



参悟Java基础 核心技术

讲师：宋红康

新浪微博：尚硅谷-宋红康

—、
Why is Java ?



Java在各领域中的应用

- 从Java的应用领域来分，Java语言的应用方向主要表现在以下几个方面：
 - **企业级应用**：主要指复杂的大企业的软件系统、各种类型的网站。Java的安全机制以及它的跨平台的优势，使它在分布式系统领域开发中有广泛应用。应用领域包括金融、电信、交通、电子商务等。
 - **Android平台应用**：Android应用程序使用Java语言编写。Android开发水平的高低很大程度上取决于Java语言核心能力是否扎实。
 - 移动领域应用，主要表现在消费和嵌入式领域，是指在各种小型设备上的应用，包括手机、PDA、机顶盒、汽车通信设备等。

1. 从java语言的诞生、特点说起

java之父Jgosling团队在开发“Green”项目时，发现C缺少垃圾回收系统，还有可移植的安全性、分布程序设计、和多线程功能。最后，他们想要一种易于移植到各种设备上的平台。

Java确实是从C语言和C++语言继承了许多成份，甚至可以将Java看成是**类c语言**发展和衍生的产物。比如Java语言的变量声明，操作符形式，参数传递，流程控制等方面和C语言、C++语言完全相同。但同时，Java是一个**纯粹的面向对象**的程序设计语言，它继承了 C++语言面向对象技术的核心。Java舍弃了C语言中容易引起错误的指针（以引用取代）、运算符重载（operator overloading）、多重继承（以接口取代）等特性，增加了垃圾回收器功能用于回收不再被引用的对象所占据的内存空间。JDK1.5又引入了泛型编程（Generic Programming）、类型安全的枚举、不定长参数和自动装/拆箱



java语言的主要特性

Java语言是易学的。Java语言的语法与C语言和C++语言很接近，使得大多数程序员很容易学习和使用Java。

Java语言是强制面向对象的。Java语言提供类、接口和继承等原语，为了简单起见，只支持类之间的单继承，但支持接口之间的多继承，并支持类与接口之间的实现机制（关键字为implements）。

Java语言是分布式的。Java语言支持Internet应用的开发，在基本的Java应用编程接口中有一个网络应用编程接口（java net），它提供了用于网络应用编程的类库，包括URL、URLConnection、Socket、ServerSocket等。Java的RMI（远程方法激活）机制也是开发分布式应用的重要手段。

Java语言是健壮的。Java的强类型机制、异常处理、垃圾的自动收集等是Java程序健壮性的重要保证。对指针的丢弃是Java的明智选择。

java语言的主要特性

Java语言是安全的。Java通常被用在网络环境中，为此，Java提供了一个安全机制以防恶意代码的攻击。如：安全防范机制（类ClassLoader），如分配不同的名字空间以防替代本地的同名类、字节代码检查。

Java语言是体系结构中立的。Java程序（后缀为java的文件）在Java平台上被编译为体系结构中立的字节码格式（后缀为class的文件），然后可以在实现这个Java平台的任何系统中运行。

Java语言是解释型的。如前所述，Java程序在Java平台上被编译为字节码格式，然后可以在实现这个Java平台的任何系统的解释器中运行。

Java是性能略高的。与那些解释型的高级脚本语言相比，Java的性能还是较优的。

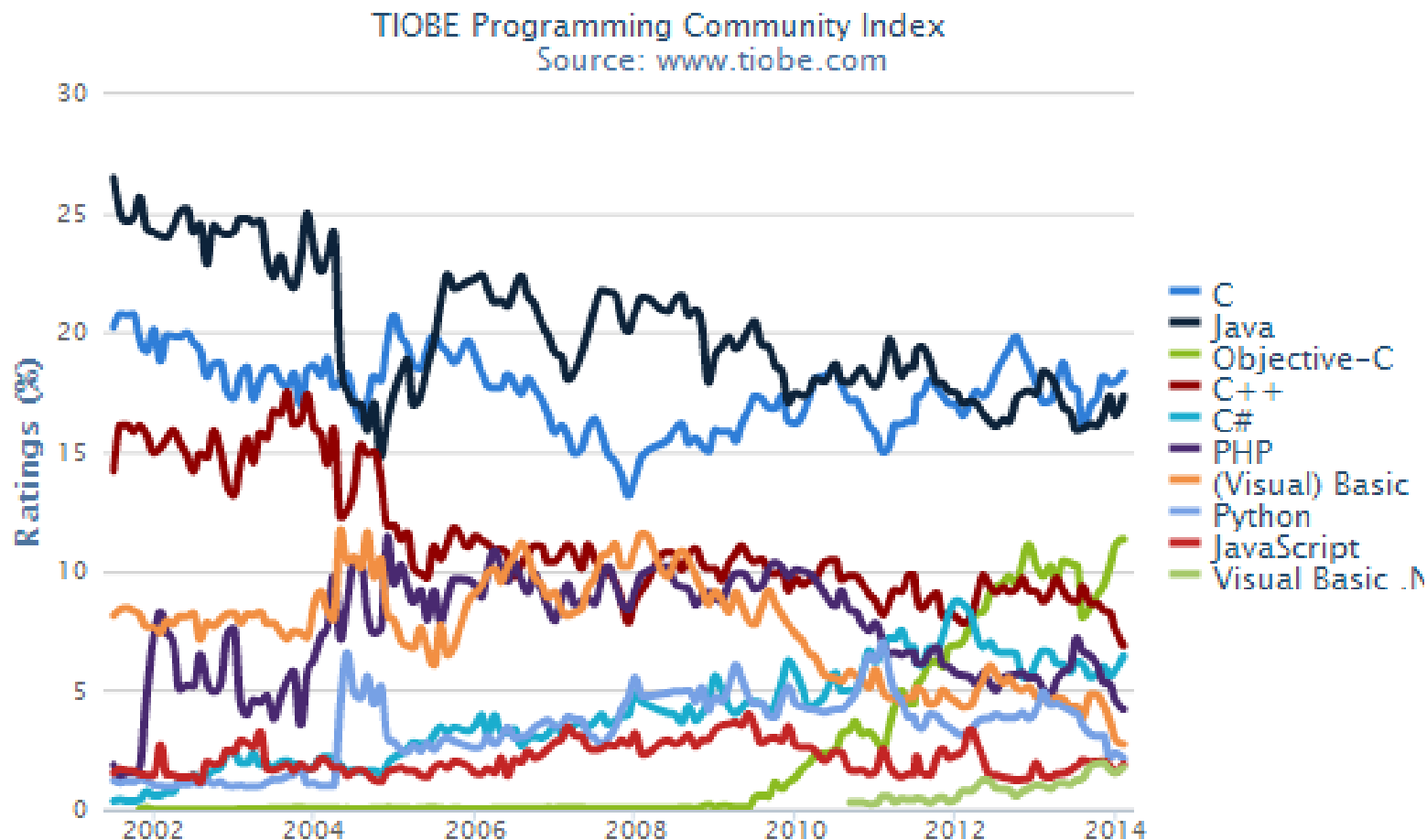
Java语言是原生支持多线程的。在Java语言中，线程是一种特殊的对象，它必须由Thread类或其子（孙）类来创建。

2. 从java语言的市场需求来看

Feb 2014	Feb 2013	Change	Programming Language	Ratings	Change
1	2	▲	C	18.334%	+1.25%
2	1	▼	Java	17.316%	-1.07%
3	3		Objective-C	11.341%	+1.54%
4	4		C++	6.892%	-1.87%
5	5		C#	6.450%	-0.23%
6	6		PHP	4.219%	-0.85%
7	8	▲	(Visual) Basic	2.759%	-1.89%
8	7	▼	Python	2.157%	-2.79%
9	11	▲	JavaScript	1.929%	+0.51%
10	12	▲	Visual Basic .NET	1.798%	+0.79%

