Module 2
Introduction to OOP

Module 2

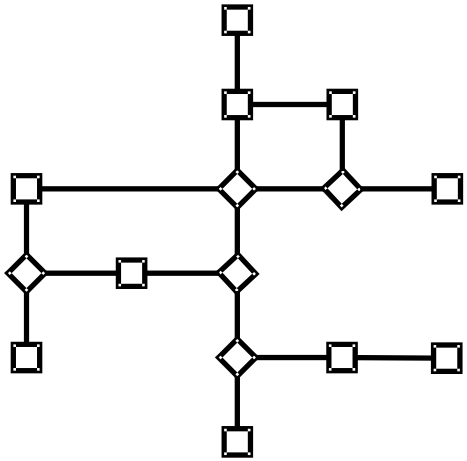
- **Programming paradigms**
- Classes and objects
- Inheritance
- Polymorphism
- Encapsulation
- OOP goals

Programming paradigms

- A programming paradigm is a collection of ideas and concepts defining a style of computer programming

- Structured
- Logic
- Functional
- Object-oriented
- Aspect-oriented
- etc.

Structured programming



- Structured programming is a programming paradigm that describes the computation process as instructions that change program state.

- Structured program is very similar to imperative orders of a natural language, i.e. it is a sequence of instructions that a computer has to execute

- Example – C language.

Logic programming

- Logic programming is programming based on formal logic.
- Logic programming languages usually specify what has to be computed, rather than the way the computation must be done
- Example - Prolog

Finding the factorial:

```
fact(1,1).

fact(N,F) :- N>1, N1 is N-1,
fact(N1,F1), F is F1*N.

?- fact(5,X).

X=120
```

Functional programming

- Functional programming is a programming paradigm that treats computation results as the evaluation of mathematical functions.

- Any functions is a superposition of other functions.

- Example: Lisp, Haskel, Closure.
- Support for functional programming: Java 8

- OOP is a programming paradigm whose basic concepts are objects and classes

- OOP appeared as the result of procedural programming evolution, when it was suggested to combine data and methods into OOP classes

- OOP is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class

OOP

- OOP is the most common programming paradigm

- Java is a fully OOP language, in other words, it doesn't support procedural programming style

The three most important words in OOP

Module 2

- Programming paradigms
- **Classes and objects**
- Inheritance
- Polymorphism
- Encapsulation
- OOP goals

Classes and objects

- A class is an OOP language structure that represents a prototype used for creating instances of the class.

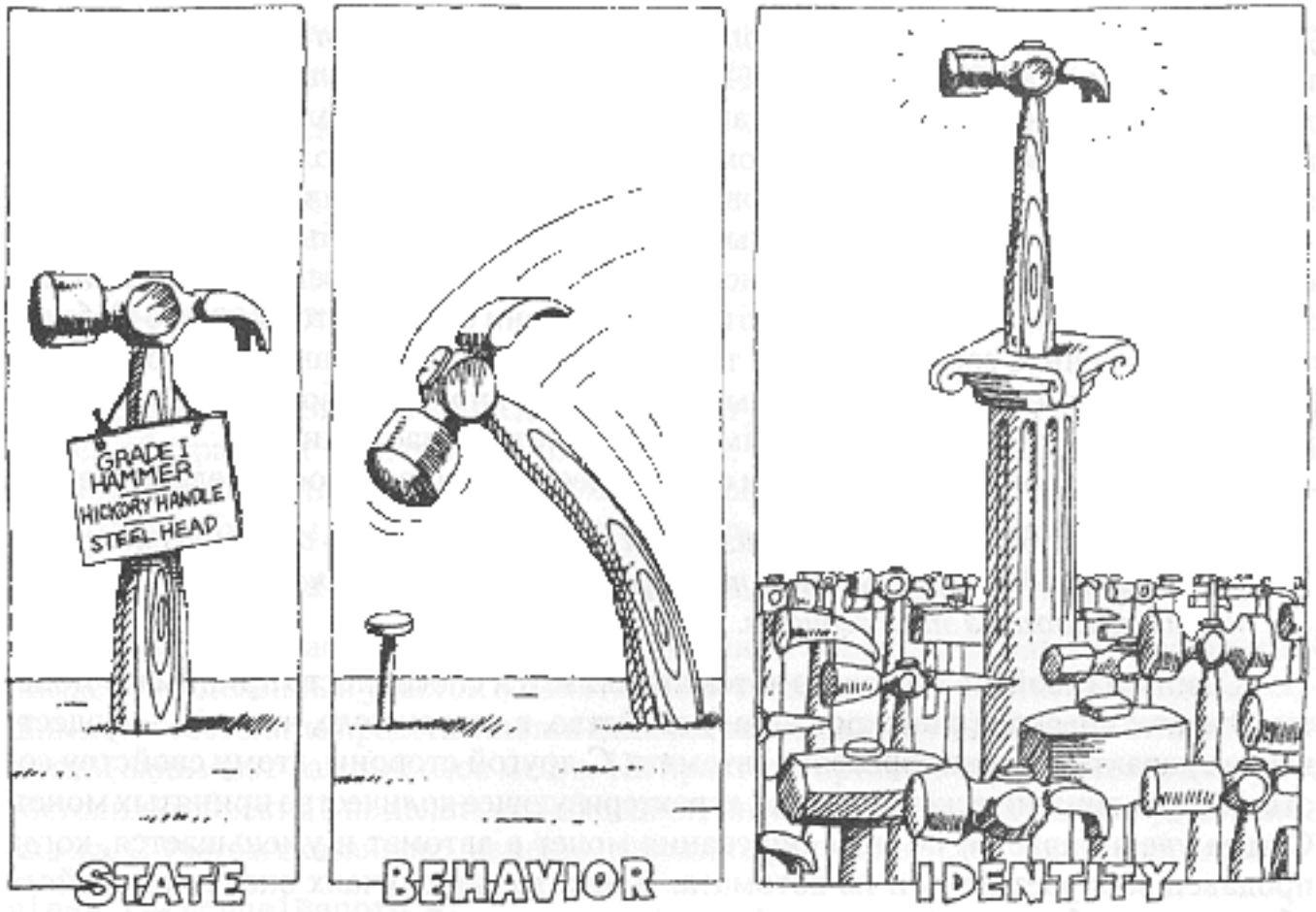
- An object is a tangible entity that exhibits some well-defined behavior and plays an important role in the domain.

Grady Booch

- An object may receive messages

Classes and objects

- What is object and what is not?



Classes and objects

- A class encapsulates state and behavior of real world object (concept) it is modeling.

- A class encapsulates the state with the help of data which is called attributes (class properties).

- As far as a class is an object prototype, when creating an object of a given class (corresponding to the prototype) a copy of data defined by classes is created in the memory.

Classes and objects

- A class encapsulates behavior with the help of code snippets that handle the state.

- This code snippets are called class methods.

- Sometimes it is convenient when class data is not duplicated at creation of each instance, but pertains to the class itself.

