**类的抽象和封装**

**类的抽象是指将类的实现和类的使用分离开，实现的细节被封装并且对用户隐藏，这被称为类的封装。**

**java提供了多层次的抽象。**

**类抽象（class abstraction）是将类的实现和使用分离。类的创建者描述类的功能，让使用者明白如何才能使用类。从类外可以访问的方法和数据域的集合以及预期这些成员如何行为的描述，合称为类的合约（class's contract）。**

**类的使用者不需要知道类是如何实现的。实现的细节经过封装，对用户隐藏起来，这称为类的封装（class encapsulation）。例如，**

**可以，可以创建一个Circle对象，并且可以在不知道面积是如何计算出来的情况下，求出这个圆的面积。由于这个原因，类也称为抽象数据类型（Abstract Data Type，ADT）。**

**将基本数据类型值作为对象处理**

**基本数据类型不是一个对象，但是可以使用javaAPI中的包装类来包装成一个对象。**

**处于对性能的考虑，在java中基本数据类型不作为对象使用。因为处理对象需要额外的系统开销，所以，如果将基本数据类型当做对象，就会对语言性能带来负面影响。**

**然而，java中的许多方法需要将对象作为参数。java提供了一个方便的办法，即将基本数据类型并入对象或包装成对象（例如，**

**int包装成Integer类，double包装成Double类，char包装成Character类）。**

**通过使用包装类，可以将基本数据类型值作为对象处理。**

**大多数基本类型的包装类的名称与对应的基本数据类型名称一样，第一个字母要大写。Integer和Character例外。**

**数值包装类相互之间都非常相似。每个都包含了doubleValue()、floatValue()、intValue()、longValue()、shortValue()和byteValue()方法。这些方法将对象“转换”为基本类型值。**

**既可以用基本数据类型值，也可以用表示数字的字符串来构造包装类。例如**

**new Double(5.0)、new Double("5.0")、new Integer(5)和new Integer("5")。**

**包装类没有无参构造方法。所有包装类的实例都是不可变的，这意味着一旦创建对象后，它们的内部值就不能再改变。**

**每一个数值包装类都有常量MAX\_VALUE和MIN\_VALUE。对Float和Double类而言，MIN\_VALUE表示float型和double型的最小正值。**

**每个数值包装类都会包含方法doubleValue()、floatValue()、intValue()、longValue()和shortValue()。这些方法返回包装对象的double、float、int、long或short值。例如**

**new Double(12.4).intValue() //returns 12;**

**new Integer(12).doubleValue() //returns 12.0;**

**数值包装类中包含compareTo方法用于比较两个数值，并且如果该数值大于、等于或者小于另外一个数值时，分别返回1、0、-1。例如**

**new Double(12.4).compareTo(new Double(12.3)) //returns 1;**

**数值包装类有一个有用的静态方法valueOf(String s)。该方法创建一个新对象，并将它初始化为指定字符串表示的值。例如**

**Double doubleObject=Double.valueOf("12.4");**

**Integer integerObject=Integer.valueOf("12");**

**使用Integer类中的parseInt方法将一个数值字符串转换为了一个int值。**

**使用Double类中的parseDouble方法将一个数值字符串转换为了一个double值。**

**每个数值包装类都有两个重载的方法，将数值字符串转换为以10（十进制）或指定值为基数（例如，2为二进制，8为八进制，16为十六进制）的数值。例如**

**Integer.parseInt("11",2) //returns 3;**

**Integer.parseInt("12",8) //returns 10;**

**Integer.parseInt("12",10) //returns 12;**

**根据上下文环境，基本数据类型值可以使用包装类自动转换成一个对象，反过来的自动转换也可以。**

**将基本类型值转换为包装类对象的过程称为装箱（boxing），相反的转换过程称为开箱（unboxing）。java允许基本类型和包装类类型之间进行自动装箱。**

**如果一个基本类型值出现在需要对象的环境中，编译器会将基本类型值进行自动装箱；如果一个对象出现在需要基本类型值的环境中，编译器会将对象进行自动开箱。这称为自动装箱和自动开箱。例如**

**Integer intObject=new Integer(2);**

**等价于**

**Integer intObject=2;**

**BigInteger和BigDecimal类**

**BigInteger类和BigDecimal类可以用于表示任意大小和精度的整数或者十进制数。在java.math包中。**

**可以使用new BigInteger(String)和new BigDecimal(String)来创建BigInteger和BigDecimal的实例，它们之间只能使用add、subtract、multiple、divide和remainder方法完成算数运算，使用compareTo方法比较两个大数字。例如**

**BigInteger a=new BigInteger("9999999999");**

**BigInteger b=new BigInteger("2");**

**BigInteger c=a.multiply(b);//a与b相乘**

**对BigDecimal对象的精度没有限制。如果结果不能终止，呢么divide方法会抛出ArithmeticException异常。但是，可以使用重载的divide(BigDecimal d,int scale,int roundingMode)方法来指定尺度和舍入方式来避免这个异常，这里的scale是指小数点后最小的整数位数。例如**

