Design Patterns

Kris.Hermans@pxl.be

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Dep. PXL-IT – Elfde-Liniestraat 26 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

1

It’s about sharing knowledge and experience

2

It will improve your OO skills

3

Language independent.

The book uses Java, but examination will be in C#.

4

It takes a while to grok.

Read, reread, think, code, discuss, reread, rethink, code again, ...

5

Class lectures

- Strategy
- Observer

6

Next: study group

- Strategy
- Observer
- Decorator
- Factory
- Singleton
- Command
- Template Method
- Composite / Iterator
- State

PXL IT

7

For each DP: read and discuss

- Explain the pattern to each other
- Execute and step through the code (in pairs)
- Code the examples in C#
  - https://github.com/jkhines/hfpatternsincsharp
- Where in the curriculum did you use this DP already?
- Search on github for other examples / uses
- Think how it could be applied to the DevOps Case (App). Why (not)?

PXL IT

8

Evaluation (Examination)

- **Define** a pattern in a concise but exact manner
- **Draw** the standard UML of a DP
- **Recognize** a DP from (pseudo)-code
- **Apply** a DP on a new problem
  - Draw a UML applied to the problem
  - Write the code in C#

9

Evaluation (DevOps Case)

- Which DP did you use (at least 2)? Explain the code!
- Did you use DP X? Why (not)?
- Are there other DP you considered not mentioned in the book?

10

Intro to Design Patterns

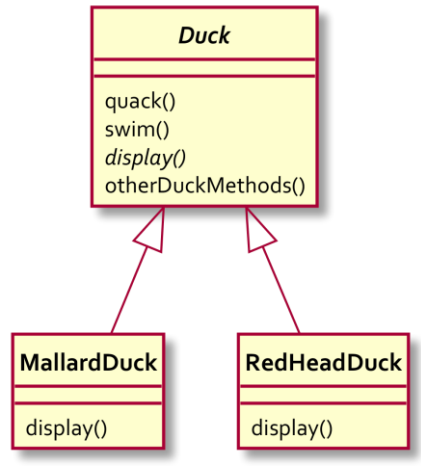
Your first pattern

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Dep. PXL-IT – Elfde-Liniestraat 26 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

11

It started with a simple SimUDuck app

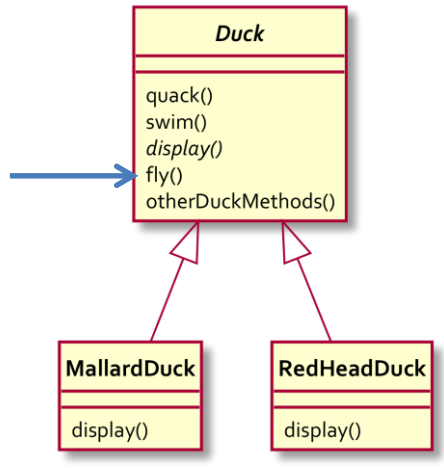


12

New killer feature: flying ducks

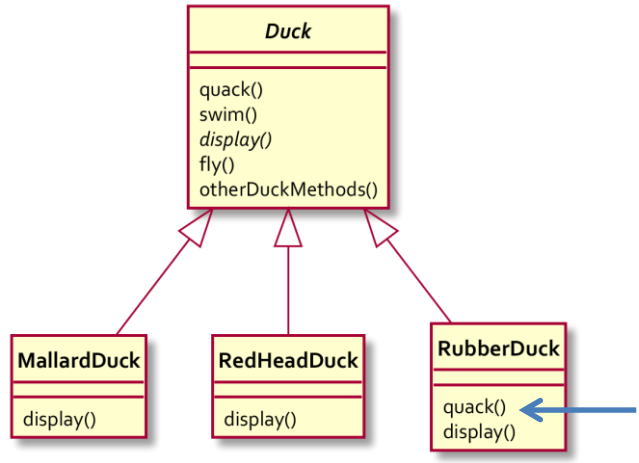
13

Flying Ducks



14

Something went wrong



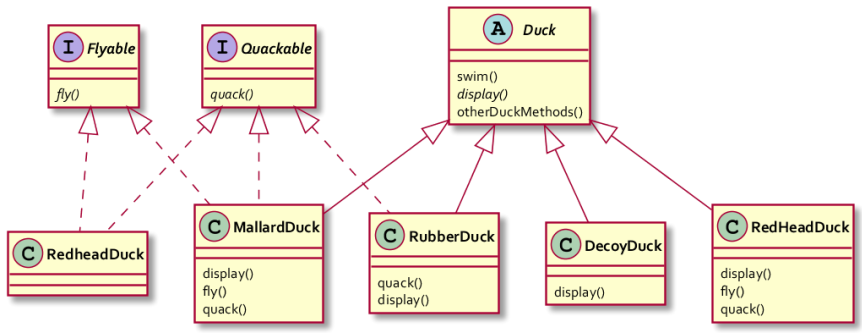
15

RubberDuck

```
quack() { // squeak}

display() { // rubber duck}

fly() { // override to do nothing }
```

