一. OOP中的基本概念

Java的编程语言是面向对象的，采用这种语言进行编程称为面向对象编程(Object-Oriented Programming, OOP)， 它允许

设计者将面向对象设计实现为一个可运行的系统。Java的编程单位是类，对象最后要通过类进行实例化(即“创建”)。

面向对象编程有三个特性：

. 封装：以前我讲过java是以类为基础的，所有的private 属性和方法（实现细节不可见）都是封装在类中的，不像C++在类外还可以定义函数。

隐藏具体的实现,或者不想让其他人知道的内容

. 多态：表面看是多种状态的意思。同一个行为不同人执行起来不相同

ojdbc.jar class.forName("jdbc.orcale 开源

三. 类和对象

面向对象的开发方法把软件系统看成各种对象的集合，对象就是最小的子系统，一组相关的对象能够组合成更复杂的

子系统。面向对象的开发方法将软件系统看成各种对象的集合，接近人的自然思维方式。

对象是对问题领域中事件的抽象。对象具有以下特性：

1) (大需求)万物皆为对象。问题领域中的实体和概念都可以抽象为对象。例如学生，成绩单、教师、课和教室。

2) 每个对象都是惟一的。正如世界上不存在一模一样的叶子。

3) 对象具有属性和行为。

4) 对象具有状态。状态是指某个瞬间对象的各个属性的取值。对象的某些行为会改变对象自身的状态，即属性的取值。

5) 每个对象都是某个类的实例。小张和小王都属于学生类、中国和美国都属于国家类、中文和英文都属于语言类。

类是具有相同属性和行为的对象的集合。

同一个类的所有实例都有相同属性，但属性取值不一定相同，它们的状态不一定相同。例如小张和小王都属

于学生类，都有姓名、性别、年龄、身高和体重这些属性，但是他们的属性取值不同。

同一个类的所有实例都有相同行为，意味着它们具有一些相同的功能。

类是一组具有相同属性和行为对象的模板。面向对象编程的主要任务就是定义对象模型中的各个类。

1) 类是一种类型：是引用类型；

2) 类是元数据： 描述数据的数据，数据在面向对象领域里以对象的形式存在，类是对象共有属性和方法的抽象描述。

Java程序是各种对象相互交互作用、而不是类。举例：

1) 早上到公司上班，在电梯中碰到人我们会说，张总早或王总早，会不会说人早呀！那非得把你抄鱿鱼不可。

2) 我们要看电视，是买台电视机，而不是买制作电视机的模具；

四. 定义方法形式

接下来我们来看一下类中定义的方法的格式。

修饰符 返回类型 方法名（参数列表 可有可无）

异常抛出类型(可有可无) ｛代码块｝

e.g

public void add(int a) throws Exception {

a++;

System.out.println(“a:”+a);

}

1) 必须有返回类型，如果方法没有返回值，必须用void申明返回类型。

2) 构造器没有返回类型

方法中定义的参数我们通常叫做形参，调用有参数的方法时，我们通常会

传递一些实参给方法，那么在java中方法的参数是如何的传递呢？

五. 参数传递

参数传递分为两种：

1) 对于基本数据类型，参数通过值传递。

基本类型(包括string类型)作为参数传递时，传递的是这个值的拷贝。无论你怎么改变这个拷贝，原值是不会改变的

2) 对于类类型，参数通过引用(对象的引用)传递。

对象类型作为参数传递时，传递的是对象的引用，也就是对象的地址

3) 只有引用传递的内容能被改变，而按值传递不会变化。

无论是基本类型作为参数传递，还是对象作为参数传递，实际上传递的都是值，只是值的的形式不用而已。

六. this关键字

1.当成员变量和局部变量重名时，在方法中使用this时，表示的是该方法所在类中的成员变量。（this是当前对象自己）

public class Student{

String name;

int age;

public Student(String name,int a) {

this.name = name;

age = a;

}

}

2.在构造函数中，通过this可以调用同一类中别的构造函数

构建无参构造器,产生多参构造器的结果.

