



主讲教师 张 智
计算机学院软件工程系
课程群: 421694618

6 继承和多态

——> 面向对象的三大核心特性：封装、**继承**、**多态**

6.1 继承

6.2 方法重写

6.3 super关键字

6.4 多态

6.1 继承

继承的主要目的是代码重用

```
public class Animal {  
    public String name;  
    public int age;  
}
```

```
public String getName() {  
    return name;  
}
```

```
public String getInfo() {  
    return "我叫" + name + ",今年" + age + "岁";  
}
```

```
}
```

```
public class Cat {  
    public String name;  
    public int age;
```

```
    public String hobby;
```

新属性

```
    public String getName() {  
        return name;  
    }
```

新方法

```
    public String getHobby() {  
        return hobby;  
    }
```

```
    public String getInfo() {  
        return "我叫" + name + ",今年" + age + "岁" + ",我爱吃" + hobby;  
    }
```

```
}
```

相同

相同

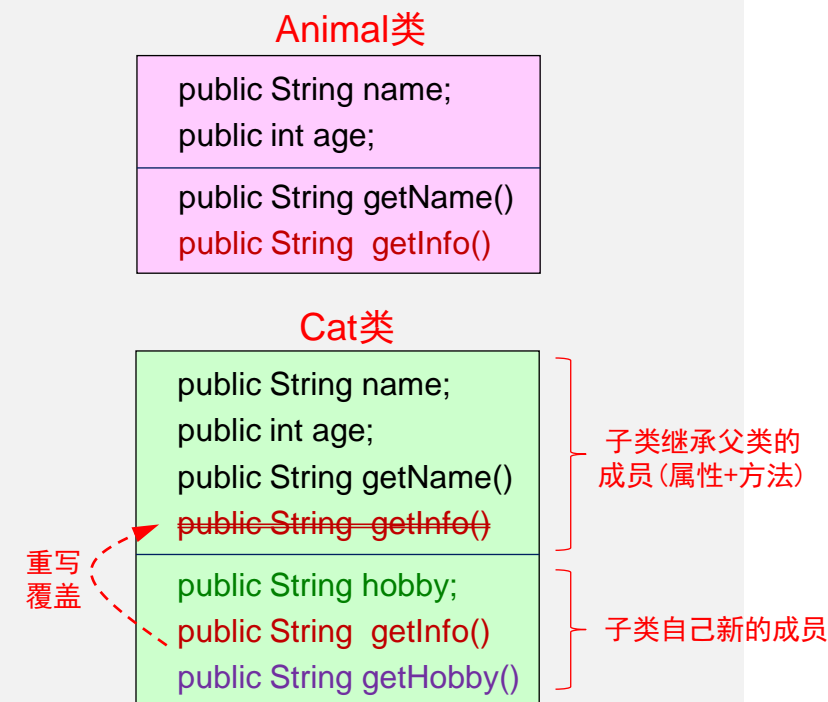
相似

关键字extends来实现继承

子类(派生类) 继承 父类(基类)

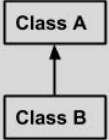
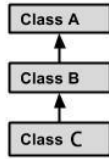
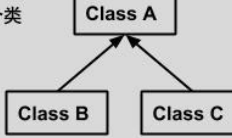
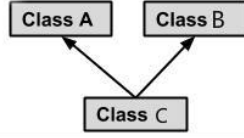
```
public class Cat extends Animal {  
    public String hobby;           //新属性  
  
    @override  
    public String getInfo() {      //重写的方法  
        return "我叫" + name + ",今年" + age + "岁" + ",我爱吃" + hobby;  
    }  
  
    public String getHobby() {     //新方法  
        return hobby;  
    }  
}
```