2014年2月 TIOBE 编程语言排行榜单

2. 从java语言的市场需求来看



2014年2月 TIOBE 编程语言排行榜单

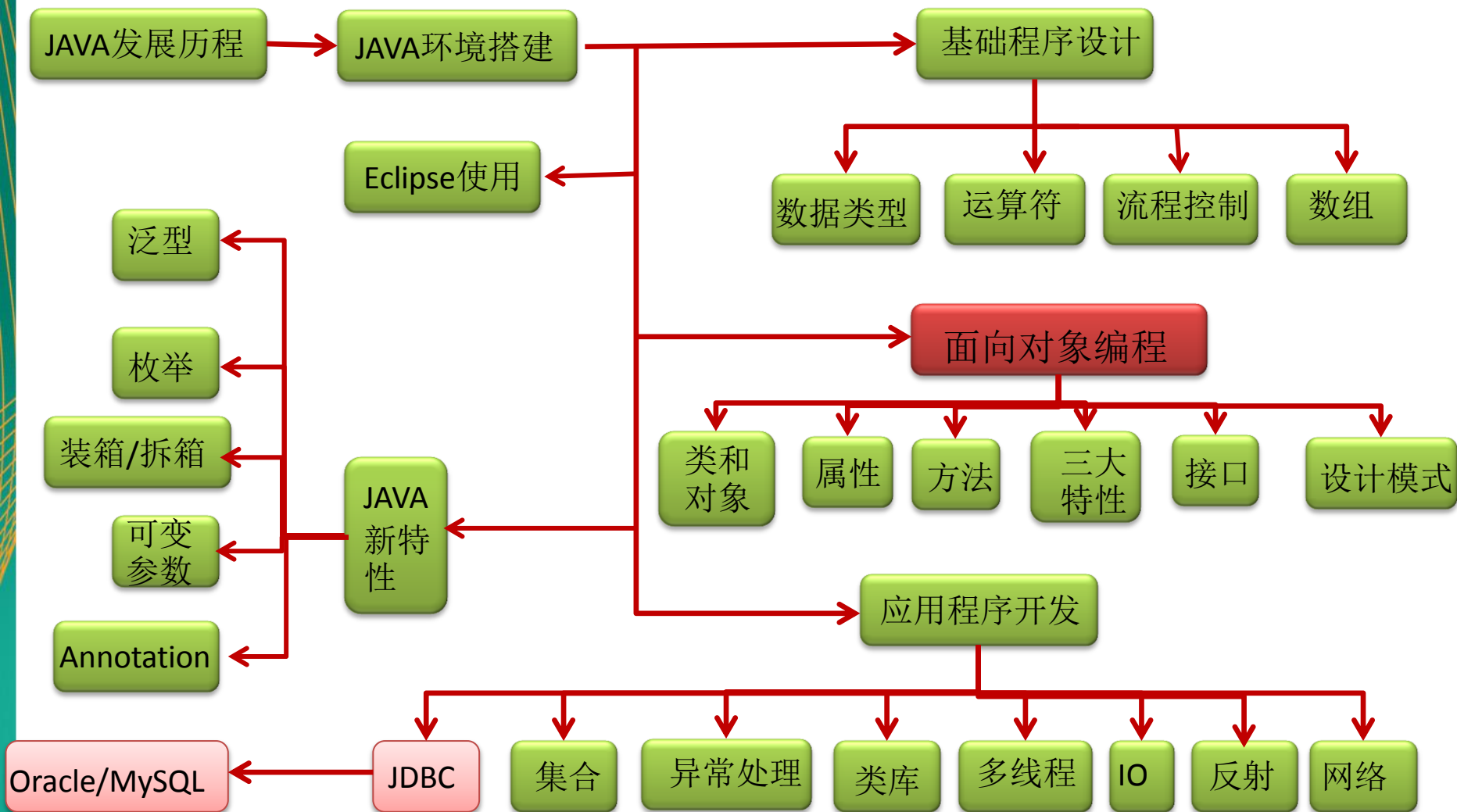
二、

如何看待java基础？

如何理解面向对象思想？

Java 基础

- 深入讲授 java 语言的核心内容：
 - 涵盖 java 的基本语法结构、java的面向对象特征、java集合框架体系、java泛型、异常处理、java 注释、java的 io 流体系、java多线程编程、java网络通信编程和 java 反射机制；
 - 覆盖了java.lang、java.util、java.text、java.io 和 java.nio、java.sql 包下绝大部分类和接口。
 - 全面介绍 **java 7** 的二进制整数、菱形语法、增强 switch语句、多异常捕获、自动关闭资源的try语句、nio.2、aio等新特性。




何谓“面向对象”的编程思想？

顿悟？ OR 渐悟？



例子：人把大象装冰箱

1. 打开冰箱
 2. 把大象装进冰箱
 3. 把冰箱门关上
- 

面向过程

```
人{  
    打开（冰箱）{  
        冰箱.开门();  
    }  
    操作(大象){  
        大象.进入(冰箱);  
    }  
    关闭(冰箱){  
        冰箱.关门();  
    }  
}  
  
冰箱{  
    开门(){} 关门(){}  
}
```

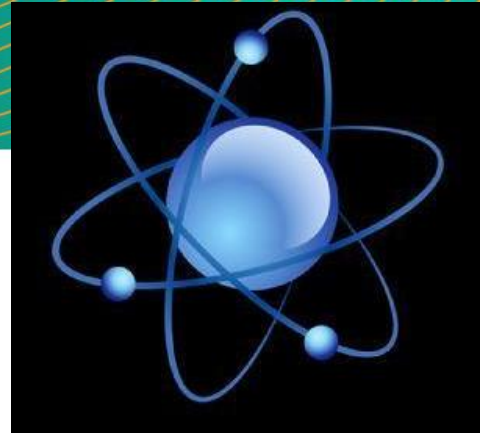
人
冰箱
大象

```
大象{  
    进入(冰箱){ }  
}
```

面向对象

学习面向对象内容的三条主线

1. java类及类的成员
2. 面向对象的三大特征
3. 其它关键字



1. java类及类的成员

- 现实世界万事万物是由分子、原子构成的。同理，Java代码世界是由诸多个不同功能的类构成的。
- 面向对象程序设计的重点是类的设计
- 现实世界中的分子、原子又是由什么构成的呢？原子核、电子！那么，Java中用类class来描述事物也是如此
 - 属性：对应类中的成员变量
 - 行为：对应类中的成员方法

Field = 属性 = 成员变量， Method = (成员)方法 = 函数

1. java类及类的成员

```
class Person {  
    String name;  
    int age;  
    boolean isMarried;  
  
    public void walk(){  
        System.out.println("人走路...");  
    }  
    public String display(){  
        return "名字是: "+name+", 年龄是: "+age+", Married: "+isMarried;  
    }  
}
```

属性，或成员变量

方法，或函数

类的成员构成 version 1.0

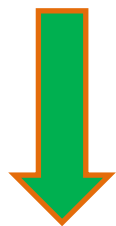
语法格式：

- 定义类的属性：修饰符 类型 属性名 = 初值 ；
- 定义类的方法：修饰符 返回值类型 方法名 (参数列表) {
方法体语句；
}

类的成员构成 version 2.0

```
class Person {  
    //属性，或成员变量  
    String name;  
    boolean isMarried;  
    //构造器  
    public Person(){}  
    public Person(String n,boolean im){  
        name = n;isMarried = im;  
    }  
    //方法，或函数  
    public void walk(){  
        System.out.println("人走路...");  
    }  
    public String display(){  
        return "名字是: "+name+",Married:"+isMarried;  
    }  
    //代码块  
    {  
        name = "HanMeiMei";  
        age = 17;  
        isMarried = true;  
    }  
    //内部类  
    class pet{  
        String name;  
        float weight;  
    }  
}
```


1. java类及类的成员

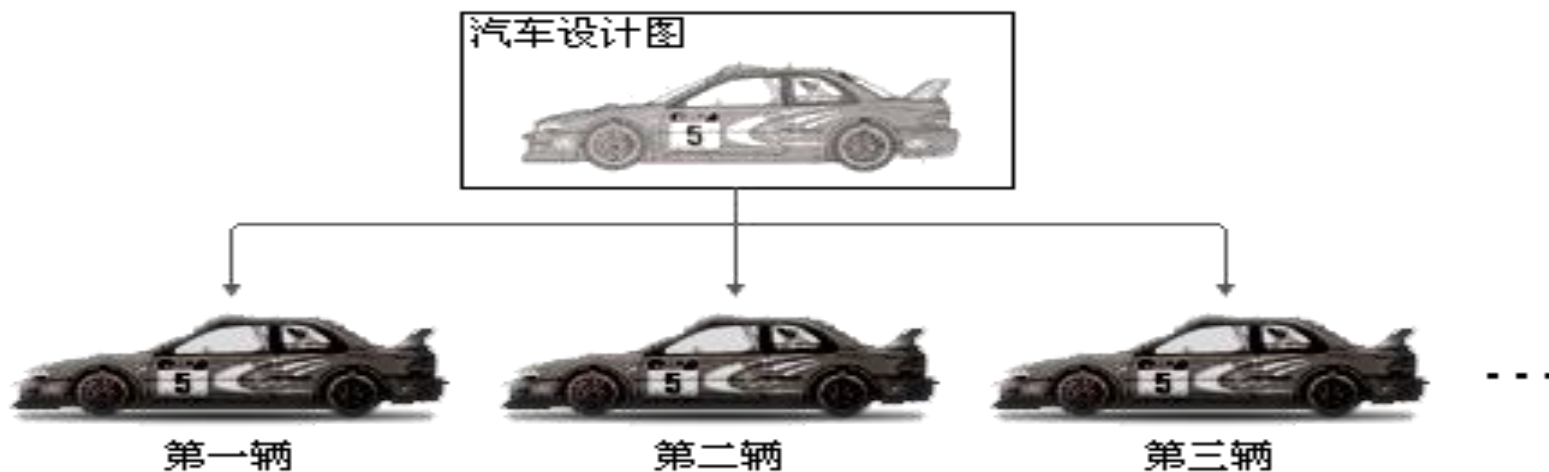


如何使用java类?

