

# Inheritance

## Exploring Polymorphism

---

Produced Dr. Siobhán Drohan  
by: Mr. Colm Dunphy  
Mr. Diarmuid O'Connor  
Dr. Frank Walsh



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 (<https://www.bluej.org/objects-first/>)

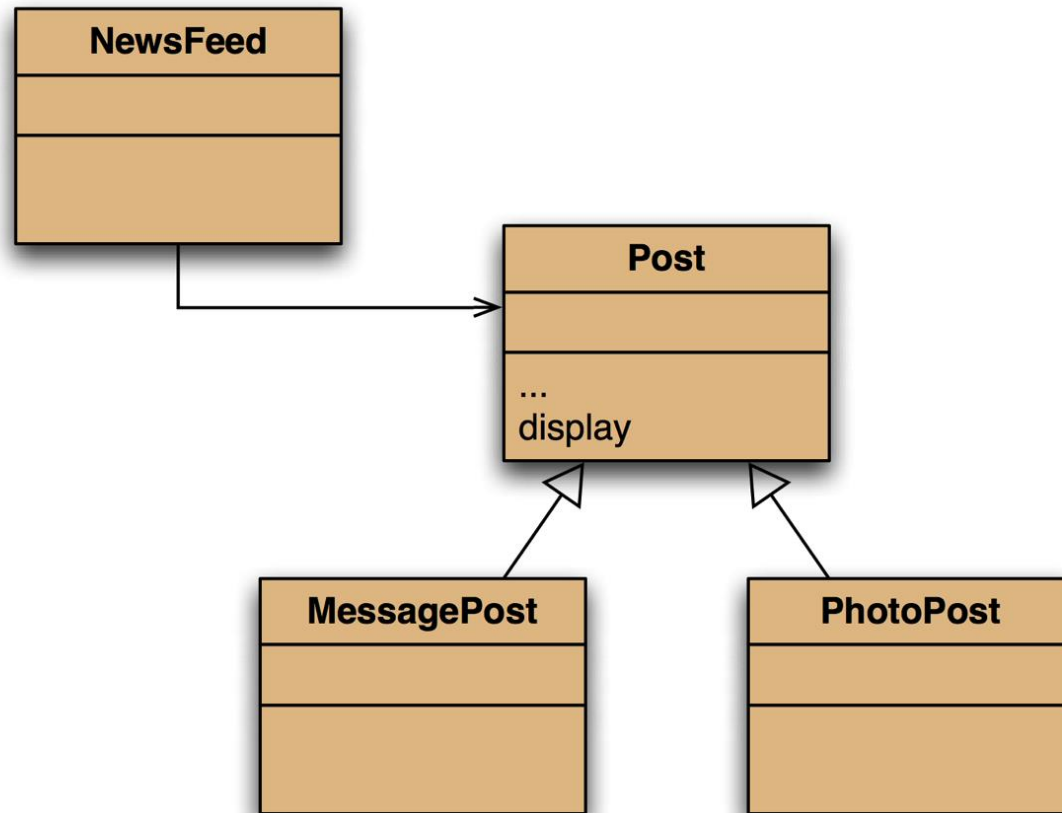
# Topic List

---

- 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

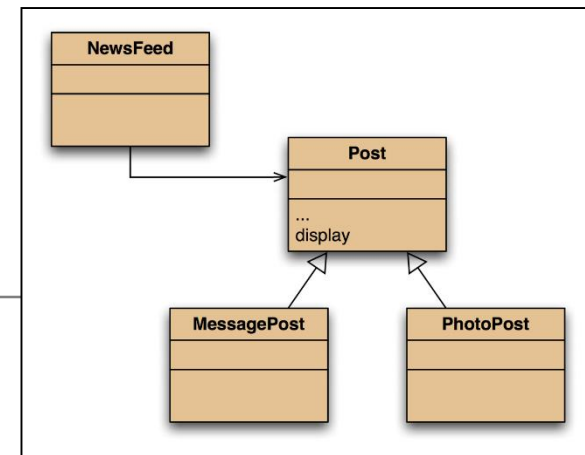
12 minutes ago - 4 people like this.

No comments.

