第 8 章 面向对象编程(中级部分)

8.1 IDE（集成开发环境）-IDEA

8.1. 1IDEA 介绍

1) IDEA 全称 IntelliJ IDEA

2) 在业界被公认为**最好的** **Java开发工具**

3) IDEA 是 JetBrains 公司的产品，总部位于捷克的首都布拉格

4) 除了支持 Java 开发，还支持 HTML ，CSS ，PHP ，MySQL ，Python 等

8.2 IDE（集成开发环境）-Eclipse

8.2. 1Eclipse 介绍

1) Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台。

2) 最初是由 IBM 公司耗资 3000 万美金开发的下一代 IDE 开发环境

3) 2001 年 11 月贡献给开源社区

4) Eclipse 是目前最优秀的 Java 开发 IDE 之一

8.3 IDE（集成开发环境）-IDEA 的使用

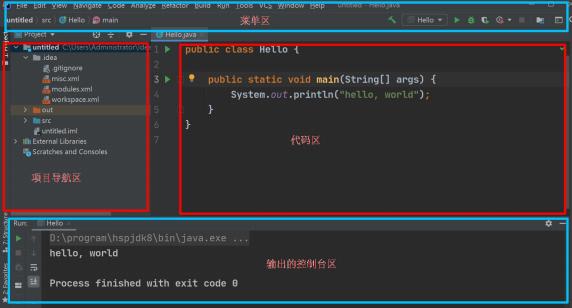
8.3. 1IDEA 的安装

1) 官网: https://www.jetbrains.com/

2) IDEA 下载后，就可以开始安装。

3) 看老师给同学们演示如何安装

4) IDEA 工作界面介绍



**老韩建议**：

1. 使用老师讲课用的版本，学习起来比较顺畅

2. 老师从官网下载的 IDEA 2020.2 会提供给各位

8.3.2IDEA 的基本介绍和使用

使用 IDEA 创建 Java 项目(project) ，看看 IDEA 是如何使用的，IDEA 是以项目的概念，来管理我们的java 源码的

1) 创建一个java 项目 - hello

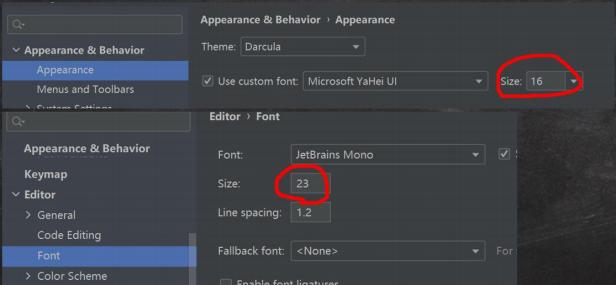
2) 5min 练习一下.

|  |
| --- |
| public class Hello {  public static void main(String[] args) {  System.out.println("hello, idea, 你好北京~~");  }  } |

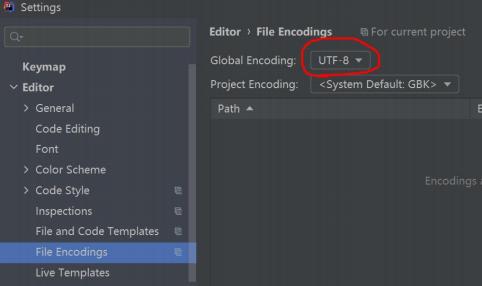
8.3.3IDEA 使用技巧和经验

 设置字体 [如图] 和 颜色主题

菜单 file -> settings



 字符编码设置



8.3.4课堂练习

使用 IDEA 开发一个java 项目 testpro01，创建一个类 MyTools, 编写一个方法，可以完成对 int 数组冒泡排序的功能

学员练习 ， 使用快捷键的开发项目



8.3.5IDEA 常用快捷键

1) 删除当前行, 默认是 ctrl + Y 自己配置 ctrl + d

2) 复制当前行, 自己配置 ctrl + alt + 向下光标

3) 补全代码 alt + /

4) 添加注释和取消注释 ctrl + / 【第一次是添加注释，第二次是取消注释】

5) 导入该行需要的类 先配置 auto import , 然后使用 alt+enter 即可

6) 快速格式化代码 ctrl + alt + L

7) 快速运行程序 自己定义 alt + R

8) 生成构造器等 alt + insert [提高开发效率]

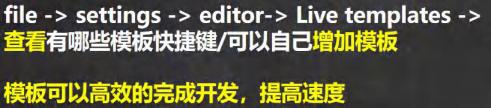
9) 查看一个类的层级关系 ctrl + H [学习继承后，非常有用]

10) 将光标放在一个方法上，输入 ctrl + B , 可以定位到方法 [学继承后，非常有用]

11) 自动的分配变量名 , 通过 在后面假 .var [老师最喜欢的]

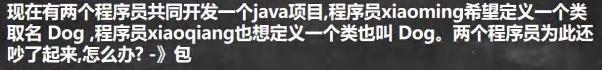
12) 还有很多其它的快捷键...

8.3.6模板/自定义模板



8.4 包

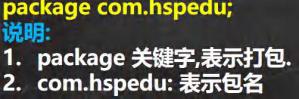
8.4. 1看一个应用场景



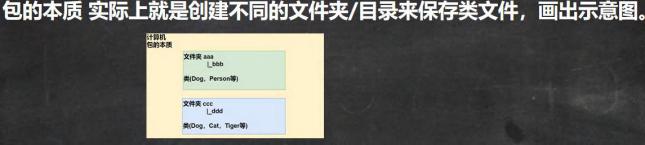
8.4.2包的三大作用



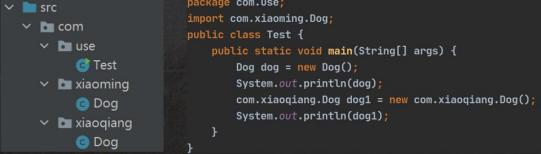
8.4.3包基本语法



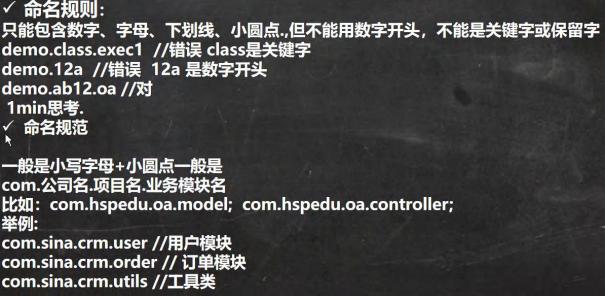
8.4.4包的本质分析(原理)



8.4.5快速入门



8.4.6包的命名



8.4.7常用的包

一个包下,包含很多的类,java 中常用的包有:

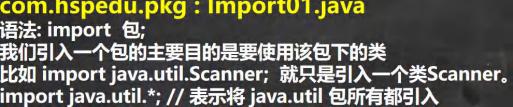
1) java.lang.\* //lang 包是基本包，默认引入，不需要再引入.

2) java.util.\* //util 包，系统提供的工具包, 工具类，使用 Scanner

3) java.net.\* //网络包，网络开发

4) java.awt.\* //是做java 的界面开发，GUI

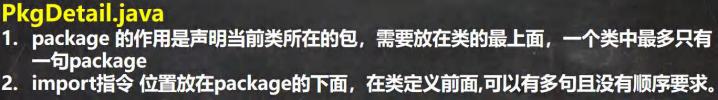
8.4.8如何引入包



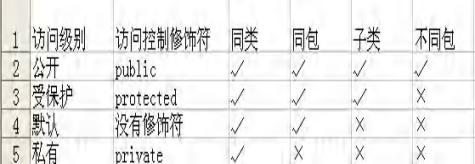
|  |
| --- |
| package com.hspedu.pkg;  import java.util.Arrays;  //注意:  //老韩建议：我们需要使用到哪个类，就导入哪个类即可，不建议使用 \*导入  //import java.util.Scanner; //表示只会引入java.util 包下的 Scanner  //import java.util.\*;//表示将java.util 包下的所有类都引入(导入)  public class Import01 {  public static void main(String[] args) {  //使用系统提供 Arrays 完成 数组排序  int[] arr = {- 1, 20, 2, 13, 3};  //比如对其进行排序  //传统方法是， 自己编写排序(冒泡)  //系统是提供了相关的类，可以方便完成 Arrays  Arrays.sort(arr);  //输出排序结果  for (int i = 0; i < arr.length ; i++) {  System.out.print(arr[i] + "\t");  } |

|  |
| --- |
| }  } |

8.4.9注意事项和使用细节



|  |
| --- |
| //package 的作用是声明当前类所在的包，需要放在类(或者文件)的最上面，  // 一个类中最多只有一句 package  package com.hspedu.pkg;  //import 指令 位置放在 package 的下面，在类定义前面,可以有多句且没有顺序要求  import java.util.Scanner;  import java.util.Arrays;  //...  //类定义  public class PkgDetail {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  int[] arr = {0, - 1, 1};  Arrays.sort(args); |



|  |
| --- |
| }  } |

8.5 访问修饰符

8.5. 1基本介绍

java 提供四种访问控制修饰符号，用于控制方法和属性(成员变量)的访问权限（范围）:

1) 公开级别:用 public 修饰,对外公开

2) 受保护级别:用 protected 修饰,对子类和同一个包中的类公开

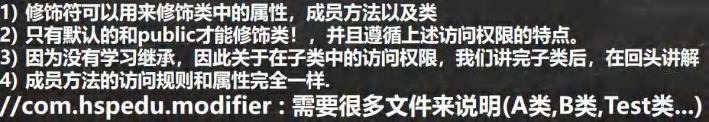
3) 默认级别:没有修饰符号, 向同一个包的类公开.

