COMP8710 Advanced Java for Programmers

# Lecture 4 Inheritance & polymorphism

Yang He

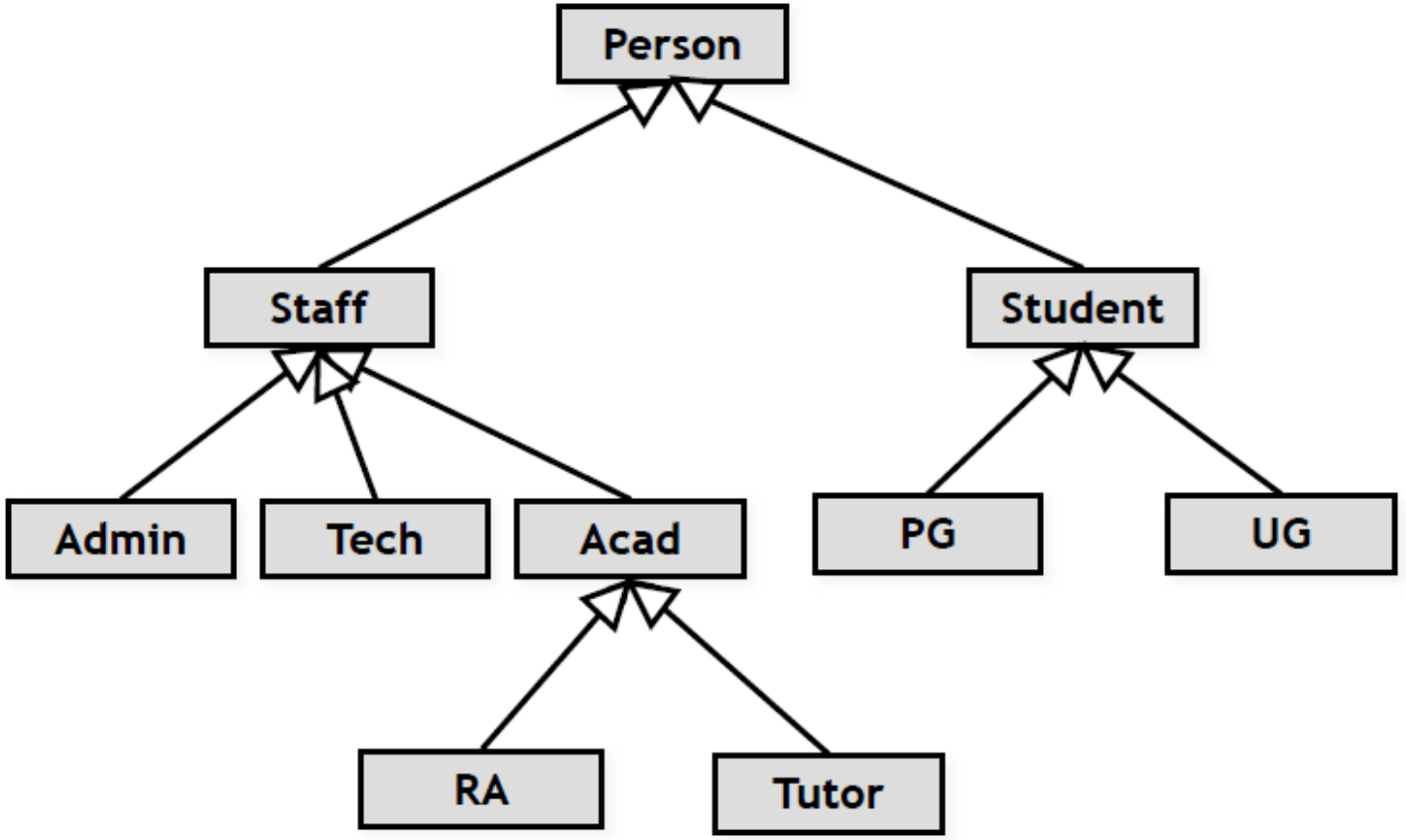
# Topics

- Inheritance
- Subtyping
- Substitution
- Polymorphic variables
- Casting
- Protected access

# Hierarchies are natural

- A university has both staff and students
  - Staff may be either academics (research assistant or tutor), technicians or administrator
  - Students are either research postgraduates (PG) or undergraduates (UG)
- Everyone has a login and a department
  - Staff have payroll and National Insurance numbers
  - Academics may teach modules; tutors may have tutees
  - Each student has a student number and an entry year
  - UG students have a programme and a tutor
  - PG students have a research topic and supervisor

# University hierarchy



# Is-A or Has-A relationships?

- What is the relationship between UG and Student? UG is a student.

- What is the relationship between UG and Tutor? UG has a tutor.

