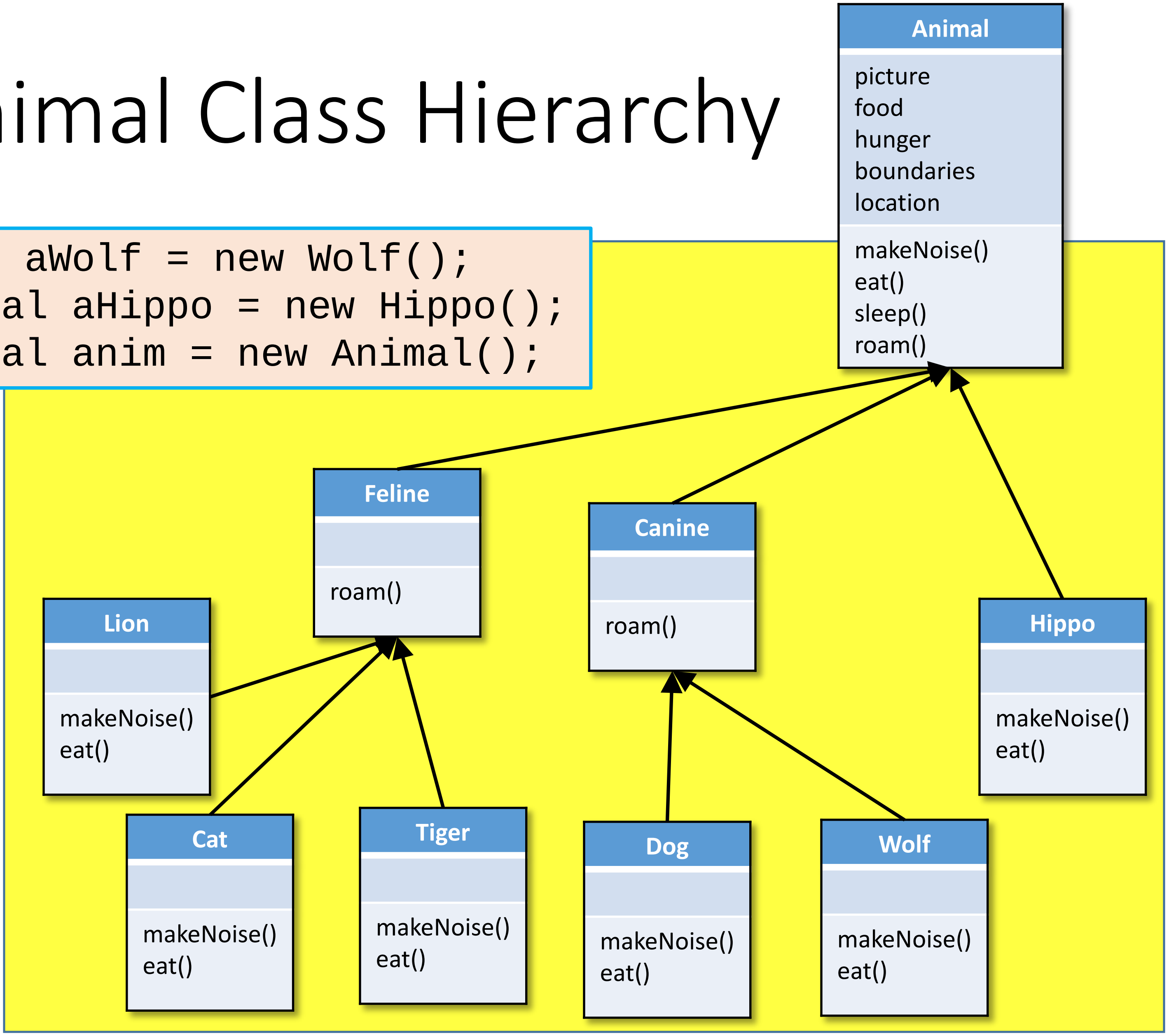
Interfaces and Polymorphism

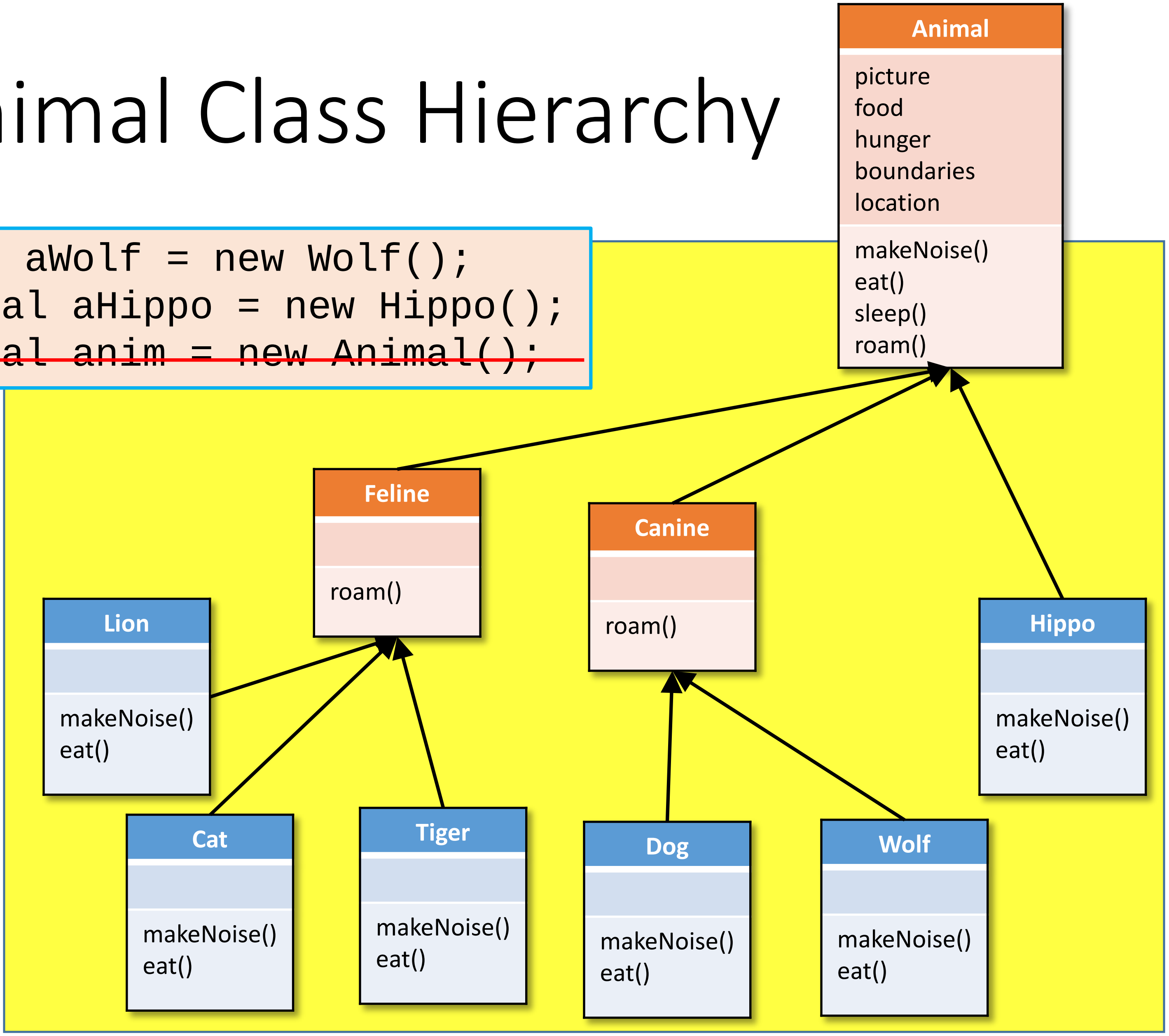
Animal Class Hierarchy

```
1. Wolf aWolf = new Wolf();
2. Animal aHippo = new Hippo();
3. Animal anim = new Animal();
```



