Inheritance

Subtyping, substitution and polymorphic collections & variables.

Produced by: Dr. Siobhán Drohan
Ms. Mairéad Meagher
Ms Siobhan Roche

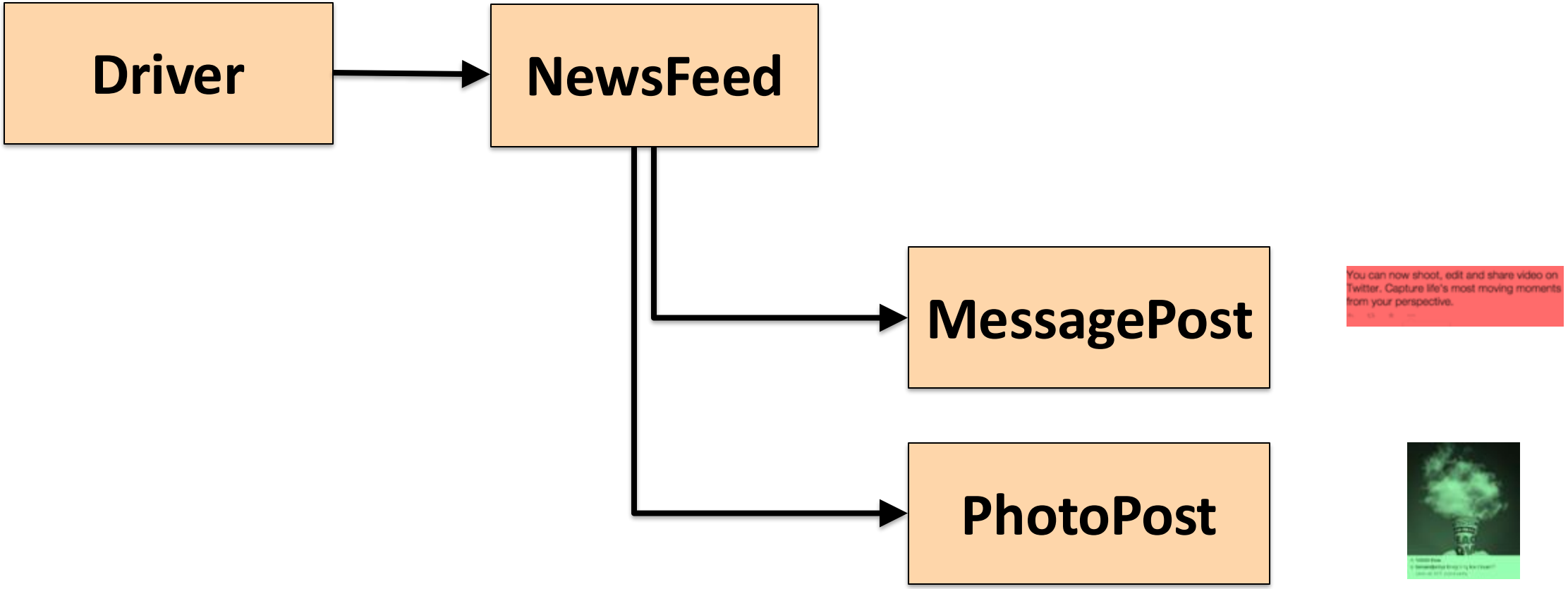
Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Department of Computing and Mathematics
http://www.setu.ie/

