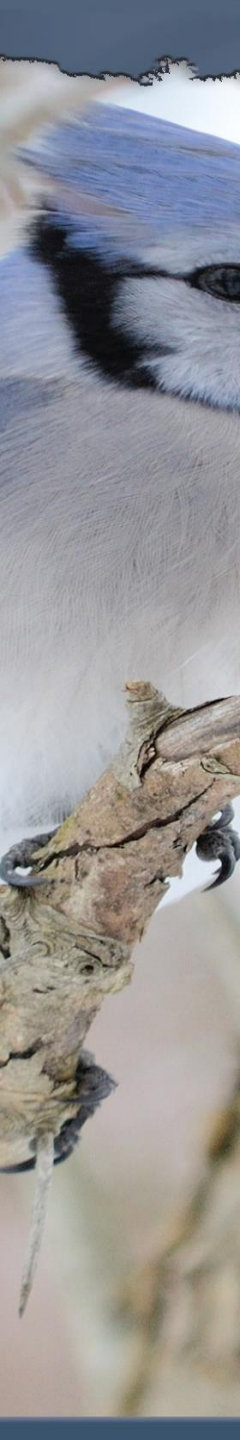
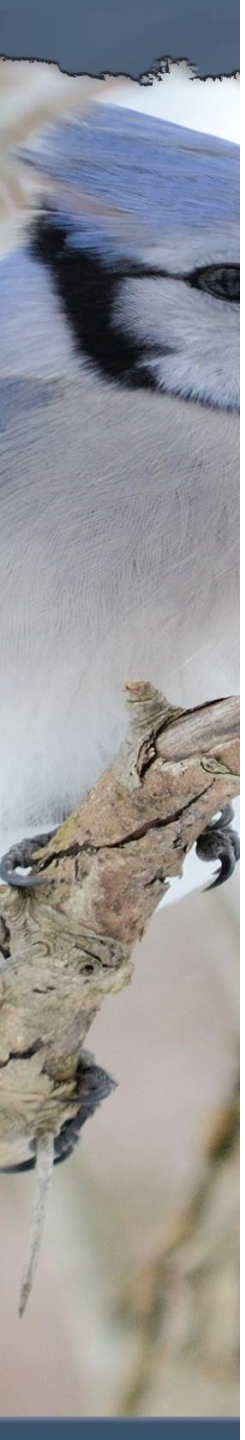
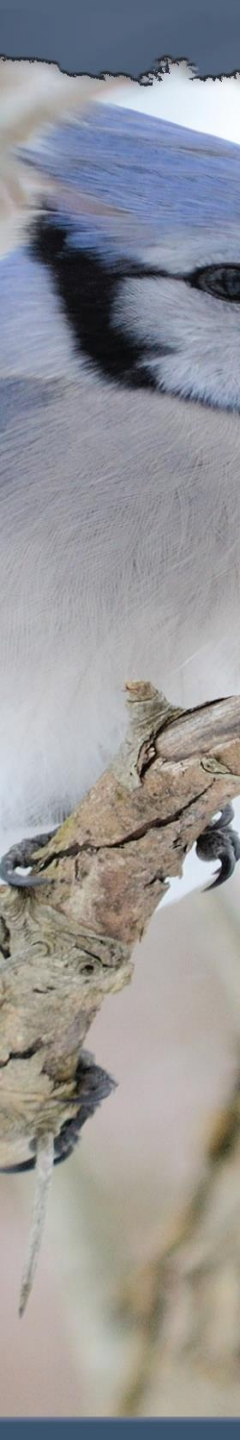


Improving structure with inheritance



Main concepts to be covered

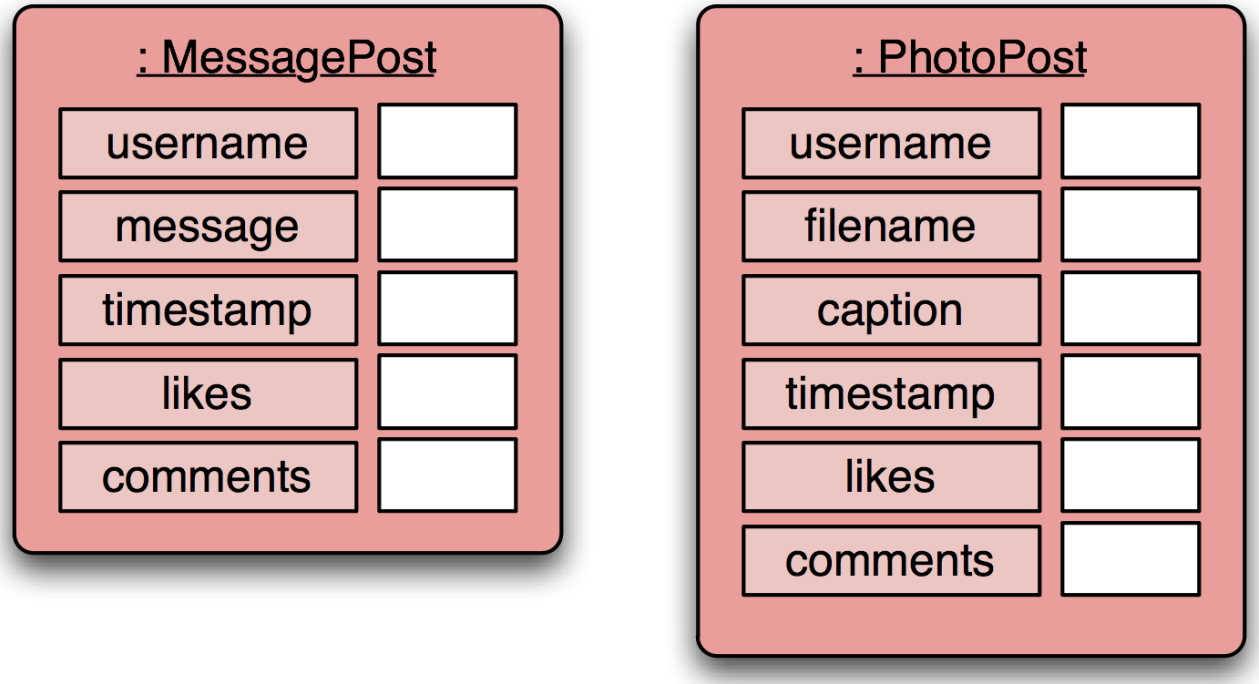
- Inheritance
- Subtyping
- Substitution
- Polymorphic variables



