

# Inheritance

## Improving Structure with Inheritance

---

Produced Dr. Siobhán Drohan  
by: Mr. Colm Dunphy  
Mr. Diarmuid O'Connor  
Dr. Frank Walsh



Waterford Institute *of* Technology  
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics  
<http://www.wit.ie/>

# 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 (<https://www.bluej.org/objects-first/>)

# Topic List

---

- 
1. Social Network V1
  2. Inheritance hierarchies
  3. Social Network V2
  4. Coding inheritance hierarchies
    - Super and subclasses
    - Using constructors in these hierarchies
  5. Social Network V3
    - Deeper hierarchies
    - Advantages of using inheritance
  6. Subtyping and Substitution
  7. Polymorphic variables / Collections
    - Includes casting, wrapper classes, autoboxing /unboxing

# Social Network V1

- A small, prototype SOCIAL NETWORK.
- Supports a **News Feed** with posts.
- **POSTS:**
  - **MessagePost:**
    - multi-line text message.
  - **PhotoPost:**
    - photo and caption.
  - **Operations**
    - e.g., search, display and remove.

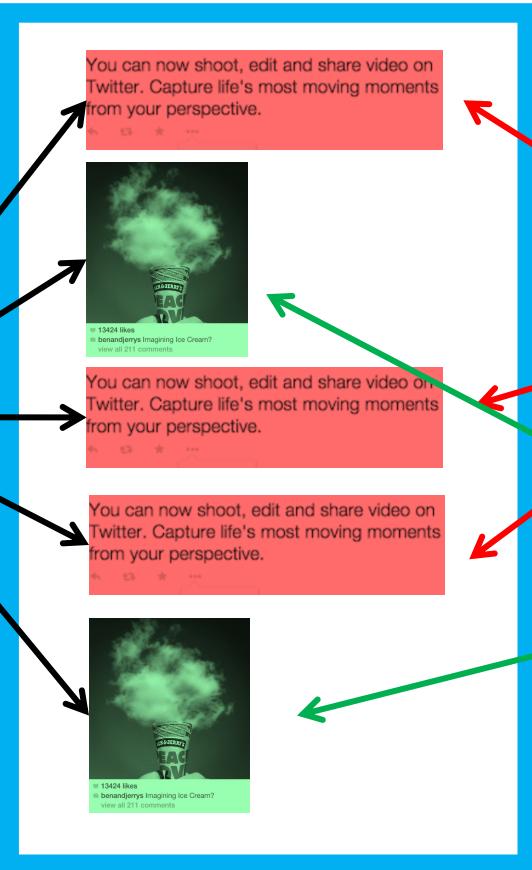


# SOCIAL NETWORK



## NEWS FEED

POSTS

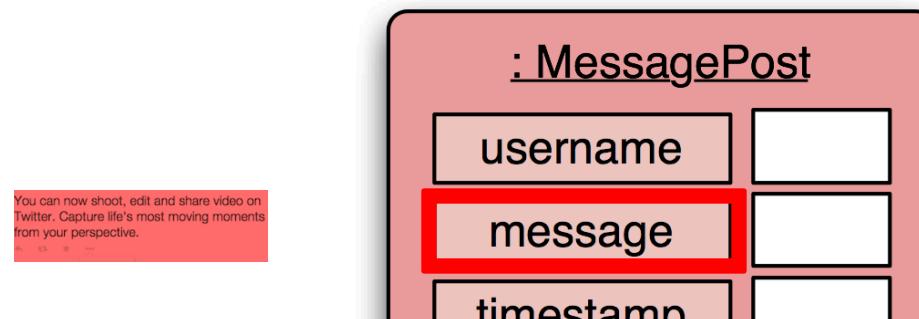


Multi Line Text

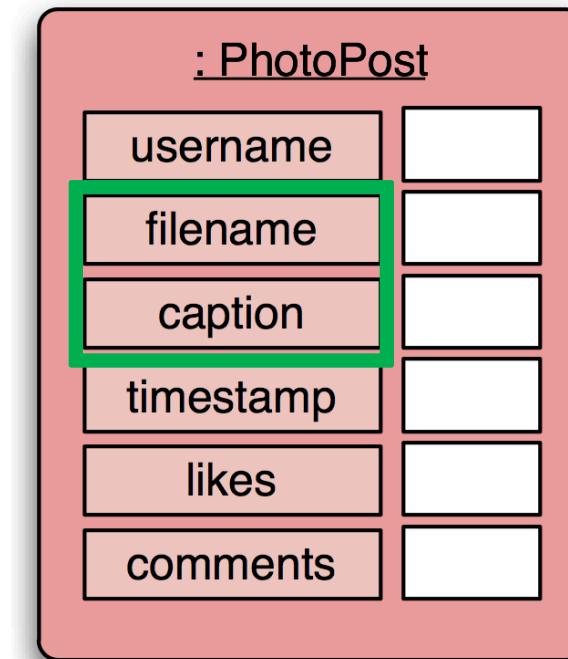
Photo

# Social Network V1 - Objects

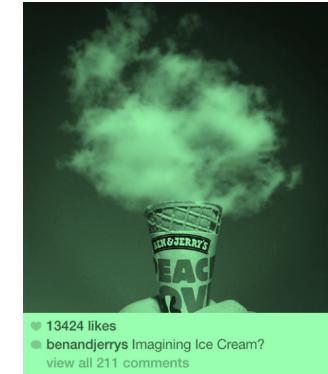
---



**MessagePost:** multi-line text message.

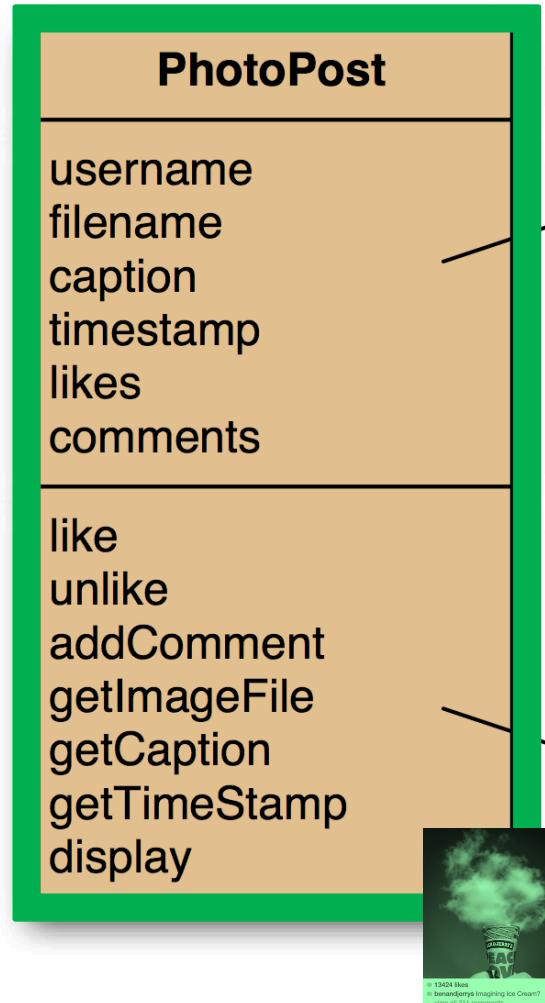


**PhotoPost:** photo and caption.



# Social Network V1 - Classes

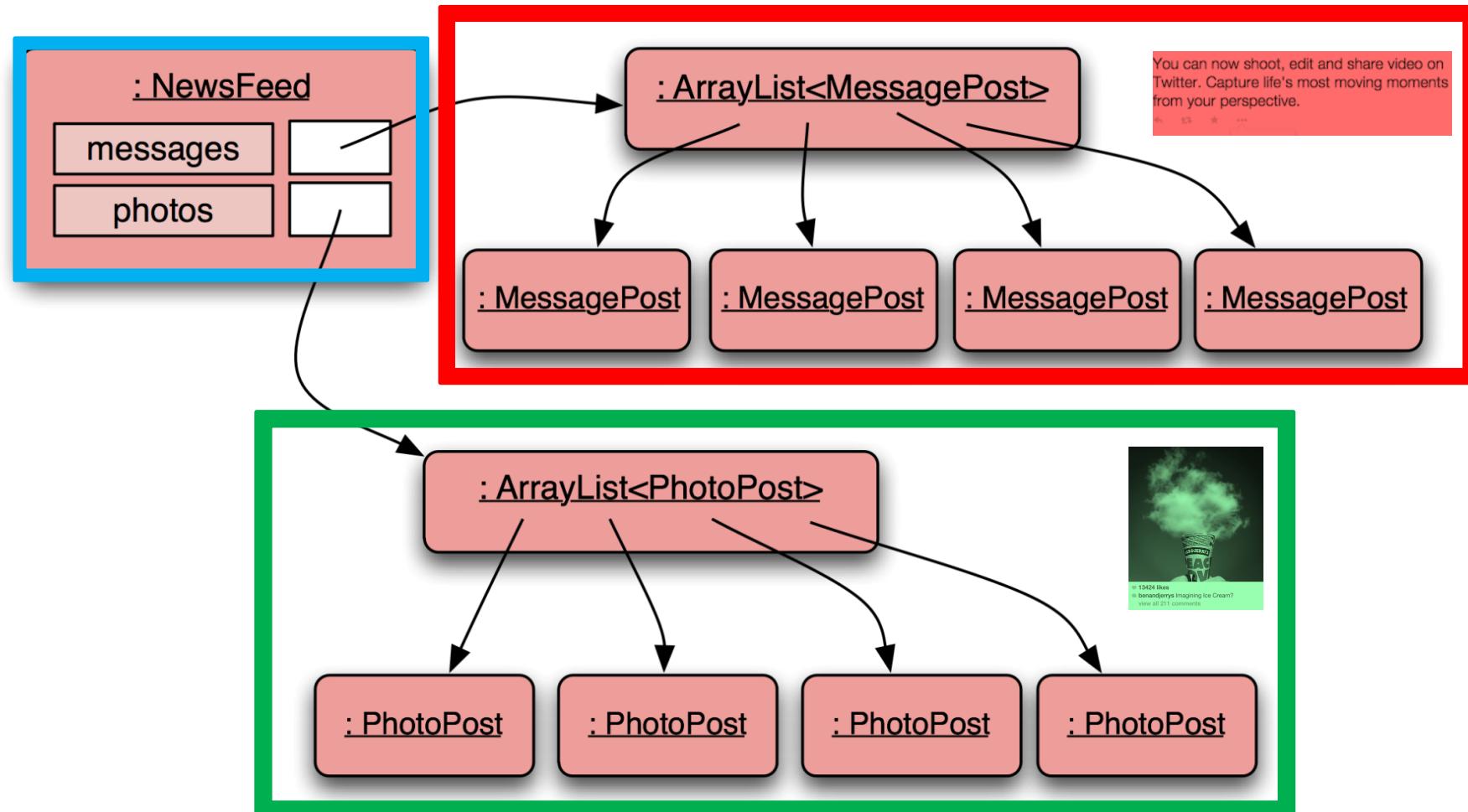
---



*top half shows fields*

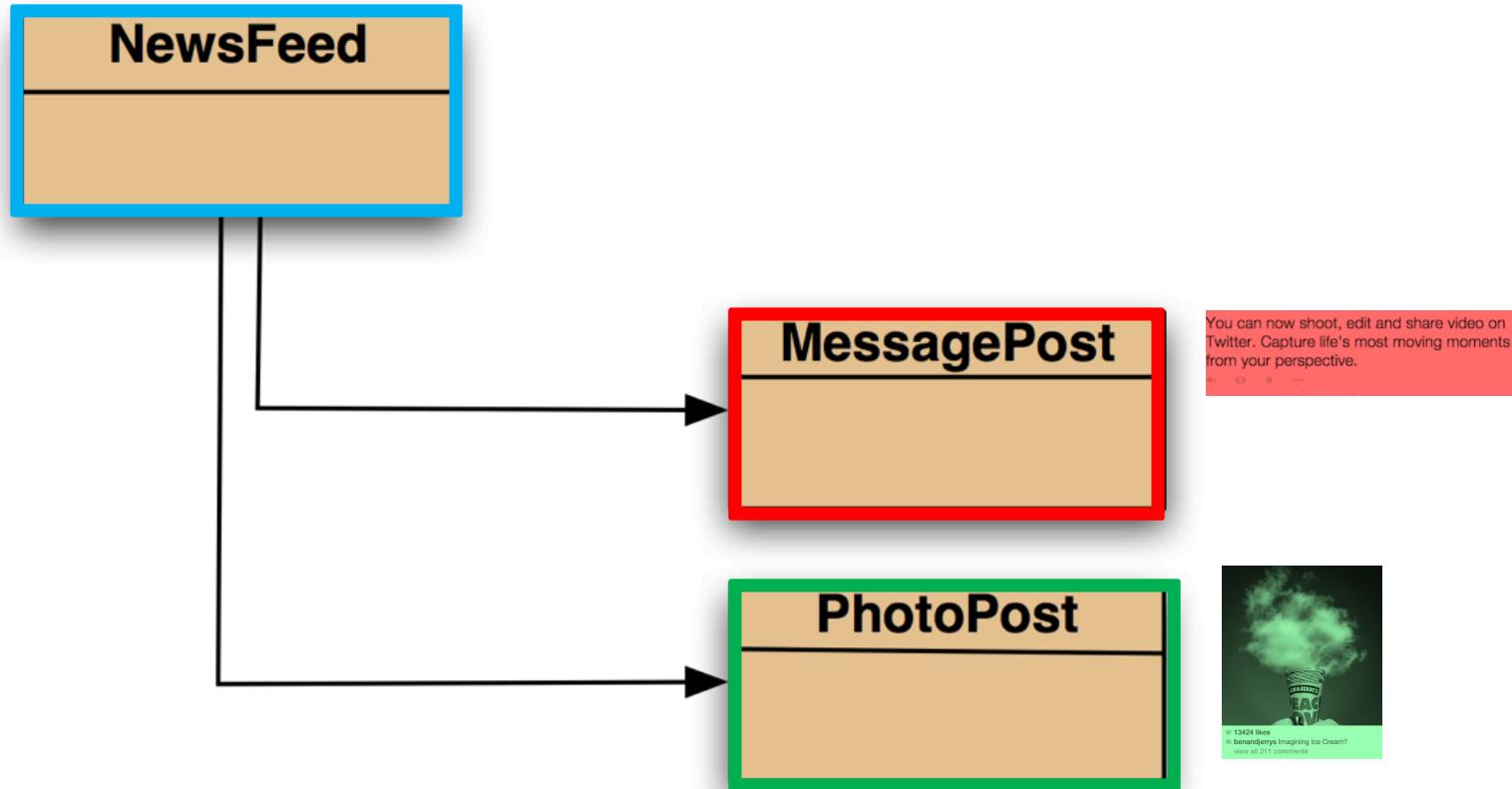
*bottom half shows methods*

# Social Network V1 - Object model



# Social Network V1 - Class diagram

---



## MessagePost source code

You can now shoot, edit and share video on Twitter. Capture life's most moving moments from your perspective.

*Just an outline...*

```
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...

```
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() ...

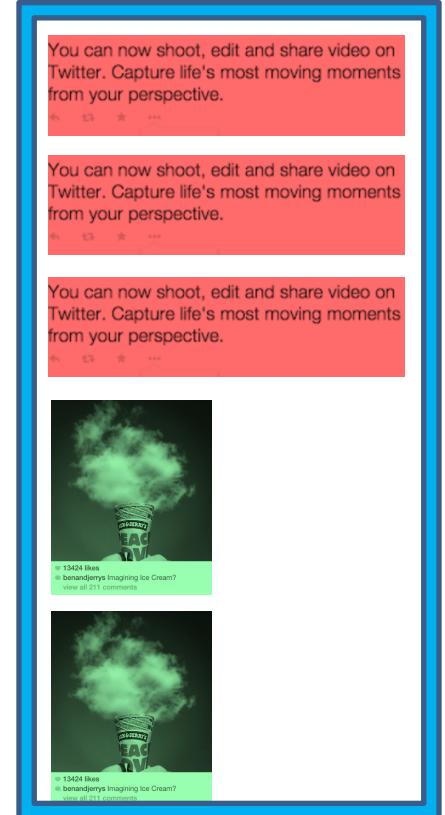
    public void display() ...
    ...
}
```

# NewsFeed

## source code

