第5章　类的继承

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

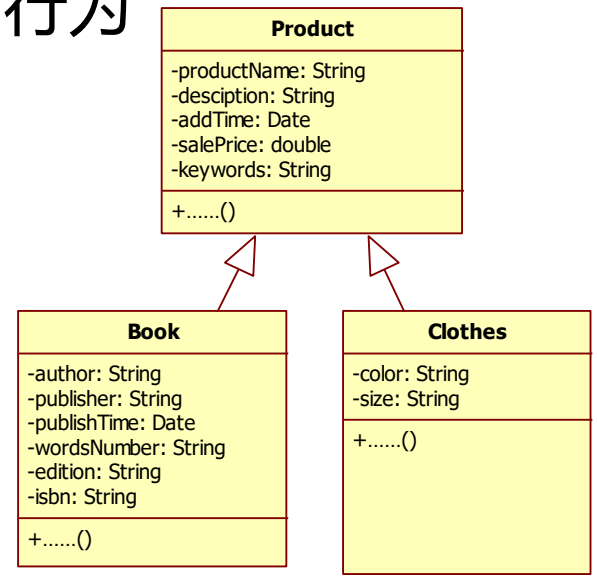
- 继承的概念和作用
- 子类的声明
- 类成员修饰符与继承的关系
- 理解父类和子类的关系，在子类中使用父类成员(方法)
- 继承机制下，对象创建的过程(构造方法的使用,super调用)
- 子类对父类方法的重写(super调用)

5.1.1 继承的概念

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- 代码复用的手段
  - “is-a” 关系: 类之间是继承的关系
  - “has-a”关系: 类之间是组合的关系

- 继承：从已有的类中派生出新的类。
  - 新的类能吸收已有类的数据属性和行为
  - 并能扩展新的能力
  - 举例：商品

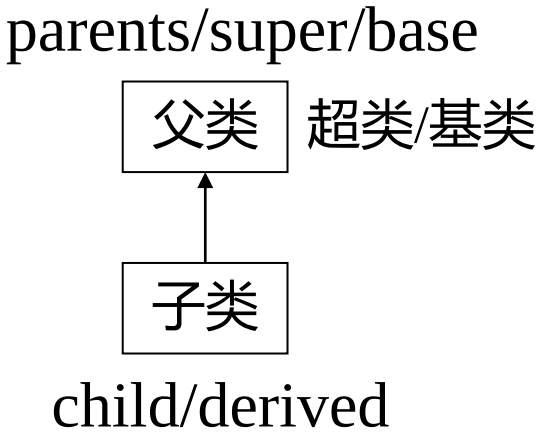


当你创建一个类时，你总是在继承。除非你明确指出要从其他类中继承，否则你就隐式的从 Java’s 的标准根源类 **Object** 进行继承.

5.1.1 继承的概念

- 父类是子类的一般化，子类是父类的特例化(具体化)。
- 父类也称为超类或基类。

父 类	子 类
学生	研究生、大学生、小学生
形状	三角形、圆、矩形
雇员	教师、医生、职员
交通工具	轿车、卡车、公交车
水果	苹果、梨、桃、桔



- 继承分类
  - 单继承：一个子类最多只能有一个父类。
  - 多继承：一个子类可有两个以上的父类。

- Java类只支持单继承，每个子类只能有一个直接父类；

- 而Java接口支持多继承。Java多继承的功能则是通过接口方式来间接实现的。

- 子类定义的一般格式

```
[类修饰符]  class 子类名 extends 父类名{
        成员变量定义;
        成员方法定义;
}
```

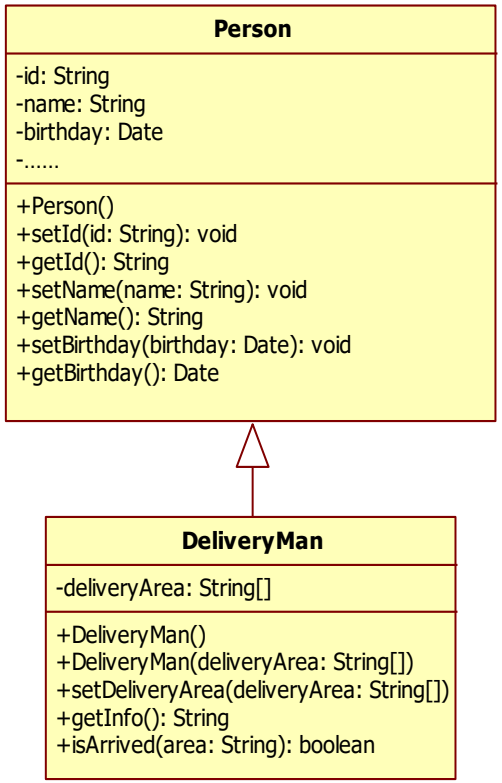
  - 在子类的定义中，用关键字extends来明确指出它所继承的父类。
  - 子类会自动的得到基类中所有的数据乘员和成员方法

【例5-1】通过继承Person类派生DeliveryMan类。

快递公司的不同职工

职工Person
快递员 DeliveryMan

- 数据成员配送区域deliveryArea
- 获取配送区域方法getDeliveryArea()
- 获取快递员信息方法getInfo()
- 判断某区域是否在配送范围内方法isArrived()



