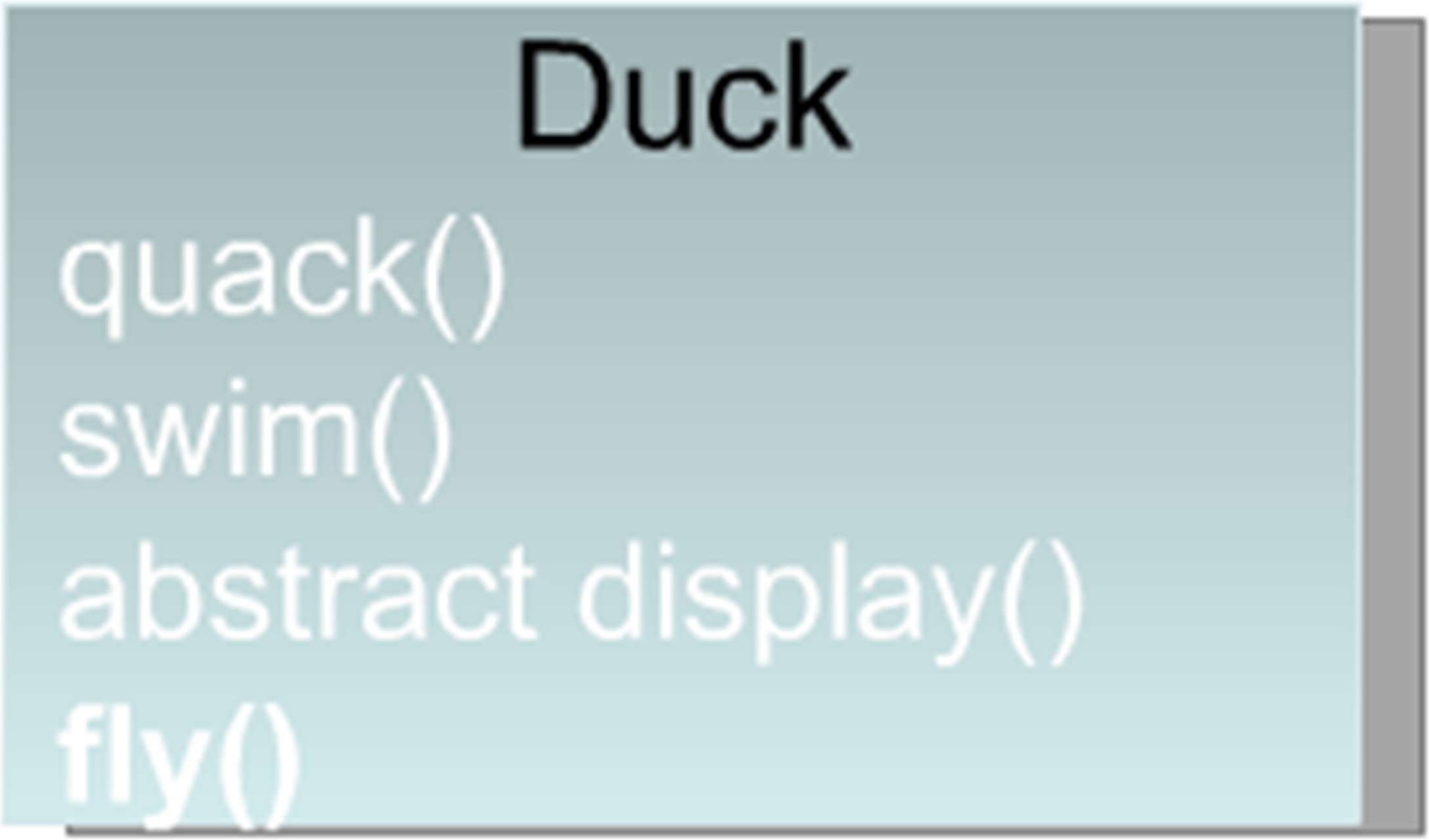
GoF Design Patterns: Startegy

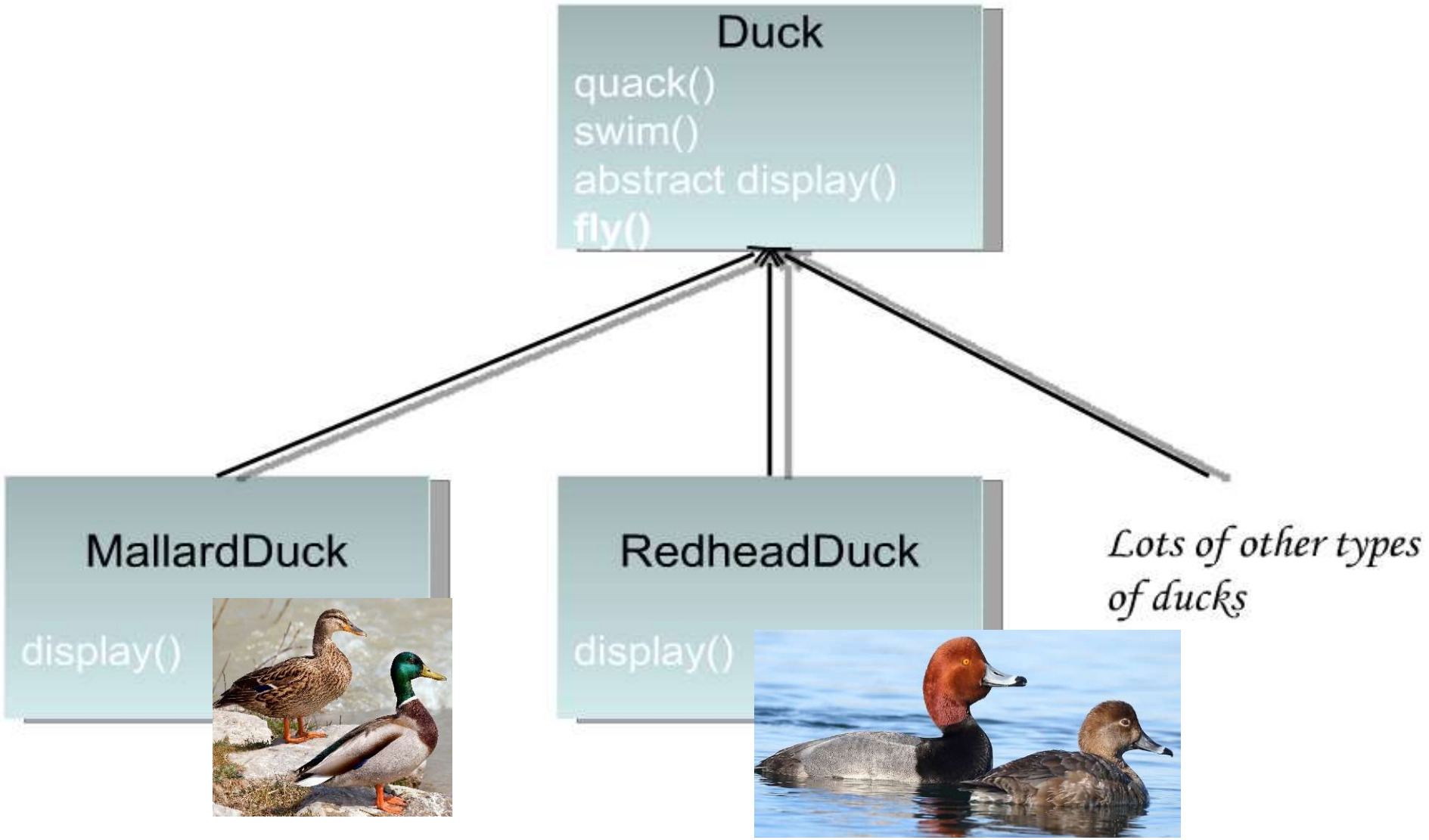
Laura Semini, Ingegneria del Software

Dipartimento di Informatica, Università di Pisa

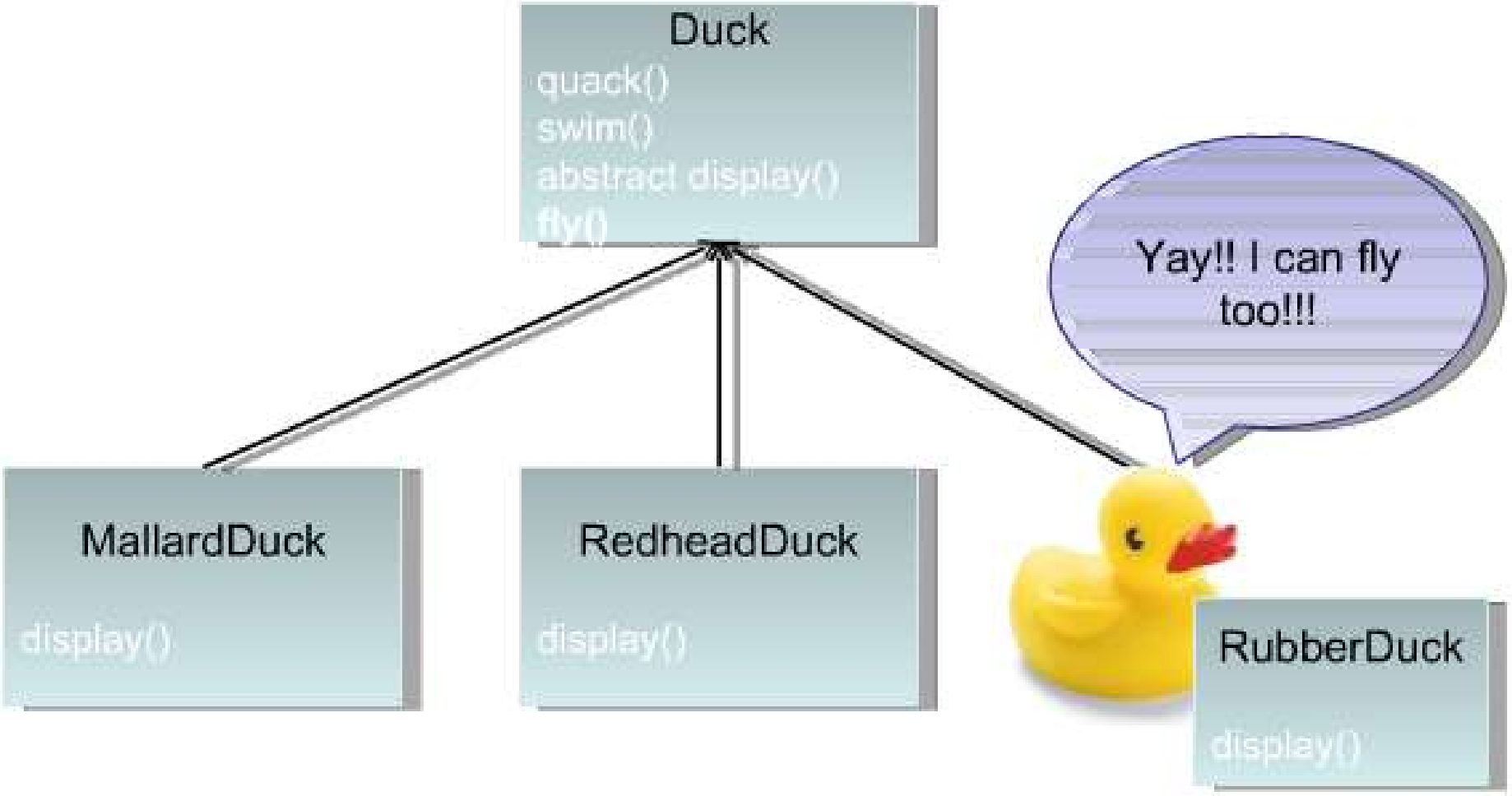
Strategy pattern: the duck



Strategy pattern: the duck



The rubber duck



First solution: override

Override fly()

```
