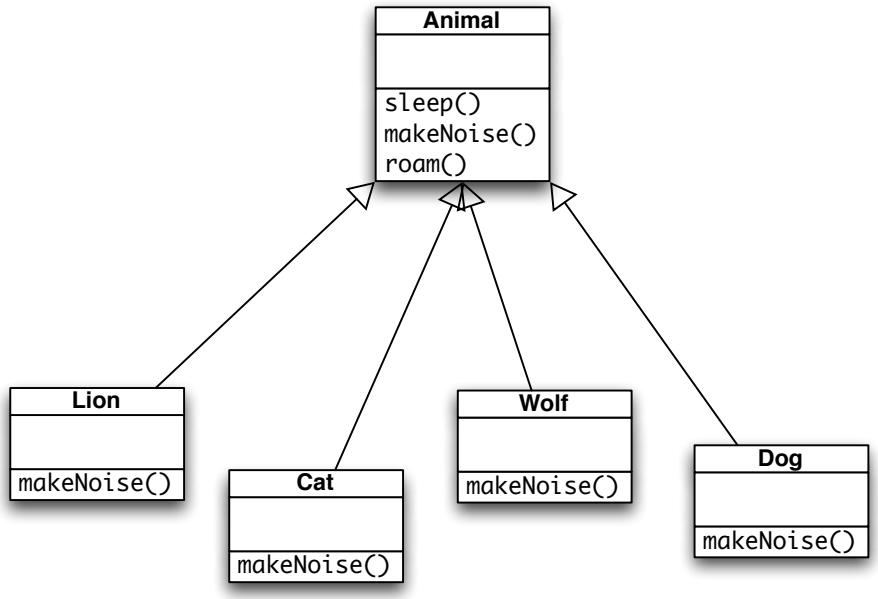
Object-Oriented Programming
Inheritance & Polymorphism

Ewan Klein

School of Informatics

Inf1 :: 2009/10

Flat Animal Hierarchy



Animals Example, 1

Our base class: `Animal`

Animal

```
public class Animal {
    public void sleep() {
        System.out.println("Sleeping: Zzzzz");
    }
    public void makeNoise() {
        System.out.println("Noises...");
    }
    public void roam() {
        System.out.println("Roamin' on the plain.");
    }
}
```

Animals Example, 2

1. `Lion IS-A Animal`
2. Override the `makeNoise()` method.

Lion

```
public class Lion extends Animal {
    public void makeNoise() {
        System.out.println("Roaring: Rrrrrr!");
    }
}
```


Animals Example, 3

1. `Cat IS-A Animal`
2. Override the `makeNoise()` method.

Cat

```
public class Cat extends Animal {
    public void makeNoise() {
        System.out.println("Miaowing: Miaooo!");
    }
}
```


Animals Example, 4

1. `Wolf IS-A Animal`
2. Override the `makeNoise()` method.

Wolf

```
public class Wolf extends Animal {
    public void makeNoise() {
        System.out.println("Howling: Ouoooooo!");
    }
}
```


Animals Example, 5

1. `Dog IS-A Animal`
2. Override the `makeNoise()` method.

Dog

```
public class Dog extends Animal {
    public void makeNoise() {
        System.out.println("Barking: Woof Woof!");
    }
}
```

Animals Example, 6

The Launcher

```
public class AnimalLauncher {
    public static void main(String[] args) {
        System.out.println("\nWolf\n=====");
        Wolf wolfie = new Wolf();
        wolfie.makeNoise(); // from Wolf
        wolfie.roam(); // from Animal
        wolfie.sleep(); // from Animal

        System.out.println("\nLion\n=====");
        Lion leo = new Lion();
        leo.makeNoise(); // from Lion
        leo.roam(); // from Animal
        leo.sleep(); // from Animal
    }
}
```


Animals Example, 7

Output

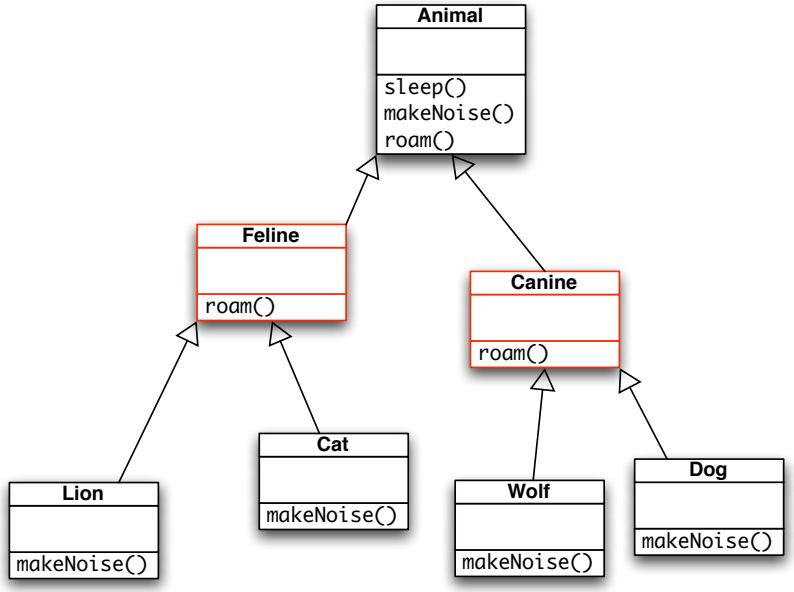
```
Wolf
=====
Howling: Ouoooooo!
Roamin' on the plain.
Sleeping: Zzzzz

Lion
=====
Roaring: Rrrrrr!
Roamin' on the plain.
Sleeping: Zzzzz
```


