面向对象程序设计

第 4 讲 OOP 概述封装隐藏

刘进

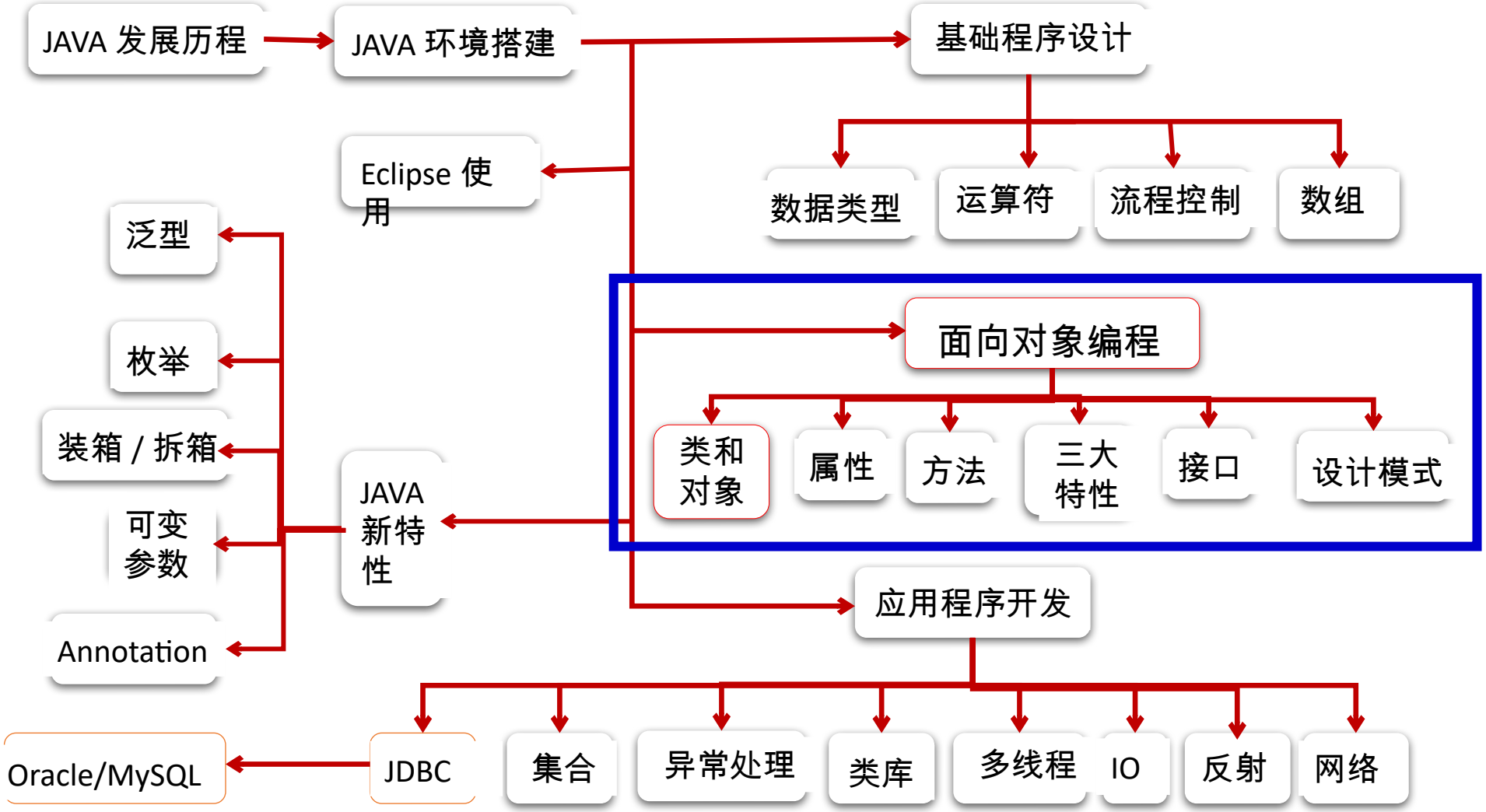
223065259 7@qq.com

OOP 教辅 2022 秋季 QQ 群：

837966056

Java 基础知识图解

主要知识点



第 4 讲 Java 面向对象编程 -1

语言设计
方式和思想
形式语义学

4.1 面向对象概述

4.2 面向对象特征之封装与隐藏

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

4. 其它关键字

学习内容

4.1 面向对象与面向过程

4.2 java 语言的基本元素：类和对象

4.3 **类的成员之一**：属 性

4.4 **类的成员之二**：方 法

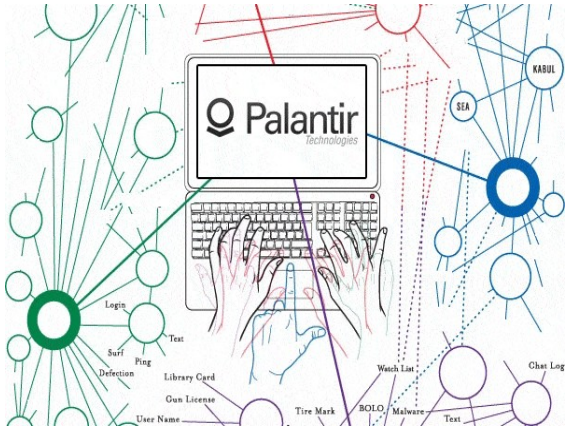
4.5 对象的创建和使用

4.6 再谈方法

4.7 **面向对象特征之一**：封装和隐藏

4.8 **类的成员之三**：构造器（构造方法）

4.9 几个关键字： this 、 package 、 import



封装和隐藏
该知道的让知道
不该知道的不让知道
任务驱动的权限管理

4.1 面向对象与面向过程

面向过程：
1）功能行为有哪些？
2）功能行为的过程如何？

- **面向对象 (OOP) 与面向过程**
  - 二者都是一种思想，面向对象是相对于面向过程而言的。面向过程，强调的是功能行为。面向对象，将功能封装进对象，强调具备了功能的对象。
  - 面向对象更加强调运用人类在日常的思维逻辑中采用的思想方法与原则，如抽象、分类、继承、聚合、多态等。
- **面向对象的三大特征**
  - 封装 (Encapsulation)
  - 继承 (Inheritance)
  - 多态 (Polymorphism)

面向对象：相互交互的实体
1）实体有哪些？（功能归属于实体）
2）实体之间通过什么关系来发生交互

OOP: Object Oriented Programming
面向过程： procedure oriented programming

面向对象

基本方式

道法自然

五禽戏

vs

少林洪拳

面向过程：

重点分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，解决问题时将各个实现步骤依次调用出来就行了；

1. 打开冰箱

2. 把大象装进冰箱

3. 把冰箱门关住

面向过程

面向过程：

不再是为了完成一个步骤，而是为叙某个事物在整个解决问题中的位置和行为，即找出（知识的显性化）构成问题事务中的各个对象（名词性概念），描述它们和它们之间的关系。

概念主体和他们之间的关系

例子：**人把大象装冰箱**

人
冰箱
大象

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

```
冰箱 {
   开门 (){}  关门 (){}
}
```

```
大象 {
   进入 (){ }
}
```

面向对象

五子棋

面向过程：

重点分析出解决问题所需要的步骤；

设计思路就是首先分析问题的步骤：

1、开始游戏，

2、黑子先走，

3、**绘制画面**，

4、判断输赢，

5、轮到白子，

6、**绘制画面**，

7、判断输赢，

8、返回步骤 2 ，

9、输出最后结果。

将上述步骤用不同的方法来实现。

面向对象：

重点分析找出各个对象（概念）并描述关系；

五子棋可以分为：

1、**黑白双方**，这两方的行为是一模一样的，

2、**棋盘系统**，负责绘制画面，

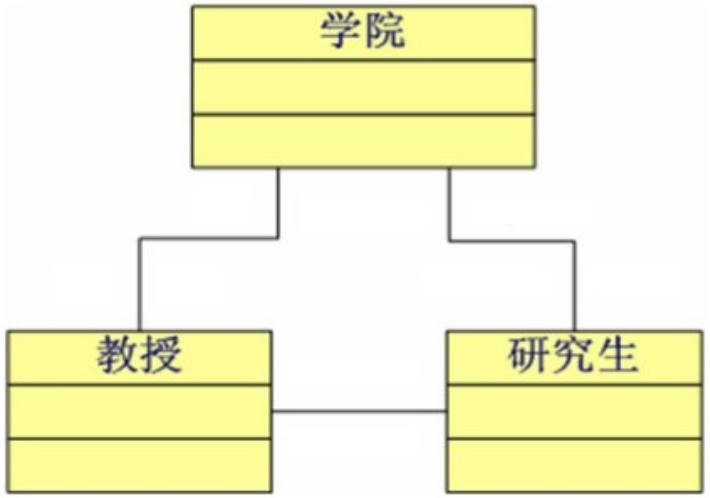
3、**规则系统**，负责判定诸如犯规、输赢等。

交互关系：

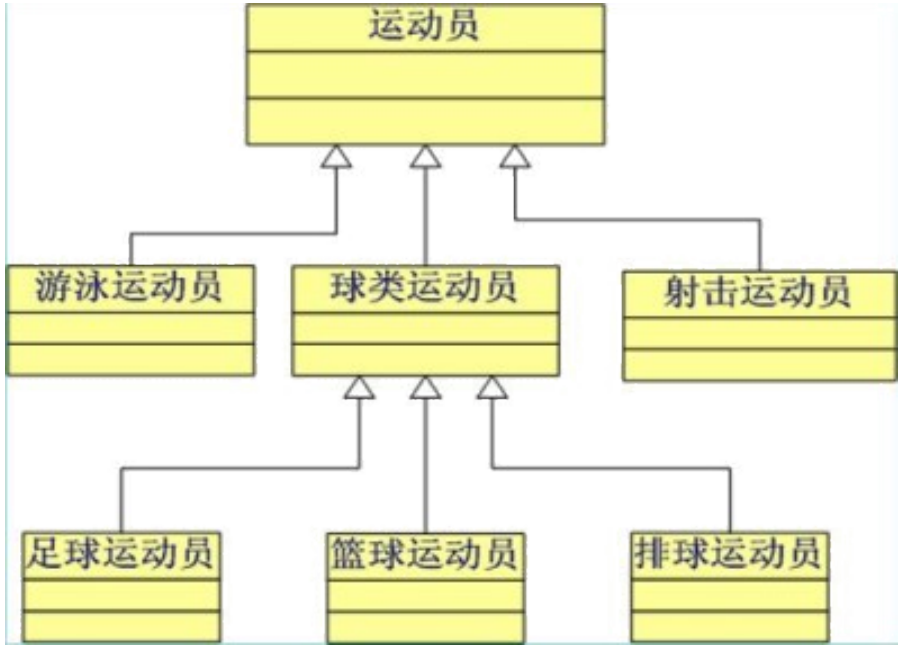
第一类对 象（**玩家对象**）负责接受用户输入，并告知第二类对象（**棋盘对象**）棋子布局的变化；

棋盘对象接收到了棋子的变化就要负责在屏幕上面显示出这种变化，同时利用 第三类对象（**规则系统**）来对棋局进行判定。

类与类之间的关系

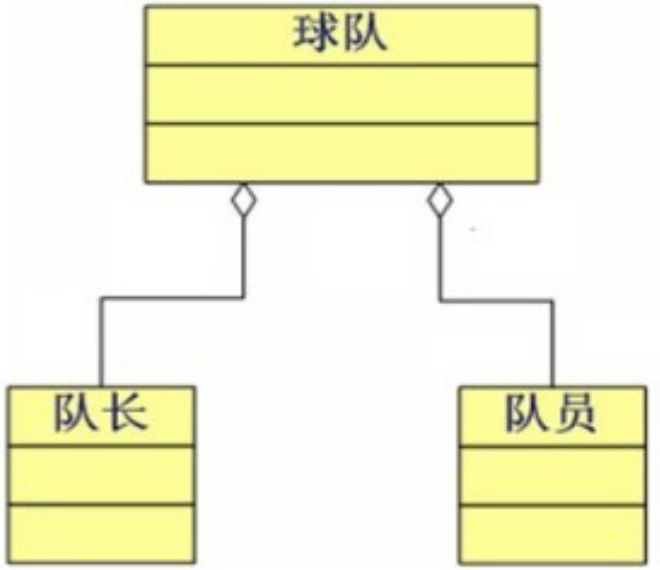


关联关系

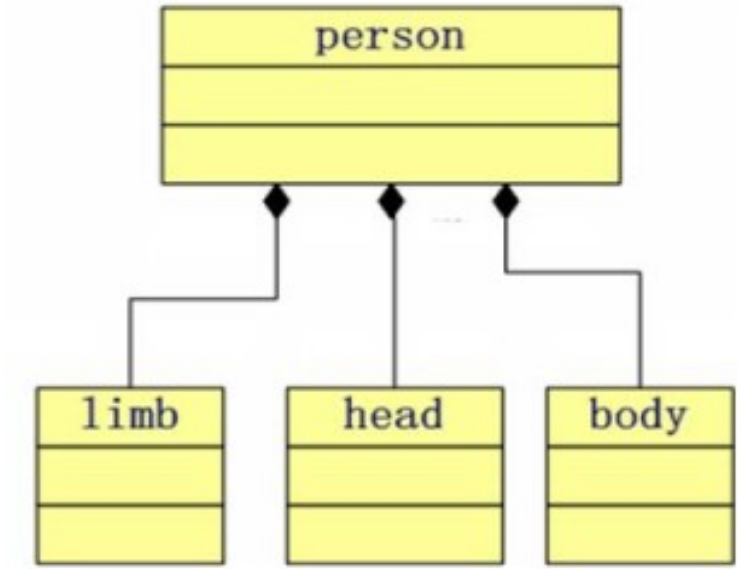


继承关系

类与类之间的关系



聚 集



组
合

聚合关系

面向对象的思想概述

- 程序员从执行者转化成了指挥者。

- 完成需求时：
  - 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 (class) 和对象 (object) 是面向对象的核心概念。
  - 类是对一类事物描述，是抽象的、概念上的定义
  - 对象是实际存在的该类事物的每个个体，因而也称实例 (instance) 。
- “万事万物皆对象”

面向对象

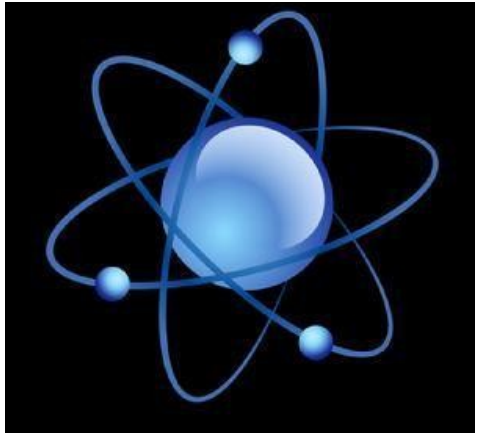
程序设计

基层结果

类（对象）

类型（实例）

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。

- 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

面向对象

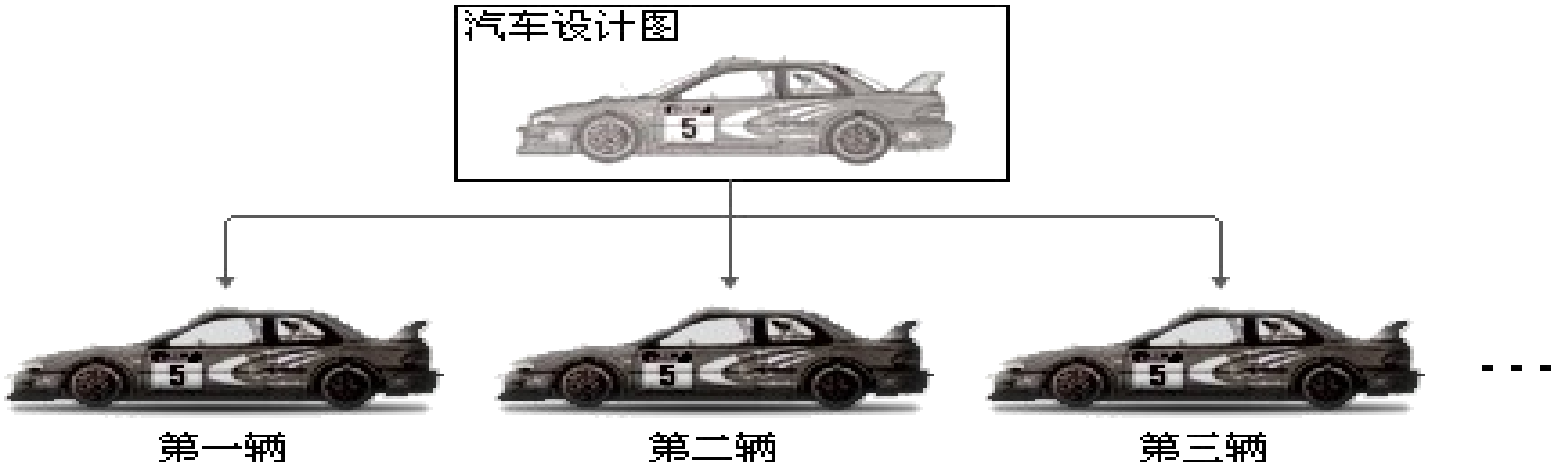
基本方式

静态属性和

动态行为

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

面向对象的思想概述



- 可以理解为：**类 = 汽车设计图；对象 = 实实在在的汽车**
- 面向对象程序设计的重点是**类的设计**
- 定义类其实是定义类中的成员（成员变量和成员方法）

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

面向对象编程举例：

确定概念

定义类（类型）

类的成员构成 version 1.0

类的成员构成
verson 2.0

```java
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;
    }
}
```

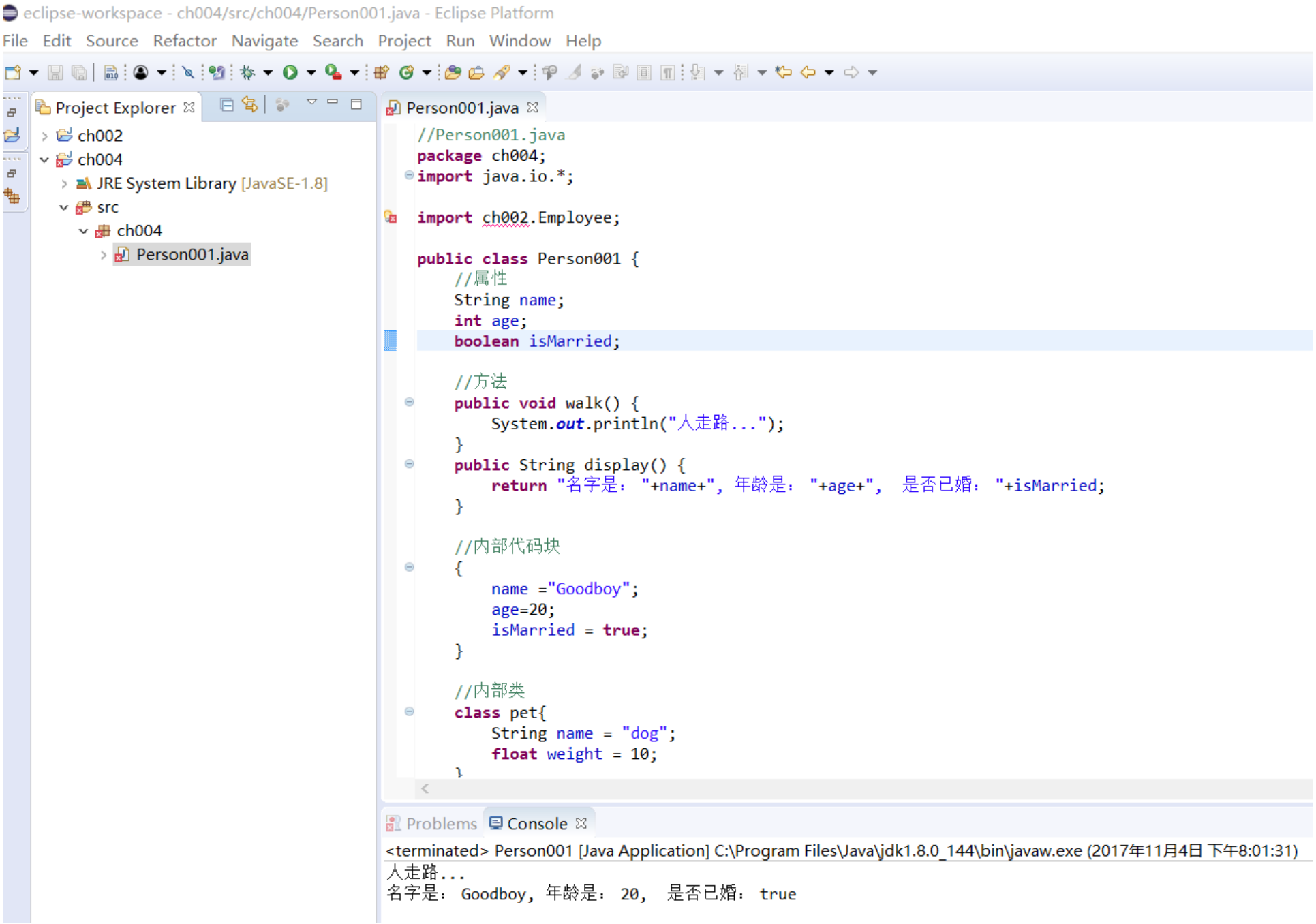
细化细节

```
//Person001.java
package ch004;

public class Person001 {
// 属性
String name;
int age;
boolean isMarried;

// 方法
public void walk() {
System.out.println(" 人走路...");
}
public String display() {
return " 名字是： "+name+", 年龄是： "+age+", 是否已婚： "+isMarried;
}

// 内部代码块
{ name ="Goodboy";
  age=20;
  isMarried = true;
}

// 内部类
class pet{
String name = "dog";
float weight = 10;
}

// 执行代码
public static void main(String args[]){
    Person001 p1 = new Person001();
    p1.walk();
    System.out.println(p1.display());
}
}
```

细化细节

合法的语法形态

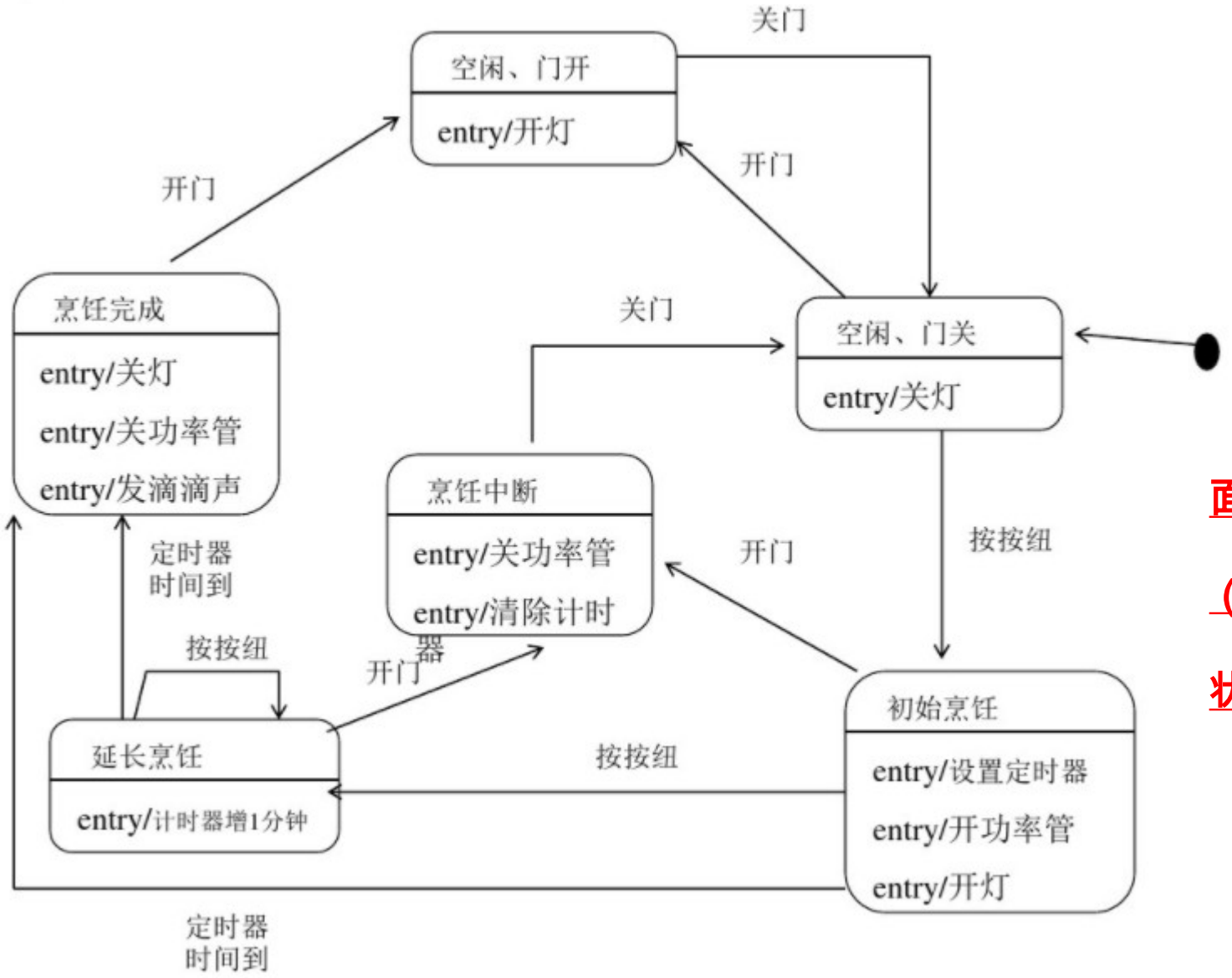
Well formed



面向对象

编程举例：

编译调试

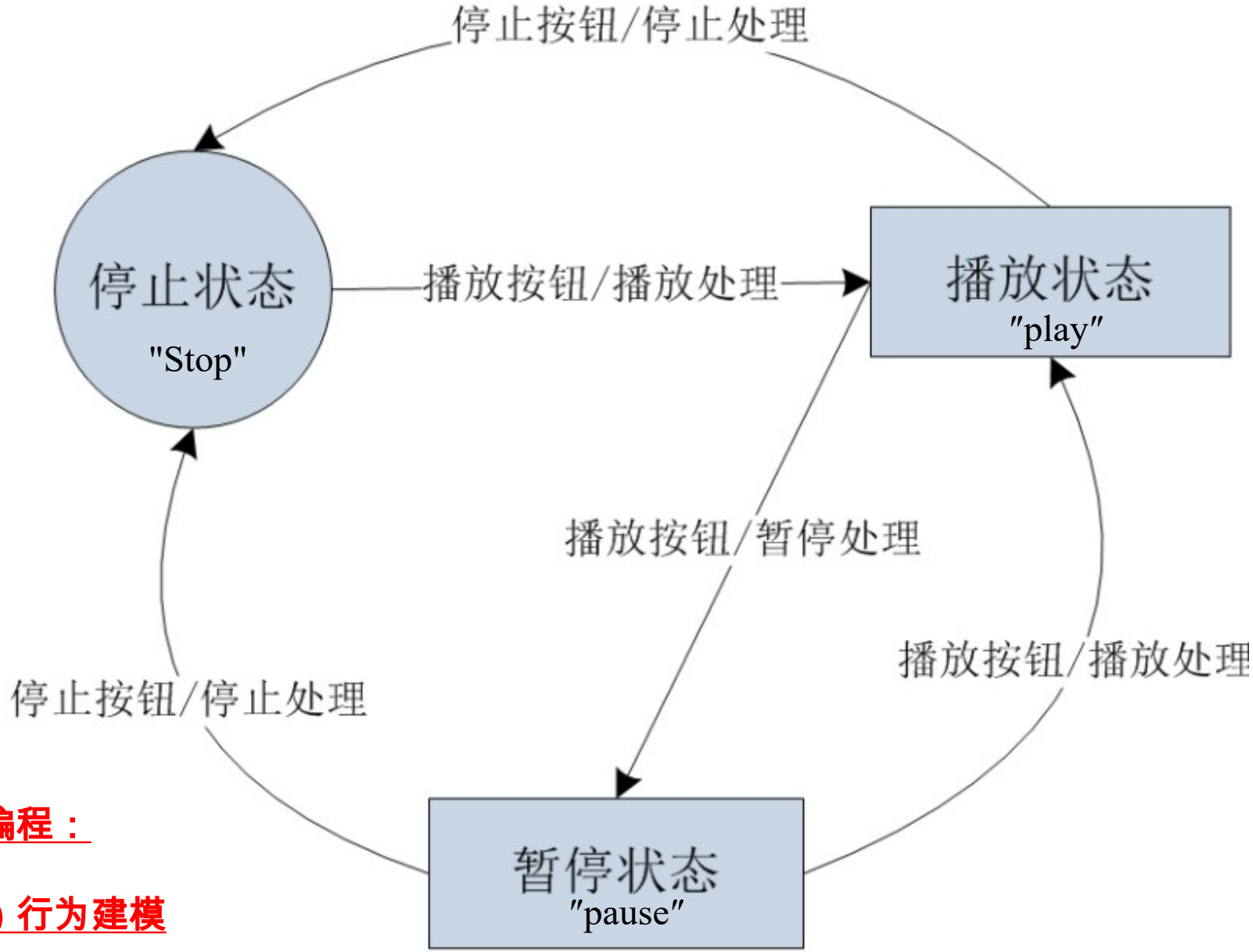


面向对象编程：

（动态的）行为建模

状态机、序列图

```
public class MP3Player{
// 属性
String[] musicitem ={"one", "two", "three"};
String[] button ={"play", "pause", "stop"};
String[] state ={"on", "off"};
String[] currentMusic = "one";

// 方法
public void stateButton() {
If (Button == "off")  {Button = "on";}
else  {Button = "off";}
}

public void playButton() {
   Button = "play";
}

public void pauseButton() {
   Button = "pause";
}

public void stopButton() {
   Button = "stop";
}

public void setMusic (string music){
   currentMusic = music;
}
```



