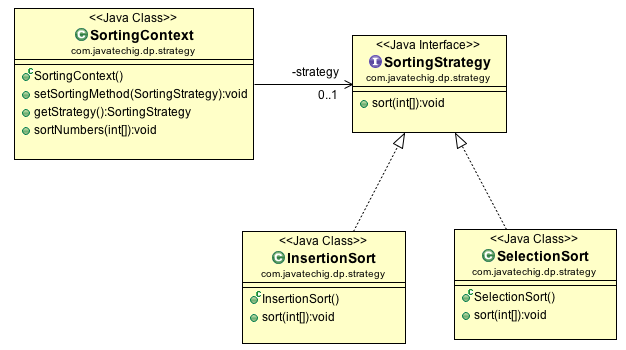
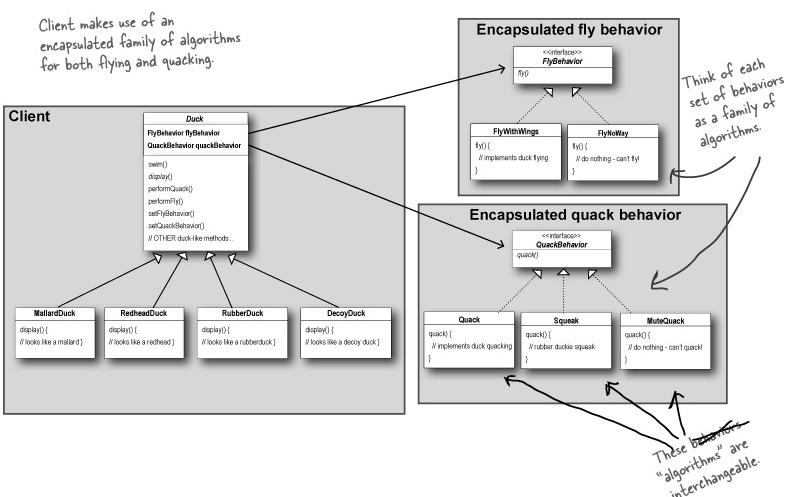
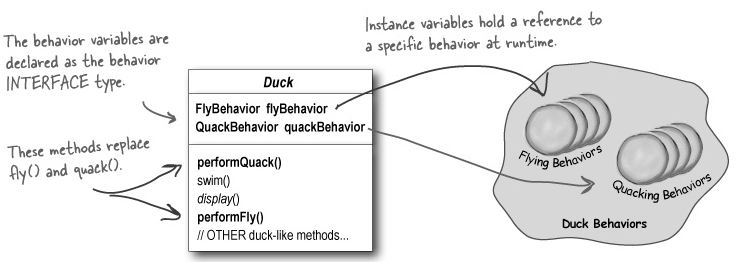
**Strategy –**

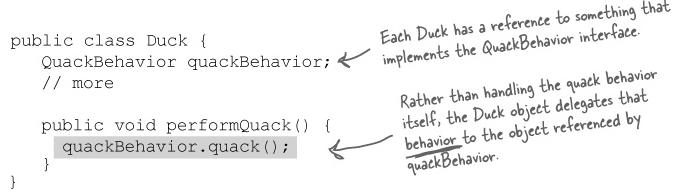


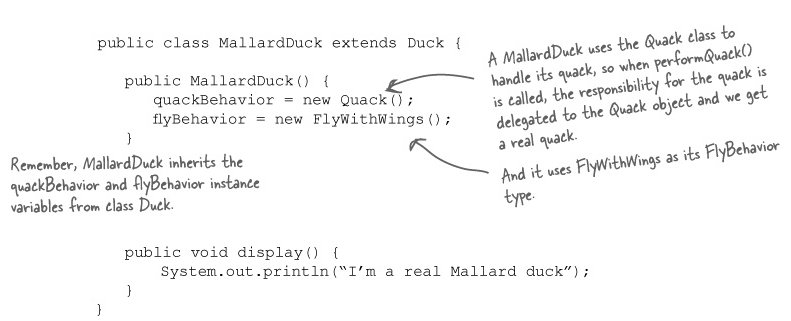
**Example 1 – Duck simulator from headfirst book**