Animal Class Hierarchy

```
1. Wolf aWolf = new Wolf();
2. Animal aHippo = new Hippo();
3. ~~Animal anim = new Animal();~~
```



Abstract Class

- Abstract classes c

```
public abstract c
        ...
}

public abstract c
        ...
}

public abstract c
        ...
}
```

```
public class MakeCanine {
        public void go() {
                Canine c;
                c = new Dog();
                c = new Canine();
                c.roam();
        }
}

% java MakeCaine.java
MakeCanine.java:5: Canine is abstract;
cannot be instantiated
        c = new Canine();
            ^
1 error.
```

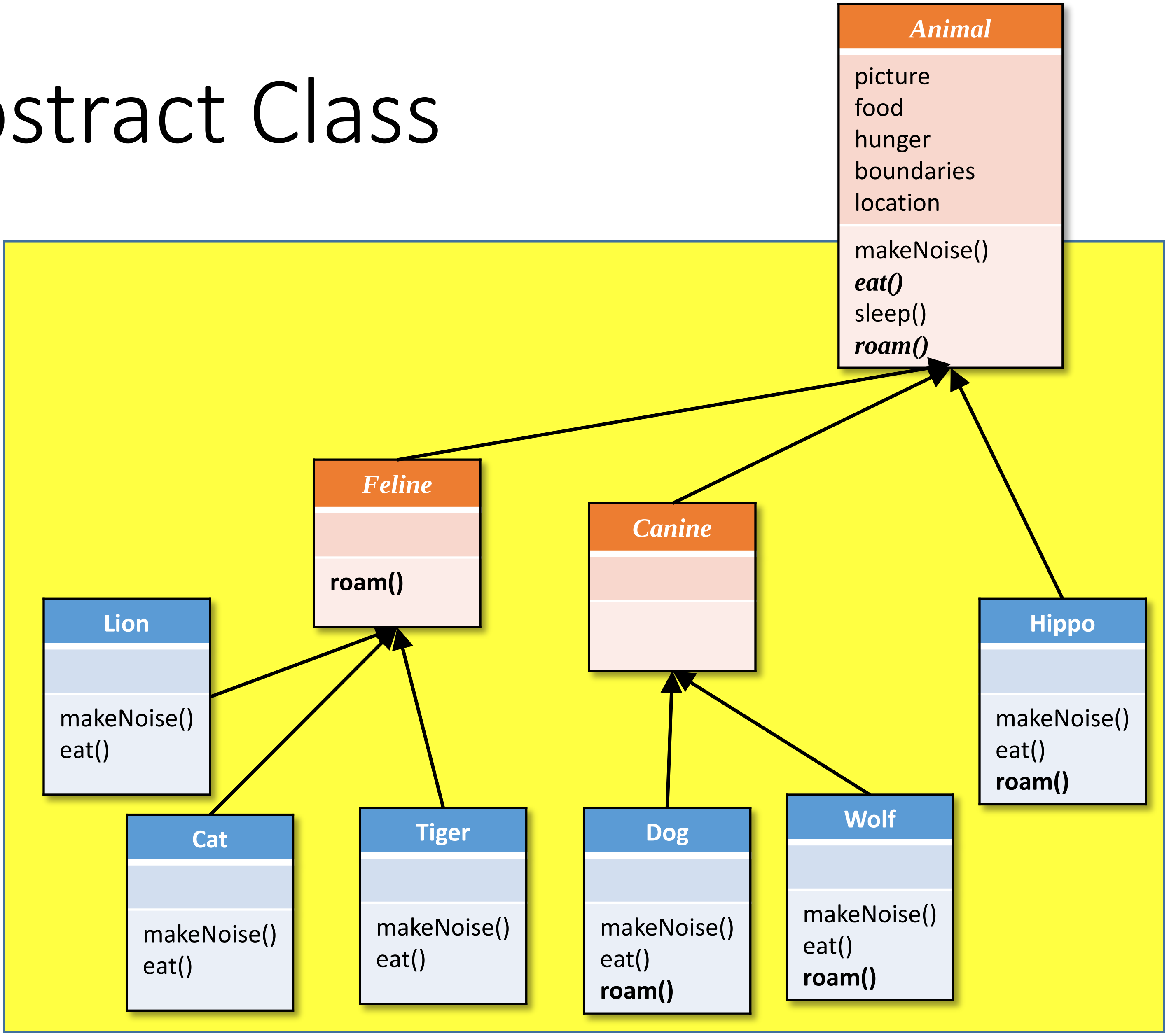
Abstract method

- Abstract method

```
public abstract class Animal {
        ...
        public abstract void eat();
        ...
        public void sleep() {
            ...
        }
}
```

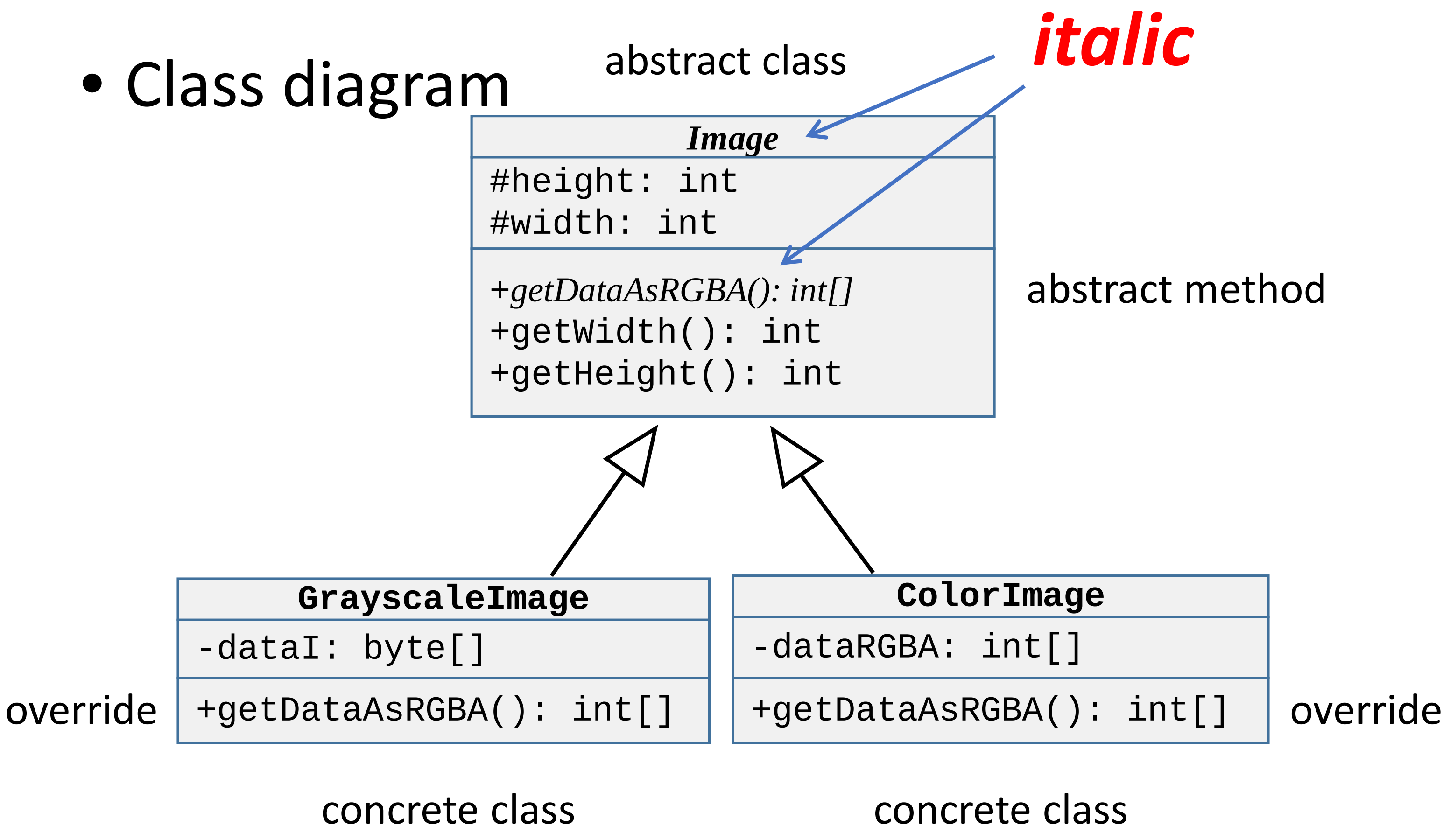
  - You can't have an abstract method in a non-abstract class.
  - All abstract methods must be implemented in an abstract subclass above, or in the concrete subclass itself.

