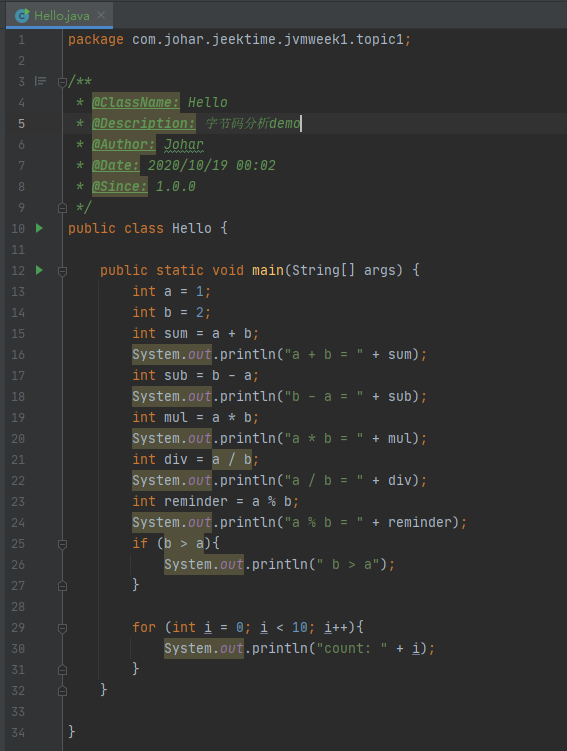
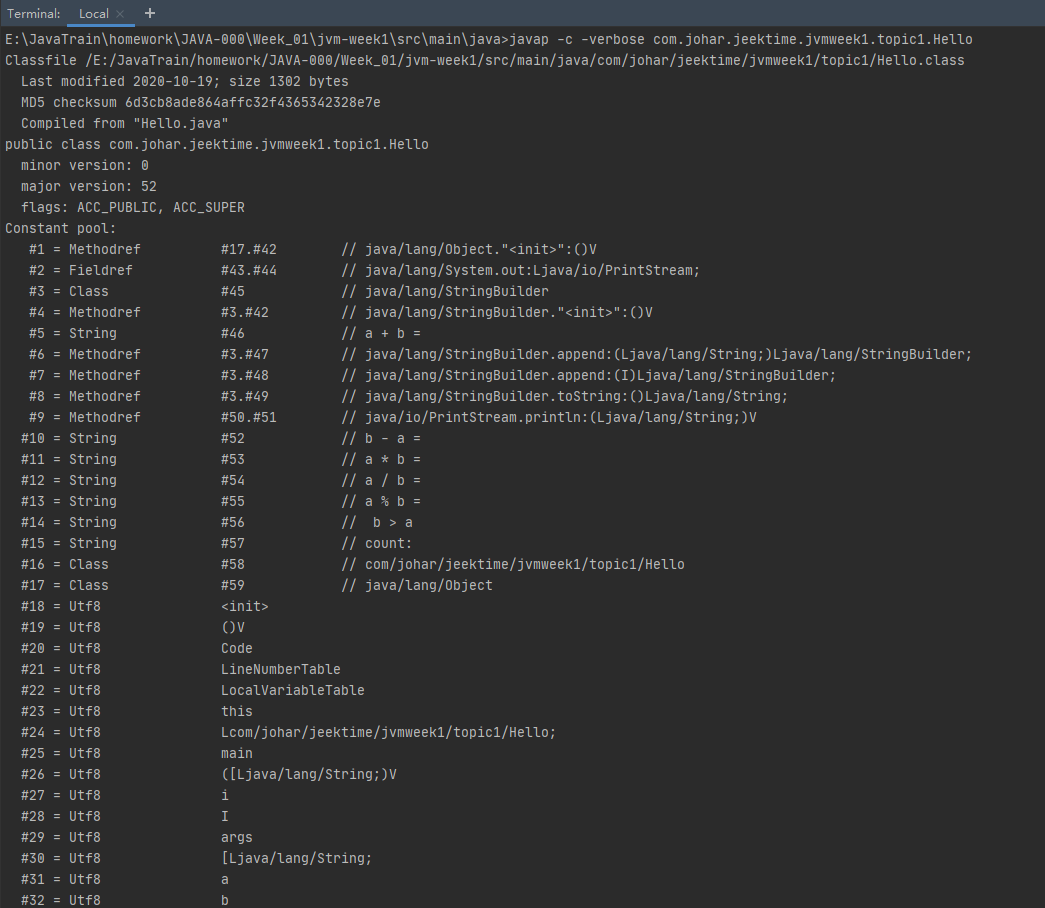
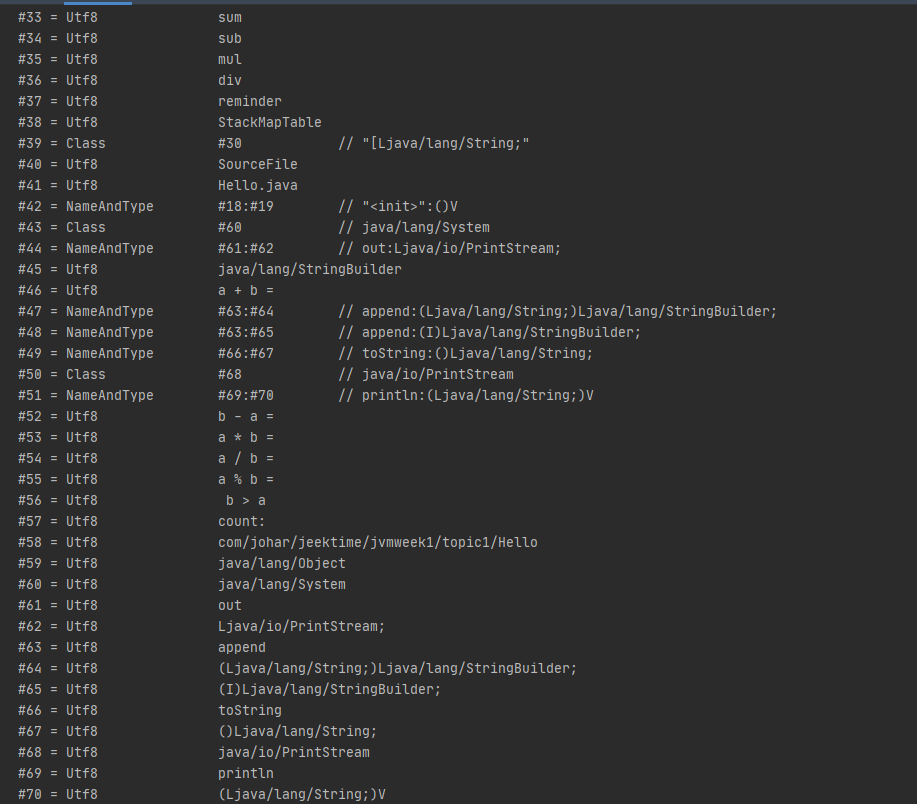
**Hello字节码分析**