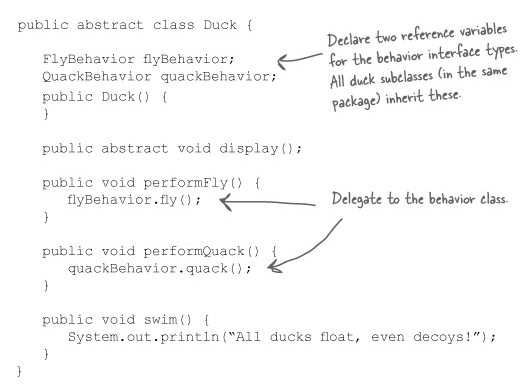
**Strategy defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from the client that uses it.**

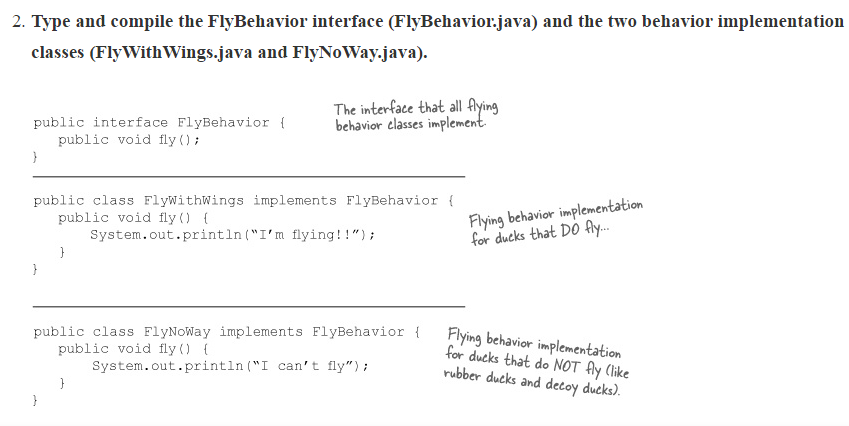


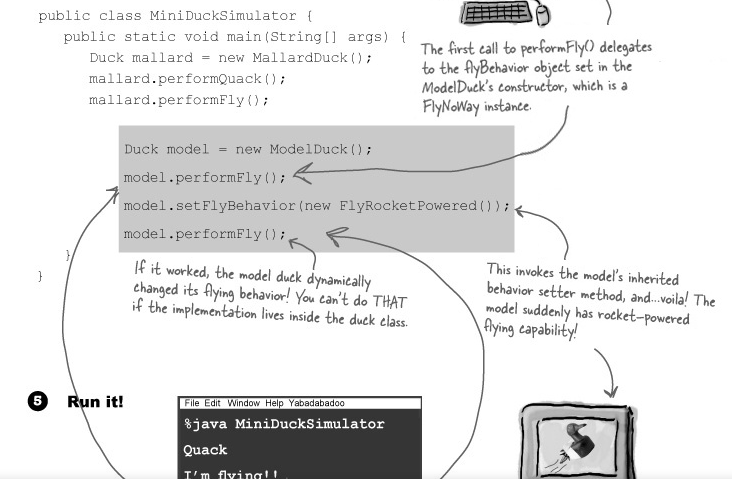












Practical example –

I have implemented in my project. I had a scenario where there are many different persons which were to be inserted/saved based on different algorithms.

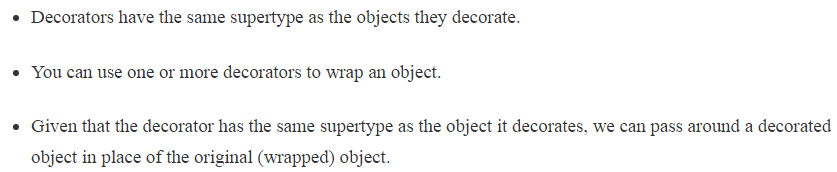
1. PI – to be saved in contacts as well as PI table.
2. Drug Person – to be saved in contacts as well as drug table.
3. Visitors – to be saved in contacts only.

