1.（选做）自己写一个简单的 Hello.java，里面需要涉及基本类型，四则运行，if 和 for，然后自己分析一下对应的字节码，有问题群里讨论。

代码实现：

```
package test3;
public class HellloTest {
 private static int[] numbers = {1, 6, 8};
 public static void main(String[] args) {
 TestAverage testAverage = new TestAverage();
 for (int number : numbers) {
 testAverage.submit(number);
 }
 double avg = testAverage.getAvg();
  }
}

class TestAverage {
 private int count = 0;
 private double sum = 0.0D;
 public void submit(double value){
 this.count ++;
 this.sum += value;
 }
 public double getAvg(){
 if(0 == this.count){ return sum;}
 return this.sum/this.count;
 }
}
```

字节码分析：

```
 //  -encoding UTF-8  解决javac运行时，“编码GBK 的不可映射字符”问题
E:\GeekCode\homeWork_Week1\geekCode\src\main\java>javac -g -encoding UTF-8 test3/HelloTest.java

E:\GeekCode\homeWork_Week1\geekCode\src\main\java>javap -c -verbose test3/HelloTest
Classfile /E:/GeekCode/homeWork_Week1/geekCode/src/main/java/test3/HelloTest.class
  Last modified 2021-11-6; size 775 bytes
  MD5 checksum e17333989875c696744c90aec9fff4f6
  Compiled from "HelloTest.java"
public class test3.HelloTest
```

```
  //小版本号
  minor version: 0
  //大版本号
  major version: 52
  //访问控制符
  flags: ACC_PUBLIC, ACC_SUPER
  // 常量池就是一个常量的大字典，使用编号的方式把程序里用到的各类常量统一管理起来，
  //这样在字节码操作里，只需要引用编号即可
  //#1 常量编号，该文件中其他地方可以引用，= 等号就是分隔符
  //Methodref 表明这个常量指向的是一个方法
  //Fieldref 表明这个常量指向的是一个属性
Constant pool:
  #1 = Methodref #8.#35 // java/lang/Object."<init>":()V
  #2 = Class #36 // test3/TestAverage
  #3 = Methodref #2.#35 // test3/TestAverage."<init>":()V
  #4 = Fieldref #7.#37 // test3/HelloTest.numbers:[I
  #5 = Methodref #2.#38 // test3/TestAverage.submit:(D)V
  #6 = Methodref #2.#39 // test3/TestAverage.getAvg:()D
  #7 = Class #40 // test3/HelloTest
  #8 = Class #41 // java/lang/Object
  #9 = Utf8 numbers
  #10 = Utf8 [I
  #11 = Utf8 <init>
  #12 = Utf8 ()V
  #13 = Utf8 Code
  #14 = Utf8 LineNumberTable
  #15 = Utf8 LocalVariableTable
  #16 = Utf8 this
  #17 = Utf8 Ltest3/HelloTest;
  #18 = Utf8 main
  #19 = Utf8 ([Ljava/lang/String;)V
  #20 = Utf8 number
  #21 = Utf8 I
  #22 = Utf8 args
  #23 = Utf8 [Ljava/lang/String;
  #24 = Utf8 testAverage
  #25 = Utf8 Ltest3/TestAverage;
  #26 = Utf8 avg
  #27 = Utf8 D
```

```
  #28 = Utf8 StackMapTable
  #29 = Class #23 // "[Ljava/lang/String;"
  #30 = Class #36 // test3/TestAverage
  #31 = Class #10 // "[I"
  #32 = Utf8 <clinit>
  #33 = Utf8 SourceFile
  #34 = Utf8 HelloTest.java
  #35 = NameAndType #11:#12 // "<init>":()V
  #36 = Utf8 test3/TestAverage
  #37 = NameAndType #9:#10 // numbers:[I
  #38 = NameAndType #42:#43 // submit:(D)V
  #39 = NameAndType #44:#45 // getAvg:()D
  #40 = Utf8 test3/HelloTest
  #41 = Utf8 java/lang/Object
  #42 = Utf8 submit
  #43 = Utf8 (D)V
  #44 = Utf8 getAvg
  #45 = Utf8 ()D
{
  public test3.HelloTest();
