马士兵教育

JVM Runtime Data Area and JVM Instructions

马士兵

http://mashibing.com

简历的写法

- 高能低写
  - 100分 80分 内心不自信 自尊心强 怕丢人 怕挫折
- 低能高写
  - 60分 80分
    - 敢
- 普能平写
  - 80分 80分
    - 至少

三种都不对！！！

自身角度
怎样写能拿下工作就怎样写！
跟自身水平无关！

简历唯一的作用：
拿到面试机会

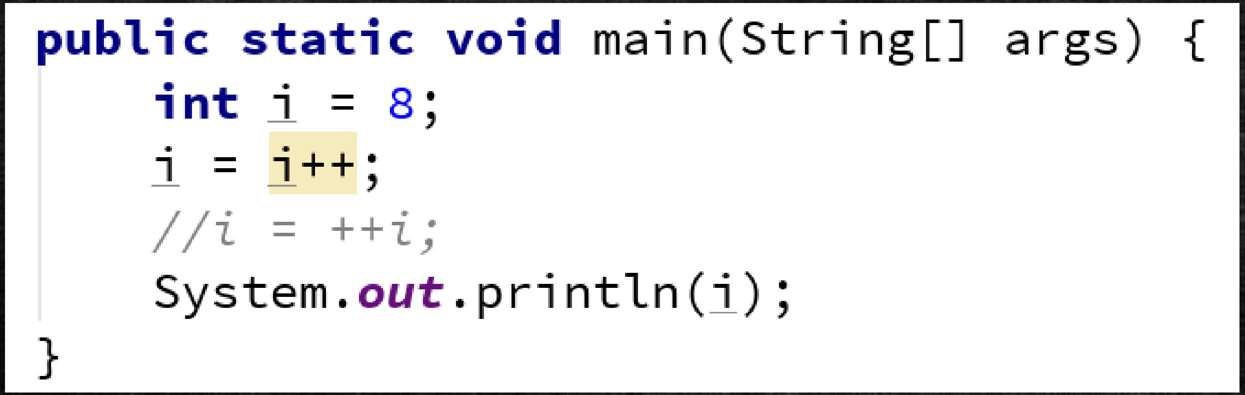
如果你是一个挖掘机er

- 能不能拿到程序员的面试机会?
  - 程序员的简历
- 1 润色简历 – 初级的面试机会 – 死！ - 录音 – 准备问题 – 下一家 – 死 loop , … 1X 成功
- 常见问题也就那么多
- 过不了试用期 – 不开就不走，使劲儿玩命学 – 1个月被开 –下一家 – 3个月 – 下一家 留下

- 精通Java 核心，有良好的算法和编码能力
- 精通面向对象编程并已构建厚实知识体系并灵活运用学习新的知识
- 精通计算机工作原理，操作系统原理，计算机网络原理
- 精通JVM，JMM模型
- 精通微服务设计方案和原理
- 精通常见垃圾回收算法、垃圾回收器及JVM调优
- 精通常见算法和数据结构并灵活运用在项目开发中
- 精通常见IO模型和优化策略
- 精通J2EE技术栈
- 精通Spring，Spring Boot，Spring Cloud技术栈
- 精通常用设计模式并灵活运用
- 精通多进程、多线程并发解决方案和编程思想
- 精通JavaScript，HTML5，CSS，Ajax，jQuery，Layui，ElementUI，Bootstrap，Vue技术栈
- 熟悉Redis、MongoDB、Memcache
- 熟悉Python、Shell脚本
- 熟悉TCP/IP协议栈
- 熟练阅读框架源码并定制框架
- 熟悉UML
- 熟悉分布式常见解决方案包括：分布式事务、分布式一致性、分布式锁
- 熟练使用Eclipse、IDEA、SVN、Git、Maven项目管理和项目构建工具
- 熟悉常见机器学习算法要点
- 熟悉应用服务器软件Tomcat，JBoss等容器配置和部署
- 熟悉使用Linux操作系统并可根据命令逐步排错和软件性能优化
- 熟悉MySql调优和常用引擎模型和核心技术，熟练掌握编写sql语句与存储过程
- 熟悉分布式文件系统HDFS原理、HDFS的Java接口应用
- 熟悉ES集群搭建及Java接口应用
- 熟悉linux内核工作原理
- 熟悉C、C++、汇编以及其工作原理
- 熟悉DDD领域驱动开发包括ES，EDA，DCI，Actors模型，Saga，CQRS和四色原型
- 熟悉大数据框架Hadoop，HBase，ELK，Spark，Hive，Impala，Storm，Kafka，Flume，Avro、Zookeeper
- 熟悉HotSpot源码
- 熟悉Android开发
- 了解RPC框架Hessian，RMI，Thrift
- 了解集群下的并发解决方案，支持(HA)高可用(采用nginx，apache，lvs，KeepAlive，HProxy)
- 对NLP领域有自己的见解和了解常用算法模型和分词算法
- 对开源异步高性能处理框架netty做过贡献：通过改造Recycle类的stack对象修改成软引用避免大量FastThreadlocal线程数过多情况下导致full gc