What we have

# The problem

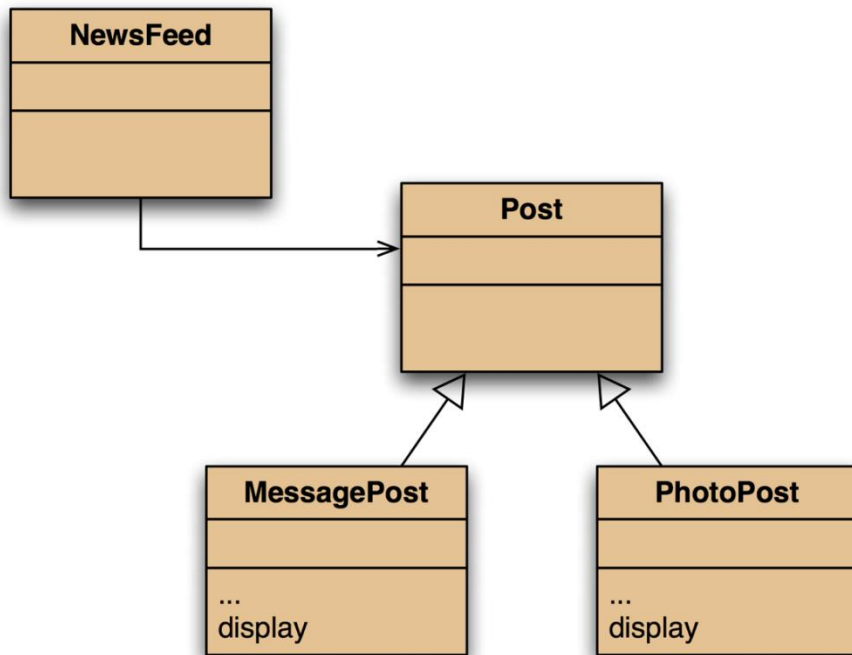
---



- 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

---

What is the type of c1?

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

What is the type of v1?

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

# Static and dynamic type

---

What is the type of c1?

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

What is the type of v1?

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

The declared  
type of a variable  
is its *static* type.

The type of the  
object a variable  
refers to is its  
*dynamic* type.

# Static and dynamic type

---

*The compiler's job is to check for static-type violations.*

What is the type of v1?

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

The declared type of a variable is its *static* type.

The type of the object a variable refers to is its *dynamic* type.

# Recall our attempt to solve this problem...

---

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.

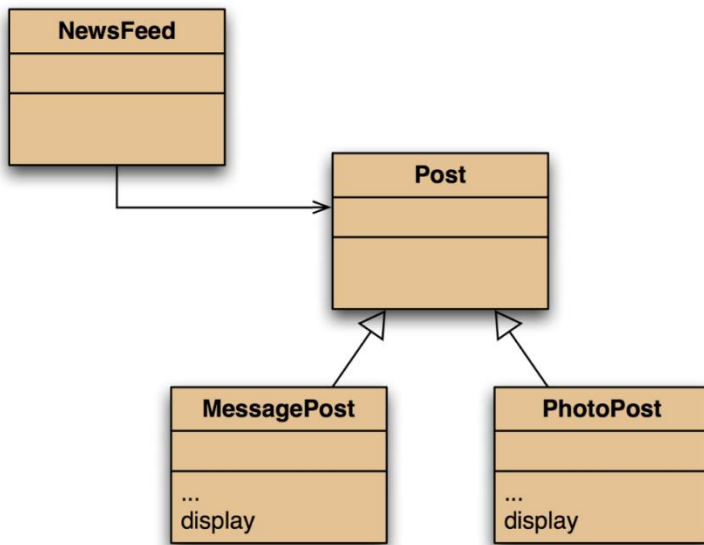
Alexander Graham Bell

12 minutes ago - 4 people like this.

No comments.

What we have

# Recall our attempt to solve this problem...



- We placed **display** in each subclass where it has access to the information it needs.

