# 第十三章 抽象类和接口

## 13.1引言

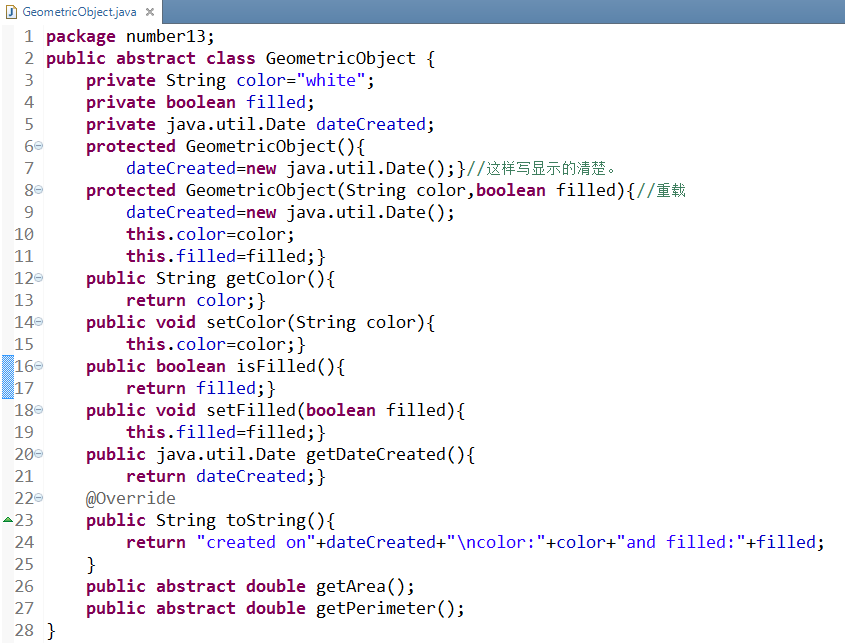
1.要点提示：父类中定义了相关子类中的共同行为。接口可以用于定义类的共同行为（包括非相关的类）。

## 13.2抽象类

1.要点提示：抽象类不可以用于创建对象。抽象类可以包含抽象方法，这些方法将在具体的子类中实现。

2.**精辟：**在继承的层次结构中，每个新子类都使类变得明确、具体。如果从一个子类追溯到父类，类就会变得更通用、更不明确。类的设计应该确保父类包含它的子类的共同特征。有时候，一个父类的设计非常抽象，以至于它没有任何具体的实例，这样的类称为“抽象类”。

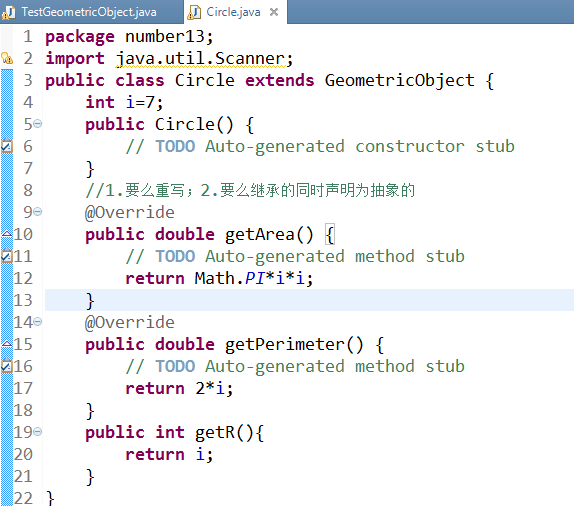
3.抽象类和抽象方法使用修饰符abstract修饰符表示。

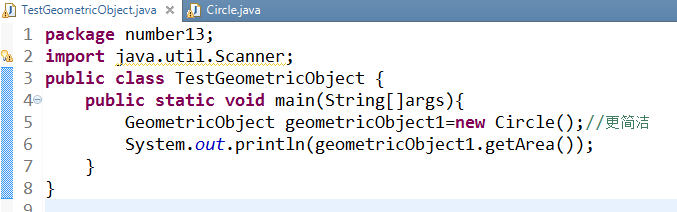


4.抽象类和常规类很像，但是不能使用new操作符创建它的实例。抽象方法只有定义而没有实现，它的实现由子类提供。一个抽象的方法类必须声明为抽象类。抽象类的构造方法定义为protected，因为它只被子类使用。