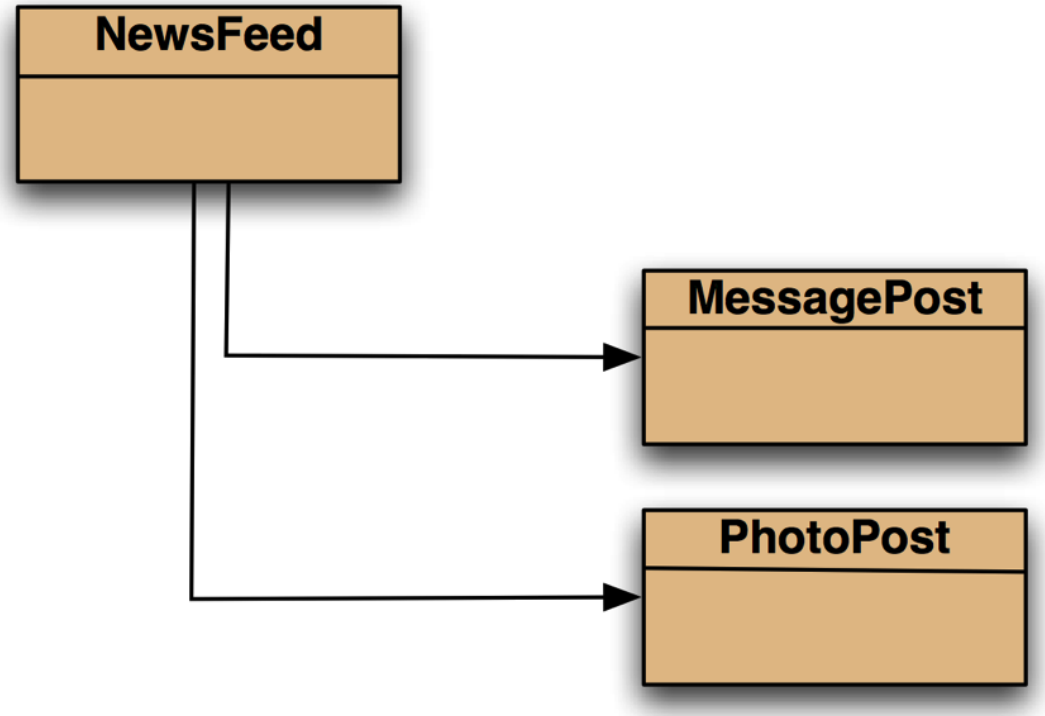
# Inheritance or aggregation?

- The distinction between Is-A and Has-A relationships is crucial to OOP design
  - Inheritance (behaves like) is an Is-A relationship
  - Aggregation (member fields) is a Has-A relationship
- The choice between using Is-A and Has-A is not always obvious, depending on the specific requirements
  - E.g. a manager is a member of staff. Alternatively, we might model staff as having a role attribute, which could be “Manager”

# The Network example

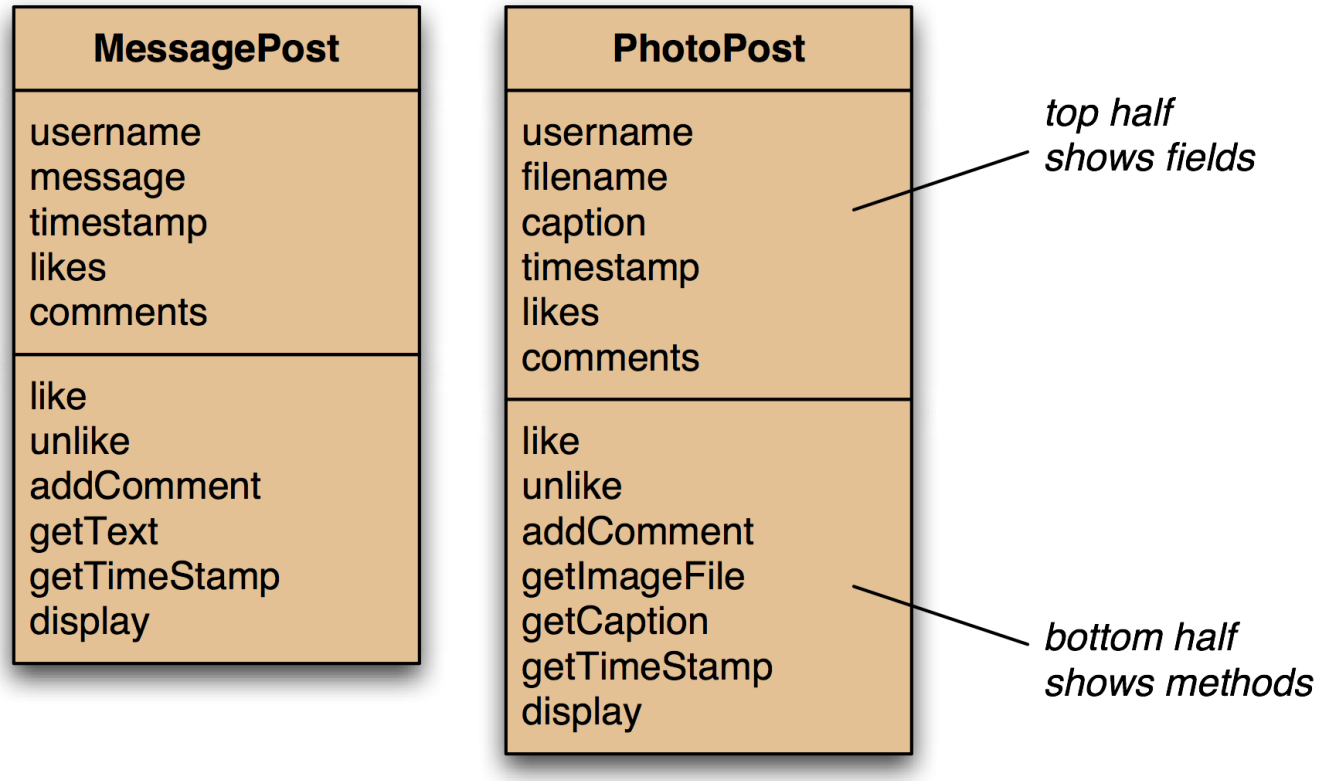
- A small, prototype social network
- Supports a news feed with posts
- Stores text posts and photo posts
  - MessagePost: multi-line text message
  - PhotoPost: photo and caption
- Allows operations on the posts:
  - E.g. search, display and remove

# Network class diagram




- Static view of linkage (coupling) between classes:
  - Single direction
  - No information about degree

