Java 为什么要加 final 关键字了！

我正在读一本有关Java的书，书上说可以声明整个类为final。我能想到的任何地方，都不会使用这个。对于编程，我只是一个新手，我想知道程序员是否实际在他们的程序中这样做。如果他们这样做，他们什么时候使用它，这样我能更好地了解它，知道什么时候使用它。如果Java是面向对象的，并且你声明一个final类，它是否不再认为类拥有这个对象的特征？

回答：

- final类只是一个不能扩展的类。如果你不想让其它人再次扩展你的类，就可以将其声明为final。如果Java是面向对象的，并且你声明一个final类，它是否不再认为类拥有这个对象的特征？
  在某种意义上是的。通过将类标记为final，您将禁用该部分代码的强大而灵活的语言特性。然而，一些类不应该（在某些情况下不能）被设计为以良好的方式考虑子类。在这些情况下，将类标记为final是有意义的，即使它限制了OOP。（但是请记住，final类仍然可以扩展为另一个非final类）。

什么样子的情况下一个类需要被Final进行修饰呢？

- 说白一个类如果你觉得这个类的已经很完美了。我不需要任何人进行修改和扩展的时候，你可以考虑设置final
- jdk中就存在大量的final修饰的比如：String、Integer、Long、Float、枚举类。
- 作用1：起到一个保护和安全的一个作用

- 作用2：对内修改开放，对外修改关闭。（String，对内修改开放（oracle的团队），对于程序员来说就是关闭的，）
- 一句话：被final修饰的类就是只读的类。

final关键字的好处

下面总结了一些使用final关键字的好处

- final关键字提高了性能。JVM和Java应用都会缓存final变量。
- final变量可以安全的在多线程环境下进行共享，而不需要额外的同步开销。
- 使用final关键字，JVM会对方法、变量及类进行优化。

不可变类

创建不可变类要使用final关键字。不可变类是指它的对象一旦被创建了就不能被更改了。String是不可变类的代表。不可变类有很多好处，譬如它们的对象是只读的，可以在多线程环境下安全的共享，不用额外的同步开销等等。

- final关键字可以用于成员变量、本地变量、方法以及类。
- final成员变量必须在声明的时候初始化或者在构造器中初始化，否则就会报编译错误。
- 你不能够对final变量再次赋值。
- 本地变量必须在声明时赋值。
- 在匿名类中所有变量都必须是final变量。
- final方法不能被重写。
- final类不能被继承。
- final关键字不同于finally关键字，后者用于异常处理。final关键字容易与finalize()方法搞混，后者是在Object类中定义的方法，是在垃圾回收之前被JVM调用的方法。
- 接口中声明的所有变量本身是final的。

- final和abstract这两个关键字是反相关的，final类就不可能是abstract的。
- final方法在编译阶段绑定，称为静态绑定(static binding)。
- 没有在声明时初始化final变量的称为空白final变量(blank final variable)，它们必须在构造器中初始化，或者调用this()初始化。不这么做的话，编译器会报错“final变量(变量名)需要进行初始化”。
- 将类、方法、变量声明为final能够提高性能，这样JVM就有机会进行估计，然后优化。
- 按照Java代码惯例，final变量就是常量，而且通常常量名要大写

String在内存中如何存储（Java）

JDK1.8中JVM把String常量池移入了堆中，同时取消了“永久代”，改用元空间代替（Metaspace）
java中对String对象特殊对待，所以在heap区域分成了两块，一块是字符串常量池(String constant pool)，用于存储java字符串常量对象，另一块用于存储普通对象及字符串对象。
string的创建有两种方法：

```
public static void main(String[] args) {
    String a = "abc";  //第一种
    String b= new String("abc"); //第二种
    String c = "abc";
    System.out.println(a == b);//false
    System.out.println(a == c);//true
}
```

