类之间的关系

依赖 users– a

是一种最明显的、最常见的关系。如果一个类的方法操作另一个类的对象，我们就说一个类依赖于另一个类。

尽可能地将相互依赖的类减至最少。如果类A不知道B的存在，它就不会关心B的任何改变（这意味着B的改变不会导致A产生任何bug）。让类之间的耦合度最小。

聚合 has – a

是一种具体且易于理解的关系。一个Order对象包含一些Item对象。聚合关系意味着类A的对象包含类B的对象

继承 is – a

是一种用于表示特殊与一般关系的。一般而言，如果类A扩展类B，类A不但包含从类B继承的方法，还会拥有一些额外的功能。

一个对象变量并没有实际包含一个对象，而仅仅引用一个对象

GregorianCalendar Date类区别

java.util. GregorianCalendar 1.1

GregorianCalendar()

构造一个日历对象，用来表示默认地区、默认时区和当前时间

GregorianCalendar(int year,int month,int day)

GregorianCalendar(int year,int month,int day,int hour,int minutes,int seconds)

用给定的日期和时间构造一个Gregorian日历对象。

参数： year 该日期所在的年份

month 该日期所在的月份，以0位基准，0表示一月

day 该月份中的日期

hour 小时(0到23之间)

minutes 分钟(0到59之间)

seconds 秒(0到59之间)

int get(int field)

返回给定域的值

参数： field 可以是下述选项之一：

void set(int field,int value)  
设置特定域的值

参数 filed get接受的常量之一

value 新值

void set(int year,int month,int day)

void set(int year,int month,int day,int hour,int minutes,int seconds)

将日期域和时间域设置为新值

参数： year 该日期所在的年份

month 该日期所在的月份，以0位基准，0表示一月

day 该月份中的日期

hour 小时(0到23之间)

minutes 分钟(0到59之间)

seconds 秒(0到59之间)

void add(int field,int amount)

这是一个可以对日期信息实施算术运算的方法。对给定的时间域增加指定数量的时间。

例如，可以通过调用c.add(Calendar.DAY\_OF\_MONTH.7)将当前的日历日期加上7

参数： field 需要修改的域(可以使用get方法文档中给出的一个常量)

amount 域被改变数量(可以为负值)

int getFirstDayOfWeek()

获得当前用户所在地区，一个星期中的第一天。例如，在美国一个星期中的第一天是Calendar.SUNDAY.

void setTime(Date time)

将日历设置为指定的时间点

参数：time 时间点

Date getTime()

获得这个日历对象当前值所表达的时间点

java.text.DateFormatSymbols 1.1

String[] getShortWeekdays()

String[] getShortMonths()

String[] getWeekdays()

String[] getMonths()

获得当前地区的星期几或月份的名称。利用Calendar的星期和月份常量作为数组索引值。

构造器与类同名

每个类可以有一个以上的构造器

构造器可以有0个、1个、或多个参数

构造器没有返回值

构造器总是伴随着new操作一起调用

每一个方法中，关键字this表示隐式参数。

位于方法名后面括号中的数值，这是一个显式参数。

封装的优点

一个私有的数据域

一个公有的域访问器方法

一个公有的域更改器方法

可以改变内部实现，除了该类的方法外，不会影响其他代码

更改器方法可以执行错误检查，然而直接对域进行赋值将不会进行这些处理

final实例域

可以将实例域定义为final。

这个域的值被设置，并且在后面的操作中，不能够再对它进行修改。

final修饰符大都应用于基本类型域或不可变类的域（如果类中的每个方法都不会改变其对象，这种类就是不可变类，例如String）

静态域与静态方法

静态域：

如果将域定于为static，每个类中只要有一个这样的域。而每一个对象对于所有的实例域却**都有自己的一份拷贝。共享一个静态域，静态域属于一个类，而不属于任何一个独立的对象**

静态常量

静态变量使用的比较少，静态常量却使用得比较多。

例如 Math中的PI

public static final double PI = 3.14159…

如果static被省略，PI就变成了Math类的一个实例域。需要通过Math类的对象访问PI，并且每一个Math对象都有它自己的一份PI拷贝。

System中的out

public static final PrintWriter out =

由于每个类对象都可以对公有域进行修改，所以，最好不要将域设计为public。然而，公有常量（即final域）却没问题。因为out被声明为final，所以，不允许再将其他打印流赋给它。

静态方法

Math.pow(x,a)

可以认为静态方法是没有this参数的方法（在一个非静态的 方法中，this参数表示这个方法的隐式参数）

因为静态方法不能操作对象，所以不能再静态方法中访问实例域，但是，静态方法可以访问自身类中的静态域。

如果省略了静态方法的static修饰，需要通过类对象的引用调用这个静态方法。

建议使用类名调用静态方法，所以最好加上static修饰。

下面两种情况下使用静态方法：

一个方法不需要访问对象状态，其所需参数都是通过显示参数提供

一个方法只需要访问类的静态域

工厂方法

静态方法还有一种常见得用途。NumberFormat类使用工厂方法产生不同风格的格式对象。

NumberFormat currencyFormatter = NumberFormat.getCurrencyInstance();

NumberFormat precentFormatter = NumberFormat.getPercentInstance();

double x = 0.1;

