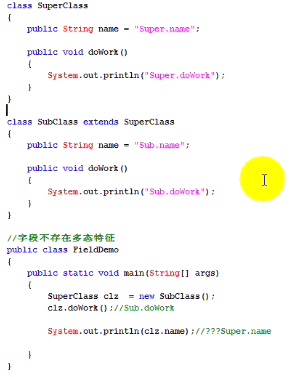
Day 09

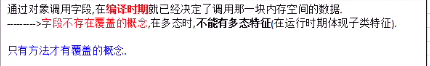
**01字段不存在多态特征**

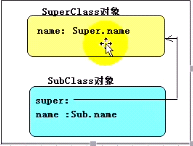


通过对象调用字段，在编译时期就已经决定了调用哪一块内存空间的数据，因为字段不存在多态的概念，在多态时，不能有多态特征。

而且子类对象的存在是以父类的对象为前提的（因为构造函数会默认或者显示的调用父类的构造函数）

只有方法才有覆盖，字段只能叫隐藏，而且字段不存在多态。

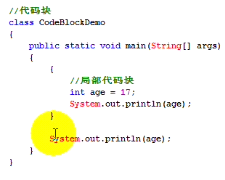


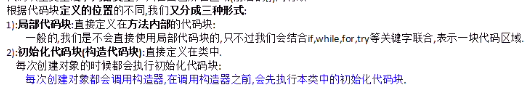


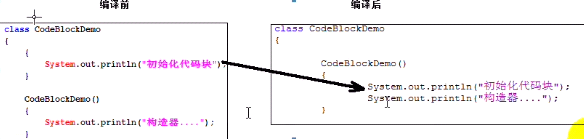
父类和子类的字段都是各自存储的。

**02 代码块语句**





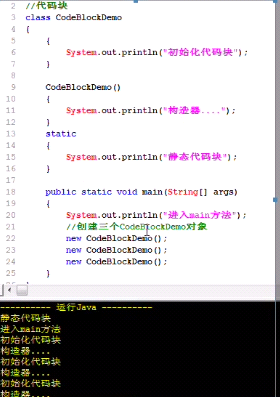




通过反编译工具，初始化代码么块在编译后也是作为构造器的最初语句。

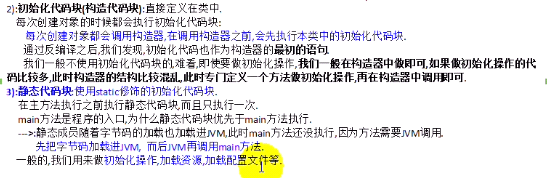
我们一般不使用初始化代码块，即使要做初始化操作，我们一般写在构造器中即可。

静态代码块：



静态代码块在主方法之前执行，而且只执行一次。

因为静态成员随着字节码的加载也加载进jvm，此时main方法还未执行，因为方法需要jvm调用。



03代码执行顺序

A类依赖B类，则B类先编译

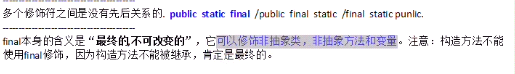
非Static字段的初始化其实都是在构造器中来执行的，即使书写时已经初始化，即：

Public string cc=”cc”; 底层实际也是在构造器中执行的而且优先于构造器中的其他代码。

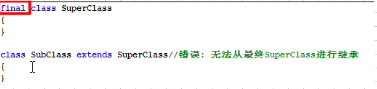
静态成员在底层是在静态代码块中做初始化的，而非静态成员是在构造器中做的出初始化，无论定义时是否指定值都是如此。