4) 私有级别:用 private 修饰,只有类本身可以访问,不对外公开.

8.5.24 种访问修饰符的访问范围

背下来

8.5.3使用的注意事项



代码:

|  |
| --- |
| package com.hspedu.modifier; |

|  |
| --- |
| public class A {  //四个属性,分别使用不同的访问修饰符来修饰  public int n1 = 100;  protected int n2 = 200;  int n3 = 300;  private int n4 = 400;  public void m1() {  //在同一类中，可以访问 public protected 默认 private 修饰属性和方法  System.out.println("n1=" + n1 + " n2=" + n2 + " n3=" + n3 + " n4=" + n4);  }  protected void m2() { }  void m3() { }  private void m4() { }  public void hi() {  //在同一类中，可以访问 public protected 默认 private 修饰属性和方法  m1();  m2();  m3();  m4();  }  } |
| package com.hspedu.modifier;  public class B { |

|  |
| --- |
| public void say() {  A a = new A();  //在同一个包下，可以访问 public , protected 和 默认修饰属性或方法,不能访问 private 属性或方法  System.out.println("n1=" + a.n1 + " n2=" + a.n2 + " n3=" + a.n3 );  a.m1();  a.m2();  a.m3();  //a.m4(); 错误的  }  } |
| package com.hspedu.modifier;  public class Test {  public static void main(String[] args) {  A a = new A ();  a.m1();  B b = new B();  b.say();  }  }  //只有 默认和 public 可以修饰类  class Tiger{ } |

8.6 面向对象编程三大特征

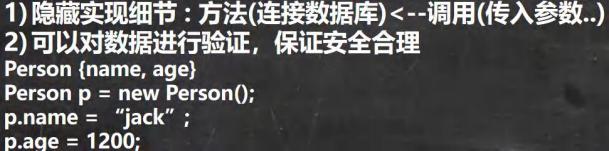
8.6. 1基本介绍

面向对象编程有三大特征：封装、继承和多态。

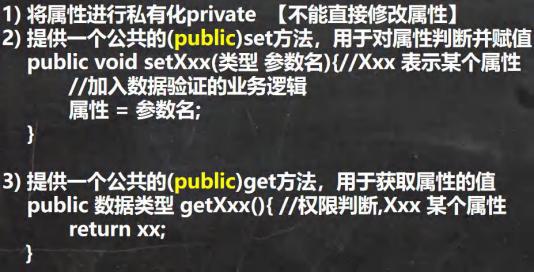
8.6.2封装介绍



8.6.3封装的理解和好处

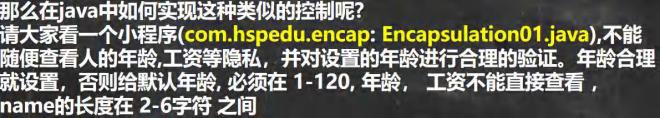


8.6.4封装的实现步骤 (三步)



8.7 快速入门案例

 看一个案例



|  |
| --- |
| package com.hspedu.encap;  public class Encapsulation01 {  public static void main(String[] args) {  //如果要使用快捷键 alt+r, 需要先配置主类  //第一次，我们使用鼠标点击形式运算程序，后面就可以用  Person person = new Person();  person.setName("韩顺平");  person.setAge(30);  person.setSalary(30000);  System.out.println(person.info());  System.out.println(person.getSalary());  //如果我们自己使用构造器指定属性  Person smith = new Person("smith", 80, 50000);  System.out.println("====smith 的信息======");  System.out.println(smith.info()); |

|  |
| --- |
| }  }  /\*  那么在java 中如何实现这种类似的控制呢?  请大家看一个小程序(com.hspedu.encap: Encapsulation01.java),  不能随便查看人的年龄,工资等隐私，并对设置的年龄进行合理的验证。年龄合理就设置，否则给默认  年龄, 必须在 1- 120, 年龄， 工资不能直接查看 ， name 的长度在 2-6 字符 之间  \*/  class Person {  public String name; //名字公开  private int age; //age 私有化  private double salary; //..  public void say(int n,String name) {  }  //构造器 alt+insert  public Person() {  }  //有三个属性的构造器  public Person(String name, int age, double salary) {  // this.name = name;  // this.age = age;  // this.salary = salary; |

|  |
| --- |
| //我们可以将 set 方法写在构造器中，这样仍然可以验证  setName(name);  setAge(age);  setSalary(salary);  }  //自己写 setXxx 和 getXxx 太慢，我们使用快捷键  //然后根据要求来完善我们的代码.  public String getName() {  return name;  }  public void setName(String name) {  //加入对数据的校验,相当于增加了业务逻辑  if(name.length() >= 2 && name.length() <=6 ) {  this.name = name;  }else {  System.out.println("名字的长度不对，需要(2-6)个字符，默认名字");  this.name = "无名人";  }  }  public int getAge() {  return age;  }  public void setAge(int age) { |

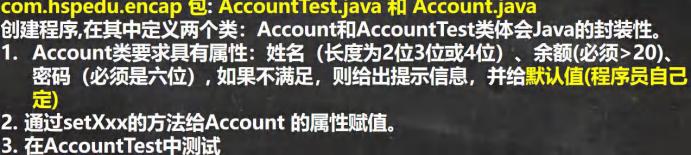
|  |
| --- |
| //判断  if(age >= 1 && age <= 120) {//如果是合理范围  this.age = age;  } else {  System.out.println("你设置年龄不对，需要在 (1- 120), 给默认年龄 18 ");  this.age = 18;//给一个默认年龄  }  }  public double getSalary() {  //可以这里增加对当前对象的权限判断  return salary;  }  public void setSalary(double salary) {  this.salary = salary;  }  //写一个方法，返回属性信息  public String info() {  return "信息为 name=" + name + " age=" + age + " 薪水=" + salary;  }  } |

8.7. 1将构造器和 setXxx 结合

 看一个案例

|  |
| --- |
| //有三个属性的构造器  public Person(String name, int age, double salary) {  // this.name = name;  // this.age = age;  // this.salary = salary;  //我们可以将 set 方法写在构造器中，这样仍然可以验证  setName(name);  setAge(age);  setSalary(salary);  } |

8.7.2课堂练习



|  |
| --- |
| package com.hspedu.encap;  /\*\*  \* 创建程序,在其中定义两个类：Account 和 AccountTest 类体会 Java 的封装性。  \* Account 类要求具有属性：姓名（长度为 2 位 3 位或 4 位）、余额(必须>20)、  \* 密码（必须是六位）, 如果不满足，则给出提示信息，并给默认值(程序员自己定)  \* 通过 setXxx 的方法给 Account 的属性赋值。  \* 在 AccountTest 中测试 |

|  |
| --- |
| \*/  public class Account {  //为了封装，将 3 个属性设置为 private  private String name;  private double balance;  private String pwd;  //提供两个构造器  public Account() {  }  public Account(String name, double balance, String pwd) {  this.setName(name);  this.setBalance(balance);  this.setPwd(pwd);  }  public String getName() {  return name;  }  //姓名（长度为 2 位 3 位或 4 位）  public void setName(String name) {  if (name.length() >= 2 && name.length() <= 4) {  this.name = name;  } else { |

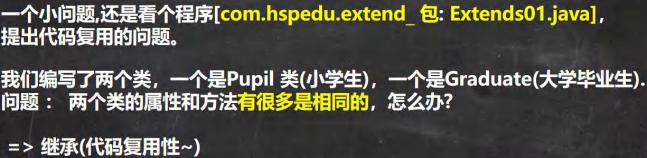
|  |
| --- |
| System.out.println("姓名要求（长度为 2 位 3 位或 4 位），默认值 无名");  this.name = "无名";  }  }  public double getBalance() {  return balance;  }  //余额(必须>20)  public void setBalance(double balance) {  if (balance > 20) {  this.balance = balance;  } else {  System.out.println("余额(必须>20) 默认为 0");  }  }  public String getPwd() {  return pwd;  }  //密码（必须是六位）  public void setPwd(String pwd) {  if (pwd.length() == 6) {  this.pwd = pwd; |

|  |
| --- |
| } else {  System.out.println("密码（必须是六位）默认密码为 000000");  this.pwd = "000000";  }  }  //显示账号信息  public void showInfo() {  //可以增加权限的校验  System.out.println("账号信息 name=" + name + " 余额=" + balance + " 密码" + pwd);  // if() {  // System.out.println("账号信息 name=" + name + " 余额=" + balance + " 密码");  // }else {  // System.out.println("你无权查看...");  // }  }  } |
| package com.hspedu.encap;  public class TestAccount {  public static void main(String[] args) {  //创建 Account  Account account = new Account();  account.setName("jack");  account.setBalance(60);  account.setPwd(" 123456"); |

|  |
| --- |
| account.showInfo();  }  } |

8.8 面向对象编程-继承

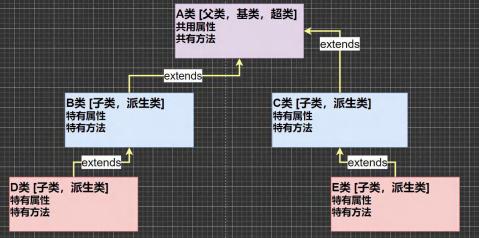
8.8. 1为什么需要继承



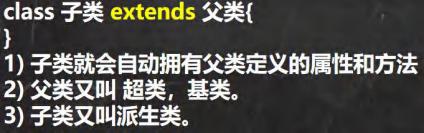
8.8.2继承基本介绍和示意图

继承可以解决代码复用,让我们的编程更加靠近人类思维. 当多个类存在相同的属性(变量)和方法时,可以从这些类中 抽象出父类,在父类中定义这些相同的属性和方法，所有的子类不需要重新定义这些属性和方法，只需要通过 extends 来