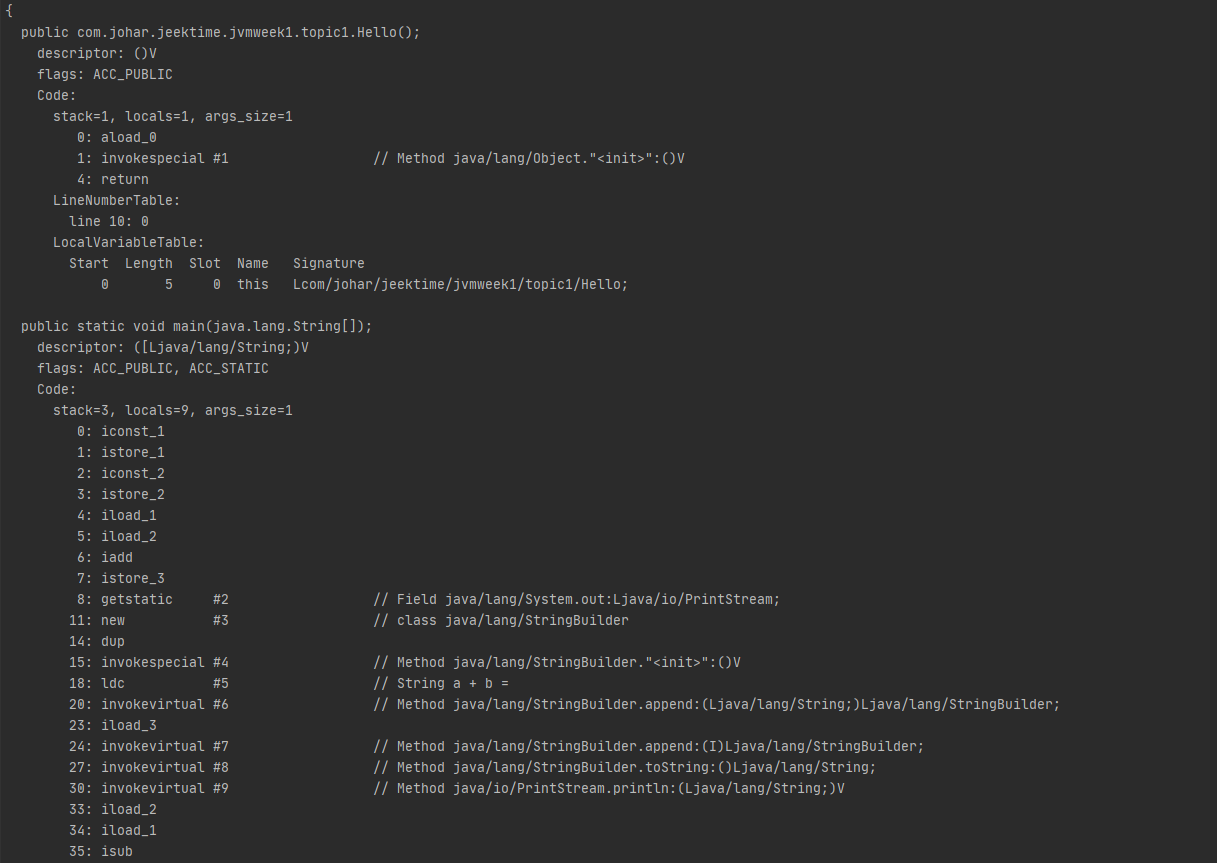
# 源码

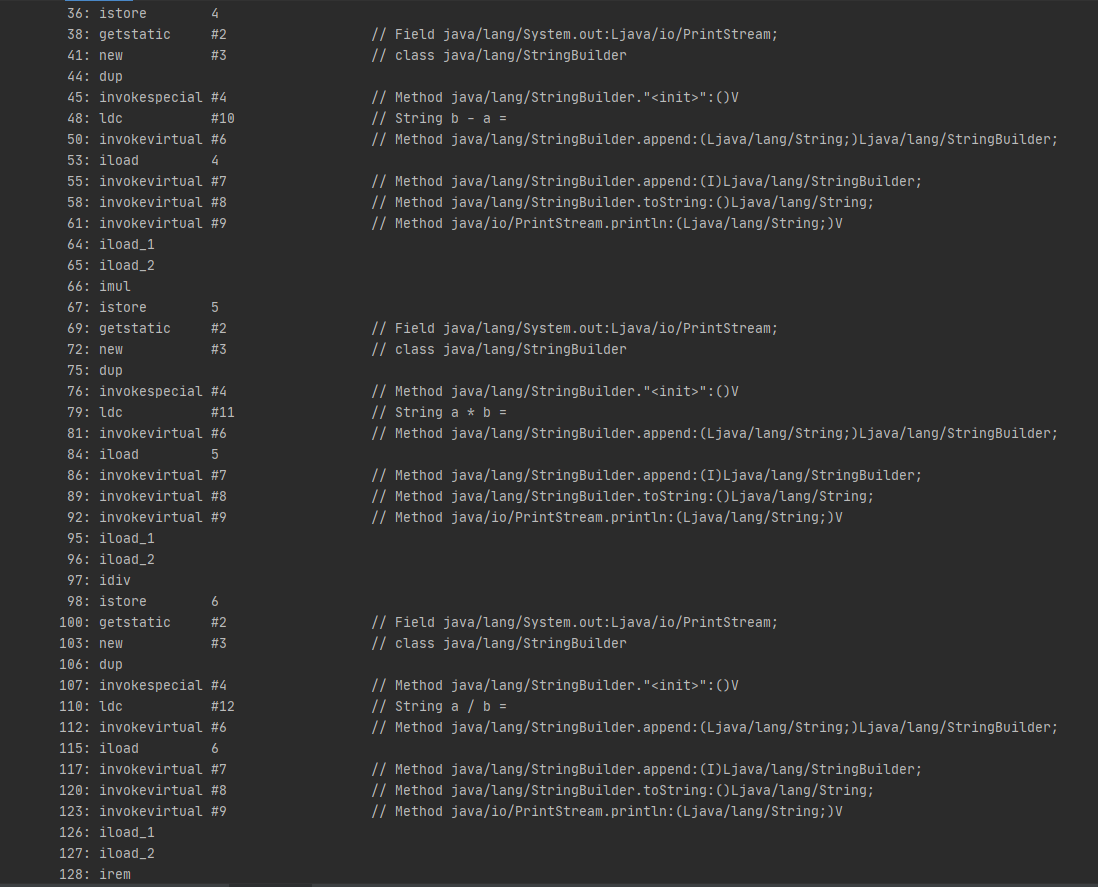


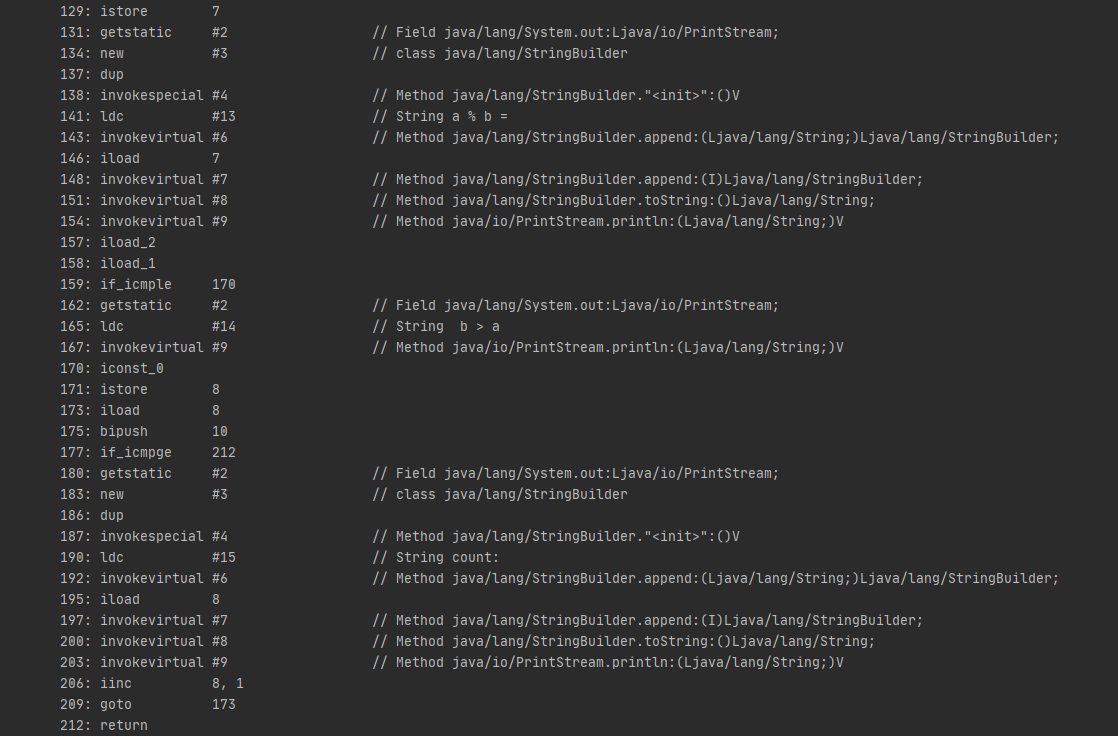
