Socrative room: F27SB

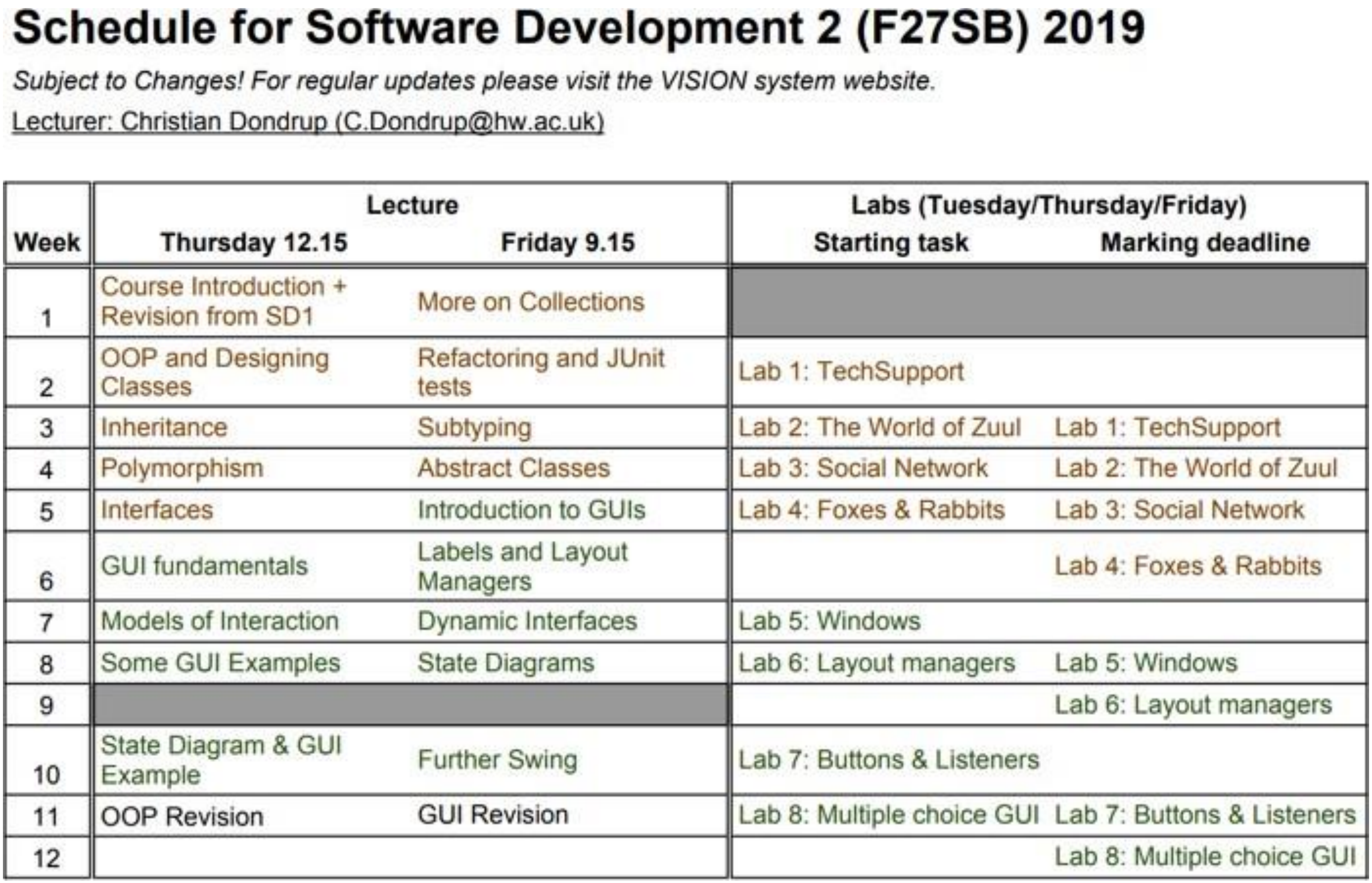
Software Development 2

Subtyping

F27SB

Labs and Marks

- “Submission” max **5 working days** after the deadline
  - Present your code to a lab helper and explain it
- Example Lab1:
  - Released: week 2
  - Due: week 3 (during your assigned lab!)
  - Latest possible: week 4 (70% of mark)



Schedule for Software Development 2 (F27SB) 2019

Subject to Changes! For regular updates please visit the VISION system website.

