考虑这样一个场景：根据传入的起始地点和结束地点，分别计算：快车、专车、站点拼车、豪华车、顺风车…的价格？

问题抽象：根据距离和选择的车的类型输出价格。

方案一：

1. 定义不同类型的车的枚举；

2. 根据传入的车子类型和距离参数计算价格；

3. 通过函数调用计算价格。

特点：

1. 简单

2. 不便于扩展

3. 不遵循“开闭原则”（当软件需要变化时，尽量通过扩展软件实体的行为来实现变化，而不是通过修改已有的代码来实现变化）

优化思路：

1. 都是根据距离参数计算价格，可以抽离出一个计算接口，负责计算部分；

2. 具体的计算算法都实现计算接口，并自行维护自己的计算细节。

3. 实现一个计算管理类，它持有一个具体的计算算法，并提供计算方法；

4. 使用方只需要持有计算管理类，并给它设置不用的计算算法，就可以通过计算管理类得到相应算法对应的结果。

**策略模式(Strategy Pattern)**：定义一系列算法，将每一个算法封装起来，并让它们可以相互替换。策略模式让算法独立于使用它的客户而变化，也称为政策模式(Policy)。

策略模式是一种对象行为型模式。

策略模式是对算法的封装，它把算法的责任和算法本身分割开，委派给不同的对象管理。策略模式通常把一个系列的算法封装到一系列的策略类里面，作为一个抽象策略类的子类。用一句话来说，就是“准备一组算法，并将每一个算法封装起来，使得它们可以互换”。

解决的问题：根据外部条件选择不同策略来解决不同问题。

**优点**

策略类之间可以自由切换

由于策略类都实现同一个接口，所以使它们之间可以自由切换。

易于扩展

增加一个新的策略只需要添加一个具体的策略类即可，基本不需要改变原有的代码，符合“开闭原则“

避免使用多重条件选择语句（if else），充分体现面向对象设计思想。

**缺点：**

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。

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。

**应用场景：**

一个系统需要动态地在几种算法中选择一种的情况

如果在一个系统里面有许多类，它们之间的区别仅在于它们的行为，那么使用策略模式可以动态地让一个对象在许多行为中选择一种行为

如果一个对象有很多的行为，如果不使用合适的模式，这些行为就只好使用多重的if-else语句来实现，此时，可以使用策略模式，把这些行为转移到相应的具体策略类里面，就可以避免使用难以维护的多重条件选择语句，并体现面向对象涉及的概念。

不希望客户端知道复杂的、与算法相关的数据结构，在具体策略类中封装算法和相关的数据结构，提高算法的保密性与安全性。