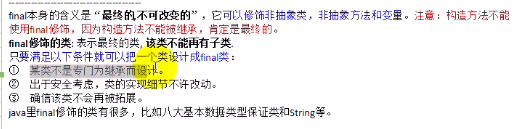
04final类和final方法



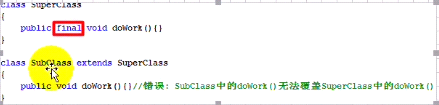


Final修饰的类表示最终类，该类不能在有此类





Final修饰的方法：最终方法，该方法不能被子类覆盖



构造器本来就是最终的，因此没有必要再用final修饰

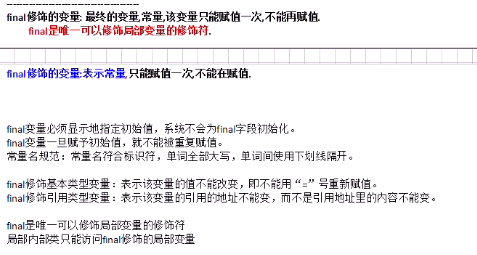
Final修饰的字段



05final变量

Final修饰的变量：最终的变量，改变量只能是常量，只能赋值一次不能子啊赋值

Final是唯一可以修饰局部变量的修饰符



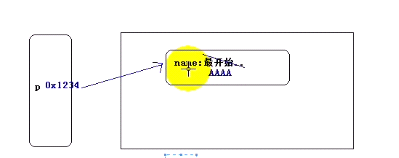
Final修饰的字段必须显示的指定初始值，系统不会为final字段初始化。

命名规范：全部使用大写字母，多个单词之间使用\_隔开

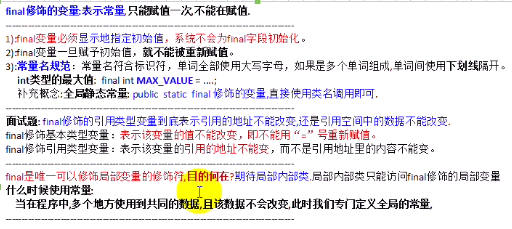
全局静态常量：public static final 修饰的变量



Final修饰的基本类型不能被再赋值；final修饰的引用类型变量的引用地址不能变，而不是引用地址的数据不能变。



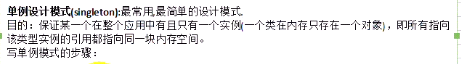


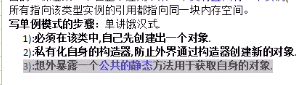




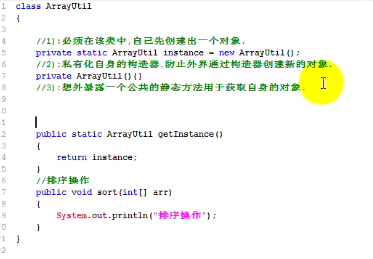
06单例设计模式

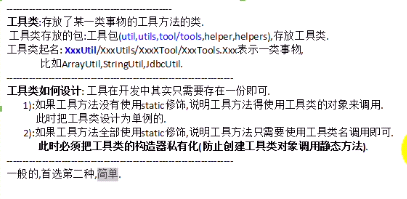






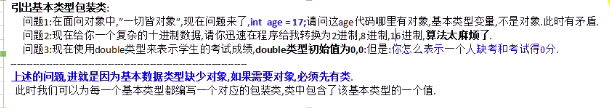
07工具类的设计





官方的工具类也是如此设计的

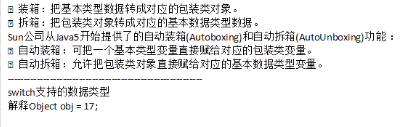
08引出基本类型包装类

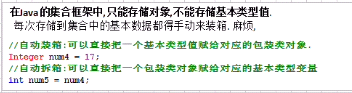


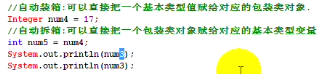
09装箱和拆箱操作



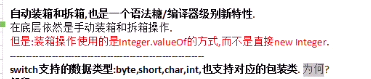


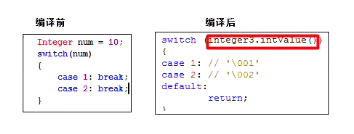






自动装箱和自动拆箱也是语法糖





Object obj=17是正确的：

自动装箱为Integer

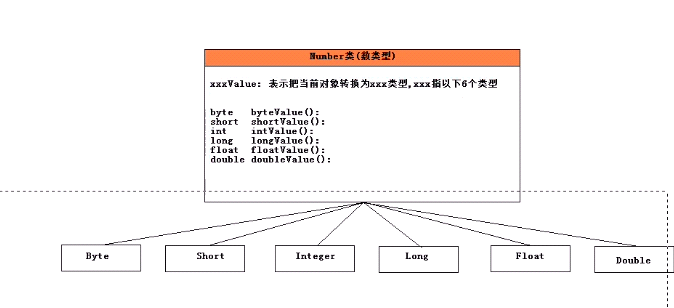
多态

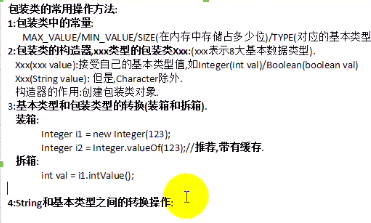
Object可以接受一切数据类型的值。

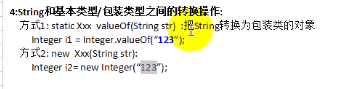
Object[]可以装一切类型

Object arr=[“a”,1,’a’]