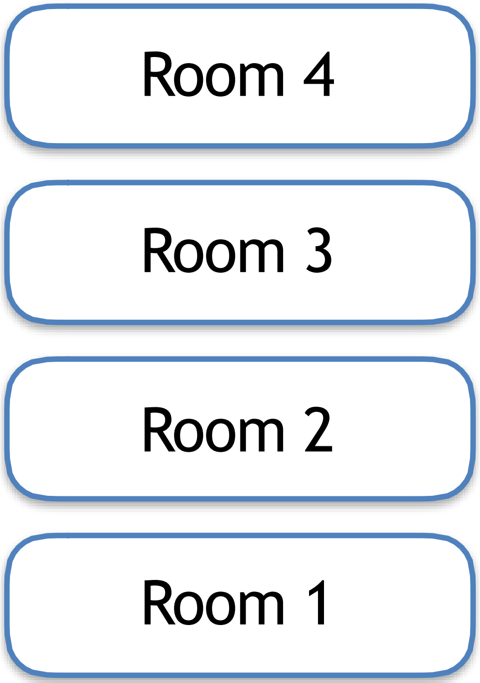
Lecturer: Christian Dondrup (C.Dondrup@hw.ac.uk)

	Lecture		**Labs (Tuesday/Thursday/Friday)**	
Week	**Thursday 12.15**	**Friday 9.15**	**Starting task**	**Marking deadline**
1	Course Introduction + Revision from SD1	More on Collections		
2	OOP and Designing Classes	Refactoring and JUnit tests	Lab 1: TechSupport	
3	Inheritance	Subtyping	Lab 2: The World of Zuul	Lab 1: TechSupport
4	Polymorphism	Abstract Classes	Lab 3: Social Network	Lab 2: The World of Zuul
5	Interfaces	Introduction to GUIs	Lab 4: Foxes & Rabbits	Lab 3: Social Network
6	GUI fundamentals	Labels and Layout Managers		Lab 4: Foxes & Rabbits
7	Models of Interaction	Dynamic Interfaces	Lab 5: Windows	
8	Some GUI Examples	State Diagrams	Lab 6: Layout managers	Lab 5: Windows
9				Lab 6: Layout managers
10	State Diagram & GUI Example	Further Swing	Lab 7: Buttons & Listeners	
11	OOP Revision	GUI Revision	Lab 8: Multiple choice GUI	Lab 7: Buttons & Listeners
12				Lab 8: Multiple choice GUI

Vision -> Course Information

Lab 2 hints

- 2 different interpretations of `goBack`
  1. A list/stack of all rooms that you have been to
  2. Going back and forth from one room to the other

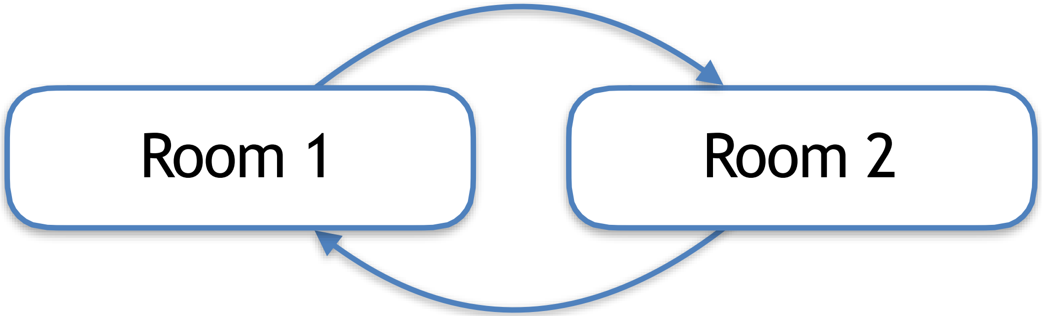


Room 4

Room 3

Room 2

Room 1



Room 1

Room 2

Lab 2 hints

- In case JUnit doesn’t work on your machine: https://www.toddlahman.com/import-org-junit-cannot-resolved-eclipse/
- Getting the name of the player:
  - Hard code it
  - Create a `Scanner` object in the `Player` class and read from input
  - Create a command “name” where the second word is the name of the player