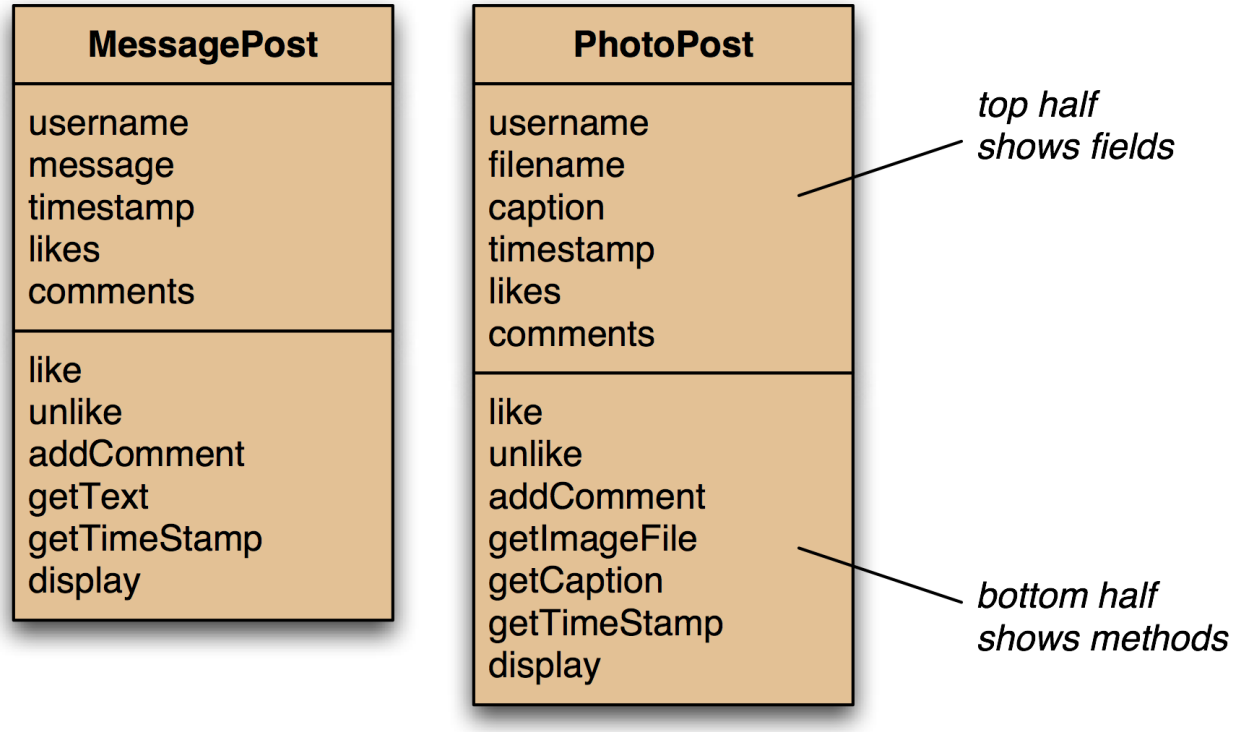
The Network example

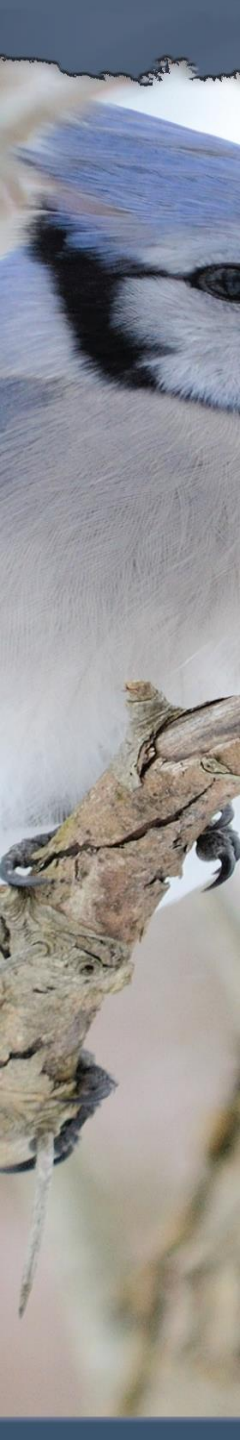
- A small, prototype social network.
- Supports a news feed with posts.
- Stores *text posts* and *photo posts*.
  - **`MessagePost`**: multi-line text message.
  - **`PhotoPost`**: photo and caption.
- Allows operations on the posts:
  - E.g., search, display and remove.

Network objects



Network classes

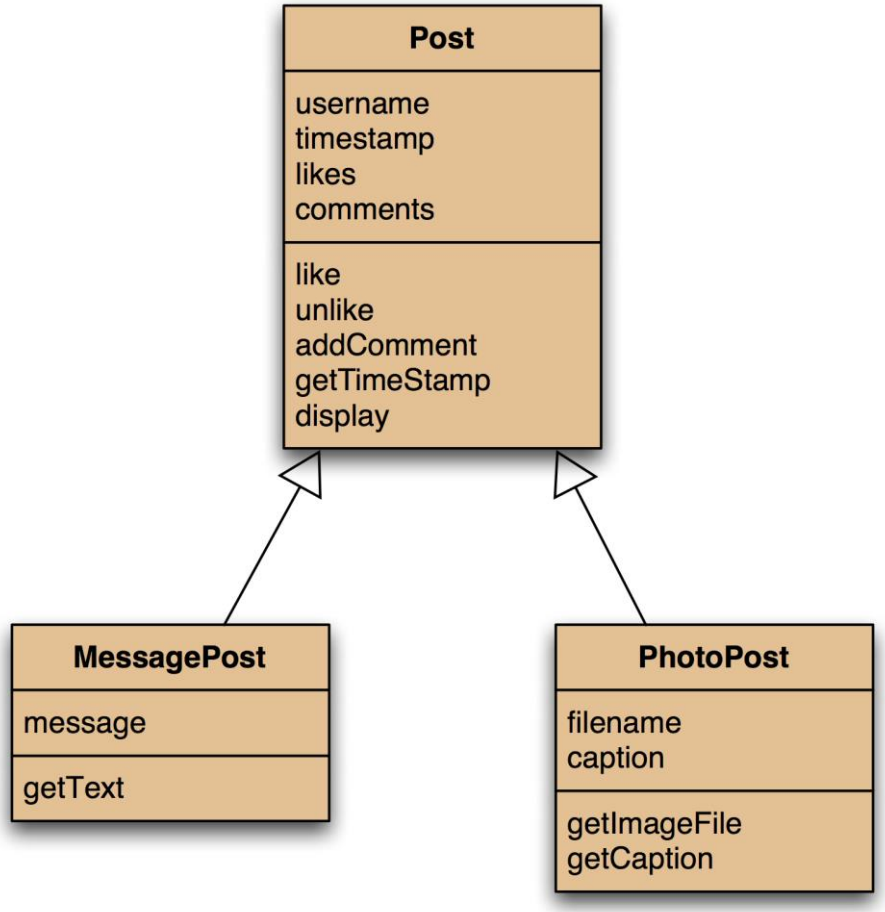


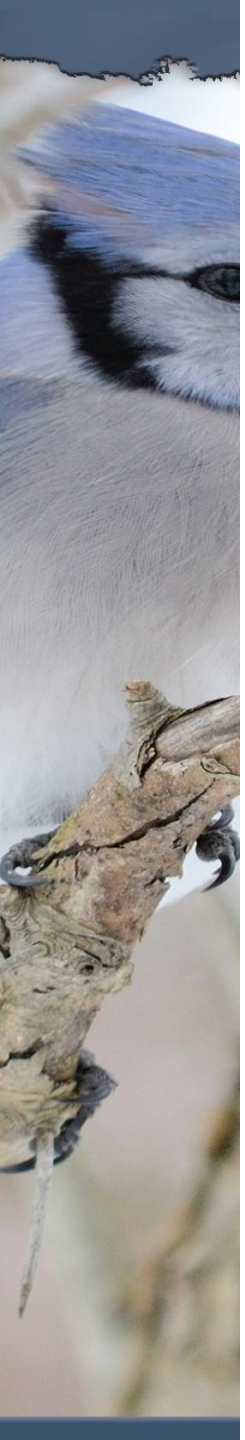


Critique of Network

- 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.