Abstract Class



Abstract class (1)

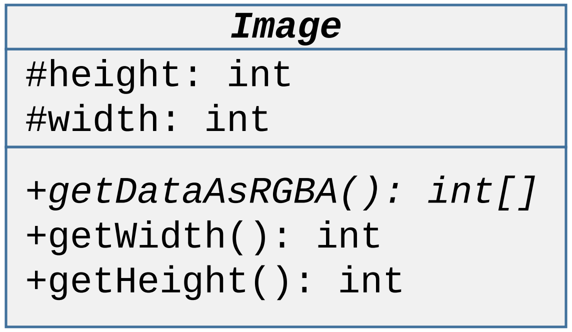
- Class diagram



Abstract class (2)

- *Image* class

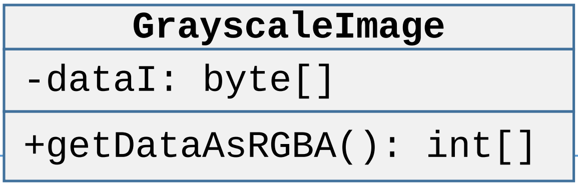
```java
abstract public class Image {
    protected int height;
    protected int width;

    public abstract int[] getDataAsRGBA();

    public int getWidth() {
        return this.width;
    }
    public int getHeight() {
        return this.height;
    }
}
```



Abstract class (3)

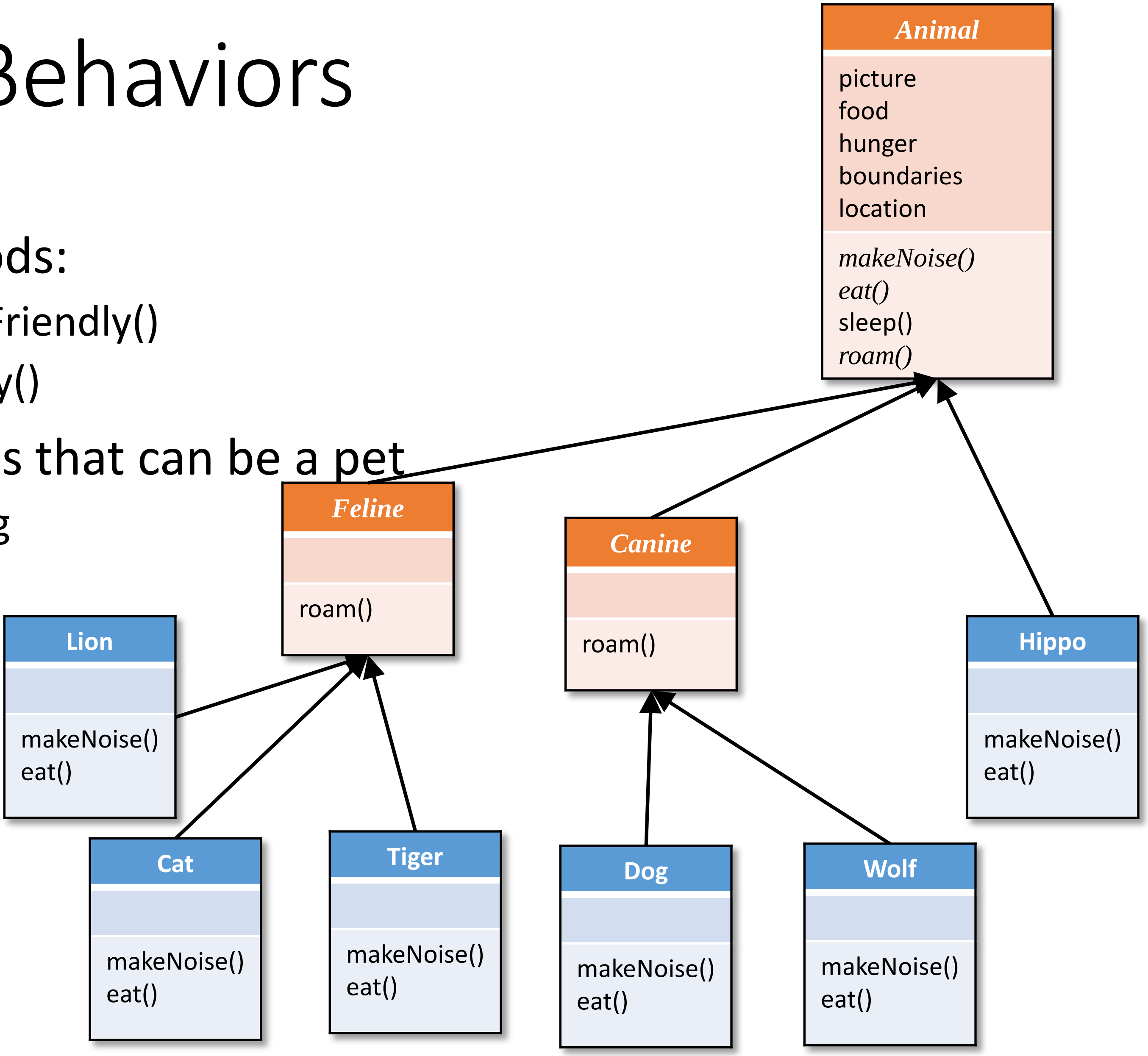
- *GrayscaleImage* class (concrete class)

```
public class GrayscaleImage extends Image {
   protected byte[] dataI;

   GrayscaleImage(int width, int height, byte[] data) {
      // ...
   }

   public int[] getDataAsRGBA() {
      // ...
   }
}
```



