设计模式代表了最佳的实践，通常被有经验的面向对象的软件开发人员所采用的。设计模式是软件开发人员在软件开发过程中面临的一般问题的解决方案。这些解决方案是众多软件开发人员经过相当长的一段时间的试验和错误总结出来的。

设计模式是对面向对象设计中反复出现的问题的解决方案。算法不是设计模式，因为算法致力于解决问题而非设计问题。设计模式通常描述了一组相互紧密作用的类与对象。设计模式提供一种软件设计的公共语言，使得熟练设计者的设计经验可以被初学者和其他设计者掌握。

总的来说，设计模式分为三大类：

创建型模式，共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。

结构性模式，共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式

行为性模式，共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

设计模式的六大原则：

总原则：开闭原则

开闭原则就是说对扩展开放，对修改关闭。在程序需要进行扩展的时候，不能去修改原有的代码，而是要扩展原有代码，实现一个热插拔的效果。所以一句话概括就是：为了使程序的扩展性好，易于维护和升级。想要达到这个效果，我们需要使用接口和抽象类等。

1. 单一职责原则：

不要存在多于一个导致类变更的原因，也就是说每个类应该实现单一的职责，若不然，就应该把类拆分

1. 里氏替换原则

里氏替换原则中，子类对父类的方法尽量不要重写或重载。因为父类代表了定义好的结构，通过这个规范的接口与外界交互，子类不应该随意破坏它。

1. 依赖倒置原则：

这是开闭原则的基础，具体内容：面向接口编程，依赖于抽象而不依赖于具体。写代码时用到具体类时，不要与具体类交互，而与具体类的上层接口交互。

1. 接口隔离原则：

这个原则的意思是：每个接口中不存在子类用不到却必须实现的方法，如果不然，就要将接口拆分。使用多个隔离的接口，比使用单个接口（多个接口方法集合到一个的接口）要好。

1. 最少知道原则

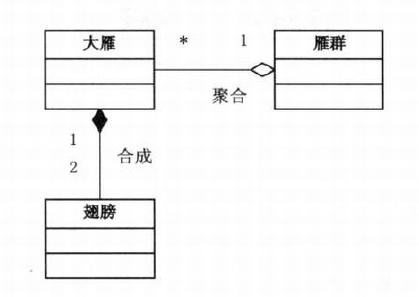
就是说：一个类对自己依赖的类知道的越少越好。也就是说无论被依赖的类多么复杂，都应该将逻辑封装在方法的内部，通过public方法提供给外部。这样当被依赖的类变化时也能最小的影响该类。

1. 合成复用原则

原则是尽量首先使用合成/聚合的方式，而不是使用继承。

聚合（aggrevation）表示一种弱的“拥有”关系，体现的是A对象可以包含B对象但B对象不是A对象的一部分

合成（Composition）则是一种强的“拥有”关系，体现了严格的部分和整体关系，部分和整体的生命周期一样。



继承是一种强耦合的结构。子类随父类改变而改变，一定要在是‘is-a’的关系在考虑使用。

创建类模式：

单例模式（singleton pattern）

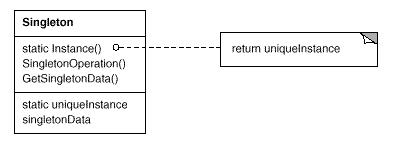
动机（motivation）：

在软件系统中，经常有这样一些特殊的类，必须保证它们在系统中只存在一个实例，才能保证他们的逻辑正确性、以及良好的效率。

如何绕过常规的构造器，提供一种机制来保证一个类只创建一个实例呢？

这应该是类设计者的责任，而不是类的使用者的责任。

结构图：



意图：

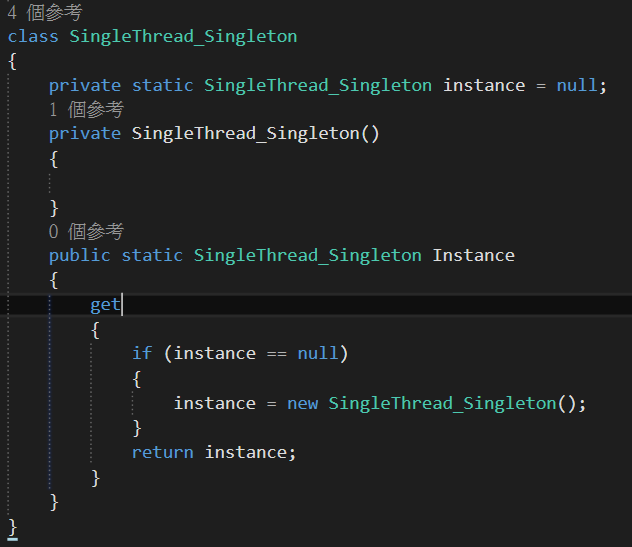
保证一个类仅有一个实例，并提供一个访问它的全局访问点。

适用性：

1. 当类只能有一个实例而且客户可以从一个总所周知的访问点访问它时。
2. 当这个唯一实例应该是通过子类化可扩展的，并且客户应该无需更改代码就能使用一个扩展的实例时

代码实现：

1. 单线程singleton实现



以上代码在单线程的情况下不会出现任何情况。但是在多线程的情况下是不安全的。

如两个线程同时运行到if(instance == null)判断是否被实例化，一个线程判断为True后，在进行创建instance=new SingleThread\_Singleton()；之前，另一个线程也在判断(instance == null)，结果也为True。这样就违背了Singleton模式的原则（保证一个类只有一个实例）。