```
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
        }
    }
}
```



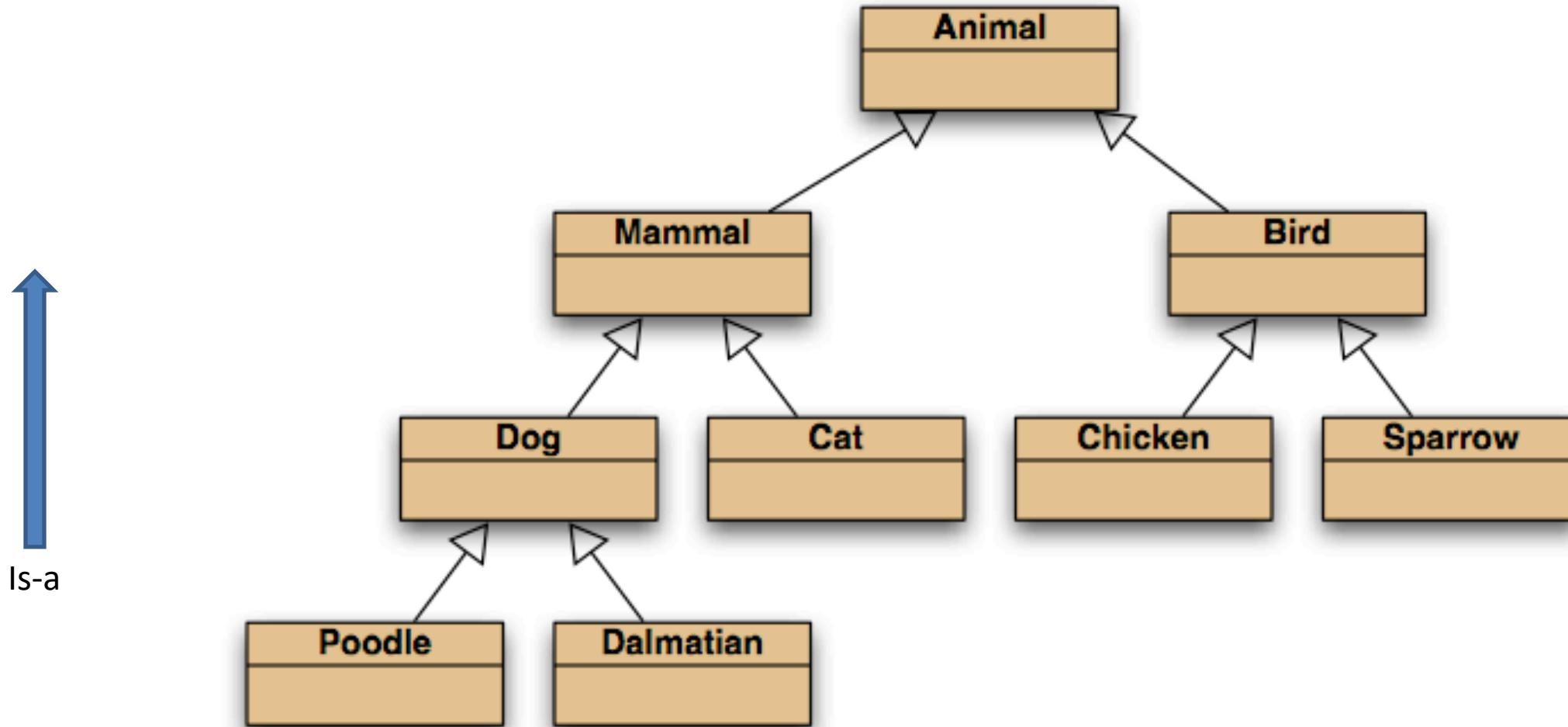
# Topic List

---

- 
1. Social Network V1
  2. Inheritance hierarchies
  3. Social Network V2
  4. Coding inheritance hierarchies
    - Super and subclasses
    - Using constructors in these hierarchies
  5. Social Network V3
    - Deeper hierarchies
    - Advantages of using inheritance
  6. Subtyping and Substitution
  7. Polymorphic variables / Collections
    - Includes casting, wrapper classes, autoboxing /unboxing

# Inheritance hierarchies

---



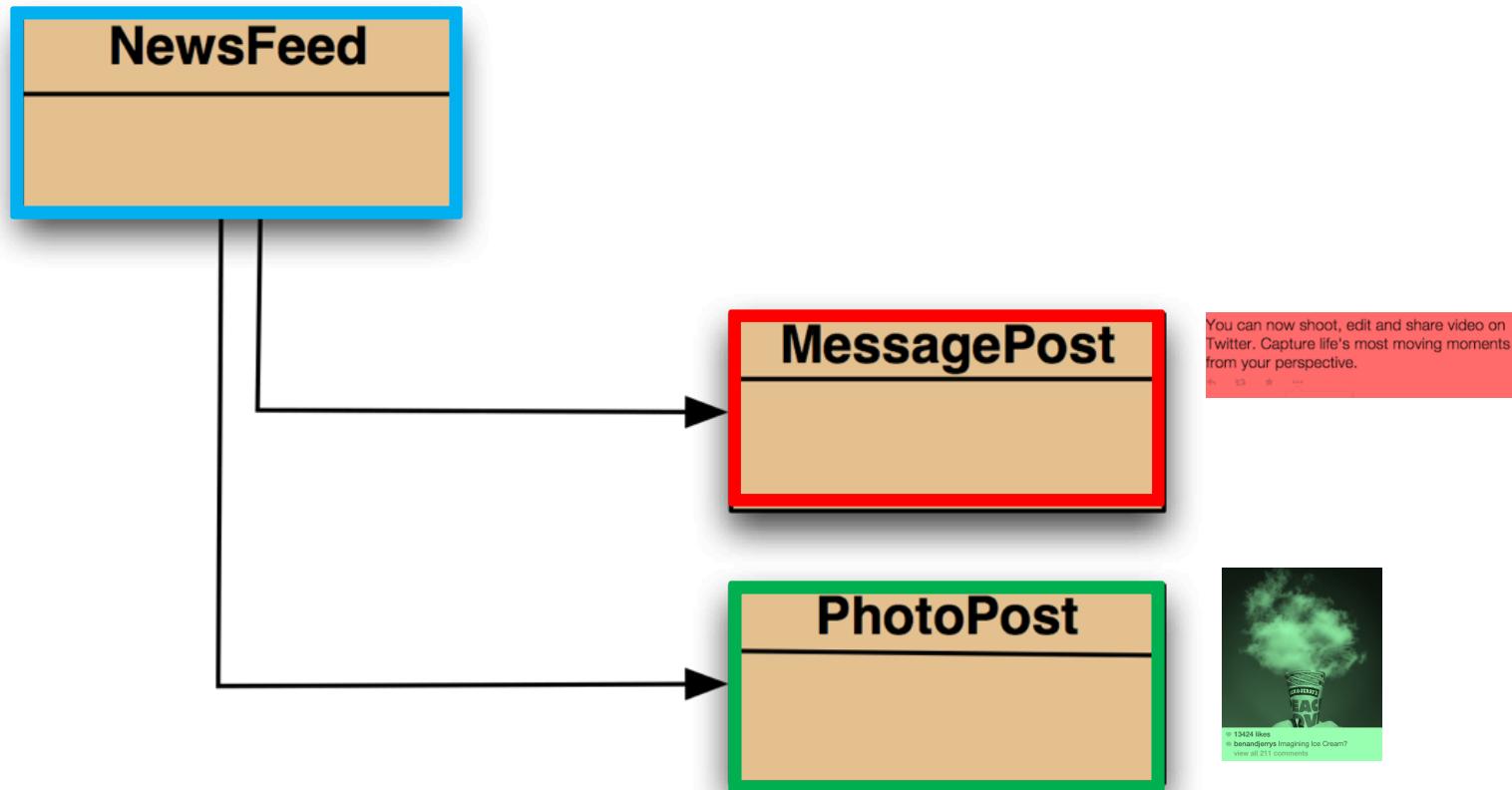
# Topic List

---

- 
1. Social Network V1
  2. Inheritance hierarchies
  3. Social Network V2
  4. Coding inheritance hierarchies