Using inheritance

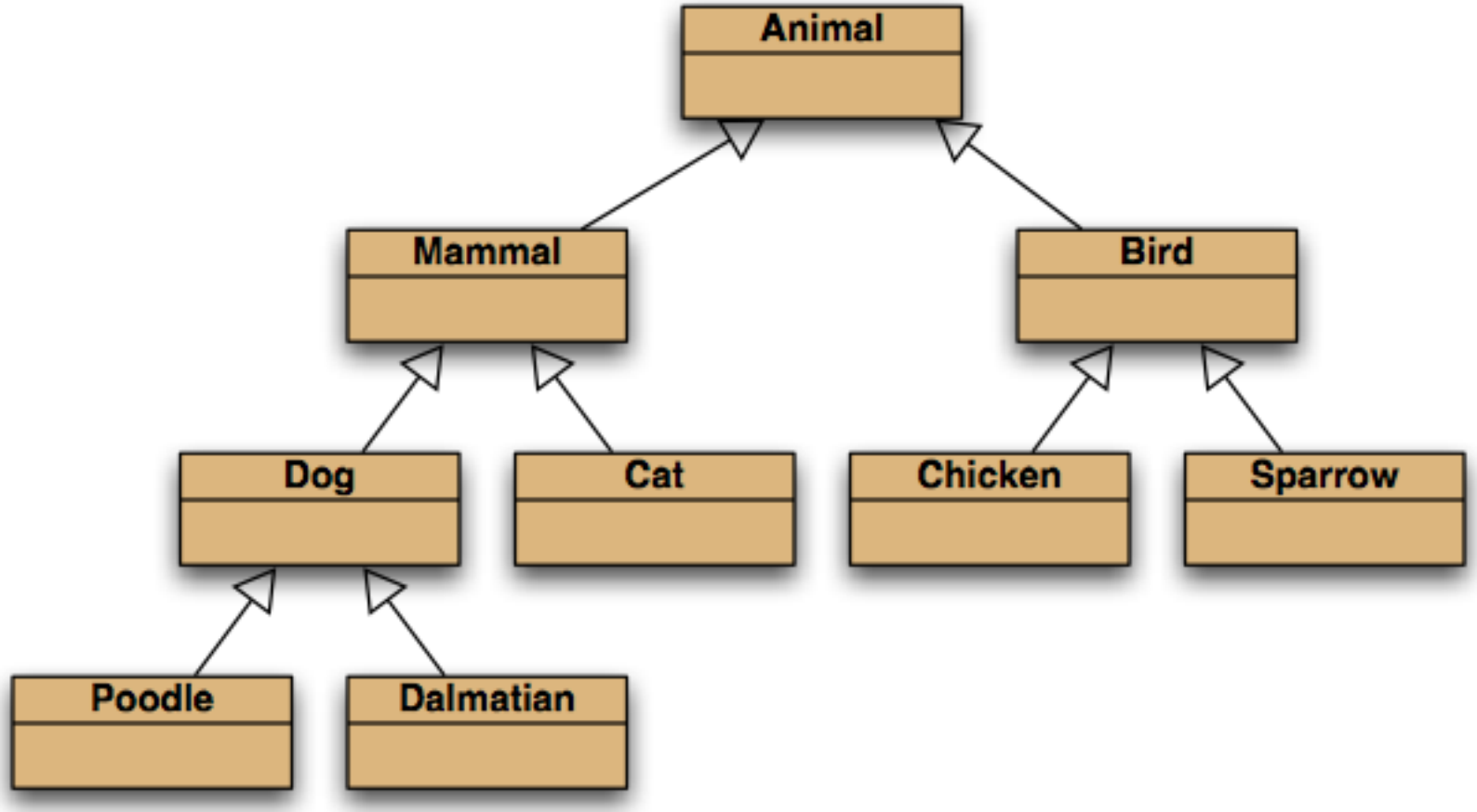


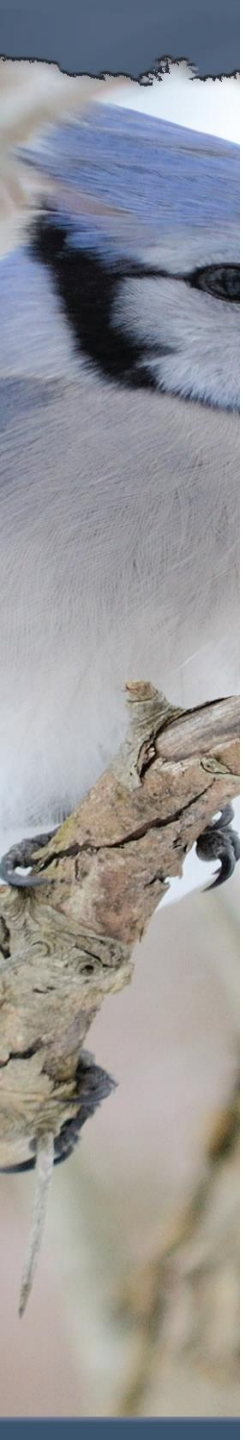


Using inheritance

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

Inheritance hierarchies

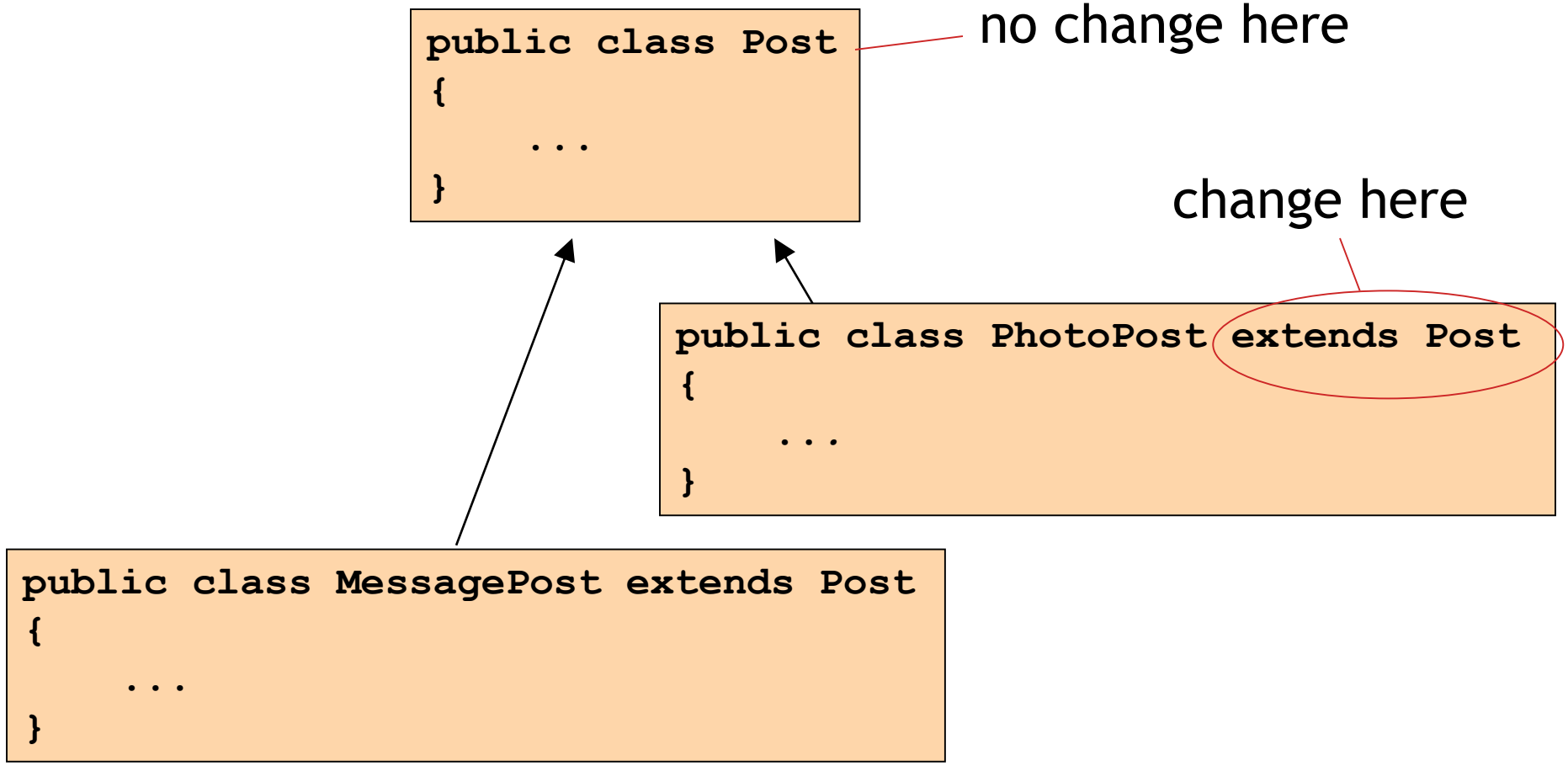


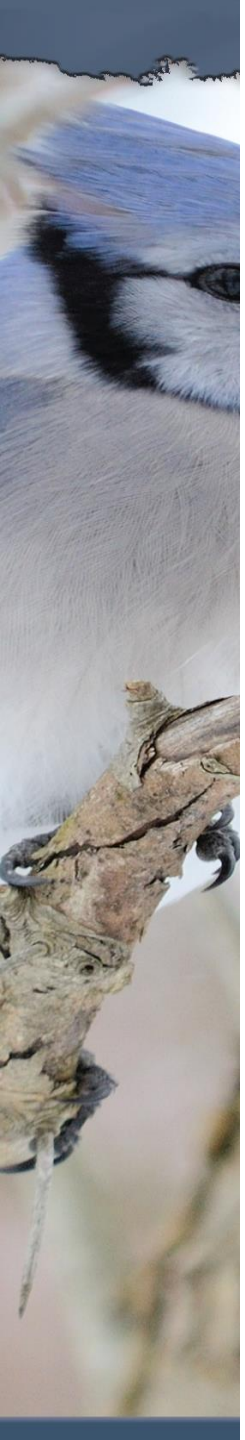


is-a? has-a? relationships

- Is-a: Arv. Subklassen **er** en spesialisering av superklassen. Eks: En hund ***er*** et pattedyr.
- Has-a: Komposisjon. En klasse kan ***ha*** tilgang til en annen klasse. En hund kan ***ha*** en eier.

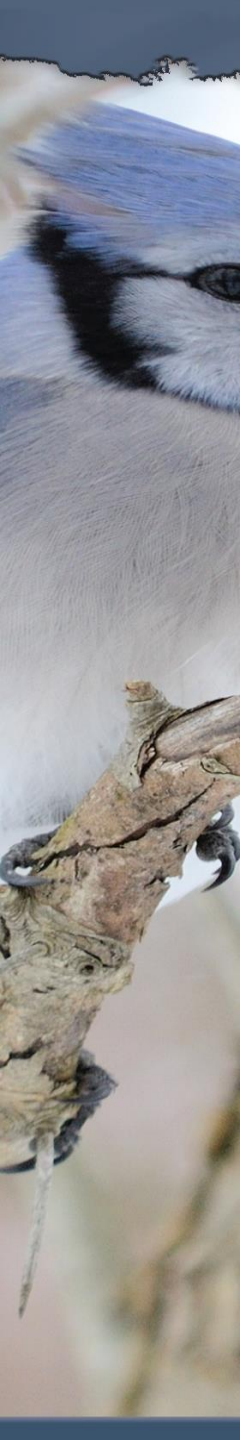
Inheritance in Java





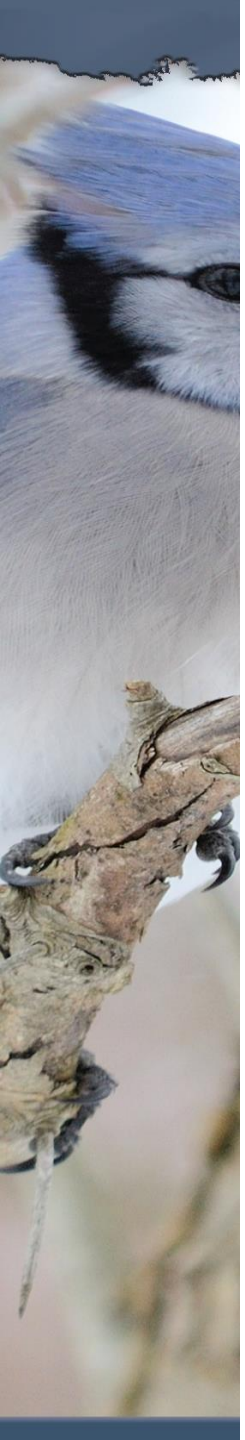
Superclass constructor call

- Subclass constructors must always contain a 'super' call.
- If none is written, the compiler inserts one (without parameters)
  - only compiles if the superclass has a constructor without parameters
- Must be the first statement in the subclass constructor.



