Inheritance - polymorphism

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/

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)
```

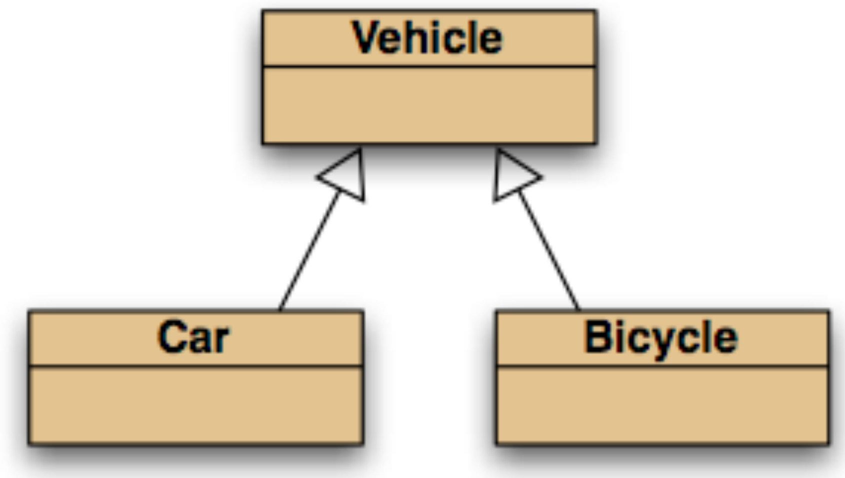
We call this method with:

```
  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

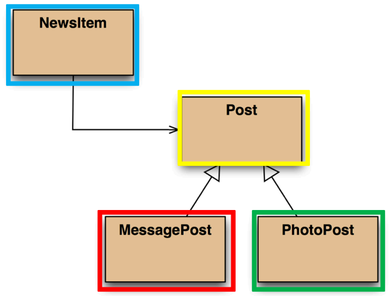


***sub**class objects*
may be assigned to
***super**class variables*

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

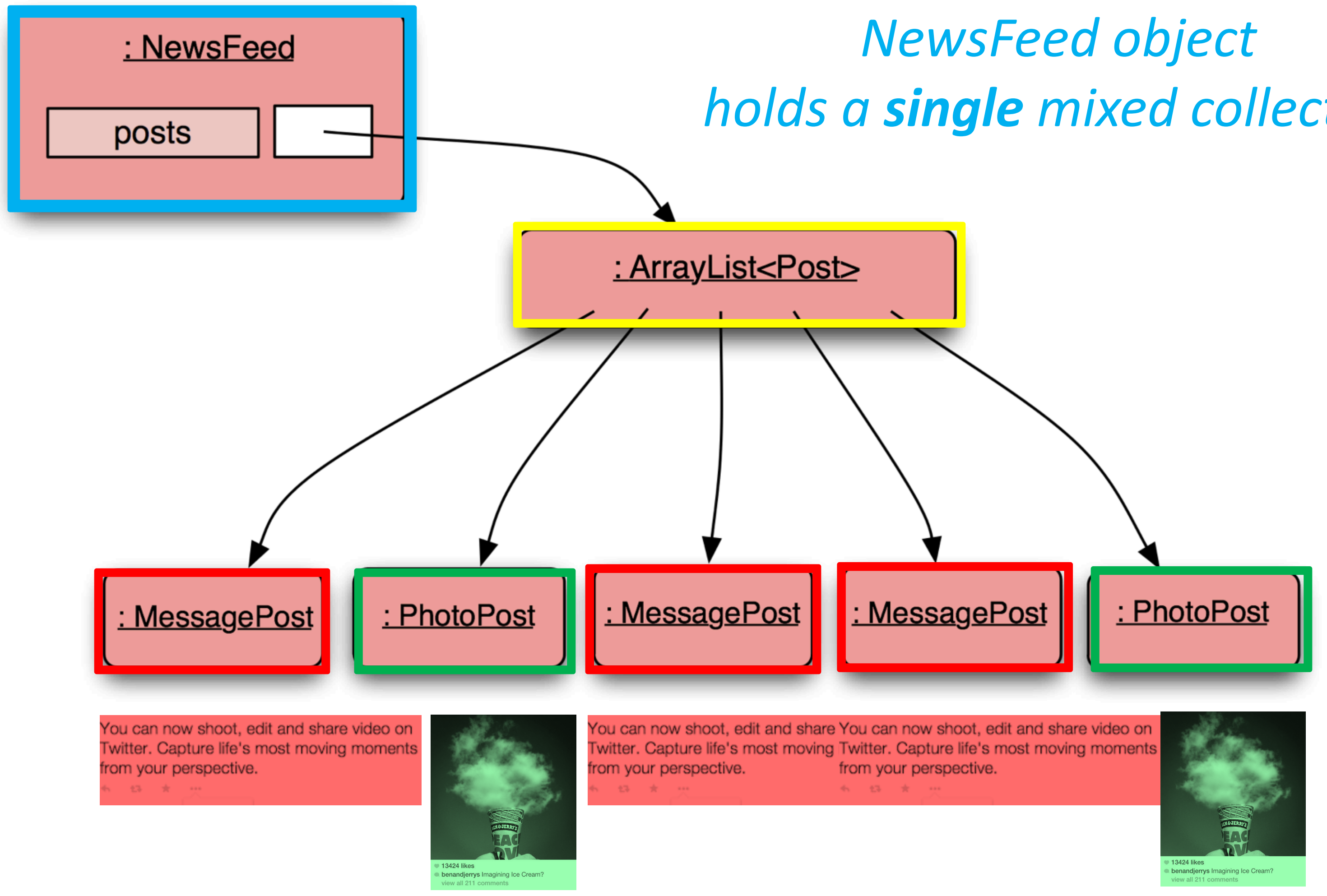
Subtyping and parameter passing

```
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**



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

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

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

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

Topic List

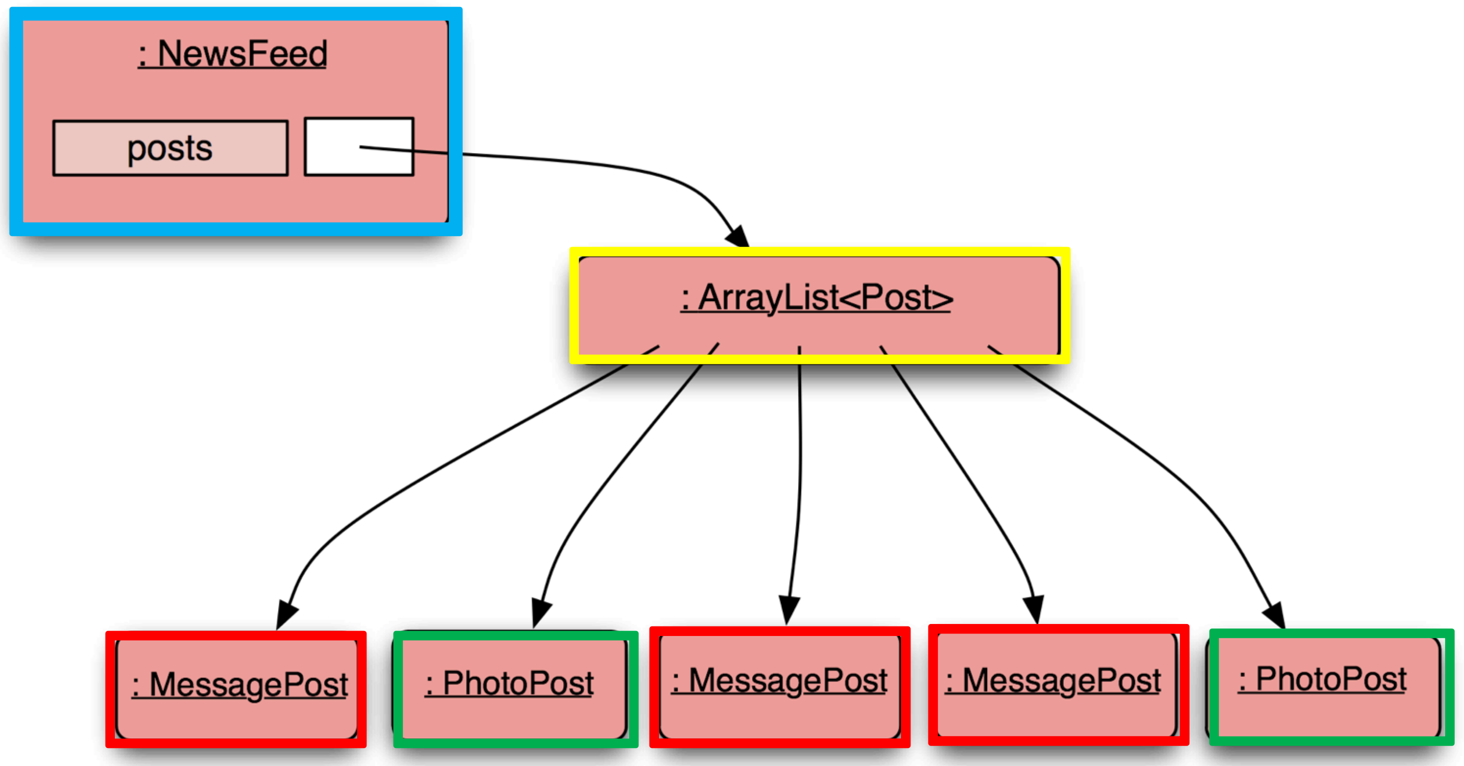
1. Social Network V1
2. Inheritance hierarchies
3. Social Network V2
4. Coding inheritance hierarchies
   – Super and subclasses
   – Using constructors in these hierarchies
5. Social Network V3
   – Deeper hierarchies
   – Advantages of using inheritance

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

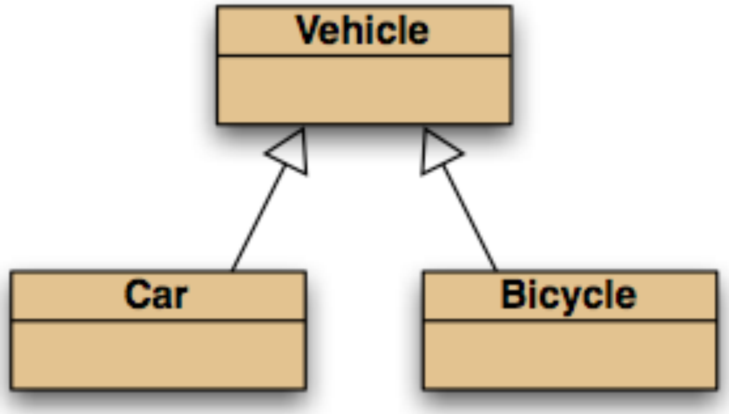
7 a) Polymorphic variables

- Object variables in Java are **polymorphic**
  - They can hold objects of
    1. more than one **type**
    2. the declared **type**
    3. sub**types** *(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!
```