  descriptor: ()V
  flags: ACC_PUBLIC
  Code:
  // stack表示栈深度，locals表示局部变量表保留槽位数量，args_size表示方法的参数个数
  stack=1, locals=1, args_size=1
  //栈操作指令，本地变量表的第0个位置加载到栈上
  0: aload_0
  //触发静态初始化方法
  1: invokespecial #1 // Method java/lang/Object."<init>":()V
  4: return
  LineNumberTable:
  line 2: 0
  //局部变量表
  LocalVariableTable:
  Start Length Slot Name Signature
  0 5 0 this Ltest3/HelloTest;

  public static void main(java.lang.String[]);
  descriptor: ([Ljava/lang/String;)V
```

```
88   //访问标志
89   flags: ACC_PUBLIC, ACC_STATIC
90   Code:
91   // stack表示栈深度，locals表示局部变量表保留槽位数量，args_size表示方法的参数个数
92   stack=3, locals=6, args_size=1
93   //编号 0 的字节码 new ，创建TestAverage类的对象
94   0: new #2 // class test3/TestAverage
95   //编号 3 的字节码 dup 复制栈顶引用值
96   3: dup
97   //编号 4 的字节码 invokespecial 执行对象初始化
98   4: invokespecial #3 // Method test3/TestAverage."<init>":()V
99    //编号 7 开始，使用 astore_1 指令将引用地址值(addr.)存储(store)到编号为 1
100  //的局部变量中： astore_1 中的 1 指代 LocalVariableTable 中 ma 对应的槽位编号
101   7: astore_1
102   // 获取静态数据
103   8: getstatic #4 // Field numbers:[I
104   11: astore_2
105   12: aload_2
106   13: arraylength
107   14: istore_3
108   15: iconst_0
109   16: istore 4
110  //这段指令将局部变量表中 4 号槽位 和 3 号槽位的值加载到栈中，并调用if_icmpge指令来比较他们的值。
111  //【 if_icmpge 解读: if, integer, compare, great equal】, 如果一个数的值大于或等于
112  //另一个值，则程序执行流程跳转到 pc=43 的地方继续执行
113   18: iload 4
114   20: iload_3
115   21: if_icmpge 43
116   24: aload_2
117   25: iload 4
118   27: iaload
119   28: istore 5
120   30: aload_1
121  //将一个 int 类型局部变量的值，作为整数加载到栈中，然后用 i2d 指令将其转换为 double 值
122   31: iload 5
123   33: i2d
```

```
124   //invokevirtual 用于调用公共，受保护和打包私有方法
125   34: invokevirtual #5 // Method test3/TestAverage.submit:(D)V
126   //唯一不需要将数值load到操作数栈的指令是 iinc ，它可以直接对LocalVariableTa
ble 中的值进行运算。 其他的所有操作均使用栈来执行
127   // 4号槽位的值加1
128   37: iinc 4, 1
129   // 跳到循环开始的地方
130   40: goto 18
131   43: aload_1
132   44: invokevirtual #6 // Method test3/TestAverage.getAvg:()D
133   47: dstore_2
134   48: return
135   LineNumberTable:
136   line 5: 0
137   line 6: 8
138   line 7: 30
139   line 6: 37
140   line 9: 43
141   line 10: 48
142   LocalVariableTable:
143   Start Length Slot Name Signature
144   //5 号槽位被 number 占用了
145   30 7 5 number I
146   //0 号槽位被main方法的参数 args 占据了
147   0 49 0 args [Ljava/lang/String;
148   //1 号槽位被 testAverage 占用了
149   8 41 1 testAverage Ltest3/TestAverage;
150   //2 号槽位是for循环之后才被 avg 占用的
151   48 1 2 avg D
152   //StackMapTable主要用来验证跳转前后locals、stack中的类型和大小一致
153   StackMapTable: number_of_entries = 2
154   frame_type = 255 /* full_frame */
155   offset_delta = 18
156   locals = [ class "[Ljava/lang/String;", class test3/TestAverage, class
"[I", int, int ]
157   stack = []
158   frame_type = 248 /* chop */
159   offset_delta = 24
160 // 初始化数据
161   static {};
```

