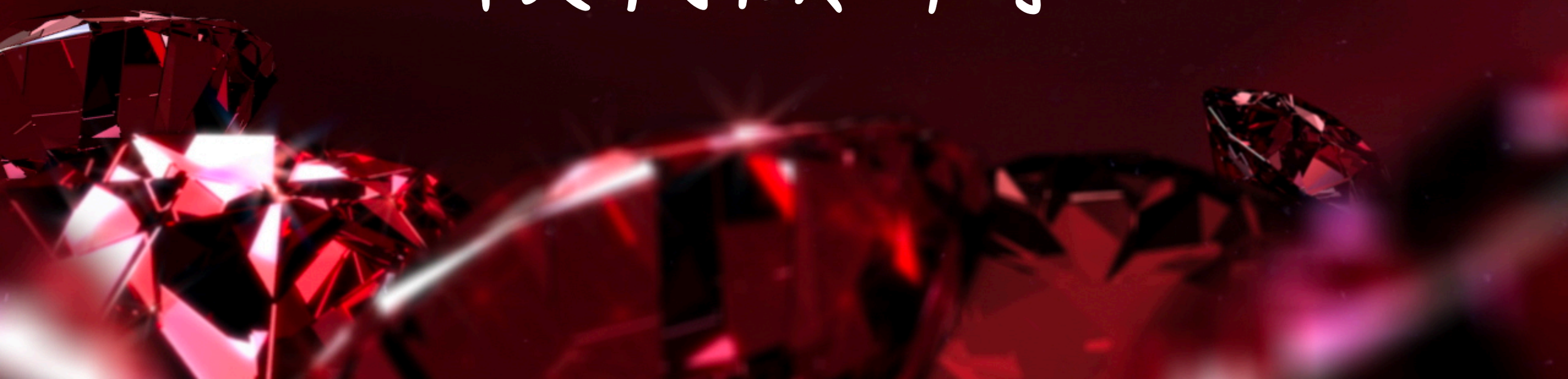
面向对象

杨亮


IDEA
升维思考

FORBIDDEN

限制做坏事



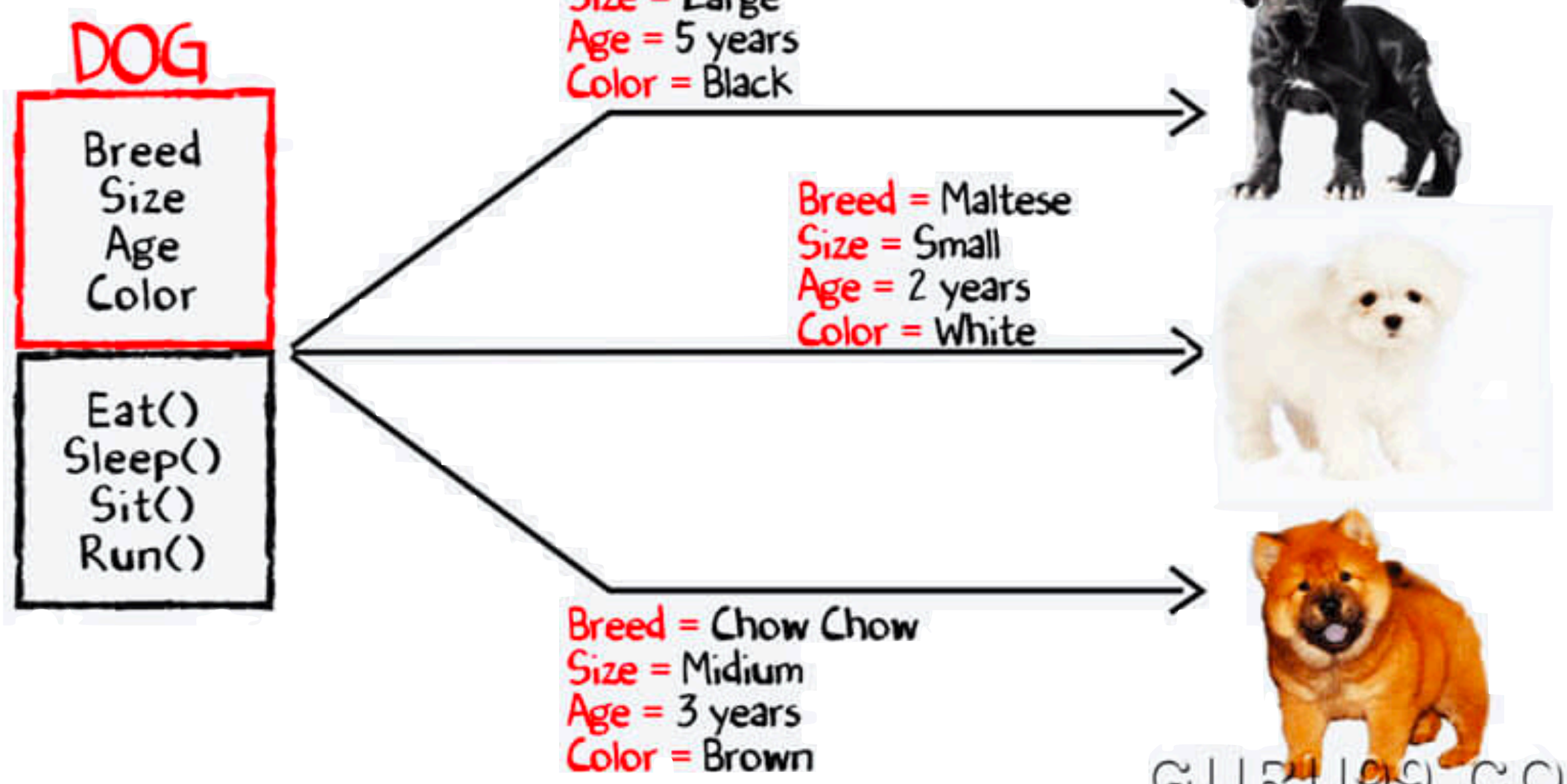
Objects

万物皆为对象

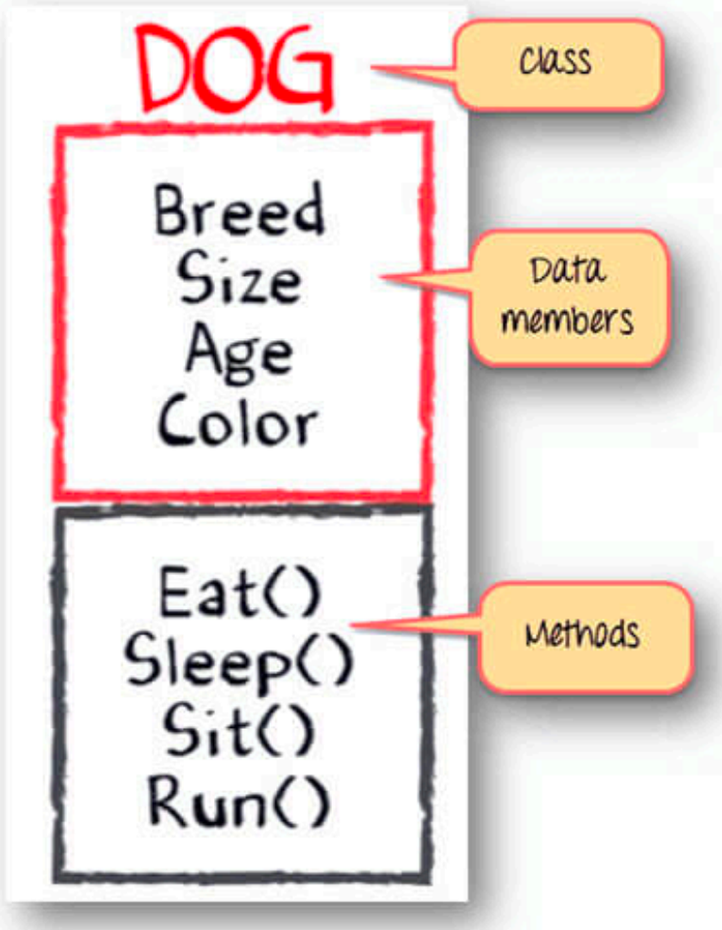
Everything is an object

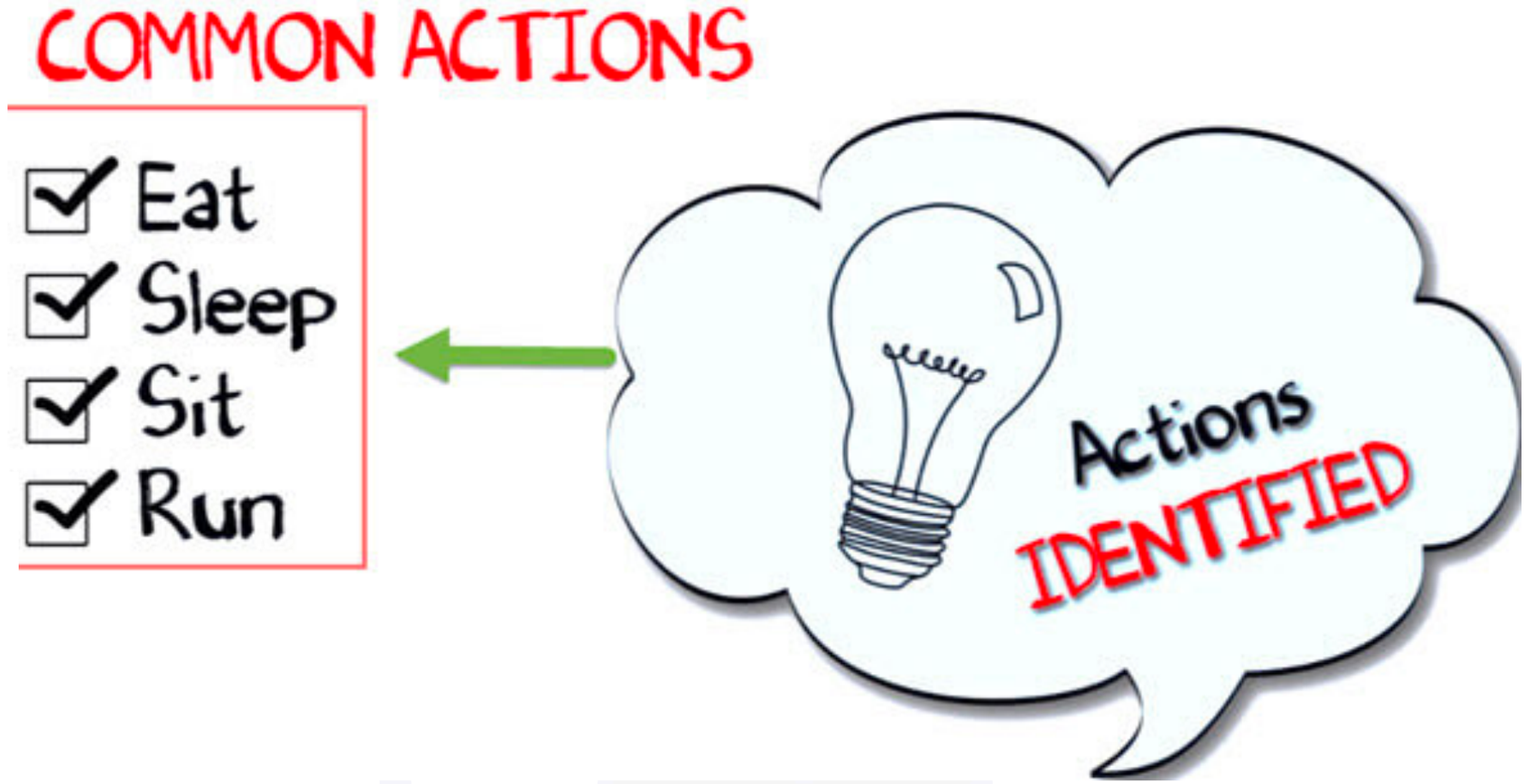
抽象 (Abstraction)