声明继承父类即可。画出继承的示意图



8.8.3继承的基本语法



8.8.4快速入门案例

我们对 Extends01.java 改进，使用继承的方法,请大家注意体会使用继承的好处

|  |
| --- |
| package com.hspedu.extend\_.improve\_;  import com.hspedu.extend\_.Graduate;  import com.hspedu.extend\_.Pupil;  public class Extends01 {  public static void main(String[] args) {  com.hspedu.extend\_.Pupil pupil = new Pupil();  pupil.name = "银角大王~";  pupil.age = 11;  pupil.testing();  pupil.setScore(50);  pupil.showInfo();  System.out.println("=======");  com.hspedu.extend\_.Graduate graduate = new Graduate();  graduate.name = "金角大王~"; |

|  |
| --- |
| graduate.age = 23;  graduate.testing();  graduate.setScore(80);  graduate.showInfo();  }  }  package com.hspedu.extend\_.improve\_;  //父类,是 Pupil 和 Graduate 的父类  public class Student {  //共有属性  public String name;  public int age;  private double score;//成绩  //共有的方法  public void setScore(double score) {  this.score = score;  }  public void showInfo() {  System.out.println("学生名 " + name + " 年龄 " + age + " 成绩 " + score);  }  }  package com.hspedu.extend\_.improve\_;  //让 Pupil 继承 Student 类 |

|  |
| --- |
| public class Pupil extends Student {  public void testing() {  System.out.println("小学生 " + name + " 正在考小学数学..");  }  }  package com.hspedu.extend\_.improve\_;  public class Graduate extends Student {  public void testing() {//和 Pupil 不一样  System.out.println("大学生 " + name + " 正在考大学数学..");  }  } |

8.8.5继承给编程带来的便利

1) 代码的复用性提高了

2) 代码的扩展性和维护性提高了

8.8.6继承的深入讨论/细节问题

1) 子类继承了所有的属性和方法，非私有的属性和方法可以在子类直接访问, 但是私有属性和方法不能在子类直接访

问，要通过父类提供公共的方法去访问

2) 子类必须调用父类的构造器， 完成父类的初始化

3) 当创建子类对象时，不管使用子类的哪个构造器，默认情况下总会去调用父类的无参构造器，如果父类没有提供无

参构造器，则必须在子类的构造器中用 super 去指定使用父类的哪个构造器完成对父类的初始化工作，否则，编译

不会通过(怎么理解。) [举例说明]

4) 如果希望指定去调用父类的某个构造器，则显式的调用一下 : super(参数列表)

5) super 在使用时，必须放在构造器第一行(super 只能在构造器中使用)

6) super() 和 this() 都只能放在构造器第一行，因此这两个方法不能共存在一个构造器

7) java 所有类都是 Object 类的子类, Object 是所有类的基类.

8) 父类构造器的调用不限于直接父类！将一直往上追溯直到 Object 类(顶级父类)

9) 子类最多只能继承一个父类(指直接继承) ，即java 中是单继承机制。

思考：如何让 A 类继承 B 类和 C 类？ 【A 继承 B ， B 继承 C】

10) 不能滥用继承，子类和父类之间必须满足 is-a 的逻辑关系

**代码**

|  |
| --- |
| package com.hspedu.extend\_;  public class ExtendsDetail {  public static void main(String[] args) {  // System.out.println("===第 1 个对象====");  // Sub sub = new Sub(); //创建了子类对象 sub  // System.out.println("===第 2 个对象====");  // Sub sub2 = new Sub("jack"); //创建了子类对象 sub2  System.out.println("===第 3 对象====");  Sub sub3 = new Sub("king", 10); //创建了子类对象 sub2  //sub.sayOk();  }  }  package com.hspedu.extend\_; |

|  |
| --- |
| public class Base extends TopBase { //父类  //4 个属性  public int n1 = 100;  protected int n2 = 200;  int n3 = 300;  private int n4 = 400;  public Base() { //无参构造器  System.out.println("父类 Base()构造器被调用....");  }  public Base(String name, int age) {//有参构造器  //默认 super()  System.out.println("父类 Base(String name, int age)构造器被调用....");  }  public Base(String name) {//有参构造器  System.out.println("父类 Base(String name)构造器被调用....");  }  //父类提供一个 public 的方法,返回了 n4  public int getN4() {  return n4;  }  public void test100() {  System.out.println("test100");  }  protected void test200() {  System.out.println("test200"); |

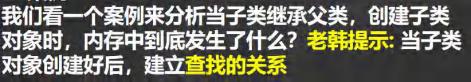
|  |
| --- |
| }  void test300() {  System.out.println("test300");  }  private void test400() {  System.out.println("test400");  }  //call  public void callTest400() {  test400();  }  }  package com.hspedu.extend\_;  import java.util.Arrays;  //输入ctrl + H 可以看到类的继承关系  public class Sub extends Base { //子类  public Sub(String name, int age) {  //1. 老师要调用父类的无参构造器, 如下或者 什么都不写,默认就是调用 super()  //super();//父类的无参构造器  //2. 老师要调用父类的 Base(String name) 构造器  //super("hsp");  //3. 老师要调用父类的 Base(String name, int age) 构造器 |

|  |
| --- |
| super("king", 20);  //细节： super 在使用时，必须放在构造器第一行  //细节: super() 和 this() 都只能放在构造器第一行，因此这两个方法不能共存在一个构造器  //this() 不能再使用了  System.out.println("子类 Sub(String name, int age)构造器被调用....");  }  public Sub() {//无参构造器  //super(); //默认调用父类的无参构造器  super("smith", 10);  System.out.println("子类 Sub()构造器被调用....");  }  //当创建子类对象时，不管使用子类的哪个构造器，默认情况下总会去调用父类的无参构造器  public Sub(String name) {  super("tom", 30);  //do nothing...  System.out.println("子类 Sub(String name)构造器被调用....");  }  public void sayOk() {//子类方法  //非私有的属性和方法可以在子类直接访问  //但是私有属性和方法不能在子类直接访问  System.out.println(n1 + " " + n2 + " " + n3); |

|  |
| --- |
| test100();  test200();  test300();  //test400();错误  //要通过父类提供公共的方法去访问  System.out.println("n4=" + getN4());  callTest400();//  }  }  package com.hspedu.extend\_;  public class TopBase { //父类是 Object  public TopBase() {  //super(); Object 的无参构造器  System.out.println("构造器 TopBase() 被调用...");//1  }  } |

8.8.7继承的本质分析(重要)

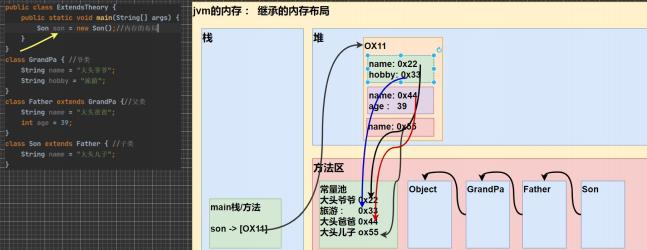
 案例



|  |
| --- |
| package com.hspedu.extend\_;  /\*\*  \* 讲解继承的本质  \*/  public class ExtendsTheory {  public static void main(String[] args) {  Son son = new Son();//内存的布局  //?-> 这时请大家注意，要按照查找关系来返回信息  //(1) 首先看子类是否有该属性  //(2) 如果子类有这个属性，并且可以访问，则返回信息  //(3) 如果子类没有这个属性，就看父类有没有这个属性(如果父类有该属性，并且可以访问，就返回信息..)  //(4) 如果父类没有就按照(3)的规则，继续找上级父类，直到 Object...  System.out.println(son.name);//返回就是大头儿子  //System.out.println(son.age);//返回的就是 39  //System.out.println(son.getAge());//返回的就是 39  System.out.println(son.hobby);//返回的就是旅游  }  }  class GrandPa { //爷类  String name = "大头爷爷"; |

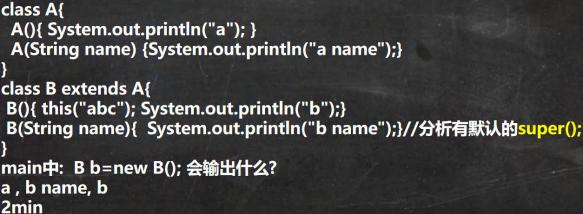
|  |
| --- |
| String hobby = "旅游";  }  class Father extends GrandPa {//父类  String name = "大头爸爸";  private int age = 39;  public int getAge() {  return age;  }  }  class Son extends Father { //子类  String name = "大头儿子";  } |