Private access

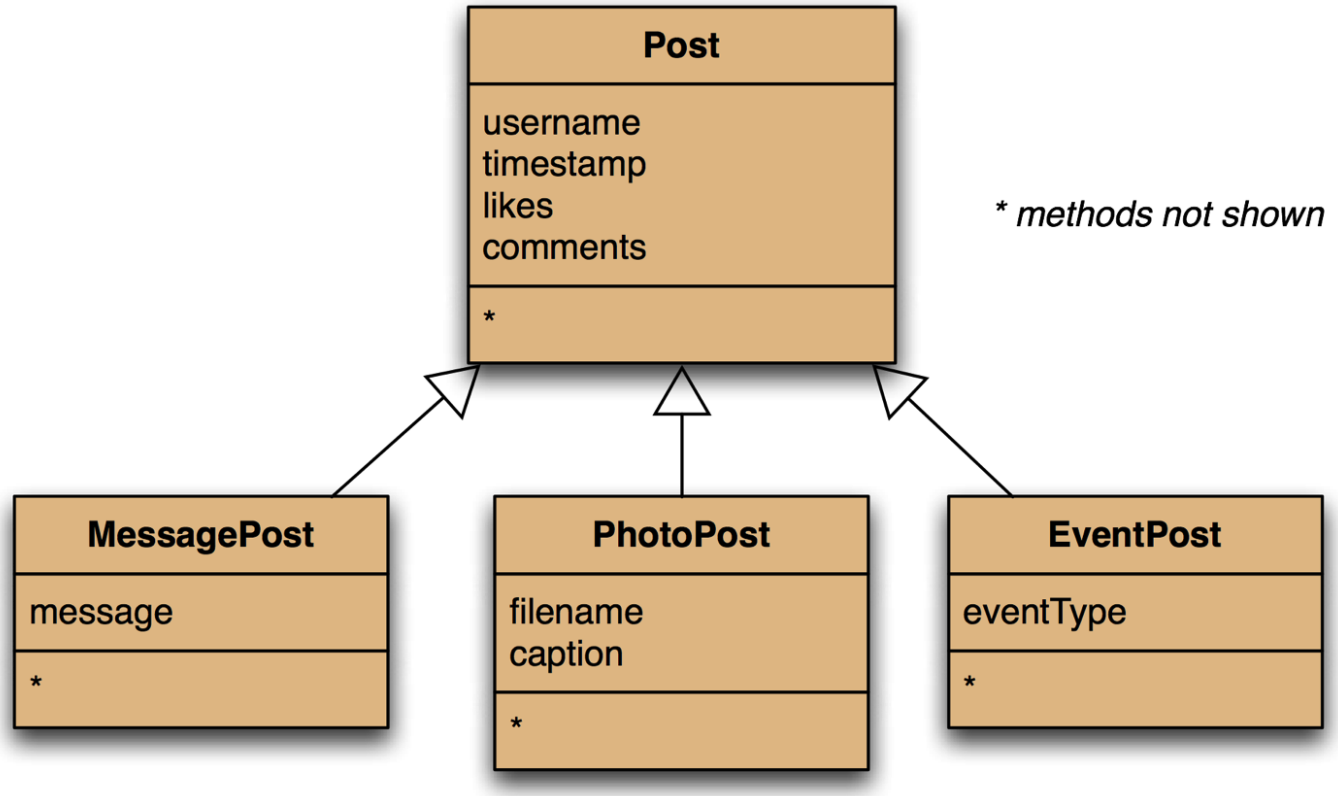
- Subclasses cannot access private fields in a superclass.
- Use getters in superclass if you want access…



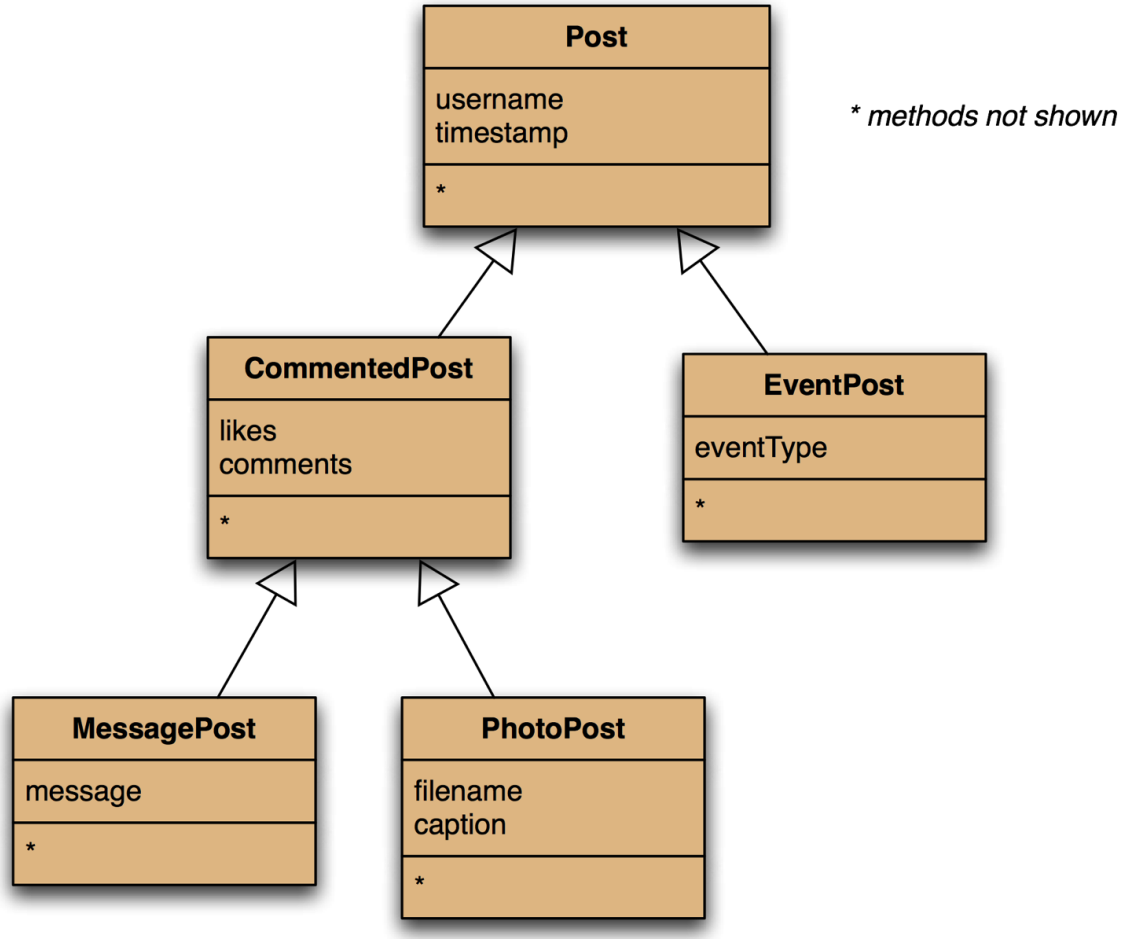
Oppgave!

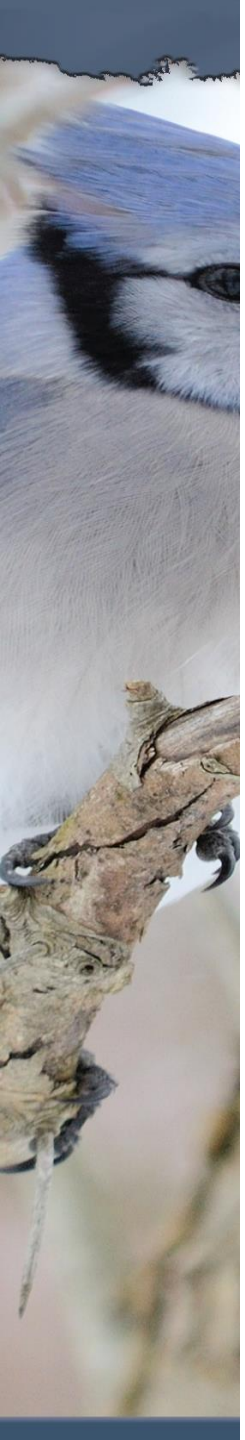
- Lag superklassen Post, og la klassene MessagePost og PhotoPost arve fra denne.
- Unngå unødvendig duplisering av kode.
- Koden skal funksjonelt sett fungere som tidligere.

