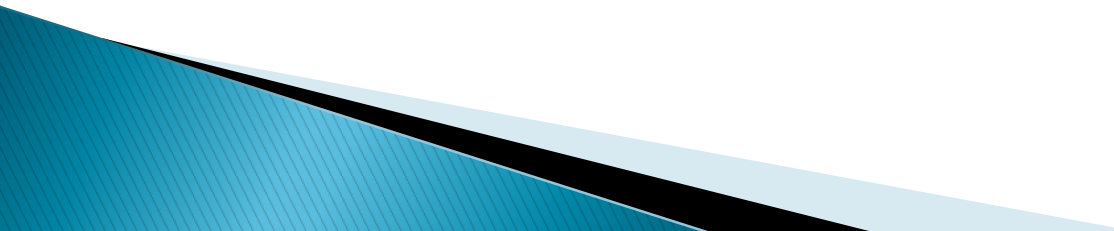


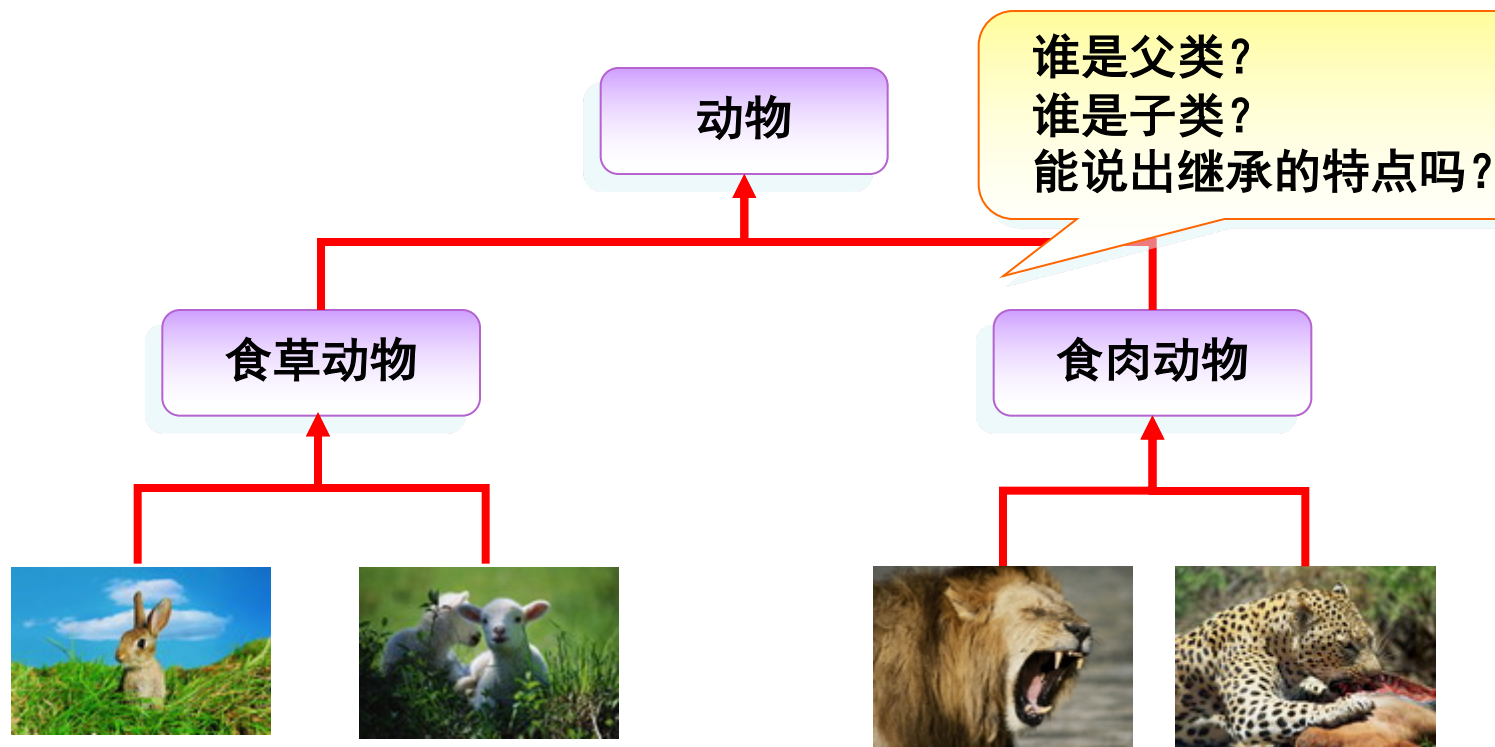
# 第5章 类的继承

# 本章知识点

- ▶ 继承的概念和作用
  - ▶ 类成员修饰符
  - ▶ 继承的实现（子类的声明/super关键字的使用）
  - ▶ super关键字和this关键字的对比
  - ▶ 方法的重写
  - ▶ final修饰符
- 

# 1. 继承的概念

- ▶ 生活中，继承的例子随处可见



继承需要符合的关系：**is-a**，父类更通用、子类更具体

# 1. 继承的概念（为什么—好处）

## ▶ 学生类

特征：学号、姓名、年龄

行为：学习、吃饭、娱乐

## ▶ 教师类

特征：薪水、年龄、姓名

行为：讲课、娱乐、吃饭



# 1. 继承的概念（为什么—好处）

- ▶ 人类

特征：姓名、年龄

行为：吃饭、娱乐

- ▶ 学生类

特征：学号、姓名、年龄

行为：学习、吃饭、娱乐

- ▶ 教师类

特征：薪水、年龄、姓名

行为：讲课、娱乐、吃饭

# 1. 继承的概念（为什么—好处）

- ▶ 人类

特征：姓名、年龄

行为：吃饭、娱乐

- ▶ 学生类

特征：学号

行为：学习

- ▶ 教师类

特征：薪水

行为：讲课

# 1. 继承的概念（为什么—好处）

- ▶ 当多个类之间有相同的特征和行为的时候，就可以将相同的内容提取出来组成新类，让原来的这些类继承新类，从而实现原来的这些类吸收新类中成员的效果，此时在原来类中只需要编写自己独有的成员即可。

