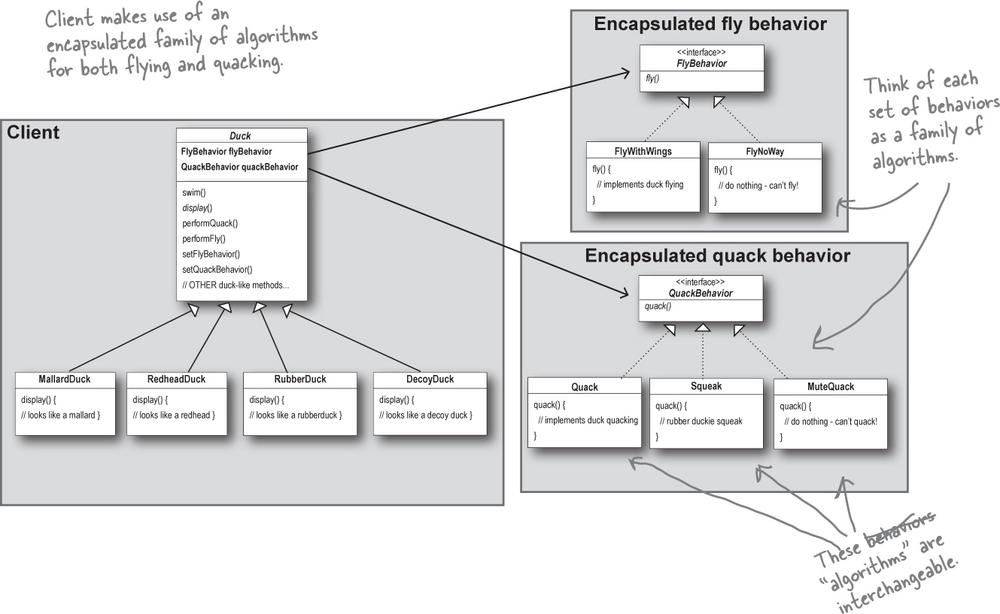
**Strategy Pattern**

The Strategy Pattern deﬁnes a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.



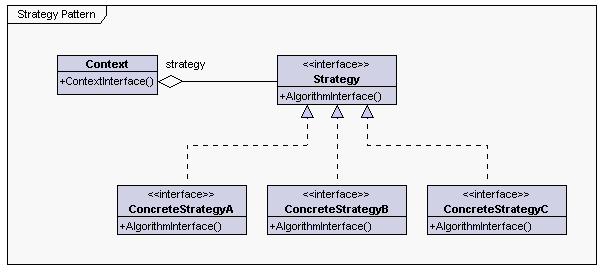
**Design Principle**: identify the aspects of your application that vary and separate them from what stays the same.

**HAS-A can be better than IS-A**

HAS-A can be better than IS-A: The HAS-A relationship is an interesting one: each duck has a *FlyBehavior* and a *QuackBehavior* to which it delegates flying and quacking.

When you put two classes together like this, you’re using ***composition***. Instead of inheriting their behaviour, the ducks get their behaviour by being *composed* with the right behaviour object.

**Design Principle**: Favour composition over inheritance.



Let us try to understand each component of this class diagram.

* **Strategy**: This is the interface common to all algorithms. Context uses this interface to perform the operations.
* **ConcreteStrategy**: This is the class that implements the actual algorithm.
* **Context**: This is the client application that performs the decision making for which strategy should be used and uses the Strategy interface (which is referring to a ConcreteStrategy object) to perform the operations.