Adding more item types



Deeper hierarchies

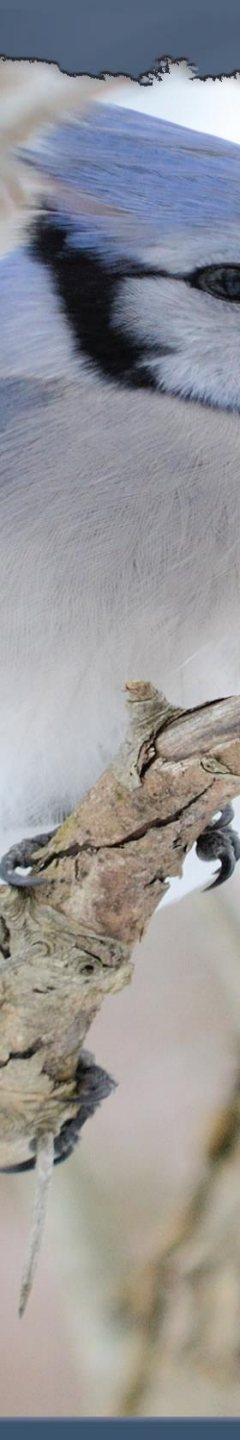




Review (so far)

Inheritance (so far) helps with:

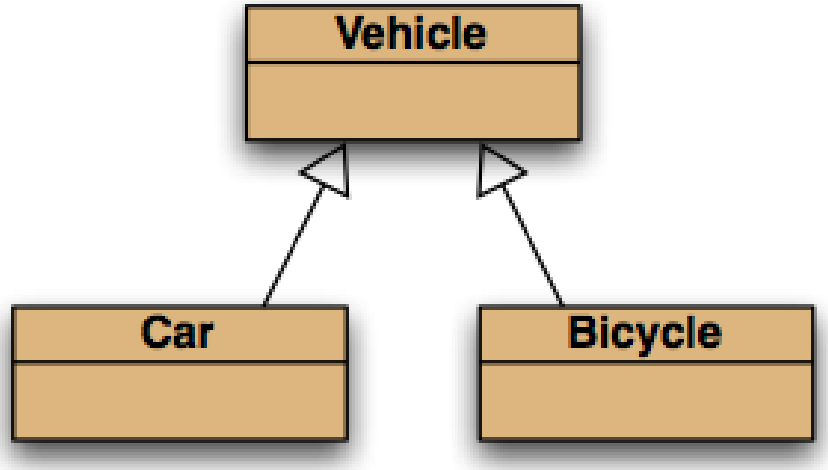
- Avoiding code duplication
- Code reuse
- Easier maintenance
- Extendibility



Subclasses and subtyping

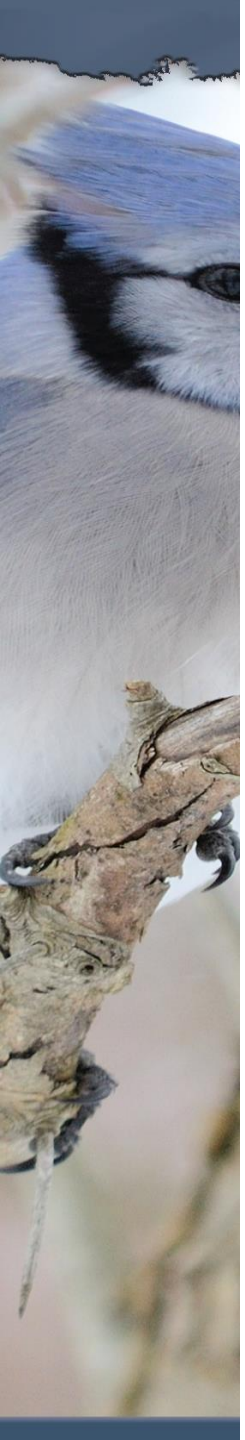
- Classes define types.
- Subclasses define *subtypes*.
- Objects of subclasses can be used where objects of supertypes are required.
  (This is called **substitution** .)

Subtyping and assignment



subclass objects may be assigned to superclass variables

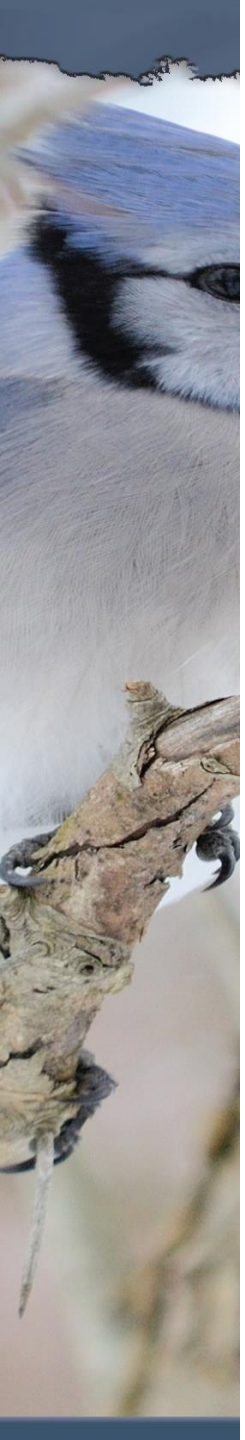
```
Vehicle v1 = new Vehicle();
Vehicle v2 = new Car();
Vehicle v3 = new Bicycle();
```



```
public class NewsFeed
{
    private ArrayList<Post> posts;

    /**
     * Construct an empty news feed.
     */
    public NewsFeed()
    {
        posts = new ArrayList<>();
    }

    /**
     * Add a post to the news feed.
     */
    public void addPost(Post post)
    {
        posts.add(post);
    }
    ...
}
```

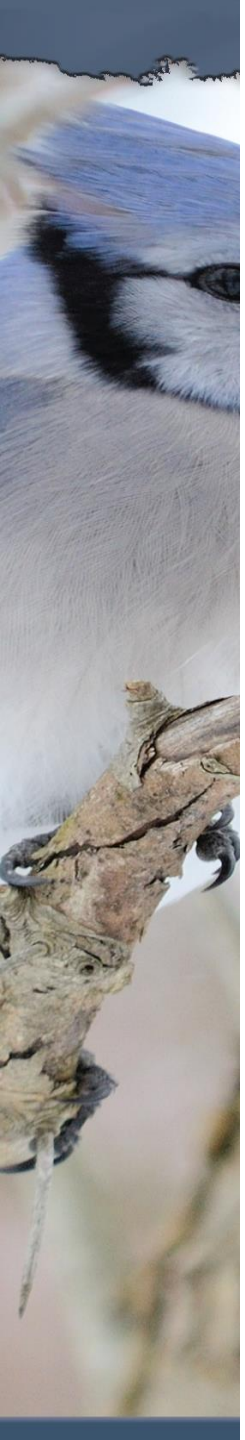
Revised **NewsFeed** source code

avoids code duplication in the client class!



New `NewsFeed` source code

```
/**
 * Show the news feed. Currently: print the
 * news feed details to the terminal.
 * (Later: display in a web browser.)
 */
public void show()
{
    for(Post post : posts) {
        post.display();
        System.out.println(); // Empty line ...
    }
}
```



Subtyping

First, we had:

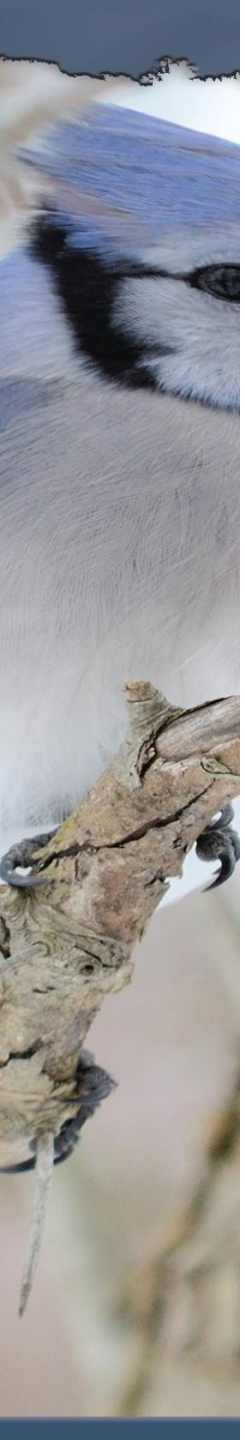
```
public void addMessagePost(
                    MessagePost message)
public void addPhotoPost(
                    PhotoPost photo)
```

Now, we have:

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

We call this method with:

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



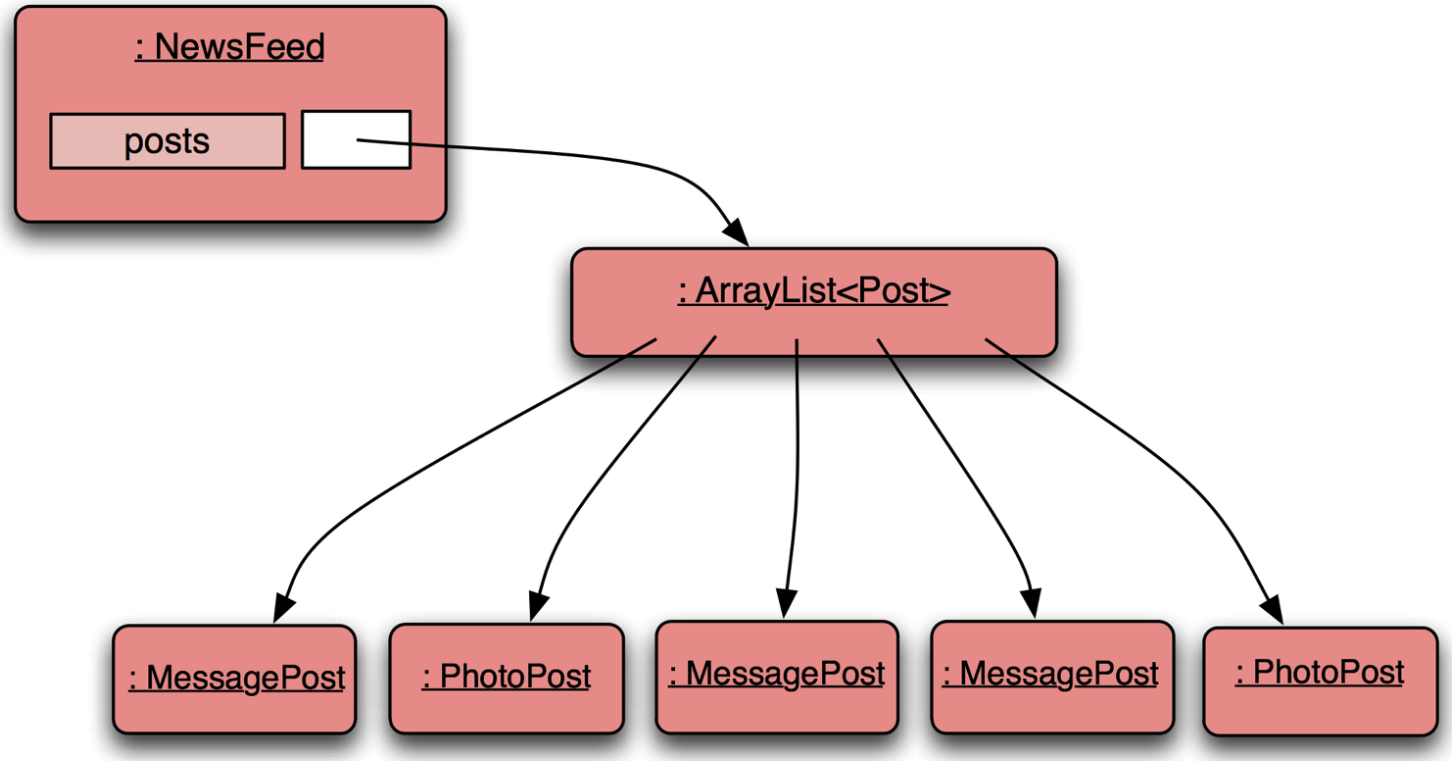
Subtyping and parameter passing

```
public class NewsFeed
{

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

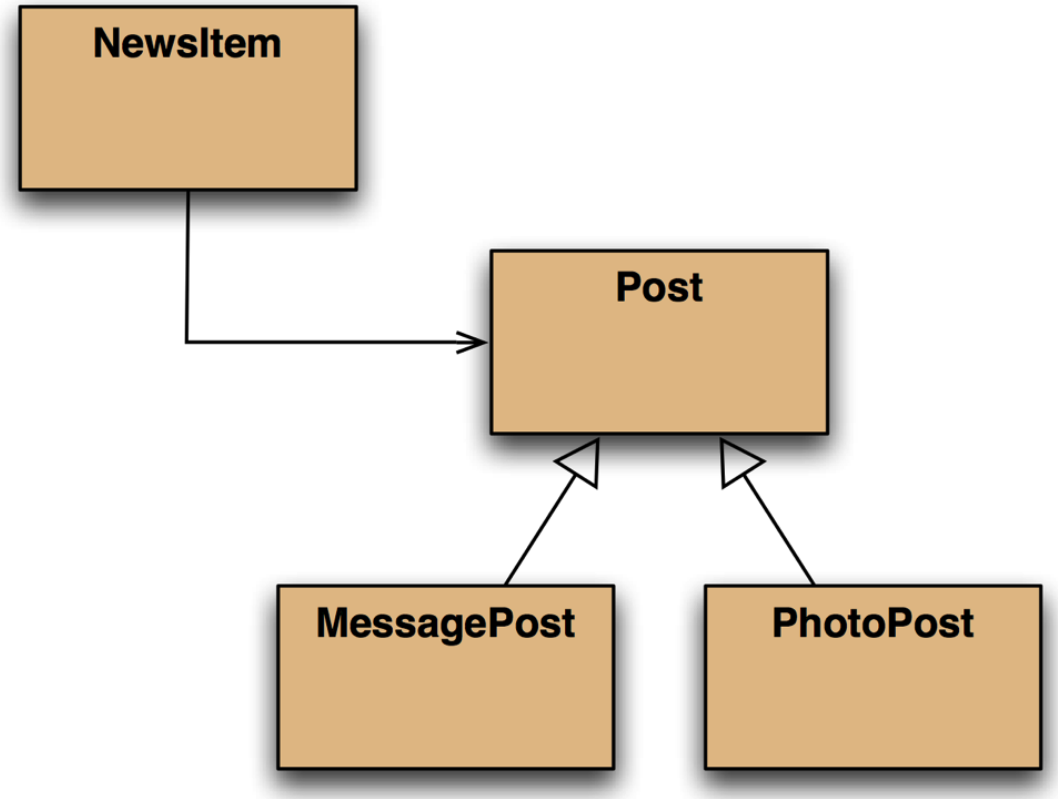
subclass objects may be used as actual parameters for the superclass

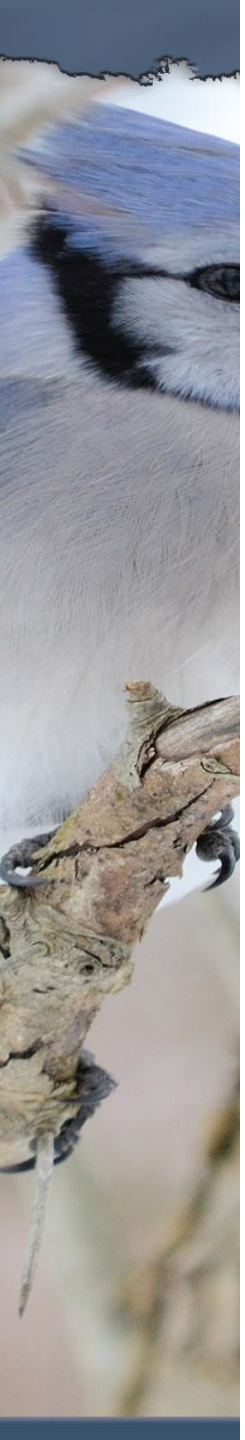
```
PhotoPost photo = new PhotoPost(...);
MessagePost message = new MessagePost(...);

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

Object diagram



Class diagram

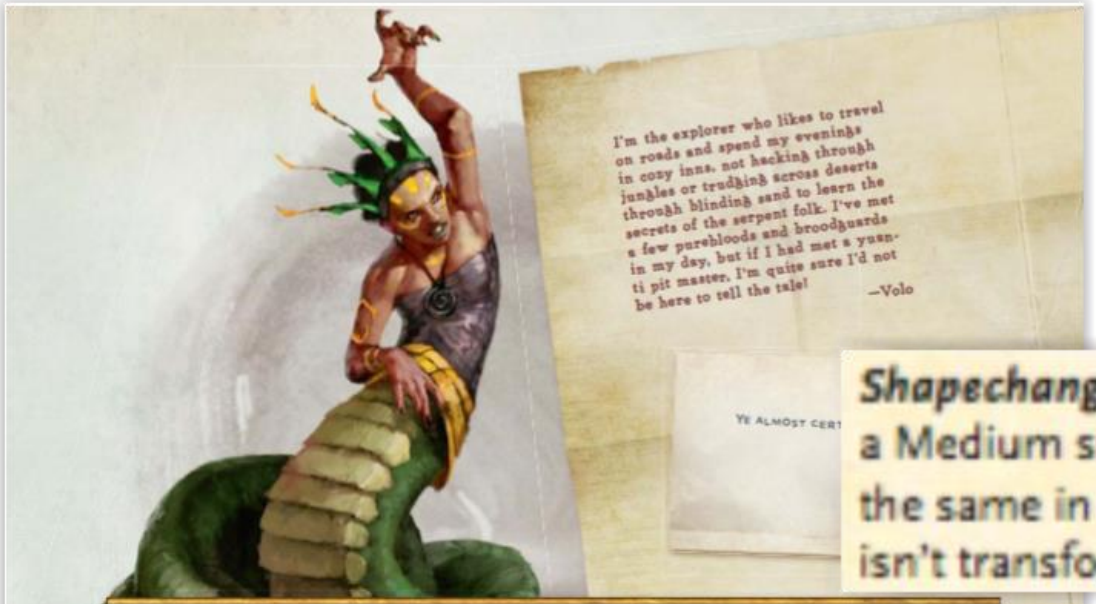




Polymorphic variables

- 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.