- 继承需要符合的关系是：is-a (表示属于关系，如cat is a animal)
- 为了达到更好的封装性，父类属性建议用protected修饰

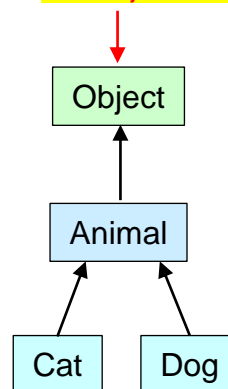


继承的概念

- 在OOP中，允许用现有类来定义新类，新类就叫做原有类的子类(派生类)，原有类称为父类(基类、超类)
- 注意：Java类只支持单继承(一个类只能有一个直接父类)，且都是公有继承

| | | |
|-----------|---|---|
| 单继承 |  | <pre>public class A { } public class B extends A { }</pre> |
| 多重继承 |  | <pre>public class A { } public class B extends A { } public class C extends B { }</pre> |
| 不同类继承同一个类 |  | <pre>public class A { } public class B extends A { } public class C extends A { }</pre> |
| 多继承 (不支持) |  | <pre>public class A { } public class B { } public class C extends A, B { } // Java 不支持多继承</pre> |

注：Object是所有类的父类，即所有类都继承Object(下页说明)



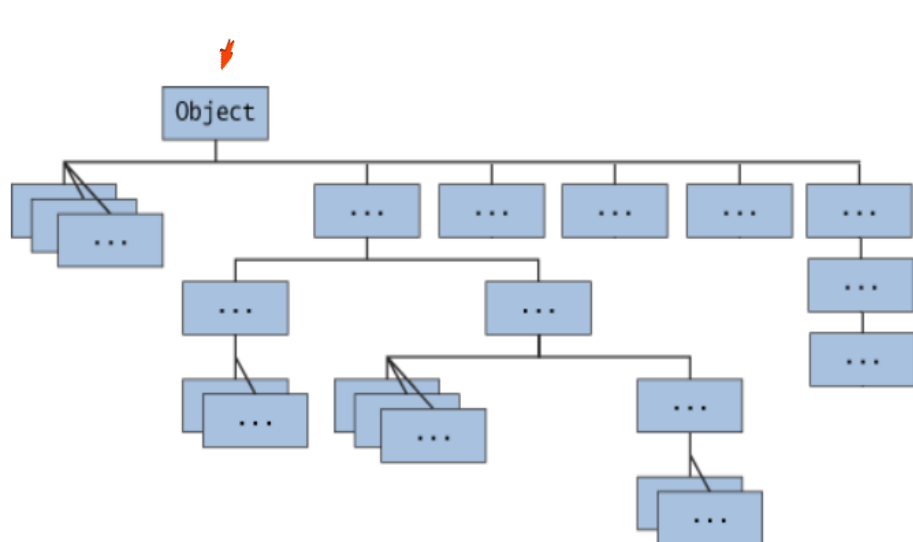
继承需要符合的关系是：is-a
父类更通用，子类更具体

补充词语释义：

- is-a: 表示属于关系，如cat is a animal
- has-a: 表示包含关系，如cat has a head

关于Object类

- `java.lang.Object`类是所有类的父类，也就是说Java的所有类都继承了Object，子类可以使用Object的所有方法
- 定义类时如未显式指定类的直接父类，则该类默认继承Object类



以下两个类表示的含义是一样的：

```
public class Animal {  
}  
public class Animal extends Object {  
}
```

Object常见方法

| 方法 | 说明 | |
|-------------------------|------------------------------------|--|
| Object clone() | 创建与该对象的类相同的新对象 | |
| boolean equals(Object) | 比较两对象是否相等 | |
| void finalize() | 当垃圾回收器确定不存在对该对象的更多引用时，对象垃圾回收器调用该方法 | |
| 反射编程 → Class getClass() | 返回一个对象运行时的实例类 | |
| int hashCode() | 返回该对象的hash散列码值 | |
| void notify() | 多线程编程 | 唤醒在该对象上等待的某个线程 |
| void notifyAll() | | 唤醒在该对象上等待的所有线程 |
| void wait() | | 让当前线程进入等待状态。直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法 |
| String toString() | 返回对象的字符串表示形式 | |