面向对象编程：

（动态的）行为建模

状态机、序列图

面向对象的分解 - 系统分解从对象开始

- **根据问题领域关键抽象概念对系统进行分解，强调代理之间的交互**（更新主控文件 master file）

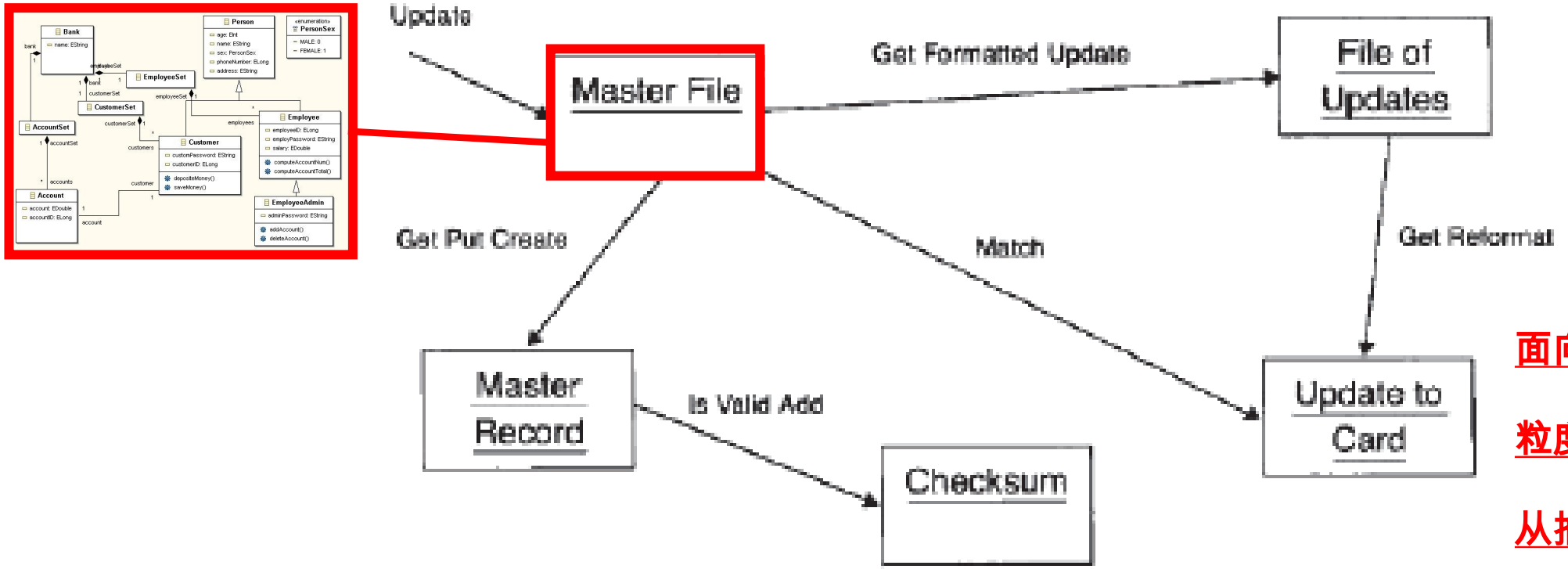


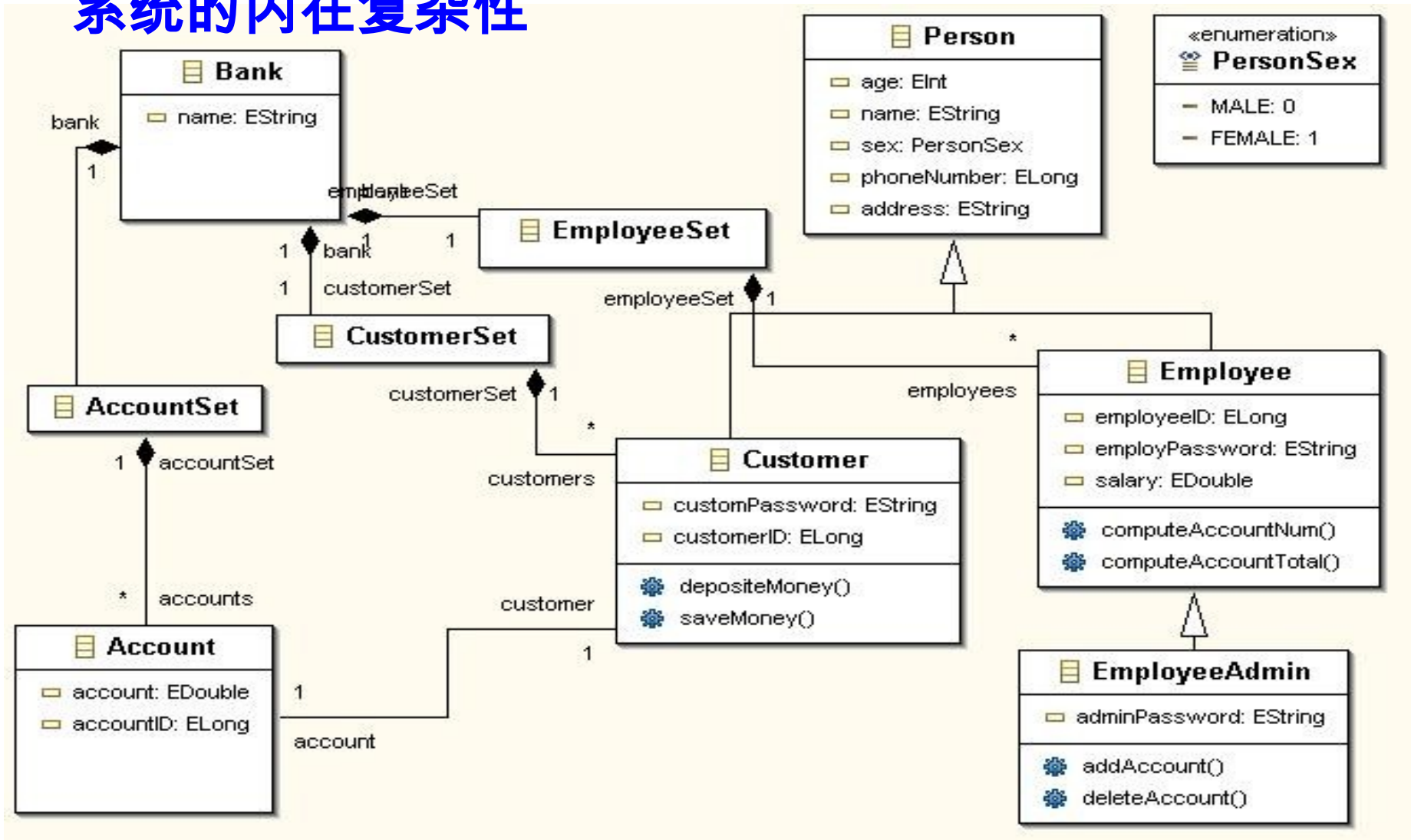
图 1-4　面向对象的分解

面向对象分析设计：

粒度从粗到细，

从抽象到具体

- 根据问题领域关键抽象概念对系统进行分解，更加偏重于领域经验，以代理之间交互的方式用于组织软件系统的内在复杂性



面向对象分析

设计：

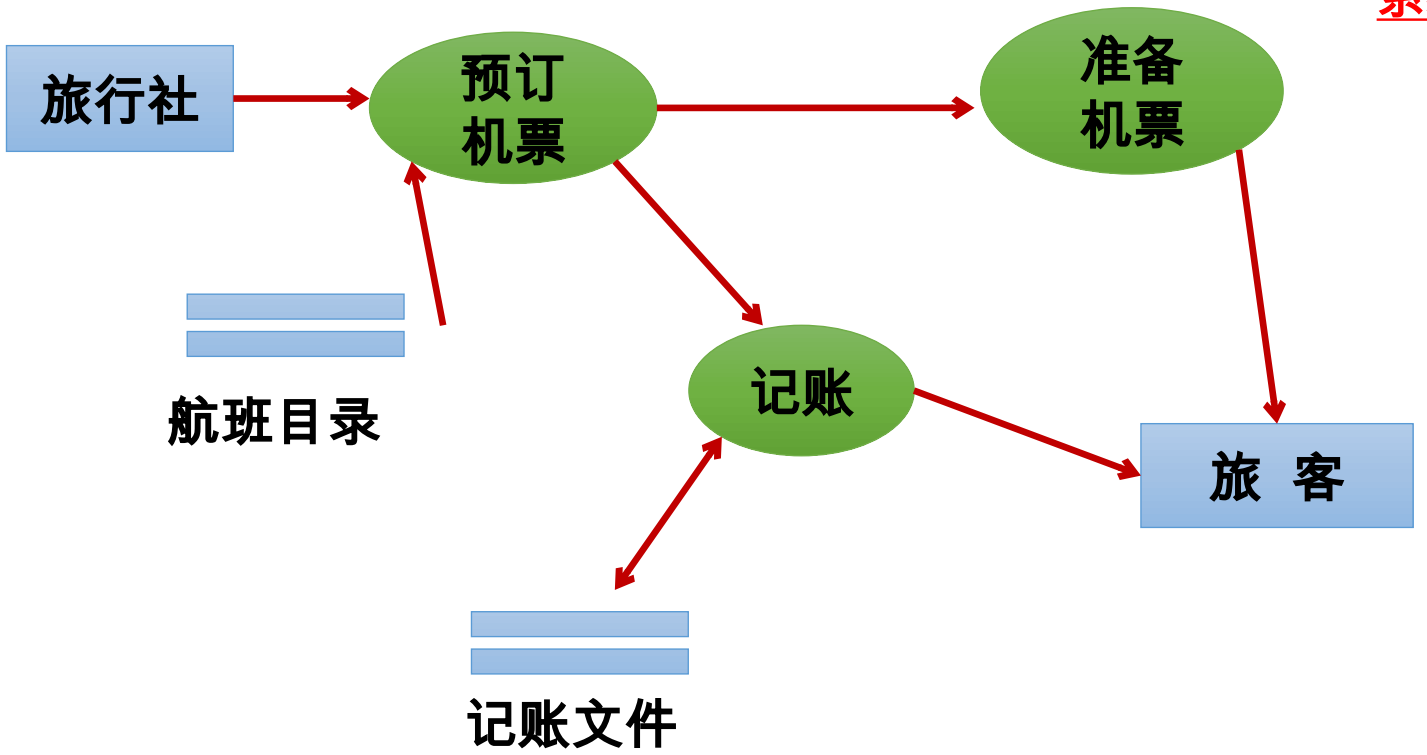
粒度从粗到细，

从抽象到具体

4. 抽象出下面系统中的“类”及其关系。

面向对象分析设计：

系统分解



面向对象的分解 - 系统分解从对象开始

- 面向对象通过复用共同的机制得到一些精炼的系统表达，**支持增量式的系统建模**。
- 面向对象的设计是基于稳定的中间状态的，具有随时间变化而变化的弹性。
- **将复杂系统巨大的状态空间进行了关注点分离，直接关注了软件的内在复杂性**。

业务建模

需求分析

概念建模

PIM

PSM

分解 - 拆解复杂问题

分解：就是把复杂的大问题，拆解成更可执行、更好理解的小步骤。

举例：为全家做一个暑假出游计划，全家人这么多，听起来就很繁琐，学会分解的孩子就会化繁为简，将难题拆解为几个容易解决的小任务 - 确定目的、目的地、安排游玩行程、预定酒店机票等。

抽象能力，锻炼找出问题本质的能力

抽象：是指聚焦最重要的信息，忽视无用细节。简单来说就是找到问题的本质，过滤掉其他无关紧要的因素。

在编程世界里，包含“子系统、模块、包、类、方法和语句”等不同等级的抽象，需要通过编程的学习不断提升聚焦关键信息的能力。抽象思维高的人，能在大量信息中抓住关键信息，提高分析效率。

例如：不论是在做语文还是英语的阅读理解中，抽象思维能力强的人总能很容易地找出关键句和中心思想，学习会更加轻松。

模式识别 - 锻炼整合“重复规律”

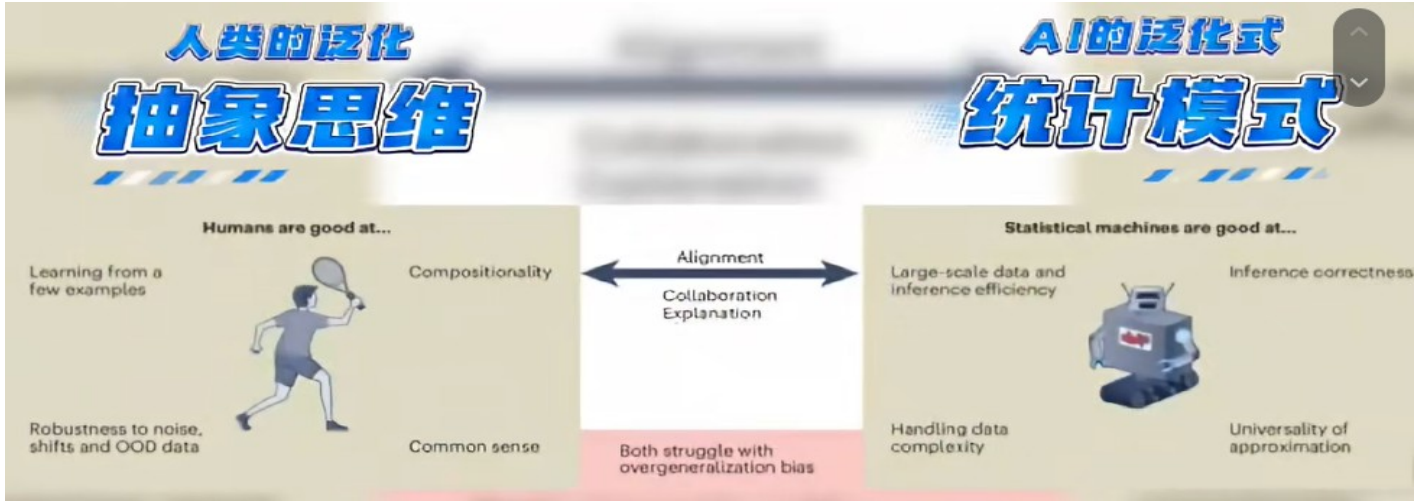
模式识别：就是识别不同问题的模式和趋势 (共同点) 的过程，在我们的经验库里找出类似问题的解决办法，套用解决。识别的模式越多，解决问题的速度也就越快。

举例：如果需要画一百辆不同的自行车

运用“模式识别”：不难发现自行车的共同点有：车架、脚蹬、链条、车轮 ... 按照这些特点建立起自行车的“模式”。有了这个模式就能批量画自行车，不同的自行车只需要更换局部特征比如颜色、大小等等，就能轻松解决这个问题。

编程思维 -- 人类的泛化靠抽象思维

编程思维：由分解、模式识别、抽象、算法四个步骤组成，一种强调逻辑性、系统性、创造性和实践性的思维模式。自觉快速反应 + 深度思考，自动忽略无关细节，抓住本质特征。将复杂问题分解成简单的步骤，并用清晰的逻辑和算



域外泛化　VS　域内泛化；

创造性思考、伦理判断、模糊决策（极端解难题）VS 数据处理、重复任务（常规抄作业）

抽象 - 控制复杂性

I、抽象的概念

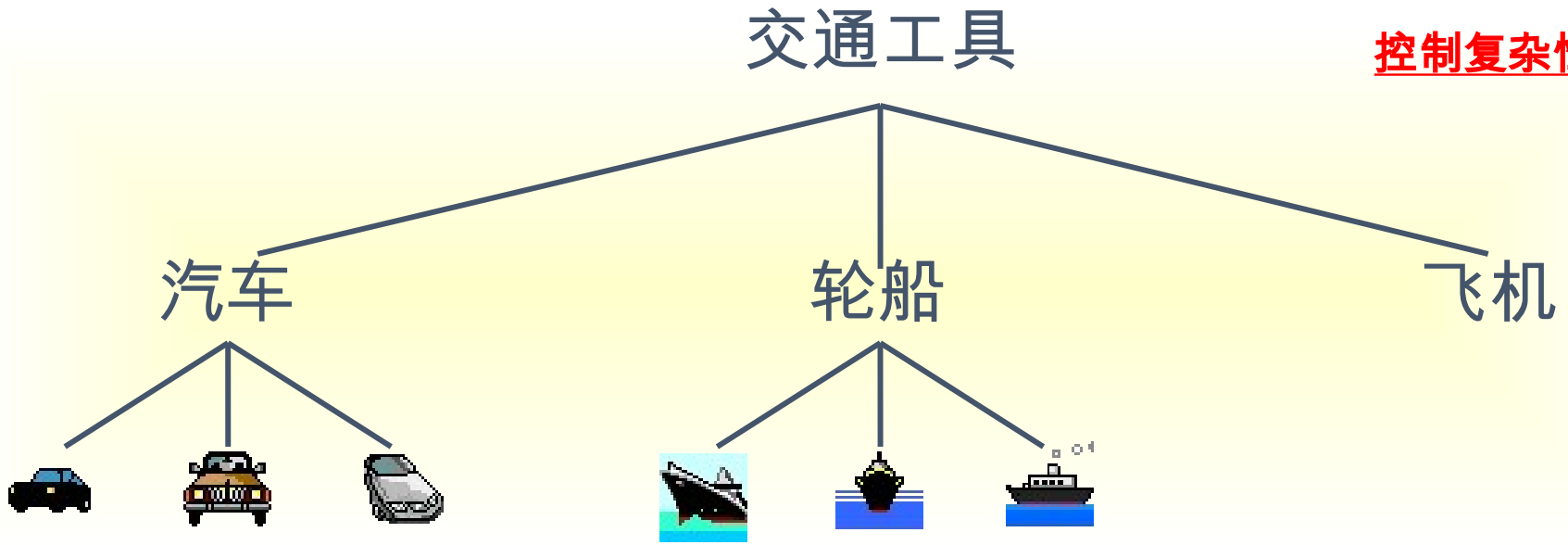
抽象代表着一个对象的本质特征，这个特征将这个对象与所有其他种类的对象区别开来；

抽象是通过从特定的实例中抽取共同的性质以形成一般化的概念的过程；

抽象具有层次；

面向对象分析设计：

控制复杂性



面向对象中的抽象

面向对象分析设计：

控制复杂性（抽象）

模型及服务

“models as services”

面向对象抽象的原理（面向对象计算的本质）

数据抽象、行为共享、进化、确定性

1、数据抽象

为程序员提供了一种对数据和为操作这些数据所需要的算法的抽象；是面向对象方法的核心，包括：

类

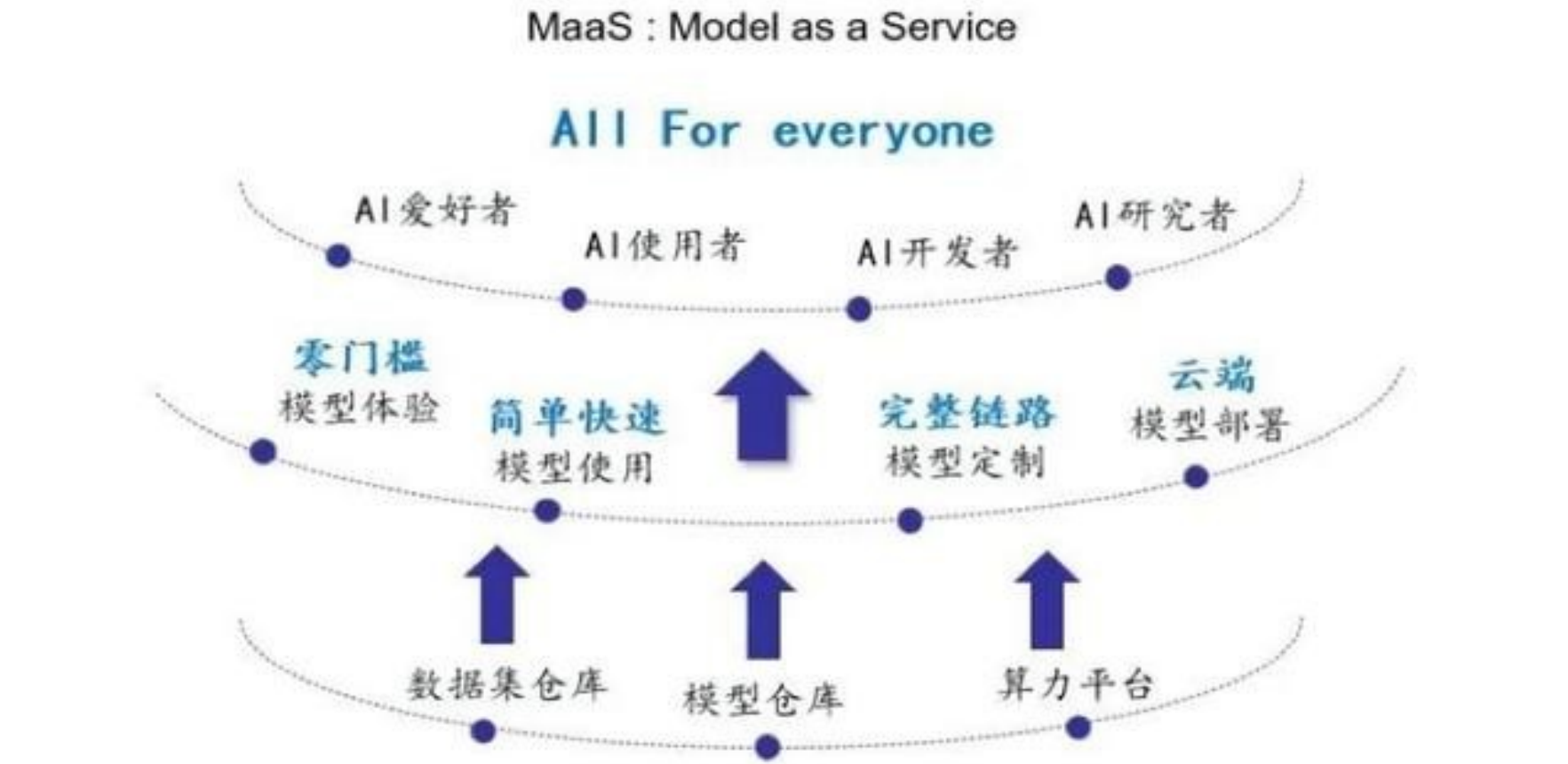
公有成员

模块化：构成了面向对象计算的本质；

信息隐藏：将一个模块的细节部分对用户隐藏起来，用户只能通过一个受保护的接口来访问某个模块，而不能直接访问一个模块内部的细节；

MaaS 是“ Model as a Service ”的缩写，即**“模型即服务”，指以模型为核心提供服务**。 MaaS 最底层的含义是要把模型作为重要的生产元素，围绕模型的生命周期设计产品和技术，从模型的开发入手，包括数据处理、特征工程、模型的训练和调优、模型的服务等，提供各种各样的产品和技术

MaaS 基本产业架构



资料来源：阿里云栖大会，HTI

在阿里云发起的 AI 模型社区魔搭中，集聚了 180 多万 AI 开发者和 900 多个 AI 模型。模型贡献者基本覆盖国内大模型赛道核心玩家，如百川智能、哔哩哔哩、粤港澳大湾区数字经济研究院（ IDEA 研究院）、澜舟科技、清华大学人工智能研究院、深势科技、浙江大学、智谱 AI 等。

“有算力、无模型，不够。有模型，无生态，也不够。发展大模型，算力、模型、生态，缺一不可。”

简单来说， MaaS 最核心的就是让模型的使用更简洁，简单几行代码就可以调用模型。

这就涉及 AI 模型落地应用的现状，即一个 AI 模型难以覆盖各行各业的 AI 应用需求，面对新场景往往需要进行二次开发或优化，否则许多模型难以适配到特定环境应用中。而 AI 模型定制化门槛较高，同时目前缺乏 AI 模型开发和使用交流分享的平台。也就是说当开发者遇到相关问题后，无法找到对应的模型服务，也比较难找到人来解答相关问题。

“最终的目标是，甚至小学生也可以调用模型，能做业务系统的开发。”

魔搭是阿里达摩院与中国计算机学会（ CCF ）开源发展委员会在 2022 年联合推出的国内首个 AI 模型开源社区，把 300 多个模型开放给中国的 AI 研究者与团队，涵盖了自然语言处理，视觉、语音、多模态等模型。

新的模型工具 ModelScopeGPT（魔搭 GPT），目的是有效帮助使用者在海量模型里面找到最合适的模型，“复杂的系统需要多个模型完成联合的任务，今天可以通过这样的一个流程自动化把各种模型融合在一起。

比如，用户在魔搭 GPT 的对话框输入任务：“用 20 字描述一款新的 VR（虚拟现实）眼镜，并用女声朗读，随后转成视频。”魔搭 GPT 会展示整个任务规划过程，先由中枢模型生成一段描述 VR 眼镜的文案，接着调用语音生成模型，生成语音并用女声念出，最后调用视频生成模型，输出最终的视频内容。过程中，魔搭 GPT 先后调用了一大二小 3 个模型。

大模型的研发不应该是一场少数机构的竞赛，而应该通过大小模型的协同进化走向更高级的应用，尤其是适应中国本土需求的应用。

面向对象中的抽象（续）

2、行为共享

对象

公有成员函数名

行为是由实体的外部接口定义的

行为共享指许多实体具有相同的接口，可增加系统的灵活性；

支持行为共享的方式

❊分类与层次分类

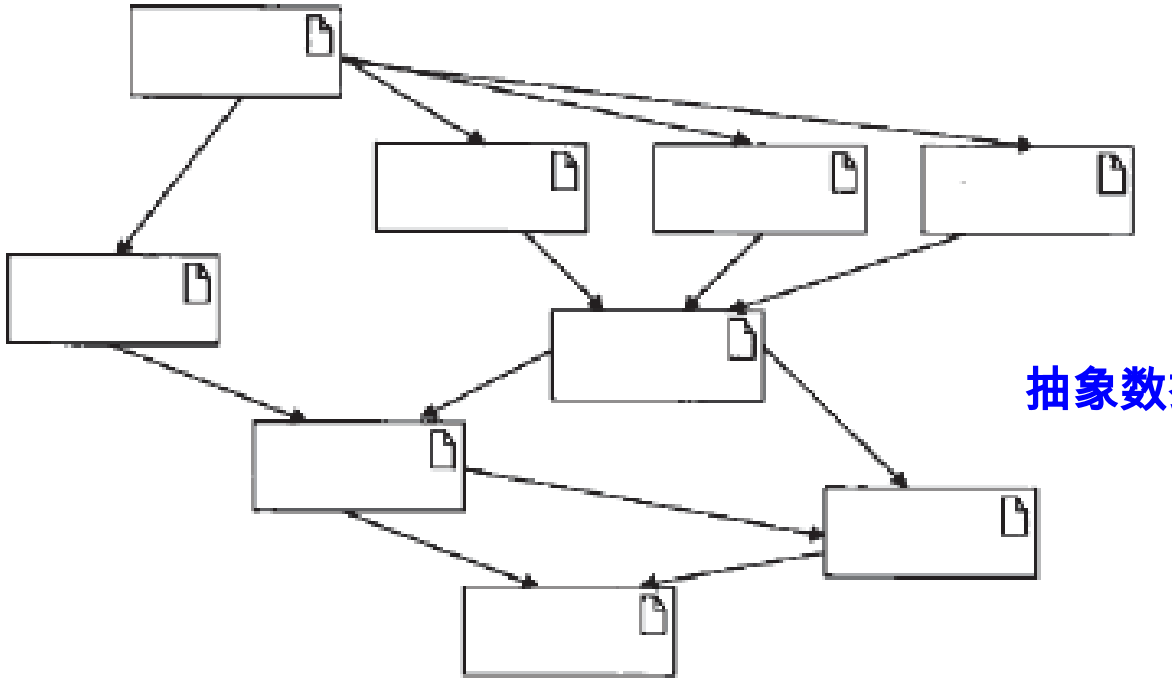
❊多态与继承

面向对象分析设计：

控制复杂性（行为共享）

行为统一于类型（类），

对象属性值千差万别



抽象数据类型

面向对象分析设计：

图或格

图 2-4　使用基于对象或面向对象编程语言的小型或中型应用的结构

这类语言的构建块是模块，它表现为逻辑上的一组类或对象，而不像早期语言那样是子程序。换言之，“如果过程和函数是动词，数据是名词，那么面向过程语言的程序就是围绕动词组织的，面向对象的程序就是围绕名词组织的”[6]。出于这个原因，小型或中型面向对象应用的物理结构表现为一个图，而不像面向算法的语言那样通常是一棵树。另外，基本上很少或没有全局数据。数据和操作被放在一个单元中，系统的基本逻辑构建块不再是算法，而是类或对象。