    - Super and subclasses
    - Using constructors in these hierarchies
  5. Social Network V3
    - Deeper hierarchies
    - Advantages of using inheritance
  6. Subtyping and Substitution
  7. Polymorphic variables / Collections
    - Includes casting, wrapper classes, autoboxing /unboxing

# Recap: Social Network V1 - Class diagram

---



# Critique of Social Network V1

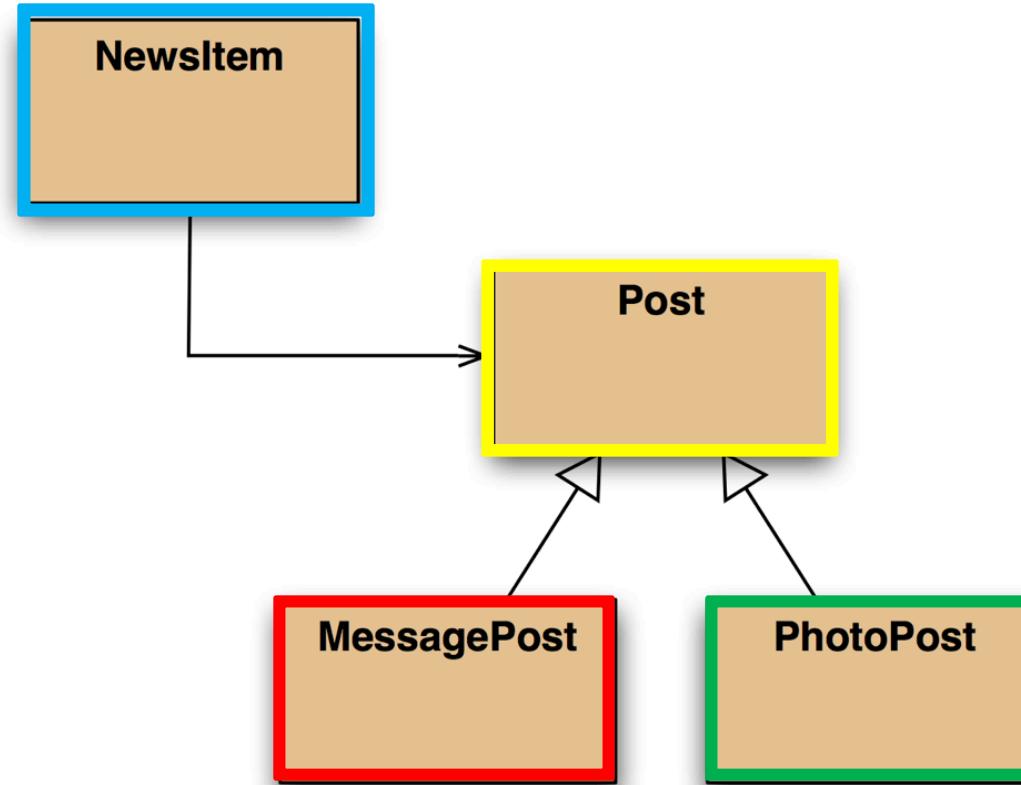
---

- 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.

v2

SN

# Social Network V2 - Class diagram

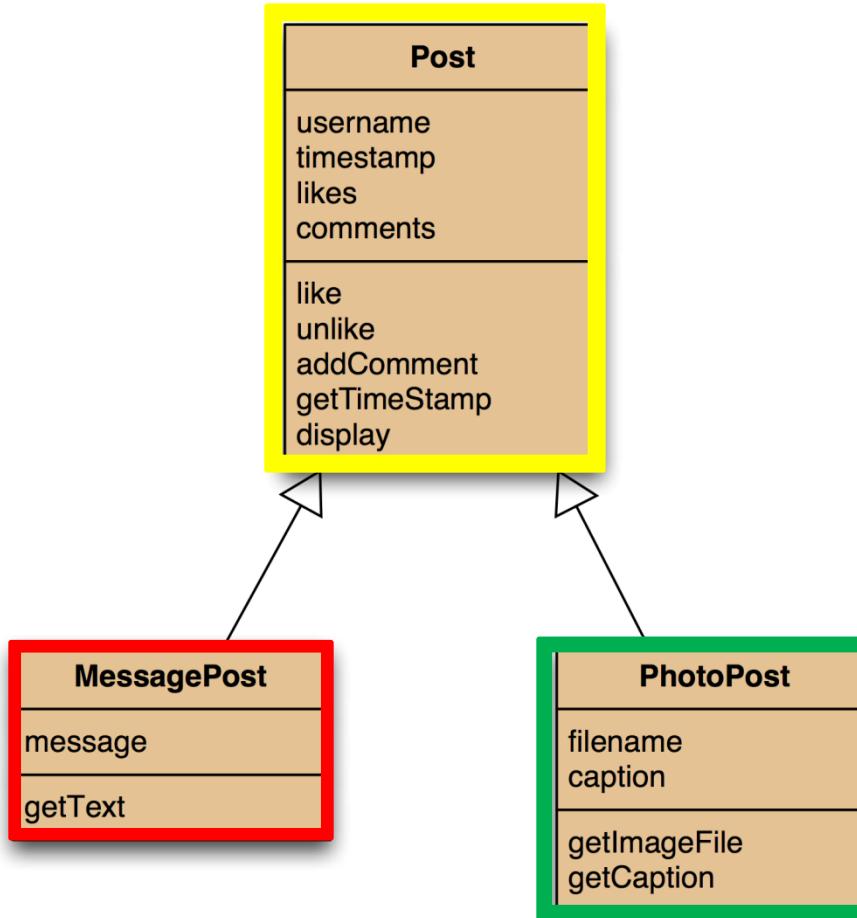


You can now shoot, edit and share video on Twitter. Capture life's most moving moments from your perspective.



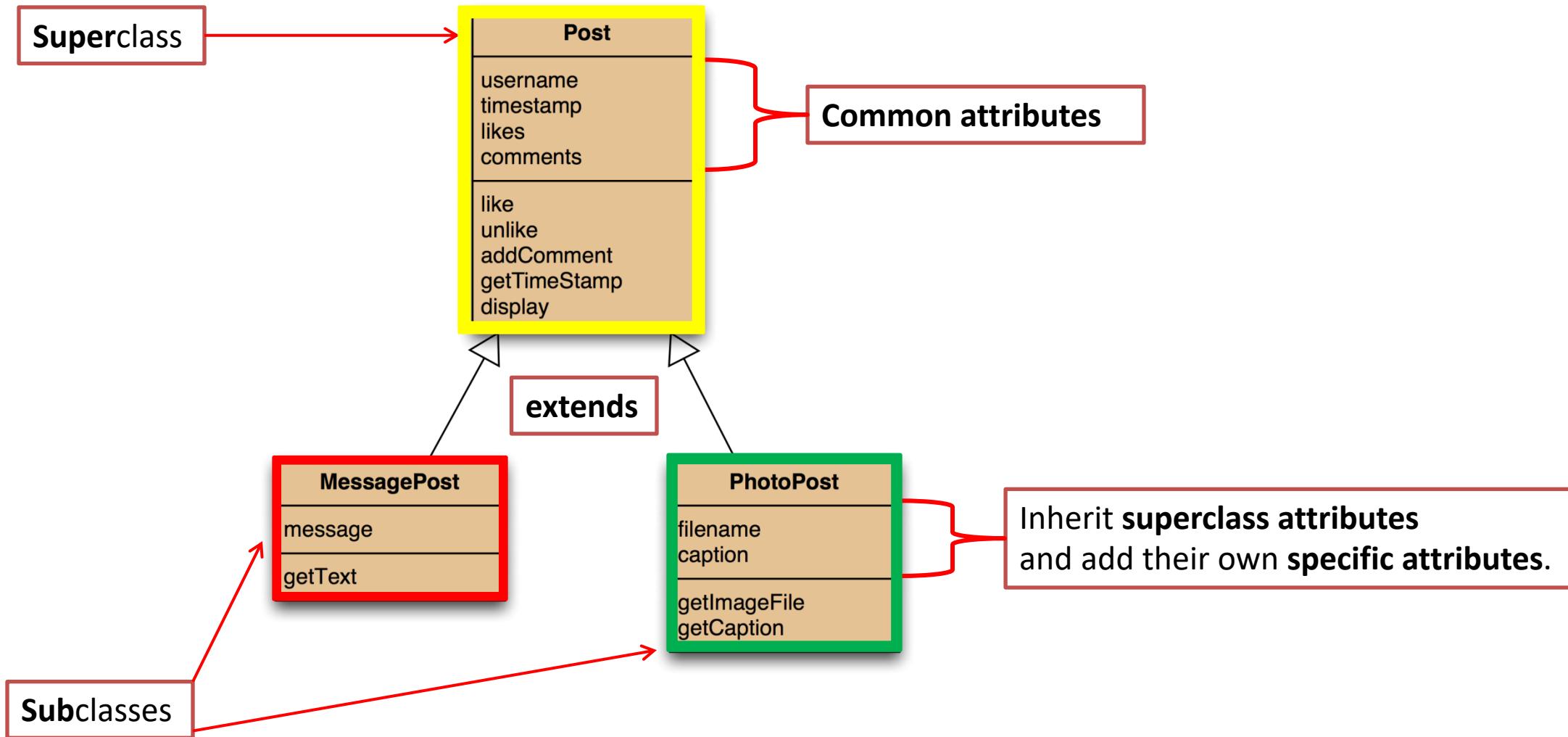
# Social NetworkV2 - Using inheritance

---



