# day08

## 继承

### 继承的概述

1. 继承是面向对象中的一大特征
2. 继承 : 类与类之间的关系, 子父类继承关系, 使用extends关键字表示, extends表示继承, 扩展概念

举例 :

class A{}

class B extends A{}// B类继承A类

A类 : 被继承的类, 称为父类, 超类, 基类

B类 : 用于继承的类, 称为子类, 派生类

### 继承发生的场景

同一个类别的多个事物之间具有共性(成员变量, 方法), 向上抽取出来, 抽取到一个父类中, 由父类将共性内容实现, 剩下的类型做为这个父类的子类存在, 子类中, 设计的就是 每一个子类特有功能即可, 共性内容子类不需要再重复实现, 可以从父类中继承使用

继承关系都是属于关系 : 猫属于动物; 狗属于动物

举例 :

猫类 : 吃eat功能, 抓老鼠catchMouse功能

狗类 : 吃eat功能, 看家lookHome功能

猫类和狗类都属于动物(同类别), 多个事物之间共性eat方法, 抽象到父类Animal中实现, 在猫类和狗类中, 只需要实现特有功能, 猫--->抓老鼠; 狗--->看家, 猫和狗作为Animal类的子类存在

代码

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Cat extends Animal{  /\*public void eat() {  System.out.println("吃饭");  }\*/    public void catchMouse() {  System.out.println("猫可以抓老鼠");  }  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Dog extends Animal{  /\*public void eat() {  System.out.println("吃饭");  }\*/    public void lookHome() {  System.out.println("狗可以看家");  }  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Animal {  public void eat() {  System.out.println("吃饭");  }  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class TestAnimalExtends {  public static void main(String[] args) {  // 1. 创建出一个猫类对象  Cat c = new Cat();  c.eat(); // eat方法来自于从父类Animal中继承来的  c.catchMouse();// catchMouse方法来自于Cat类型本身    // 2. 创建出一个狗类对象  Dog d = new Dog();  d.eat();// eat方法来自于从父类Animal中继承来的  d.lookHome();// lookHome方法来自于Dog类型本身  }  } |

### 继承设计的案例练习 :

1. 需求:

程序员类 : 姓名, 年龄, 工号, 工资, 功能: 工作

项目经理类 : 姓名, 年龄, 工号, 工资, 奖金, 功能 : 工作

1. 分析:

程序员类和项目经理类, 都属于员工, 将共性抽取出来(姓名, 年龄, 工号, 工资, 功能: 工作), 在父类员工类(Employee)中实现, 子类程序员类和项目经理类只需要实现特有内容

1. 继承关系中代码实现步骤:

先设计出父类中共性内容, 再到子类中补充特有内容

代码

|  |
| --- |
| package com.ujiuye.extendsDemo;  // Employee员工类表示公司中所有员工  public class Employee {  // 姓名, 年龄, 工号, 工资, 功能: 工作  String name;  int age;  String id;  double salary;    public void work() {  System.out.println(name + "员工,工作了" + age + "年, 员工编号为" + id + ",薪资为" + salary);  }  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  // Programmer表示程序员类, 属于员工的一种  public class Programmer extends Employee{  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  // XingMuManager表示项目经理类, 属于员工  public class XingMuManager extends Employee{  private double jiangJin;  public double getJiangJin() {  return jiangJin;  }  public void setJiangJin(double jiangJin) {  this.jiangJin = jiangJin;  }  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class TestEmployeeExtends {  public static void main(String[] args) {  Programmer p = new Programmer();  p.name = "程序员";  p.age = 20;  p.id = "001";  p.salary = 10000;    p.work();    XingMuManager m = new XingMuManager();  m.name = "项目经理";  m.age = 25;  m.id = "xm001";  m.salary = 10000;    m.setJiangJin(3000);  m.work();  System.out.println("本月奖金为:" + m.getJiangJin());  }  } |

