Inheritance

Improving Structure with Inheritance

Produced by: Dr. Siobhán Drohan
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/)

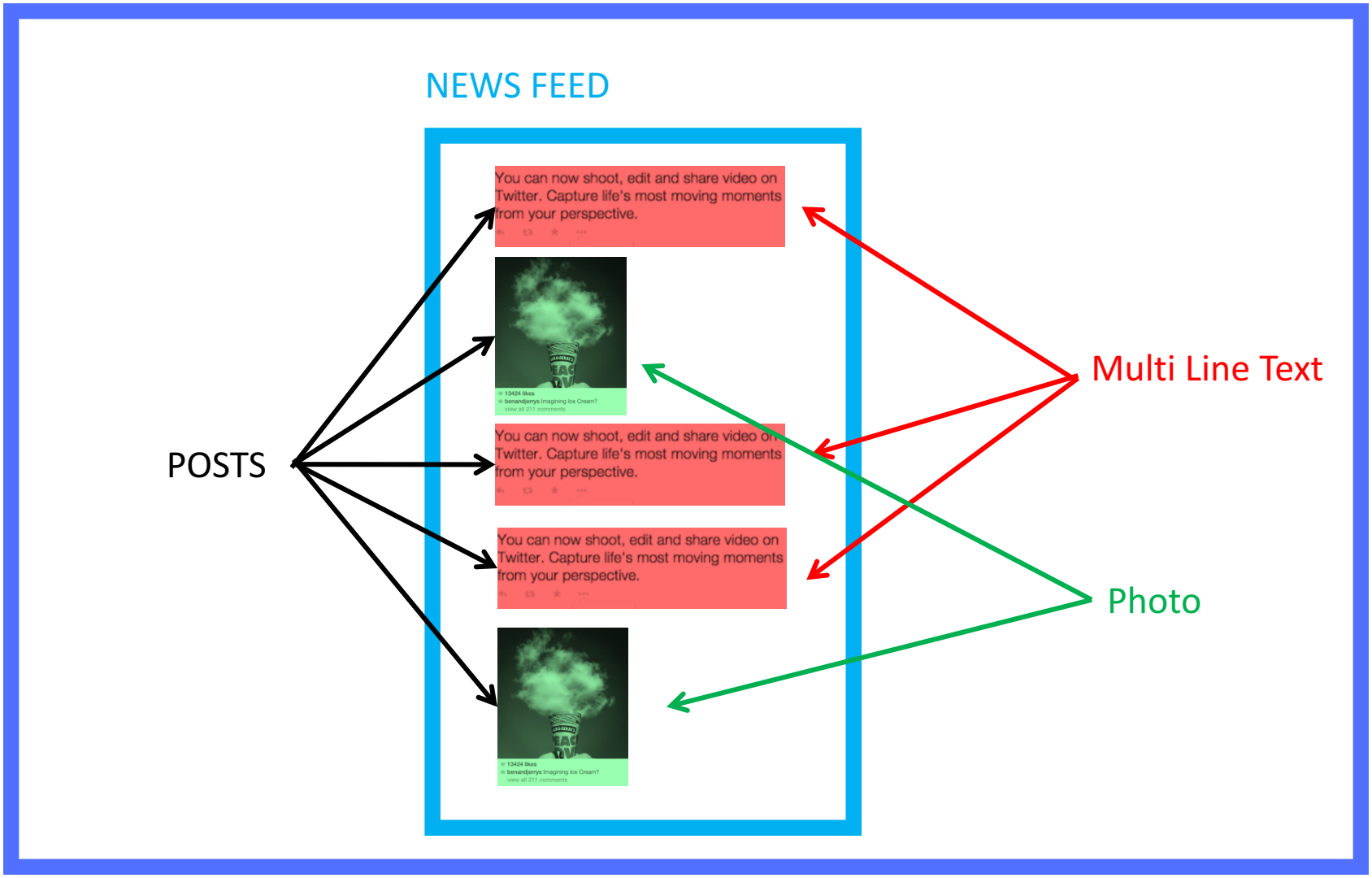
Topic List

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

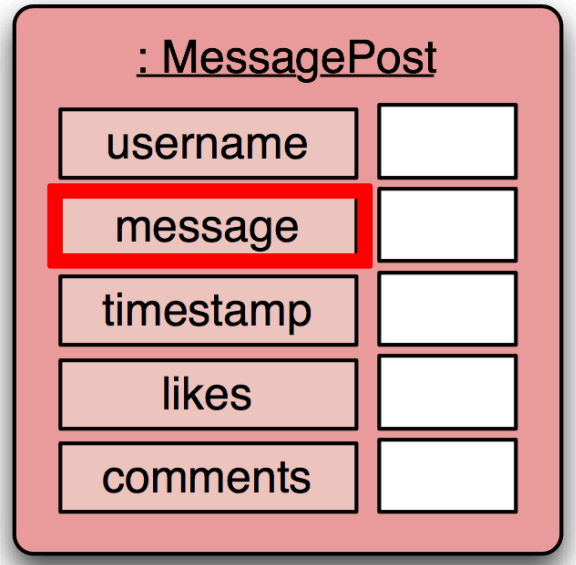
Social Network V1

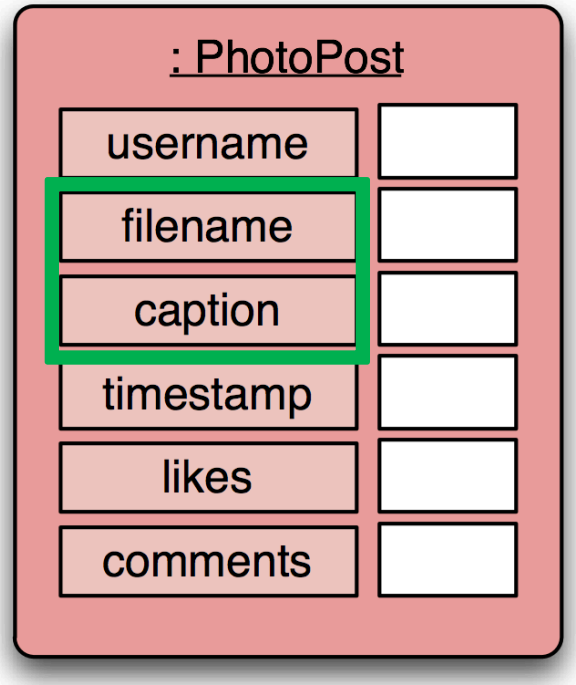
- A small, prototype SOCIAL NETWORK.
- Supports a News Feed with posts.
- POSTS:
  - MessagePost:
    - multi-line text message.
  - PhotoPost:
    - photo and caption.
  - Operations
    - e.g., search, display and remove.

