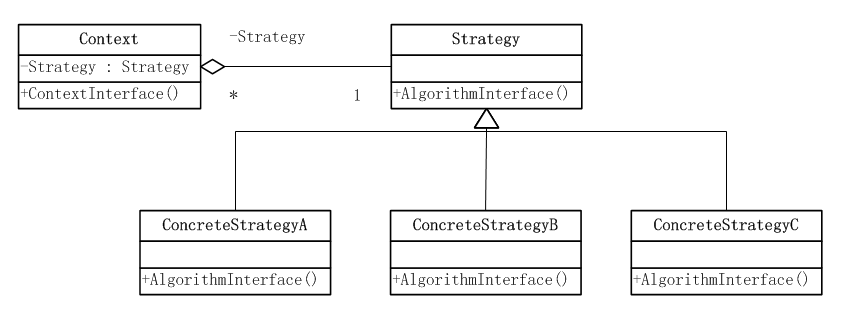
1. 概述

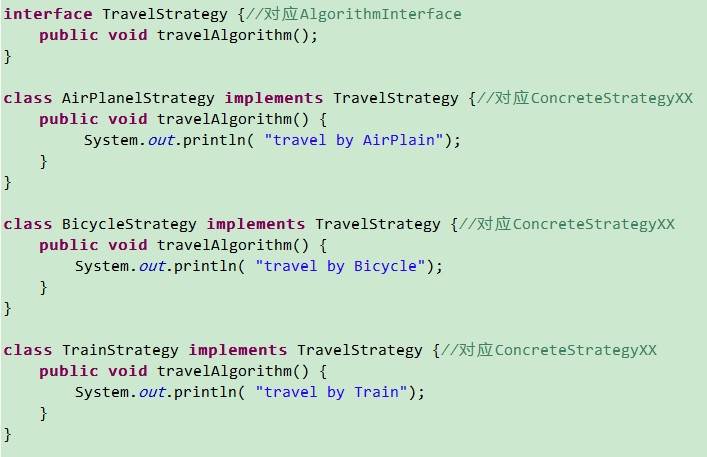
在软件开发中也常常遇到类似的情况，实现某一个功能有多种算法或者策略，我们可以根据环境或者条件的不同选择不同的算法或者策略来完成该功能。如查找、排序等，一种常用的方法是硬编码(Hard Coding)。在一个类中，如需要提供多种查找算法，可以将这些算法写到一个类中，在该类中提供多个方法，每一个方法对应一个具体的查找算法；当然也可以将这些查找算法封装在一个统一的方法中，通过if…else…或者case等条件判断语句来进行选择。这两种实现方法我们都可以称之为硬编码，如果需要增加一种新的查找算法，需要修改封装算法类的源代码；更换查找算法，也需要修改客户端调用代码。在这个算法类中封装了大量查找算法，该类代码将较复杂，维护较为困难。如果我们将这些策略包含在客户端，这种做法更不可取，将导致客户端程序庞大而且难以维护，如果存在大量可供选择的算法时问题将变得更加严重。

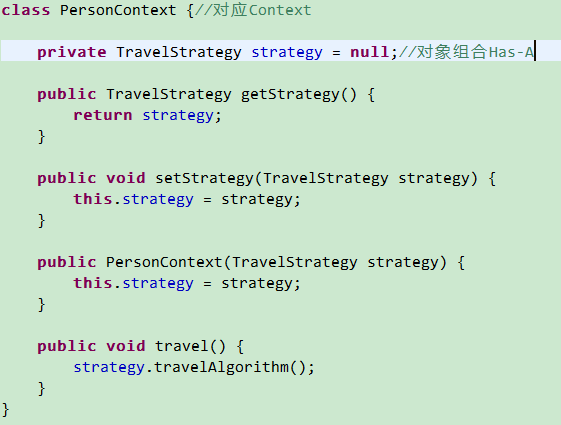
1. 策略模式对应于解决某一个问题的一个算法族, 允许用户从该算法族中任选一个算法解决某一问题同时可以方便的更换算法或者增加新的算法. 并且由客户端决定调用哪个算法. 例如根据不同用户选择不同的报价策略等
2. 本质: 分离算法, 选择实现
3. 策略模式结构图



