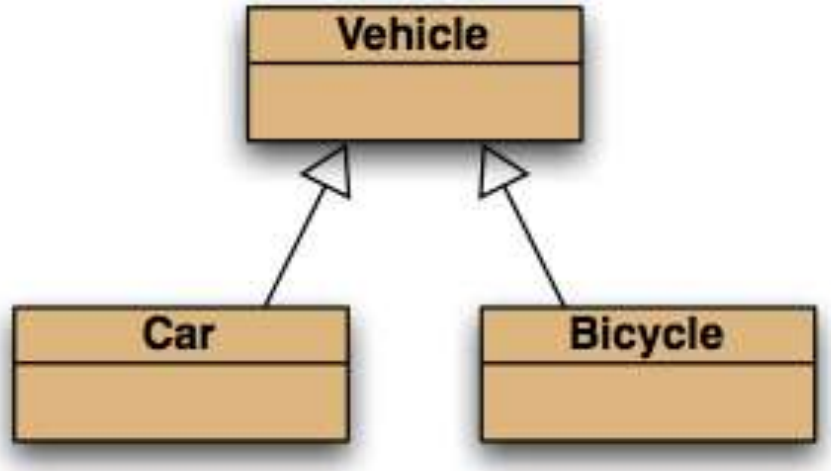
زیرکلاس‌ها و زیرنمونه‌ها

- کلاس‌ها، نوع‌هایی را تعریف می‌کنند.
- زیرکلاس‌ها، زیرنوع‌هایی را تعریف می‌کنند.
- اشیای زیرکلاس‌ها زمانی که اشیایی که از نوع ابرکلاس نیاز است می‌تواند مورد استفاده قرار گیرد.

(این امر جایگزینی – **substitution** خوانده می‌شود.)

زیرنوع‌ها و انتساب

اشیای یک زیرکلاس می‌تواند به متغیرهای از نوع ابرکلاس منتسب شود.




```
Vehicle v1 = new Vehicle();
Vehicle v2 = new Car();
Vehicle v3 = new Bicycle();
```

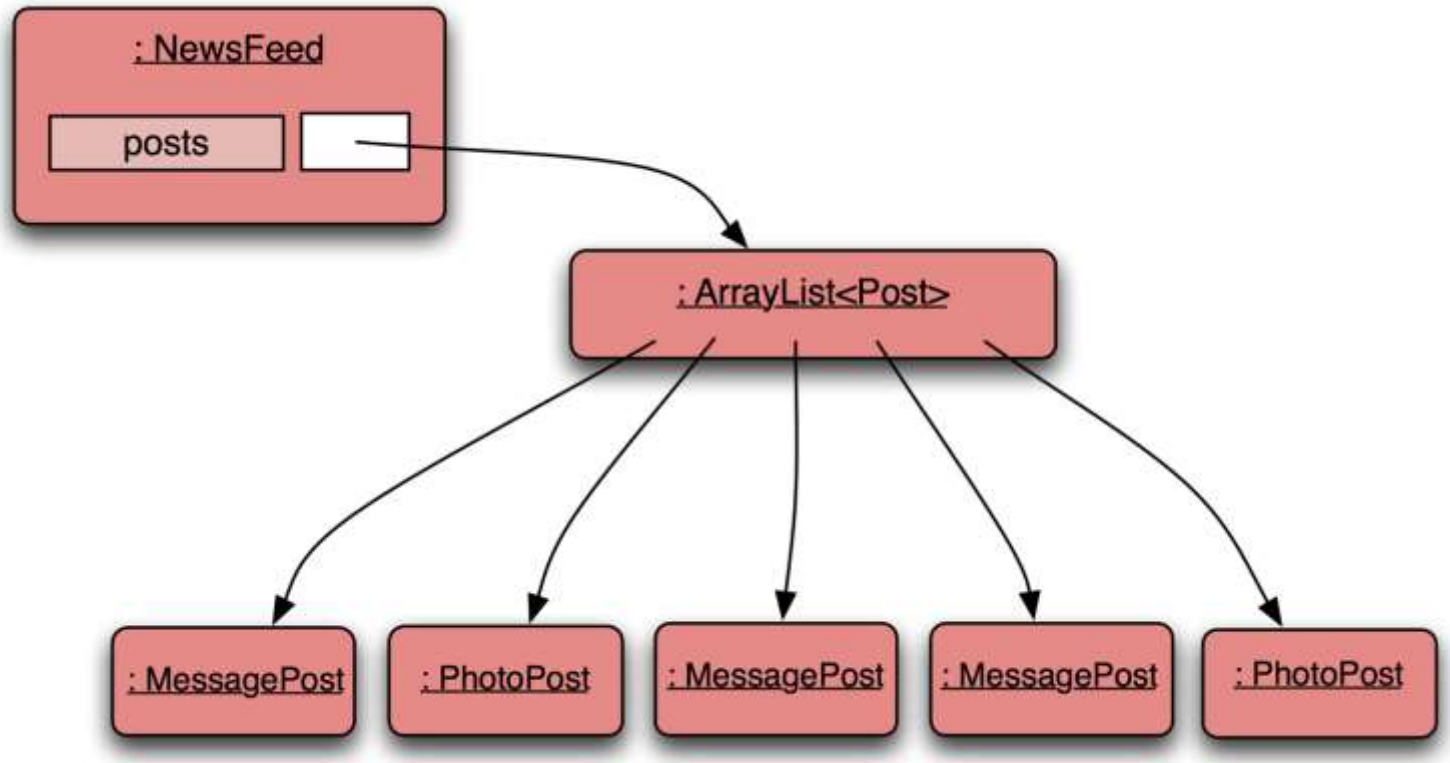
زیرنوع‌ها و ارسال پارامتر

```
public class NewsFeed
{

    public void addPost(Post post)
    {
        ...
    }
}
```