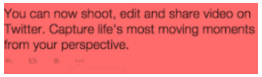
SOCIAL NETWORK



Social Network V1 - **Objects**



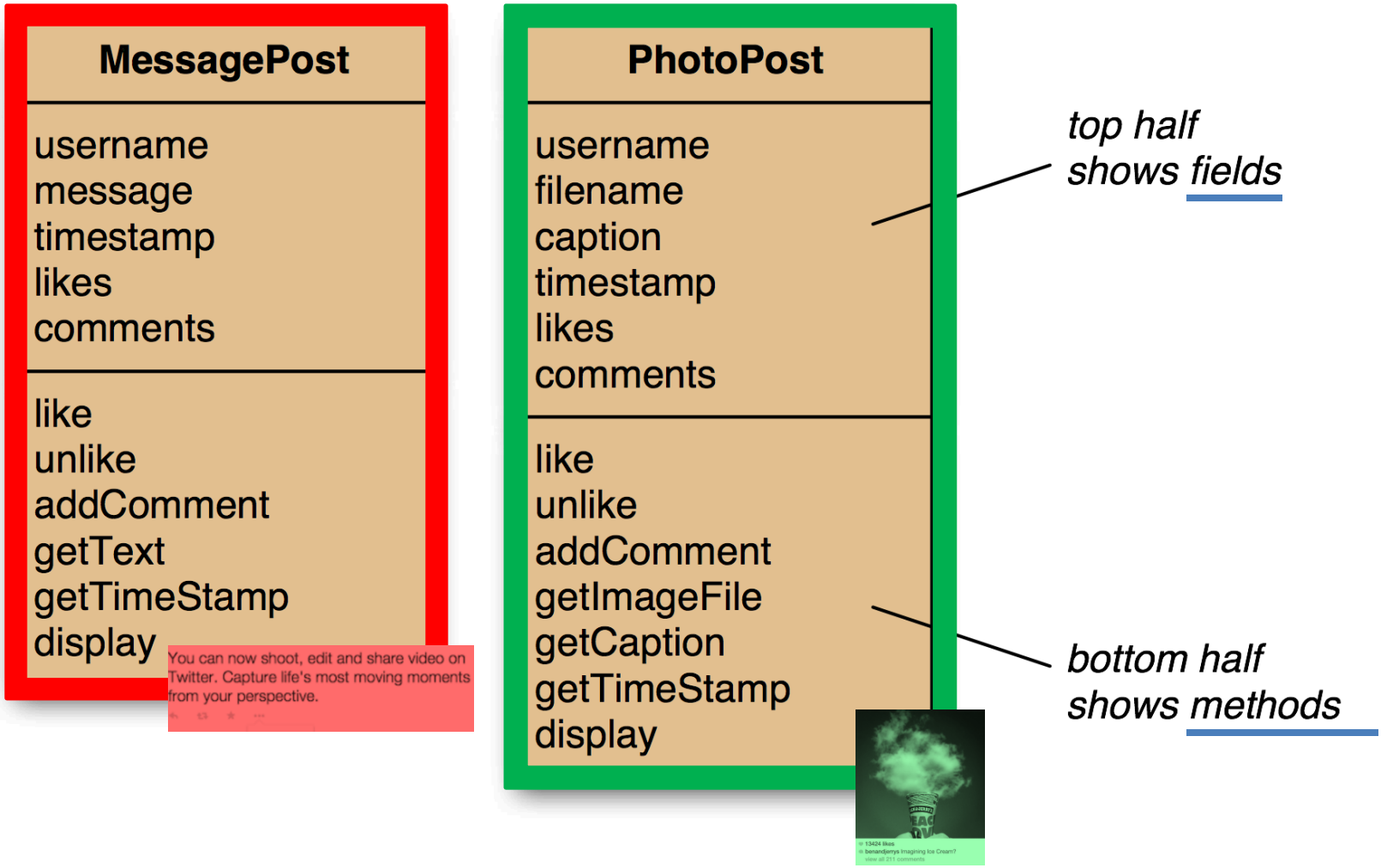




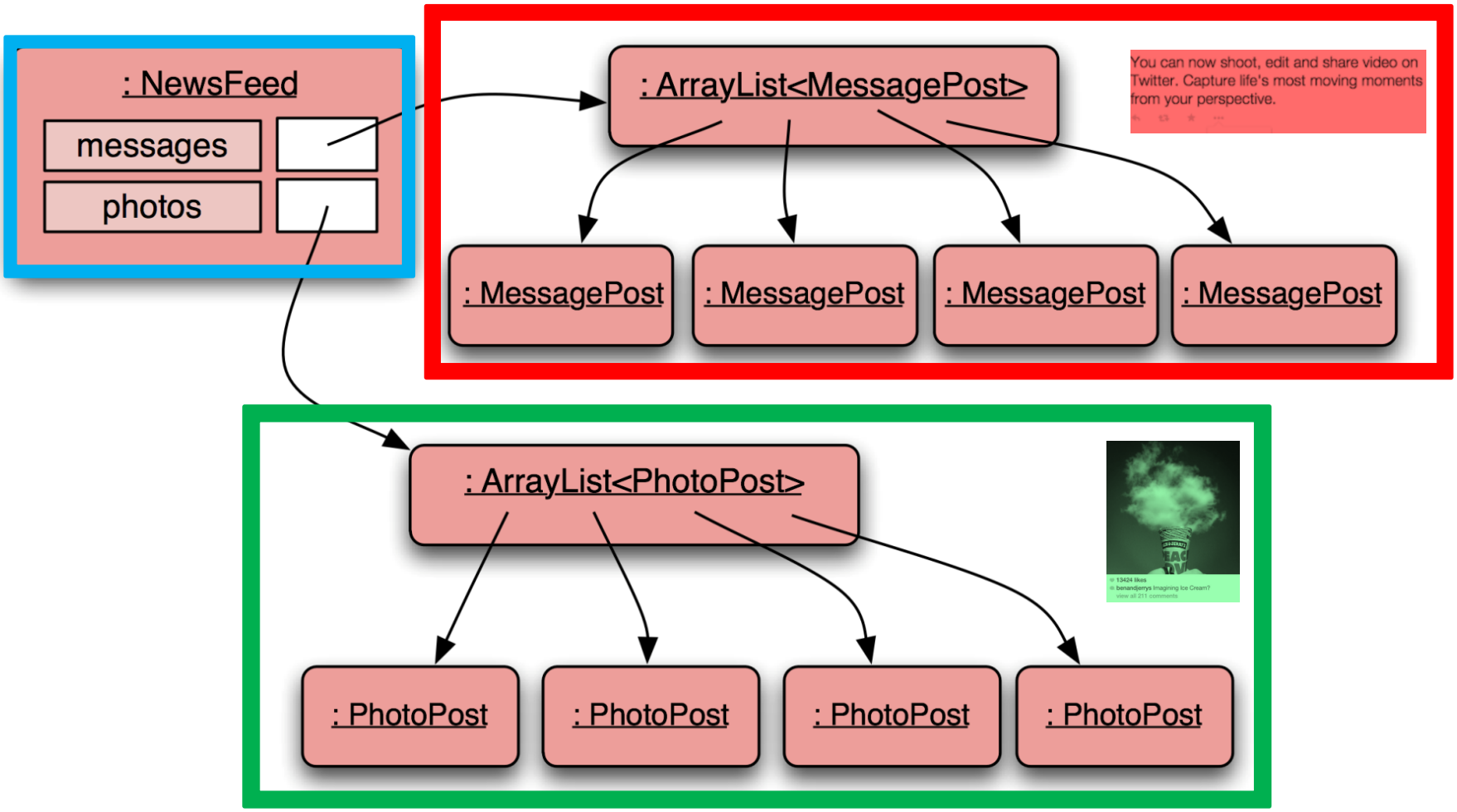
MessagePost:
multi-line text message.

PhotoPost:
photo and caption.