5.1.2 继承的实现

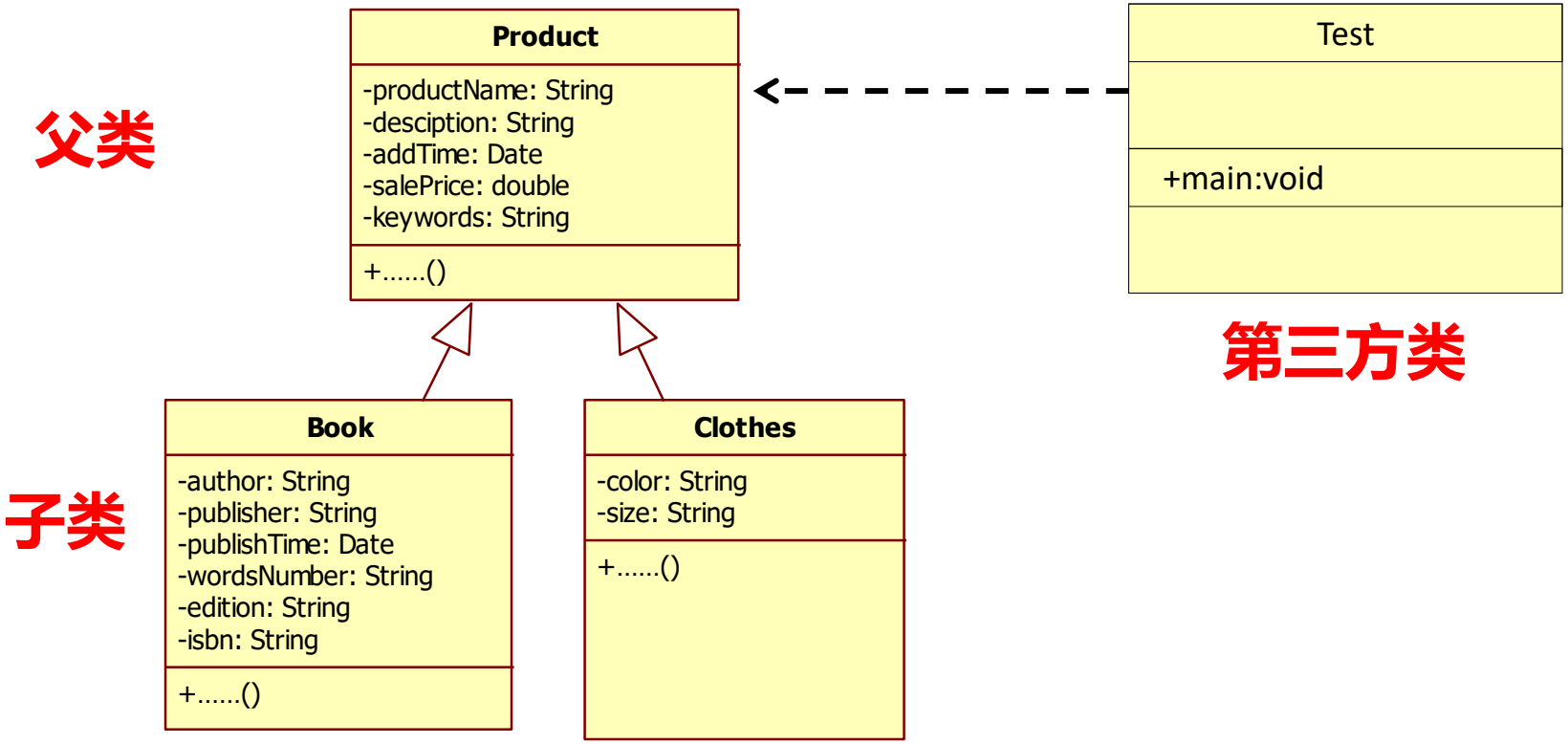
```
public class DeliveryMan extends Person {
        //新增数据成员
        private String[] deliveryArea;

        //新增方法
        public void setDeliveryArea(String[] deliveryArea) {
                this.deliveryArea = deliveryArea;
        }
        public String getInfo(){//拼写快递员信息字符串
                String str = getId()+","+getName()+"\n配送范围:";
                int i;
                for(i=0; i<deliveryArea.length-1; i++){//除最后一个外，都有一个逗号
                        str+=deliveryArea[i]+",";
                }
                return str+deliveryArea[i];
        }
```

```
public boolean isArrived(String area){//检查area是否在配送范围
                for(String d: deliveryArea){
                        if(d.equalsIgnoreCase(area)){
                                return true;
                        }
                }
                return false;
        }
}
```

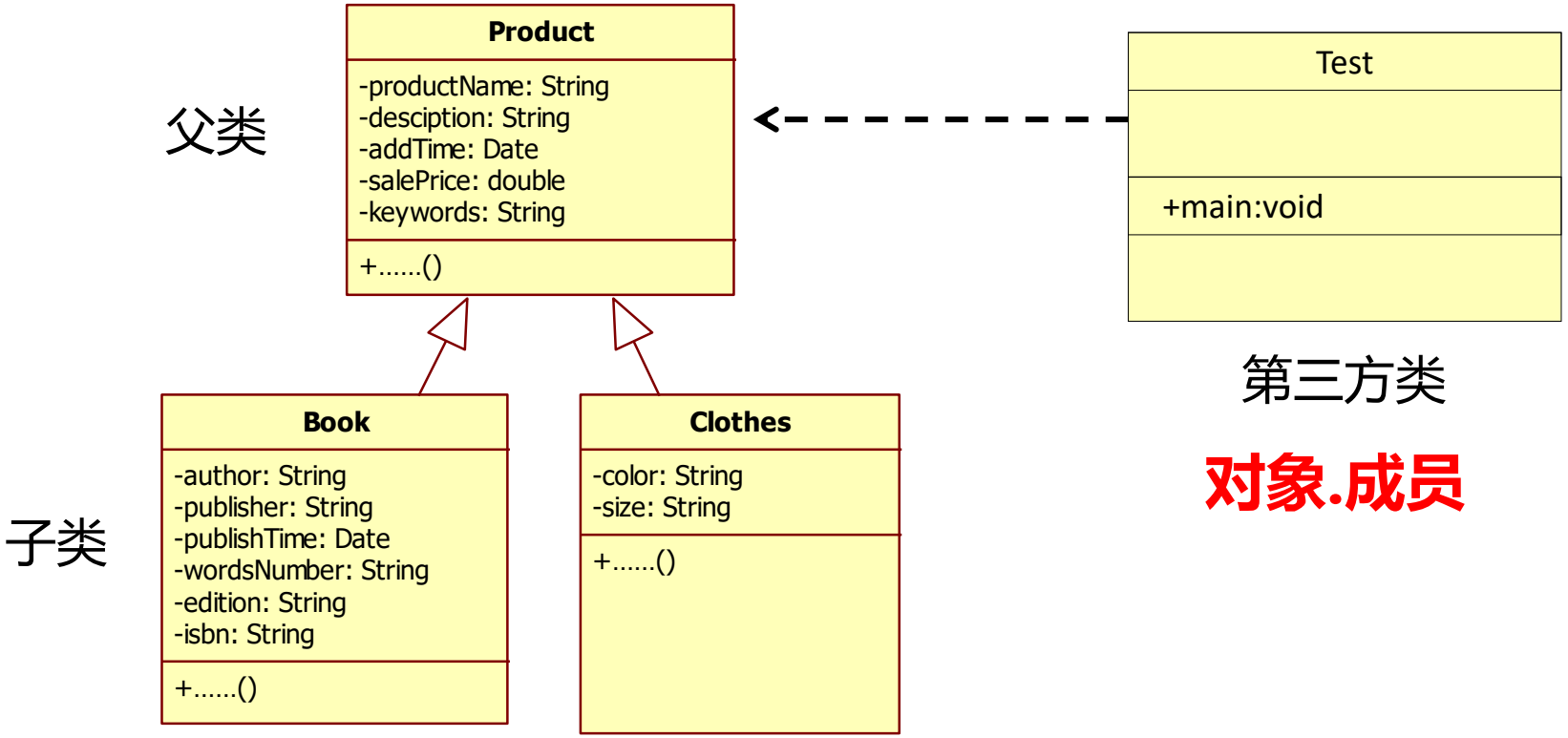
- 要区分“存在”与“可见”之间的关系。private的成员与其他成员一样都被继承到子类中（是存在的），只是它们不能被子类直接使用而已（不可见）。