# Social NetworkV2 - Using inheritance

---



# Social Network V2 – Inheritance Summary

---

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

# Topic List

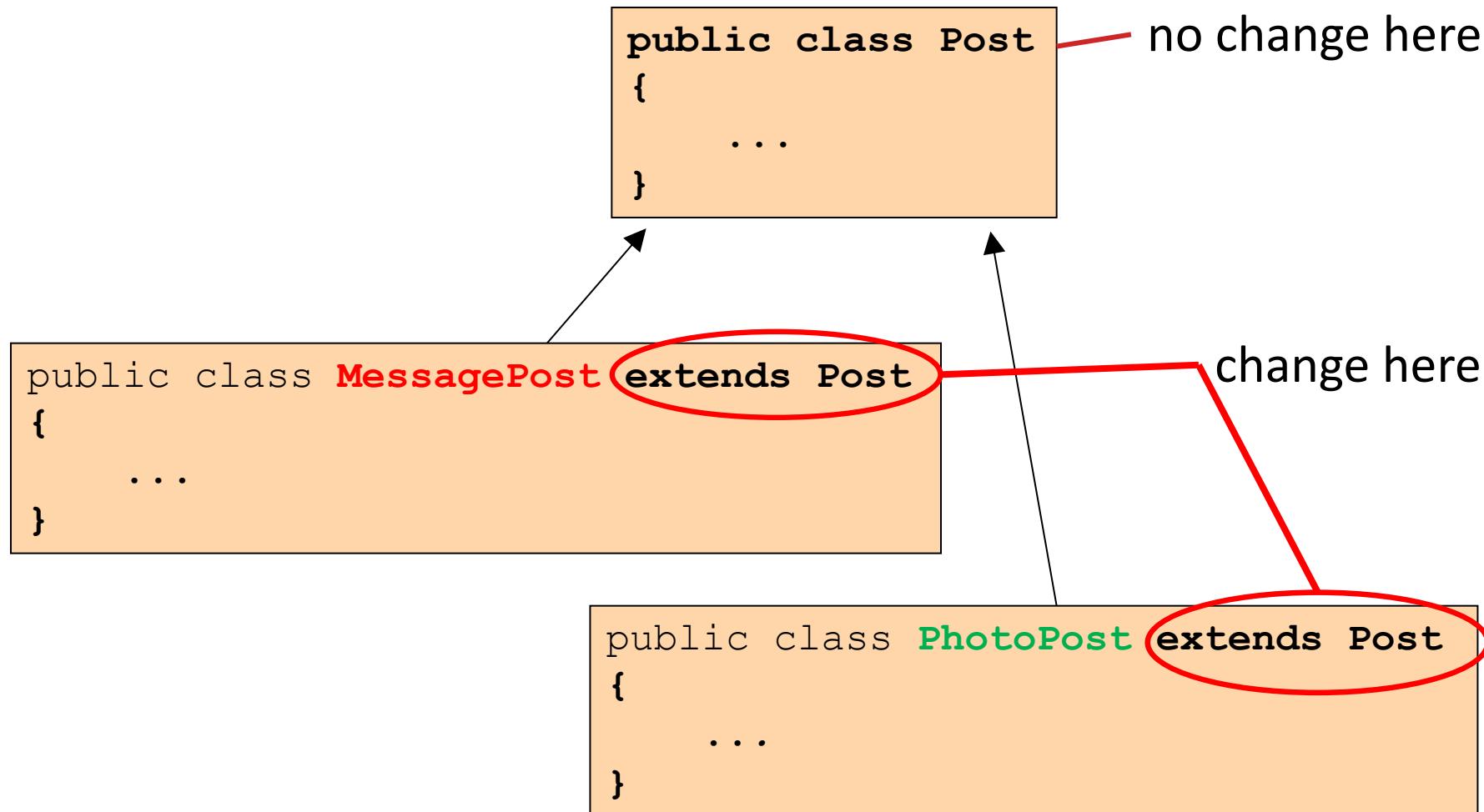
---

1. Social Network V1
2. Inheritance hierarchies
3. Social Network V2
4. Coding inheritance hierarchies
  - **Super** and **subclasses**
  - Using **constructors** in these hierarchies
5. Social Network V3
  - Deeper hierarchies
  - Advantages of using inheritance
6. Subtyping and Substitution
7. Polymorphic variables / Collections
  - Includes casting, wrapper classes, autoboxing /unboxing



# Inheritance in Java - `extends`

---



# Superclass

---

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

    // constructor and methods omitted.
}
```

