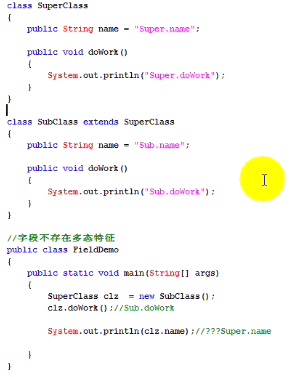
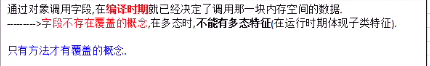
Day 09

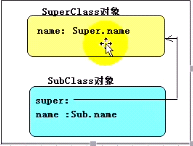
01字段不存在多态特征



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。

只有方法才有覆盖，字段只能叫隐藏，而且字段不存在多态。

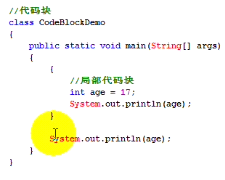


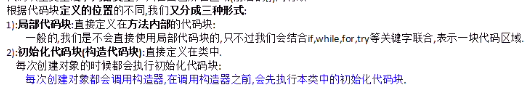


父类和子类的字段都是各自存储的。

02 代码块语句





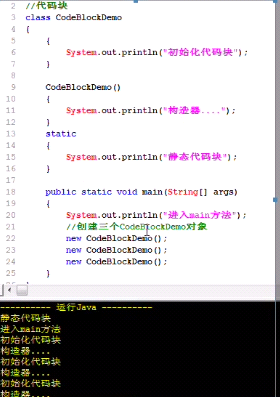




通过反编译工具，初始化代码么块在编译后也是作为构造器的最初语句。

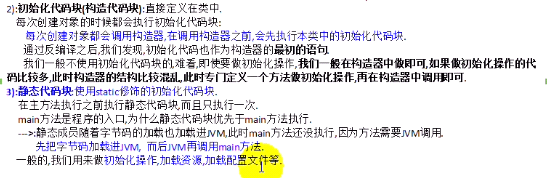
我们一般不使用初始化代码块，即使要做初始化操作，我们一般写在构造器中即可。

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

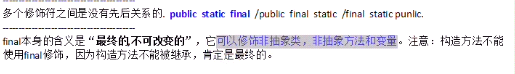
非Static字段的初始化其实都是在构造器中来执行的，即使书写时已经初始化，即：

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。

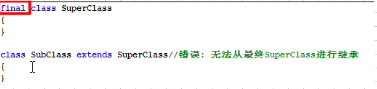
静态成员在底层是在静态代码块中做初始化的，而非静态成员是在构造器中做的出初始化，无论定义时是否指定值都是如此。