 子类创建的内存布局



8.8.8课堂练习

1) 案例 1 ExtendsExercise01.java



2) 案例 2 ExtendsExercise02.java

|  |
| --- |
| package com.hspedu.extend\_.exercise;  public class ExtendsExercise02 {  public static void main(String[] args) {  C c = new C();  }  } |

|  |
| --- |
| class A {//A 类  public A() {  System.out.println("我是 A 类");  }  }  class B extends A { //B 类,继承 A 类 //main 方法中： C c =new C(); 输出么内容? 3min  public B() {  System.out.println("我是 B 类的无参构造");  }  public B(String name) {  System.out.println(name + "我是 B 类的有参构造");  }  }  class C extends B { //C 类，继承 B 类  public C() {  this("hello");  System.out.println("我是 c 类的无参构造");  }  public C(String name) {  super("hahah"); |

|  |
| --- |
| System.out.println("我是 c 类的有参构造");  }  } |

3) 案例 3 ExtendsExercise03.java

编写 Computer 类，包含 CPU 、内存、硬盘等属性，getDetails 方法用于返回 Computer 的详细信息

编写 PC 子类，继承 Computer 类，添加特有属性【品牌 brand】

编写 NotePad 子类，继承 Computer 类，添加特有属性【color】

编写 Test 类，在 main 方法中创建 PC 和 NotePad 对象，分别给对象中特有的属性赋值，以及从 Computer 类继承的

属性赋值，并使用方法并打印输出信息

|  |
| --- |
| package com.hspedu.extend\_.exercise;  //编写 Computer 类，包含 CPU 、内存、硬盘等属性，getDetails 方法用于返回 Computer 的详细信息  public class Computer {  private String cpu;  private int memory;  private int disk;  public Computer(String cpu, int memory, int disk) {  this.cpu = cpu;  this.memory = memory;  this.disk = disk;  }  //返回 Computer 信息  public String getDetails() {  return "cpu=" + cpu + " memory=" + memory + " disk=" + disk;  } |

|  |
| --- |
| public String getCpu() {  return cpu;  }  public void setCpu(String cpu) {  this.cpu = cpu;  }  public int getMemory() {  return memory;  }  public void setMemory(int memory) {  this.memory = memory;  }  public int getDisk() {  return disk;  }  public void setDisk(int disk) {  this.disk = disk;  }  } |

|  |
| --- |
| package com.hspedu.extend\_.exercise;  //编写 PC 子类，继承 Computer 类，添加特有属性【品牌 brand】  public class PC extends Computer{  private String brand;  //这里 IDEA 根据继承的规则， 自动把构造器的调用写好  //这里也体现： 继承设计的基本思想，父类的构造器完成父类属性初始化  //子类的构造器完成子类属性初始化  public PC(String cpu, int memory, int disk, String brand) {  super(cpu, memory, disk);  this.brand = brand;  }  public String getBrand() {  return brand;  }  public void setBrand(String brand) {  this.brand = brand;  }  public void printInfo() {  System.out.println("PC 信息=");  // System.out.println(getCpu() + getMemory() + getDisk());  //调用父类的 getDetails 方法，得到相关属性信息..  System.out.println(getDetails() + " brand=" + brand);  } |

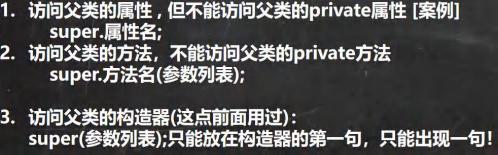
|  |
| --- |
| } |
| package com.hspedu.extend\_.exercise;  public class ExtendsExercise03 {  public static void main(String[] args) {  PC pc = new PC("intel", 16, 500, "IBM");  pc.printInfo();  }  }  /\*  编写 Computer 类，包含 CPU 、内存、硬盘等属性，getDetails 方法用于返回 Computer 的详细信息  编写 PC 子类，继承 Computer 类，添加特有属性【品牌 brand】  编写 NotePad 子类，继承 Computer 类，添加特有属性【color】//同学们自己写。  编写 Test类，在 main 方法中创建 PC 和 NotePad 对象，分别给对象中特有的属性赋值，  以及从 Computer类继承的属性赋值，并使用方法并打印输出信息  \*/ |

8.9 super 关键字

8.9. 1基本介绍

super 代表父类的引用，用于**访问父类的属性、方法、构造**器

8.9.2基本语法



代码

|  |
| --- |
| package com.hspedu.super\_;  public class A extends Base{  //4 个属性  //public int n1 = 100;  protected int n2 = 200;  int n3 = 300;  private int n4 = 400;  public A() {}  public A(String name) {}  public A(String name, int age) {}  // public void cal() {  // System.out.println("A 类的 cal() 方法...");  // }  public void test100() {  } |

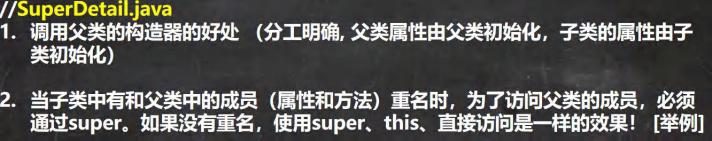
|  |
| --- |
| protected void test200() {  }  void test300() {  }  private void test400() {  }  } |
| package com.hspedu.super\_;  public class B extends A {  public int n1 = 888;  //编写测试方法  public void test() {  //super 的访问不限于直接父类，如果爷爷类和本类中有同名的成员，也可以使用 super 去访问爷爷类的成员；  // 如果多个基类(上级类)中都有同名的成员，使用 super 访问遵循就近原则。A->B->C  System.out.println("super.n1=" + super.n1);  super.cal();  } |

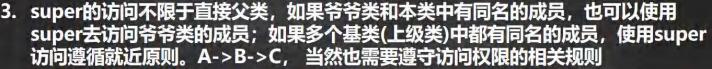
|  |
| --- |
| //访问父类的属性 , 但不能访问父类的 private 属性 [案例]super.属性名  public void hi() {  System.out.println(super.n1 + " " + super.n2 + " " + super.n3 );  }  public void cal() {  System.out.println("B 类的 cal() 方法...");  }  public void sum() {  System.out.println("B 类的 sum()");  //希望调用父类-A 的 cal 方法  //这时，因为子类 B 没有cal 方法，因此我可以使用下面三种方式  //找 cal 方法时(cal() 和 this.cal()) ，顺序是:  // (1)先找本类，如果有，则调用  // (2)如果没有，则找父类(如果有，并可以调用，则调用)  // (3)如果父类没有，则继续找父类的父类,整个规则，就是一样的,直到 Object 类  // 提示：如果查找方法的过程中，找到了，但是不能访问， 则报错, cannot access  // 如果查找方法的过程中，没有找到，则提示方法不存在  //cal();  this.cal(); //等价 cal  //找 cal 方法(super.call()) 的顺序是直接查找父类，其他的规则一样  //super.cal();  //演示访问属性的规则 |

|  |
| --- |
| //n1 和 this.n1 查找的规则是  //(1) 先找本类，如果有，则调用  //(2) 如果没有，则找父类(如果有，并可以调用，则调用)  //(3) 如果父类没有，则继续找父类的父类,整个规则，就是一样的,直到 Object 类  // 提示：如果查找属性的过程中，找到了，但是不能访问， 则报错, cannot access  // 如果查找属性的过程中，没有找到，则提示属性不存在  System.out.println(n1);  System.out.println(this.n1);  //找n1 (super.n1) 的顺序是直接查找父类属性，其他的规则一样  System.out.println(super.n1);  }  //访问父类的方法，不能访问父类的 private 方法 super.方法名(参数列表);  public void ok() {  super.test100();  super.test200();  super.test300();  //super.test400();//不能访问父类 private 方法  }  //访问父类的构造器(这点前面用过)：super(参数列表);只能放在构造器的第一句，只能出现一句！  public B() {  //super();  //super("jack", 10);  super("jack");  } |

|  |
| --- |
| } |
| package com.hspedu.super\_;  public class Super01 {  public static void main(String[] args) {  B b = new B();//子类对象  //b.sum();  b.test();  }  } |
| package com.hspedu.super\_;  public class Base { //父类是 Object  public int n1 = 999;  public int age = 111;  public void cal() {  System.out.println("Base 类的 cal() 方法...");  }  public void eat() {  System.out.println("Base 类的 eat().....");  }  } |

8.9.3super 给编程带来的便利/细节





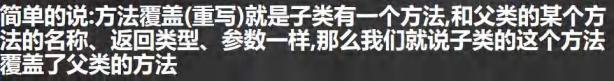
代码, 看前面的案例即可：

8.9.4super 和 this 的比较



8. 10 方法重写/覆盖(override)

8.10.1 基本介绍



8.10.2 快速入门

代码

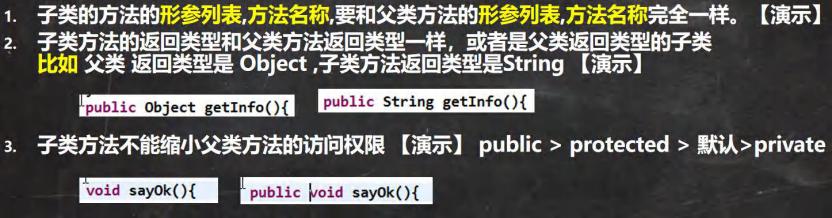
|  |
| --- |
| package com.hspedu.override\_;  public class Animal {  public void cry() {  System.out.println("动物叫唤..");  }  public Object m1() {  return null;  }  public String m2() {  return null;  }  public AAA m3() {  return null;  }  protected void eat() {  } |

|  |
| --- |
| } |
| package com.hspedu.override\_;  public class Dog extends Animal{  //老韩解读  //1. 因为 Dog 是 Animal 子类  //2. Dog 的 cry 方法和 Animal 的 cry 定义形式一样(名称、返回类型、参数)  //3. 这时我们就说 Dog 的 cry 方法，重写了 Animal 的 cry 方法  public void cry() {  System.out.println("小狗汪汪叫..");  }  //细节: 子类方法的返回类型和父类方法返回类型一样，  // 或者是父类返回类型的子类  比如 父类 返回类型是 Object ,  // 子类方法返回类型是 String  public String m1() {  return null;  }  //这里 Object 不是 String 的子类，因此编译错误  // public Object m2() {  // return null;  // }  // public BBB m3() { |

|  |
| --- |
| // return null;  // }  //细节: 子类方法不能缩小父类方法的访问权限 【演示】  //public > protected > 默认>private  public void eat() {  }  }  class AAA {  }  class BBB extends AAA {  } |
| package com.hspedu.override\_;  public class Override01 {  public static void main(String[] args) {  //演示方法重写的情况  Dog dog = new Dog();  dog.cry();//ctrl+b  }  } |