### 继承的注意事项

1. 父类中的私有成员(成员变量, 方法功能)无法被子类继承使用

因为私有使用private关键字修饰的, 私有只能在当前类型中使用; 父类中私有成员也只能在父类中使用, 除了父类不能再用, 子类不是父类本类, 所以私有无法被子类继承使用

1. 父类中构造方法不能被子类继承, 子类构造方法中可以调用父类构造

父类构造父类名Fu, 子类构造需要是子类名Zi, 父类构造方法名字Fu与子类类名不同, 冲突, 违背构造方法定义规则, 因此构造方法不能继承

代码

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Fu {  // 1. 成员变量  int i = 10;  private int j = 20;    // 2. 定义出方法功能  public void fun() {  System.out.println("我是父类中fun方法");  }    private void sleep() {  System.out.println("我是父类私有方法sleep");  }  } |

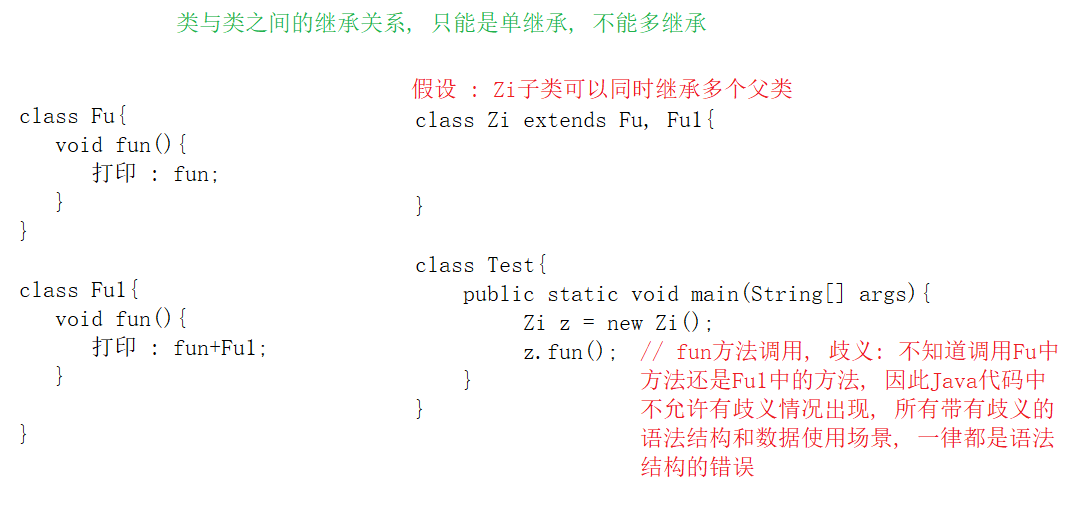
|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Zi extends Fu {  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class TestFuAndZi {  public static void main(String[] args) {  Zi z = new Zi();  System.out.println(z.i);// 10  // The field Fu.j is not visible  // 1. 子类中不能继承使用父类中的私有成员变量  // System.out.println(z.j);    z.fun();// 我是父类中fun方法  // The method sleep() from the type Fu is not visible  // 2. 子类中不能继承使用父类中的私有方法  // z.sleep();  }  } |