- 成员访问控制修饰符在继承中的性质
  - public、private、package、protected
  - 父类的public成员可以在父类中使用，也可以在子类使用。程序可以在任何地方访问public父类成员。
  - 父类的private成员仅在父类中使用，在子类中不能被访问。
  - 父类的protected成员可在子类被访问，无论子类与父类是否存储在同一个包下。
  - 父类的package成员可在同一包的子类中被访问。

5.1.3 类成员的访问控制



5.1.3 类成员的访问控制



子类对于可以访问的父类成员，直接使用

- 成员访问控制修饰符的具体应用
  - 父类中属性的修饰符：子类应依赖于父类的服务，而不应依赖于父类的数据。所以，应该将父类中的成员变量声明为private，并在父类中定义访问这些private成员变量的public型的方法。(public、protected和package较少修饰属性)
  - 父类中方法的修饰符：关于父类中的private方法，如果某个方法是为类中其他的方法提供服务，只在类中使用，将其定义为private，对外界隐藏。

5.1.3 类成员的访问控制

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- 基类（父类）的protected成员（包括成员变量和成员方法）对本包内可见，并且对子类可见；
- 若子类与基类（父类）不在同一包中，那么在子类中，只有子类实例可以访问其从基类继承而来的protected方法，而在子类中不能访问基类实例（对象）（所调用）的protected方法。
- 不论是否在一个包内，父类中可以访问子类实例（对象）继承的父类protected修饰的方法。（子父类访问权限特点：父类访问域大于子类）
- 若子类与基类（父类）不在同一包中,子类只能在自己的类（域）中访问父类继承而来的protected成员，无法访问别的子类实例（即便同父类的亲兄弟）所继承的protected修饰的方法。
- 若子类与基类（父类）不在同一包中，父类中不可以使用子类实例调用（父类中没有）子类中特有的（自己的）protected修饰的成员。（毕竟没有满足同一包内和继承获得protected成员的关系）

- 子类从父类继承成员时，父类的所有public、protected、package成员，在子类中都保持它们原有的访问修饰符。
  - 例如，父类的public成员成为子类的public成员。父类的protected成员也会成为子类的protected成员。
  - 子类只能通过父类所提供的非private方法来访问父类的private成员。

对象.成员　　直接使用

类A成员的访问控制符	类A对类A成员的访问权限	第三方类对类A成员的访问权限		子类B对父类A成员的访问权限	
		与A同包	与A不同包	与A同包	与A不同包
public	√	√	√	√	√
protected	√	√	×	√	√
默认(package)	√	√	×	√	×
private	√	×	×	×	×

【例5-2】分析同包下无继承关系的类之间成员的访问控制权限。

```
package chap5.example.access;
public class C {
        public int a=1;
        protected int b=2;
        int c=3;
        private int d=4;
        public int getD(){
                return d;
        }
}
```

```
package chap5.example.access;
public class AccessDemo {
        public static void  main(String[] args) {
                C coo = new C();
                System.out.println(coo.a);
                System.out.println(coo.b);
                System.out.println(coo.c);
                //System.out.println(coo.d);
               System.out.println(coo.getD());

        }
}
```

【例5-3】分析不同包下子类对父类成员的访问控制权限。

```
package chap5.example.access.sub;
public class C {
        public int a;
        protected int b;
        int c;
        private int d;
        public int getD(){
                return d;
        }
}
```

```
package chap5.example.access;
import chap5.example.access.sub.C;
public class AccessDemo extends C{
        public void getInfo(){
                System.out.println(a);
                System.out.println(b);
                System.out.println(c);
                System.out.println(d);
        }
}
```

总结：

1. 为了继承，一般的规则是所有的数据成员都指定为 private 将所有的方法指定为 public or protected.

2. 对在基类中定义的方法可以进行修改

3. 你并不一定非要使用基类的方法，你也可以在子类当中添加新的方法

- Object类：Java中所有类的父类，定义和实现了Java系统下所有类的共同行为，所有的类都是由这个类继承、扩充而来的。
- 认识Object类中的方法

5.2.1 重写及其意义

- 子类扩展父类，大部分时候是子类以父类为基础，额外增加新的数据成员和方法。但除此之外，重写（override）父类的方法也是普遍存在的。

- 覆盖方法的返回类型，方法名称，参数的个数及类型必须和被覆盖的方法一摸一样
- 只需在方法名前面使用不同的类名或不同类的对象名即可区分覆盖方法和被覆盖方法
- 覆盖方法的访问权限可以比被覆盖的宽松，但是不能更为严格

- 为什么要重写呢，本章从现实世界的角度来说。设一个动物Animal类中有一个表示动物行动方式的move()方法（它的功能可能仅仅是告诉你动物是可以运动的）。而子类Bird中move的方式是飞翔，子类Fish中move的方式是游水，子类Wolf的move方式的奔跑……，可见，这些子类都需要重写父类的move()方法以表达自身的更为贴切的行为方式。

【例5-4】定义Animal类的子类Bird，并重写它的move()方法。

```
public class Animal {
        public void move(){
                System.out.println("我可以move...");
        }
}
public class Bird extends Animal{
        public void move(){
                System.out.println("我可以在天空飞翔.....");
        }
        public staticvoid main(String[] args){
                Bird bird = newBird();
                bird.move();   //输出"我可以在天空飞翔....."
        }
}
```

- 方法的重写遵循“两同两小一大”的规则
  - “两同”：方法名称相同、形参列表相同
  - “两小”
    - 子类方法返回值类型≤父类方法返回值类型
    - 子类方法抛出的异常≤父类方法抛出的异常
  - “一大”：子类方法的访问权限≥父类方法的访问权限