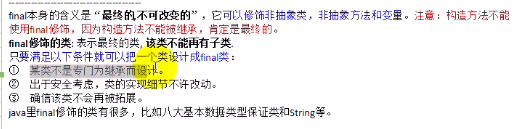
04final类和final方法



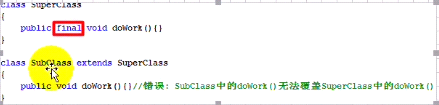


Final修饰的类表示最终类，该类不能在有此类





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

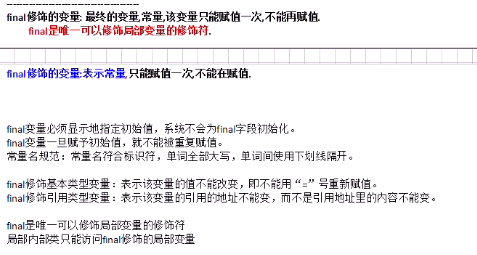
Final修饰的字段



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



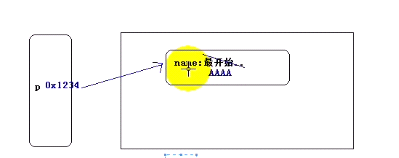
Final修饰的字段必须显示的指定初始值，系统不会为final字段初始化。

命名规范：全部使用大写字母，多个单词之间使用\_隔开

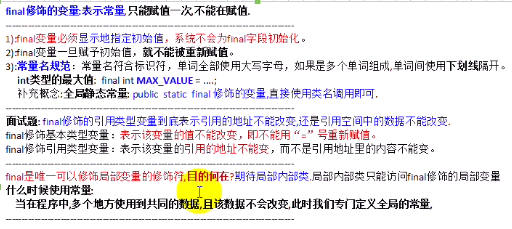
全局静态常量：public static final 修饰的变量



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。



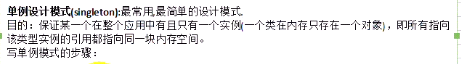


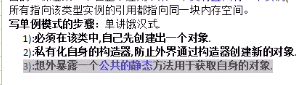




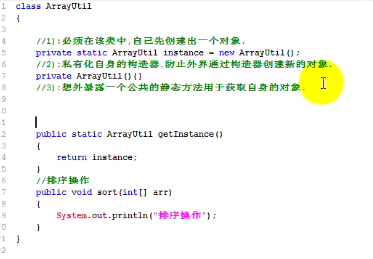
06单例设计模式

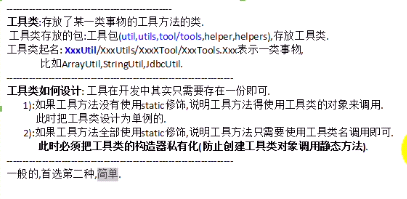






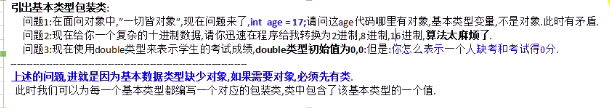
07工具类的设计





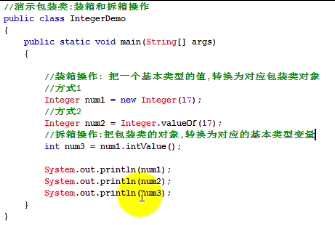
官方的工具类也是如此设计的

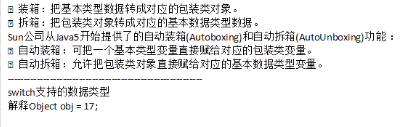
08引出基本类型包装类

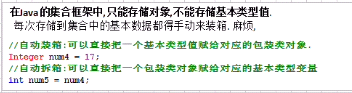


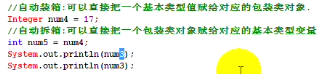
09装箱和拆箱操作



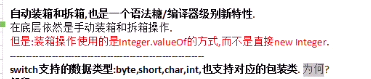


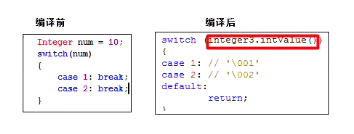






自动装箱和自动拆箱也是语法糖





Object obj=17是正确的：

自动装箱为Integer

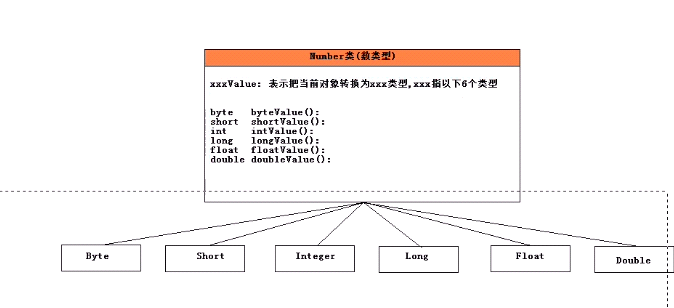
多态

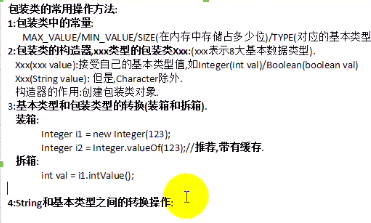
Object可以接受一切数据类型的值。

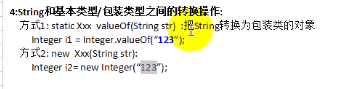
Object[]可以装一切类型

Object arr=[“a”,1,’a’]