java类的实例化，即创建类的对象

- 类(class)和对象(object)是面向对象的核心概念
- 完成工程需求时：
 - 先去找具有所需功能的对象来用
 - 如果该对象不存在，那么创建一个具有所需功能的对象
 - 这样简化开发并提高复用

类和对象的关系

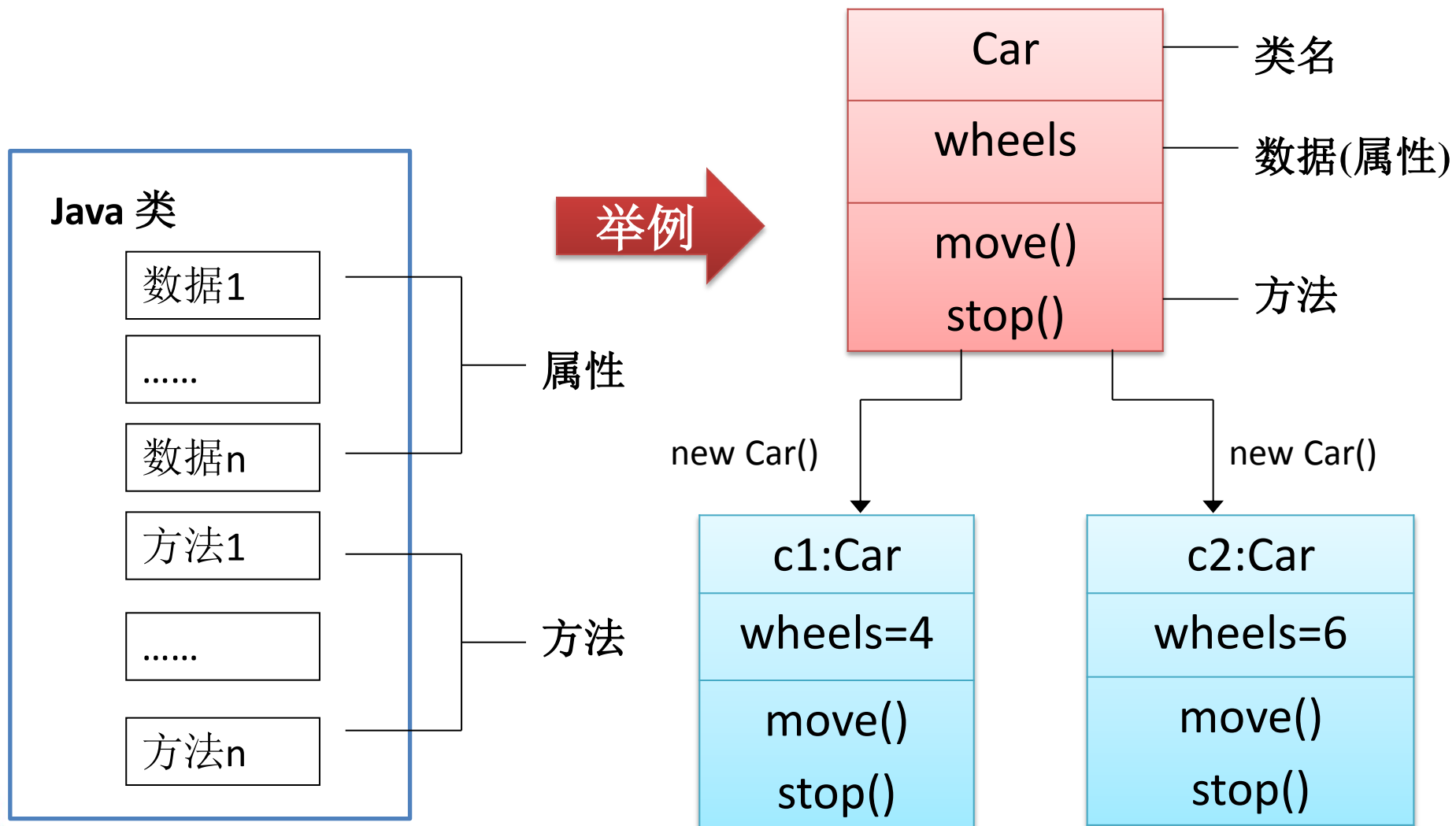


- 可以理解为：类 = 汽车设计图；对象 = 实实在在的汽车
- 类的实例化：`Car c = new Car();`

面向对象思想“落地”法则(一)

- 使用 **new + 构造器** 创建一个新的对象；
- 使用 “**对象名.对象成员**” 的方式访问对象成员（包括属性和方法）；
- 如果创建了一个类的多个对象，对于类中定义的属性，每个对象都拥有各自的一套副本，且互不干扰。
- 在一个类中：类中的方法可以直接访问类中的成员变量或其它方法。（例外：**static方法访问非static，编译不通过。**）

Java 中类与对象举例



练习：创建Java自定义类

步骤：

1. 定义类（考虑修饰符、类名）
2. 编写类的属性（考虑修饰符、属性类型、属性名、初始化值）
3. 编写类的方法（考虑修饰符、返回值类型、方法名、形参等）

提示：

定义Person、Animal、Bird、Zoo等类，加以体会。

2. 面向对象的三大特征

- 封装 (Encapsulation)
- 继承 (Inheritance)
- 多态 (Polymorphism)

2.1 面向对象的特征一：封装性

使用者对类内部定义的属性(对象的成员变量)的直接操作会导致数据的错误、混乱或安全性问题。

```
public class Animal {  
    public int legs;  
    public void eat(){  
        System.out.println("Eating.");  
    }  
    public void move(){  
        System.out.println("Moving.");  
    }  
}
```

```
public class Zoo{  
    public static void main(String args[]){  
        Animal xb=new Animal();  
        xb.legs=4;  
        System.out.println(xb.legs);  
        xb.eat();xb.move();  
    }  
}
```

应该将legs属性保护起来，防止乱用。

保护的方式：信息隐藏

问题： xb.legs = -1000;

面向对象思想“落地”法则(二)

Java中通过将数据声明为私有的(private)，再提供公共的(public)方法:**getXxx()**和**setXxx()**实现对该属性的操作，以实现下述目的：

- **隐藏**一个类中不需要对外提供的实现细节；
- 使用者只能通过事先定制好的**方法来访问数据**，可以方便地加入控制逻辑，限制对属性的不合理操作；
- 便于修改，增强代码的可维护性；

```
public class Animal{
```

```
    private int legs;//将属性legs定义为private，只能被Animal类内部访问
```

```
    public void setLegs(int i){ //在这里定义方法 eat() 和 move()
```

```
        if (i != 0 && i != 2 && i != 4){
```

```
            System.out.println("Wrong number of legs!");
```

```
            return;
```

```
        }
```

```
        legs=i;
```

```
    }
```

```
    public int getLegs(){
```

```
        return legs;
```

```
    } }
```

```
public class Zoo{
```

```
    public static void main(String args[]){
```

```
        Animal xb=new Animal();
```

```
        xb.setLegs(4);      //xb.setLegs(-1000);
```

```
        xb.legs=-1000;     //非法
```

```
        System.out.println(xb.getLegs());
```

```
    } }
```