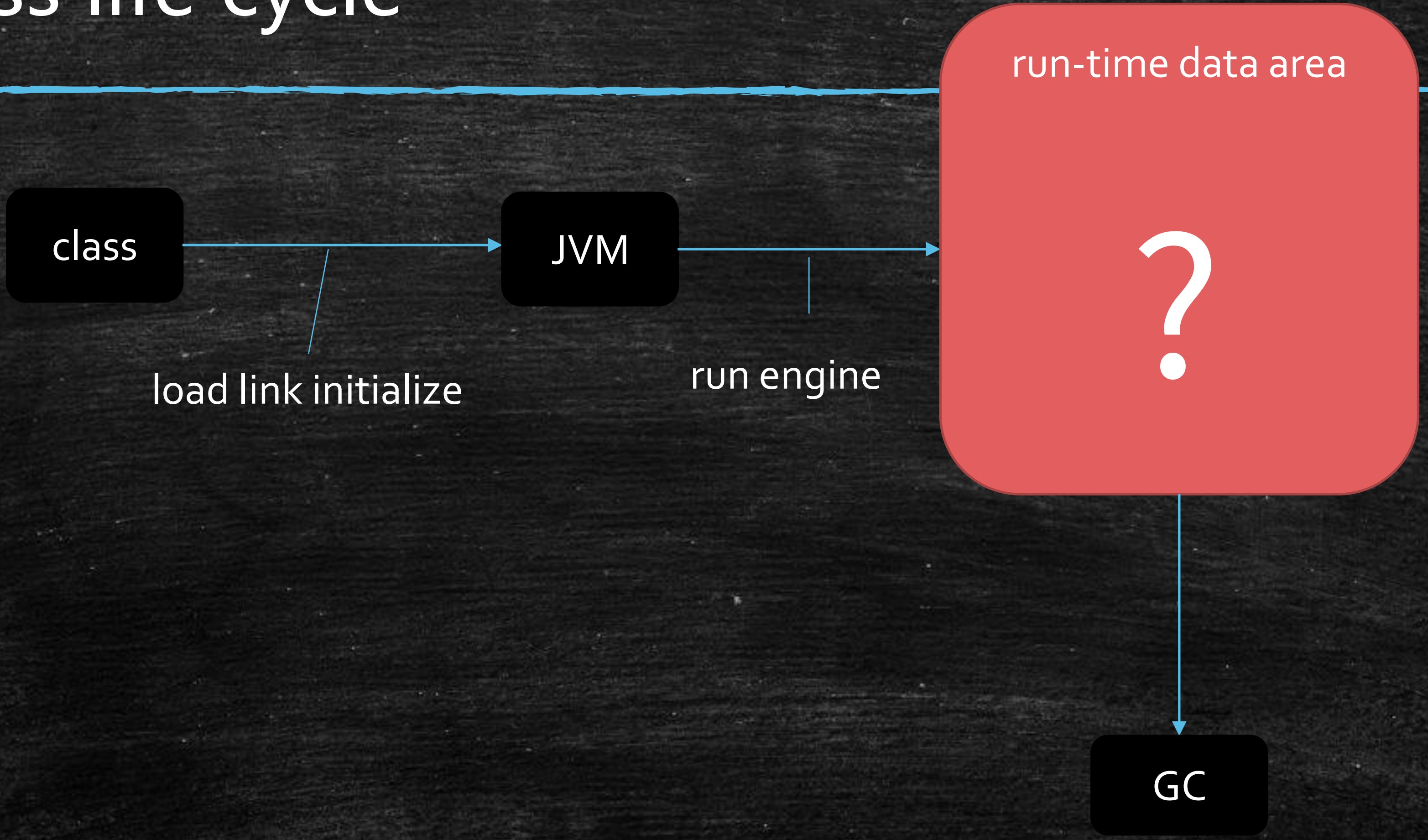
从一道面试题谈起

- TestIPlusPlus.java

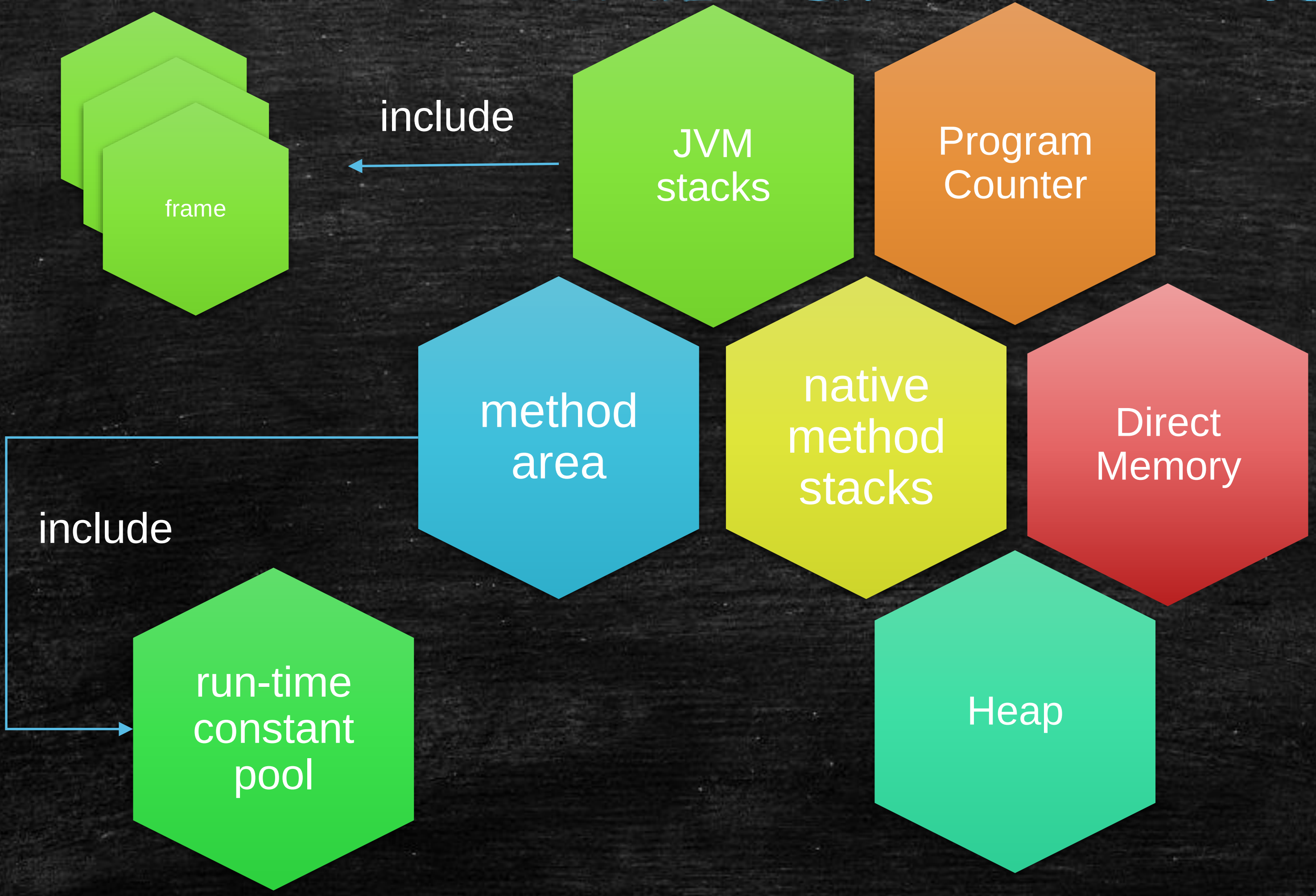


```
public static void main(String[] args) {
    int i = 8;
    i = i++;
    //i = ++i;
    System.out.println(i);
}
```

a class life cycle



Run-time data areas



PC

- Each Java Virtual Machine thread has its own pc (program counter) register.
- At any point, each Java Virtual Machine thread is executing the code of a single method, namely the current method for that thread.
- If that method is not native , the pc register contains the address of the Java Virtual Machine instruction currently being executed.

JVM Stacks

- Each Java Virtual Machine thread has a private Java Virtual Machine stack, created at the same time as the thread.
- A Java Virtual Machine stack stores frames

Heap

- The Java Virtual Machine has a heap that is shared among all Java Virtual Machine threads.
- The heap is the run-time data area from which memory for all class instances and arrays is allocated.

Method Area

- The Java Virtual Machine has a method area that is shared among all Java Virtual Machine threads.
- It stores per-class structures

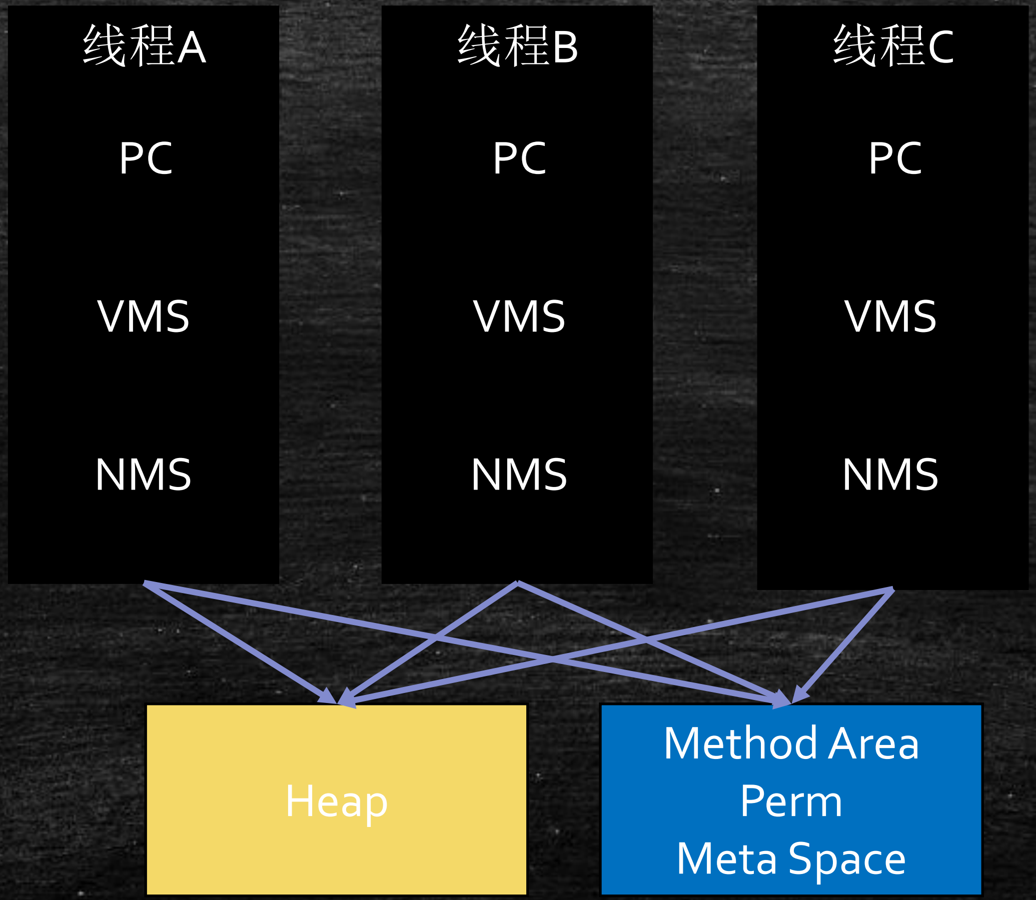
Run-Time Constant Pool

- A run-time constant pool is a per-class or per-interface run-time representation of the constant_pool table in a class file