值得注意的是：

1：在构造调用另一个构造函数，调用动作必须置于最起始的位置。

2：不能在构造函数以外的任何函数内调用构造函数。

3：在一个构造函数内只能调用一个构造函数。

七. 数据隐藏

那如何对属性进行隐藏呢？

在前面用private修饰，表示该属性不能被其它类访问和修改，它只能被本类访问和修改，范围限制在本类内。

那前面讲了那么多，怎么还没有讲到OOP的第一个特性封装呢？

封装有两个方面：

其中数据隐藏就是封装的一个方面

pirvate修饰的属性,不可以被其他类访问或者修改.可以通过给该属性提供公开的set和get

方法来进行赋值与取值操作

e.g

private String name;

public:set方法一般都是public修饰,表示可以被其他类使用

void:set方法没有返回值

setName:方法名称 set关键字后面添加需要赋值的属性名称(名称首字母大写）

(String name):参数类型必须与需要添加的属性的类型一致

public void setName(String name) {

this.name = name;

}

public:get方法一般都是public修饰,表示可以被其他类使用

String:表示通过该方法返回得到的数据的类型,该类型必须与属性的类型一致

getName:方法名称 get关键字后面添加需要取值的属性名称(名称首字母大写)

return:关键字 必须有

public String getName(){

return name;

}

八. 封装

另一个方面就是让实现细节不可见(方法(行为)实现的具体细节隐藏。)

那我如何去访问那些private的属性呢？

封装：在属性(实例变量)前加private, 然后通过统一的方法访问以及修改这些属性值的

注意：实现封装的关键在于绝不让其它类访问该类的实例字段。

如何实现封装，实现封装的具体方法？

（1）、修改属性的可见性来限制对属性的访问。

（2）、为每个属性创建一对赋值方法和取值方法，用于对这些属性的访问。

（3）、在赋值和取值方法中，加入对属性的存取的限制。

九. 方法重载

有时侯，类的同一种功能有多种实现方式，换句话说，有很多相同名称的方法，参数不同。这给用户对这种功能的调用使用提供了

很大的灵活性。

对于类的方法(包括从父类中继承的方法), 如果有两个方法的方法名相同，但参数不一致，那么可以说，一个方法是另一个方法的重载方法。这种现象叫重载。

方法名相同，参数不同 称之为重载

重载必须满足以下条件：

1.必须是同一个类

2.方法名（也可以叫函数）一样

3.方法参数个数或参数类型（或参数的类型不是同一种的情况下参数顺序）不同

注：

方法的返回类型以及方法的修饰符都不能称为重载的条件

1.参数顺序指的是参数类型顺序，与参数名字无关，比如

show(int a,String a,int c)

和show(int c,String b,int a)是一样的方法，非方法重载,因为他们的参数类型顺序一样是int，String，int

2.方法重载与访问权限修饰符和方法返回值无关

在一个类中不允许定义两个方法名相同，并且参数签名也完全相同的方法。因为假如存在这样的两个方法，Java虚拟机在运行时就无法

决定到底执行哪个方法。参数签名是指参数的类型、个数和顺序。

具体理解书上的例子。

十. 创建和初始化对象

按照前面讲述的定义类的形式、定义方法的形式构建好类好了之后，程序要真实的运行，还是得通过对象的交互来完成。创建好了类，

只是创建了构建对象的模板。接下来，我们可以通过new操作符，快速地构建出对象。使用new有以下作用：

. 为对象分配内存空间，将对象的实例变量自动初始化为其变量类型的默认值；

. 如实例变量显示初始化，将初始化值赋给实例变量；

. 调用构造方法；

. 返回对象的引用；

十一. 构造方法

1.jvm默认给类添加无参数构造器

2.可以显示构建无参数构造器,如果用户自己构建了构建器（包括无参数或者有参数的）,系统不在构建无参数构造器.此时如果调用无参数构造器会报找不到构造器的错误.

