Inheritance

Exploring Polymorphism

Produced by: Mairead Meagher
Dr. Siobhán Drohan

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

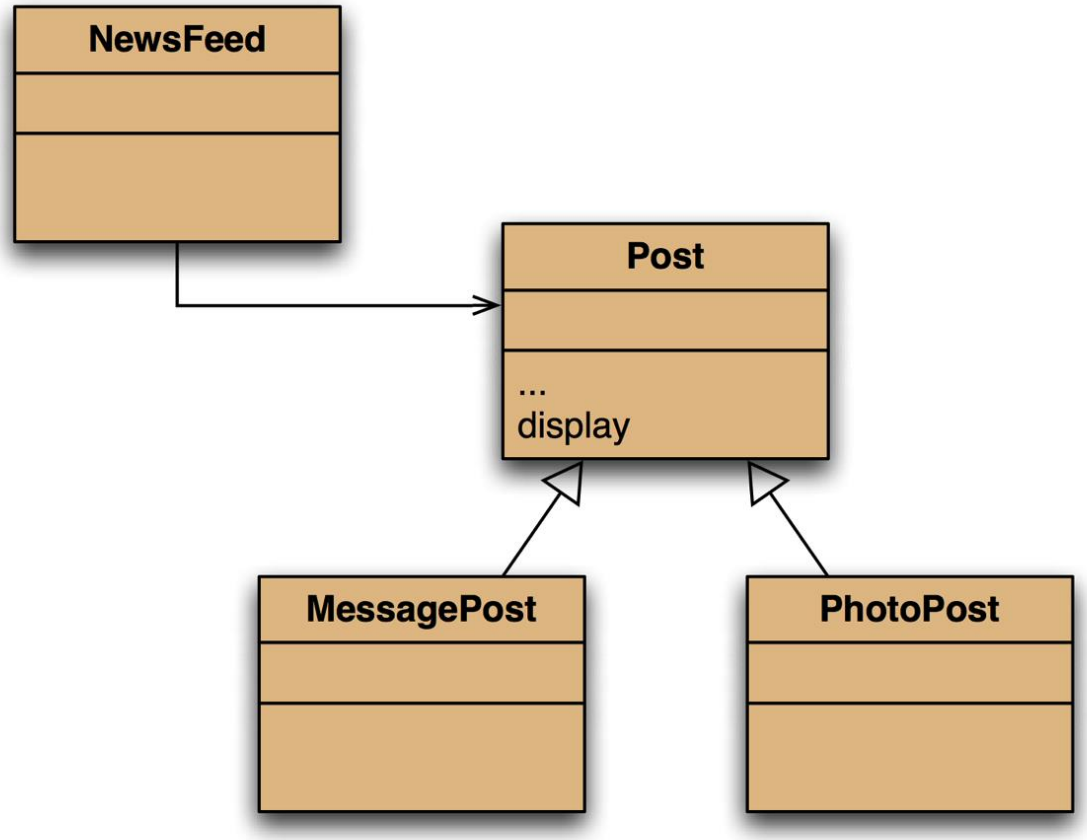
Lectures and Labs

- This weeks lectures and labs are based on examples in:
  - Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Topic List

- Method polymorphism
- Static and dynamic type
- Overriding
- Dynamic method lookup
- Protected access

NetworkV2 – Inheritance Hierarchy



Testing the display method…

Create this MessagePost

```
Leonardo da Vinci
Had a great idea this morning.
But now I forgot what it was. Something to do
with flying ...
40 seconds ago - 2 people like this.
  No comments.
```

Create this PhotoPost

```
Alexander Graham Bell
[experiment.jpg]
I think I might call this thing 'telephone'.
12 minutes ago  -  4 people like this.
  No comments.
```

Testing the display method…

```
Leonardo da Vinci
Had a great idea this morning.
But now I forgot what it was. Something to do with flying ...
40 seconds ago - 2 people like this.
  No comments.

Alexander Graham Bell
[experiment.jpg]
I think I might call this thing 'telephone'.
12 minutes ago  -  4 people like this.
  No comments.
```