Pet Behaviors

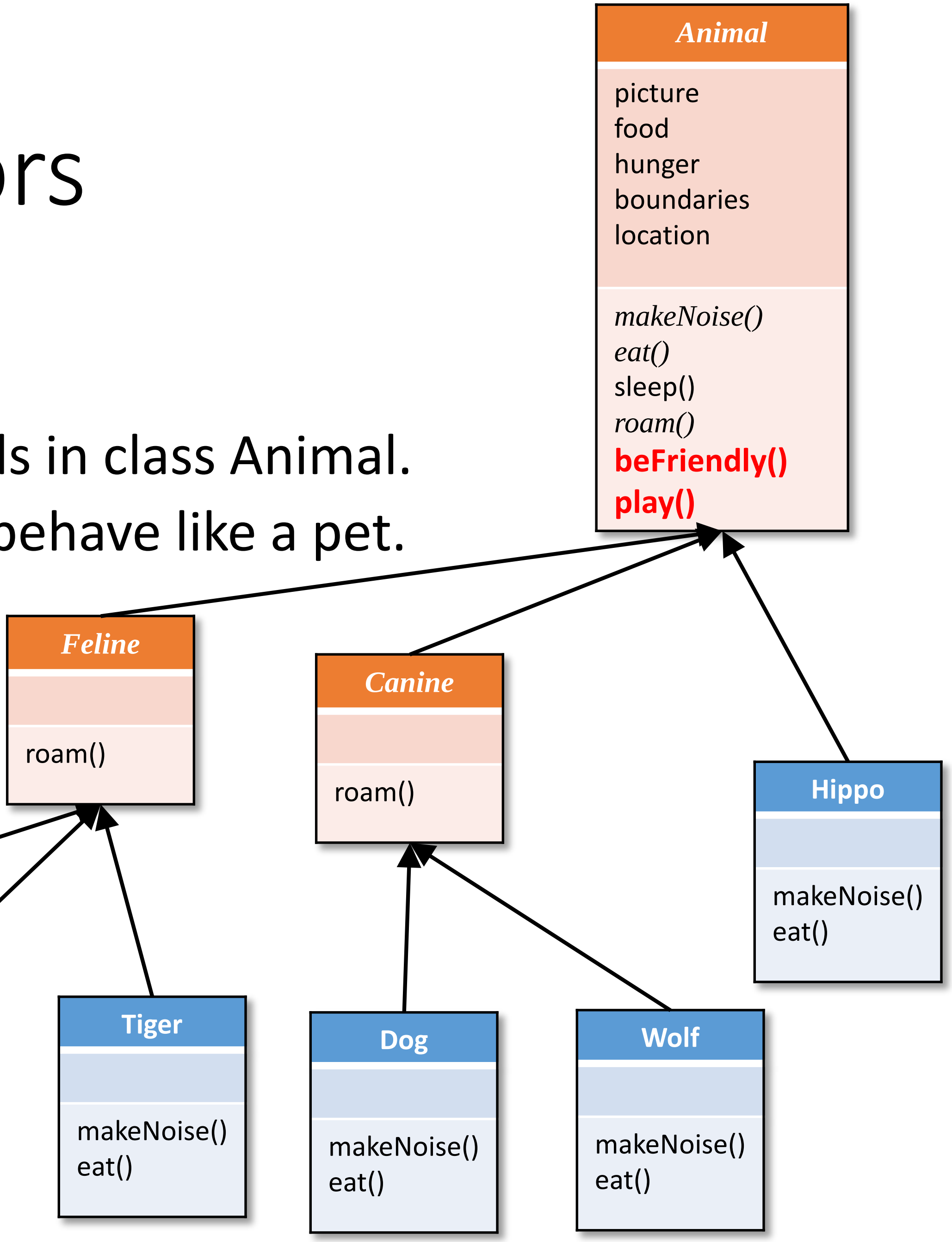
- Methods:
  - beFriendly()
  - play()
- Objects that can be a pet
  - Dog
  - Cat



Pet Behaviors

- Option 1:
  - Put pet methods in class Animal.
  - Even non-pets behave like a pet.

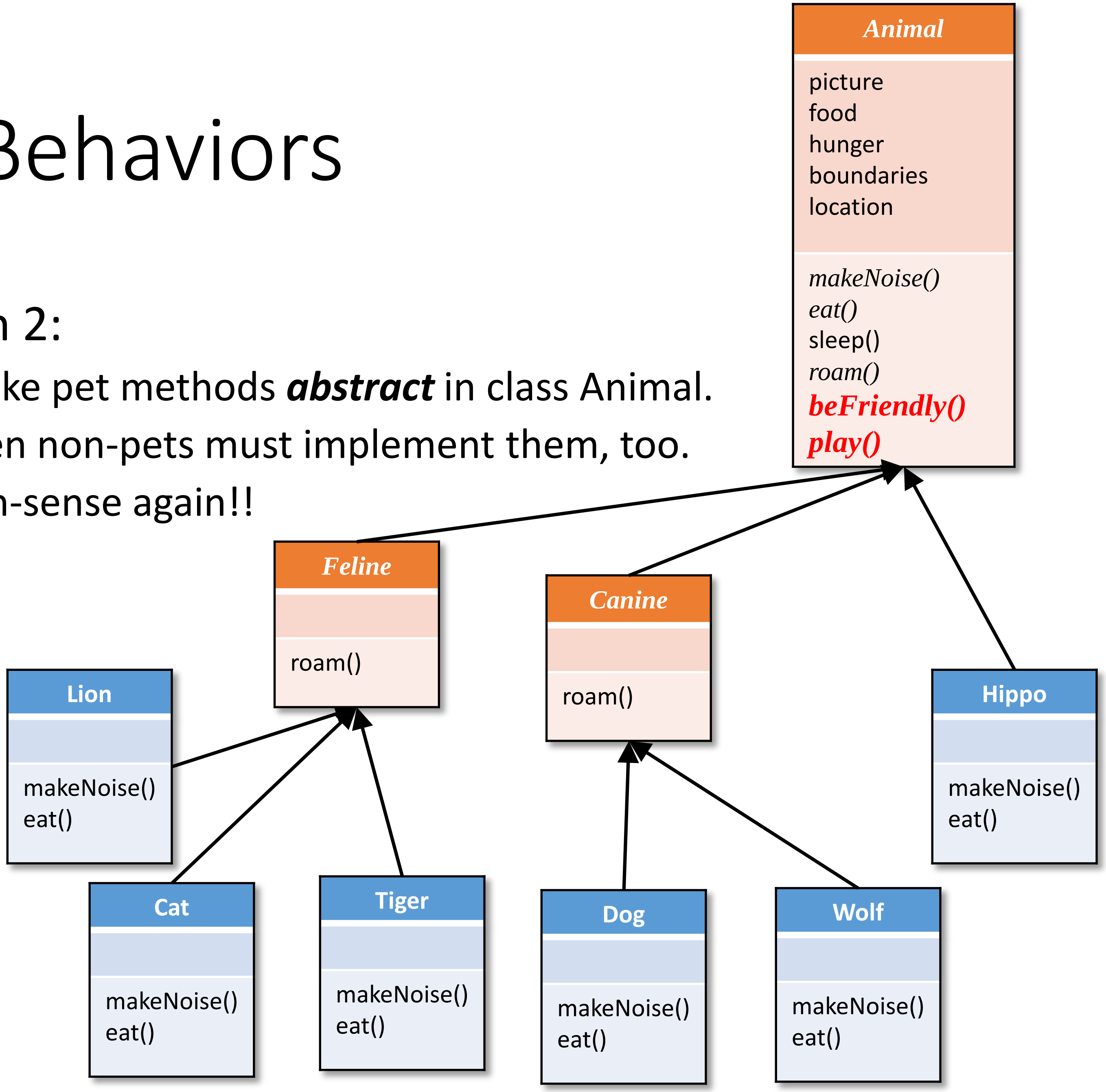
    Non-sense !!



