类和对象的关系

- **面向对象的思想（Object Oriented Programming，简称OOP）**
  - ✓ 面向对象的基本思想是，从现实世界中客观存在的事物出发来构造软件系统，并在系统的构造中尽可能运用人类的自然思维方式
  - ✓ 面向对象更加强调运用人类在日常的思维逻辑中经常采用的思维方法与原则

- **面向对象的三个特征**
  - ✓ 封装
  - ✓ 继承
  - ✓ 多态

类和对象的关系

■ 面向对象与面向过程

面向过程

```
void OpenIcebox()
{
   printf("把冰箱门打开\n");
}
void PutElephantI()
{
   printf("把大象放冰箱\n");
}
void CloseIcebox()
{
   printf("把冰箱门关上\n");
}
int main()
{
   OpenIcebox();
   PutElephantIn();
   CloseIcebox();
}
```

putElephantBox.c

面向对象

```
public class Icebox {
  static void putThings(String things){
   System.out.println("把冰箱门打开");
   System.out.println("把"+things+"放进来");
   System.out.println("把冰箱门关上");
   }
}
```

Icebox.java

```
public class Mine {
    public static void main(String args[ ]){
       Icebox.putThings("大象");
    }
}
```

Mine.java

类的定义

- 把相似的对象划归成一个类。
- 在软件设计中，类就是一个模板，它定义了通用于一个特定种类的所有对象的属性（变量）和行为（方法）。

类和对象的关系

类的定义

```
[访问权限控制符] class 类名
{
        类的成员
          …
}
```

- 修饰符一般为public，也可以没有修饰符。
  - 注意类名的命名规范。类名一般大写
  - 类的成员：
    - 成员变量（属性）
    - 成员函数（方法）
  - 通过“.”调用属性和方法

■ 类的定义

Employee.java

```
class Employee {
   String name;
   int age;
   double salary;
    public String showName(){
        System.out.println(name);
        return name;
    }
   public int showAge(){
        System.out.println(age);
        return age;
   }
   public void updateName(String name2){
        name = name2;
   }
   public void getSalary(){
      System.out.println("The salary of this month is 2000");
   }
}
```

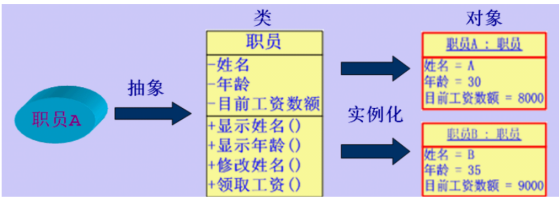
属性

方法

类和对象的关系

■ 类和对象的关系

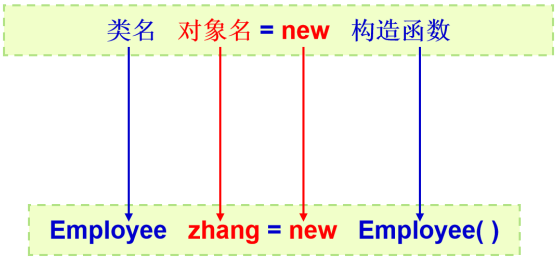
- ✓ 类(class) — 是对某一类事物的描述
- ✓ 对象(object) — 是实际存在的某类事物的个体，也称为实例(instance)
- ✓ 类是创建对象的模板，对象是类的实例。



类和对象的关系

■ 对象的创建

✓ 语法格式



✓ 示例

Employee zhang = new Employee( )

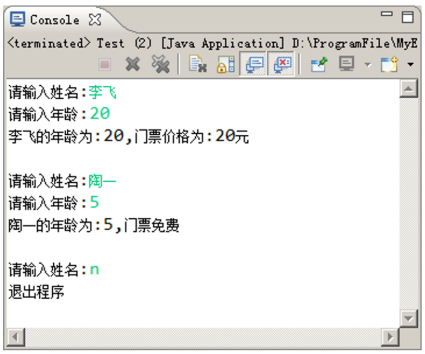
创建和使用对象示例

问题

■ 一个景区根据游人的年龄收取不同价格的门票。请编写游人类，根据年龄段决定能够购买的门票价格并输出

分析

游人类
姓名 年龄
显示姓名及门票价格



创建和使用对象示例

示例

```
import java.util.Scanner;
public class InitialVistor {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        Visitor v = new Visitor();
        System.out.print("请输入姓名：");
        v.name = input.next();
        System.out.print("请输入年龄：");
        v.age = input.nextInt();
        v.show();
    }
}
```

创建对象

给每个属性赋值

调用方法

构造方法

- **构造方法（构造函数/构造器，Constructor）**
  - ✓ 具有与类相同的名称
  - ✓ 不含返回值类型
  - ✓ 不能在方法中用return语句返回一个值
  - ✓ 一般访问权限为public