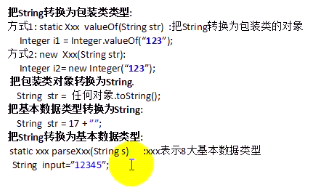
10B包装类的常用操作

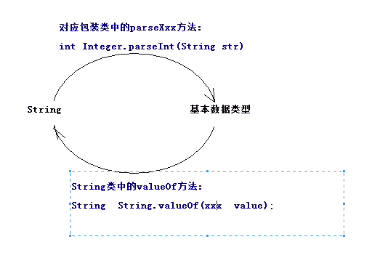




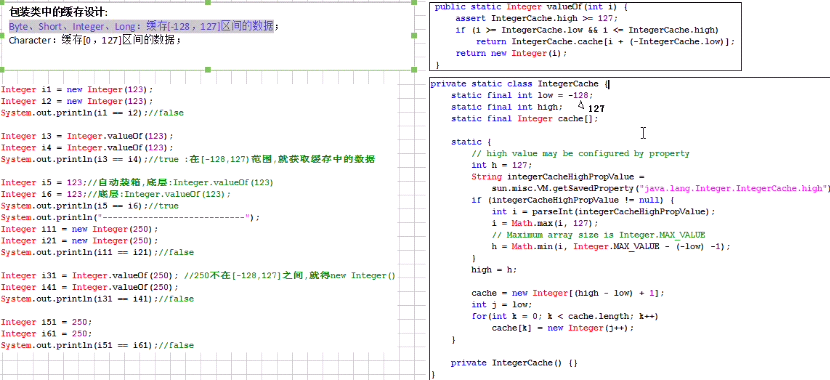






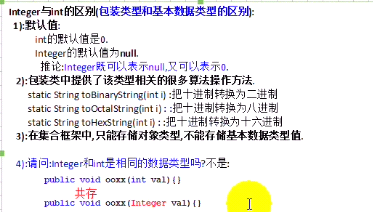


11包装类中的缓存设计



享元设计-本质就是缓存设计

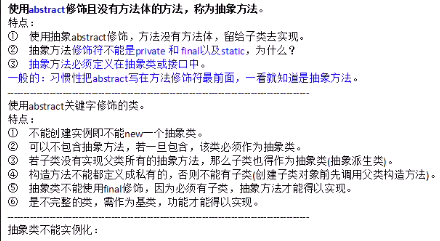
12integer和int的区别



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

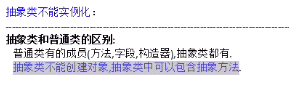


抽象类即使创建出类对象，调用抽象方法根本没有方法体。

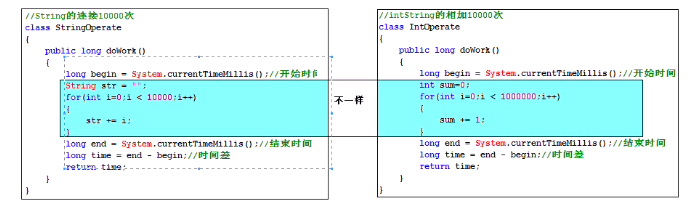
抽象类必须有子类才有意义。

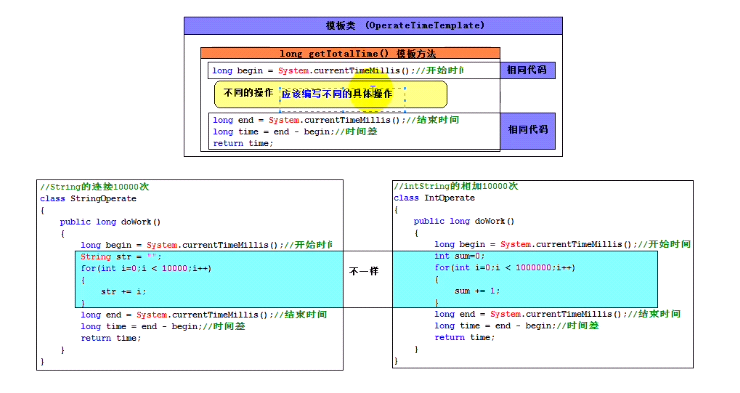




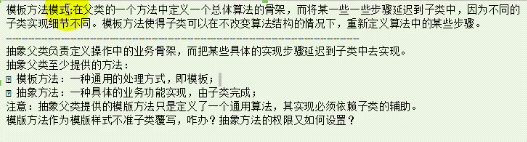


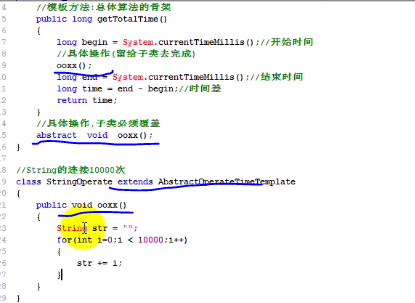
15引出模板方法设计模式

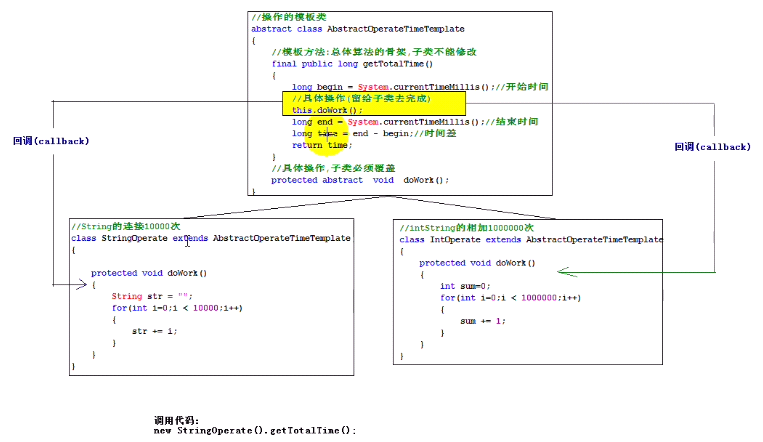




16分析模板方法设计模式







17小结



Day 10

01什么是接口和为什么需要接口



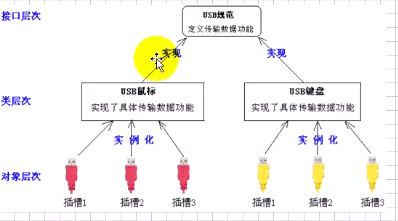








USB是一种协议（规范）：应该规范所有的useb设备具有的功能。



面向接口的编程：

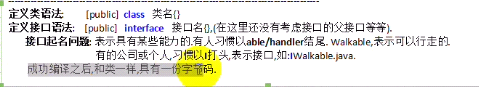
接口和实现类体现了真正的多态。

02接口的定义和接口的成员







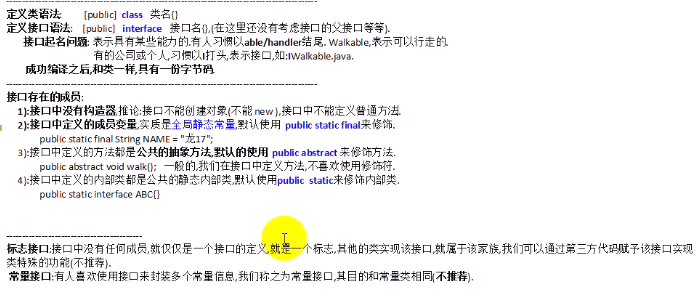


一个接口编译后就会生成一个字节码文件

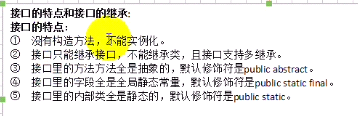
接口存在的成员：

* 接口中没有构造器，接口与不能创建对象
* 接口汇总定义的成员变量，实质是全局静态常量，默认使用public static final修饰
* 接口中定义的方法都是公共的抽象方法，默认使用public abstract修饰
* 接口中定义的内部类都是public static修饰的





