Software Development 2

Improving Structure with Inheritance

F27SB

Marks for Labs

- Lab 2 is due this week
- In general:
  - “No individual extensions are permitted under any circumstances” *- University’s Submission of Coursework Policy.*
  - Need to apply for Mitigating Circumstances or Temporal Suspension of Studies

So far

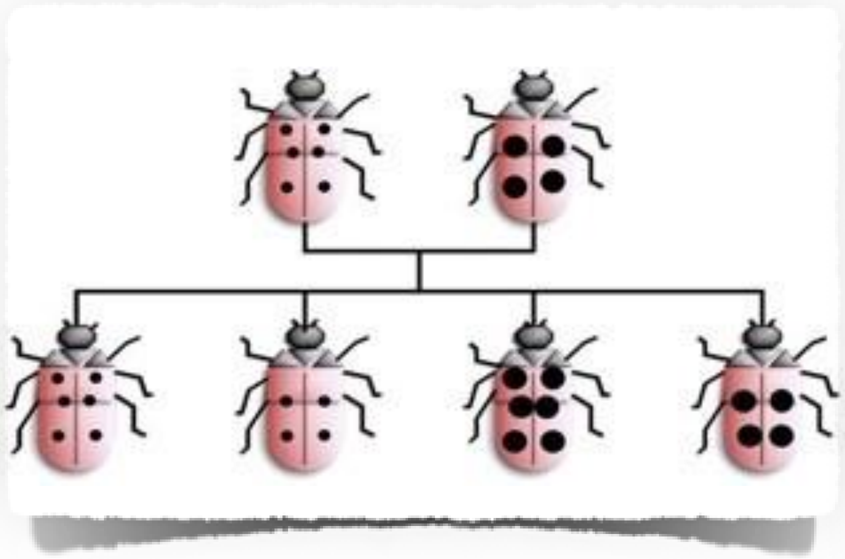
- Java as a Object-Oriented language.
- Low coupling and high cohesion make good code.
- Code duplication is a bad sign.
- Refactoring: Code needs to be maintained.
- JUnit tests for test-driven development.

Today

INHERITANCE

Inheritance?