What we want

```
Leonardo da Vinci
40 seconds ago  -  2 people like this.
  No comments.

Alexander Graham Bell
12 minutes ago  -  4 people like this.
  No comments.
```

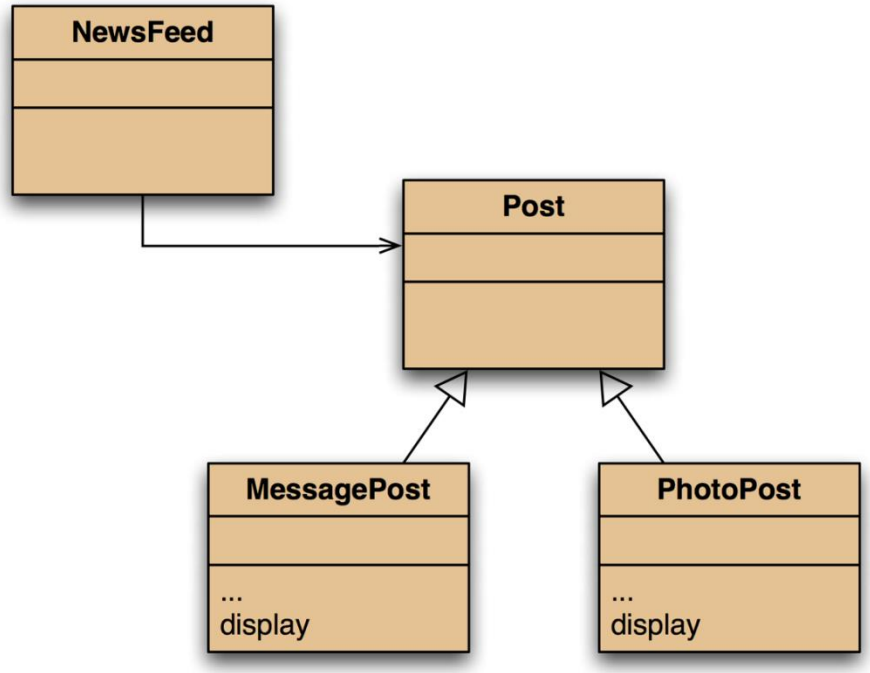
What we have

The problem

- The **`display`** method in **`Post`** only prints the common fields.

- Inheritance is a one-way street:
  - A subclass inherits the superclass fields.
  - The superclass knows nothing about its subclass's fields.

Attempting to solve the problem?



- Place **`display`** where it has access to the information it needs.
- Each subclass has its own version.

But:

- **`Post`**'s fields are private.
- **`NewsFeed`** cannot find a **`display`** method in **`Post`**.

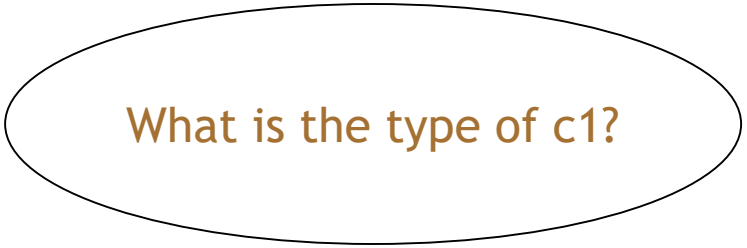
Topic List

- Method polymorphism
- Static and dynamic type
- Overriding
- Dynamic method lookup
- Protected access

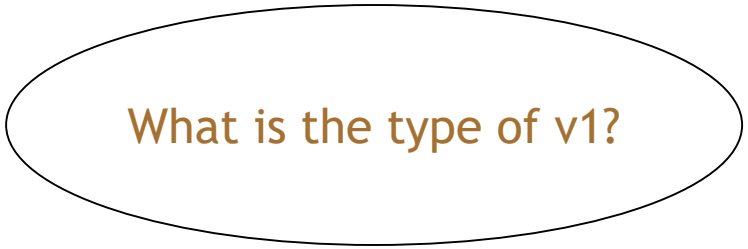
Static type and dynamic type

- A more complex type hierarchy requires further concepts to describe it.
- Some new terminology:
  - static type
  - dynamic type
  - method dispatch/lookup