对于第一种，此创建方法会在String constant pool中创建对象。jvm会首先在String constant pool 中寻找是否已经存在"abc"常量，如果没有则创建该常量，并且将此常量的引用返回给String a；如果已有"abc" 常量，则直接返回String constant pool 中“abc” 的引用给String a。
对于第二种，jvm会直接在非String constant pool 中创建字符串对象，然后把该对象引用返回给String b，并且不会把"abc”加入到String constant pool中。new就是在堆中创建一个新的String对象，不管"abc"在内存中是否存在，都会在堆中开辟新空间。

虽然new String()方法并不会把"abc” 加入到String constant pool中，但是可以手动调用String.intern()，将new 出来的字符串对象加入到String constant pool中。

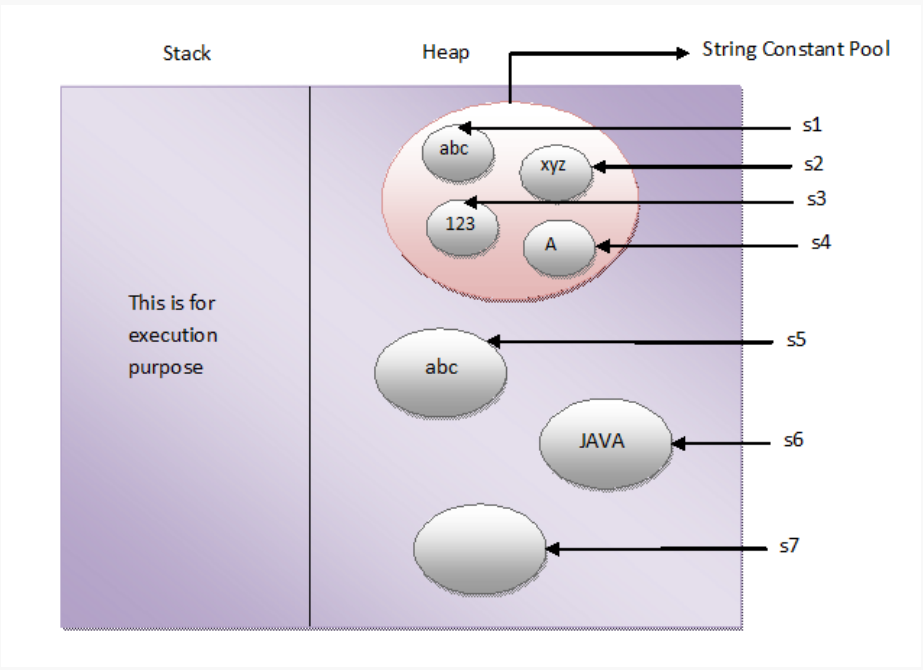
```
String s1 = new String("abc");
String s2 = "abc";
System.out.println(s1 == s2); //false
System.out.println(s1.intern() == s2); //true
```

当一个String实例调用intern()方法时，会查找常量池中是否有相同的字符串常量，如果有，则返回其的引用，如果没有，则在常量池中增加一个等于str的字符串并返回它的引用，由于s2已经在常量池中，所以s1.intern()不会再创建，而是直接引用同一个"aaa"。

```
public static void main(String[] args) {
String s1 = "abc";//字符串常量池

String s2 = "xyz";//字符串常量池

String s3 = "123";//字符串常量池

String s4 = "A";//字符串常量池

String s5 = new String("abc"); //堆里

```

```
12  char[] c = {'J','A','V','A'};
13
14  String s6 = new String(c);//堆里
15
16  String s7 = new String(new
    StringBuffer());//堆里
17  }
18
```

字符串在内存中的存储情况如下图所示：



总结：

对于字符串：其对象的引用都是存储在栈中的，如果是【编译期已经创建好(直接用双引号定义的)的就存储在常量池中】，如果是【运行期（new出来的）才能确定的就存储在堆中】。对于equals相等的字符串，在常量池中永远只有一份，在堆中有多份。