  - 重写方法时不能改变方法的static或非static性质。

5.2.2 Object类与重写toString()方法

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- Object类中定义了9个方法
  - clone()：clone() 是 Object 方法，通过该方法克隆一个一模一样的对象。一般通过实现 cloneable 接口实现该方法；
  - finalize()：在对象被垃圾收集器析构(回收)之前调用，用来清除回收对象；
  - toString()：把非字符串的数据类型转化为字符串，输出结果：包名.类名+@+16进制的哈希值；
  - equals()：用于检测一个对象是否等于另外一个对象；equals 方法判断两个对象是否具有相同的引用。如果两个对象具有相同的引用，则返回true，否者返回false；hashCode()：在散列存储结构中确定对象的存储地址，用hashcode来代表对象就是在hash表中的位置；
  - notify()：唤醒在此对象锁(监视器)上等待的单个线程。
  - notifyAll()：唤醒在此对象锁(监视器)上等待的所有线程
  - wait()：当前线程释放对象锁(监视器)的拥有权，在其他线程调用此对象的 notify() 方法或 notifyAll() 方法前，当前线程处于等待状态。
  - getClass()：getClass() 返回此 Object 的运行时类

5.2.2 Object类与重写toString()方法

- toString()方法：对对象的文字描述，返回一个字符串。
- 如果在新创建的类中重写toString ()方法，返回一个与该类具体相关的字符串，则打印对象的操作将变得非常简单，直接调用System.*out*.println()方法即可。正因为如此，toString()方法通常都会被重写。

【例5-5】在DeliveryMan类中重写toString方法。

```
public class DeliveryMan extends Person {
        private String[] deliveryArea;
……
        public String toString(){
                String str = getId()+","+getName()+"\n配送范围:";
                int i;
              for(i=0; i<deliveryArea.length-1; i++){//除最后一个外，
都有一个逗号
                        str+=deliveryArea[i]+",";
                }
                return str+deliveryArea[i];
        }
}
```

【例5-5】在DeliveryMan类中重写toString方法。

```
public class Test {
        public static void main(String[] args) {
                DeliveryMan dm = new DeliveryMan();
                dm.setId("007");
                dm.setName("Bang");

                dm.setDeliveryArea(new String[]{"南锣鼓巷","烟袋斜
街","雨儿胡同","帽儿胡同","黑芝麻胡同"});
                System.out.println("快递员信息:"+dm);
        }
}
```

5.2.3 调用父类被重写的方法

- 如果子类重写了父类的方法，那么在子类对象默认调用的是自己重写后的方法，如何能调用父类被重写了的方法呢，可以使用super。
- super是Java提供的一个关键字，用于对象调用它从父类继承得到的数据成员或方法（通常在子类成员与父类成员同名的情况下使用，否则画蛇添足）。
- 与this一样，super也不能出现在static修饰的方法中。

【例5-6】利用super调用父类的同名方法。

```
public class Animal {
        public void move(){
                System.out.println("我可以move...");
        }
}
public class Bird extends Animal{
        public void move(){
                super.move();     //调用父类的move()方法
                System.out.println("我可以在天空飞翔.....");
        }
        public static void main(String[] args){
                Bird bird = new Bird();
                bird.move();   //输出"我可以move...我可以在天空飞
翔....."
        }
}
```

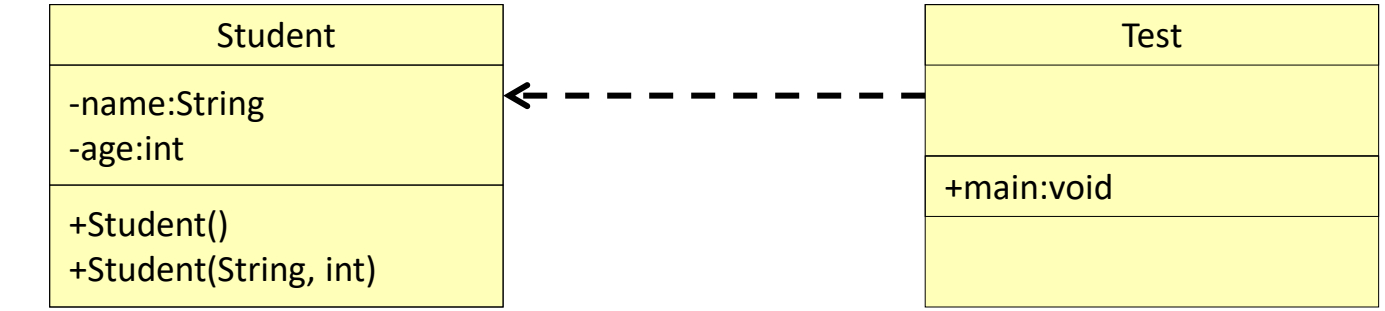
5.2.4 Object类的clone()方法与对象的复制

- 复制：在堆内存中制造副本。

- Object类中的clone()方法可以将对象复制一份并返回给调用者，程序在运行时， clone()方法可以识别要复制的对象，然后为新对象分配存储空间，并将原始对象的内容一一复制到新对象的存储空间中。

```
protected Object clone() throws
CloneNotSupportedException
```

5.2.4 Object类的clone()方法与对象的复制



clone(): java.lang.Object

第三方类

5.2.4 Object类的clone()方法与对象的复制

【例5-7】在Student类中重写Object的clone()方法，实现Student对象的深复制。

Object类中的clone()方法的访问修饰符是protected，所以不能用下面的方式复制对象。

```
public class Student {
        private String name;
        private int age;
        public Student() {}
        public Student(String name, int age) {
                this.name = name;
                this.age = age;
        }
}
```

5.2.4 Object类的clone()方法与对象的复制

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

```
public class Student {
        private String name;
        private int age;
        public Student() {}
        public Student(String r
                this.name = r
                this.age = ag
        }
}
```

- Test类相对于Student类来讲，身份是“第三方类”（既不是Student类本身，也不是Student的子类）
- clone()方法来自于Object，与Test不同包，所以Test没有权利访问Student从Object继承来的protected修饰的方法。

```
public class Test {
        public static void main(String[] args){
                Student stu1 = new Student("Lucy",15);
                Student stu2 = null;
                try {
                        //此句报错
                        stu2 = (Student)stu1.clone();
                } catch (CloneNotSupportedException e) {
                        e.printStackTrace();
                }
......
        }
}
```

5.2.4 Object类的clone()方法与对象的复制

- 实现方法：
  - 在Student类中重写Object类的clone()方法，并将clone()方法的访问修饰符改为public（重写父类方法时，访问修饰符可以比父类更宽泛）。
  - 在clone()方法中用super.clone()的形式调用父类Object中的clone()方法，完成复制行为。
  - 此外，依照语法规则，Student类要实现Cloneable接口（接口将在第6章学习），该接口仅仅起到标识该类的对象是可以复制的作用，并没有实际功能。