### 13.2.1为何要使用抽象方法

目的：更简洁





（我在这里自行修改了一下，因为原有程序不能跑没有意义。遗憾的是把半径写死了，不能在主函数中读取输入流的同时调用子类，大概是这种冲突导致的）

### 13.2.2抽象类的几点说明

（比较难懂）

1.在抽象类扩展到非抽象子类中，必须实现所有的抽象方法。

2.抽象类是不能使用new操作符来初始化的。但是，仍然可以定义它的构造方法，这个构造方法在它的子类的构造方法中调用。

3.包含抽象方法的类必须是抽象的。但是，可以定义一个不包含抽象方法的抽象类。在这种情况下，不能使用new操作符创建该类的实例。

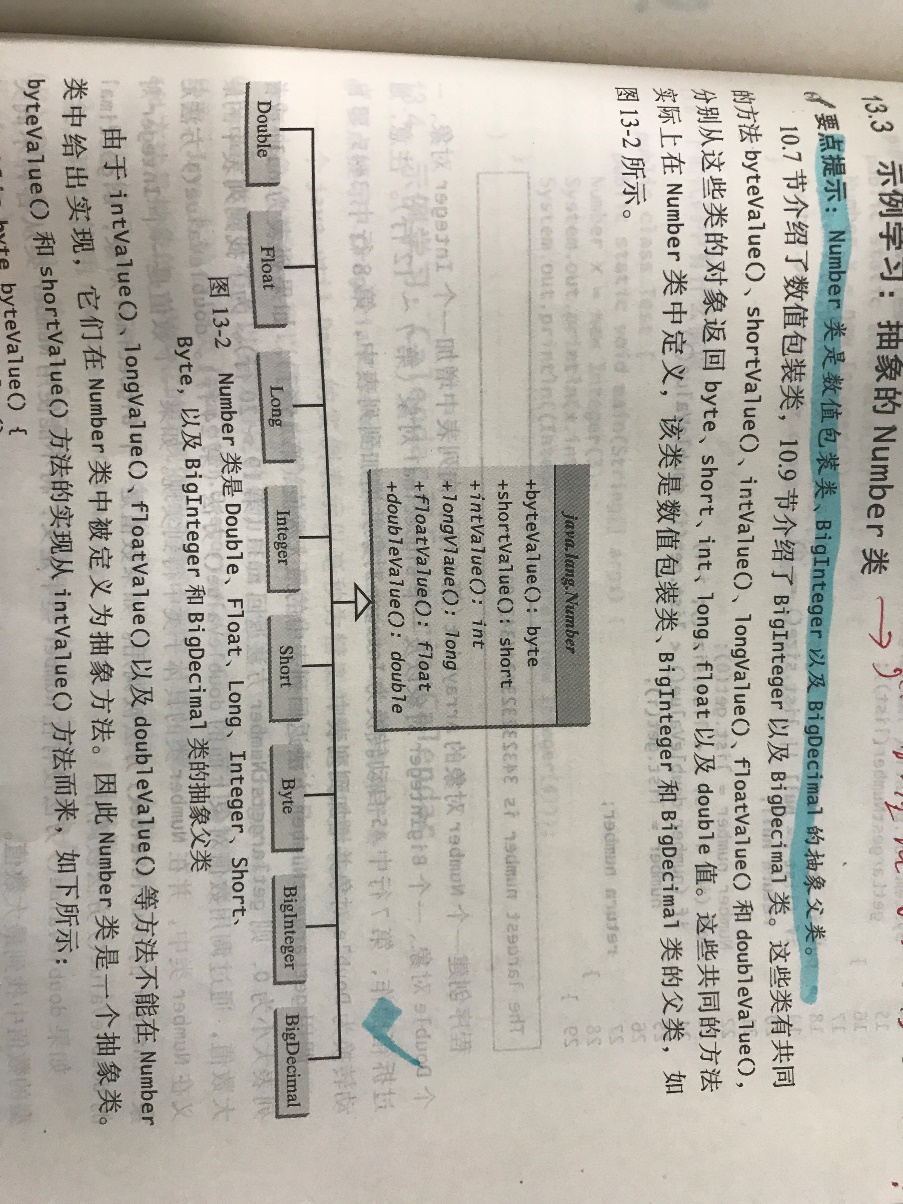
4.子类可以覆盖父类的方法并将其定义为抽象的。（重写）

5.即使子类的父类是具体的，这个子类也可以是抽象的。（跟4相反）

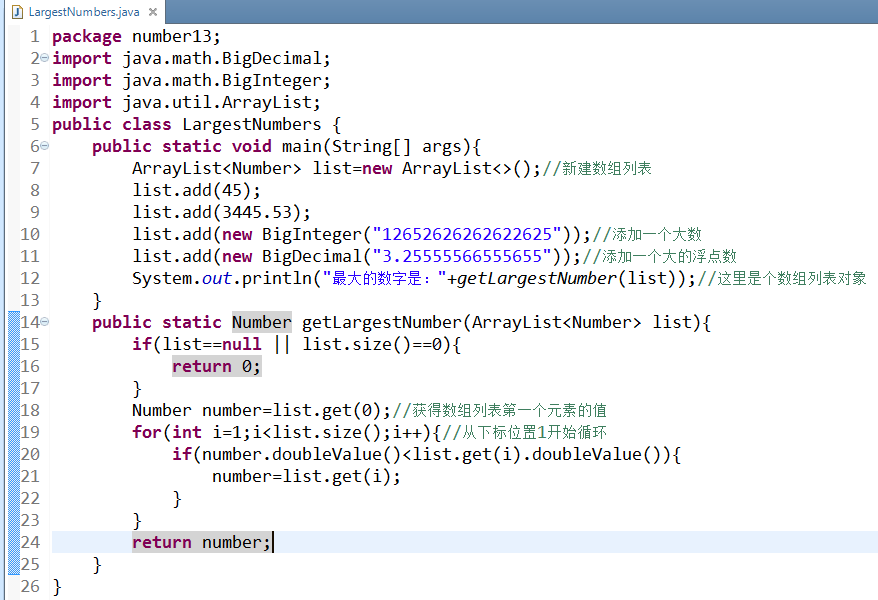
6.不能使用new操作符从一个抽象类创建一个实例，但是抽象类可以用作一种数据类型。