Module 2

- Programming paradigms
- Classes and objects
- **Inheritance**
- Polymorphism
- Encapsulation
- OOP goals

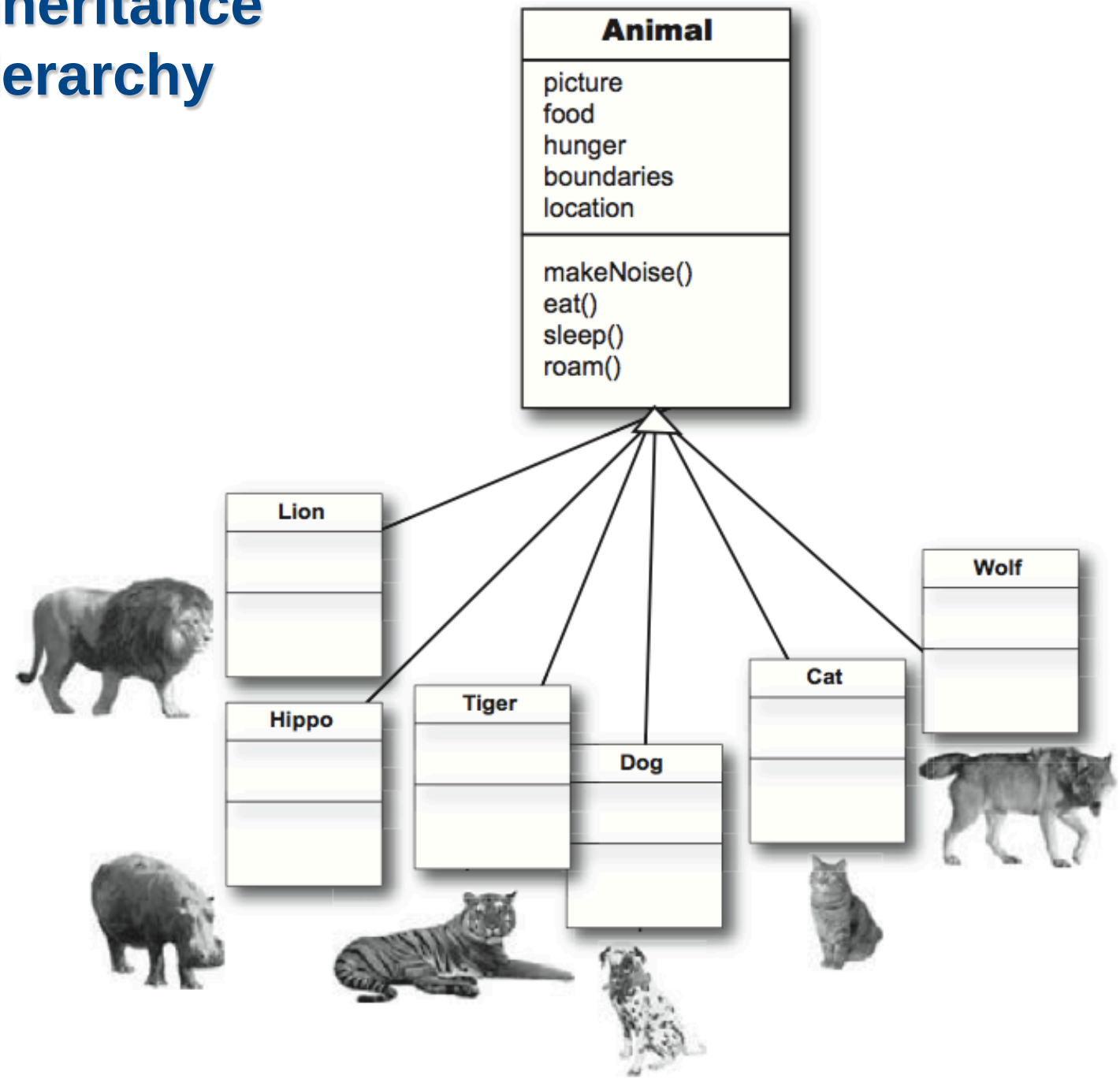
Inheritance

1. Look for objects that have common attributes and behaviors.

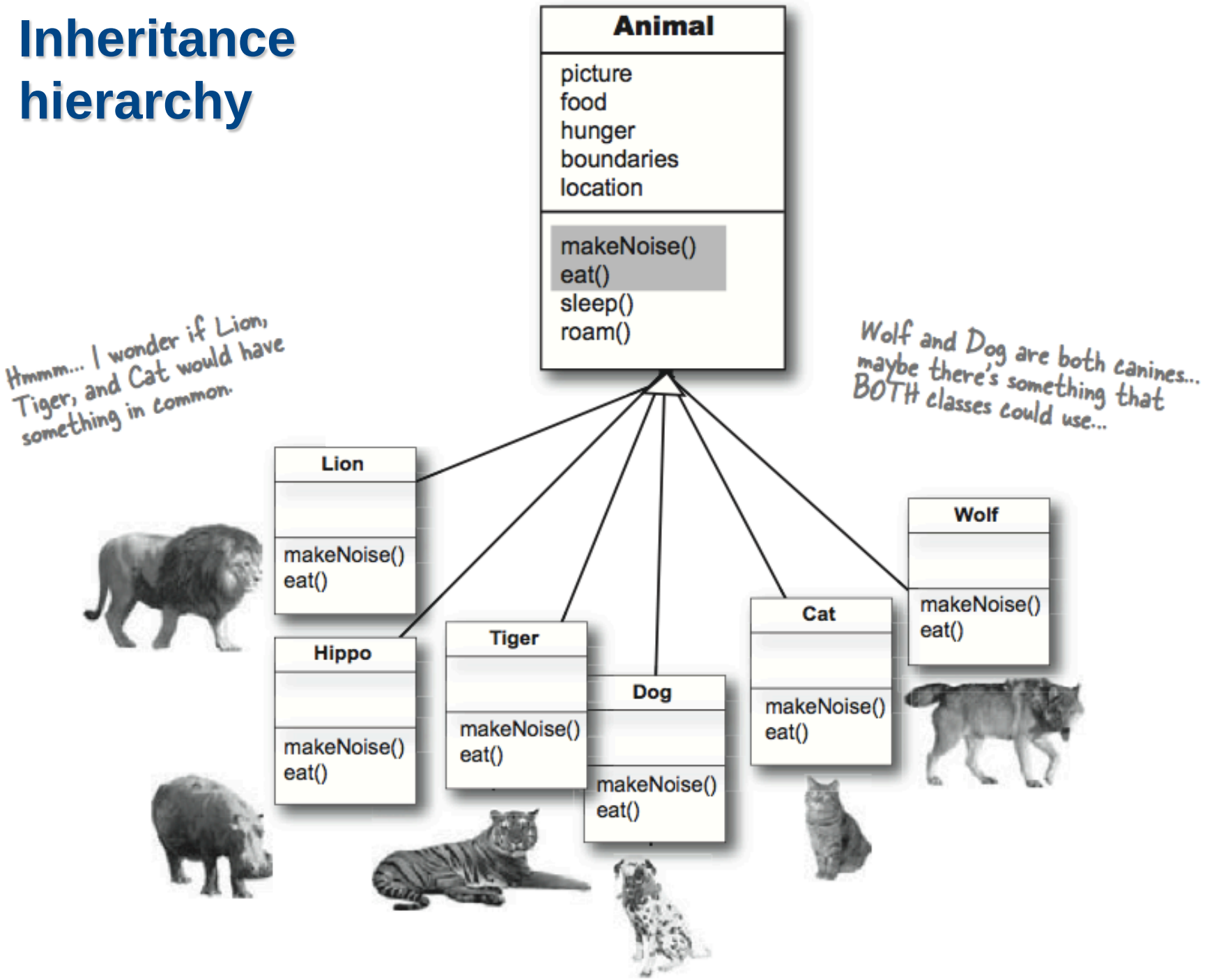
What do these six types have in common? This helps you to abstract out behaviors. (step 2)