继承的优缺点

■ 优点：

- 实现代码共享，减少创建类的工作量，使子类可拥有父类方法和属性
- 提高代码维护性和可重用性
- 提高代码的可扩展性，更好的实现父类的方法

■ 缺点：

- 继承是侵入性的。继承拥有父类的属性和方法
- 降低代码灵活性。子类拥有父类的属性和方法后多了些约束
- 增强代码耦合性。当父类被修改时，子类可能需要重构

继承的两个重要问题：

子类：继承现有类 + 扩展

子类除了继承父类属性和操作外(继承不改变访问权限)，还可定义自己特有的属性和操作，也可对父类的操作进行**重写**

重写规则? (6.2节)

注意：尽管子类从父类继承了方法和变量，但它**不继承父类构造函数** ★

强调：子类对象在创建时，不仅要用调用本类的构造函数，而且还要调用父类相应的构造函数，**且父类构造函数首先调用**，子类对象如何调用父类构造函数? (6.3节)

[【返回】](#)

6.2 方法重写 (Override)

- 通过继承可以获得父类的属性和方法。在此基础上，不仅可在子类中增加属性和方法，而且还可修改父类方法。

方法重写3个条件

强调

在子类中创建一个与父类相同名称、相同参数列表、相同返回值类型的方法，只是方法体中的实现不同，以实现不同于父类的功能，这种方式被称为方法重写（方法覆盖）

注：Java 5之前返回值类型必须一样，之后放宽了限制，返回值类型必须小于或者等于父类方法的返回值类型（涉及多态）

方法重写示例

```
public class Animal {  
    protected String name;  
    protected int age;  
    public String getInfo() {  
        return "我叫" + name + ",今年" + age + "岁";  
    }  
}
```

方法
重写

```
public class Cat extends Animal {
```

```
    String hobby;
```

```
    @Override //加个注解更规范
```

```
    public String getInfo() {
```

```
        return "我叫" + name + ",今年" + age + "岁" + ",我爱吃" + hobby;
```

```
    }
```

```
}
```

说明：重写方法上方加 **@Override** 注解
可让编译器帮助检查是否正确覆写

方法重写规则

强调：小于或等于父类方法的返回值类型

- 子类方法的名称、参数表、返回类型必须与与父类被重写的方法相同
- 重写方法的访问权限不能比父类的权限低 (public>protected>default>private) ★
- 重写方法一定不能抛出新的非运行时异常或者比被重写方法声明更加宽泛的非运行时异常 (详见异常处理)

这些规则源自多态性和Java语言必须保证“类型安全”的需要

无效覆盖示例：

原因：子类方法缩小了父类方法的访问权限

```
public class Animal {  
    ...  
    public String getInfo() {  
        return "我叫" + name + ",今年" + age + "岁";  
    }  
}
```

无效覆盖

原因：private < default < protected < public

```
public class Cat extends Animal {  
    ...  
    String getInfo() {  
        return "我叫" + name + ",今年" + age + "岁" + ",我爱吃" + hobby;  
    }  
}
```

- 问题1：去掉父类的public呢？
- 问题2：子类能调用父类被覆盖的方法吗？

方法重写注意点：

- 重写的方法可以使用 @Override 注解来标识
- 父类的成员方法只能被它的子类重写
- 不能继承 private方法、final不能修改 final方法、属于全体实例 static方法不能被重写（下页示例）
- 构造函数不能被重写
构造函数不能继承当然谈不上重写

注：被标记为static或private的方法被自动final，不能被重写 ★

强调

| | |
|---|---|
| <pre>public class Father { public void f1() { System.out.println("f1"); } public final void f2() { System.out.println("f2"); } public static void f3() { System.out.println("f3"); } private void f4() { System.out.println("f4"); } public final void f2(int x) { } }</pre> | <pre>public class Son extends Father { public void f1(){ // OK System.out.println("重写父类f1"); } public void f2(){ // 报错 System.out.println("重写父类f2"); } public void f3(){ // 报错 System.out.println("重写父类f3"); } public static void f3(){ } // OK public void f4(){ // OK System.out.println("f4"); } }</pre> |
|---|---|