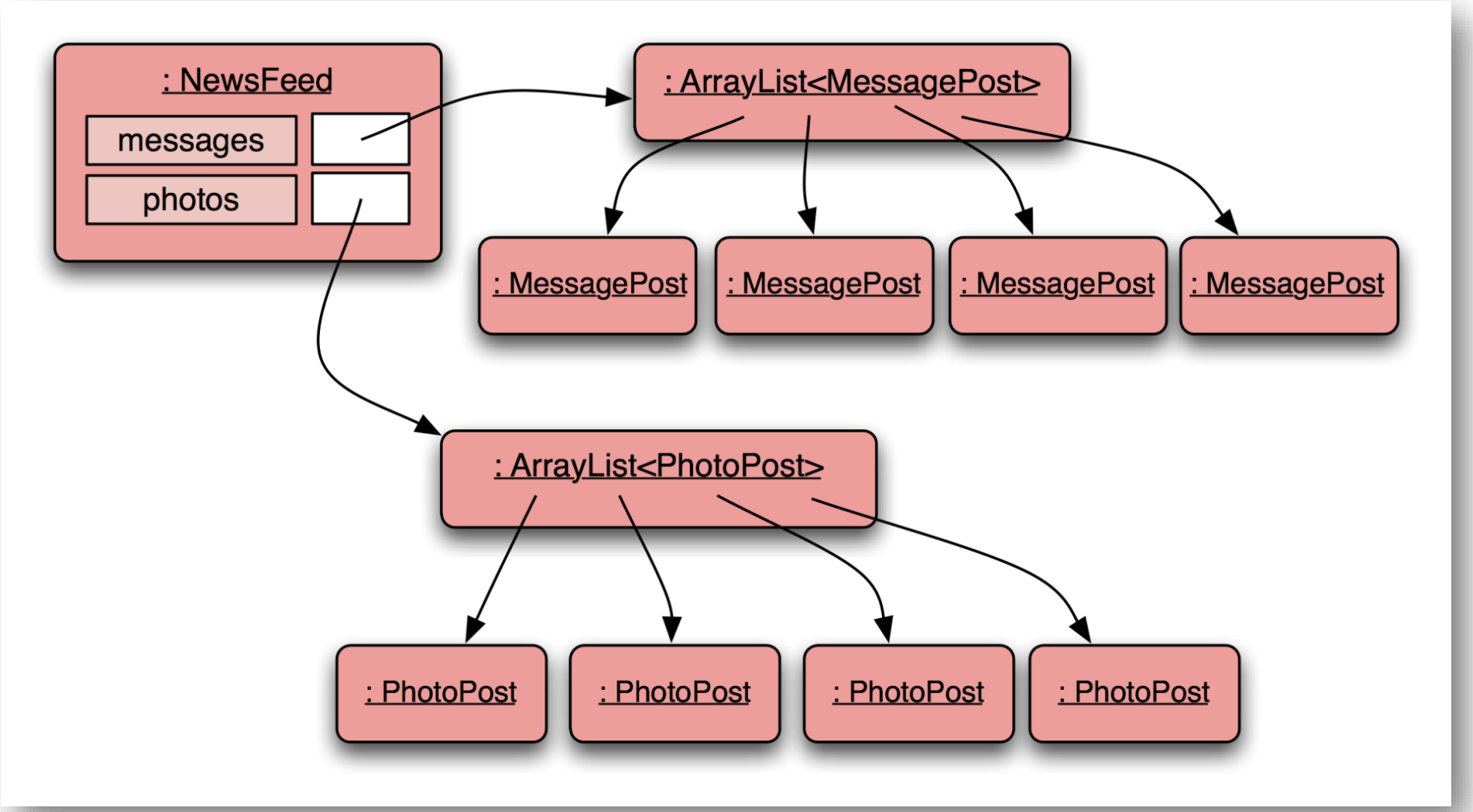
Topic List

1. Subtyping and Substitution
2. Polymorphic Variables
3. Polymorphic Collections
   - Includes
     - Casting
     - Wrapper classes
     - Autoboxing
     - Unboxing