## 13.3示例学习：抽象的Number类

1.提示：Number类是数值包装类、BigInteger已经BigDecimal的抽象父类。



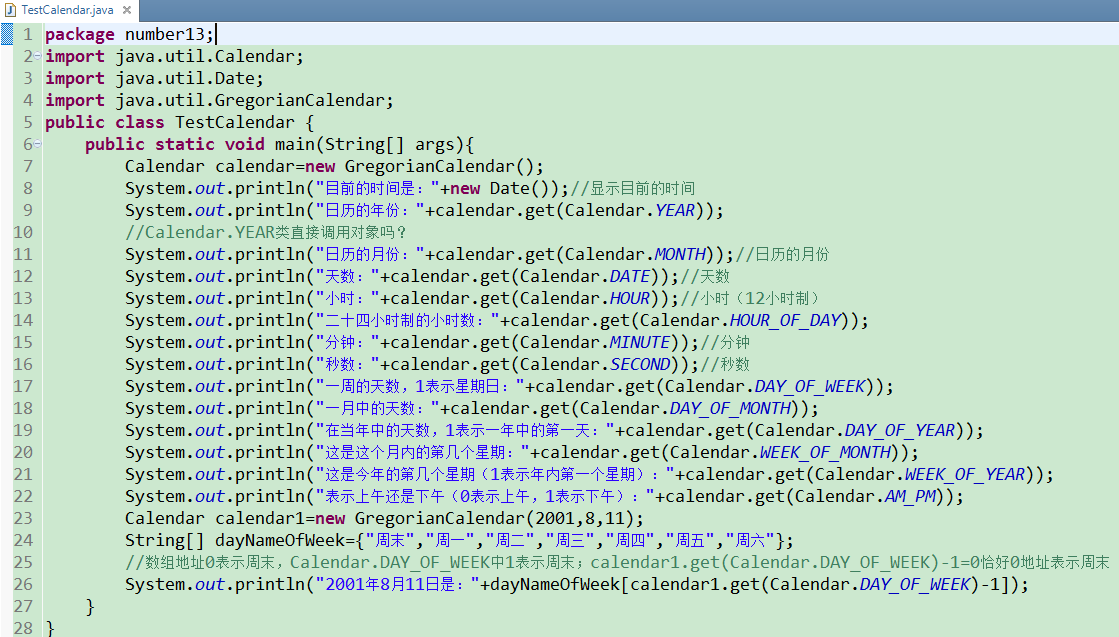
2.示例程序：找到一个Number对象列表中的最大数。



## 13.4示例学习：Calendar和GregorianCalendar

1.要点提示：GregorianCalendar是抽象类Calendar的一个具体子类。

2.一个Java.util.Date的实例表示以毫秒为精度的特定时刻。Java.util.Calendar是一个抽象的基类，可以取出详细的日历信息，例如，年、月、日、小时、分钟、秒。



## 13.5接口

1.要点提示：接口是一种与类相似的结构，只包含常量和抽象方法。

2.接口在很多方面都与抽象类很相似，但是它的目的是指明相关或者不相关的多个对象的共同行为。

3.为了区分接口和类，Java采用下面的语法来定义接口：

修饰符 interface 接口名 {

//常量声明

//方法签名

}

4.类和接口之间的关系称为“接口继承”，因为接口继承和类的继承本质上是相同的，所以我们都将它们简称为“继承”。以下是一个接口继承的示例程序（原程序有错，此处略去）

## 13.6Comparable接口

1.要点提示：Comparable接口定义了compareTo方法，用于比较对象。假设要设计一个求两个相同类型对象中较大者的通用方法，这两个对象必须是可比较的。

2.Comparable接口的定义如下：

Package Java.lang；

Public interface Comparable<E>{

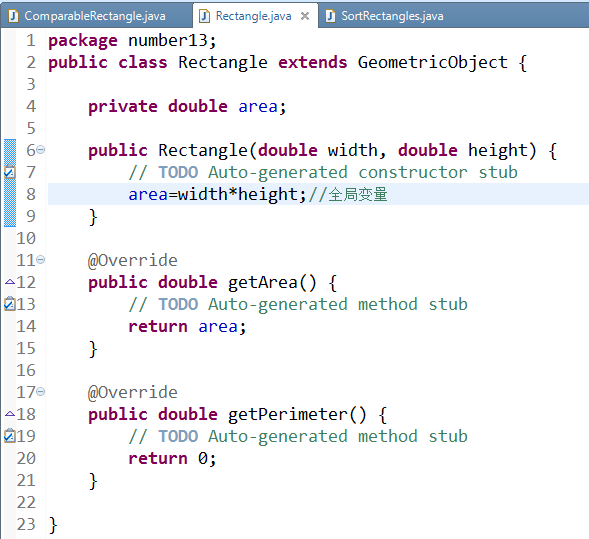
Public int comparableTo(E o)；}

compareTo方法判断这个对象对于给定对象o的顺序并且当这个对象小于、等于或大于给定对象o时，分别返回负整数、0或正整数。Comparable接口是一个泛型接口。在实现该接口时，泛类型E被替换成一种具体的类型。

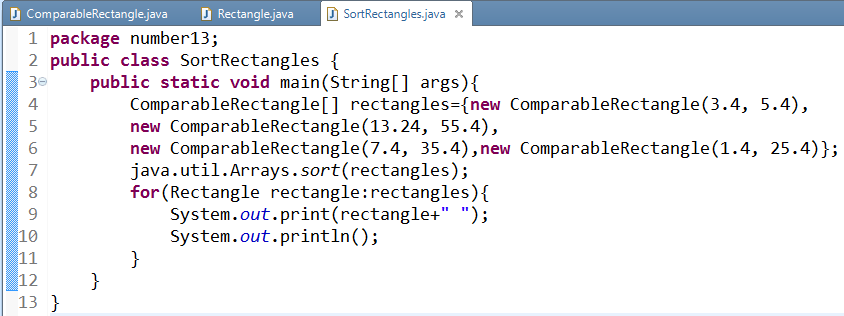
3.Java API 中的java.util.Arrays.sort(Object[])方法就可以使用compareTo方法来对数组中的对象进行比较和排序。以下程序给出一个对字符串数组和BigInteger对象数组进行排序的示例：