How are these types related? This helps you to define the inheritance tree relationships (step 4-5)

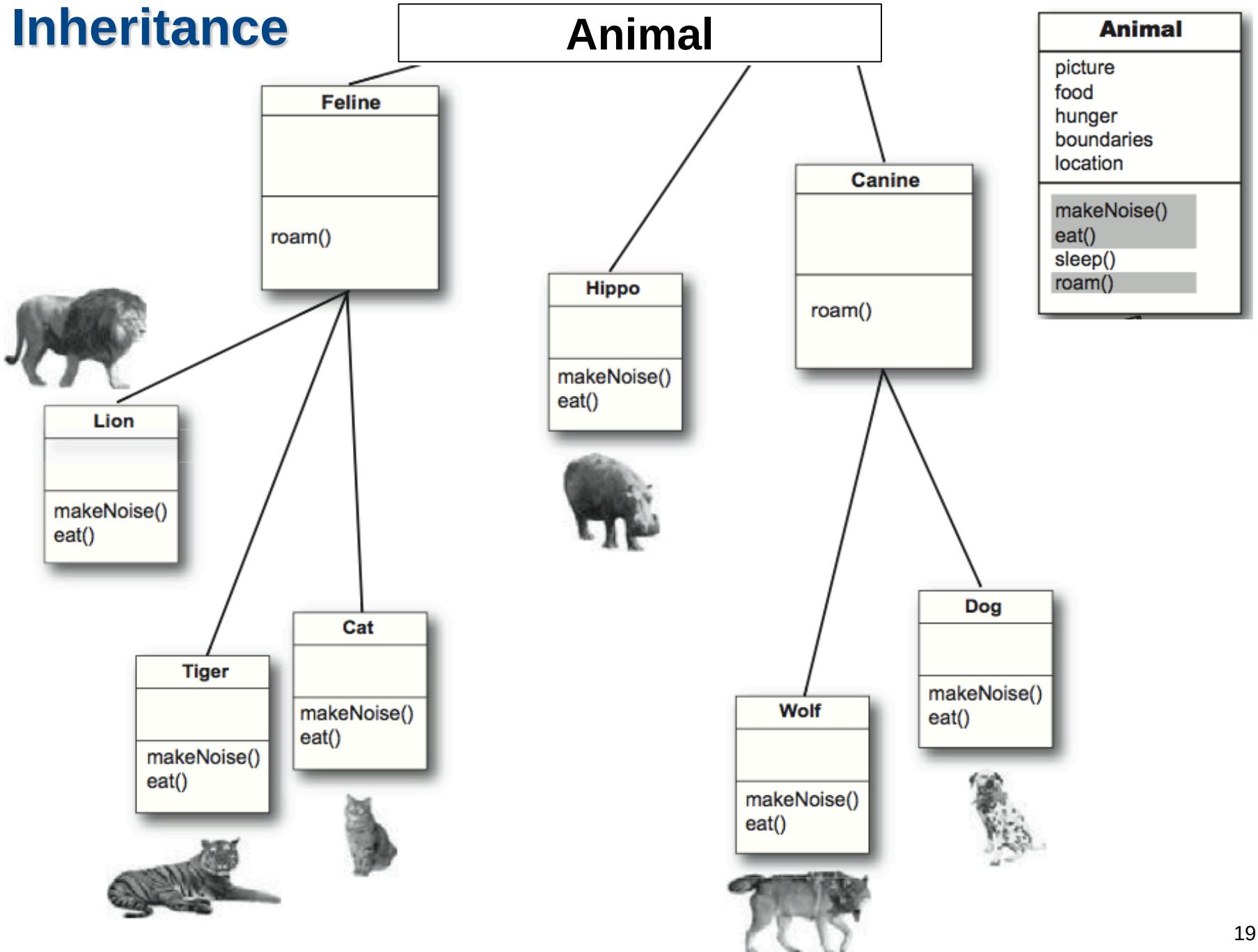
Inheritance hierarchy