Static and dynamic type




```
Car c1 = new Car();
```




```
Vehicle v1 = new Car();
```

Static and dynamic type

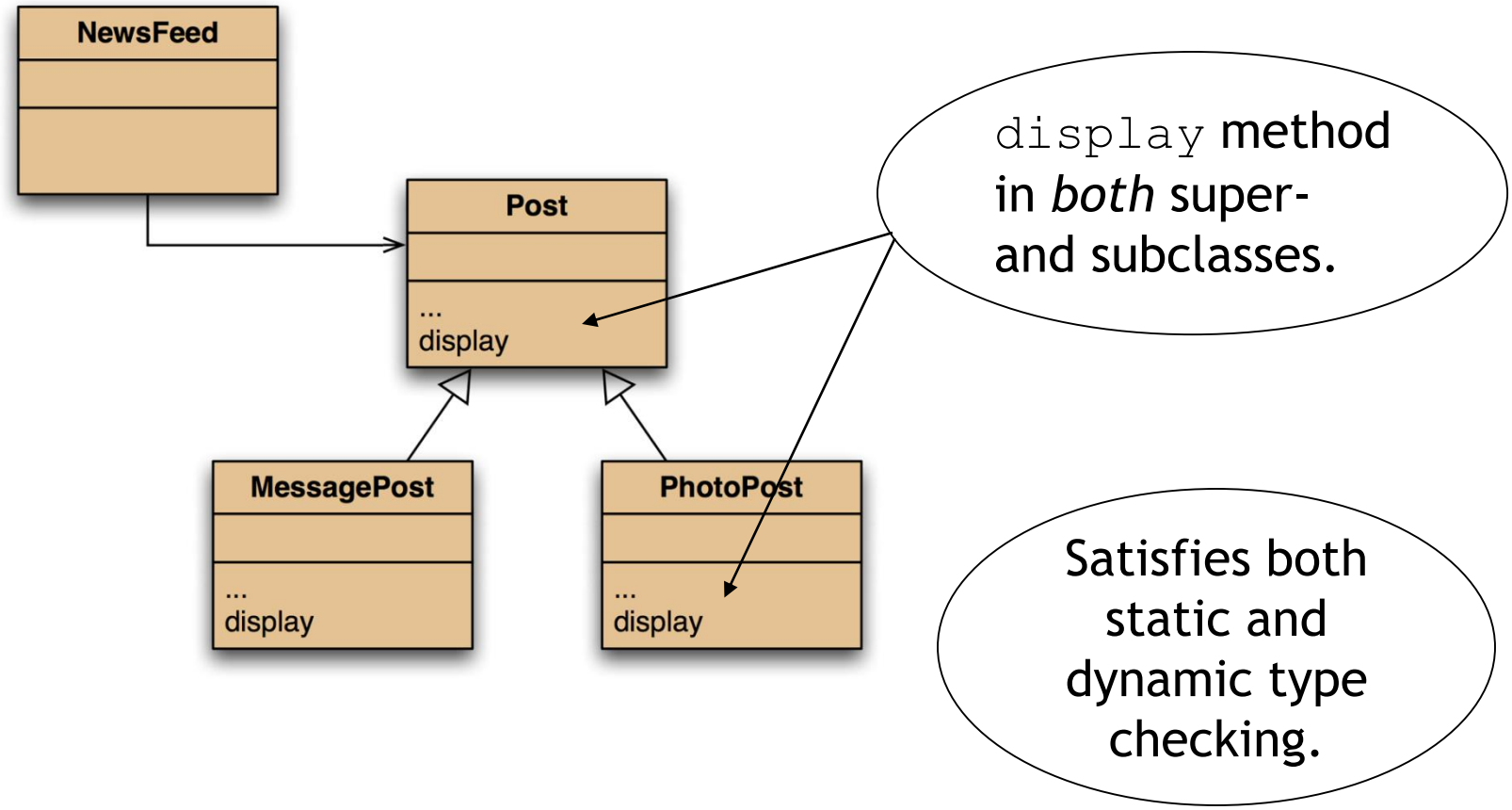
- The declared type of a variable is its *static type*.
- The type of the object a variable refers to is its *dynamic type*.
- The compiler's job is to check for static-type violations.

```
for(Post post : posts) {
    post.display();    // Compile-time error.
                       // display is not in
                  // Post class.
}
```