构造方法就是与类同名的那个方法，它的作用是可以用来初始化

class Person //人类{

public Person(String n,int a) //构造方法

{

name = n; age = a;

}

private string name;

private int age;

}

static void main(String[] args){

Person p = new Person("张三",14);//这就是作用

}

new一个对象的时候要用到构造函数,例如Hello hello = new Hello();这时调用的是Hello的无参数构造方法;

Hello hello = new Hello("hi");这个是调用Hello有参数构造方法,在JAVA中如果不写构造方法的话,会默认加上一个无参数的构造方法,

但是如果已经有了一个有参数的构造方法,那么无参数的构造方法就不会默认被加上.

如果Hello类中已经有了一个有参数的构造方法,这时再使用 Hello hello = new Hello();来创建对象的时候就会出错,

这就是为什么书上要强调写了有参数的构造方法就最好加一个无参数的构造方法.

1. 定义：

. 有和类名相同的名字 大小写一致

. 没有返回类型，有返回类型的构造器就变成了普通方法。

2. 调用时刻：

在创建对象的时候调用；

3. 作用：

方法有什么作用，构造方法就有什么作用；只不过构造方法的作用在创建对象的时候生效。一般放置属性初始化代码；

4. 构造方法的作用域：

构造方法只能通过以下方式被调用：

. 当前类的其他构造方法通过this语句调用它；

. 当前类的子类的构造方法通过super语句调用它；

. 在程序中通过new语句调用它；

不能通过使用.方法的手段来调用构造方法

十二. 构造方法重载

class Student {

public Student(){}

public Student(String name){}

}

当通过new语句创建一个对象时，在不同的条件下，对象可能会有不同的初始化行为。例如对于公司新来的一

个雇员，在一开始的时侯，有可能他的姓名和年龄是未知的，也有可能仅仅他的姓名是已知的，也有可能姓名和

年龄都是已知的。如果姓名是未知的，就暂且把姓名设为“无名氏”，如果年龄是未知的，就暂且把年龄设为-1.

可通过重载构造方法来表达对象的多种初始化行为。在一个类的多个构造方法中，可能会出现一些重复操作。为了

提高代码的可重用性，Java语言允许在一个构造方法中，用this语句来调用另一个构造方法。

使用this语句来调用其他构造方法时，必须遵守以下语法规则。

. 假如在一个构造方法中使用了this语句，那么它必须作为构造方法的第一条语句(不考虑注释语句)。

public Employee() {

String name="无名氏"; this(“无名”) this.name = “无名”

this(name); //编译错误，this语句必须作为第一条语句

}

. 只能在一个构造方法中用this语句来调用类的其他构造方法，而不能在实例方法中用this语句来调用类的其他构造

方法；

. 只能用this语句来调用其他构造方法，而不能通过方法名来直接调用构造方法。

public Employee() {

String name="无名氏";

Employee(name); //编译错误，不能用方法名来调用构造方法

}

课堂练习：1) 在student.java类基础上多增加几个构造方法，多个构造方法相互调用。然后相应的修改测

试类的创建方式，编译、执行，通过输出内容观察构造方法调用的先后时刻；

十三. 默认的构造方法

默认构造方法：没有参数的构造方法，可分为两种：

1) 隐含的默认构造方法；

2) 程序显示定义的构造方法；

在Java语言中，每个类至少有一个构造方法。为了保证这一点，如果用户定义类中没有提供任何构造方法，那么Java语

言将自动提供一个隐含的默认构造方法。该构造方法没有参数，用public修饰，而且方法体为空，格式如下：

public ClassName(){} //隐含的默认构造方法

在程序中也可以显示地定义默认构造方法，它可以是任意的访问级别。

如果类中显式定义了一个或多个构造方法，那么Java语言便不再分配隐含的默认构造方法。举例：

public class Sample{

public Sample(int a) {

System.out.println("My Constructor");