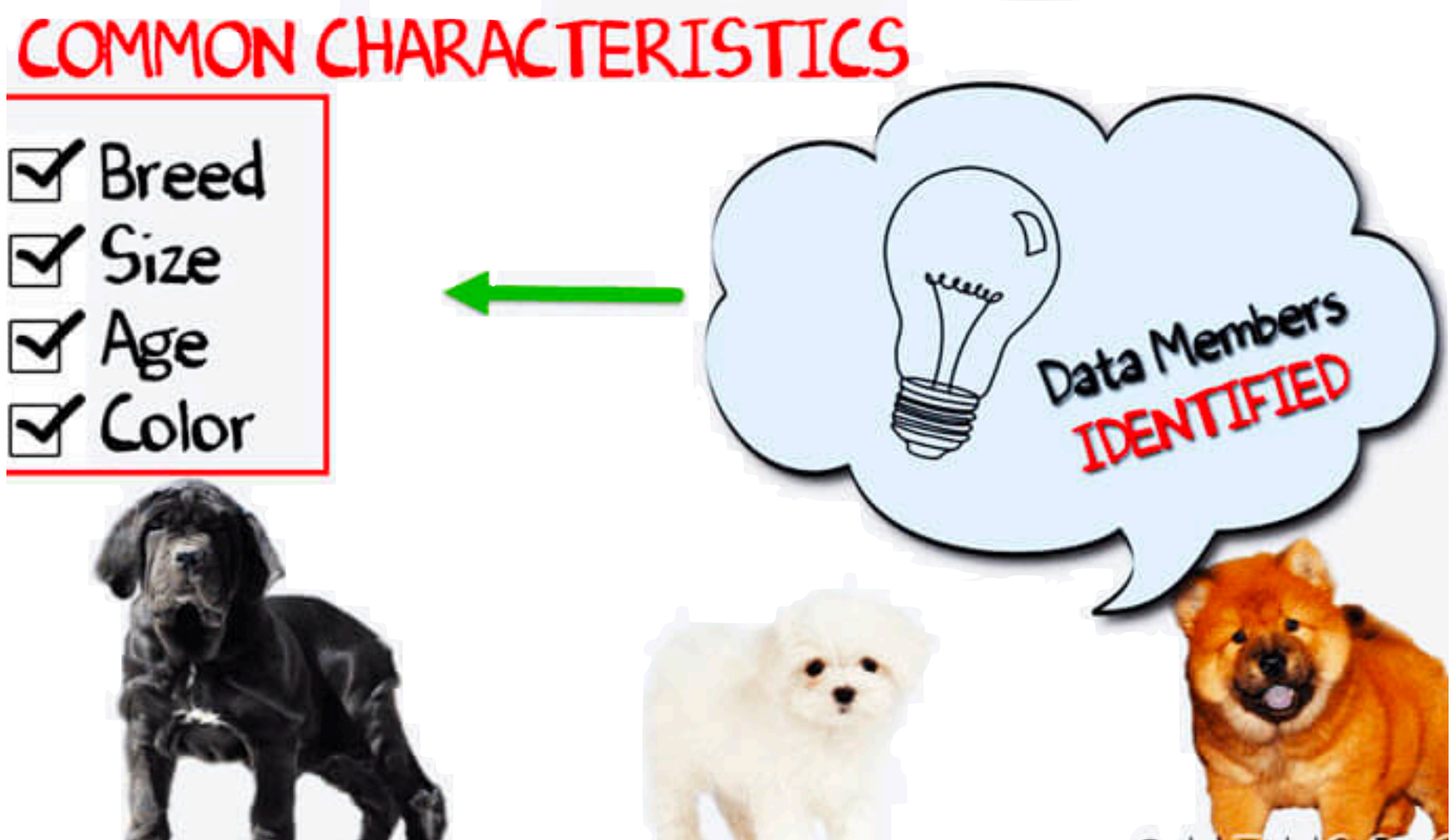
抽象特征与行为 ← 共同特征与行为 ← 真实事物



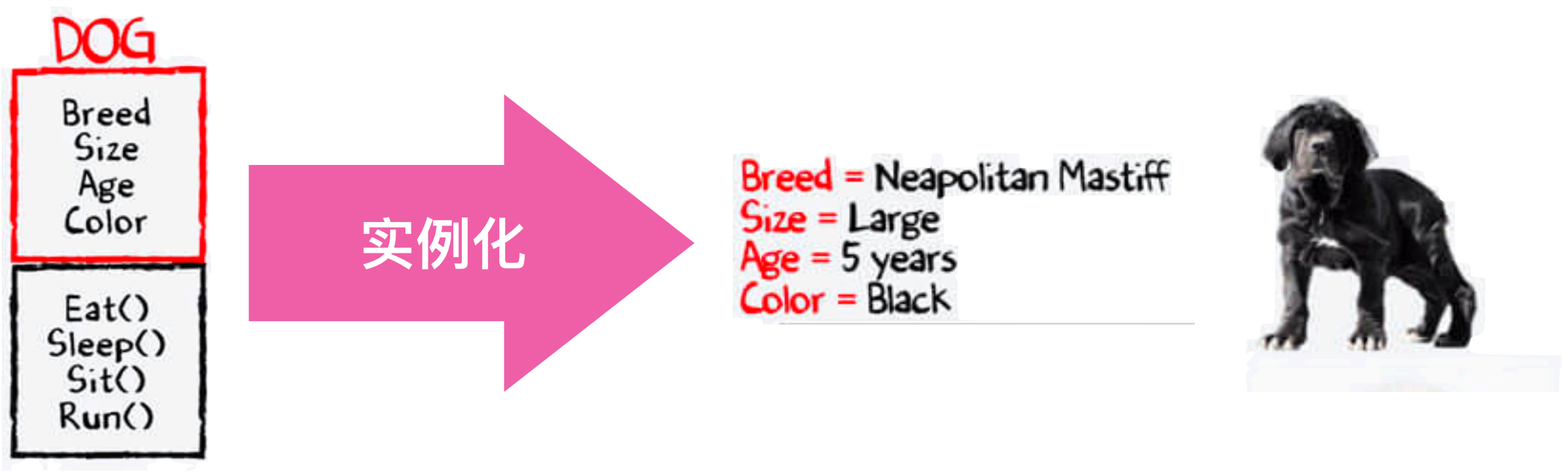
类
Class


DOG
class
Breed
Size
Age
Color
Data members
Eat()
Sleep()
Sit()
Run()
Methods


COMMON ACTIONS
Eat
Sleep
Sit
Run
Actions
IDENTIFIED


COMMON CHARACTERISTICS
Breed
Size
Age
Color
Data Members
IDENTIFIED

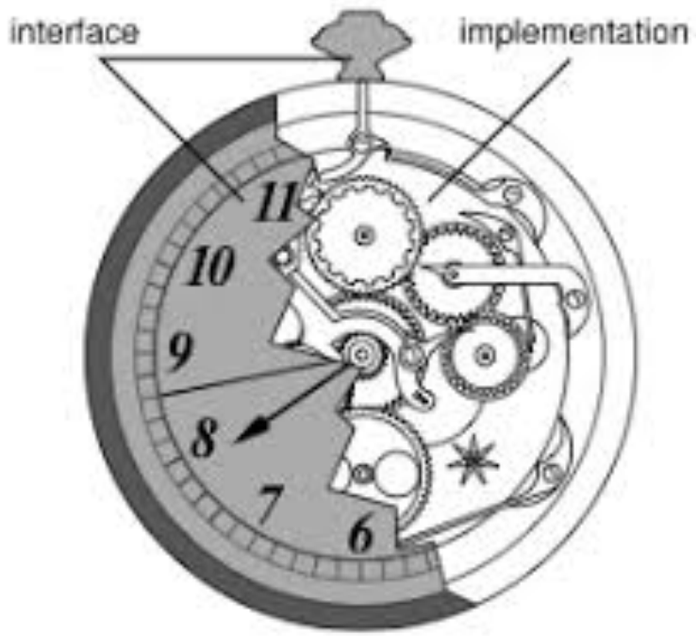
分析与设计



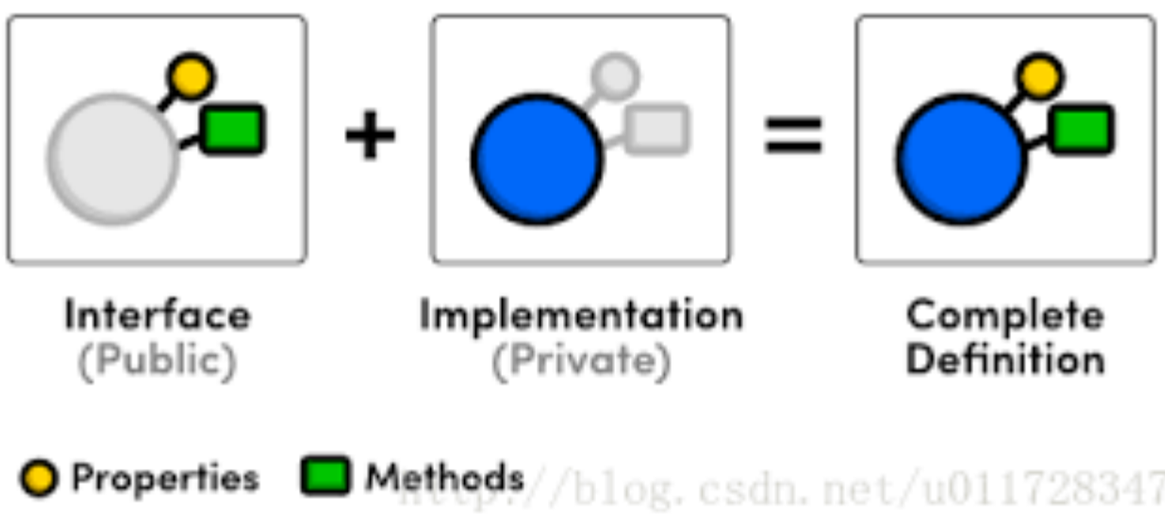
分析 设计与实现

类的定义（声明+实现）

[修饰符] class 类名
{
    成员变量的声明;
    成员方法的声明及实现;
}



[修饰符] 变量类型 变量名;
[修饰符] 返回值类型 方法名([参数列表])
{
    语句序列;
    [return [返回值]];
}



Example

```
public class MyDate
{
    int year, month, day;

    void set(int y, int m, int d)
    {
        this.year = y;
        this.month = m;
        this.day = d;
    }

    public String toString()
    {
        return this.year+”年”+this.month+”月"+this.day+"日";
    }
}
```

对象的声明和实例化

声明：　类名 对象名;

实例化：　对象名=new 类名([参数列表])

使用：　对象名.成员变量

　　对象名.成员方法([参数列表])

类定义中
用this代替对象名
通常省略?!!!

```
MyDate d1;
d1 = new MyDate();
System.out.println("d1:"+d1.toString());
d1.set(2012,1,1);
MyDate d2 = d1;
System.out.println("d1:"+d1.toString()+", d2:"+d2.toString());
d2.month = 10;
System.out.println("d1:"+d1+", d2:"+d2);