5.2.4 Object类的clone()方法与对象的复制


```
public class Student implements Cloneable{
        // implements Cloneable表示实现Cloneable接口
        private String name;
        private int age;
……
        public Object clone() throws CloneNotSupportedException{
                //重写Object类的clone()方法
                return  (Student)super.clone();
                //调用Object类的clone()功能完成复制
```

super.clone()是Object类型，要强制转换为Student类型

5.2.4 Object类的clone()方法与对象的复制

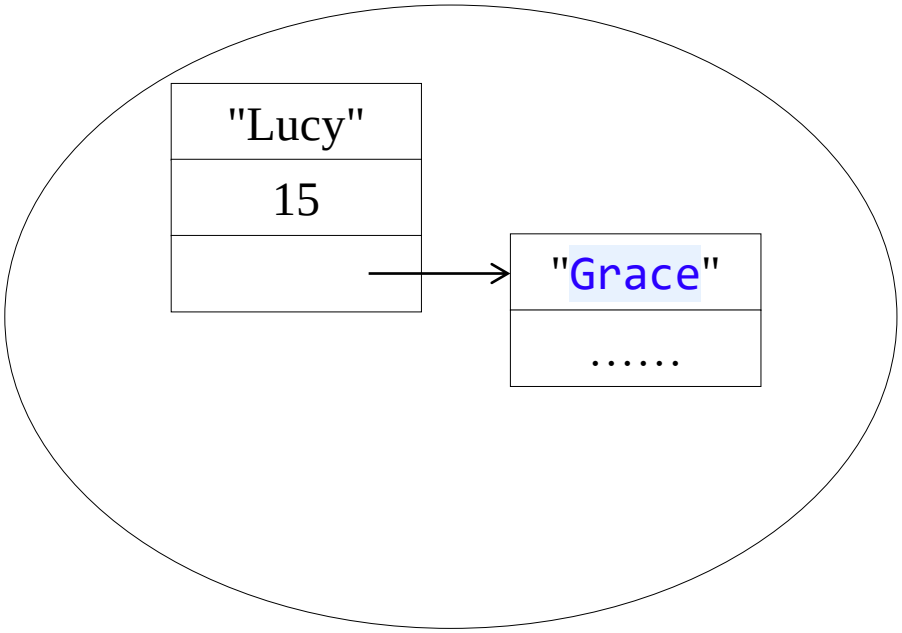

```
public class Test {
        public static void main(String[] args) throws
CloneNotSupportedException{
                Student stu1 = new Student("Lucy",15);
                Student stu2 = stu1.clone();
                System.out.println(stu1);
                System.out.println(stu2);
        }
}
```

5.2.4 Object类的clone()方法与深、浅复制

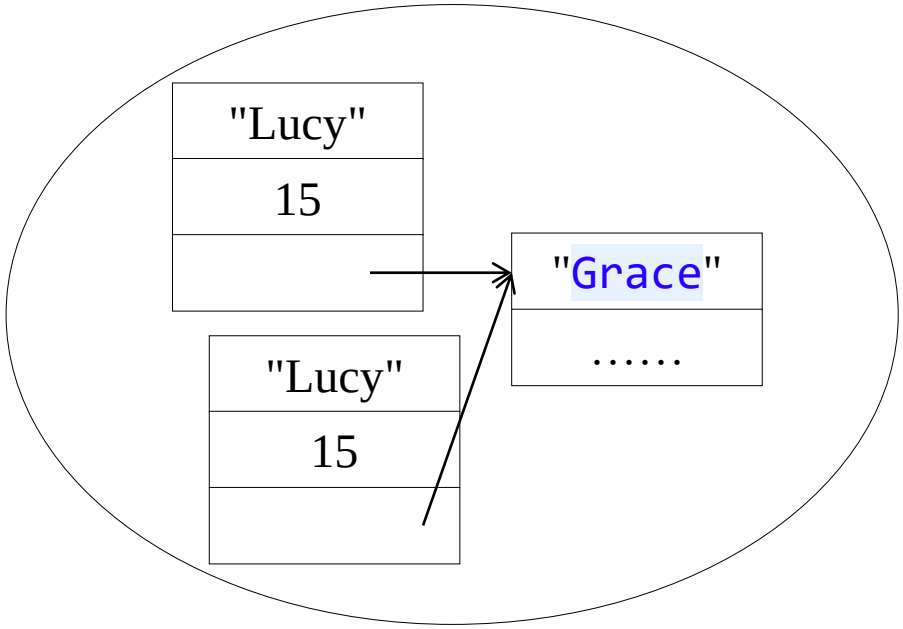
- Java中将对象的复制分为浅复制与深复制两种（复制也称克隆(clone)）。
- 浅复制指被复制对象的所有数据成员都含有与原来对象相同的值，包括引用类型的数据成员，即不复制引用类型数据成员所指向的对象。
- 深复制实现的是数据成员的全面复制，包括复制产生引用成员所指向的对象的副本。

