# 单例模式简介

## 概述

1. 单例模式可以确保系统中的某个类只有一个实例，从而方便对实例个数进行控制，节约系统资源。
2. 单例模式有3个要点：一是某个类只能有一个实例；二是这个类必须自行创建这个实例；三是它必须向整个系统提供这个实例。
3. 单例模式的定义如下：确保某个类只有一个实例，并提供一个全局访问点来访问这个唯一实例。

## 结构



1. 单例模式只包含一个角色，即单例类（Singleton）；
2. 对于Singleton（单例），在单例类的内部创建它的唯一实例，并通过静态方法getInstance( )让客户端可以使用它的唯一实例；
3. 为了防止在外部对单例类实例化，单例类中有且只有一个构造方法，并且该构造方法是私有化的，即将访问权限设置为private。

# 实现

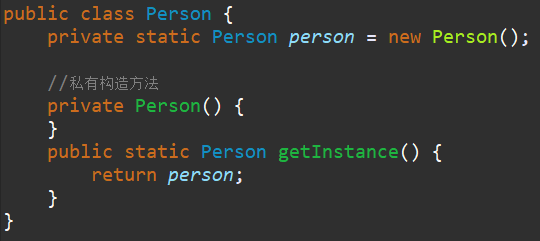
单例模式根据其实例化的时机不同，可分为饿汉式和懒汉式。

## 饿汉式

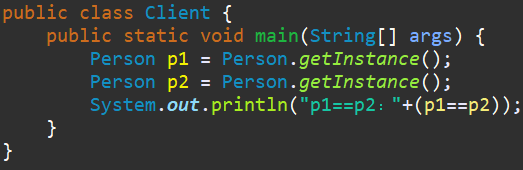
饿汉式单例类（Eager Singleton）是在将单例对象定义为静态变量的时候就进行实例化，因此在类加载时单例对象就已创建。



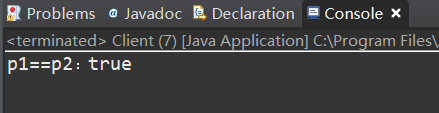
1）Person（饿汉式单例类）



2）编写客户端测试类，测试饿汉式单例模式：



运行结果如下：



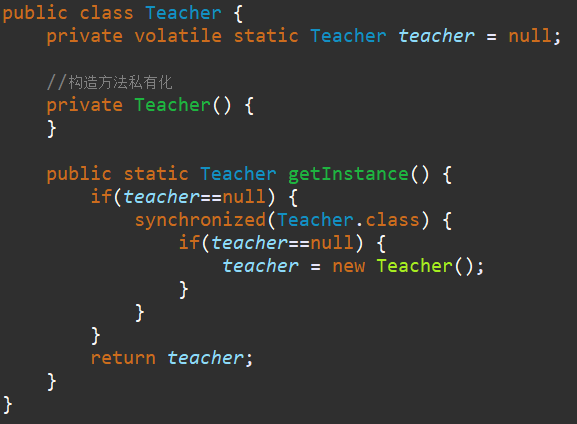
由此可以证明单例模式可以确保某个类只有一个实例。

## 懒汉式

懒汉式单例类是在第一次引用时才创建单例对象。

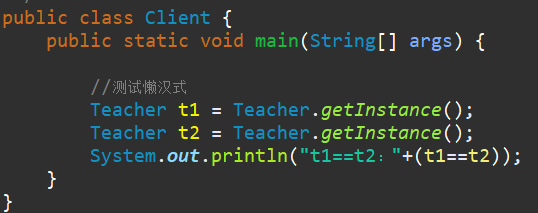


1. Teacher（懒汉式单例类）

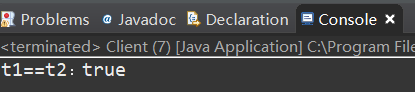


为了保证线程问题，需要在获取单例对象的方法中使用双重检查锁定（关于双重检查锁定技术请参考教材），也需要给静态成员变量加上修饰符volatile，被volatile修饰的成员变量可以确保多个线程都能够正确处理，且只有JDK1.5及以上版本才支持，由于valatile关键字会屏蔽Java虚拟机所做的一些代码优化，所有可能会导致系统的运行效率降低，因此使用懒汉式实现单例模式不是一种完美的方式。