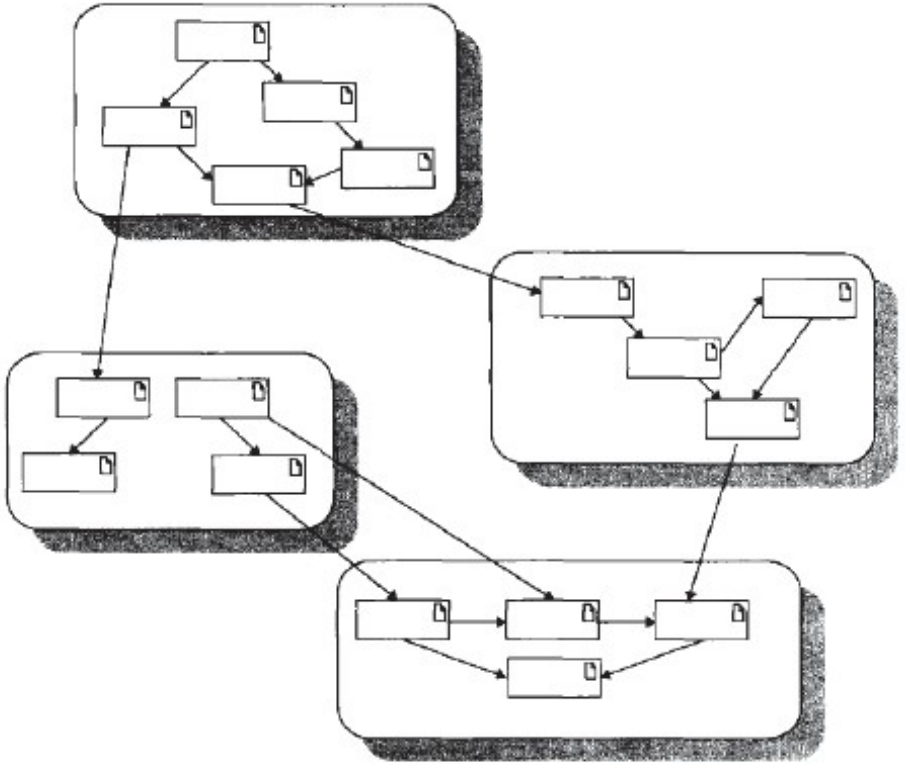


图 2-5　使用基于对象或面向对象编程语言的大型应用的结构

面向对象分析设计：

控制复杂性（巨型软件）

高内聚、低耦合

到目前为止，我们已经超越了“大型编程”（programming-in-the-large），必须面对“巨型编程”（programming-in-the-colossal）。对于非常复杂的系统，我们发现类、对象和模块提供了基本的抽象手段，但这还不够。幸运的是，对象模型在规模上可以扩展。在大型系统中，一组抽象可以构建在另一组抽象层之上。在任何一个抽象层上，都可以找到某些有意义的对象，它们可以协作实现更高层的行为。如果我们仔细看一组对象的实现，会看到另一组协作的抽象。

程序设计编程 OOP 的定义

“面向对象编程是一种实现的方法，在这种方法中，程序被组织成许多组相互协作的对象，每个对象代表某个类的一个实例，而类则属于一个通过继承关系形成的层次结构。”

这个定义有三个要点：（1）利用对象作为面向对象编程的基本逻辑构建块（第 1 章中介绍的“组成部分”层次结构），而不是利用算法；（2）每个对象都是某个类的一个实例；（3）类与类之间可以通过继承关系联系在一起（第 1 章中介绍的“是一个”层次结构）。一个程序可能看起来像是面向对象的，但是如果不满足这三点之一，它就不是一个面向对象的程序。具体来说，没有继承的编程显然不是面向对象的，那只是利用抽象数据类型在编程。

“面向对象”是面向概念的，而非面向算法（“面向过程则是”），自然地形成软件系统的边界；

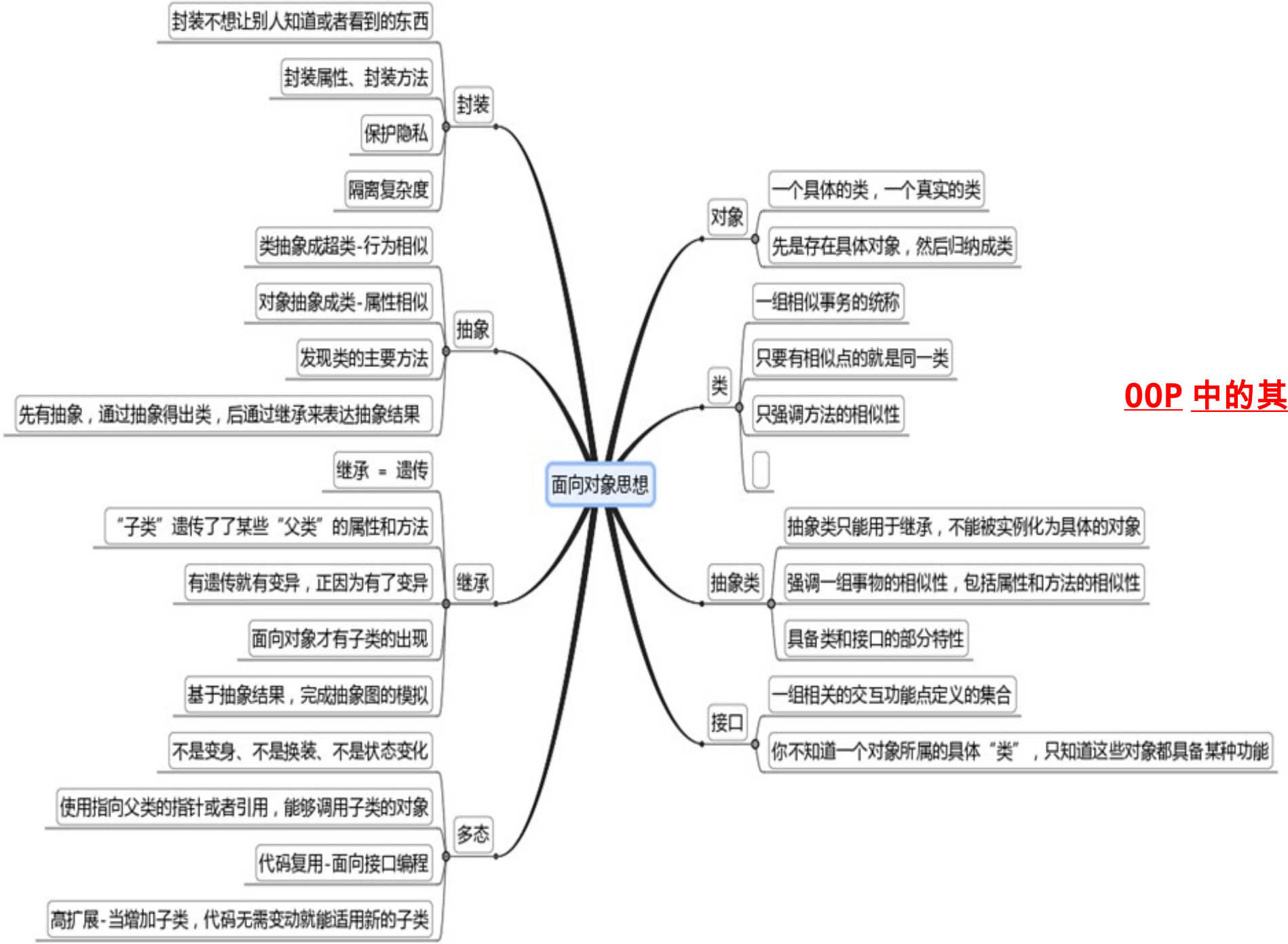
对象模型中的抽象

“抽象描述了一个对象的基本特征，可以将这个对象与所有其他类型的对象区分开来，因此提供了清晰定义的概念边界，它与观察者的视角有关。”

抽象关注一个对象的外部视图，所以可以用来分离对象的基本行为和它的实现。Abelson 和 Sussman 将这种行为/实现的分界称为抽象壁垒[45]，它是通过应用“最少承诺”原则来达成的。根据这个原则，对象的接口只提供它的基本行为，此外别无其他[46]。我们还想使用另一个原则，我们称之为“最少惊奇”原则，这个原则是指抽象捕捉了某个对象的全部行为，不多也不少，并且不提供抽象之外的惊奇效果或副作用。

对于给定的问题域决定一组正确的抽象，就是面向对象设计的核心问题。

原子性，确保软件中单元内部高内聚、单元之间松耦合


面向对象思想
封装
封装不想让别人知道或者看到的东西
封装属性、封装方法
保护隐私
隔离复杂度
抽象
类抽象成超类-行为相似
对象抽象成类-属性相似
发现类的主要方法
先有抽象，通过抽象得出类，后通过继承来表达抽象结果
继承
继承 = 遗传
“子类”遗传了了某些“父类”的属性和方法
有遗传就有变异，正因为有了变异
面向对象才有子类的出现
基于抽象结果，完成抽象图的模拟
多态
不是变身、不是换装、不是状态变化
使用指向父类的指针或者引用，能够调用子类的对象
代码复用-面向接口编程
高扩展-当增加子类，代码无需变动就能适用新的子类
对象
一个具体的类，一个真实的类
先是存在具体对象，然后归纳成类
类
一组相似事务的统称
只要有相似点的就是同一类
只强调方法的相似性
抽象类
抽象类只能用于继承，不能被实例化为具体的对象
强调一组事物的相似性，包括属性和方法的相似性
具备类和接口的部分特性
接口
一组相关的交互功能点定义的集合
你不知道一个对象所属的具体“类”，只知道这些对象都具备某种功能

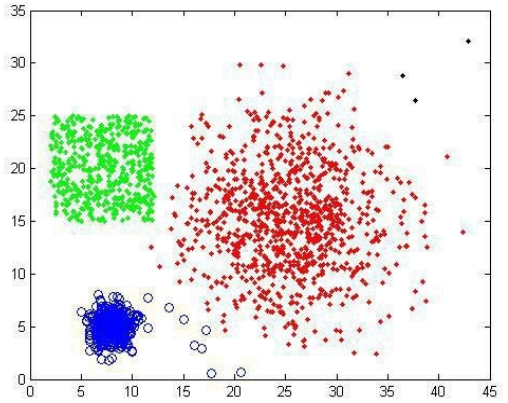
OOP 中的其它概念

类 Class

- 类 Class 是一组相似事物的统称
- 站在观察者的角度，具有相似点的事物就是同一类
- 类 = 方法 + 属性
- 设计原则：

  1、属性最小化，不可再分（适应性最强）。

  2、方法单一原则，一个方法只做一件事



类（类型）内部高度内聚、类型（对象）之间松耦合（保持必要的联系，类型 / 对象 可用不可见）

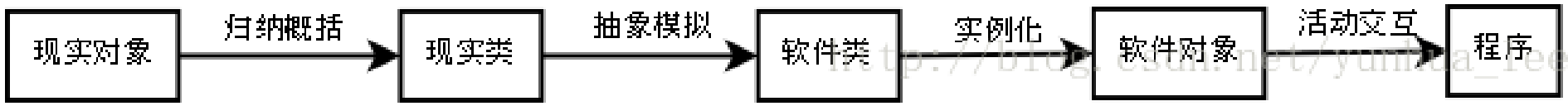
对象 Object

类与对象

回望面向对象程序

- 对象是类的实例

对于存在现实对象，人们根据自己的观察角度和要求将现实对象抽象成现实类。软件设计人员基于现实类模拟出软件类，最后在程序中将软件类实例化成软件对象，程序就是软件对象的活动和交互。



对象 Object

- **现实类与软件类并不一定是一 一对应的**

例如：令 ATM 机是现实世界真实存在的类，但在做软件设计的时候，可能将 ATM 机拆分为“ ATM 认证”、“ ATM 打印”、“ ATM 取款”等几个软件类，这些软件类互相配合，最后完成现实世界的 ATM 机器的功能。

- **软件类并不一定是现实存在的**

例如：策略（ Strategy ）是一个人的概念，也是软件领域常见的类，但并不是程序员可以直观观察到的

道法自然与人工系统：

概念与类（类型）

类的语法格式

修饰符 class 类名 {

属性声明；

方法声明；

}

说明：修饰符 public ：类可以被任意访问

类的正文要用 { } 括起来

举例：

```
public class  Person{
   private int age ;                    // 声明私有变量  age
   public void showAge(int i) { // 声明方法 showAge( )
            age = i;
   }
}
```

语法：类的形式

创建 Java 自定义类

语法、程序、人工系统

有一定便利性、有利于专注解决问题

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）

建议练习：

定义 Person 、 Animal 、 ClassRoom 、 Zoo 等类，加以体会。

4.3 类的成员之一：属性

语法、程序、人工系统

有一定便利性、有利于专注解决问题

但还要自己“编码”

- **语法格式：**

  修饰符 类型 属性名 = 初值 ；

  - **说明：修饰符 private:** 该属性只能由该类的方法访问。

    修饰符 public: 该属性可以被该类以外的方法访问。

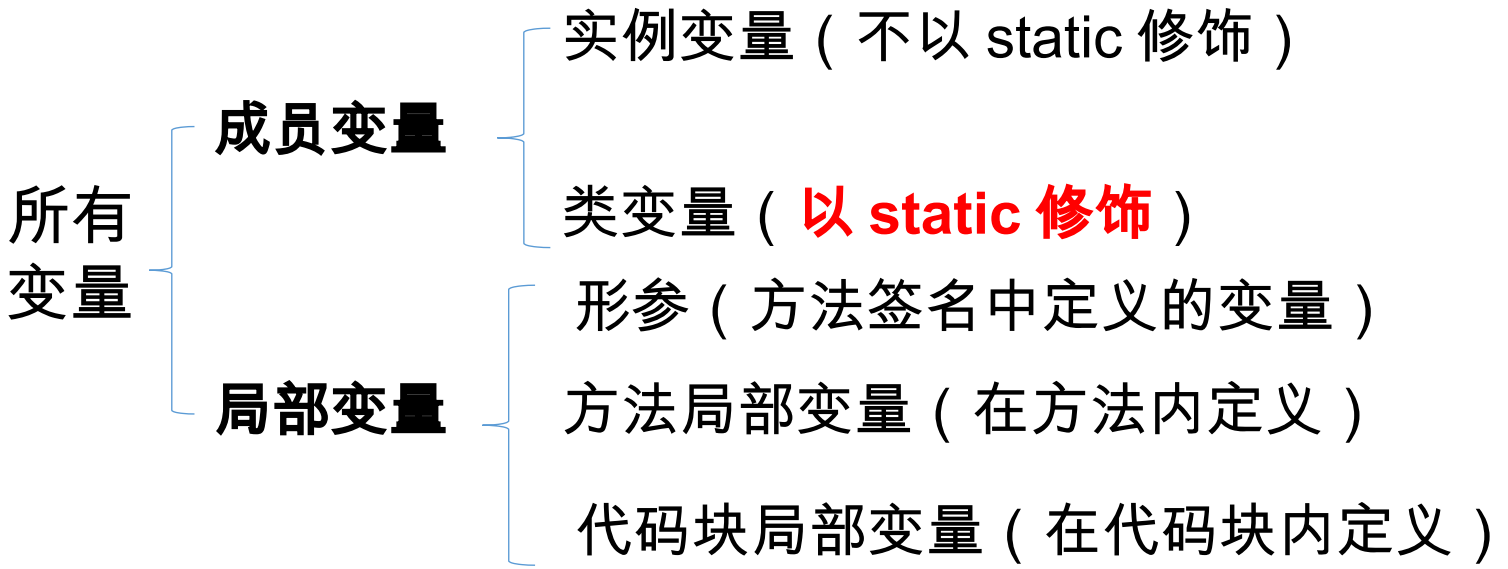
    类型：任何基本类型，如 int 、 boolean 或任何类。

- **举例：**

```
public class Person{
    private int age;          // 声明 private 变量  age
    public String name = “ Lila ”;   // 声明 public 变量  name
}
```

补：变量的分类：成员变量与局部变量

- 在方法体外，类体内声明的变量称为成员变量。
- 在方法体内部声明的变量称为局部变量。



成员变量与局部变量

- 注意：二者在初始化值方面的异同：

  同：都有生命周期

  异：局部变量除形参外，需显式初始化。

成员变量（属性）和局部变量的区别？

- **成员变量：**
  - 成员变量定义在类中，在整个类中都可以被访问。
  - 成员变量分为类成员变量和实例成员变量，实例变量存在于对象所在的堆内存中。
  - 成员变量**有默认初始化值**。
  - 成员变量的权限修饰符可以根据需要，选择任意一个

- **局部变量：**
  - 局部变量只定义在局部范围内，如：方法内，代码块内等。
  - 局部变量存在于栈内存中。
  - 作用的范围结束，变量空间会自动释放。
  - 局部变量没有默认初始化值，每次**必须显式初始化**。
  - 局部变量声明时不指定权限修饰符

属性 / 变量

成员变量与局部变量

4.4 类的成员之二：方 法

方法

语法格式：

修饰符 返回值类型 方法名 (参数列表){
    方法体语句；
}

说明： 修饰符： public, private, protected 等。

返回值类型： return 语句传递返回值。没有返回值： void 。

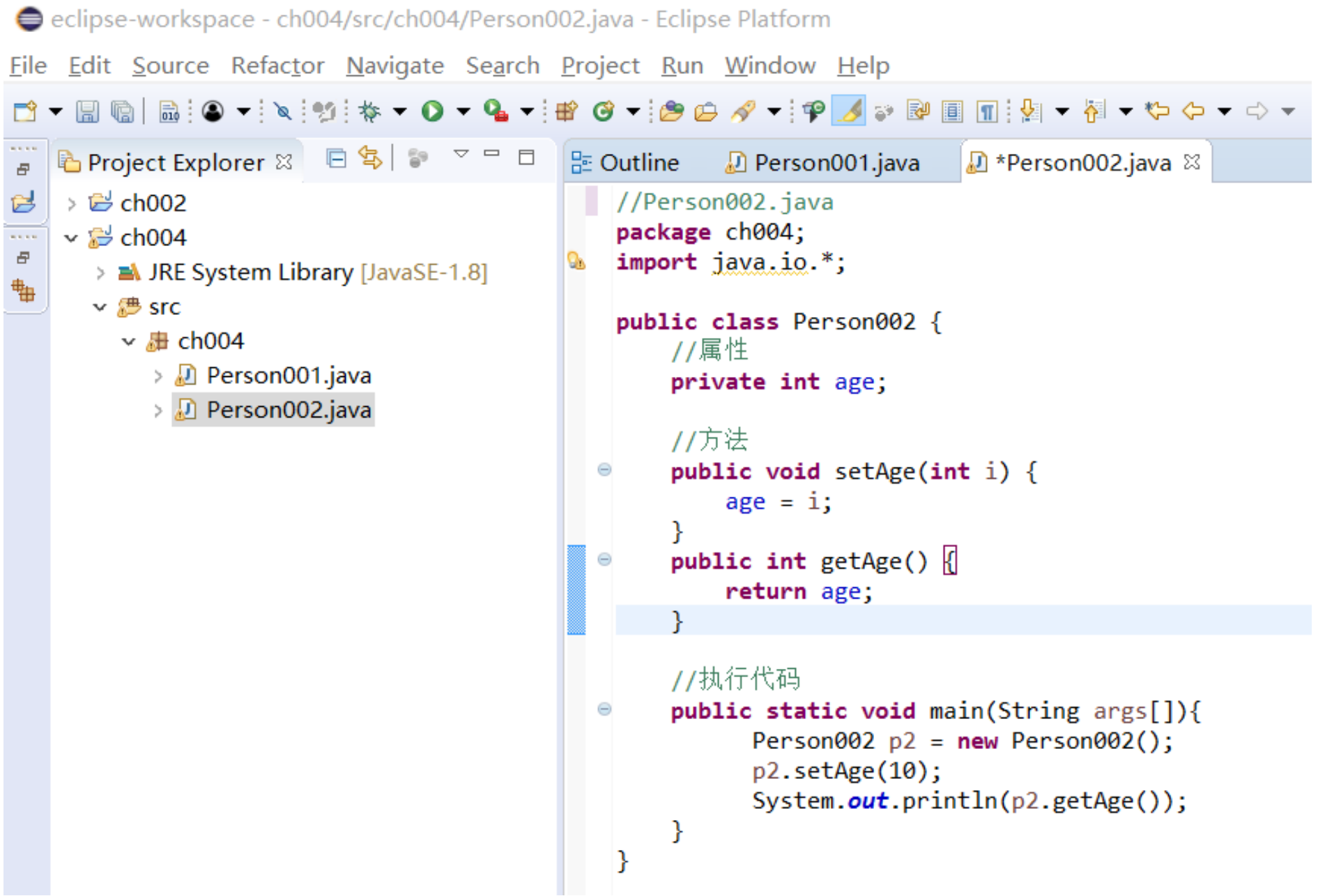
举例：

```
public class Person{
   private int age;
   public int getAge()  { return age; } // 声明方法 getAge
   public void setAge(int i) {        // 声明方法 setAge
          age = i;      // 将参数 i 的值赋给类的成员变量 age
   }
}
```

```java
//Person002.java
package ch004;
import java.io.*;

public class Person002 {
// 属性
private int age;

// 方法
public void setAge(int i) {
age = i;
}
public int getAge() {
return age;
}

// 执行代码
public static void main(String args[]){
	Person002 p2 = new Person002();
	p2.setAge(10);
	System.out.println(p2.getAge());
}
}
```

属性 & 方法

举例


eclipse-workspace - ch004/src/ch004/Person002.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
ch002
ch004
JRE System Library [JavaSE-1.8]
src
ch004
Person001.java
Person002.java
Outline
Person001.java
*Person002.java
//Person002.java
package ch004;
import java.io.*;
public class Person002 {
//属性
private int age;
//方法
public void setAge(int i) {
age = i;
}
public int getAge() {
return age;
}
//执行代码
public static void main(String args[]){
Person002 p2 = new Person002();
p2.setAge(10);
System.out.println(p2.getAge());
}
}

属性 & 方法

举例

调试

4.5 对象的创建和使用

创建和使用对象

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

对象的创建和使用

动物和动物园

构成的问题域

- 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）；

举例：

```
public class Animal {
  public int legs;
  public void  eat(){
    System.out.println(“Eating.”);
  }
  public void move(){
     System.out.println(“Move.”);
  }
}
```

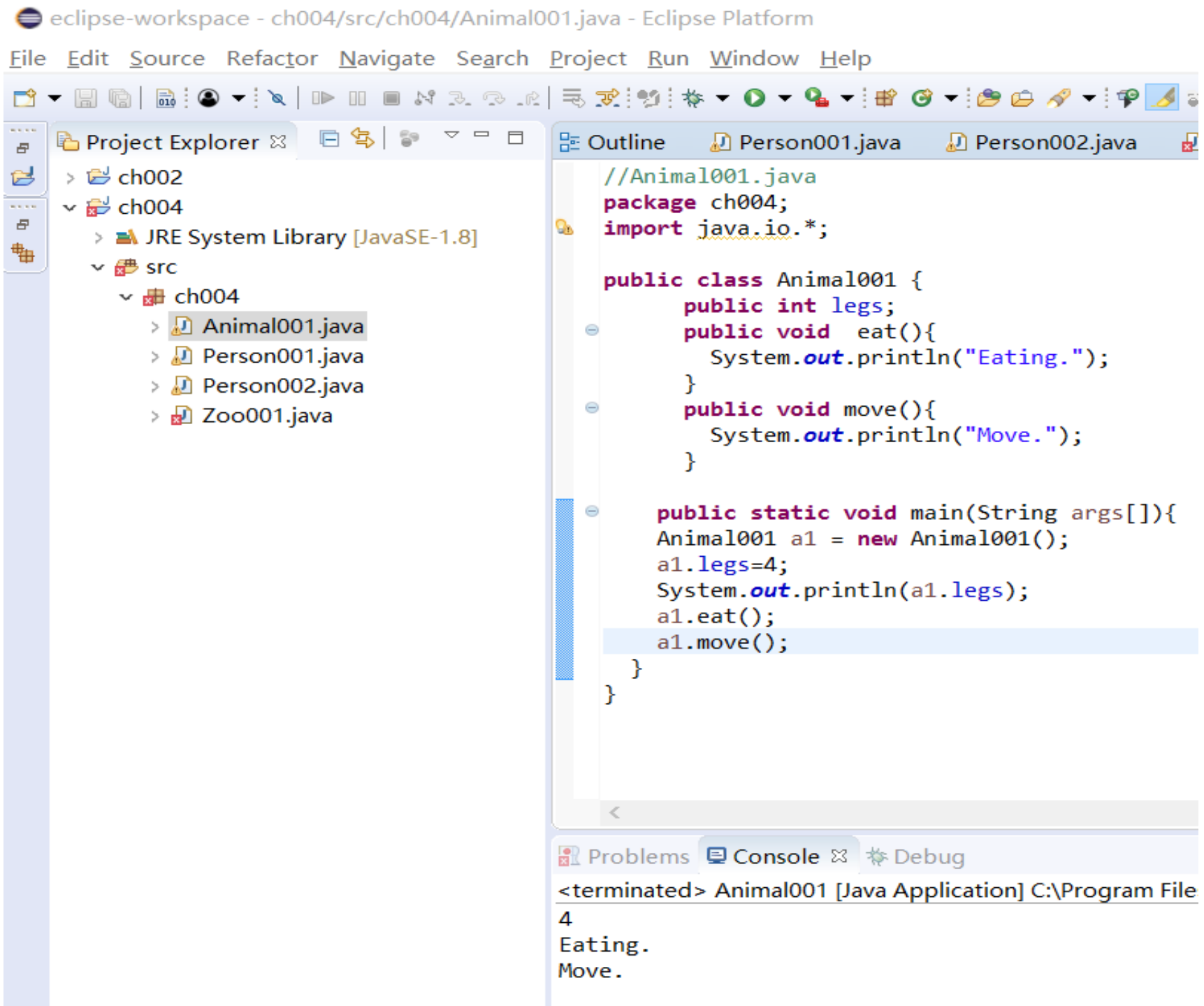
```
public class Zoo {
   public static void main(String args[]){
            Animal xb=new Animal();
            xb.legs=4;
            System.out.println(xb.legs);
            xb.eat();
            xb.move();
   }
}
```

```java
//Animal001.java
package ch004;
import java.io.*;

public class Animal001 {
  public int legs;
  public void  eat(){
System.out.println("Eating.");
  }
  public void move(){
System.out.println("Move.");
  }

public static void main(String args[]){
Animal001 a1 = new Animal001();
a1.legs=4;
System.out.println(a1.legs);
a1.eat();
a1.move();
  }
}
```

动物类型

和

实例

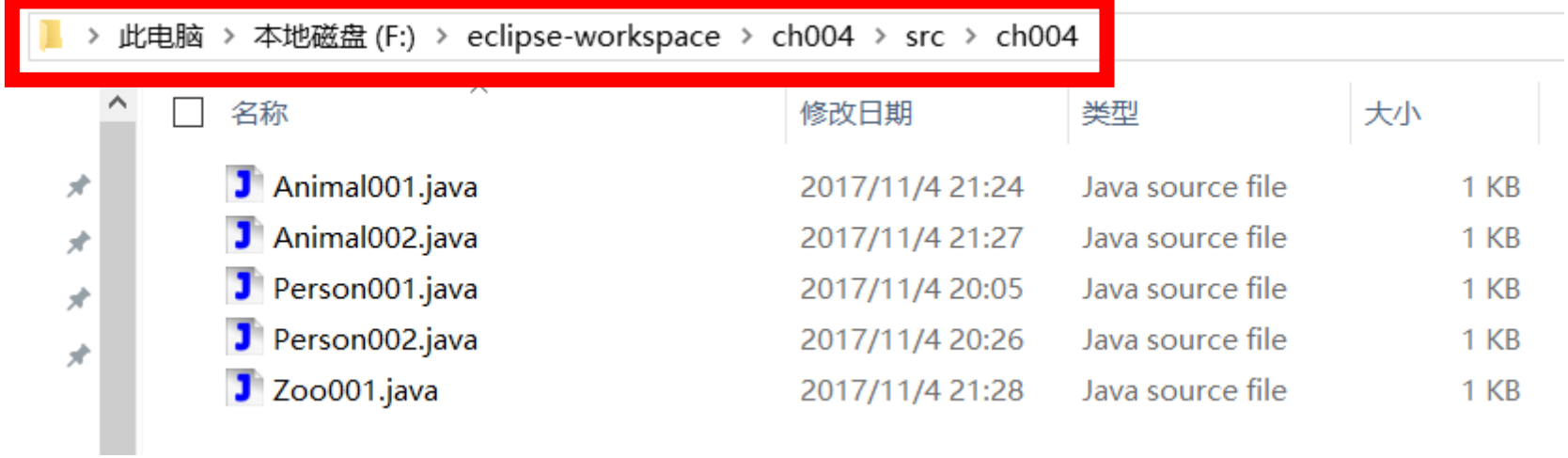

eclipse-workspace - ch004/src/ch004/Animal001.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
ch002
ch004
JRE System Library [JavaSE-1.8]
src
ch004
Animal001.java
Person001.java
Person002.java
Zoo001.java
Outline
Person001.java
Person002.java
//Animal001.java
package ch004;
import java.io.*;
public class Animal001 {
public int legs;
public void eat(){
System.out.println("Eating.");
}
public void move(){
System.out.println("Move.");
}
public static void main(String args[]){
Animal001 a1 = new Animal001();
a1.legs=4;
System.out.println(a1.legs);
a1.eat();
a1.move();
}
}
Problems
Console
Debug
<terminated> Animal001 [Java Application] C:\Program File
4
Eating.
Move.