- ▶ 继承就是用于提供代码复用性和可维护性以及可扩展性。

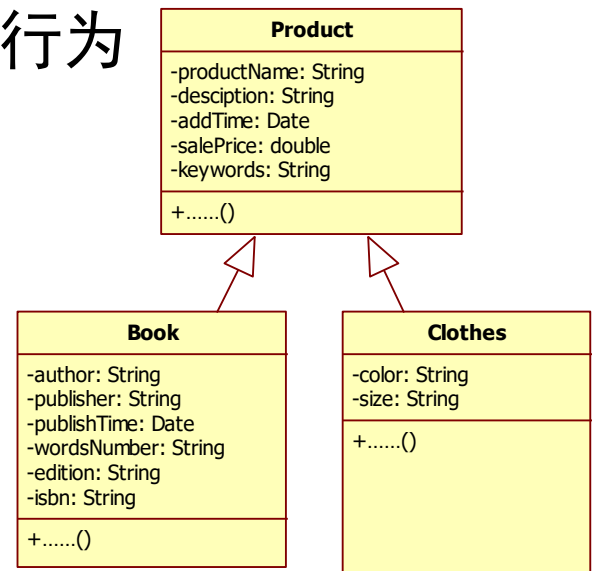
# 1. 继承的概念（为什么—好处）

## ▶ 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



# 1. 继承的概念--（继承的特征）

## ▶ 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，而接口支持多继承**。Java多继承的功能则是通过接口方式来间接实现的。

## 2. 继承的实现

### ▶ 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。

例如：`class Student extends person{ }`

其中person类叫做父类也称为超类或基类parents/super/base。

其中Student类叫做子类也叫派生类child/derived

◦

## 2. 继承的实现

- ▶ 人类

特征：姓名、年龄

行为：吃饭、娱乐

- ▶ 学生类

特征：学号

行为：学习

- ▶ 教师类

特征：薪水

行为：讲课

```
public class Person { //Person.java
    private int age;
    private String name;
    public Person(){
    public Person(String name,int age)
        {setAge(age); setName(name);}
    public int getAge(int age){return age;}
    public void setAge(int age){
    if(age>0 && age<150){ this.age=age; } else{
    System.out.println("年齡不合理"); }}
    public String getName(String name){return name;}
    public void setName(String name){this.name=name ;}

    public void show(){
    System.out.println(name+" "+age);
    }
}
```



```
public class Student extends Person //Student.java
{ private int id;
  public Student(){
    super(); //表示调用父类的无参构造方法
  public Student(String name,int age, int id){
    setAge(age);
    setName(name);
    setId(id); }
  public int getId(int id){ return id; }
  public void setId(int id){ this.id=id; }
}
```

```
public class TestStudent { // TestStudent.java
public static void main(String[] args){

Student s1 = new Student();
s1.show();}
}
```

null 0

```
public class TestStudent { // TestStudent.java
public static void main(String[] args){
Student s1=new Student();
s1.show();

System.out.println("-----
-");

Student s2=new Student("如花",18,1);
s2.show();
}
}
```

null 0

-----  
如花 18

## 需要在子类中重写show()方法

- ◆当子类中没有重写show（）方法时，调用从父类中继承的的show（）方法；
- ◆当子类重写了show（）方法后，则调用子类重写后的show（）方法。

```
public class Student extends Person //Student.java
```

```
{ private int id;
```

```
public Student(){
```

```
super();//表示调用父类的无参构造方法 }
```

```
public Student(String name,int age, int id){
```

```
setAge(age);
```

```
setName(name);
```

```
setId(id); }
```

```
public int getId(int id){ return id; }
```

```
public void setId(int id){ this.id=id; }
```

```
public void show(){
```

```
super.show();
```

```
System.out.println("学号是: "+id);
```

```
}
```

```
}
```

```
public class TestStudent { // TestStudent.java
public static void main(String[] args){
Student s1=new Student();
s1.show();

System.out.println("-----
-");

Student s2=new Student("如花",18,1);
s2.show();
}
}
```

```
null 0
学号是： 0
```

```
-----
如花 18
学号是： 1
```

```
public class Student extends Person //Student.java
```

```
{ private int id;
```

```
public Student(){
```

```
super();//表示调用父类的无参构造方法 }
```

```
public Student(String name,int age, int id){
```

```
super(name,age);
```

```
setId(id); //表示调用父类的有参构造方法 }
```

```
public int getId(int id){ return id; }
```

```
public void setId(int id){ this.id=id; }
```

```
public void show(){
```

```
super.show();
```

```
System.out.println("学号是: "+id);
```

```
}
```

```
}
```