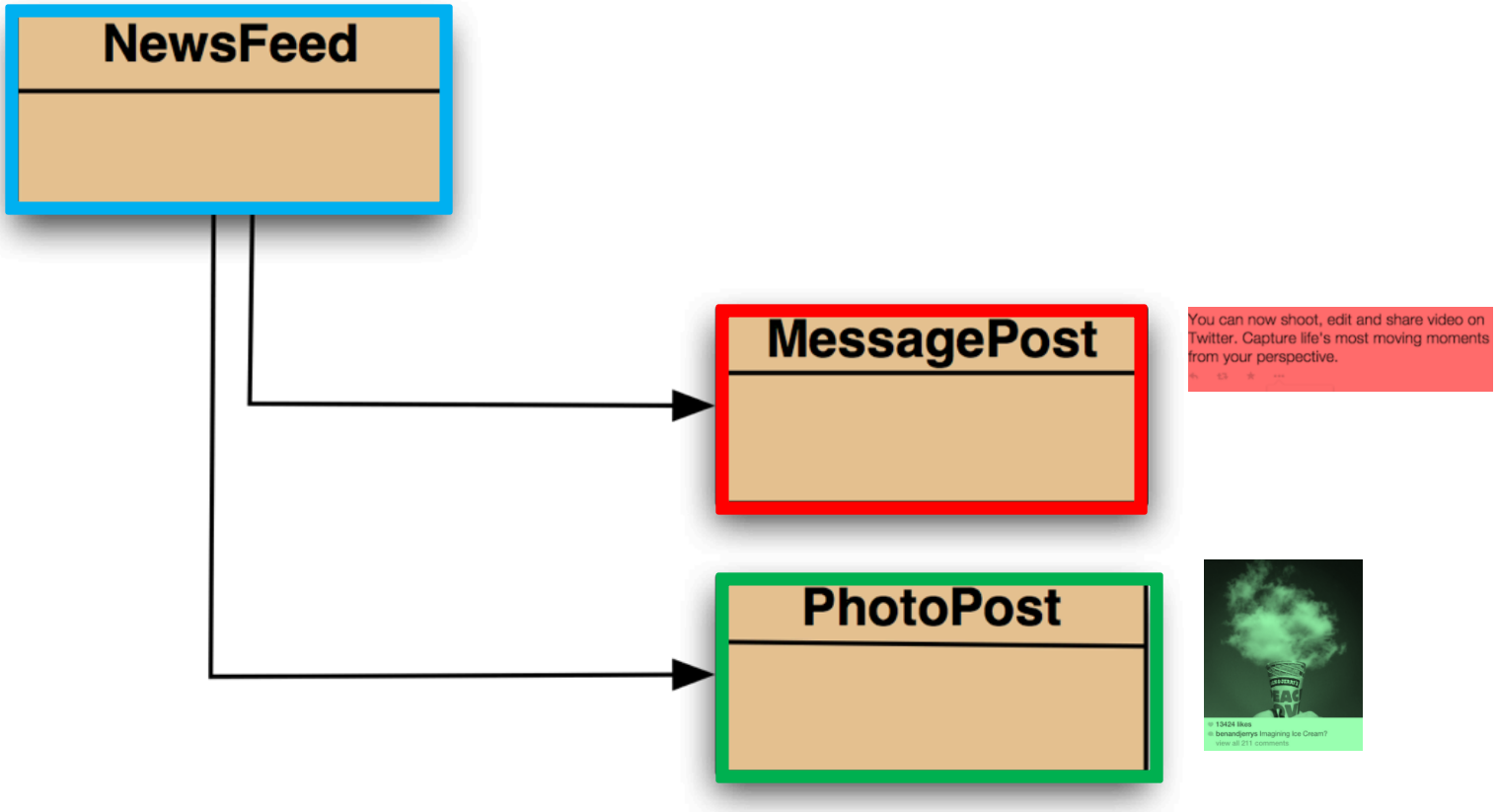
Social Network V1 - **Classes**



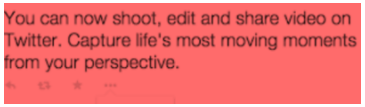
Social Network V1 - **Object model**



Social Network V1 - **Class diagram**



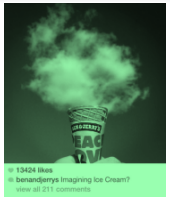
MessagePost source code



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() ...

    ...
}
```

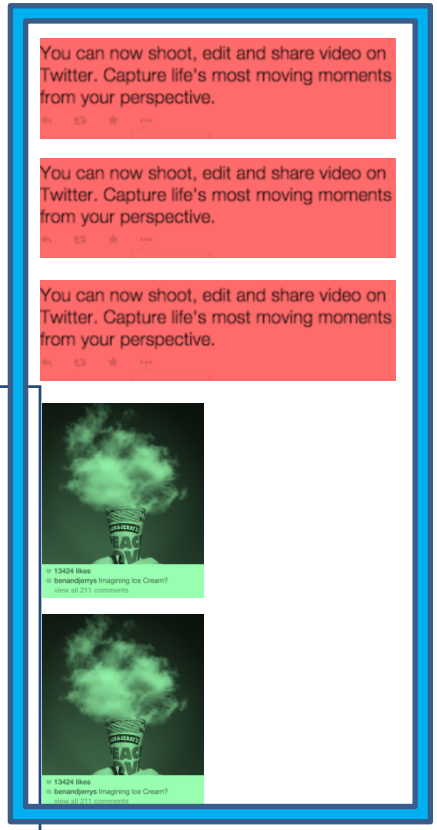
PhotoPost
source code



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

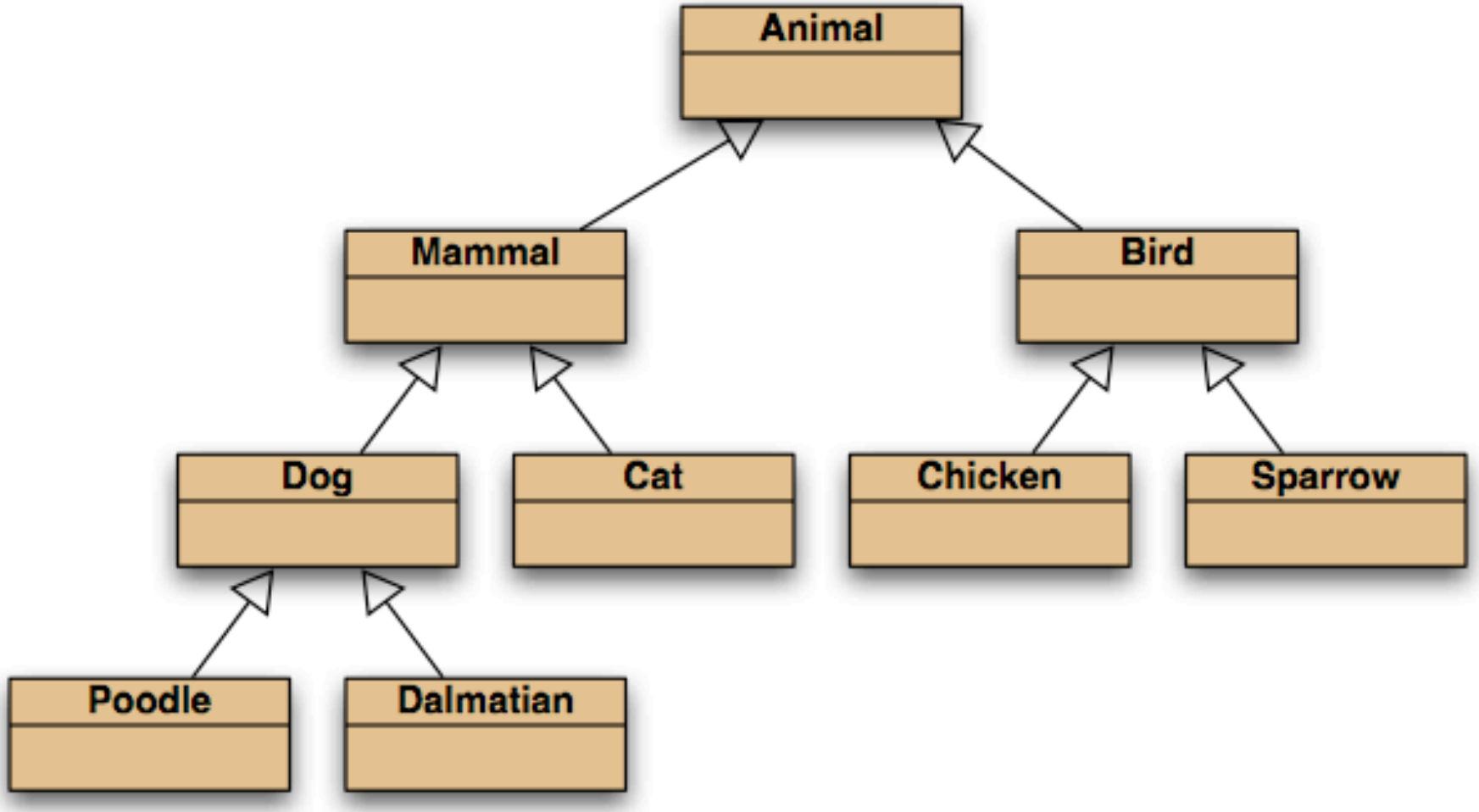



```
public class NewsFeed
{
    private ArrayList<MessagePost> messages;
    private ArrayList<PhotoPost> photos;
    ...
    public void show()
    {
        for(MessagePost message : messages) {
            message.display();
            System.out.println();  // empty line
between posts
        }