**//下面的代码创建两个尺度为20、舍入方式为BigDecimal.ROUND\_UP的BigDecimal对象**

**BigDecimal c=a.divide(b,20,BigDecimal.ROUND\_UP);**

**可以用字符串直接量或字符数组创建一个字符串对象。使用如下语法，用字符串直接量创建一个字符串：**

**Stirng newString=new String(stringLiteral);**

**String对象是不可变的，它的内容是不能改变的。**

**因为字符串在程序设计中是不可变的，但同时又会频繁地使用，所以java虚拟机为了提高效率并节约内存，对具有相同字符序列的字符串直接量使用同一个实例。这样的实例称为限定的（interned）字符串。例如**

**String s1="Welcome to java";**

**String s2=new String("Welcome to java");**

**String s3="Welcome to java";**

**//s1==s2 is false**

**//s1==s3 is true**

**由于s1和s3指向相同的限定字符串“Welcome to java”，因此，s1==s3为true，但是，s1==s2为false，这是因为尽管s1和s2内容相同，但它们是不同的字符串对象。**

**字符串的替换和分隔**

**String类提供替换和分隔字符串的方法：**

**replace(oldChar:char,newChar char):char**

**replaceFirst(oldString:String,newString:String):String**

**replaceAll(oldString:String,newString:String):String**

**split(delimiter:String):String[] //返回一个字符串数组，其中包含被分隔符分隔的子字符串集**

**一旦创建了字符串，它的内容就不能改变。但是replace、replaceFirst和replaceAll会返回一个源自原始字符串的新字符串（并未改变原始字符串！）。**

**检测用户输入格式，正则表达式（regular expression）（缩写regex）是一个字符串，用于描述匹配一个字符串集的模式。**

**String类的matches方法和equals方法非常相似，但是matches方法的功能更加强大。它不仅可以匹配定长的字符串，还能匹配一套遵从某种模式的字符串。**

**字符串与数组之间的转换**

**字符串不是数组，但是字符串可以转换成数组，反之亦然。为了将字符串转换成一个字符数组，可以使用toCharArray方法。例如**

**char[] chars="java".toCharArray();**

**因此，chars[0]是'j'、...**

**还可以使用方法getChars(int srcBegin,int srcEnd,char[] dst,int dstBegin)将下标从srcBegin到srcEnd-1的子串复制到字符数组dst中下标从dstBegin开始的位置。**

**为了将一个字符数组转换成一个字符串，应该使用构造方法String(char[])或者方法valueOf(char[])。例如**

**String str=new String(new char[]{'j','a','v','a'});**

**String str=String.valueOf(new char[]{'j','a','v','a'});**

**将字符和数值转换成字符串**

**可以使用Double.parseDouble(str)或者Integer.parseInt(str)将一个字符串转换为一个double值或者一个int值，也可以使用字符串的连接操作符来将字符或者数字转换为字符串。另外一种方法是使用重载的静态valueOf方法。该方法可以用于将字符和数值转换成字符串。**

**valueOf(c:char):String**

**valueOf(data:char[]):Stirng**

**valueOf(d:double):String**

**valueOf(f:float):String**

**valueOf(i:int):String**

**valueOf(l:long):String**

**valueOf(b:boolean):String**

**格式化字符串**

**String类包含静态format方法，它可以创建一个格式化的字符串。调用该方法的语法是：**

**String.format(format,item1,item2,...,itemk)**

**这个方法很像printf方法，只是format方法返回一个格式化的字符串，而printf方法显示一个格式化的字符串。**

**StringBuilder和StringBuffer类**

**StringBuilder和StringBuffer类似于String类，区别在于String类是不可改变的。**

**一般来说，只要使用字符串的地方，都可以使用StsringBuilder/StringBuffer类。**

**StringBuilder/StringBuffer类比String类更加灵活。可以给一个StringBuilder或StringBuffer中添加、插入或追加新的内容，但是String对象一旦创建，它的值就确定了。**

**除了StringBuffer中修改缓冲区的方法是同步的，这意味着只有一个任务被允许执行方法之外，StringBuilder类与StirngBuffer类是很相似的。如果是多任务并发访问，就使用StringBuffer，因为这种情况下，需要同步以防止StringBuffer崩溃。**

**如果是单任务访问，使用StringBuilder会更有效。StringBuffer和StringBUilder中的构造方法和其他方法几乎是完全一样的。**

**可以在字符串构建器的末尾追加新内容，在字符串构建器的特定位置插入新的内容，还可以删除或替换字符串构建器中的字符。**

**如果一个字符串不需要任何改变，则使用String类而不使用StringBuffer类。java可以完成对String类的优化，例如，共享限定字符串等。**

**StringBuilder类提供了许多其他处理字符串构建器和获取它的属性的方法。**

**capacity()方法返回字符串构建器当前的容量。容量是指在不增加构建器大小的情况下能存储的字符数量。**

**length()方法返回字符串构建器中实际存储的字符数量。setLength(newLength)方法设置字符串构建器的长度。如果参数newLength小于字符串构建器的当前长度，则字符串构建器会被截短到敲好能包含有参数newLength给定的字符个数。如果参数newLength大于或者等于当前长度，则给字符串构建器追加足够多的空字符（'\u0000')，使其长度length变成新参数newLength。参数newLength必须大于等于0.**

**charAt(index)方法返回字符串构建器中某个特定下标index的字符。下标是基于0的，字符串构建器中的第一个字符的下标为0，第二个字符的下标为1，以此类推。参数index必须大于或等于0，并且小于字符串构建器的长度。**

**字符串的长度总是小于或等于构建器的容量。长度是存储在构建器中的字符串的实际大小，而容量是构建器的当前大小。如果有更多的字符添加到字符串构建器，超出它的容量，则构建器的容量会自动增加。在计算机内部，字符串构建器是一个字符数组，因此，构建器的容量就是数组的大小。如果超出构建器的熔炼，就用新的数组替换现有数组。新数组的大小为2\*（之前数组的长度+1）。**

**可以使用new StringBuilder(initialCapacity)创建指定初始容量的StringBuilder。通过仔细选择初始容量能够使程序更有效。**

**如果容量总是超过构建器的实际使用长度，JVM将永远不需要为构建器重新分配内存。另一方面，如果容量过大将会浪费内存空间。可以使用trimToSize()方法将容量降到实际的大小。**