Inheritance

Improving Structure with Inheritance

Produced by: Dr. Siobhán Drohan

Mairead Meagher

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Lectures and Labs

- This weeks lectures and labs are based on examples in:

  – Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Topic List

- NetworkV1
- Inheritance hierarchies
- Network V2
- Coding inheritance hierarchies
  - Super and subclasses
  - Using constructors in these hierarchies
- Network V3
  - Deeper hierarchies
  - Advantages of using inheritance
- Subtyping and Substitution
- Polymorphic variables / Collections
  - Includes casting, wrapper classes, autoboxing /unboxing

NetworkV1

- A small, prototype social network.
- Supports a news feed with posts.

- Stores *text-based posts* and *photo posts*.
  - MessagePost: multi-line text message.
  - PhotoPost: photo and caption.

- Allows operations on the posts:
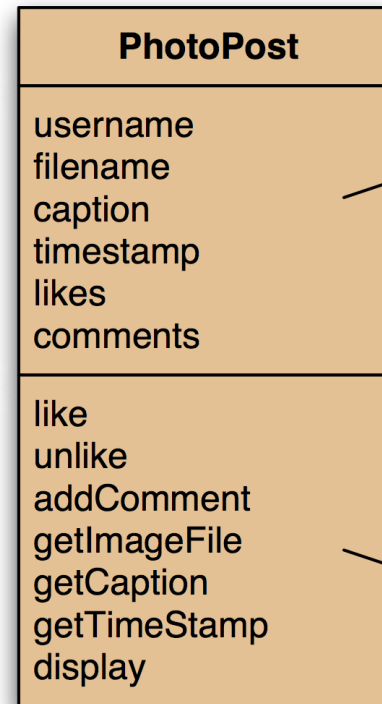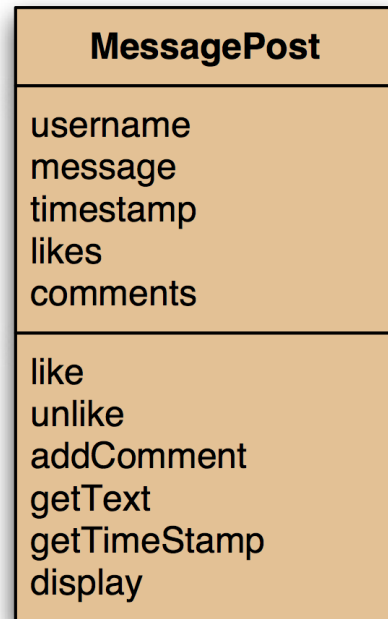 - e.g., search, display and remove.