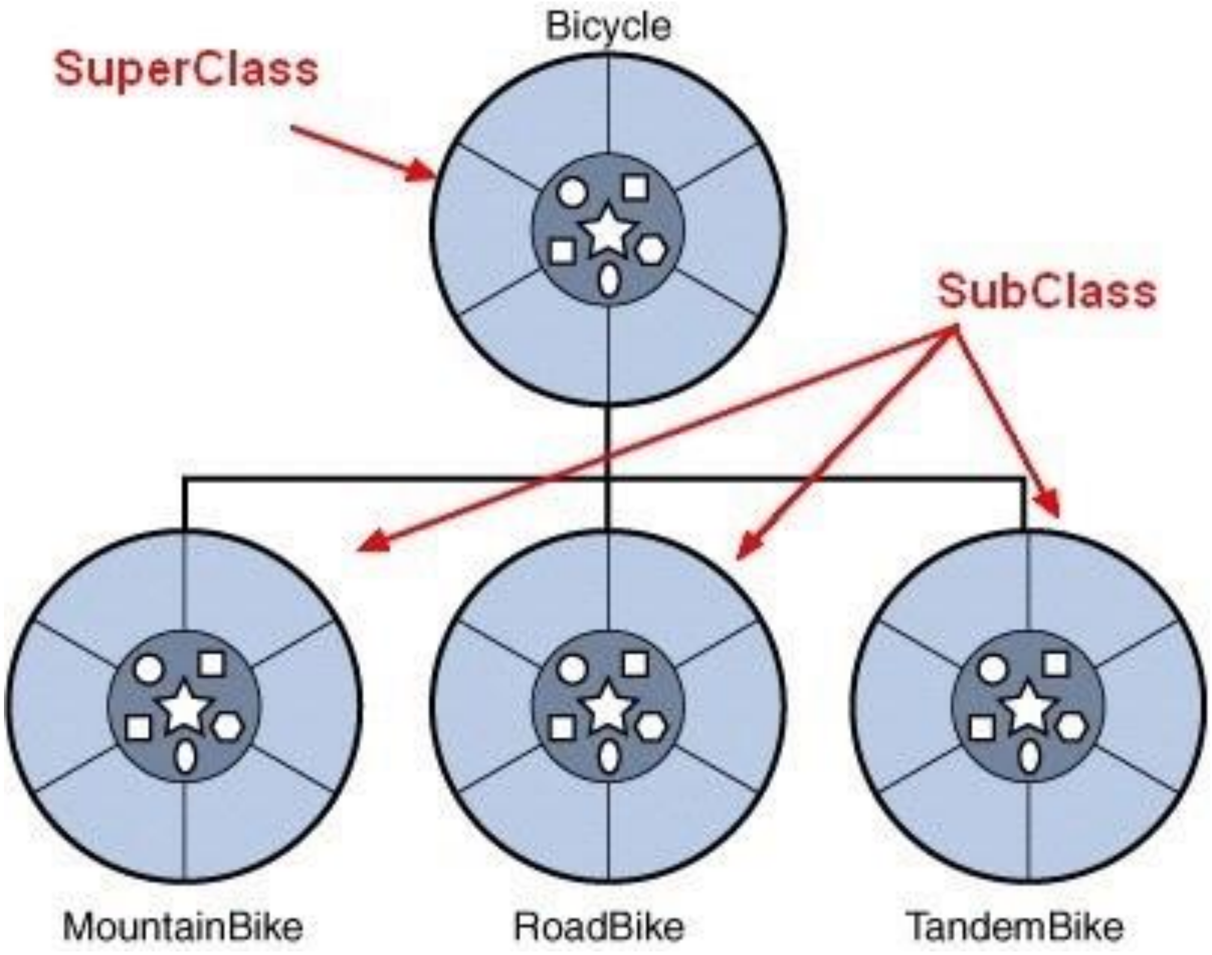
Inheritance

RECAP

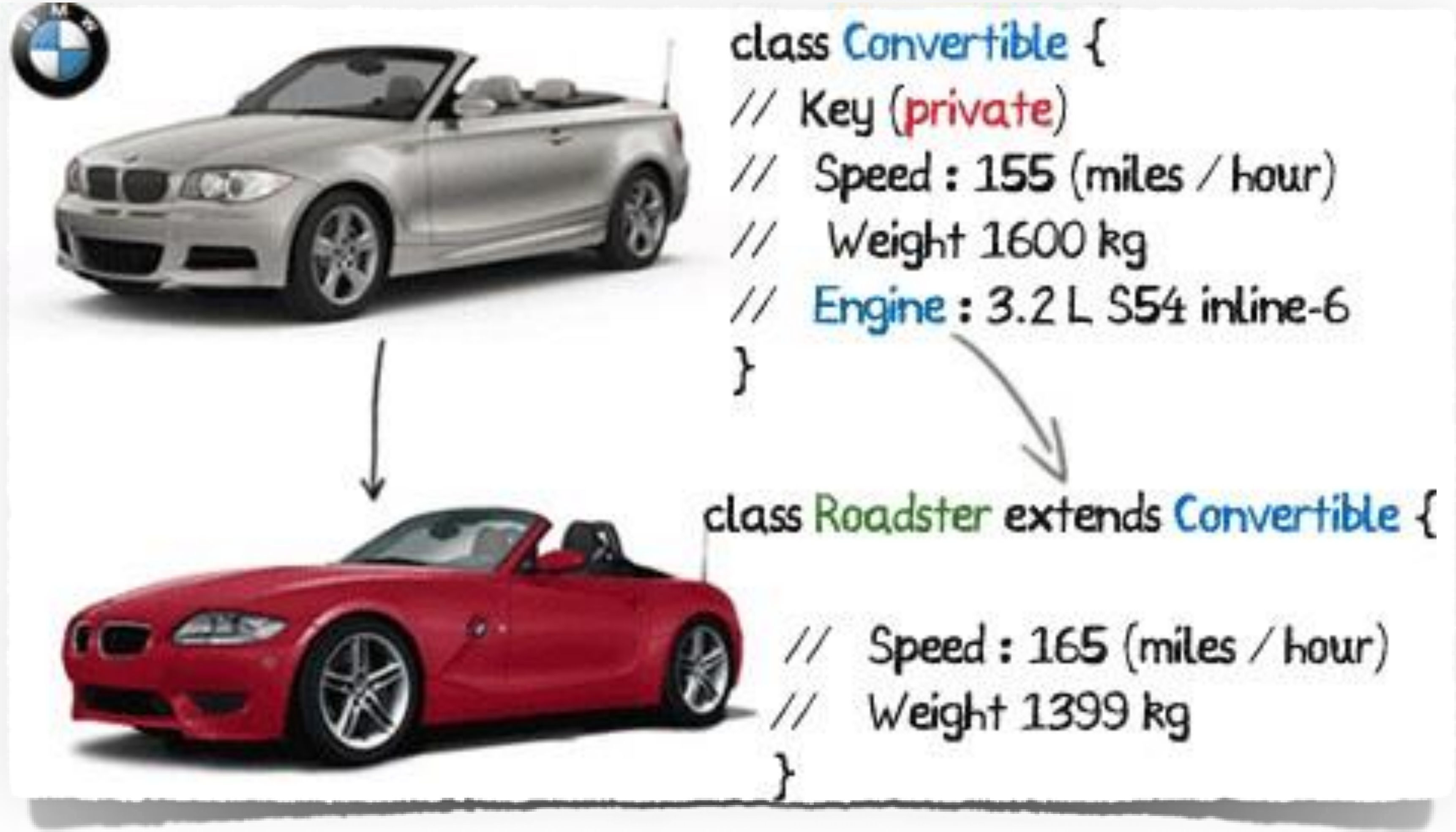
Recap: Inheritance

- Inheritance improves the structure of your code.
- A superclass defines common attributes (and methods).
- The subclass inherits these and adds its own.

Recap: Hierarchy of Classes



Recap: Inheritance in Java



```
class Convertible {
// Key (private)
//  Speed : 155 (miles / hour)
//   Weight 1600 kg
//  Engine : 3.2 L S54 inline-6
}