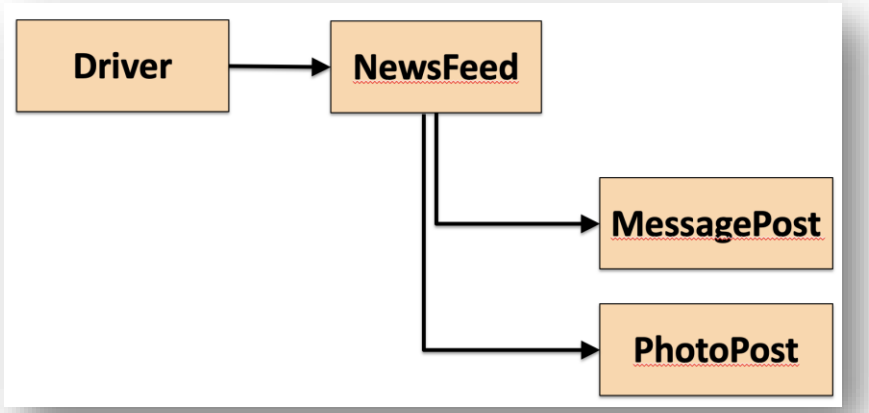
RECAP - Social Network V4.0 – NO Inheritance



RECAP - Social Network V4.0 – TWO ArrayLists



RECAP - Social Network V4.0

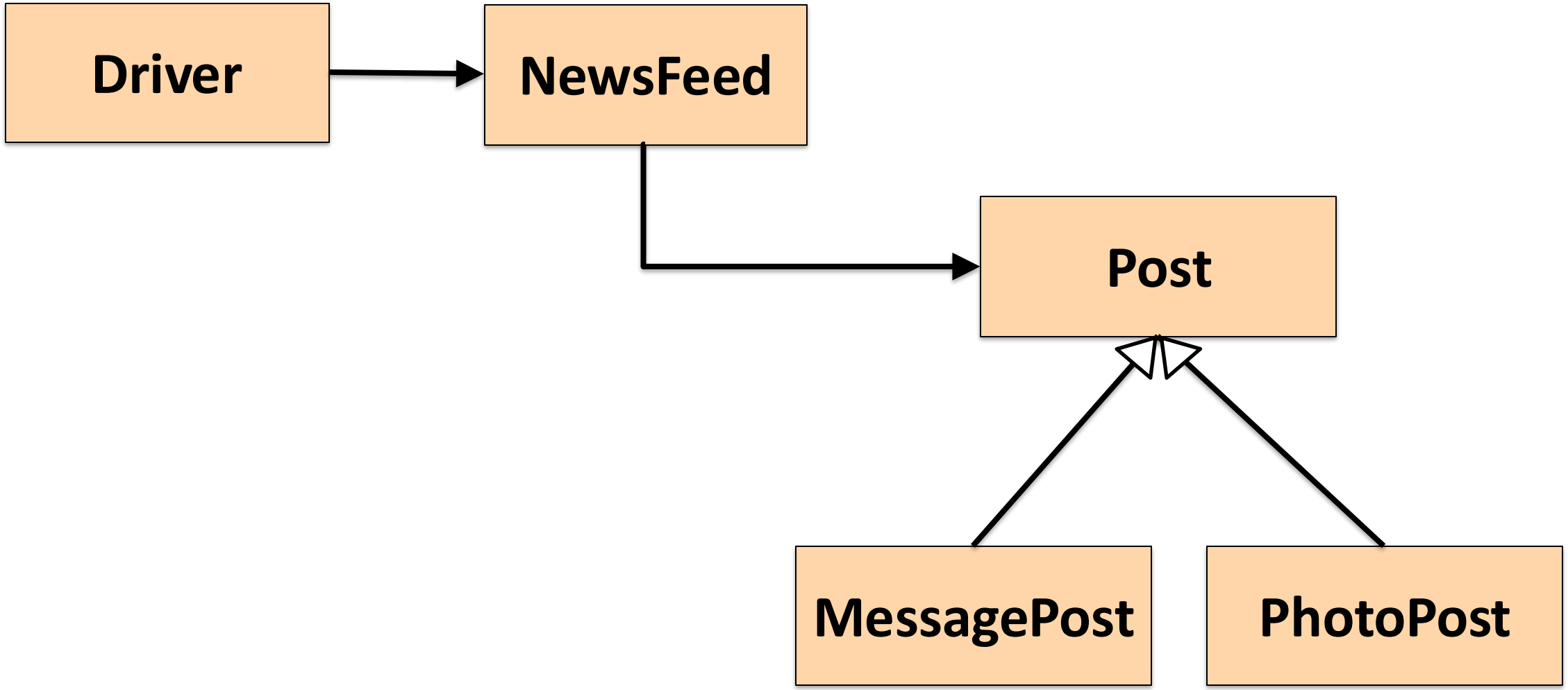


- NewsFeed
  - NewsFeed()
  - addMessagePost(MessagePost): boolean
  - addPhotoPost(PhotoPost): boolean
  - show(): String
  - showPhotoPosts(): String
  - showMessagePosts(): String
  - deleteMessagePost(int): MessagePost
  - deletePhotoPost(int): PhotoPost
  - updateMessagePost(int, String, String): boolean
  - updatePhotoPost(int, String, String, String): boolean
  - findMessagePost(int): MessagePost
  - findPhotoPost(int): PhotoPost
  - numberOfMessagePosts(): int
  - numberOfPhotoPosts(): int
  - load(): void
  - save(): void
  - isValidMessagePostIndex(int): boolean
  - isValidPhotoPostIndex(int): boolean
  - messagePosts: ArrayList<MessagePost>
  - photoPosts: ArrayList<PhotoPost>

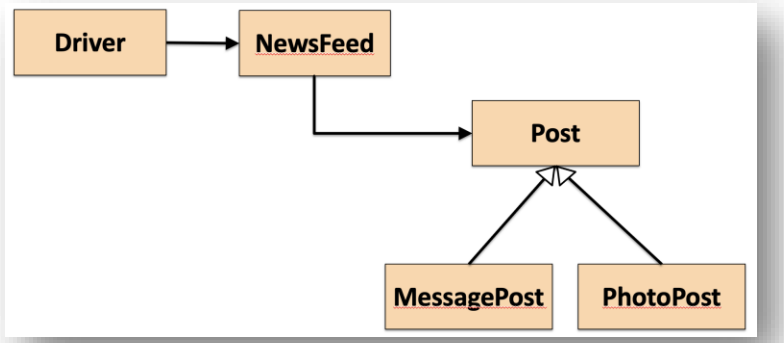
In V4.0, we had (no inheritance):

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

