Lab 04 Assignment

> 班级：212113
>
> 学号：21371220
>
> 姓名：杨硕

Question1

程序的输出为：

```
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
```

Question2

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”。

这段代码不能够证明“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”。

Question3

对于静态属性，初始化方法有：

- 在属性定义处显式初始化；
- 在静态代码块或静态方法中初始化；

静态属性的初始化顺序为：

在属性定义处初始化的静态属性，比在方法中初始化的属性先被初始化；

在属性定义处初始化的静态属性，初始化顺序等同于在类定义出现的顺序；

静态属性比非静态属性先被初始化；

Question4

不能，修改代码部分

```
public class Initialization {
    //static B b1 = new B(1);
    static B b2;
    public static void main(String[] args) {
        System.out.println("main begins");
        A a9 = new A(9);
        b2 = new B(2);
        System.out.println("main ends");

    }
}
```

将`static B b1 = new B(1);`这一句注释掉，再次运行代码，输出结果为：

```
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
```

可以看到，先输出了`main begins`，后才进行了初始化，与原输出对比，说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

Question5

不能，因为`Singleton()`在`Singleton`中是`private`访问控制。

Question6

`Singleton`类的构造方法为`private`访问级别，确保了在`Singleton`类外用户无法通过`new`方法创建新实例，并且在`Singleton`类中创建了`uniqueInstance`这一唯一实例，向外部提供的获得该类实例的方法`getInstance`返回的也是这一唯一实例，这样就确保了`Singleton`类最多只能有一个实例同时存在。这个唯一的实例在类装载的时候被构造。

Question7

外部类调用 Singleton 类的 foo()的方法：

```
Singleton singleton = Singleton.getInstance();
singleton.foo();
```