5.2.4 Object类的clone()方法与深、浅复制

浅复制



堆内存

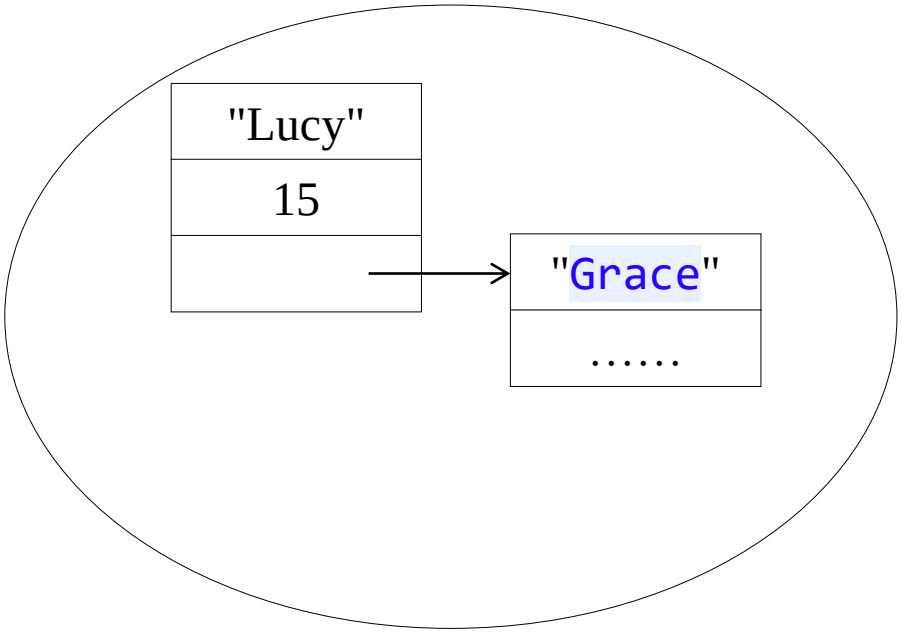


堆内存

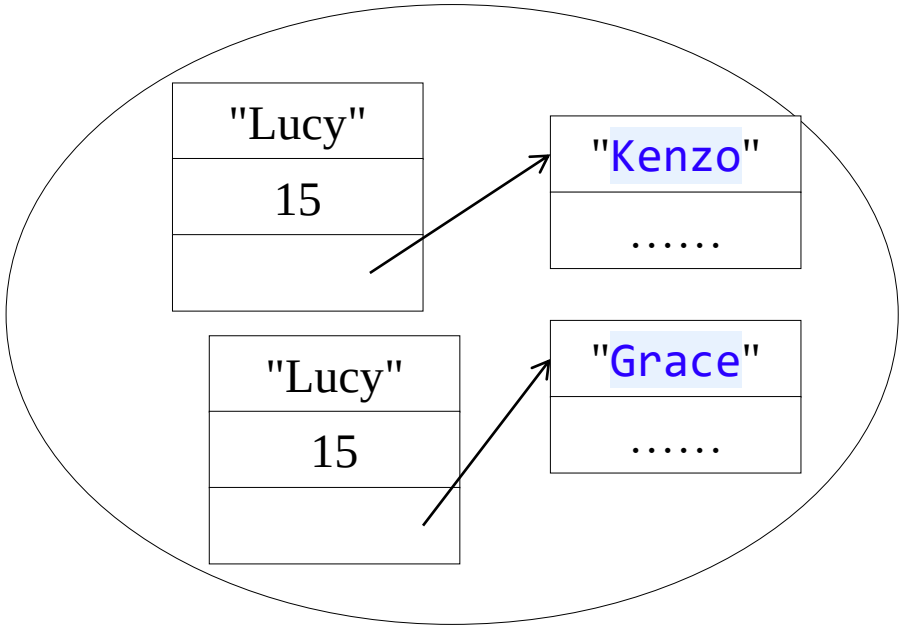
5.2.4 Object类的clone()方法与深、浅复制

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

深复制



堆内存



堆内存

5.2.4 Object类的clone()方法与深、浅复制

【例】向Student中组合一个Teacher类的引用成员。

实现深复制时，如果对象中包含引用成员，则该引用成员所属类也需要重写clone()方法。在复制对象时，再单独复制引用成员所指对象。

浅复制

```
public class Teacher{
        private String name;
        public Teacher(String name) {this.name = name;}
        public Teacher() {}
        public String getName() {       return name;}
        public void setName(String name) {this.name = name;}
}
public class Student implements Cloneable{
        private String name;
        private int age;
        private Teacher teacher;  //增加Teacher类型的引用成员

        public Student() {}
        public Student(String name, int age, Teacher teacher) {
                this.name = name;
                this.age = age;
                this.teacher = teacher;
        }
```

浅复制

```
    public Teacher getTeacher() {return teacher;}
    public void setTeacher(Teacher teacher) {
            this.teacher = teacher;
    }
    public Object clone() throws CloneNotSupportedException{
            return (Student)super.clone();
    }
    public String toString(){
            return name+","+age+","+teacher.getName();
    }
}
```

浅复制

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

请思考下面代码的运行结果是什么?

```
public class Test {
        public static void main(String[] args) throws
CloneNotSupportedException{
                Teacher teacher = new Teacher("Grace");
                Student stu1 = new Student("Lucy",15, teacher);
                Student stu2 = (Student)stu1.clone();

                stu1.getTeacher().setName("Kenzo");
                System.out.println("stu1:"+stu1);
                System.out.println("stu2:"+stu2);
        }
}
```

深复制

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- 代码改进如下：

```
public class Teacher implements Cloneable{
        ……
        public Object clone() throws CloneNotSupportedException{
                //重写clone方法
                return (Teacher)super.clone();
        }
}
```

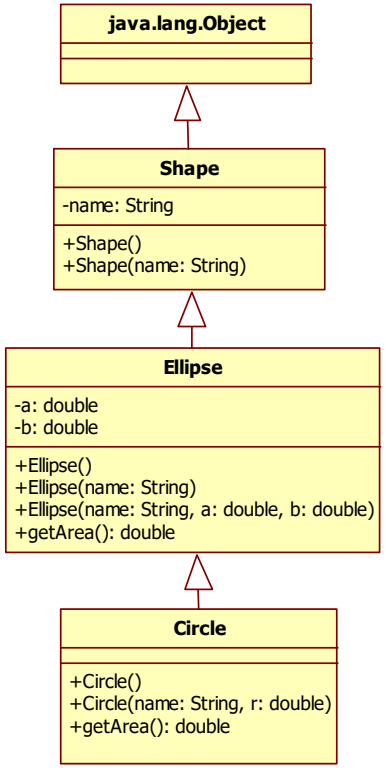
深复制

```
public class Student implements Cloneable{
        ……
        public Object clone() throws CloneNotSupportedException{
                Student stu = (Student)super.clone();
                //复制引用成员所指对象
                this.teacher = (Teacher)stu.getTeacher().clone();
                return stu;
        }
}
```

- **继承下的构造方法的调用次序**
  - 子类构造方法在执行自己的任务之前，将**显式**地(通过super引用)或**隐式**地(调用父类默认的无参数构造方法)调用其直接父类的构造方法。
  - 类似地，如果父类派生于另一个类，则要求父类的构造方法调用上一级类的构造方法，依此类推。调用请求中，最先调用的一定是Object类的构造方法。
  - 创建对象的过程：先父后子。

5.3.1 子类对象的构造过程

【例5-8】已知圆Circle→椭圆Ellipse→形状Shape的继承关系，查看Circle对象构建的过程。




```
public class Shape {
        private String name;
        public Shape() {
                System.out.println("Shape()...");
        }                                          super();
        ......
}
public class Ellipse extends Shape{
        private double a; //短轴
        private double b;  //长轴
        public Ellipse() {
                System.out.println("Ellipse()...");
        }                                           super();
        ......
}
public class Circle extends Ellipse{
        public Circle() {
                System.out.println("Cicle()...");
        }                                           super();
}
```

```
public class Test {

    public static void main(String[] args) {
        new Circle();
    }
}
```

Shape()...
Ellipse()...
Cicle()...

为了避免错误，父类中至少定义一个无参的构造方法。

```
public class Test {

    public static void main(String[] args) {
        new Circle("大圆",100);
    }
}
```

Shape(String)...
Ellipse(String)...
Ellipse(String,double,double)...
Circle(String,double)...