Pet Behaviors

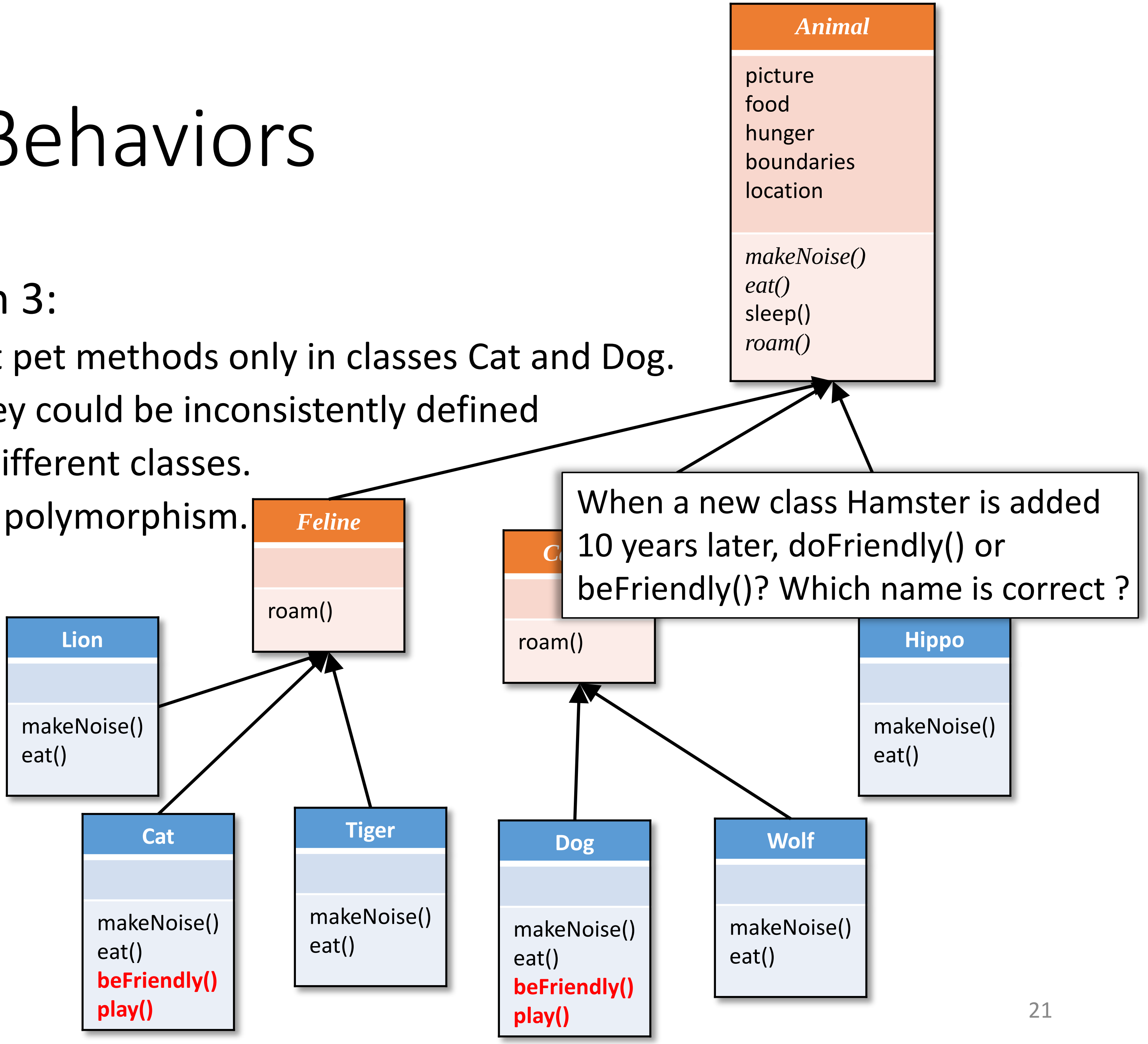
- Option 2:
  - Make pet methods ***abstract*** in class Animal.
  - Even non-pets must implement them, too.

  Non-sense again!!



Pet Behaviors

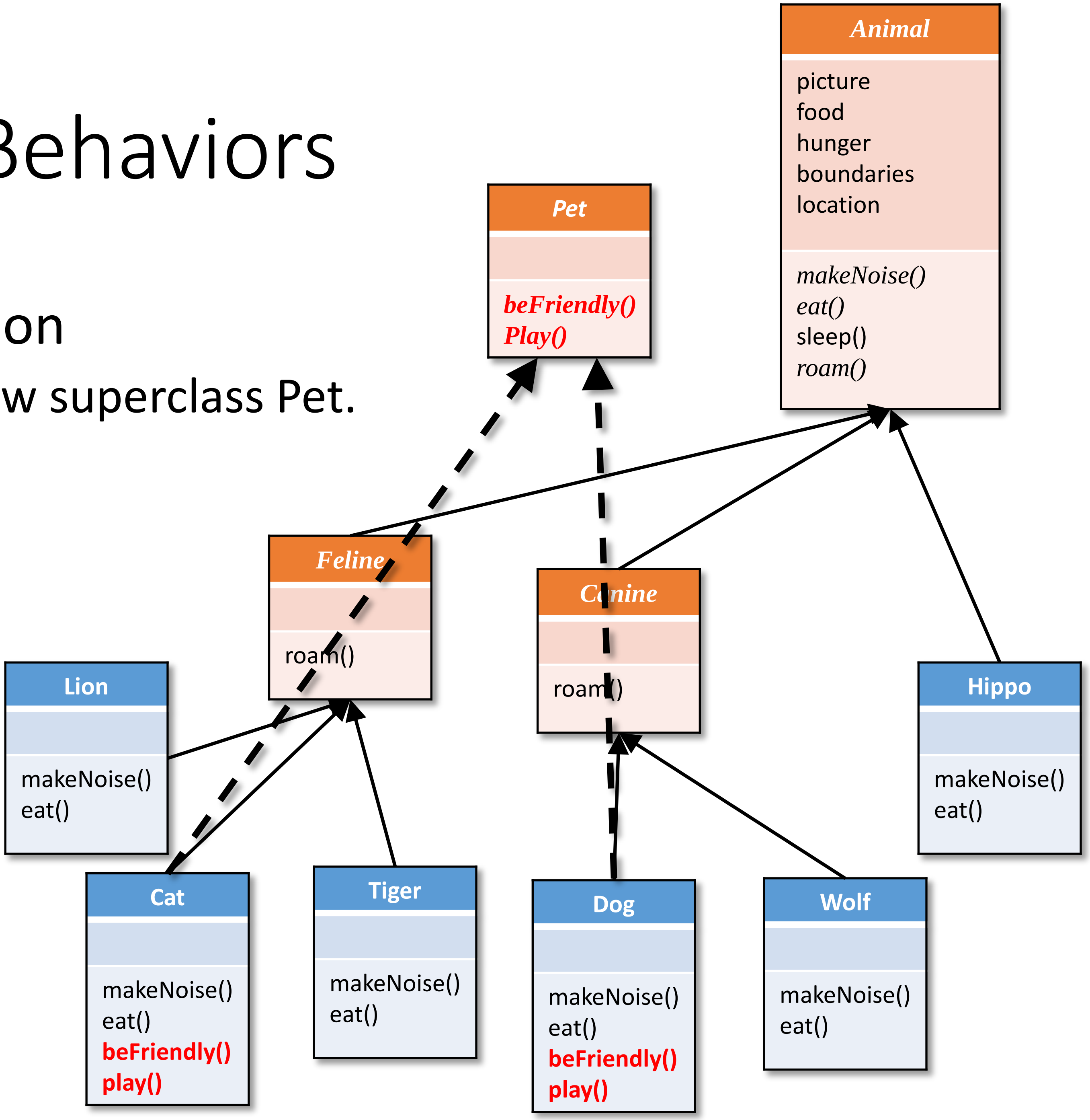
- Option 3:
  - Put pet methods only in classes Cat and Dog.
  - They could be inconsistently defined in different classes.
  - No polymorphism.



When a new class Hamster is added 10 years later, doFriendly() or beFriendly()? Which name is correct ?