16

Let's try interfaces



17

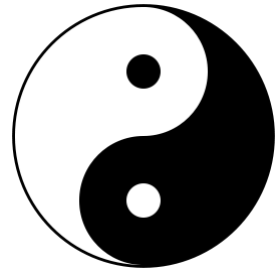
Some observations

- Flyable/Quackable destroy code reuse
- Change is the one constant in software development
- Design Patterns provide a way to let some part of a system vary independently of all other parts

18

Design Principle 1

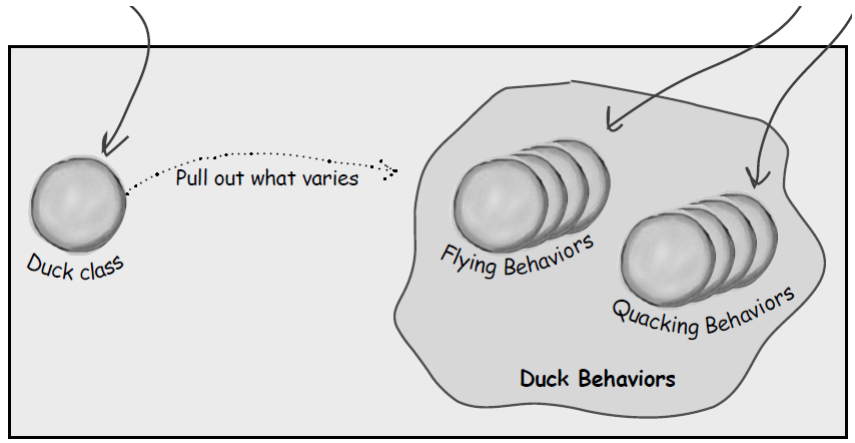
Indentify the aspects of your application that vary and separate them from what stays the same.



PXL IT

19

Applied for Duck problem



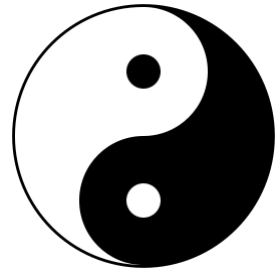
PXL IT

20

Design Principle 2

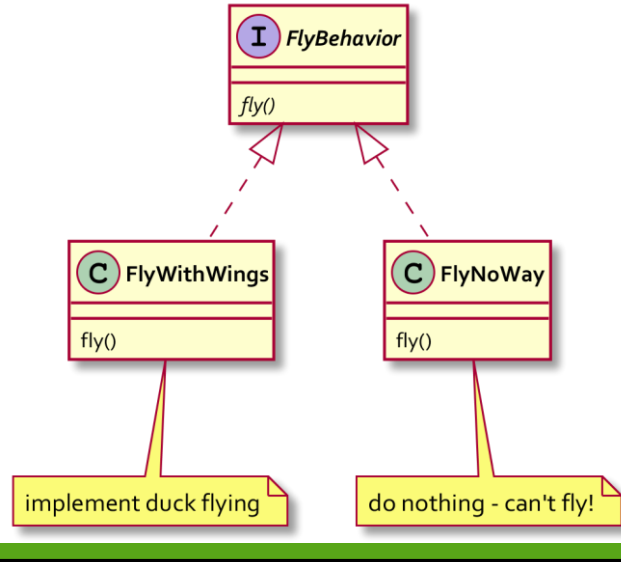
Program to an interface,

Not an implementation.