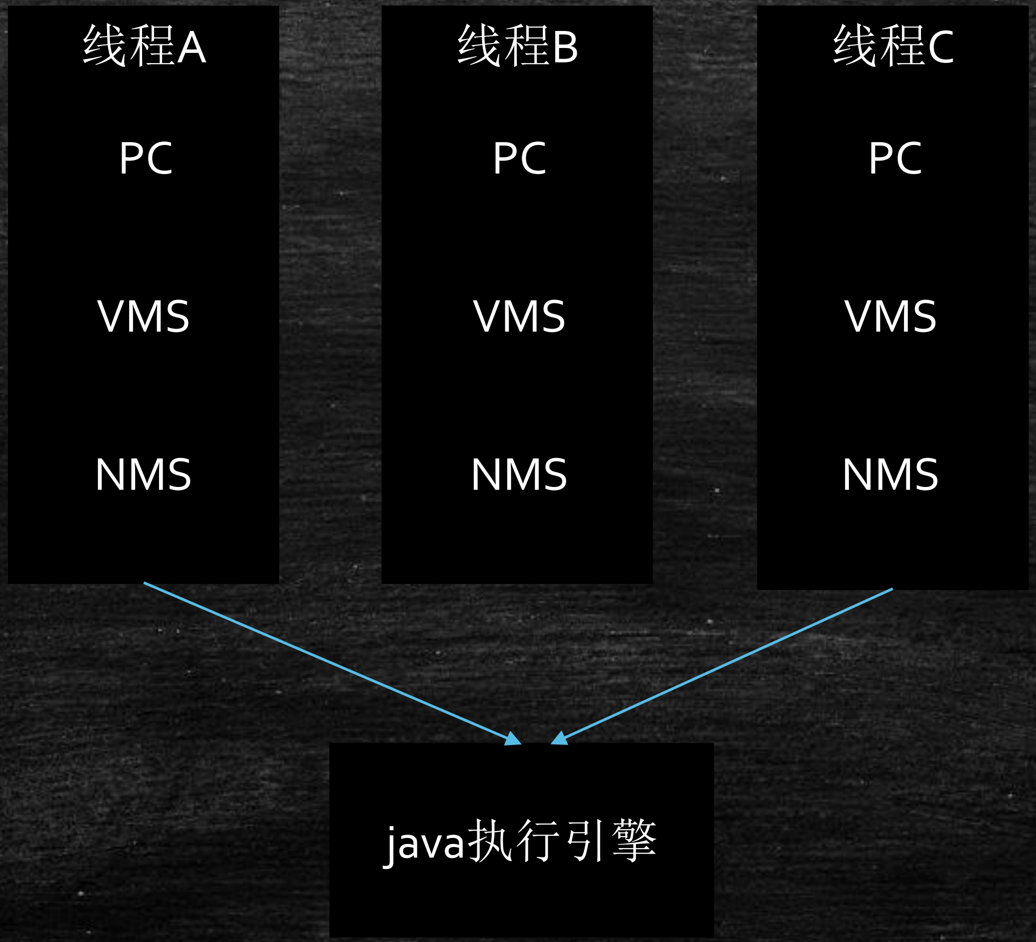
Native Method Stacks

- An implementation of the Java Virtual Machine may use conventional stacks called native method stacks

next: 线程共享区域

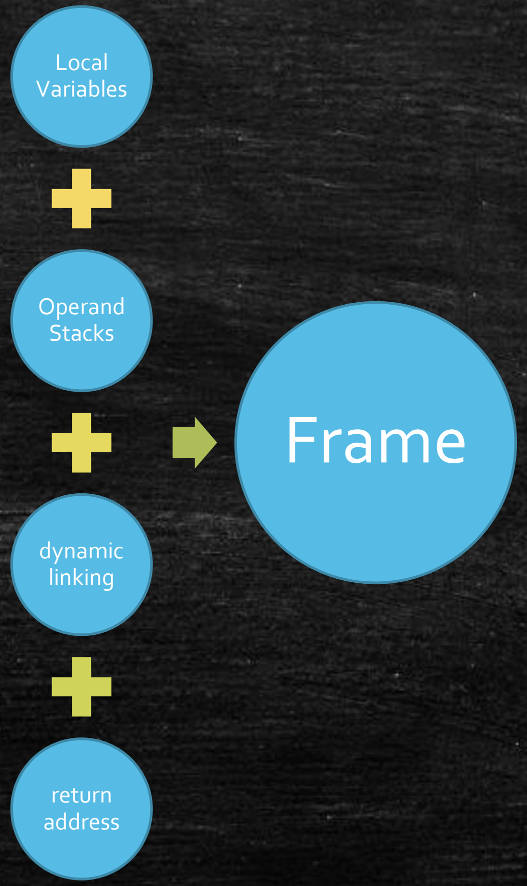

线程A
PC
VMS
NMS
线程B
PC
VMS
NMS
线程C
PC
VMS
NMS
Heap
Method Area
Perm
Meta Space

为什么需要记录当前线程的执行地址？



栈帧Frame

- A frame is used to store data and partial results, as well as to perform dynamic linking, return values for methods, and dispatch exceptions.



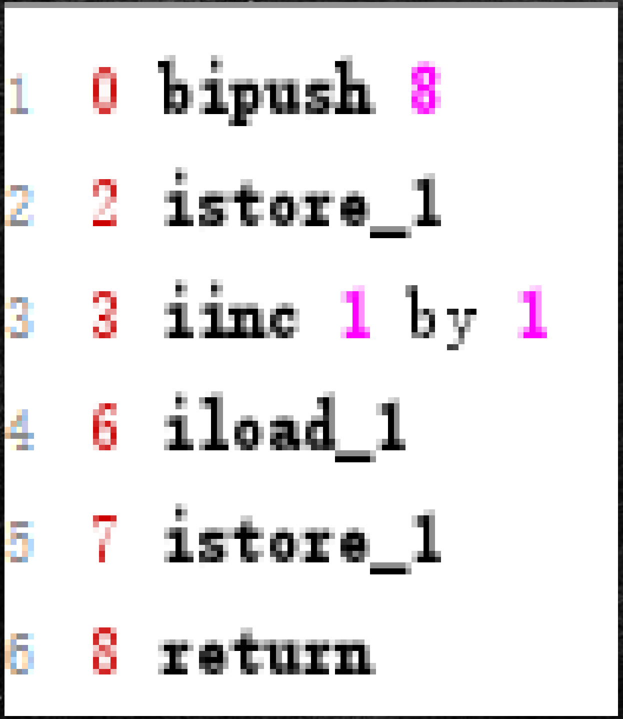
补充：

- 基于栈的指令集
- 基于寄存器的指令集

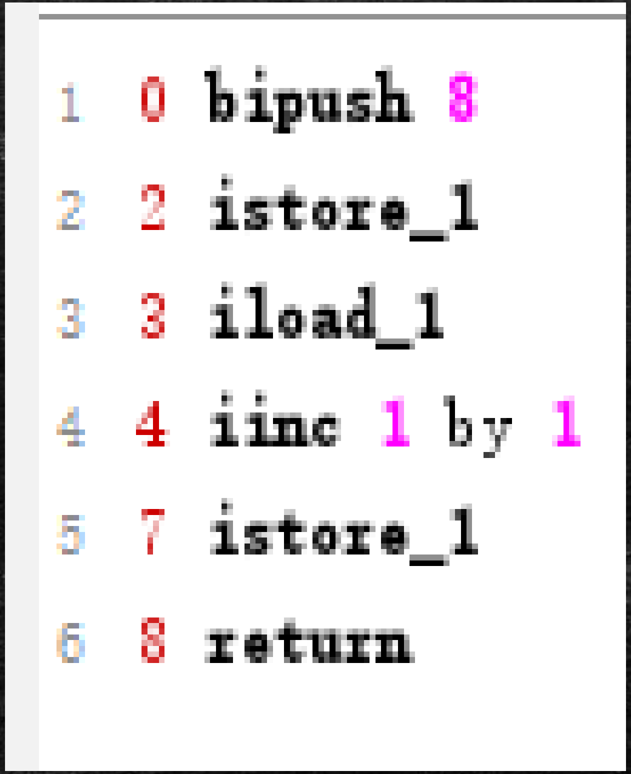
hotspot的local variable table
类似于寄存器

面试题解答：

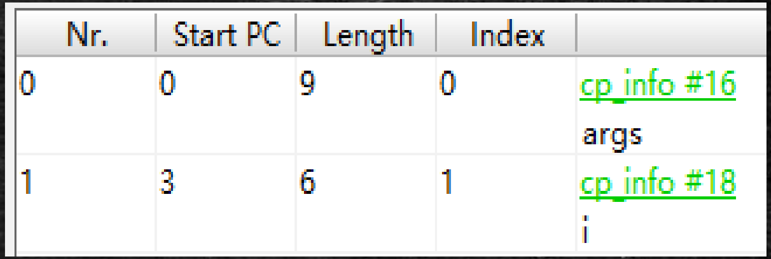
- int I = 8



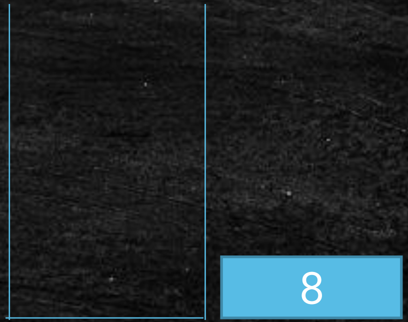
```
0 bipush 8
2 istore_1
3 iinc 1 by 1
6 iload_1
7 istore_1
8 return
```

