



面向对象程序设计Java

江春华

电子科技大学信息与软件工程学院

内 容

第5章 超类、子类和继承性

1

类的继承性

2

`null`, `this`和`super`

3

方法覆盖与运行时的多态

4

`final`和`abstract`类和方法

5

`Object`类

类的继承性

- ❖ 继承性是面向对象编程技术的最重要的基础概念之一。类的继承性允许创建分等级层次的类。
- ❖ 在继承中，被继承的类叫超类（Super class）或基类（Base class）、父类（Parent class），继承的类叫继承类（Derived class）或子类（Sub class）。

类的继承性

❖ 继承性表现在一个类对另一个类的成员的继承。继承类是对被继承类的扩展，由声明中的关键字 `extends` 指明。

❖ 例：

```
class Student extends Person{  
    ...  
}
```

超类 `Person` 被子类 `Student` 继承或扩展。

类的继承性

❖ 继承性特性如下：

- 单一继承性：子类只能有一个超类，而超类可以有多个子类；
- 子类继承超类的所有成员；
- 子类可以创建自己的成员；
- 子类不能继承超类的构造器，只能在构造器中通过`super()`调用超类的构造器；
- 子类的构造器首先要调用超类的构造器；
- 多态性之一：子类的成员隐藏和覆盖超类中相同的成员；
- 多态性之二：超类的对象可以对子类的实例引用；
- 由`abstract`和`final`修饰的类指示类的是否必须或不能被继承。

类的继承性

❖ 创建子类，格式如下：

```
[abstract|final] class SubCls extends SupCls{  
    SubClassBody  
}
```

- **abstract**是可选项，修饰的类叫抽象类，指示其对象引用的必须是其子类实例；
- **final**是可选项，修饰的类叫终结类，指示其不能被继承，不能有子类；
- **SubCls**是创建的类，称为子类；
- **extends**是关键字，指示两个类存在的继承关系；
- **SuperCls**是**SubCls**类的超类。

类的继承性

- ❖ 在创建类时，若缺省`extends`，则该类就为`Object`类的直接子类。`Object`类是Java语言中所有类的直接或间接超类。`Object`类存放在`java.lang`包中。
- ❖ 子类继承了超类的所有成员，这些成员就象子类创建的成员一样的使用。

类的继承性

- ❖ 虽然子类可以继承超类所有成员，但是因为超类中成员的访问控制，子类无法访问某些受限成员。
- ❖ 在超类中，由`private`修饰的访问权限的成员变量和方法，虽然被子类继承，但是子类不能访问。

类的继承性示例

```
class Person {
    String name;
    char sex;
    Date birthday;
    void setData(String n, char s, Date b){
        name = n;
        sex = s;
        birthday = b;
    }
}

class Student extends Person{
    String stuID;
    String speciality;
    void setData(String n, char s, Date b, String id,
                  String spec){
        setData(n, s, b);
        stuID = id;
        speciality = spec;
    }
}
```

类的继承性示例

❖ **类Person**有成员：

- 成员变量：**name**、**sex**、**birthday**
- 成员方法：**setData (String, char, Date)**

❖ **类Student**有成员：

- 成员变量：**name**、**sex**、**birthday**、**stuID**、**speciality**
- 成员方法：**setData (String, char, Date)**、

setData (String, char, Date, String, String)

其中：**红色标识**的成员是类**Student****继承**超类**Person**的成员。

类的继承性

❖ 成员变量的隐藏:

在子类的创建中，如果出现了与其超类**相同的成员变量**，则超类中的成员变量被子类中的成员变量所隐藏。

❖ 方法覆盖:

在子类的创建中，如果出现与超类中有**相同名、同参数及同返回类型**的成员方法，则超类中的成员方法被子类中的成员方法所覆盖。

类的继承性

```
class SuperClass {  
    int x;  
    void setX(){  
        x = 10;  
    }  
}  
class SubClass extends SuperClass {  
    int x;  
    void setX() {  
        x = 100;  
    }  
    String getMessage() {  
        return "super.x = "+super.x+"\tsub.x = "+x;  
    }  
}
```

类的继承性

```
public class TestCovert{  
    public static void main(String[] args){  
        SubClass sc = new SubClass();  
        sc.setX();  
        System.out.println(sc.getMessage());  
    }  
}
```

结果为:

super.x = 0 sub.x = 100

超类**SuperClass**的成员**x**和**setX()**被子类**SubClass**的成员**x**和**setX()**所**隐藏**和**覆盖**。

类的null, this和super三个量

❖ Java语言中，每个类均有三个量，它们具有特别的含义及用途。

➤ null

➤ this

➤ super

类的null, this和super三个量

❖ null

null表示变量的值为“空”，用于表示对象或数组还没有相应的实例引用。例如：

```
Point pNull = null;
```

❖ this

表示对类的实例访问，它也表示了对象对该实例引用访问。；

- 在类中可以来指向成员变量，以示区别于非成员变量；
- 在构造器中，使用this()形式对另一个构造器的调用；
- 在类的创建中，需要表示对自身的实例访问时，用this表示。