1. 编写客户端测试类，测试懒汉式单例模式



运行结果如下：



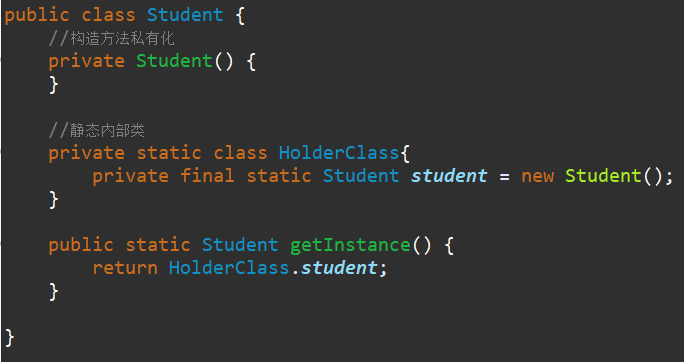
由此可以证明单例模式可以确保某个类只有一个实例。

## 饿汉式与懒汉式的对比

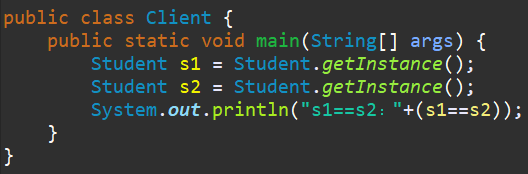
饿汉式单例类在类加载时就将自己实例化，它的优点在于无需考虑多个线程同时访问的问题，可以确保实例的唯一性；从调用速度和反应时间角度来讲，由于单例对象一开始就得以创建，因此要优先于懒汉式单例。但是无论系统在运行时是否需要使用该单例对象，由于在类加载时该对象就需要创建，因此从资源利用效率角度来讲饿汉式单例不及懒汉式单例，而且在系统加载时由于需要创建饿汉式单例对象，加载时间可能会比较长。

懒汉式单例类在第一次使用时创建，无须一直占用系统资源，实现了延迟加载，但是必须处理多个线程同时访问的问题，特别是当单例类作为资源控制器，在实例化时必然涉及资源初始化，而资源初始化很有可能耗费大量时间，这意味着出现多线程同时首次引用此类的几率变得较大，需要通过双重检查锁定等机制进行控制，这将导致系统性能受到一定影响。

由此可见，无论是饿汉式还是懒汉式，都会存在一些问题，为此，在Java语言中可以通过Initialization on Demand Holder（IoDH）技术来实现单例模式。在IoDH中，需要在单例类中增加一个静态内部类，在该内部类中创建单例对象，再将该单例对象通过getInstance( )方法返回给外部使用，实现代码如下：



进行测试：



通过使用IoDH既可以实现延迟加载，又可以保证线程安全，不影响性能，不失为一种最好的Java语言单例模式实现方式。

# 优缺点及适用环境

## 优点

单例模式的优点主要如下：

1. 单例模式提供了对唯一实例的受控访问。因为单例类封装了它的唯一实例，所以它可以严格控制客户端怎样以及何时访问它。
2. 由于在系统内存中只存在一个对象，因此可以节约系统资源，对于一些需要频繁创建和销毁的对象，单例模式无疑可以提高系统的性能。
3. 允许可变数目的实例。基于单例模式可以进行扩展，使用与控制单例对象相似的方法来获得指定个数的实例对象，既节省系统资源，又解决了由于单例对象共享过多有损性能的问题。
4. 在单例模式的基础上可以演变为多例模式，自行提供指定书目实例对象的类可称为多例类。

## 缺点

单例模式的缺点主要如下：

1. 由于单例模式中没有抽象层，因此单例类的扩展有很大的困难。
2. 单例类的职责过重，在一定程度上违背了单一职责原则。因为单例类既提供了业务方法，又提供了创建对象的方法，将对象的创建和对象本身的功能耦合在一起。
3. 现在有很多面向对象语言（如Java）的运行环境都提供了自动垃圾回收技术，因此如果实例化的共享对象长时间不被利用，系统会认为它是垃圾，会自动销毁并回收资源，下次利用时又将重新实例化，这将导致共享的单例对象状态的丢失。

## 适用环境

在以下情况下可以考虑使用单例模式：

1. 系统只需要一个实例对象，例如系统要求提供一个唯一的序列号生成器或资源管理器，或者因为资源消耗太大而只允许创建一个对象。
2. 只允许使用一个公共访问点，除了该公共访问点，不能通过其他途径访问该类实例。