## But:

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

```
for(Post post : posts) {
    post.display(); // Compile-time error (static-type violation)
                   // method display() is not found in the
                   // Post class
}
```

# Topic List

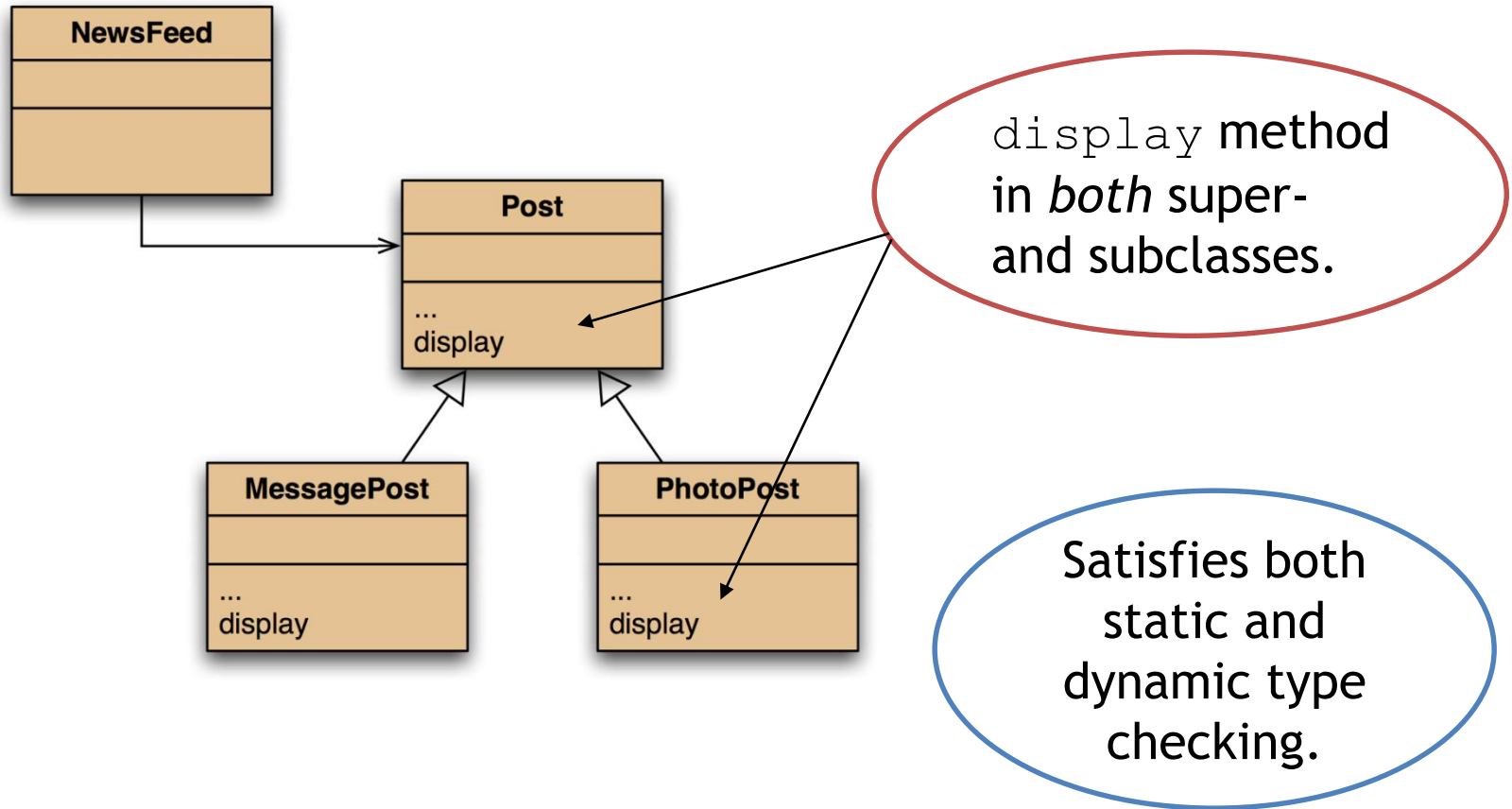
---