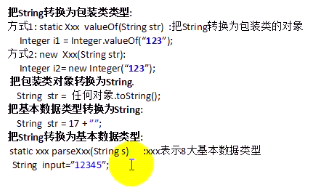
10B包装类的常用操作

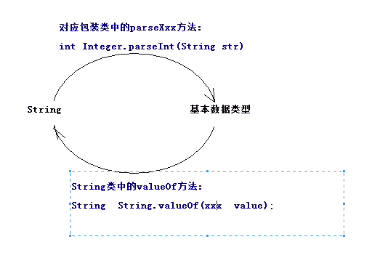




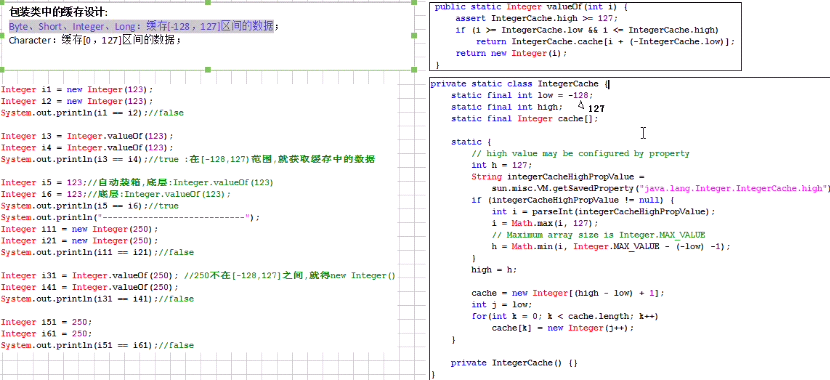






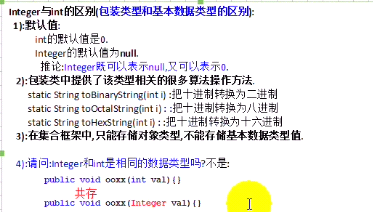


11包装类中的缓存设计



享元设计-本质就是缓存设计

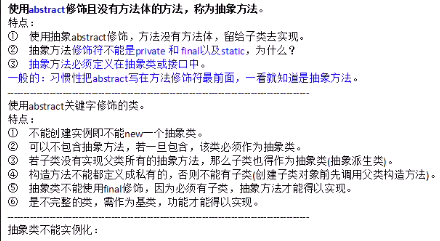
12integer和int的区别



开发中建议使用包装类型，基本数据类型存储在栈中方法的栈帧中，包装类存储在堆中，操作栈较快。

13引出抽象方法设计

14抽象方法和抽象类