类的null, this和super三个量

❖ super

在子类中，使用super访问被隐藏的超类变量，被覆盖的超类方法。

❖ 使用有三种情况：

- 访问被隐藏的超类成员变量；

例如： **super.varName**

- 调用超类中被覆盖的方法；

例如： **super.methodName ([paramList])**

- 调用超类中的构造方法。

例如： **super ([paramList])**

方法覆盖

❖ 方法覆盖:

- 在子类的创建中，具有与超类中有相同的方法名、相同的参数以及相同的返回数据类型。
- 它还具有比超类中被覆盖的方法可访问性，即权限限制更宽松。

public > protected > 缺省 > private

- 不同于方法重载。方法重载是在一个类中具有相同方法名的方法，它们之间有~~不同的参数~~。

运行时的多态

- ❖ 对于覆盖或继承的方法，Java运行时系统根据调用该方法的实例的类型来决定选择哪个方法调用。
- ❖ 对于类的一个实例，如果子类覆盖了超类的方法，则运行时系统调用子类的方法。
- ❖ 如果子类继承了超类的方法 (未覆盖)，则运行时系统调用超类的方法。

运行时的多态

```
class ClassA{
    void callMe() {
        System.out.println("在ClassA中的callMe() 方法!");
    }
}

class ClassB extends ClassA{
    void callMe() {
        System.out.println("在ClassB中的callMe() 方法!");
    }
}
```

运行时的多态

```
public class TestConvert{  
    public static void main(String arg[]) {  
        ClassA vA = new ClassB();  
        vA.callMe();  
    }  
}
```

程序运行输出的是：

在**ClassB**中的**callMe()**方法！

结果说明了vA调用callMe()是子类ClassB中的方法。

运行时的多态

- ❖ 当超类的对象对子类实例引用时，这个对象所访问的成员必须是超类中所具有的。
- ❖ 这个对象不能访问子类自己创建的成员。
- ❖ 当这个对象访问的是被覆盖的方法，则调用的是子类中覆盖方法。
- ❖ 只有当这个对象被强制转换成子类类型时，这个子类的所有成员才有可能被访问。

final类和方法

- ❖ 由final修饰的类称终结类，不能被继承。由于安全性的原因或者是面向对象设计的考虑，限定一些类不能被继承。
- ❖ final类不能被继承，保证了该类的唯一性。
- ❖ 对于一个类的定义已经很完善，不需再创建它的子类，也可以将其修饰为final类。
- ❖ 格式：

```
final class finalClassName{  
    ...  
}
```

final类和方法

- ❖ 使用修饰符final修饰的变量就像常量一样地使用，称其为常量符号。
- ❖ 以final修饰的方法是不能被子类的方法所覆盖。
- ❖ 其格式为：

```
final returnType methodName ([paraList]) {  
    ...  
}
```

abstract类和方法

- ❖ **抽象类**是指由abstract修饰符声明的类，它的方法中有**未实现**的方法即**抽象方法**。
- ❖ 与final类和final方法相反，abstract类必须**被子类继承**，abstract方法则必须**被子类中的方法覆盖**。

abstract类和方法

- ❖ 当一个类的定义完全表示抽象概念时，它不能够被实例化为一个对象。
- ❖ 抽象类本身存在未实现的方法 (abstract方法)，这些方法不具备实际功能，它只能衍生出子类，抽象方法则由衍生子类时所覆盖。
- ❖ abstract方法必须是在abstract类中，但是abstract类中也可以有非abstract方法。

abstract类和方法

❖ abstract类格式:

```
abstract class abstractClassName{  
    ...  
}
```

❖ abstract方法格式为:

```
abstract returnType methodName([paraList]);
```

- abstract方法是没有语句实现部分，直接由;结束。
- abstract方法必须是在abstract类中，并由其子类的方法覆盖。

abstract类和方法

❖ 在创建抽象方法时，要注意有下面三种方法不能作为抽象方法定义：

- 构造方法
- 类方法
- 私有方法

Object类

❖ Object类

- **Object**类处于Java开发环境的类层次树的根部，处于Java类层的最高层的一个类，是所有类的超类。
- 其它所有的类都直接或间接地为它的子类。
- 该类定义了一些所有对象的最基本的状态和行为，包括与同类对象相比较，转化为字符串等

Object类的方法

- ❖ `clone()` 方法：对象必须支持接口 `Cloneable`。
生成一个类的实例的拷贝，返回值就是这个拷贝的实例。
- ❖ `getClass()` 方法：是 `final` 方法。
返回对象的类的信息，并保存在类 `Class` 的对象中。
- ❖ `equals()` 方法
比较两个对象类是否相同，如果相同则方法返回值 `true`，否则返回值 `false`。
- ❖ `toString()` 方法
以字符串的形式返回当前对象的有关信息。

思考问题

1

面向对象
中的继承关系
是什么含义？

2

子类是如
何继承超类的
成员？构造器
能继承吗？

3

面向对象
程序设计的多
态性是如何体
现的？

第5章作业

本章习题

习题1-10题

1-8题必做

9题选做

10题选做



Q & A

电子科技大学信息与软件工程学院