        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

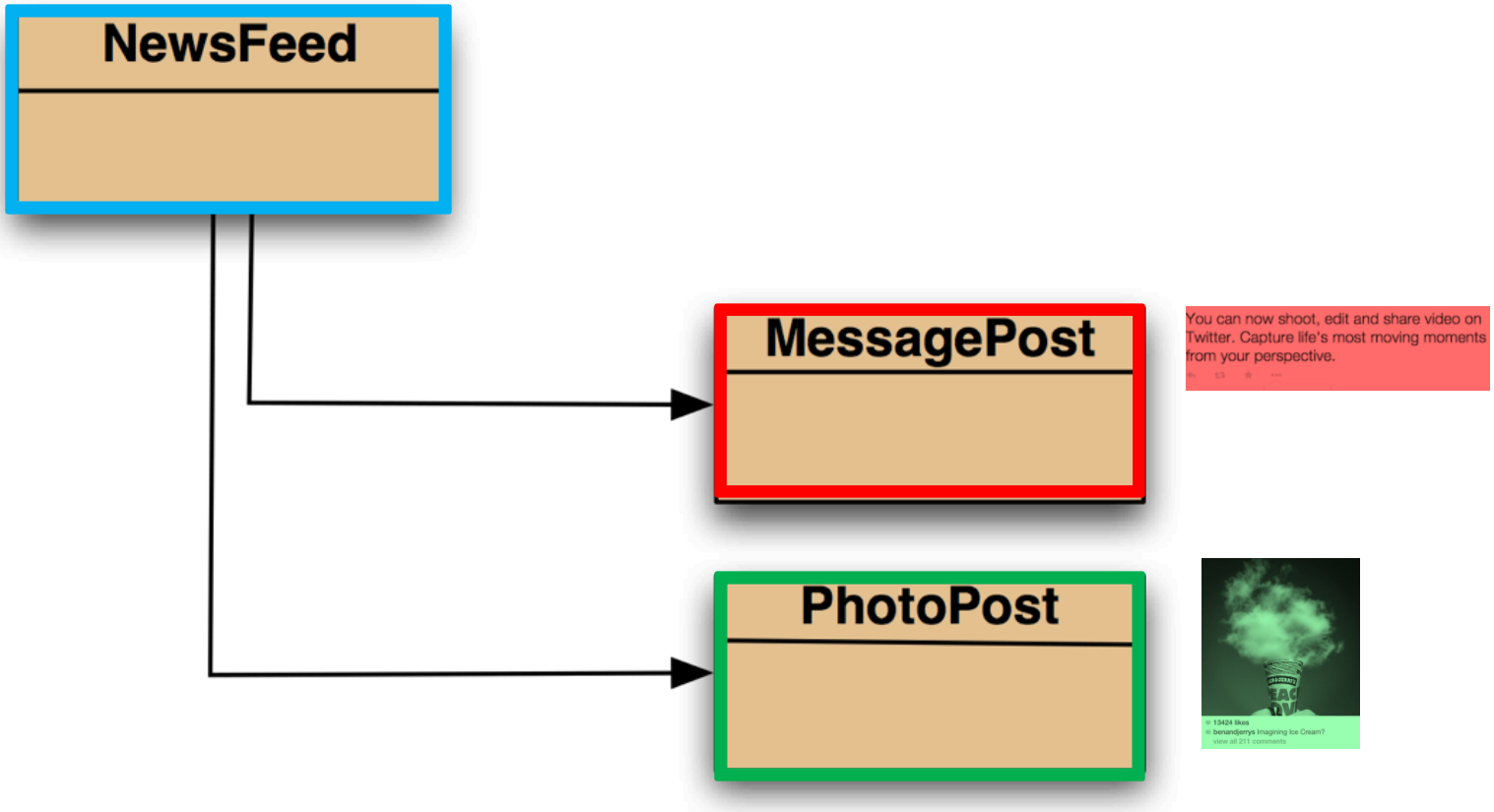
Inheritance hierarchies



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**

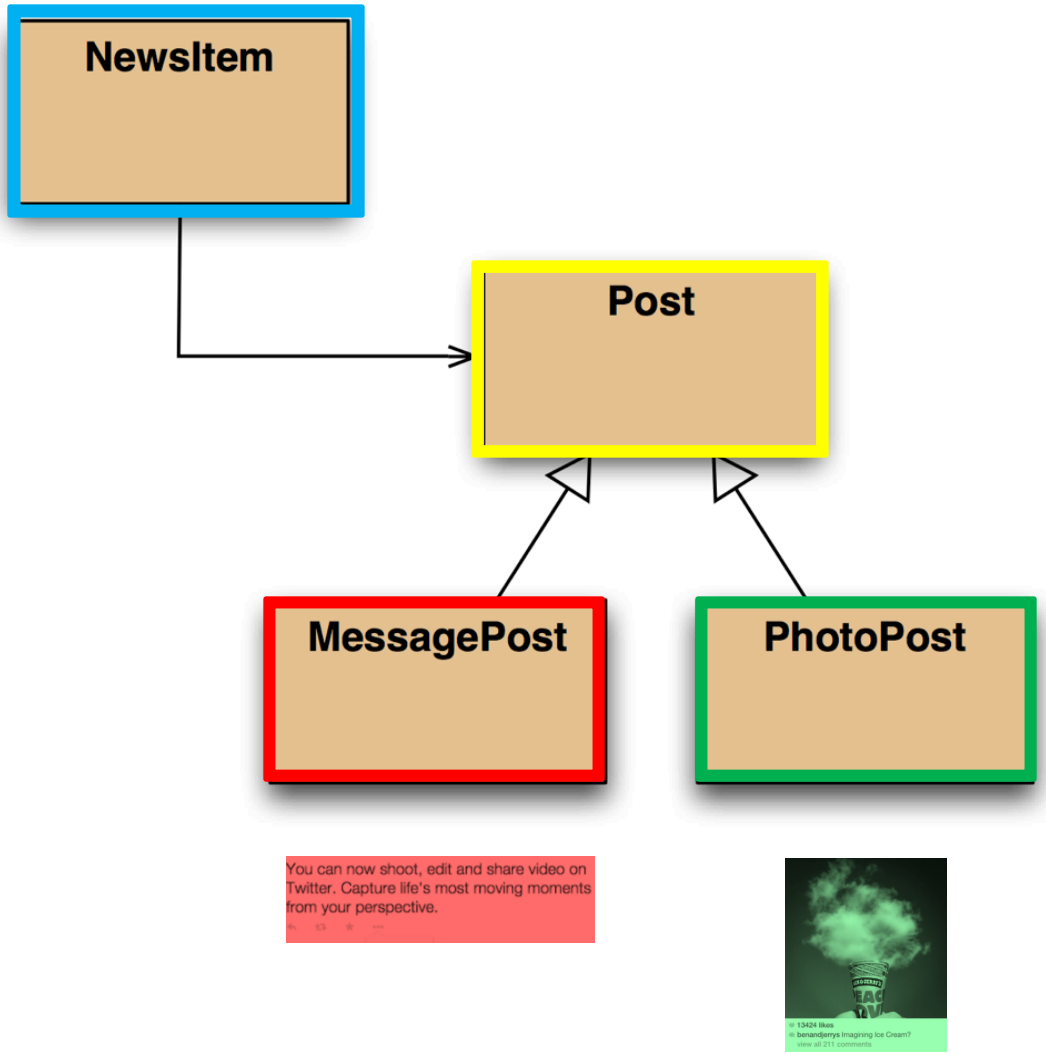


Critique of Social Network V1

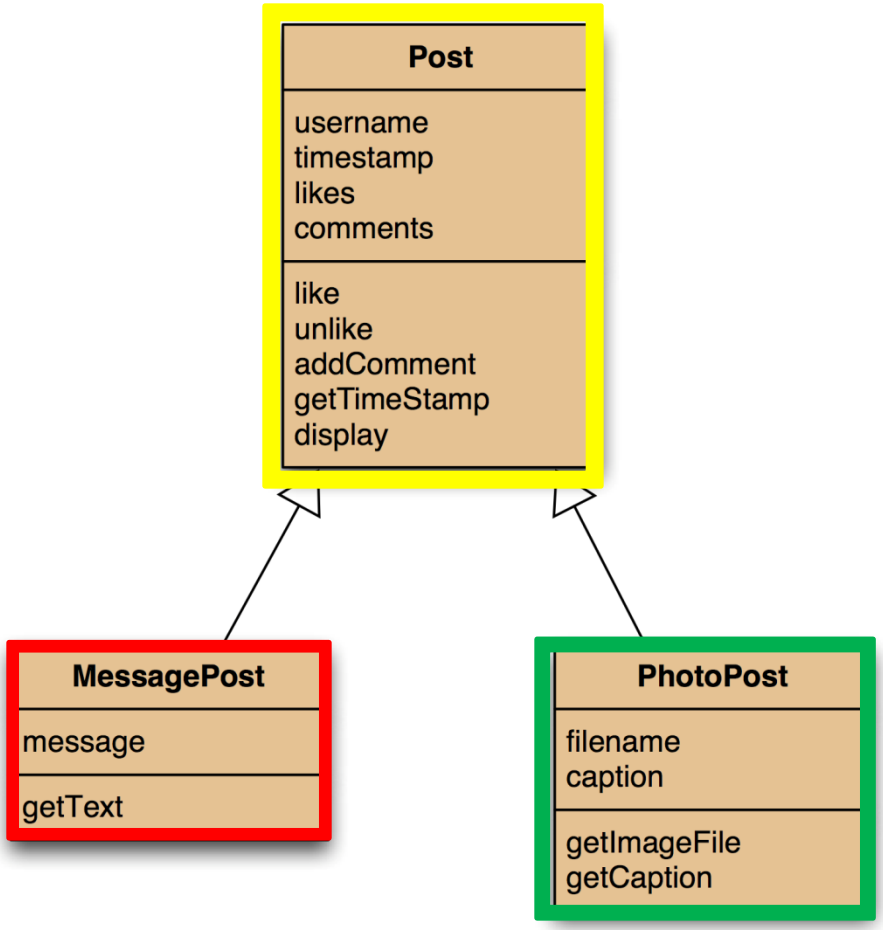
- Code duplication:
  - MessagePost and PhotoPost classes very similar (large parts are identical)
  - makes maintenance difficult/more work
  - introduces danger of bugs through incorrect maintenance

- Code duplication in NewsFeed class as well.