03接口的特点和接口的继承



接口不能显示的定义构造器

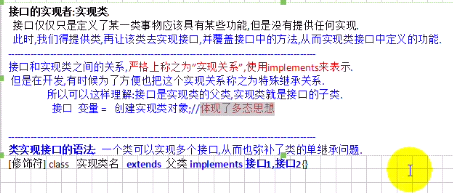
类是单继承，接口支持多继承

接口继承关键词为extends

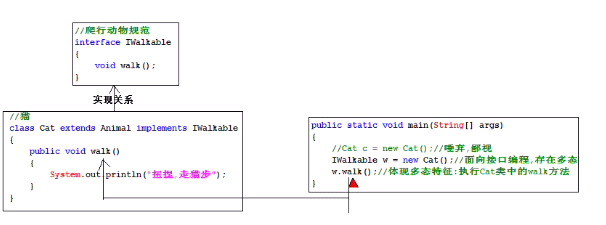
类实现接口关键词为implements

04接口的实现关系

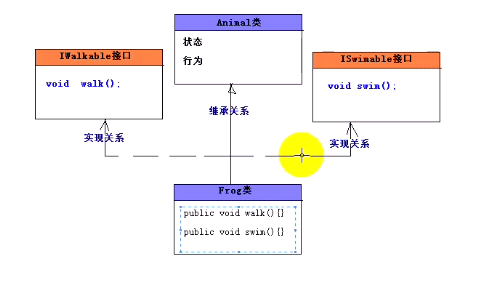




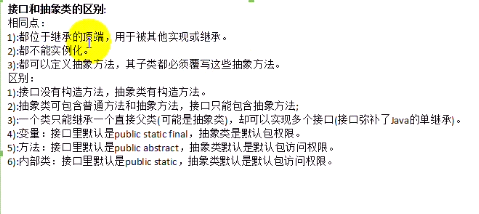




05类和类以及类和接口的关系图



06接口和实现类的区别



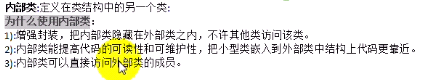
如果接口和抽象类可完成相同的功能，尽量使用接口，面向接口编程

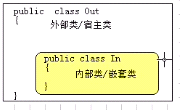
07面向接口编程思想

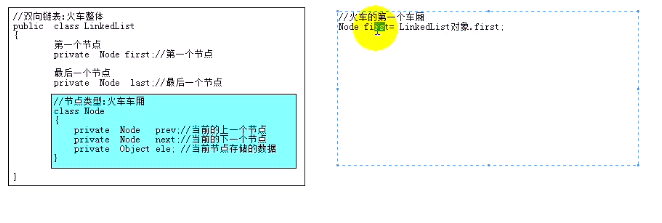




08内部类概述





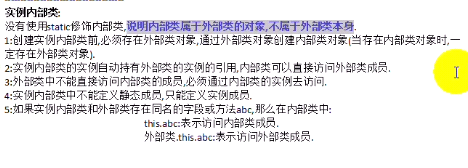


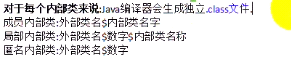


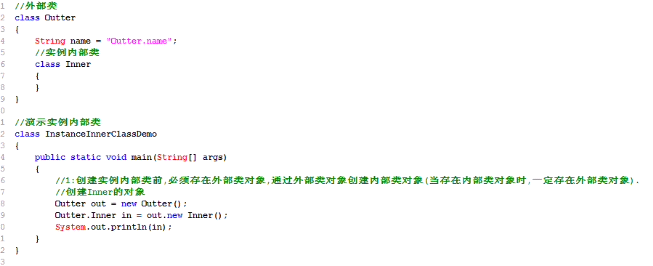
09实例内部类和静态内部类分析



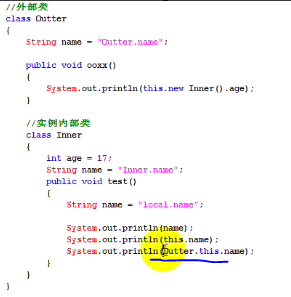
实例内部类：



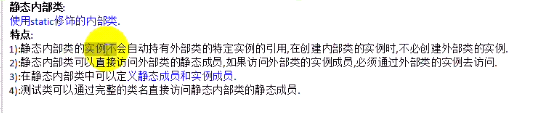




内部类访问外部类成员使用outter.this.字段或方法名



静态内部类：

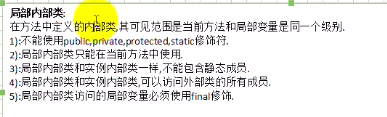


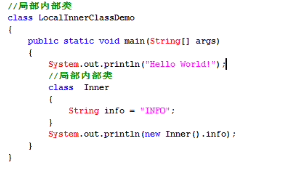
有静态内部类的时候可以没有外部类对象

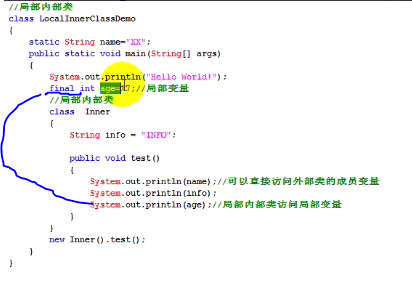


10局部内部类

可以把局部内部类当做局部局部变量



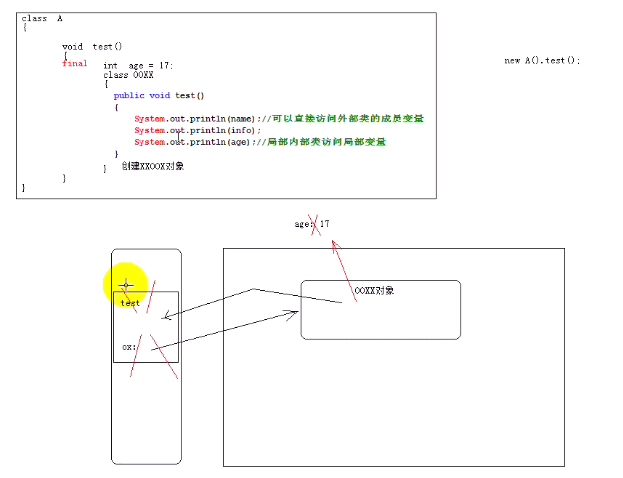






原因：如果当前方法不是main方法，那么当前方法调用完毕之后，当前方法的栈帧被销毁，方法内的局部变量的空间全部被销毁的时候，还存在堆内存，依然持有对局部的引用，但是方法被销毁的时候局部变量已经被销毁了。、

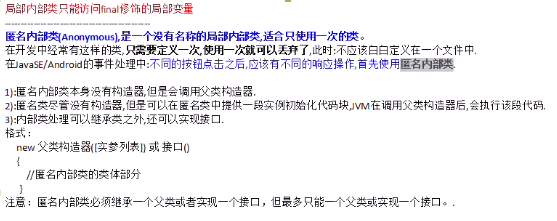
此时出现：在对内存中，一个对象引用这一个不存在的数据，为了不免该问题我们使用final修饰局部变量，从而编程常量永驻内存空间，即使方法销毁后也可以被持有。



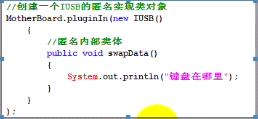
Age作为常量时就会被放在常量池里面，所以当方法调用完毕栈帧被销毁后仍然能够存在，而此时未被垃圾回收器回收的对象也可以引用此常量。