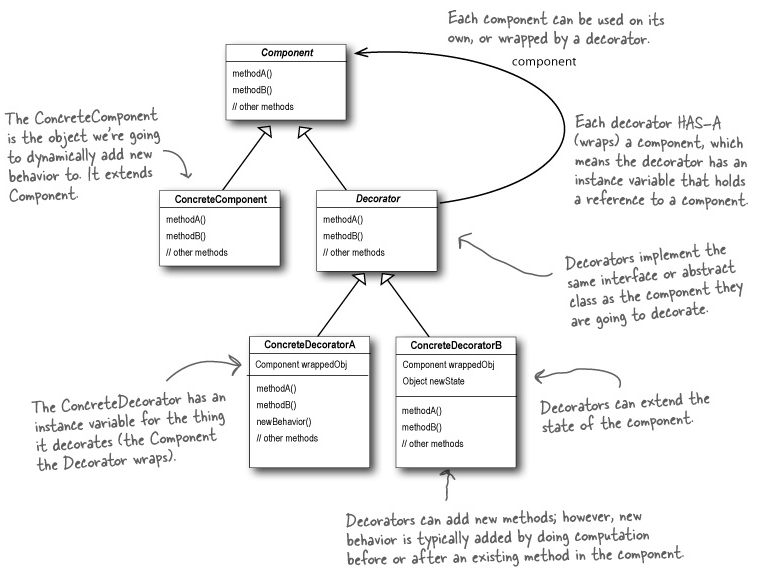
I have created a BusinessPerson interface and let this concrete classes implemented this. From service layer, I dynamically created different persons based on **FactoryPattern** and saved the person accordingly. BusinessPerson had abstract save () method and this method is implemented in different concrete person classes.

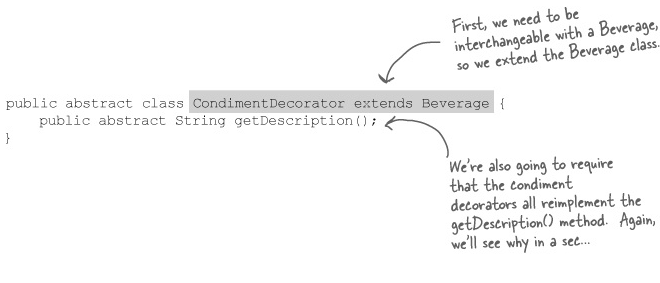
<http://stackoverflow.com/questions/370258/real-world-example-of-the-strategy-pattern>