# 继承的主要事项

1. 子类可以继承父类中的成员变量，包括私有成员变量，但不能直接访问；

子类不可以继承父类中的构造方法以及私有的成员方法。

2. 构造子类对象的时候会自动调用父类的无参构造方法，用于初始化从父类中继承下的成员变量信息，相当于在子类构造方法中的第一行增加代码`super()`；

3. 在java语言中只支持单继承，也就是一个类只能有一个父类，但一个父类可以有多个子类；

4. 只有满足：子类 is a 父类的逻辑关系才能继承，不能滥用继承。



## 3.this关键字和super关键字的对比（P99-101）

### ▶ 基本概述

this关键字代表本类对象。

super关键字代表父类对象。

### ▶ 使用方式

使用this.的方式可以访问本类的成员方法和成员变量。

使用super.的方式可以访问父类的成员方法和成员变量

使用this()的方式在构造方法的第一行表示调用本类的构造方法。

使用super()的方式在构造方法的第一行表示调用父类的构造方法。

## 3.this关键字和super关键字的对比（P99-101）

### ▶ 要求大家掌握的用法：

1.使用this.的方式可以区分同名的形参变量和成员变量。

2.使用super(实参)的方式在构造方法的第一行可以调用父类的构造方法。

3.使用super.的方式可以调用父类中被重写的方法。

# 4.重写父类方法（p92--98）

## ◆ 为什么要重写

当父类中继承下来的方法不足以满足子类的需求的时候，就需要在子类中重写一个与父类一模一样的（方法名相同、参数列表相同、返回值类型相同）的方法，叫做方法的重写/覆盖。

# 4. 重写父类方法

## ▶ Dog类

特征：名字、年龄

行为：吃（吃骨头）

## ▶ Cat类

特征：名字、年龄

行为：吃（吃鱼）

## ▶ Tiger类

特征：名字、年龄

行为：吃（吃肉）

## ▶ 动物类

特征：名字、年龄

行为：吃东西、叫

# 4. 重写父类方法

- ▶ Dog类

特征：

行为：吃骨头

- ▶ Cat类

特征：

行为：吃鱼

- ▶ Tiger类

特征：

行为：吃肉

- ▶ 动物类

特征：名字、年龄

行为：吃东西、叫



# Dog.java文件

```
public class Dog extends Animal{  
    public Dog(){super(); //无参构造方法  
}  
    public Dog(String name,int age) {  
        super(name,age); //有参构造方法  
    }  
  
}
```

# Test.java文件

```
public class Test {  
    public static void main(String[] args){  
        Dog dog=new Dog();//创建Dog类的对象  
        dog.show();//调用从父类继承的show()方法  
    }  
}
```

unll 0



# Dog.java文件

```
public class Dog extends Animal{  
    public Dog(){super(); //无参构造方法  
}  
    public Dog(String name,int age) {  
        super(name,age); //有参构造方法  
    }  
    public void show(){  
        show(); //证明super关键字，必须加  
    }  
}
```

# Test.java文件

```
public class Test {  
    public static void main(String[] args){  
        Dog dog=new Dog();//创建Dog类的对象  
        dog.show();//调用show()方法  
    }  
}
```

报错

Exception in thread "main" java.lang.StackOverflowError

# Dog.java文件

```
public class Dog extends Animal{
    public Dog(){super(); //无参构造方法
    }
    public Dog(String name,int age) {
        super(name,age); //有参构造方法
    }
    public void show(){
        show(); //调用父类的方法error, 调用自己的方法进入无限循环状态, 提示栈溢出
    }
}
```

# Dog.java文件

```
public class Dog extends Animal{  
    public Dog(){super(); //无参构造方法  
}  
    public Dog(String name,int age) {  
        super(name,age); //有参构造方法  
    }  
    public void show() { //1.调用父类的show方法  
        super.show();  
    }  
}
```

# Dog.java文件

```
public class Dog extends Animal{
    public Dog(){super(); //无参构造方法
    }
    public Dog(String name,int age) {
        super(name,age); //有参构造方法
    }
    public void show() { //2.调用父类的set方法
        System.out.println(setName()+" "+setAge());
    }
}
```

# Dog.java文件

```
public class Dog extends Animal{  
    public Dog(){super(); //无参构造方法  
}  
    public Dog(String name,int age) {  
        super(name,age); //有参构造方法  
    }  
    public void show2(){ //3.修改子类的方法名称  
        show();  
    }  
}
```

# Test.java文件

```
public class Test {  
    public static void main(String[] args){  
        Dog dog=new Dog();//创建Dog类的对象  
        dog.show();//从父类继承的show()方法  
        Dog dog1=new Dog("金毛",3);  
        dog1.show2();  
    }  
}
```

```
unll 0  
金毛 3
```

# 4. 重写父类方法

- ▶ Dog类

特征：

行为：吃骨头

- ▶ Cat类

特征：

行为：吃鱼

- ▶ Tiger类

特征：

行为：吃肉

- ▶ 动物类

特征：名字、年龄

行为：吃东西、叫



# Animal.java文件

```
public class Animal {  
    private String name;  
    private int age;  
    public Animal(){} //无参构造方法  
    public Animal(String name,int age){ //有参构造方法  
        setName(name);  
        setAge(age);    }  
  
    public void eat(){System.out.println(“动物吃东西”);} //eat()方法  
    public void show(){System.out.println(name+” ”+age);}  
  
    public String getName(){return name;}  
    public void setName(String name){this.name=name;}  
    public int getAge(){return age;}  
    public void setAge(int age){ if(age>0&&age<120)  
        {this.age=age;}  
        else{System.out.println(“年龄不合理”)} }  
}
```

# 4.重写父类方法（ p92--98）

## ◆ 为什么要重写

当父类中继承下来的方法不足以满足子类的需求的时候，测需要在子类中重写一个与父类一模一样的（方法名相同、参数列表相同、返回值类型相同）的方法，叫做方法的重写/覆盖。

# Dog.java文件

```
public class Dog extends Animal{
    public Dog(){super(); //无参构造方法
    }
    public Dog(String name,int age) {
        super(name,age); //有参构造方法
    }
    @override
    public void eat(){System.out.println("狗吃骨头");}
    public void show2(){//3.修改子类的方法名称
        show();
    }
}
```