Class Rubberduck{
    fly() {
    \\ do nothing
    }
    quack(){
    \\ override to squeak
    }
}
```

PROBLEM:
subclassing when only part of the behaviour is inherited

All time a new duck is added, the designer has to check if methods fly and quack have to be overiden

Second solution: interfaces

<<Interfaces>> !!!

Yes!!! That's it! I make an IFlyable interface and RubberDuck doesn't get to implement it…

MallardDuck
<<IFlyable>>
display()

RedheadDuck
<<IFlyable>>
display()

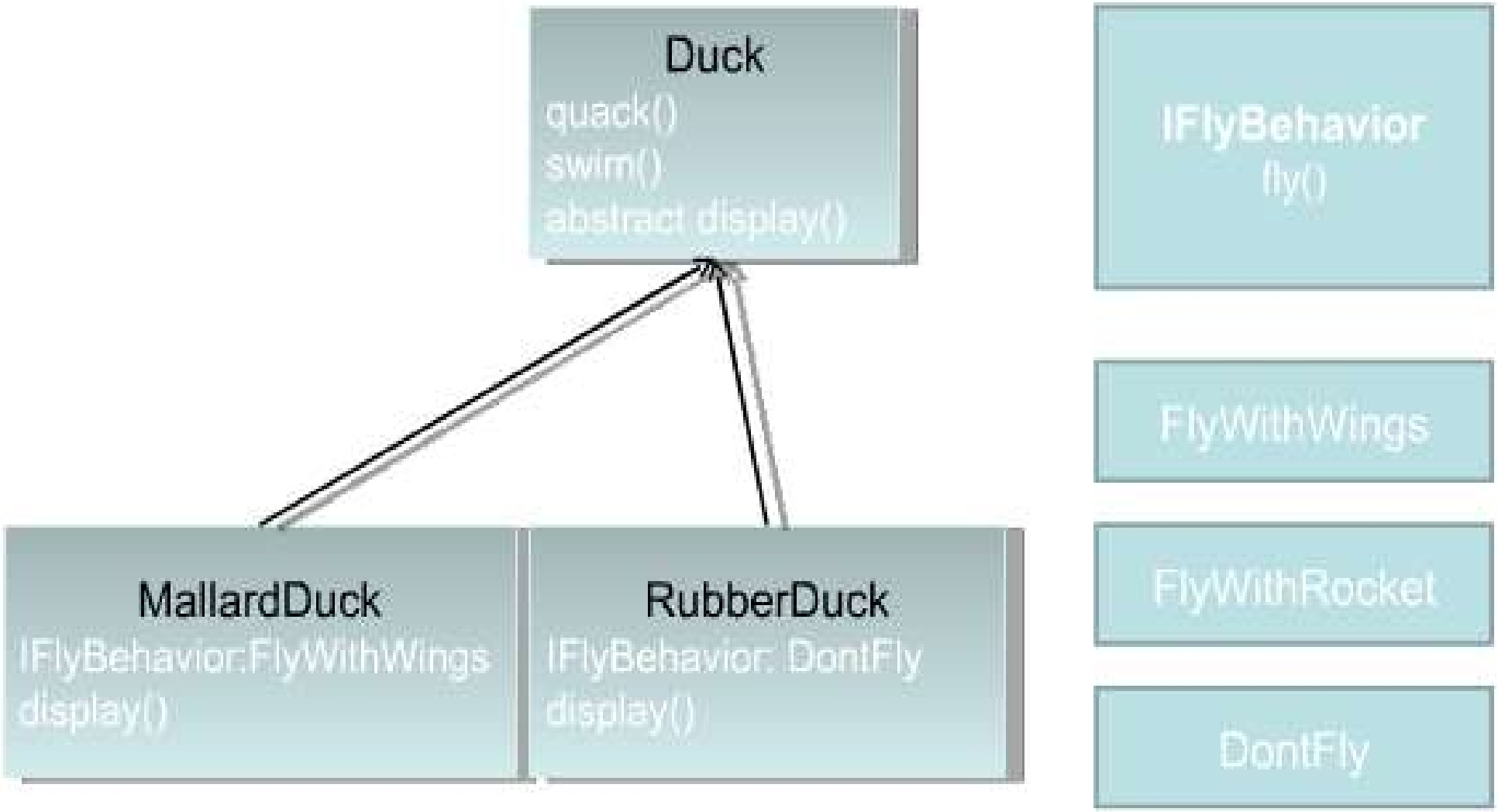
I… I… I…
I have a Question!
SimCorp simulates 50 ducks… are you saying you are going to write 50 fly methods? What if there is a change in flying style and it effects 20 ducks… will all 20 ducks change?