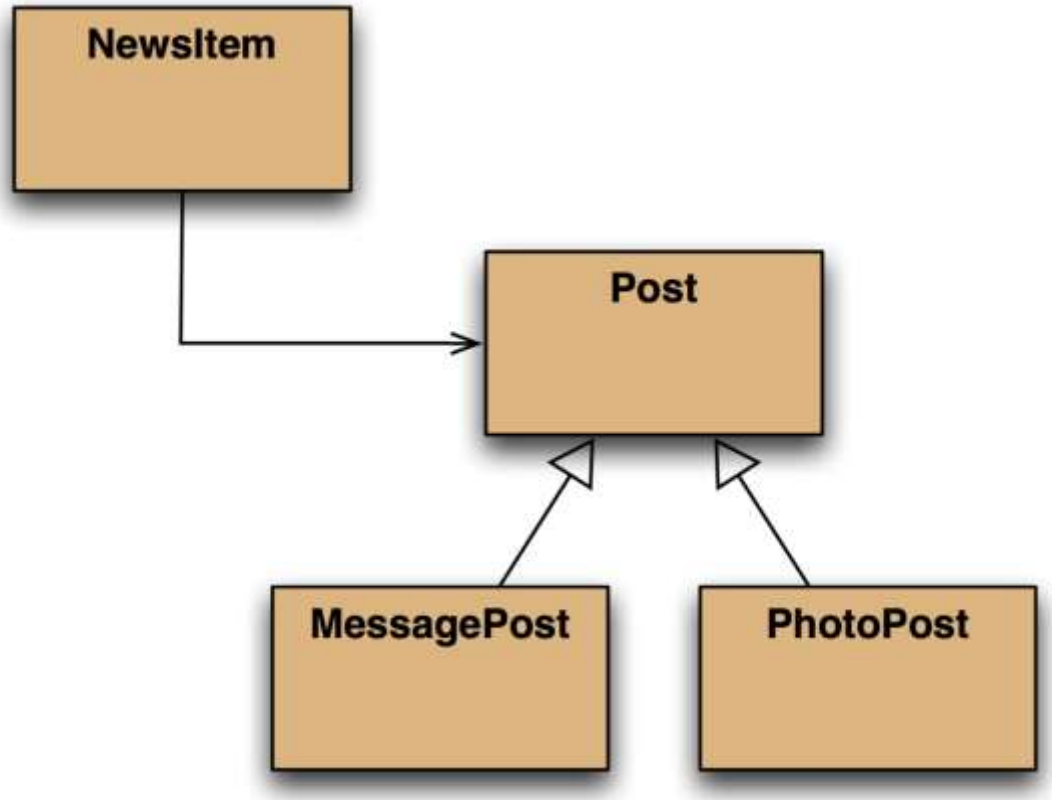
اشیا زیرکلاس می‌توانند به جای پارامترهای مربوط ابرکلاس ارسال شوند.

```
PhotoPost photo = new PhotoPost(...);
MessagePost message = new MessagePost(...);

feed.addPost(photo);
feed.addPost(message);
```

دیاگرام شی


: NewsFeed
posts
: ArrayList<Post>
: MessagePost
: PhotoPost
: MessagePost
: MessagePost
: PhotoPost

دیاگرام کلاس



متغیرهای چندریخت

- چندریختی یعنی: شکل‌های متعدد.
- متغیرهای شی در جاوا چندریخت (`polymorphic`) هستند.
  - می‌توانند اشیایی از بیشتر از یک نوع را نگهداری کنند.
- می‌توانند اشیایی از نوع تعریف شده (نوع کلاس) و زیرنوع‌های نوع تعریف شده را نگهداری کنند.