### 继承的特点

1. Java中类与类之间继承关系: 只能单继承, 不能多继承, 可以多层继承
2. 单继承 : 一个子类只能有一个直接父类

不能多继承 : 一个子类不能同时继承多个父类

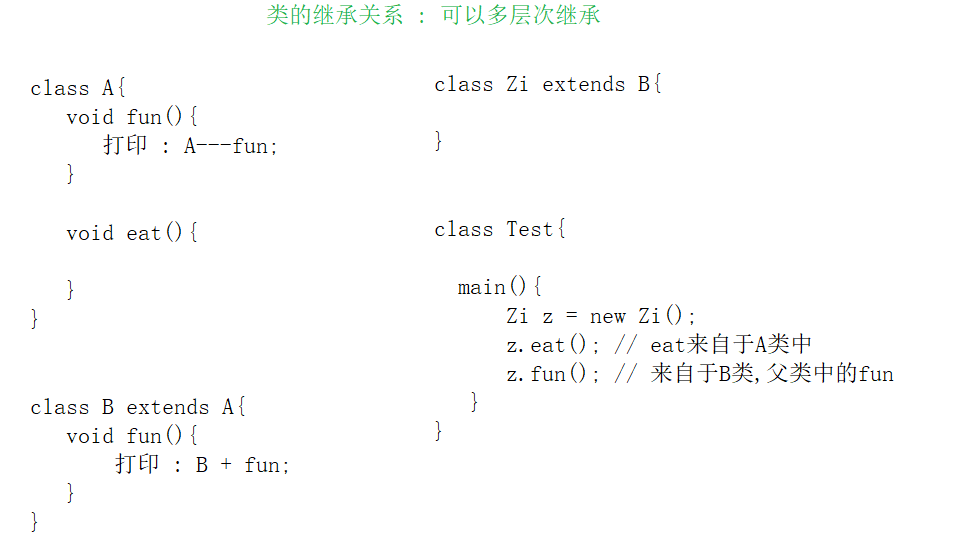


多层继承 : 传承概念

举例 : class A{}

class B extends A{}

class C extends B{} // C类可以继承到B类和A类中所有可继承内容, 先以B类为第一继承顺位



代码

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class A {  void fun() {  System.out.println("A----fun");  }    void eat() {  System.out.println("A---eat");  }  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class B extends A{  void fun() {  System.out.println("B----fun");  }  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  /\*  \* 类与类之间只能单继承  \* public class C extends B,A{  }\*/  // 实现类多层次继承, C类中可以继承到B类和B的父类A类中的所有可继承内容, 以B类第一继承顺序  public class C extends B{    } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Test继承的特点 {  public static void main(String[] args) {  C c = new C();  c.eat();// A---eat  c.fun();// B----fun  }  } |

### 继承中的成员变量

1. 子类中定义的成员变量与父类中成员变量名字不同, 子类可以使用自己定义出的成员变量也可以使用从父类继承来的成员变量(父类中私有成员变量不被子类继承使用)
2. 子类中定义出了与父类同名成员变量, 变量访问具有就近原则:

当使用某一个变量时, 优先使用离变量最近的一次定义数据; 现象: 如果有局部变量, 优先使用局部变量; 没有局部变量, 再使用成员变量; 如果没有成员变量, 再到父类中使用可继承成员...一直找到最后(Object类), 没有才会报出变量不存在错误

1. super 关键字 : 表示当前类型对应的父类引用

super关键字使用在子类中, 用于表示当前子类对应的父类的引用, 因此super第一种使用场景就是 : 可以区分子父类中重名的成员变量或者方法功能, 带有super关键字的就表示父类中的成员变量或者方法的调用