Diagram annotations:

- 重写 (Overwrite): Indicated by a red dashed arrow from Father.f1() to Son.f1().
- final方法不能被重写 (final method cannot be overwritten): Indicated by a red dashed arrow with an 'X' from Father.f2() to Son.f2().
- static方法也不能被重写 (static method cannot be overwritten): Indicated by a red dashed arrow with an 'X' from Father.f3() to Son.f3().
- 自定义static方法不属于重写 (Custom static method does not belong to overwrite): Indicated by a red dashed arrow from Father.f3() to Son.f3().
- private不会继承不属于重写 (private will not be inherited, does not belong to overwrite): Indicated by a red dashed arrow from Father.f4() to Son.f4().
- final方法只是不能被重写，但重载还是可以的 (final method is just cannot be overwritten, but overload is still possible): Indicated by a red arrow from Father.f2(int x) to Son.f2().
- static方法不能被重写，但自定义还是可以的 (static method cannot be overwritten, but custom is still possible): Indicated by a red arrow from Father.f3() to Son.f3().
- 添加新方法，不属于覆盖范畴 (Add new method, not in the scope of overwriting): Indicated by a red arrow from Son.f4() to the text.

[【返回】](#)


6.3 关键字super


- 用法1：super可被子类用来引用其直接父类

语法：super.父类成员变量或方法（非private类型）

- 用法2：子类调用直接父类构造函数

语法：super([参数列表])

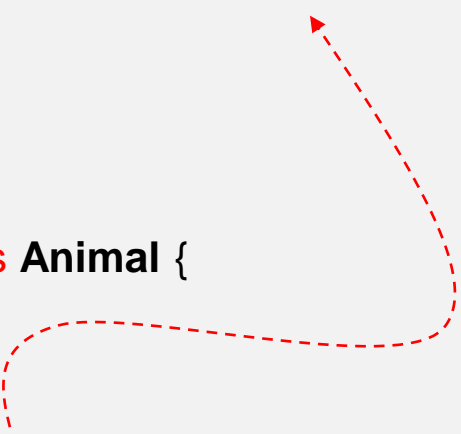
- 
- super() 仅在子类的构造方法中使用
 - super() 必须是第一个可执行语句



思考一下：super()和this()能同时出现吗？（不能）

1、用super调用直接父类成员

```
public class Animal {  
    ...  
    public String getInfo() {  
        return "我叫" + name + ",今年" + age + "岁";  
    }  
}  
  
public class Cat extends Animal {  
    ...  
    String getInfo() {  
        return super.getInfo() + ",我爱吃" + hobby;  
    }  
}
```



super.父类成员

2、用super调用直接父类构造函数

```
public class Animal {  
    protected String name;  
    protected int age;  
  
    public Animal(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public String getInfo() {  
        return "我叫" + name + ",今年" + age + "岁";  
    }  
}
```

父类构造函数

```
public class Cat extends Animal {  
    private String hobby;  
  
    public Cat(String name, int age, String hobby) {  
        super(name, age);  
        this.hobby = hobby;  
    }  
  
    @Override  
    public String getInfo() {  
        return super.getInfo() + ",我爱吃" + hobby;  
    }  
}
```

子类构造函数

调用直接父类构造函数

提醒：子类不继承父类构造函数

测试类

```
public class Test {  
    public static void main(String args[]) {  
        Animal animal = new Animal("某动物", 10);  
        System.out.println( animal.getInfo() );  
  
        Cat cat = new Cat("大橘", 3, "小鱼干");  
        System.out.println( cat.getInfo() );  
    }  
}
```

运行结果

我叫某动物, 今年10岁

我叫大橘, 今年3岁, 我爱吃小鱼干

super衍生的问题

```
public class Animal {  
    protected String name;  
    protected int age;  
  
    public Animal(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public String getInfo() {  
        return "我叫" + name + ",今年" + age + "岁";  
    }  
}
```

父类构造函数

```
public class Cat extends Animal {  
    private String hobby;  
  
    public Cat(String name, int age, String hobby) {  
        super.name = name;  
        super.age = age;  
        this.hobby = hobby;  
    }  
  
    @Override  
    public String getInfo() {  
        return super.getInfo() + ",我爱吃" + hobby;  
    }  
}
```

子类构造函数

表面上看绕过构造函数

会报错!