we define common fields in superclass

# Subclasses

---

```
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.
}
```

we add subclass fields; inherit superclass fields  
subclass objects will have all fields

# Inheritance and Constructors

## - superclass

```
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
}
```

# Inheritance and Constructors

## - subclass

```
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
}
```

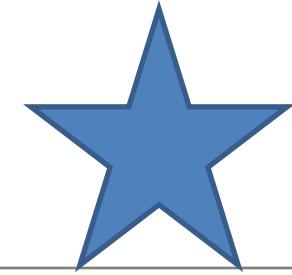


subclass: must call superclass constructor!

Must take values for all fields that we want to initialise.

# Superclass constructor call

---



- Subclass constructors **must** 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
- 'super' call must be the **first statement** in the subclass constructor.

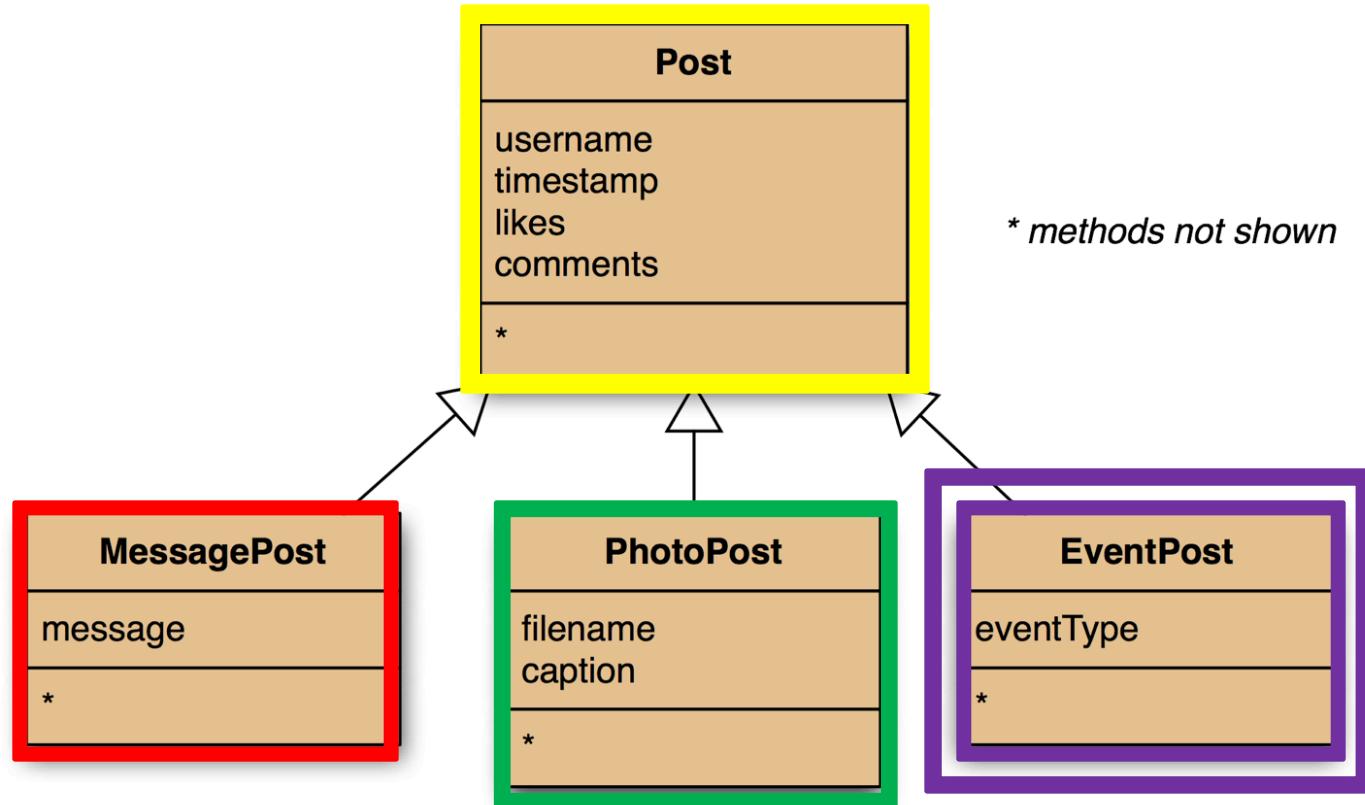
# Topic List

---

1. Social Network V1
2. Inheritance hierarchies
3. Social Network V2
4. Coding inheritance hierarchies
  - Super and subclasses
  - Using constructors in these hierarchies
5. Social Network V3
  - Deeper hierarchies
  - Advantages of using inheritance
6. Subtyping and Substitution
7. Polymorphic variables / Collections
  - Includes casting, wrapper classes, autoboxing /unboxing

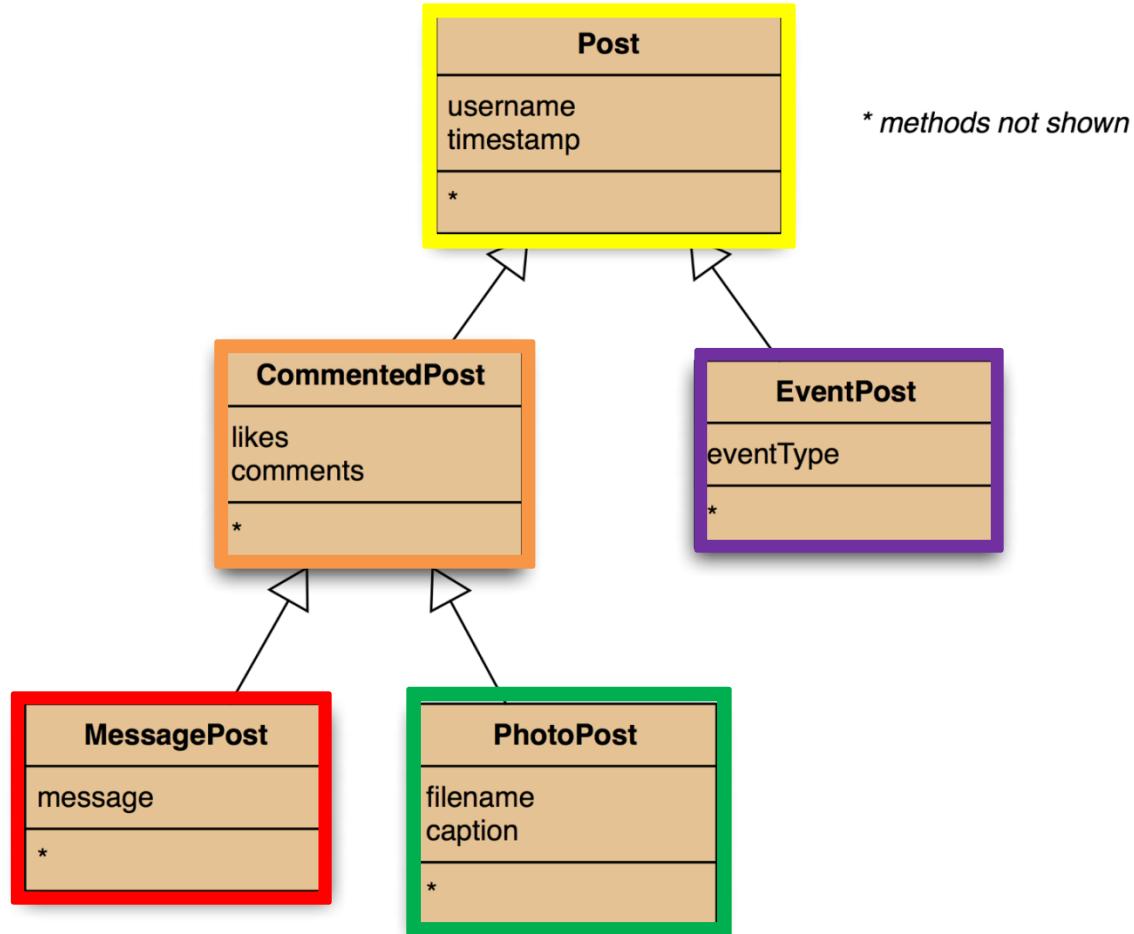


# Social Network V3 - Adding more item types



# Social Network V3 - Deeper hierarchies

---



# Topic List

---

1. Social Network V1
2. Inheritance hierarchies
3. Social Network V2
4. Coding inheritance hierarchies
  - Super and subclasses
  - Using constructors in these hierarchies
5. Social Network V3
  - Deeper hierarchies
  - Advantages of using inheritance
6. Subtyping and Substitution
7. Polymorphic variables / Collections
  - Includes casting, wrapper classes, autoboxing /unboxing



# Advantages of inheritance

---

Inheritance (so far) helps with:

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

```
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);
    }

    ...
}
```