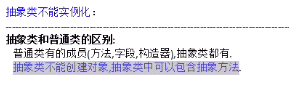


抽象类即使创建出类对象，调用抽象方法根本没有方法体。

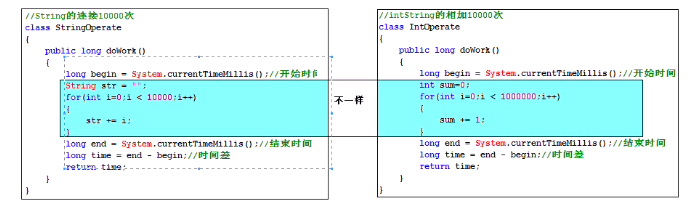
抽象类必须有子类才有意义。

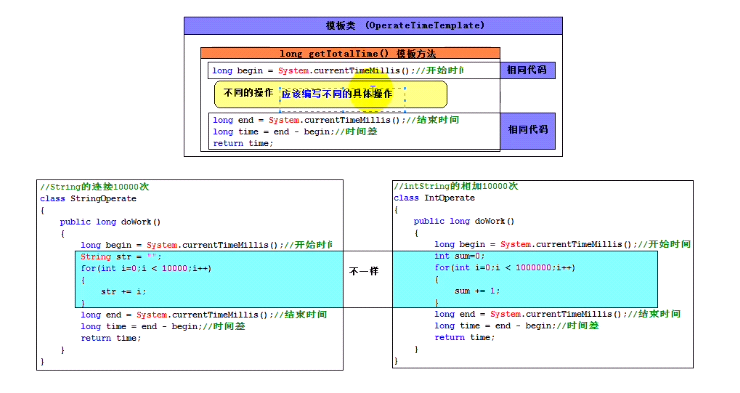




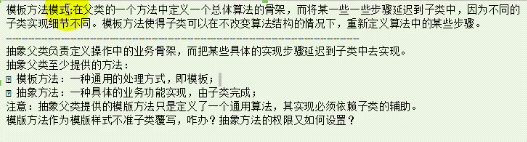


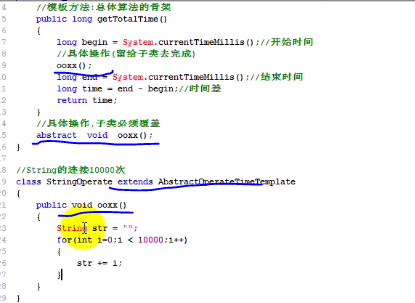
15引出模板方法设计模式

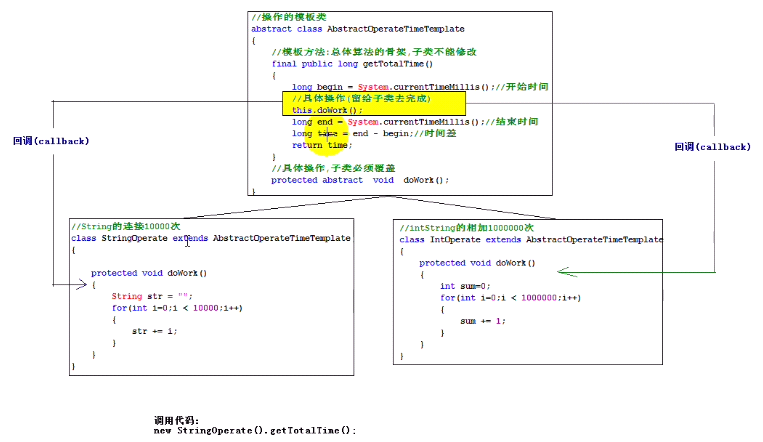




16分析模板方法设计模式







17小结