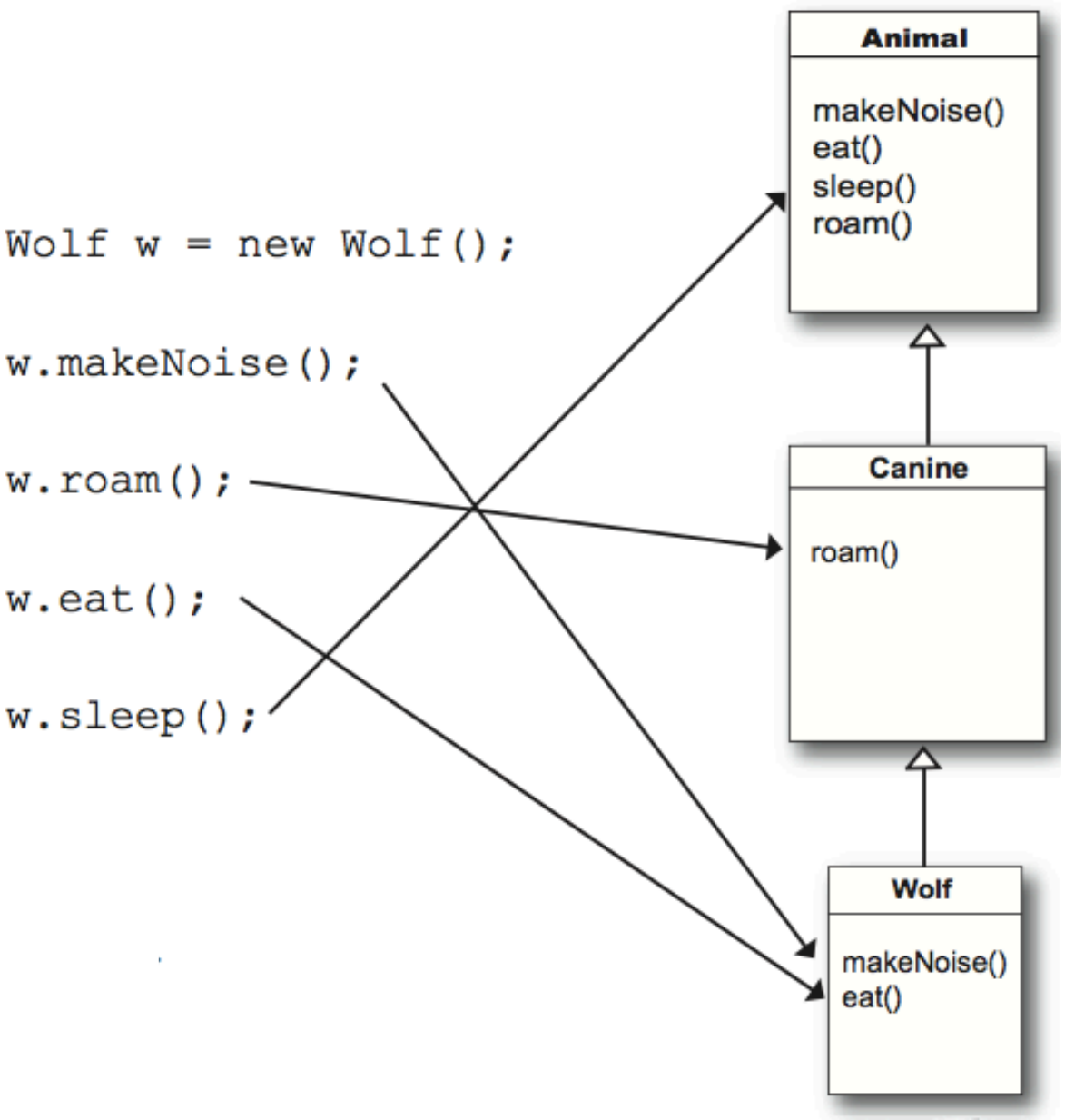
Inheritance hierarchy



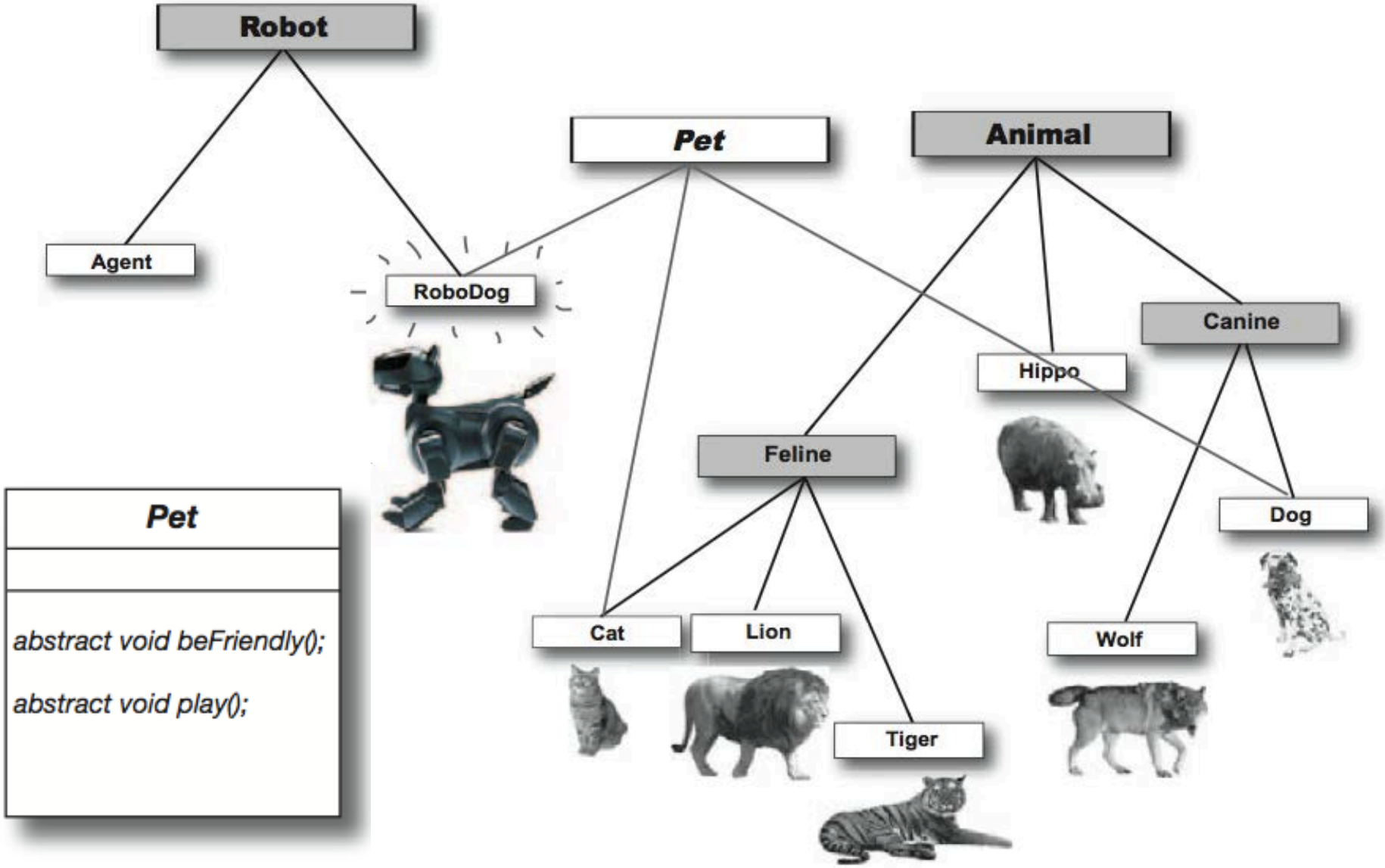
Inheritance



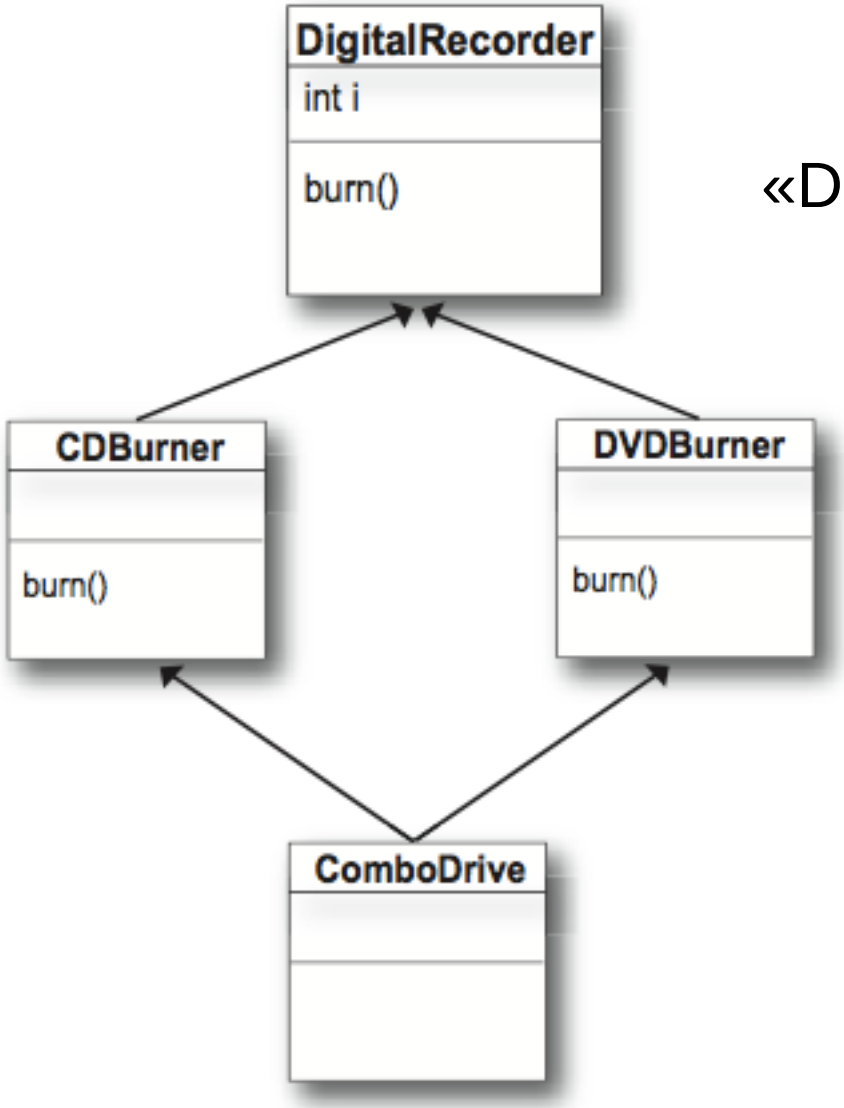
Which method is called?



