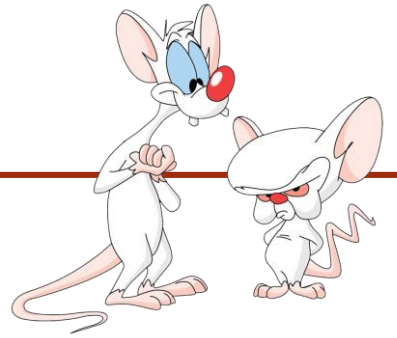
COMP 250

INTRODUCTION TO COMPUTER SCIENCE

Week 4-3: OOP 6

Giulia Alberini, Fall 2020

WHAT ARE WE GOING TO DO IN THIS VIDEO?



OOD6

- Modifiers and Inher
- Type Conversion

MODIFIERS

ACCESS CONTROL MODIFIERS

- Recall that a class can be declared to be either `public` or package-private (no keyword).
- A class can *extend* another class only if the former is visible from where the former is located.

ACCESS CONTROL MODIFIERS

- Recall that a class can be declared to be either `public` or package-private (no keyword).
- A class can *extend* another class, and a member is visible from where the former is located.


```
package assignments.a1;

public class A {
    :
}
```

```
package assignments.a2;
import assignments.a1.A;

public class B extends A{
    :
}
```

All public classes can be extended (even across packages)

ACCESS CONTROL MODIFIERS

- Recall that a class can be declared to be either `public` or package-private (no keyword).
- A class can *extend* another class, but the latter is visible from where the former is located.


```
package assignments.a1;

class A {
    :
}
```

```
packag          s;

public class B extends A{
    :
}
```

Not allowed, since A is not visible from B.

WHICH MEMBERS ARE INHERITED?

- Every superclass' member visible from where the subclass is located is inherited by the subclass (with the exception of constructors)

- Members include: fi c nested cla .

- Note that *a subclass cannot reduce the visibility of an inherited method*. The visibility can only be increased. (we'll understand better why in the next few classes)

ASIDE: NESTED CLASSES

- Note that a nested class is not a subclass.
- Outer and inner clas ds and methods of each other. Details are out of the scope of .

`final` KEYWORD

- **A class that has been declared `final` cannot be *extended*.**


```
public final class Dog {
    :
}
```

```
publi                eagle extends Dog {
    :
}
```

compile-time error!

final KEYWORD

- **A method that has been declared `final` cannot be *overridden*.**


```
public class Dog {
   public final void bark() {
      :
   }
}
```

```
class Beagle extends Dog {
   void bark() {
      :
   }
}
```

compile-time error!

TY ON

FROM LAST A COUPLE OF VIDEOS AGO

class Dog

Person owner

```
public void bark() {
    print("woof!");
}
    :
```

extends

class Beagle

void hunt ()

```
public void bark() {
    print("aowwwuuu");
}
    :
```

```
public class Test {

    public static void main(String[] args) {
        Dog snoopy = new Beagle();
    }
}
```


Is this allowed??

OBJECTS TYPE

- We have seen that an object is of the type of the class from which it was instantiated.

- For example, if we write


```
Dog myDog = new Dog();
```

then `myDog` points to an object of type `Dog`.

OBJECT TYPES

- But `Dog` is a subclass of `Animal` which is a subclass of `Object`.

- Thus, a `Dog` is an `Anim` We can use an object of type `Dog` wherever obj `An` `ject` are called for.

- Note that the reverse is not necessarily true: an `Animal` could be a `Dog`, but not necessarily. Similarly, an `Object` could be an `Animal` or a `Dog`, but it isn't necessarily.

TYPE CASTING – REFERENCE TYPES

- Casting allows us to use an object of one type in place of another type, if permitted.

- For example we can wri


```
Animal myPet          g();
```

This will not cause a compile-time error because there is an ***implicit upcasting*** since a `Dog` is for sure also an `Animal`.