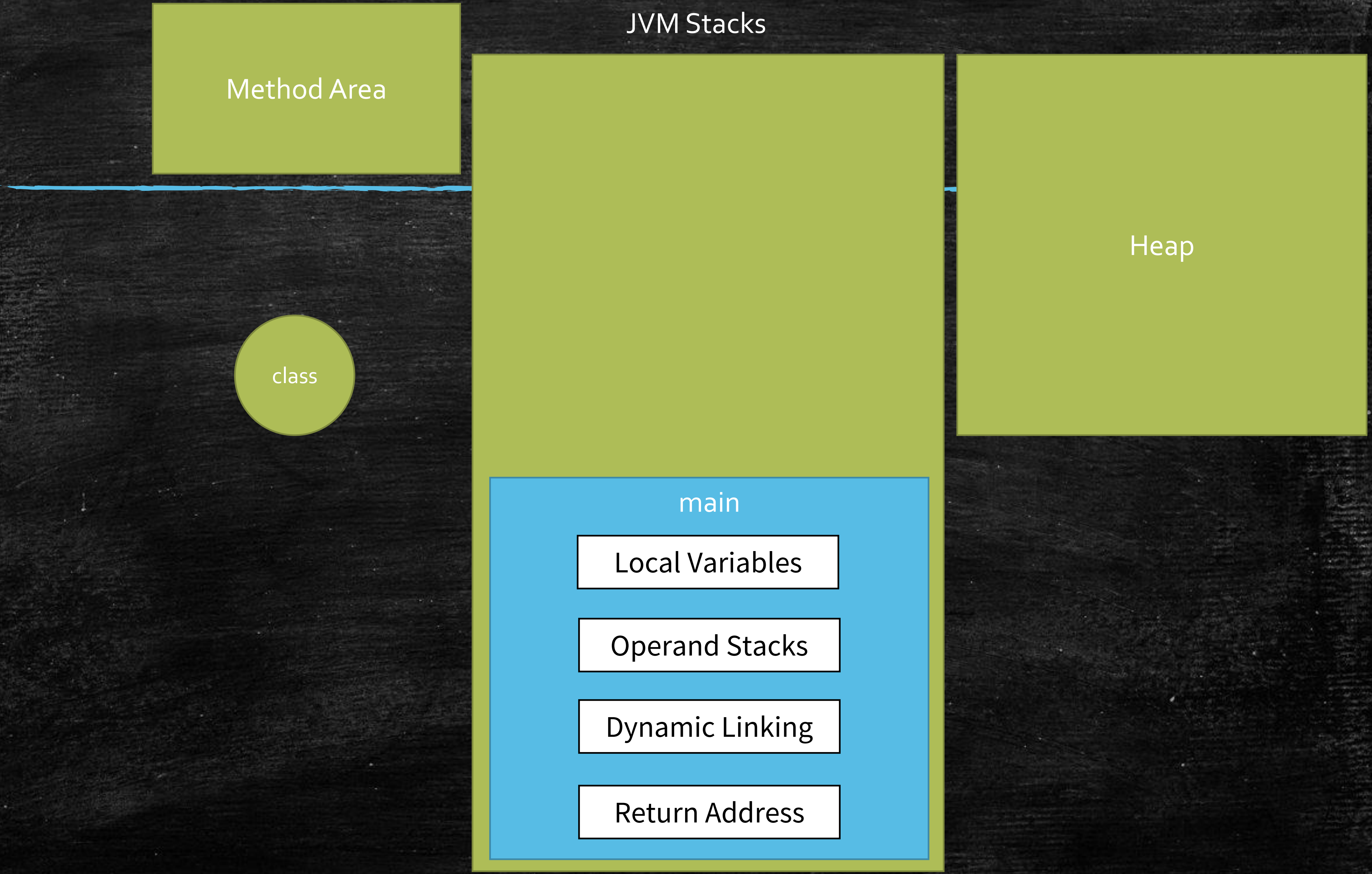


```
0 bipush 8
2 istore_1
3 iload_1
4 iinc 1 by 1
7 istore_1
8 return
```

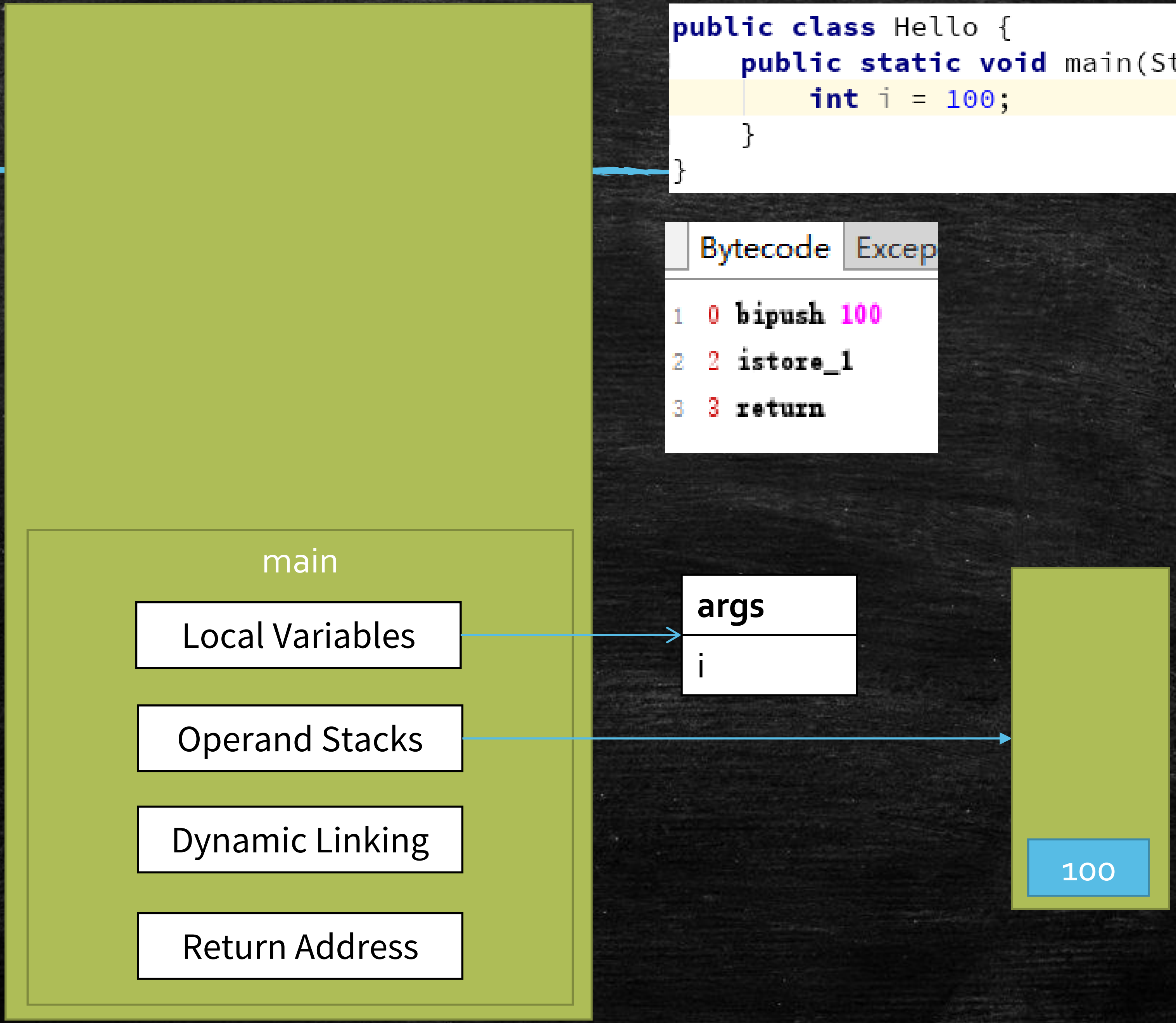


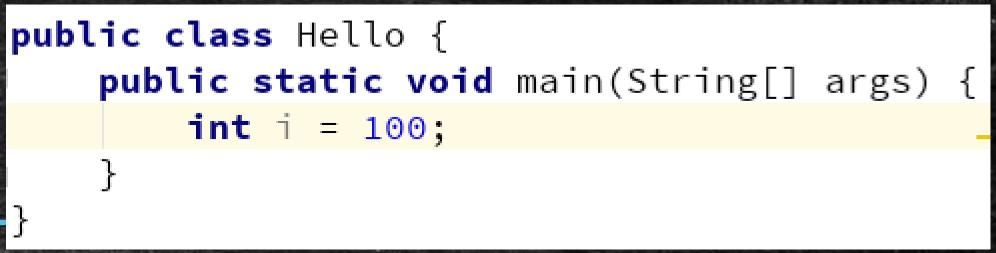
Nr.	Start PC	Length	Index	
0	0	9	0	cp_info #16 args
1	3	6	1	cp_info #18 i



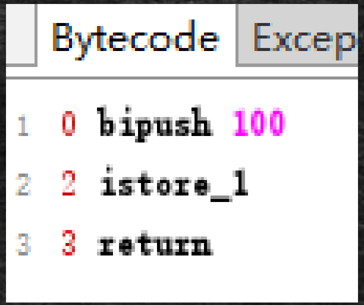

Method Area
JVM Stacks
Heap
class
main
Local Variables
Operand Stacks
Dynamic Linking
Return Address