Multiple inheritance

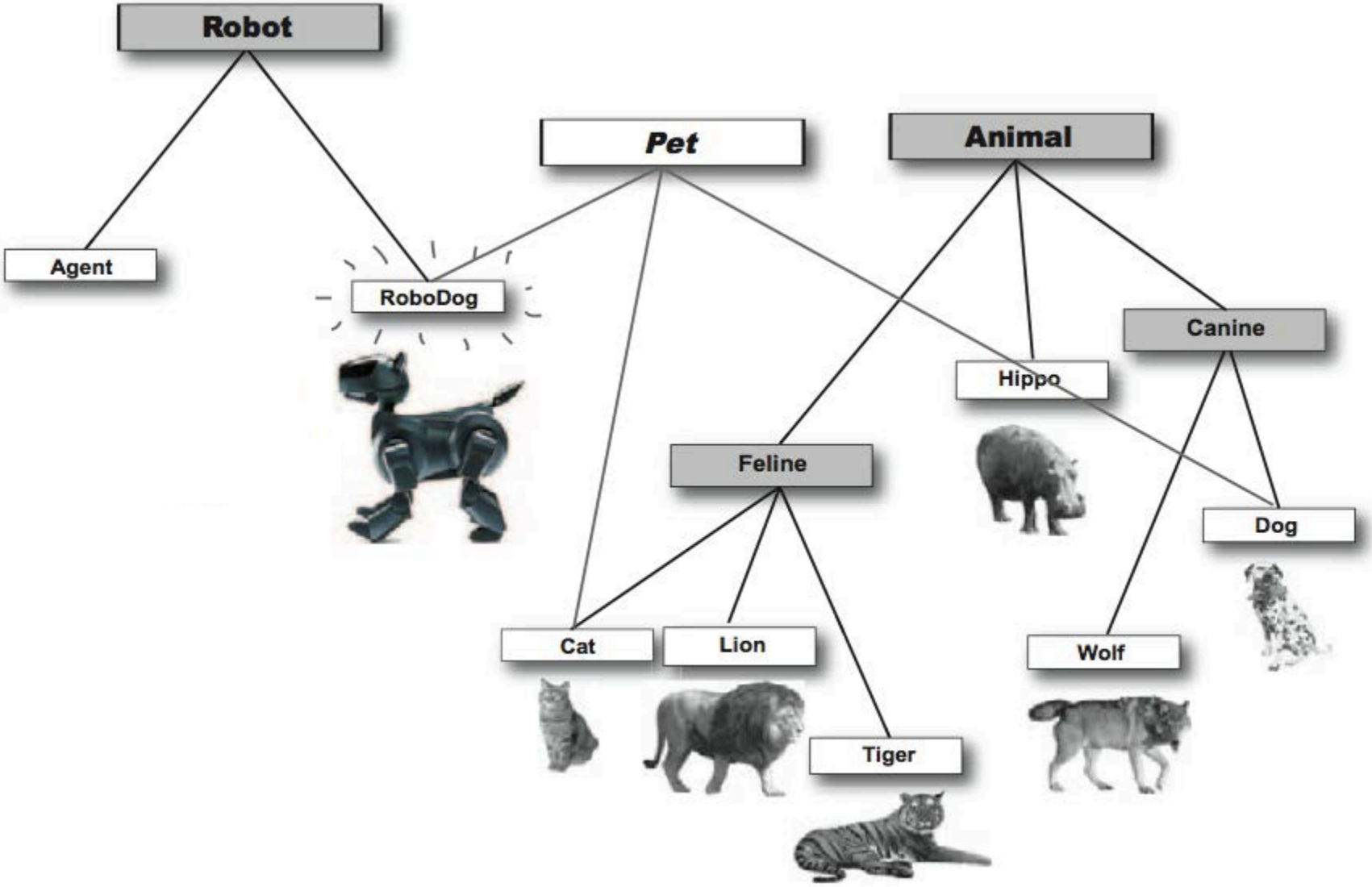


Multiple inheritance: problem



«Diamond problem»

Multiple inheritance



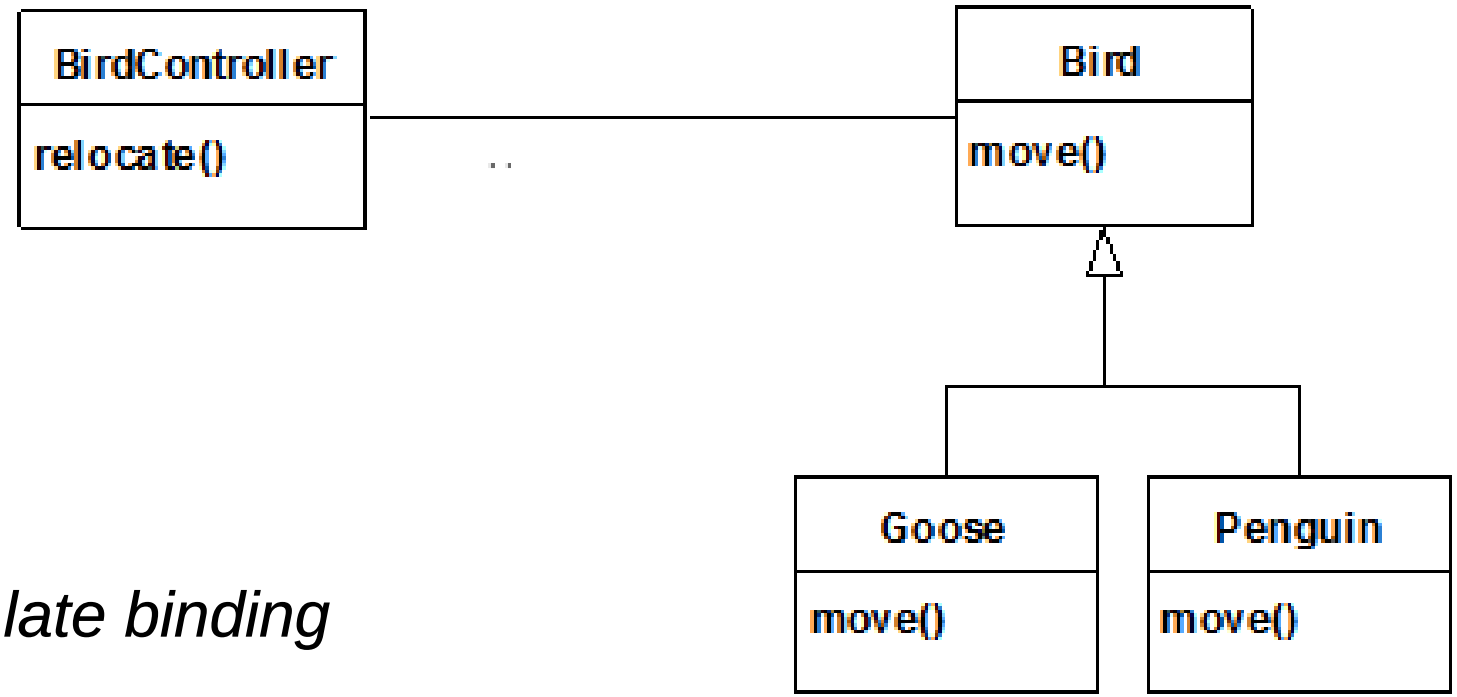

```
public class Dog extends Animal implements
Pet, Saveable, Paintable { ... }
```