8.10.3 注意事项和使用细节

方法重写也叫方法覆盖，需要满足下面的条件



8.10.4 课堂练习

 题 1

请对方法的重写和重载做一个比较



 题 2

1) 编写一个 Person 类，包括属性/private（name 、age），构造器、方法 say(返回自我介绍的字符串）。

2) 编写一个 Student 类，继承 Person 类，增加 id、score 属性/private，以及构造器，定义 say 方法(返回自我介绍的信息)。

3) 在 main 中,分别创建 Person 和 Student 对象，调用 say 方法输出自我介绍

代码

|  |
| --- |
| package com.hspedu.override\_;  //编写一个 Person 类，包括属性/private（name 、age），构造器、方法 say(返回自我介绍的字符串）  public class Person {  private String name;  private int age;  public Person(String name, int age) {  this.name = name;  this.age = age;  }  public String say() {  return "name=" + name + " age=" + age;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  public void setAge(int age) { |

|  |
| --- |
| this.age = age;  }  } |
| package com.hspedu.override\_;  //编写一个 Student 类，继承 Person 类，增加 id 、score 属性/private ，以及构造器，定义 say 方法(返回自我介绍的信息)。  public class Student extends Person{  private int id;  private double score;  public Student(String name, int age, int id, double score) {  super(name, age);//这里会调用父类构造器  this.id = id;  this.score = score;  }  //say  public String say() { //这里体现 super 的一个好处，代码复用.  return super.say() + " id=" + id + " score=" + score;  }  public int getId() {  return id;  }  public void setId(int id) { |

|  |
| --- |
| this.id = id;  }  public double getScore() {  return score;  }  public void setScore(double score) {  this.score = score;  }  } |
| package com.hspedu.override\_;  public class OverrideExercise {  public static void main(String[] args) {  //在 main 中,分别创建 Person 和 Student 对象，调用 say 方法输出自我介绍  Person jack = new Person("jack", 10);  System.out.println(jack.say());  Student smith = new Student("smith", 20, 123456, 99.8);  System.out.println(smith.say());  }  } |

8. 11 面向对象编程-多态

8.11.1 先看一个问题



 使用传统的方法来解决（private 属性）

 传统的方法带来的问题是什么? 如何解决?

问题是： 代码的复用性不高，而且不利于代码维护

解决方案： 引出我们要讲解的多态

代码:

|  |
| --- |
| package com.hspedu.poly\_;  public class Animal {  private String name;  public Animal(String name) {  this.name = name;  } |

|  |
| --- |
| public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  }  package com.hspedu.poly\_;  public class Bone extends Food {  public Bone(String name) {  super(name);  }  }  package com.hspedu.poly\_;  public class Cat extends Animal {  public Cat(String name) {  super(name);  }  }  package com.hspedu.poly\_;  public class Dog extends Animal {  public Dog(String name) { |

|  |
| --- |
| super(name);  }  }  package com.hspedu.poly\_;  public class Fish extends Food {  public Fish(String name) {  super(name);  }  }  package com.hspedu.poly\_;  public class Food {  private String name;  public Food(String name) {  this.name = name;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  } |

|  |
| --- |
| }  package com.hspedu.poly\_;  public class Master {  private String name;  public Master(String name) {  this.name = name;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  //使用多态机制，可以统一的管理主人喂食的问题  //animal 编译类型是 Animal,可以指向(接收) Animal 子类的对象  //food 编译类型是 Food ,可以指向(接收) Food 子类的对象  public void feed(Animal animal, Food food) {  System.out.println("主人 " + name + " 给 " + animal.getName() + " 吃 " + food.getName());  }  //主人给小狗 喂食 骨头 |

|  |
| --- |
| // public void feed(Dog dog, Bone bone) {  // System.out.println("主人 " + name + " 给 " + dog.getName() + " 吃 " + bone.getName());  // }  // //主人给 小猫喂 黄花鱼  // public void feed(Cat cat, Fish fish) {  // System.out.println("主人 " + name + " 给 " + cat.getName() + " 吃 " + fish.getName());  // }  //如果动物很多，食物很多  //===> feed 方法很多，不利于管理和维护  //Pig --> Rice  //Tiger ---> meat ...  //...  }  package com.hspedu.poly\_;  public class Pig extends Animal {  public Pig(String name) {  super(name);  }  }  package com.hspedu.poly\_;  public class Rice extends Food {  public Rice(String name) { |

|  |
| --- |
| super(name);  }  } |

8.11.2 多[多种]态[状态]基本介绍

方法或对象具有多种形态。是面向对象的第三大特征，多态是建立在封装和继承基础之上的。

8.11.3 多态的具体体现

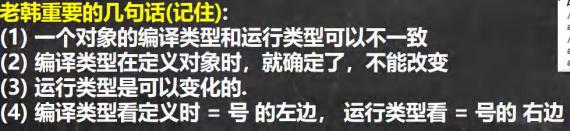
1) 方法的多态 PloyMethod.java

重写和重载就体现多态 [案例说明：]

|  |
| --- |
| package com.hspedu.poly\_;  public class PloyMethod {  public static void main(String[] args) {  //方法重载体现多态  A a = new A();  //这里我们传入不同的参数，就会调用不同 sum 方法，就体现多态  System.out.println(a.sum(10, 20));  System.out.println(a.sum(10, 20, 30));  //方法重写体现多态  B b = new B();  a.say();  b.say(); |

|  |
| --- |
| }  }  class B { //父类  public void say() {  System.out.println("B say() 方法被调用...");  }  }  class A extends B {//子类  public int sum(int n1, int n2){//和下面 sum 构成重载  return n1 + n2;  }  public int sum(int n1, int n2, int n3){  return n1 + n2 + n3;  }  public void say() {  System.out.println("A say() 方法被调用...");  }  } |

2) 对象的多态 **(核心，困难，重点**)





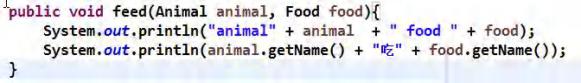
代码：

|  |
| --- |
| package com.hspedu.poly\_.objectpoly\_;  public class Animal {  public void cry() {  System.out.println("Animal cry() 动物在叫....");  }  }  package com.hspedu.poly\_.objectpoly\_;  public class Cat extends Animal {  public void cry() {  System.out.println("Cat cry() 小猫喵喵叫...");  }  }  package com.hspedu.poly\_.objectpoly\_;  public class Dog extends Animal { |

|  |
| --- |
| public void cry() {  System.out.println("Dog cry() 小狗汪汪叫...");  }  }  package com.hspedu.poly\_.objectpoly\_;  public class PolyObject {  public static void main(String[] args) {  //体验对象多态特点  //animal 编译类型就是 Animal , 运行类型 Dog  Animal animal = new Dog();  //因为运行时 , 执行到改行时，animal 运行类型是 Dog,所以 cry 就是 Dog 的 cry  animal.cry(); //小狗汪汪叫  //animal 编译类型 Animal,运行类型就是 Cat  animal = new Cat();  animal.cry(); //小猫喵喵叫  }  } |

8.11.4 多态快速入门案例

使用多态的机制来解决主人喂食物的问题，走代码。 Poly01.java

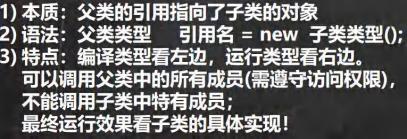


8.11.5 多态注意事项和细节讨论

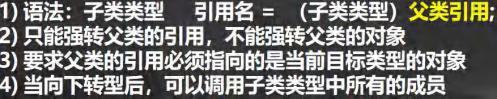
com.hspedu.poly\_.detail\_ 包 : PolyDetail.java

 多态的**前提是**：两个对象(类)存在继承关系

 多态的向上转型



 多态向下转型



|  |
| --- |
| package com.hspedu.poly\_.detail\_;  public class Animal {  String name = "动物";  int age = 10;  public void sleep(){  System.out.println("睡");  }  public void run(){  System.out.println("跑"); |

|  |
| --- |
| }  public void eat(){  System.out.println("吃");  }  public void show(){  System.out.println("hello,你好");  }  }  package com.hspedu.poly\_.detail\_;  public class Cat extends Animal {  public void eat(){//方法重写  System.out.println("猫吃鱼");  }  public void catchMouse(){//Cat 特有方法  System.out.println("猫抓老鼠");  }  }  package com.hspedu.poly\_.detail\_;  public class Dog extends Animal {//Dog 是 Animal 的子类  }  package com.hspedu.poly\_.detail\_;  public class PolyDetail { |

|  |
| --- |
| public static void main(String[] args) {  //向上转型: 父类的引用指向了子类的对象  //语法：父类类型引用名 = new 子类类型();  Animal animal = new Cat();  Object obj = new Cat();//可以吗? 可以 Object 也是 Cat 的父类  //向上转型调用方法的规则如下:  //(1)可以调用父类中的所有成员(需遵守访问权限)  //(2)但是不能调用子类的特有的成员  //(#)因为在编译阶段，能调用哪些成员,是由编译类型来决定的  //animal.catchMouse();错误  //(4)最终运行效果看子类(运行类型)的具体实现, 即调用方法时，按照从子类(运行类型)开始查找方法  // ，然后调用，规则我前面我们讲的方法调用规则一致。  animal.eat();//猫吃鱼..  animal.run();//跑  animal.show();//hello,你好  animal.sleep();//睡  //老师希望，可以调用 Cat 的 catchMouse 方法  //多态的向下转型  //(1)语法：子类类型 引用名 =（子类类型）父类引用;  //问一个问题? cat 的编译类型 Cat,运行类型是 Cat  Cat cat = (Cat) animal;  cat.catchMouse();//猫抓老鼠  //(2)要求父类的引用必须指向的是当前目标类型的对象 |

|  |
| --- |
| Dog dog = (Dog) animal; //可以吗？  System.out.println("ok~~");  }  } |