NetworkV1 - Objects

: MessagePost

username	
message	
timestamp	
likes	
comments	

: PhotoPost

username	
filename	
caption	
timestamp	
likes	
comments	

NetworkV1 - Classes

MessagePost

username
message
timestamp
likes
comments

like
unlike
addComment
getText
getTimeStamp
display

PhotoPost

username
filename
caption
timestamp
likes
comments
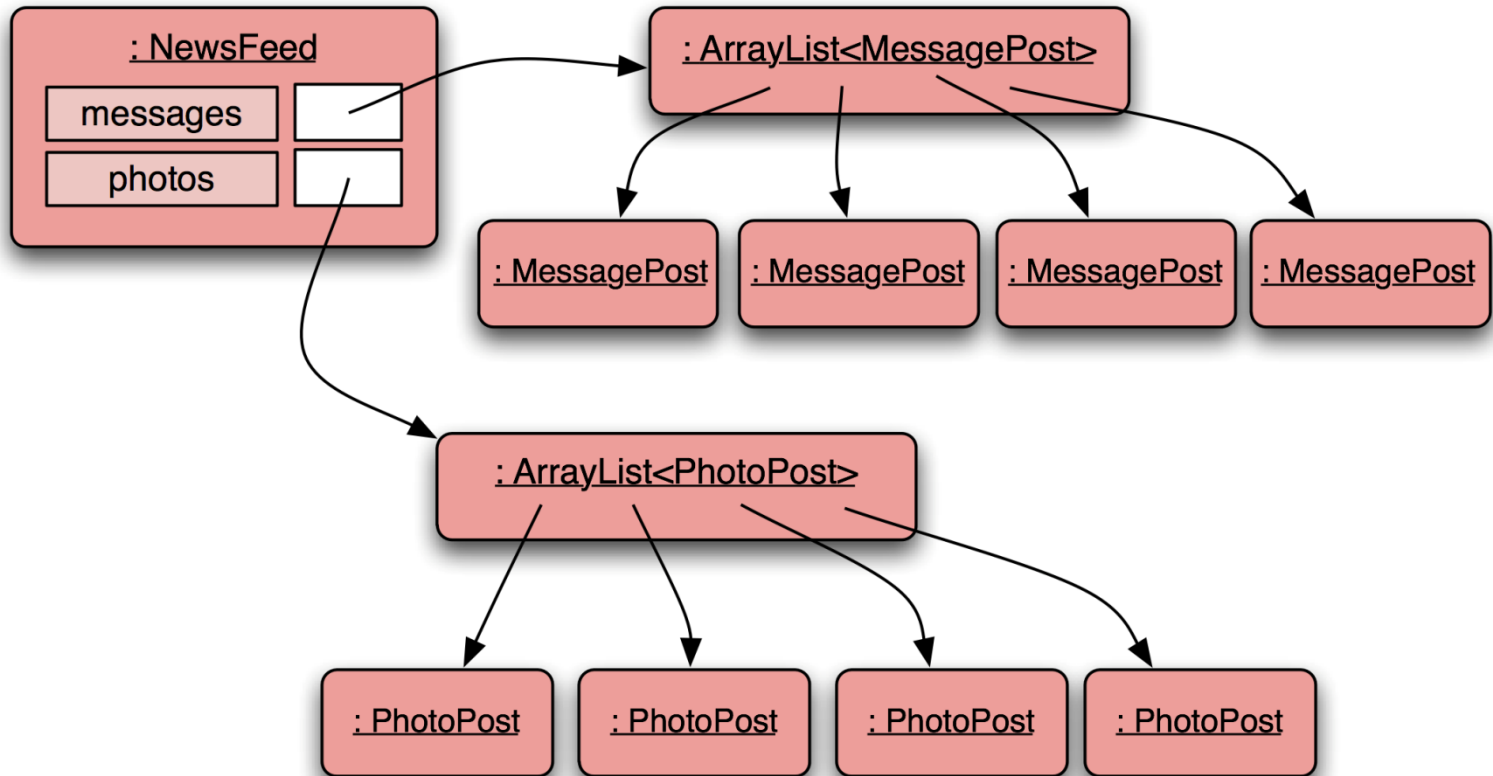
like
unlike
addComment
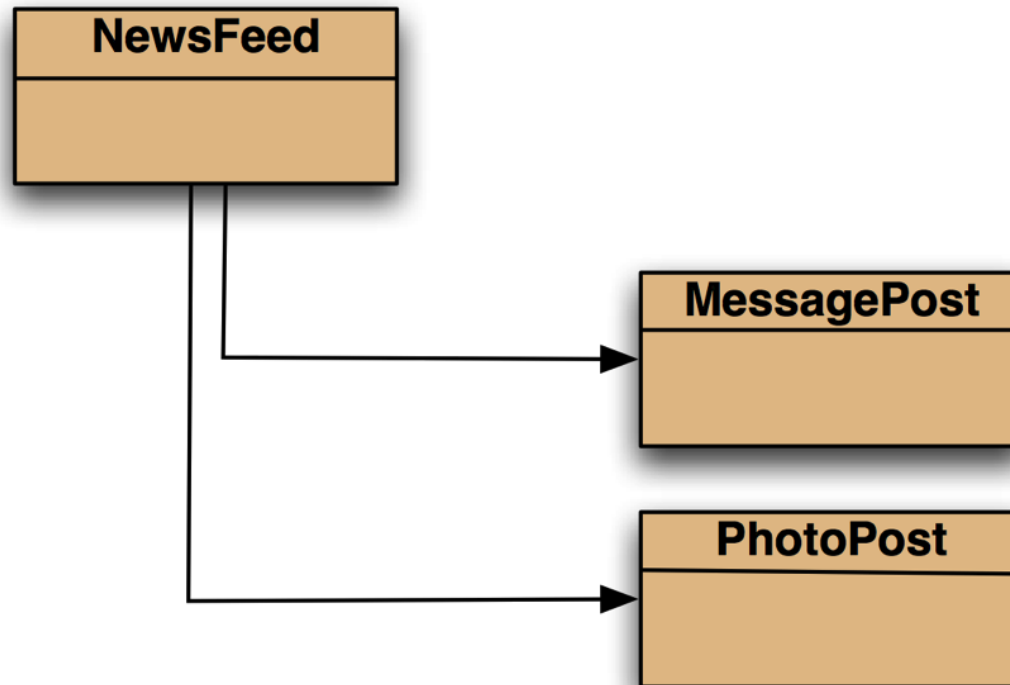getImageFile
getCaption
getTimeStamp
display

*top half
shows fields*

*bottom half
shows methods*

NetworkV1 - Object model

NetworkV1 - Class diagram

MessagePost
source code

*Just an
outline*

```java
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() ...

    ...
}
```

