Java5:超类、子类和继承性

授课教师：邱元杰　电子邮箱：yuanjiq@126.com，微信电话: 13679081552

第5章 超类、子类和继承性

类的继承性

成员变量的隐藏和方法覆盖

null，this和super

运行时的多态

final和abstract

Object类

类的继承

- 继承：类继承另一个类，这个类除了创建自己的成员外，还能够继承或扩展另一个类的成员
- 运用继承，你能够创建一个通用类，它定义了一系列相关项目的一般特性。该类可以被更具体的类继承，每个具体的类都增加一些自己特有的东西。
- 被继承的类叫超类（superclass），继承超类的类叫子类（subclass）

类的继承

- 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

- 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

- 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，但是无法访问某些受限的成员
- 子类中可以访问从超类继承下来的三种访问权限设定的成员
  - public;
  - protected;
  - 缺省。

类的继承

- 父类（超类）

```
class Pe
 public
 protected char sex;
 Date birthday;
 private String address;
 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

- 类Person有成员：
  - 成员变量：name、sex、birthday 、address
  - 成员方法：setData(String,char,Date)
- 类Student有成员：
  - 成员变量：name、sex、birthday、 address、 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

- 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

- 一个更实际的例子：
  - 为了新的功能而编写子类

DemoBoxWeight.java

第5章 超类、子类和继承性

类的继承性

成员变量的隐藏和方法覆盖

null，this和super

运行时的多态

final和abstract

Object类

成员变量的隐藏和方法覆盖

- 在子类的创建中，如果出现了与其超类中相同的成员变量，则超类中的成员变量被子类中的成员变量所隐藏。
- 如果出现了与超类中有相同名、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。

TestCovert.java

方法覆盖与运行时的多态

- 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方法更大的可访问性，即权限限制更宽松。
  - public > protected > 缺省 > private
  - 不同于方法过载。方法过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。

第5章 超类、子类和继承性

类的继承性

成员变量的隐藏和方法覆盖

null，this和super

运行时的多态

final和abstract

Object类

null，this和super

- 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

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。

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：super.varName
  - 调用超类中被覆盖的方法：
    super.methodName([paramList])
  - 调用超类中的构造方法： super([paramList])

何时调用构造函数

- super( )必须是子类构造函数的第一个执行语句，无论你用到了super( )没有，这个次序不变。
- 如果super( )没有被用到，每个超类的默认的或无参数的构造函数将执行。

第5章 超类、子类和继承性

类的继承性

成员变量的隐藏和方法覆盖

null，this和super

运行时的多态

final和abstract

Object类

运行时的多态

- 超类的对象可以引用子类的实例
- 该对象仍然只能够调用超类中定义的方法和变量
- 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。
- 对子类的一个实例，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。

方法覆盖与运行时的多态

```
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!");
        }
}
```

```
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。

方法覆盖与运行时的多态

- 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。

Dispatch.java　　FindAreas.java

TestMethodHidden.java

第5章 超类、子类和继承性

类的继承性

成员变量的隐藏和方法覆盖

null，this和super

运行时的多态

final和abstract

Object类

final关键字

- “这个东西不能改变”
- 考虑到两方面的因素：设计或效率

final 数据

- final type identifier=[初值]

- 许多程序设计语言都有自己的办法告诉编译器某个数据是“常数”。常数主要应用于下述两个方面：

(1) 编译期常数，它永远不会改变

(2) 在运行期初始化的一个值，我们不希望它发生变化

- 对于编译期的常数，编译器（程序）可将常数值“封装”到需要的计算过程里。也就是说，计算可在编译期间提前执行，从而节省运行时的一些开销。

FinalData.java

空白final

- 尽管被声明成final，但却未得到一个初始值。
- 空白final必须在实际使用前得到正确的初始化。
- 空白final 具有最大的灵活性:
  - 位于类内部的一个final 字段现在对每个对象都可以有所不同，同时依然保持其“不变”的本质。

BlankFinal.java

final 参数

- 将参数设成final属性，方法是在参数列表中对它们进行适当的声明。
- 在一个方法的内部，不能改变该final参数。

FinalArguments.java

final类

- 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方法

- 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]){
  ...
}
```

- 可有效地“关闭”动态绑定，或者告诉编译器不需要进行动态绑定。编译器就可为final 方法调用生成效率更高的代码。

abstract类和方法

- 当一个类的定义完全表示抽象概念时，它不需要（也不应该）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

TestAbst.java　　AbstractAreas.java

第5章 超类、子类和继承性

类的继承性

成员变量的隐藏和方法覆盖

null，this和super

运行时的多态

final和abstract

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 类定义的方法及其用途

方法	用途
Object clone( )	创建一个和被复制的对象完全一样的新对象
boolean equals(Object object)	判定对象是否相等
void finalize( )	在一个不常用的对象被使用前调用
Class getClass( )	获取运行时一个对象的类
int hashCode( )	返回调用对象有关的散列值
void notify( )	恢复一个等待调用对象线程的执行
void notifyAll( )	恢复所有等待调用对象线程的执行
String toString( )	返回描述对象的一个字符串
void wait( )	等待另一个线程的执行
void wait(long milliseconds)	
void wait(long milliseconds,	
int nanoseconds)	

思考

- Java语言中类的继承性有什么特点?
- 子类可以继承超类的什么，不能继承的如何在子类中访问?
- 子类对超类的扩展表现在哪些方面？举例说明。
- 方法覆盖是指什么，它与方法过载有什么不同？举例说明。
- 在Java中有哪些是多态性的表现？举例说明。
- 用final和abstract修饰的类各有什么特点?
- 用final和abstract修饰的方法各有什么特点?
- 举例说明Object类的方法equals()的使用情况。