d2 = new MyDate();
```

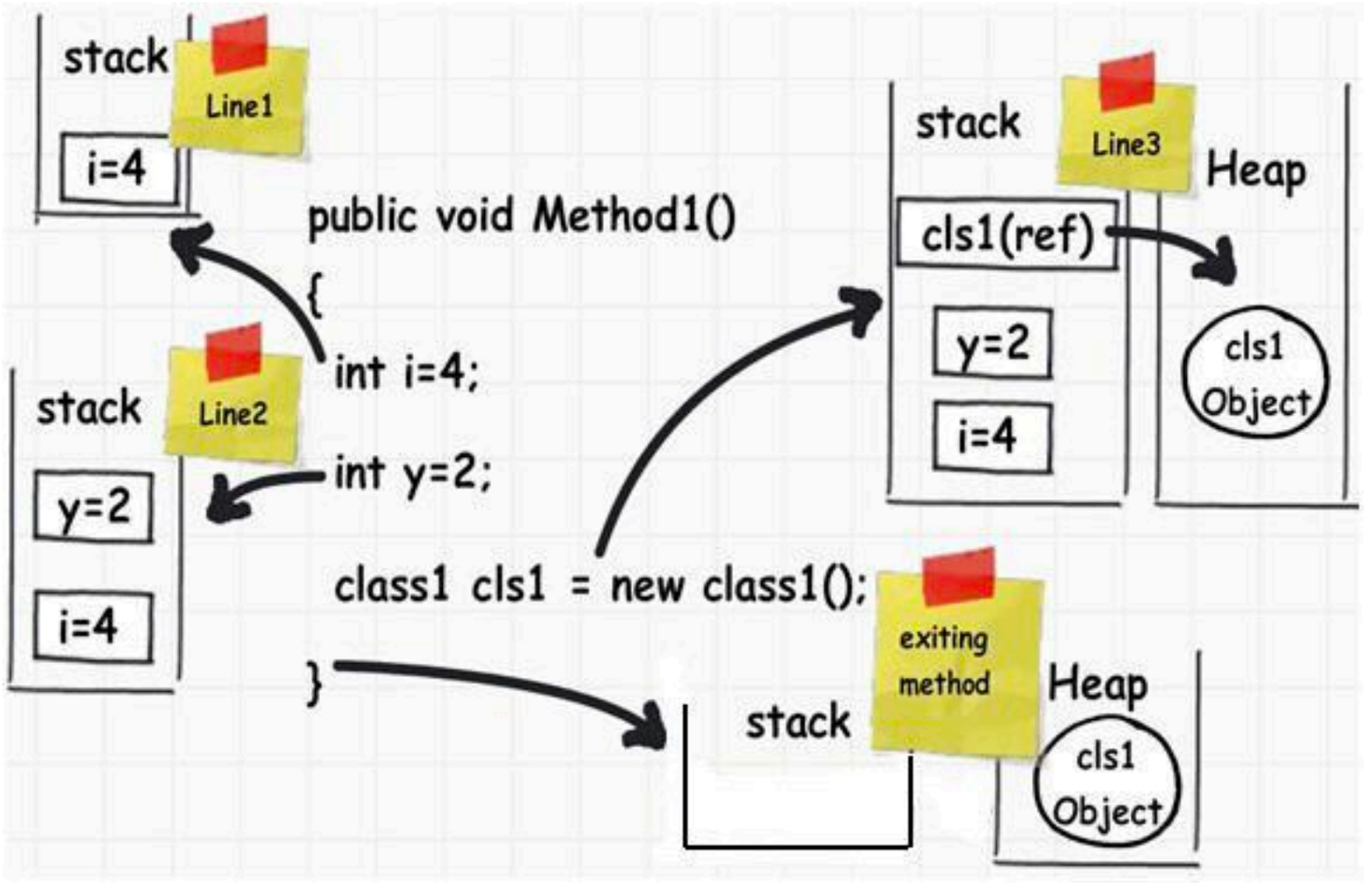
堆 vs. 栈

栈中分配的是基本类型和自定义对象的引用。

堆中分配的是对象，也就是new出来的东西。

参数传递、赋值和比较中只针对栈中的内容（值）

```
public void Method1()
{
    int i = 4;
    int y = 2;
    class1 cls1 = new class1();
}
```



```
MyDate d1;
d1 = new MyDate();
System.out.println(“d1:”+d1.toString());
d1.set(2012,1,1);
MyDate d2 = d1;
System.out.println(“d1:”+d1.toString()+”, d2:”+d2.toString());
d2.month = 10;
System.out.println(“d1:”+d1+”, d2:”+d2);
d2 = new MyDate();
```

画出栈和堆的变化情况

—类特殊的成员函数

```
public class MyDate
{
    int year, month, day;

    void set(int y, int m, int d)
    {
        this.year = y;
        this.month = m;
        this.day = d;
    }

    public MyDate(int year, int month, int day)
    {
            this.set(year, month, day);
    }

    public String toString()
    {
        return this.year+”年”+this.month+”月"+this.day+"日"
   }
}
```

构造函数

函数名与类名相同

在创建对象时调用

函数重载

功能相同、函数名相同、参数不同

```
public void set(int year, int month, int day)
{
    this.year = year;
    this.month = (month>=1 && month<=12) ? month : 1;
    this.day = (day>=1 && day<=31) ? day : 1;
}

public void set(MyDate d)
{
    this.set(d.year, d.month, d.day);
}
```