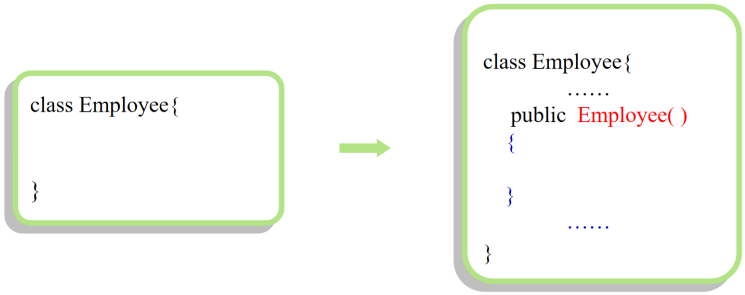
  在一个类中，具有上述特征的方法就是构造方法。

- **构造方法的作用**
  - ✓ 完成对象的创建，即完成对象的实例化
  - ✓ 一般使用构造方法来完成对成员变量的初始化

构造方法

■ 默认的构造方法

- ✓ 在Java中，每个类都至少要有一个构造方法，如果程序员没有在类里定义构造方法，系统会自动为这个类产生一个默认的构造方法



- ➢ 一旦编程者为该类定义了构造方法，系统就不再提供默认的构造方法

方法的重载

■ 方法的重载（overload）

- ✓ 函数的重载就是在同一个类中允许同时存在一个以上同名的函数

• 方法重载的规则

- ✓ 函数名称相同
- ✓ 函数的参数必须不同
  - ➢ 参数个数不同　或　参数类型不同
- ✓ 函数的返回值类型可以相同，也可以不同

方法的重载

■ 方法的重载（overload）

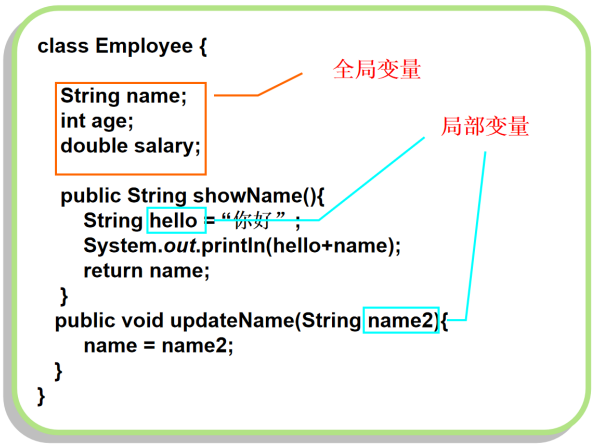
```
class SumFunc {

public int getSum(int end){
      int sum = 0;
      for(int i=1;i<=end;i++){
          sum+=i;
      }
      return sum;
 }

public int getSum(int start,int end){
      int sum = 0;
      for(int i=start;i<end;i++){
          sum+=i;
      }
      return sum;
  }

}
```

变量的作用域

■ 变量的作用域

- ✓ 全局变量
  - 类体中声明的成员变量为全局变量
  - 全局变量在类的整个生命周期中都有效
- ✓ 局部变量
  - 方法体中声明的变量，方法中的参数，或代码块中声明的变量，都是局部变量
  - 局部变量只在方法调用的过程中有效，方法调用结束后失效

■ 变量的作用域



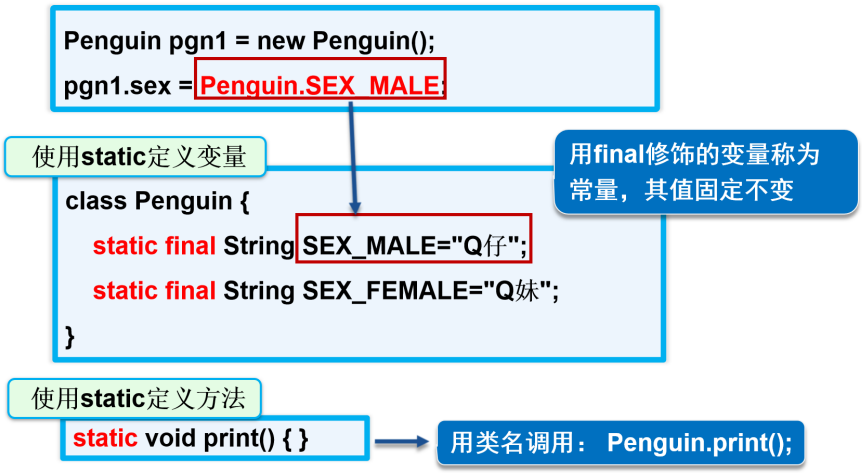
■ this关键字

- ✓ 代表对象自身的引用
  - ➢ 一个引用
  - ➢ 指向调用该方法的当前对象
- ✓ 通常在类的方法定义中使用

- 用this关键字的情况
  - ✓ 方法中的变量与成员变量重名
  - ✓ 在一个构造方法中，调用其它重载的构造方法
  - ✓ 返回当前对象的引用

问题

- 可否通过类名直接访问成员变量?