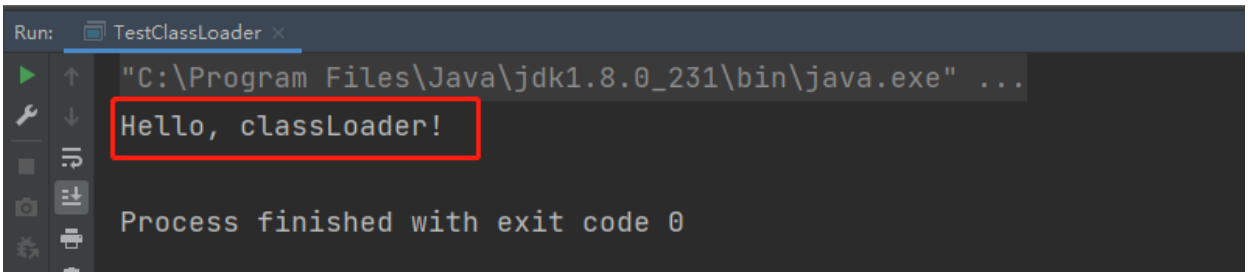
```
  descriptor: ()V
  //访问标志
  flags: ACC_STATIC
  Code:
  // stack表示栈深度，locals表示局部变量表保留槽位数量，args_size表示方法的参数个数
  stack=4, locals=0, args_size=0
  0: iconst_3
  1: newarray int
  //dup 指令复制栈顶元素的值
  3: dup
  4: iconst_0
  5: iconst_1
  6: iastore
  7: dup
  8: iconst_1
  9: bipush 6
  11: iastore
  12: dup
  13: iconst_2
  14: bipush 8
  16: iastore
  // 将值赋给静态字段 与之对应的有 putfield 将值赋给实例字段
  17: putstatic #4 // Field numbers:[I
  20: return
  LineNumberTable:
  line 3: 0
}
SourceFile: "HelloTest.java"
```

2.（必做）自定义一个 Classloader，加载一个 Hello.xlass 文件，执行 hello 方法，此文件内容是一个 Hello.class 文件所有字节（x=255-x）处理后的文件。文件群里提供。