JVM Stacks



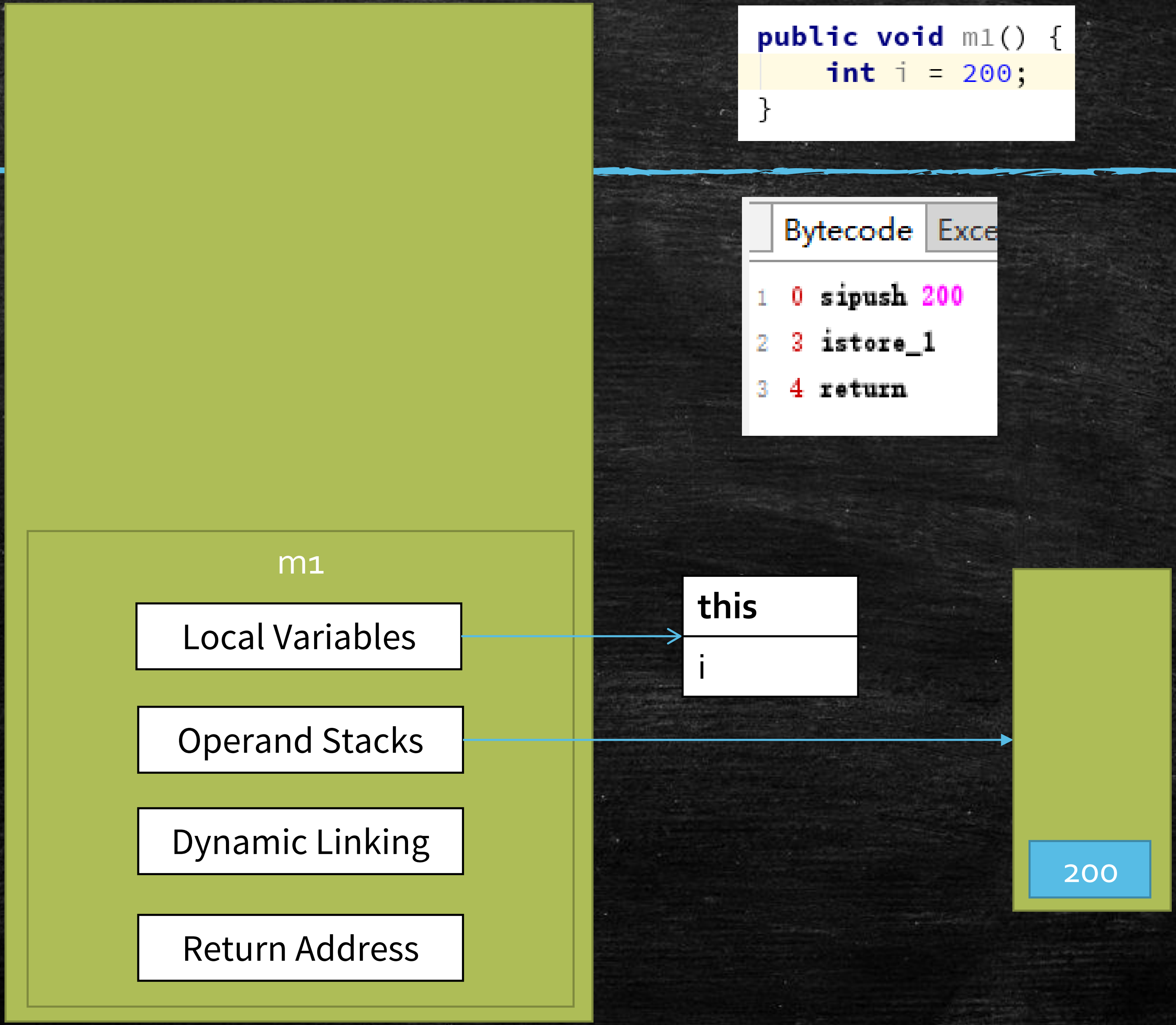


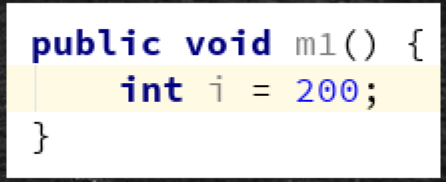
```
public class Hello {
    public static void main(String[] args) {
        int i = 100;
    }
}
```



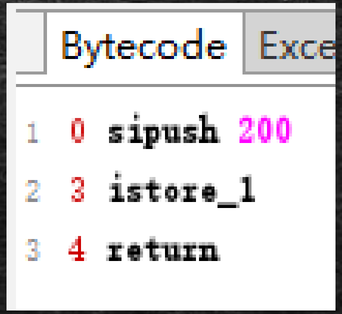
```
0 bipush 100
2 istore_1
3 return
```

JVM Stacks





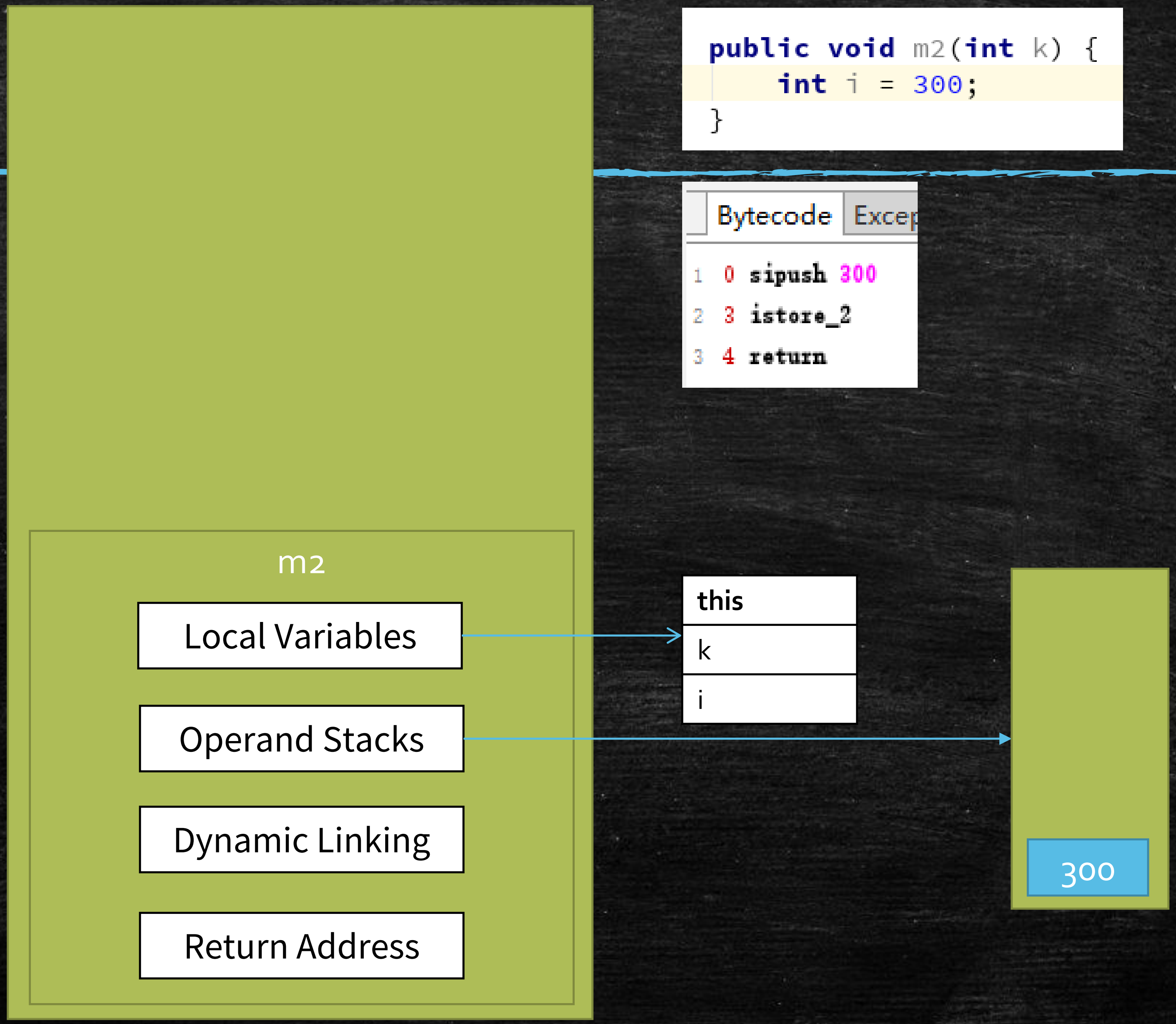
```
public void m1() {
    int i = 200;
}
```

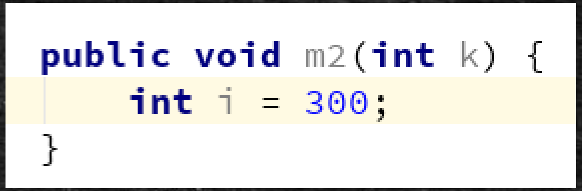


Bytecode Exce

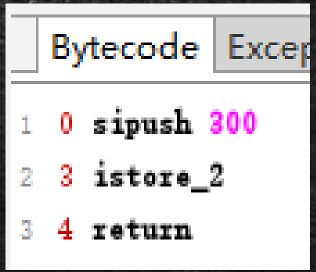
```
0 sipush 200
3 istore_1
4 return
```