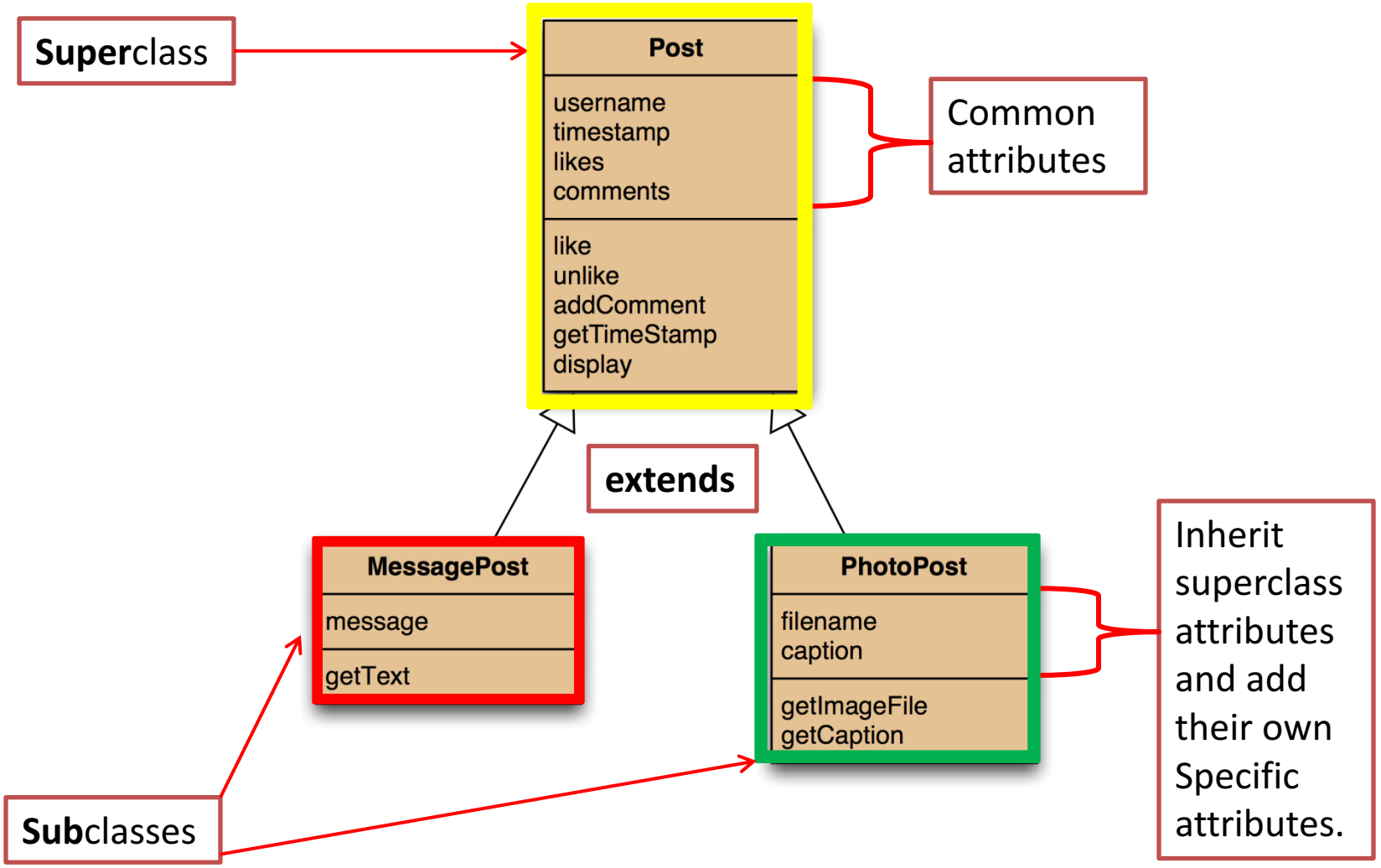
Social Network V2 - **Class** diagram



Social NetworkV2 - Using **inheritance**



Social NetworkV2 - Using **inheritance**



Social Network V2 – Inheritance Summary

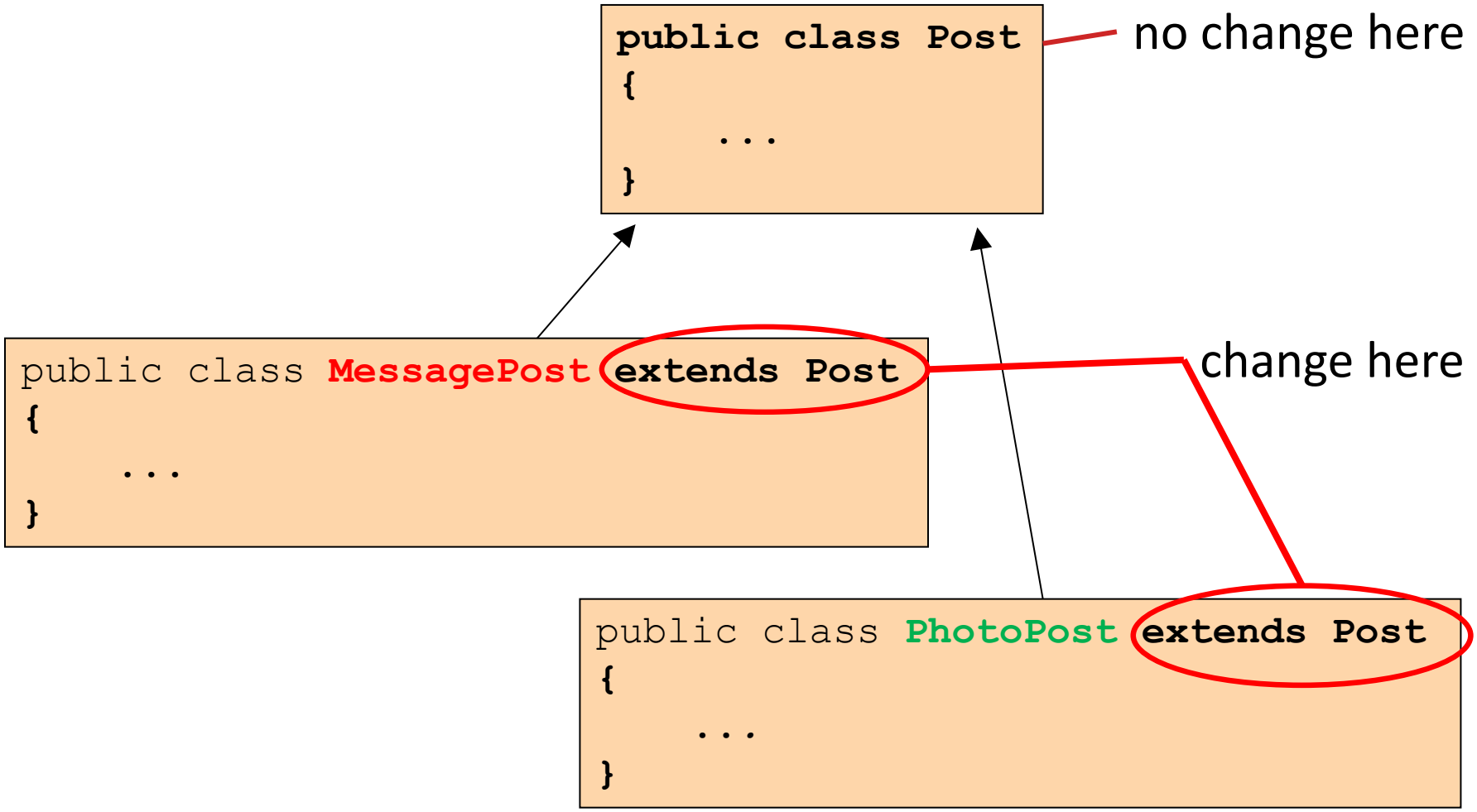
- define one **superclass**
  - `Post`
- define **subclasses** for
  - `MessagePost`
  - `PhotoPost`

- the **super**class
  - defines common attributes (via fields)
- the **sub**classes
  - **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.