动物类型

和

实例

调试



动物类型

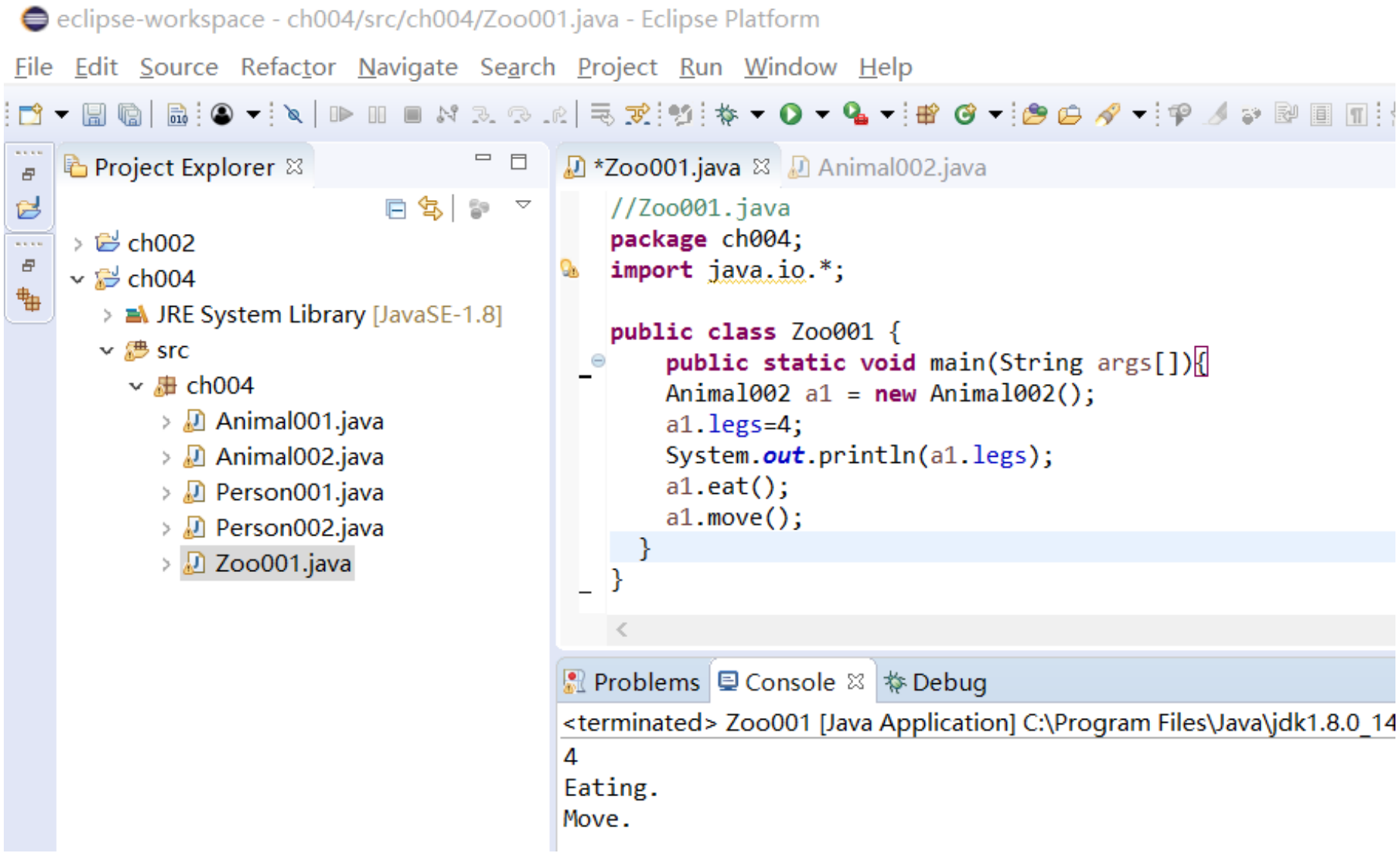
和

实例

代码

```java
//Animal001.java
package ch004;
import java.io.*;

public class Animal002 {
     public int legs;
  public void  eat(){
System.out.println("Eating.");
  }
  public void move(){
System.out.println("Move.");
  }
}
```

```java
//Zoo001.java
package ch004;
import java.io.*;

public class Zoo001 {
public static void main(String args[]){
Animal002 a1 = new Animal002();
a1.legs=4;
System.out.println(a1.legs);
a1.eat();
a1.move();
  }
}
```


eclipse-workspace - ch004/src/ch004/Zoo001.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
ch002
ch004
JRE System Library [JavaSE-1.8]
src
ch004
Animal001.java
Animal002.java
Person001.java
Person002.java
Zoo001.java
*Zoo001.java
Animal002.java
//Zoo001.java
package ch004;
import java.io.*;
public class Zoo001 {
public static void main(String args[]){
Animal002 a1 = new Animal002();
a1.legs=4;
System.out.println(a1.legs);
a1.eat();
a1.move();
}
}
Problems
Console
Debug
<terminated> Zoo001 [Java Application] C:\Program Files\Java\jdk1.8.0_14
4
Eating.
Move.

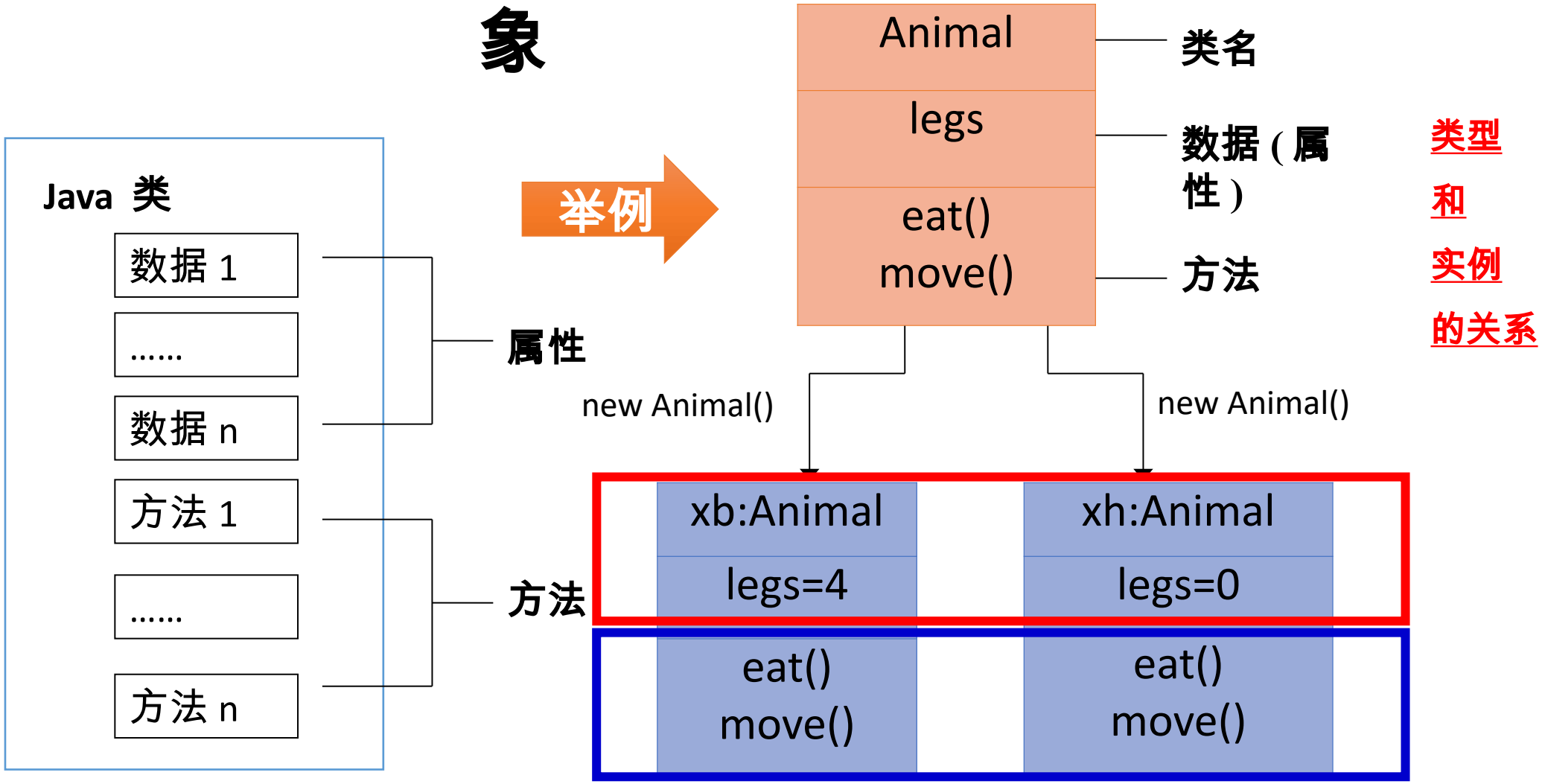
动物类型

和

实例

代码

Java 中类与对象



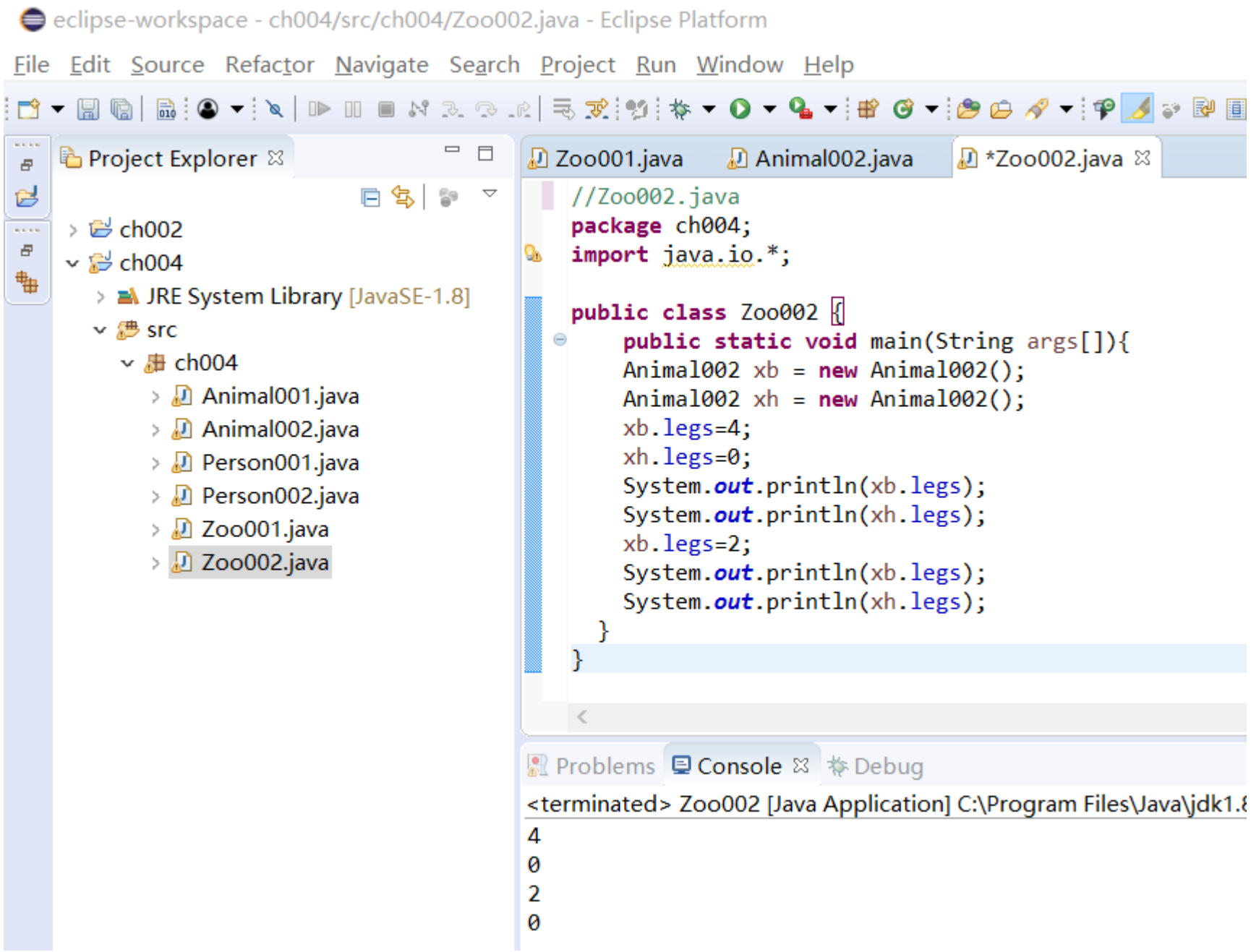
类型和实例的关系

对象的创建和使用

- 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。

举例：

```
public class Zoo {
    public static void main(String args[]){
        Animal xb=new Animal();
        Animal xh=new Animal();
        xb.legs=4;
        xh.legs=0;
        System.out.println(xb.legs);   //4
        System.out.println(xh.legs);   //0
        xb.legs=2;
        System.out.println(xb.legs);   //2
        System.out.println(xh.legs);   //0
    } }
```



类型

和

实例

的关系

举例调试

推荐练习

编写教师类和学生类，并通过测试类创建对象进行测试

学生类
属性： 姓名 年龄 参加的课程 兴趣
方法： 显示学生的个人信息

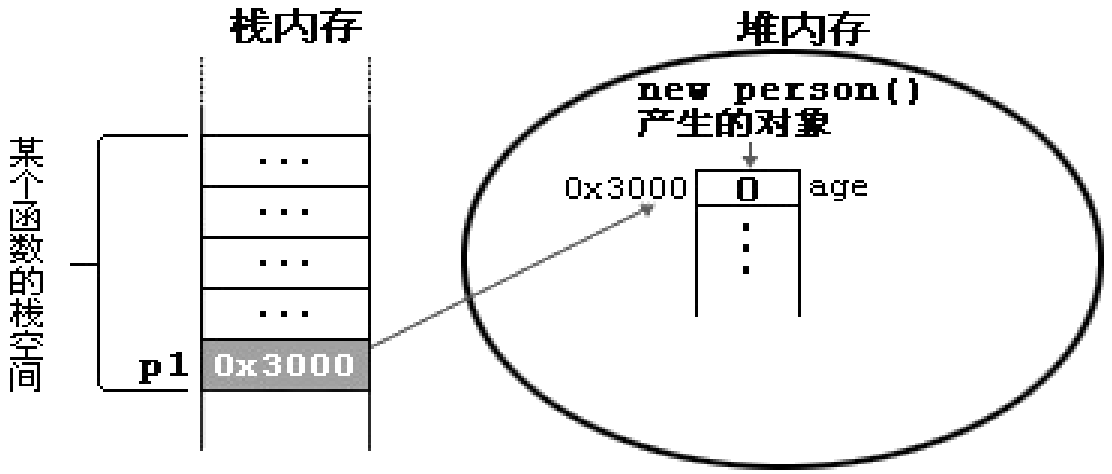
教师类
属性： 姓名 专业 教授的课程 教龄
方法： 显示教师的个人信息

课堂作业

对象的产生

```
class Person{
   int age;
   void shout(){
            System.out.println("oh,my god! I am " + age);
   }
}
```

Person p1 = new Person(); 执行完后的内存状态



类型

和

实例

存储

对象的产生

当一个对象被创建时，会对其中各种类型的成员变量自动进行初始化赋值。除了基本数据类型之外的变量类型都是引用类型，如上面的 Person 及前面讲过的数组。

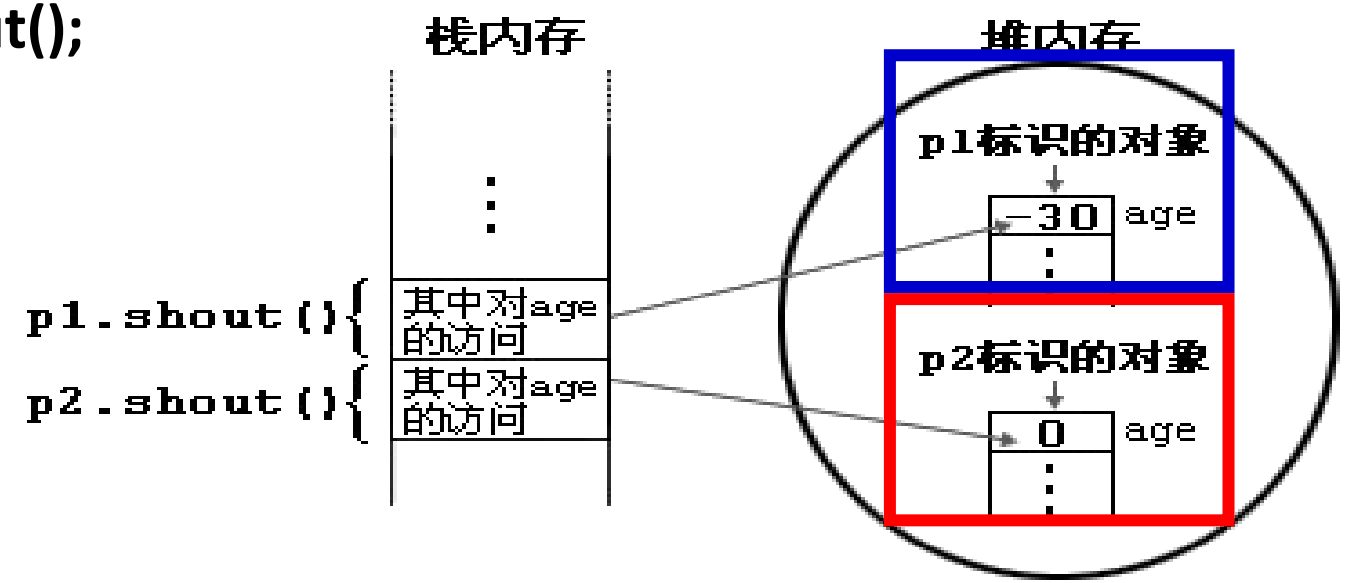
成员变量类型	初始值
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0D
char	‘ \u0000 ’ (表示为空)
boolean	false
引用类型	null

成员变量默认的初始化赋值

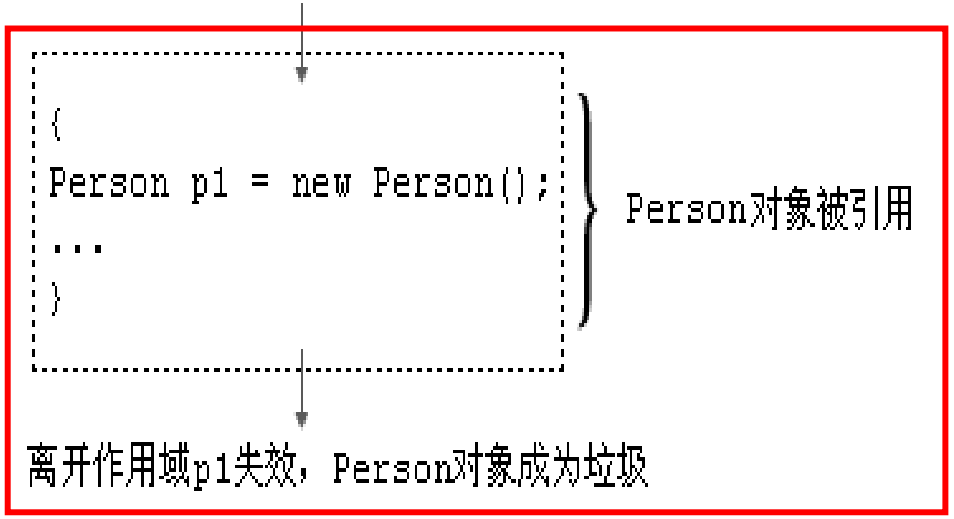
对象的使用

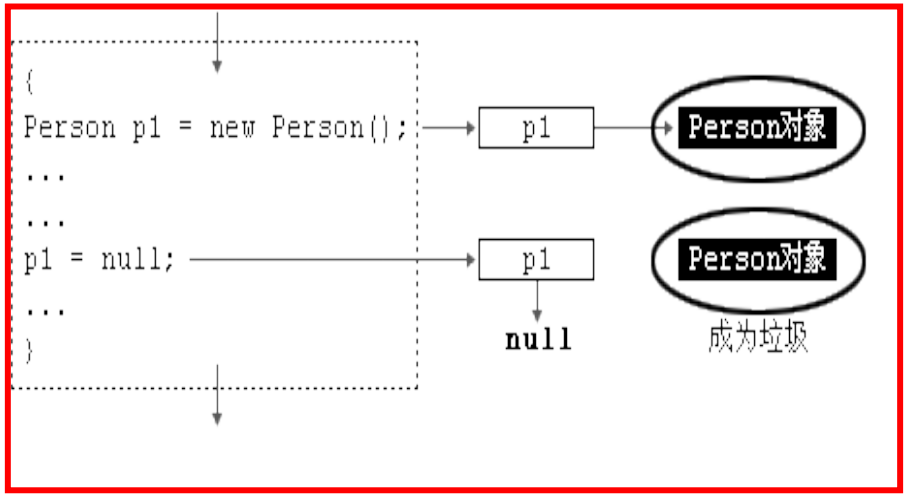
```
class TestPerson{
    public static void main(String[] args) {   // 程序运行的内存布局如下图
        Person p1 = new Person();
        Person p2 =new Person();
        p1.age = -30;
        p1.shout();
        p2.shout();
    }
}
```

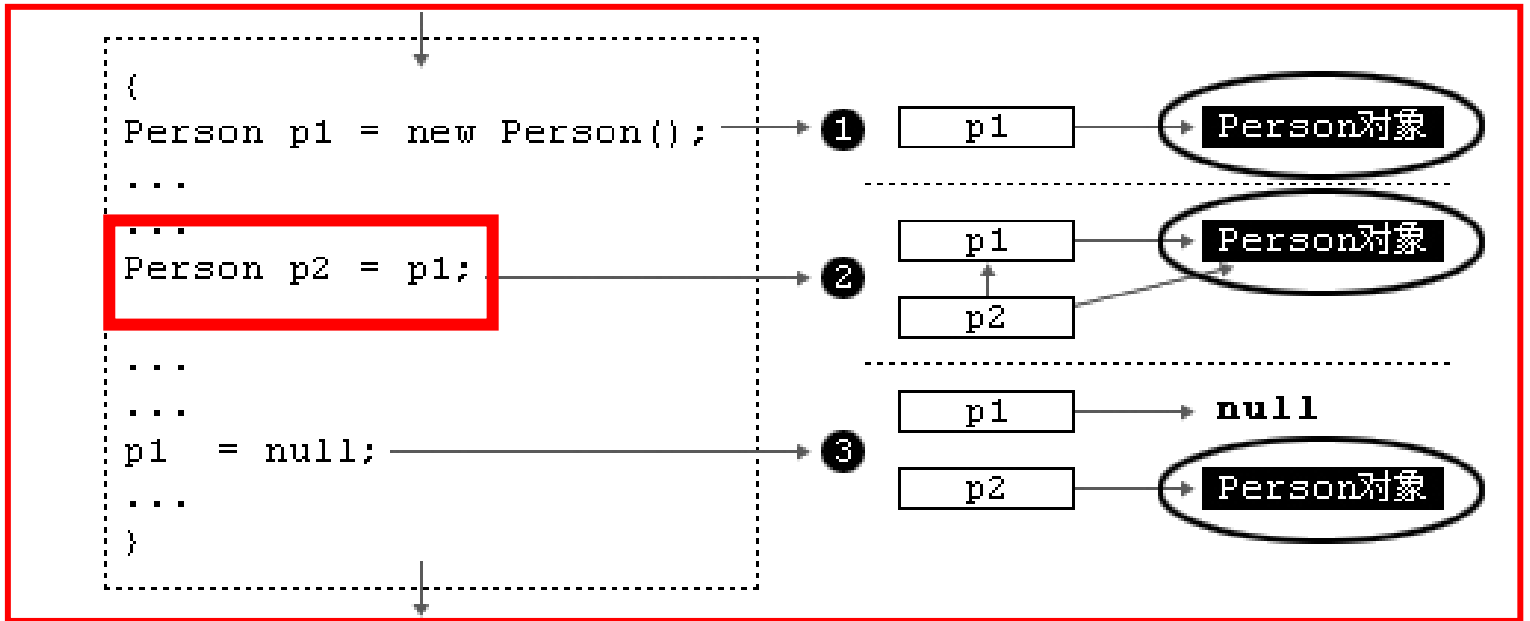
成员变量的赋值



对象的生命周期







对象在内存中的存储情况

对象名、对象内容

代码的内存图

```
class Car{
        String color = "red";
        int num = 4;
        void show(){
                System.out.println("color="+color+"..num="+num);
         }
 }
class TestCar {
        public static void main(String[] args) {
                Car c1 = new Car();   // 建立对象 c1
                Car c2 = new Car();   // 建立对象 c2
                c1.color = "blue";   // 对对象的属性进行修改
                c1.show();   // 使用对象的方法
                c2.show();
        } }
```

对象在内存中的

存储情况

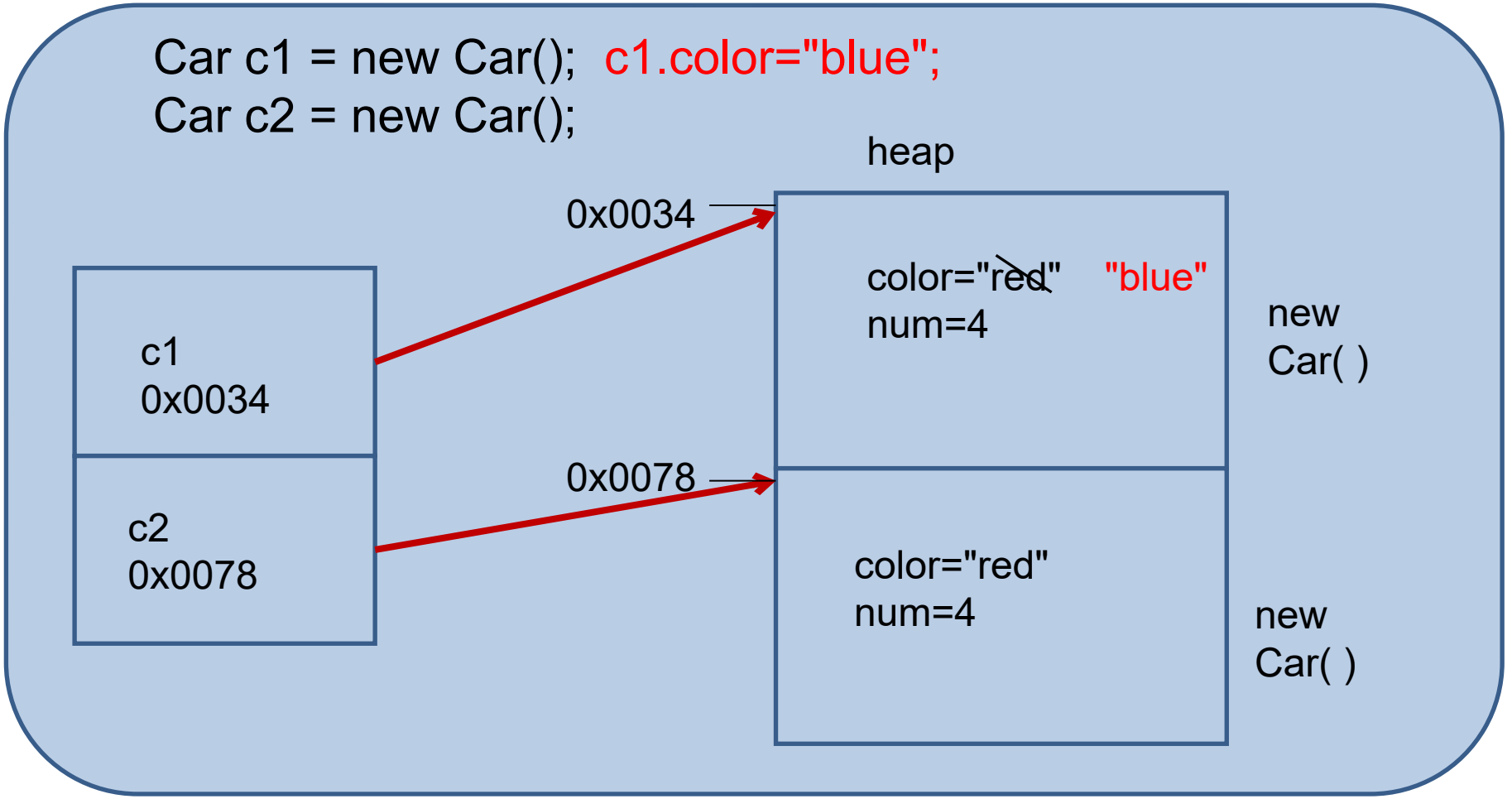
对象名、对象内容

对象在内存中的

存储情况

对象名、对象内容

对象内存结构



匿名对象

- 我们也可以不定义对象的句柄，而直接调用这个对象的方法。这样的对象叫做匿名对象。
  - 如： **new Person().shout();**

匿名对象

- 使用情况
  - 如果对一个对象只需要进行一次方法调用，那么就可以使用匿名对象。
  - 我们经常将匿名对象作为实参传递给一个方法调用。

推荐练习

1. 创建一个 Person 类，其定义如下：

Person
name:String age:int sex:int
+study():void +showAge():void +addAge(int i):int

要求：(1) 创建 Person 类的对象，设置该对象的 name、 age 和 sex 属性，调用 study 方法，输出字符串“ studying ”，调用 showAge() 方法显示 age 值，调用 addAge() 方法给对象的 age 属性值增加 2 岁。

(2) 创建第二个对象，执行上述操作，体会同一个类的不同对象之间的关系。

2. 利用面向对象的编程方法，设计类 Circle 计算圆的面积。

4.6 再谈方法 (method)