PhotoPost
source code

*Just an
outline*

```java
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() …
    ...
}
```

```java
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
        }
    }
}
```
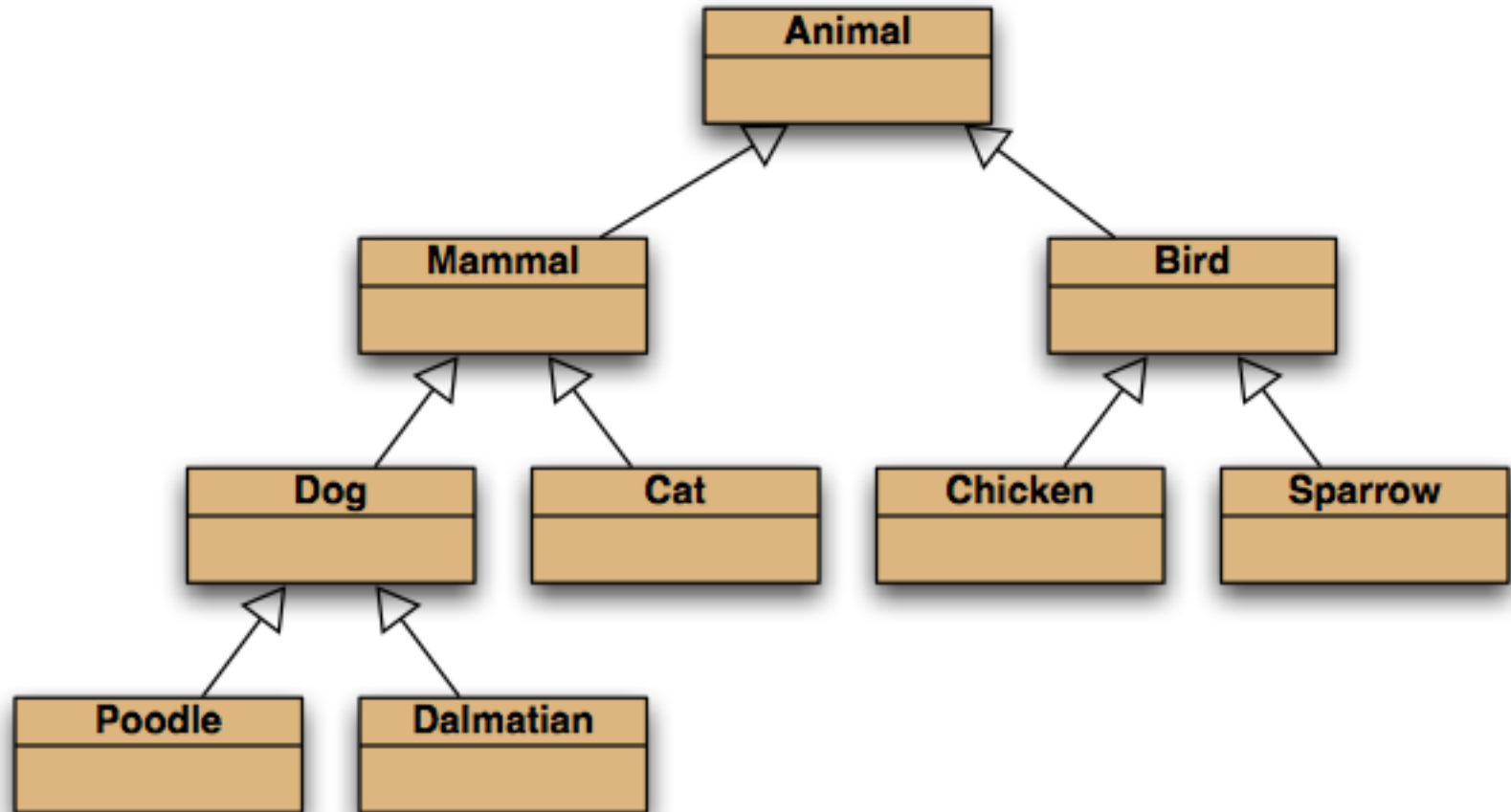
Topic List

- NetworkV1
- Inheritance hierarchies
- Network V2
- Coding inheritance hierarchies
  - Super and subclasses
  - Using constructors in these hierarchies
- Network V3
  - Deeper hierarchies
  - Advantages of using inheritance
- Subtyping and Substitution
- Polymorphic variables / Collections
  - Includes casting, wrapper classes, autoboxing /unboxing
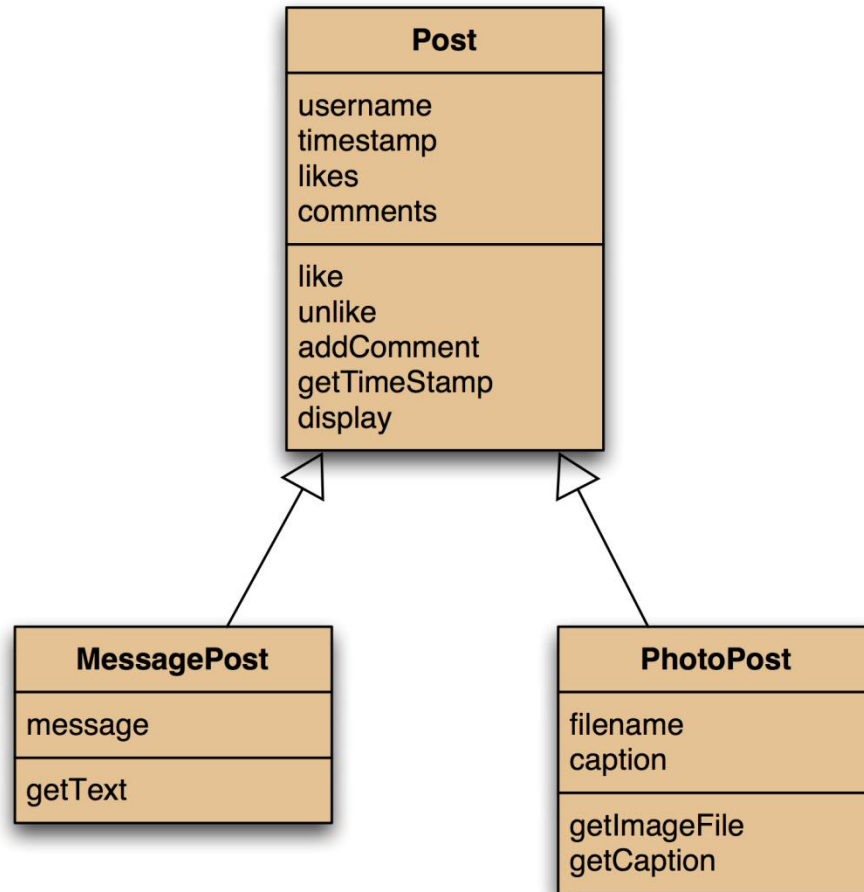
Inheritance hierarchies

Topic List

- NetworkV1
- Inheritance hierarchies
- Network V2
- Coding inheritance hierarchies
  - Super and subclasses
  - Using constructors in these hierarchies
- Network V3
  - Deeper hierarchies
  - Advantages of using inheritance
- Subtyping and Substitution
- Polymorphic variables / Collections
  - Includes casting, wrapper classes, autoboxing /unboxing