Nested Animal Hierarchy

- Lions and cats can be grouped together into `Felines`, with common `roam()` behaviours.
- Dogs and wolves can be grouped together into `Canines`, with common `roam()` behaviours.

Nested Animal Hierarchy



Animals Example, 1

Same as before.

Animal

```
public class Animal {
    public void sleep() {
        System.out.println("Sleeping: Zzzzz");
    }
    public void makeNoise() {
        System.out.println("Noises...");
    }
    public void roam() {
        System.out.println("Roamin' on the plain.");
    }
}
```


Animals Example, 2

The new class `Feline`

Feline

```
public class Feline extends Animal {
    public void roam() {
        // Override roam()
        System.out.println("Roaming: I'm roaming alone.");
    }
}
```


Animals Example, 3

The new class `Canine`

Canine

```
public class Canine extends Animal {
    public void roam() {
        // Override roam()
        System.out.println("Roaming: I'm with my pack.");
    }
}
```


Animals Example, 4

1. `Lion IS-A Feline`
2. Override the `makeNoise()` method.

Lion

```
public class Lion extends Feline {
    public void makeNoise() {
        System.out.println("Roaring: Rrrrrr!");
    }
}
```

- Similarly for `Cat`.

Animals Example, 5

1. Wolf IS-A Canine
2. Override the makeNoise() method.

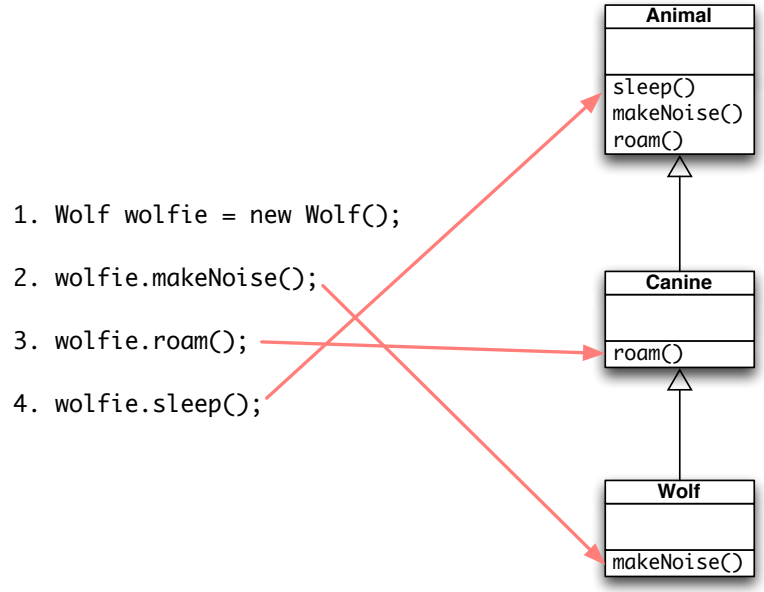
Wolf

```
public class Wolf extends Canine {
    public void makeNoise() {
        System.out.println("Howling: Ouoooooo!");
    }
}
```

- Similarly for Dog.

Which method gets called?

```
1. Wolf wolfie = new Wolf();
2. wolfie.makeNoise();
3. wolfie.roam();
4. wolfie.sleep();
```



Animals Example, 6

The Launcher

```
public class AnimalLauncher {
    public static void main(String[] args) {
        System.out.println("\nWolf\n=====");
        Wolf wolfie = new Wolf();
        wolfie.makeNoise(); // from Wolf
        wolfie.roam(); // from Canine
        wolfie.sleep(); // from Animal

        System.out.println("\nLion\n=====");
        Lion leo = new Lion();
        leo.makeNoise(); // from Lion
        leo.roam(); // from Feline
        leo.sleep(); // from Animal
    }
}
```


Animals Example, 7

Output

```
Wolf
=====
Howling: Ouoooooo!
Roaming: I'm with my pack.
Sleeping: Zzzzz

Lion
=====
Roaring: Rrrrrr!
Roaming: I'm roaming alone.
Sleeping: Zzzzz
```

Animals Example, 7

Animal

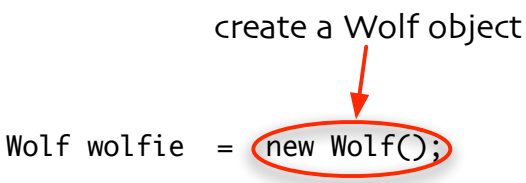
```
public class Animal {
    public void sleep() {
        System.out.println("Sleeping: Zzzzz");
    }
    public void makeNoise() {
        System.out.println("Noises...");
    }
    public void roam() {
        System.out.println("Roamin' on the plain.");
    }
}
```