TYPE CASTING – REFERENCE TYPES

On the other hand, consider the following

```
Animal myPet = new Dog();
Dog myDog = myPet;
```

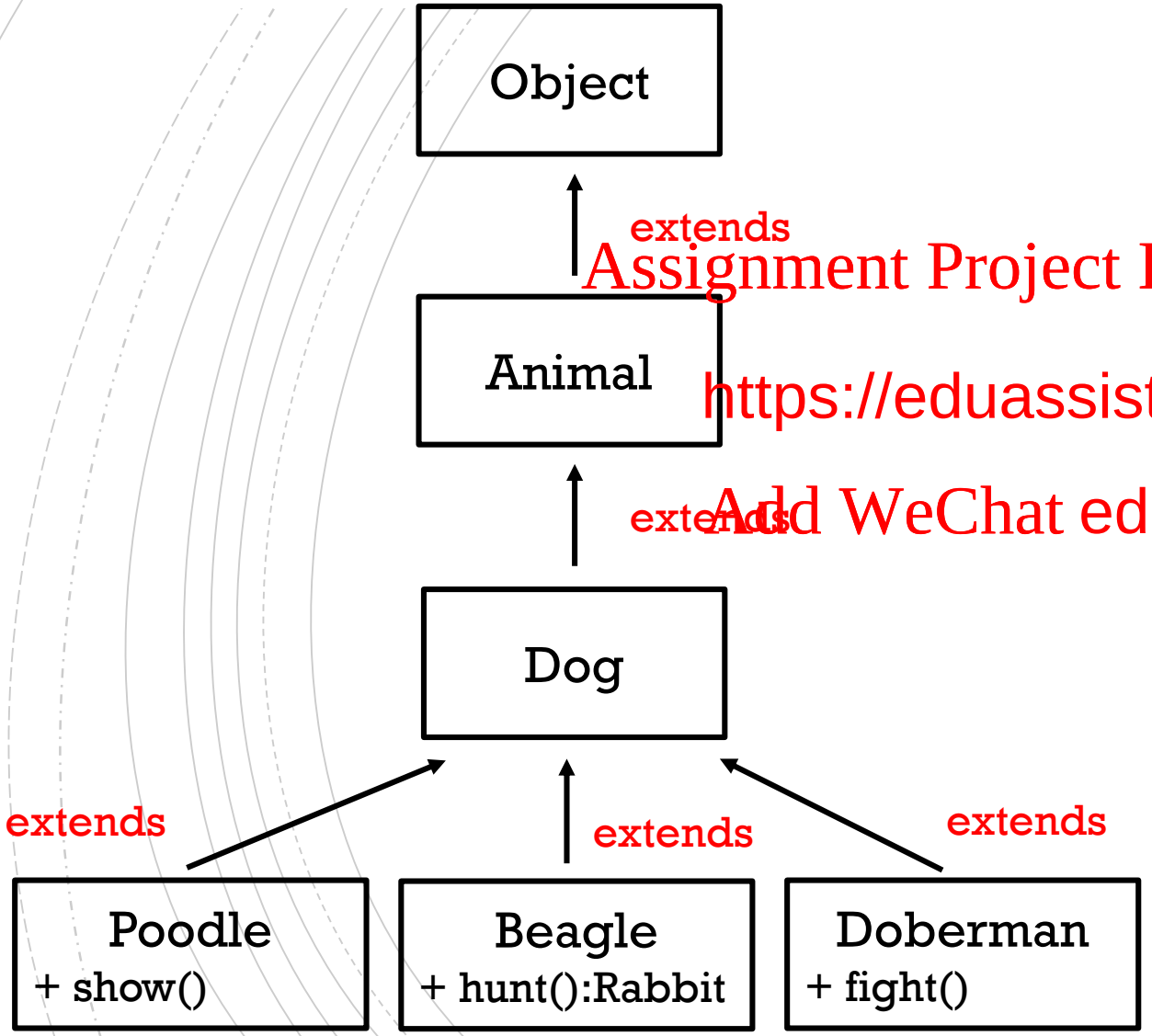

The second line will cause a compile-time . the compiler point of view, `myPet` is of type `Animal` and an `Animal` mi `Dog`.