局部内部类近乎不会使用。

11匿名内部类分析



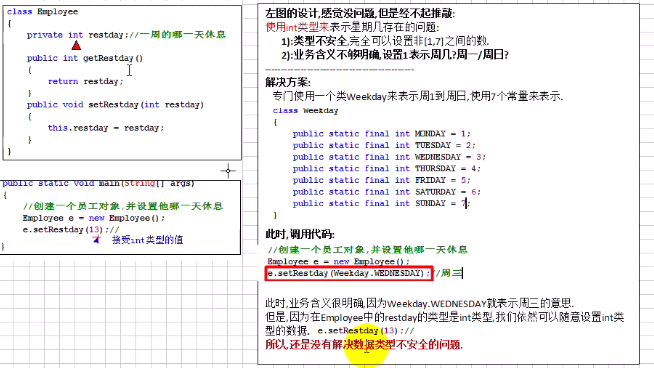
匿名内部类连类名都没有因此也就没有构造器。



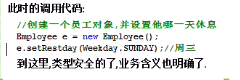
虽然是new接口，这只是语法格式，表名该匿名类实现了此接口。

匿名内部类访问局部变量也要加final修饰符，因为匿名内部类是特殊的局部内部类。

12引入枚举类型和枚举的模拟



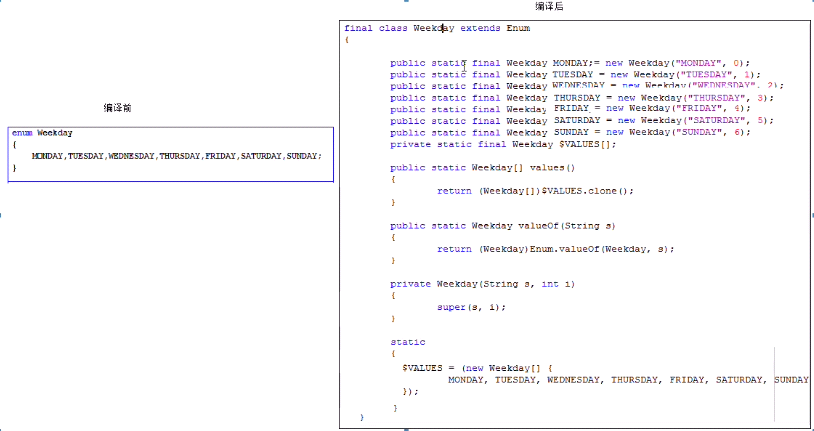


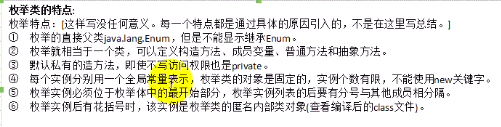


13枚举的定义和特点以及底层分析



反编译即可

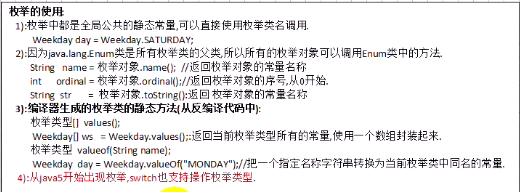


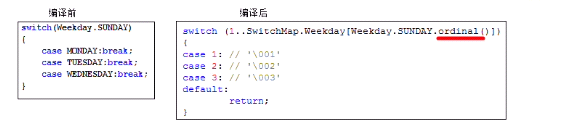


底层实现是

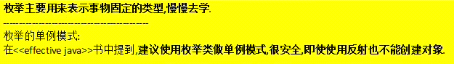
枚举底层没有无参数构造器

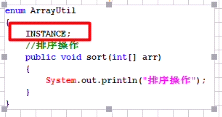
14枚举的操作细节





Switch只支持int类型，其他的都是自动转换了或者做了其他的改变





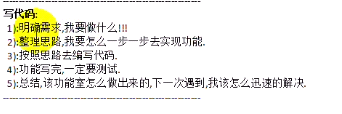
15小结

、

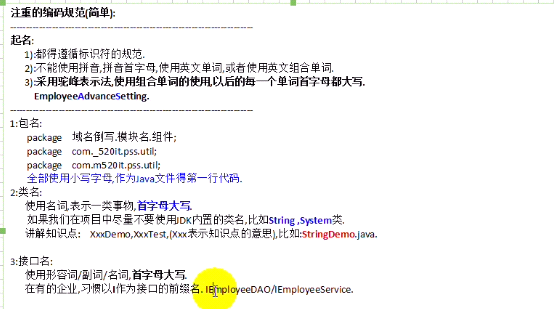
Day 11

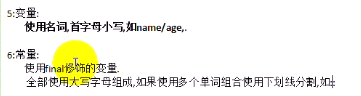
01为什么得有编码规范





02开发中命名规范概述







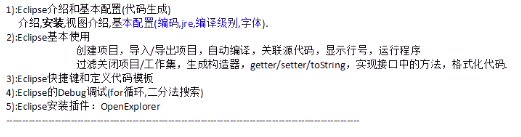
03java开发工具介绍和eclipse概述

04eclipse的安装和基本配置

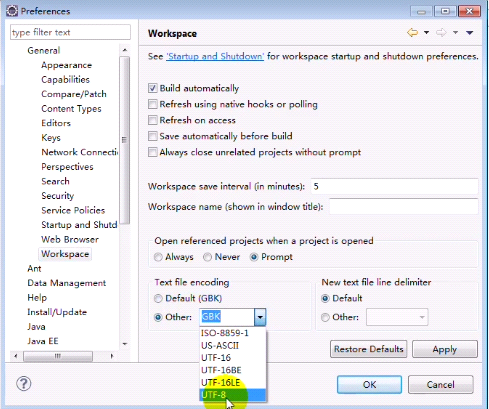
设置工作空间—>工作目录

偏好设置视图

偏好—reset



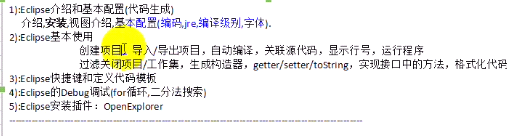
编码：



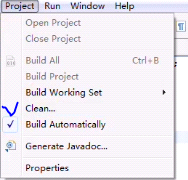
当前项目：右键属性

05eclipse基本使用上