}
```

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.
}
```

Inheritance and Constructors - **super**class

```
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 - **sub**class

```
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 must always contain a **'super'** call.

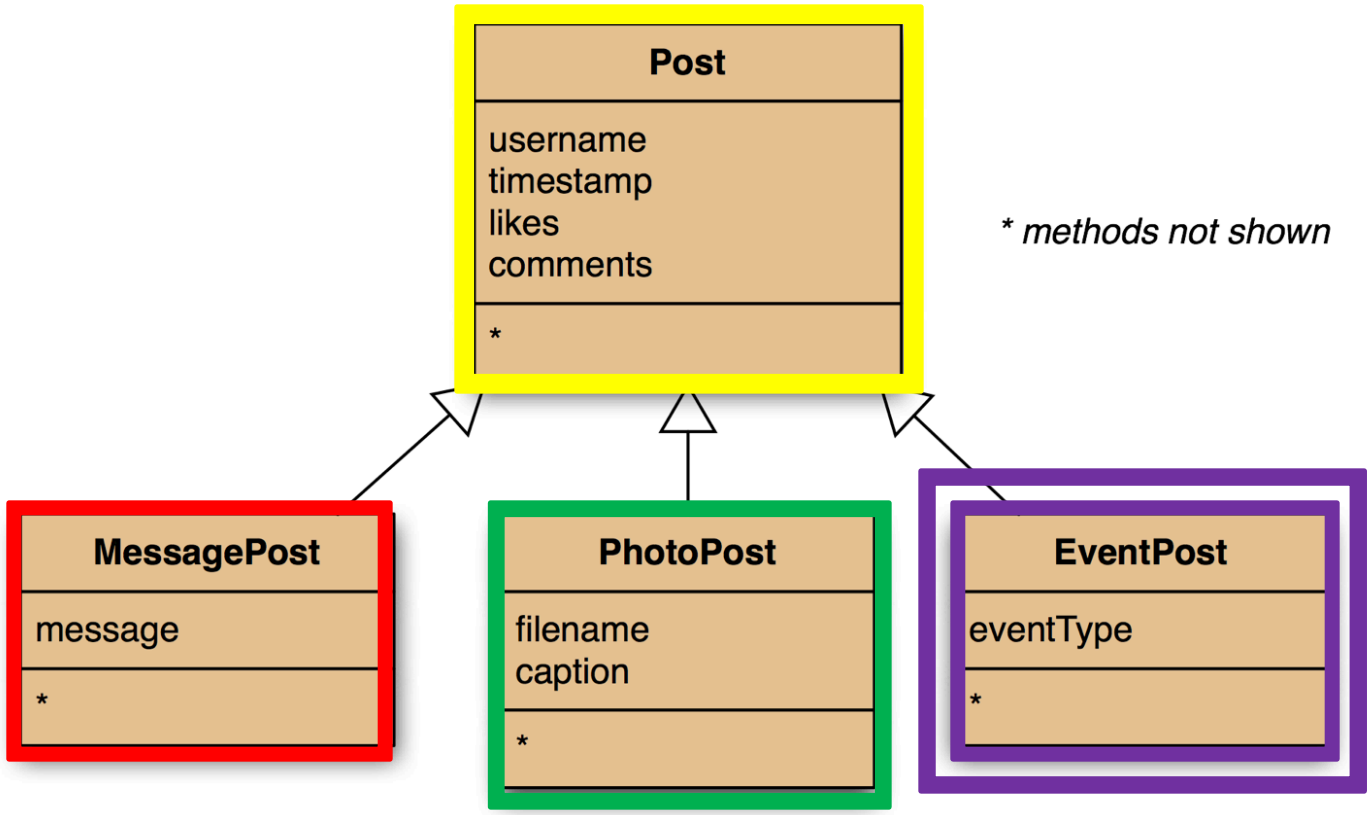
- If none is written, the compiler inserts one (without parameters)
  - works only, if the superclass has a constructor without parameters