I am losing money you know…

I thought she was non-technical…

- Well here is my situation
- I can't put the fly() method in the base class
- If I use interface, I can't reuse code
- Alright, so this calls for a dependency split
- Flying is a behavior and should be separate from the Duck object
- Flying behaviors could be reused on different objects
- Different ducks could fly in different ways

Strategy


Duck
quack()
swim()
abstract display()
IFlyBehavior
fly()
FlyWithWings
FlyWithRocket
DontFly
MallardDuck
IFlyBehavior:FlyWithWings
display()
RubberDuck
IFlyBehavior: DontFly
display()

Recall some OO design principles

“Identify what vary and encapsulate them, so that later you can alter or extend the parts that vary without affecting those that don’t”;

“Program to an interface, not an implementation”;

“Favor composition over inheritance”.

Strategy

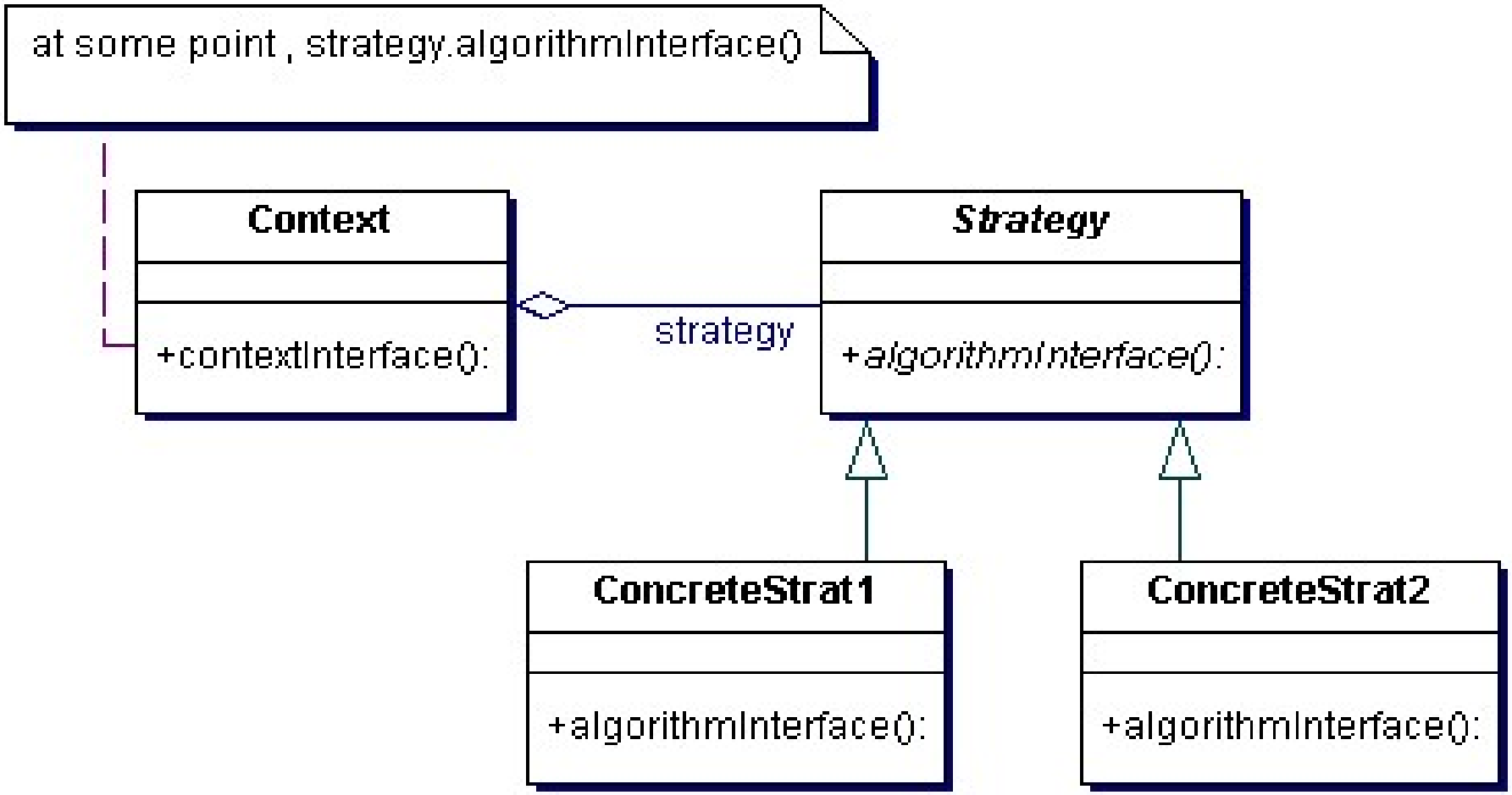
Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