Module 2

- Programming paradigms
- Classes and objects
- Inheritance
- **Polymorphism**
- Encapsulation
- OOP goals

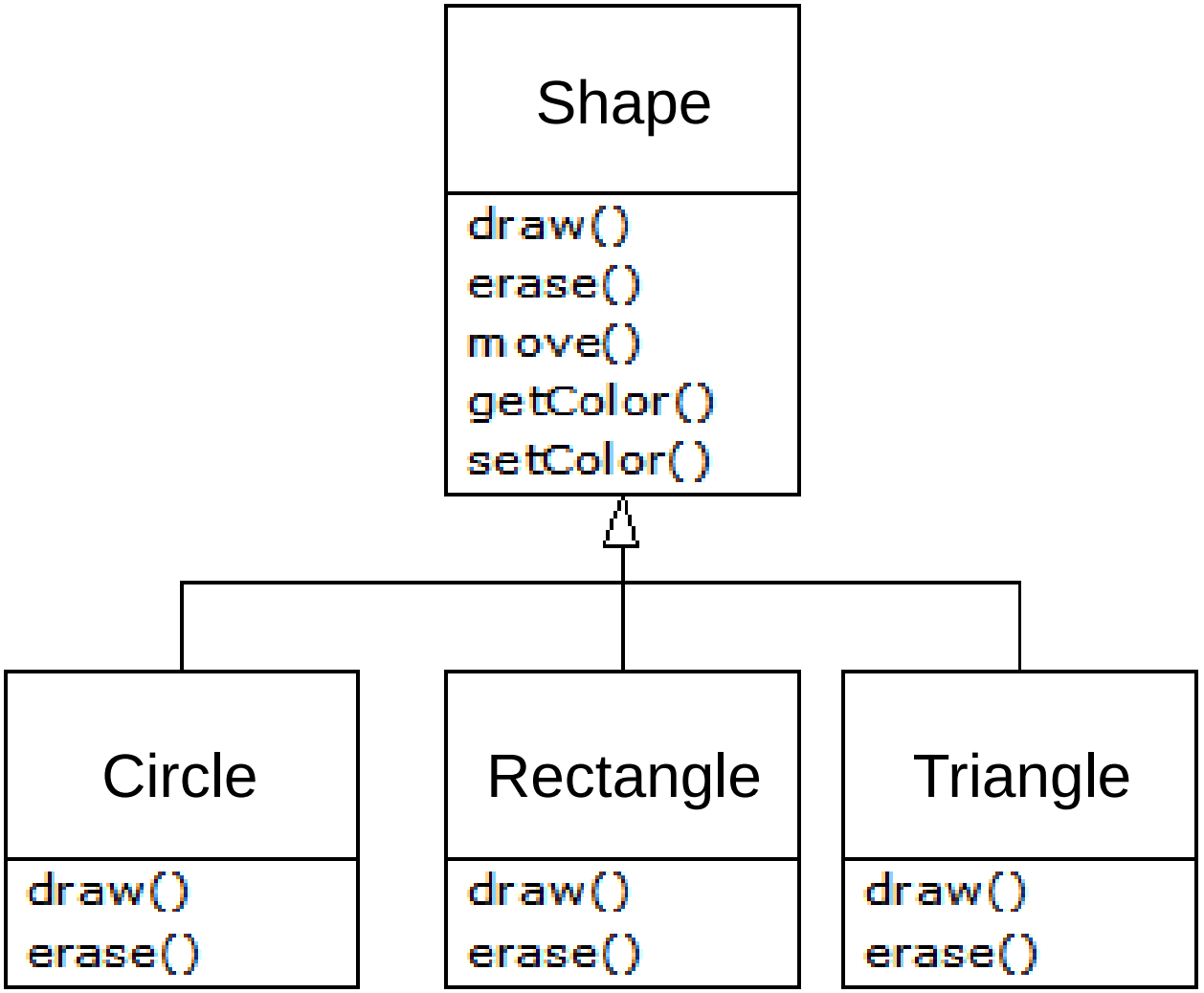
Polymorphism

- Polymorphysm is a possibility of the objects of the same type to have different behavior