class Roadster extends Convertible {

//  Speed : 165 (miles / hour)
//  Weight 1399 kg
}
```

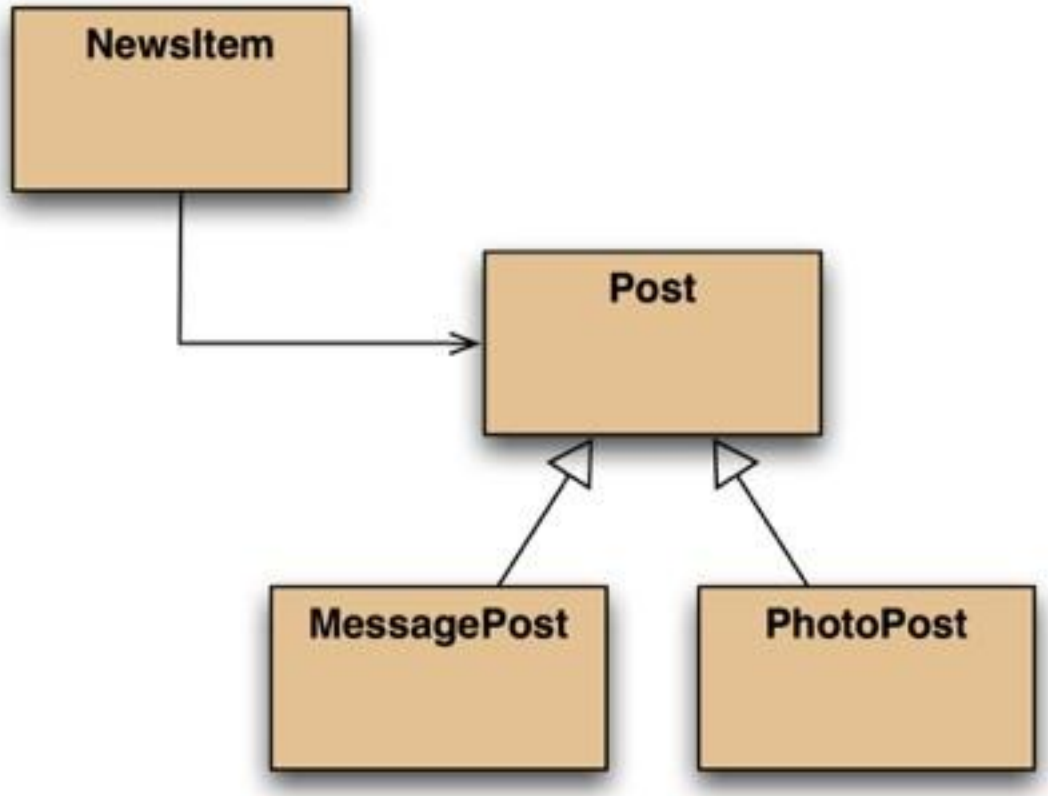
TODAY’S LECTURE

Today’s lecture:

- Subtyping
- Polymorphic* variables
- Upcasting and Downcasting
- Object class
- Polymorphic collections

*Polymorphism = lit.: The occurrence of something in different forms

Class diagram (reminder)



Subtyping

First we had:

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

Now we have:

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

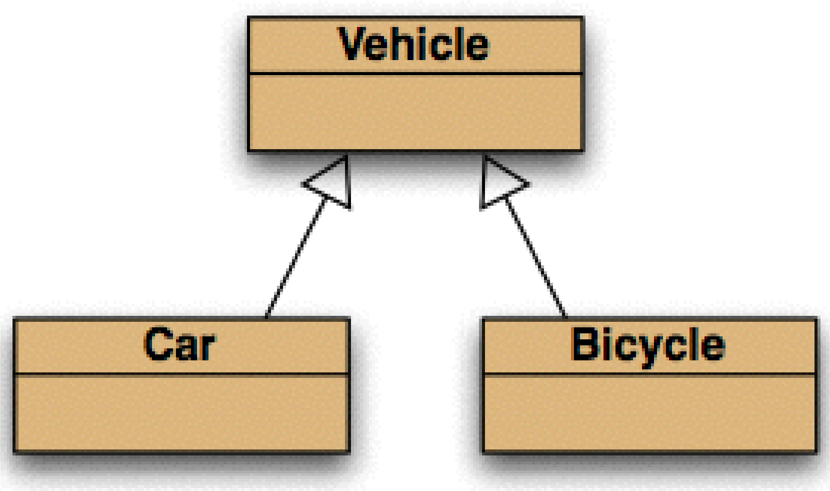
We call this method with:

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

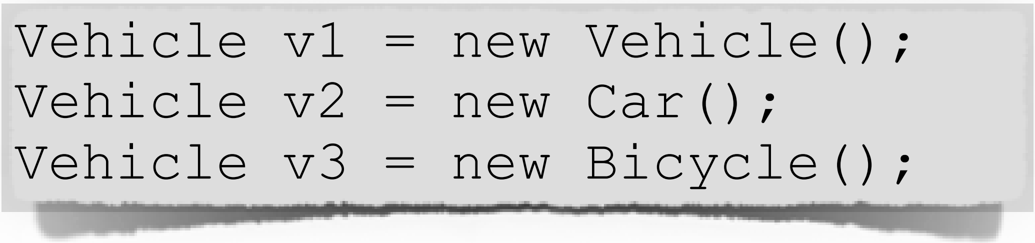
Subclasses and subtyping

- 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



subclass objects may be assigned to superclass variables



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

Subtyping and parameter passing

```
public class NewsFeed
{
    public void addPost(Post post)
    {
        ...
    }
}

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

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

subclass objects may be used as actual parameters for the superclass

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

- 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;
```

(only ok if the vehicle really is a `Car`!)

Casting

- 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:
  - `ClassCastException` if it isn't!
- Use it sparingly.

Automatic upcasting to Bird

```
Duck d = new Duck();
Bird b = d;
```

Automatic upcasting

✓

Socrative room: F27SB

Quiz

Will this run?

(Given that a duck is a subtype of bird.)

```
Bird b;
Duck d = new Duck();
b = d;
d = b;
```

Downcasting error!

Manual downcasting to Duck

Will this run?

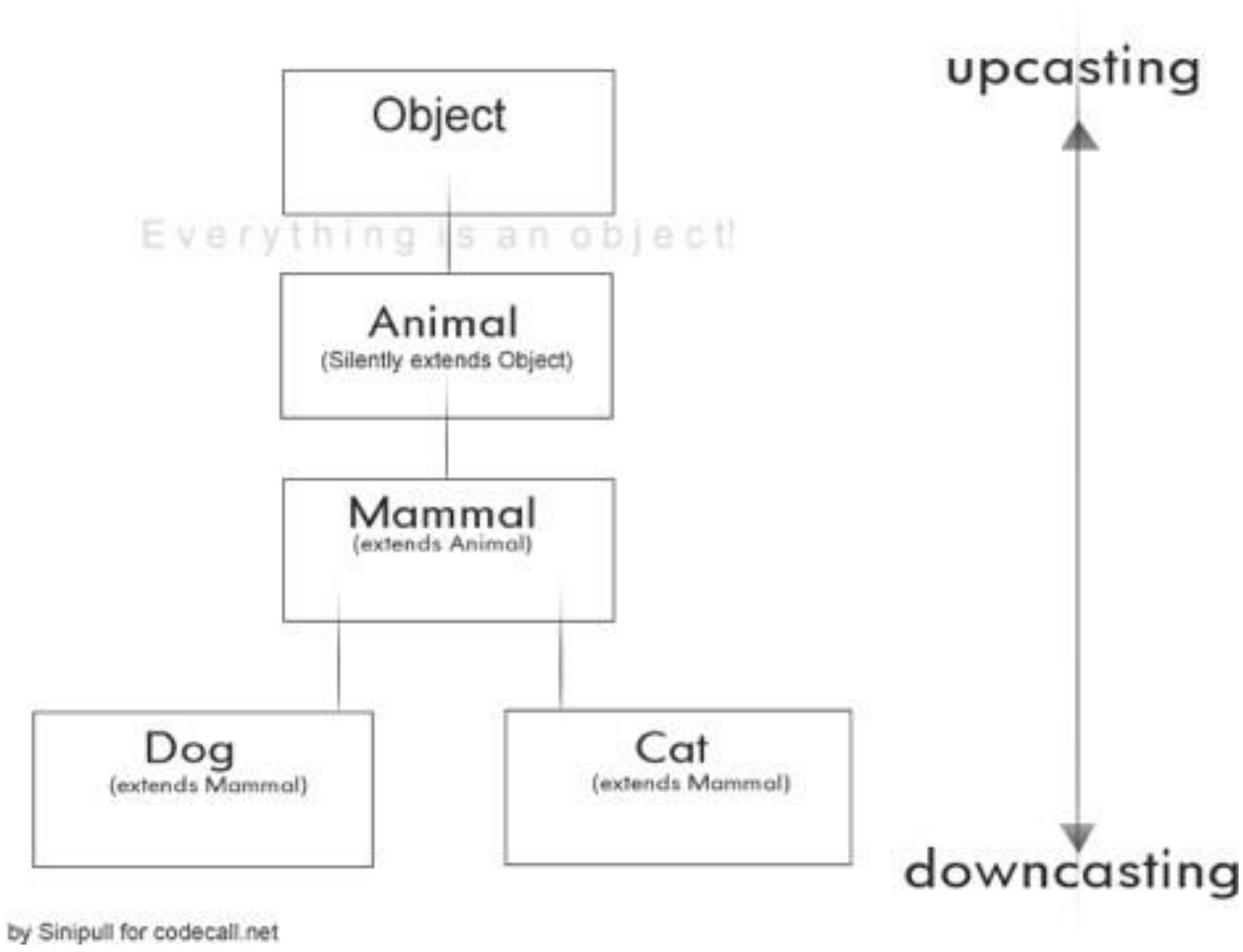
(Given that a duck is a subtype of bird.)

```
Bird b;
Duck d = new Duck();
b = d;
d = (Duck) b;
```