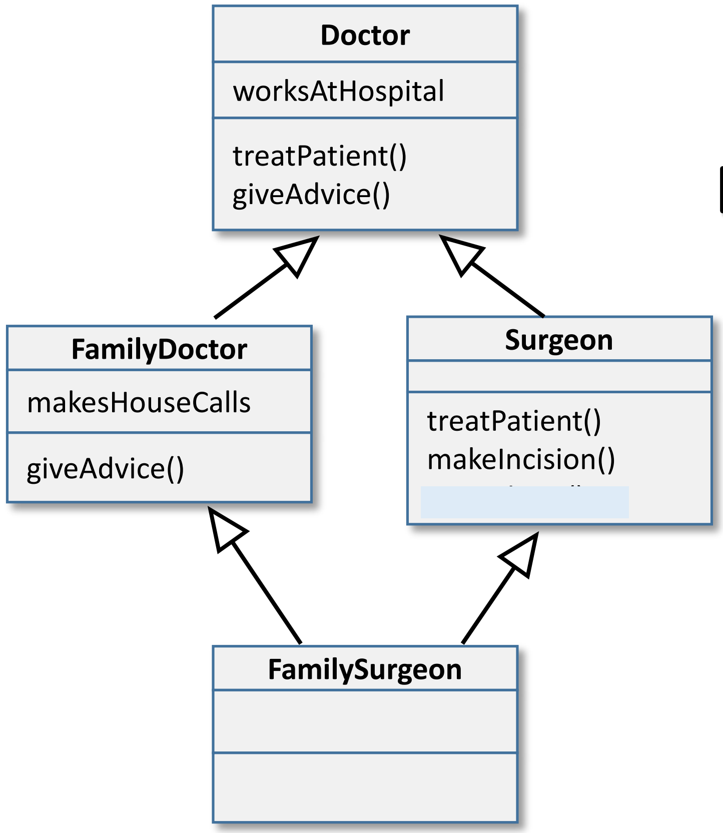
Pet Behaviors

- Solution
  - New superclass Pet.



Review: Multiple Inheritance

- It's *NOT* allowed in Java



Deadly Diamond of Death

Interface

- Interface
  - A group of related **methods** with **empty bodies**.

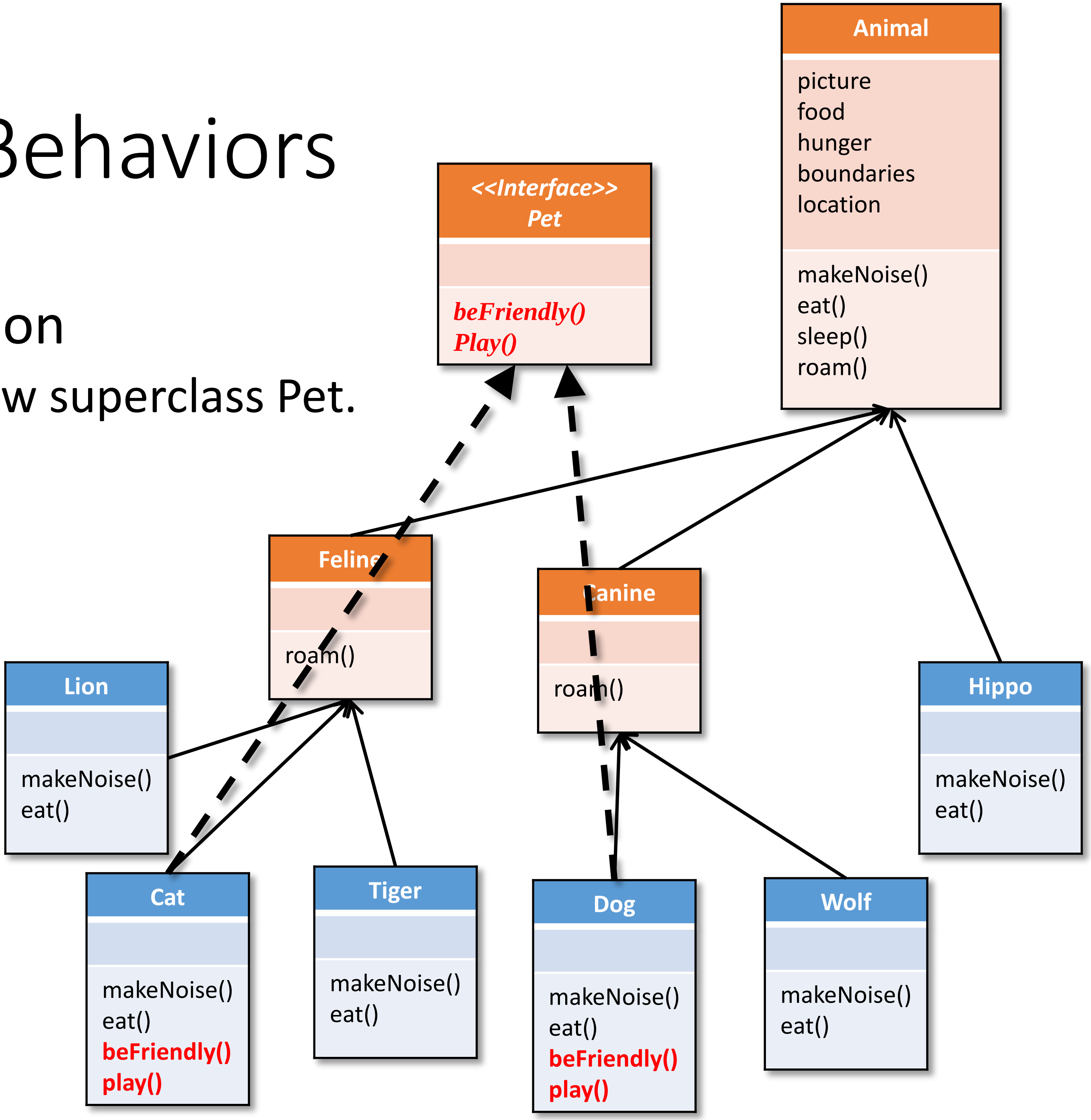
```
public interface Pet {
   public abstract void beFriendly();
   public abstract void play();
}
```

```
public class Dog extends Canine implements Pet {
   public void beFriendly() { ... }
   public void play() { ... }

   pu
   pu
}
```

```
public class Cat extends Feline implements Pet {
   public void beFriendly() { ... }
   public void play() { ... }

   public void roam() { ... }
   public void eat() { ... }
}
```

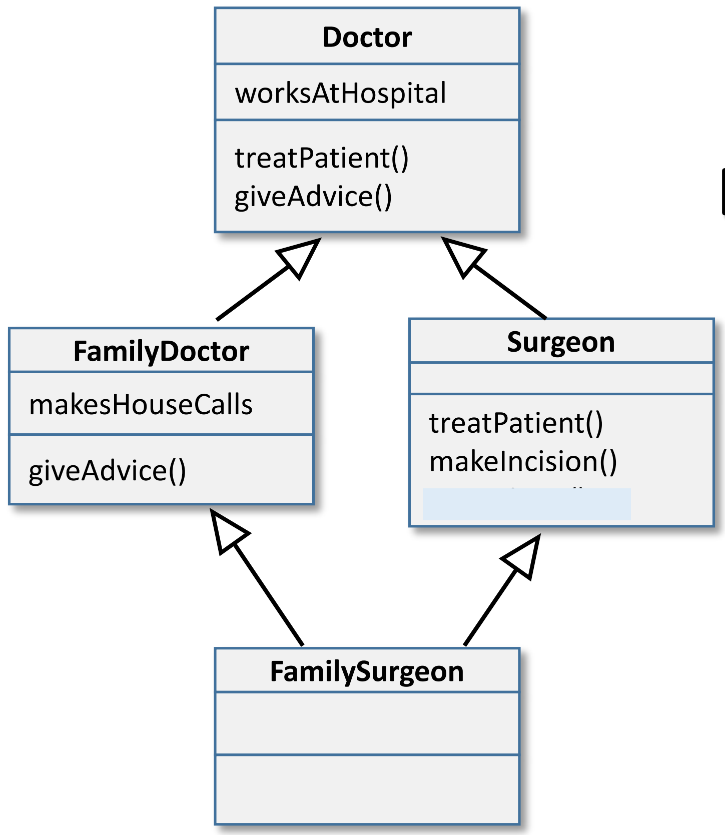
Pet Behaviors

- Solution
  - New superclass Pet.