- *late binding*

Polymorphism



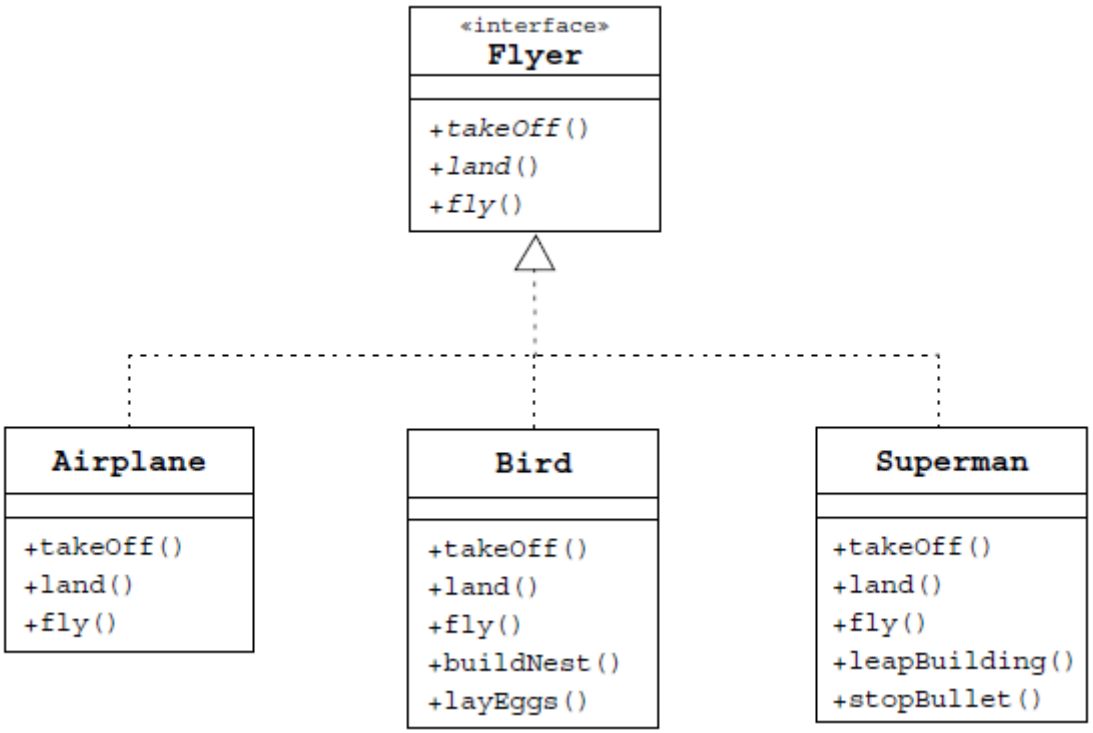
Interface

- An interface is a semantic and syntactic structure in the program code used for specifying services provided by a class or a component.

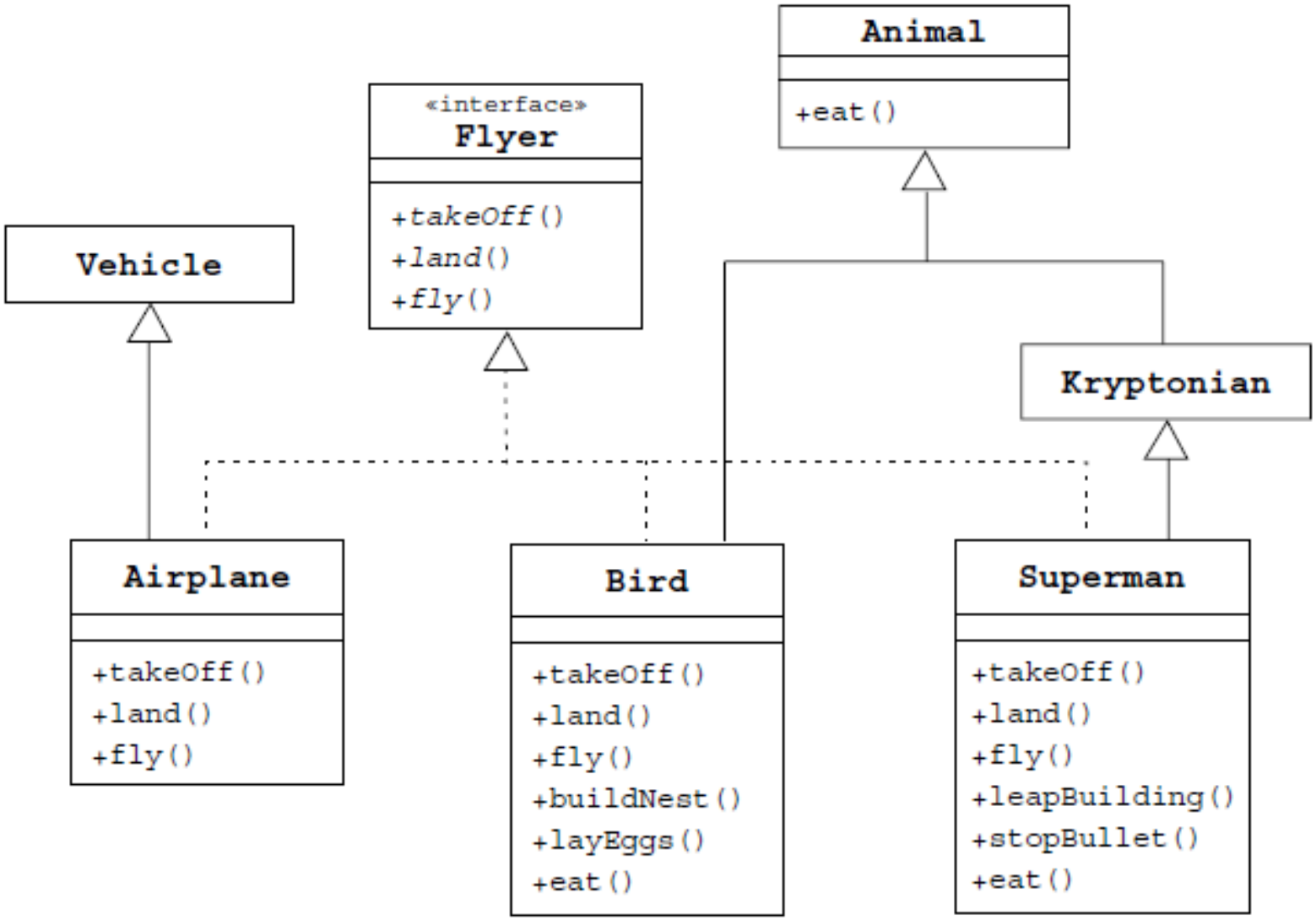
- The reason for semantic interface loading is that sometimes you should know *how* to use the object without knowing anything about its implementation.