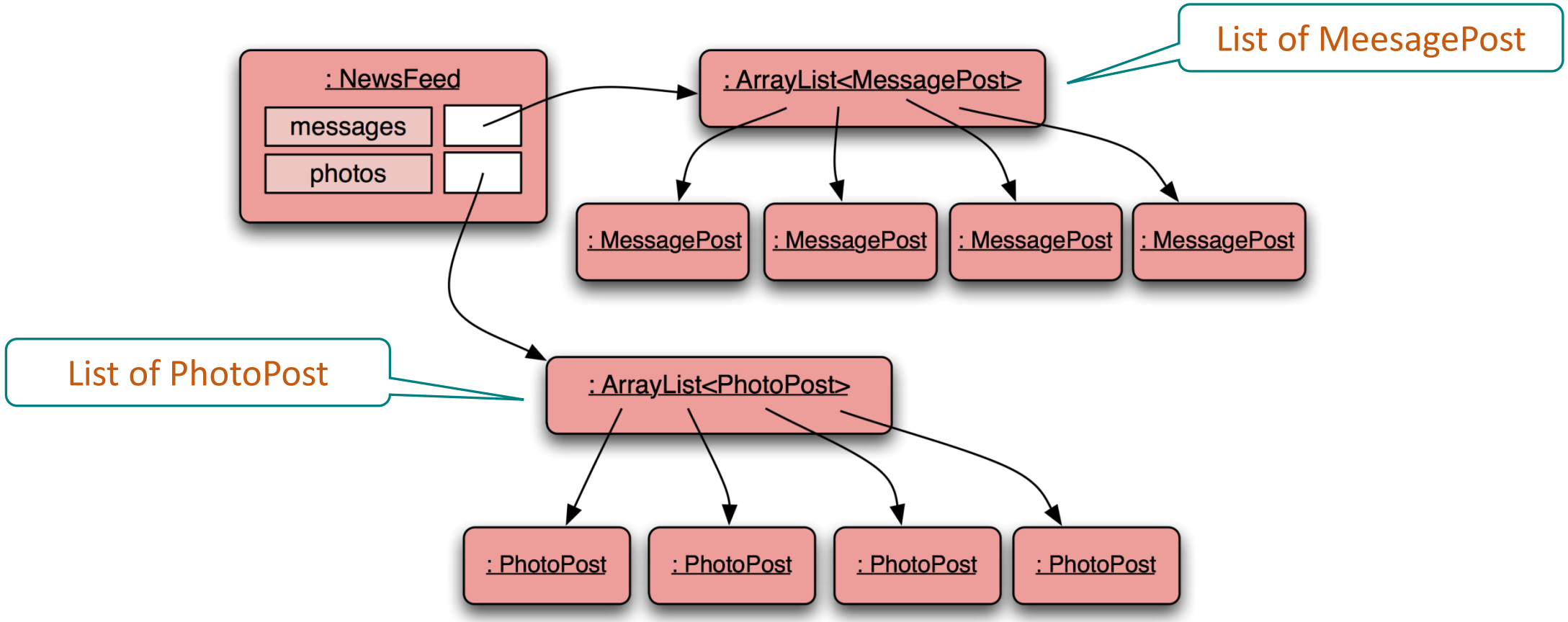
# Network classes




*MessagePost and PhotoPost have common fields and methods!*

# Object diagram

- Dynamic view of objects of difference classes



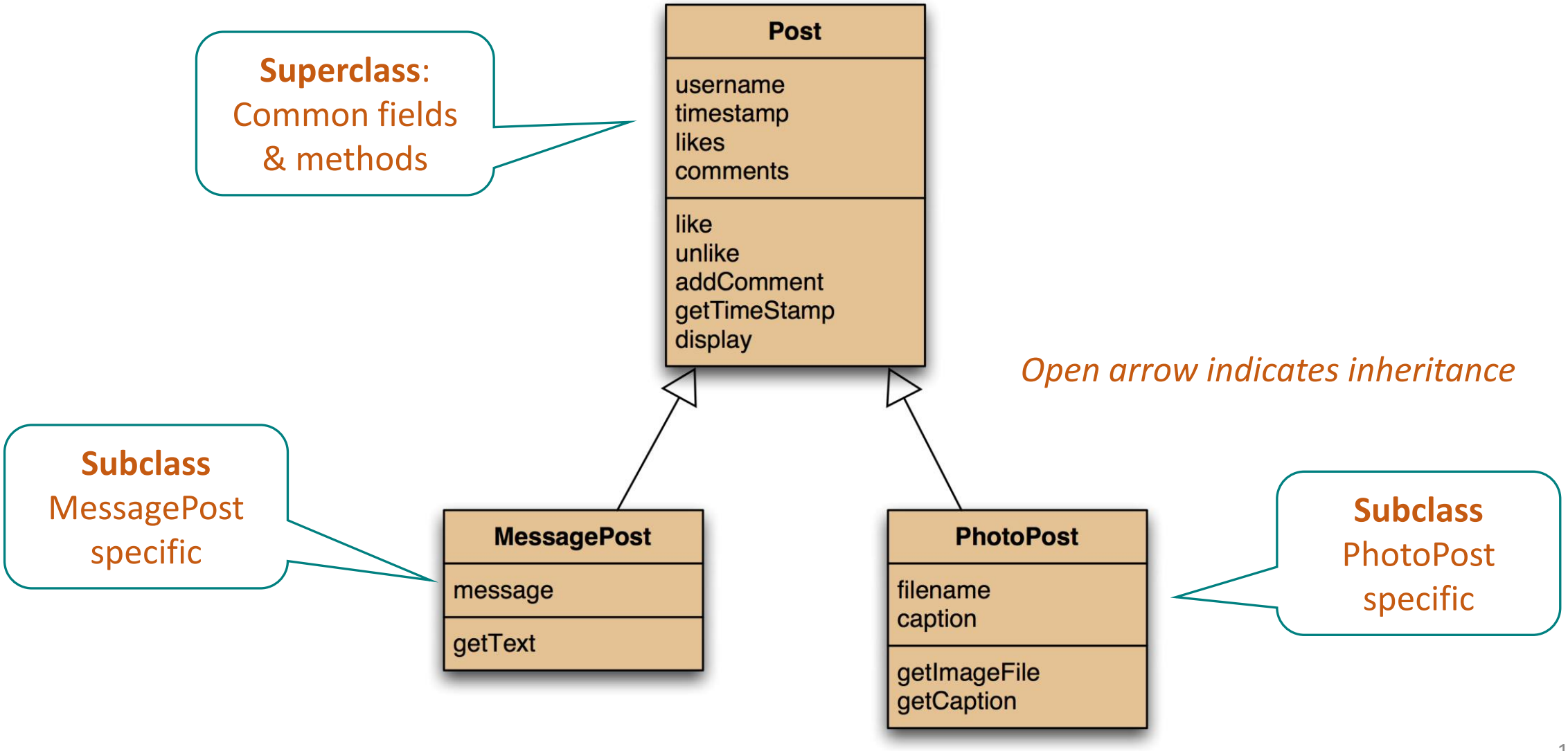
# Critique of Network

- 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.