# 字节码

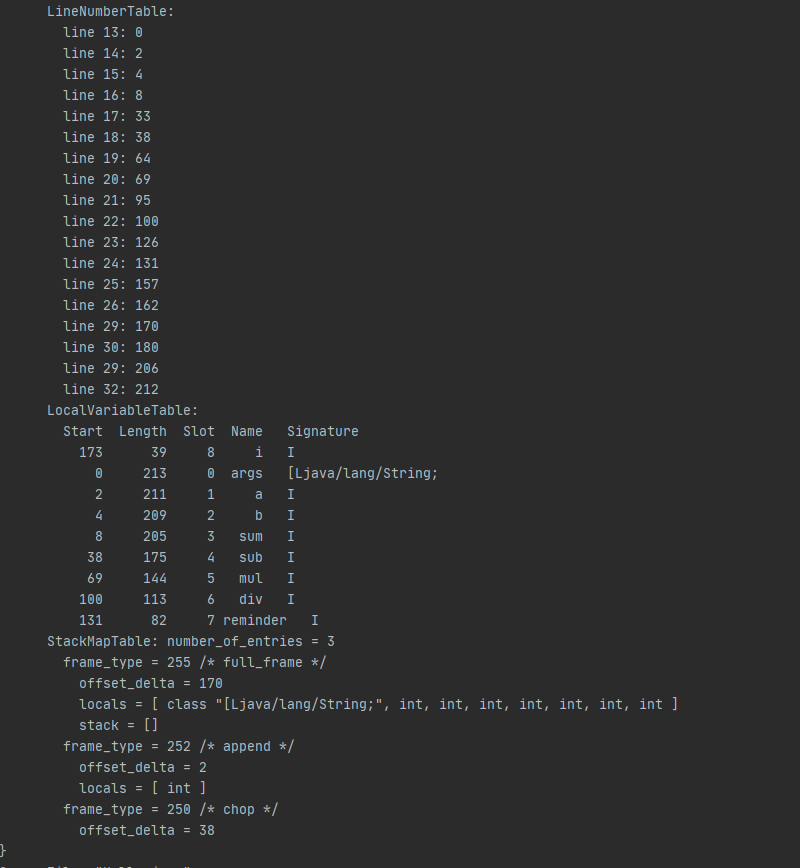








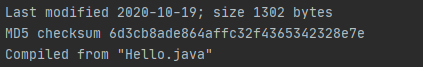




# 字节码分析

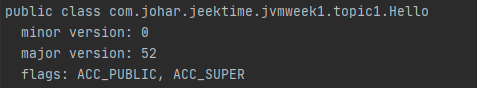
为了方便对字节码文件的理解，个人把字节码文件由文件信息、类信息、常量池、方法区四个部分构成，其中方法区有多个，又主要分成：方法描述、代码（代码逻辑、LineNumberTable、LocalVariableTable）。