换一种初始化方法

super衍生的问题（续）

报错!

```
java: 无法将类 edu.wust.examples.Animal中的构造器 Animal应用到给定类型;  
需要: java.lang.String,int  
找到: 没有参数  
原因: 实际参数列表和形式参数列表长度不同
```



原因解释:

- 子类对象创建和初始化时，需要在子类构造函数中**首先调用父类构造函数**
- 这种调用要么是**显式调用**，要么是**隐式自动调用（默认构造函数）**

问题解决:

```
public class Animal {  
    protected String name;  
    protected int age;  
    public Animal(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
    public Animal() {  
    }  
    public String getInfo() {  
        return "我叫" + name + ",今年" + age + "岁";  
    }  
}
```

② 开启父类默认构造函数

```
public class Cat extends Animal {  
    private String hobby;  
    public Cat(String name, int age, String hobby) {  
        super(); ← ① 调用父类默认构造函数(可省略)  
        super.name = name;  
        super.age = age;  
        this.hobby = hobby;  
    }  
    @Override  
    public String getInfo() {  
        return super.getInfo() + ",我爱吃" + hobby;  
    }  
}
```

子类构造函数

养成编程习惯：自定义构造函数+默认构造都开启

再次强调：

- 子类对象在创建时，不仅要用调用本类的构造函数，而且还要调用父类相应的构造函数，且父类构造函数首先调用。
- 因此，子类构造函数如果不显示地通过super调用父类构造函数，则系统将自动隐式调用默认的父亲构造函数（带0个参数的）。
- 在这种情况下，如果没有缺省的父类构造函数，将导致编译错误。

继承编程示例：

■ 父类 Box:

- 数据成员: length、width、height (均为double类型)
- 3个构造函数:
 - Box(double,double,double);
 - Box(double);
 - Box();
- 方法: 求体积-getVolume(), toString()

■ 子类 WeightBox:

- 新增属性: 重量weight (double类型)
- 2个构造函数:
 - WeightBox()和WeightBox (double,double,double,double)
- 方法: 求单位体积重量-getWeightPerVolume(), toString()

Box.java

```

public class Box {
    protected double length;
    protected double width;
    protected double height;

    public Box(double length, double width, double height) {
        this.length = length;
        this.width = width;
        this.height = height;
    }

    public Box(double a) {
        this(a, a, a);
    }

    public Box() {
    }

    //求体积
    public double getVolume() {
        return length * width * height;
    }

    @Override
    public String toString() {
        return "length=" + length + ", width=" + width + ", height=" + height;
    }
}

```

WeightBox.java

```

public class WeightBox extends Box {
    private double weight;

    public WeightBox() {
    }

    public WeightBox(double length, double width, double height, double weight) {
        super(length, width, height);
        this.weight = weight;
    }

    //求单位体积
    public double getWeightPerVolume() {
        if( super.getVolume() > 0 ){
            return weight / super.getVolume();
        }
        else {
            return 0;
        }
    }

    @Override
    public String toString() {
        return super.toString() + ", weight=" + weight;
    }
}

```

测试类

```
public class Test {  
    public static void main(String args[]) {  
        Box box = new Box(3,4,5);  
        System.out.println( box );  
        System.out.println( box.getVolume() );  
  
        WeightBox wbox = new WeightBox(3,4,5,100);  
        System.out.println( wbox );  
        System.out.println( wbox.getWeightPerVolume() );  
    }  
}
```

运行结果

```
length=3.0, width=4.0, height=5.0  
60.0  
length=3.0, width=4.0, height=5.0, weight=100.0  
1.6666666666666667
```

[【返回】](#)

6.4 多态 (Polymorphic)

引例: **B extends A** 以下定义哪些成立?