*How can we improve the structure of the program?*

# Inheritance

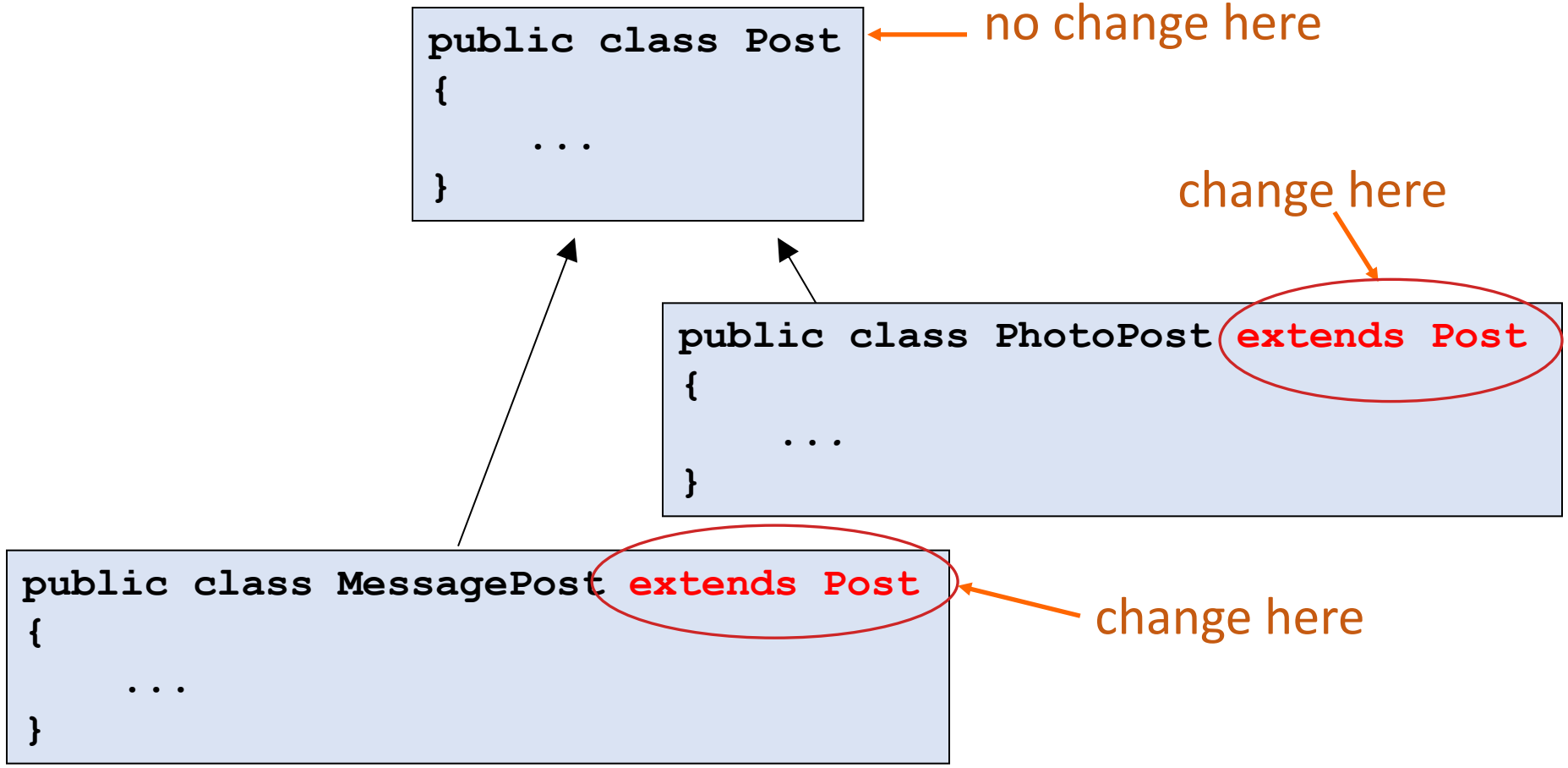
# Refactor using inheritance



# 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, they add other characteristics

# Inheritance in Java



# Superclass

Common fields

Constructor

```
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<>();
    }

    // methods omitted
}
```

# Subclasses

- E.g. MessagePost

```
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
}
```