RECAP - Social Network V5.0 – Inheritance



RECAP - Social Network V5.0



- NewsFeed
  - NewsFeed()
  - addPost(Post): boolean
  - show(): String
  - showPhotoPosts(): String
  - showMessagePosts(): String
  - deletePost(int): Post
  - updateMessagePost(int, String, String): boolean
  - updatePhotoPost(int, String, String, String): boolean
  - findPost(int): Post
  - numberOfPosts(): int
  - numberOfMessagePosts(): int
  - numberOfPhotoPosts(): int
  - load(): void
  - save(): void
  - isValidIndex(int): boolean
  - isValidMessagePostIndex(int): boolean
  - isValidPhotoPostIndex(int): boolean
  - posts: ArrayList<Post>

V5.0 had inheritance:

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

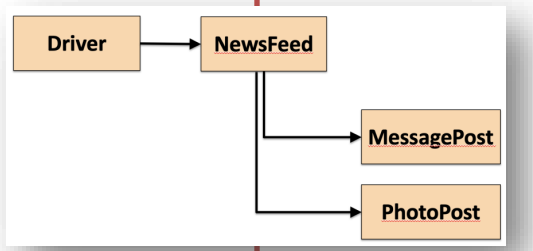
We call this method with:

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

Subtyping

V4.0 had no inheritance:

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

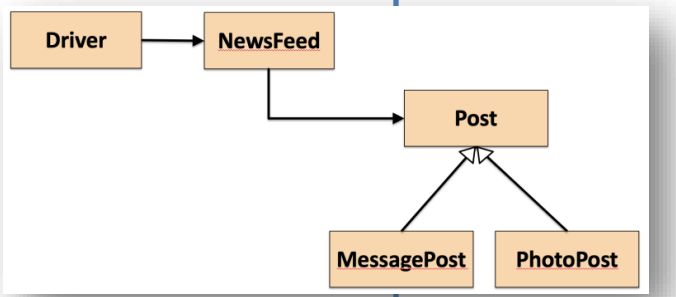


V5.0 had inheritance:

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

We call this method with:

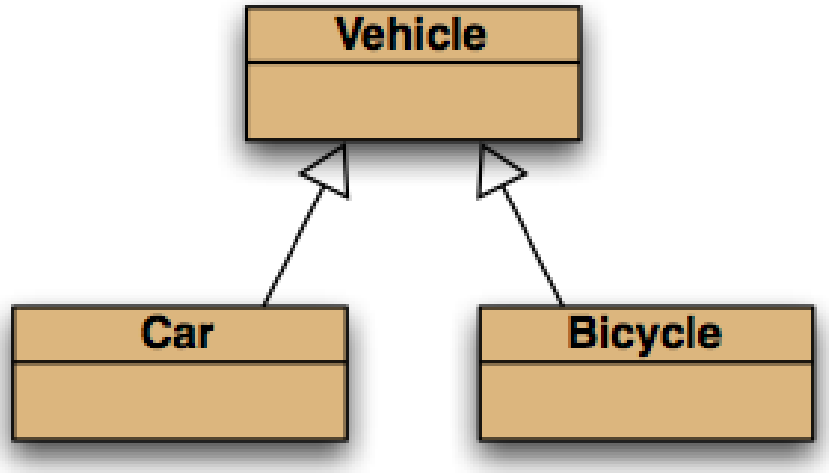
```
PhotoPost myPhoto = new PhotoPost(...);
newsFeed.addPost(myPhoto);
```



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*

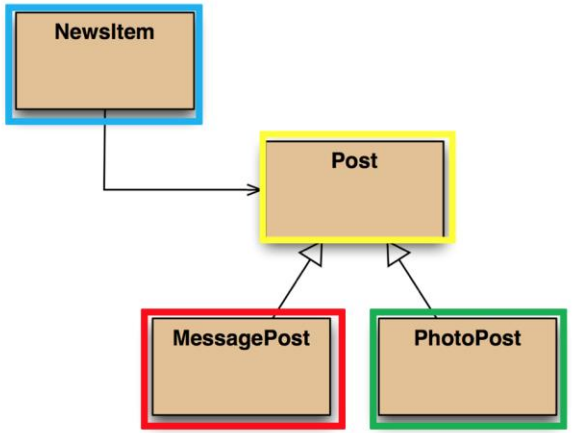
```
Vehicle v1 = new Vehicle();
Vehicle v2 = new Car();
Vehicle v3 = new Bicycle();
```