## REVISED NewsFeed source code

*Code is simplified  
&  
code duplication  
in the client class is avoided!*

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

## REVISED NewsFeed source code



# Topic List

---

1. Social Network V1
2. Inheritance hierarchies
3. Social Network V2
4. Coding inheritance hierarchies
  - Super and subclasses
  - Using constructors in these hierarchies
5. Social Network V3
  - Deeper hierarchies
  - Advantages of using inheritance
6. Subtyping and Substitution
7. 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);
```

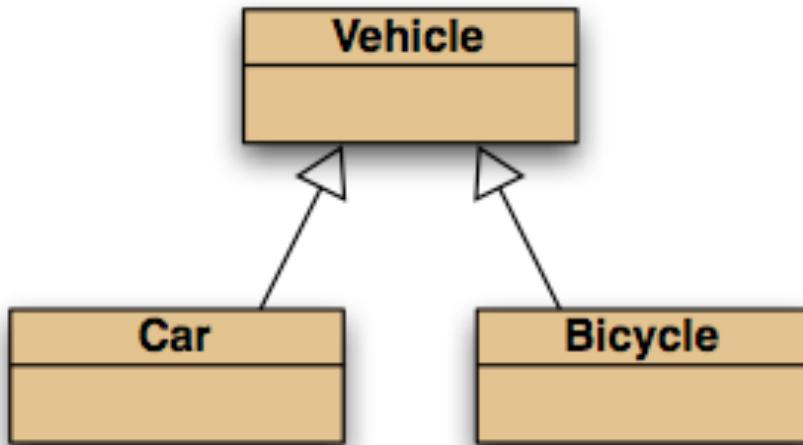
# Subclasses and subtyping

---

- 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*

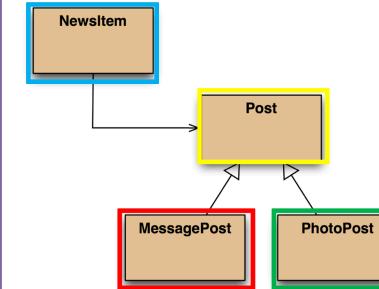
```
Vehicle v1 = new Vehicle();
Vehicle v2 = new Car();
Vehicle v3 = new Bicycle();
```

# Subtyping and parameter passing

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

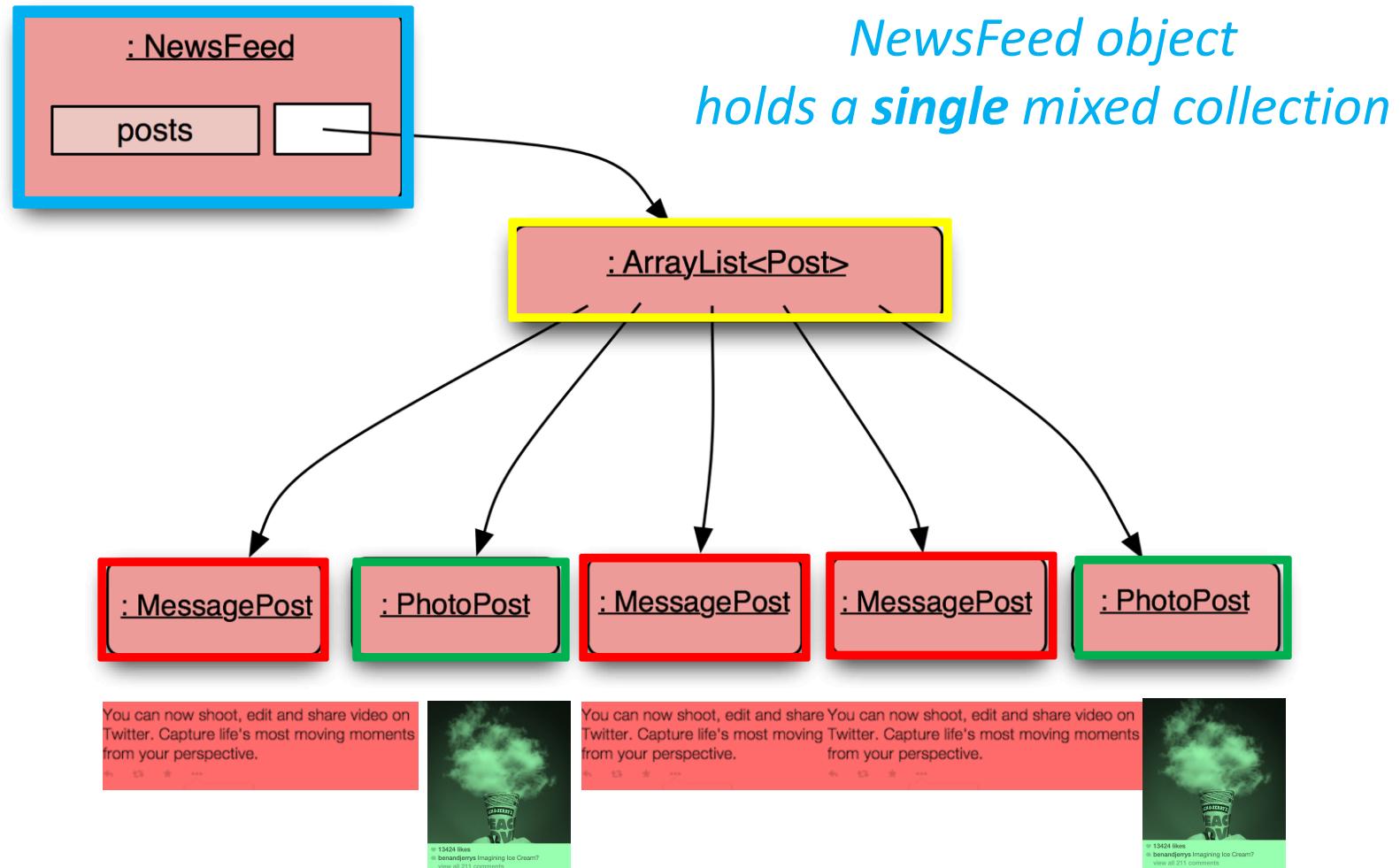
    PhotoPost photo = new PhotoPost(...);
    MessagePost message = new MessagePost(...);

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



*subclass objects may be used as actual parameters  
when a superclass is required.*

# Social Network V2 - Object diagram



# Topic List

---

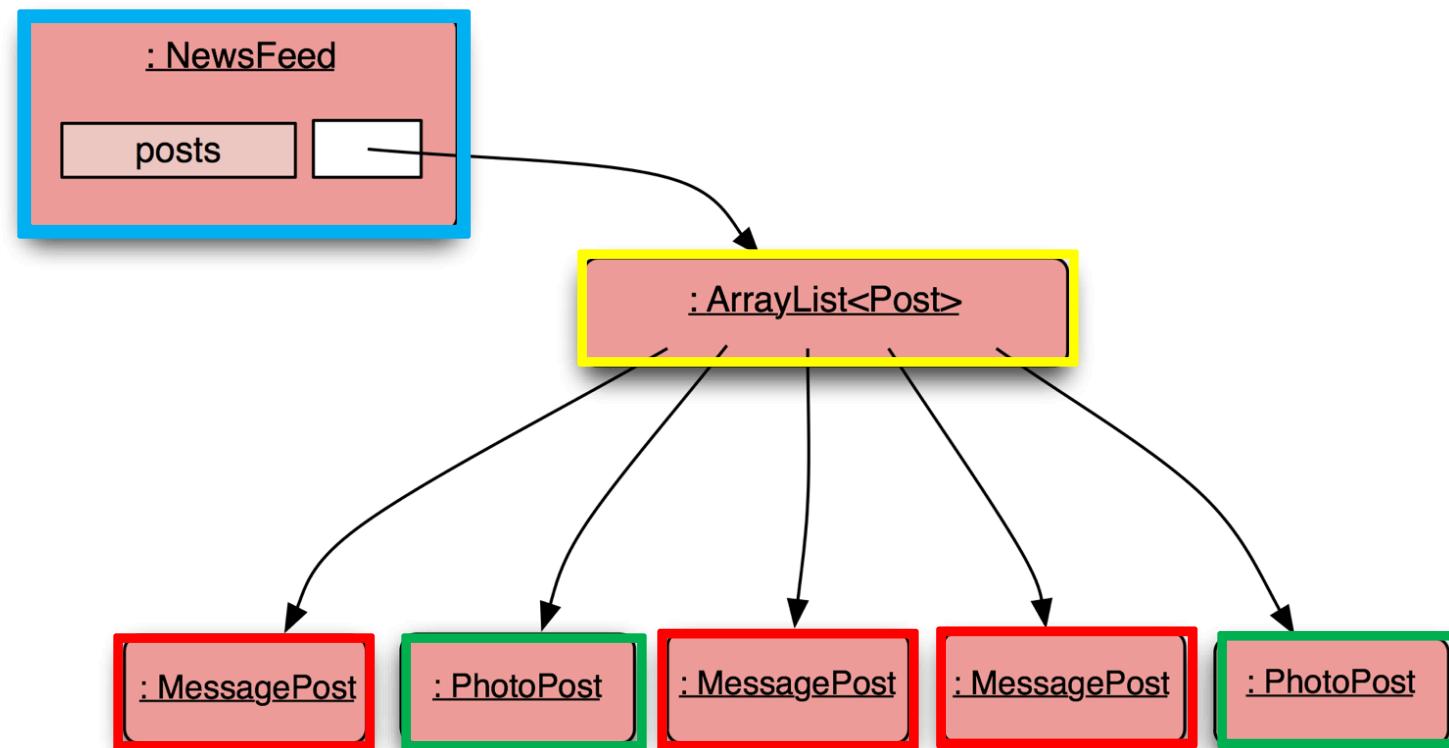
1. Social Network V1
2. Inheritance hierarchies
3. Social Network V2
4. Coding inheritance hierarchies
  - Super and subclasses
  - Using constructors in these hierarchies
5. Social Network V3
  - Deeper hierarchies
  - Advantages of using inheritance
6. Subtyping and Substitution
7. **Polymorphic**
  - a) Variables
  - b) Collections
    - casting, wrapper classes, autoboxing /unboxing

## 7 a) Polymorphic variables

---

- Object variables in Java are **polymorphic**
  - They can hold objects of
    - i. more than one **type**
    - ii. the declared **type**
    - iii. **subtypes** (*of the declared type*).

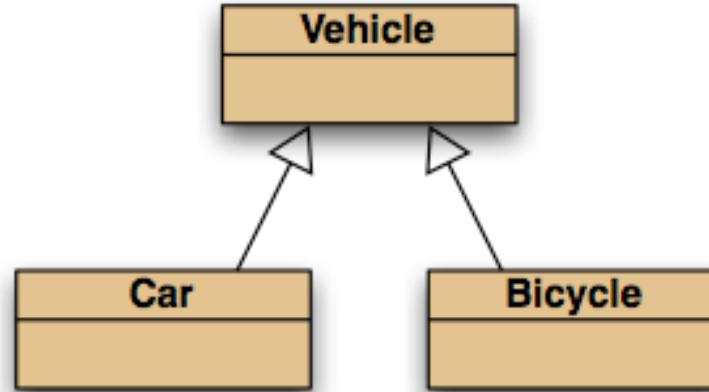
# Social Network V2 – polymorphic ArrayList of Post



# Casting

---

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



We can assign **subtype** to **supertype** (note arrow direction)!

