1. java面向对象的三大特征是封装、继承、多态。
2. （1）final关键字修饰变量时，变量就变成了常量，也即其值不能改变，和c++中的“const”作用相似。

代码如下：

**package** Test;  
**public class** test {  
 **public static void** main(String[] args) {  
 **final int** a = 10;  
 a =20;//编译器报错：无法将值赋值给final变量’a’  
 }  
}

编译器报错信息如下：



(2)final关键字修饰方法时，表示该方法不可以被覆盖，也即该方法不能被重写。

代码如下：

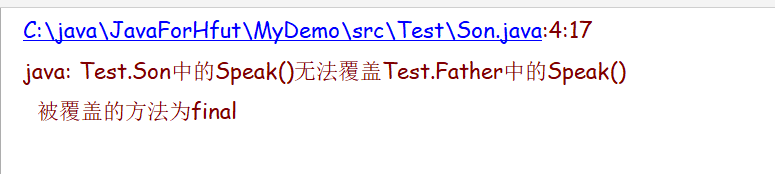
**package** Test;//这是父类

**public class** Father {  
 **public final void** Speak(){  
 System.***out***.println(**"该方法被调用了！"**);  
 }  
}

**package** Test;//这是子类  
**public class** Son **extends** Father {  
 **public void** Speak(){  
 System.***out***.println(**"son重写了该方法！！"**);  
 }  
}

**package** Test;//这是测试类  
**public class** finalWithMethod {  
 **public static void** main(String[] args) {  
 Son son = **new** Son();  
 son.speak();//编译器报错，该方法不可被覆盖  
 }  
}

编译器报错信息如下：



(3)final修饰类，即这个类不可以被继承。也即这个类不能有子类。

代码如下：

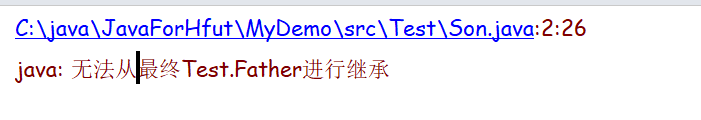
**package** Test;//这是父类

**public final class** Father {  
 **public void** Speak(){  
 System.***out***.println(**"该方法被调用了！"**);  
 }  
}

**package** Test;//这是子类  
**public class** Son **extends** Father {//extends使用错误！Father类不能被继承  
 **public void** Speak(){  
 System.***out***.println(**"son重写了该方法！！"**);  
 }  
}

**package** Test;//这是测试类  
**public class** finalWithMethod {  
 **public static void** main(String[] args) {  
 Son son = **new** Son();  
 son.speak();  
 }  
}

编译器报错信息如下：



1. **知识点总结：**

* **Class类及Class对象的简介：**Class对象表示每个类运行时的一个类型信息的一个表示，也就是说，每个类在编译的时候都会产生一个class对象，这些类在创建实例化对象的时候，就是根据之前Class对象进行创建的。
* **类加载进入内存的过程包含着Class对象的创建：**Class类是在一个类被加载到内存的时候通过JVM以及通过类加载器中的defineClass方法来构造的。类加载器通过获取Class字节码，然后根据字节码在java堆中生成代表这个类的Class对象。
* **类在java程序中采用动态加载方式：**类加载是在对其第一次使用时，动态加载到JVM的，JVM先检查Class对象是否被加载，Class对象载入内存之后，类加载器就会来创建这个类的所有实例化对象
* **获得一个类的Class对象的方式有三种：**

1. 类的全限定名（包括包含该类的包的名称）
2. 通过调用实例对象的getClass方法
3. 通过类字面常量

* **通过类的全限定名方法获得Class对象：**可以不需要创建该类的实例化对象来获取Class对象，个人感觉有点点类似于匿名对象的思想。
* **使用.forName()方法获得Class对象：**此时类的Class对象会自动初始化。
* **同一个类的Class对象是唯一的：**不管以哪种方式获得Class对象，任意一个类对应的Class对象，都通过类加载器和类本身来确定其在JVM中的唯一性，JVM中使用双亲委派模型来组织加载器之间的关系，来保证Class对象唯一性
* **Class类的三种方法的辨析与区别：**getSimpleName和getCanonicalName类似，但是后者返回更加容易理解的，且在内部类和数组中表示形式不同

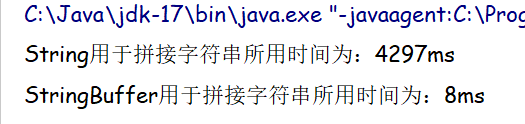
而getName是在JVM中Class的表示的名字，可以用于动态加载Class对象

以上八条知识点的大致逻辑顺序是按照首先对Class的介绍，然后再是其与类加载进内存所关联的创建过程，再是创建了Class对象之后我们该如何获取，介绍了几种方法，最后是对Class类中几个成员方法的辨析。

1. **代码如下：**

**package** Test;  
**public class** TestString {  
 **public static void** main(String[] args) {  
 *//String* String s1 =**""**;  
 **long** timeStart1 = System.*currentTimeMillis*();  
 **for** (**int** i=0;i<100000;i++){  
 s1 += (String.*valueOf*(i));  
 }  
 **long** timeEnd1 = System.*currentTimeMillis*();  
 **long** result1 = timeEnd1 -timeStart1;*//差值表示运行该程序所需要的时间* System.***out***.println(**"String用于拼接字符串所用时间为："**+result1+**"ms"**);  
  
 *//StingBuffer* StringBuffer sb = **new** StringBuffer();  
 **long** timeStart2 = System.*currentTimeMillis*();  
 **for** (**int** i=0;i<100000;i++){  
 sb.append(String.*valueOf*(i));  
 }  
 String s2 = sb.toString();  
 **long** timeEnd2 = System.*currentTimeMillis*();  
 **long** result2 = timeEnd2 - timeStart2;  
 System.***out***.println(**"StringBuffer用于拼接字符串所用时间为："**+result2+**"ms"**);  
 }  
}

**代码运行结果为：**



**可见StringBuffer用于拼接字符串的效率尤其当任务量大时，明显比String要高得多。**

1. 子类重写（覆盖）父类方法所需要遵循的原则有：
2. 子类重写的方法名、方法的参数列表必须与父类的方法相同

**理解**：若重写的方法名不同，则这是创造了一个新的方法；若参数不同，则这是进行了重载，并不是重写。

1. 子类重新定义的方法，不能比父类的方法有更严格的的访问权限。

**理解**：里氏原则中，子类必须能够替换成它们的基类。也就是说，父类可以调用的方法，子类也一定可以调用。

1. 子类重新定义的方法，不能比父类的方法传递更多的异常。

**理解**：根据里氏原则，若某处父类的方法抛出异常可被处理，但替换成子类之后，重写的方法若是抛出更多异常，此处不会处理抛出的更多异常，则会造成代码出错，程序不能运行。

1. 当子类需要重载父类中方法的时候，子类方法的形参要比父类方法输入的参数更宽松，也就是范围更广。

**总结**：遵循里氏原则在实际开发中是程序员应当牢记在心的最基础原则，可以提高我们程序的健壮性和后期的扩展以及移植。