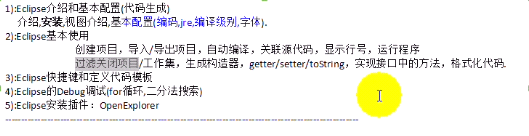
.metadata存储 当前工作空间的配置信息



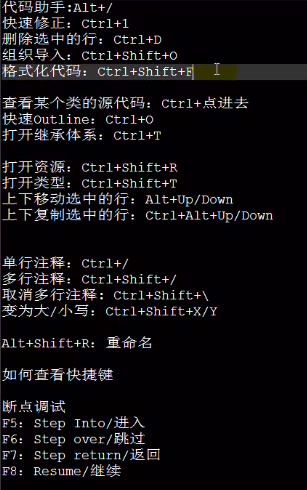
重新生成所有的字节码文件



06eclipse的基本使用下

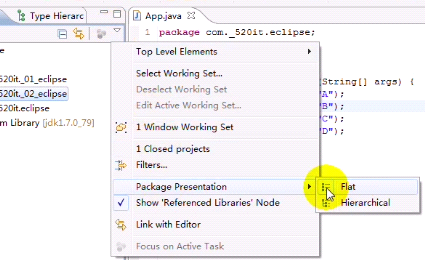


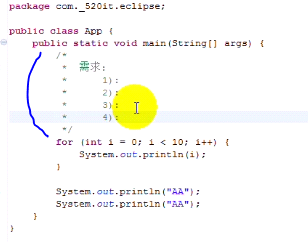
07eclipse常用快捷键



08eclipse定义和修改模板



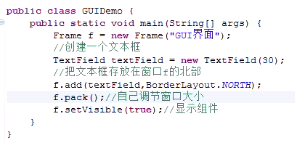




10gui简单操作和匿名内部类

Awt包

Javax包的swing



11制作java文件-jar包和jar包使用

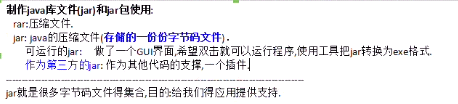
NetBeans javaFX gui制作工具

Jar—类四java的压缩文件（因此可以使用wrar打开），存储的是java的字节码文件

* 可运行的jar:做了一个GUI界面，希望双击就可以运行程序，所使用工具吧jar转换为exe格式
* 作为第三方的jar：作为其他代码的支撑

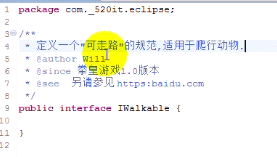
添加jar:添加到lib后建立buildPath

导出jar包：右键导出选择jar包即可



12java制作文档

编写代码注释：



使用@符号输出文档注释提示

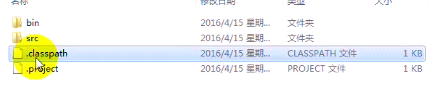


选中文件或者整个项目或者工程，右键导出javadoc即可

13eclipse安装插件

Dropins形式安装

14eclipse下的classpath文件



工程下的文件

Src 源文件

Bin字节码文件

.project 项目的配置文件

.classpath



Kind表示类别

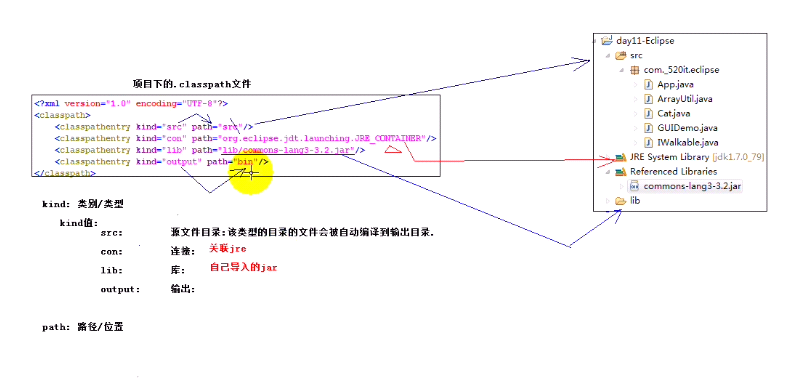
Kind值：src 源文件目录：该目录下的文件会被自动编译到输出目录

Con 连接：关联jre

Lib 库：自己导入的jar文件目录

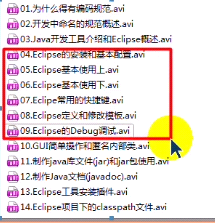