- **什么是方法（函数）？**
  - 方法是类或对象行为特征的抽象，也称为函数。
  - Java 里的方法不能独立存在，所有的方法必须定义在类里。

方法

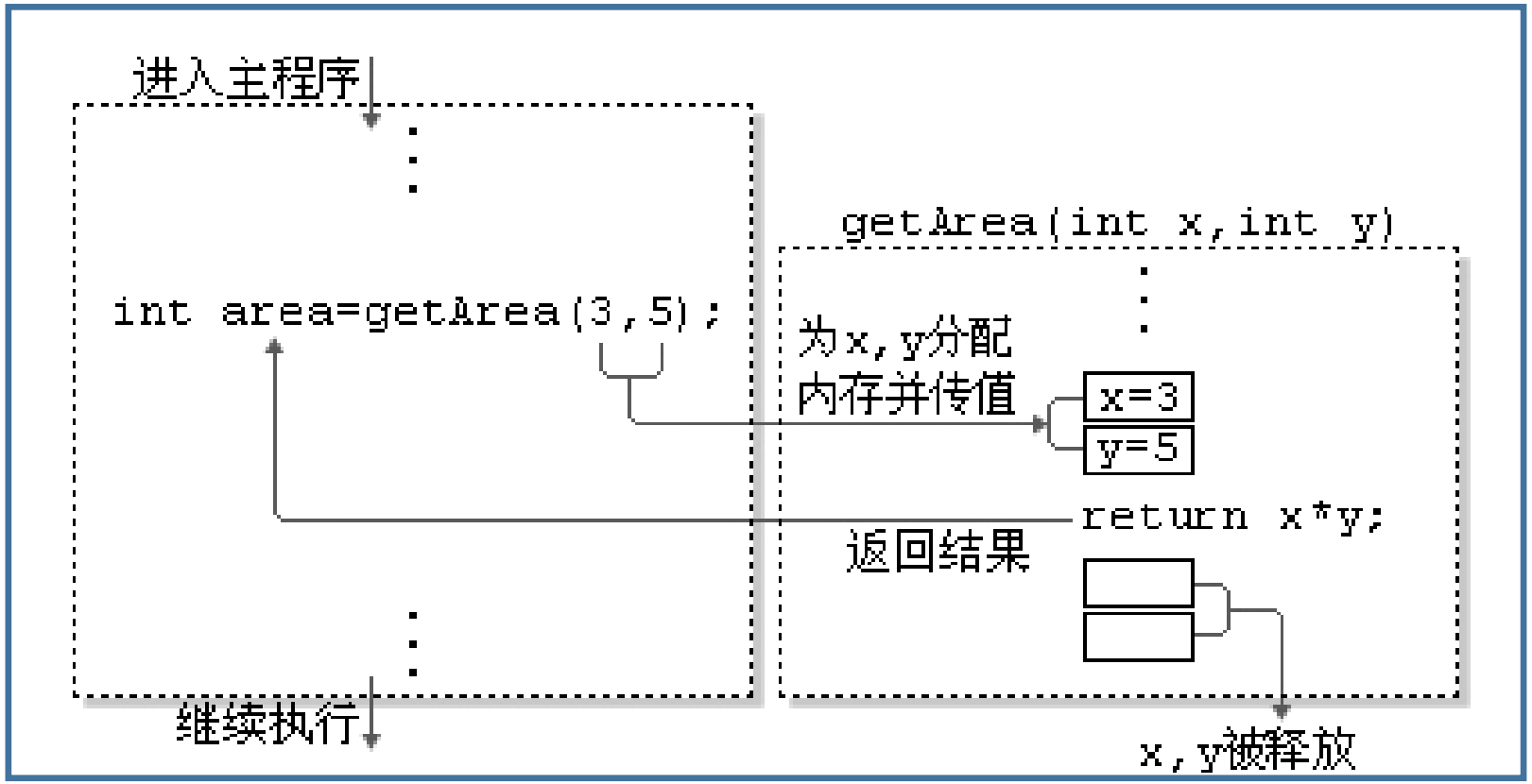
修饰符 返回值类型 方法名（参数类型 形参 1 ，参数类型 形参 2 ，. . . . ）{
  程序代码
  return 返回值 ;
}

其中：
形式参数：在方法被调用时用于接收外部传入的数据的变量。
参数类型：就是该形式参数的数据类型。
返回值：方法在执行完毕后返还给调用它的程序的数据。
返回值类型：方法要返回的结果的数据类型。
实参：调用方法时实际传给函数形式参数的数据。

◆ 如何理解方法返回值类型为 void 的情况 ？

方法的调用

- 方法只有被调用才会被执行
- 方法调用的过程分析



方法

值传递

方法的调用

●注　意：

- 没有具体返回值的情况，返回值类型用关键字 void 表示，那么该函数中的 return 语句如果在最后一行可以省略不写。

- **定义方法时，方法的结果应该返回给调用者，交由调用者处理。**

- **方法中只能调用方法，不可以在方法内部定义方法。**

面向对象思想“落地”法则（一）

- 1. 关注于类的设计，即设计类的成员：属性、方法
- 2. 类的实例化，即创建类的对象（比如： Person p = new Person()）
- 3. 通过“**对象.属性**”、“**对象.方法**”执行

△方法的重载(overload)

重载的概念

在同一个类中，允许存在一个以上的同名方法，只要它们的参数个数或者参数类型不同即可。

重载的特点：

与返回值类型无关，只看参数列表，且参数列表必须不同。（参数个数或参数类型）。调用时，根据方法参数列表的不同来区别。

重载示例：

```
// 返回两个整数的和
  int add(int x,int y){return x+y;}
// 返回三个整数的和
  int add(int x,int y,int z){return x+y+z;}
// 返回两个小数的和
  double add(double x,double y){return x+y;}
```

一个方法名字

多个解决方案（不同参数方案）

根据参数方案调用对应方案

函数的重载

```java
//PrintStream.java
package ch004;

public class PrintStream {
  public static void print(int i) {
        System.out.println(i);
  }
  public static void print(float i) {
        System.out.println(i);
  }
  public static void print(String i) {
        System.out.println(i);
  }
  public static void main(String[] args){
        print(1);
        print(1.2f);
        print("hello!");
 }
}
```

重载举例

参考练习 3

1. 判 断 ：　　与返回值类型无关，只看参数列表

与 void show(int a, char b, double c){} 构成重载的有：

a)　void show(int x, char y, double z){}　//no

b)　int show(int a, double c, char b){}　//yes

c)　void show(int a, double c, char b){}　//yes

d)　boolean show(int c, char b){}　//yes

e)　void show(double c){}　//yes

f)　double show(int x, char y, double z){}　//no

g)　void shows(){double c}　//no

判断是否存在重载

方法名一样

个数不一样

相同次序上的类型不一样

参考练习 3（续）

课堂练习

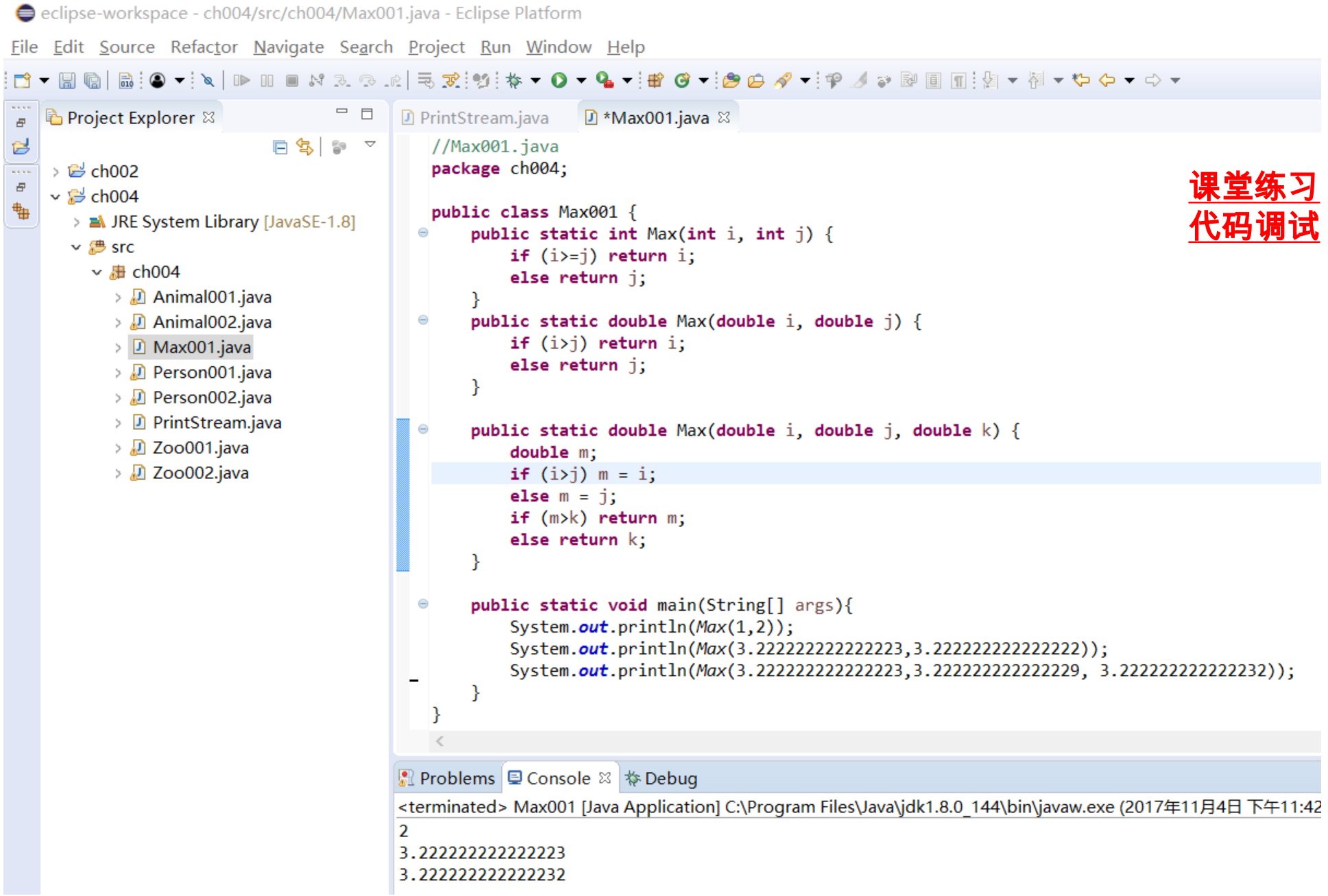
2. 编写程序，定义三个重载方法并调用。方法名为 mOL 。

- 三个方法分别接收一个 int 参数、两个 int 参数、一个字符串参数。分别执行平方运算并输出结果，相乘并输出结果，输出字符串信息。
- 在主类的 main () 方法中分别用参数区别调用三个方法。

4. 定义三个重载方法 max() ，第一个方法求两个 int 值中的最大值，第二个方法求两个 double 值中的最大值，第三个方法求三个 double 值中的最大值，并分别调用三个方法。

课堂练习
参考答案

```java
//Max001.java
package ch004;

public class Max001 {
 public static int Max(int i, int j) {
    if (i>=j) return i;
    else return j;
 }
 public static double Max(double i, double j) {
    if (i>j) return i;
    else return j;
 }

 public static double Max(double i, double j, double k) {
    double m;
    if (i>j) m = i;
    else m = j;
    if (m>k) return m;
    else return k;
 }

 public static void main(String[] args){
  System.out.println(Max(1,2));
  System.out.println(Max(4.222222222222223,4.222222222222222));
  System.out.println(Max(4.222222222222223,4.222222222222229, 4.222222222222232));
 }
}
```


eclipse-workspace - ch004/src/ch004/Max001.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
ch002
ch004
JRE System Library [JavaSE-1.8]
src
ch004
Animal001.java
Animal002.java
Max001.java
Person001.java
Person002.java
PrintStream.java
Zoo001.java
Zoo002.java
PrintStream.java
*Max001.java
//Max001.java
package ch004;
public class Max001 {
public static int Max(int i, int j) {
if (i>=j) return i;
else return j;
}
public static double Max(double i, double j) {
if (i>j) return i;
else return j;
}
public static double Max(double i, double j, double k) {
double m;
if (i>j) m = i;
else m = j;
if (m>k) return m;
else return k;
}
public static void main(String[] args){
System.out.println(Max(1,2));
System.out.println(Max(3.222222222222223,3.222222222222222));
System.out.println(Max(3.222222222222223,3.222222222222229, 3.222222222222232));
}
}
课堂练习
代码调试
Problems
Console
Debug
<terminated> Max001 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年11月4日 下午11:42
2
3.222222222222223
3.222222222222232

☆ 方法的参数传递

- **方法，必须有其所在类或对象调用才有意义。若方法含有参数：**
  - **形参**：方法声明时的参数
  - **实参**：方法调用时实际传给形参的参数值

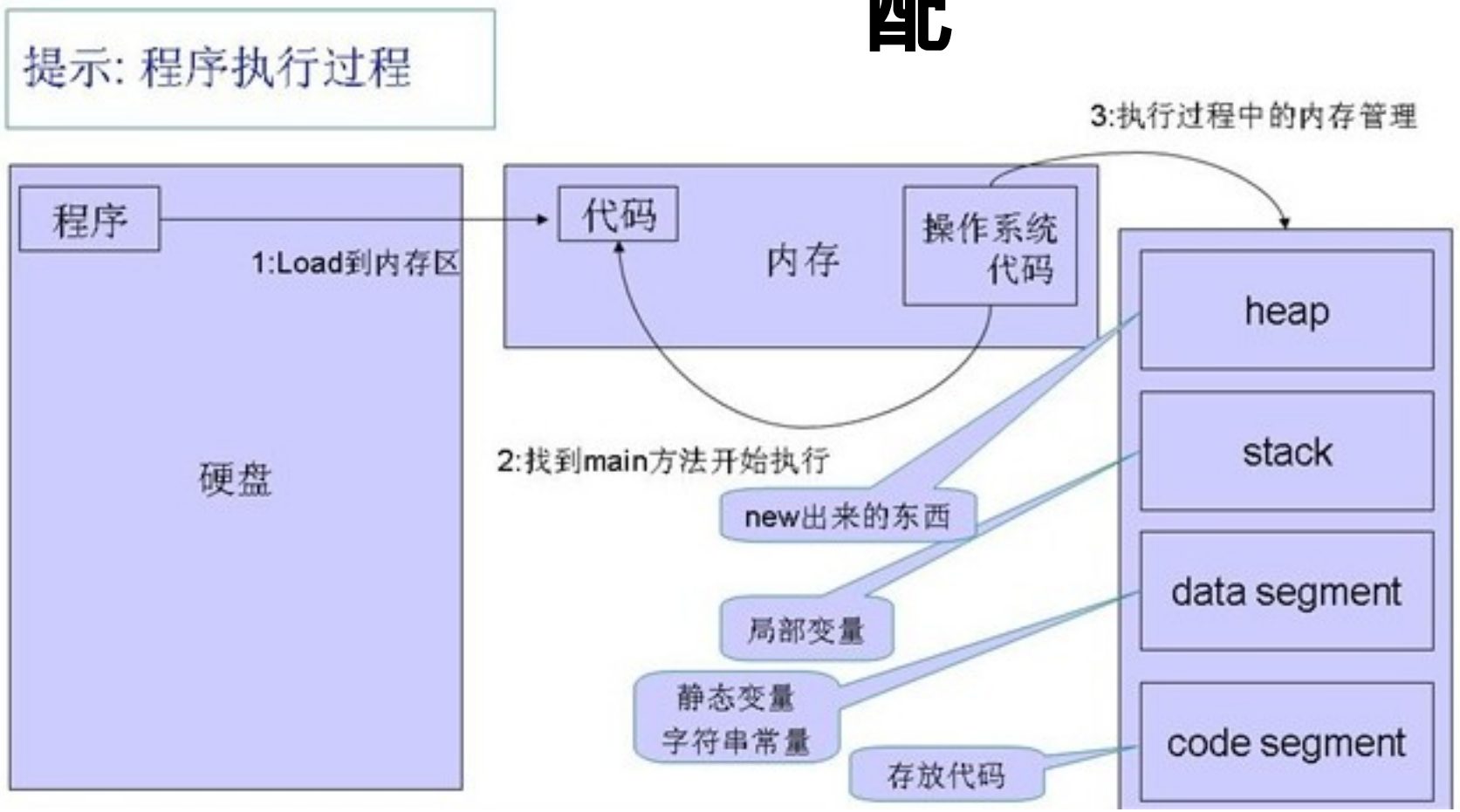
值传递

- Java 的实参值如何传入方法呢？

Java 里方法的参数传递方式只有一种：值传递。 即将实际参数值的副本（复制品）传入方法内，而参数本身不受影响。

Java 内存分配

程序执行过程中的信息存储方式



heap: 动态申请内存用，可以理解为 new 出来的东西

stack: 局部变量

data segment: 静态变量 + 字符串常量

code segment: 存放代码

Java 内存分配

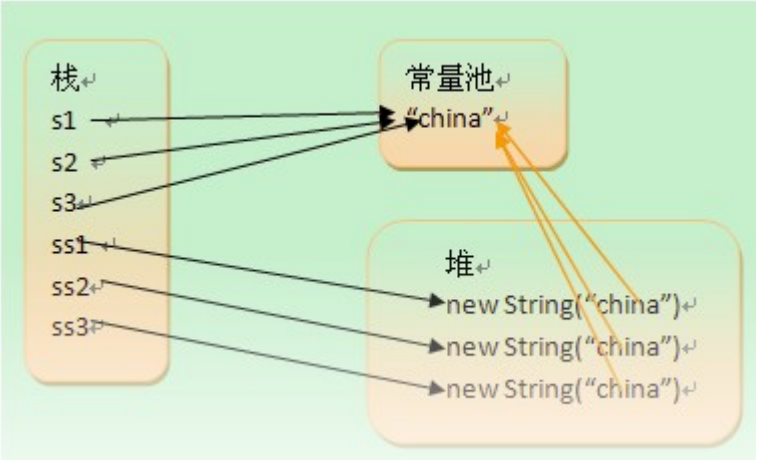
程序执行过程中的信息存储方式

主要包括以下几个区域：

1. 寄存器：我们在程序中无法控制
2. **栈：存放基本类型的数据和对象的引用，但对象本身不存放在栈中，而是存放在堆中**
3. **堆：存放用 new 产生的数据**
4. 静态域：存放在对象中用 static 定义的静态成员 (data segment)
5. 常量池：存放常量 (data segment)
6. 非 RAM(随机存取存储器) 存储：硬盘等永久存储空间

Java代码

```
1. String s1 = "china";
2. String s2 = "china";
3. String s3 = "china";
4. String ss1 = new String("china");
5. String ss2 = new String("china");
6. String ss3 = new String("china");
```



对于栈和常量池中的对象可以共享，对于堆中的对象不可以共享。栈中的数据大小和生命周期是可以确定的，当没有引用指向数据时，这个数据就会消失。**堆中的对象的由垃圾回收器负责回收，因此大小和生命周期不需要确定，具有很大的灵活性。**

程序执行过程中的信息存储方式

	栈内存	堆内存
存储变量种类	局部变量（基本类型变量和对象的引用变量） 对象的引用变量--用于指向堆内存new出来的内存空间	new创建的对象和数组
回收	超出作用域，java自动释放内存空间	由java虚拟机的自动垃圾回收器管理
优势	存取速度比堆要快，仅次于寄存器，栈数据可以共享	动态分配内存，生存期不必事先告诉编译器 运行时动态分配
缺点	存在栈中的数据大小和生存期必须是确定的，缺乏灵活性 栈中主要存放基本类型的变量和对象句柄	存取速度较慢

Java代码

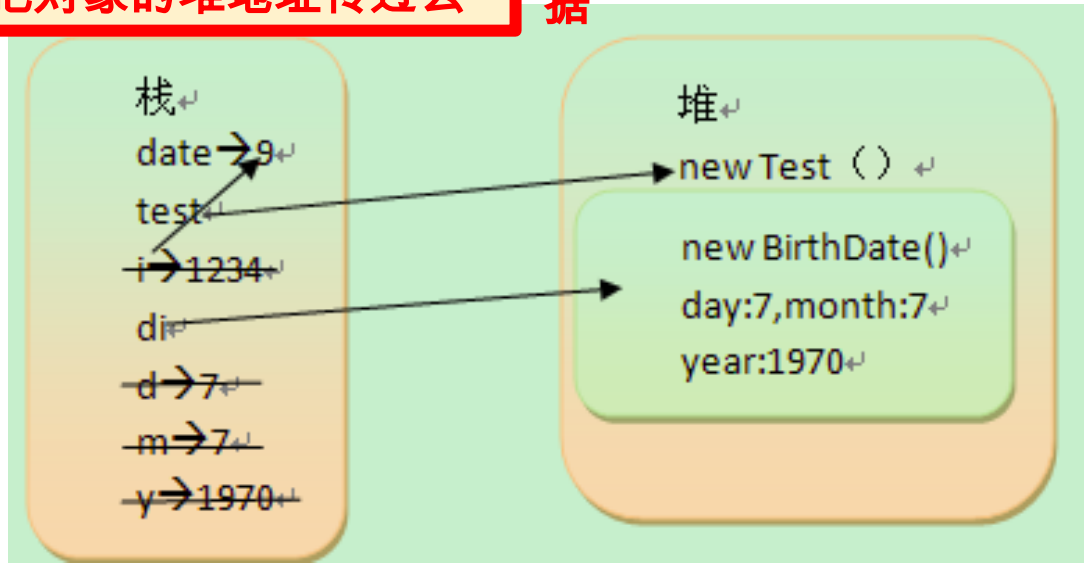
```java
class BirthDate {
    private int day;
    private int month;
    private int year;
    public BirthDate(int d, int m, int y) {
        day = d;
        month = m;
        year = y;
    }
    省略get,set方法.
}
```

基本类型数据作为参数是值拷贝

对象作为参数是传引用，可以认为是把对象的堆地址传过去

```java
public class Test{
    public static void main(String args[]){
int date = 9;
        Test test = new Test();
            test.change(date);
        BirthDate d1= new BirthDate(7,7,1970);
    }

    public void change1(int i){
        i = 1234;
    }
}
```

堆：存放用 new 产生的数据



栈：存放基本类型的数据和对象的引用，但对象本身不存放在栈中，而是存放在堆中

	栈内存	堆内存
存储变量种类	局部变量（基本类型变量和对象的引用变量） 对象的引用变量--用于指向堆内存new出来的内存空间	new创建的对象和数组
回收	超出作用域，java自动释放内存空间	由java虚拟机的自动垃圾回收器管理
优势	存取速度比堆要快，仅次于寄存器，栈数据可以共享	动态分配内存，生存期不必事先告诉编译器 运行时动态分配
缺点	存在栈中的数据大小和生存期必须是确定的，缺乏灵活性 栈中主要存放基本类型的变量和对象句柄	存取速度较慢

方法的参数传递

参数传递（值传递）

基本类型数据作为参数是值拷贝

对象作为参数是传引用，可以认为是把对象的堆地址传过去

```
//TestTransfer.java
package ch004;

public class TestTransfer {
 public static void swap(int a , int b){
  int tmp = a;
  a = b;
  b = tmp;
  System.out.println("swap 方法里，a 的值是" + a + "；b 的值是" + b);
 }

 public static void main(String[] args) {
  int a = 6;
  int b = 9;
  swap(a , b);
  System.out.println("交换结束后，变量 a 的值是" + a + "；变量 b 的值是" + b);
 }
}
```

请输出结果

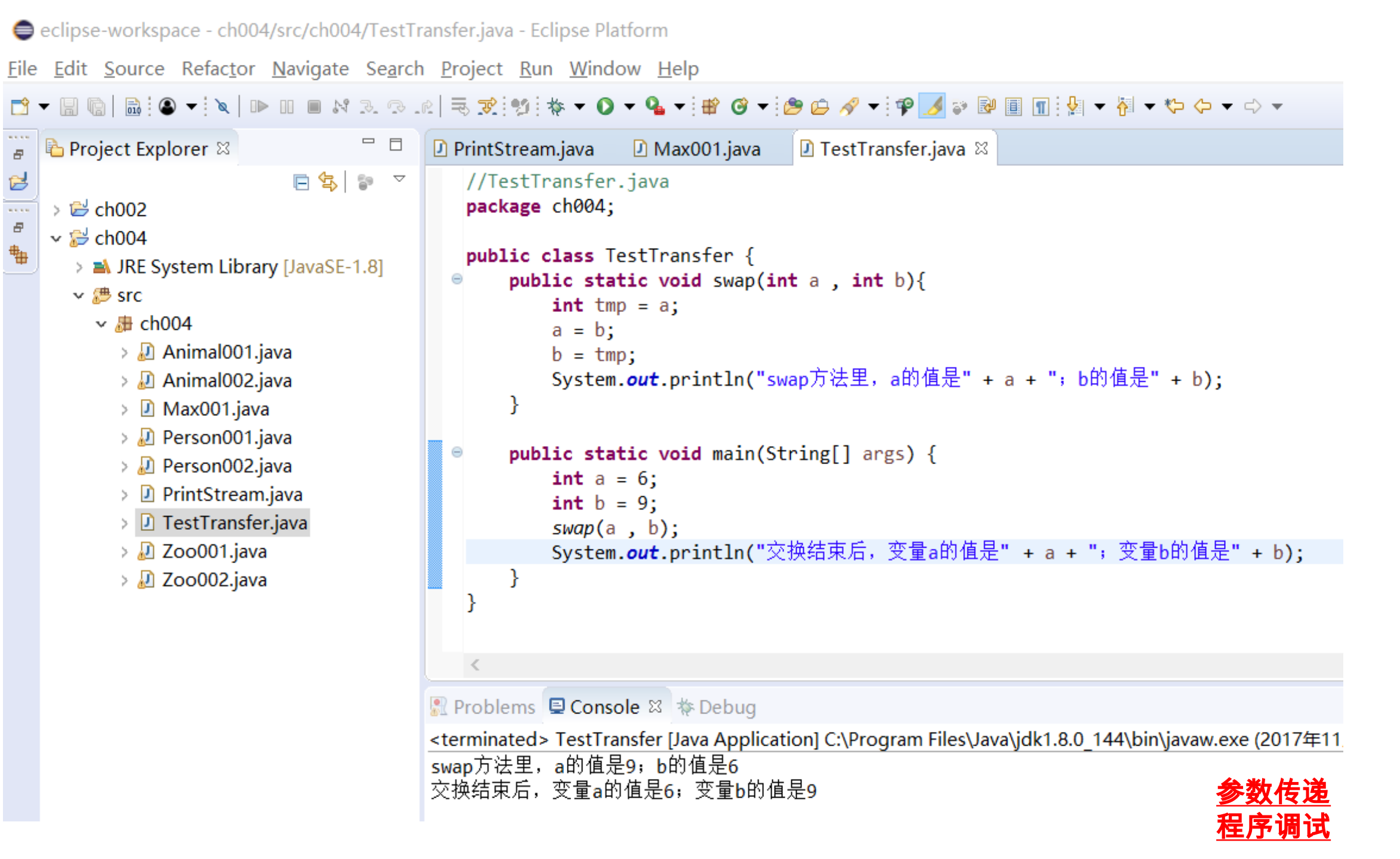
栈（2）
a->6
b->9
a'->6
b'->9
tmp->6

栈（3）
a->6
b->9
a'->9
b'->6
tmp->6

栈（1）
a->6
b->9

栈（5）

栈（4）
a->6
b->9

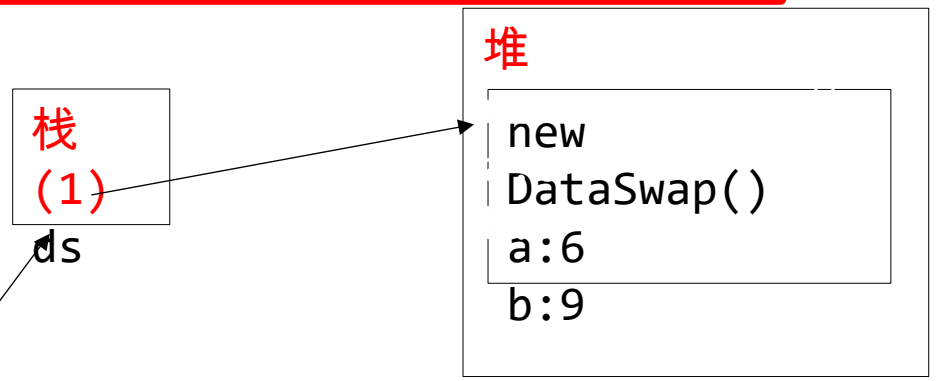

eclipse-workspace - ch004/src/ch004/TestTransfer.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
ch002
ch004
JRE System Library [JavaSE-1.8]
src
ch004
Animal001.java
Animal002.java
Max001.java
Person001.java
Person002.java
PrintStream.java
TestTransfer.java
Zoo001.java
Zoo002.java
PrintStream.java
Max001.java
TestTransfer.java
//TestTransfer.java
package ch004;
public class TestTransfer {
public static void swap(int a , int b){
int tmp = a;
a = b;
b = tmp;
System.out.println("swap方法里，a的值是" + a + "；b的值是" + b);
}
public static void main(String[] args) {
int a = 6;
int b = 9;
swap(a , b);
System.out.println("交换结束后，变量a的值是" + a + "；变量b的值是" + b);
}
}
Problems
Console
Debug
<terminated> TestTransfer [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年11
swap方法里，a的值是9；b的值是6
交换结束后，变量a的值是6；变量b的值是9
参数传递
程序调试

方法的参数传递

```
//DataSwap.java
package ch004;
class DataSwap{
 public int a;
 public int b;
}
```

基本类型数据作为参数是值拷贝

对象作为参数是传引用，可以认为是把对象的堆地址传过去




```
//TestTransfer1.java
package ch004;
public class TestTransfer1 {
 public static void swap(DataSwap ds){
  int tmp = ds.a;
  ds.a = ds.b;
  ds.b = tmp;
  System.out.println("swap 方法里，a Field 的值是"+ ds.a + " ；b Field 的值是" + ds.b);
 }

 public static void main(String[] args) {
  DataSwap ds = new DataSwap();
  ds.a = 6;
  ds.b = 9;
  swap(ds);
  System.out.println(" 交换结束后，a Field 的值是"+ ds.a + " ；b Field 的值是" + ds.b);
 }
}
```

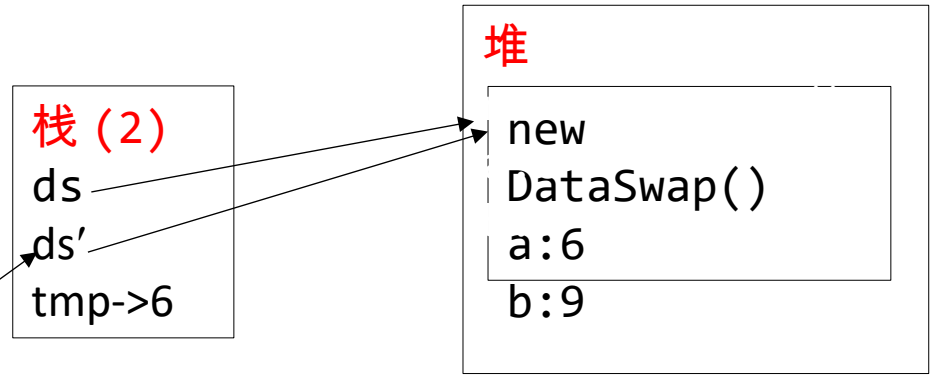
请输出结果

方法的参数传递

```java
//DataSwap.java
package ch004;
class DataSwap{
 public int a;
 public int b;
}
```