## 3.1文件信息



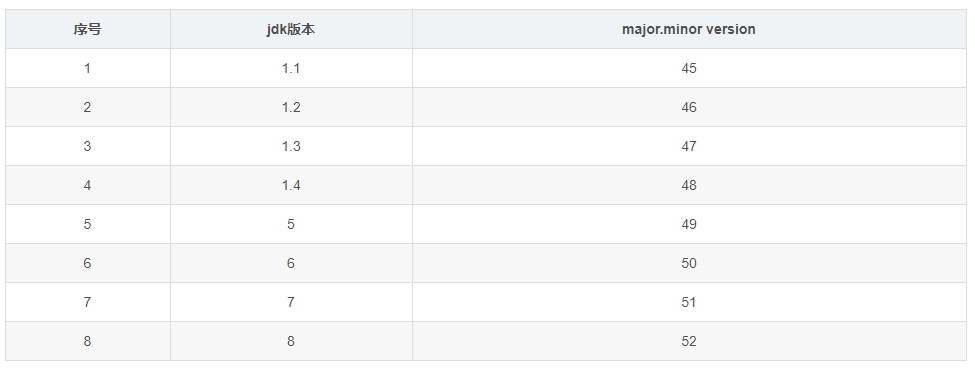
如上图所示，依次为：编译时间、MD5校验和、从哪个文件编译的。

## 3.2类信息



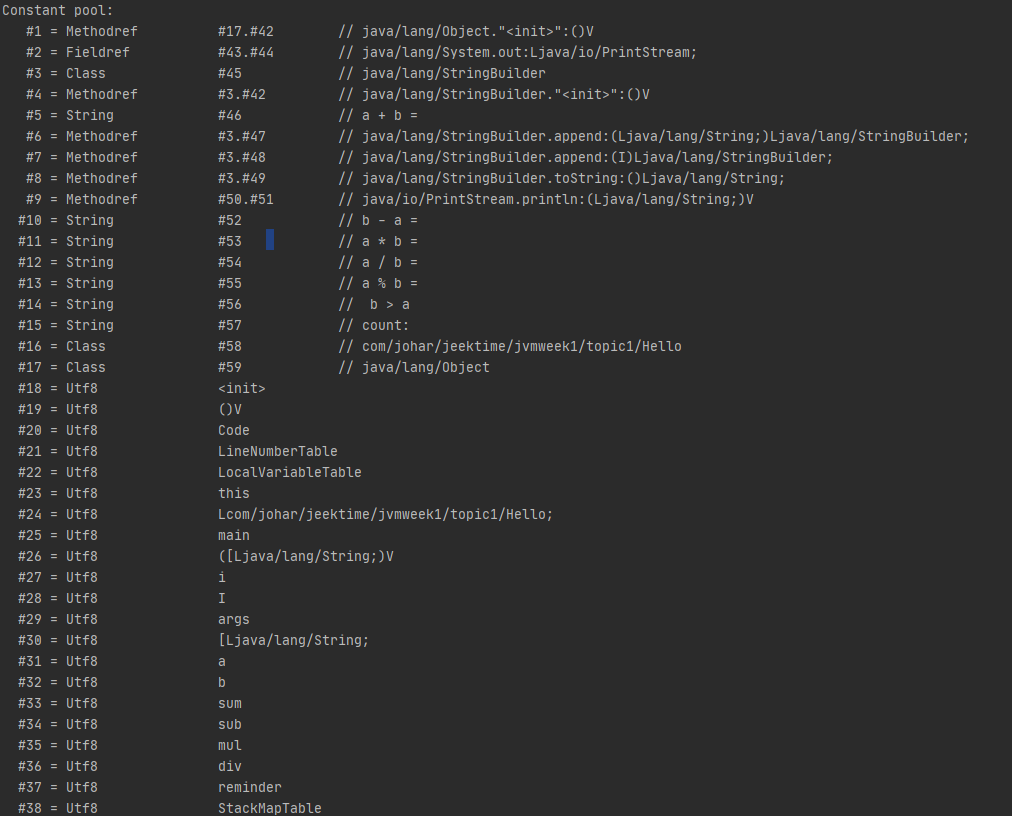
如上图所示：依次为全量类名、JDK版本信息、访问标识符。

JDK版本与major.minor版本关系对应表



ACC\_PUBLIC标识public；ACC\_SUPER主要用来修正invokespecial调用super类方法的问题

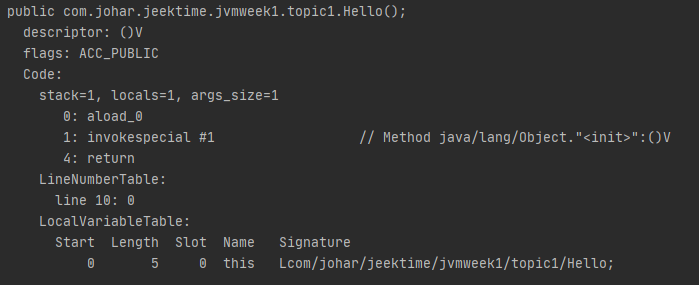