# Test.java文件

```
public class Test {  
    public static void main(String[] args){  
        Dog dog=new Dog();//创建Dog类的对象  
        dog.show();//从父类继承的show()方法  
        Dog dog1=new Dog("金毛",3);  
        dog1.show2();  
        dog1.eat();  
    }  
}
```

```
unll 0  
金毛 3  
狗吃骨头
```

- ▶ 当子类继承了父类的行为的同时还具有自己的特有行为，也就是说在子类行为中改写的时候还要保留父类的行为，此时用super.的方式调用父类行为。

# Dog.java文件

```
public class Dog extends Animal{
    public Dog(){super(); //无参构造方法
    }
    public Dog(String name,int age) {
        super(name,age); //有参构造方法
    }
    @override
    public void eat(){ super.eat();
        System.out.println("狗吃骨头");}
    public void show2(){//3.修改子类的方法名称
        show(); }
}
```

# Test.java文件

```
public class Test {  
    public static void main(String[] args){  
        Dog dog=new Dog();//创建Dog类的对象  
        dog.show();//从父类继承的show()方法  
        Dog dog1=new Dog("金毛",3);  
        dog1.show2();  
        dog1.eat();  
    }  
}
```

```
null 0  
金毛 3  
动物吃东西  
狗吃骨头
```

# 4. 重写父类方法

- ▶ Dog类

特征：

行为：吃骨头

- ▶ Cat类

特征：

行为：吃鱼

- ▶ Tiger类

特征：

行为：吃肉

- ▶ 动物类

特征：名字、年龄

行为：吃东西、叫



# Cat.java文件

```
public class Cat extends Animal{  
    public Cat(){ super(); //无参构造方法  
    }  
  
    public void eat(){ //eat()方法,重写了Animal  
    类的eat()方法  
        System.out.println("猫吃鱼");  
    }  
}
```

# Tiger.java文件

```
public class Tiger extends Animal {  
  
}
```

1. JVM自动提供类一个无参构造;
2. 构造子类对象的时候会自动调用父类的无参构造方法，用于初始化从父类中继承下的成员变量信息，相当于在子类构造方法中的第一行增加代码super();

# Test.java文件

```
public class Test {
```

```
    public static void main(String[] args){
```

```
        Dog dog=new Dog();//创建Dog类的对象
```

```
        dog.show();//从父类继承的show()方法
```

```
        Dog dog1=new Dog("金毛",3);
```

```
        dog1.show2();
```

```
        dog1.eat();
```

```
        Cat cat=new Cat();//创建Cat类的对象
```

```
        cat.eat();//调用eat()方法
```

```
        Tiger tiger=new Tiger();//创建Tiger对象
```

```
        tiger.eat();//调用eat()方法，此时调用的是父类的eat()方法
```

```
    }
```

```
}
```

```
unll 0  
金毛 3  
狗吃骨头  
猫吃鱼  
动物吃东西
```

# 4.重写父类方法（p92--98）

## ◆ 方法重写的原则

- 1.相同的方法名称，相同的参数列表，相同的返回值类型或者返回子类。
- 2.访问权限不能变小，可以变大。理解：子类的方法所具有的访问修饰符必须跟父类的方法的访问修饰符相同，或更宽的访问修饰符；
- 3.不能抛出更大的异常。

在子类重写的方法中，可以通过super关键字调用父类的“原始”方法。

# Animal.java文件

```
public class Animal {  
    private String name;  
    private int age;  
    public Animal(){} //无参构造方法  
    public Animal(String name,int age){ //有参构造方法  
        setName(name);  
        setAge(age);    }  
  
    public void eat(){System.out.println(“动物吃东西”);} //eat()方法  
    public void show(){System.out.println(name+” ”+age);}  
  
    public String getName(){return name;}  
    public void setName(String name){this.name=name;}  
    public int getAge(){return age;}  
    public void setAge(int age){ if(age>0&&age<120)  
        {this.age=age;}  
        else{System.out.println(“年龄不合理”)} }  
}
```

# Dog.java文件

```
public class Dog extends Animal{
    public Dog(){super(); //无参构造方法
    }
    public Dog(String name,int age) {
        super(name,age); //有参构造方法
    }
    @override
    public void eat(){System.out.println("狗吃骨头");}
    public void show2(){//3.修改子类的方法名称
        show();
    }
}
```

# 5.final修饰符(P101~103)

final本意为“最终的”，“无法更改的”。可以修饰类、成员方法以及成员变量。

final修饰类	最终类
final修饰方法	最终方法
final修饰变量	最终变量

## 5.final修饰符(P101~103)

**A.java**

```
public class A {  
  
}
```

**SubA.java**

```
public class SubA extends A {  
  
}
```




# 5.final修饰符(P101~103)

A.java

```
public final class A {  
  
}
```

SubA.java

```
 public class SubA extends A {  
  
}
```

SubA 不能成为终态类 A 的子类

# 5.final修饰符(P101~103)

final本意为“最终的”，“无法更改的”。可以修饰类、成员方法以及成员变量。

final修饰类	最终类	表示该类不能被继承
final修饰方法	最终方法	
final修饰变量	最终变量	

## 5.final修饰符(P101~103)

**A.java**

```
public class A {  
    public void show(){  
        System.out.println("A类中的show()方法");  
    }  
}
```

**SubA.java**

```
public class SubA extends A {  
    public void show(){//子类可以重写父类方法  
        System.out.println("SubA类中的show()方法");  
    }  
}
```

## 5.final修饰符(P101~103)

A.java

```
public class A {  
    public final void show(){  
        System.out.println("A类中的show()方法");  
    }  
}
```

SubA.java

```
public class Sub extends A {  
    ❌ public void show(){//子类可以重写父类方法  
        System.out.println("SubA类中的show()方法");  
    }  
}
```