Subtyping and parameter passing

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

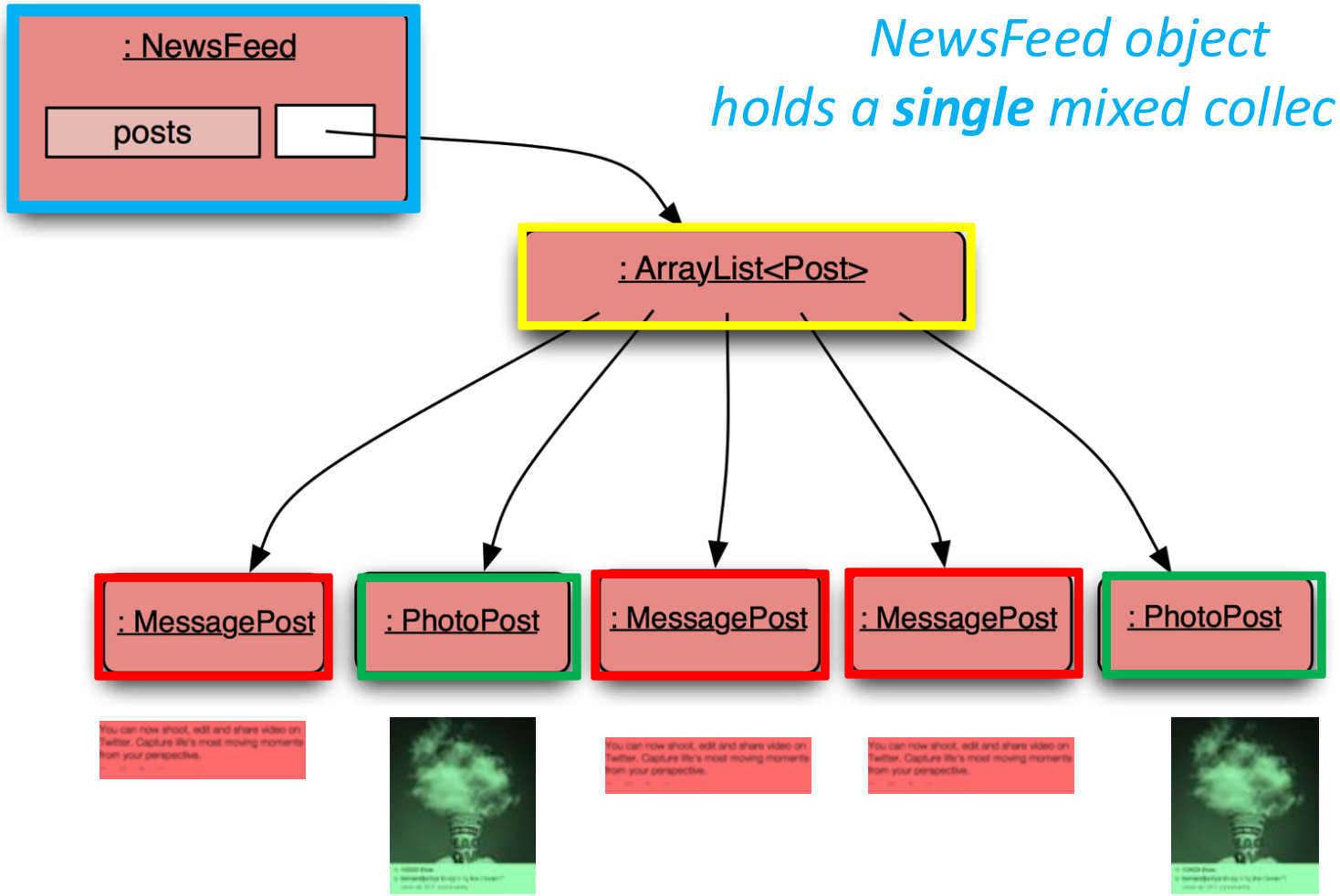
```
PhotoPost photo = new PhotoPost(...);
MessagePost message = new MessagePost(...);

newsFeed.addPost(photo);
newFeed.addPost(message);
```