重载
Overload

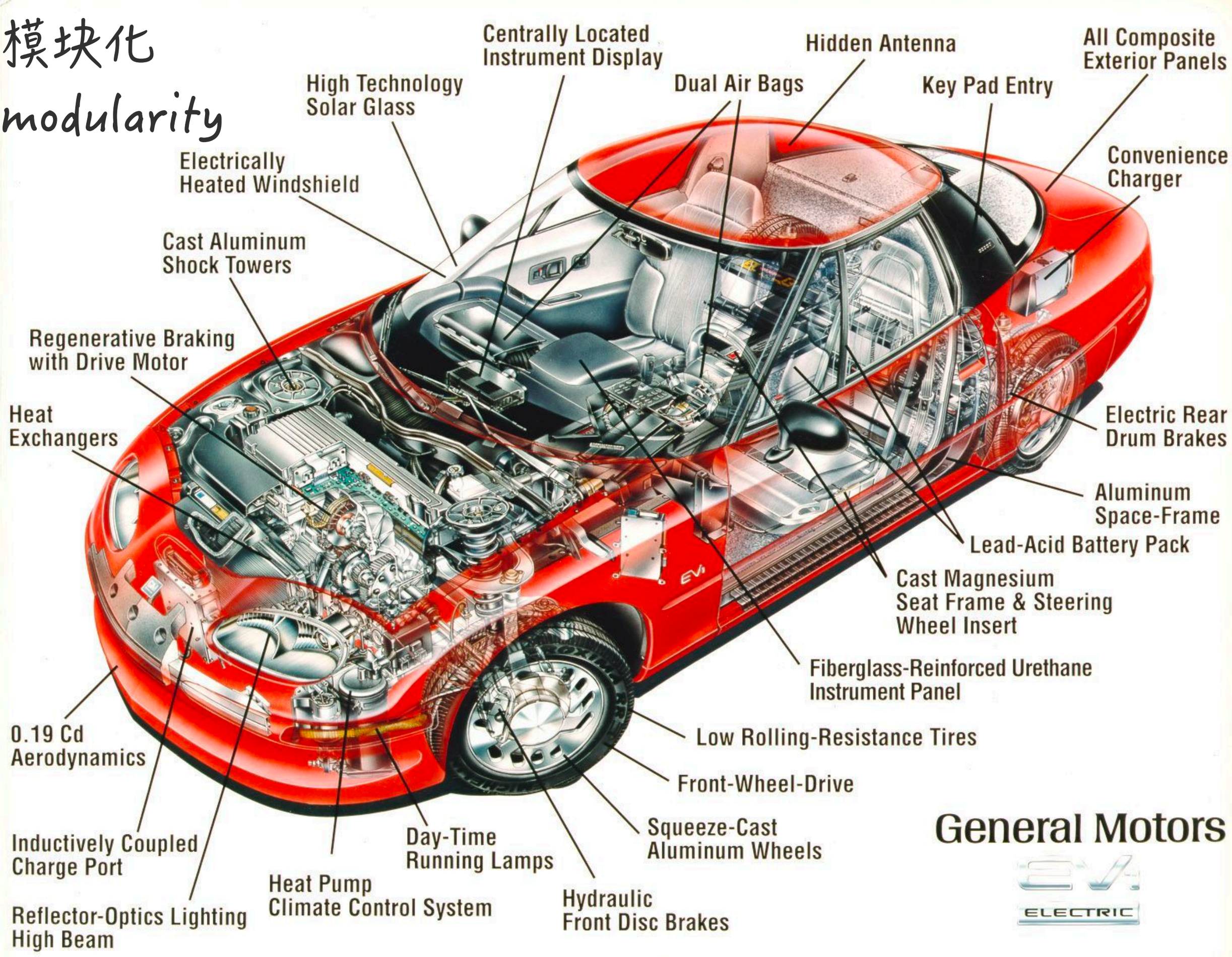
构造函数重载

```
public MyDate()
{
    Calendar today = java.util.Calendar.getInstance();
    this.year = today.get(Calendar.YEAR);
    this.month = today.get(Calendar.MONTH)+1;
    this.day = today.get(Calendar.DAY_OF_MONTH);
}

public MyDate(int year, int month, int day)
{
    this.set(year, month, day);
}

public MyDate(MyDate d)
{
    this.set(d);
}
```

模块化
modularity

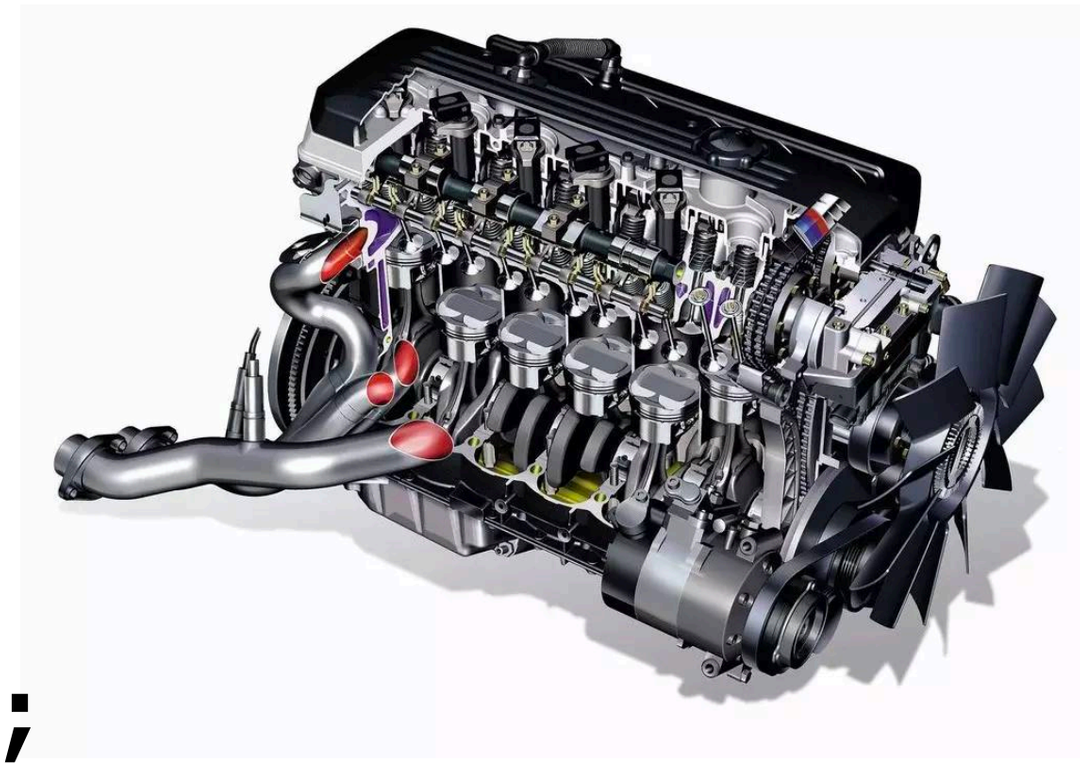


程序就是对象的集合

通过对象之间的相互协作实现特定功能

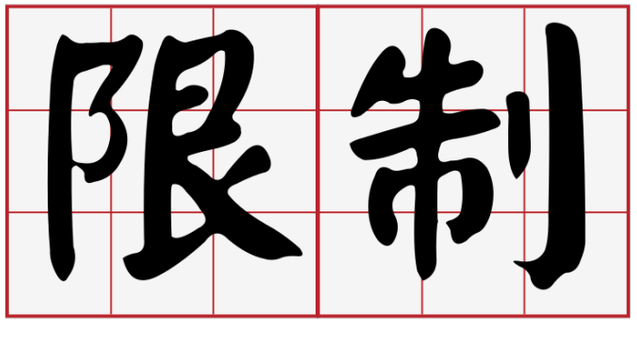
封装性—隐藏实现细节

[修饰符] class 类名
{
　　成员变量的声明;
　　成员方法的声明及实现;
}



[修饰符] 变量类型 变量名;

