建造者模式：

Builder模式的起源：

假设创建游戏中的一个房屋设施，该房屋的构建由几部分组成，且各自部分富于变化。如果使用最直观的设计方法，每一个房屋部分的变化，都将导致房屋构建的重新修正。

动机：

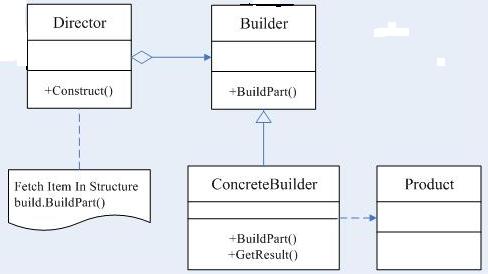
在软件系统中，有时候面临一个“复杂对象”的创建工作，其通常由各个部分的子对象用一定算法构成；由于需求的变化，这个复杂对象的各个部分经常面临着剧烈的变化，但是将它们组合到一起的算法却相对稳定。

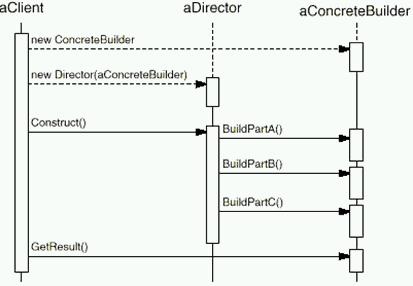
如何应对这种变化呢？如果提供一种“封装机制”来隔离出“复杂对象的各个部分”的变化，从而保持系统中的“稳定构建算法”不随需求的改变而改变？

意图：

将一个复杂对象的构建与其表示相分离，使得同样的构建过程可以创建不同的表示。

结构图：





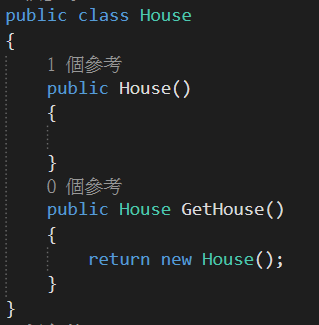
客户要求创建一个复杂对象，Director指导该对象的各个部分的组成，而builder根据director的指导来实际构建这个对象。

适用性：

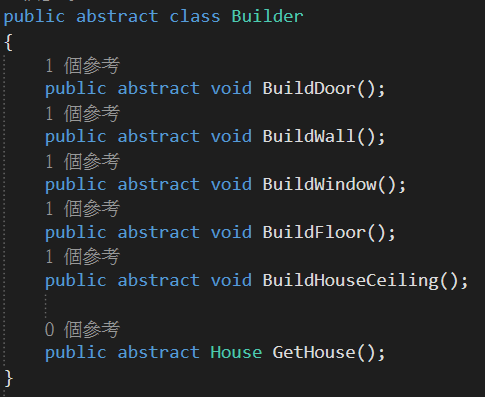
1. 当创建复杂对象的算法应该独立于该对象的组成部分以及它们的装配方式时。
2. 当构造过程必须允许被构造的对象有不同的表示时。

实例代码：

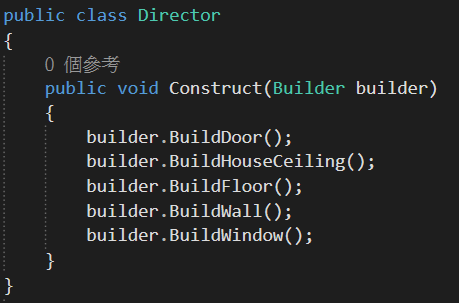
House类：



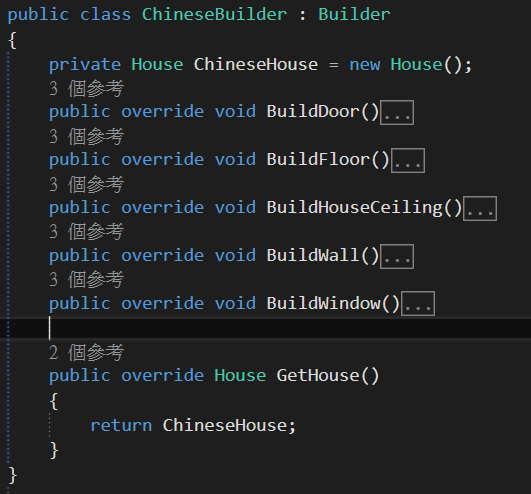
Builder类：

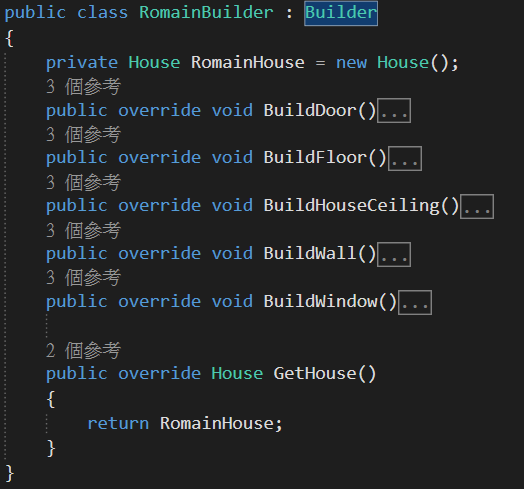


Director类：这一部分是组合到一起的算法（相对稳定）

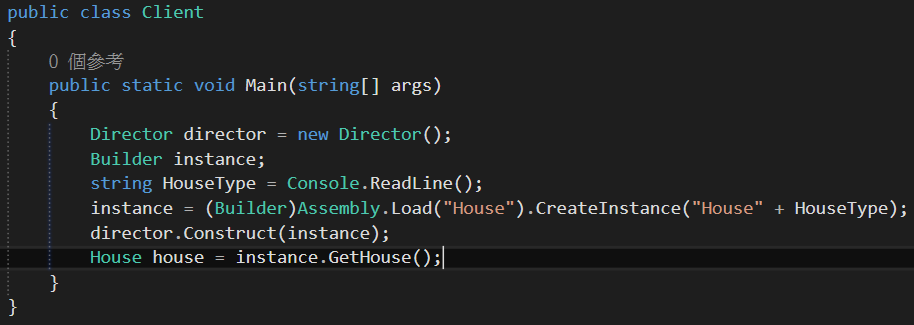


两个不同表示的Builder：





ChineseBuilder和RomanBuilder这两个是：这个复杂对象的两个部分进程面临着剧烈的变化



Abstract Factory模式解决“系列对象”的需求变化。

Builder模式解决“对象部分”的需求变化。