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

Social Network V5.0 - **Object diagram**



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

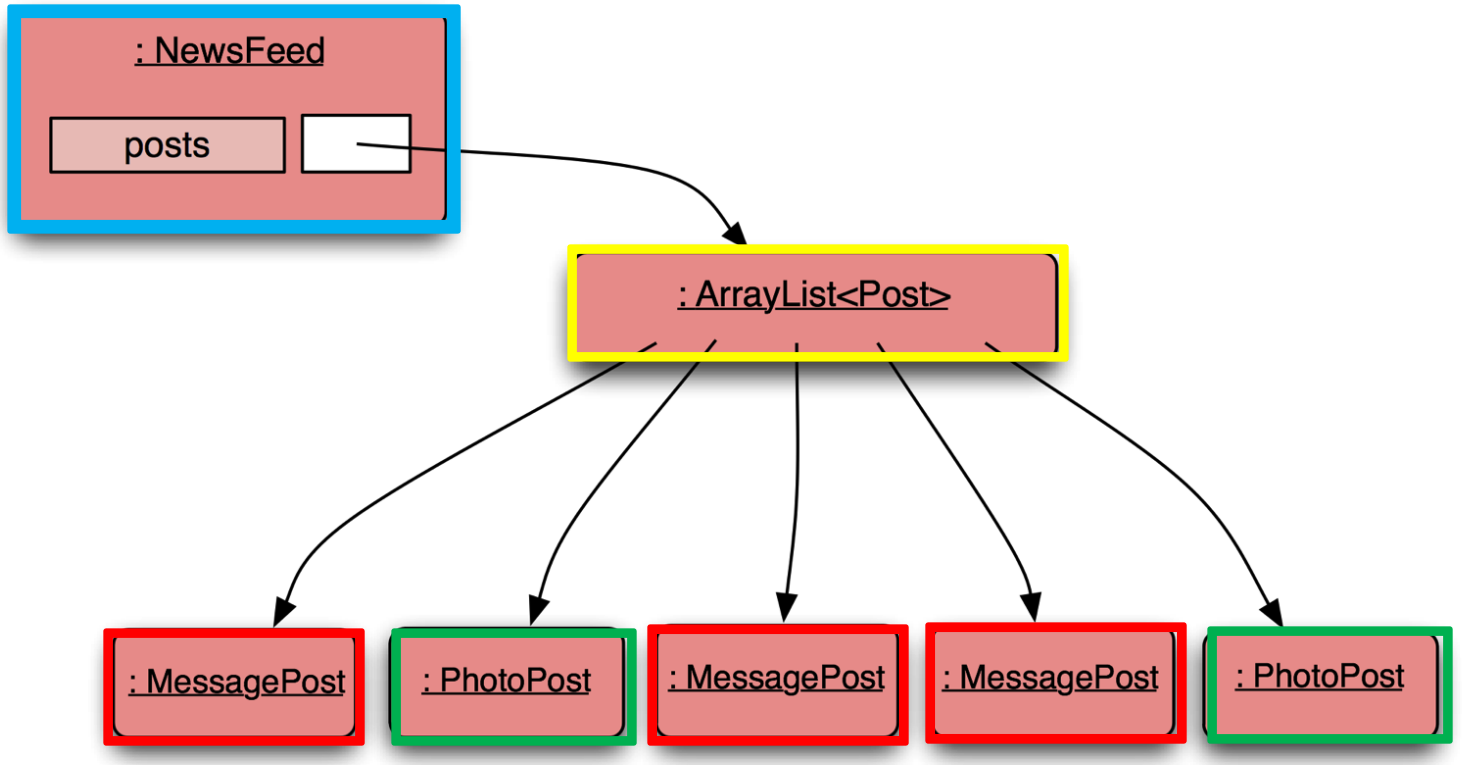
Topic List

1. Subtyping and Substitution
2. Polymorphic Variables
3. Polymorphic Collections
   - Includes
     - Casting
     - 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.

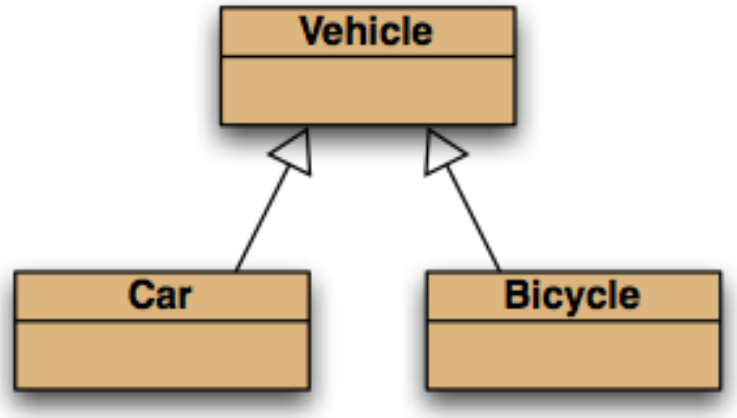
Social Network V5.0 – polymorphic ArrayList of Post