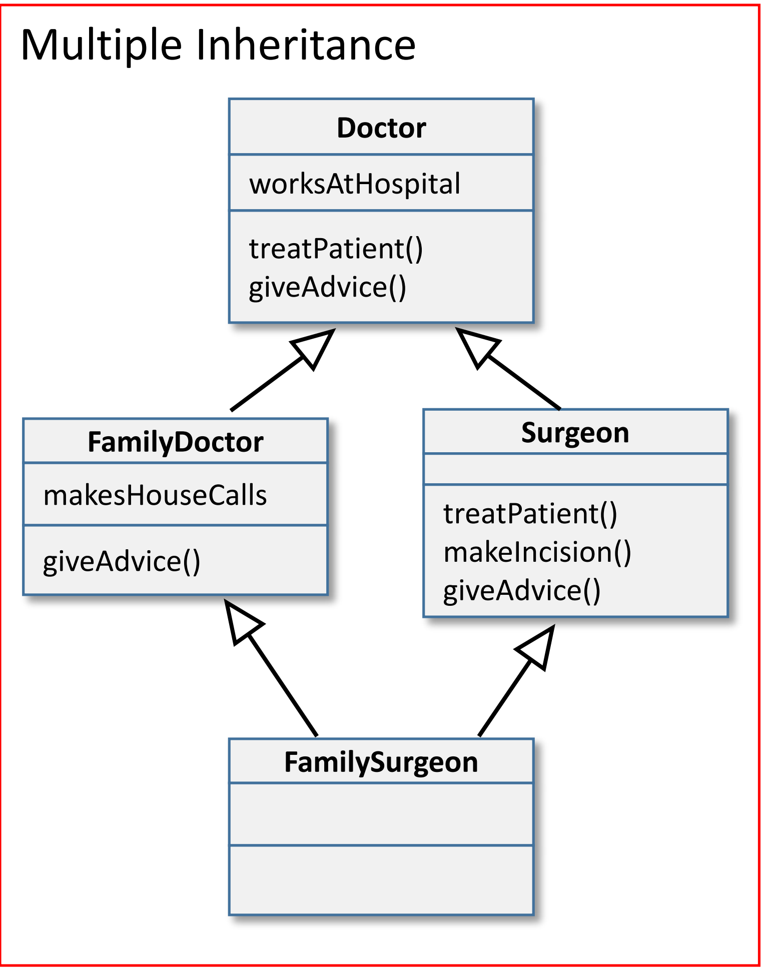
Multiple Inheritance

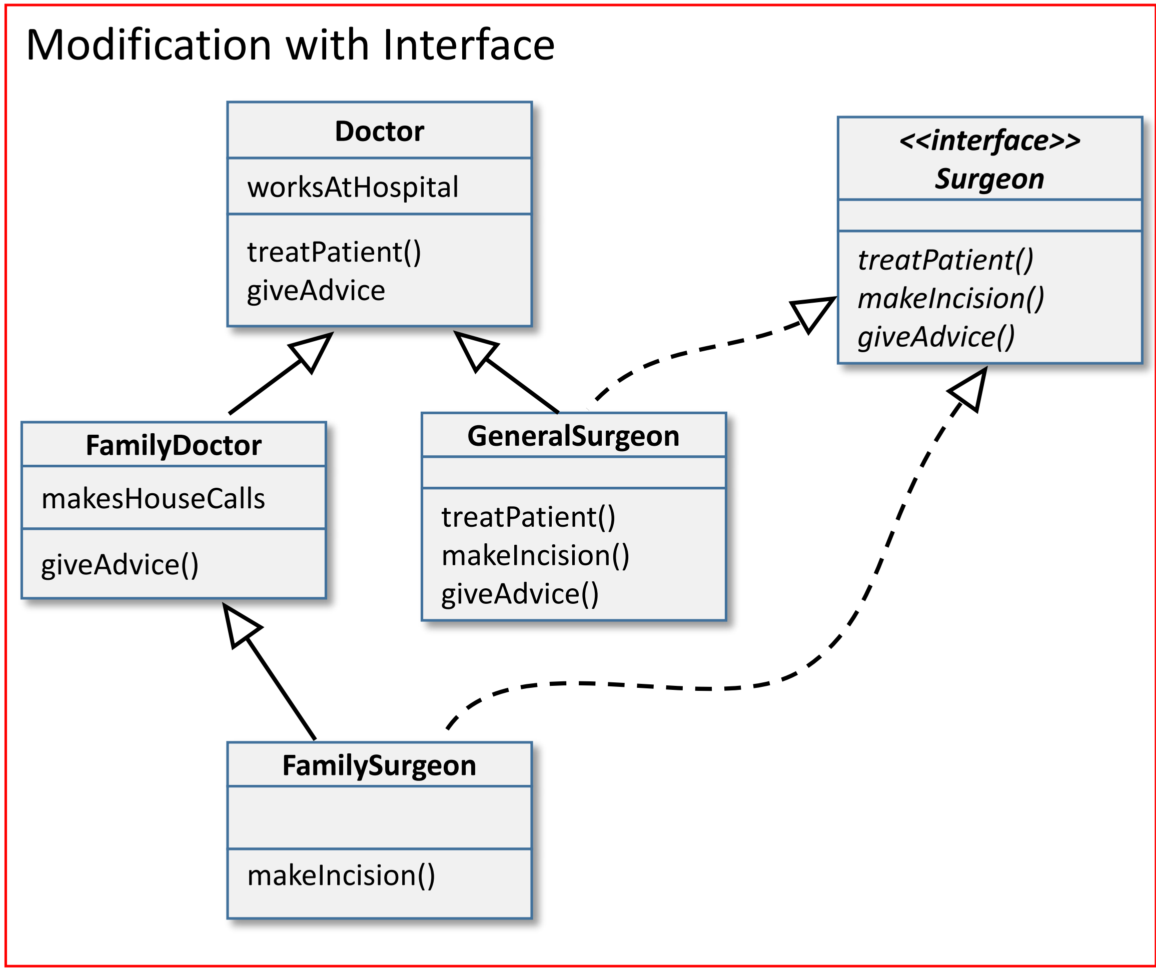
- It's *NOT* allowed in Java



Deadly Diamond of Death

Interface


Multiple Inheritance
Doctor
worksAtHospital
treatPatient()
giveAdvice()
FamilyDoctor
makesHouseCalls
giveAdvice()
Surgeon
treatPatient()
makeIncision()
giveAdvice()
FamilySurgeon


Modification with Interface
Doctor
worksAtHospital
treatPatient()
giveAdvice
<<interface>>
Surgeon
treatPatient()
makeIncision()
giveAdvice()
FamilyDoctor
makesHouseCalls
giveAdvice()
GeneralSurgeon
treatPatient()
makeIncision()
giveAdvice()
FamilySurgeon
makeIncision()

Interface

- A class can implement multiple interfaces.

```
public class Dog extends Canine implements
        Pet, Savable, Paintable  {

   public void beFriendly() { ... }
   public void play() { ... }

   public void roam() { ... }
   public void eat() { ... }
}
```

- All interface methods are implicitly public and abstract.
- An interface can extend several other interfaces.

```
public interface TinyPet extends Pet {
   public void putIntoPocket();
}
```

- An interface can have ***public final*** variables.