[修饰符] 返回值类型 方法名([参数列表])
{
　　语句序列;
　　[return [返回值]];
}



访问权限

先确定类是否有权限，再确定成员是否有权限

共有 public　　**包内 default**　　**私有 private ***

宽松 ⟶ 严格

Modifier	Class	Package	World
public	Y	Y	Y
no modifier	Y	Y	X
private	Y	X	X

* 普通的类不能直接定义为私有

Package & Files

代码

myobject

Person.java — public Class Person {}

Car.java — public Class Car {} | Class Wheel {}

Other.java — Class Cat {} | Class Dog {} | Class Apple {}

二进制码

myobject

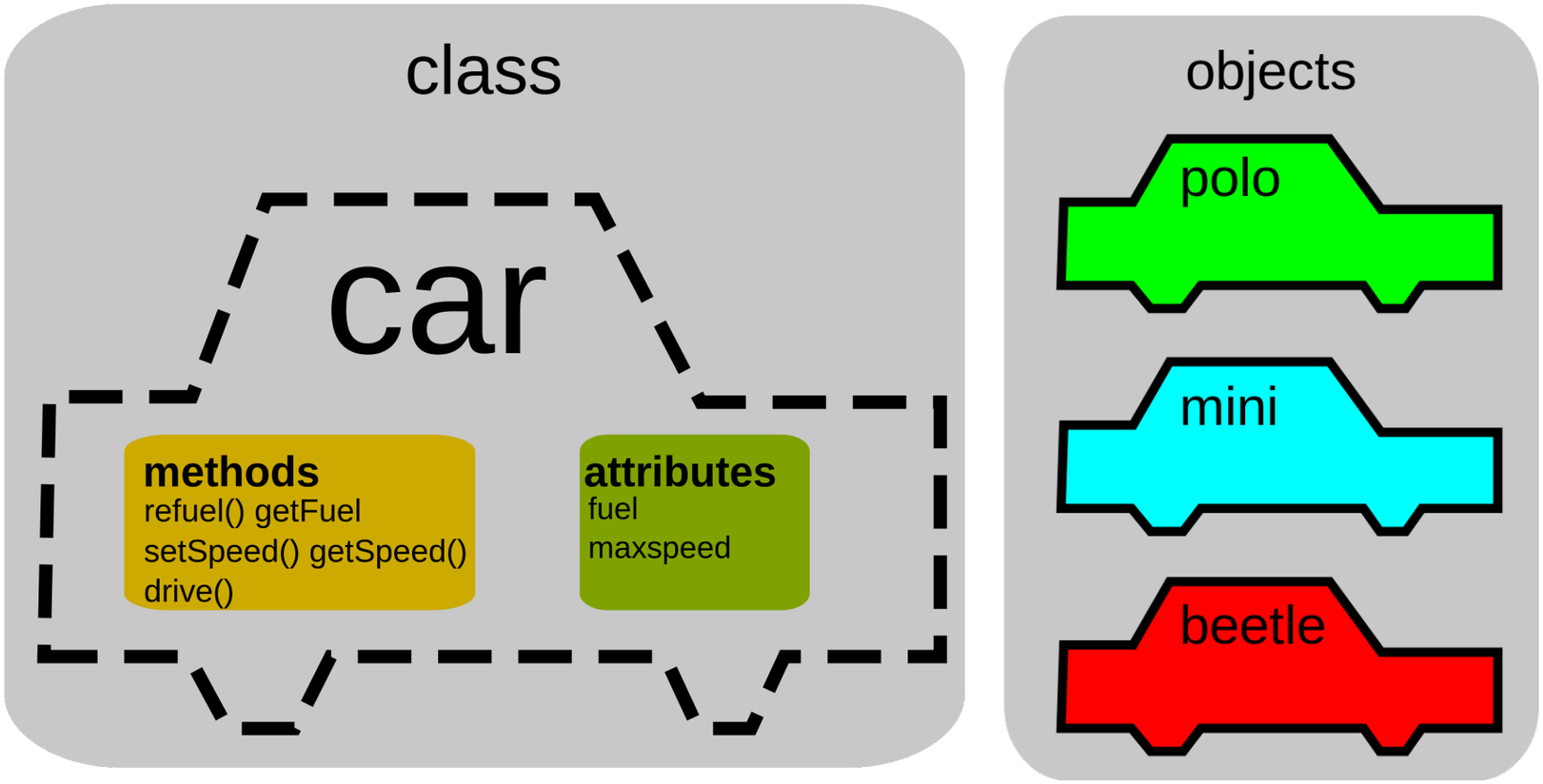
Car.class | Person.class | Cat.class

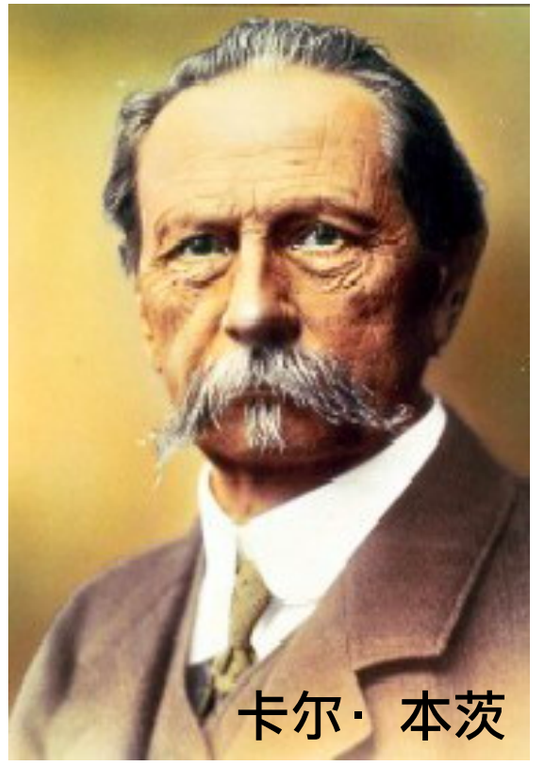
Dog.class | Apple.class | Wheel.class

Objects

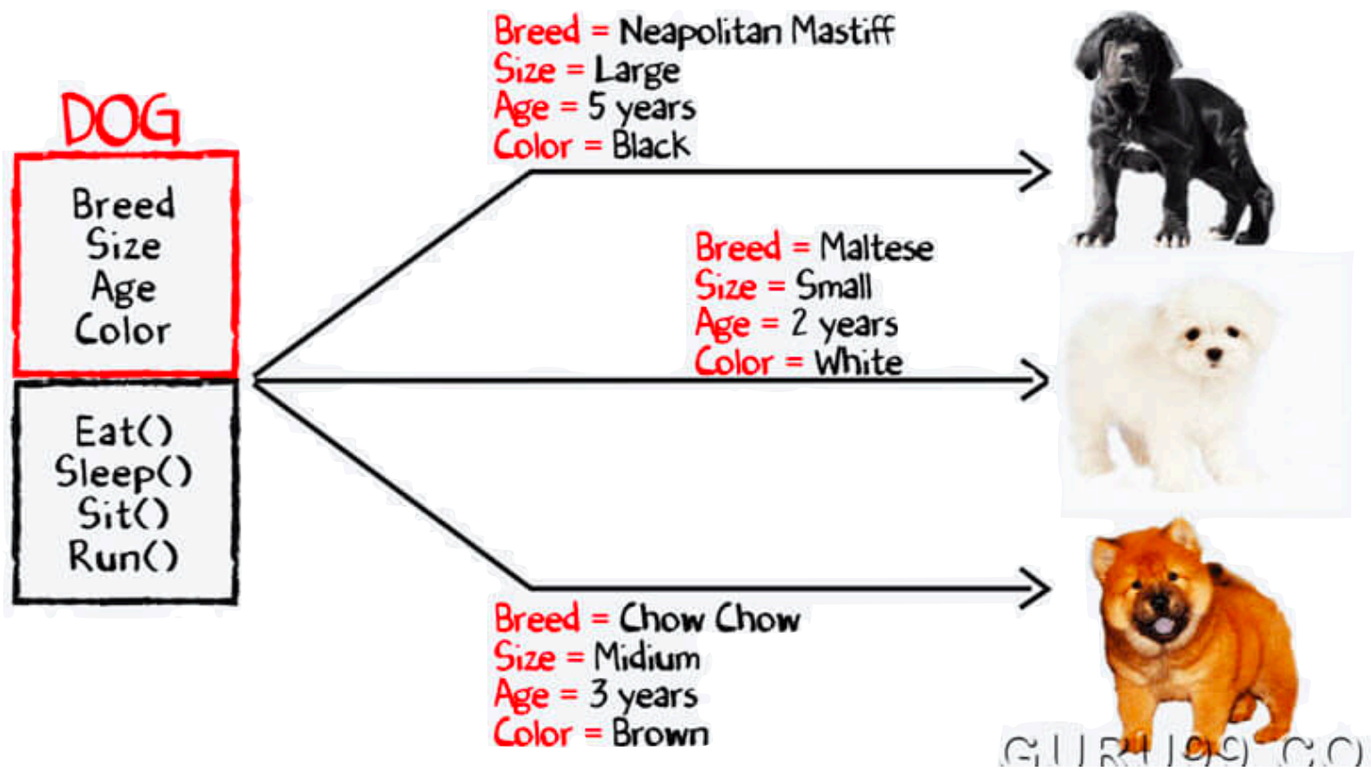
万物皆为对象

Everything is an object





很多属性与具体的对象无关，而与类直接相关



科学分类	
界：	动物界 Animalia
门：	脊索动物门 Chordata
纲：	哺乳纲 Mammalia
目：	食肉目 Carnivora
科：	犬科 Canidae
属：	犬属 *Canis*
种：	狼 *C. lupus*
亚种：	**犬 *C. l. familiaris***

java.lang.Math

- Abs
- Ceil
- Pow
- Max
- Sqrt
- Sin
- Cos

java.lang.System

- getenv ()
- console ()
- lineSeparator ()
- setIn ()
- setOut ()
- loadLibrary ()

