INHERITANCE 1

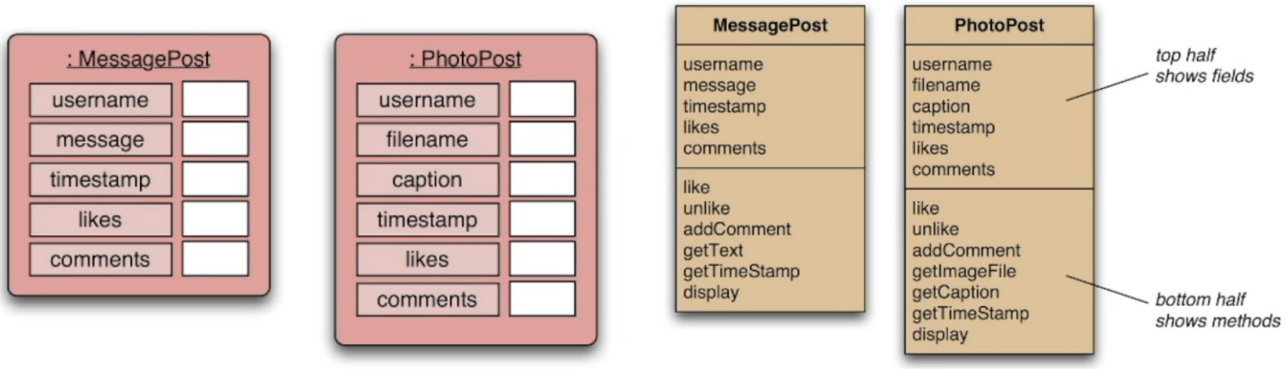
THE NETWORK EXAMPLE

This example will be based around a social-networking application like Twitter, Facebook, etc.

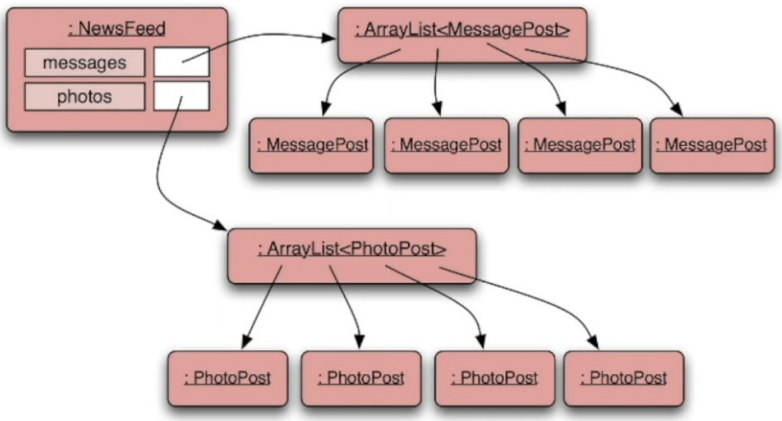
It will:

- Support 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.

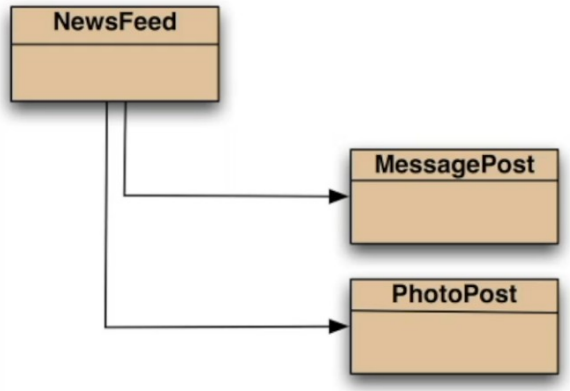
POST MODEL



NEWSFEED OBJECT



CLASS DIAGRAM



SOURCE CODE INSPECTION

MESSAGEPOST AND PHOTOPOST

```
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<>();
   }

   public void addComment(String text) ...

   public void like() ...

   public void display() ...

   ...
}
```

```
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<>();
   }

   public void addComment(String text) ...
   public void like() …
   public void display() …
   ...
}
```

NEWSFEED

```
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
        }
   }
}
```

CRITIQUE OF NETWORK

Code Duplication

- MessagePost and PhotoPost classes are very similar in the sense that large parts are identical.
    - Makes maintenance difficult/more work
    - Introduces danger of bugs through incorrect maintenance
- NewsFeed also has a lot of code duplication.

So how do we solve these problems?