代码

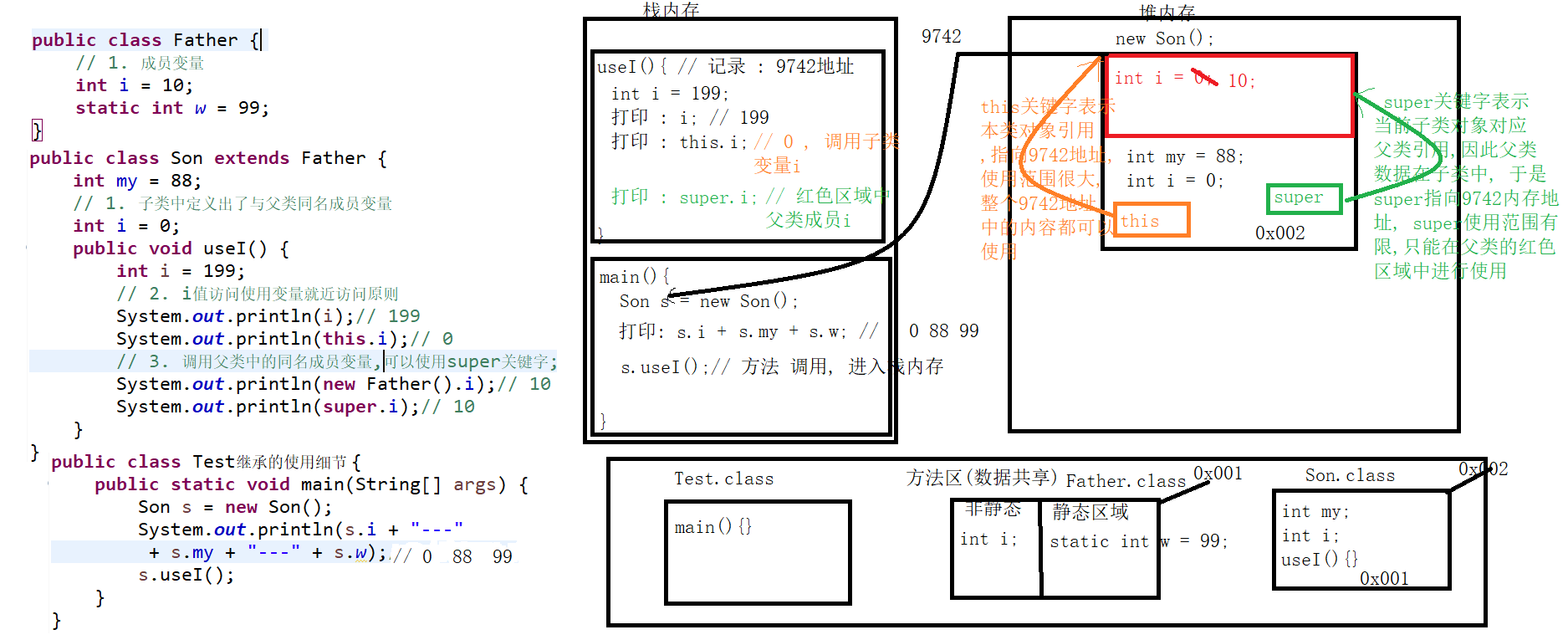
|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Father {  // 1. 成员变量  int i = 10;  static int w = 99;  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Son extends Father {  int my = 88;  // 1. 子类中定义出了与父类同名成员变量  int i = 0;  public void useI() {  int i = 199;  // 2. i值访问使用变量就近访问原则  System.out.println(i);// 199  System.out.println(this.i);// 0  // 3. 调用父类中的同名成员变量,可以使用super关键字; i = 10  System.out.println(new Father().i);// 10  System.out.println(super.i);// 10  }  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Test继承的使用细节 {  public static void main(String[] args) {  Son s = new Son();  System.out.println(s.i + "---" + s.my + "---" + s.w);// 10, 88, 99  s.useI();  }  } |

### this和super关键字在内存中的理解

1. 具有继承关系的类, 如果子类要进入到内存中, 父类必须优先于子类进入到内存, 因为子类可以继承父类中数据(成员变量, 方法), 因此需要父类先进入内存将可继承数据加载子类才能使用
2. 父类优先于子类进入内存体现在:
3. 父类的.class字节码文件优先于子类的.class字节码文件进入到方法区中,加载数据
4. 在堆内存中, 父类中的数据, 会存在于子类的对象中, 并且先将父类中的数据加载进入到堆内存, 后加载子类数据进入堆内存



### 继承中的方法

1. 子类中定义的方法功能与父类中继承到的功能不同(方法名不同), 那么子类可以使用自定义出的方法, 也可以使用从父类继承来方法(私有方法不能被子类继承使用)
2. 如果子类对于从父类中继承到的方法功能觉得不满意, 没有实现子类最终要求, 子类可以重写从父类继承来的方法功能(重写的方法还是属于父类, 只不过子类作了新的实现)

重写为了让方法功能越来越好:

1. 方法重写注意:
2. 要求重写方法 : 返回值类型, 方法名, 方法参数列表上与父类方法保持一致
3. 方法体可以做任意实现, 可以满足子类的实际需求即可
4. 修饰符上 , 子类重写方法权限修饰上大于等于父类原有方法权限

public---->默认权限--->priavte

默认权限 : 定义一个类, 方法, 变量时, 没有给出任何权限修饰符, 那么认为是默认权限

1. 子类重写方法功能, 在方法声明之上, 尽量使用注解进行标识和验证方法属于重写方法, @Override
2. 在子类中调用父类方法功能 : super.父类方法名(实际参数);
3. 什么是方法重载(Overload), 什么是方法重写(Override)

重载 : 定义在同一个类中, 方法名相同, 参数列表不同, 与方法返回值类型无关多个方法之间是重载

重写 : 子父类的继承关系中, 子类重写父类中继承到方法功能, 修饰符在权限上大于等于父类, 返回值类型, 方法名, 参数列表与父类一致, 方法体子类自行实现

代码

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Father {  // 1. 成员变量  int i = 10;  static int w = 99;  private int fa = 888;    // 2. 方法功能  public void eat() {  System.out.println("父类中的eat功能");  }    public void sleep() {  System.out.println("父类中的sleep功能");  }  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Son extends Father {  int my = 88;  // 1. 子类中定义出了与父类同名成员变量  int i = 0;  // 2. 子类中定义出于父类不同方法功能  public void useI() {  int i = 199;  // 2. i值访问使用变量就近访问原则  System.out.println(i);// 199  System.out.println(this.i);// 0  // 3. 调用父类中的同名成员变量,可以使用super关键字; i = 10  System.out.println(new Father().i);// 10  System.out.println(super.i);// 10  }    // 3.子类重写从父类Father中继承来方法功能eat  @Override  public void eat() {  // 4. 使用super.调用父类中方法功能  super.eat();  System.out.println("子类Son重写父类方法eat, 子类不吃胡萝卜");  }  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Test继承的使用细节 {  public static void main(String[] args) {  Son s = new Son();  // 1. 测试子父类继承中成员变量  System.out.println(s.i + "---"  + s.my + "---" + s.w);// 10, 88, 99    // 2. 测试子父类继承中方法功能  s.eat();// 调用子类重写方法功能  s.sleep();// 调用从父类继承来的方法  s.useI(); // 调用子类的特有方法  }  } |

### 继承中的构造方法

1. 父类中构造方法不能被子类继承, 子类构造方法中可以调用父类构造
2. 在子类数据初始化之前, 必须先初始化父类的数据, 因此子类构造方法第一行, 默认super() 存在, super()表示父类空参数构造方法调用

为什么要先调用父类构造 :

先调用父类构造--->因为父类构造方法是为了给父类成员变量赋值--->因此父类成员变量先加载进内存中, 才能调用父类构造赋值

等父类成员变量进入内存完毕, 父类构造方法执行完毕, 父类成员变量有值, 子类就可以继承使用父类中的可继承数据

1. 如果在子类构造方法第一行手动调用了一个父类构造,系统就不会默认调用任何构造方法

注意 :

super() 父类构造调用, 必须在子类构造方法有效行第一行;

this() 本类构造方法之间调用, 必须在构造方法有效行第一行; 目的就是间接的让super()优先于子类构造进入内存

因此两者不能同时在构造中存在, 只能二选一;

代码

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Father {  // 1. 成员变量  int i = 10;  static int w = 99;  private int fa = 888;    // 2. 方法功能  public void eat() {  System.out.println("父类中的eat功能");  }    public void sleep() {  System.out.println("父类中的sleep功能");  }    // 3. 构造方法  public Father() {  System.out.println("父类构造执行了");  }    public Father(int i) {  this.i = i;  System.out.println("父类有参构造执行");  }  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Son extends Father {  int my = 88;  // 1. 子类中定义出了与父类同名成员变量  int i = 0;  // 2. 子类中定义出于父类不同方法功能  public void useI() {  int i = 199;  // 2. i值访问使用变量就近访问原则  System.out.println(i);// 199  System.out.println(this.i);// 0  // 3. 调用父类中的同名成员变量,可以使用super关键字; i = 10  System.out.println(new Father().i);// 10  System.out.println(super.i);// 10  }    // 3.子类重写从父类Father中继承来方法功能eat  @Override  public void eat() {  // 4. 使用super.调用父类中方法功能  super.eat();  System.out.println("子类Son重写父类方法eat, 子类不吃胡萝卜");  }    // 5. 子类构造方法  public Son() {  //super();// 默认存在. 调用父类空参数构造  // 子类构造方法第一行调用父类构造, 保证父类中数据优先于子类进入内存中  super(9);  }    public Son(String s) {  this();  System.out.println(s);  }  } |