基本类型数据作为参数是值拷贝

对象作为参数是传引用，可以认为是把对象的堆地址传过去




```java
//TestTransfer1.java
package ch004;
public class TestTransfer1 {
 public static void swap(DataSwap ds){
  int tmp = ds.a;
  ds.a = ds.b;
  ds.b = tmp;
  System.out.println("swap 方法里，a Field 的值是"+ ds.a + " ；b Field 的值是" + ds.b);
 }

 public static void main(String[] args) {
  DataSwap ds = new DataSwap();
  ds.a = 6;
  ds.b = 9;
  swap(ds);
  System.out.println(" 交换结束后，a Field 的值是"+ ds.a + " ；b Field 的值是" + ds.b);
 }
}
```

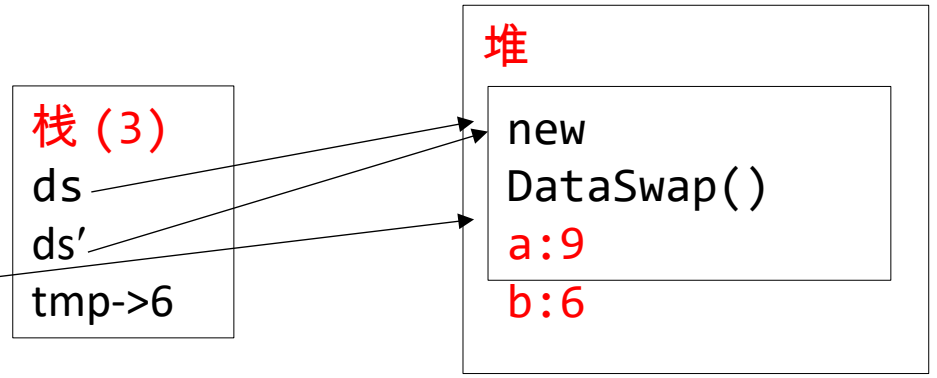
请输出结果

方法的参数传递

```java
//DataSwap.java
package ch004;
class DataSwap{
 public int a;
 public int b;
}
```

基本类型数据作为参数是值拷贝

对象作为参数是传引用，可以认为是把对象的堆地址传过去




```java
//TestTransfer1.java
package ch004;
public class TestTransfer1 {
 public static void swap(DataSwap ds){
  int tmp = ds.a;
  ds.a = ds.b;
  ds.b = tmp;
  System.out.println("swap 方法里，a Field 的值是"+ ds.a + " ；b Field 的值是" + ds.b);
 }

 public static void main(String[] args) {
  DataSwap ds = new DataSwap();
  ds.a = 6;
  ds.b = 9;
  swap(ds);
  System.out.println(" 交换结束后，a Field 的值是"+ ds.a + " ；b Field 的值是" + ds.b);
 }
}
```

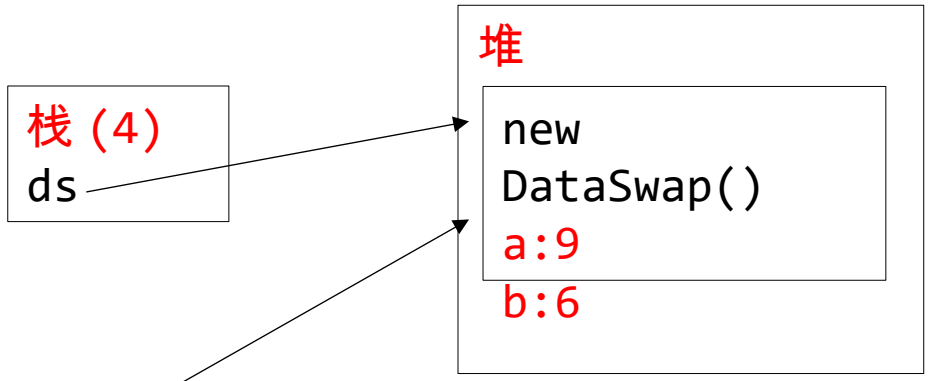
请输出结果

方法的参数传递

```java
//DataSwap.java
package ch004;
class DataSwap{
 public int a;
 public int b;
}
```

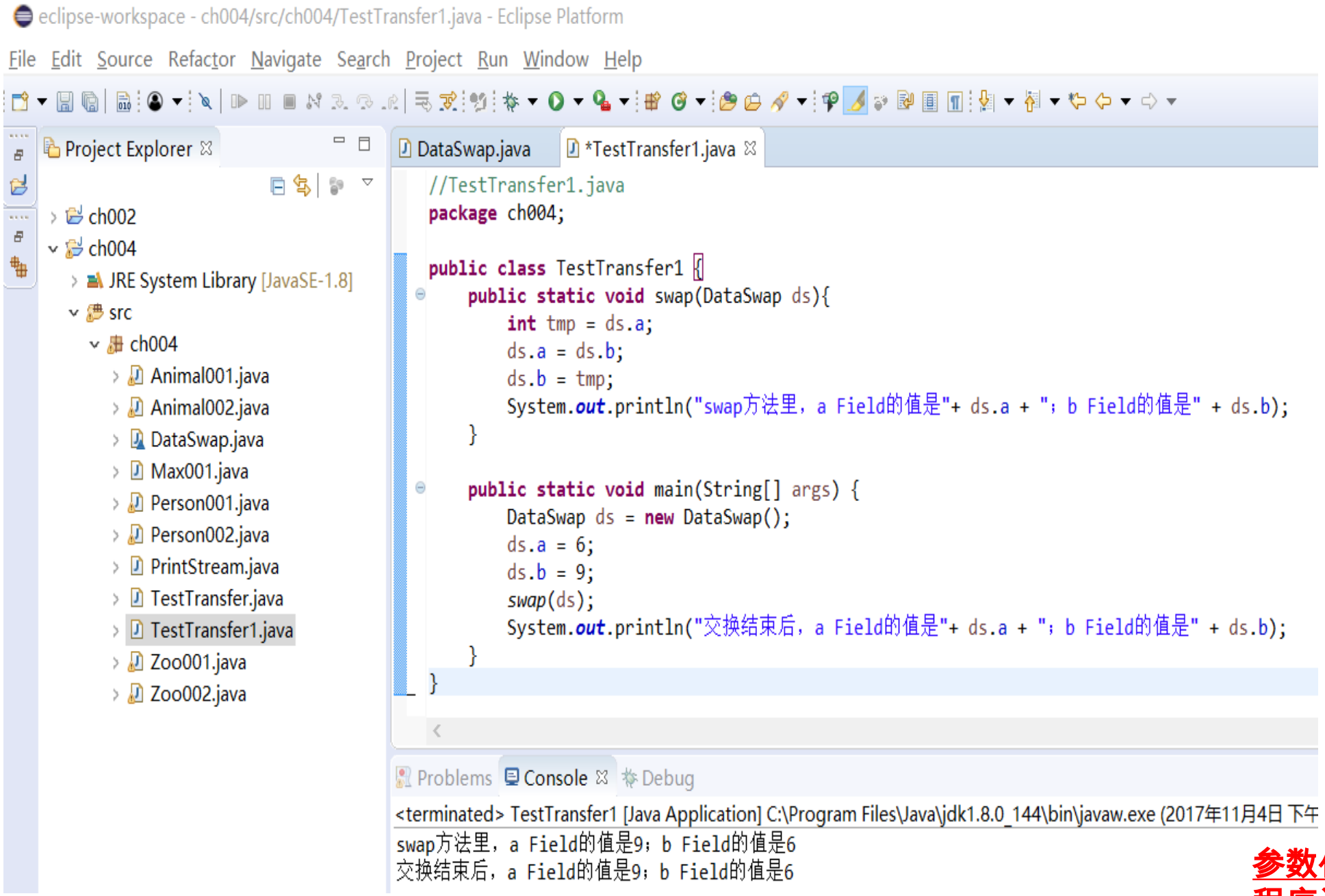
基本类型数据作为参数是值拷贝

对象作为参数是传引用，可以认为是把对象的堆地址传过去




```java
//TestTransfer1.java
package ch004;
public class TestTransfer1 {
 public static void swap(DataSwap ds){
  int tmp = ds.a;
  ds.a = ds.b;
  ds.b = tmp;
  System.out.println("swap 方法里，a Field 的值是"+ ds.a + " ；b Field 的值是" + ds.b);
 }

 public static void main(String[] args) {
  DataSwap ds = new DataSwap();
  ds.a = 6;
  ds.b = 9;
  swap(ds);
  System.out.println(" 交换结束后，a Field 的值是"+ ds.a + " ；b Field 的值是" + ds.b);
 }
}
```

请输出结果


eclipse-workspace - ch004/src/ch004/TestTransfer1.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
ch002
ch004
JRE System Library [JavaSE-1.8]
src
ch004
Animal001.java
Animal002.java
DataSwap.java
Max001.java
Person001.java
Person002.java
PrintStream.java
TestTransfer.java
TestTransfer1.java
Zoo001.java
Zoo002.java
DataSwap.java
*TestTransfer1.java
//TestTransfer1.java
package ch004;
public class TestTransfer1 {
public static void swap(DataSwap ds){
int tmp = ds.a;
ds.a = ds.b;
ds.b = tmp;
System.out.println("swap方法里，a Field的值是"+ ds.a + "；b Field的值是" + ds.b);
}
public static void main(String[] args) {
DataSwap ds = new DataSwap();
ds.a = 6;
ds.b = 9;
swap(ds);
System.out.println("交换结束后，a Field的值是"+ ds.a + "；b Field的值是" + ds.b);
}
}
Problems
Console
Debug
<terminated> TestTransfer1 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年11月4日 下午
swap方法里，a Field的值是9；b Field的值是6
交换结束后，a Field的值是9；b Field的值是6

参数传递
程序调试

```
class Value {
int i = 15;
}
```

基本类型数据作为参数是值拷贝

对象作为参数是传引用，可以认为是把对象的堆地址传过去

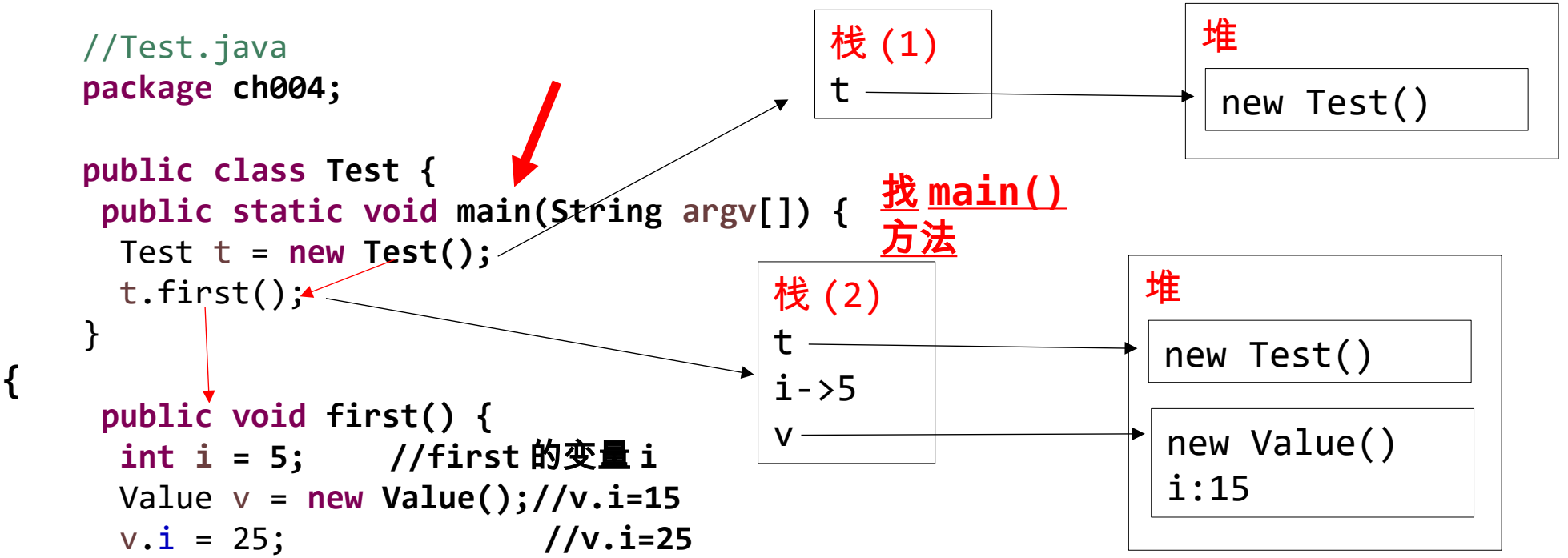
```
class Test {
public static void main(String argv[]) {
Test t = new Test();
t.first();
}
```

```
public void first() {
int i = 5;
Value v = new Value();
v.i = 25;
second(v, i);
System.out.println(v.i);
}
```

```
public void second(Value v, int i) {
i = 0;
v.i = 20;
Value val = new Value();
v = val;
System.out.println(v.i + " " + i);
}
}
```

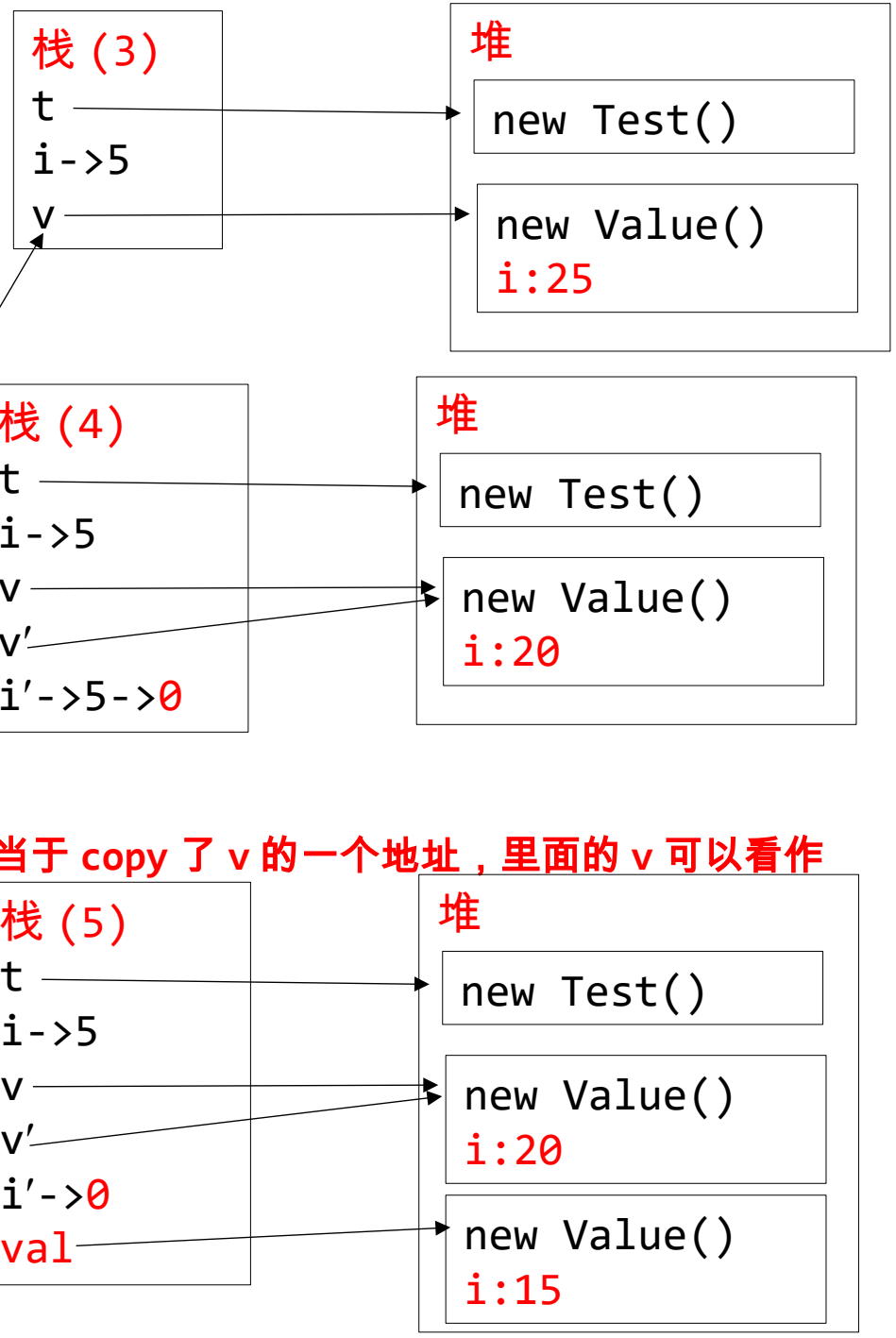
```java
//Value.java
package ch004;

public class Value {
int i = 15;
}
```

```java
//Test.java
package ch004;

public class Test {
 public static void main(String argv[]) {
  Test t = new Test();
  t.first();
}

 public void first() {
  int i = 5;     //first 的变量 i
  Value v = new Value();//v.i=15
  v.i = 25;                //v.i=25
  second(v, i);// 25, 5
  System.out.println(v.i);//20
 }

 public void second(Value v, int i) {// 相当于 copy 了 v 的一个地址，里面的 v 可以看作 v'
  i = 0;                                //i=0
  v.i = 20;//20!v 指向的地址位置改成 20
  Value val = new Value();//15
  v = val;//15
  System.out.println(v.i + " " + i);
 }
}
```



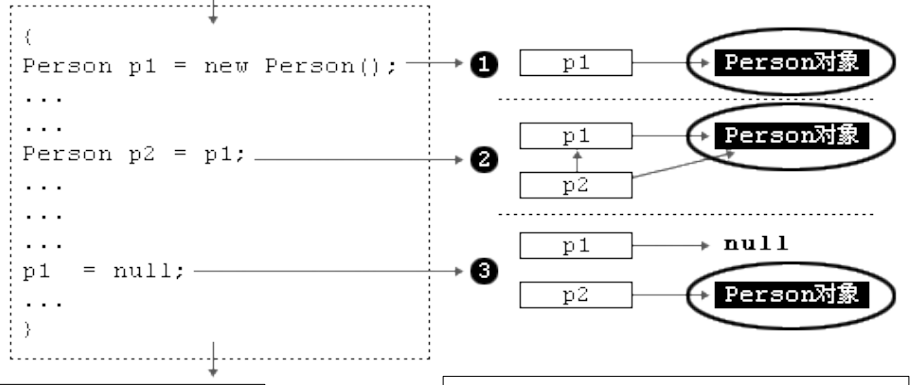
```java
//Value.java
package ch004;

public class Value {
int i = 15;
}
```

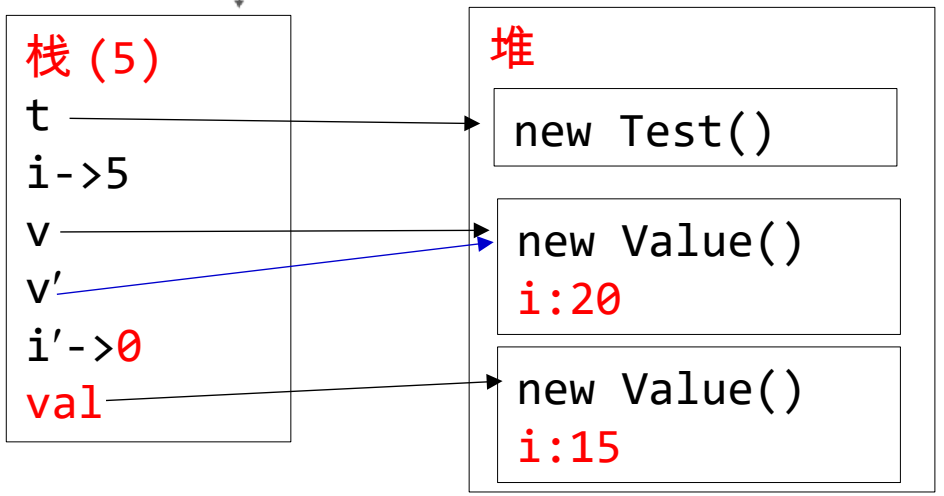
```java
//Test.java
package ch004;

public class Test {
 public static void main(String argv[]) {
  Test t = new Test();
  t.first();
 }

 public void first() {
  int i = 5;     //first 的变量 i
  Value v = new Value();//v.i=15
  v.i = 25;                //v.i=25
  second(v, i);// 25, 5
  System.out.println(v.i);//20
 }
 public void second(Value v, int i) {// 相当于 copy 了 v 的一个地址，里面的 v 可以看作 v′
  i = 0;                                //i=0
  v.i = 20;//20!v 指向的地址位置改成 20
  Value val = new Value();//15
  v = val;//15
  System.out.println(v.i + " " + i);
 }
}
```

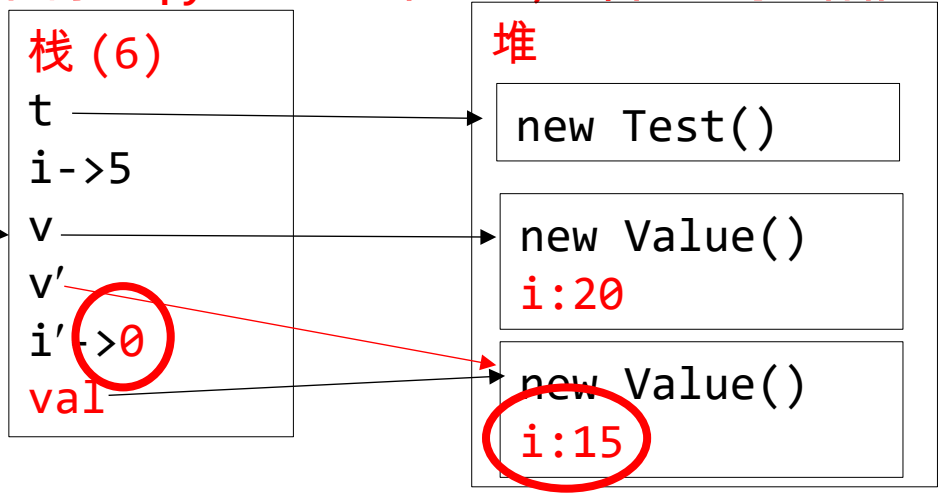


```java
//Value.java
package ch004;

public class Value {
int i = 15;
}
```

```java
//Test.java
package ch004;

public class Test {
 public static void main(String argv[]) {
  Test t = new Test();
  t.first();
}

 public void first() {
  int i = 5;     //first 的变量 i
  Value v = new Value();//v.i=15
  v.i = 25;                //v.i=25
  second(v, i);// 25, 5
  System.out.println(v.i);//20
 }
 public void second(Value v, int i) {// 相当于 copy 了 v 的一个地址，里面的 v 可以看作 v'
  i = 0;                                //i=0
  v.i = 20;//20!v 指向的地址位置改成 20
  Value val = new Value();//15
  v = val;//15
  System.out.println(v.i + " " + i);
 }
}
```

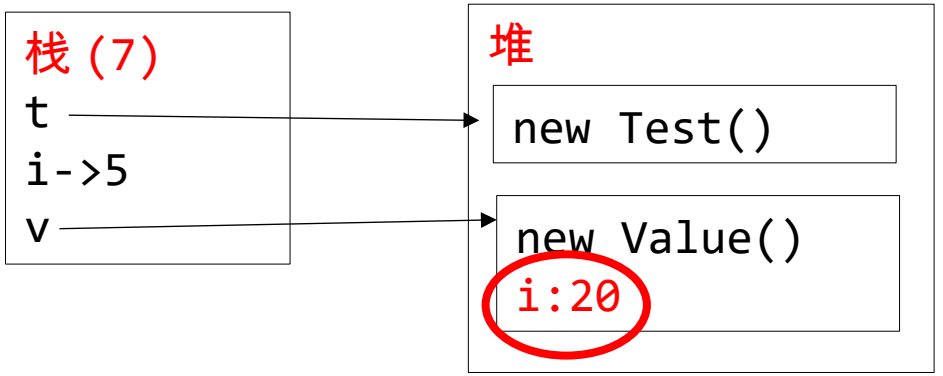


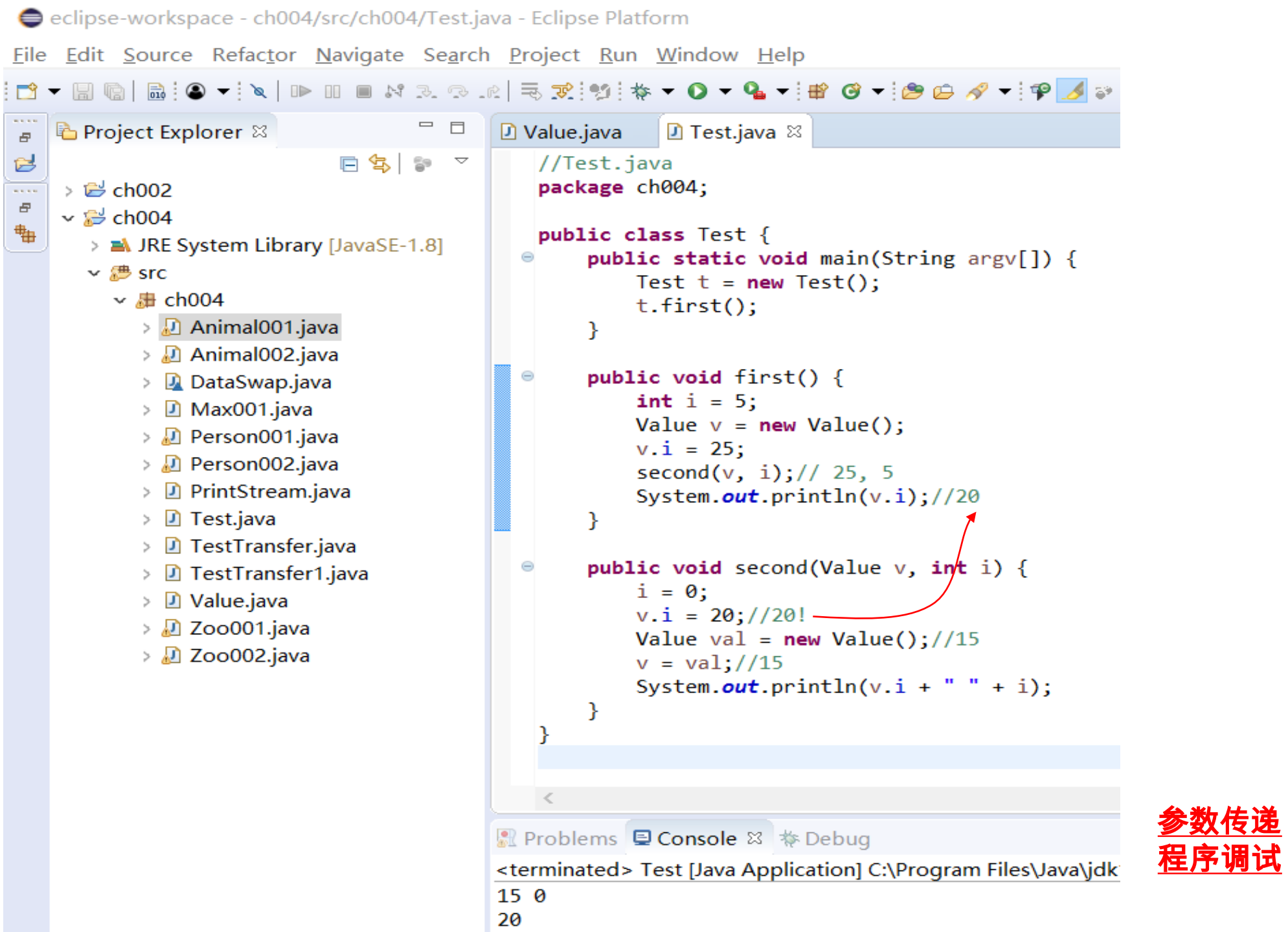




```java
//Value.java
package ch004;

public class Value {
int i = 15;
}
```

```java
//Test.java
package ch004;

public class Test {
 public static void main(String argv[]) {
  Test t = new Test();
  t.first();
}

 public void first() {
  int i = 5;     //first 的变量 i
  Value v = new Value();//v.i=15
  v.i = 25;                //v.i=25
  second(v, i);// 25, 5
  System.out.println(v.i);//20
 }
 public void second(Value v, int i) {// 相当于 copy 了 v 的一个地址，里面的 v 可以看作 v'
  i = 0;                                //i=0
  v.i = 20;//20!v 指向的地址位置改成 20
  Value val = new Value();//15
  v = val;//15
  System.out.println(v.i + " " + i);
 }
}
```