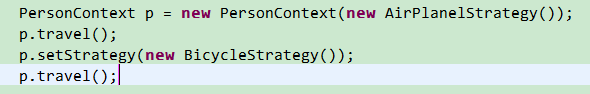
* 1. 环境类(Context): 用一个ConcreteStrategy对象来配置。**维护一个对Strategy对象的引用(对象组合Has-A)**。可定义一个接口来让Strategy访问它的数据。
  2. 抽象策略类(Strategy): 定义所有支持的算法的公共接口。 Context使用这个接口来调用某ConcreteStrategy定义的算法。
  3. 具体策略类(ConcreteStrategy): 以Strategy接口实现某具体算法。

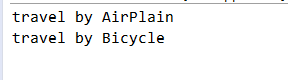
1. 例，出行方式选择





测试：



输出：

1. 优缺点：
   1. **Strategy模式有下面的一些优点：**
      1. 相关算法系列 Strategy类层次为Context定义了一系列的可供重用的算法或行为。 继承有助于析取出这些算法中的公共功能。
      2. 提供了可以替换继承关系的办法： 继承提供了另一种支持多种算法或行为的方法。你可以直接生成一个Context类的子类，从而给它以不同的行为。但这会将行为硬行编制到 Context中，而将算法的实现与Context的实现混合起来,从而使Context难以理解、难以维护和难以扩展，而且还不能动态地改变算法。最后你得到一堆相关的类 , 它们之间的唯一差别是它们所使用的算法或行为。 将算法封装在独立的Strategy类中使得你可以独立于其Context改变它，使它易于切换、易于理解、易于扩展。
      3. 消除了一些if else条件语句 ：Strategy模式提供了用条件语句选择所需的行为以外的另一种选择。当不同的行为堆砌在一个类中时 ,很难避免使用条件语句来选择合适的行为。将行为封装在一个个独立的Strategy类中消除了这些条件语句。含有许多条件语句的代码通常意味着需要使用Strategy模式。
      4. 实现的选择 Strategy模式可以提供相同行为的不同实现。客户可以根据不同时间 /空间权衡取舍要求从不同策略中进行选择。
   2. **Strategy模式缺点**:
      1. 客户端必须知道所有的策略类，并自行决定使用哪一个策略类:  本模式有一个潜在的缺点，就是一个客户要选择一个合适的Strategy就必须知道这些Strategy到底有何不同。此时可能不得不向客户暴露具体的实现问题。因此仅当这些不同行为变体与客户相关的行为时 , 才需要使用Strategy模式。
      2. Strategy和Context之间的通信开销 ：无论各个ConcreteStrategy实现的算法是简单还是复杂, 它们都共享Strategy定义的接口。因此很可能某些 ConcreteStrategy不会都用到所有通过这个接口传递给它们的信息；简单的 ConcreteStrategy可能不使用其中的任何信息！这就意味着有时Context会创建和初始化一些永远不会用到的参数。如果存在这样问题 , 那么将需要在Strategy和Context之间更进行紧密的耦合。
      3. 策略模式将造成产生很多策略类：可以通过使用享元模式在一定程度上减少对象的数量。 增加了对象的数目 Strategy增加了一个应用中的对象的数目。有时你可以将 Strategy实现为可供各Context共享的无状态的对象来减少这一开销。任何其余的状态都由 Context维护。Context在每一次对Strategy对象的请求中都将这个状态传递过去。共享的 Strategy不应在各次调用之间维护状态。
2. 详见http://blog.csdn.net/hguisu/article/details/7558249