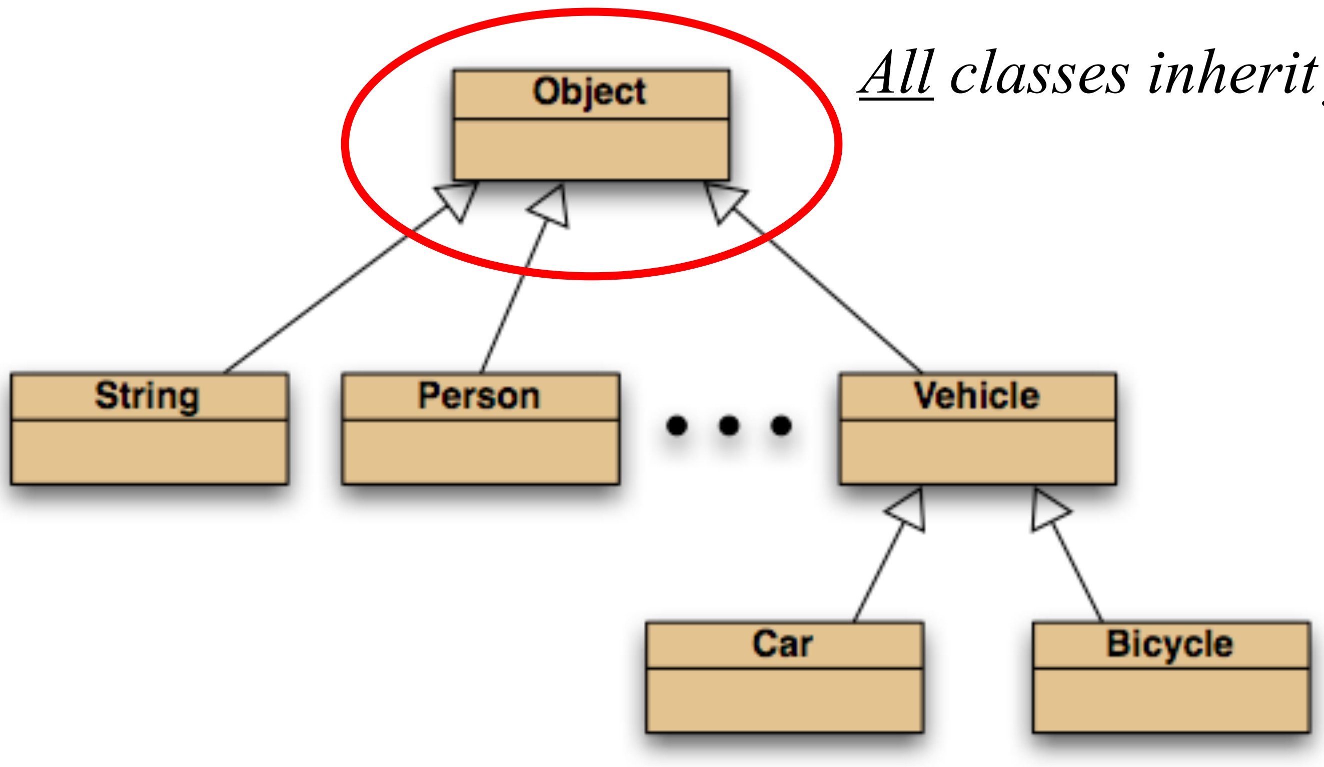
Without (CASTING)

```
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



All classes inherit from **Object**.

Topic List

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:

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

```
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).