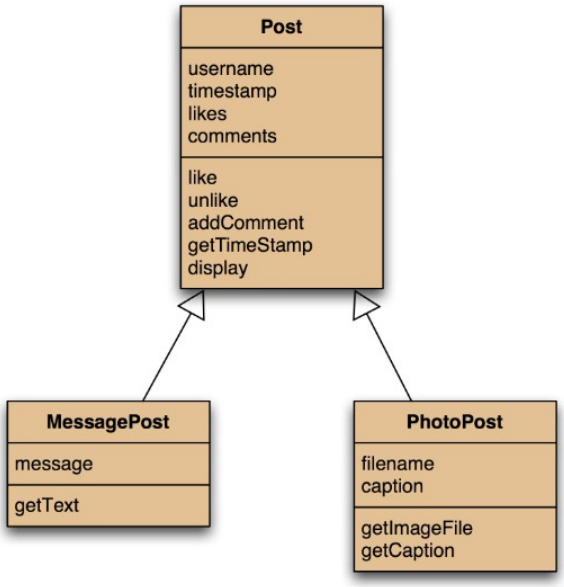
We solve them by using...

INTRODUCING INHERITANCE

We once had two separate classes that were independent.

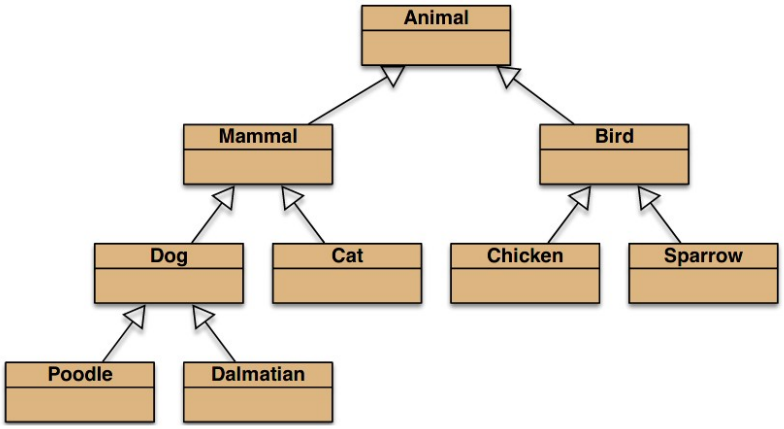
This is the diagram for Inheritance.

- Inheritance declares a single superclass that defines the common generic attributes.
- All the specific fields are left in the subclasses.
- In a UML diagram, a hollow arrow represents an inheritance arrow. It is a “special case” of the superclass.



USING INHERITANCE

- Define one **superclass**: Post
- Define subclasses as **MessagePost** and **PhotoPost**
- The superclass defines common attributes (via fields)
- The subclasses inherit the superclass characteristics
- The subclasses add other characteristics



INHERITANCE IN JAVA

SUPERCLASS

```
public class SuperClass
{
      ...
}
```

SUBCLASS

```
public class SubClass extends Superclass
{
      ...
}
```

CONSTRUCTORS

The superclass constructor is the same.

However, with the subclasses…

```
[...]
/**
 * Constructor for objects of class MessagePost
 */
public SubClass(String fromSuperclass, String newVariable)
{
      super(fromSuperclass);
      newVariable = this.newVariable;
}
[...]
```

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

    // constructor and methods omitted.
}
```

```
public class MessagePost extends Post
{
    private String message;

    // constructor and methods omitted.
}
```

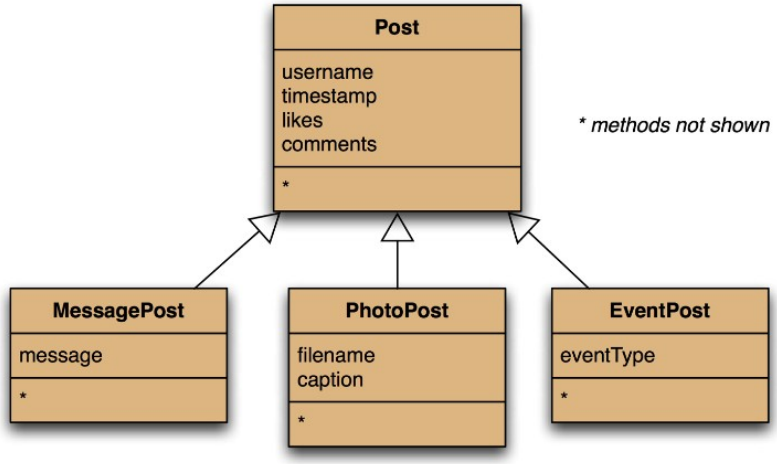
```
public class PhotoPost extends Post
{
    private String filename;
    private String caption;