MessagePost's own field

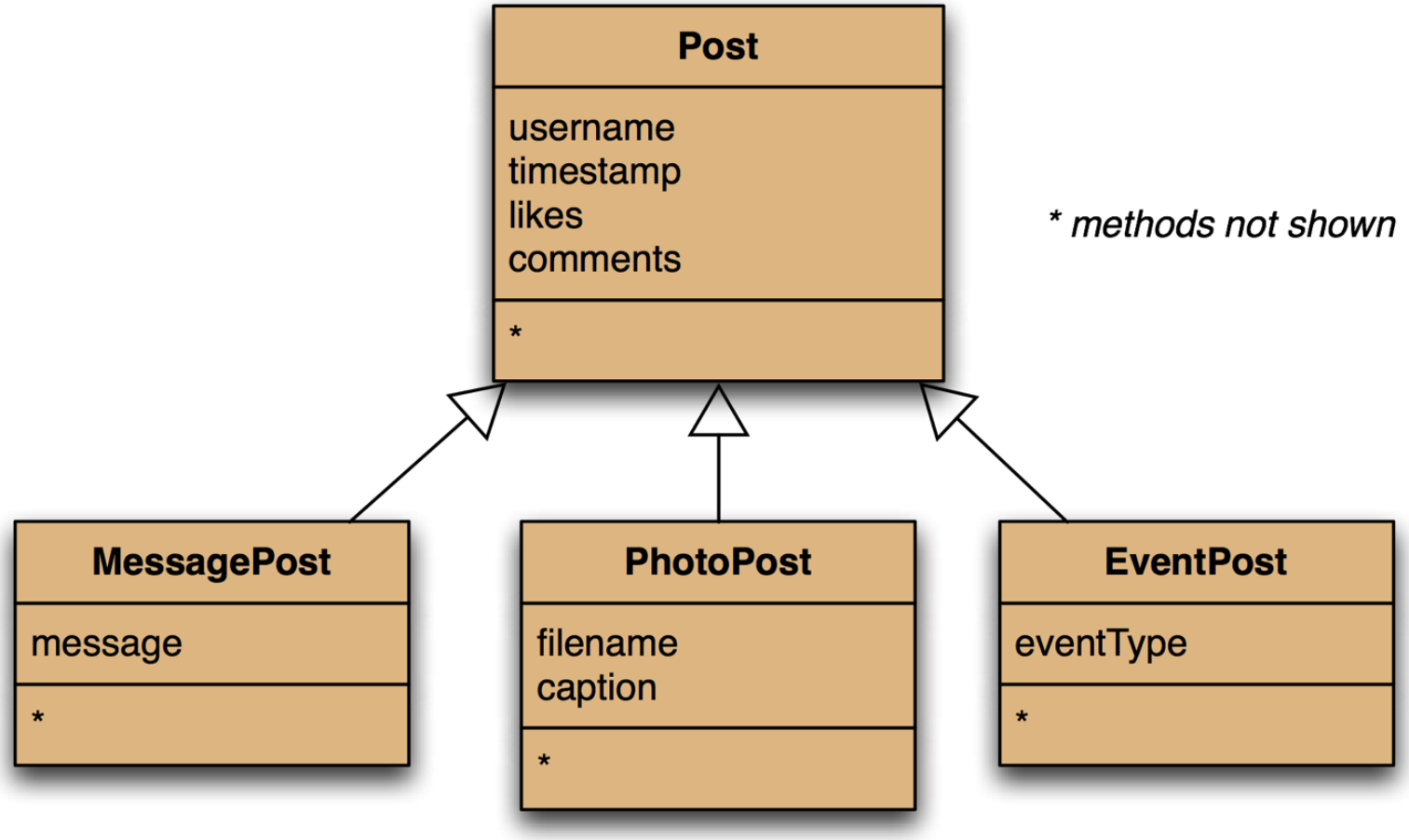
Constructor

Superclass constructor call

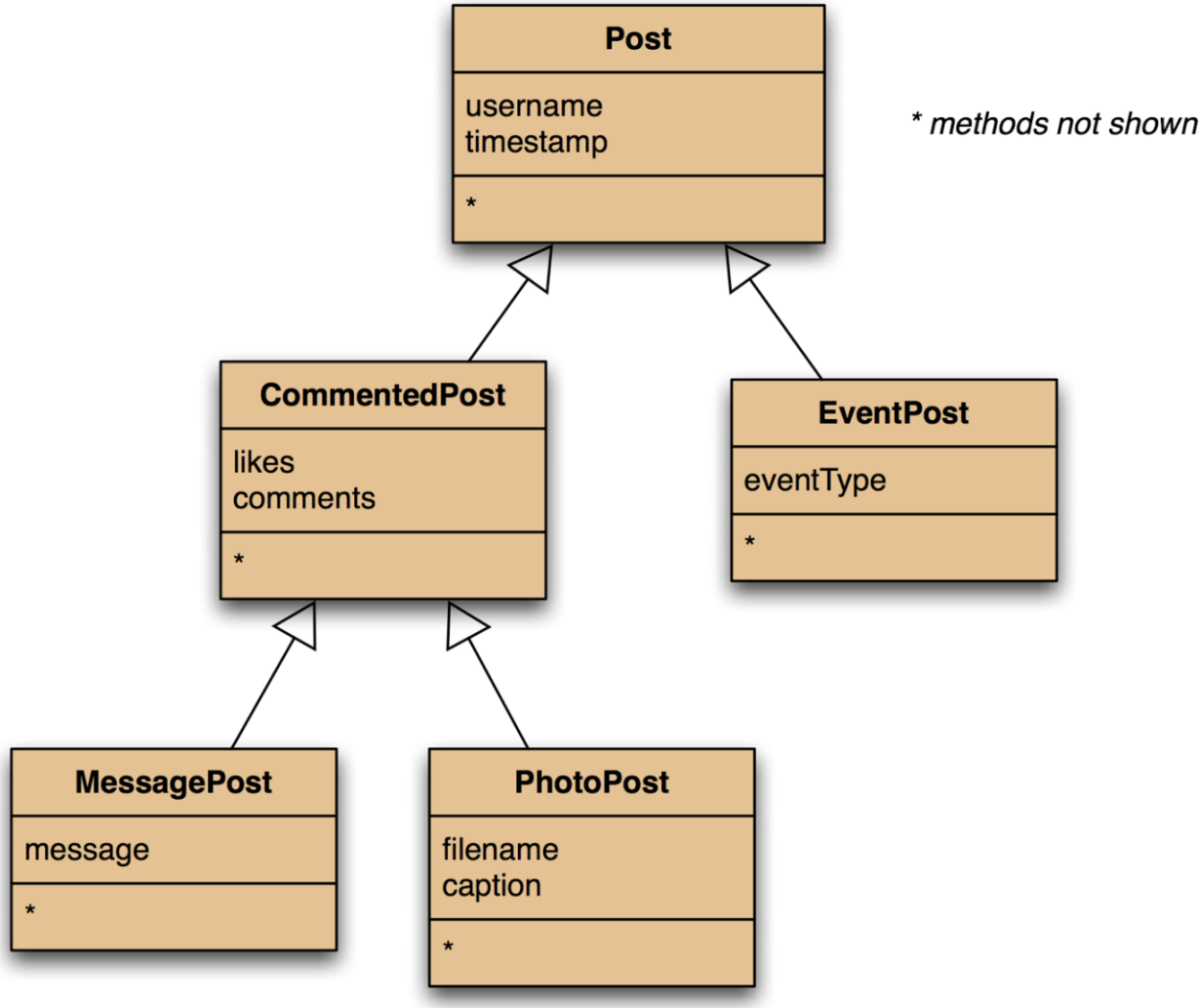
# Superclass constructor call

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

# Adding more item types



# Deeper hierarchies



# Review (so far)

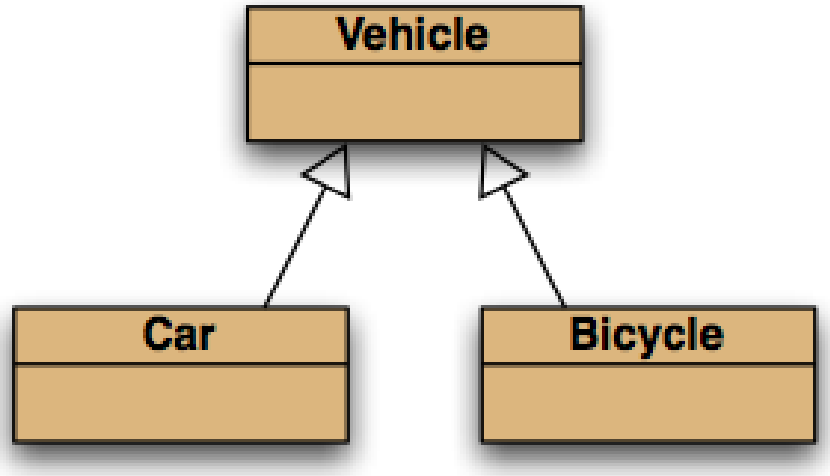
- Inheritance (so far) helps with:
  - Avoiding code duplication
  - Code reuse
  - Easier maintenance
  - Extendibility

# Subtype & supertype

# 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




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