JVM Stacks

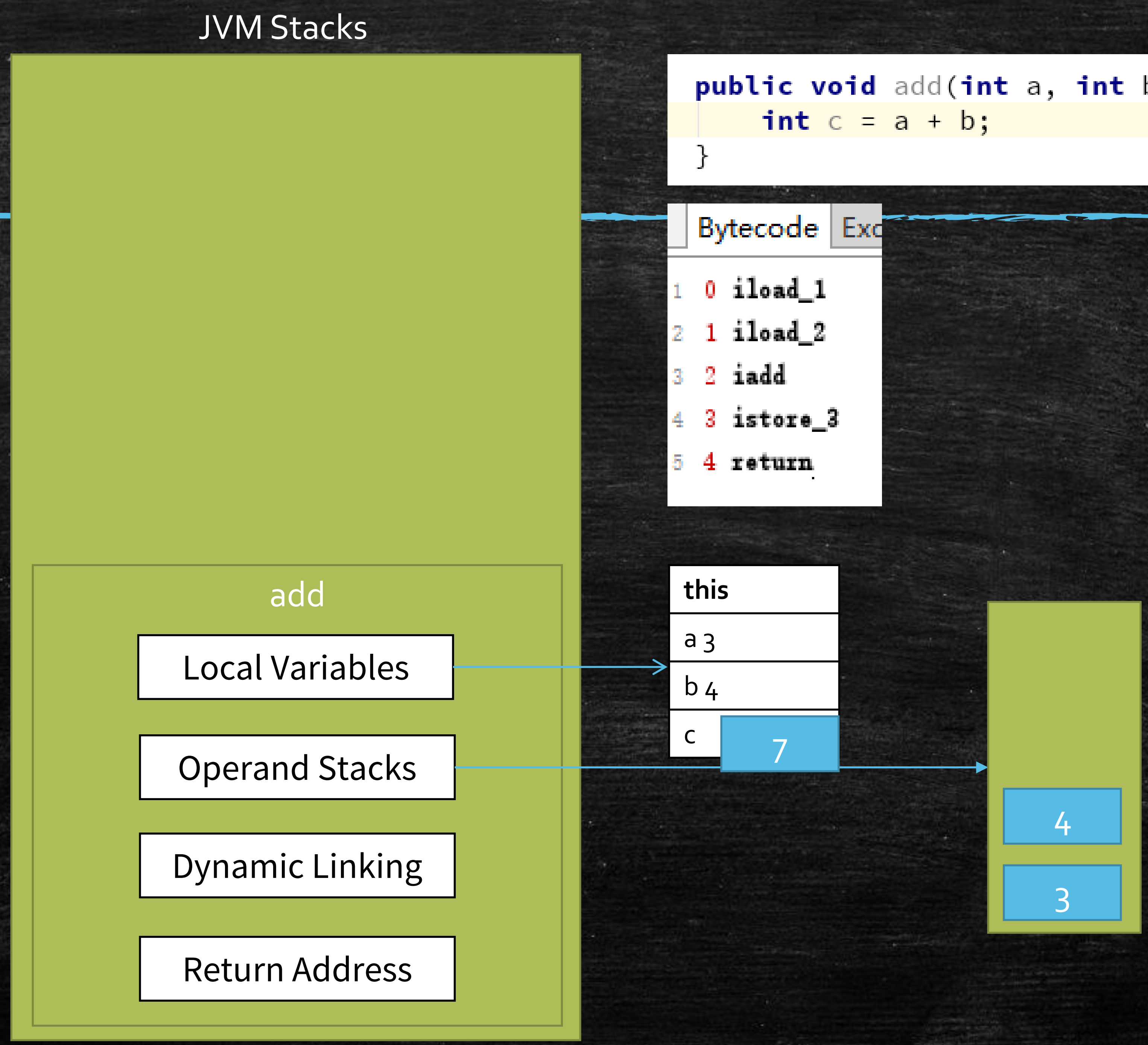


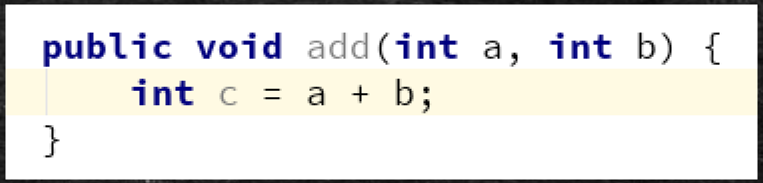


```
public void m2(int k) {
    int i = 300;
}
```

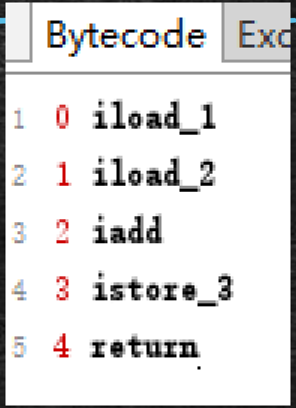


```
0 sipush 300
3 istore_2
4 return
```





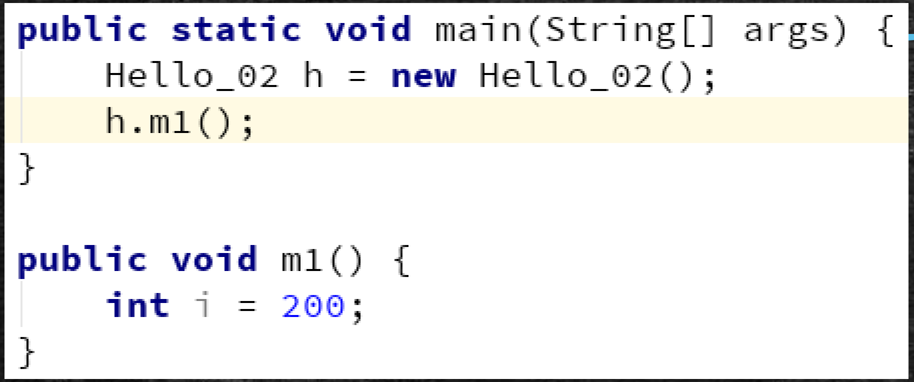
```
public void add(int a, int b) {
    int c = a + b;
}
```



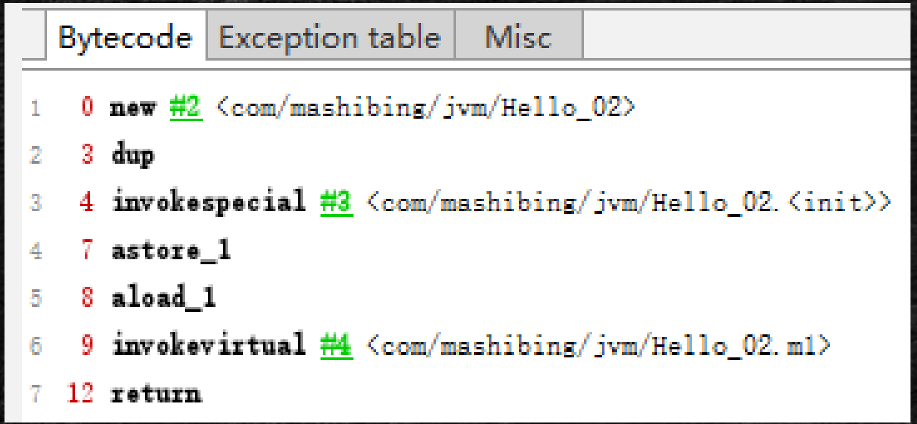
Bytecode Exc

```
0 iload_1
1 iload_2
2 iadd
3 istore_3
4 return
```