I'm the explorer who likes to travel on roads and spend my evenings in cozy inns, not hacking through jungles or trudging across deserts through blinding sand to learn the secrets of the serpent folk. I've met a few purebloods and broodguards in my day, but if I had met a yuan-ti pit master, I'm quite sure I'd not be here to tell the tale!

—Volo

YE ALMOST CER

Yuan-ti Nightmare Speaker

Medium monstrosity (shapechanger, yuan-ti), neutral evil

Armor Class 14 (natural armor)
Hit Points 71 (13d8 + 13)
Speed 30 ft.

STR	DEX	CON	INT	WIS	CHA
16 (+3)	14 (+2)	13 (+1)	14 (+2)	12 (+1)	16 (+3)

Saving Throws Wis +3, Cha +5
Skills Deception +5, Stealth +4
Damage Immunities poison
Condition Immunities poisoned
Senses darkvision 120 ft. (penetrates magical darkness), passive Perception 11
Languages Abyssal, Common, Draconic
Challenge 4 (1,100 XP)

Shapechanger. The yuan-ti can use its action to polymorph into a Medium snake or back into its true form. Its statistics are the same in each form. Any equipment it is wearing or carrying isn't transformed. If it dies, it stays in its current form.

Death Fangs (2/Day). The first time the yuan-ti hits with a melee attack on its turn, it can deal an extra 16 (3d10) necrotic damage to the target.

Innate Spellcasting (Yuan-ti Form Only). The yuan-ti's innate spellcasting ability is Charisma (spell save DC 13). The yuan-ti can innately cast the following spells, requiring no material components:

At will: *animal friendship* (snakes only)
3/day: *suggestion*

Magic Resistance. The yuan-ti has advantage on saving throws against spells and other magical effects.

Spellcasting (Yuan-ti Form Only). The yuan-ti is a 6th-level spellcaster. Its spellcasting ability is Charisma (spell save DC 13, +5 to hit with spell attacks). It regains its expended spell slots when it finishes a short or long rest. It knows the following warlock spells:

Cantrip (at will): *chill touch*, *eldritch blast* (range 300 ft., +3 bonus to each damage roll), *mage hand*, *message*, *poison spray*, *prestidigitation*
1st–3rd level (2 3rd-level slots): *arms of Hadar*, *darkness*, *fear*, *hex*, *hold person*, *hunger of Hadar*, *witch bolt*

Actions

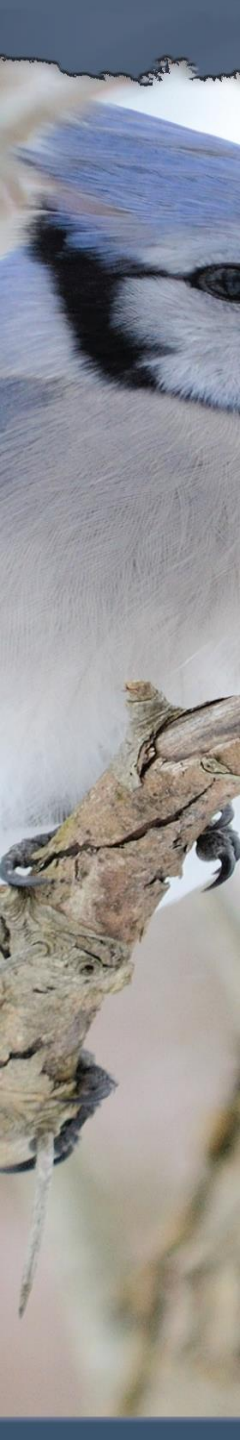
Multiattack (Yuan-ti Form Only). The yuan-ti makes one constrict attack and one scimitar attack.

Constrict. *Melee Weapon Attack:* +5 to hit, reach 10 ft., one target. *Hit:* 10 (2d6 + 3) bludgeoning damage, and the target is grappled (escape DC 14) if it is a Large or smaller creature. Until this grapple ends, the target is restrained, and the yuan-ti can't constrict another target.

Scimitar (Yuan-ti Form Only). *Melee Weapon Attack:* +5 to hit, reach 5 ft., one target. *Hit:* 6 (1d6 + 3) slashing damage.

Invoke Nightmare (Recharges after a Short or Long Rest). The yuan-ti taps into the nightmares of a creature it can see within 60 feet of it and creates an illusory, immobile manifestation of the creature's deepest fears, visible only to that creature. The target must make a DC 13 Intelligence saving throw. On a failed save, the target takes 11 (2d10) psychic damage and is frightened of the manifestation, believing it to be real. The yuan-ti must concentrate to maintain the illusion (as if concentrating on a spell), which lasts for up to 1 minute and can't be harmed. The target can repeat the saving throw at the end of each of its turns, ending the illusion on a success, or taking 11 (2d10) psychic damage on a failure.

Shapechanger. The yuan-ti can use its action to polymorph into a Medium snake or back into its true form. Its statistics are the same in each form. Any equipment it is wearing or carrying isn't transformed. If it dies, it stays in its current form.



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;
```

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

Kompileringsfeil? Kjørefeil?

```
1  public class Test
2  {
3      public void someMethod(){
4          MessagePost msgPost = new MessagePost("Per", "Hallo!");
5          PhotoPost photoPost = new PhotoPost("Gro", "C:/filen.jpg", "Flott bilde!");
6          Post post;
7          post = msgPost;
8          msgPost = post;
9      }
10 }
```

Kompileringsfeil? Kjørefeil?

```
1  public class Test
2  {
3      public void someMethod(){
4          MessagePost msgPost = new MessagePost("Per", "Hallo!");
5          PhotoPost photoPost = new PhotoPost("Gro", "C:/filen.jpg", "Flott bilde!");
6          Post post;
7          post = msgPost;
8          msgPost = (MessagePost) post;
9      }
10 }
```

Kompileringsfeil? Kjørefeil?

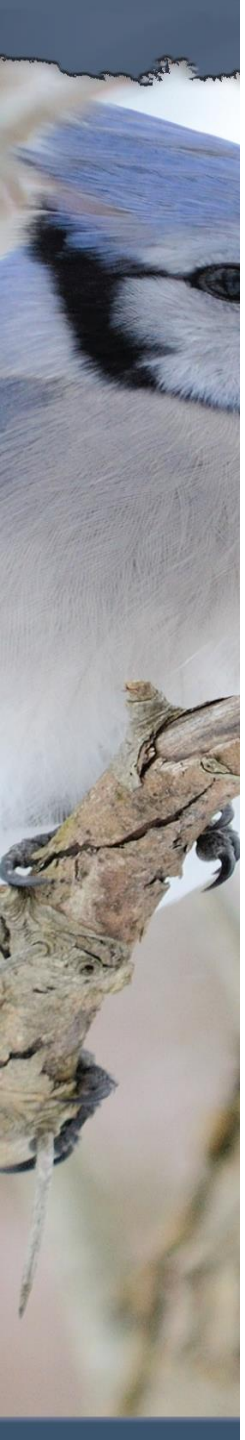
```
1  public class Test
2  {
3      public void someMethod(){
4          MessagePost msgPost = new MessagePost("Per", "Hallo!");
5          PhotoPost photoPost = new PhotoPost("Gro", "C:/filen.jpg", "Flott bilde!");
6          Post post = new MessagePost("Bente", "Heisann!");;
7          photoPost = msgPost;
8          msgPost = (MessagePost) post;
9      }
10 }
```

Kompileringsfeil? Kjørefeil?

```
1  public class Test
2  {
3      public void someMethod(){
4          Post msgPost = new MessagePost("Per", "Hallo!");
5          PhotoPost photoPost = new Post("Gro");
6          Post post = new MessagePost("Bente", "Heisann!");
7          msgPost = (MessagePost) post;
8      }
9  }
```

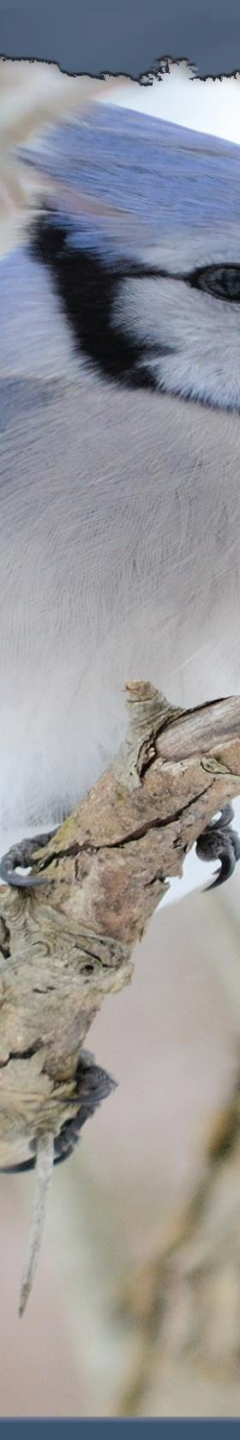
Kompileringsfeil? Kjørefeil?