- static修饰与非static修饰的区别

	static、非private修饰	非static、private修饰
属性	类属性、类变量	实例属性、实例变量
方法	类方法	实例方法
调用方式	类名.属性 类名.方法() 对象.属性 对象.方法()	对象.属性 对象.方法()
归属	类	单个对象

常见错误

代码阅读

- 请指出下面代码的错误

```
class Dog {
    private String name = "旺财"; // 昵称
    private int health = 100;    // 健康值
    private int love = 0;        // 亲密度
    public void play(int n) {
        static int localv=5;
        health = health - n;
        System.out.println(name+" "+localv+" "+health+" "+love);
    }
    public stat
        Dog d=
        d.play(
    }
}
```

在实例方法里不可以定义static变量

```
1  class Dog {
2      private String name = "旺财"; // 昵称
3      private int health = 100;     // 健康值
4      private int love = 0;         // 亲密度
5      public void play(int n) {
6  Illegal modifier for parameter localv; only final is permitted
7          health = health - n;
8          System.out.println(name+" "+localv+" "+health+" "+love);
9      }
```


为什么要使用封装

问题

- 下面代码有什么缺陷?

```
Person d = new Person();
d.age = -1000;
```

属性随意访问，不合理的赋值

- 如何解决上面设计的缺陷?

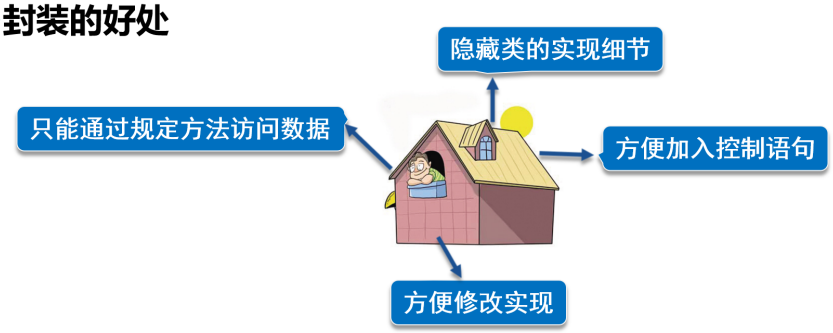
使用封装

什么是封装

- 面向对象三大特征之一 ——封装
  - ◆封装的概念

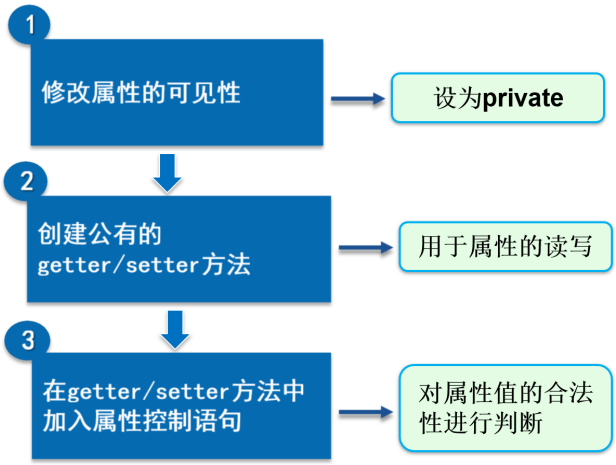
> 封装：将类的某些信息隐藏在类内部，不允许外部程序直接访问，而是通过该类提供的方法来实现对隐藏信息的操作和访问

  - ◆封装的好处



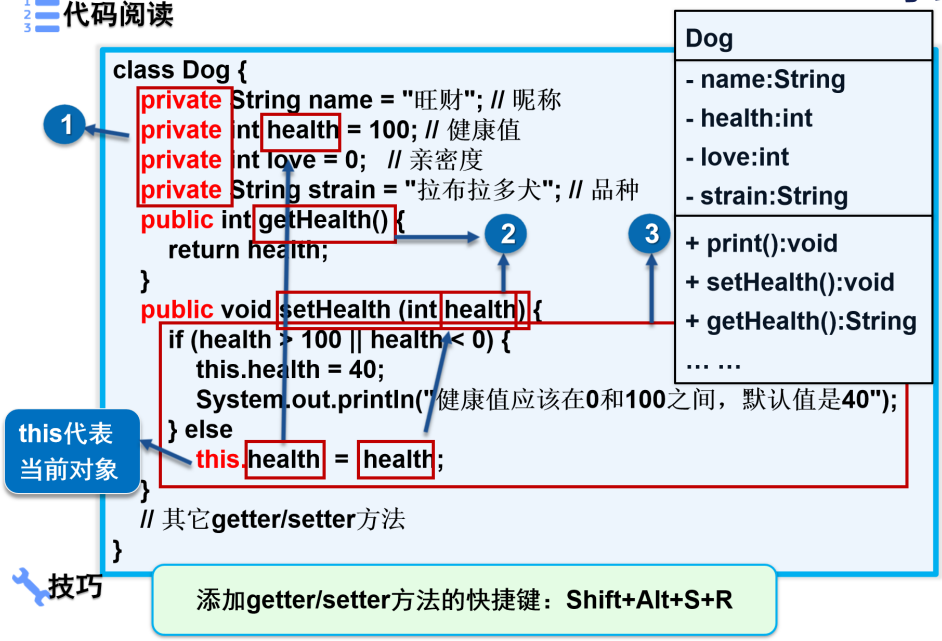
如何使用封装

- 封装的步骤



小结

代码阅读



技巧

添加getter/setter方法的快捷键：Shift+Alt+S+R

类和对象