Car c1 = new Vehicle();
```

✓
✓
✓

x

*Subclass objects may be assigned to superclass variables (substation)*

*But a superclass object cannot be assigned to a variable of any subclasses.*

*Why not?*

# Revised NewsFeed source code

```
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);
    }
    ...
```

*Avoids code duplication in the client class!*

```
public void show()
{
   for (Post post : posts) {
       post.display();
       System.out.println();
   }
}
```

# Subtyping and parameter passing

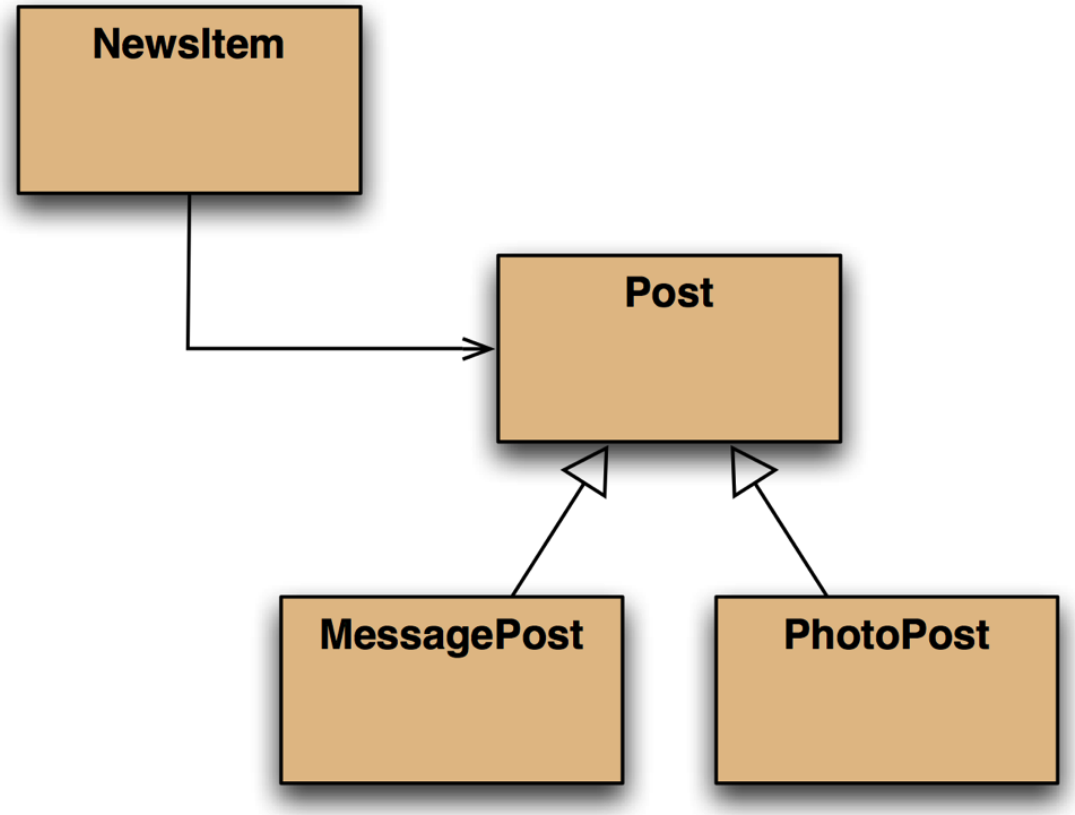
```
public class NewsFeed
{