Manual downcasting

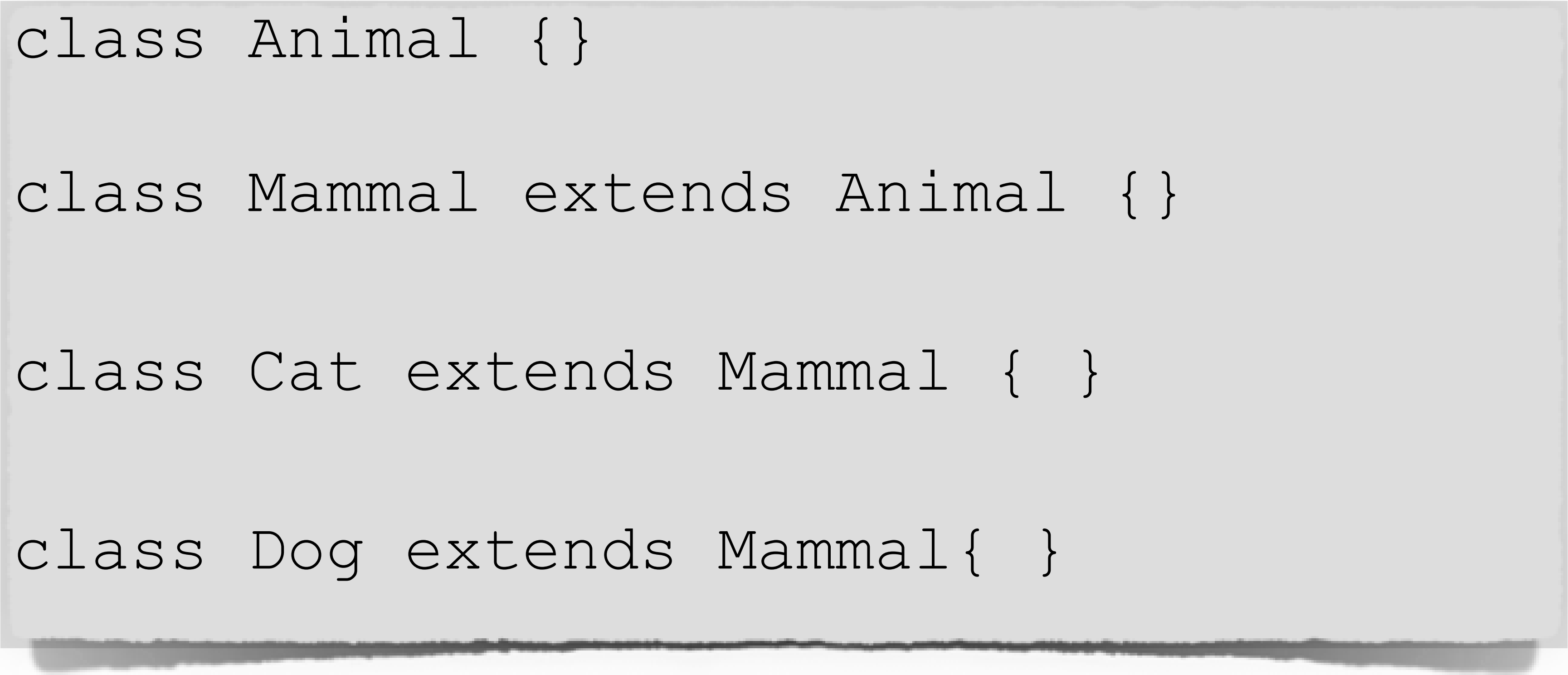
✓

Type Casting



http://forum.codecall.net/java-tutorials/20719-upcasting-downcasting.html

Type Casting Example



```
class Animal {}

class Mammal extends Animal {}

class Cat extends Mammal { }

class Dog extends Mammal{ }
```