A program may have to supply several variations of an algorithm or of a behaviour.

Solution:

- These variations are encapsulated in separate classes
- There is a uniform access to them

Strategy: structure



Strategy: participants

Strategy

- Defines an interface common to all supported algorithms

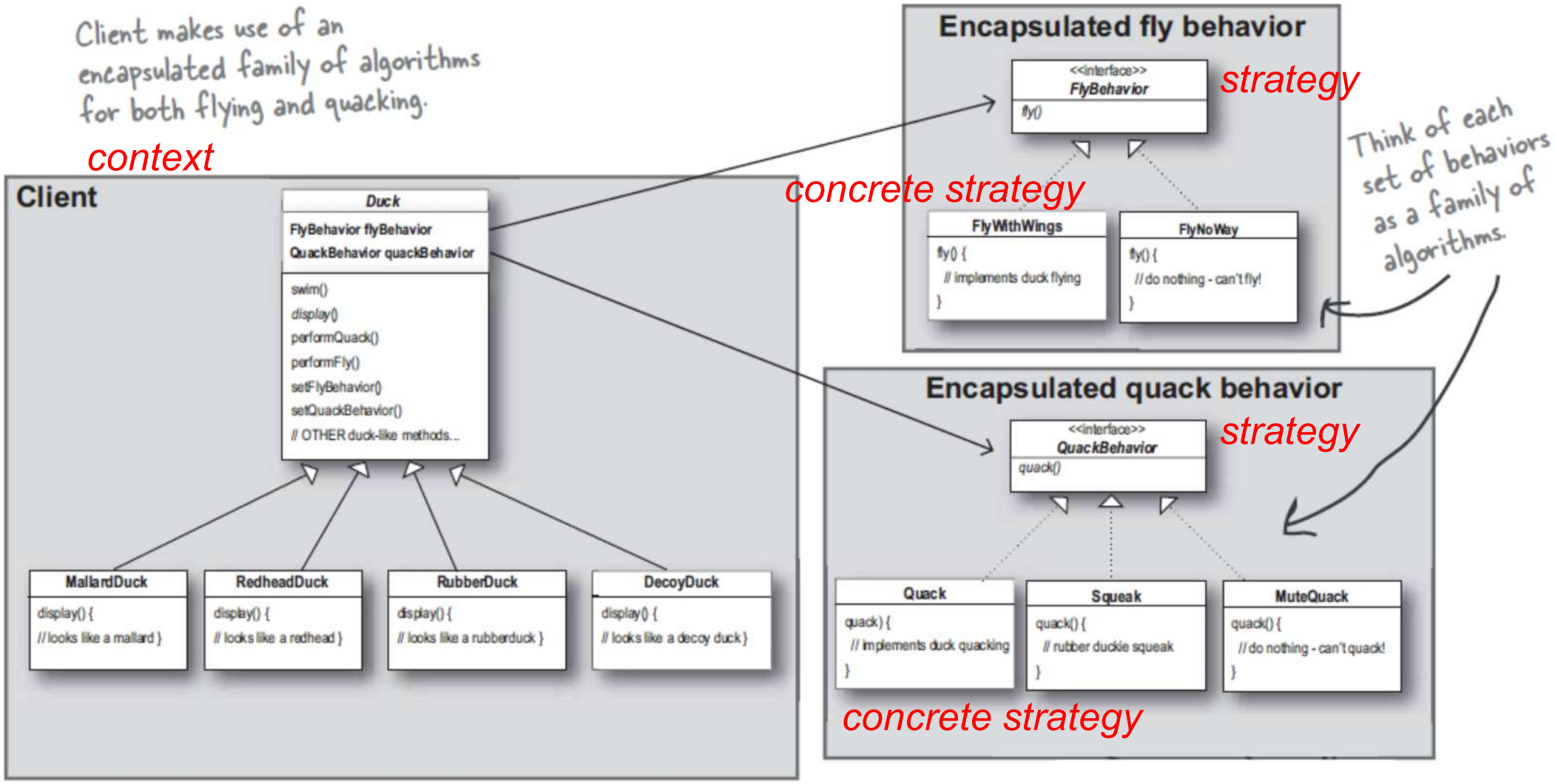
ConcreteStrategy

- Each concrete strategy implements an algorithm

Context

- Contains a reference to a strategy object (with type Strategy)
- May define an interface that lets strategy access its data. (Instead of passing them as arguments when calling the strategy methods)