super

- Usage

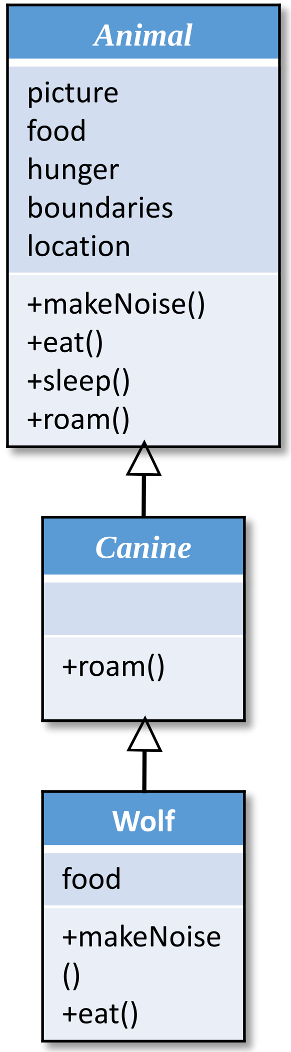
```
class HighTechnology {
   ...
   void setInteger(int value) {
      // The implementation of this method is very difficult.
      // And the source code is not available.
   }
}
```

```
class MoreSafeHighTechnology extends HighTechnology{
   ...
   void setInteger(int value) {
      if (value < 0) return;
      super.setInteger(value);
   }
}
```

- super.super is illegal in Java

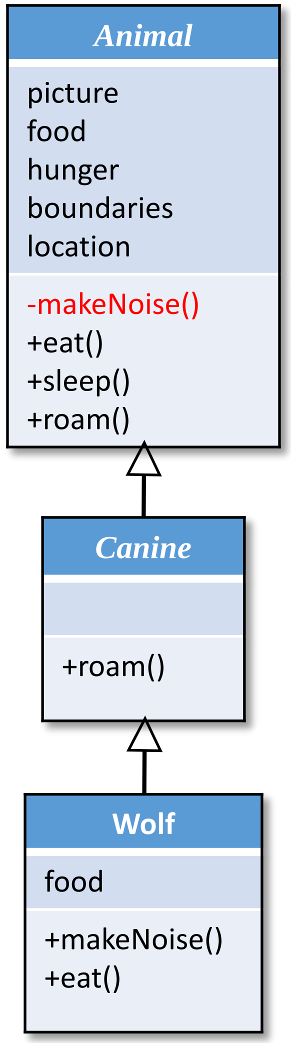
private methods overridable?

```
public abstract class Animal {
      ...
      public void sleep() {
            makeNoise();
      }
      ...
}

...
Wolf w = new Wolf();
w.sleep();
..
```



private methods

```
public abstract class Animal {
      ...
      public void sleep() {
            makeNoise();
      }
      ...
}


...
Wolf w = new Wolf();
w.sleep();
..
```

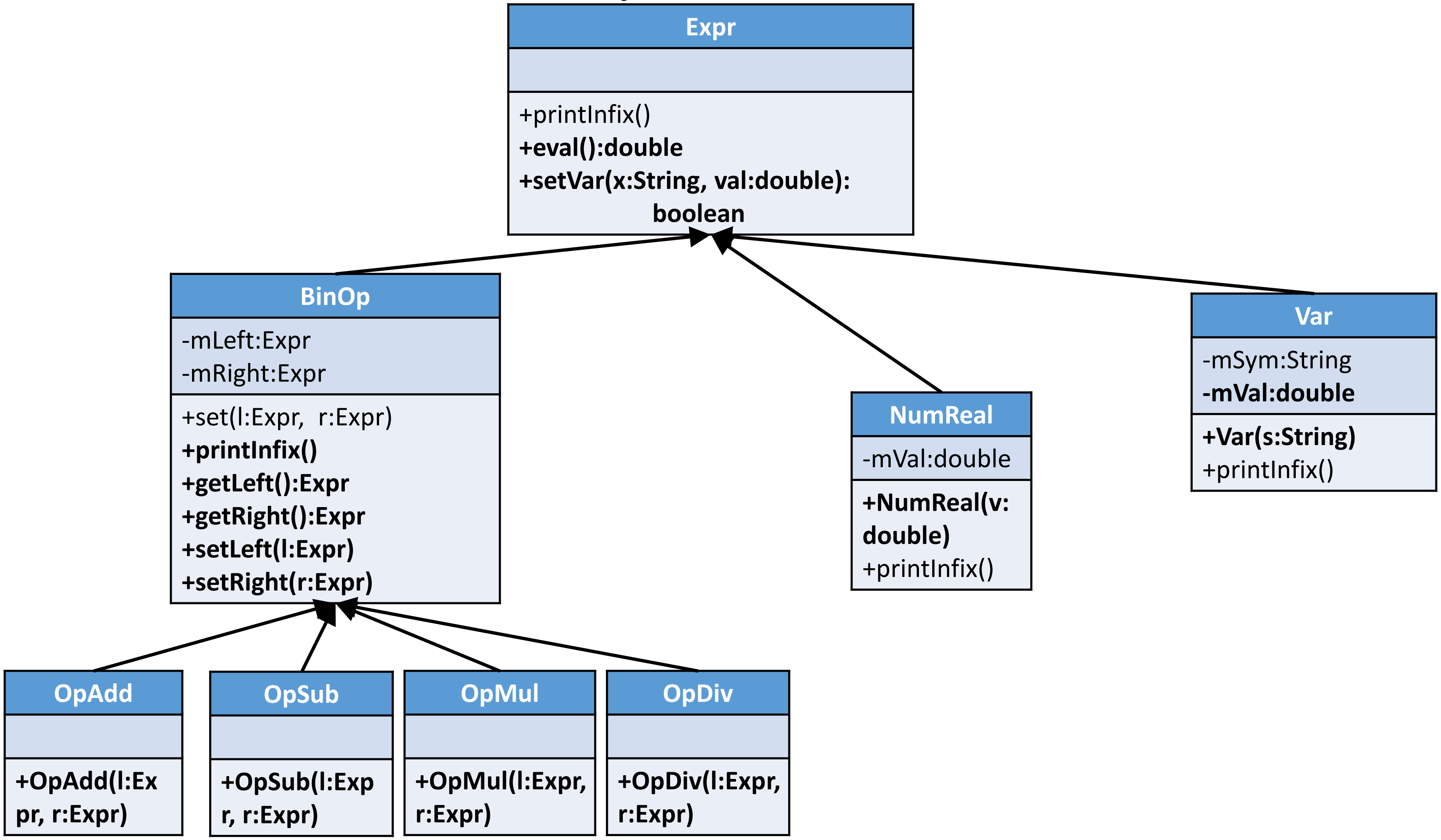


References

- Kathy Sierra and Bert Bates, *Head First Java*, O'Reilly, 2005.
- Java Tutorials
  - http://docs.oracle.com/javase/tutorial/
- Java Platform, Standard Edition 7 API Specification
  - http://docs.oracle.com/javase/7/docs/api/

Q&A

Class Hierarchy


Expr
+printInfix()
+eval():double
+setVar(x:String, val:double): boolean
BinOp
-mLeft:Expr
-mRight:Expr
+set(l:Expr, r:Expr)
+printInfix()
+getLeft():Expr
+getRight():Expr
+setLeft(l:Expr)
+setRight(r:Expr)
NumReal
-mVal:double
+NumReal(v: double)
+printInfix()
Var
-mSym:String
-mVal:double
+Var(s:String)
+printInfix()
OpAdd
+OpAdd(l:Expr, r:Expr)
OpSub
+OpSub(l:Expr, r:Expr)
OpMul
+OpMul(l:Expr, r:Expr)
OpDiv
+OpDiv(l:Expr, r:Expr)