    // constructor and methods omitted.
}
```

```
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
}
```

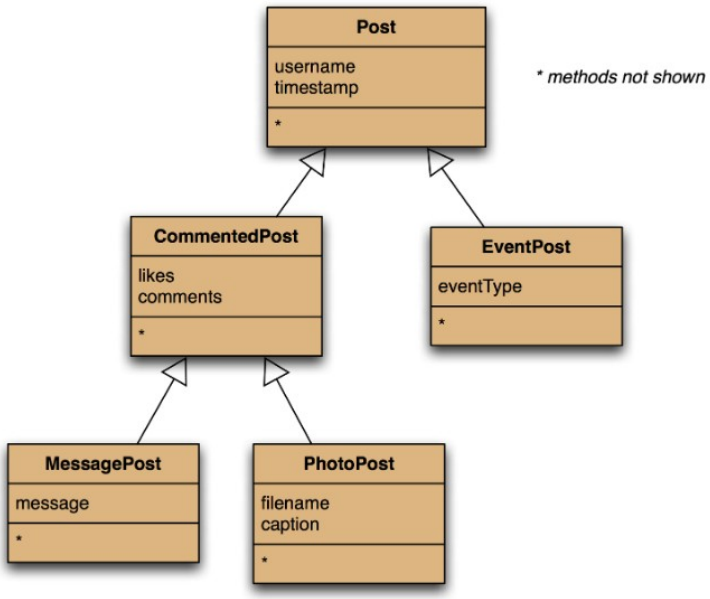
SUPERCLASS CONSTRUCTOR CALL

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

ADDING MORE ITEM TYPES



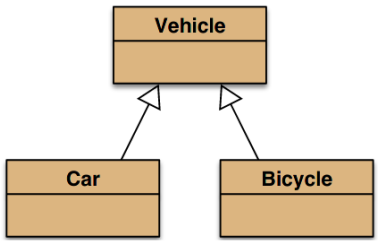
DEEPER HIERARCHIES



SUBTYPING

Subclass objects may be assigned to superclass variables.

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



i.e: Subtyping!

SUBTYPING IN JAVA

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

First we had:

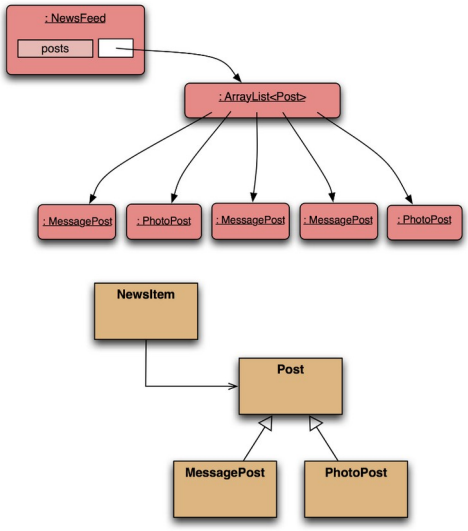
```
public void addMessagePost(MessagePost message)
public void addPhotoPost(PhotoPost photo)
```

Now we have:

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

We call this method with:

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



POLYMORPHIC VARIABLES

```
Person p = new Person();
Student s = new Student();

p = s;
s = p;
```

Which of these works?

- Object variables in Java are polymorphic.
  (They can hold objects of more than one type.)
- They can hold objects of the declared type, or subtypes of the declared type.

CASTING

We can assign a subtype to a supertype, but we cannot assign a supertype to a subtype!

```
p = s; //correct
s = p; //compile-time error
```

Casting fixes this:

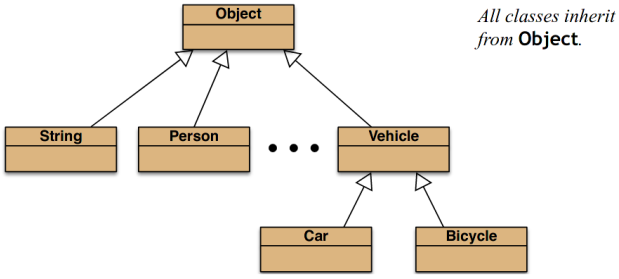
```
s = (Student) p;
```

But this is only fine if the person really is a Student.

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.
- 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.