Ducks example



Example

The MyArray class represents vectors of numbers

One of its methods print the array, in two formats:

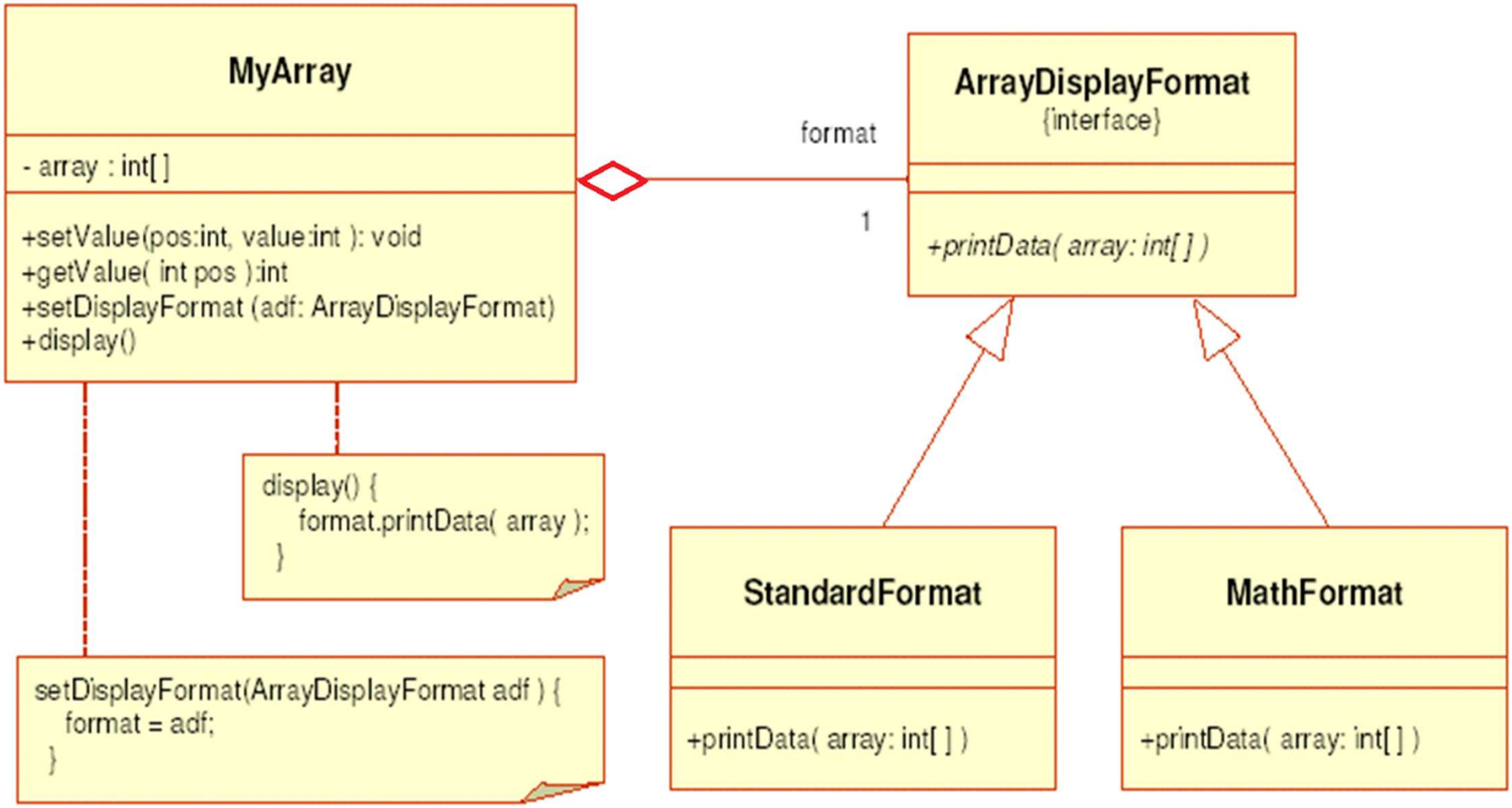
- MathFormat (es. {12, -7, 3, ...})
- StandardFormat (es. ar[0]_12, ar[1]_-7, ar[2]_3, …)

In the future these formats may be substituted by different ones….

Problem:

- How to isolate the algorithm used to format the array contents, so that it can vary independently of the other methods of the class?