|  |
| --- |
| package com.ujiuye.extendsDemo;  public class Test继承的使用细节 {  public static void main(String[] args) {  Son s = new Son();  // 1. 测试子父类继承中成员变量  System.out.println(s.i + "---"  + s.my + "---" + s.w);// 10, 88, 99    // 2. 测试子父类继承中方法功能  s.eat();// 调用子类重写方法功能  s.sleep();// 调用从父类继承来的方法  s.useI(); // 调用子类的特有方法  }  } |

### this和super关键字的使用总结

1. this : 关键字, 表示当前类型对象引用

a : 区分成员变量和局部变量的重名问题, 带有this.关键字表示调用成员变量

b : this()用于在本类构造方法之间进行调用, 需要写在构造方法有效行第一行

1. super : 关键字, 表示当前类型对象的父类引用

a : 区分子类和父类重名变量和方法功能, 带有super.关键字调用父类中的成员变量或者方法功能

b : super()用于子类构造方法中, 调用父类构造, 保证让父类数据优先于子类数据进入内存中, 需要写在构造方法有效行第一行

注意 : super() 和 this() 构造方法调用, 都必须在第一行的位置, 于是二选一

### 继承需求:

程序员类 : 姓名, 年龄, 工号, 工资, 功能: 工作

项目经理类 : 姓名, 年龄, 工号, 工资, 奖金, 功能 : 工作

分析:

程序员类和项目经理类, 都属于员工, 将共性抽取出来(姓名, 年龄, 工号, 工资, 功能: 工作), 在父类员工类(Employee)中实现, 子类程序员类和项目经理类只需要实现特有内容

Employee (员工类): 姓名, 年龄, 工号, 工资, 功能: 工作

Program (程序员类): 继承Employee

Manager(项目经理类) : 继承Employee, 额外具有奖金属性

代码

|  |
| --- |
| package com.ujiuye.extendslx;  public class Employee {  // 姓名  private String name;  // 年龄  private int age;  // 工号  private String id;  // 工资  private double salary;    // 功能: 工作  public void work() {  System.out.println("每个人都需要工作");  }    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 getId() {  return id;  }  public void setId(String id) {  this.id = id;  }  public double getSalary() {  return salary;  }  public void setSalary(double salary) {  this.salary = salary;  }  // 父类添加有参数构造  public Employee(String name, int age, String id, double salary) {  super();  this.name = name;  this.age = age;  this.id = id;  this.salary = salary;  }    public Employee() {}  } |

|  |
| --- |
| package com.ujiuye.extendslx;  public class Program extends Employee {  // Employee父类中的4个私有成员变量无法继承使用  // Program可以继承4个私有成员变量对应间接访问方式, set和get方法  // 可以继承work方法  @Override  public void work() {  // System.out.println("每个人都需要工作");  super.work();  System.out.println(super.getName() + "--" + super.getAge() + "程序员工作就是开发");  }    public Program() {    }    public Program(String name, int age, String id, double salary) {  super(name,age,id,salary);  }  } |

|  |
| --- |
| package com.ujiuye.extendslx;  public class Manager extends Employee {  // 奖金  private double jiangJin;  public double getJiangJin() {  return jiangJin;  }  public void setJiangJin(double jiangJin) {  this.jiangJin = jiangJin;  }    @Override  public void work() {  System.out.println(super.getName() + "--" + super.getAge() + "--" + super.getId() + "--" + super.getSalary()+"工作监督项目进度, 协助开发人员解决问题, 奖金" + jiangJin );  }    public Manager(String name, int age, String id, double salary,double jiangJin) {  super.setName(name);  super.setAge(age);  super.setId(id);  super.setSalary(salary);  this.jiangJin = jiangJin;  }  } |