4.不能使用sort方法来对一个Rectangle(矩形)对象数组进行排序，因为Rectangle类没有实现接口Comparable。然而，可以定义一个新的Rectangle类来实现Comparable。这个新类的实例是可比较的，将这个新类命名为ComparableRectangle，如图所示：



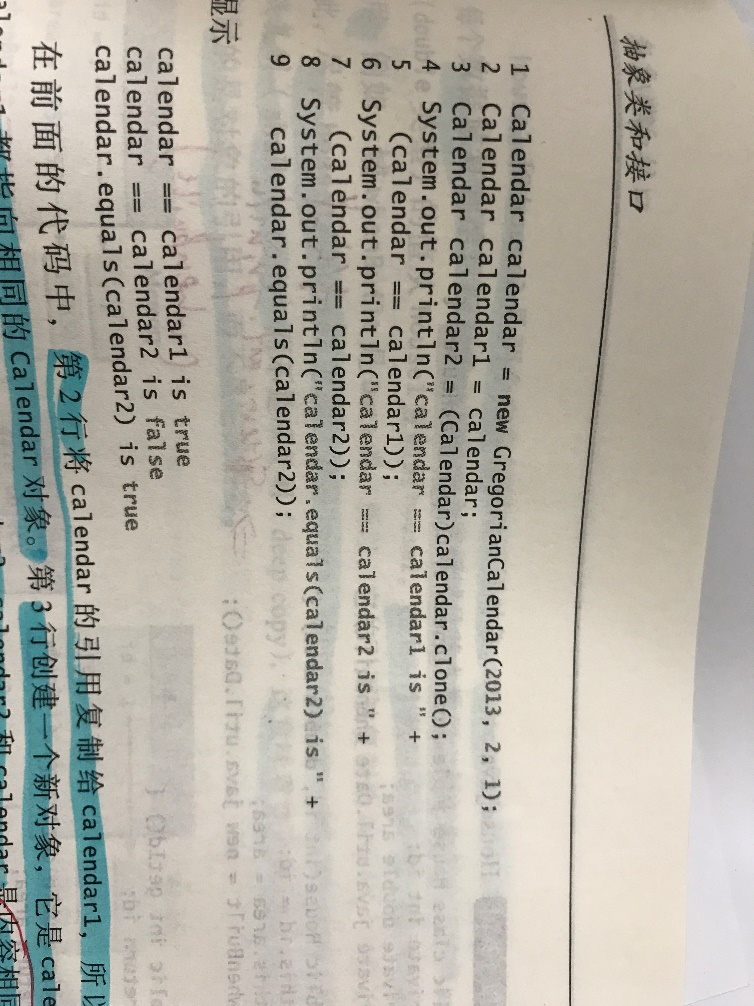




## 13.7Cloneable接口（这些接口和方法有什么区别？）

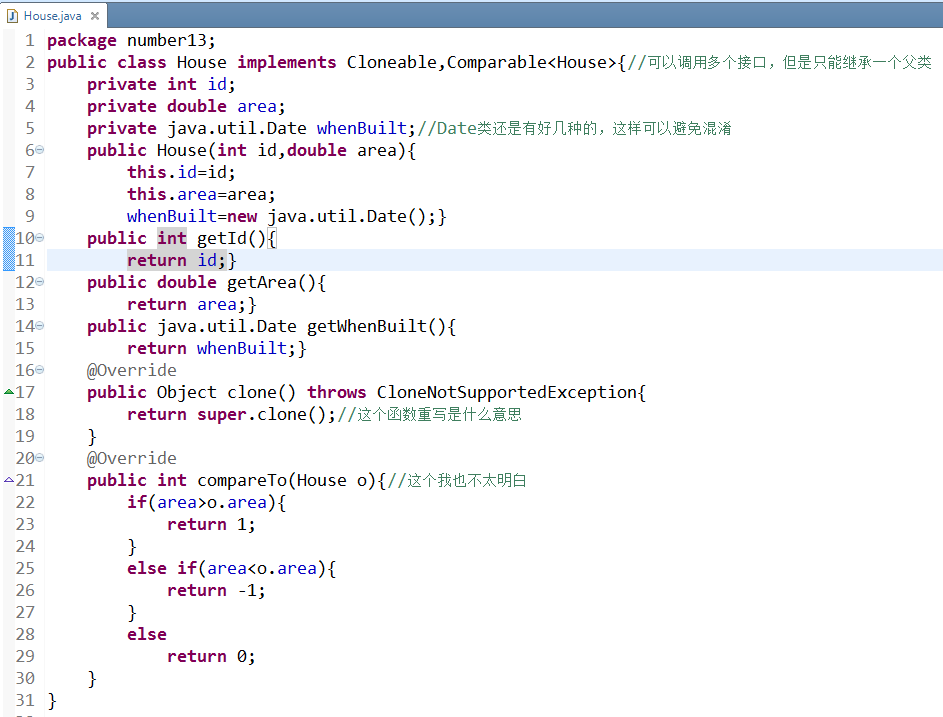
1.要点提示：Cloneable接口给出了一个可克隆的对象。

经常会出现需要创建一个对象拷贝的情况。为了实现这个目的，需要使用clone方法并理解Cloneable接口



