Lab 04 实验报告

> 班级：222111
>
> 学号：22371001
>
> 姓名：卜欣然

Question 1

```
initialize A1
initialize A2
initialize A3
initialize A4
initialize A5
initialize A6
copy from A6
initialize B1
initialize A8
main begins
initialize A9
initialize A6
copy from A6
initialize B2
initialize A8
main ends
```

Question 2

对于非静态属性，它的初始化方法有两种：

- 在属性定义处显式初始化（如本例中的 `a6`）
- 在构造方法或非静态方法中初始化（如本例中的 `a8`）

回答下面两个问题（只考虑非静态属性）：

1. 这段代码能够证明“在属性定义处初始化的属性，比在方法中初始化的属性先被初始化”吗？

   ```
   public B(int i) {
           System.out.println("initialize B" + i);
           a8 = new A(8);
       }

       A a7 = new A(a6);
   ```

   这段代码的输出为：

   ```
   copy from A6
   initialize B1
   initialize A8
   ```

   可以看出，在属性定义处初始化的属性，比在方法中初始化的属性先被初始化。

2. 这段代码能够证明“在属性定义处初始化的属性，初始化顺序等同于他们在类定义中出现的顺序”吗?

```
static A a1 = new A(1);
    public A(int i) {
        System.out.println("initialize A" + i);
        value = i;
    }
    public A(A a) {
        System.out.println("copy from A" + a.value);
        value = a.value;
    }
    static A a2 = new A(2);
```

这段代码的输出为:

```
initialize A1
initialize A2
```

可以看出，在属性定义处初始化的属性，初始化顺序等同于他们在类定义中出现的先后顺序。

¶ Question 3

请尝试仿照 Question2 的内容，**描述静态属性的初始化方式和实际初始化时的顺序。**

- `static` 静态属性定义时的初始化
- `static` 的方法块中的初始化
- 对象属性（非静态变量）定义时的初始化
- 非静态方法快的初始化
- 构造方法（函数）中的初始化

¶ Question 4

已知 `static` 属性的初始化、`static` 块的执行，只在 JVM 进行类加载的时候执行，**请回答下面的问题。**

这段代码能够证明“在类的实例第一次被构造、或类的静态属性和静态方法第一次被访问时，JVM 会执行类加载”吗？如果不能，请尝试修改代码并证明。

不能证明。将以下这行代码注释掉，

```
static B b1 = new B(1);
```

输出则会变为:

```
main begins
initialize A1
initialize A2
initialize A9
initialize A3
initialize A4
initialize A5
initialize A6
copy from A6
initialize B2
initialize A8
main ends
```

在类的实例第一次被构造、或类的静态属性和静态方法第一次被访问时，JVM 会执行类加载。类加载只会进行一次，这一次类加载会完成所有的静态初始化工作。

¶ Question 5

其他的外部类可以通过 `new Singleton()` 来构造一个新的 `Singleton` 变量吗?

不能。因为构造方法是 private 的，其他的外部类没有权利访问。

¶ Question 6

本题给出的 `Singleton` 类的写法被称为单例模式，是因为这个类最多只可能有 1 个实例同时存在。**为什么只可能有 1 个？这个唯一的实例在什么时候被构造?**

因为构造方法是 private 的，只能在类内部访问，而类内部仅仅 new 了一个实例，那么这个类就只可能有一个实例。

因为唯一的实例是静态变量，因此在类加载时被构造。

¶ Question 7

请写出任意一种外部类调用 Singleton 类的 `foo()` 的方法。