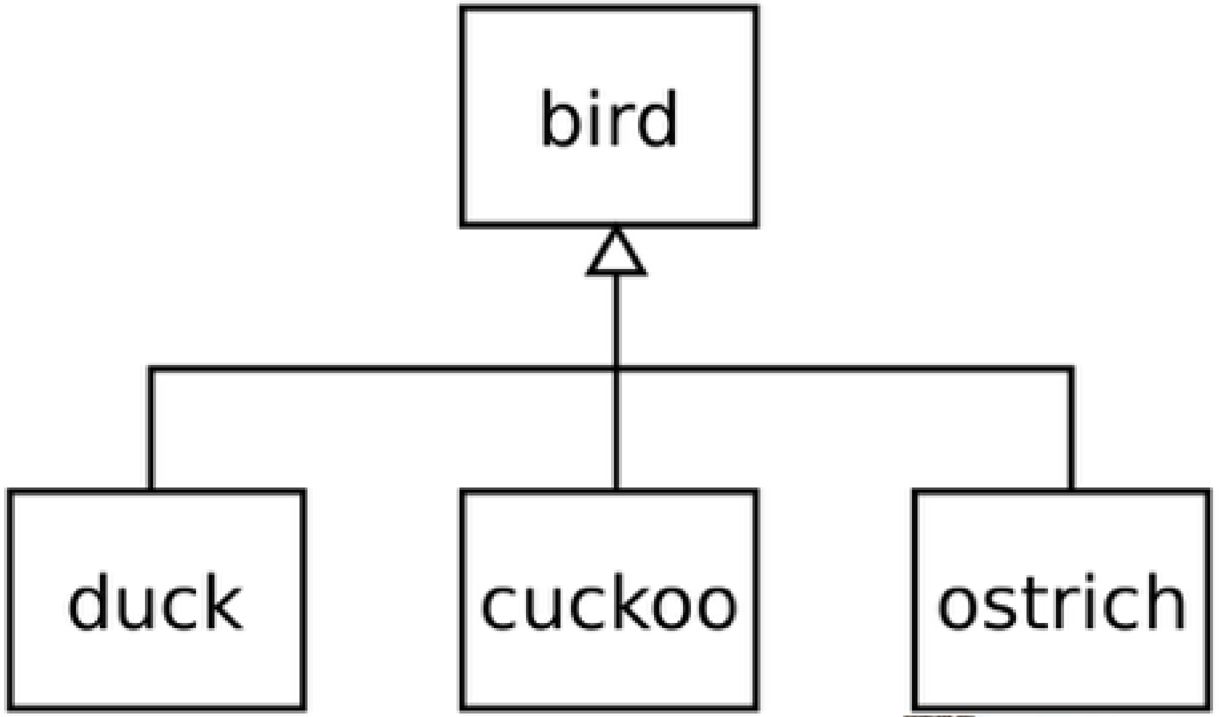
Biology



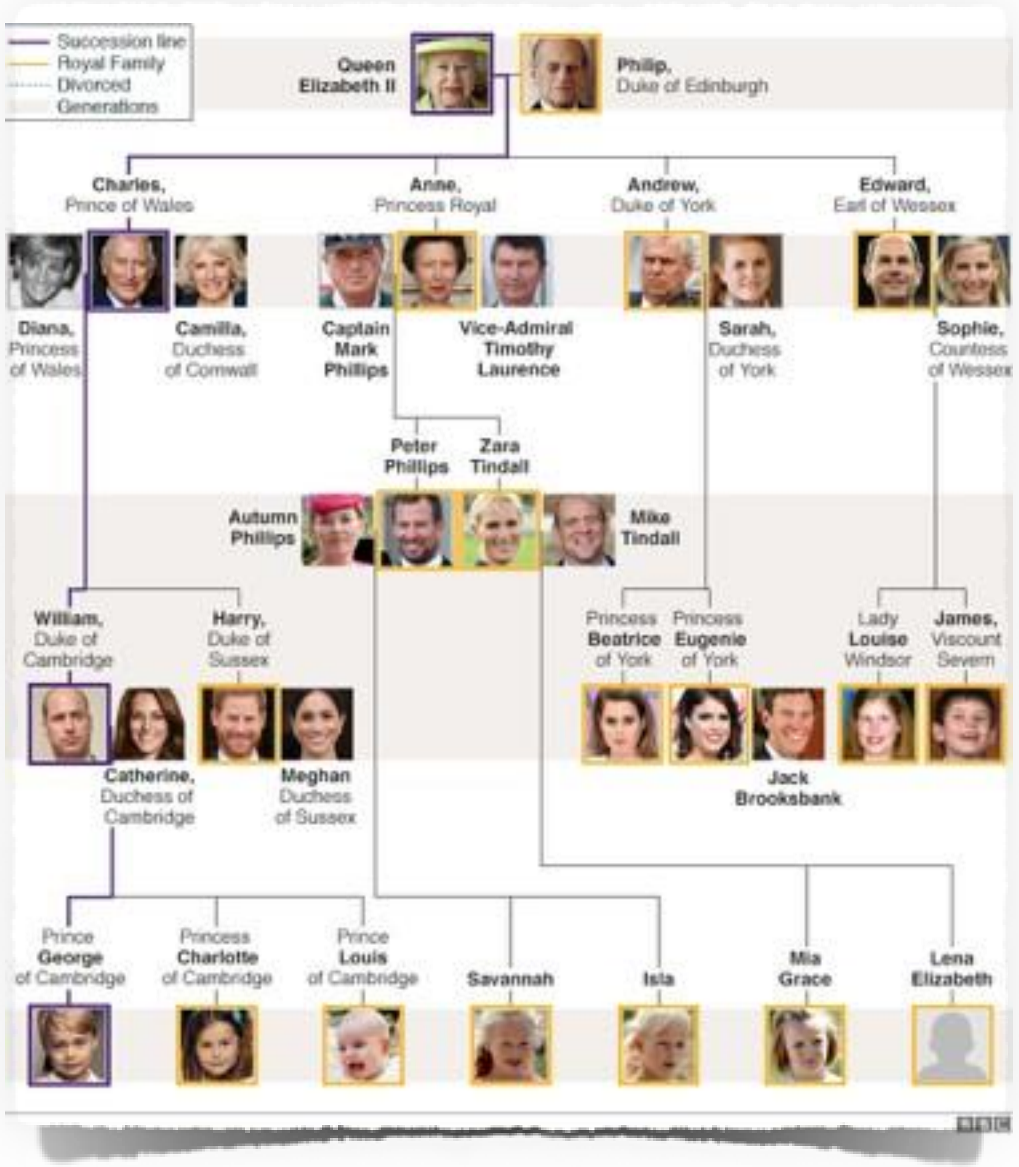
Money/legal

Inheritance as an object-oriented design principle!

OOP Inheritance is about sharing features…



… by belonging to one FAMILY (Type).



Inheritance:

- Avoids Code Duplication
- Improves Cohesion
- Reduces Coupling

Main concepts to be covered

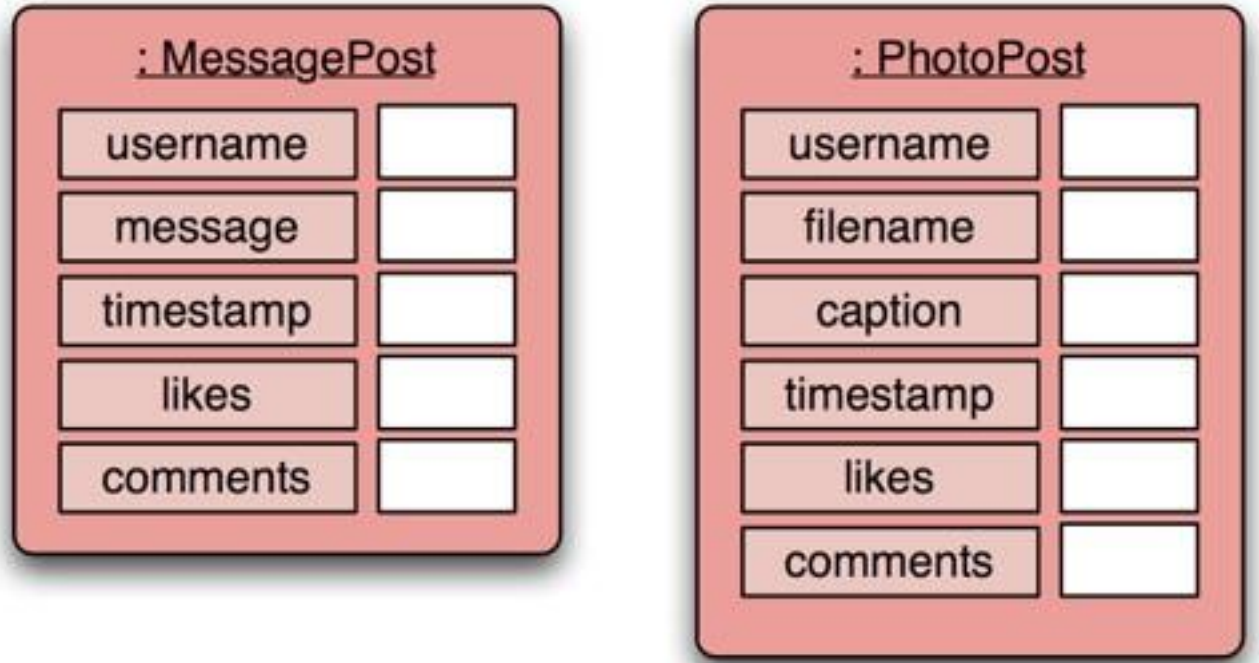
- Inheritance
- Subtyping
- Substitution
- Polymorphic variables