不能覆盖 A 中的终态方法

# 5.final修饰符(P101~103)

final本意为“最终的”，“无法更改的”。可以修饰类、成员方法以及成员变量。

final修饰类	最终类	表示该类不能被继承
final修饰方法	最终方法	表示该方法不能被改写
final此时变量	最终变量	

## 5.final修饰符(P101~103)

**TestFinal.java**

```
public class TestFinal {  
    private int count=0;  
    public static void main(String[] args) {  
        TestFinal tf=new TestFinal();  
        System.out.println(tf.count);  
    }  
}
```

0

## 5.final修饰符(P101~103)

TestFinal.java

```
public class TestFinal {  
    private int count=0;  
    public static void main(String[] args) {  
        TestFinal tf=new TestFinal();  
        System.out.println(tf.count);  
        tf.count=20;  
        System.out.println(tf.count);  
    }  
}
```

0

20

# 5.final修饰符(P101~103)

TestFinal.java

```
public class TestFinal {  
    private final int count=0;  
    public static void main(String[] args) {  
        TestFinal tf=new TestFinal();  
        System.out.println(tf.count);  
        ❌ tf.count=20;  
        System.out.println(tf.count);  
    }  
}
```

不能对终态字段 TestFinal.count 赋值



# 5.final修饰符(P101~103)

final本意为“最终的”，“无法更改的”。可以修饰类、成员方法以及成员变量。

final修饰类	最终类	表示该类不能被继承
final修饰方法	最终方法	表示该方法不能被改写
Final成员变量	最终变量	表示必须制定初始值而且不能被更改

# 5.final修饰符(P101~103)

final本意为“最终的”，“无法更改的”。可以修饰类、成员方法以及成员变量。

Final使用方式

final修饰类表示该类不能被继承

--通常用于防止滥用继承

final修饰方法表示该方法不能被改写

--通常用于防止不经意间造成的重写

Final成员变量表示必须制定初始值而且不能被更改

--通常用于描述常量的数据

# 5.final修饰符(P101~103)

## ▶ 补充:

在java语言中，很少单独使用static关键字还有final关键字，通常使用public、static、final 共同修饰成员变量来表示常量的概念。

例如：`public static final PI=3.14;`

通常用于不被修改的，例如：个人账号。

## 5.2.1 重写及其意义

【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 static void main(String[] args){
        Bird bird = new Bird();
        bird.move(); //输出"我可以在天空飞翔....."
    }
}
```

## 5.2.1 重写及其意义

- ▶ 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 $\leq$ 父类方法返回值类型
    - 子类方法抛出的异常 $\leq$ 父类方法抛出的异常
  - “一大”：子类方法的访问权限 $\geq$ 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。

## 5.1.3 类成员的访问控制（已讲）

- ▶ 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。
- ▶ 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。

## 5.1.3 类成员的访问控制（已讲）

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	✓	×	×	×	×

## 5.1.3 类成员的访问控制（已讲）

【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());  
    }  
}
```



## 5.1.3 类成员的访问控制（已讲）

【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);  
    }  
}
```

## 5.2.2 Object类与重写toString()方法

- ▶ Object类中定义了9个方法
  - clone()
  - finalize()
  - toString()
  - equals()
  - hashCode()
  - notify()
  - notifyAll()
  - wait()
  - getClass()

## 5.2.2 Object类与重写toString()方法

【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];
    }
}
```

## 5.2.2 Object类与重写toString()方法

【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

【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

【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。

## 5.2.4 Object类的clone()方法与对象的复制

```
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

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

```
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

```
public class Student {  
    private String name;  
    private int age;  
    public Student() {}  
    public Student(String
```

- 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

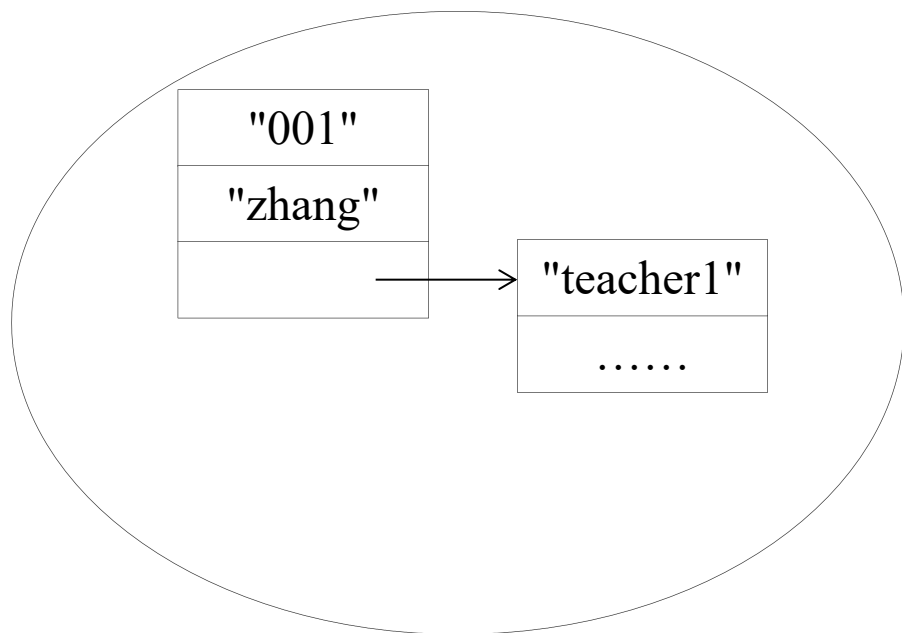
```
public class Student implements Cloneable{ // implements
Cloneable表示实现Cloneable接口
    private String name;
    private int age;
    .....
    public Object clone(){ //重写Object类的clone()方法
        Student stu = null;
        try {
            stu = (Student)super.clone(); //调用Object
类的clone()功能完成复制
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return stu;
    }
}
```