System.out.println(currencyFormatter.format(x)); // print $0.10

System.out.println(precentFormatter.format(x)); //print 10%

为什么NumberFormat类不利用构造器完成这些操作？

无法命名构造器，构造器的名字必须与类名相同。但是，这里希望将得到的货币实例和百分比实例采用不同的名字

当使用构造器时，无法改变所构造的对象类型。而Factory方法将返回一个DecimalFormat类对象，这是NumberFormat的子类

main方法

需要注意，不需要使用对象调用静态方法。例如，不需要构造Math类对象就可以访问Math.pow

同理main方法也是一个静态方法

main方法不对任何对象进行操作。事实上，在启动程序时还没有任何一个对象。静态的main方法将执行并创建程序所需要的对象。

每个类可以有一个main方法，这是一个常用于对类进行单元测试的技巧。

方法参数

值调用 call by value

表示接收的是调用者提供的值，

引用调用 call by reference

表示方法接收的是调用者提供的变量地址

一个方法可以修改传递引用所对应的变量值，而不能修改传递值调用对应的变量值

Java中总是采用按值调用。也就是说，方法得到的是所有参数值的一个拷贝，特别是，方法不能修改传递给它的任何参数变量的内容。

java程序设计语言对对象采用的不是引用调用，实际上，对象引用进行的是值传递

下面总结一下Java程序设计语言中方法参数的使用情况

一个方法不能修改一个基本数据类型的参数（即数值型和布尔型）

一个方法可以改变一个对象参数的状态

一个方法不能让对象参数引用一个新的对象

对象构造

重载

一个类有多个构造器，这种特征叫重载。

如果多个方法有相同的名字、不同的参数、便产生了重载。编译器必须挑出具体执行哪个方法，它通过用各个方法给出的参数类型与特定方法调用所使用的值类型进行匹配来挑选出相应的方法。这个过程叫做重载解析

Java允许重载任何方法，而不只是构造器方法。

要完整的描述一个方法，需要指出方法名以及参数类型，这叫做方法的签名

默认域初始化

如果在构造器中没有显式地给域赋予初值，会自动赋值，数值为0，布尔值为false，对象引用为null

无参数构造器

很多类都包含一个无参数的构造函数，对象由无参数构造函数创建时，其状态会设置为适当的默认值

如果在编写一个类时没有编写构造器，那么系统就会提供一个无参数构造器，这个构造器将所有的实例域设置为默认值。

**如果类中提供了至少一个构造器，但是没有提供无参数的构造器，现在构造对象时如果没有提供参数就会被视为不合法。**

显式域初始化

由于类的构造器方法可以重载，所以可以采用多种形式设置类的实例域的初始化状态。确保不管怎样调用构造器，每个实例域都可以被设置为一个有意义的初值。

调用另一个构造器

关键字this引用方法的隐式参数

还有一个含义：

如果构造器的第一个语句形如this(…)，这个构造器将调用同一个类的另一个构造器。

初始化块

初始化数据域的方法：

在构造器中设置值

在声明中赋值

实际上，还有第三种机制 初始化块

在一个类的声明中，可以包含多个代码块，只要构造类的对象，这些块就会被执行。

class Emloyee{

private static int nextId;

private int id;

private String name;

private double salary;

//object initialization block

{

id = nextId;

nextId++;

}

}

在这个示例中，无论使用哪个构造器构造对象，id域都在对象初始化块中被初始化。首先运行初始化块，然后才运行构造器的主体部分。

由于初始化数据域有多种途径，所以列出构造过程的所有路径可能相当混乱。下面是调用构造器的具体处理步骤：

1 所有数据域被初始化为默认值（0 false 或 null）

2 按照在类声明中出现的次序，依次执行所有域初始化语句和初始化块

3 如果构造器第一行调用了第二个构造器，则执行第二个构造器主体

4 执行这个构造器的主体

在类第一次加载的时候，将会进行静态域的初始化。static{}

对象析构与finalize方法

由于java有自动的垃圾回收器，不需要人工回收内存，所以java不支持析构器

可以为任何类添加finalize方法，将在垃圾回收器清除对象之前调用。

包

类的导入

一个类可以使用所属包中的所有类，以及其他包中的公有类（public class）。我们可以采用两种方式访问另一个包中的公有类，

第一种方式是在每个类名之前添加完整的包名。

第二种方式使用import语句。import语句是一种引用包含在包中的类的简明描述。

可以使用哦import语句导入一个特定的类或者整个包。import语句应该位于源文件的顶部（但位于package语句的后面）。

只能使用星号\*导入一个包，而不能使用import java.\*或import java.\*.\* 导入以java为前缀的所有包。

java.util 和 java.sql 都有Date类

使用一个Date类需要明确导入一个包中的Date

如果两个都使用，则在类前加上包名。

静态导入

import 语句不仅可以导入类，还增加了导入静态方法和静态域的功能

import static java.lang.System.\*

就可以使用System.类的静态方法和静态域

直接使用out.println(“”) exit(0)

将类放入包中

要想将一个类放入包中，就必须将包的名字放在源文件的开头，包中定义类的代码之前。如果没有在源文件中放置package语句，这个源文件中的类就放置在一个默认的包（default package）中。

包作用域

修饰符 ：

标记为public的部分可以被任意的类使用

标记为private的部分只能被定义它们的类使用

如果没有指定public 和 private 这个部分（类、方法或变量）可以被同一个包中的所有方法访问。

类路径

类路径必须与包名匹配

另外，类文件可以存储在JAR（java归档）文件中。

classpath

文档注释

方法注释

@param 变量描述

@return 描述

@throw 类描述

通用注释

@author 姓名

@version 文本 版本

@since 文本 始于

@deprecated 文本

@see 引用

类设计技巧

1 一定要保证数据私有

2 一定要对数据初始化

3 不要在类中使用过多的基本类型

4 不是所有的域都需要独立的域访问器和域更改器

5 将职责过多的类进行分解

6 类名和方法名要能够体现它们的职责