Critique of NetworkV1

- Code duplication:
  - MessagePost and PhotoPost classes very similar (large parts are identical)
  - makes maintenance difficult/more work
  - introduces danger of bugs through incorrect maintenance

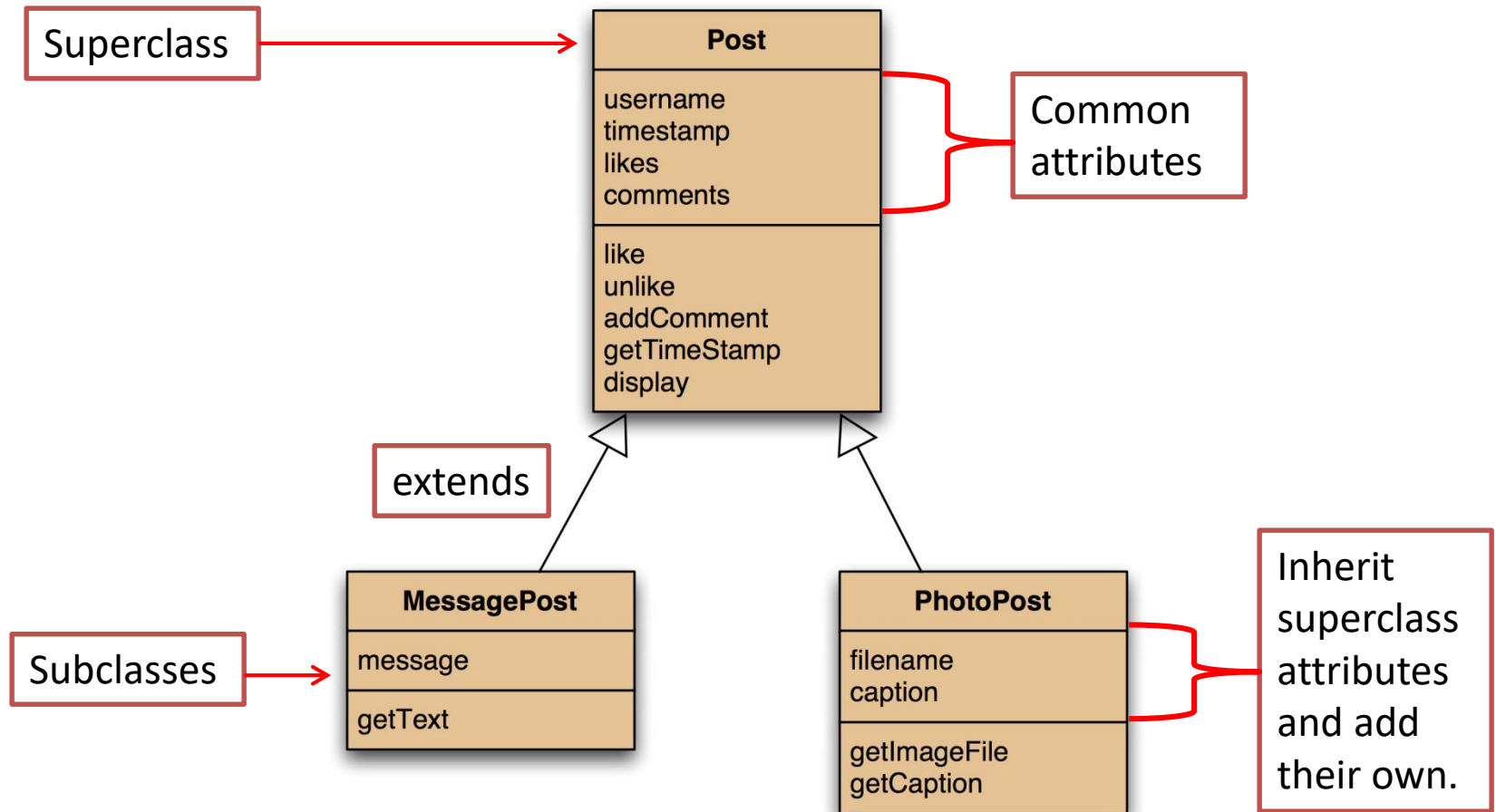- Code duplication in NewsFeed class as well.

NetworkV2 - Using inheritance

NetworkV2 - Using inheritance

- define one **superclass** : `Post`
- define **subclasses** for `MessagePost` and `PhotoPost`
- the superclass defines common attributes (via fields)
- the subclasses **inherit** the superclass attributes
- the subclasses add other attributes

NetworkV2 - Using inheritance

Superclass

Post

username
timestamp
likes
comments

Common attributes

like
unlike
addComment
getTimeStamp
display

extends

Subclasses

MessagePost

message

getText

PhotoPost

filename
caption

getImageFile
getCaption

Inherit superclass attributes and add their own.

