Java中的设计模式

1. 设计模式：对问题行之有效的解决方式，其实它是一种解决问题的思想。
2. Java中有23中设计模式。
3. **单例设计模式：**
4. **可以解决的问题：可以保证一个类在内存中的对象唯一性。**
5. **什么时候需要对象唯一性？**

**对于多个程序使用同一个配置信息对象时，就需要保证该对象的唯一性。**

1. **如何保证对象的唯一性？**
   1. **不允许其他程序利用new创建该类对象：私有化该类构造函数；**
   2. **在该类中创建一个本类实例：通过new在本类中创建一个该类对象；**
   3. **对外提供一个方法，让其他程序可以获取该对象：创建一个公有方法，将创建的对象返回。**
2. **实现单例模式需要在类中添加三句话：**
   1. **私有化构造函数 ：private Signal（）{}**
   2. **在本类中创建一个本类对象：**

**private static Signal signal = new Signal（）；**

**解释：A、为什么需要是static，因为需要定义一个方法getInstance（）返回此对象，但是这个方法需要外界利用类名进行调用，所以此方法必须是static静态的，静态方法中不能访问非静态变量，所以此类引用变量必须是static静态的。B、为什么需要设置为private；因为如果不设置为private的，则外界就可以随意通过类名进行调用此对象（引用），不可控制，定义为private后，外界只可以通过getInstance（）方法进行获取此对象的引用，同时可以在此方法中设置一些权限判断，从而进行控制哪些可以获取此对象。**

* 1. **创建一个公有方法getInstance（），将创建的对象返回。**

**public static Signal（\*\*\*\*\*\*）{**

**\*\*\*\*\*\*\*\*//通过设置些判断，例如if语句**

**return signal；**

**}**

1. 实例：饿汉式：**public** **class** Signal {

**private** **static** Signal *s* = **new** Signal();//一定要私有化，通过getInstance（）方法进行可控制

**private** Signal() {

}//构造方法私有化

**public** **static** Signal getInstance(**int** password) {

**if** (password == 328)//权限验证

**return** *s*;

**else**

**return** **null**;

}

**private** String name;// 一般属性都是private的，不能随意让人调用，所以对应两个方法对其设置和获取setName（）和getName（）。

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getName() {

**return** **this**.name;

}

}

\*\*\*\*\*\*\*\*\*主方法的类\*\*\*\*\*\*\*\*

**public** **class** SignalTest {

**public** **static** **void** main(String[] args) {

Signal ss1 = Signal.*getInstance*(328);

Signal ss2 = Signal.*getInstance*(328);

ss1.setName("赵红 ");

ss2.setName("赵西元");

System.***out***.println(ss1.getName());

System.***out***.println(ss2.getName());

}

}

**输出结果是两个“赵西元”。此题有可能也是面试题目。**

1. 单例模式的另外一种表现形式：延迟加载形式），

**public** **class** Signal {

**private** **static** Signal *s* = **null**;//一定要私有化，通过getInstance（）方法进行可控制

**private** Signal() {

}//构造方法私有化

**public** **static** Signal getInstance(**int** password) {

**if** ((s == null)&&(password == 328))//权限验证

**return** s = new Signal();//这样的好处是可以在需要创建对象时才生成对象，而第一种形式是在类已加载的时候就会生成对象。

**else**

**return** **null**;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

}

1. 单例模式的两种表现形式（饿汉式和懒汉式）的区别：

第一种形式是在类已加载的时候就会生成对象，称为**饿汉式**；第二种形式是类加载进来，但没有对象，只有在调用getInstance（）方法时才创建对象（称为延迟加载形式）又称为**懒汉式**。

1. 表面上看上去懒汉式有些好处，但是也存在很多弊端，在多线程并非访问时，可能不能保证对象的唯一性。
2. 面试的时候一般考察的是懒汉式，因为其中涉及的知识点较多；但是在实际开发中用的最多的是饿汉式。