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



# Overriding - the solution to our problem

---



# 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?

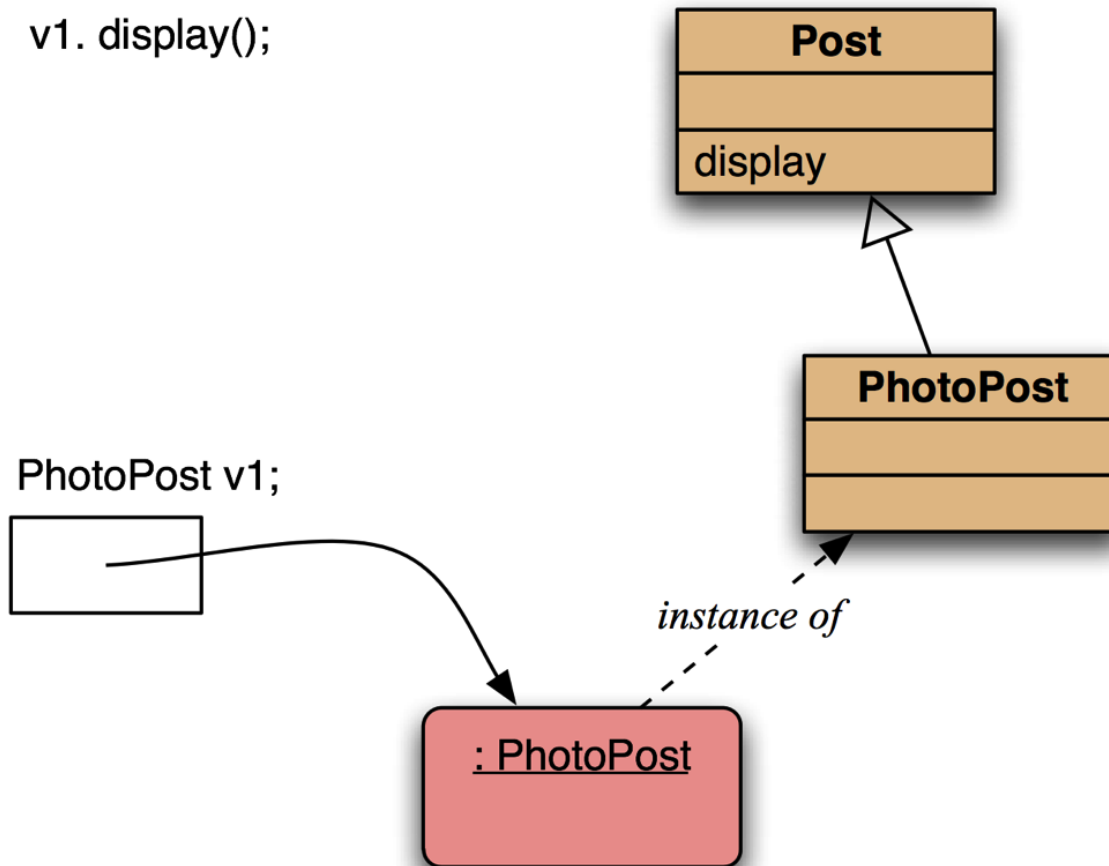
# Topic List

---

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

# Dynamic method lookup

v1.display();