Topic List

- NetworkV1
- Inheritance hierarchies
- Network V2
- Coding inheritance hierarchies
  - Super and subclasses
  - Using constructors in these hierarchies
- Network V3
  - Deeper hierarchies
  - Advantages of using inheritance
- Subtyping and Substitution
- Polymorphic variables / Collections
  - Includes casting, wrapper classes, autoboxing /unboxing
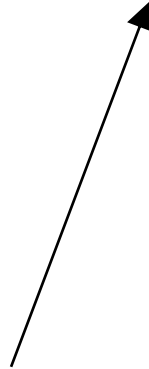
Inheritance in Java

```
public class Post
{
    ...
}
```
no change here

change here

```
public class PhotoPost extends Post
{
    ...
}
```

```
public class MessagePost extends Post
{
    ...
}
```

Superclass

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

    // constructor and methods omitted.
}
```

Subclasses

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

    // constructor and methods omitted.
}
```

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

    // constructor and methods omitted.
}
```

```
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
}
```

```java
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
}
```
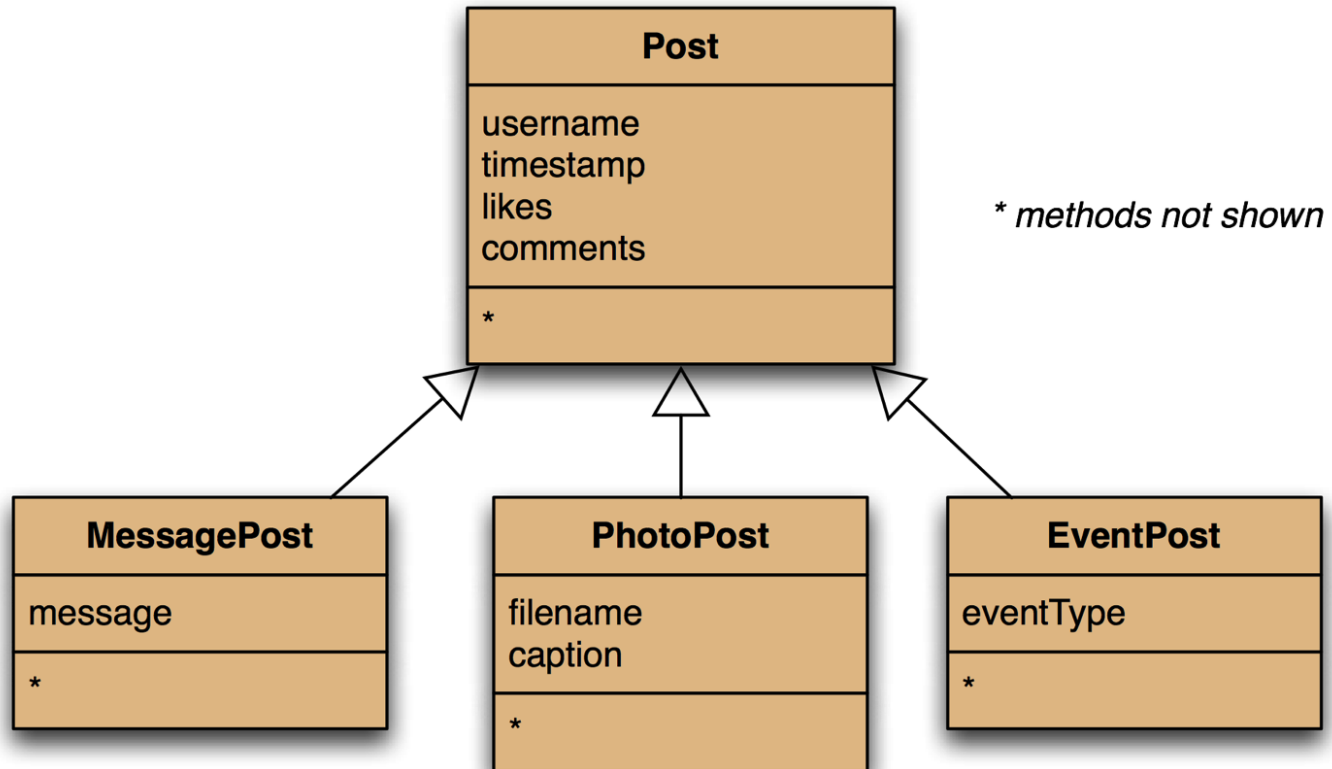
Superclass constructor call

- Subclass constructors <u>**must**</u> always contain a 'super' call.

- If none is written, the compiler inserts one (without parameters)

  - works only, if the superclass has a constructor without parameters

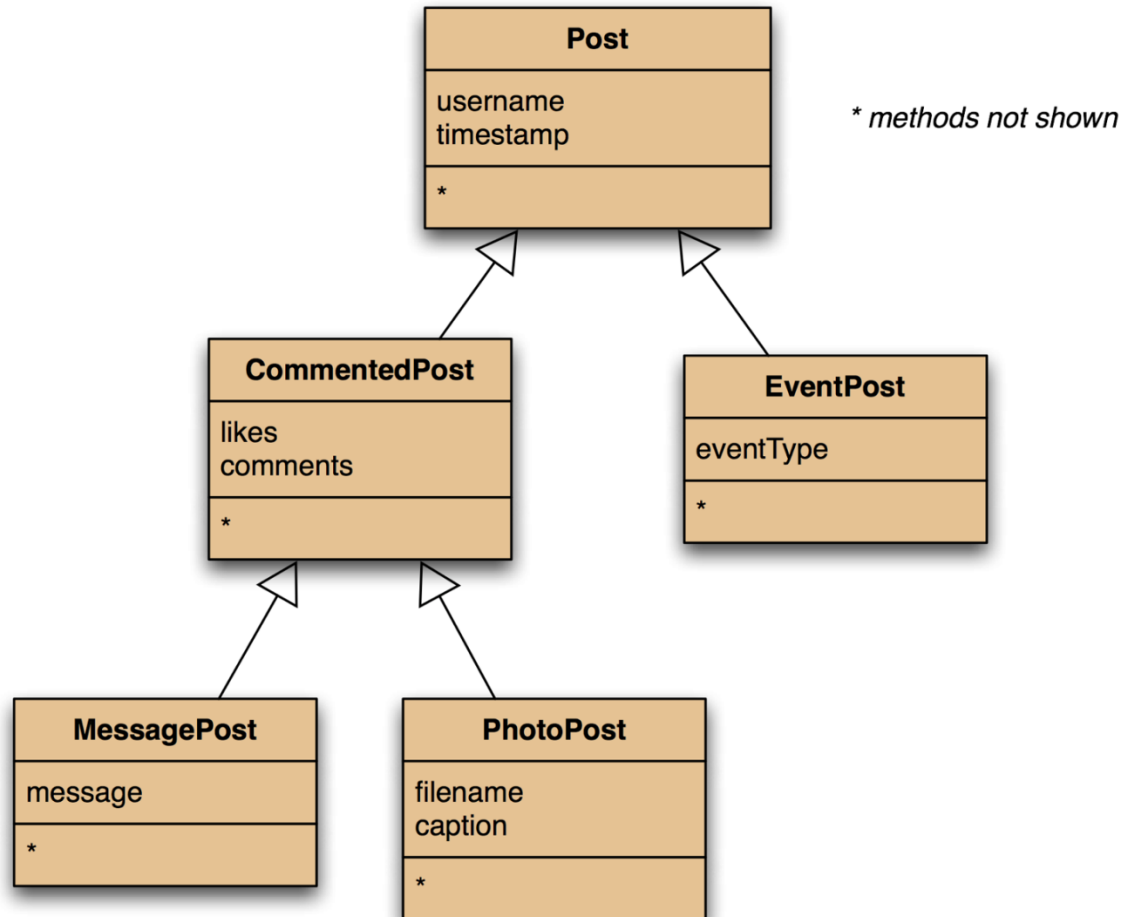- Must be the first statement in the subclass constructor.

Topic List

- NetworkV1
- Inheritance hierarchies
- Network V2
- Coding inheritance hierarchies
  - Super and subclasses
  - Using constructors in these hierarchies
- Network V3
  - Deeper hierarchies
  - Advantages of using inheritance
- Subtyping and Substitution
- Polymorphic variables / Collections
  - Includes casting, wrapper classes, autoboxing /unboxing

NetworkV3 - Adding more item types

Post

username
timestamp
likes
comments

*

methods not shown

MessagePost

message

*

PhotoPost

filename
caption

*

EventPost

eventType

*

NetworkV3 - Deeper hierarchies

Post

username
timestamp

*

CommentedPost

likes
comments

*

EventPost

eventType

*

MessagePost

message

*

PhotoPost

filename
caption

*

methods not shown

Advantages of inheritance

Inheritance (so far) helps with:

- Avoiding code duplication
- Code reuse
- Easier maintenance
- Extendibility

Revised NewsFeed source code