Casting

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

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



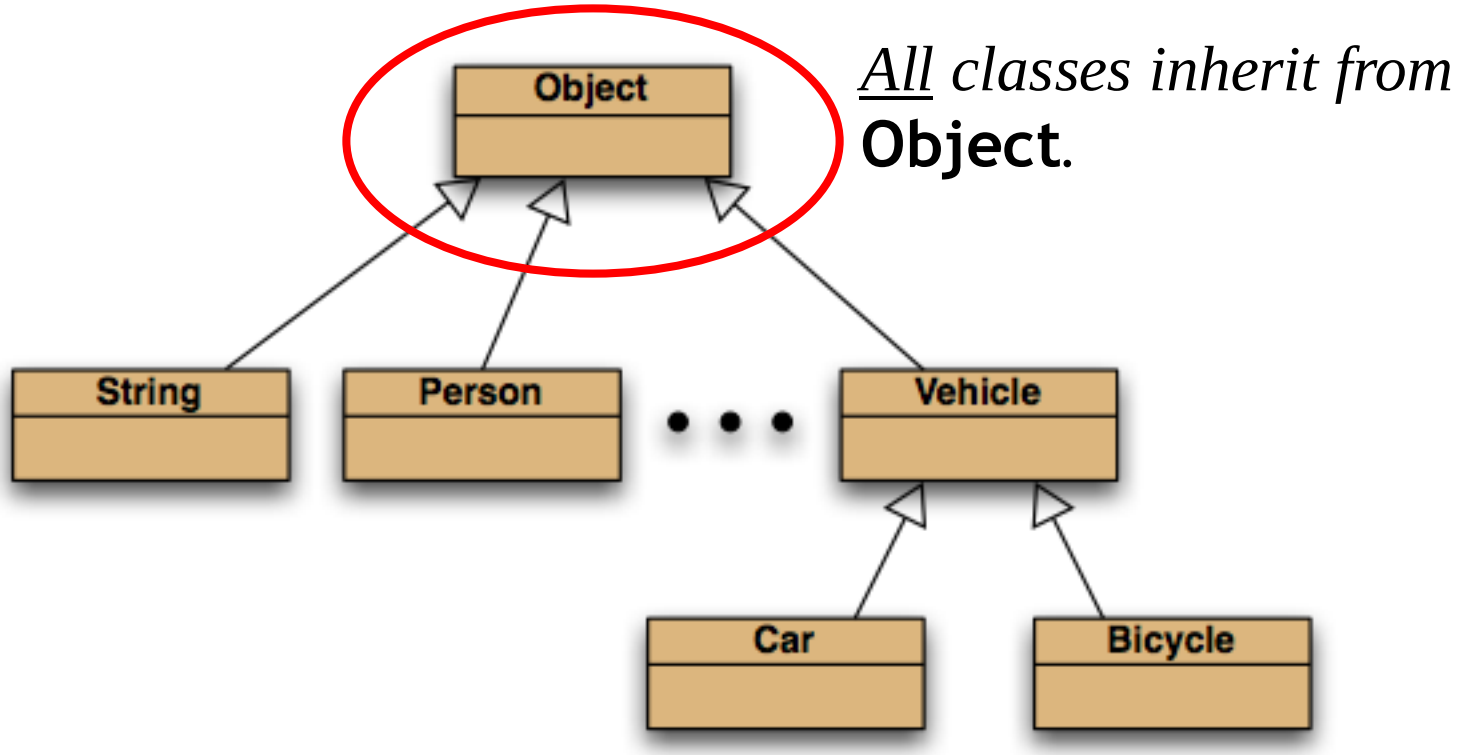

```
Vehicle v;
Car c = new Car();
v = c;          // correct (car is-a vehicle)
c = v;          // compile-time error!
c = (Car) v; //casting…correct (only if the vehicle really is a Car!)
```

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. Subtyping and Substitution
2. Polymorphic Variables
3. Polymorphic Collections
   - Includes
     - Casting
     - 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 value = 18;
Integer iwrap = new Integer(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)`**

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?