```
v = c;  
// correct (car is-a vehicle)
```

But we cannot assign a **supertype** to **subtype** (cannot go against the arrows)!

```
c = v;  
// compile-time error!
```

Without (CASTING)

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

//casting...correct (only if the vehicle really is a Car!)

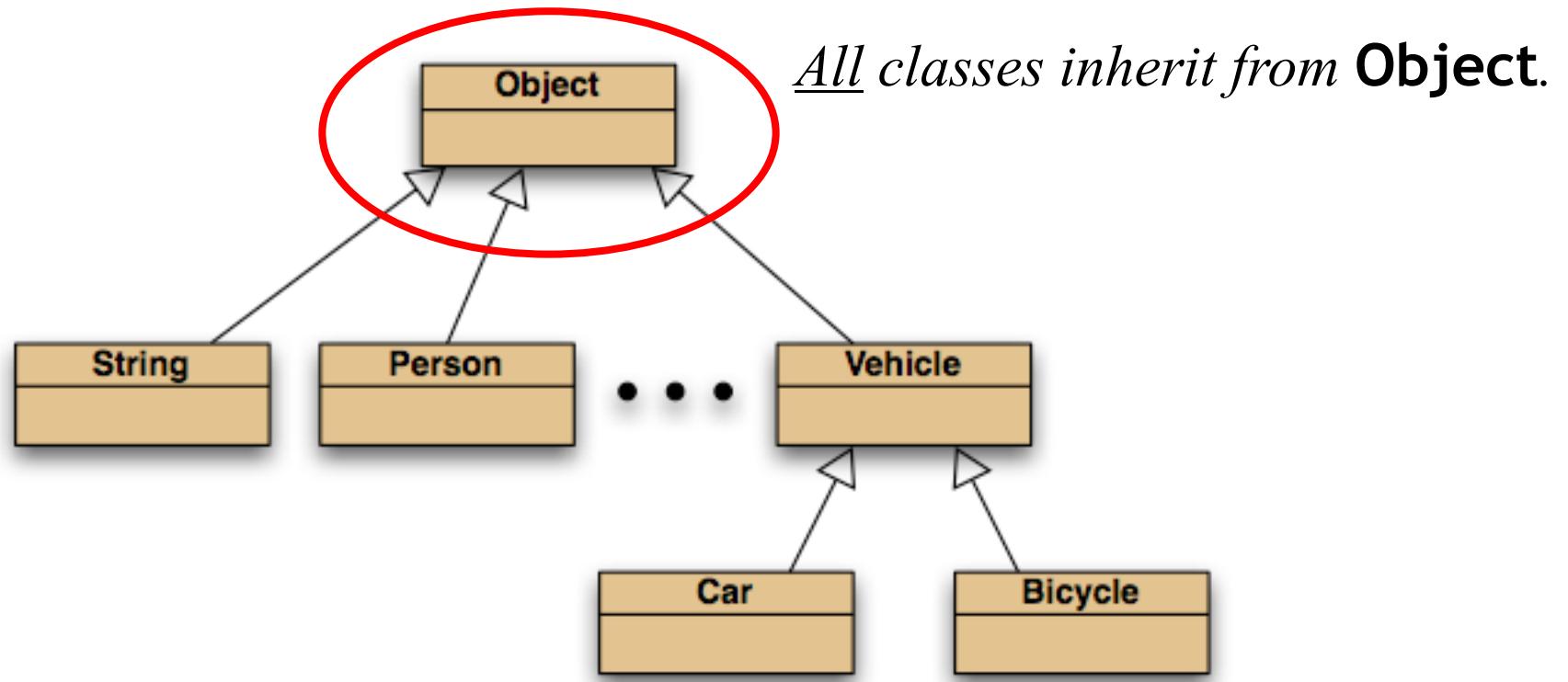
# Casting

---

- 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

---



# Topic List

---

1. Social Network V1
2. Inheritance hierarchies
3. Social Network V2
4. Coding inheritance hierarchies
  - Super and subclasses
  - Using constructors in these hierarchies
5. Social Network V3
  - Deeper hierarchies
  - Advantages of using inheritance
6. Subtyping and Substitution
7. **Polymorphic**
  - a) Variables
  - b) Collections
    - Casting
    - wrapper classes,
    - autoboxing /unboxing



## 7 b) 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...
  - we typically use a type parameter with the collection.

## 7 b) Polymorphic collections

---

- With a type parameter the degree of polymorphism:

**ArrayList<Post>** is **limited**.

- 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.?

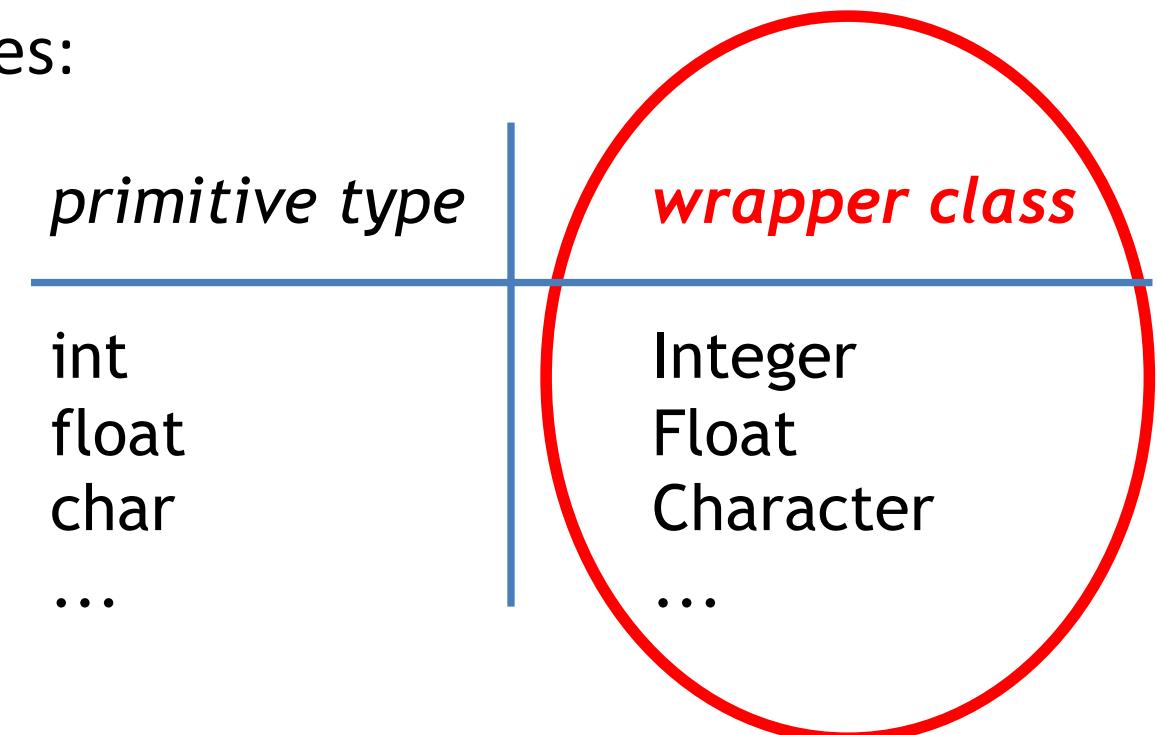
# Wrapper classes

---

- Primitive types are not object types.

Primitive-type values must be wrapped in objects, to be stored in a collection! 

- **Wrapper classes** exist for all primitive types:



Note that there is no simple mapping rule from primitive name to wrapper name!

# Wrapper classes

---

```
int i = 18;
```

```
Integer iwrap = new Integer(i); ← wrap the value
```

...

```
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

i.e. we don't have to worry about explicitly wrapping **mark** above

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

unboxing

Or explicitly unwrapping the first mark in the list **markList.get(0)**

# Summary

---

- a) Polymorphic Variables
- b) Polymorphic Collections
  - casting,
  - wrapper classes,
  - autoboxing /unboxing

**Any  
Questions?**



# 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).