在上述代码中，第2行将calendar的引用复制给calendar1，所以calendar和calendar1都是指向相同Calendar的对象。第3行创建一个新对象，它是calendar的克隆，然后将这个新对象的引用赋值给calendar2。Calendar2和calendar是内容相同的不同对象。

2.以下程序定义一个实现Cloneable和Comparable的House类。



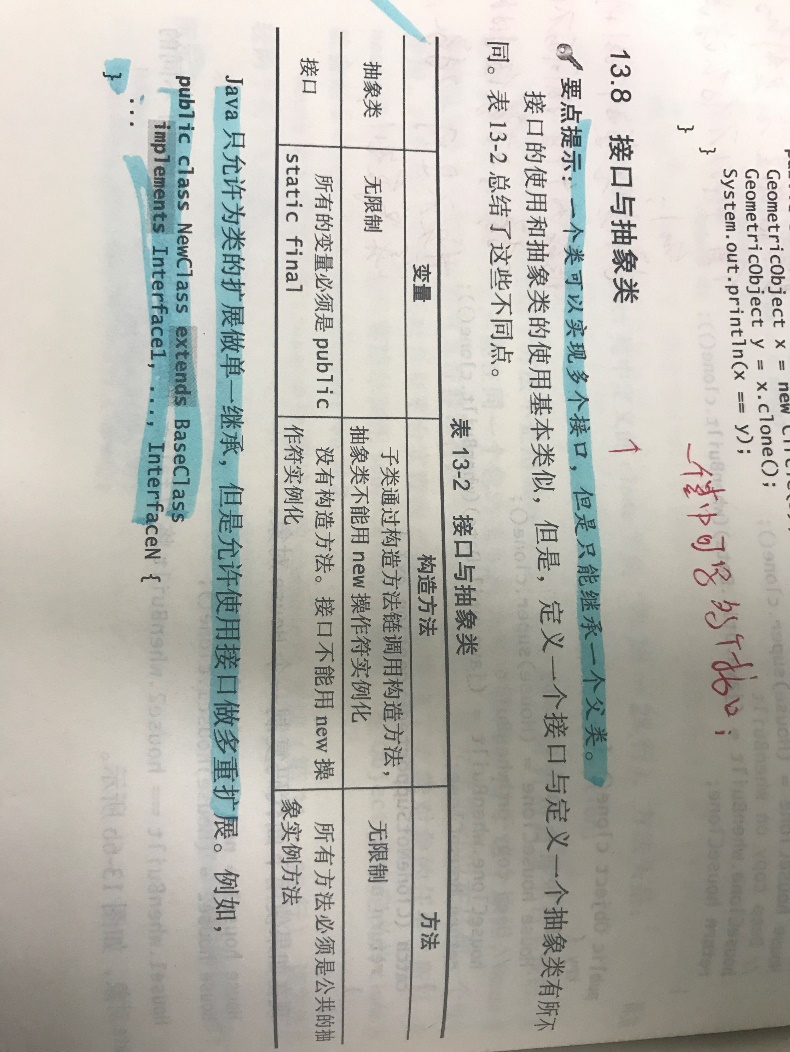
3.深复制和浅复制

基本数据类型直接复制数值，所以不存在深浅复制；对象类型复制的引用，形成两个数值相同的不同对象，存在深浅复制。两个对象完全一致叫深复制；数值相同的不同对象叫浅复制。