Output 输出目录，默认为bin：编译之后产生的文件

Path表示路径





15eclipse使用小结



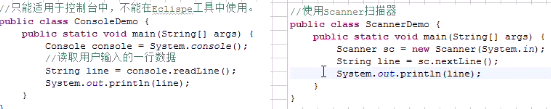
16java核心内库

Run configuration配置运行参数

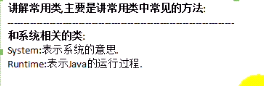
在控制台运行带有报名的文件要带有报名，即全限定名称

用户交互：1 运行之前传入参数

2 运行过程中从控制条传入参数



17常用类-系统相关类

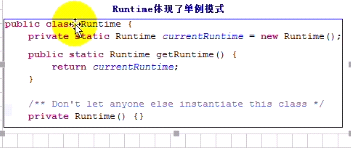


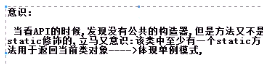




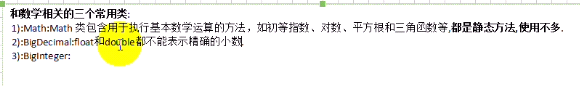
Gc：垃圾回收器方法一般不使用

Runtime类就是单例





18和数学相关的常用类

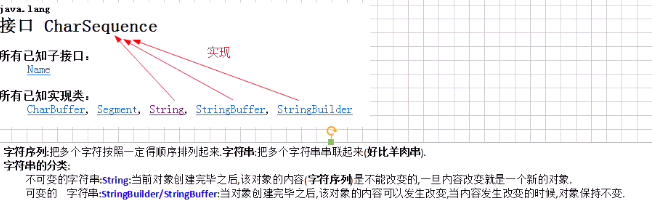


19小结



Day 20

01字符串相关的类

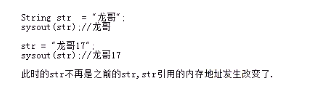


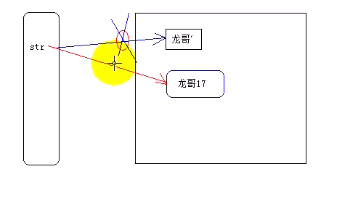


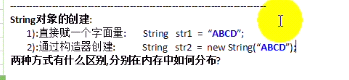
String其实就是对char数组的封装

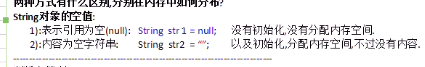
02深入string类分析

String对象：不可变的对象，一旦值改变就是一个新的对象

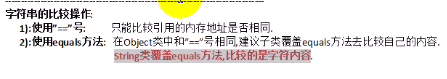


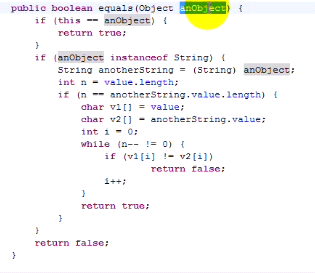










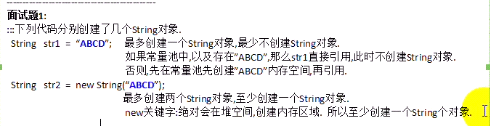


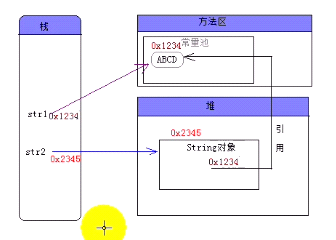
03string面试题分析

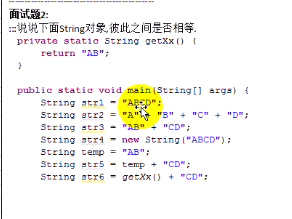
常量池：编译常量池和运行常量池，都存在于方法区中。

编译常量池：把字节码加载进jvm的时候，存储的是字节码的相关信息（一般不做研究）

运行常量池：存储常量数据（研究）