- یادآوری: چندریختی با ما اجازه می‌دهد کدهایی را برای ابرکلاس بنویسیم اما رفتار در زمان اجرا متعلق به زیرکلاس‌ها و مناسب آنها باشد.

تبدیل نوع (Casting)

- می‌توان یک زیرنوع را به یک ابرنوع منتسب کرد.
- ... اما نمی‌توان یک ابرنوع را به یک زیرنوع منتسب کرد.

```
Vehicle v;
Car c = new Car();
v = c; //correct;
c = v; //compile-time error!
```

- می‌توان این مورد را اصلاح کرد:

```
c = (Car) v;
```

(فقط در صورتی درست است که `Vehicle` مورد نظر واقعا یک `Car` باشد.)

تبدیل نوع (Casting)

- نوع شی در پرانتز
- برای جلوگیری از "از دست دادن نوع" استفاده می‌شود.
- شی به هیچ عنوان تغییر نمی‌کند.
- یک بررسی زمان اجرا برای اطمینان از اینکه شی واقعا از آن نوع است انجام می‌شود.
  - `ClassCastException` رخ می‌دهد در صورتی که نباشد.
- در استفاده از آن امساک کنید!

محدودیت‌های وراثت

وراثت مهم است اما دارای محدودیت‌هایی است از جمله:

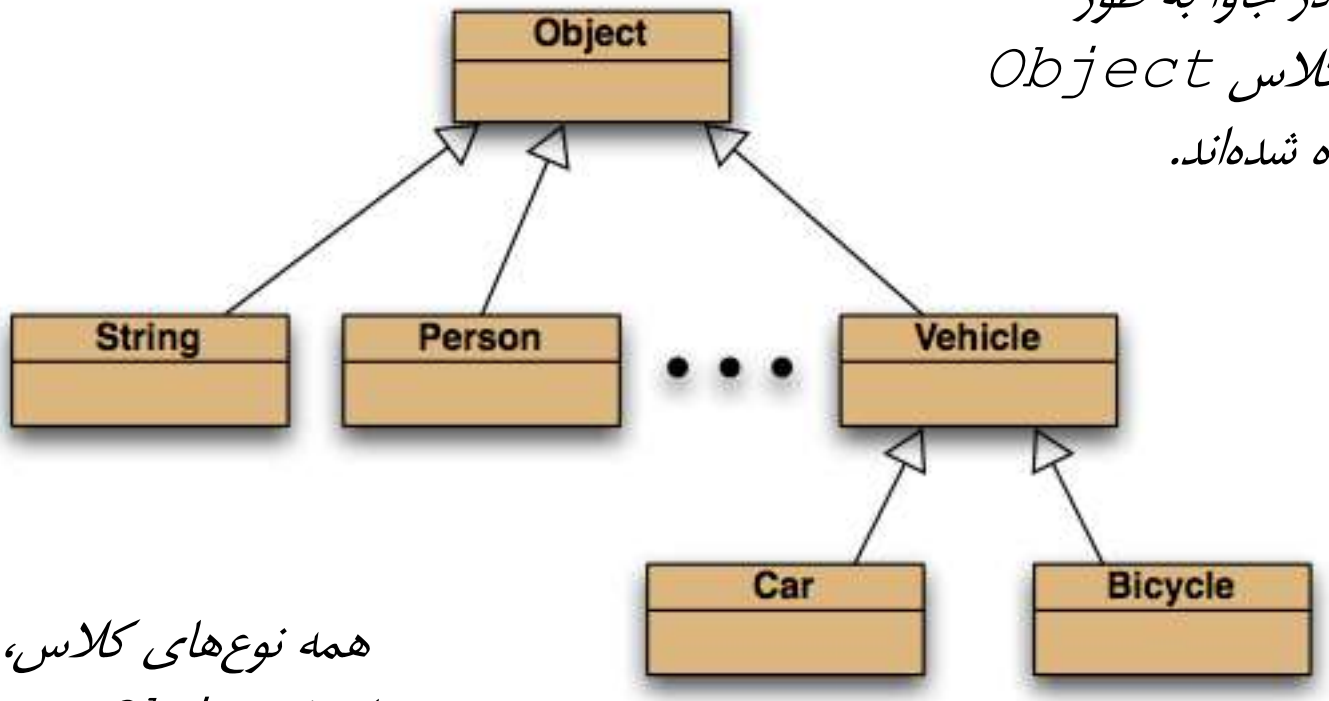
- یک زیرکلاس همه چیز را به ارث می‌برد.
  - هیچ روش مناسبی برای پنهان کردن فیلدها یا متدهای به ارث برده شده که نمی‌خواهیم وجود ندارد.
    - می‌توان با تغییر ساختار کلاس‌ها تا حدی به این مهم دست یافت
- وراثت یک رابطه ایستای معین شده در زمان کامپایل است.
  - روشی برای تغییر روابط بین اشیا در زمان اجرا وجود ندارد.
- در جاوا یک کلاس فقط می‌تواند یک کلاس دیگر را توسعه دهد.
  - در C++ امکان توسعه چند کلاس وجود دارد.