很多工具类包含了工具函数，并不需要对象

static 全局&静态

描述与类相关，不与具体的某个对象相关的属性与方法

static成员变量在JVM加载类时创建
而不是在new对象时创建，所有该类对象共享同一个

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

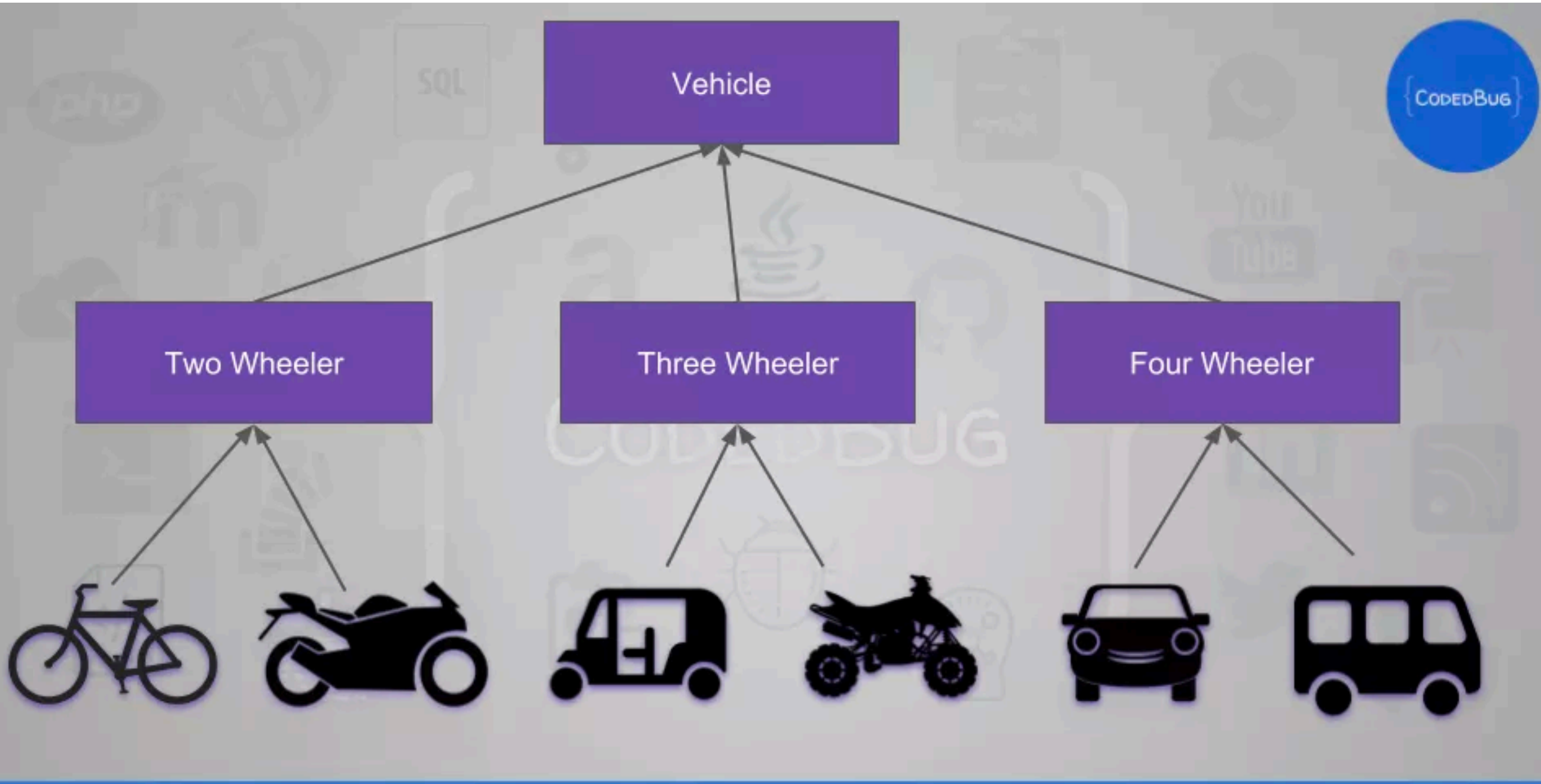
浅拷贝 vs. 深拷贝

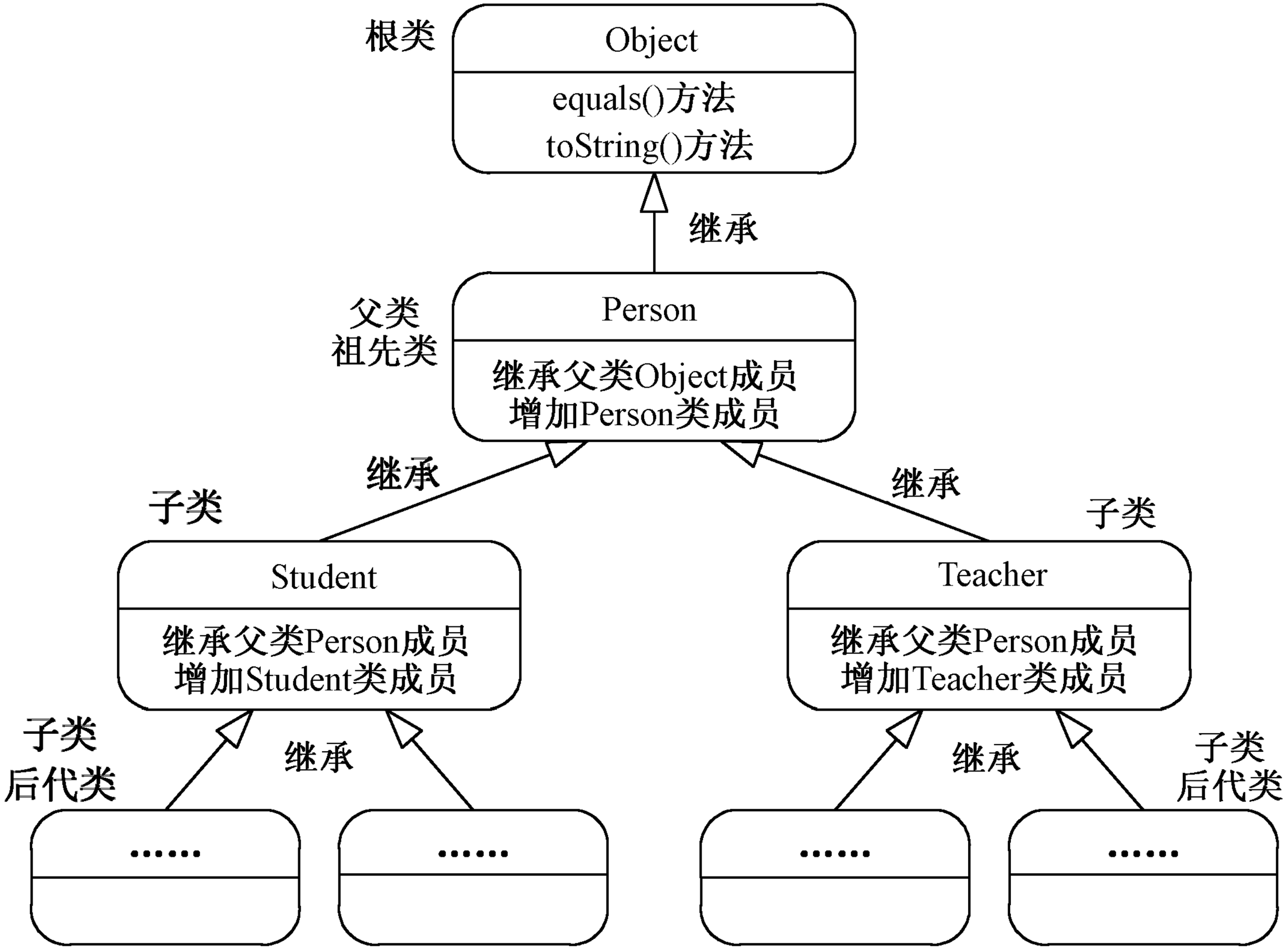
代码复用

COPY

PASTE

继承


Vehicle
Two Wheeler
Three Wheeler
Four Wheeler
CodedBug
Inheritance In Java


根类
Object
equals()方法
toString()方法
继承
父类
祖先类
Person
继承父类Object成员
增加Person类成员
继承
继承
子类
Student
继承父类Person成员
增加Student类成员
子类
Teacher
继承父类Person成员
增加Teacher类成员
子类
后代类
继承
继承
子类
后代类
.....
.....
.....
.....

继承的定义

[修饰符] class 类名 **[extends 父类名]**
{
    新增成员变量的声明;
    新增或重写成员方法的声明及实现;
}

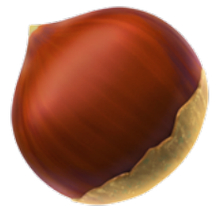
```
public class Student extends Person
{
    String speciality;
    public void learn() {}
}
```

继承的基本原则

- 子类继承父类的成员变量
- 子类继承父类除构造方法以外的成员方法
- 子类可以增加成员（变量和方法），可以重写从父类继承来的成员方法。
- Java中的类都是Object的子类

```
public class Object
{
    public Object()                                  //构造方法
    public String toString()                        //描述对象
    public boolean equals(Object obj)//比较对象相等
    protected void finalize() throws Throwable
}
```

举个例子



```
abstract class Pet{
    public abstract void makeSound();
}

class Cat extends Pet{
    public void makeSound() {
        System.out.println("Meow");
    }
}

class Dog extends Pet{
    public void makeSound() {
        System.out.println("Woof");
    }
}
```