四种访问权限修饰符

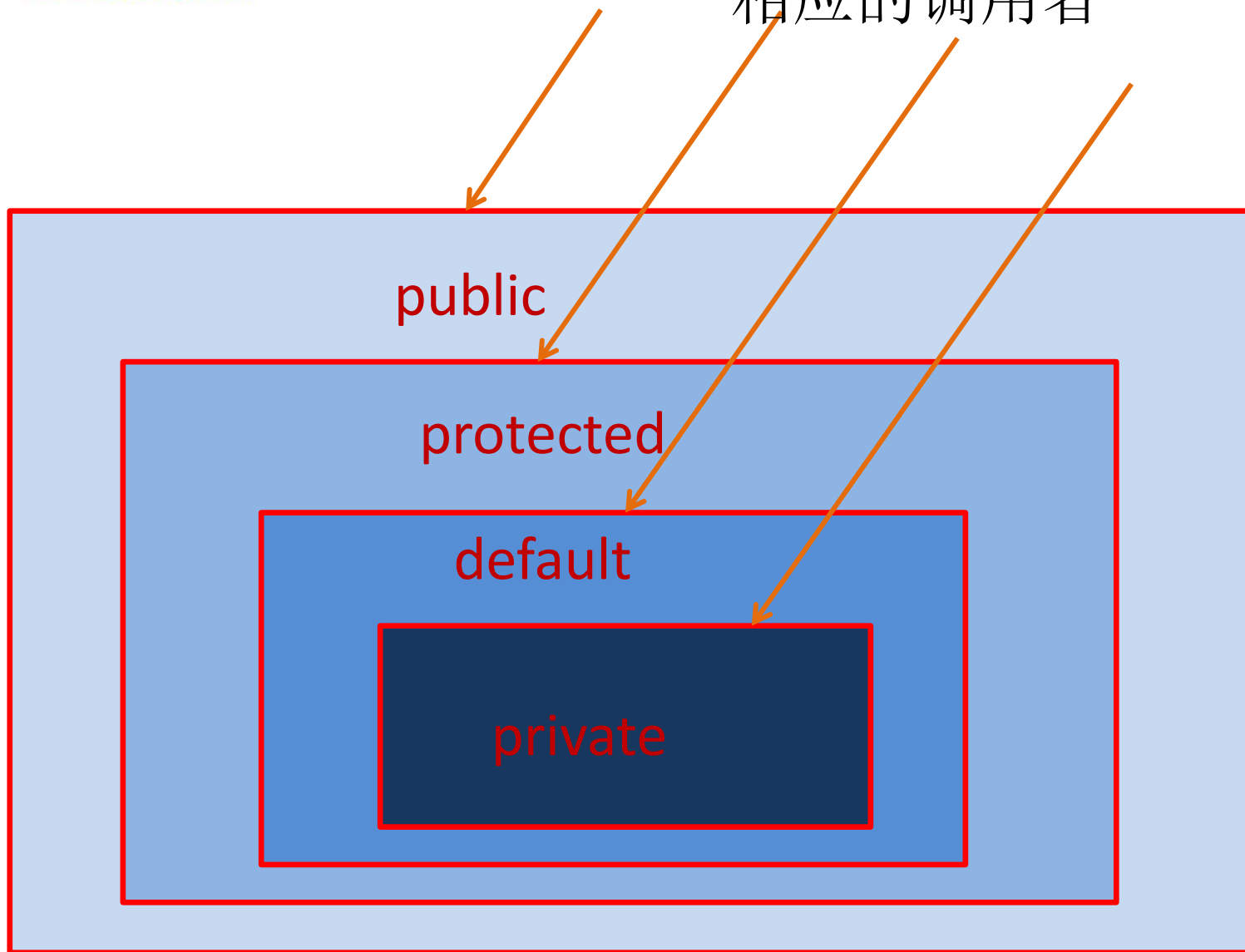
Java权限修饰符public、protected、private置于**类的成员**定义前，用来限定对象对该类成员的访问权限。

修饰符	类内部	同一个包	子类	任何地方
private	Yes			
(缺省)	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