## 5.2.4 Object类的clone()方法与对象的复制

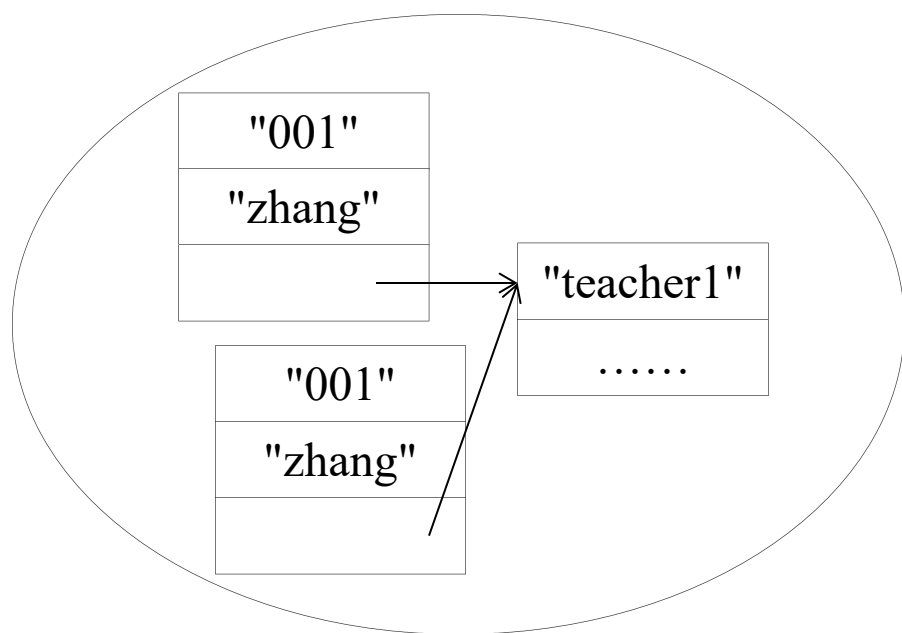
```
public class Test {  
    public static void main(String[] args){  
        Student stu1 = new Student("Lucy",15);  
        Student stu2 = (Student)stu1.clone();  
        System.out.println(stu1);  
        System.out.println(stu2);  
    }  
}
```

## 5.2.4 Object类的clone()方法与深、浅复制

### 浅复制



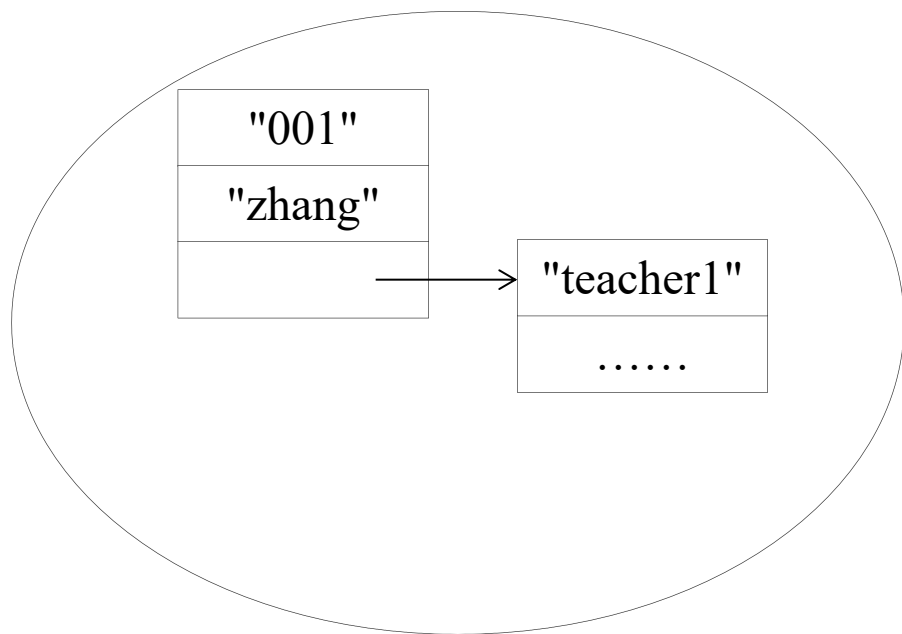
堆内存



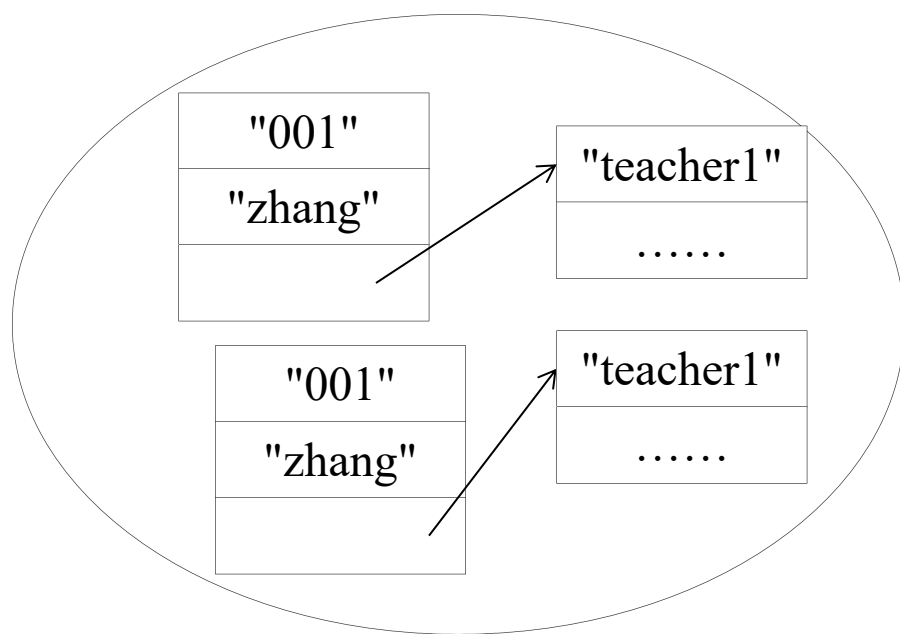
堆内存

## 5.2.4 Object类的clone()方法与深、浅复制

### 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

请思考下面代码的运行结果是什么？

```
public class Test {  
    public static void main(String[] args){  
        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

▶ 代码改进如下：