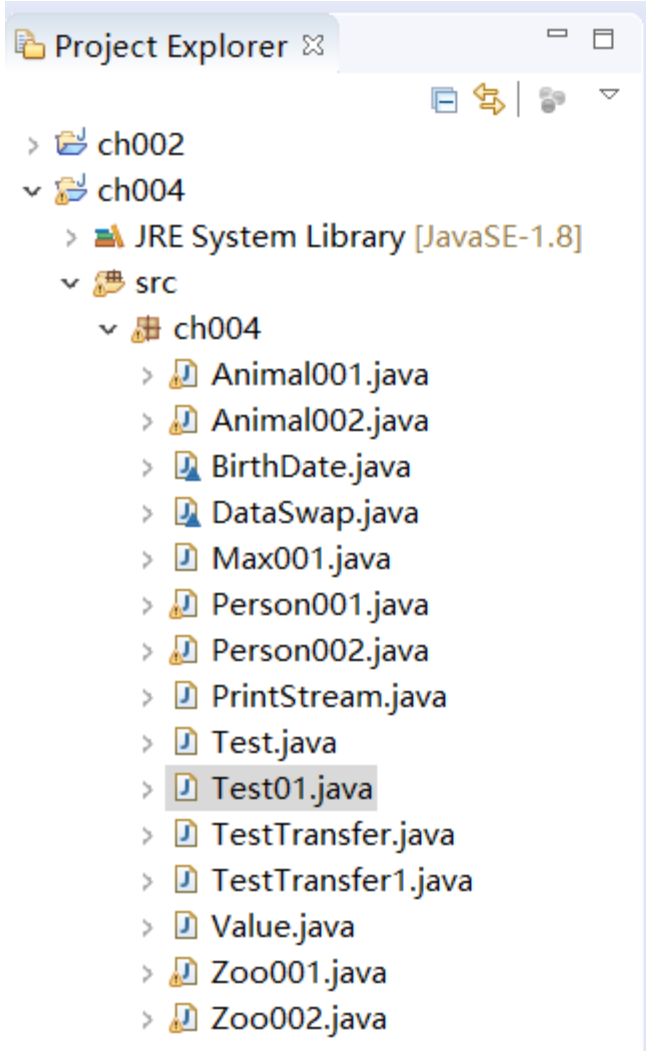

eclipse-workspace - ch004/src/ch004/Test.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
ch002
ch004
JRE System Library [JavaSE-1.8]
src
ch004
Animal001.java
Animal002.java
DataSwap.java
Max001.java
Person001.java
Person002.java
PrintStream.java
Test.java
TestTransfer.java
TestTransfer1.java
Value.java
Zoo001.java
Zoo002.java
Value.java
Test.java
//Test.java
package ch004;
public class Test {
public static void main(String argv[]) {
Test t = new Test();
t.first();
}
public void first() {
int i = 5;
Value v = new Value();
v.i = 25;
second(v, i);// 25, 5
System.out.println(v.i);//20
}
public void second(Value v, int i) {
i = 0;
v.i = 20;//20!
Value val = new Value();//15
v = val;//15
System.out.println(v.i + " " + i);
}
}
Problems
Console
Debug
<terminated> Test [Java Application] C:\Program Files\Java\jdk
15 0
20

参数传递
程序调试

再体会参数的传递

基本类型数据作为参数是值拷贝

对象作为参数是传引用，可以认为是把对象的堆地址传过去



*BirthDate.java Test01.jav

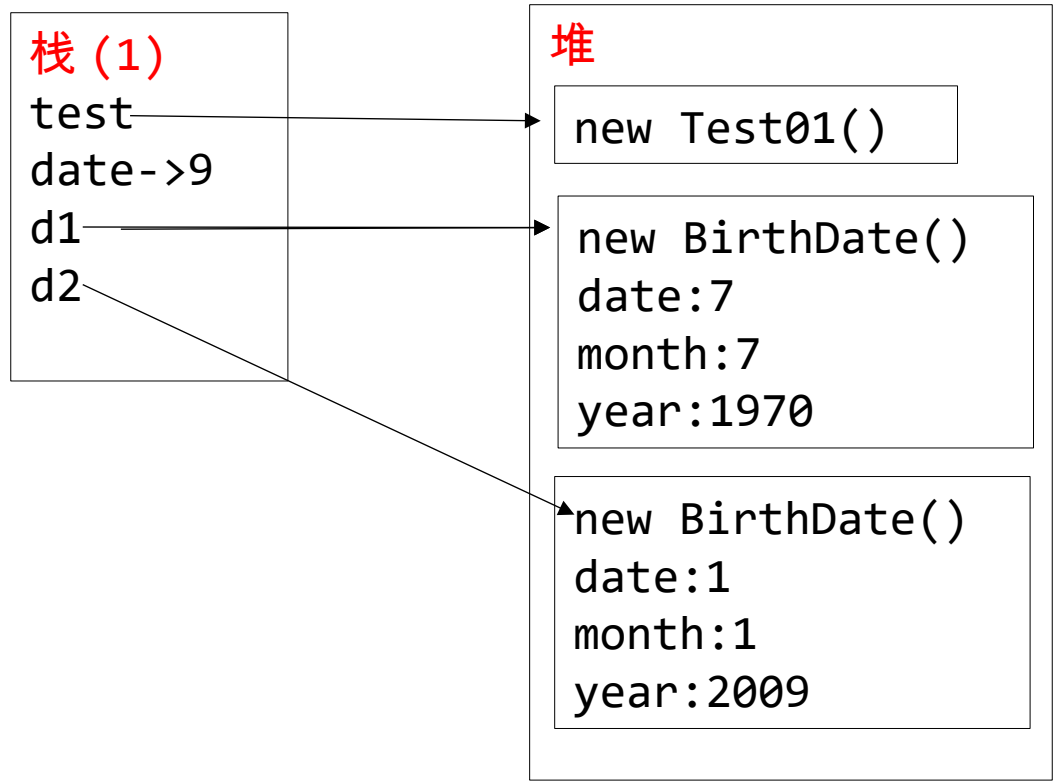
```java
//BirthData.java
package ch004;

class BirthDate{
    private int day;
    private int month;
    private int year;
    public BirthDate(int d,int m,int y){
        day = d; month = m; year = y;
    }
    public void setDay(int d){day = d;}
    public void setMonth(int m){month = m;}
    public void setYear(int y){year = y;}
    public int getDay(){return day;}
    public int getMonth(){return month;}
    public int getYear(){return year;}

    public void display(){
        System.out.println(day+"-"+month+"-"+year);
    }
}
```

*BirthDate.java | Test01.java

```java
//Test01.java
package ch004;

public class Test01 {
    public void change1(int i){
        i = 1234;
    }
    public void change2(BirthDate b){
        b = new BirthDate(22,3,2004);
    }
    public void change3(BirthDate b){
        b.setDay(22);
    }
    public static void main(String[] args) {
        Test01 test = new Test01();
        int date = 9;
        BirthDate d1 = new BirthDate(7,7,1970);
        BirthDate d2 = new BirthDate(1,1,2009);
        test.change1(date);//data = 1234形参
        test.change2(d1);  //b = new BirthDate(22,3,2004)
        test.change3(d2);  //b.setDay(22)
        System.out.println("date="+date);//形参，实际没变
        d1.display();//
        d2.display();//
    }
}
```

Problems | Console | Debug

```
<terminated> Test01 (1) [Java Application] C:\Program Files\Java\jdk1.8.0_1
date=9
7-7-1970
22-1-2009
```

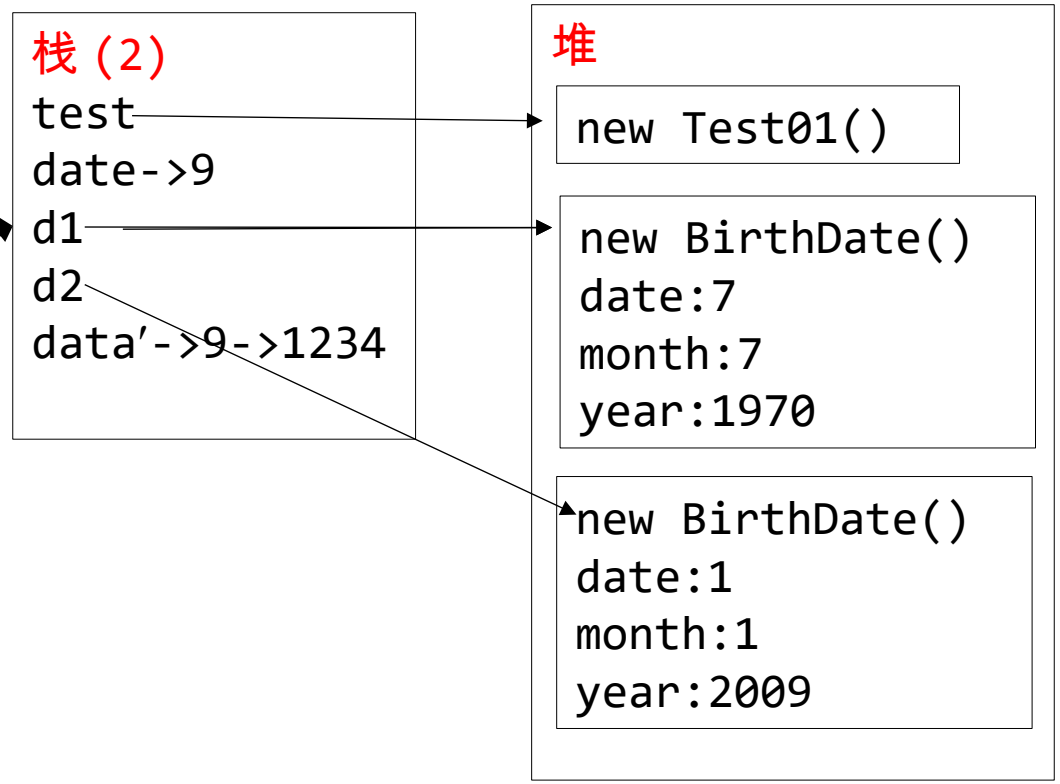


*BirthDate.java | Test01.java

```java
//Test01.java
package ch004;

public class Test01 {
    public void change1(int i){
        i = 1234;
    }
    public void change2(BirthDate b){
        b = new BirthDate(22,3,2004);
    }
    public void change3(BirthDate b){
        b.setDay(22);
    }
    public static void main(String[] args) {
        Test01 test = new Test01();
        int date = 9;
        BirthDate d1 = new BirthDate(7,7,1970);
        BirthDate d2 = new BirthDate(1,1,2009);
        test.change1(date);//data = 1234形参
        test.change2(d1);  //b = new BirthDate(22,3,2004)
        test.change3(d2);  //b.setDay(22)
        System.out.println("date="+date);//形参，实际没变
        d1.display();//
        d2.display();//
    }
}
```

Problems | Console | Debug

```
<terminated> Test01 (1) [Java Application] C:\Program Files\Java\jdk1.8.0_1
date=9
7-7-1970
22-1-2009
```

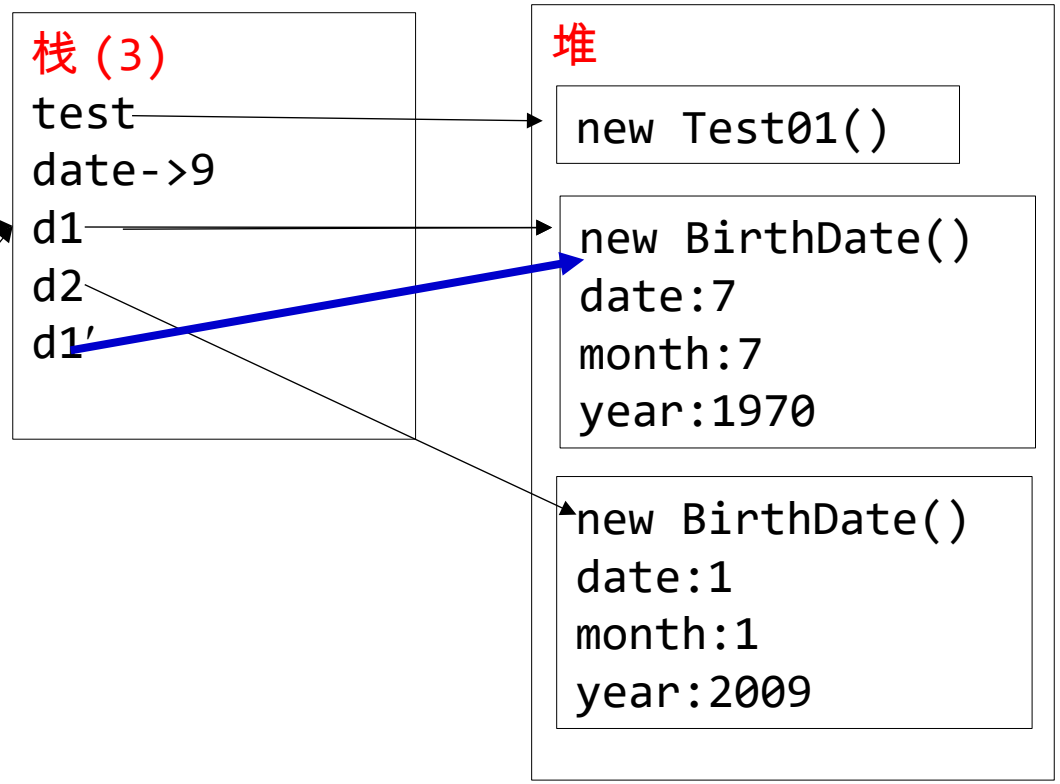


*BirthDate.java | Test01.java

```java
//Test01.java
package ch004;

public class Test01 {
    public void change1(int i){
        i = 1234;
    }
    public void change2(BirthDate b){
        b = new BirthDate(22,3,2004);
    }
    public void change3(BirthDate b){
        b.setDay(22);
    }
    public static void main(String[] args) {
        Test01 test = new Test01();
        int date = 9;
        BirthDate d1 = new BirthDate(7,7,1970);
        BirthDate d2 = new BirthDate(1,1,2009);
        test.change1(date);//data = 1234形参
        test.change2(d1);  //b = new BirthDate(22,3,2004)
        test.change3(d2);  //b.setDay(22)
        System.out.println("date="+date);//形参，实际没变
        d1.display();//
        d2.display();//
    }
}
```

Problems | Console | Debug

```
<terminated> Test01 (1) [Java Application] C:\Program Files\Java\jdk1.8.0_1
date=9
7-7-1970
22-1-2009
```



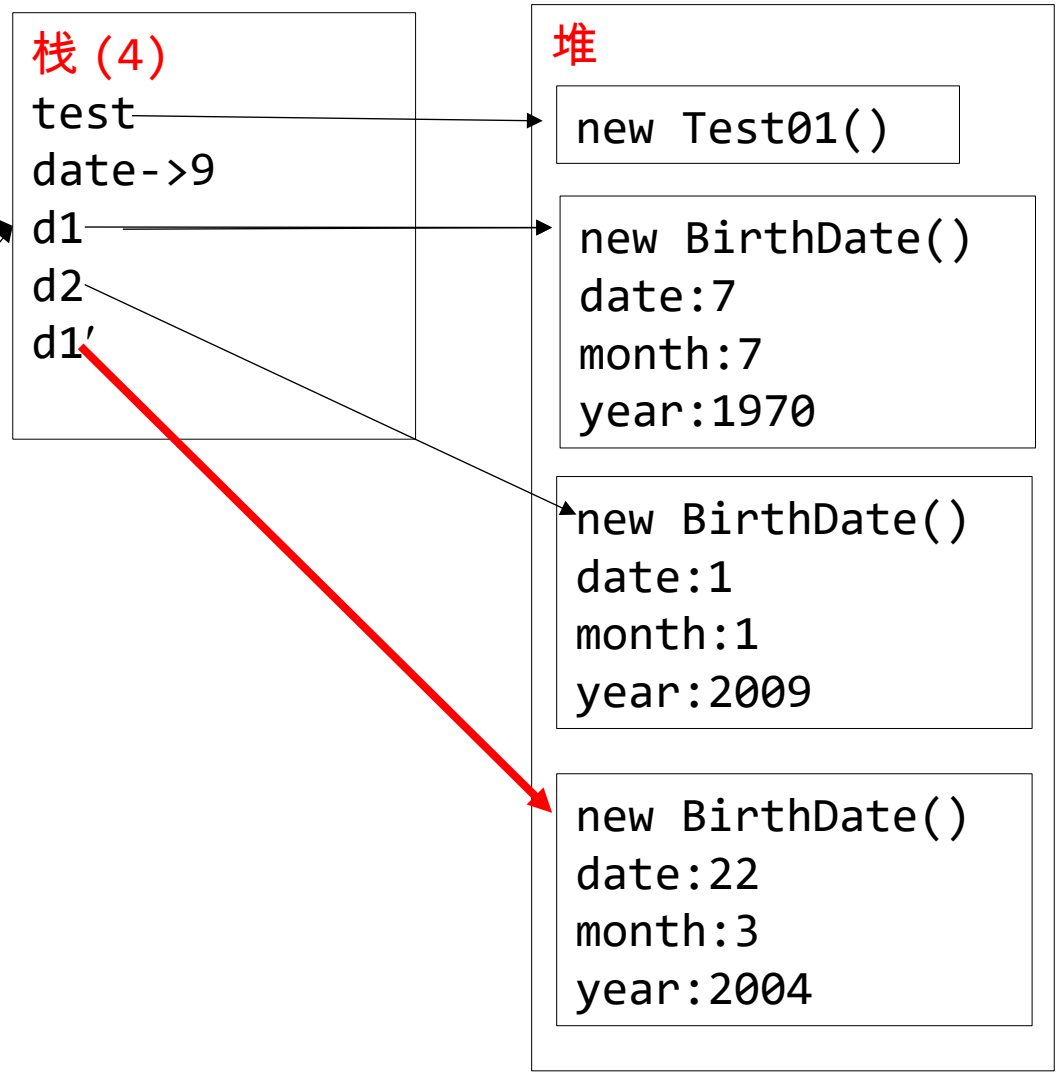
*BirthDate.java | Test01.java

```java
//Test01.java
package ch004;

public class Test01 {
    public void change1(int i){
        i = 1234;
    }
    public void change2(BirthDate b){
        b = new BirthDate(22,3,2004);
    }
    public void change3(BirthDate b){
        b.setDay(22);
    }
    public static void main(String[] args) {
        Test01 test = new Test01();
        int date = 9;
        BirthDate d1 = new BirthDate(7,7,1970);
        BirthDate d2 = new BirthDate(1,1,2009);
        test.change1(date);//data = 1234形参
        test.change2(d1);  //b = new BirthDate(22,3,2004)
        test.change3(d2);  //b.setDay(22)
        System.out.println("date="+date);//形参，实际没变
        d1.display();//
        d2.display();//
    }
}
```

Problems | Console | Debug

<terminated> Test01 (1) [Java Application] C:\Program Files\Java\jdk1.8.0_1

```
date=9
7-7-1970
22-1-2009
```

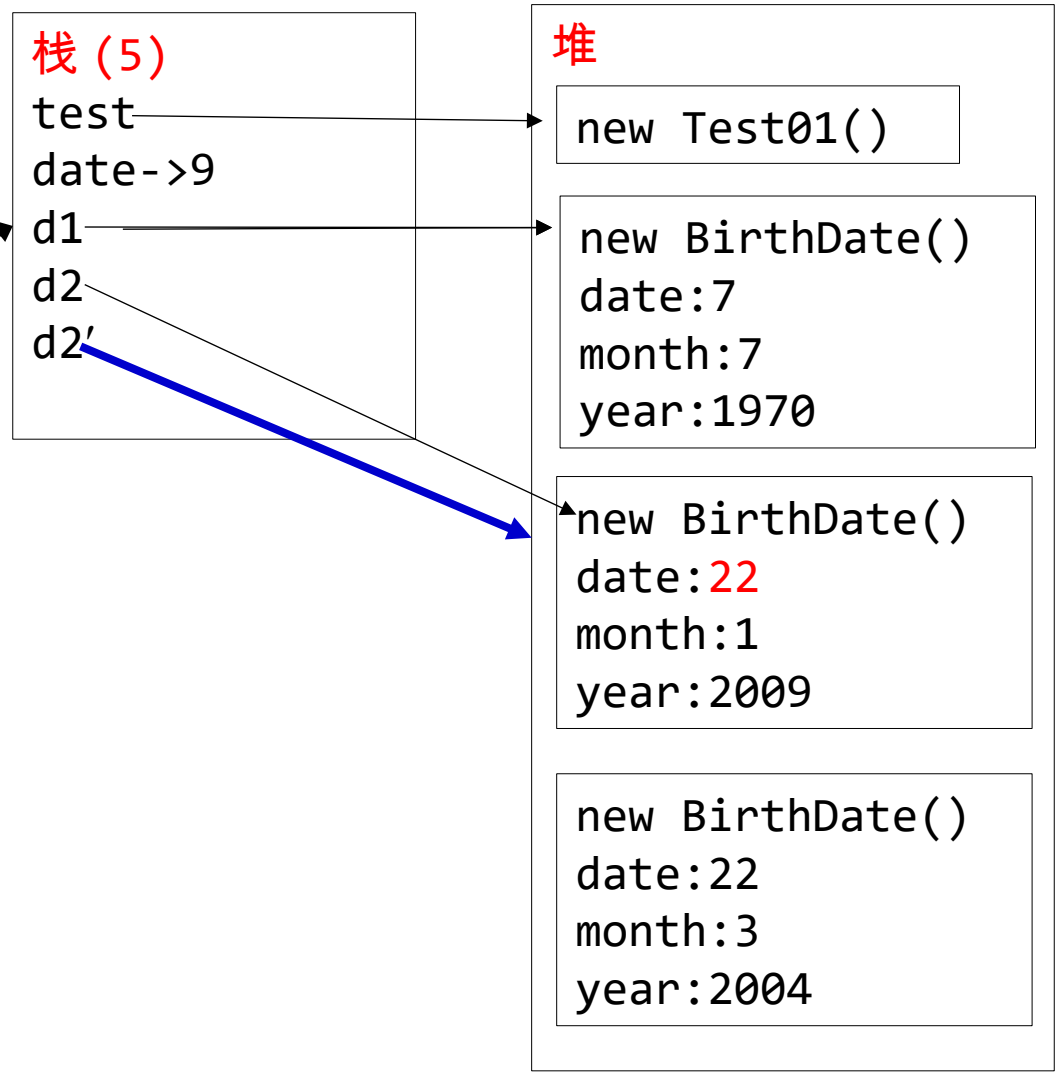


*BirthDate.java | Test01.java

```java
//Test01.java
package ch004;

public class Test01 {
    public void change1(int i){
        i = 1234;
    }
    public void change2(BirthDate b){
        b = new BirthDate(22,3,2004);
    }
    public void change3(BirthDate b){
        b.setDay(22);
    }
    public static void main(String[] args) {
        Test01 test = new Test01();
        int date = 9;
        BirthDate d1 = new BirthDate(7,7,1970);
        BirthDate d2 = new BirthDate(1,1,2009);
        test.change1(date);//data = 1234形参
        test.change2(d1);  //b = new BirthDate(22,3,2004)
        test.change3(d2);  //b.setDay(22)
        System.out.println("date="+date);//形参，实际没变
        d1.display();//
        d2.display();//
    }
}
```

Problems | Console | Debug

```
<terminated> Test01 (1) [Java Application] C:\Program Files\Java\jdk1.8.0_1
date=9
7-7-1970
22-1-2009
```

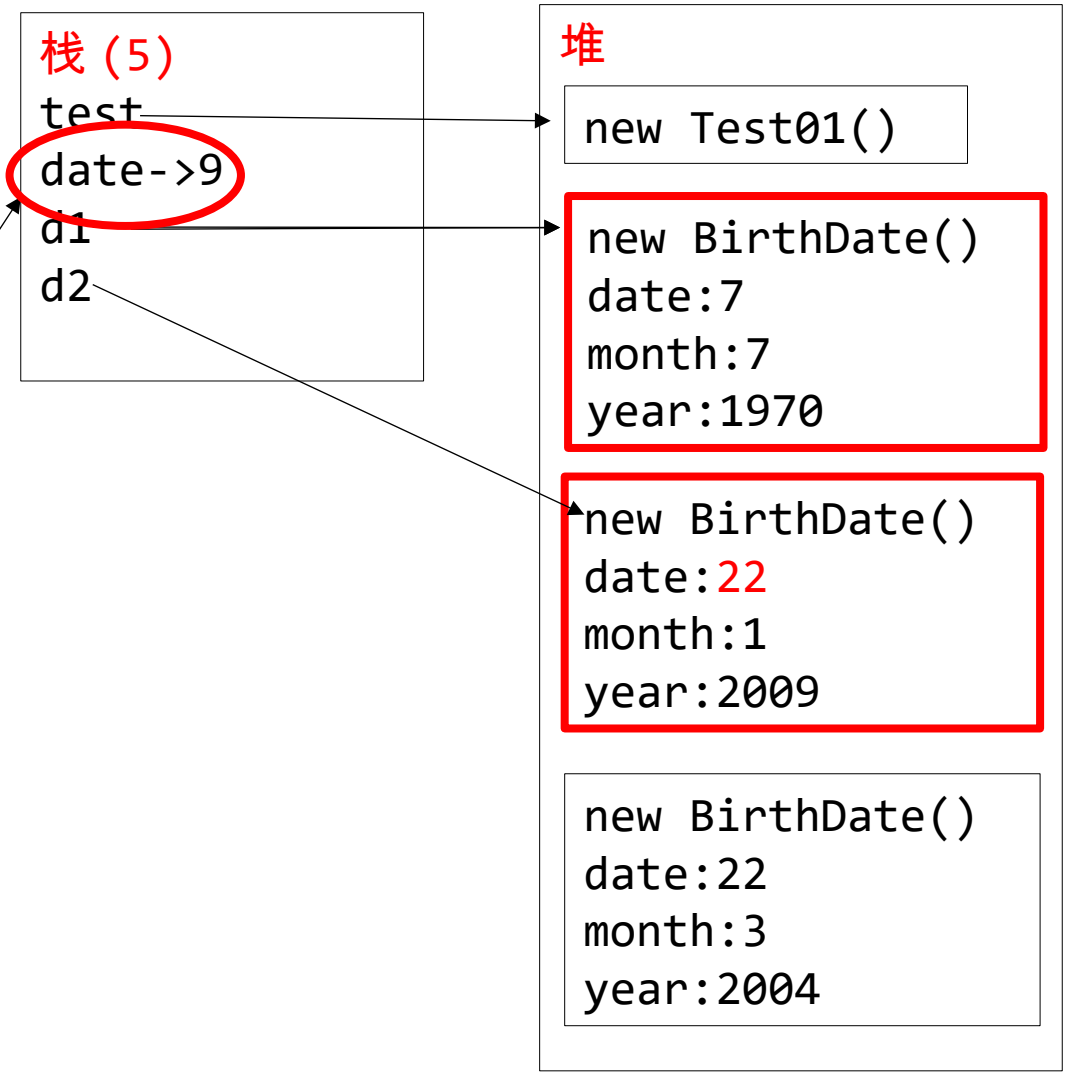


*BirthDate.java | Test01.java

```java
//Test01.java
package ch004;

public class Test01 {
    public void change1(int i){
        i = 1234;
    }
    public void change2(BirthDate b){
        b = new BirthDate(22,3,2004);
    }
    public void change3(BirthDate b){
        b.setDay(22);
    }
    public static void main(String[] args) {
        Test01 test = new Test01();
        int date = 9;
        BirthDate d1 = new BirthDate(7,7,1970);
        BirthDate d2 = new BirthDate(1,1,2009);
        test.change1(date);//data = 1234形参
        test.change2(d1);  //b = new BirthDate(22,3,2004)
        test.change3(d2);  //b.setDay(22)
        System.out.println("date="+date);//形参，实际没变
        d1.display();//
        d2.display();//
    }
}
```

Problems | Console | Debug

```
<terminated> Test01 (1) [Java Application] C:\Program Files\Java\jdk1.8.0_1
date=9
7-7-1970
22-1-2009
```



4.7 面向对象特征之一：封装和隐藏

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.");
   }
 }
```

应该将 legs 属性保护起来，防止乱用。

保护的方式：信息隐藏

```
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();
    } }
```

问题： xb.legs = -1000;

信息的封装和隐藏

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**；　　通过方法访问属性
- 便于修改，增强代码的可维护性；

信息的封装和隐藏

封装和隐藏

```
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());
  } }
```

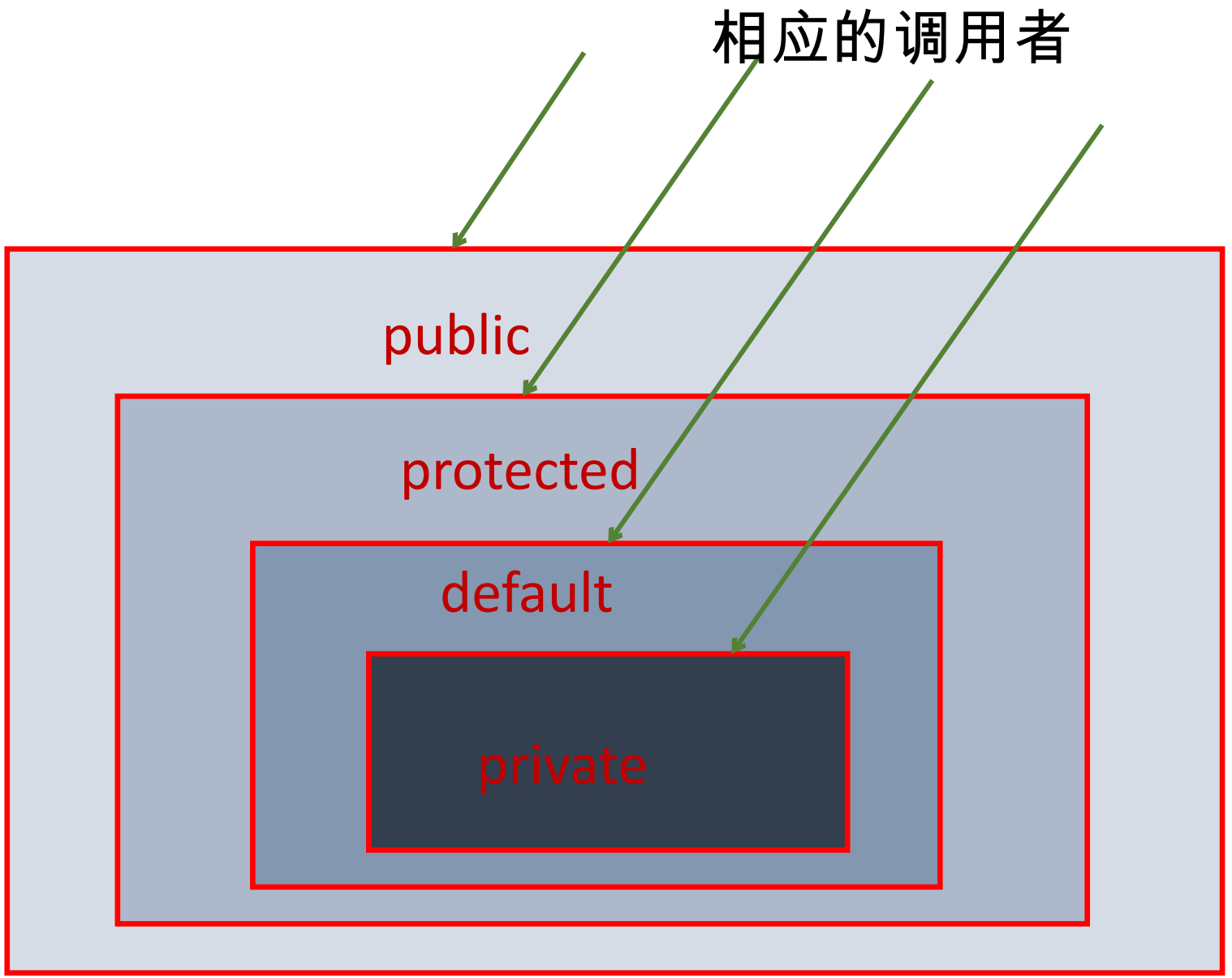
四种访问权限修饰符

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。



封装和隐藏

安全隐私措施？

1）权限分级控制

2）上（区块）链

3）各类隐私和安全算法

包括差分、安全多方计算、隐私计算、同态加密

4）联邦学习（分布式的模型联合训练）

4.8 类的成员之三：构造器 (构造方法)

构造器

●构造器的特征

- ➢它具有与类相同的名称
- ➢它不声明返回值类型。 (与声明为 void 不同)
- ➢不能被 static 、 final 、 synchronized 、 abstract 、 native 修饰 ，不能有 return 语句返回值

●构造器的作用 ：创建对象；给对象进行初始化

- ➢如： Order o = new Order(); Person p = new Person(Peter,15);
- ➢如同我们规定每个“人”一出生就必须先洗澡，我们就可以在“人”的构造方法中加入完成“洗澡”的程序代码，于是每个“人”一出生就会自动完成“洗澡”，程序就不必再在每个人刚出生时一个一个地告诉他们要“洗澡”了。

构造器

- 语法格式：

  修饰符 类名（参数列表）{

  初始化语句；

  }

- 举 例：