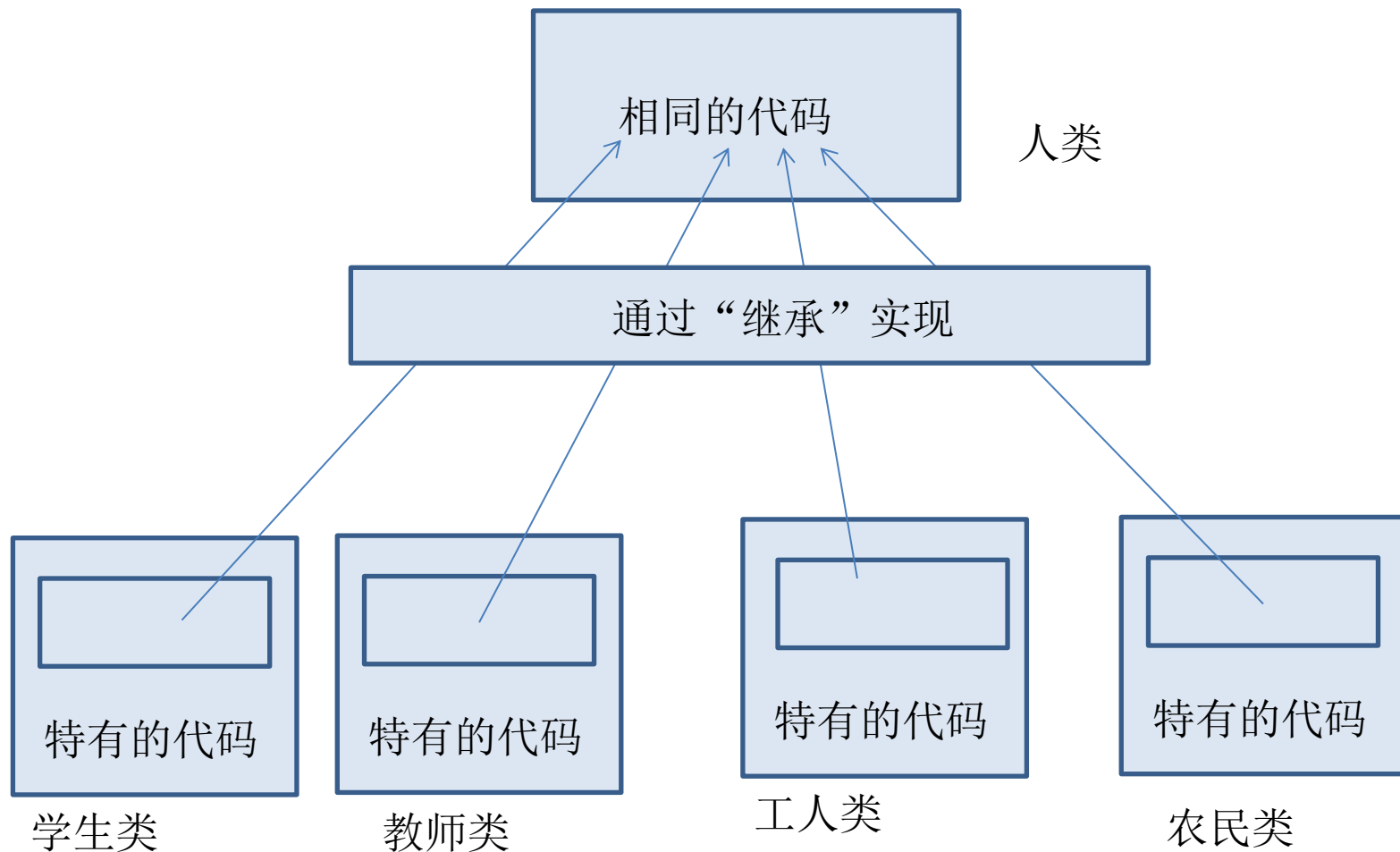
对于class的权限修饰只可以用public和default(缺省)。

- public类可以在任意地方被访问。
- default类只可以被同一个包内部的类访问。

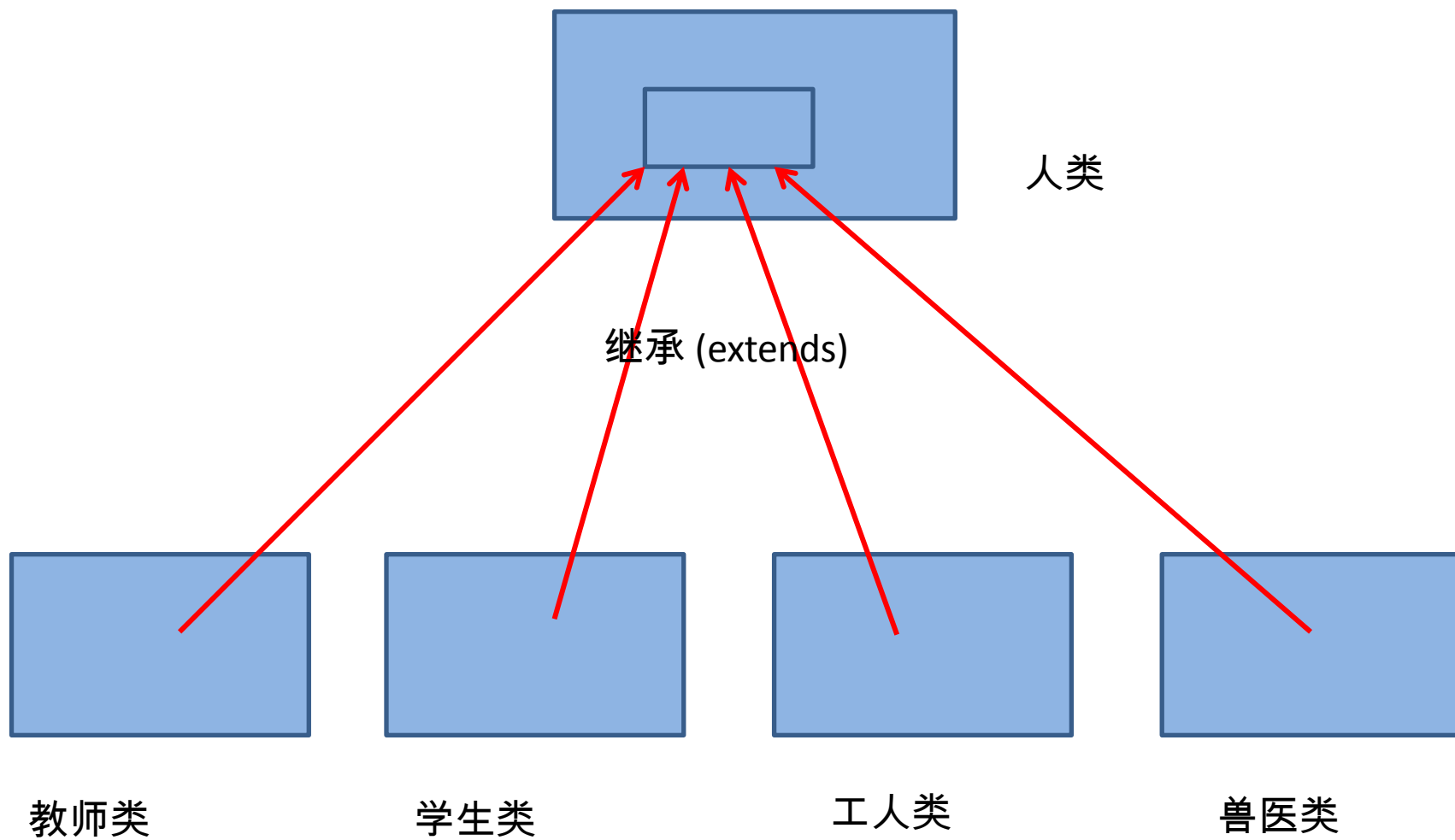
相应的调用者



2.2 面向对象的特征二：继承性



继承的思想



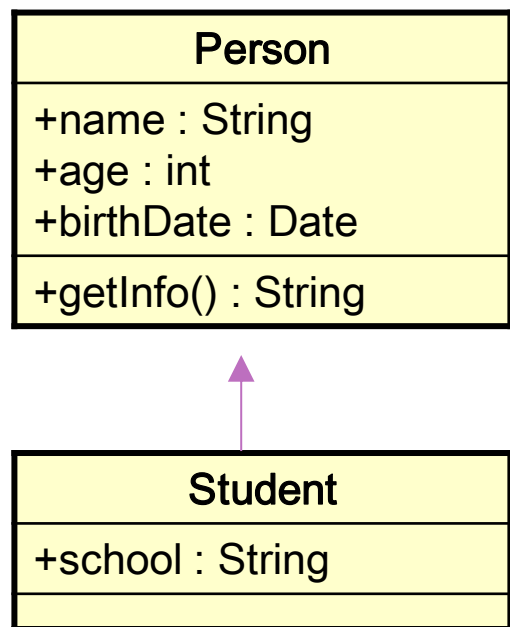
面向对象思想“落地”法则(三)

● 继承的思想

- 多个类中存在相同属性和行为时，将这些内容抽取到单独一个类中，那么多个类无需再定义这些属性和行为，只要继承那个类即可。
- 此处的多个类称为**子类**，单独的这个类称为**父类**（**基类或超类**）。可以理解为：“子类 is a 父类”
- 类继承语法规则：
class Subclass **extends** Superclass{ }

2.2 面向对象的特征二：继承性

- 通过继承，简化Student类的定义：



```
public class Person {  
    public String name;  
    public int age;  
    public Date birthDate;  
    public String getInfo() {...}  
}
```

```
public class Student extends Person{  
    public String school;  
}
```

//Student类继承了父类Person的所有属性和方法，并增加了一个属性school。Person中的属性和方法,Student都可以利用。

2.2 面向对象的特征二：继承性

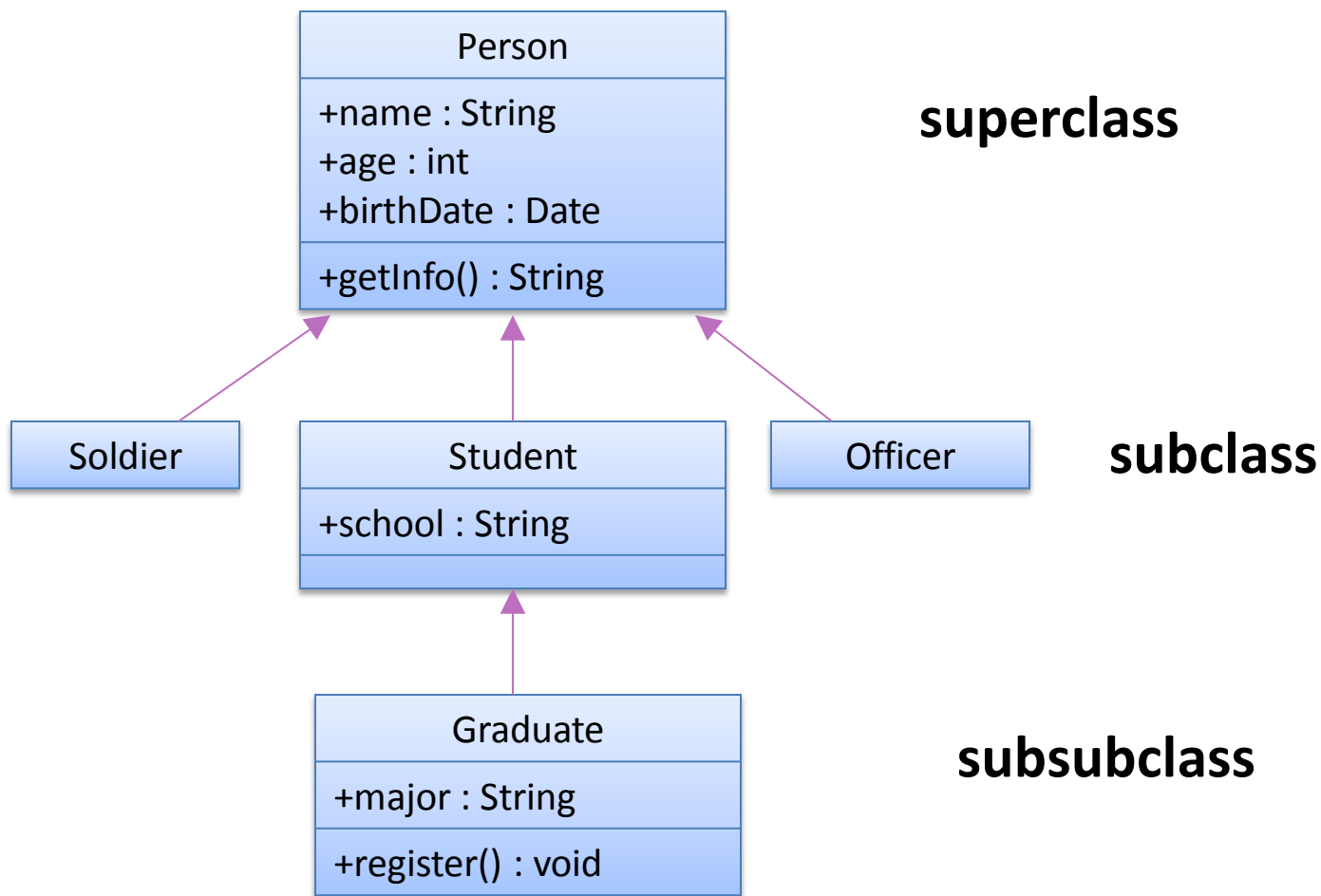
- 子类继承了父类，就继承了父类的方法和属性。
- 在子类中，可以使用父类中定义的方法和属性，也可以创建新的数据和方法。
- 在Java 中，继承的关键字用的是“**extends**”，即子类不是父类的子集，而是对父类的“扩展”。

关于继承的规则：

➤ 子类**不能直接访问**父类中私有的(**private**)的成员变量和方法。

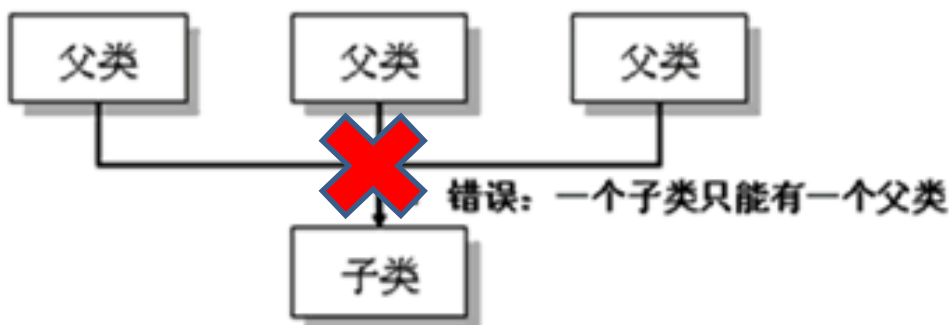


继承举例

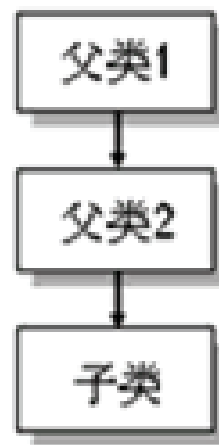


2.2 面向对象的特征二：继承性

- Java只支持单继承，不允许多重继承
 - 一个子类只能有一个父类
 - 一个父类可以派生出多个子类
 - ✓ `class SubDemo extends Demo{ }` //ok
 - ✓ `class SubDemo extends Demo1,Demo2...` //error



多重继承



多层继承

练习

1.(1)定义一个ManKind类，包括

- 成员变量int sex和int salary;
- 方法void manOrWorman(): 根据sex的值显示“man”(sex==1)或者“women”(sex==0);
- 方法void employeed(): 根据salary的值显示“no job”(salary==0)或者“job”(salary!=0)。