EXAMPLE

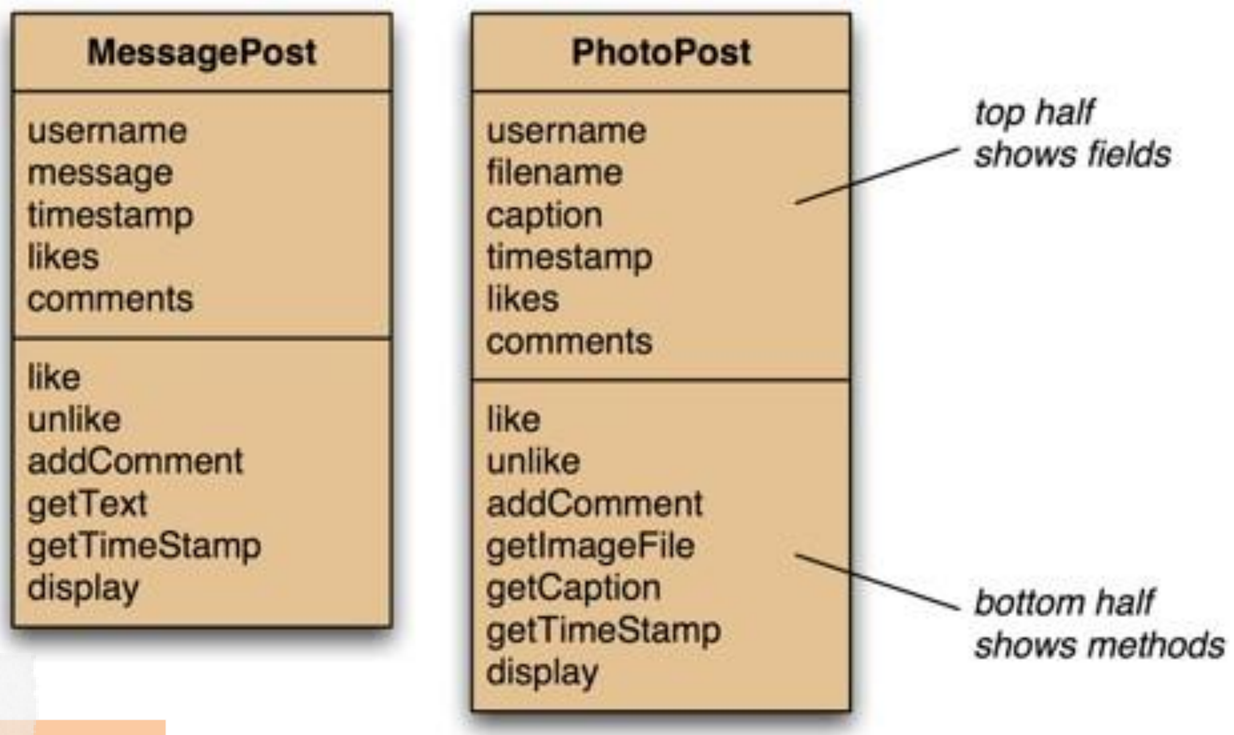
The Network example

- 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

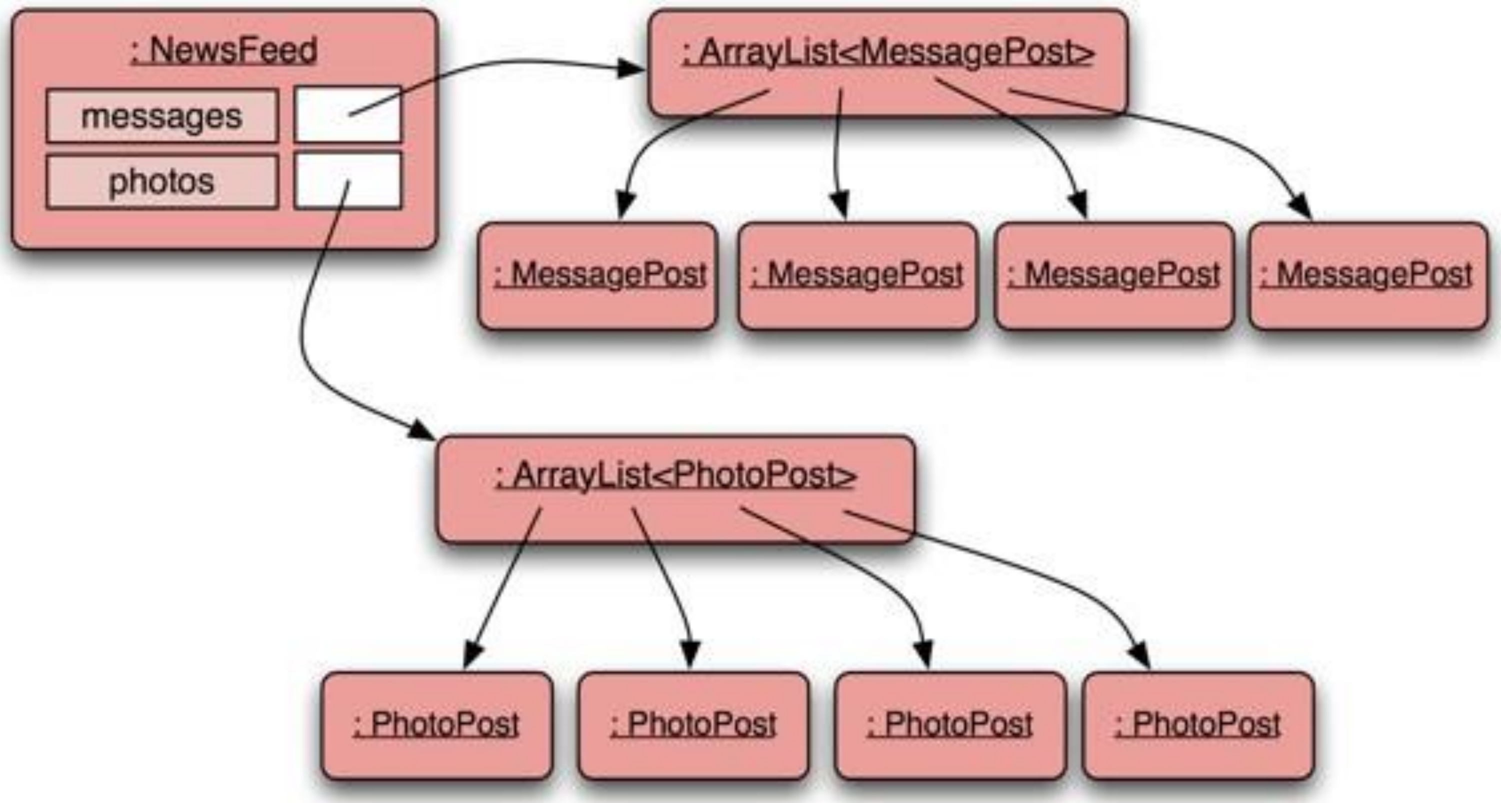


Network classes

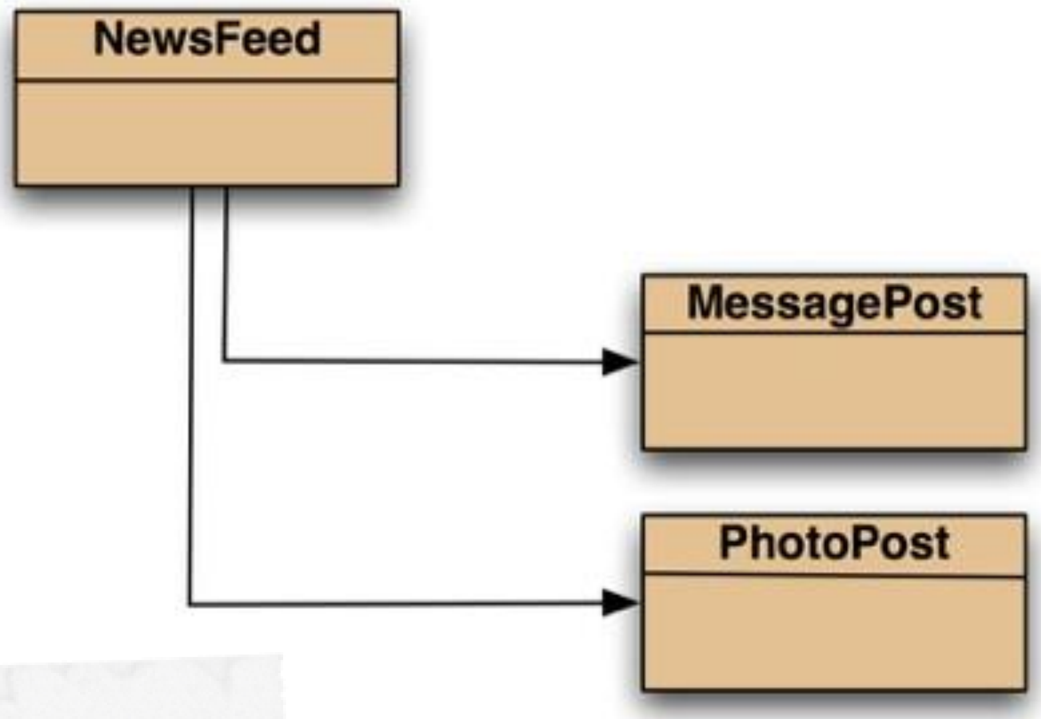


Visibility modifiers:
+Public
# Protected
- Private

Network object model



Class diagram for Network

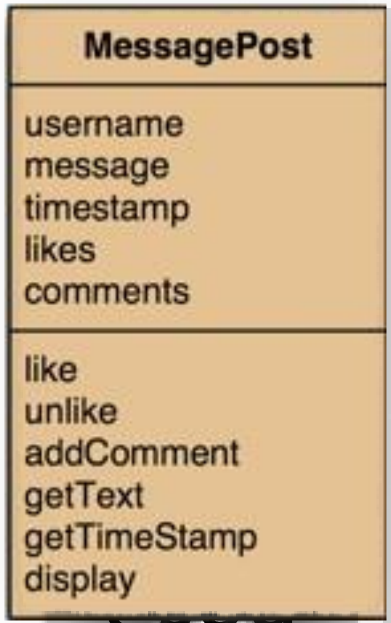


Unidirectional association

Message-Post source code

```java
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() ...

   ...
}
```

Just an outline!



M

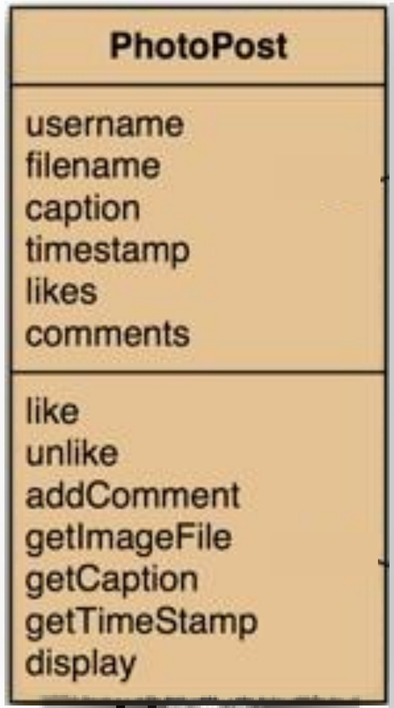
```
public class MessagePost
{
   private String username;
   private String message;
   private long timestamp;
   private int likes;
   private ArrayList<String> comments;

   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() ...

   ...
}
```

Just an outline!

Photo-Post source code

```
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() …
   ...
}
```

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() …
   ...
}
```

Just an
outline!

NewsFeed source code

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

        for(PhotoPost photo : photos) {
            photo.display();
            //empty line between posts
            System.out.println();
        }
    }
}
```

Just an outline!