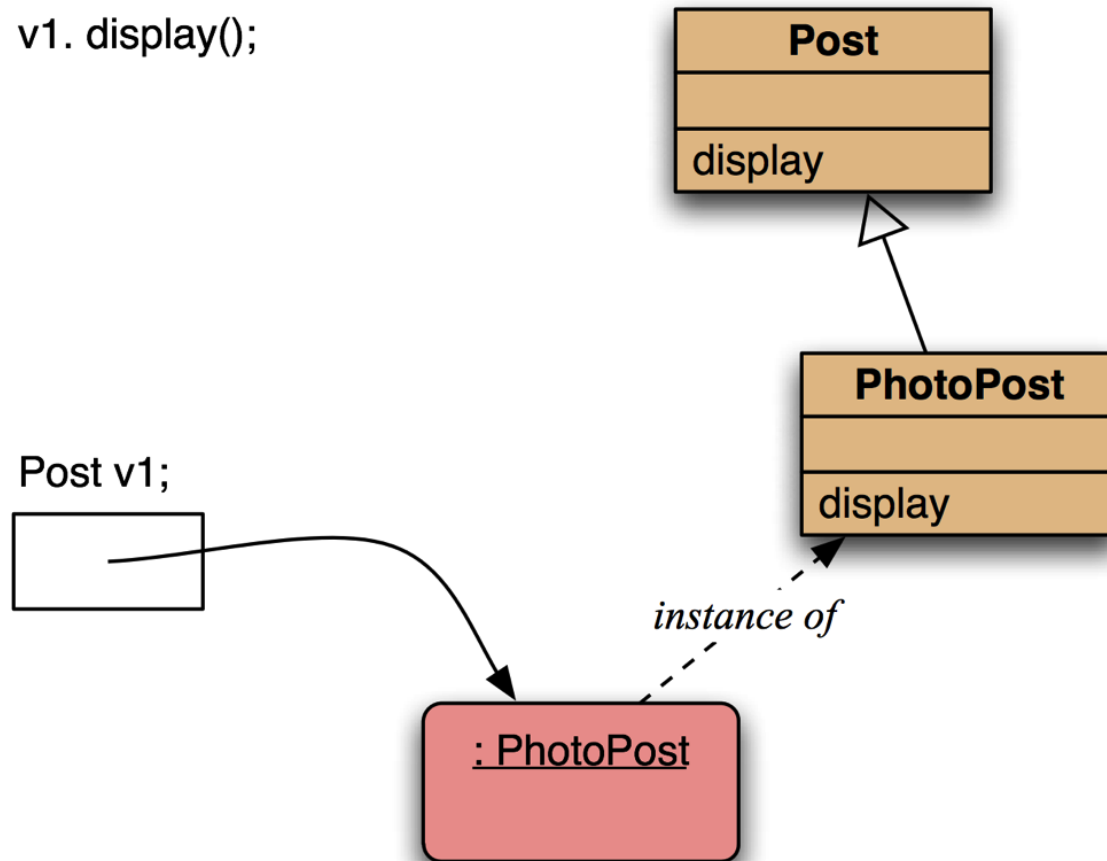
Inheritance but  
no overriding.

The inheritance  
hierarchy is  
ascended,  
searching for a  
match.

# Dynamic method lookup

---

v1. display();



Polymorphism  
and overriding.

The 'first'  
version found is  
used.

# Dynamic method lookup summary

---

1. The variable is accessed.
2. The object stored in the variable is found.
3. The class of the object is found.
4. The class is searched for a method match.
5. If no match is found, the superclass is searched.
6. This is repeated until a match is found, or the class hierarchy is exhausted.
7. 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( . . . )`**
  - Recall we used **`super`** in our 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.
- Can recover 'lost' type information.
- Usually precedes assignment with a cast to the dynamic type:
- ```
if(post instanceof MessagePost) {  
    MessagePost msg =  
        (MessagePost) post;  
    ... e.g. then access MessagePost methods via msg ...  
}
```