(2)定义类Kids继承ManKind，并包括

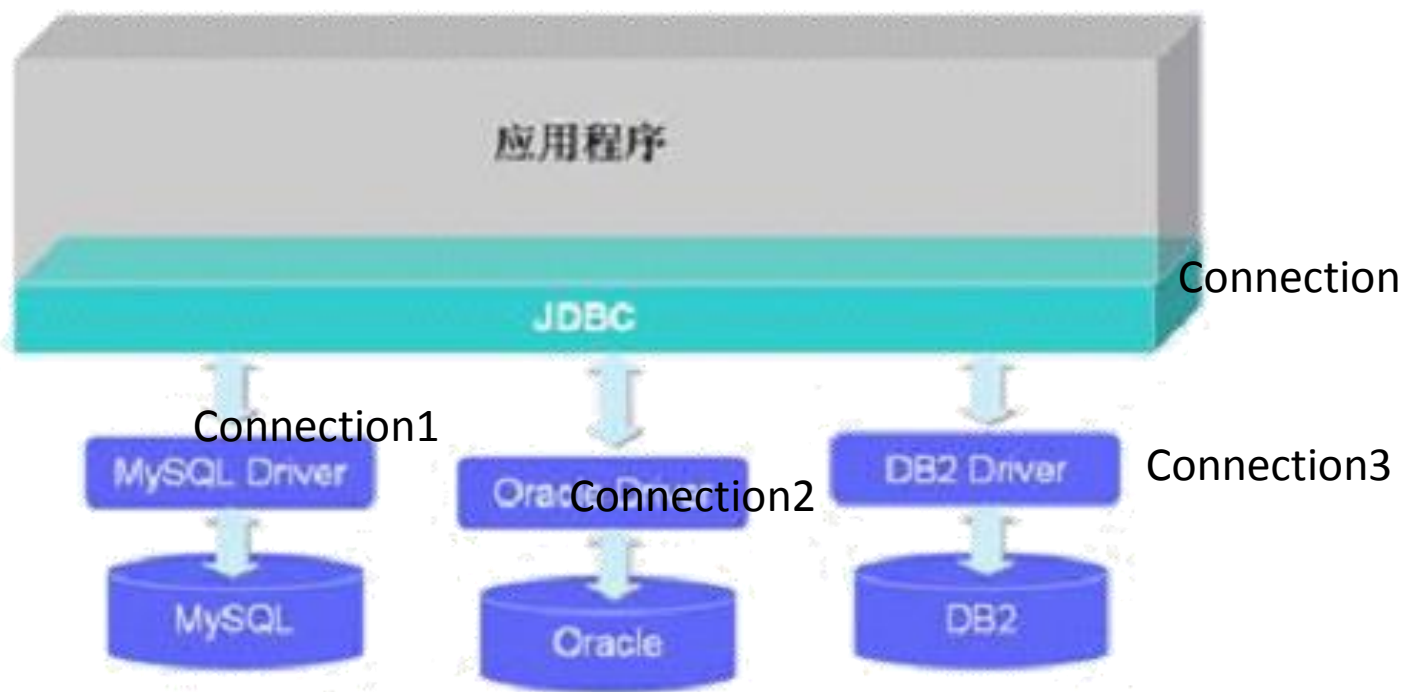
- 成员变量int yearsOld;
- 方法printAge()打印yearsOld的值。

(3)在Kids类的主方法中实例化Kids的对象someKid，用该对象访问其父类的成员变量及方法。

2.3 面向对象的特征三：多态性

- 多态性，是面向对象中最重要的概念，在java中有两种体现：
 1. 方法的重载(overload)和重写(override)。
 2. 对象的多态性：父类的引用指向子类对象实体
 - 可以直接应用在抽象类和接口上

2.3 面向对象的特征三：多态性



多态思想的应用实例

面向对象思想“落地”法则(四)

- 对象的多态

Person p = new Student();

Object o = new Person(); //Object类型的变量o，指向Person类型的对象

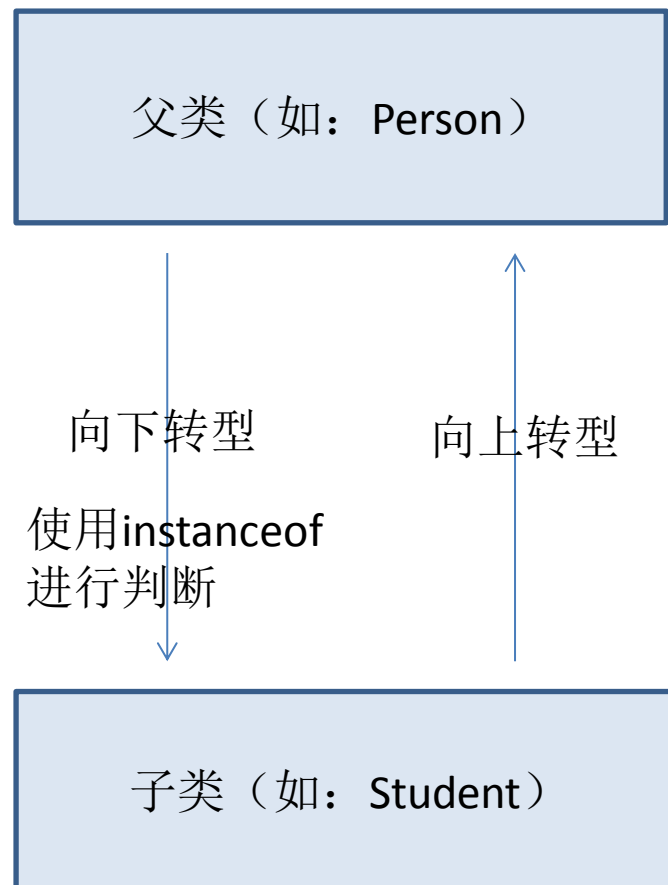
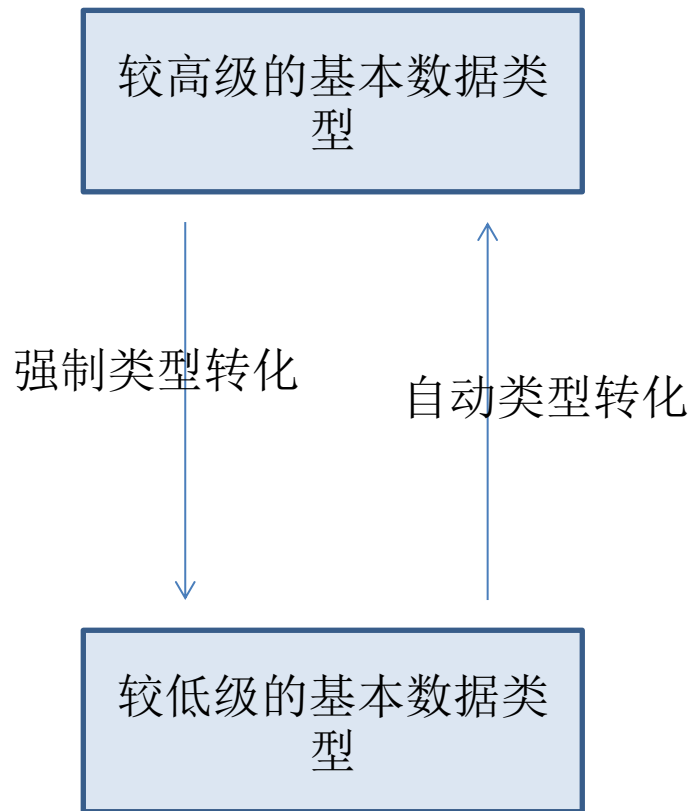
o = new Student(); //Object类型的变量o，指向Student类型的对象

- 虚拟方法调用(多态情况下)

Person e = new Student();

e.getInfo(); //调用Student类的getInfo()方法

编译时e为Person类型，而方法的调用是在运行时确定的，所以调用的是Student类的getInfo()方法。——动态绑定



3. 其它关键字

➤ **this**

➤ **super**

➤ **static**

➤ **final**

➤ **abstract**

➤ **interface**

关键字—this

this是什么？

- 在java中，this关键字比较难理解，它的作用和其词义很接近。
 - 它在方法内部使用，即这个方法所属对象的引用；
 - 它在构造器内部使用，表示该构造器正在初始化的对象。
- this表示当前对象，可以调用类的属性、方法和构造器
- 什么时候使用this关键字呢？
 - 当在方法内需要用到调用该方法的对象时，就用this。

● 使用this，调用属性、方法

```
class Person{           // 定义Person类
    private String name ;
    private int age ;
    public Person(String name,int age){
        this.name = name ;
        this.age = age ; }
    public void getInfo(){
        System.out.println("姓名： " + name) ;
        this.speak();
    }
    public void speak(){
        System.out.println("年龄： " + this.age);
    }
}
```