However, we can tell the compiler that `myPet` is of the correct type, by ***explicitly downcasting***:

```
Dog myDog = (Dog) myPet;
```

If `myPet` turns out to be of the wrong type we'll get a run-time error.

HIERARCHY FROM LAST CLASS



Upcasting

Happens automatically

IMPORTANT!

Note that casting does NOT change the object itself, it just labels it differently!

Downcasting

The programmer has to manually do it.

EXAMPLES

```
Dog myDog = new Beagle();
```


Is this allowed?

- Yes, it is an example of upcasting which happens automatically.

EXAMPLES

```
Dog myDog = new Beagle();

Poodle
```


Is this allowed?

- **Compile-time error! The variable `myDog` is of type `Dog`, and it might not be pointing to a `Poodle`. It requires explicit downcasting to compile.**

EXAMPLES

```
Dog myDog = new Beagle();

Poodle myDog;
```


Is this allowed?

- **The code compiles, but there will be a run-time error because `myDog` is not pointing to a `Poodle` after all.**

EXAMPLES

```
Dog myDog = new Beagle();
myDog.hunt();
```


Is this allowed?

- **Compile-time error! The variable `myDog` is of type `Dog`, and there is no method called `hunt` inside the `Dog` class.**

EXAMPLES

```
Dog myDog = new Beagle();
((Beagl
```


Is this allowed?

- Yes, this code will compile and run.

NEXT VIDEO!

class Dog

Person owner

```
public void bark() {
    print("woof!");
}
    :
```

extends

class Beagle

void hunt ()

```
public void bark() {
    print("aowwwuuu");
}
    :
```

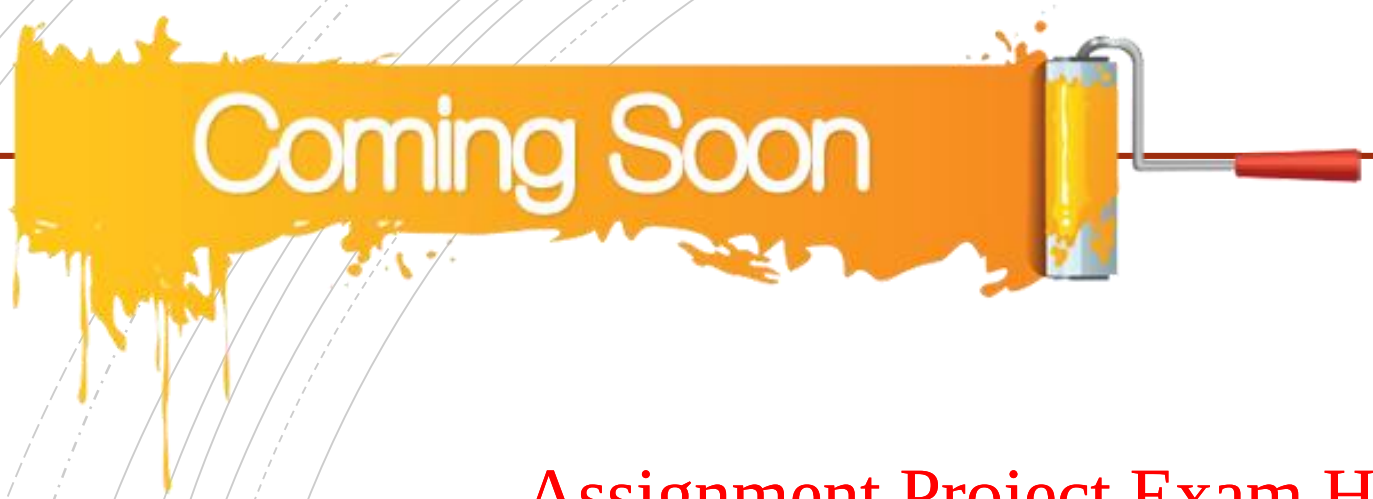
```
public class Test {

    public static void main(String[] args) {
        Dog snoopy = new Beagle();
    }
}
```


Which `bark()` will execute???

Is this allowed??

Yes, it's an example of upcasting!



In the next

- Polymor