}

}

创建Sample类对象的语句:

Sample s1 = new Sample(); //编译出错

Sample s2 = new Sample(1); //合法的

十四. 子类

到这里，基本上讲解了封装，接下来我们谈谈OOP中第二个特性继承。

Object是所有类的父类

1. 通过生活中的例子推出Java中继承;

什么是继承呢？生活中不乏这样的例子，张老头有个儿子张小头，张老头健在的时侯，张小头继承了张老头的坏脾气，国字脸，八字

脚。张老头挂了后，作为子女之一，张小头继承了张老头的财产。张小头只有亲生爸爸张老头，张老头却有包括张小头在内的多个子

女。

Java中类与类之间也有生活中类似的继承关系。在Java类继承关系中，对应于父亲的类叫父类(超类)，对应于儿子的类叫子类。父子类间的

继承关系也叫“is a”关系。这种关系通过类声明上的extends关键字体现。

一个子类只有一个父类，一个父类可有多个子类。

继承是所有OOP语言不可缺少的部分，在java中使用extends关键字来表示继承关系。当创建一个类时，总是在继承，如果没有明确指出要继承的类，就总是隐式地从根类Object进行继承。

　　类Man继承于Person类，这样一来的话，Person类称为父类（基类,超类），Man类称为子类（导出类）。

如果两个类存在继承关系，则子类会自动继承父类的方法和变量，在子类中可以调用父类的方法和变量。

在java中，只允许单继承，也就是说 一个类最多只能显示地继承于一个父类。但是一个类却可以被多个类继承，也就是说一个类可以拥有多个子类。

　　1.子类继承父类的成员变量

　　当子类继承了某个类之后，便可以使用父类中的成员变量，但是并不是完全继承父类的所有成员变量。具体的原则如下：

　　1）能够继承父类的public和protected成员变量；不能够继承父类的private成员变量；

　　2）对于父类的包访问权限成员变量，如果子类和父类在同一个包下，则子类能够继承；否则 需要导入包

　　3）对于子类可以继承的父类成员变量，如果在子类中出现了同名称的成员变量，

super关键字调用父类同名变量,this关键字调用子类同名变量.如果子类中没有出现同名称的成员变量

e.g 父类中有name 子类中没有name

this.name 先找子类中的name,不存在查找父类中name(父类name不是private修饰),如果仍然找不到.编译不通过

父类与子类产生重名的属性或者方法时才出现super于this区别

　　2.子类继承父类的方法

无论子类如何调用构造器,一定会调用父类的构造器(可以是无参或者有参的)

子类构造器中this()调用本类的其他构造方法后,其他构造方法也会调用

父类的构造器.父类构造器不能缺失

　　同样地，子类也并不是完全继承父类的所有方法。

　　1）能够继承父类的public和protected成员方法；不能够继承父类的private成员方法；

　　2）对于父类的包访问权限成员方法，如果子类和父类在同一个包下，则子类能够继承；否则，子类不能够继承；

　　3）对于子类可以继承的父类成员方法，如果在子类中出现了同名称的成员方法，则称为覆盖，即子类的成员方法会覆盖掉父类的同名成员方法。

如果要在子类中访问父类中同名成员方法，需要使用super关键字来进行引用。

　　注意：隐藏和覆盖是不同的。隐藏是针对成员变量，而覆盖是针对普通方法的。（后面会讲到）

3. Object类简略介绍

所有的Java类都直接或间接地继承了java.lang.Object类。Object类是所有Java类的祖先，在这个类中定义了所有的Java对象都具有

相同行为。

十五. 继承

那么继承有哪些细节呢？

1、构造器不能被继承

2、方法和实例变量可以被继承

3、子类构造器隐式地调用父类的默认无参构造器;

4、如果父类中没有定义无参构造器，只定义了有参构造器，那么子类构造器则必须显式地调用父类的有参构造器(通过super(…))，

且必须放置在第一条语句，否则会有语法错误。

super()调用父类的构造方法，必须出现在第一行

this()调用本类的构造方法,必须出现在第一行.

super()和this()不能同时出现.