重写
Override

区分重载与重写

Overriding 重写

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

方法名与参数都一样

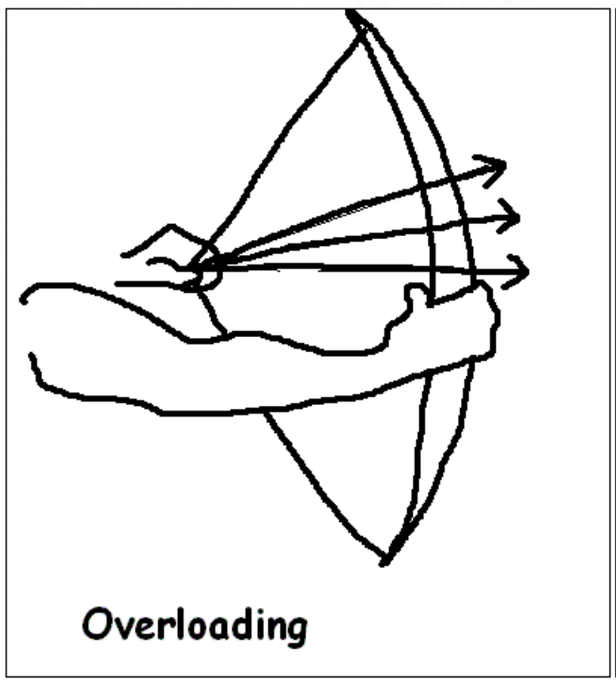
Overloading 重载

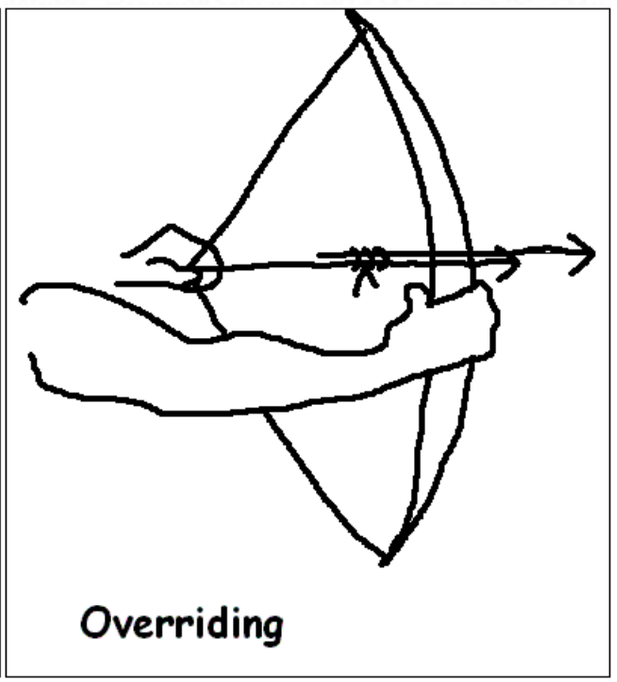
```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

方法名相同，参数不同

区别点	重载方法	重写方法
参数列表	必须修改	一定不能修改
返回类型	可以修改	一定不能修改





举个例子

Person

静态成员变量

count

实例成员变量

name, birthday, sex, province, city

构造方法

Person(name, birthday, sex, province, city)

Person(name, birthday)

Person()

Person(p)

析构方法

finalize()

静态成员方法

void howMany()

实例成员方法

void set(name, birthday, sex, province, city)

String toString()

boolean equals(obj)

继承

Student

静态成员变量

count //隐藏

实例成员变量

department, speciality, number, member

构造方法

Student(name, birthday, sex, province,city,
department, speciality, number, member)

Student()

Student(person, department, speciality, number, member)

Student(s)

析构方法

finalize() //覆盖

静态成员方法

void howMany() //覆盖

实例成员方法

void set(department, speciality, number, member) //重载

String toString() //覆盖

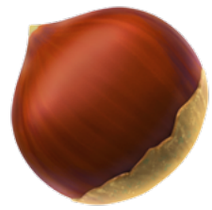
boolean equals(obj) //覆盖

成员变量隐藏

子类中的成员变量如果和父类中的成员变量同名，那么即使他们类型不一样，只要名字一样，父类中的成员变量都会被隐藏。

	this	super
作为函数	this()调用本类的构造方法	super()调用父类的构造方法
访问成员变量	当局部变量与成员变量重名时，用this.成员变量名	当子类隐藏了父类的成员变量时，用super.成员变量名
访问成员方法	------	当父类的成员方法被重写时

举个例子



```
class Super {
    String s = "Super";
    String say(){
        return "hello Super";
    }
}
```

```
class Sub extends Super {
    String s = "Sub";
    String say(){
        return "hello Sub";
    }
}
```

```
public class FieldOverriding {

    public static void main(String[] args) {
        Sub c1 = new Sub();
        System.out.println(" c1.s : " + c1.s);
        System.out.println(" c1.say : " + c1.say());

        Super c2 = new Sub();
        System.out.println(" c2.s : " + c2.s);
        System.out.println(" c2.say : " + c2.say());
    }
}
```

protected权限

子类不能访问父类的私有成员（private）。

子类能够访问父类的公有成员（public）和保护成员（protected）。

子类对父类的缺省权限成员的访问控制，以包为界分两种情况。

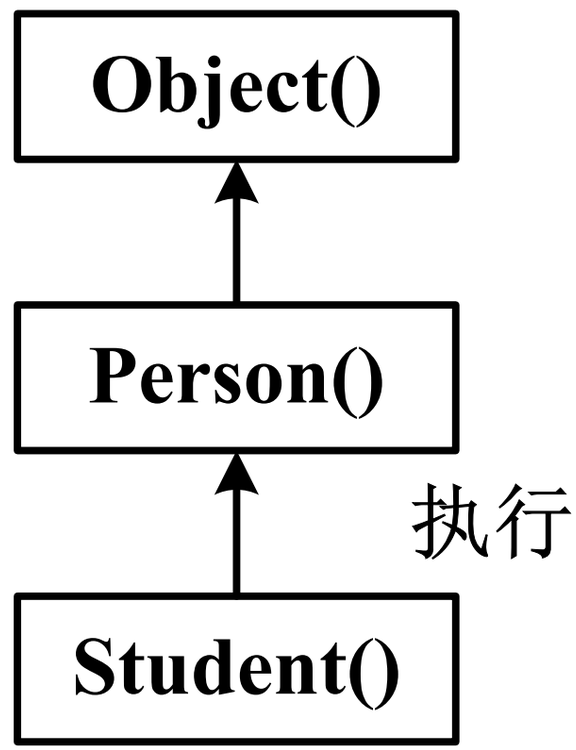
Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	X
no modifier	Y	Y	X	X
private	Y	X	X	X

子类的构造函数

使用super([参数列表])调用父类构造方法，且要放在构造函数第一行

使用this([参数列表])可以调用该类重载的构造方法

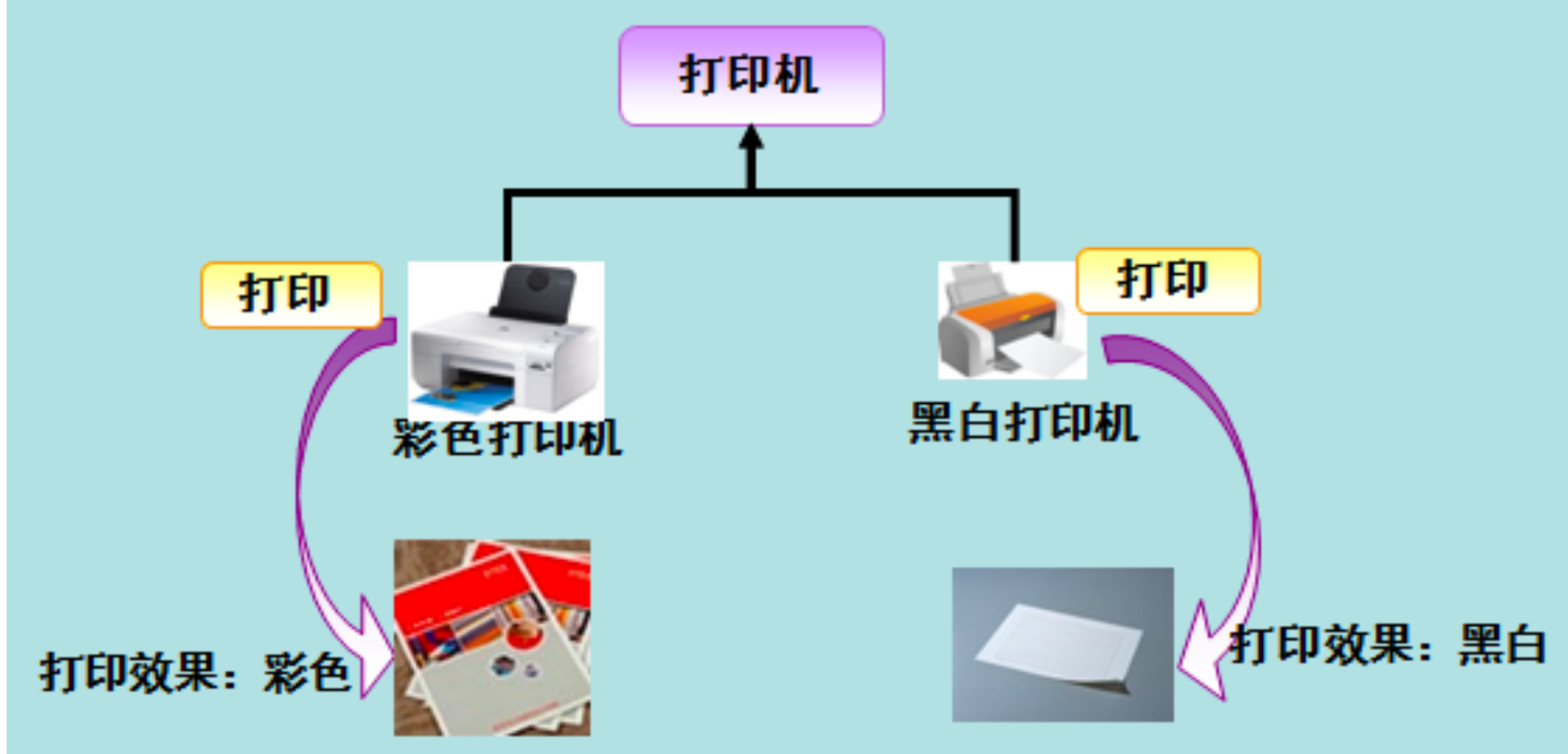
当没有构造方法时，将默认调用空super()

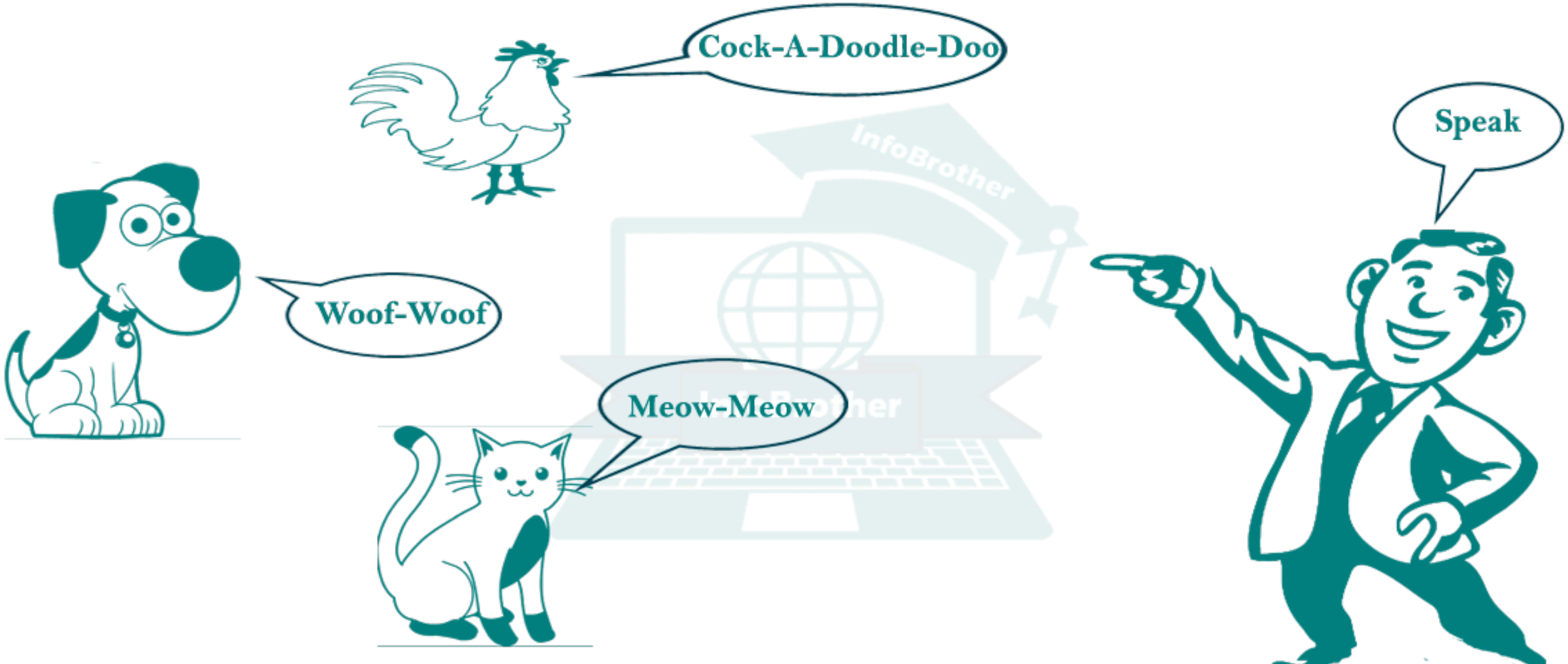


多态(覆盖)

消除类型之间的

耦合关系

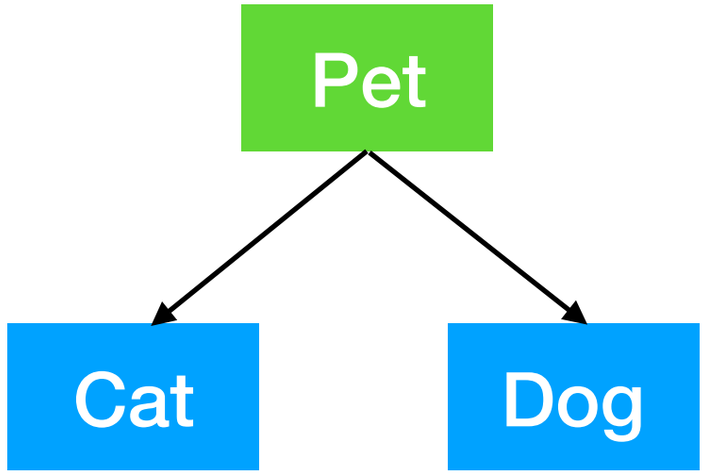