编译器优化

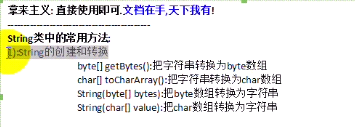
编译器优化会将str=”A”+”B”+”C”编译为就是str=”ABC”,可以查看反编译结果

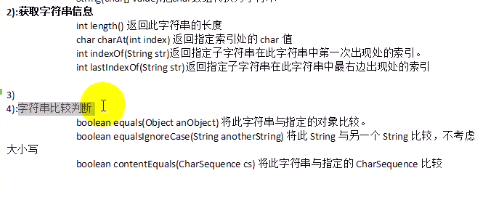
编译时期不能确定变量值为多少所以不能优化，只有在运行时期才能确定；所以编译只会检查语法





04string中的常用方法





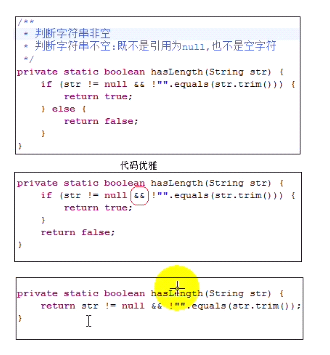


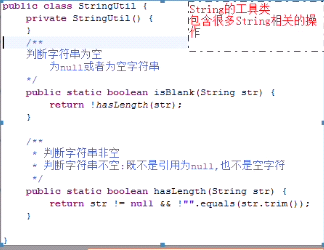
05常用类string的操作练习

06string的工具类封装



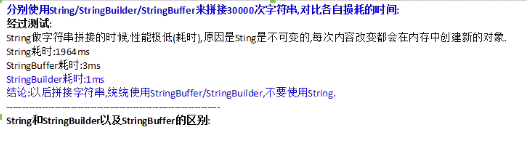
将常量放在前面可以防止str为null，可以不用判断str是否为null了

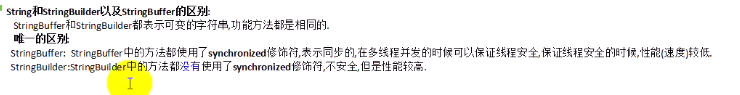




工具类：静态调用就私有化构造器

07string和stringBuilder以及stringBuffer的区别

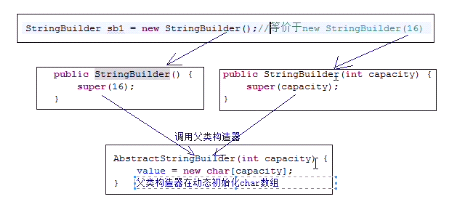


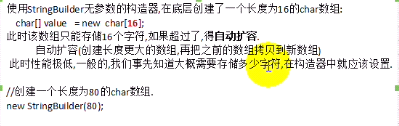


08stringbuilder的常用操作



查看源代码

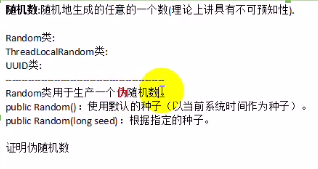






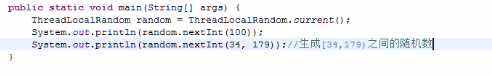
09随机数









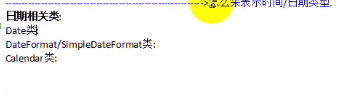






10生成验证码

11日期类

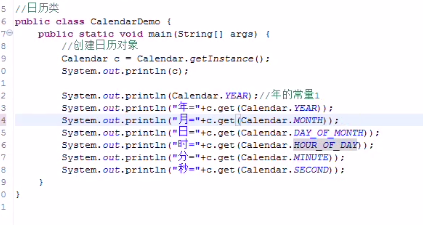


12日期格式化



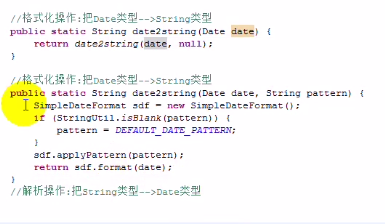


13日历类



抽象方法可以调用，因为运行时是子类实现的，多态

14实现日期的格式化和解析工作



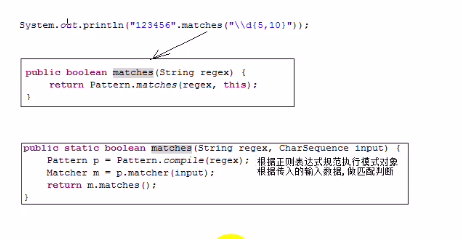
设置格式🡪返回格式化的结果

15日历操作练习

16正则表达式







17小结