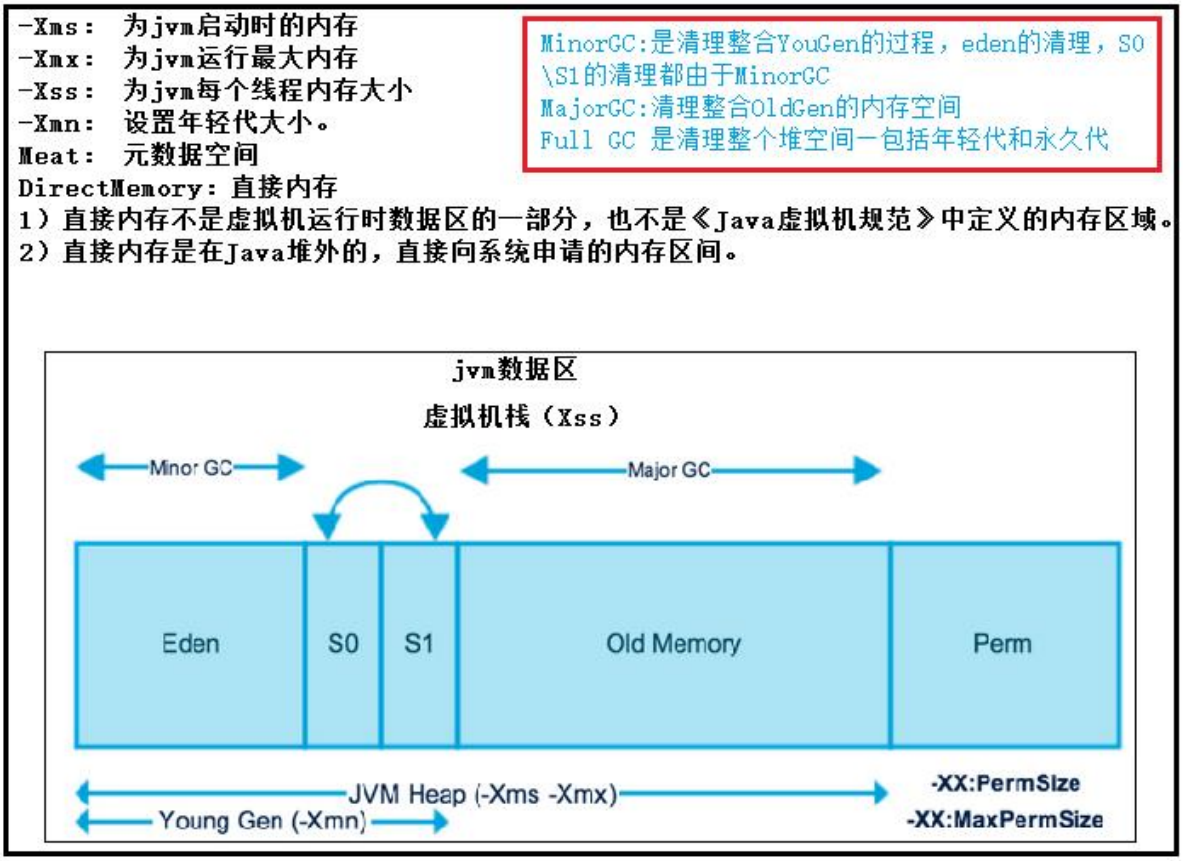
```
package test5;

import java.io.*;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

/**
 * 自定义一个 Classloader，加载一个 Hello.xlass 文件，执行 hello 方法，
```

```
 * 此文件内容是一个 Hello.class 文件所有字节（x=255-x）处理后的文件。文件群里提供。
 */
public class TestClassLoader extends ClassLoader {

 public static void main(String[] args) throws ClassNotFoundException, InstantiationException, IllegalAccessException, NoSuchMethodException, InvocationTargetException {
 // 指定类加载器加载调用
 TestClassLoader testClassLoader = new TestClassLoader();
 Class<?> TClass = Class.forName("Hello", true, testClassLoader);
 Object obj = TClass.newInstance();
 Method method = obj.getClass().getMethod("hello");
 method.invoke(obj);
 }


 @Override
 public Class<?> findClass(String name) throws ClassNotFoundException {
 // 读取resources目录下的文件
 String loadPath = ClassLoader.getSystemClassLoader().getResource("Hello.xlass").getPath();
 byte[] classData = getClassData(loadPath);
 if (classData == null) {
 throw new ClassNotFoundException();
 } else {
 return defineClass(name, classData, 0, classData.length);
 }
 }

 //获取文件class的byte数组
 private byte[] getClassData(String loadPath) {
 ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
 try {
 InputStream inputStream = new FileInputStream(loadPath);
 int bufferSize = 4096;
 byte[] buffer = new byte[bufferSize];
 int bytesNumRead = 0;
 while ((bytesNumRead = inputStream.read(buffer)) != -1) {
 byteArrayOutputStream.write(buffer, 0, bytesNumRead);
 }
```

```
46  } catch (Exception e) {
47  e.printStackTrace();
48  }
49  return decode(byteArrayOutputStream.toByteArray());
50  }
51
52  //解密 文件内容是一个Hello.xlass文件所有字节（x=255-x）处理后的文件 所以需要对文件进行解密操作
53  private byte[] decode(byte[] byteArr) {
54  byte[] bytes = new byte[byteArr.length];
55  for (int i = 0; i < byteArr.length; i++) {
56  bytes[i] = (byte) (255 - byteArr[i]);
57  }
58  return bytes;
59  }
60 }
61
```



3.（必做）画一张图，展示 Xmx、Xms、Xmn、Meta、DirectMemory、Xss 这些内存参数的关系。



Xmx、Xms、Xmn、Meta、DirectMemory、Xss内存参数关系

4.（选做）检查一下自己维护的业务系统的 JVM 参数配置，用 jstat 和 jstack、jmap 查看一下详情，并且自己独立分析一下大概情况，思考有没有不合理的地方，如何改进。

注意：如果没有线上系统，可以自己 run 一个 web/java 项目。

```
jstat -gc -t 13228 1000 1000
```

```
E:\GeekCode\homeWork_Week1\geekCode\src\main\java>jstat -gc -t 13228 1000 1000
Timestamp        S0C    S1C    S0U    S1U      EC       EU        OC         OU       MC     MU    CCSC   CCSU   YGC     YGCT    FGC    FGCT     GCT
         67692.9   0.0    0.0    0.0    0.0   12288.0   4096.0   12288.0    6430.2   28544.0 26872.5 3200.0 2646.0     54    0.255  17     2.227    2.482
         67693.9   0.0    0.0    0.0    0.0   12288.0   4096.0   12288.0    6430.2   28544.0 26872.5 3200.0 2646.0     54    0.255  17     2.227    2.482
         67694.9   0.0    0.0    0.0    0.0   12288.0   4096.0   12288.0    6430.2   28544.0 26872.5 3200.0 2646.0     54    0.255  17     2.227    2.482
         67695.9   0.0    0.0    0.0    0.0   12288.0   4096.0   12288.0    6430.2   28544.0 26872.5 3200.0 2646.0     54    0.255  17     2.227    2.482
```