怎么样在多线程的情况下实现Singleton？

1. 多线程Singleton实现：

关键字volatile：

volatile关键字指示一个字段可以由多个同时执行的线程修改。声明为volatile的字段不受编译器优化（假定由单个线程访问）的限制。这样可以确保该字段在任何时间呈现的都是最新的值。

也就是说：

Volatile多用于多线程的环境，当一个变量定义为volatile时，读取这个变量的值时每次都是从memory里面读取而不是从cache读。这样做是为了保证读取该变量的信息都是最新的，而无论其他线程如何更新这个变量。

如果还不是很理解，可以先看下背景知识点：

多个线程同时访问一个变量时，CLR（common language runtime）为了效率会进行优化，比如“允许线程进行本地缓存”，这样就可能导致变量访问的不一致性。Volatile就是为了解决这个问题；volatile修饰的变量，不允许线程进行本地缓存，每个线程的读写都直接操作在共享内存上，这就保证了变量始终具有一致性。

Lock的用法：

lock关键字可以用来确保代码块完成运行，而不会被其他线程中断。它可以把一段代码定义为互斥段。在同一时刻内只允许一个线程进入执行，而其他线程必须等待。

在多线程中，每个线程都有自己的资源，但是代码区是共享的，即每个线程都可以执行相同的函数。这样可能带来几个线程同时执行一个函数导致数据混乱的问题。

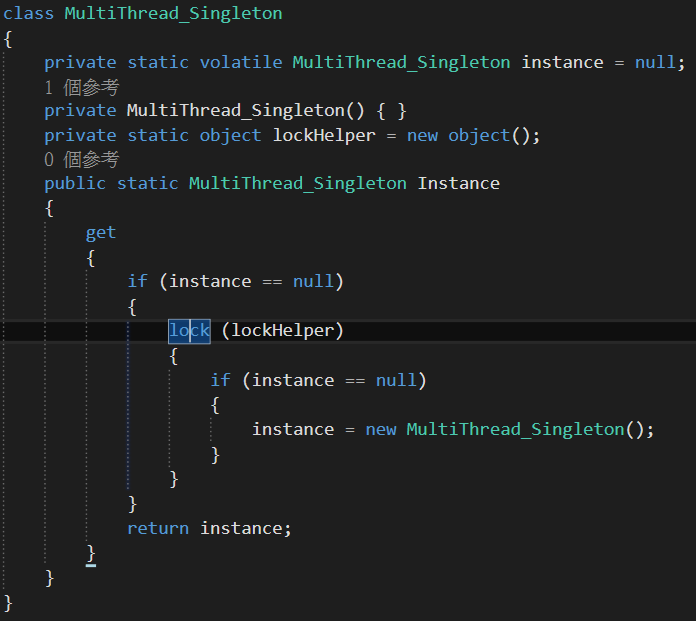
Lock是针对线程一级的（最好了解下进程、**应用域**、线程的概念），而在.NET中应用域是否会对lock起隔离作用，应该是不在同一应用域中的线程无法通过lock来中断。另外也最好能了解一下数据段、代码段、堆、栈等概念。

应用域：为了保证代码的健壮性CLR希望不同服务功能的代码之间相互隔离，这种隔离可以通过创建多个进程来实现，但操作系统中创建进程是即耗时又耗费资源的，所以在CLR中引入了APPDomain的概念，APPDomain主要是用来实现同一进程中的各APPDomain之间的隔离，APPDomain可以用以下特征来描述它的全貌：

* APPDomain概念并不存在于操作系统，而只存在于.net中并且APPDomain不可脱离进程单独存在。
* 一个进程中可以由多个APPDomain，并且每个之间相互隔离，次可以理解为APPDomain是.net程序中的“进程”，在一个APPDomain中创建的对象只属于本APPDomain，默认应用域只有在进程终止时才会被销毁，如果主动的调用Unload去卸载默认应用域，会抛出一个CannotUnloadAPPDomainException。
* <https://www.cnblogs.com/asminfo/p/3999412.html>

在C#lock关键字定义如下：

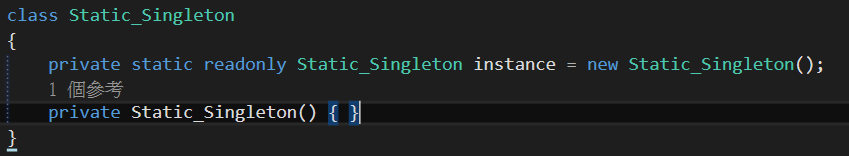
lock(expression) statement\_block，其中expression代表你希望跟踪的对象，通常是对象引用。



如上代码中的lockHelper即定义的一个obj，当线程A执行到lock语句时，判断lockHelper是否已经申请了互斥锁，如果不存在，则申请一个新的互斥锁，这时线程A进行lock里面了。当线程A执行完毕，释放互斥锁，线程B才能申请新的互斥锁并执行lock里面的代码。

此程序对多线程是安全，使用了一个辅助对象lockHelper，保证了只有一个线程创建实例且只能创建一个实例。

1. 静态Singleton实现



以上创建一个readonly静态实例只能在类中创建，所以无法在类外进行实例的创建。

以上代码等同于：