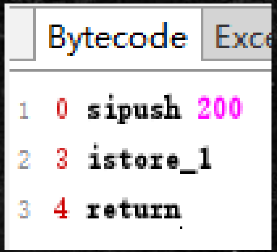
JVM Stacks



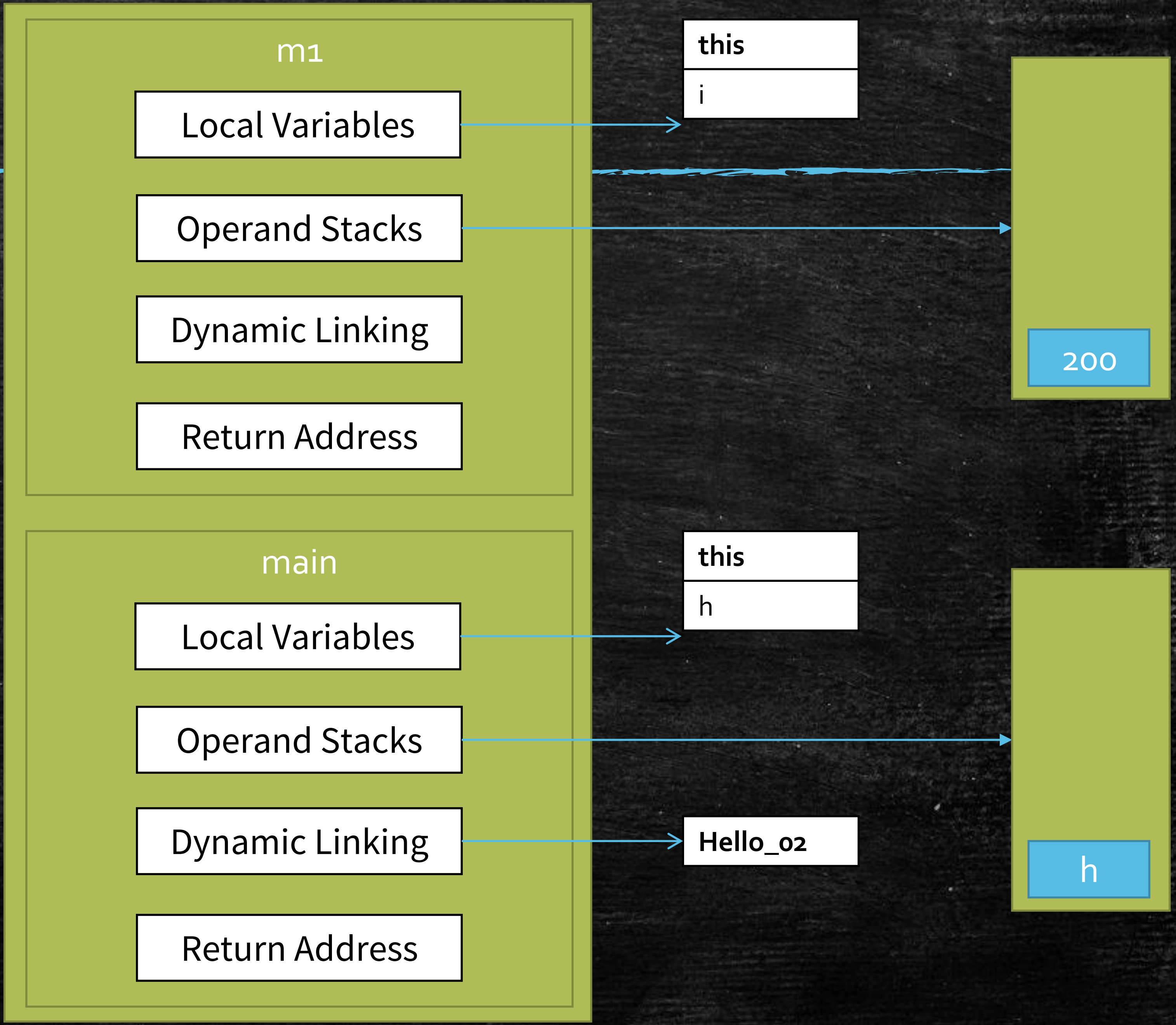
```java
public static void main(String[] args) {
    Hello_02 h = new Hello_02();
    h.m1();
}

public void m1() {
    int i = 200;
}
```



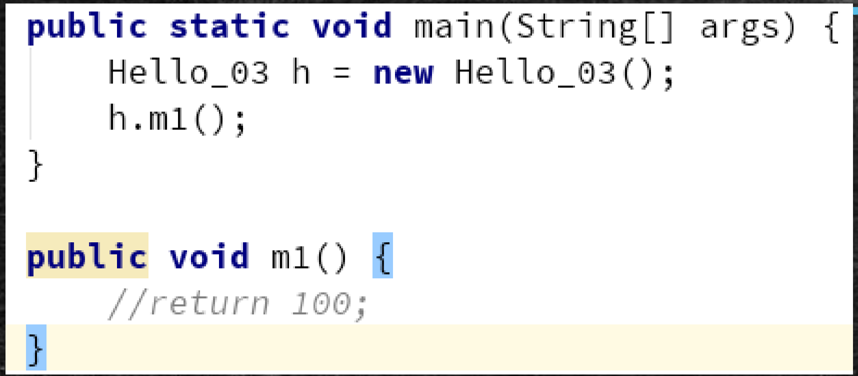
```
 0 new #2 <com/mashibing/jvm/Hello_02>
 3 dup
 4 invokespecial #3 <com/mashibing/jvm/Hello_02.<init>>
 7 astore_1
 8 aload_1
 9 invokevirtual #4 <com/mashibing/jvm/Hello_02.m1>
12 return
```



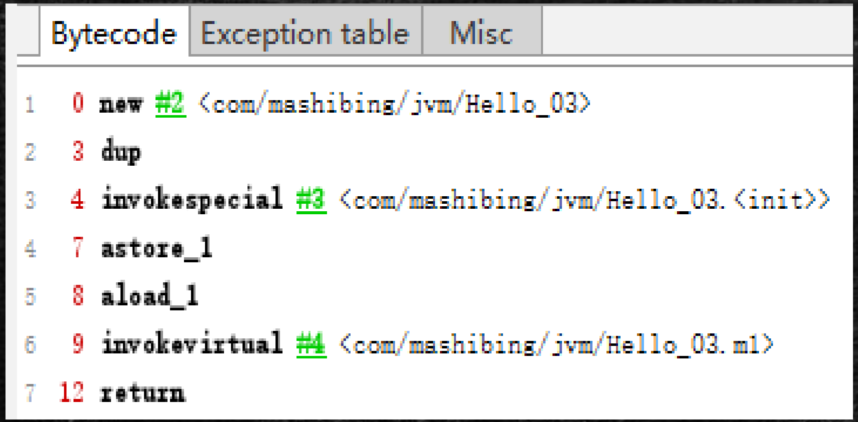
```
0 sipush 200
3 istore_1
4 return
```



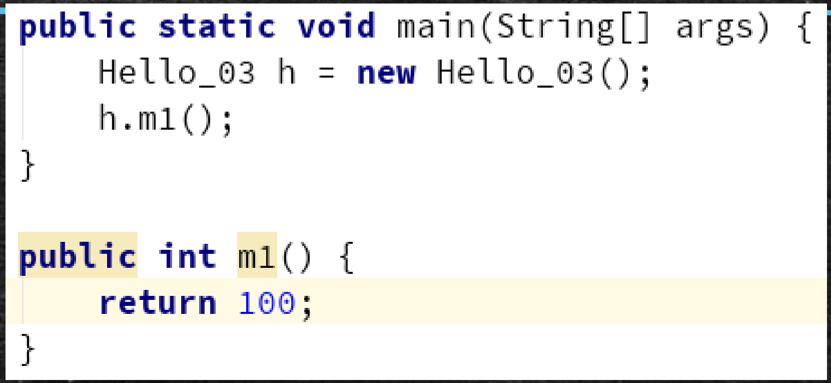
返回值



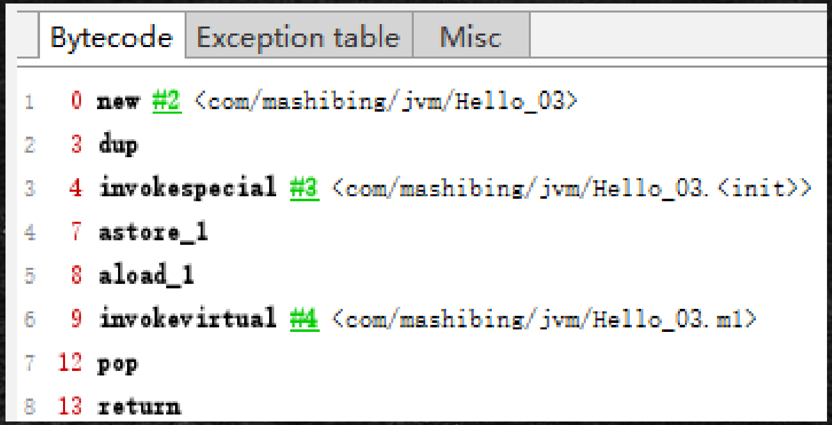
```java
public static void main(String[] args) {
    Hello_03 h = new Hello_03();
    h.m1();
}

public void m1() {
    //return 100;
}
```



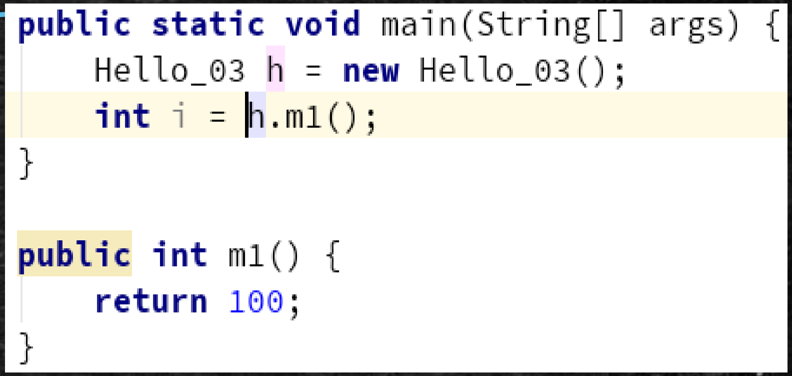
```
 0 new #2 <com/mashibing/jvm/Hello_03>
 3 dup
 4 invokespecial #3 <com/mashibing/jvm/Hello_03.<init>>
 7 astore_1
 8 aload_1
 9 invokevirtual #4 <com/mashibing/jvm/Hello_03.m1>
12 return
```