Interfaces

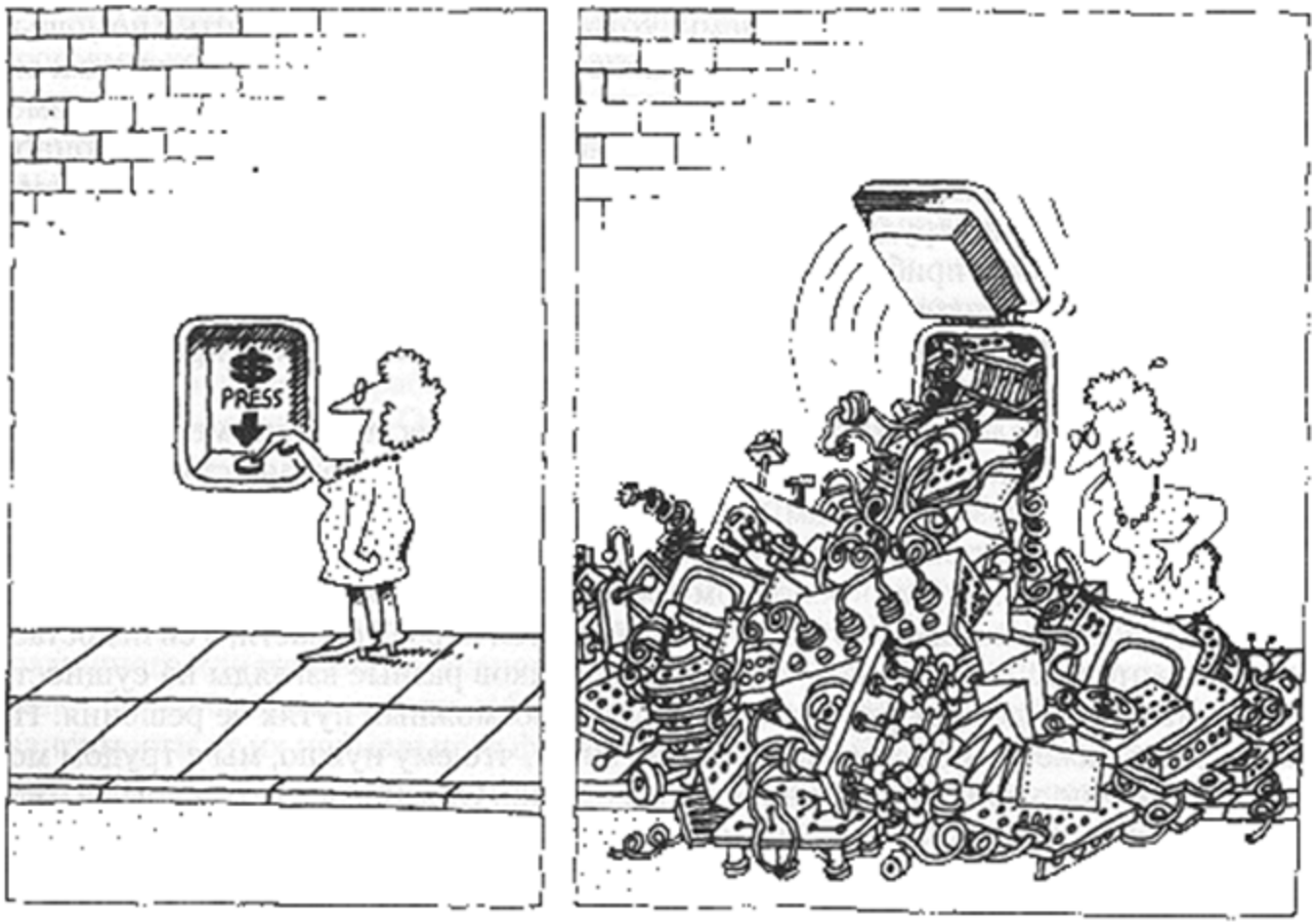
- A class may implement several interfaces.



Interfaces example



Interface



The developer's task is create an illusion of simplicity

Module 2

- Programming paradigms
- Classes and objects
- Inheritance
- Polymorphism
- **Encapsulation**
- OOP goals

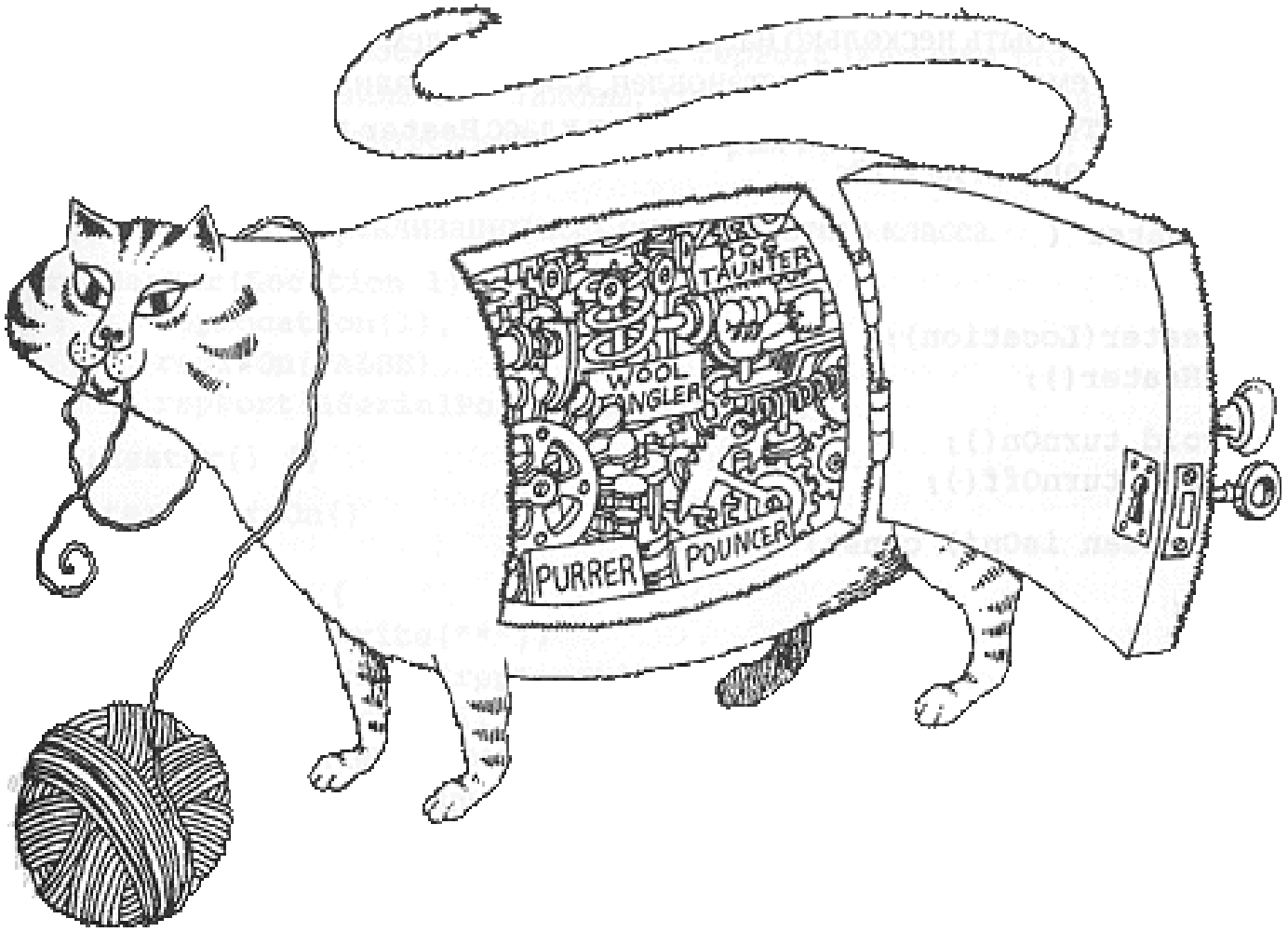
Encapsulation

- Encapsulation is a language mechanism that restricts the access to object components.

  (hiding implementation details behind allowable messages)

- Encapsulated variables can be accessed when writing class implementation.

Encapsulation



“Black box” concept

Encapsulation

Encapsulation allows not to depend on the inner structure of the object.

Examples:

- Driving a car
- Doing a job by specialist

It allows to change the inside structure and mechanisms of the object using an interface.

Encapsulation

What we can hide: fields and methods.

There are several hidden levels:

private – hide from everyone: only this class has the access

protected – hide from everyone except for this class and its descendants

public – everyone can get an access

Module 2

- Programming paradigms
- Classes and objects
- Inheritance
- Polymorphism
- Encapsulation
- **OOP goals**

OOP goals

- OOP introduces a significant number of new concepts, requires certain efforts aimed at creating a correct system design, and offers the following benefits:

  - High degree of code reuse;
  - Easy-to-understand to humans;
  - Possibility to localize code modifications;
  - Creating abstractions facilitates the understanding of applications.

Exercise

Lab guide:

- Exercise 3