## 3.3常量池



指令后面使用类似#1、#2、#3编号主要是为了常量池的引用。在常量池这个字典表中，存储着方法（MethodRef）、字段（FieldRef）、类（Class）、字符串（String）等。

## 3.4方法区

### 3.4.1 默认构造函数



方法描述中（）V表示：没有入参，返回值得类型为void

ACC\_PUBLIC：表示该方法的访问标识是pubic

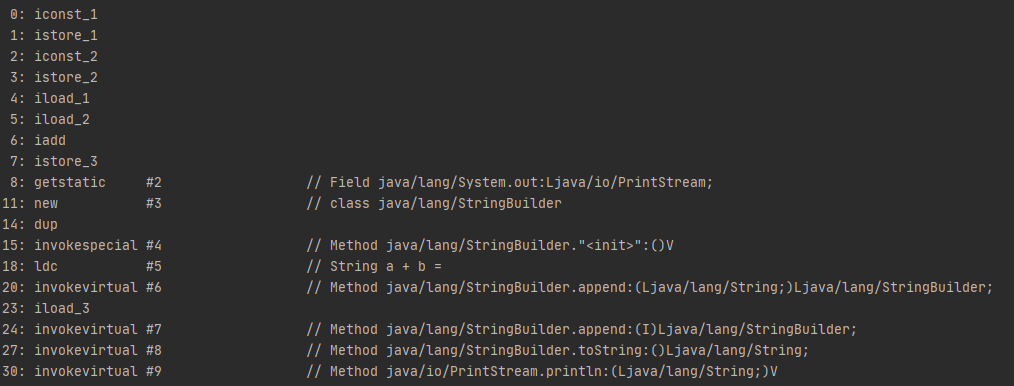
stack=1,locals=1, args\_size=1 是个很好玩的地方，和上面的方法描述表面看起来是有点冲突，但是实际上，Java非静态方法将this分配到0号位置，可以在LocalVariableTable中看见；但是静态方法中没有this引用。