子类构造方法隐式调用父类无参构造方法

```
class Pet{
    private String name;
    public Pet(){
        System.out.print(1);
    }
    public Pet(String name){
        System.out.print(2);
    }
}
class Dog extends Pet{
    public Dog(String name){
        System.out.print(3);
    }
}
class Test{
    public static void main(String[] args) {
        new Dog("秋田");
    }
}
```

子类构造方法隐式调用[illegible]造方法

```
class Pet{
    private String name;
    public Pet(String name){
        System.out.print(2);
    }
}
class Dog extends Pet{
    public Dog(String name){
        System.out.print(3);
    }
}
class Test{
    public static void main(String[] args) {
        new Dog("秋田");
    }
}
```

```
public Pet () {
}
```

子类构造方法**隐式调用**父类无参构造方法

```
class Pet{
    private String name;
    public Pet(){
        System.out.print(1);
    }
    public Pet(String name){
        System.out.print(2);
    }
}
class Dog extends Pet{
    public Dog(String name){
        super();
        System.out.print(3);
    }
}
class Test{
    public static void main(String[] args) {
        new Dog("秋田");
    }
}
```

5.3.2 super与this调用构造方法

- 子类不会继承父类的构造方法，但是子类可以调用父类的构造方法，如同一个类的构造方法可以用this()调用自己的重载构造方法一样。子类调用父类构造方法使用super()完成。

```
public class Shape {
        private String name;
        public Shape() {
                System.out.println("Shape()...");
        }
        public Shape(String name) {
                this.name = name;
                System.out.println("Shape(String)...");
        }
}
```

5.3.2 super与this调用构造方法

```
public class Ellipse extends Shape{
        private double a; //短轴
        private double b;  //长轴
        public Ellipse() {
                System.out.println("Ellipse()...");
        }
        public Ellipse(String name) {
                super(name);          //调用Shape的带参构造方法
                System.out.println("Ellipse(String)...");
        }
        public Ellipse(String name, double a, double b) {
                this(name);   //调用本类的重载构造方法
                this.a = a;
                this.b = b;
                System.out.println("Ellipse(String,double,double)...");
        }
}
```

5.3.2 super与this调用构造方法

```
public class Circle extends Ellipse{
        public Circle() {
                System.out.println("Cicle()...");
        }
        public Circle(String name, double r) {
                super(name,r,r);   //调用Ellipse的构造方法
                System.out.println("Circle(String,double)...");
        }
}
```

5.3.2 super与this调用构造方法

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- 构造方法调用的过程

  （1）子类构造方法的第一行使用super()显式调用父类的构造方法，编译系统根据super的实参列表调用对应的父类构造方法。

  （2）子类构造方法的第一行使用this()显式调用本类重载的构造方法，编译系统根据this的实参列表调用对应的本类构造方法。执行本类另一个构造方法时会显式或隐式地调用父类的构造方法。

  （3）如果子类构造方法中既没有super调用，也没有this调用，系统会在执行子类构造方法前隐式调用父类无参的构造方法。

- **5.4.1 final修饰符**
  - final修饰类时，类不能被继承
  - final修饰方法时，方法不能被重写
  - final修饰变量时，变量只能被赋值一次

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

1．final类

- final修饰的类不能有子类。像java.lang.String、java.lang.Math等就是final类，不能被继承，它们的方法禁止被重写。
- 当子类继承父类时，可以访问到父类内部的数据、可以重写父类的方法，这些可能导致一些不安全的因素。如果可以确定某个类不会再被扩展，或其实现的细节不允许有任何改动，可以使用final修饰这个类。例如：
- *public final class FinalClass{}*
- 如果试图对该类进行继承，则会出现编译错误。

2．final方法

- final修饰的方法不能被重写，例如java.lang.Object类中的getClass()方法。Object类是一定会被继承的（它是所有类直接或间接的父类），但是Java不希望子类重写这个方法，所以使用final把它保护起来。Object中的final方法还有notify()、notifyAll()和wait()，其他方法没有final修饰，均可以被重写。
- 同样，在实际项目开发中，原则上不允许使用final方法。

3．final变量

- 无论final修饰哪种变量，无论这种变量在何处被赋初值，final变量的赋值永远只能有一次。
  - final成员变量
  - final局部变量
  - final方法参数

【例5-9】final修饰成员变量示例。

【例5-10】final修饰局部变量示例。

5.4.2 Java修饰符之间的关系

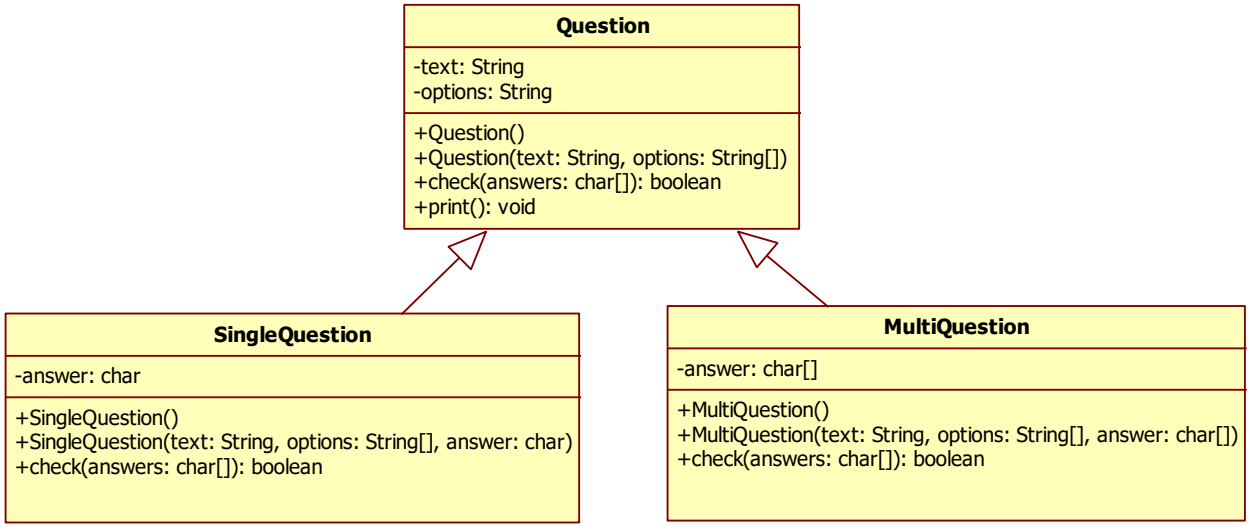
	public	protected	默认	private	static	final
类	√		√			√
数据成员	√	√	√	√	√	√
方法成员	√	√	√	√	√	√
局部变量						√