Topic List

- Method polymorphism
- Static and dynamic type
- Overriding
- Dynamic method lookup
- Protected access

Overriding: the solution



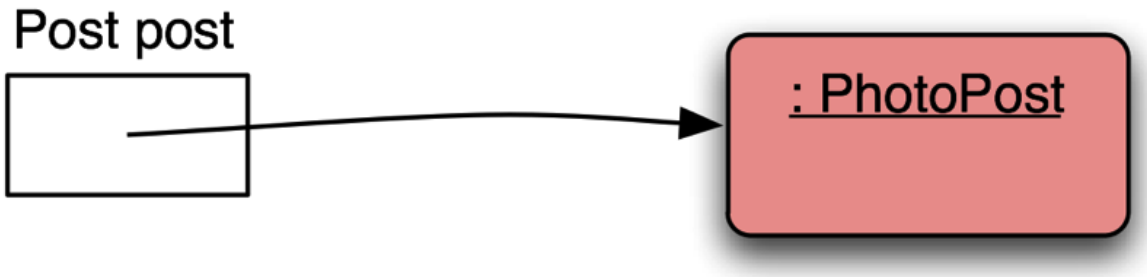
Overriding

- Superclass and subclass define methods with the same signature.
- Each has access to the fields of its class.
- Superclass satisfies static type check.
- Subclass method is called at runtime – it *overrides* the superclass version.
- What becomes of the superclass version?

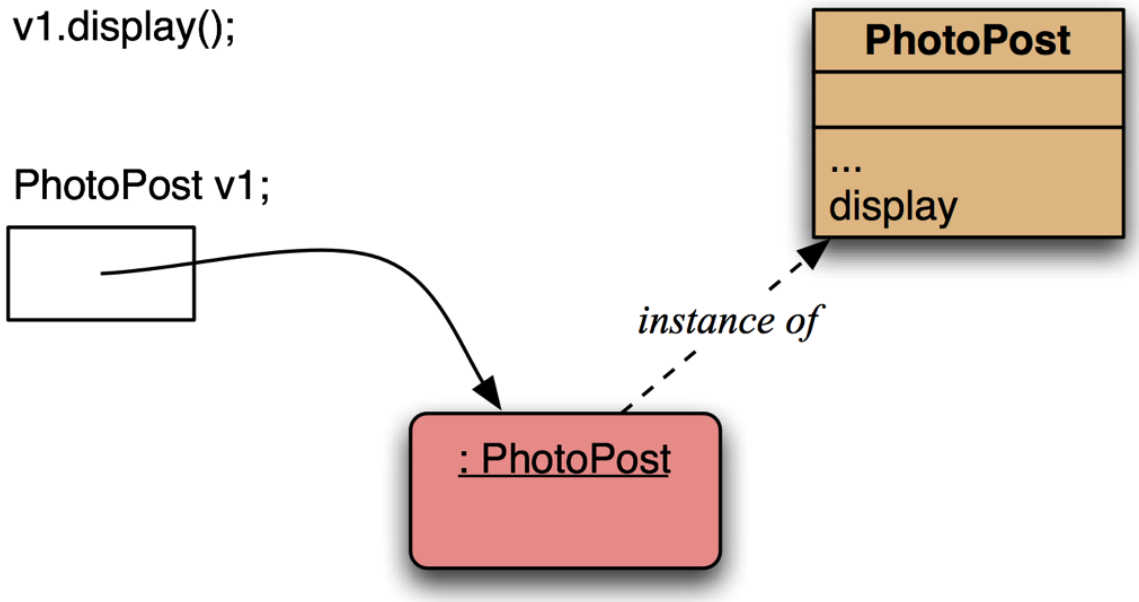
Topic List

- Method polymorphism
- Static and dynamic type
- Overriding
- Dynamic method lookup
- Protected access