5、this()和super()在构造器中都必须为第一条语句，两者不能同时出现。

6、当一个子类继承了一个父类后，父类中所有的字段和方法都被子类继承拥有，子类可以任意的支配使用，

每个子类对象中都拥有了父类中的所有字段。当构造一个子类的实例对象时，该对象的实例变量包括

了子类本身以及父类中的所有实例变量，实例方法也包括了子类和父类中的所有实例方法。

子类构造器用来初始化子类中所有的实例变量，而父类构造器super(实参)用来初始化父类中所有的实例变量。

\*\*\*\*\*\*\*\*\*那么在堆中为子类实例对象分配的内存区域中包括了子类和父类中所有初始化后的实例变量。\*\*\*\*\*\*\*\*\*\*\*

十五. 方法覆盖(重写)

重写位于子父类之间,重载位于同一个类之间

子类在重写父类方法时,对于重写的方法的修饰符必须范围大于或者等于父类的方法的

修饰符.如果小于父类的会抛出异常.

修饰符范围子类必须大于或者等于父类,抛出异常范围子类必须小于或者等于父类

1. 方法覆盖只存在于子类和父类(包括直接父类和间接父类)之间。在同一个类中方法只能被重载，不能被覆盖；

2. 静态方法： （第六章讲）

不能覆盖；

a. 父类的静态方法不能被子类覆盖为非静态方法； -> 编译错误

b. 子类可以定义与父类的静态方法同名的静态方法，以便在子类中隐藏父类的静态方法； -> 编译正常

c. 父类的非静态方法不能被子类覆盖为静态方法； -> 编译错误

3. 私有方法： 不能被子类覆盖 -> 编译正常

4. 抽象方法：（第六章讲）abstract

可以覆盖：

a. 父类的抽象方法可以被子类覆盖为非抽象方法： 子类实现父类抽象方法；

b. 父类的抽象方法可以被子类覆盖为抽象方法： 重新声明父类的抽象方法；

c. 父类的非抽象方法可以被子类覆盖为抽象方法；

十六. super关键字

1. 为什么要使用super关键字？

子类中要访问父类中屏蔽的方法或变量。

1) 子类方法中定义和父类成员变量同名变量；

2) 子类写了一个和父类中相同的方法；

3) 子类中写了一个和父类中相同的属性；

2. 使用注意事项：

a. 只能在构造方法或实例方法内使用super关键字，在静态方法和静态代码块内不能使用super关键字this关键字。

b. 在子类构造方法中如没有使用this关键字，会隐式调用父类的无参构造方法；

c. 构造方法中this(...)和super(...)不能同时出现；

十七. 多态

前面我们讲了OOP的两个特性，接下来要学习另外一个重要特性：多态

1) 一个引用变量可以指向多种实际类型的现象。

Circle extends Shape

Recting extends Shape

Shape s1 = new Circle(); eat() s1.eat(){circle}

Shape s2 = new Recting(); s1.eat(){reacting}

2) 有不同的类型

3) 一个对象有一个类型

4) 一个引用变量会有许多类型

5) 多态是出现在具有继承关系的两个类之间，所以它不像方法重载（发生在一个类中）

不在编译期间发生，而是在运行期间发生（确定下来）。

多态的作用：消除类型之间的耦合关系。

现实中，关于多态的例子不胜枚举。

买票操作.A用户是vvip,买票打五折,B用户是vip,买票打7折,C用户

是普通会员,不打折.

同一个买票操作产生多种情况,此时就称之为多态.