```
1  public class Test
2  {
3      public void someMethod(){
4          Post msgPost = new MessagePost("Per", "Hallo!");
5          PhotoPost photoPost = new PhotoPost("Gro", "C:/bildet.png", "Knallbilde!");
6          Post post = photoPost;
7          msgPost = (MessagePost) post;
8      }
9  }
```



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.



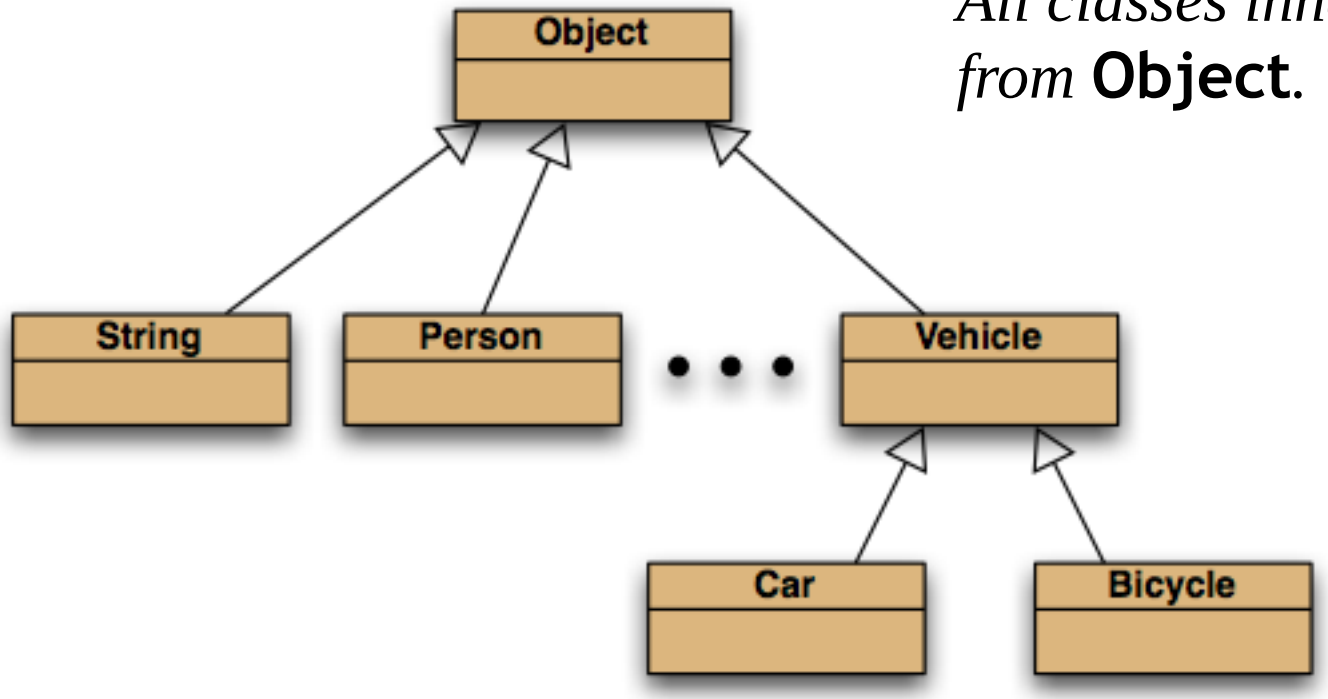
Instanceof (kap 11)

Hmm, as we may have polymorphic variables in Java… Can I investigate if my superclass reference is actually a specific subclass?

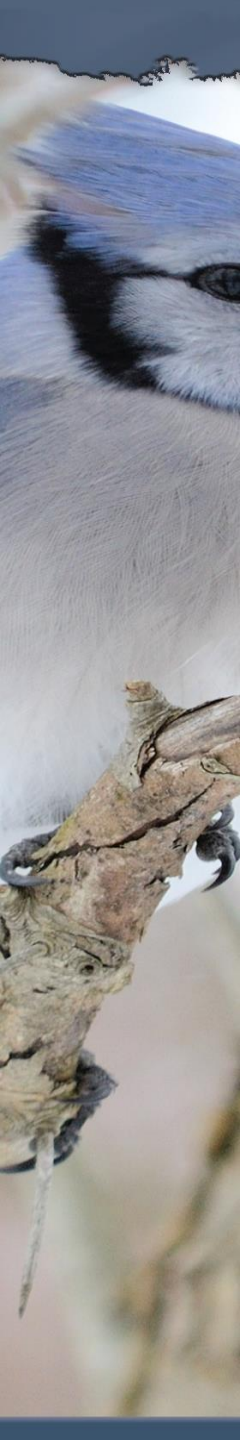
Yes, of course!

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

The Object class



All classes inherit from **Object**.

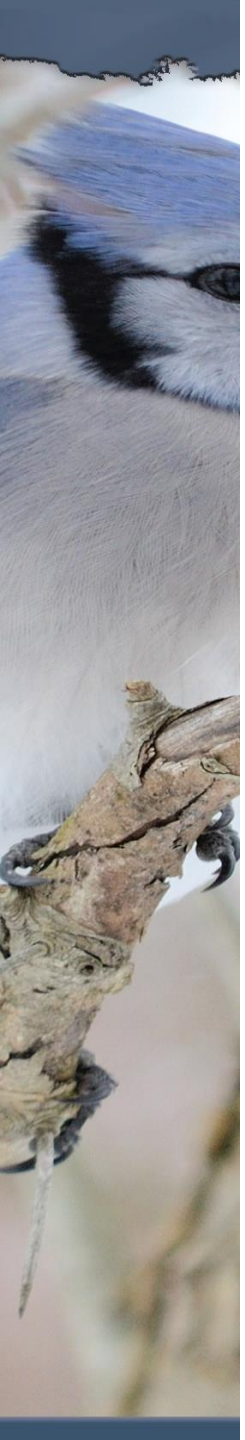


Polymorphic collections

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

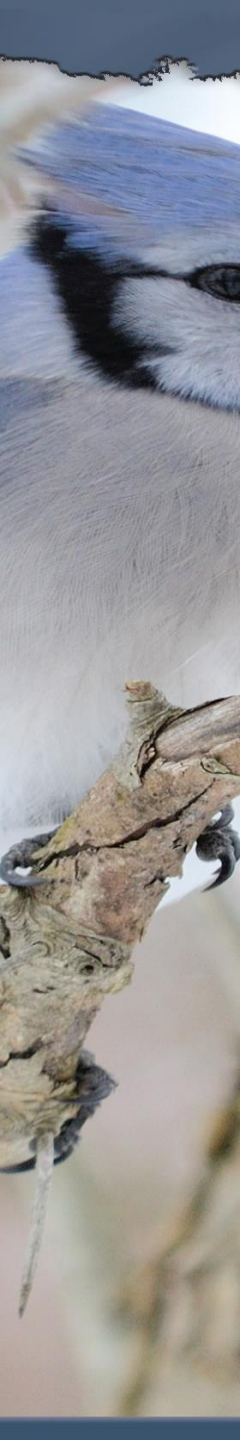
```
public void add(Object element)

public Object get(int index)
```

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



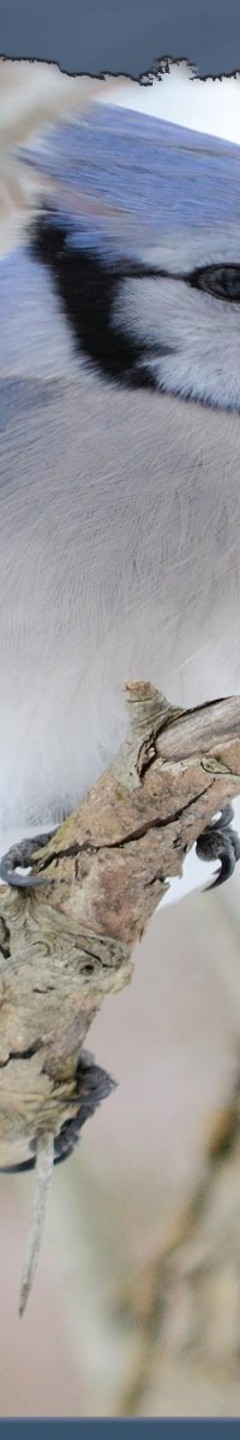
Polymorphic collections

- 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.



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).



Neste gang

- Mer arv (kapittel 11).
- Vi kommer til å bruke 2 uker på kapittel 11.

- Nå Kahoot, før øving…