Upcasting Example

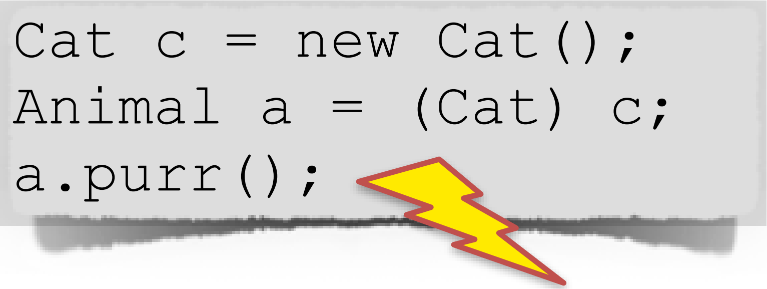
```
Cat c = new Cat();
System.out.println(c);


Animal a = (Animal) c;
System.out.println(a);



    /*
    This prints:
        Cat@a90653
        Cat@a90653
    */
```

`Cat` is still exactly the same `Cat` after upcasting.

It didn't change to a `Animal`, it's just being labeled `Animal` right now.

Possible Upcasting Errors

- If you upcast an object, it will lose all it's properties, which were inherited from below it's current position.



```
Cat c = new Cat();
Animal a = (Cat) c;
a.purr();
```

I can't, because you don't know if i'm a Cat, you must downcast me before i can do it.
Animal
Purr();

by Sinipull for codecall.net

Downcasting Example

```
Cat c1 = new Cat();

//automatic upcasting to Animal
Animal a = c1;

//manual downcasting back to a Cat
Cat c2 = (Cat) a;
```

Casting Error:
Cats cannot be casted as Dogs

I'm still a Cat after upcasting, but compiler treats me as an Object
That means i can't do anything, that's specific for Animals... or Cats.
object
animal
mammal
cat
i can't!
Be a dog!
java.lang.ClassCastException: Cat cannot be cast to Dog

by Sinipull for codecall.net

```
Cat c = new Cat();
Dog g = c;
```

Downcasting Error

```
Mammal m = new Mammal();
Cat c = (Cat)m;
```

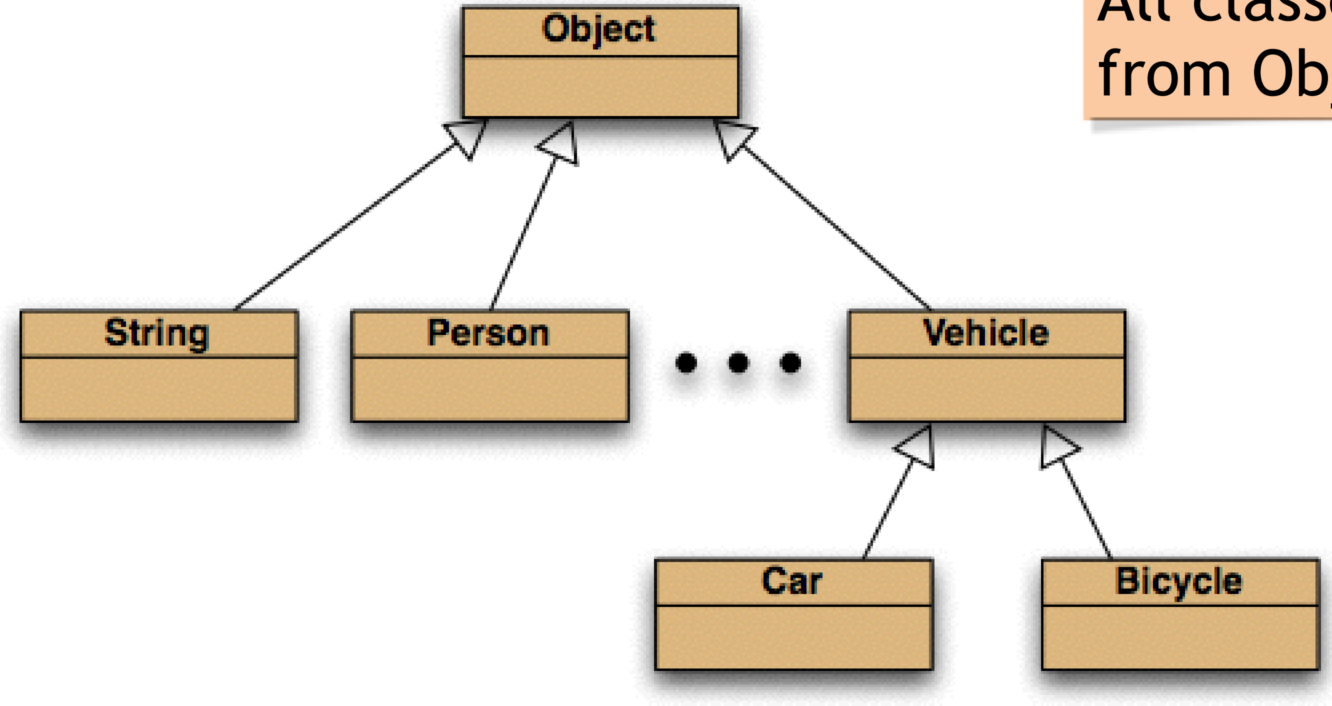
- Such code throws "`java.lang.ClassCastException: Mammal cannot be cast to Cat`" exception during running,
- Because we’re trying to cast a Mammal, which is not a Cat, to a Cat.