- | | |
|---|---------------|
| <code>A a = new A();</code> | ✓ |
| <code>B b = new B();</code> | ✓ |
| <code>A a = new B();</code> | ✓ |
| <code>A a = (A)new B();</code> | ✓ 向上转型(和上句等价) |
| <code>B b = new A();</code> | ✗ 编译不通过 |
| <code>B b = (B)new A();</code> | ✗ 编译通过, 运行时错 |
| <code>A a = new B(); B b = (B)a;</code> | ✓ 向下转型(有前提) |

程序测试

```

public class A {
}
class B extends A {
}
class Test {
    public static void main(String[] args) {
        A a = new B();           // OK
        A a = (A) new B();       // OK 两者等价
        B b = new A();           // NO, 编译不通过
        B b = (B) new A();       // NO, 编译通过, 运行时报错 ClassCastException

        A a = new B();           //前提
        B b = (B) a;             //OK, 必须强转
        if ( a instanceof B ) { //安全起见
            B b = (B)a;
        }
    }
}

```

向上转型

向下转型

| | |
|--------|-------------------------------|
| 父=子 | // OK |
| 父=(父)子 | // OK, 向上转型, 二者等价 |
| 子=父 | // 报错 编译报错 |
| 子=(子)父 | // 编译通过, 但运行报错 |
| 父=子 | // 前提 |
| 子=(子)父 | // OK, 向下转型, instanceof 判断更安全 |

多态概念

■ 什么是多态？

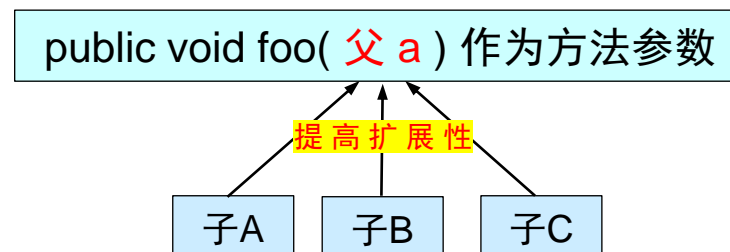
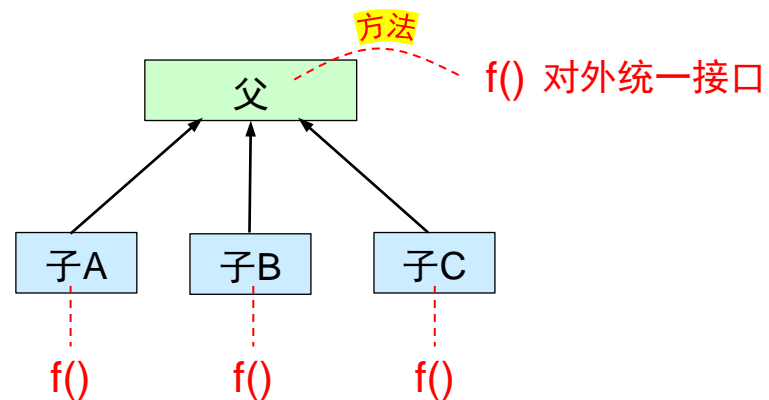
一个对象只有一个类型（是在声明时给它的）。但是，如果该对象能指向其他不同类型的对象(通常是子类)，那么这个对象就是多态性的。

多态主要体现继承过程中的动态调用问题，具体来说，可以用“一个对外接口，多个内在实现方法”表示。



- 作用1：通过统一的接口，实现调用不同的功能
- 作用2：降低模块耦合度，提高程序可扩展性

多态作用图示



父 a = {
new 父()
new 子A()
new 子B()
new 子C()
}

则 a.f() 将具有不同功能

父类引用指向子类对象称为向上转型，即：父 a = ~~(父)~~ new 子();