Solution


MyArray
- array : int[ ]
+setValue(pos:int, value:int): void
+getValue(int pos):int
+setDisplayFormat (adf: ArrayDisplayFormat)
+display()
format
1
ArrayDisplayFormat
{interface}
+printData(array: int[ ])
display() {
format.printData(array);
}
setDisplayFormat(ArrayDisplayFormat adf) {
format = adf;
}
StandardFormat
+printData(array: int[ ])
MathFormat
+printData(array: int[ ])

Context

```
public class MyArray {

   private int[] array;
   private int size;
   ArrayDisplayFormat format;

   public MyArray( int size ) {
      array = new int[ size ];
   }

   public void setValue( int pos, int value ) {
      array[pos] = value;
   }

   public int getValue( int pos ) {
      return array[pos];
   }

   public int getLength( int pos ) {
      return array.length;
   }

   public void setDisplayFormat( ArrayDisplayFormat adf ) {
      format = adf;
   }

   public void display() {
      format.printData( array );
   }

}
```

The interface (strategy)

```
public interface ArrayDisplayFormat {

   public void printData( int[] arr );

}
```

First concrete startegy

```
public class StandardFormat implements ArrayDisplayFormat {

   public void printData( int[] arr ) {
      System.out.print( "( " );
      for(int i=0; i < arr.length-1 ; i++ )
         System.out.print( arr[i] + ", " );
      System.out.println( arr[arr.length-1] + " )" );

   }
}
```

Second concrete startegy

```
public class MathFormat implements ArrayDisplayFormat {

   public void printData( int[] arr ) {
      for(int i=0; i < arr.length ; i++ )
         System.out.println( "Arr[ " + i + " ] = " + arr[i] );
   }
}
```

The client

```
public class StrategyExample {

 public static void main (String[] arg) {

   MyArray m = new MyArray( 10 );
   m.setValue( 1 , 6 );
   m.setValue( 0 , 8 );
   m.setValue( 4 , 1 );
   m.setValue( 9 , 7 );
   System.out.println("This is the array in 'standard' format");
   m.setDisplayFormat( new StandardFormat() );
   m.display();
   System.out.println("This is the array in 'math' format:");
   m.setDisplayFormat( new MathFormat() );
   m.display();
 }

}
```

The result

```
C: \Design Patterns\Behavioral\Strategy>java StrategyExample

This is the array in 'standard' format :
{ 8, 6, 0, 0, 1, 0, 0, 0, 0, 7 }

This is the array in 'math' format:
Arr[ 0 ] = 8
Arr[ 1 ] = 6
Arr[ 2 ] = 0
Arr[ 3 ] = 0
Arr[ 4 ] = 1
Arr[ 5 ] = 0
Arr[ 6 ] = 0
Arr[ 7 ] = 0
Arr[ 8 ] = 0
Arr[ 9 ] = 7
```

Applicability

Use the Strategy pattern whenever:

- Many related classes differ only in their behavior
- You need different variants of an algorithm
- An algorithm uses data that clients shouldn't know about.
- Use the Strategy pattern to avoid exposing complex, algorithm-specific data structures.
- A class defines many behaviors, and these appear as multiple conditional statements in its operations. Instead of many conditionals, move related conditional branches into their own Strategy class.

Discussion

Benefits

- Provides an alternative to subclassing the Context class to get a variety of algorithms or behaviors
- Eliminates large conditional statements
- Provides a choice of implementations for the same behavior

Liabilities

- Increases the number of objects
- All algorithms must use the same Strategy interface

Discussion (cont'd)

Different ConcreteStrategy may need different data.

Most probably some ConcreteStrategy will not use all the data passed through the generic interface

- Hence: the context create and initializes parameters that will never be used by anybody
- When this is a problem:
  - stronger coupling between ConcreteStrategy and Context
  - the former accessing the latter to ask for the data it needs