# Recall the Object class...

---

java.lang

## **Class Object**

java.lang.Object

---

```
public class Object
```

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

**Since:**

JDK1.0

# Recall the Object class...

*All classes inherit from **Object**.*

java.lang

## Class Object

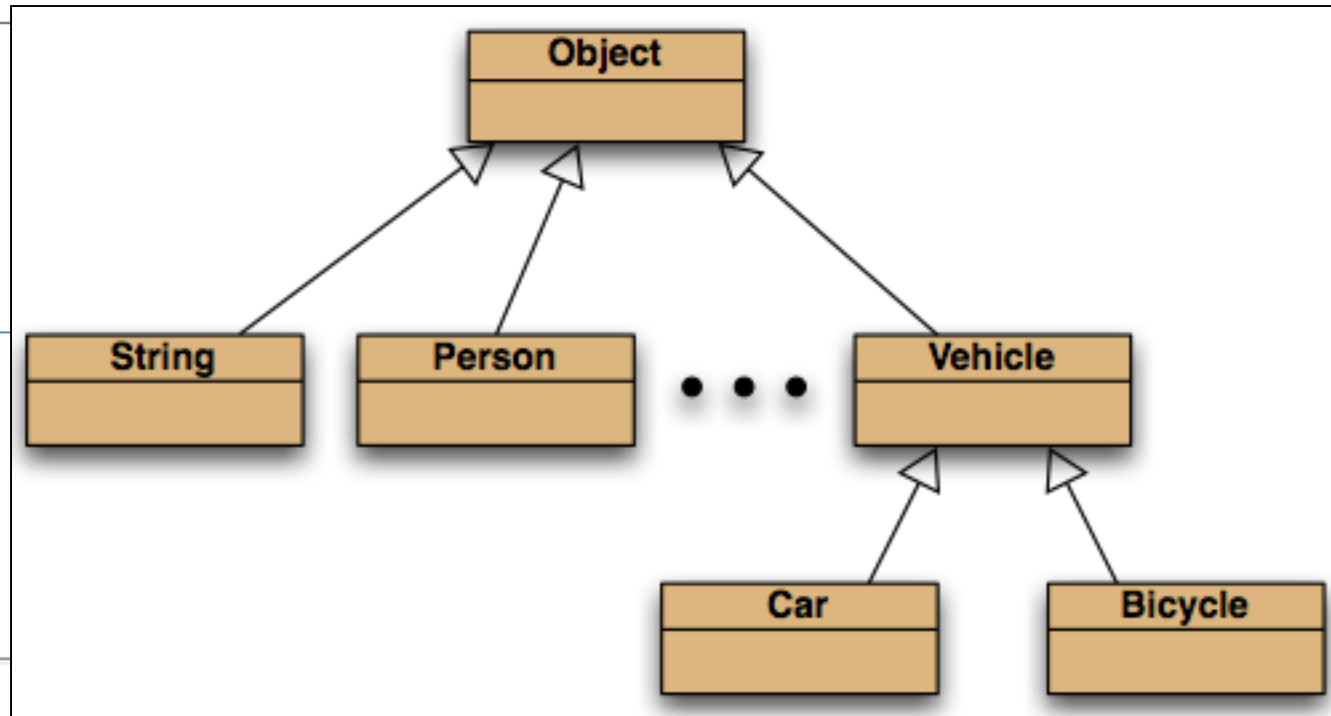
java.lang.Object

```
public class Object
```

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

**Since:**

JDK1.0



Methods in **Object** are inherited by all classes.

Any of these may be overridden.

| Methods                 |                                                                                                                                                                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Modifier and Type       | Method and Description                                                                                                                                                                                                                                                                |
| protected <b>Object</b> | <b>clone()</b><br>Creates and returns a copy of this object.                                                                                                                                                                                                                          |
| boolean                 | <b>equals(Object obj)</b><br>Indicates whether some other object is "equal to" this one.                                                                                                                                                                                              |
| protected void          | <b>finalize()</b><br>Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.                                                                                                                                 |
| <b>Class&lt;?&gt;</b>   | <b>getClass()</b><br>Returns the runtime class of this Object.                                                                                                                                                                                                                        |
| int                     | <b>hashCode()</b><br>Returns a hash code value for the object.                                                                                                                                                                                                                        |
| void                    | <b>notify()</b><br>Wakes up a single thread that is waiting on this object's monitor.                                                                                                                                                                                                 |
| void                    | <b>notifyAll()</b><br>Wakes up all threads that are waiting on this object's monitor.                                                                                                                                                                                                 |
| <b>String</b>           | <b>toString()</b><br>Returns a string representation of the object.                                                                                                                                                                                                                   |
| void                    | <b>wait()</b><br>Causes the current thread to wait until another thread invokes the <b>notify()</b> method or the <b>notifyAll()</b> method for this object.                                                                                                                          |
| void                    | <b>wait(long timeout)</b><br>Causes the current thread to wait until either another thread invokes the <b>notify()</b> method or the <b>notifyAll()</b> method for this object, or a specified amount of time has elapsed.                                                            |
| void                    | <b>wait(long timeout, int nanos)</b><br>Causes the current thread to wait until another thread invokes the <b>notify()</b> method or the <b>notifyAll()</b> method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed. |

| Methods                 |                                                                                                                                                                   |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Modifier and Type       | Method and Description                                                                                                                                            |
| protected <b>Object</b> | <b>clone()</b><br>Creates and returns a copy of this object.                                                                                                      |
| boolean                 | <b>equals(Object obj)</b><br>Indicates whether some other object is "equal to" this one.                                                                          |
| protected void          | <b>finalize()</b><br>Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.             |
| <b>Class&lt;?&gt;</b>   | <b>getClass()</b><br>Returns the runtime class of this Object.                                                                                                    |
| int                     | <b>hashCode()</b><br>Returns a hash code value for the object.                                                                                                    |
| void                    | <b>notify()</b><br>Wakes up a single thread that is waiting on this object's monitor.                                                                             |
| void                    | <b>notifyAll()</b><br>Wakes up all threads that are waiting on this object's monitor.                                                                             |
| <b>String</b>           | <b>toString()</b><br>Returns a string representation of the object.                                                                                               |
| void                    | <b>wait()</b><br>Causes the current thread to wait until another thread invokes the <code>wait()</code> method on the <code>wait()</code> method for this object. |