- 继承允许根据其他类的实现来定义一个新类，这种生成子类的复用通常被称为白箱复用（Whitebox reuse）。“白箱”是相对可视性而言，在继承方式中父类的内部细节对子类可见，所以称为白箱。
- 组合是类继承之外的另一种复用选择，新的更复杂的功能通过组合对象来获得，这种复用被称为黑箱复用（Blackbox reuse），被组合的对象的内部细节是不可见的，对象只以“黑箱”的形式出现（被组合的对象必须定义了良好的数据访问接口）。

当你在开始设计是，一般应该优先选择使用组合，只有确实必要时才使用继承。

我们的目标是找到或创建某些类，其中，每个类都有具体的用途，而且既不是太大（包含太多功能而难以复用），也不是太小（不添加其他功能就无法使用。）

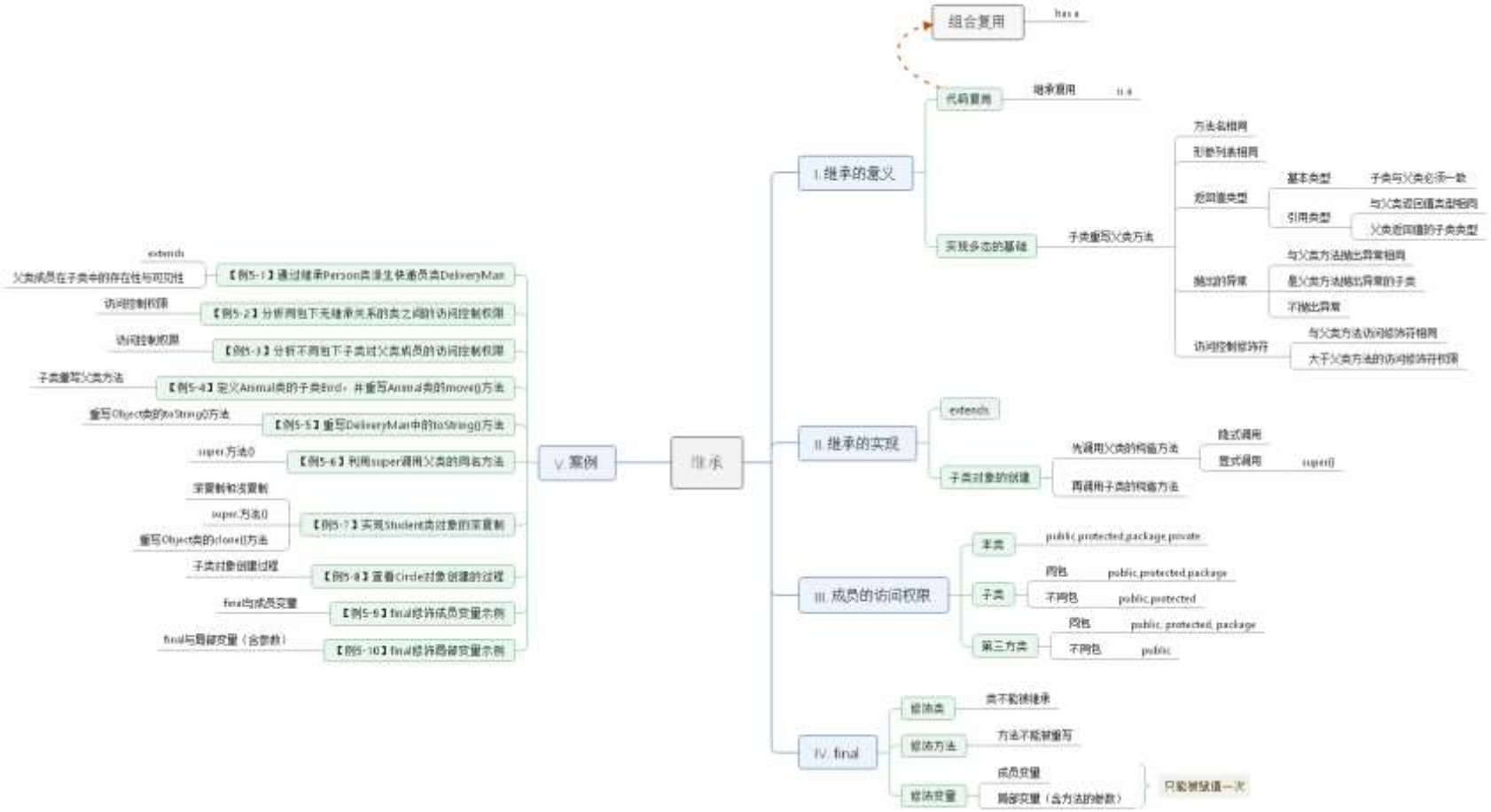
作业



三国演义中的三绝是谁?
A.曹操　　B.刘备　C.关羽　D.诸葛亮
请选择:cad
恭喜, 答对了!

最早向刘备推荐诸葛亮的是谁?
A.徐庶　　B.司马徽　　C.鲁肃　D.关羽
请选择:A
还得努力呀!

本章思维导图


继承
I. 继承的意义
组合复用
has a
代码复用
继承复用
is a
实现多态的基础
子类重写父类方法
方法名相同
形参列表相同
返回值类型
基本类型
子类与父类必须一致
引用类型
与父类返回值类型相同
父类返回值的子类类型
抛出的异常
与父类方法抛出异常相同
是父类方法抛出异常的子类
不抛出异常
访问控制修饰符
与父类方法访问修饰符相同
大于父类方法的访问修饰符权限
II. 继承的实现
extends
子类对象的创建
先调用父类的构造方法
隐式调用
显式调用
super()
再调用子类的构造方法
III. 成员的访问权限
本类
public,protected,package,private
子类
同包
public,protected,package
不同包
public,protected
第三方类
同包
public,protected,package
不同包
public
IV. final
修饰类
类不能被继承
修饰方法
方法不能被重写
修饰变量
成员变量
局部变量（含方法的参数）
只能被赋值一次
V. 案例
extends
父类成员在子类中的存在性与可见性
【例5-1】通过继承Person类派生快递员类DeliveryMan
访问控制权限
【例5-2】分析两包下无继承关系的类之间的访问控制权限
访问控制权限
【例5-3】分析不同包下子类对父类成员的访问控制权限
子类重写父类方法
【例5-4】定义Animal类的子类Bird，并重写Animal类的move()方法
重写Object类的toString()方法
【例5-5】重写DeliveryMan中的toString()方法
super.方法()
【例5-6】利用super调用父类的同名方法
深复制和浅复制
super.方法()
重写Object类的clone()方法
【例5-7】实现Student类对象的深复制
子类对象创建过程
【例5-8】查看Circle对象创建的过程
final与成员变量
【例5-9】final修饰成员变量示例
final与局部变量（含参数）
【例5-10】final修饰局部变量示例