 属性没有重写之说！属性的值看编译类型 PolyDetail02.java

|  |
| --- |
| package com.hspedu.poly\_.detail\_;  public class PolyDetail02 {  public static void main(String[] args) {  //属性没有重写之说！属性的值看编译类型  Base base = new Sub();//向上转型  System.out.println(base.count);// ？ 看编译类型 10  Sub sub = new Sub();  System.out.println(sub.count);//? 20  }  }  class Base { //父类  int count = 10;//属性  }  class Sub extends Base {//子类  int count = 20;//属性  } |

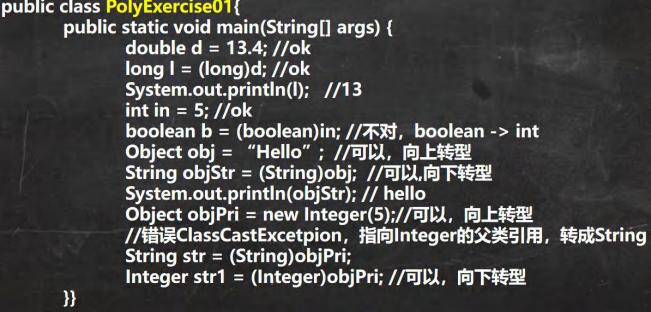
 instanceOf 比较操作符，用于判断对象的运行类型是否为 XX 类型或 XX 类型的子类型【举例说明】PolyDetail03.java

|  |
| --- |
| package com.hspedu.poly\_.detail\_;  public class PolyDetail03 {  public static void main(String[] args) {  BB bb = new BB();  System.out.println(bb instanceof BB);// true  System.out.println(bb instanceof AA);// true  //aa 编译类型 AA, 运行类型是 BB  //BB 是 AA 子类  AA aa = new BB();  System.out.println(aa instanceof AA);  System.out.println(aa instanceof BB);  Object obj = new Object();  System.out.println(obj instanceof AA);//false  String str = "hello";  //System.out.println(str instanceof AA);  System.out.println(str instanceof Object);//true  }  }  class AA {} //父类 |

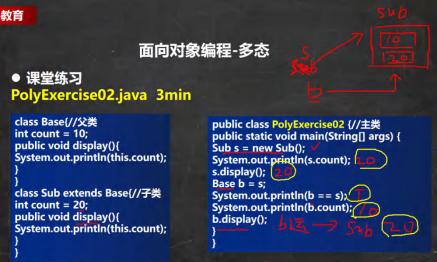
|  |
| --- |
| class BB extends AA {}//子类 |

8.11.6 课堂练习

请说出下面的每条语言，哪些是正确的，哪些是错误的，为什么? 2min 后老师评讲

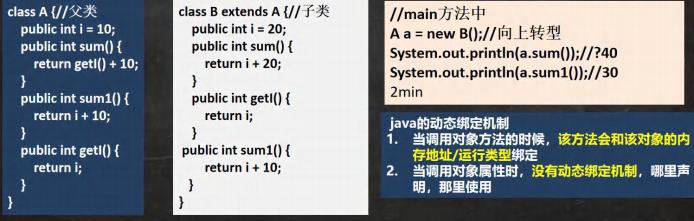


PolyExercise02.java 3min



8.11.7 java 的动态绑定机制(非常非常重要.)

Java 重要特性: 动态绑定机制



代码

|  |
| --- |
| package com.hspedu.poly\_.dynamic\_;  public class DynamicBinding {  public static void main(String[] args) {  //a 的编译类型 A, 运行类型 B  A a = new B();//向上转型  System.out.println(a.sum());//?40 -> 30  System.out.println(a.sum1());//?30-> 20  }  }  class A {//父类  public int i = 10;  //动态绑定机制:  public int sum() {//父类 sum()  return getI() + 10;//20 + 10 |

|  |
| --- |
| }  public int sum1() {//父类 sum1()  return i + 10;//10 + 10  }  public int getI() {//父类 getI  return i;  }  }  class B extends A {//子类  public int i = 20;  // public int sum() {  // return i + 20;  // }  public int getI() {//子类 getI()  return i;  }  // public int sum1() {  // return i + 10;  // }  } |

8.11.8 多态的应用

1) 多态数组 com.hspedu.poly\_.polyarr\_ 包 PloyArray.java

数组的定义类型为父类类型，里面保存的实际元素类型为子类类型

应用实例:现有一个继承结构如下：要求创建 1 个 Person 对象、2 个 Student 对象和 2 个 Teacher 对象, 统一放在数组

中，并调用每个对象

say 方法.

应用实例升级：如何调用子类特有的方法，比如

Teacher 有一个 teach , Student 有一个 study

怎么调用？

代码:

|  |
| --- |
| package com.hspedu.poly\_.polyarr\_;  public class Person {//父类  private String name;  private int age;  public Person(String name, int age) {  this.name = name;  this.age = age;  }  public String getName() { |

|  |
| --- |
| return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  public String say() {//返回名字和年龄  return name + "\t" + age;  }  } |
| package com.hspedu.poly\_.polyarr\_;  public class Student extends Person {  private double score;  public Student(String name, int age, double score) { |

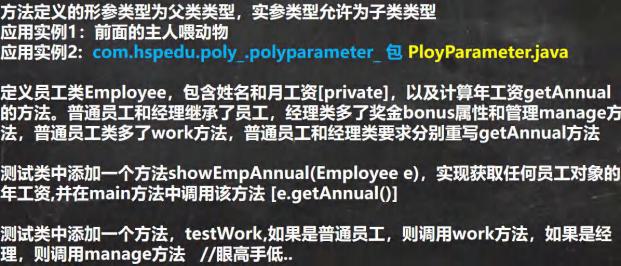
|  |
| --- |
| super(name, age);  this.score = score;  }  public double getScore() {  return score;  }  public void setScore(double score) {  this.score = score;  }  //重写父类 say  @Override  public String say() {  return "学生 " + super.say() + " score=" + score;  }  //特有的方法  public void study() {  System.out.println("学生 " + getName() + " 正在学java...");  }  } |
| package com.hspedu.poly\_.polyarr\_;  public class Teacher extends Person { |

|  |
| --- |
| private double salary;  public Teacher(String name, int age, double salary) {  super(name, age);  this.salary = salary;  }  public double getSalary() {  return salary;  }  public void setSalary(double salary) {  this.salary = salary;  }  //写重写父类的 say 方法  @Override  public String say() {  return "老师 " + super.say() + " salary=" + salary;  }  //特有方法  public void teach() {  System.out.println("老师 " + getName() + " 正在讲java 课程...");  }  } |

|  |
| --- |
| package com.hspedu.poly\_.polyarr\_;  public class PloyArray {  public static void main(String[] args) {  //应用实例:现有一个继承结构如下：要求创建 1 个 Person对象、  // 2 个 Student 对象和 2 个 Teacher 对象, 统一放在数组中，并调用每个对象 say 方法  Person[] persons = new Person[5];  persons[0] = new Person("jack", 20);  persons[1] = new Student("mary", 18, 100);  persons[2] = new Student("smith", 19, 30. 1);  persons[3] = new Teacher("scott", 30, 20000);  persons[4] = new Teacher("king", 50, 25000);  //循环遍历多态数组，调用 say  for (int i = 0; i < persons.length; i++) {  //老师提示: person[i] 编译类型是 Person ,运行类型是是根据实际情况有 JVM 来判断  System.out.println(persons[i].say());//动态绑定机制  //这里大家聪明. 使用 类型判断 + 向下转型.  if(persons[i] instanceof Student) {//判断 person[i] 的运行类型是不是 Student  Student student = (Student)persons[i];//向下转型  student.study();  //小伙伴也可以使用一条语句 ((Student)persons[i]).study();  } else if(persons[i] instanceof Teacher) {  Teacher teacher = (Teacher)persons[i];  teacher.teach(); |

|  |
| --- |
| } else if(persons[i] instanceof Person){  //System.out.println("你的类型有误, 请自己检查...");  } else {  System.out.println("你的类型有误, 请自己检查...");  }  }  }  } |