```java
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);
    }
    ...
}
```

Avoids code duplication in the client class!

New NewsFeed source code

```java
/**
 * Show the news feed. Currently: print the
 * news feed details to the terminal.
 */

public void show()
{
    for(Post post : posts) {
        post.display();
        System.out.println(); // Empty line ...
    }
}
```

Topic List

- NetworkV1
- Inheritance hierarchies
- Network V2
- Coding inheritance hierarchies
  - Super and subclasses
  - Using constructors in these hierarchies
- Network V3
  - Deeper hierarchies
  - Advantages of using inheritance
- Subtyping and Substitution
- Polymorphic variables / Collections
  - Includes casting, wrapper classes, autoboxing /unboxing

Subtyping

First, we had:

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

Now, we have:
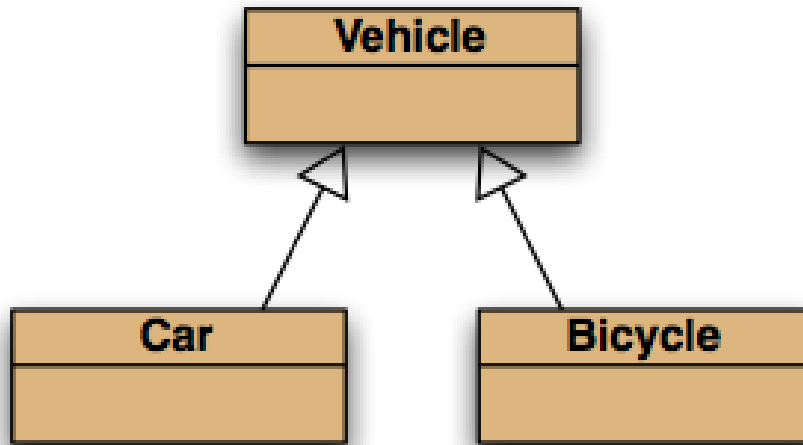
```
public void addPost(Post post)
```

We call this method with:

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

Subclasses and subtyping

- Classes define types.

- Subclasses define *subtypes*.

- Substitution: objects of subclasses can be used where objects of supertypes are required.

Subtyping and assignment

subclass objects may be assigned to superclass variables