```
public class Teacher implements Cloneable{
    .....
    public Object clone(){//重写clone方法
        Teacher tea = null;
        try {
            tea = (Teacher)super.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return tea;
    }
}
```



# 深复制

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

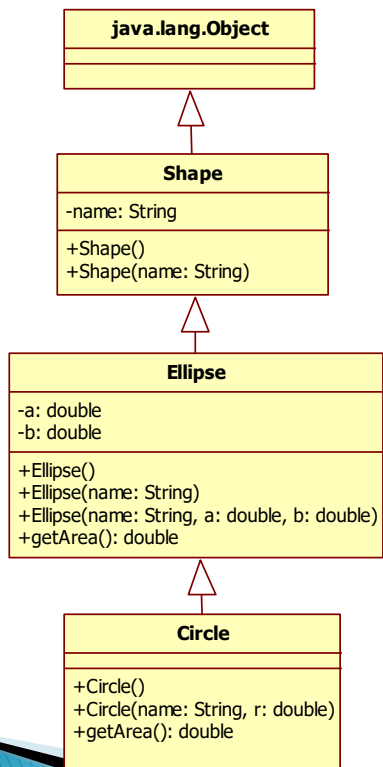
## 5.3 子类对象的构造

### ▶ 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()...");
    }
    .....
}

public class Ellipse extends Shape{
    private double a; //短轴
    private double b; //长轴
    public Ellipse() {
        System.out.println("Ellipse()...");
    }
    .....
}

public class Circle extends Ellipse{
    public Circle() {
        System.out.println("Cicle()...");
    }
}
```

# 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。



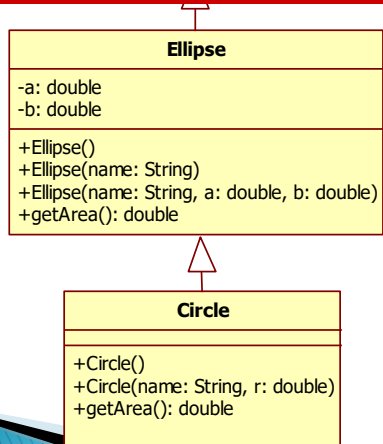
```
public class Shape {
    private String name;
}

public class Ellipse extends Shape {
    private double a; //短轴
    private double b; //长轴
    public Ellipse() {
        System.out.println("Ellipse()...");
    }
}

.....

public class Circle extends Ellipse {
    public Circle() {
        System.out.println("Cicle()...");
    }
}
```