aload\_0:表示将this装载到操作数栈的栈顶

Invokespecial：表示调用构造方法

### 3.4.2 main函数

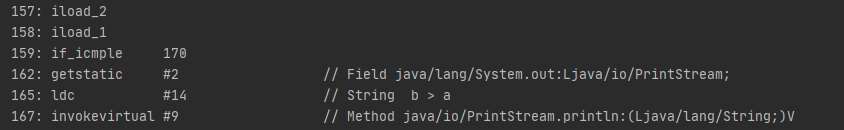
main函数中就跳过其他的分析，直接看其中的代码逻辑。



上图是加法计算部分：

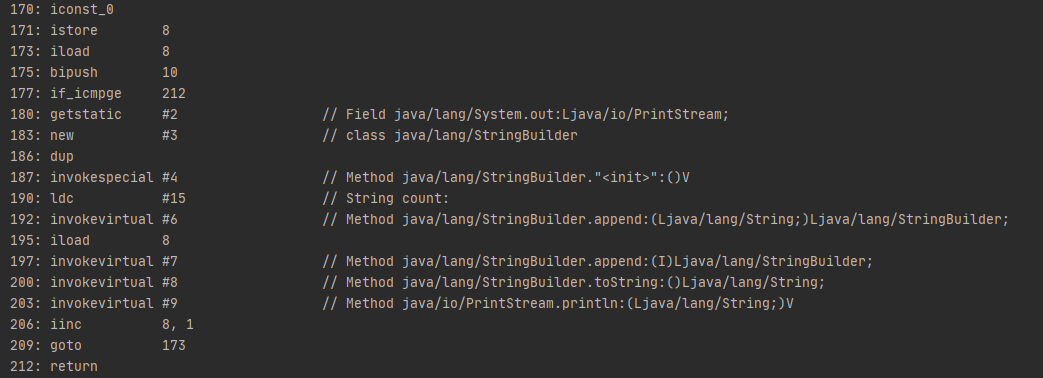
1. 将常量1压入栈顶
2. 将常量1保存到变量槽1
3. 将常量2压入栈顶
4. 将常量2保存到变量槽2
5. 将变量槽1变量压入栈顶
6. 将变量槽2变量压入栈顶
7. 将栈顶变量1、变量2进行相加
8. 将加法的结果保存到变量槽中3的位置
9. 将静态方法System.out.println压到栈顶，结合常量池中#2
10. 创建一个StringBuilder对象，这个可以看出来Java编译的时间默认进行了优化
11. 赋值栈顶的值
12. 调用StringBuilder的构造函数
13. 将常量”a + b = ”压入栈顶
14. 调用StringBuilder.apend方法
15. 将变量槽3变量，也就是a+b的结果压入栈顶
16. 将常量”a + b = ”和a+b的结果拼接起来
17. 调用StringBuilder.toString方法
18. 调用System.out.println方法，打印结果

减法、乘法、除法类似，不再叙述。



上图是if条件跳转语句：

1. 将变量槽2，也就是b的值压入栈顶
2. 将变量槽1，也就是a的值压入栈顶
3. 判断b是否大于a，若是继续往下执行，否则，跳转到170执行代码
4. 将System.out.println压入栈顶
5. 将”b > a”常量压入栈顶
6. 调用System.out.println方法，打印”b > a”



上面是for循环语句：  
（170）将常量0压入栈顶

1. 将栈顶值，也就是常量0保存到变量槽8中
2. 加载变量槽8的值
3. 将常量10压入栈顶
4. 比较常量10是否大于变量槽8中的值，若是，则继续往下执行，否则跳转到212行执行
5. 183-203 就是打印常量8的调用过程，和上面加法，if条件判断类似，不再赘述
6. 变量槽8中的值进行自增
7. 跳转到173行执行