```
Vehicle v1 = new Vehicle();
Vehicle v2 = new Car();
Vehicle v3 = new Bicycle();
```
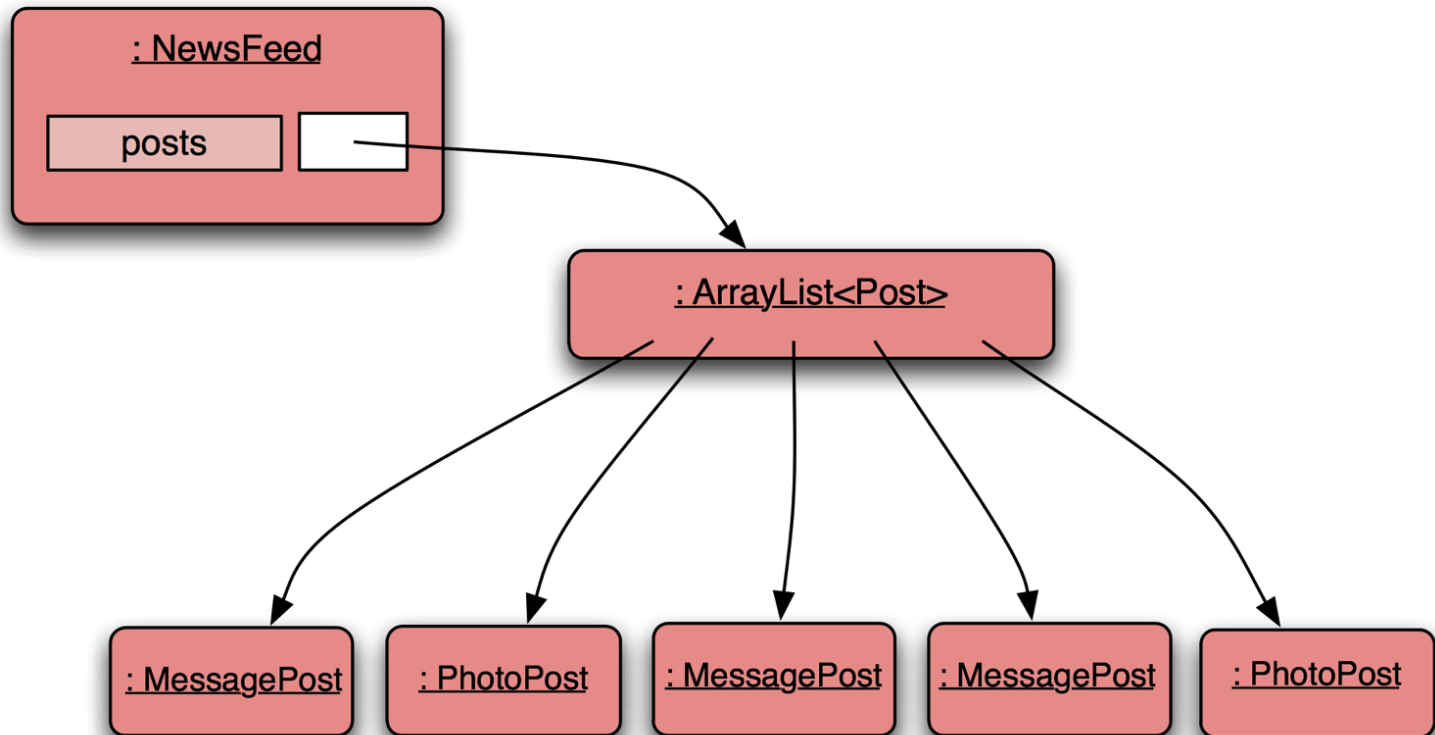
Subtyping and parameter passing

