The Flying Fish Solution
Using the Strategy Pattern

Animal

FlyBehavior fb;
SwimBehavior sb;

+Eat()

<<interface>>
FlyBehavior

+Fly()

FlyFast

+Fly()

FlyNoWay

+Fly()

Bird

+MethodCommonToBirds()

Fish

+MethodCommonToFish()

<<interface>>
SwimBehavior

+Swim()

Robin

fb = FlyFast()
sb = SwimNoWay()

Woodpecker

fb = FlyFast()
sb = SwimNoWay()

Trout

fb = FlyNoWay()
sb = SwimWell()

FlyingFish

fb = FlyFast()
sb = SwimWell()

SwimWell

+Swim()

SwimNoWay

+Swim()
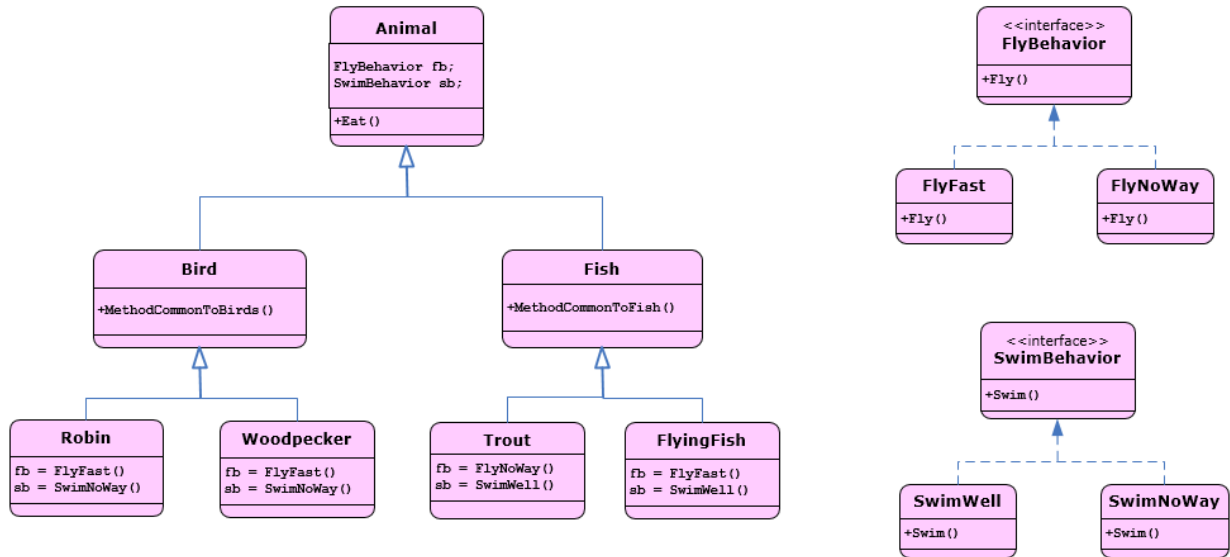
Benefits
- Favor object composition over inheritance
- Forces developer to make a conscious decision about behavior when coding it (as opposed to creating default implementations
  in the base class that the developer must remember to override).
- By abstracting the varying behavior, it is now available for use by other clients.
- Being more loosely coupled means it's easier to adhere to the open/closed principle, where code should be open for extension
  but closed for modification.

The varying behavior (flying and swimming) is now encapsulated and can vary independently from other parts of the system.