2) 多态参数



代码:

|  |
| --- |
| package com.hspedu.poly\_.polyparameter\_;  public class Employee { |

|  |
| --- |
| private String name;  private double salary;  public Employee(String name, double salary) {  this.name = name;  this.salary = salary;  }  //得到年工资的方法  public double getAnnual() {  return 12 \* salary;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public double getSalary() {  return salary;  }  public void setSalary(double salary) {  this.salary = salary; |

|  |
| --- |
| }  } |
| package com.hspedu.poly\_.polyparameter\_;  public class Manager extends Employee{  private double bonus;  public Manager(String name, double salary, double bonus) {  super(name, salary);  this.bonus = bonus;  }  public double getBonus() {  return bonus;  }  public void setBonus(double bonus) {  this.bonus = bonus;  }  public void manage() {  System.out.println("经理 " + getName() + " is managing");  }  //重写获取年薪方法  @Override |

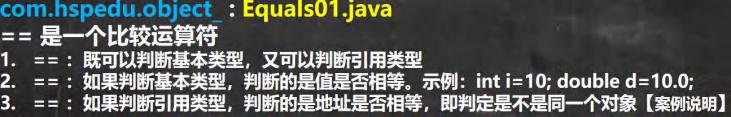
|  |
| --- |
| public double getAnnual() {  return super.getAnnual() + bonus;  }  } |
| package com.hspedu.poly\_.polyparameter\_;  public class Worker extends Employee {  public Worker(String name, double salary) {  super(name, salary);  }  public void work() {  System.out.println("普通员工 " + getName() + " is working");  }  @Override  public double getAnnual() { //因为普通员工没有其它收入，则直接调用父类方法  return super.getAnnual();  }  } |
| package com.hspedu.poly\_.polyparameter\_;  public class PloyParameter {  public static void main(String[] args) {  Worker tom = new Worker("tom", 2500); |

|  |
| --- |
| Manager milan = new Manager("milan", 5000, 200000);  PloyParameter ployParameter = new PloyParameter();  ployParameter.showEmpAnnual(tom);  ployParameter.showEmpAnnual(milan);  ployParameter.testWork(tom);  ployParameter.testWork(milan);  }  //showEmpAnnual(Employee e)  //实现获取任何员工对象的年工资,并在 main 方法中调用该方法 [e.getAnnual()]  public void showEmpAnnual(Employee e) {  System.out.println(e.getAnnual());//动态绑定机制.  }  //添加一个方法，testWork,如果是普通员工，则调用work 方法，如果是经理，则调用 manage 方法  public void testWork(Employee e) {  if(e instanceof Worker) {  ((Worker) e).work();//有向下转型操作  } else if(e instanceof Manager) {  ((Manager) e).manage();//有向下转型操作  } else {  System.out.println("不做处理...");  }  }  } |

8. 12 Object 类详解

8.12.1 equals 方法

 ==和 equals 的对比 [面试题]





|  |
| --- |
| package com.hspedu.object\_;  public class Equals01 {  public static void main(String[] args) {  A a = new A();  Ab = a;  A c = b;  System.out.println(a == c);//true  System.out.println(b == c);//true  B bObj = a;  System.out.println(bObj == c);//true  int num1 = 10; |

|  |
| --- |
| double num2 = 10.0;  System.out.println(num1 == num2);//基本数据类型，判断值是否相等  //equals 方法，源码怎么查看.  //把光标放在 equals 方法，直接输入 ctrl+b  //如果你使用不了. 自己配置. 即可使用.  /\*  //带大家看看 Jdk 的源码 String 类的 equals 方法  //把 Object 的 equals 方法重写了,变成了比较两个字符串值是否相同  public boolean equals(Object anObject) {  if (this == anObject) {//如果是同一个对象  return true;//返回 true  }  if (anObject instanceof String) {//判断类型  String anotherString = (String)anObject;//向下转型  int n = value.length;  if (n == anotherString.value.length) {//如果长度相同  char v1[] = value;  char v2[] = anotherString.value;  int i = 0;  while (n-- != 0) {//然后一个一个的比较字符  if (v1[i] != v2[i])  return false;  i++;  } |

|  |
| --- |
| return true;//如果两个字符串的所有字符都相等，则返回 true  }  }  return false;//如果比较的不是字符串，则直接返回 false  }  \*/  "hello".equals("abc");  //看看 Object 类的 equals 是  /\*  //即 Object 的 equals 方法默认就是比较对象地址是否相同  //也就是判断两个对象是不是同一个对象.  public boolean equals(Object obj) {  return (this == obj);  }  \*/  /\*  //从源码可以看到 Integer 也重写了 Object 的 equals 方法,  //变成了判断两个值是否相同  public boolean equals(Object obj) {  if (obj instanceof Integer) {  return value == ((Integer)obj).intValue(); |

|  |
| --- |
| }  return false;  }  \*/  Integer integer1 = new Integer(1000);  Integer integer2 = new Integer(1000);  System.out.println(integer1 == integer2);//false  System.out.println(integer1.equals(integer2));//true  String str1 = new String("hspedu");  String str2 = new String("hspedu");  System.out.println(str1 == str2);//false  System.out.println(str1.equals(str2));//true  }  }  class B {}  class A extends B {} |

8.12.2 如何重写 equals 方法

应用实例: 判断两个 Person对象的内容是否相等，如果两个 Person对象的各个属性值都一样，则返回 true，反之 false。

EqualsExercise01.java

|  |
| --- |
| package com.hspedu.object\_;  public class EqualsExercise01 {  public static void main(String[] args) {  Person person1 = new Person("jack", 10, '男');  Person person2 = new Person("jack", 20, '男');  System.out.println(person1.equals(person2));//假  }  }  //判断两个 Person 对象的内容是否相等，  //如果两个 Person 对象的各个属性值都一样，则返回 true ，反之 false  class Person{ //extends Object  private String name;  private int age;  private char gender;  //重写 Object 的 equals 方法  public boolean equals(Object obj) {  //判断如果比较的两个对象是同一个对象，则直接返回 true  if(this == obj) {  return true;  }  //类型判断  if(obj instanceof Person) {//是 Person ，我们才比较 |

|  |
| --- |
| //进行 向下转型, 因为我需要得到obj 的 各个属性  Person p = (Person)obj;  return this.name.equals(p.name) && this.age == p.age && this.gender == p.gender;  }  //如果不是 Person ，则直接返回 false  return false;  }  public Person(String name, int age, char gender) {  this.name = name;  this.age = age;  this.gender = gender;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age; |

|  |
| --- |
| }  public void setAge(int age) {  this.age = age;  }  public char getGender() {  return gender;  }  public void setGender(char gender) {  this.gender = gender;  }  } |

8.12.3 课堂练习题

|  |
| --- |
| package com.hspedu.object\_;  public class EqualsExercise02 {  public static void main(String[] args) {  Person\_ p1 = new Person\_();  p1.name = "hspedu"; |

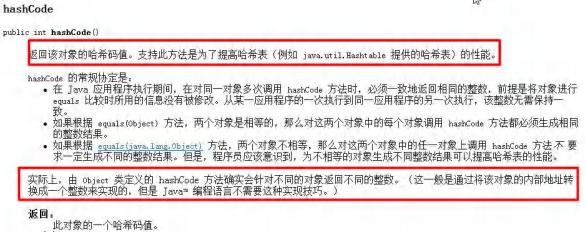
|  |
| --- |
| Person\_ p2 = new Person\_();  p2.name = "hspedu";  System.out.println(p1==p2); //False  System.out.println(p1.name .equals( p2.name));//T  System.out.println(p1.equals(p2));//False  String s1 = new String("asdf");  String s2 = new String("asdf");  System.out.println(s1.equals(s2));//T  System.out.println(s1==s2); //F  }  }  class Person\_ {//类  public String name;  } |

代码如下 EqualsExercise03.java 2min

|  |
| --- |
| //代码如下 EqualsExercise03.java 2min  int it = 65;  float fl = 65.0f;  System.out.println(“65 和 65.0f 是否相等？” + (it == fl));//T |

|  |
| --- |
| char ch1 = ‘A’; char ch2 = 12;  System.out.println(“65 和‘A’是否相等？” + (it == ch1));//T  System.out.println(“ 12 和 ch2 是否相等？” + (12 == ch2));//T  String str1 = new String("hello");  String str2 = new String("hello");  System.out.println("str1 和 str2 是否相等？"+ (str1 == str2)); //F  System.out.println(“str1 是否 equals str2？”+(str1.equals(str2)));//T  System.out.println(“hello” == new java.sql.Date()); //编译错误 |

8.12.4 hashCode 方法



> 老韩的 6 个小结:

1) 提高具有哈希结构的容器的效率！

2) 两个引用，如果指向的是同一个对象，则哈希值肯定是一样的！

3) 两个引用，如果指向的是不同对象，则哈希值是不一样的

4) 哈希值主要根据地址号来的！， 不能完全将哈希值等价于地址。

5) 案例演示[HashCode\_.java]: obj.hashCode() [测试：A obj1 = new A(); A obj2 = new A(); A obj3 = obj1] 6) 后面在集合，中 hashCode 如果需要的话，也会重写, 在讲解集合时，老韩在说如何重写 hashCode()

代码

|  |
| --- |
| package com.hspedu.object\_;  public class HashCode\_ {  public static void main(String[] args) {  AA aa = new AA();  AA aa2 = new AA();  AA aa3 = aa;  System.out.println("aa.hashCode()=" + aa.hashCode());  System.out.println("aa2.hashCode()=" + aa2.hashCode());  System.out.println("aa3.hashCode()=" + aa3.hashCode());  }  }  class AA {} |

8.12.5 toString 方法

1) 基本介绍

默认返回：全类名+@+哈希值的十六进制，【查看 Object 的 toString 方法】

子类往往重写toString 方法，用于返回对象的属性信息

2) 重写toString 方法，打印对象或拼接对象时，都会自动调用该对象的toString 形式.