```
public class NewsFeed
{

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


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

feed.addPost(photo);
feed.addPost(message);
```
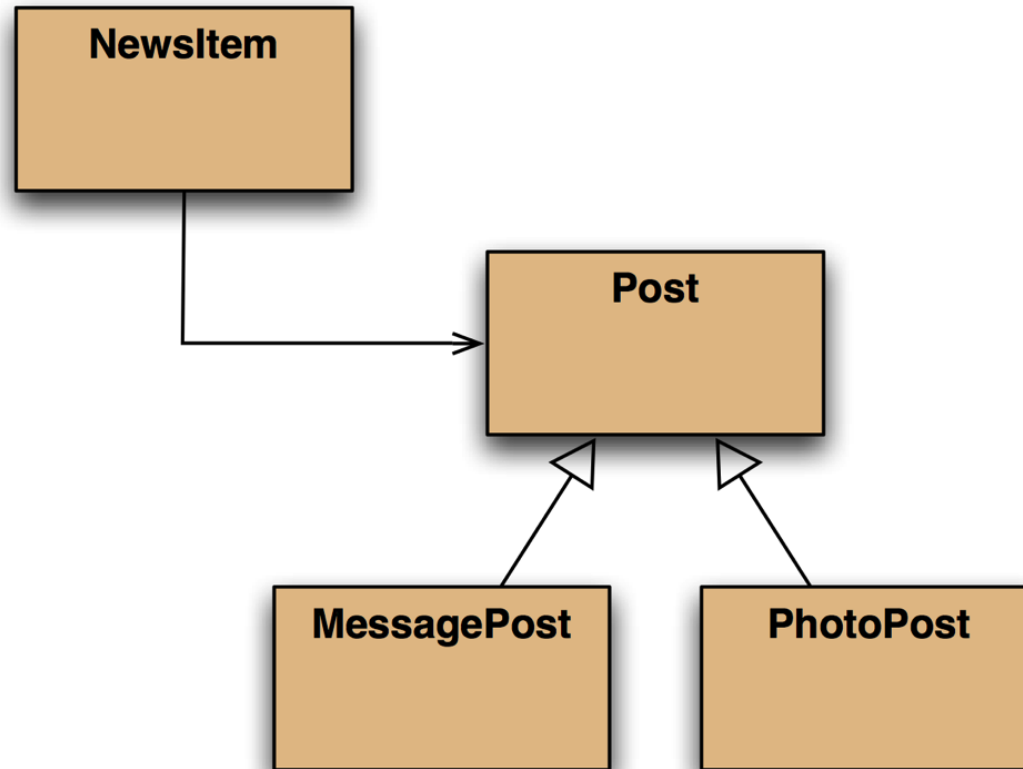
subclass objects may be used as actual parameters for the superclass

Object diagram

Class diagram

Topic List

- NetworkV1
- Inheritance hierarchies
- Network V2
- Coding inheritance hierarchies
  - Super and subclasses
  - Using constructors in these hierarchies
- Network V3
  - Deeper hierarchies
  - Advantages of using inheritance
- Subtyping and Substitution
- Polymorphic variables / Collections
  - Includes casting, wrapper classes, autoboxing /unboxing

Polymorphic variables

- Object variables in Java are **polymorphic** (they can hold objects of more than one type.)

- They can hold objects of the declared type, or of subtypes of the declared type.

Casting

- We can assign subtype to supertype …

- … but we cannot assign supertype to subtype!

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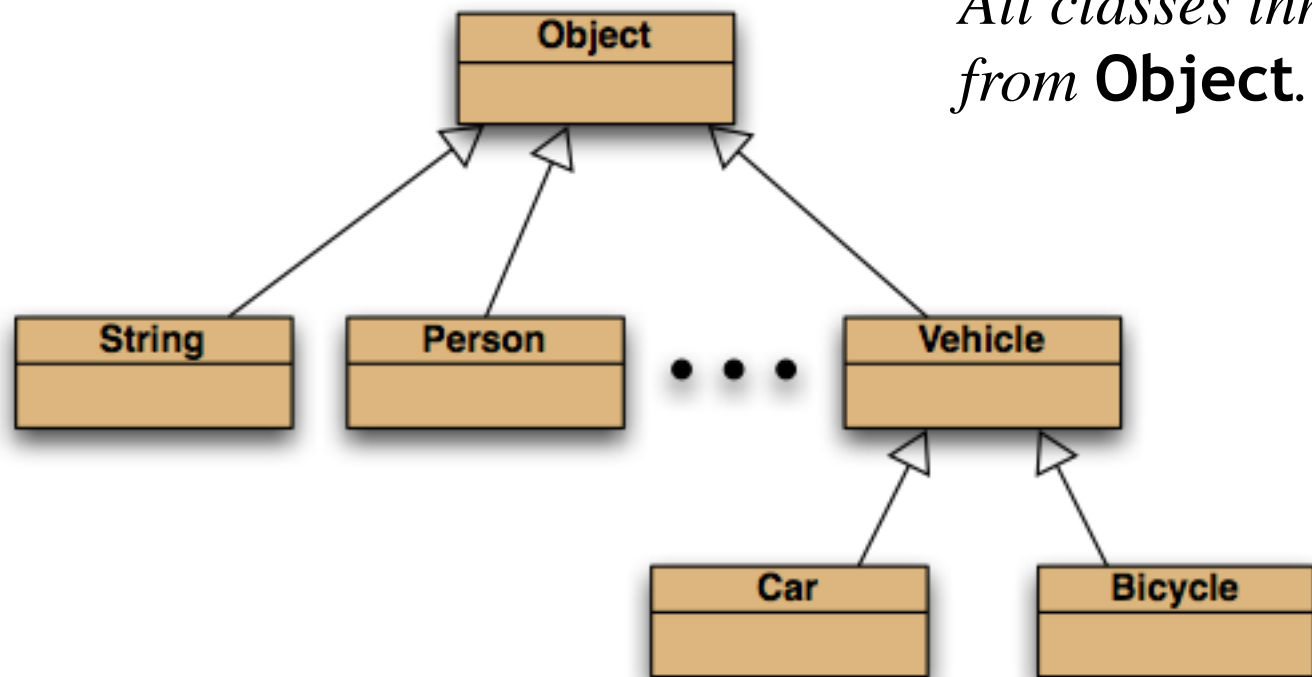
- Casting fixes this:

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

(only ok if the vehicle really is a Car!)

Casting

- An object type in parentheses.

- Used to overcome 'type loss'.

- The object is not changed in any way.

- A runtime check is made to ensure the object really is of that type:

  - **`ClassCastException`** if it isn't!

- Use it sparingly.

The Object class

All classes inherit from **Object***.*

Polymorphic collections

- All collections are polymorphic.
- The elements could simply be of type **Object**.

  ```
  public void add(Object element)
  ```

  ```
  public Object get(int index)
  ```

- Usually avoided by using a type parameter with the collection.

Polymorphic collections

- A type parameter limits the degree of polymorphism: `ArrayList<Post>`

- Collection methods are then typed.

- Without a type parameter, `ArrayList<Object>` is implied.

- Likely to get an *"unchecked or unsafe operations"* warning.

- More likely to have to use casts.

Collections and primitive types

- Potentially, all objects can be entered into collections …

  … because collections can accept elements of type **`Object`** …

  … and all classes are subtypes of **`Object`**.

- Great! But what about *the primitive types*: **`int`**, **`boolean`**, etc.?

Wrapper classes

- Primitive types are not objects types. Primitive-type values must be wrapped in objects to be stored in a collection!

- Wrapper classes exist for all primitive types:

simple type	wrapper class
int	Integer
float	Float
char	Character
…	…

Wrapper classes

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.get(0);
```

unboxing

Review

- Inheritance allows the definition of classes as extensions of other classes.

- Inheritance
  - avoids code duplication
  - allows code reuse
  - simplifies the code
  - simplifies maintenance and extending

- Variables can hold subtype objects.

- Subtypes can be used wherever supertype objects are expected (substitution).