1.当形参与成员变量重名时，如果在方法内部需要使用成员变量，必须添加this来表明该变量时类成员

2.在任意方法内，如果使用当前类的成员变量或成员方法可以在其前面添加this，增强程序的阅读性

● 使用this调用本类的构造器

```
class Person{           // 定义Person类
    private String name ;
    private int age ;
    public Person(){    // 无参构造
        System.out.println("新对象实例化");
    }
    public Person(String name){
        this();    // 调用本类中的无参构造方法
        this.name = name ;
    }
    public Person(String name,int age){
        this(name) ; // 调用有一个参数的构造方法
        this.age = age;
    }
    public String getInfo(){
        return "姓名: " + name + ", 年龄: " + age ;
    } }
}
```

3.this可以作为一个类中，构造方法相互调用的特殊格式

关键字—**super**

- 在Java类中使用**super**来调用父类中的指定操作：
 - **super**可用于访问父类中定义的属性
 - **super**可用于调用父类中定义的成员方法
 - **super**可用于在子类构造方法中调用父类的构造器
- 注意：
 - 尤其当子父类出现同名成员时，可以用**super**进行区分
 - **super**的追溯不仅限于直接父类
 - **super**和**this**的用法相像，**this**代表本类对象的引用，**super**代表父类的内存空间的标识

关键字super举例

```
class Person {  
    protected String name="张三";  
    protected int age;  
    public String getInfo() {  
        return "Name: " + name + "\nage: " + age;  
    }  
}  
  
class Student extends Person {  
    protected String name = "李四";  
    private String school = "New Oriental";  
    public String getSchool(){ return school; }  
    public String getInfo() {  
        return super.getInfo() + "\nschool: " + school;  
    }  
}  
  
public class TestStudent{  
    public static void main(String[] args){  
        Student st = new Student();  
        System.out.println(st.getInfo());  
    }  
}
```

关键字—static

当我们编写一个类时，其实就是在描述其对象的属性和行为，而并没有产生实质上的对象，只有通过new关键字才会产生出对象，这时系统才会分配内存空间给对象，其方法才可以供外部调用。我们有时候希望无论是否产生了对象或无论产生了多少对象的情况下，某些特定的数据在内存空间里只有一份，例如所有的中国人都有个国家名称，每一个中国人都共享这个国家名称，不必在每一个中国人的实例对象中都单独分配一个用于代表国家名称的变量。



关键字—static

- `class Circle{`
 `private double radius;`
 `public Circle(double radius){this.radius=radius;}`
 `public double findArea(){return Math.PI*radius*radius;}}`
- 创建两个Circle对象
 - `Circle c1=new Circle(2.0);` `//c1.radius=2.0`
 - `Circle c2=new Circle(3.0);` `//c2.radius=3.0`
- Circle类中的变量radius是一个实例变量(instance variable), 它属于类的每一个对象, 不能被同一个类的不同对象所共享。
- 上例中c1的radius独立于c2的radius, 存储在不同的空间。c1中的radius变化不会影响c2的radius, 反之亦然。

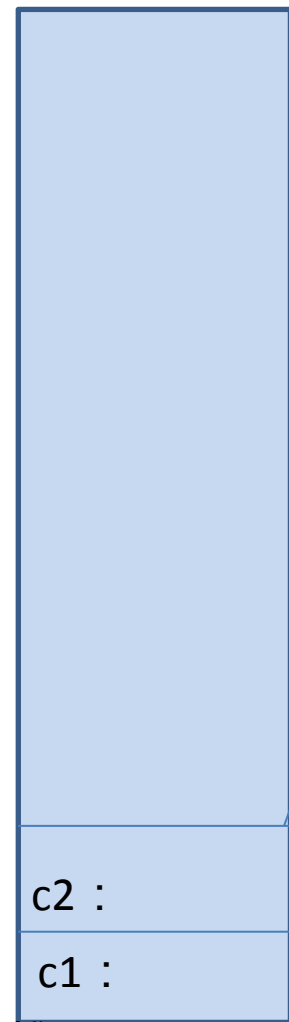
如果想让一个类的所有实例共享数据, 就用类变量!

类属性、类方法的设计思想

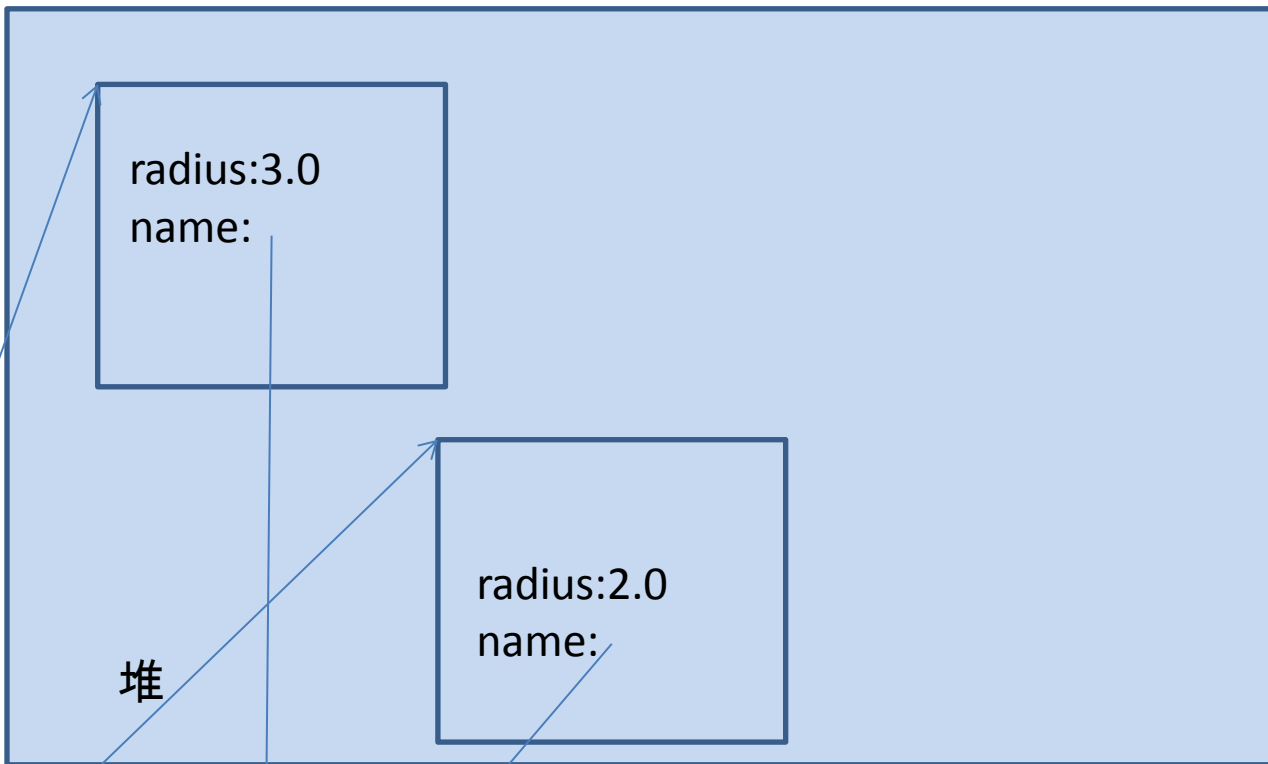
- 类属性作为该类各个对象之间共享的变量。在设计类时,分析哪些类属性不因对象的不同而改变,将这些属性设置为类属性。相应的方法设置为类方法。
- 如果方法与调用者无关,则这样的方法通常被声明为类方法,由于不需要创建对象就可以调用类方法,从而简化了方法的调用


```
class Circle {  
    private double radius;  
    public static String name = "这是一个  
圆";  
    public static String getName(){  
        return name;  
    }  
    public Circle(double radius) {  
        getName();  
        this.radius = radius;  
    }  
    public double findArea() {  
        return Math.PI * radius * radius;  
    }  
    public void display(){  
        System.out.println("name:"+name+"ra  
dius:"+radius);  
    }  
}
```

```
public class TestStatic {  
    public static void main(String[]  
args) {  
        Circle c1 = new Circle(2.0);  
        Circle c2 = new Circle(3.0);  
        c1.display();  
        c2.display();  
    }  
}
```



栈



堆



静态域

关键字—final

- 在Java中声明类、属性和方法时，可使用关键字final来修饰,表示“最终”。
 - final标记的类不能被继承。提高安全性，提高程序的可读性。
 - final标记的方法不能被子类重写。
 - ✓ Object类中的getClass()。
 - final标记的变量(成员变量或局部变量)即称为常量。名称大写，且只能被赋值一次。
 - ✓ final标记的成员变量必须在声明的同时或在每个构造方法中或代码块中显式赋值，然后才能使用。
 - ✓ final double PI=3.14;

1.final修饰类

```
final class A{  
}  
class B extends A{    //错误，不能被继承。  
}
```

中国古代，什么人不能有后代，就可以被final声明，称为太监类！

2.final修饰方法

```
class A{  
    public final void print(){  
        System.out.println("A");  
    }  
}  
class B extends A{  
    public void print(){ //错误，不能被重写。  
        System.out.println("尚硅谷");  
    }  
}
```

