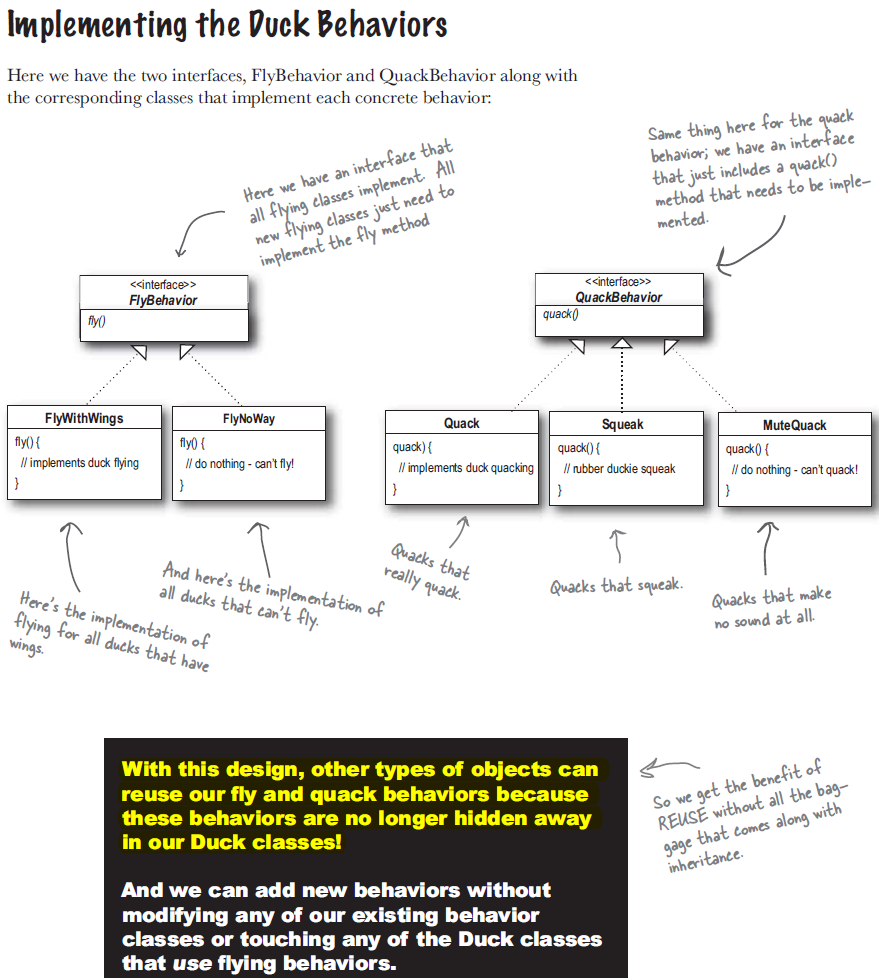
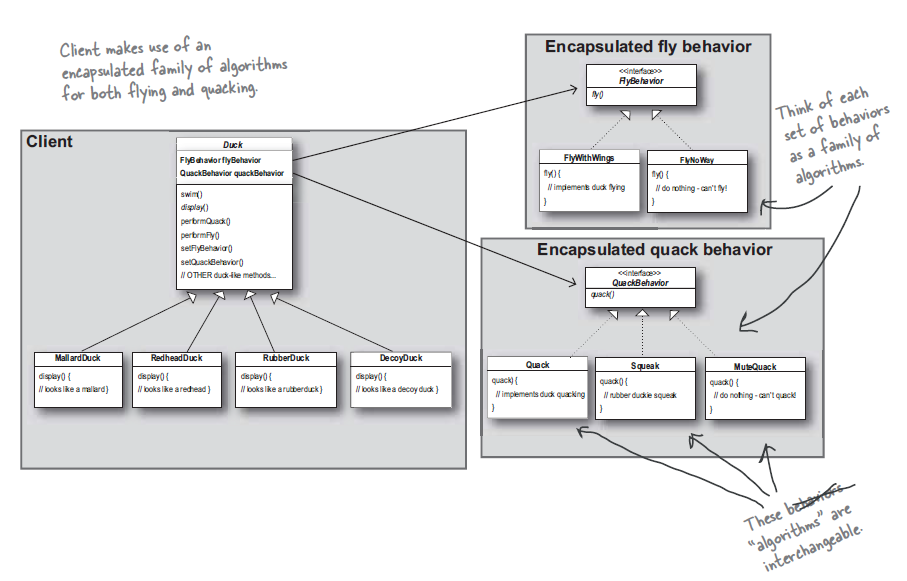
**Strategy pattern:**

1. **Definition**: Strategy defines a family of algorithms encapsulates each of them and make them interchangeable. Strategy let the algorithm vary independent of the client uses them.
2. **Inheritance problems**:
   1. With inheritance a localized change in superclass would be reflected on all subclasses. Now if want to add new child with new behavior that means at first adding it to superclass then overriding it in child classes properly.
   2. Now even separating the behavior in interface won’t be the best idea. Since it would destroy the code reusability provided by superclass, also adds maintenance problems.
3. **Design Principle**: Identify the aspect of your code that doesn’t vary and separate that from what stays the same.
4. **Design Principle**: Program to interfaces not to implementation.
   1. While using inheritance where a behavior either comes from superclass or implemented in specialize child class. In both cases we’re relying on implementation that we can’t change.
   2. So, in summary changing behaviors should stay in different classes that implement a behavior interface.
   3. Program to interface means program to super types interface, abstract class to take advantage of polymorphism.
   4. With inheritance out development time may be less but there would be much worse maintenance impact.





1. **Design Principle**: Favor composition over inheritance.
   1. Composition not only let us encapsulate family of algorithm in their own classes but also let us change the behavior at runtime.