|  |
| --- |
| package com.ujiuye.extendslx;  public class Test {  public static void main(String[] args) {  // 1. 程序员类  Program p = new Program();  p.setName("小明");  p.setAge(20);  p.setId("001");  p.setSalary(12000);    p.work();    // 2. 项目经理类 : 想要通过构造方法赋值  Manager m = new Manager("小强", 25, "jl001", 13000, 3000);  m.work();    // 3. 程序员类使用构造赋值  Program p1 = new Program("美丽", 22, "002", 10000);  p1.work();  }  } |

## final关键字

1. final : 关键字, 修饰符, 表示最终的, 最后的, 不可改变的
2. final修饰:
3. 修饰类 : 这个类就是一个最终类, 不可改变的类; 使用final修饰的类, 不能当父类; 其他一切使用正常(JDK的核心类库中比较常见使用final修饰类)
4. 修饰方法 : 使用final修饰方法, 可以被子类继承使用, 但是子类不能重写(JDK中核心类库里面的部分方法功能比较常见使用final修饰方法)
5. 修饰变量 : 实际开发中最常用,final修饰变量只能有一次手动赋值(JVM虚拟机默认赋值不生效), 赋值之后这个变量的值不能再修改, 数据不能修改,称为常量

final修饰的变量 : 称为常量, 行业内部命名规范, 每一个英文单词全部都大写, 多个单词之间使用\_进行分隔

a : final修饰基本数据类型, 表示这个类型的数值不可改变

b : final修饰引用数据类型, 表示这个引用类型变量的地址值不可改变

c : final修饰成员变量, 赋值机制, 使用变量之前必须完成手动赋值, 如果使用fianl修饰成员变量, 建议创建成员变量同时马上赋值

final修饰类代码

|  |
| --- |
| package com.ujiuye.finaldemo;  // 1. FinalClass就是一个使用final修饰的最终类, 不能当父类  public final class FinalClass {  int i = 10;  } |

|  |
| --- |
| package com.ujiuye.finaldemo;  //The type FinalClassZi cannot subclass the final class FinalClass  // 子类  /\*public class FinalClassZi extends FinalClass {  }\*/ |

final修饰方法代码

|  |
| --- |
| package com.ujiuye.finaldemo;  public class FinalMethod {  // 1. 定义出一个普通方法  public void fun() {  System.out.println("父类的fun");  }    // 2. 定义出一个使用final修饰方法  public final void play() {  System.out.println("父类中的final最终方法, 可以继承, 不能重写");  }  } |

|  |
| --- |
| package com.ujiuye.finaldemo;  public class FinalMethodZi extends FinalMethod {  // 继承到了父类中的两个方法 : fun() ; play();  // fun可以重写, play不能重写  @Override  public void fun() {  System.out.println("我是子类重写fun");  }    // Cannot override the final method from FinalMethod  // 无法重写使用final修饰方法  /\*public void play() {  System.out.println("父类中的final最终方法, 可以继承, 不能重写");  }\*/  } |

final修饰变量代码

|  |
| --- |
| package com.ujiuye.finaldemo;  public class FinalVariable {  //The blank final field SCHOOL\_NAME may not have been initialized  static final String SCHOOL\_NAME = "第一中学";    int j = 88;  public static void main(String[] args) {  // 1. final修饰基本数据类型  final int I = 10;  // The final local variable i cannot be assigned.  // It must be blank and not using a compound assignment  // I = 20;    // 2. final修饰引用数据类型, 表示这个引用类型变量的地址值不可改变  final FinalVariable fv = new FinalVariable();  // fv = new FinalVariable();  fv.j = 8888;  // final修饰的变量不能修改  // fv.SCHOOL\_NAME = "二中";  }  } |