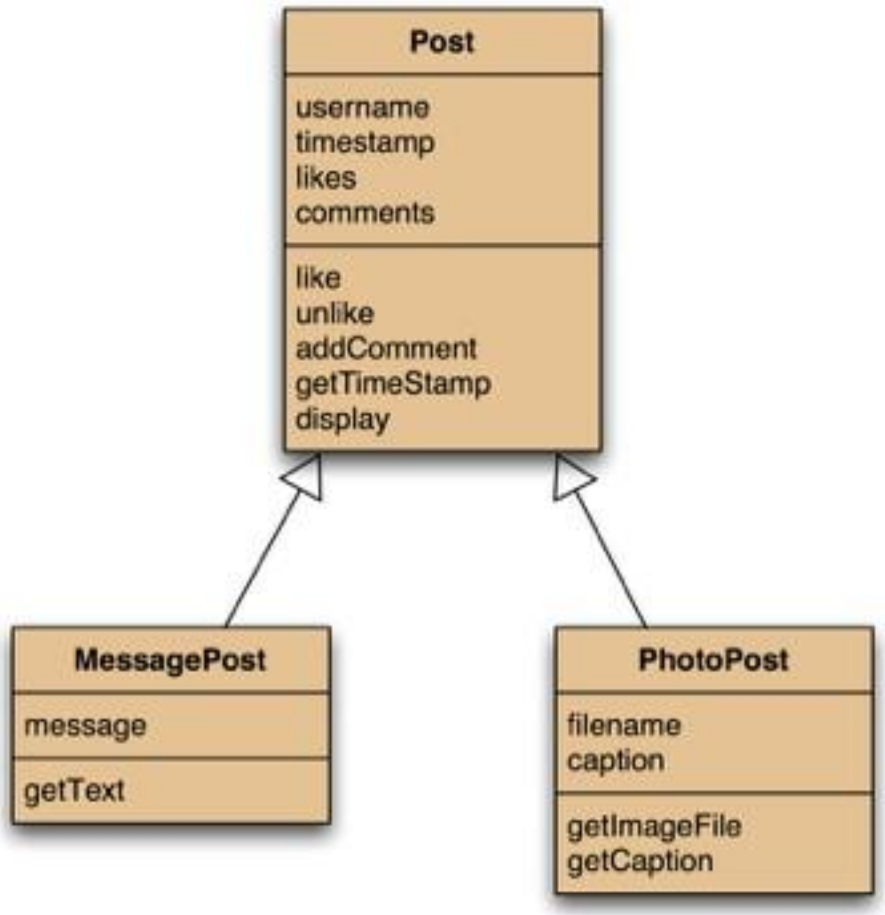
Solving things

USING INHERITANCE

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

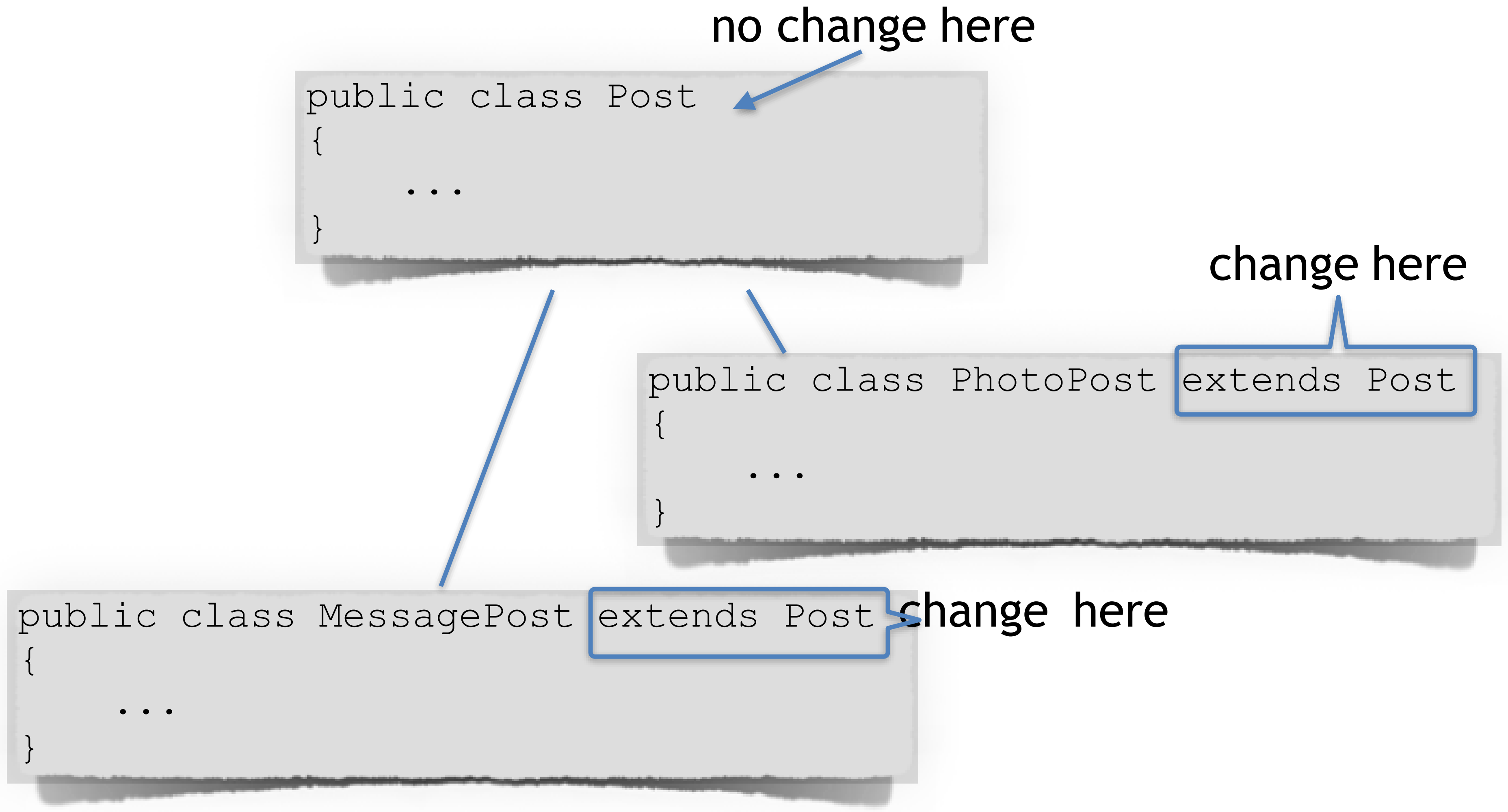


Inherits from

Using inheritance

- define one $\boxed{\textbf{superclass}}$: `Post`
- define $\boxed{\textbf{subclasses}}$ for `MessagePost` and `PhotoPost`
- the superclass defines common attributes (via fields)
- the subclasses **inherit** the superclass attributes
- the subclasses add other attributes