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

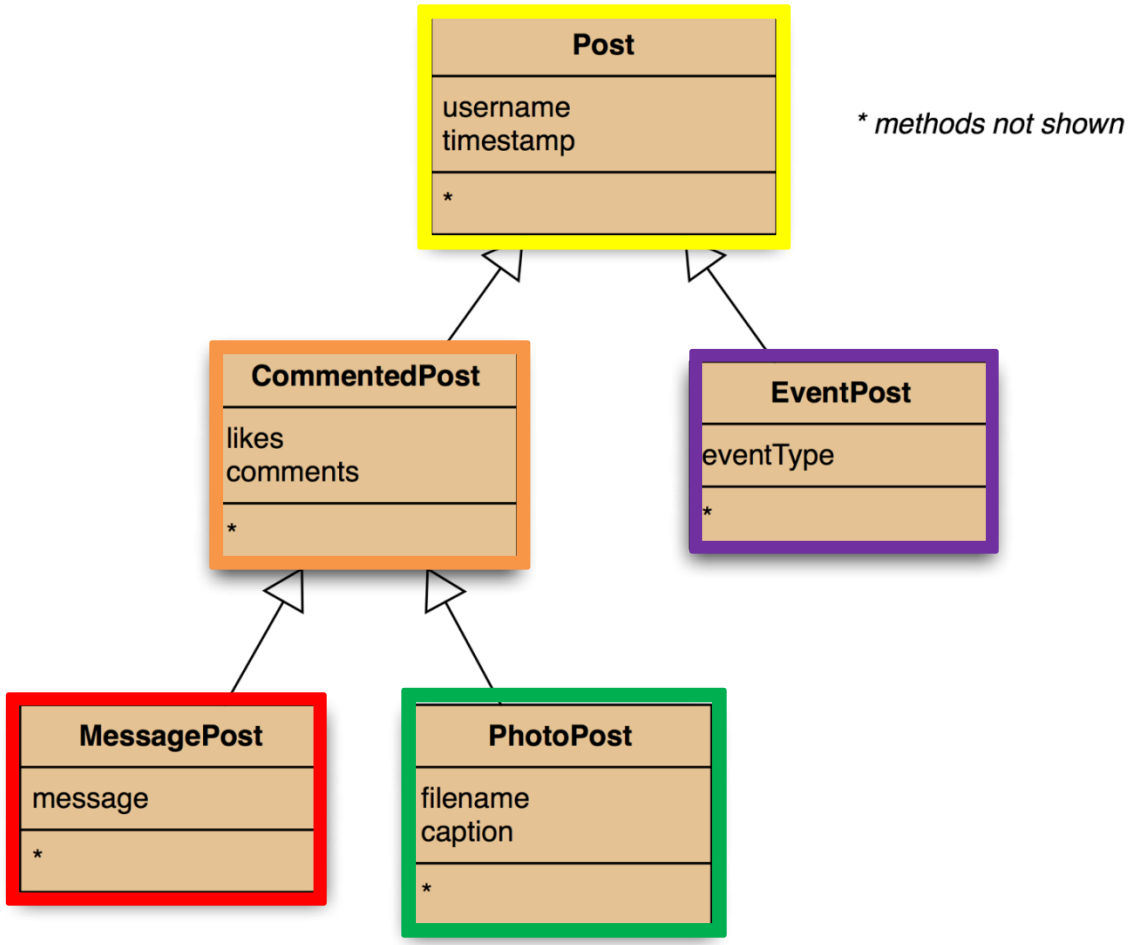
Topic List

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

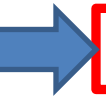
Code is simplified & code duplication in the client class is avoided!

REVISED NewsFeed source code

```
/**
 * 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

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)
```

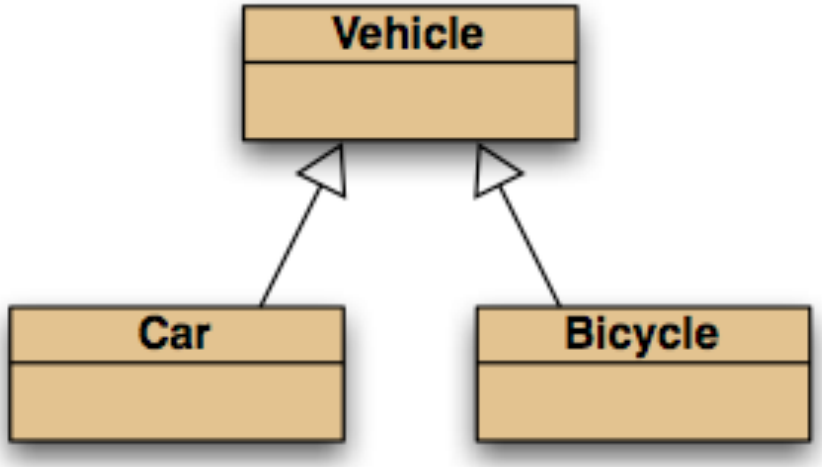
We call this method with:

```
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

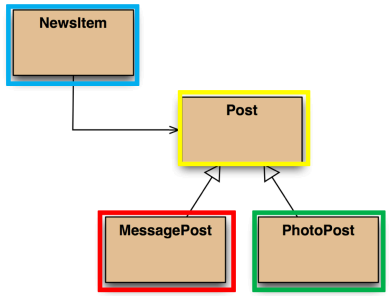


***sub**class objects*
may be assigned to
***super**class variables*

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

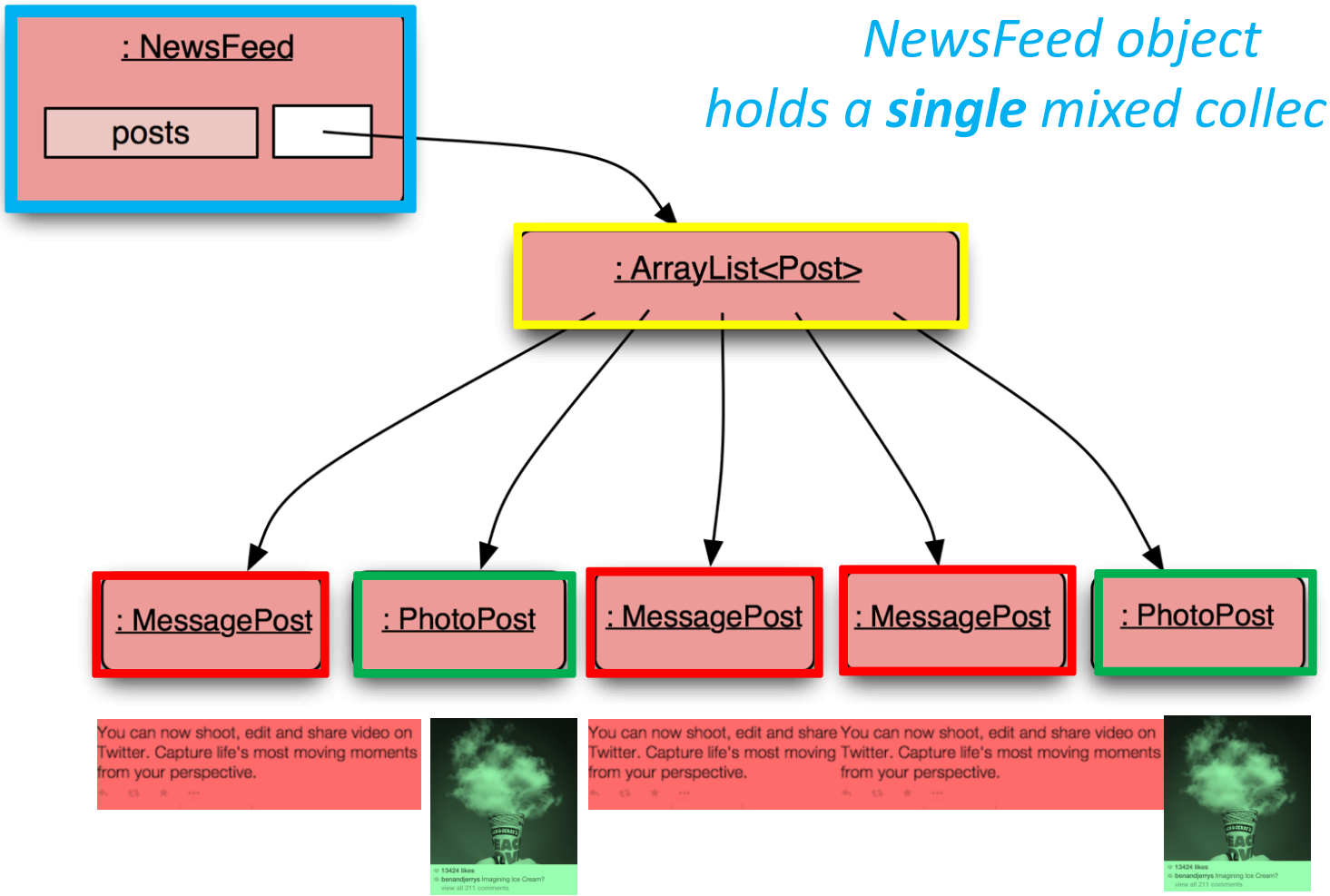
Subtyping and parameter passing