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

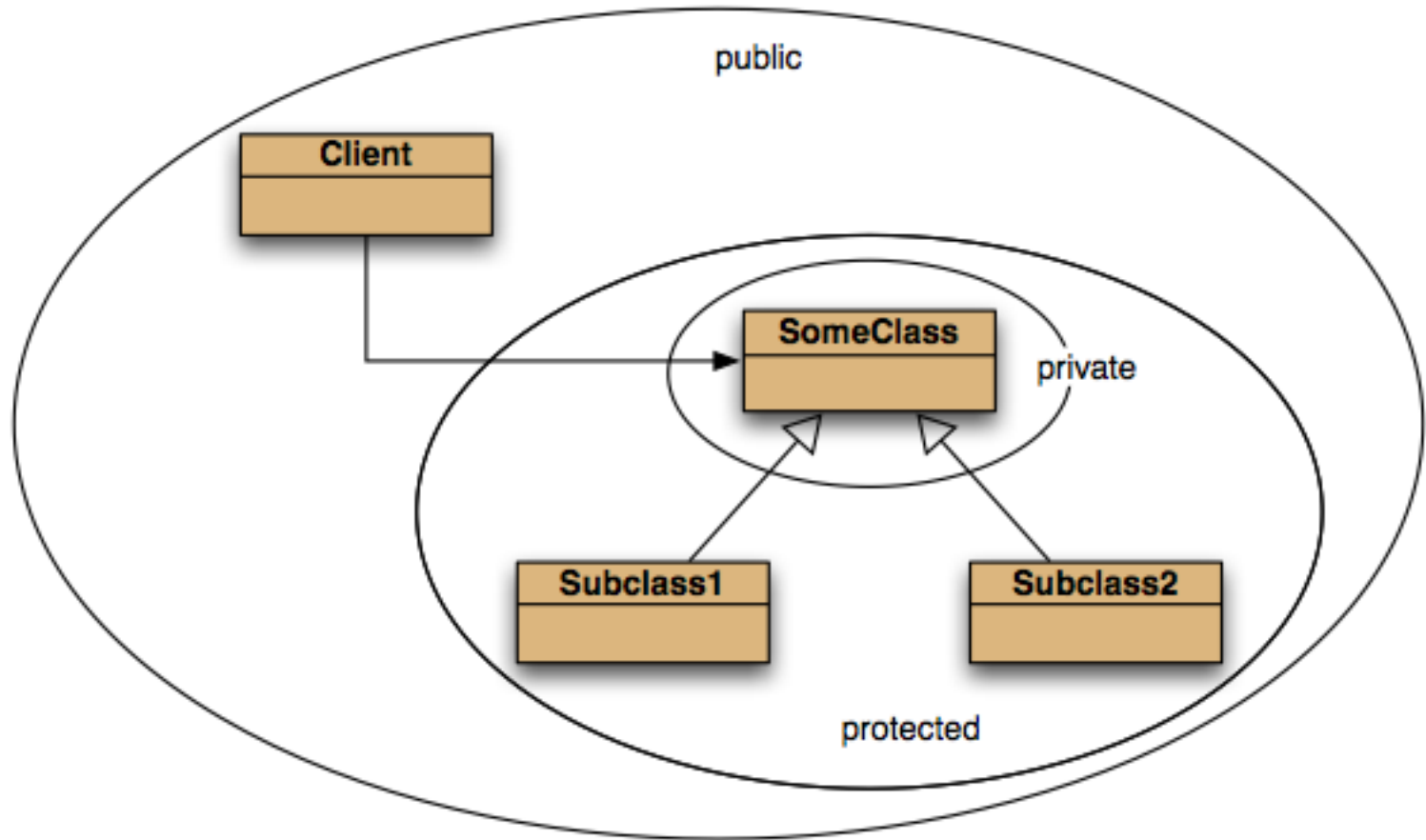
# Protected access

---

- *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.

# 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?**