Inheritance in Java

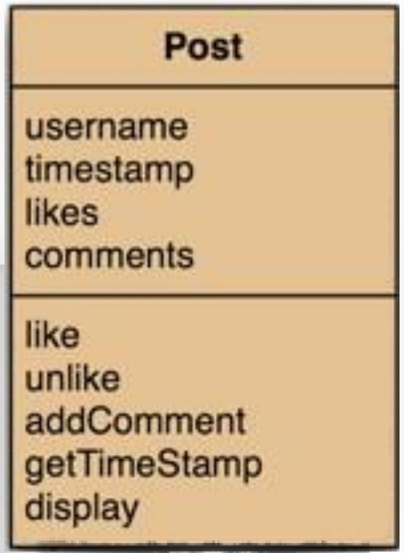


```
public class Post
{
      ...
}
```

```
public class PhotoPost extends Post
{
      ...
}
```

```
public class MessagePost extends Post
{
      ...
}
```

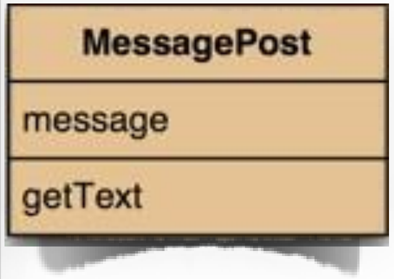
Superclass



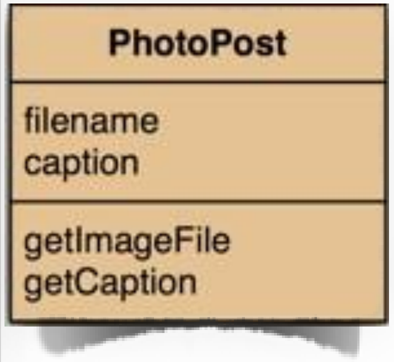
```
public class Post
{
    private String username;
    private long timestamp;
    private int likes;
    private ArrayList<String> comments;

    //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

```
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

```java
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
- Must be the first statement in the subclass constructor.

New NewsFeed source code

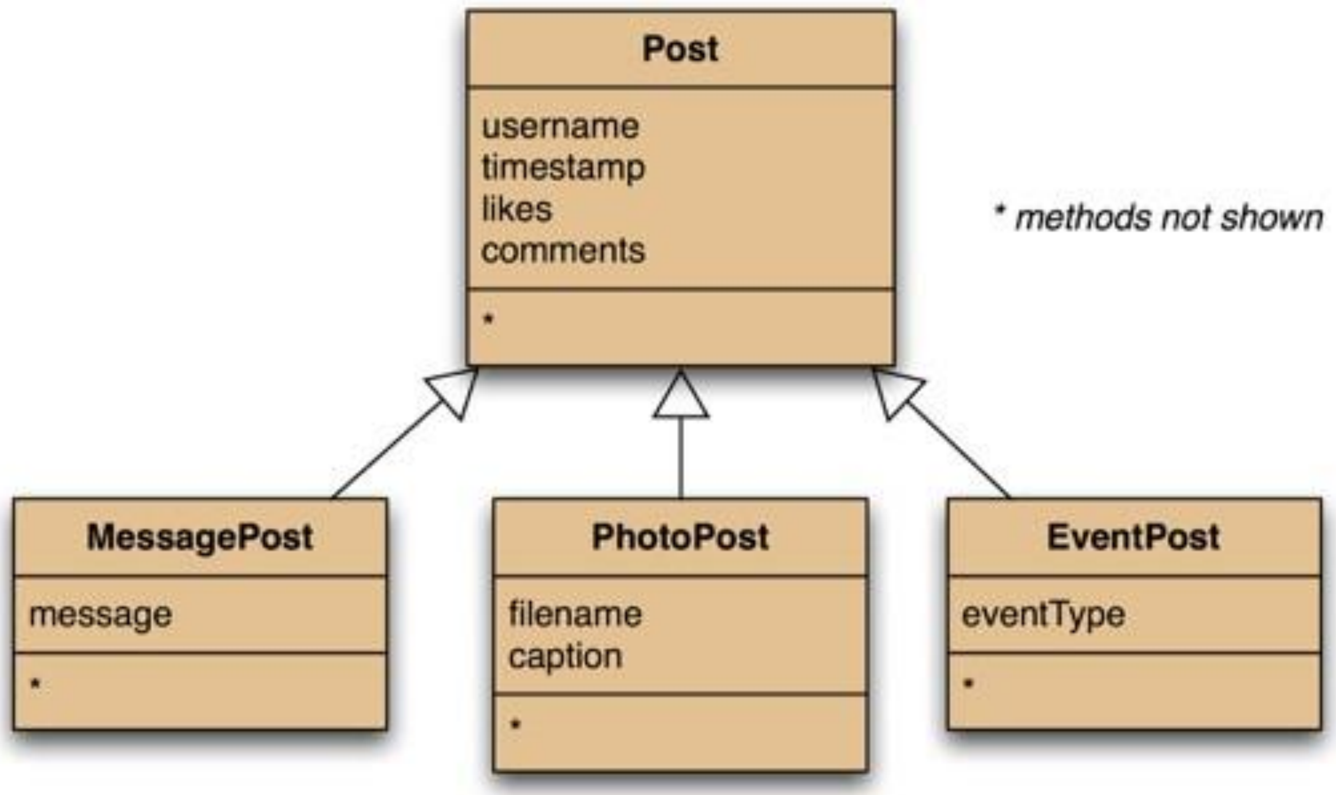
```
public class NewsFeed
{
    private ArrayList<Post> posts;