多态示例

父类：Game

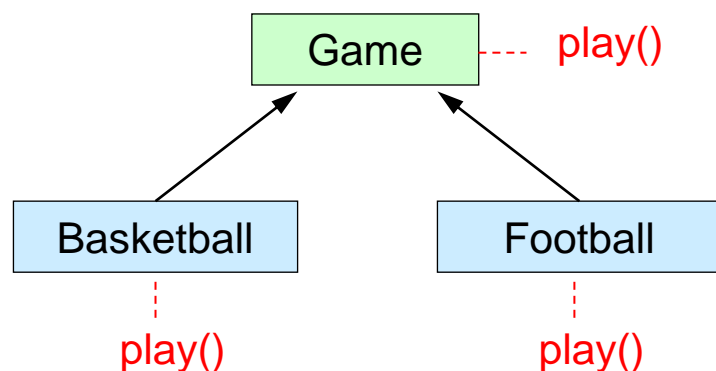
```
public class Game {  
    public void play() {  
        System.out.println("play game");  
    }  
}
```

子类：Basketball 、Football

```
public class Basketball extends Game {  
    @Override  
    public void play() {    //重写方法  
        System.out.println("play basketball");  
    }  
}
```

```
public class Football extends Game {  
    @Override  
    public void play() {    //重写方法  
        System.out.println("play football");  
    }  
}
```


测试类：



Game g = {
 New Game()
 New Basketball()
 New Football()
}

则 g.play()具有不同的功能

```
public class Test {  
    public static void main(String args[]) {  
        Game g = new Game();  
        // Game g = new Basketball();  
        // Game g = new Football();  
        g.play();  
    }  
}
```

play game
play basketball
play football

程序阅读

```
class Father {  
    public void f1() {  
        f2();  
    }  
    public void f2() {  
        System.out.println("AAA");  
    }  
}
```

```
class Child extends Father {  
    public void f1( int i ) {  
        System.out.println("BBB");  
    }  
}
```

```
public void f2() {  
    System.out.println("CCC");  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Father fa = new Child();  
        fa.f1();  
    }  
}
```

运行结果

CCC

多态实现条件

■ Java 实现多态有 3 个必要条件：

- (1) **继承**：在多态中必须存在有继承关系的子类 and 父类
- (2) **重写**：子类对父类方法进行重新定义，在调用这些方法时会调用子类的方法
- (3) **向上转型**：父类引用指向子类对象

两种多态性

■ 编译时多态（静态多态）

- 主要是指方法的重载(overload)，程序在编译时就会根据参数列表的不同来区分不同的方法，在运行时谈不上多态。

■ 运行时多态（动态多态）

- 是指程序在运行时才能动态确定操作所指的对象，通过动态绑定来实现调用不同方法(override)，这是通常所说的多态性。

多态下子类特有方法的访问问题

问题提出：

Son是Father的一个子类

```
Father fa = new Son();    // fa具有多态性
```

```
fa.sonMethod();           // fa能不能调用子类特有的方法？
```

说明：

- fa 是一个Father类型 (声明时定义的), 而不是一个Son类型
- 因此, 通过 fa 能访问的部分只是与Father类相关的部分, 而Son子类的部分是隐藏的不可访问的

程序测试

```
public class A {  
    public void f1(){  
        System.out.println("A");  
    }  
  
    public void foo(){  
        System.out.println("foo");  
    }  
}
```

```
class B extends A {  
    @Override  
    public void f1() { ← 重写的方法  
        System.out.println("B");  
    }  
  
    public void f2(){ ← 子类新方法  
        System.out.println("f2");  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        A a = new B();  
        a.f1();    //多态 输出 "B"  
        a.foo();   // OK 父类方法 输出 "foo"  
        a.f2();    // 报错 无法访问子类特有方法  
    }  
}
```