## 13.8接口与抽象类

1.要点提示：一个类可以实现多个接口，但是只能继承一个父类。

2.以下是接口与抽象类的比较：



3.Java只允许为类扩展做单一的继承，但是允许使用接口做多重扩展。例如：

Public class NewClass extends BaseClass implements Interface1,…,InterfaceN{ }

利用关键字extends，接口可以继承其他接口。这样的接口称为“子接口”。例如，在下面的代码中，newInterface 是interface1…interfaceN的子接口。

Public interface NewInterface extends interface1,…,interfaceN{ }

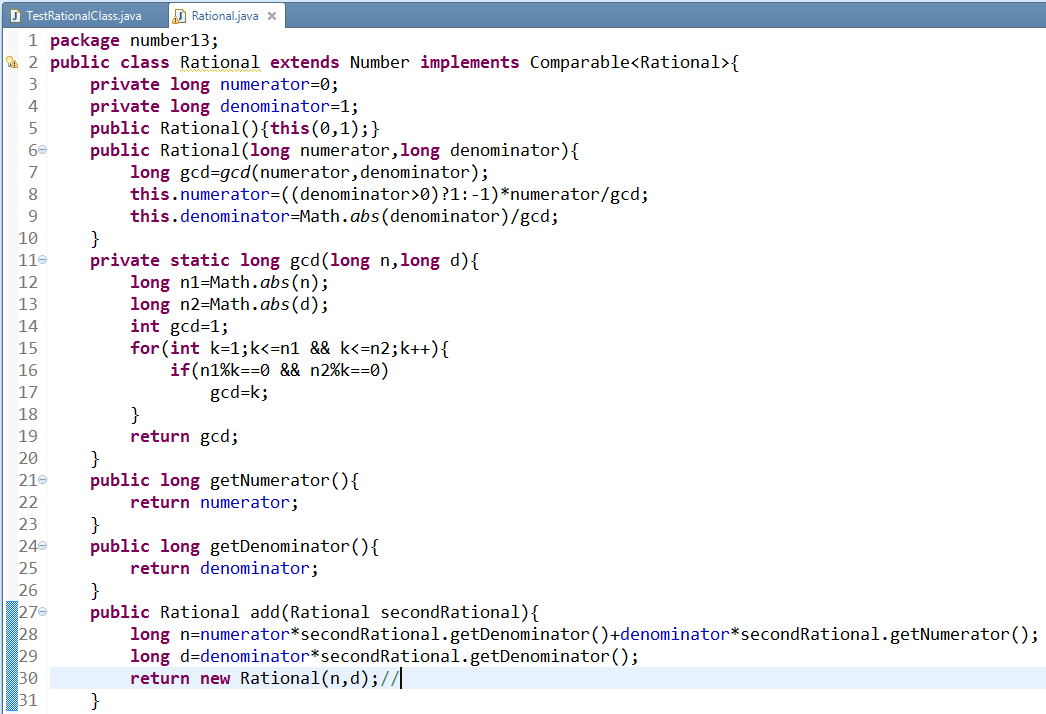
4.所有的类共享一个根类Object，但是接口没有共同的根。

5.那么该如何确定在什么情况下使用接口？什么情况下使用类呢？

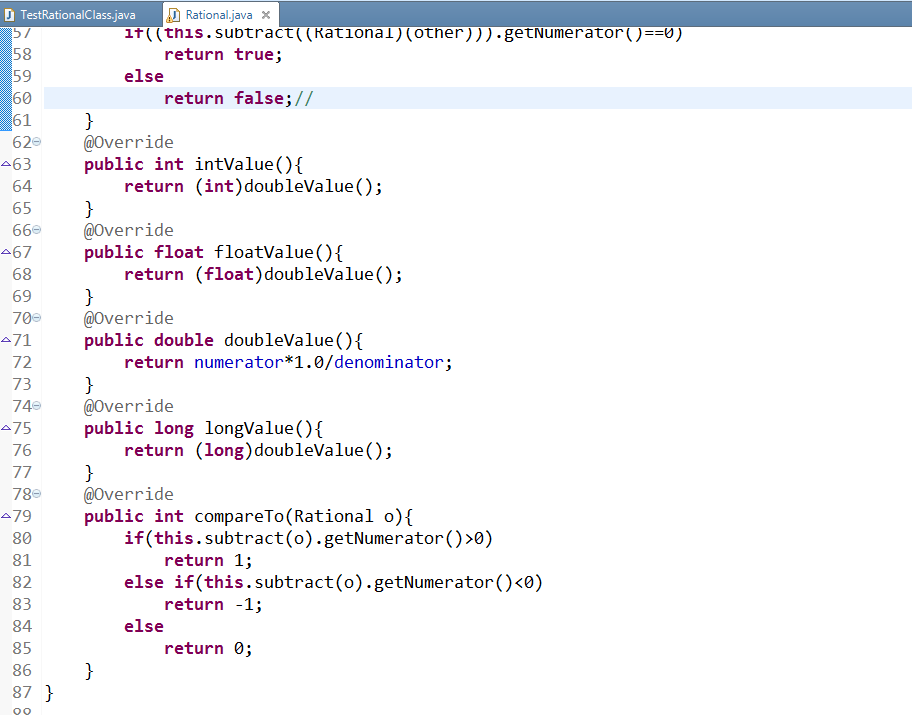
一般来说清晰的描述父子关系的强的“是一种”的关系，应该用类来建模；弱的“是一种”关系，也成为类属关系，它表明对象拥有某种属性，可以用接口来建模。

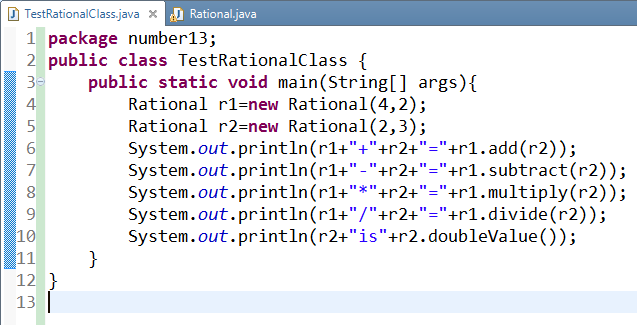