- S0C:年轻代第一个survivor的容量是0(字节)
- S1C:年轻代第二个survivor的容量是0(字节)
- S0U:年轻代中第一个survivor目前已使用空间0(字节)
- S1U:年轻代中第二个survivor目前已使用空间0(字节)
- EC:年轻代中Eden的容量 12288(字节)
- EU:年轻代中Eden目前已使用空间4096(字节)
- OC:Old代的容量 12288(字节)
- OU:Old代目前已使用空间6430.2(字节)
- MC:元空间的容量28544(字节)
- MU:元空间目前已使用的容量26872.5(字节)
- CCSC:压缩类空间大小3200(字节)

- CCSU:压缩类空间目前使用大小2646(字节)
- YGC:从应用程序启动到采样时年轻代中gc 54次数
- YGCT:从应用程序启动到采样时年轻代中gc所用时间0.255(s)
- FGC:从应用程序启动到采样时old代(全gc)gc 17次数
- FGCT:从应用程序启动到采样时old代(全gc)gc所用时间2.227(s)
- GCT:从应用程序启动到采样时gc用的总时间2.482(s)

```
jstat -gcutil -t 13228 1000 1000
```

```
E:\GeekCode\homeWork_Week1\geekCode\src\main\java>jstat -gcutil -t 13228 1000 1000
Timestamp         S0     S1     E      O      M     CCS    YGC     YGCT    FGC    FGCT     GCT
        67443.5   0.00   0.00  33.33  52.33  94.14  82.69     54    0.255     17    2.227    2.482
        67444.5   0.00   0.00  33.33  52.33  94.14  82.69     54    0.255     17    2.227    2.482
        67445.5   0.00   0.00  33.33  52.33  94.14  82.69     54    0.255     17    2.227    2.482
        67446.5   0.00   0.00  33.33  52.33  94.14  82.69     54    0.255     17    2.227    2.482
        67447.5   0.00   0.00  33.33  52.33  94.14  82.69     54    0.255     17    2.227    2.482
```

- S0:年轻代中第一个survivor（幸存区）已使用的占当前容量百分比0%
- S1:年轻代中第二个survivor（幸存区）已使用的占当前容量百分比0%
- E:年轻代中Eden区已使用的占当前容量百分比33.33%
- O:old代已使用的占当前容量百分比52.33%
- M:元空间当前容量百分比94.14%
- CCS:压缩类空间占当前容量百分比82.69%
- YGC:从应用程序启动到采样时年轻代中gc 54次数
- YGCT:从应用程序启动到采样时年轻代中gc所用时间0.255(s)
- FGC:从应用程序启动到采样时full gc 17次数
- FGCT:从应用程序启动到采样时full gc所用时间2.227 (s)
- GCT:从应用程序启动到采样时gc用的总时间2.482(s)

```
jstat -gcutil -t 13228 1000 1000
```

Timestamp	S0C	S1C	S0U	S1U	EC	EU	OC	OU	MC	MU	YGC	YGCT	FGC	FGCT
14254245.3	1152.0	1152.0	145.6	0.0	9600.0	2312.8	11848.0	8527.3	31616.0	26528.6	11378	206.082	4	0.122
14254246.3	1152.0	1152.0	145.6	0.0	9600.0	2313.1	11848.0	8527.3	31616.0	26528.6	11378	206.082	4	0.122
14254247.3	1152.0	1152.0	145.6	0.0	9600.0	2313.4	11848.0	8527.3	31616.0	26528.6	11378	206.082	4	0.122

- S0C:年轻代第一个survivor的容量是11520(字节)
- S1C:年轻代第二个survivor的容量是68096(字节)
- S0U:年轻代中第一个survivor目前已使用空间117.9(字节)