**Decorator –**



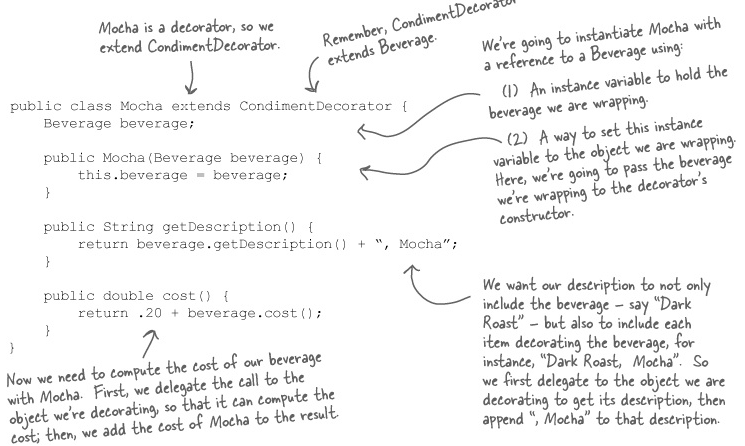


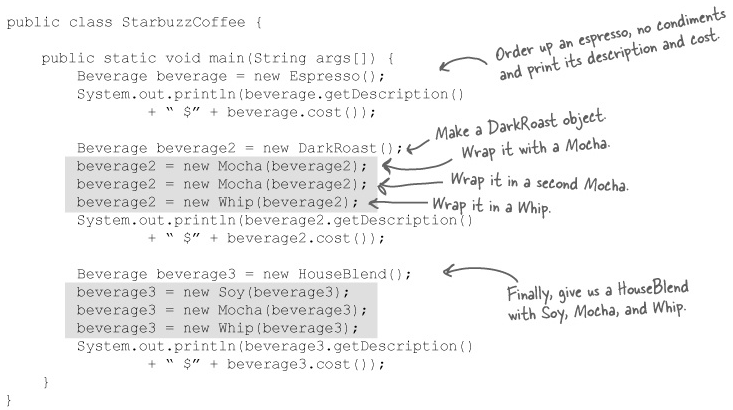




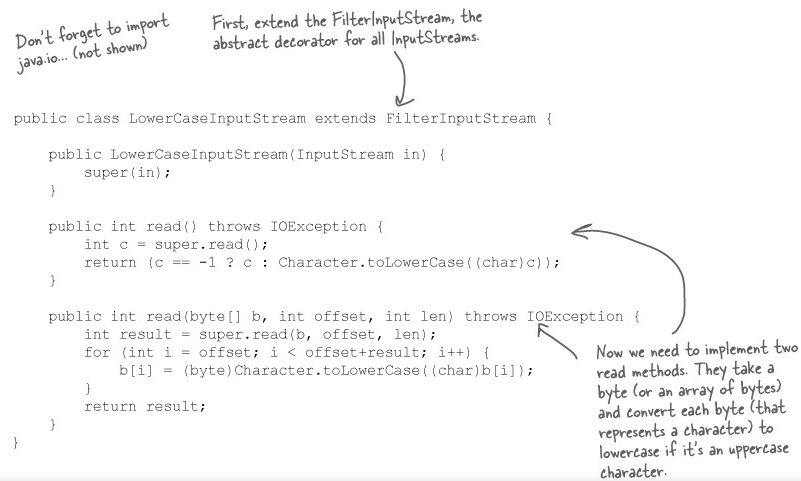
OOP concept – Open closed principal

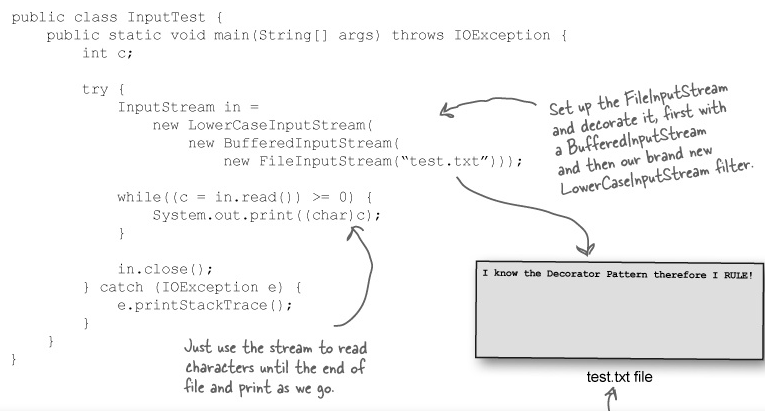
Classes should be open for extension but closed for modification. In this design pattern, we are extending the functionality of existing classes but not changing the functionality.



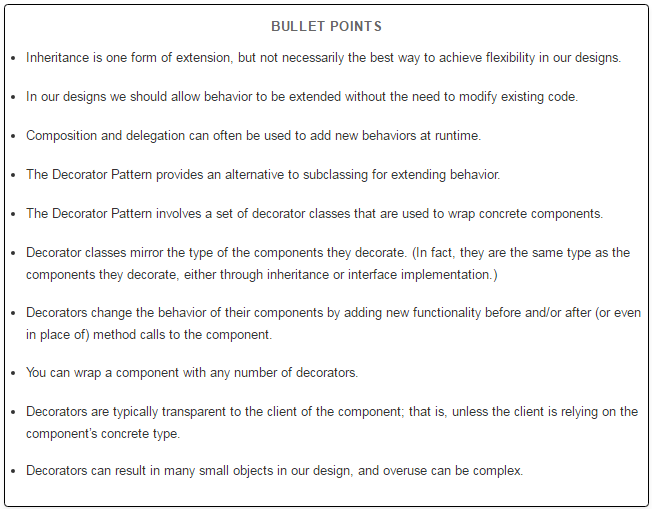


We can create our own user defined decorator





Finally notes for the design patterns in the below page



**State –**