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

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

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

*Subclass objects may be used as actual parameters for the superclass*

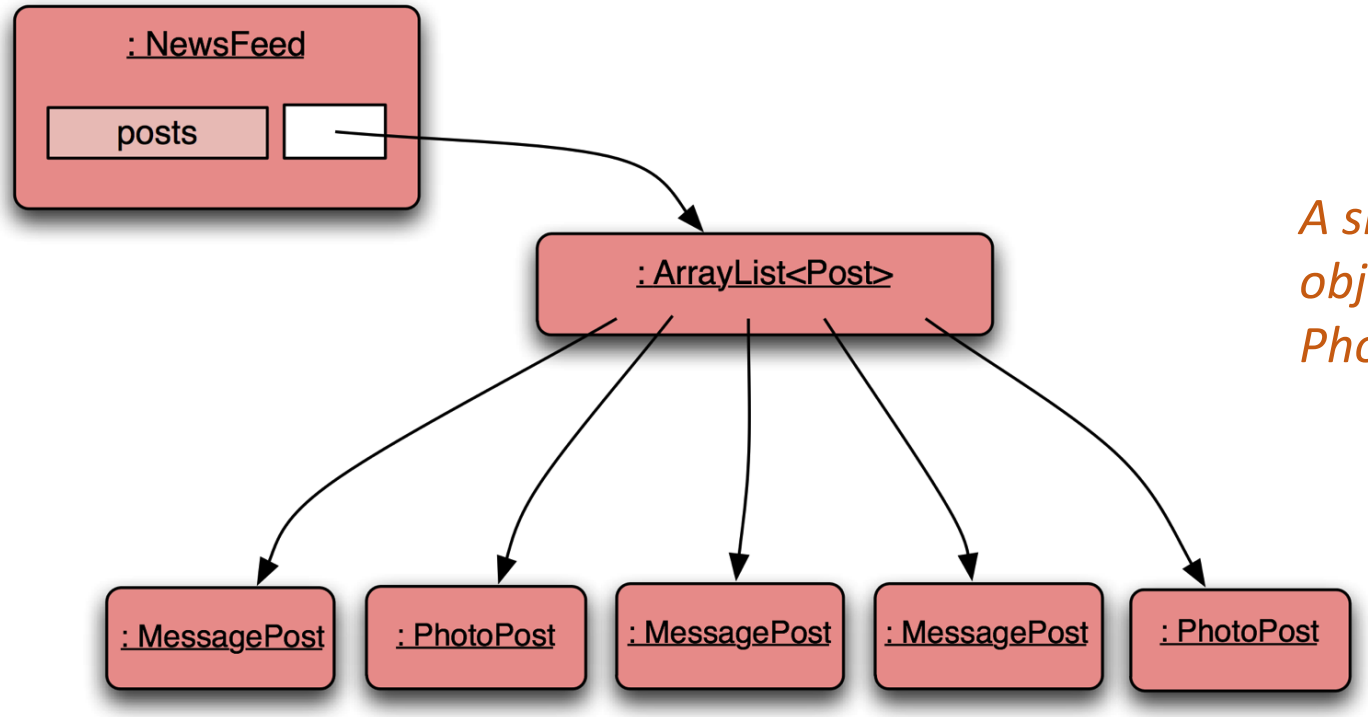
# Polymorphic variables

# Class diagram

*NewItem is now de-coupled from MessagePost and PhotoPost*



# Object diagram




*A single list containing Post objects: MessagePost, PhotoPost, etc.*

# 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

# Casting (1)

- 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!*)

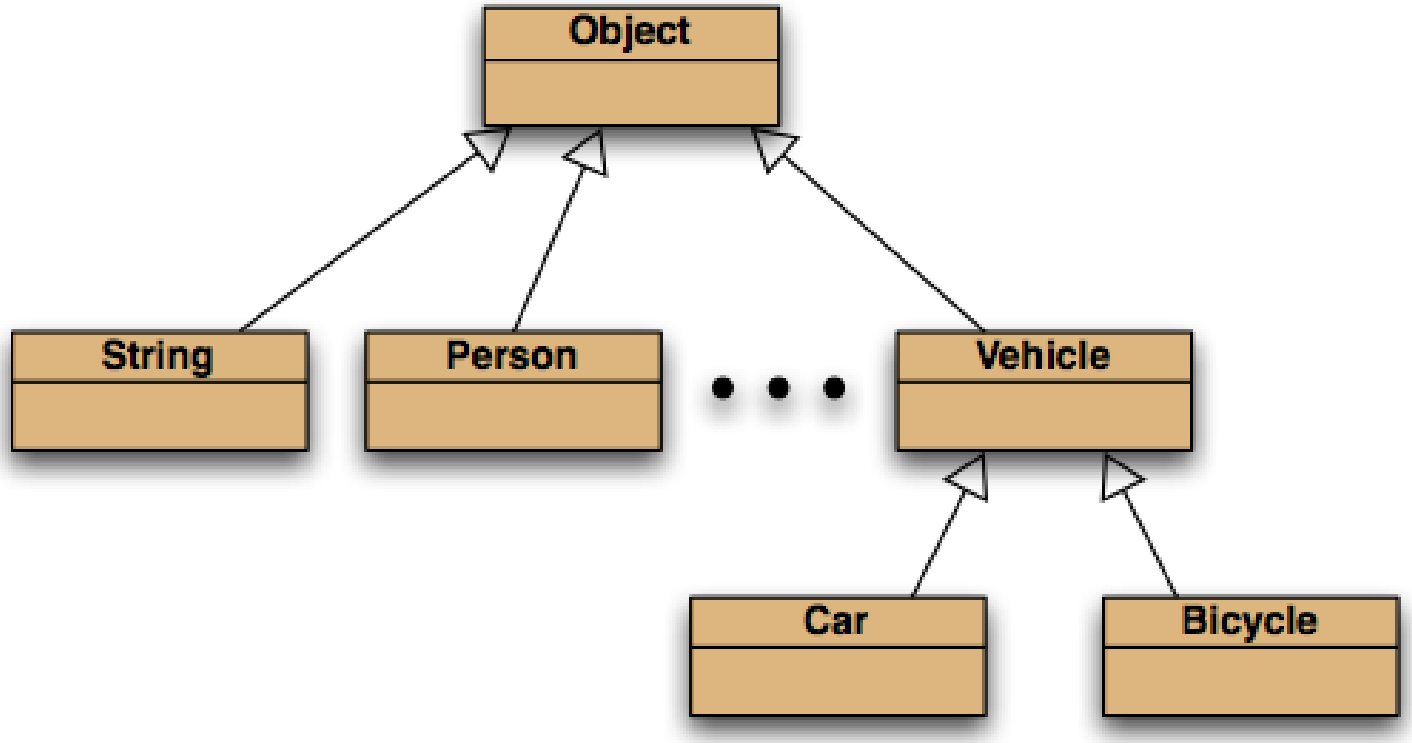
# Casting (2)