下面是多态存在的三个必要条件，要求大家做梦时都能背出来！

类对象调用getClass()［Object中的方法］方法返回Class类型,

在调用getName()返回该Class类型对象的String类型(类或者接口的名称)

多态存在的三个必要条件

一、要有继承；

二、要有重写；

三、父类引用指向子类对象。

如果没有写toString,默认调用引用对象时也是调用toString方法.

此时由于子类没有toString,但是确实在调用toString方法.所以猜测

应该是调用父类的toString.所有的类默认都继承于Object.因此在

object对象中应该有一个toString方法.

引用对象默认调用toString方法,如果没有没有写toString,调用父类

object中的toString方法,该方法返回当前类所在内存地址.建议每个类

都重写该方法.

Grade g = new VIP();

//此时g取决于运行时类型,而不是

//前面的声明类型(编译时类型)

十八. 类型转换

转换：

1) 先使用instanceof 识别类型

2) 子类型隐式地扩展到父类型（自动转换）

3) 父类型必须显式地缩小到子类型

java.lang.ClassCastException

类型转换异常

直接转换异常发生在运行阶段,编译阶段不会抱错.换言之,任何类型转换之前都有必要

进行判断.如果类型相同再执行转换(有大范围的数据类型转换成小范围的数据类型).

如果发现本身就不是同一种类型,就不需要进行转换.

e.g

Employee extends Person

Person范围大 Employee范围小 如果判断employee和person是同一类型

数据,可以将person类型转换成employee类型

判断类型是否相同 可以使用instanceof关键字

instanceof主要处理,当一个类有多个子类.开始用父类指向其中一个子类时,

再后来的使用中忘记到底指向的是哪一个子类时，可以使用该关键字进行判断

避免将真实的构建出来的子类型转换不相关的另一个子类型

类型转换:

基本数据类型:小数据类型转换成大数据类型.默认转换

大数据类型转换成小数据类型 显示转换并且精度可能丢失

引用类型转换:小范围(子类)类型转换成大范围(父类)类型 默认转换

大范围(父类)类型转换成小范围(子类)类型 需要显示转换并且

转换之前进行类型判断(使用instanceof),如果instanceof

返回为true才能转换.否则转换后会抛出类型转换异常

Person p = new Person();

Student s = (Student)p; 编译不会错，运行时错误

Person p2 = new Student();

Student s = (Student)p2 正确

注意：这里可以在com.briup.ch05.StudentTest.java中演示一下类型的强制转换。

子类重写父类方法不能抛出比父类更宽的异常类型

重写规则之一：重写方法不能比被重写方法限制有更严格的访问级别

Person {

setName(){}

}

Employee extends Person {

setName(String name){}

setName()

}

e.g

父类的方法是public修饰,子类方法只能是public修饰.如果private修饰就

抛出异常.

重写规则之二： 参数列表必须与被重写方法的相同

方法名称 方法返回类型 参数个数 参数类型 参数类型顺序

重写规则之三：返回类型必须与被重写方法的返回类型相同

重写规则之四：重写方法不能抛出新的异常或者比被重写方法声明的检查异常更广的检查异常。但是可以抛出更少，更有限或者不抛出异常

重写规则之五： 不能重写被标识为final的方法。

重写规则之六：如果一个方法不能被继承，则不能重写它。如private方法

e.g

父类的构造方法不能被继承,所以父类的构造方法不能被重写.

继承现象总结：

1. 子类重写父类方法，调用子类方法；

2. 子类属性与父类同名(不管子类属性前修饰符如何均允许)，如获取属性，看获取属性方法位置，如在父类中，获取的是父类属性，

如在子类中，获取的是子类属性；

3. 子类私有方法与父类私有方法同名，如调用该方法，看私有方法被调用的位置，如在父类中，调用的是父类方法，

如在子类中，调用的是子类方法；

4. 子类静态方法与父类静态方法同名，子类静态方法屏蔽父类静态方法。如调用该静态方法，看实例化对象时所声明的类型，如声明

为父类，调用的是父类中静态方法，反之是子类中静态方法。