案例演示：Monster [name, job, sal] 案例: ToString\_.java

3) 当直接输 出一个对 象 时 ，toString 方法会被默认 的调用, 比如 System.out.println(monster) ； 就会默认调用

monster.toString()

代码:

|  |
| --- |
| package com.hspedu.object\_;  public class ToString\_ {  public static void main(String[] args) {  /\*  Object 的 toString() 源码  (1)getClass().getName() 类的全类名(包名+类名 )  (2)Integer.toHexString(hashCode()) 将对象的 hashCode 值转成 16 进制字符串  public String toString() {  return getClass().getName() + "@" + Integer.toHexString(hashCode());  }  \*/  Monster monster = new Monster("小妖怪", "巡山的", 1000);  System.out.println(monster.toString() + " hashcode=" + monster.hashCode());  System.out.println("==当直接输出一个对象时，toString 方法会被默认的调用==");  System.out.println(monster); //等价 monster.toString() |

|  |
| --- |
| }  }  class Monster {  private String name;  private String job;  private double sal;  public Monster(String name, String job, double sal) {  this.name = name;  this.job = job;  this.sal = sal;  }  //重写 toString 方法, 输出对象的属性  //使用快捷键即可 alt+insert -> toString  @Override  public String toString() { //重写后，一般是把对象的属性值输出，当然程序员也可以自己定制  return "Monster{" +  "name='" + name + '\'' +  ", job='" + job + '\'' +  ", sal=" + sal +  '}';  }  @Override |

|  |
| --- |
| protected void finalize() throws Throwable {  System.out.println("fin..");  }  } |

8.12.6 finalize 方法

1) 当对象被回收时，系统自动调用该对象的 finalize方法。子类可以重写该方法，做一些释放资源的操作【演示】

2) 什么时候被回收：当某个对象没有任何引用时，则jvm 就认为这个对象是一个垃圾对象，就会使用垃圾回收机制来

销毁该对象，在销毁该对象前，会先调用 finalize方法。

3) 垃圾回收机制的调用，是由系统来决定(即有自己的 GC 算法), 也可以通过 System.gc() 主动触发垃圾回收机制，测

试：Car [name]

老韩提示： 我们在实际开发中，几乎不会运用 finalize , 所以更多就是为了应付面试.

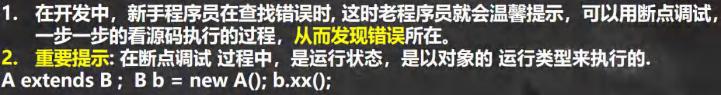
代码:

|  |
| --- |
| package com.hspedu.object\_;  //演示 Finalize 的用法  public class Finalize\_ {  public static void main(String[] args) {  Car bmw = new Car("宝马");  //这时 car 对象就是一个垃圾,垃圾回收器就会回收(销毁)对象, 在销毁对象前，会调用该对象的 finalize方法  //,程序员就可以在 finalize 中，写自己的业务逻辑代码(比如释放资源：数据库连接,或者打开文件..)  //,如果程序员不重写 finalize,那么就会调用 Object 类的 finalize, 即默认处理 |

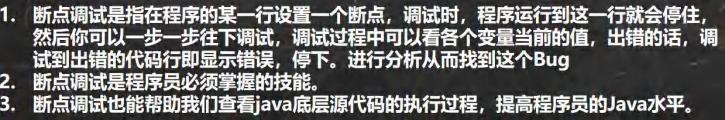
|  |
| --- |
| //,如果程序员重写了 finalize, 就可以实现自己的逻辑  bmw = null;  System.gc();//主动调用垃圾回收器  System.out.println("程序退出了....");  }  }  class Car {  private String name;  //属性, 资源。。  public Car(String name) {  this.name = name;  }  //重写 finalize  @Override  protected void finalize() throws Throwable {  System.out.println("我们销毁 汽车" + name );  System.out.println("释放了某些资源...");  }  } |

8. 13 断点调试(debug)

8.13.1 一个实际需求



8.13.2 断点调试介绍



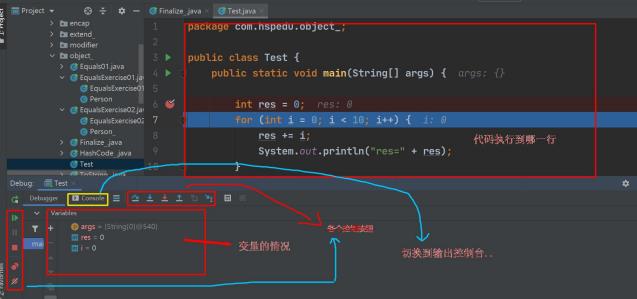
8.13.3 断点调试的快捷键

F7(跳入) F8(跳过) shift+F8(跳出) F9(resume,执行到下一个断点)

F7：跳入方法内

F8: 逐行执行代码.

shift+F8: 跳出方法



8.13.4 断点调试应用案例

看几段代码，演示调试过程

8.13.5 断点调试应用案例

1) 案例 1 com.hspedu.debug\_ 包 Debug01.java

看一下变量的变化情况等

|  |
| --- |
| package com.hspedu.debug\_;  public class Debug01 {  public static void main(String[] args) {  //演示逐行执行代码  int sum = 0;  for (int i = 0; i < 5; i++) {  sum += i;  System.out.println("i=" + i);  System.out.println("sum=" + i);  }  System.out.println("退出 for....");  }  } |

2) 案例 2

看一下数组越界的异常 Debug02.java

|  |
| --- |
| package com.hspedu.debug\_; |

|  |
| --- |
| public class Debug02 {  public static void main(String[] args) {  int[] arr = {1, 10, - 1};  for (int i = 0; i <= arr.length; i++) {  System.out.println(arr[i]);  }  System.out.println("退出 for");  }  } |

3) 案例 3

演示如何追源码，看看java 设计者是怎么实现的。(提高编程思想)。

小技巧：将光标放在某个变量上，可以看到最新的数据。 Debug03.java

|  |
| --- |
| package com.hspedu.debug\_;  import java.util.Arrays;  public class Debug03 {  public static void main(String[] args) {  int[] arr = {1, - 1, 10, -20 , 100};  //我们看看 Arrays.sort 方法底层实现.->Debug  Arrays.sort(arr);  for (int i = 0; i < arr.length; i++) {  System.out.print(arr[i] + "\t");  } |

|  |
| --- |
| }  } |

4) 案例 4

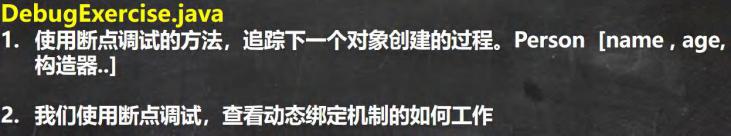
演示如何直接执行到下一个断点 F9 resume。

老韩小技巧: 断点可以在 debug 过程中，动态的下断点

|  |
| --- |
| package com.hspedu.debug\_;  import java.util.Arrays;  //演示执行到下一个断点，同时支持动态的下断点.  public class Debug04 {  public static void main(String[] args) {  int[] arr = {1, - 1, 10, -20 , 100};  //我们看看 Arrays.sort 方法底层实现.->Debug  Arrays.sort(arr);  for (int i = 0; i < arr.length; i++) {  System.out.print(arr[i] + "\t");  }  System.out.println("hello100");  System.out.println("hello200");  System.out.println("hello300");  System.out.println("hello400"); |

|  |
| --- |
| System.out.println("hello500");  System.out.println("hello600");  System.out.println("hello700");  }  } |

8.13.6 断点调试课后练习



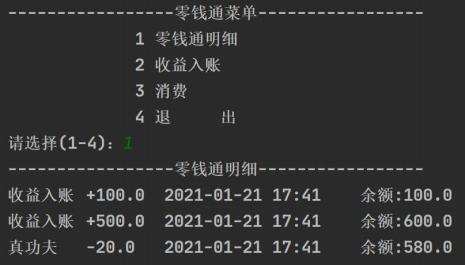
8. 14 项目-零钱通

8.14.1 项目开发流程说明

8.14.2 项目需求说明

使用 Java 开发 零钱通项目 , 可以完成收益入账，消费，查看明细，退出系统等功能.

8.14.3 项目的界面



化繁为简.

1) 先完成显示菜单，并可以选择

2) 完成零钱通明细.

3) 完成收益入账

4) 消费

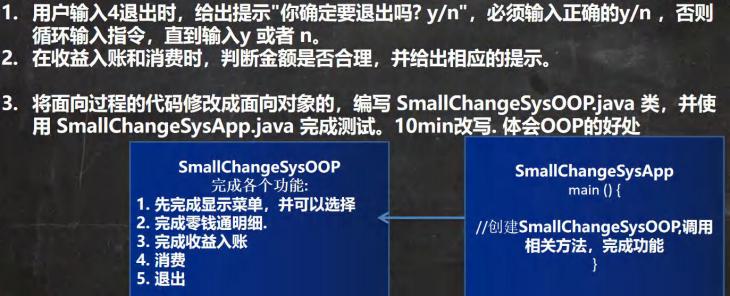
5) 退出

8.14.4 项目代码实现

编写文件 SmallChangeSys.java 完成基本功能 (过程编程)

老师提示：先使用过程编程，后面改成 OOP 版本，请小伙伴体会 OOP 编程带来的好处

8.14.5 项目代码实现改进



项目代码打包：



smallchange.zip

8. 15 本章作业