جایگزین‌هایی برای وراثت

- در بسیاری از موارد جایگزین‌های بهتری وجود دارد: ترکیب / نمایندگی (composition / delegation)
  - بسیاری از کتاب‌ها تاکید بیش از اندازه‌ای روی وراثت می‌کنند، بسیاری از طراحان استفاده بیش از اندازه‌ای از آن می‌کنند.
  - در آینده بیشتر در این مورد صحبت می‌کنیم.

کلاس Object

تمام اشیا در جاوا به طور ضمنی از کلاس Object به ارث برده شده‌اند.



همه نوع‌های کلاس، زیرنوع‌های از Object هستند.

مجموعه‌های چندریخت

- تمام مجموعه‌ها چندریخت هستند.
- المان‌ها به سادگی می‌توانند از نوع `Object` باشند.

```
public void add(Object element)

public Object get(int index)
```

- معمولا محدودیتی توسط تعریف نوع پارامتری برای استفاده از مجموعه تعریف می‌شود.

مجموعه‌های چندریخت

- یک نوع پارامتری، درجه چند ریختی را محدود می‌کند.

```
ArrayList<Post>
```

  - نوع‌های `Post`، `MessagePost` و `PhotoPost` قابل استفاده‌اند.

- در این صورت مجموعه‌ها نوع‌دار خواهند بود.
- بدون نوع پارامتری، به طور ضمنی `ArrayList<Object>` در نظر گرفته می‌شود.
- با اخطار زیر مواجه خواهید شد:

  - *"unchecked or unsafe operations"*

- به احتمال زیاد مجبور به استفاده از `Casting` خواهید شد.

مجموعه‌ها و نوع‌های اولیه

- تمام اشیا می‌توانند در مجموعه‌ها درج شوند...
- ... چون اشیا المان‌هایی از نوع `Object` را می‌پذیرد...
- ... و تمام کلاس‌ها یک زیرنوع از `Object` هستند.
- بیسار عالی! اما نوع‌های اولیه چطور؟ **`int`**، **`boolean`** و...

کلاس‌های Wrapper

- نوع‌های اولیه نوع‌های شی نیستند. نوع‌های اولیه باید به شکل اشیا بسته بندی (wrapped) شوند تا بتوان آنها را در مجموعه‌ها ذخیره شوند.
- کلاس‌های wrapper برای تمام نوع‌های اولیه وجود دارند.

Primitive type	Wrapper class
int	Integer
float	Float
char	Character
…	…

کلاس‌های Wrapper

wrap the value

```
int i = 18;
Integer iwrap = new Integer(i);
…
int value = iwrap.intValue();
```

unwrap it

In practice, *autoboxing* and *unboxing* mean we don't often have to do this explicitly

Autoboxing and unboxing

```
private ArrayList<Integer> markList;
…
public void storeMark(int mark)
{
    markList.add(mark);
}
```

autoboxing

```
int firstMark = markList.remove(0);
```

unboxing

مرور

- وراثت اجازه می‌دهد کلاس‌هایی را با گسترش کلاس‌های دیگر ایجاد کنیم.
- وراثت
  - جلوگیری از تکرار کد
  - استفاده مجدد از کد
  - ساده‌سازی کد
  - ساده‌سازی نگهداری و گسترش
- متغیرها می‌توانند اشیای زیرنوع را نگهداری کنند.
- زیرنوع‌ها می‌توانند در جاهایی که ابرنوع‌ها قابل پذیرش هستند استفاده شوند (جایگزینی – `substitution`).

مرور

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