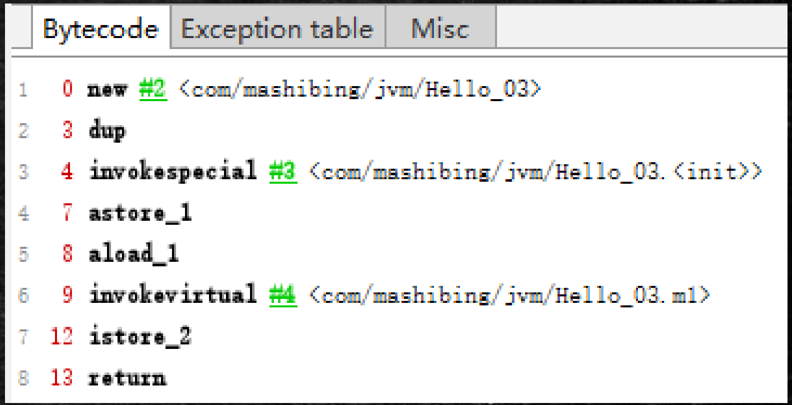
```java
public static void main(String[] args) {
    Hello_03 h = new Hello_03();
    h.m1();
}

public int m1() {
    return 100;
}
```



```
 0 new #2 <com/mashibing/jvm/Hello_03>
 3 dup
 4 invokespecial #3 <com/mashibing/jvm/Hello_03.<init>>
 7 astore_1
 8 aload_1
 9 invokevirtual #4 <com/mashibing/jvm/Hello_03.m1>
12 pop
13 return
```

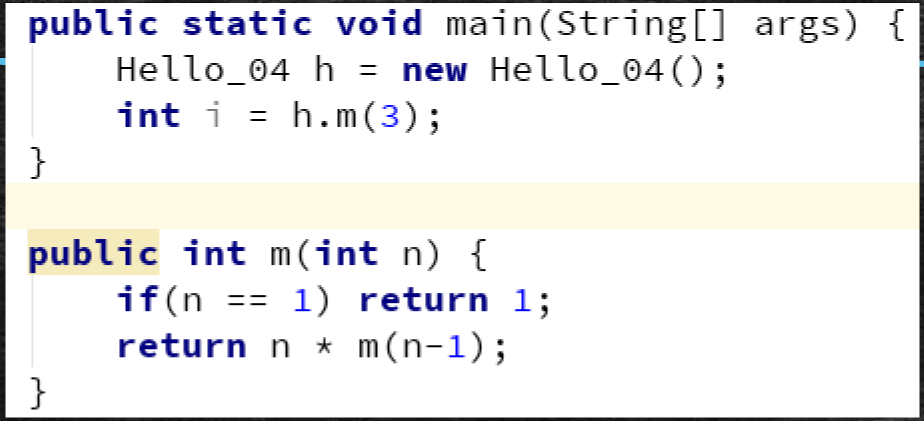


```java
public static void main(String[] args) {
    Hello_03 h = new Hello_03();
    int i = h.m1();
}

public int m1() {
    return 100;
}
```

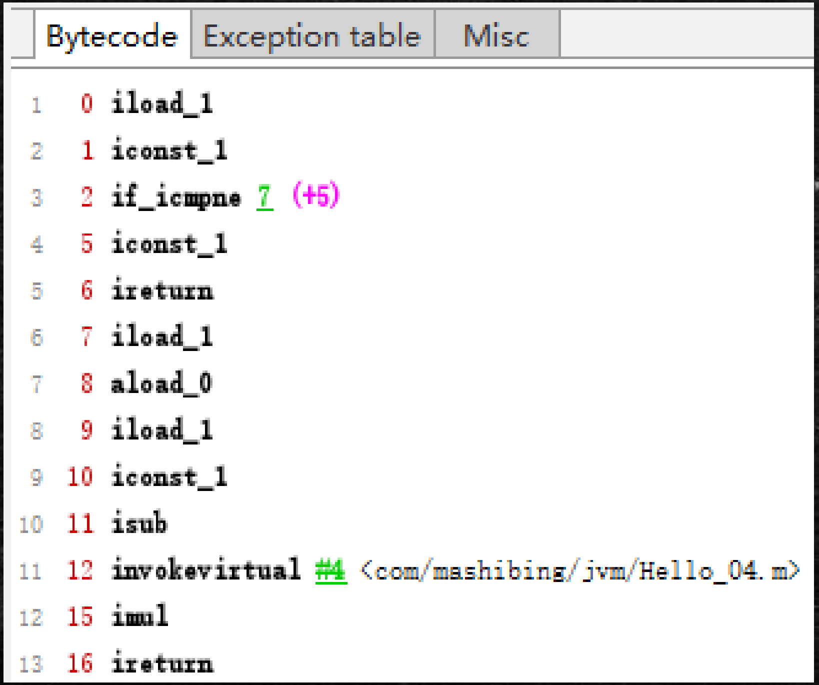


```
 0 new #2 <com/mashibing/jvm/Hello_03>
 3 dup
 4 invokespecial #3 <com/mashibing/jvm/Hello_03.<init>>
 7 astore_1
 8 aload_1
 9 invokevirtual #4 <com/mashibing/jvm/Hello_03.m1>
12 istore_2
13 return
```

frames of recursion

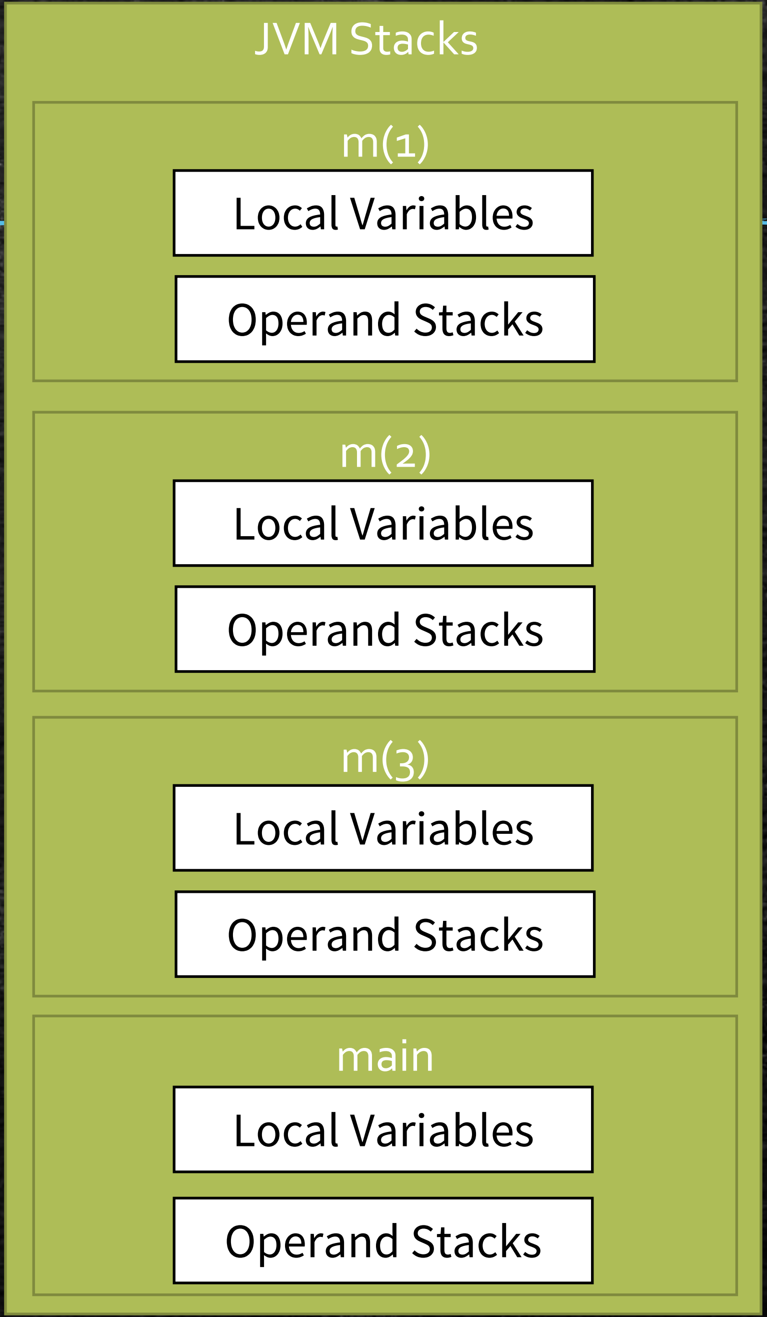


```
public static void main(String[] args) {
    Hello_04 h = new Hello_04();
    int i = h.m(3);
}

public int m(int n) {
    if(n == 1) return 1;
    return n * m(n-1);
}
```



Bytecode | Exception table | Misc

```
 0 iload_1
 1 iconst_1
 2 if_icmpne 7 (+5)
 5 iconst_1
 6 ireturn
 7 iload_1
 8 aload_0
 9 iload_1
10 iconst_1
11 isub
12 invokevirtual #4 <com/mashibing/jvm/Hello_04.m>
15 imul
16 ireturn
```



总结

- <clinit>
- <init>
- _store
- _load
- invoke_XXX

添加幻灯片标题 - 4

添加幻灯片标题 - 5