问题解决：通过**向下转型**来恢复子类对象全部功能

```
Father fa = new Son();           // 前提 父=子
if ( fa instanceof Son ) {       -----安全保证一下-----
    Son s = (Son) fa;             // 向下强转(向下转型)
    s.sonMethod();                // OK
}
```

■ instanceof 关键字用于判断对象是否为一个类(或接口/抽象类/父类)的实例

简写

`((Son) fa).sonMethod();`

前例解决

```
public class Test {
    public static void main(String[] args) {
        A a = new B();
        ((B)a).f2();    // OK 向下转型恢复
    }
}
```

程序阅读

```
class Shape {  
    public String name = "shape";  
    public Shape(){  
        System.out.println("shape constructor");  
    }  
    public void printType() {  
        System.out.println("this is shape");  
    }  
    public static void printName() {  
        System.out.println("shape");  
    }  
}
```

```
class Circle extends Shape {  
    public String name = "circle";  
    public Circle() {  
        System.out.println("circle constructor");  
    }  
}
```

```
    public void printType() {  
        System.out.println("this is circle");  
    }  
    public static void printName() {  
        System.out.println("circle");  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Shape shape = new Circle(); //先调用父类构造函数  
        System.out.println(shape.name); //变量不属于重写概念  
        shape.printType(); //调用子类重写方法  
        shape.printName(); //调用父类静态方法(静态方法没重写)  
    }  
}
```

运行结果

```
shape constructor  
circle constructor  
shape  
this is circle  
shape
```


多态程序阅读注意点

- 多态情况下，子类中存在跟父类同名的成员变量时，访问的是父类的成员变量 (变量不属于重写)
- 多态情况下，子类中存在跟父类同名的非静态方法时，调用的是子类中的方法 (重写的方法)
- 多态情况下，子类中存在跟父类同名的静态方法时，访问的是父类中的方法 (静态方法不能重写)
- 多态情况下，访问不到子类中特有的成员 (需要向下转换)

多态编程示例：

■ 父类Person：

- 成员变量：name(String)，要求为private（不让继承）
- 2个构造函数：Person()、Person(String)
- 成员方法：getter、setter（成员私有了，getter/setter要打开，方便操作）
- equals()方法：return false；具体判断由子类确定

■ 子类Student：

- 新增属性：school(String)，private
- 2个构造函数：Student()、Student(String, String)
- toString()方法：显示name和school信息
- equals()方法：假设当姓名和校名相同认为相等

```

public class Person {
    private String name;
    public Person() {
    }
    public Person(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    //必须写出来，由子类重写
    public boolean equals(Person person) {
        return false;
    }
}

class Student extends Person {
    private String school;

```

```

    public Student() {
    }
    public Student(String name, String school) {
        super(name);
        this.school = school;
    }

```

```

    @Override
    public String toString() {
        return "name=" + getName() + ",school=" + school;
    }

```

这个override是重写来自Object类的toString

继承可以直接用，也可加super或this

```

    @Override
    public boolean equals( Person person ) {
        if ( person instanceof Student ) {
            Student student = (Student) person;
            return this.school.equals( student.school ) &&
                getName().equals( student.getName() );
        }
        return false;
    }
}

```

多态体现扩展性（能不能换成Object？）

此处可以访问私有

这个override是重写
Person类的equals

测试类：

```
public class Test {  
    public static void main(String args[]) {  
        Person p1 = new Student("小明", "武科大");  
        Person p2 = new Student("小明", "武大");  
        Person p3 = new Student("小明", "武科大");  
        System.out.println( p1 );  
        System.out.println( p2 );  
        System.out.println( p3 );  
  
        System.out.println( p1.equals(p2) );    // 输出 false  
        System.out.println( p1.equals(p3) );    // 输出 true  
    }  
}
```

思考：Person类没有定义toString，为何结果显示的是子类的toString？

运行结果

```
name=小明,school=武科大  
name=小明,school=武大  
name=小明,school=武科大  
false  
true
```

【完】