**为了避免错误，父类中至少定义一个无参的构造方法。**



## 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。

## 5.3.2 super与this调用构造方法

### ▶ 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 Java修饰符

## ▶ 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

# 5.4.1 final修饰符

## 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。

## 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。

## 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。

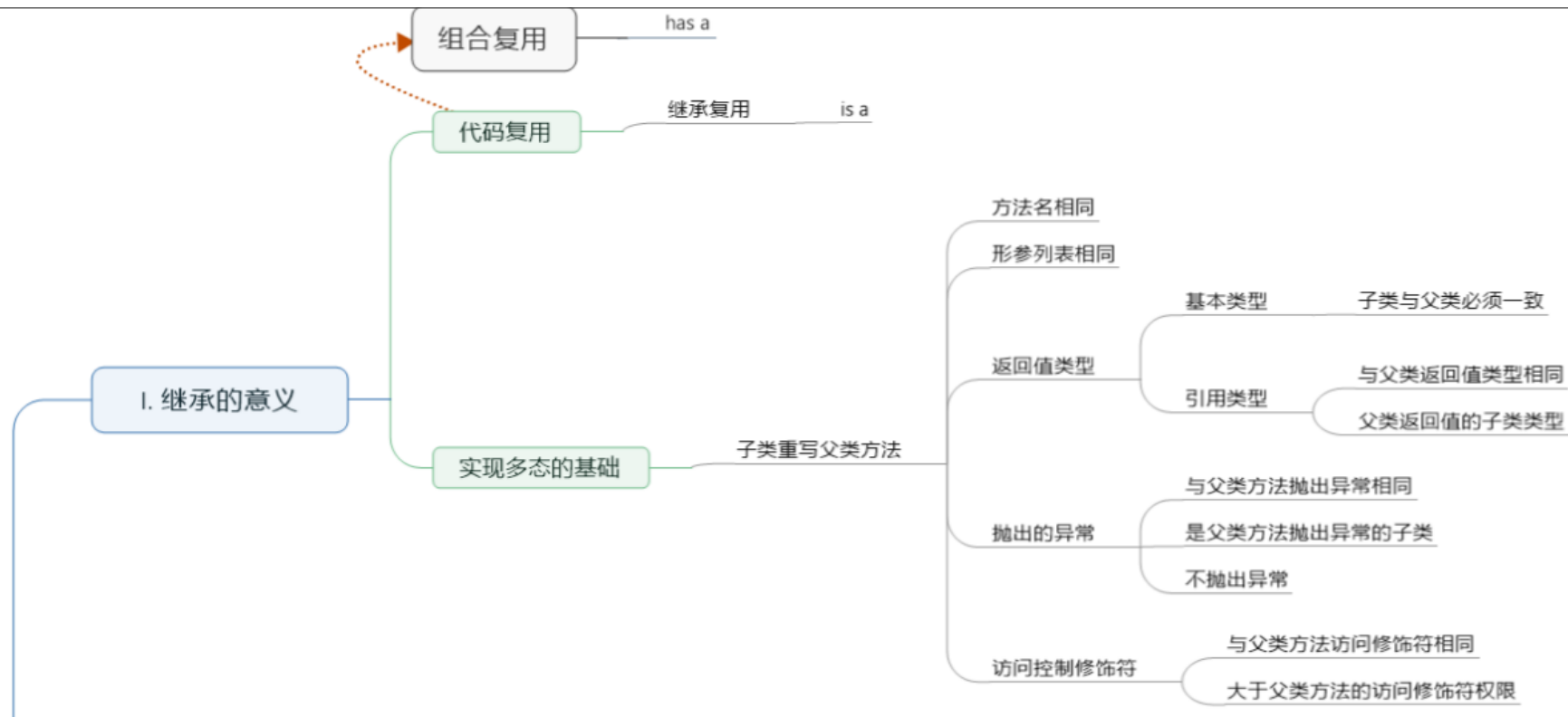
- final成员变量
- final局部变量
- final方法参数



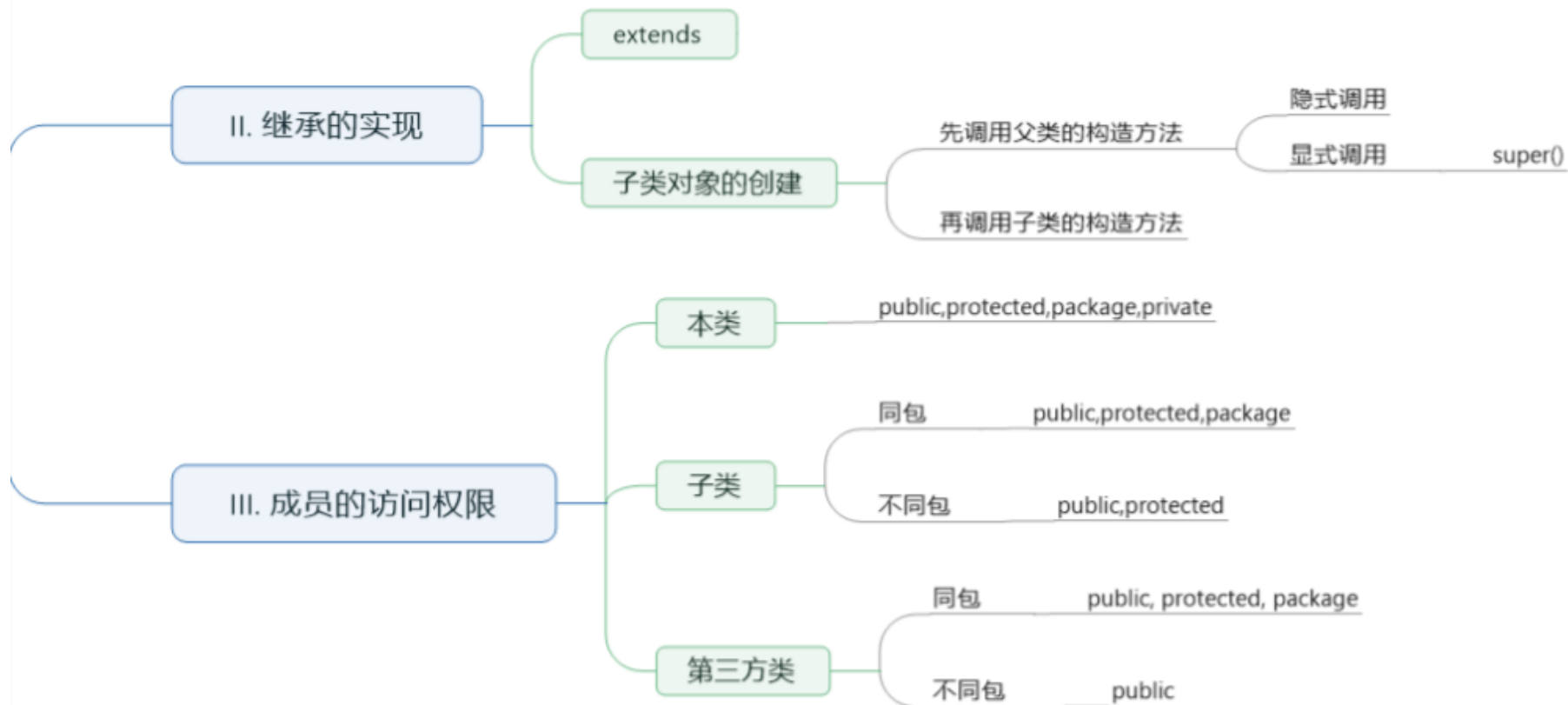
## 5.4.2 Java修饰符之间的关系

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

# 本章思维导图



# 本章思维导图



# 本章思维导图



# 本章思维导图



# Java toUpperCase() 方法

作用：toUpperCase() 方法用于将小写字符转换为大写。

语法：char toUpperCase(char ch)

**参数ch** -- 要转换的字符;**返回值**:返回转换后字符的大写形式，如果有的话；否则返回字符本身。

实例:

```
public class Test {    public static void main(String ar
    gs[]) { System.out.println(Character.toUpperCase('a
    ')); System.out.println(Character.toUpperCase('A'))
    ;    } }
```

以上程序执行结果为：A

A