链滴

010 面向对象思考

作者：pzs233
原文链接：https://ld246.com/article/1537285898013
来源网站：链滴

本文为《Java 语言程序设计》第十版 章节笔记

10.2 类的抽象和封装

- **类的抽象**： 类的实现和类的使用分离开
- **类的封装**： 类的实现被封装并且都用户隐藏

类的抽象和封装是一个问题的两个方面。

从类的开发者角度来看，设计类是为了让很多的不同用户多使用。为了在更大的应用范围内使用类，应该通过构造方法、属性和方法提供各种方式的定制。

10.3 面向对象的思考

- 面向过程的范式重点在于设计方法。
- 面向对象的范式将数据和方法耦合子在一起构成对象。

使用面向对象范式的软件设计重点在对象以及对对象的操作上。

在面向过程的程序设计中，数据和数据上的操作是分离的，而且这种做法要求传递数据给方法。

10.4 类的关系

类中间的关系通常是 关联、聚合、组合、继承。

关联

关联 是一种常见的二元关系，描述两个类之间的活动。

在 Java 代码中，可以通过使用 数据域 以及 方法 来实现关联。

聚集和组合

聚集 是关联的一种特殊形式，代表了两个对象之间的归属关系。

一个对象可以被多个其他的聚集对象所拥有。如果一个对象只归属于一个聚集对象，那么它和聚集对之间的关系就称为 组合（composition）。

聚集关系 通常被表示为 聚集类 中的一个 数据域。

聚集 可以存在于同一个类的多个对象之间。

10.7 将基本数据类型值作为对象处理

基本数据类型值不是一个对象，但是可以使用 Java API 中的 包装类 来包装成一个对象。

Java 为基本数据类型提供了 Boolean、Character、Double、Float、Byte、Short、Integer 和 Long 等包装类。

包装类有的一些方法：

- doubleValue()
- floatValue()
- intValue()
- longValue()
- shortValue()
- byteValue()
- compareTo()
- valueOf(String s)
- paraseInt()

包装类没有无参构造方法。

每个数值包装类都有常量 MAX_VALUE 和 MIN_VALUE。

10.8 基本类型和包装类型之间的自动转换

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。

- 装箱（boxing）：将基本类型值转换为包装类对象的过程
- 开箱（unboxing）：将包装类对象转换成基本类型的过程
- 自动装箱：如果一个基本类型值出现在需要对象的环境装，编译器会将基本类型值进行自动装箱
- 自动开箱：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

10.9 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。

二者都是不可变的。

一些方法：add、subtract、multiple、divide、remainder

10.10 String 类

String 对象是不可改变的。

Java 将字符串直接量看做 String 对象。

限定的（interned）字符串：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。

字符串的替换和分隔

```
java.lang.String
+repalce(oldChar: char, newChar: char): String
+replaceFirst(oldString: String, newString:  String): String
```

+replaceAll(lodString: String, newString: String): String
+split(delimiter: String): String[]

split 方法 可以从一个指定分隔符的字符串中提取标识。

依照模式匹配、替换和分隔

String 类的 matches 方法用于匹配字符串，它不仅可以匹配定长的字符串，还能匹配一套遵从某种式（正则表达式）的字符串。

"Java is fun".matches("Java.*"); // return true

"400-02-4534".matches("\\d{3}-\\d{2}-\\d{4}); // return true

字符串与数组之间的转换

- 字符串转换成字符数组： toCharArrya()
- 字符串数组转换成字符串： String(char []) 或 valueO(char [])

格式化字符串

String.format(format, item1, item2, ..., tiemk)

System.out.printf(format, item1, tiem2, ..., itemk);等价于：

System.out.print(String.format(format, item1, item2, ..., itemk));

10.11 StringBuilder 和 StringBuffer 类

一般来说，只要使用字符串的地方，都可以使用 StringBuilder/StringBuffer 类。可以给一个 StringBilder或 StringBuffer 中添加、插入或追加新的在内容，但 String 对象一旦创建，它的值就确定了。

StringBuffer 中修改缓冲区的方法时同步的。如果是多任务并发访问，就是用 StringBuffer，因为这情况下需要同步一防止 StringBuffer 崩溃。如果是单任务访问，使用 StringBuilder 会更高效。

java.lang.StringBuilder
//构造方法
+StringBuilder()
+StringBuilder(capacity: int)
+StringBuilder(s: String)

//修改 StringBuilder 中的字符串
+append(data: char[]): StringBuilder
+append(data: char[], offset: int, len: int): StringBuilder
+append(v: aPrimitiveType): StringBuilder
+append(s: String): StringBuilder
+delete(startIndex: int, endIndex: int): StringBuilder
+deletaCharAt(index: int): StringBuilder
+insert(index: int, data: char[], offset: int, len: int); StirngBuilder
+insert(offset: int, data: char[]): StringBuilder
+insert(offset: int, b: aPrimitiveType): StringBuilder
+insert(offset: int, s: String): StringBuilder

+replace(startIndex: int, endIndex: int, s: String): StringBuilder
+reverse(): StringBuilder
+setCharAt(index: int, ch: char): void

//其他的一些方法
+toStrin(): String
+capacity(): int
+charAt(index: int): char
+length(): int
+setLength(newLentth: int): void
+substring(startIndex: int): String
+substring(startIndex: int, endIndex: int): String
+trimToSzie(): void

END