- S1U:年轻代中第二个survivor目前已使用空间0(字节)
- EC:年轻代中Eden的容量 545344(字节)
- EU:年轻代中Eden目前已使用空间20984.9(字节)
- OC:Old代的容量 3512768(字节)
- OU:Old代目前已使用空间1965283.4(字节)
- MC:元空间的容量104308(字节)
- MU:元空间目前已使用的容量93620.4(字节)
- CCSC:压缩类空间大小15688(字节)
- CCSU:压缩类空间目前使用大小12746.5(字节)
- YGC:从应用程序启动到采样时年轻代中gc 6232次数
- YGCT:从应用程序启动到采样时年轻代中gc所用时间225.331(s)
- FGC:从应用程序启动到采样时old代(全gc)gc 110次数
- FGCT:从应用程序启动到采样时old代(全gc)gc所用时间12.023(s)
- GCT:从应用程序启动到采样时gc用的总时间237.354(s)

```
jmap -heap 21304
```

```
C:\Windows\system32>jmap -heap 21304
Attaching to process ID 21304, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.231-b11

using thread-local object allocation.
Parallel GC with 4 thread(s)

 Heap Configuration:
   MinHeapFreeRatio = 0
   MaxHeapFreeRatio = 100
   MaxHeapSize = 2120220672 (2022.0MB)
   NewSize = 44564480 (42.5MB)
   MaxNewSize = 706740224 (674.0MB)
   OldSize = 89653248 (85.5MB)
   NewRatio = 2
   SurvivorRatio = 8
   MetaspaceSize = 21807104 (20.796875MB)
   CompressedClassSpaceSize = 1073741824 (1024.0MB)
   MaxMetaspaceSize = 17592186044415 MB
   G1HeapRegionSize = 0 (0.0MB)
```

```
23
24 Heap Usage:
25 PS Young Generation
26 Eden Space:
27  capacity = 210239488 (200.5MB)
28  used = 135964208 (129.6655731201172MB)
29  free = 74275280 (70.83442687988281MB)
30  64.67110878808838% used
31 From Space:
32  capacity = 9961472 (9.5MB)
33  used = 9951024 (9.490036010742188MB)
34  free = 10448 (0.0099639892578125MB)
35  99.89511590254934% used
36 To Space:
37  capacity = 12582912 (12.0MB)
38  used = 0 (0.0MB)
39  free = 12582912 (12.0MB)
40  0.0% used
41 PS Old Generation
42  capacity = 55574528 (53.0MB)
43  used = 11563480 (11.027793884277344MB)
44  free = 44011048 (41.972206115722656MB)
45  20.807158272221404% used
46
47 15867 interned Strings occupying 2119776 bytes.
```

5. (选做) 本机使用 G1 GC 启动一个程序，仿照课上案例分析一下 JVM 情况。

```
1 java -Xmx1g -Xms1g -XX:-UseAdaptiveSizePolicy -XX:+UseG1GC -XX:MaxGCPause
Millis=200 -jar C:\Users\XM\Desktop\gateway-server-0.0.1-SNAPSHOT.jar
```

```
1 C:\Windows\system32>jmap -heap 19492
2 Attaching to process ID 19492, please wait...
3 Debugger attached successfully.
4 Server compiler detected.
5 JVM version is 25.231-b11
6
7 using thread-local object allocation.
8 Garbage-First (G1) GC with 4 thread(s)
9
10 Heap Configuration:
```

```
   MinHeapFreeRatio = 40
   MaxHeapFreeRatio = 70
   MaxHeapSize = 1073741824 (1024.0MB)
   NewSize = 1363144 (1.2999954223632812MB)
   MaxNewSize = 643825664 (614.0MB)
   OldSize = 5452592 (5.1999969482421875MB)
   NewRatio = 2
   SurvivorRatio = 8
   MetaspaceSize = 21807104 (20.796875MB)
   CompressedClassSpaceSize = 1073741824 (1024.0MB)
   MaxMetaspaceSize = 17592186044415 MB
   G1HeapRegionSize = 1048576 (1.0MB)

Heap Usage:
G1 Heap:
   regions = 1024
   capacity = 1073741824 (1024.0MB)
   used = 414187504 (394.99998474121094MB)
   free = 659554320 (629.0000152587891MB)
   38.57421725988388% used
G1 Young Generation:
Eden Space:
   regions = 384
   capacity = 664797184 (634.0MB)
   used = 402653184 (384.0MB)
   free = 262144000 (250.0MB)
   60.56782334384858% used
Survivor Space:
   regions = 11
   capacity = 11534336 (11.0MB)
   used = 11534336 (11.0MB)
   free = 0 (0.0MB)
   100.0% used
G1 Old Generation:
   regions = 0
   capacity = 397410304 (379.0MB)
   used = 0 (0.0MB)
   free = 397410304 (379.0MB)
   0.0% used

```

```
16109 interned Strings occupying 2161400 bytes.
```