- Any class that extends `Animal` must support the methods:
  - `sleep()`
  - `makeNoise()`
  - `roam()`

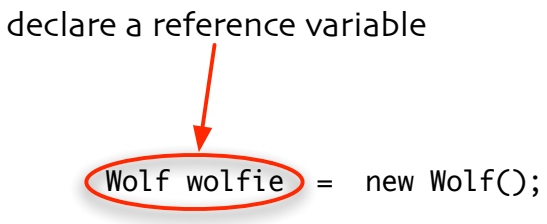
Typing and Polymorphism

- polymorphism (= 'many shapes'): the same piece of code can be assigned multiple types.
- A class defines a type, namely the signatures of its methods.
- S is a subtype of T, written S <: T, if a value of type S can be used in any context where a value of type T is expected.
- The relation <: is reflexive: T <: T
- The relation <: is transitive: if S <: T and T <: U, then S <: U.
- (NB: We say T is a supertype of S if S is a subtype of T.)
- Inclusion polymorphism: objects of different types S1, S2, ...may be treated uniformly as instances of a common supertype T.

Declaring and Initializing a Reference Variable

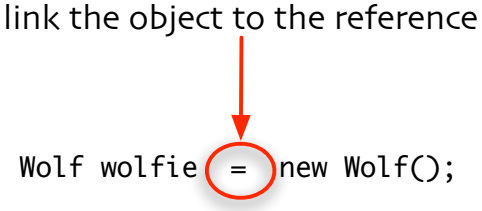


Declaring and Initializing a Reference Variable

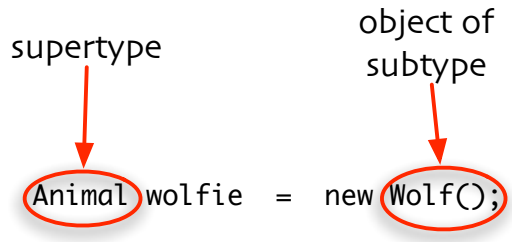


Declaring and Initializing a Reference Variable

link the object to the reference



Declaring and Initializing a Reference Variable



- Reference type can be supertype of the object type.
- E.g., `Wolf <: Animal`.

Polymorphic `ArrayList`

The Launcher

```
public class AnimalLauncher2 {
   public static void main(String[] args) {
       Wolf wolfie = new Wolf();
       Lion leo = new Lion();
       Cat felix = new Cat();
       Dog rover = new Dog();
       ArrayList< Animal > animals = new ArrayList<Animal>();
       animals.add(wolfie);
       animals.add(leo);
       animals.add(felix);
       animals.add(rover);
       for ( Animal a  : animals) {
            a.makeNoise();
       }
   }
}
```


Polymorphic Arrays

`ArrayList<Animal>` is polymorphic.

- `animals.add(wolfie)`
  add an object of type `Wolf`. OK since `Wolf <: Animal`.
- `for (Animal a : animals)`
  for each object `a` of type `T` such that `T <: Animal` ...
- `a.makeNoise()`
  if `a` is of type `T`, use `T`'s `makeNoise()` method.

Method Overriding, 1

If a class C overrides a method m of superclass D, then:

- Parameter lists must be same and return type must be compatible:
  1. signature of m in C must be same as signature of m in D; i.e. same name, same parameter list, and
  2. return type S of m in C must such that S <: T, where T is return type of m in D.
- m must be at least as accessible in C as m is in D

Method Overriding, 2

method in Animal

```
public void makeNoise() {
...
}
```

Wrong: method in Wolf

```
public void makeNoise( int volume ) {
...
}
```

Wrong: method in Wolf

```
private void makeNoise() {
...
}
```


Method Overloading, 1

Overloading: two methods with same name but different parameter lists.

Overloaded makeNoise

```
public void makeNoise() {
...
}
public void makeNoise(int volume) {
...
}
```

Overloaded println

```
System.out.println(3); // int
System.out.println(3.0); // double
System.out.println((float) 3.0); // cast to float
System.out.println("3.0"); // String
```


Method Overloading, 2

1. Return types can be different.
2. You can't just change the return type — gets treated as an invalid override.
3. Access levels can be varied up or down.

Incorrect override of makeNoise

```
public String makeNoise() {
String howl = "Ouooooo!";
return howl;
}
Exception in thread "main" java.lang.Error: Unresolved compila-
tion problem:
        The return type is incompatible with Animal.makeNoise()

        at week06.Wolf.makeNoise(Wolf.java:15)
        at week06.AnimalLauncher.main(AnimalLauncher.java:11)
```

Reading, etc

- This week, read Chapters 7 & 8 of *Head First Java*.
- We'll look at Chapter 8 on Friday.
- Try using Eclipse this week (see links from OOP Webpage).