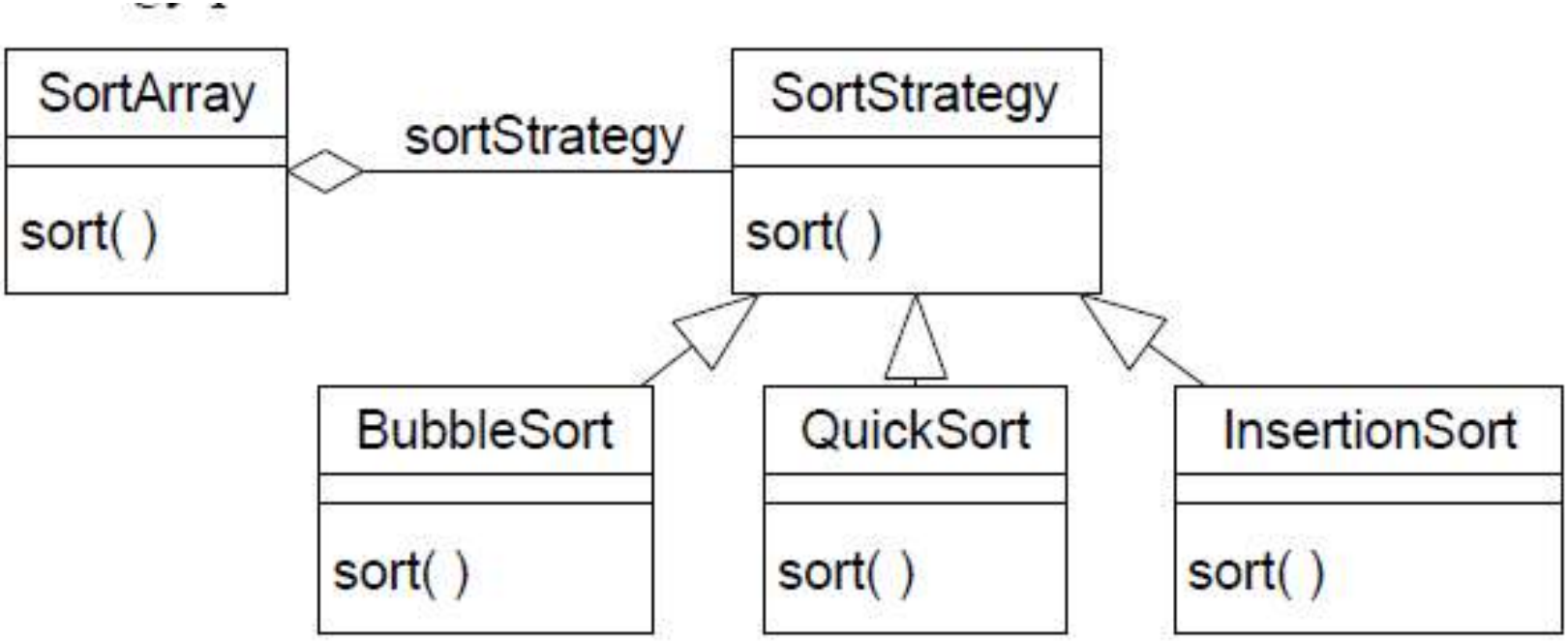
Strategy Pattern Example: SORT

Problem:

- A class wants to decide at run-time what algorithm it should use to sort an array. Many different sort algorithms are already available.

Solution

- Encapsulate the different sort algorithms using the Strategy pattern



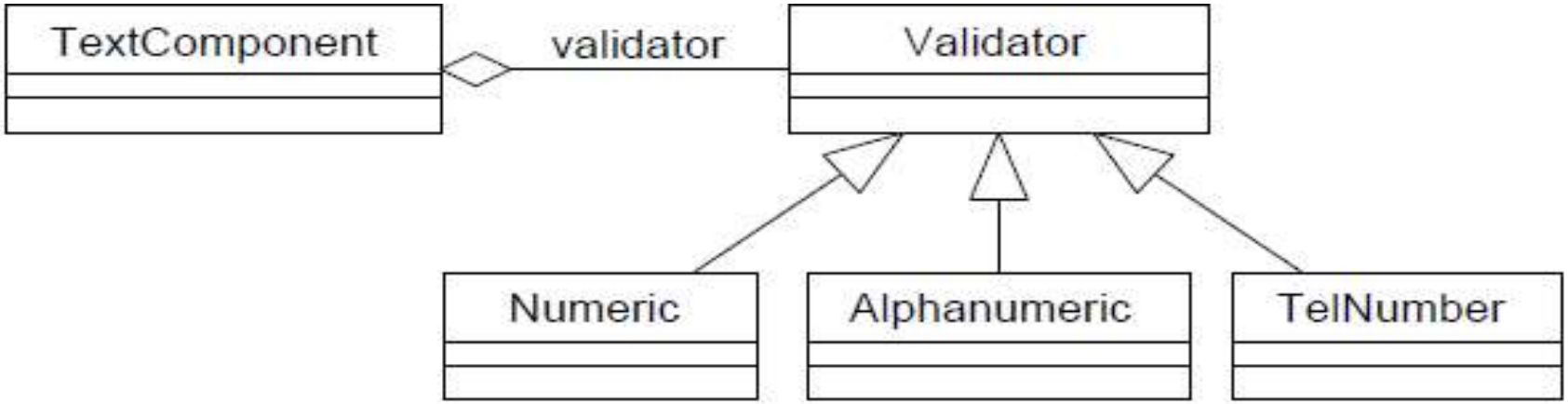
Strategy Pattern Example: GUI

A GUI text component object wants to decide at runtime what strategy it should use to validate user input. Many different validation strategies are possible: numeric fields, alphanumeric fields, telephone-number fields, etc.

Solution

- Encapsulate the different input validation strategies using the Strategy pattern

This is the technique used by the Java Swing GUI text components. Every text component has a reference to a document model which provides the required user input validation strategy.



Homework: design and code

PizzaCap is a company that sells pizzas and offres discounts to clients.

> There are many kinds of discount calculation methods such as: 10% off, reduce 5$ each 20$ spent, 3 pizzas at the price of 2, 20% discount on pepperoni pizza.

Now PizzaCap asks **you to develop a sales management system**, they want you to design a schema to calculate the discount when selling pizzas. Your design should be capable of selecting the discount calculation methods. Also, when PizzaCap needs new discount calculation methods or wants to modify old ones, it must be very easy to implement the changes without affecting the existing system.

Discuss the issue with the data passed. What happens with "reduce 5$ on any purchase”? or “reduce 5$ on first purchase”?