Lab 2: 类与对象&封装

> **写在前面**：文件提交格式如下，与Lab1相同，提交压缩包格式为 `zip`
>
> ```
> 23373000-XXX-Lab02
> |-- 23373000-XXX-Lab02-问答报告.pdf
> |-- 23373000-XXX-Lab02-实验报告.pdf
> |-- Q2
> |   \-- Main.java
> |-- Q3
> |   |-- Test.java
> |   \-- Hello.java
> \-- Q6
>  \-- Test.java
> ```
>
> **温馨提示：**本次实验仅有Question10和Question11两道编程题需要提交 `.java` 文件，其它题目不需要提交代码文件，只需要按照题目要求在问答报告里体现即可
> 祝各位实验顺利🧑

实验目的

- 理解并掌握类的概念
- 理解并掌握对象的概念
- 理解类与对象的关系
- 理解面向对象中抽象过程
- 理解面向对象中的消息
- 理解 Java 程序的基本结构并能灵活使用
- 理解并掌握 Java 类的定义（成员变量、成员方法和方法重载）
- 理解并掌握 Java 类的构造函数（默认构造函数、带参数构造函数），理解重载的构造函数并灵活使用
- 理解 Java 垃圾内存自动回收机制
- 理解并掌握 Java 类变量和类方法
- 理解封装含义
- 理解信息隐藏的必要性
- 掌握访问控制修饰符的使用

    - 私有成员（变量和方法）的理解和使用
    - 共有成员的理解和使用
    - 保护成员的理解和使用
    - 使用不加任何权限修饰符的成员
- 加深对“类和对象”的理解

注意事项

本次代码量相对较多，请大家注意文件结构的规划。🧐

随着代码量的增加，建议建立一个自己的统一且良好的**代码风格**，比如**命名风格**（camelCase、PascalCase 等）、缩进方式（空格数量、switch-case 缩不缩进等）、开闭大括号换不换行等容易引发战争（迫真）的东西，以养成良好的编程习惯。

> 比如命名规范，可以参考 Oracle 官网的 Naming Conventions。

编程题最好为每一个类编写一个完备的测试类，覆盖尽可能多的输入、函数调用、输出，以证明代码正确实现了功能。

如果编程题使用了 `package` 语句，应当确保提交时目录结构和 `package` 语句表达的包结构相同。（IDE 很多时候会帮你做。）

编程题在给出了具体需求的情况下，**可以根据自己的需要添加额外的方法**，合理即可。

---

1. 类的定义

Question 1

编译 Sample 类，并查看编译结果。

```java
public class Sample {
    int x; // 1
    long y = x; // 2

    public void f(int n) {
        int m; // 3
        int t = n + m; // 4
        m = 20;
    }

    public static void main(String[] args) {
        Sample t = new Sample();
```

```
13          t.f(5);
14          System.out.println(t.x);
15      }
16  }
```

1. 注释标记的哪些行会导致编译错误？**将错误截图，并说明原因。**
2. 区分变量 `x` 和 `m`，回答**谁需要初始化才能使用，为什么**

---

2. 函数重载

Question 2

阅读下面代码

```
1   public class Overload {
2       Overload(int m) {}
3
4       Overload(double m) {}
5
6       int Overload(int m) {
7           return 23;
8       }
9
1       void Overload(double m) {}
0   }
```

对于 `Overload` 类，下面哪些叙述是错误的？**给出你的回答，并说明原因。**

- a. `Overload(int m)` 与 `Overload(double m)` 互为重载的构造方法。
- b. `int Overload(int m)` 与 `void Overload(double m)` 互为重载的非构造方法。
- c. `Overload` 类有 2 个构造方法，尝试调用默认构造方法 `Overload()` 会无法通过编译。
- d. `Overload` 类有 3 个构造方法。

---

3. 初始化（Initialization）

阅读并运行下面这段代码，尝试理解 Java 中初始化的顺序。

```
1   class A {
2       int value;
3       static A a1 = new A(1);
```

```
4
5       public A(int i) {
6           System.out.println("initialize A" + i);
7           value = i;
8       }
9
10      public A(A a) {
11          System.out.println("copy from A" + a.value);
12          value = a.value;
13      }
14
15      static A a2 = new A(2);
16
17  }
18
19  class B {
20      A a8;
21      // A a7 = new A(a6);
22      A a6 = new A(6);
23      static A a3 = new A(3);
24      static A a4;
25
26      static {
27          a4 = new A(4);
28      }
29
30      static A a5 = new A(5);
31
32      public B(int i) {
33          System.out.println("initialize B" + i);
34          a8 = new A(8);
35      }
36
37      A a7 = new A(a6);
38  }
39
40  public class Initialization {
41      static B b1 = new B(1);
42      static B b2;
43
44      public static void main(String[] args) {
45          System.out.println("main begins");
46          A a9 = new A(9);
47          b2 = new B(2);
48          System.out.println("main ends");
49      }
```

```
50    }
```

Quesion 3

阅读上面这段代码，**给出程序的输出**。

Question 4

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？
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？