- 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
  - It throws ClassCastException if it isn’t!
- Use casting sparingly!

```
Vehicle v = new Bicycle();
Car c = new Car();

c = (Car) v;  // runtime error!
```

# The Object class

- In Java, all classes inherit from the Object class



# Polymorphic collections (1)

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

```
public void add(Object element)
public Object get(int index)
public boolean equals(Object obj)
```

- Usually avoided by using a type parameter with the collection

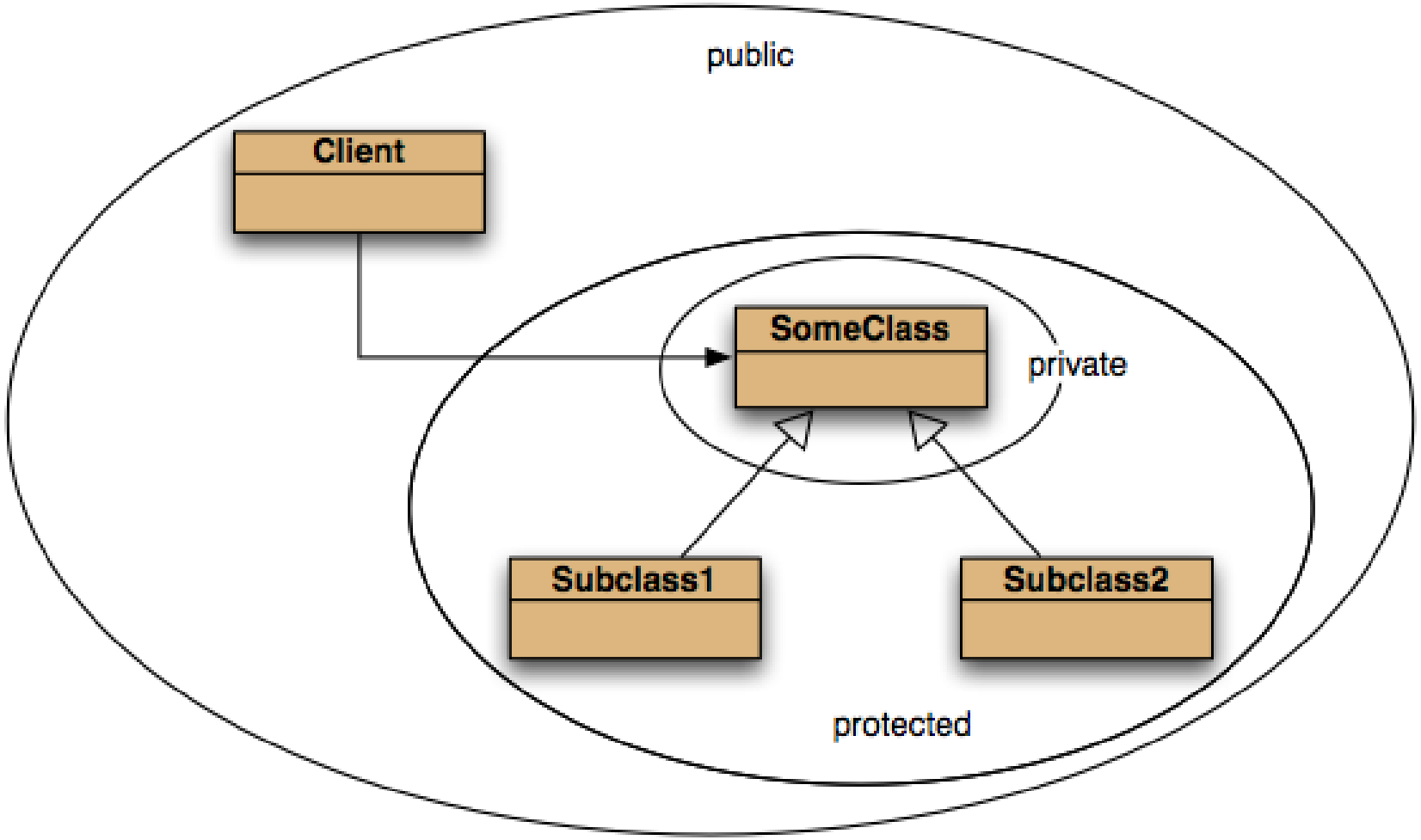
# Polymorphic collections (2)

- A type parameter limits the degree of polymorphism
  - E.g.  ArrayList<Post> posts;
- 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

# Protected access

- Private access in the superclass may be too restrictive for a subclass
- The closer inheritance relationship is supported by protected access
- Protected access is more restricted than public access
- We still recommend keeping fields private
  - Define protected accessors and mutators.

# Access levels



# 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
- Protected access supports inheritance