Distinct static and dynamic types

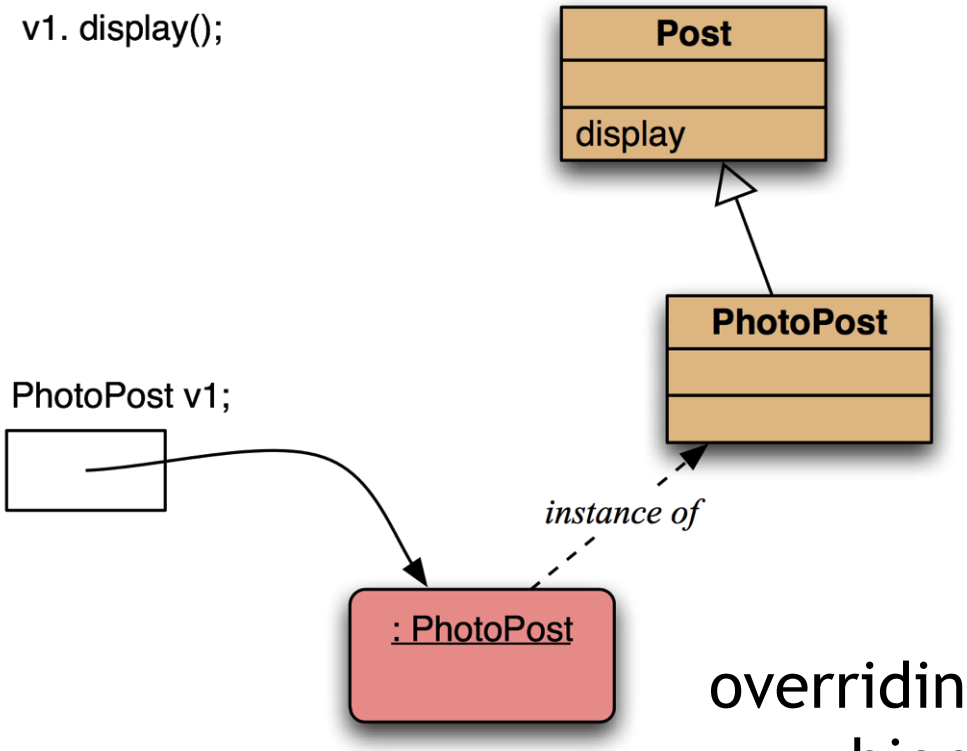


Method lookup



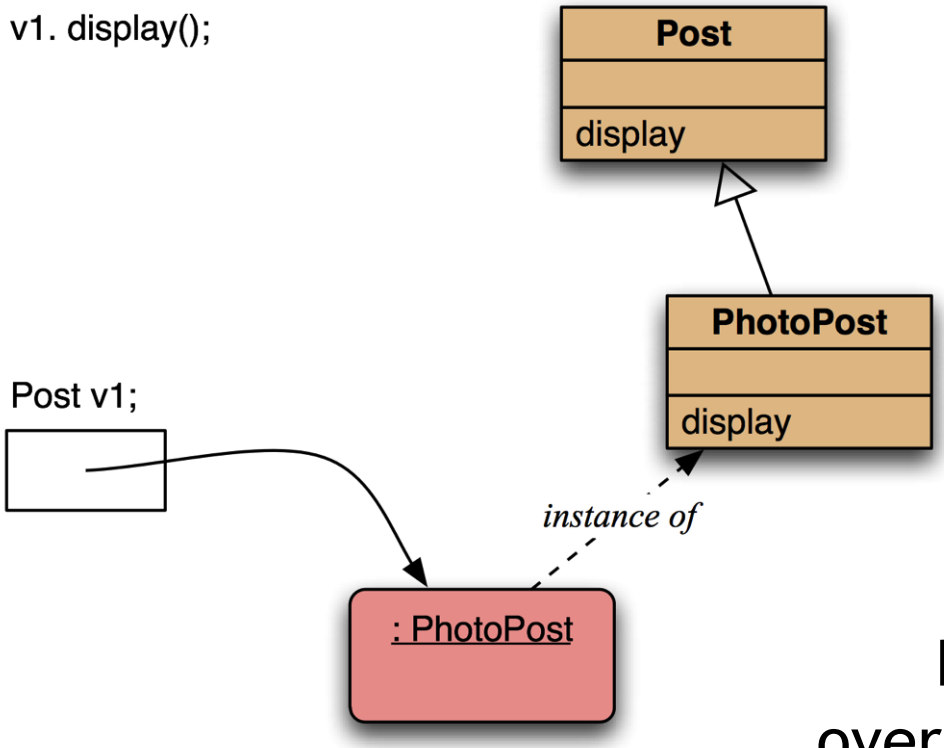
No inheritance or polymorphism.
The obvious method is selected.

Method lookup



Inheritance but no overriding. The inheritance hierarchy is ascended, searching for a match.

Method lookup



Polymorphism and overriding. The ‘first’ version found is used.

Method lookup summary

- The variable is accessed.
- The object stored in the variable is found.
- The class of the object is found.
- The class is searched for a method match.
- If no match is found, the superclass is searched.
- This is repeated until a match is found, or the class hierarchy is exhausted.
- Overriding methods take precedence.

Super call in methods

- Overridden methods are hidden ...
- ... but we often still want to be able to call them.
- An overridden method *can* be called from the method that overrides it.
  - **`super.method(...)`**
  - Compare with the use of **`super`** in constructors.

Calling an overridden method

```
public void display()
{
    super.display();
    System.out.println(" [" +
                       filename +
                       "]");
    System.out.println(" " + caption);
}
```

Method polymorphism

- We have been discussing *polymorphic method dispatch*.
- A polymorphic variable can store objects of varying types.
- Method calls are polymorphic.
  - The actual method called depends on the dynamic object type.

The `instanceof` operator

- Used to determine the dynamic type.
- Recovers ‘lost’ type information.
- Usually precedes assignment with a cast to the dynamic type:

```
if(post instanceof MessagePost) {
    MessagePost msg =
        (MessagePost) post;
    … access MessagePost methods via msg …
}
```

The Object class's methods

- Methods in **`Object`** are inherited by all classes.
- Any of these may be overridden.
- The **`toString`** method is commonly overridden:
  - **`public String toString()`**
  - Returns a string representation of the object.

Overriding `toString` in `Post`

```
public String toString()
{
    String text = username + "\n" +
                  timeString(timestamp);
    if(likes > 0) {
        text += " - " + likes + " people like this.\n";
    }
    else {
        text += "\n";
    }
    if(comments.isEmpty()) {
        return text + " No comments.\n";
    }
    else {
        return text + " " + comments.size() +
               " comment(s). Click here to view.\n";
    }
}
```

Overriding toString

- Explicit print methods can often be omitted from a class:

```
System.out.println(post.toString());
```

- Calls to **`println`** with just an object automatically result in **`toString`** being called:

```
System.out.println(post);
```