3.final修饰变量——常量

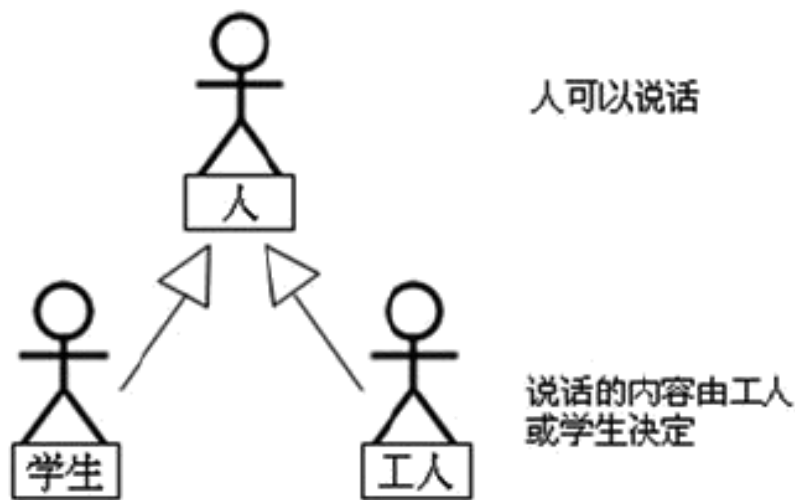
```
class A{  
    private final String INFO = "atguigu"; //声明常量  
    public void print(){  
        //INFO = "尚硅谷";  
    }  
}
```

常量名要大写，内容不可修改。——如同古代皇帝的圣旨。

- **static final: 全局常量**

抽象类(**abstract class**)

- 随着继承层次中一个个新子类的定义，类变得越来越具体，而父类则更一般，更通用。类的设计应该保证父类和子类能够共享特征。有时将一个父类设计得非常抽象，以至于它没有具体的实例，这样的类叫做**抽象类**。



关键字—**abstract**(抽象)

- 用**abstract**关键字来修饰一个类时，这个类叫做**抽象类**；
- 用**abstract**来修饰一个方法时，该方法叫做**抽象方法**。
 - 抽象方法：只有方法的声明，没有方法的实现。以分号结束：**abstract int abstractMethod(int a);**
- 含有抽象方法的类必须被声明为抽象类。
- 抽象类不能被实例化。抽象类是用来被继承的，抽象类的子类必须重写父类的抽象方法，并提供方法体。若没有重写全部的抽象方法，仍为抽象类。
- 不能用**abstract**修饰属性、私有方法、构造器、静态方法、**final**的方法。

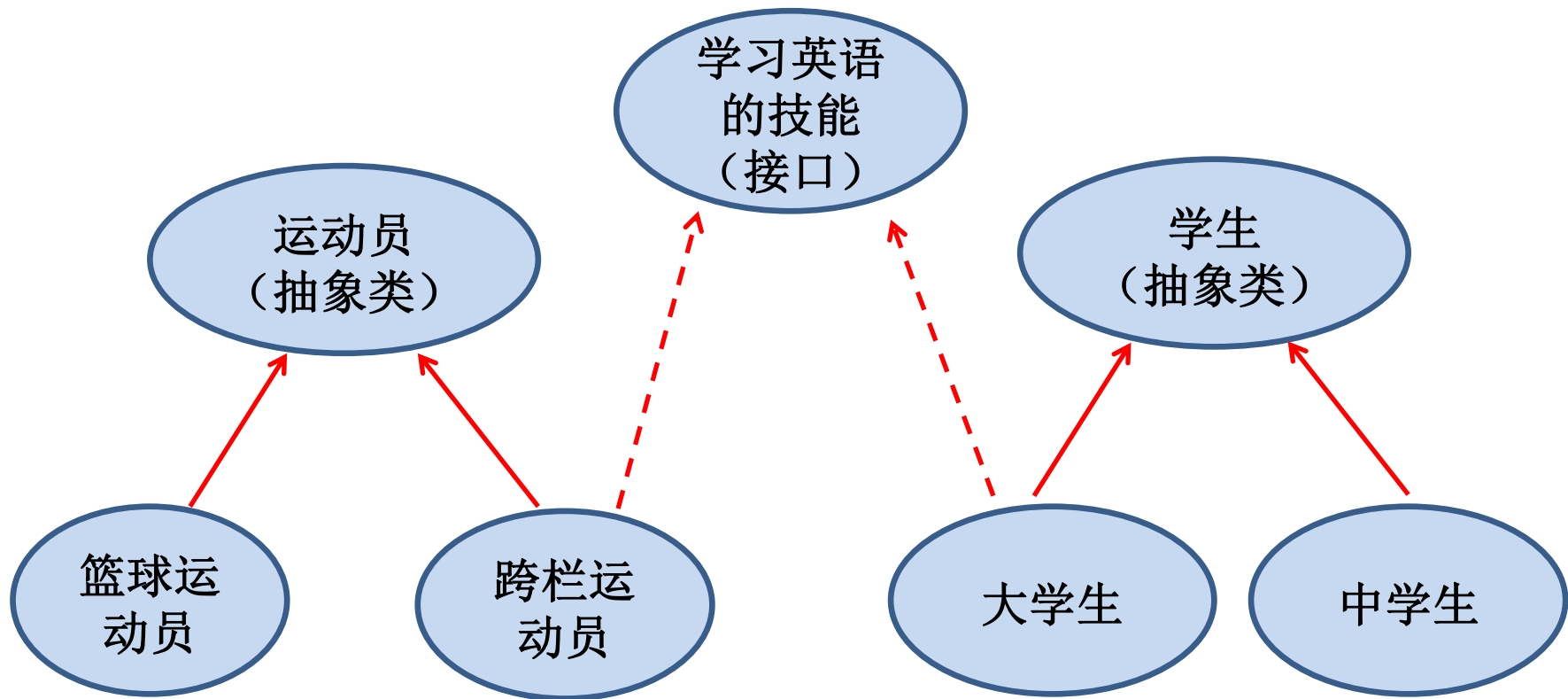
抽象类举例

```
abstract class A{  
    abstract void m1( );  
    public void m2( ){  
        System.out.println("A类中定义的m2方法");  
    }  
}  
  
class B extends A{  
    void m1( ){  
        System.out.println("B类中定义的m1方法");  
    }  
}  
  
public class Test{  
    public static void main( String args[ ] ){  
        A a = new B( );  
        a.m1( );  
        a.m2( );  
    }  
}
```

关键字—**interface**(接 口)

- 有时必须从几个类中派生出一个子类，继承它们所有的属性和方法。但是，Java不支持多重继承。有了接口，就可以得到多重继承的效果。
- 接口(**interface**)是**抽象方法**和**常量值**的定义的集合。
- 从本质上讲，接口是一种**特殊的抽象类**，这种抽象类中只包含常量和方法的定义，而没有变量和方法的实现。
- 实现接口类：
 - `class SubClass implements InterfaceA{ }`

关键字—interface(接口)



关键字—interface(接口)

● 接口的特点：

- 用interface来定义。
- 接口中的所有成员变量都默认是由public static final修饰的。
- 接口中的所有方法都默认是由public abstract修饰的。
- 接口没有构造器。
- 接口采用多继承机制。

● 接口定义举例

```
public interface Runner {  
    int ID = 1;  
    void start();  
    public void run();  
    void stop();  
}
```



```
public interface Runner {  
    public static final int ID = 1;  
    public abstract void start();  
    public abstract void run();  
    public abstract void stop();  
}
```


接口应用举例(1)

```
public interface Runner {  
    public void start();  
    public void run();  
    public void stop();  
}  
  
public class Person implements Runner {  
    public void start() {  
        // 准备工作：弯腰、蹬腿、咬牙、瞪眼  
        // 开跑  
    }  
    public void run() {  
        // 摆动手臂  
        // 维持直线方向  
    }  
    public void stop() {  
        // 减速直至停止、喝水。  
    }  
}
```

接口应用举例(2)

- 一个类可以实现多个无关的接口

```
interface Runner { public void run();}  
interface Swimmer {public double swim();}  
class Creator{public int eat(){...}}  
class Man extends Creator implements Runner ,Swimmer{  
    public void run() {.....}  
    public double swim() {.....}  
    public int eat() {.....}  
}
```

- 与继承关系类似，接口与实现类之间存在多态性

```
public class Test{  
    public static void main(String args[]){  
        Test t = new Test();  
        Man m = new Man();  
        t.m1(m);  
        t.m2(m);  
        t.m3(m);  
    }  
    public String m1(Runner f) { f.run(); }  
    public void m2(Swimmer s) {s.swim();}  
    public void m3(Creator a) {a.eat();}  
}
```

三、java基础其它内容

- java集合类 & 数组
- 多线程
- IO流:input & output “人与程序互动”
- 网络编程
- JDK新特性: 泛型、枚举、可变参数、自动装箱/拆箱

关于此处各章节详细内容, 可关注微博@尚硅谷-宋红康索取