    /**
     * 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);
    }
    ...
}
```

avoids code duplication in the client class!

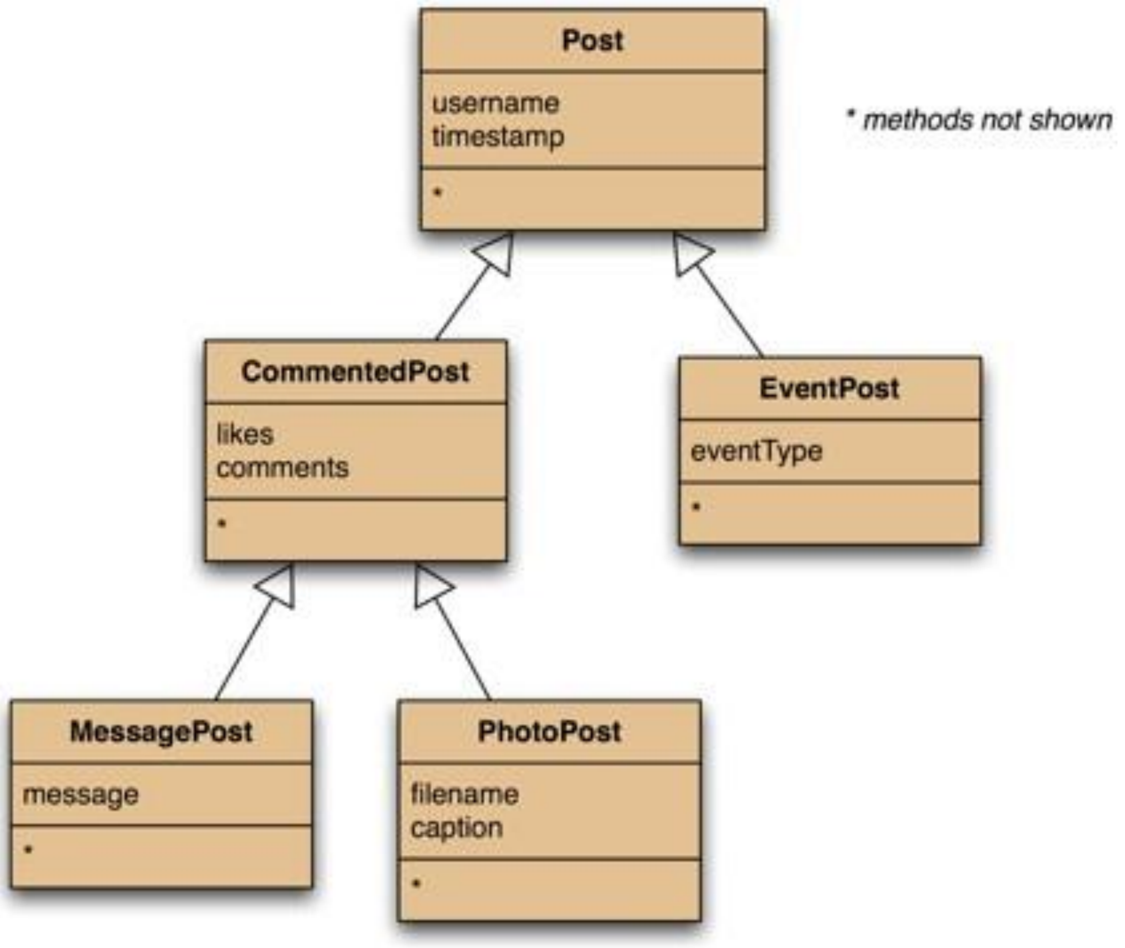
New NewsFeed source code

```
/**
 * 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();
       //Empty line ...
       System.out.println();
   }
}
```

Adding more item types



Deeper hierarchies



Review (so far)

Inheritance (so far) helps with:

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

No multiple inheritance!

THAT’S IT!

Homework

- Chapters 8.1 (“Network example”) – 8.6 (“Advantages of Inheritance”)