Quiz: will this run?

Socrative room: F27SB

```
public class Test(){
   public static void main(String[] args)
   {
      Cat c = new Cat();
      Mammal m = c;
      Dog d = (Dog) m;
   }
}
```

The Object class: The mother of all classes



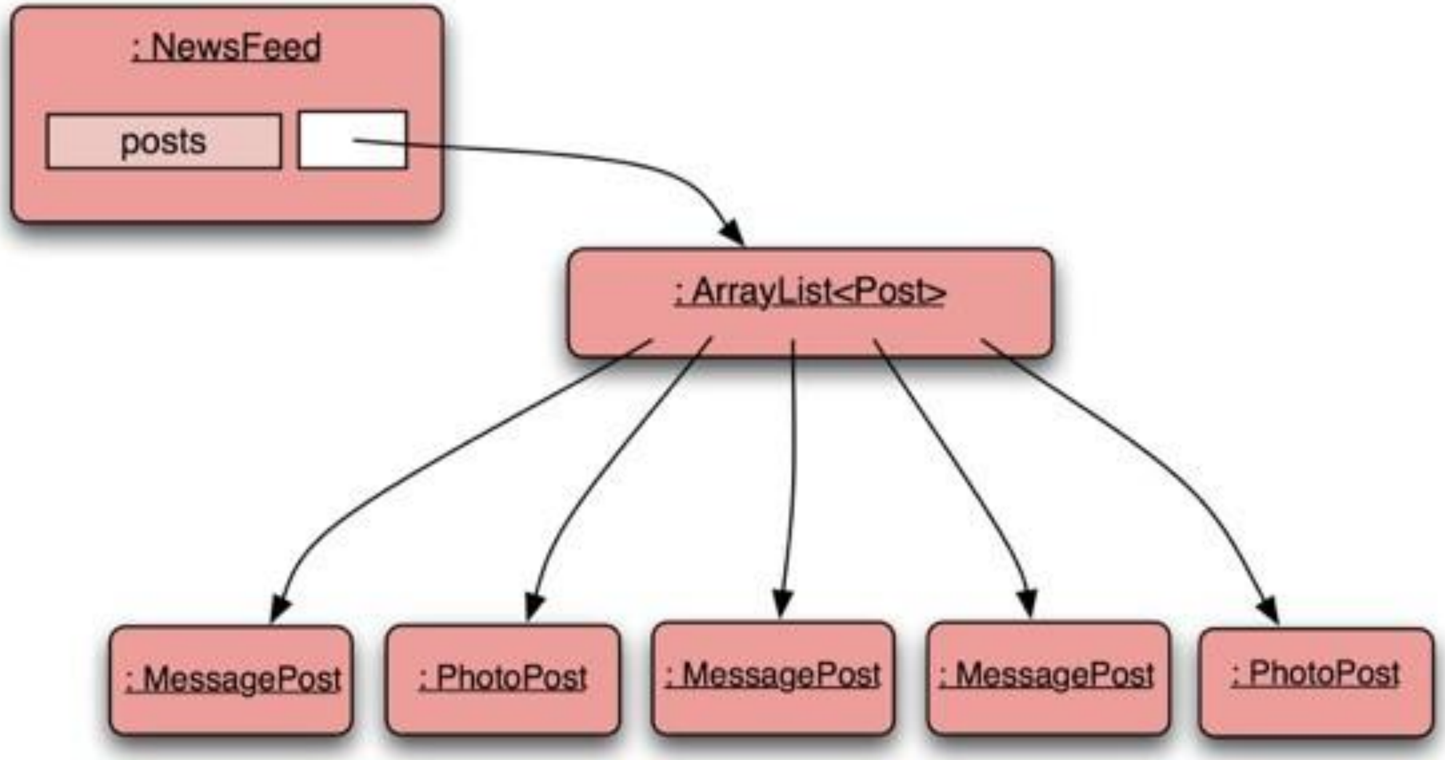
All classes inherit from Object.

Polymorphic collections

```
public class NewsFeed
{
    public void addPost(Post post)
    {
        ...
    }
}

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

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

subclass objects may be used as actual parameters for the superclass

Object diagram:
ArrayList holding Subtypes of Post



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.

Polymorphic collections

- 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.
- Java 1.5 onwards uses Generics.

Summary

- **Subtyping**: Objects of subclasses **substitute** objects of supertypes.
- **Upcasting** and **Downcasting** is used to treat supertypes as subtypes and the other way round
- **Polymorphism** (lit.: The occurrence of something in different forms)
  - Polymorphic variables
  - Polymorphic collections

THAT'S IT!

Homework

- Chapters 8.7 (“Subtyping”) – 8.11.

Other references:

- http://forum.codecall.net/topic/50451-upcasting-downcasting/
- Video code demo: http://www.youtube.com/watch?v=Uj4JdvFKTNo