Question 5

请尝试仿照 Question4 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**。

Question 6

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**。

1. 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。
2. 基于static关键字，`带有static关键字的方法、变量、代码块` 可以调用什么？反过来 `不带有static关键字的方法、变量、代码块` 可以调用什么？（`带有static关键字的方法、变量` or `不带有static关键字的方法、变量` or `都可以`）

> **题外话**
>
> 懒加载：Lazy Load，对某资源只在需要时才寻找其存在并初始化；对立面是预加载。
>
> 预加载：提前加载好所有资源，等待使用资源的那一刻。
>
> 对于一些使用频率较低但初始化开销很大的资源，懒加载可以避免他们给程序的初始化增加过多的负担。
>
> JVM 的类加载是懒加载，**只有在程序第一次使用到某个类时才去尝试读取其 `.class` 文件**。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。更多内容会在后续课程讲解 RTTI 和反射的时候提到。

> 例如，在游戏编程中，当某一个类的所有实例都使用同一批贴图文件时，可以将贴图资源声明为 static 属性并直接（或在 static 块）初始化。让类的贴图属性引用这些静态资源，这样就可以避免为每一个对象构造单独的贴图文件导致的内存浪费和时间浪费。

---

4. 单例模式（Singleton Pattern）

阅读下面这段代码，它实现了经典设计模式之一：单例模式。

```java
/**
 * Singleton 一个最简单的单例模式的实现
 */
public class Singleton {
    private static final Singleton uniqueInstance = new Singleton();


    private Singleton() {
    }

    public static Singleton getInstance() {
        return uniqueInstance;
    }

    public void foo() {
        System.out.println("Aha!");
    }
}
```

Question 7

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

Question 8

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?**

Question 9

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

> **题外话**

> 这里的 `uniqueInstance` 初始化方法不是懒加载（Lazy Load）的，因为 `uniqueInstance` 在类加载时就被初始化了，虽然我们可能最终并用不到它。你可以思考一下如何实现一个懒加载的单例模式。(不要求写在问答报告里。)

---

5. 编程题

Question 10

编写程序，在其中定义两个类

- `Person` 类:
  - 属性有 `name`、`age` 和 `sex`；
  - 提供你认为必要的构造方法;
  - 方法 `setAge()` 设置人的合法年龄（`0` ~ `130`）；
  - 方法 `getAge()` 返回人的年龄;
  - 方法 `work()` 输出字符串 `working`；
  - 方法 `showAge()` 输出 `age` 值。
- `TestPerson` 类:
  - 创建 `Person` 类的对象，设置该对象的 `name`、`age` 和 `sex` 属性;
  - 调用 `setAge()` 和 `getAge()` 方法，体会 Java 的封装性;
  - 创建第二个对象，执行上述操作，体会同一个类的不同对象之间的关系。

除了提交 `.java` 代码外，你需要在解答中说明如下内容

```
目录名为：Question10（或者你自己的命名）
文件名有：（如果你放了一个项目进来，则说明你的项目结构以及入口位置）
```

Question 11

编写一个 Java 命令行程序，**只从标准输入读取一行用户输入**，判断这行输入是否是一个没有前导 0 的无符号整数；如果是，则还要判断该数字是否是一个回文数。**自定义你程序的输出**，要求能直观展现程序判断的结果。

对于"没有前导 0 的无符号整数"的定义：

- 是一个字符串 `s`；
- `s` 的长度至少是 `1`，没有上限要求;
- `s` 的字符集 `{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}`，其他所有字符都不应该出现在 `s` 中;

- 当 `s` 的长度大于 `1` 时，若从其首部开始有若干个连续字符 `0`，那么这些字符 `0` 都叫做 `s` 的“前导 0”。
  - 比如数字串 00010020，有三个前导 0
  - 比如数字串 01，有 1 个前导 0
  - 比如数字串 102030，没有前导 0
  - 比如数字串 0，没有前导 0

本题对于回文数的定义：

- 是一个字符串 `s`
- `s` 的长度至少是 `1`，没有上限要求
- `s` 的字符集 `{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}`，其他所有字符都不应该出现在 `s` 中
- 将 `s` 中的字符逆序排列并去除前导 `0` 得到的数字串 `r`，有 `s` 和 `r` 完全相同
  - s = 123 时，r = 321，不相同，s 和 r 都不是回文数
  - s = 12321 时，r = 12321，相同，s 是回文数
  - s = 12100 时，r = 121，不相同，s 不是回文数，但 r 是回文数
  - s = 1 时，r = 1，相同，s 是回文数
  - s = 0 时，r = 0，相同，s 是回文数

本题将输入的一行字符视为一个完整的字符串，如果输入的是诸如“121 121”这样包含空格的串，虽然 121 是回文数，**但是整个串不应该被认为是回文数**。如果将字符集扩充为包含空格的其他字符集，那么“121 121”就是一个该字符集下的回文串，不过本题的字符集限定为由 0 ~ 9 这十个数字组成的字符集。

除了提交 `.java` 代码外，你需要在解答中说明如下内容：

```
目录名为：Question11 （或者你自己的命名）
文件名有：（如果你放了一个项目进来，则说明你的项目结构以及入口位置）
```