```
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**



NewsFeed object
*holds a **single** mixed collection*

Topic List

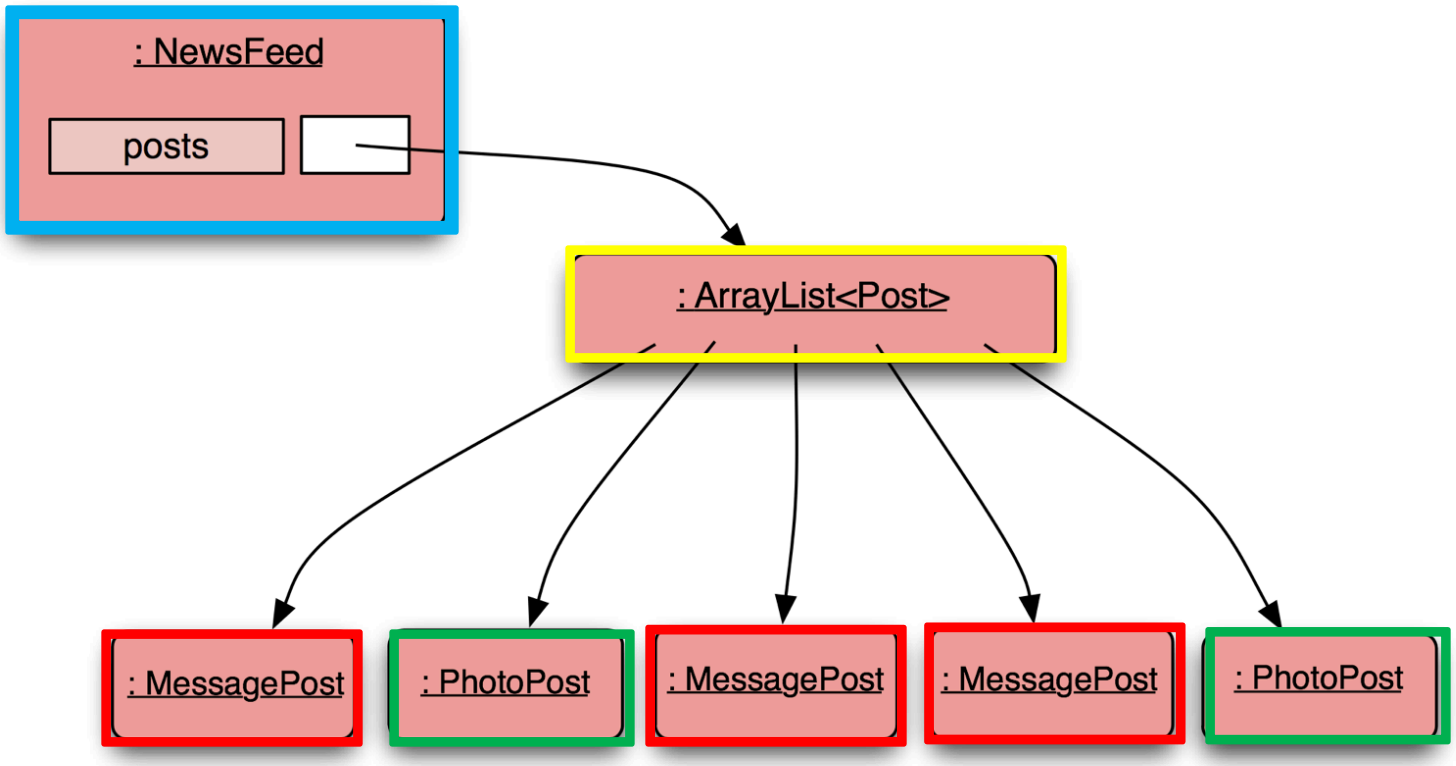
1. Social Network V1
2. Inheritance hierarchies
3. Social Network V2
4. Coding inheritance hierarchies
   - Super and subclasses
   - Using constructors in these hierarchies
5. Social Network V3
   - Deeper hierarchies
   - Advantages of using inheritance

6. Subtyping and Substitution
7. **Polymorphic**
   a) Variables
   b) Collections
   - ccasting, wrapper classes, autoboxing /unboxing

Polymorphic variables

- Object variables in Java are **polymorphic**
  - they can hold objects
    - of more than one type
    - of the declared type
    - or of subtypes of the declared type.

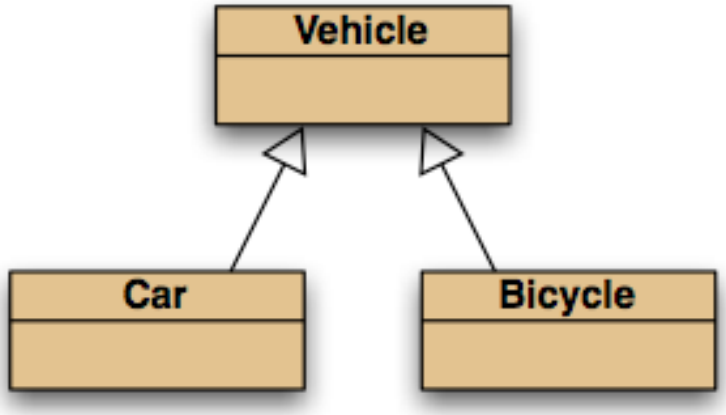
Social Network V2 – polymorphic ArrayList of Post



Casting

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

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



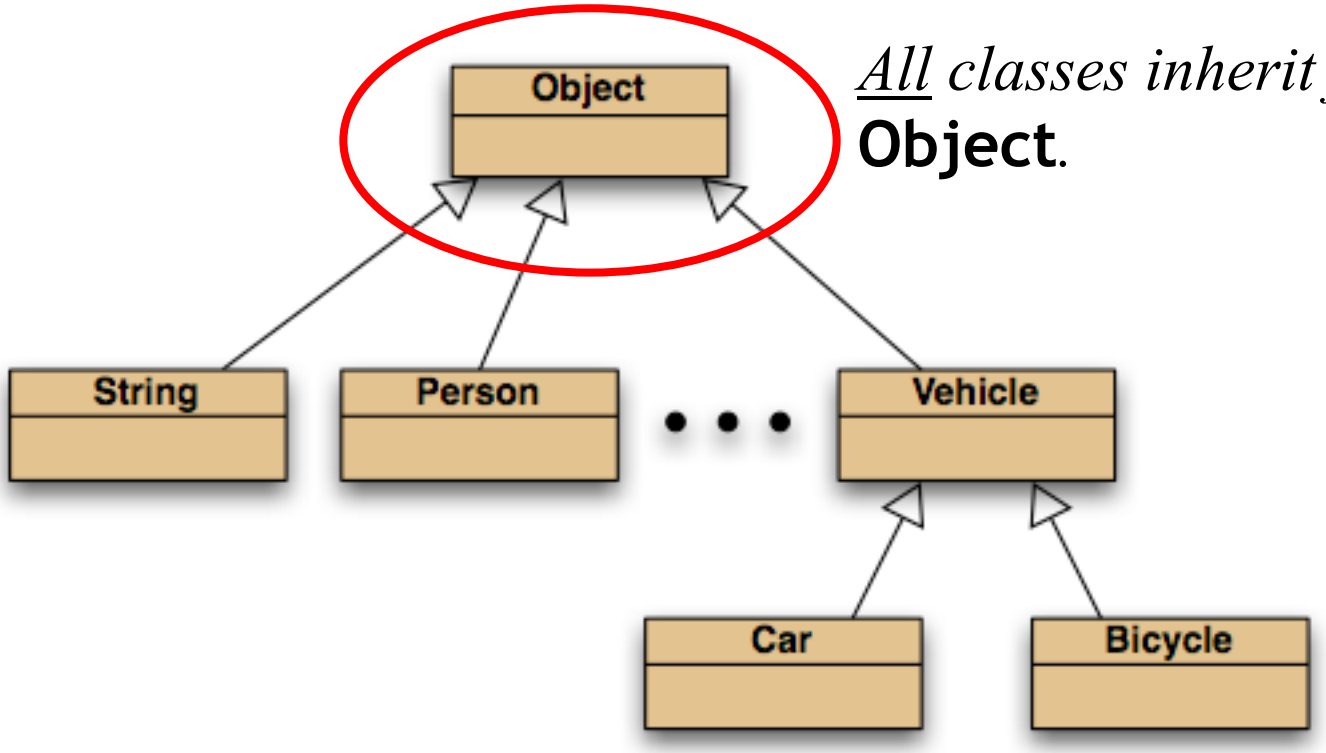

```
Vehicle v;
Car c = new Car();
v = c;          // correct (car is-a vehicle)
c = v;          // compile-time error!
c = (Car) v; //casting…correct (only if the vehicle really is a Car!)
```

Without (CASTING)

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**.

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
   - ccasting, wrapper classes, autoboxing /unboxing

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.

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:

primitive type	*wrapper class*
int	Integer
float	Float
char	Character
…	…

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

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

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)`**

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).

Any Questions?