```
public class Animal {
    private int legs;
    public Animal() {legs = 4; }        // 构造器
    public void setLegs(int i) { legs = i; }
    public int getLegs(){return legs;}
}
```

构造器举例

创建 Animal 类的实例： Animal a=new Animal();
// 调用构造器，将 legs 初始化为 4 。

构造器

- 根据参数不同，构造器可以分为如下两类：
  - 隐式无参构造器（系统默认提供）
  - 显式定义一个或多个构造器（无参、有参）

构造器

- 注　意：
  - Java 语言中，每个类都至少有一个构造器
  - 默认构造器的修饰符与所属类的修饰符一致
  - 一旦显式定义了构造器，则系统不再提供默认构造器
  - 一个类可以创建多个重载的构造器
  - 父类的构造器不可被子类继承

构造器重载

- 构造器一般用来创建对象的同时初始化对象。如

```
class Person{
    String name;
    int age;
    public Person(String n , int a){  name=n; age=a;}
}
```

构造器重载
即有多个构造方案

- 构造器重载使得对象的创建更加灵活，方便创建各种不同的对象。

构造器重载举例：

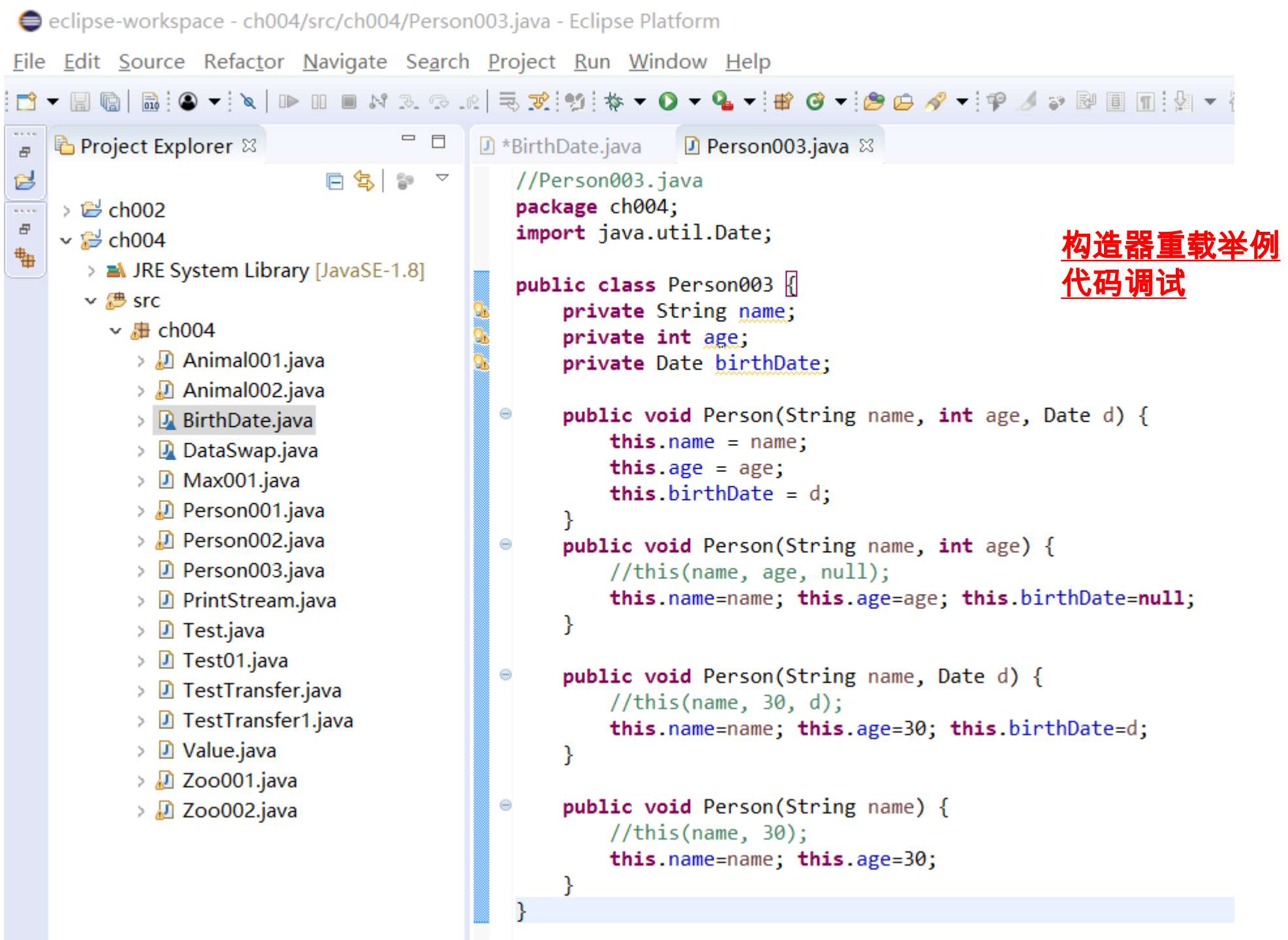
```
public class Person{
    public Person(String name, int age, Date d) {this(name,age);…}
    public Person(String name, int age) {…}
    public Person(String name, Date d) {…}
    public Person(){…}
}
```

- 构造器重载，参数列表必须不同

构造器重载举例

```
public class Person {
    private String name;
    private int age;
    private Date birthDate;
    public Person(String name, int age, Date d) {
            this.name = name;
            this.age = age;
            this.birthDate = d;
     }
     public Person(String name, int age) {
            this(name, age, null);
            //this.name=name; this.age=age; this.birthDate=null;
     }
     public Person(String name, Date d) {
            this(name, 30, d);
            //this.name=name; this.age=30; this.birthDate=d;
     }
     public Person(String name) {
             this(name, 30);      //this.name=name; this.age=30;
     }
}
```

构造器重载举例


eclipse-workspace - ch004/src/ch004/Person003.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
ch002
ch004
JRE System Library [JavaSE-1.8]
src
ch004
Animal001.java
Animal002.java
BirthDate.java
DataSwap.java
Max001.java
Person001.java
Person002.java
Person003.java
PrintStream.java
Test.java
Test01.java
TestTransfer.java
TestTransfer1.java
Value.java
Zoo001.java
Zoo002.java
*BirthDate.java
Person003.java
//Person003.java
package ch004;
import java.util.Date;
public class Person003 {
private String name;
private int age;
private Date birthDate;
public void Person(String name, int age, Date d) {
this.name = name;
this.age = age;
this.birthDate = d;
}
public void Person(String name, int age) {
//this(name, age, null);
this.name=name; this.age=age; this.birthDate=null;
}
public void Person(String name, Date d) {
//this(name, 30, d);
this.name=name; this.age=30; this.birthDate=d;
}
public void Person(String name) {
//this(name, 30);
this.name=name; this.age=30;
}
}
构造器重载举例
代码调试

4.9 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 。** Java 中的 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

this 注意事项

- **使用 this 调用本类的构造器**

4.this 可以作为一个类中，构造器相互调用的特殊格式

```
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 ;
        } }
```

this 调用本类构造器

this 调用本类构造器

注意：

1. 使用 this() 必须放在构造器的首行！ this 调用本类构造器

2. 使用 this 调用本类中其他的构造器，保证至少有一个构造器是不用 this 的。 this 调用本类构造器

当前正在操作本方法的对象称为当前对象。

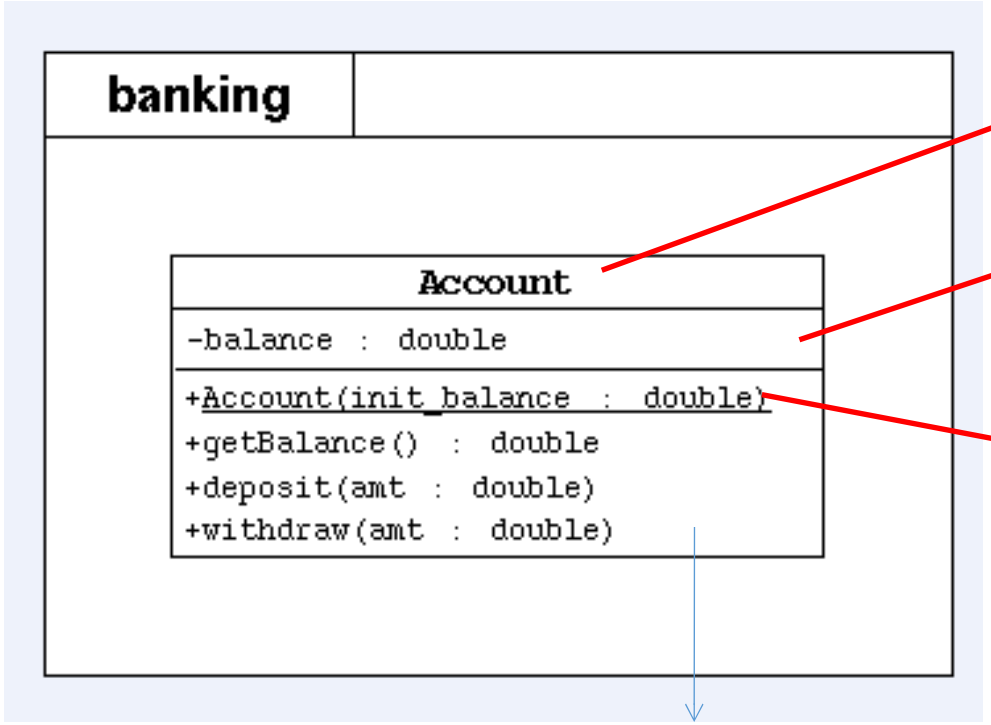
Person004.java | TestPerson.java

```java
//Person004.java
package ch004;

public class Person004 {
    String name;
    Person004(String name){
        this.name = name;
    }
    public void getInfo(){
        System.out.println("Person类 --> " + this.name) ;
    }
    public boolean compare(Person004 p){
        return this.name==p.name;
    }
}
```

this 用法举例

this 用法举例

补充： UML 类图



类名

属性：“：”前是属性名，“：”后是属性的类型

若方法有下划线
表示为构造方法

方法

类 / 对象的图示化表示
统一建模语言

1. + 表示 public 类型， - 表示 private 类型， # 表示 protected 类型

2. 方法的写法：
方法的类型 (+ 、 -) 方法名 (参数名： 参数类型) ：返回值类型

关键字— package

源文件布局：

- Java 源文件的基本语法：

  [<包声明>]
  [<导入声明>]
  <类声明>+

- 示例，VehicleCapacityReport.java 文件：

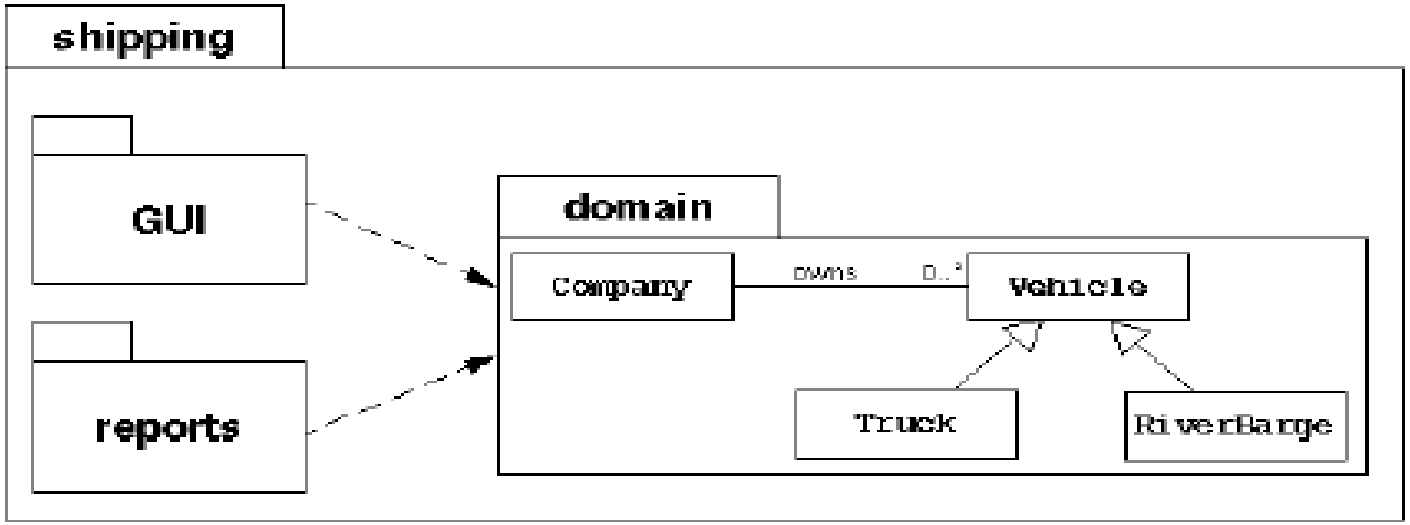
```
package shipping.reports;

import shipping.domain.*;
import java.util.List;
import java.io.*;

public class VehicleCapacityReport {
   private List  vehicles;
   public void generateReport(Writer output) {...}
}
```

统一建模语言
UML
package

统一建模语言
UML
package

软件包：

- 包帮助管理大型软件系统：将语义近似的类组织到包中；解决类命名冲突的问题。
- 包可以包含类和子包。
- 例：某航运软件系统包括：一组域对象、 GUI 和 reports 子系统



关键字— package

- package 语句作为 Java 源文件的第一条语句，指明该文件中定义的类所在的包。（若缺省该语句，则指定为无名包）。它的格式为：

  package 顶层包名 . 子包名 ；

  举例： pack\Test.java

```
package p1;   // 指定类 Test 属于包 p1
public class Test{
    public void display(){
        System.out.println("in  method display()");
    }
}
```

统一建模语言
UML
package

- **包对应于文件系统的目录， package 语句中，用“ . ”来指明包（目录）的层次；**
- **包通常用小写单词，类名首字母通常大写。**

关键字— import

- 为使用定义在不同包中的 Java 类，需用 import 语句来引入指定包层次下所需要的类或全部类 (.*) 。 import 语句告诉编译器到哪里去寻找类。

统一建模语言
UML
import

- **语法格式：**

import 包名 [. 子包名…]. < 类名 |*>

- **应用举例：**

```
import  p1.Test;   //import p1.*; 表示引入 p1 包中的所有类
public class TestPackage{
        public static void main(String args[]){
            Test t = new Test();        //Test 类在 p1 包中定义
            t.display();
        }
 }
```

import 语句

统一建模语言
UML
import

- **注意：**
  - 若引入的包为： java.lang ，则编译器默认可获取此包下的类，不需要再显示声明。
  - import 语句出现在 package 语句之后、类定义之前
  - 一个源文件中可包含多个 import 语句
  - 可以使用 **import lee.* ;** 语句，表明导入 lee 包下的所有类。而 lee 包下 sub 子包内的类则不会被导入。 import lee.sub.*;
  - import 语句不是必需的，可坚持在类里使用其它类的全名
  - JDK 1.5 加入 import static 语句

JDK 中主要的包介绍

统一建模语言 UML 标准库 JDK 中的 package

1. **java.lang----** 包含一些 Java 语言的核心类，如 String、Math、Integer、System 和 Thread，提供常用功能。
2. **java.net----** 包含执行与网络相关的操作的类和接口。
4. **java.io ----** 包含能提供多种输入 / 输出功能的类。

4. **java.util----** 包含一些实用工具类，如定义系统特性、接口的集合框架类、使用与日期日历相关的函数。

5. **java.text----** 包含了一些 java 格式化相关的类

6. **java.sql----** 包含了 java 进行 JDBC 数据库编程的相关类 / 接口

7. **java.awt----** 包含了构成抽象窗口工具集（abstract window toolkits）的多个类，这些类被用来构建和管理应用程序的图形用户界面 (GUI)。

8. **java.applet----** 包含 applet 运行所需的一些类。

面向过程 (OP) 和面向对象 (OO) 是不是就是指编码的两种方式呢？不是！

拿到了一个用户需求，要编个软件，需要经过需求分析，然后进行总体 / 详细设计，最后编码，才能最终写出软件，交付给用户。

这个过程是符合人类基本行为方式的：先想做什么，再想如何去做，最后才是做事情。有的同学说："我没按照你说的步骤做啊，我是直接编码的"。其实，你一定会经历了这三个阶段，只不过你潜意识里没有分得那么清楚。对于拿到需求就编码的人，可能编着编着，又得倒回去重新琢磨，还是免不了这些过程，

以 OO 为例，对应于软件开发的过程，**OO 衍生出 3 个概念： OOA 、 OOD 和 OOP** 。

采用面向对象进行分析的方式称为 OOA ，采用面向对象进行设计的方式称为 OOD ，采用面向对象进行编码的方式称为 OOP 。

面向过程 (OP) 和面向对象 (OO) 本质的区别在于分析方式的不同，最终导致了编码方式的不同。

之前课堂作业参考答案

1、 Java 的变量名称如何组成？

2、如何提高程序可读性？

4、请写出下面程序的运行结果

```
1  public class Draw {
2      public static void main(String[] args) {
3          for (int m = 1; m <= 4; m++) {
4              for (int n = 0; n <= m; n++) {
5                  System.out.print(" ");
6              }

7              for (int x = 1; x <= 7 -2 * (m - 1); x++) {
8                  System.out.print("*");
9              }
10             System.out.println();
11         }
12     }
13 }
```

3、请写出下面程序的运行结果

参考答案

```
1  public class Test1 {

2      private static int a = 2;

3      private static int b = 3;
4
5      public static int MultiplicationOne(int m)

6      {
7          b = a * m;

8          return b;
9      }
10
11     public static int AddTwo(int a, int b)

12     {
13         b += MultiplicationOne(a);
14         return b;

15     }
16
17     public static void main(String[] args)
18     {
19         b = AddTwo(4, 7);
20         System.out.print(b);
21     }
22 }
```

之前课堂作业部分参考答案

参考答案

1、 Java 的变量名称如何组成？

答： Java 的变量名称可以由英文字母、数字、下划线（ _ ）和美元符号（ $ ）组成，但标识符不能以数字开头，也不能是 Java 中的保留关键字。此外， Java 的变量有大小写之分。

2、如何提高程序可读性？

答：

（ 1 ）在程序中加上批注；

（ 2 ）为变量取个有意义的名称；

（ 3 ）保持每一行只有一个语句；

（ 4 ）适当的缩进。

4、请写出下面程序的运行结果

```
1  public class Draw {
2      public static void main(String[] args) {
3          for (int m = 1; m <= 4; m++) {
4              for (int n = 0; n <= m; n++) {
5                  System.out.print(" ");
6              }

7              for (int x = 1; x <= 7 -2 * (m - 1); x++) {
8                  System.out.print("*");
9              }
10             System.out.println();
11         }
12     }
13 }
```

参考答案

答：

****

*****

***

*

3、请写出下面程序的运行结果

```
1  public class Test1 {
2      private static int a = 2;
3      private static int b = 3;
4
5      public static int MultiplicationOne(int m)
6      {
7          b = a * m;
8          return b;
9      }
10
11     public static int AddTwo(int a, int b)
12     {
13         b += MultiplicationOne(a);
14         return b;
15     }
16
17     public static void main(String[] args)
18     {
19         b = AddTwo(4, 7);
20         System.out.print(b);
21     }
22 }
```

参考答案

输出结果：

15

3 个月起每个月都生一对兔子，小兔子长到第三个月后每个月又生一对兔子，假如兔子都不死，问每个月的兔子总数为多少？（斐波那契数列）

题目的意思指的是每个月的兔子总对数；假设将兔子分为小中大三种，兔子从出生后三个月后每个月就会生出一对兔子，那么假定第一个月的兔子为小兔子，第二个月为中兔子，第三个月之后就为大兔子，那么第一个月分别有 1 、 0 、 0 ，第二个月分别为 0 、 1 、 0 ，第三个月分别为 1 、 0 、 1 ，第四个月分别为 ,1 、 1 、 1 ，第五个月分别为 2 、 1 、 2 ，第六个月分别为 3 、 2 、 3 ，第七个月分别为 5 、 3 、 5…… 。兔子总数分别为 ： 1 、 1 、 2 、 3 、 5 、 8 、 13…… 。

于是得出了一个规律，从第三个月起，后面的兔子总数都等于前面两个月的兔子总数之和，即为斐波那契数列。

参考答案

```
public class Test{
    public static void main(String[] args){
        int i = 1;
        for(i=1;i<=20;i++){
            System.out.println(" 兔子第 "+i+" 个月的总数为 :"+f(i));
        }
    }
    public static int f(int x){
        if(x==1 || x==2){
            return 1;
        }else{
            return f(x-1)+f(x-2);
        }
    }
}
```

周三课后作业部分答案

参考答案

16. 有一对兔子，从 3 个月起每个月都生一对兔子，小兔子长到第三个月后每个月又生一对兔子，假如兔子都不死，问每个月的兔子总数为多少？（斐波那契数列）

分析：

月份	兔子数	说明
1	1（对）	从开始有一对兔子
2	1	
3	1+1	原本有一对 从第三个月开始 生了一对 一共是两对兔子
4	1+1+1	生了第二对
5	1+1+1+1+1	生了第三对兔子 同时 3 月生的第一对兔子又生了一对
6	5+3=8	

兔子数目序列： 1 1 2 3 5 8 . 。 。 。 。 。 。 。 。 。

总结出规律：前两项之和就是第三项

所以：第 n 个月的兔子数目为： f(n)=f(n-1)+f(n-2)

参考答案

```
public class Rabbit {
    public static void main(String [] args){
        long [] m ;
        System.out.print("Please input month : ");
        Scanner s = new Scanner(System.in);
        int i = s.nextInt();
        m = new long[i];

        if(m.length >= 1){
            m[0] = m[1] = 1;// 第一和第二个月的兔子都是： 1
        }
        for(int j = 0; j < m.length; j++){
            if(j == 0 || j == 1){
                System.out.println("Month : " + (j+1) + "\tRabbit number is : " + m[j]);
            }else {
                m[j] = m[j-2] + m[j-1];// 第三个月以后都满足规律： m[j] = m[j-2] + m[j-1]
                System.out.println("Month : " + (j+1) + "\tRabbit number is : " + m[j]);
            }
        }
    }
}
```

课堂作业

1、 Java 的访问控制修饰符有哪些？它们的定义和使用对象分别是什么？

2、程序实现：某五星级酒店，资金雄厚，要招聘多名员工（经理、厨师、服务员）。入职的员工需要记录个人信息（姓名、工号、经理特有奖金属性）。要求如下：

1. 构造员工类（ Employee ）：包含 Name （姓名）、 NO （工号）和构造方法（通过形参完成对成员变量赋值）；

2. 构造经理（ Manager ）、厨师（ Cook ）和服务员（ Waiter ）三个员工类的子类，其中经随机方法得出获奖金员工；

3. 编写测试类：

向酒店中增加多名员工（其中包含 1 名经理， 1 名厨师、 2 名服务员）；

打印出获得奖金的员工。

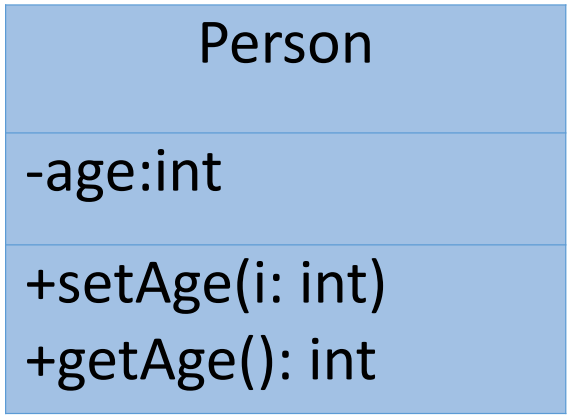
3、定义一个交通工具 (Vehicle) 类，其中有： 属性速度 (speed) 和体积 (size) 等等，方法有设置速度 (setSpeed(int speed)) 、加速 speedUp() 和减速 speedDown() 等等。然后在测试类 Vehicle 中的 main() 方法中实例化一个交通工具对象，并通过方法给它初始化 speed,size 的值并且通过打印出来。最后调用加速和减速的方法对其速度进行改变。

B- 第四周课后作业 -1

课后作业

1. 创建程序，在其中定义两个类： Person 和 TestPerson 类。定义如下：

   用 setAge() 设置人的合法年龄 (0~130) ，用 getAge() 返回人的年龄。在 TestPerson 类中实例化 Person 类的对象 b ，调用 setAge() 和 getAge() 方法，体会 Java 的封装性。

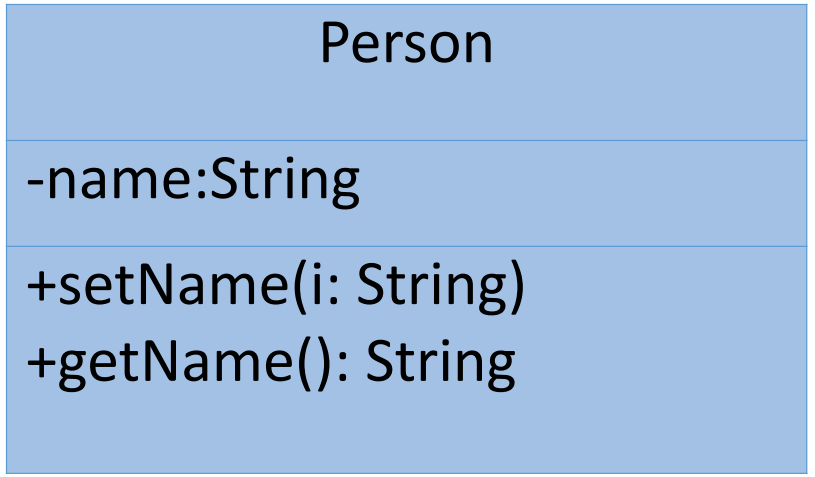


B- 第四周课后作业 -2

课后作业

2. 在前面定义的 Person 类中添加构造器，利用构造器设置所有人的 age 属性初始值都为 18 。

3. 修改上题中类和构造器，增加 name 属性，使得每次创建 Person 对象的同时初始化对象的 age 属性值和 name 属性值。



B- 第四周课后作业 -3

课后作业

(1) 定义 Person 类 , 有 4 个属性： String name; int age; String school;

String major

(2) 定义 Person 类的 3 个构造方法 :

- 第一个构造方法 Person(String n, int a) 设置类的 name 和 age 属性；
- 第二个构造方法 Person(String n, int a, String s) 设置类的 name, age 和 school 属性；
- 第三个构造方法 Person(String n, int a, String s, String m) 设置类的 name, age ,school 和 major 属性；

(3) 在 main 方法中分别调用不同的构造方法创建的对象，并输出其属性值。

B- 第四周课后作业 -4

课后作业

4. 添加必要的构造器，综合应用构造器的重载， this 关键字。

Girl
-name:String
+setName(i: String) +getName(): String +marry(boy:Boy)

Boy
-name:String -age:int
+setName(i: String) +getName(): String +setAge(i: int) +getAge(): int +marry(girl:Girl) +shout():void

B- 第四周课后作业 -5

课后作业

5. 定义一个“点”（ Point ）类用来表示三维空间中的点（有三个坐标）。要求如下：

1 ）可以生成具有特定坐标的点对象。

2 ）提供可以设置三个坐标的方法。

3 ）提供可以计算该“点”距原点距离平方的方法。

6. 编写两个类， TriAngle 和 TestTriAngle ，其中 TriAngle 中声明私有的底边长 base 和高 height ，同时声明公共方法访问私有变量；另一个类中使用这些公共方法，计算三角形的面积。

B- 第四周课后作业 -6

7. 第 4 讲 ppt 上的所有示例程序代码编写一遍（与调试）

作业（标注学号）发邮件到：
2230652597@qq.com