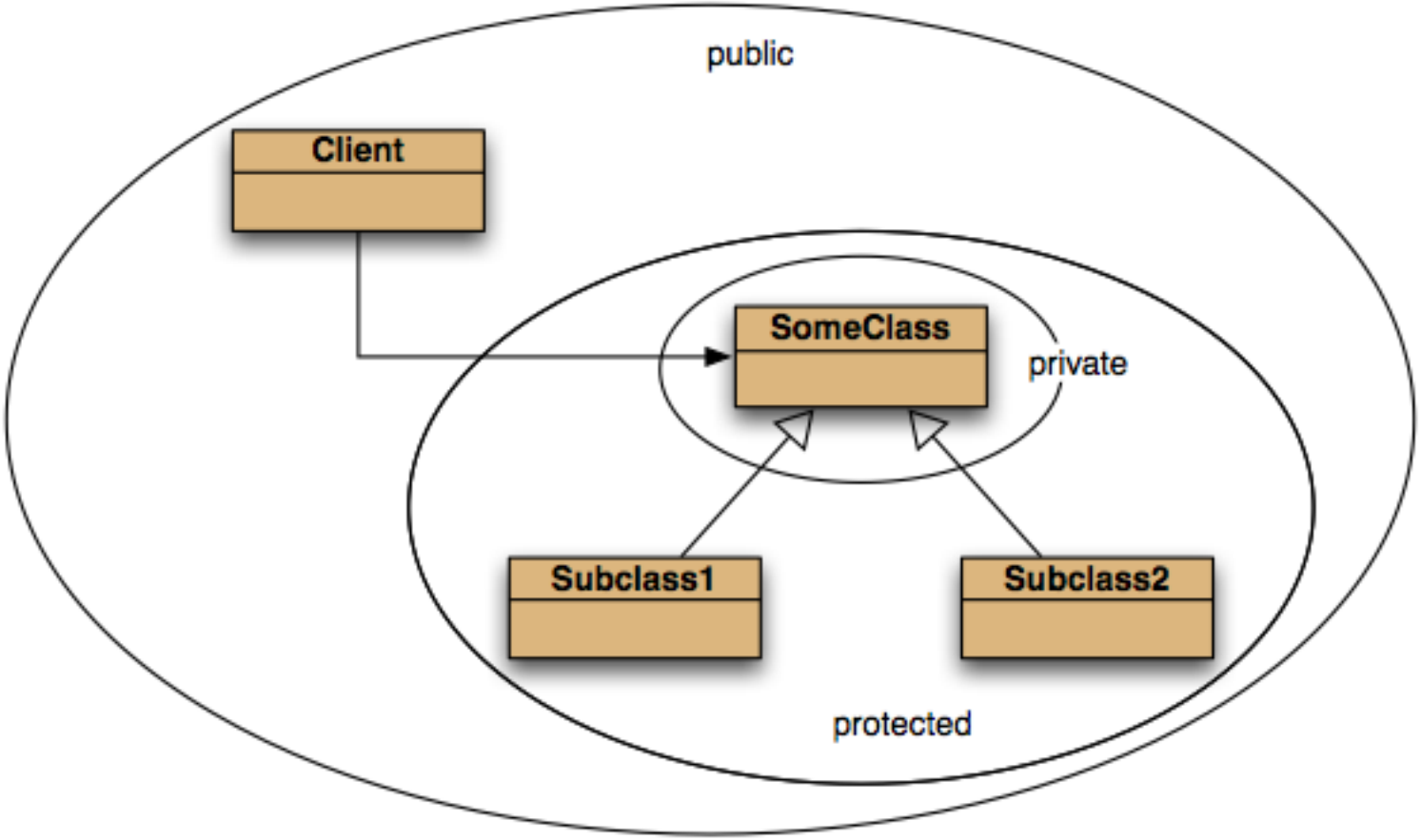
Topic List

- Method polymorphism
- Static and dynamic type
- Overriding
- Dynamic method lookup
- Protected access

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

- The declared type of a variable is its static type.
  - Compilers check static types.
- The type of an object is its dynamic type.
  - Dynamic types are used at runtime.
- Methods may be overridden in a subclass.
- Method lookup starts with the dynamic type.
- Protected access supports inheritance.

Any Questions?