```java
abstract class Pet{
    public abstract void makeSound();
}

class Cat extends Pet{
    public void makeSound() {
        System.out.println("Meow");
    }
}

class Dog extends Pet{
    public void makeSound() {
        System.out.println("Woof");
    }
}
```



多类的条件

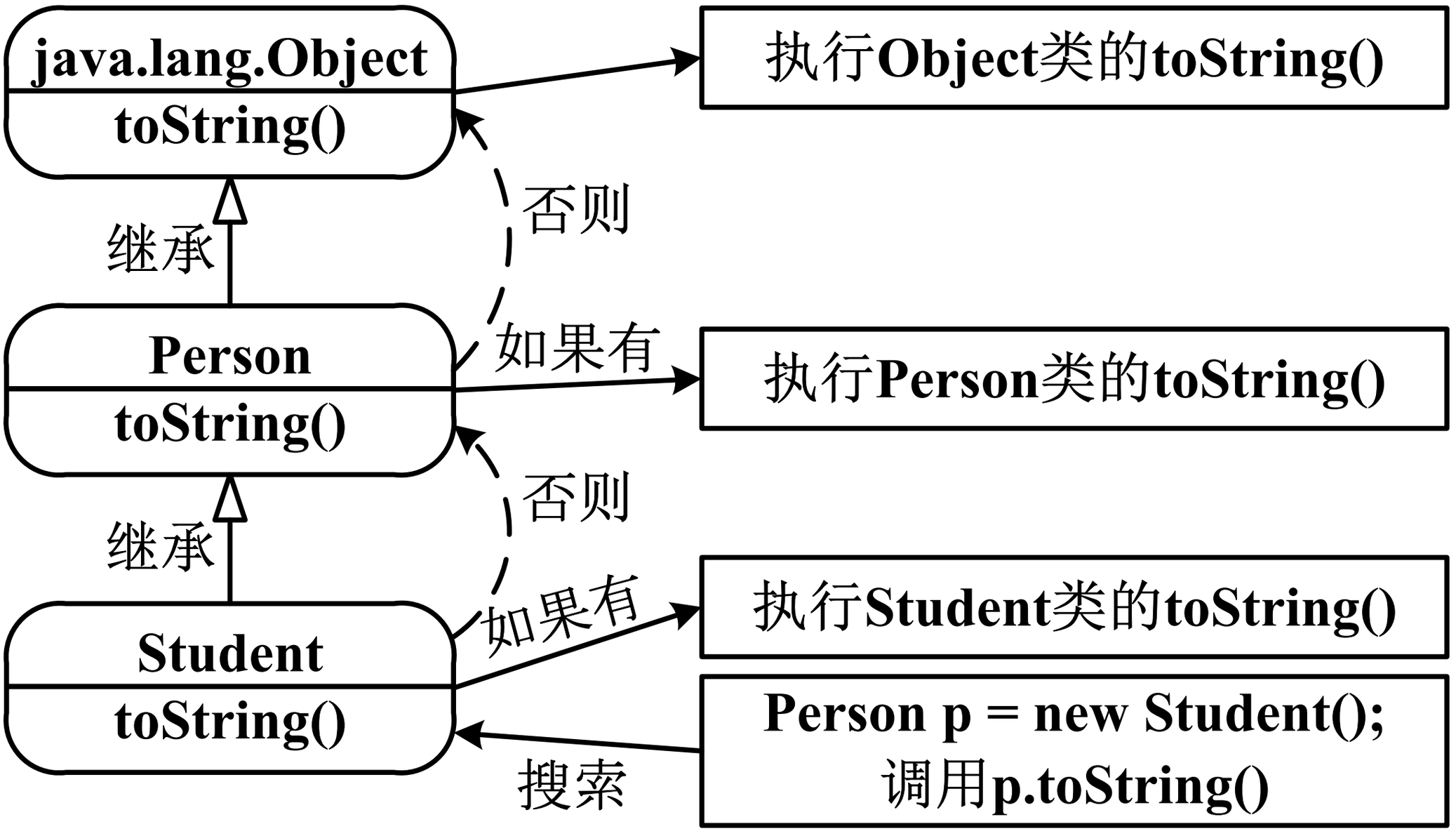
要有继承 要有重写

父类引用指向子类对象

```java
public class PolymorphismDemo{
    public static void main(String args[]) {
        Pet a = new Cat();
        Pet b = new Dog();
        a.makeSound();
        b.makeSound();
    }
}
```

Dynamic binding

Dynamic binding



多态的好处

指允许不同类的对象对同一消息做出响应。即同一消息可以根据发送对象的不同而采用多种不同的行为方式。（发送消息就是函数调用）

- 1.可替换性（substitutability）。多态对已存在代码具有可替换性。例如，多态对圆Circle类工作，对其他任何圆形几何体，如圆环，也同样工作。

- 2.可扩充性（extensibility）。多态对代码具有可扩充性。增加新的子类不影响已存在类的多态性、继承性，以及其他特性的运行和操作。实际上新加子类更容易获得多态功能。例如，在实现了圆锥、半圆锥以及半球体的多态基础上，很容易增添球体类的多态性。

- 3.接口性（interface-ability）。多态是超类通过方法签名，向子类提供了一个共同接口，由子类来完善或者覆盖它而实现的。如图8.3 所示。图中超类Shape规定了两个实现多态的接口方法，computeArea()以及computeVolume()。子类，如Circle和Sphere为了实现多态，完善或者覆盖这两个接口方法。

- 4.灵活性（flexibility）。它在应用中体现了灵活多样的操作，提高了使用效率。

- 5.简化性（simplicity）。多态简化对应用软件的代码编写和修改过程，尤其在处理大量对象的运算和操作时，这个特点尤为突出和重要。