通常，推荐使用接口而非抽象类，因为接口可以定义非相关类共有的父类型。接口比类更加灵活。

## 13.9示例学习：Rational类









## 13.10类的设计原则

1.要点提示：类的设计原则有助于设计出合理的类。

### 13.10.1内聚性

类应该描述一个单一的实体，而所有的类操作应该在逻辑上相互配合，支持一个一致的目的。

### 13.10.2一致性

遵循标准Java程序设计风格和命名习惯。

### 13.10.3封装性

一个类应该使用private修饰符隐藏其数据，以免用户直接访问它，这使得类更易于维护。

### 13.10.4清晰性

### 13.10.5完整性（我对自带方法的源码比较感兴趣）

类是为许多不同用户的使用而设计的。为了能在一个广泛的应用中使用，一个类应该通过属性和方法提供多种方案以适应用户的不同需求。例如：为满足不同的应用需求，String类包含了40多种很使用的方法。

### 13.10.6实例和静态

依赖于类的具体实例的变量或方法必须是一个实例变量或方法。如果一个变量被类的所有实例所共享，那就应该将它声明为静态的。

### 13.10.7继承与聚合

### 13.10.8接口和抽象类

接口和抽象类都可以用于为对象指定共同的行为。比较强的“是一种”的关系应该使用类来建模；弱的“是一种”的关系可以使用接口来建模。接口比抽象类更加灵活，因为一个子类只能继承一个父类，但是却可以实现任意个数的接口。