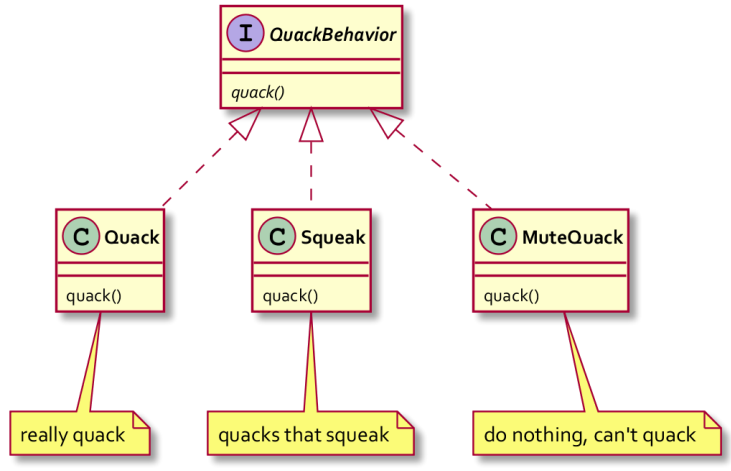
21

Fly behaviors



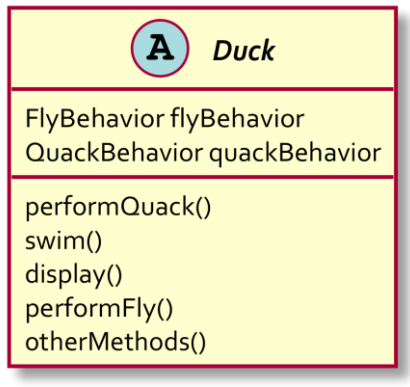
22

Quack behaviors



23

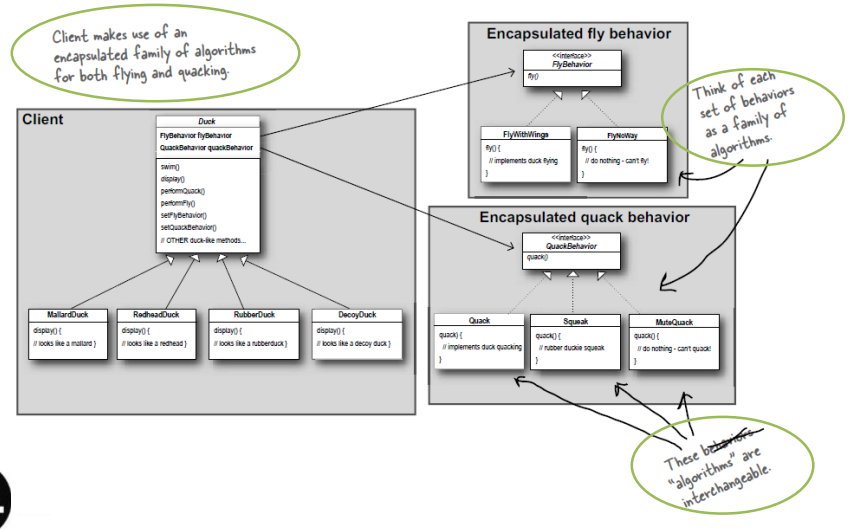
Integrating Duck behavior



- Abstract class
- Instance variables hold reference to behavior at runtime
- performXXX
  - Delegate to behaviors
- Subclasses constructor initialize behaviors

24

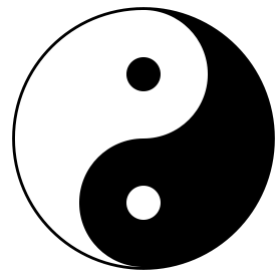
The Big Picture on encapsulated behaviors



26

Design Principle 3

Favor composition over inheritance



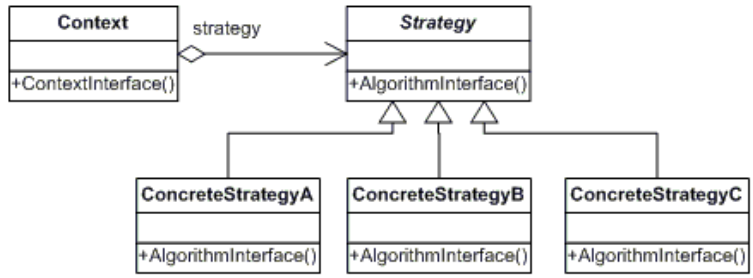
27

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.

28

UML



Bron: http://www.dofactory.com/Patterns/PatternStrategy.aspx

29

Have I seen this before?

```
interface IArtistRepository
{
    IList<Artist> GetAll();
    void Update(Artist artist);
}
```

```
public class ArtistService
{
    private IArtistRepository _artistRepository;

    public ArtistService(IArtistRepository repo)
    {
        _artistRepository = repo;
    }
}
```

30

Observer

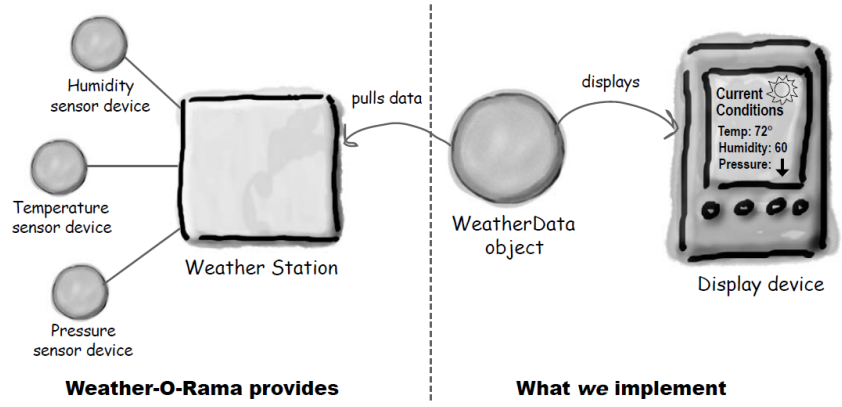
Keeping your objects in the know

DE HOGESCHOOL
MET HET NETWERK

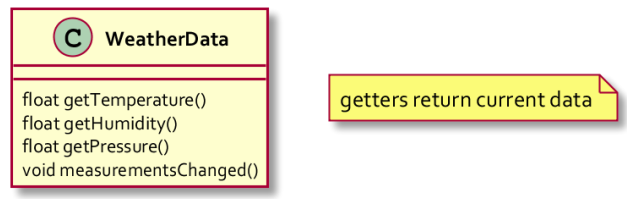
Hogeschool PXL – Dep. PXL-IT – Elfde-Liniestraat 26 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

31

Wheater-O-Rama



WeatherData



measurementsChanged is called by the system, we need to implement it.

Display System

- Current conditions
- Statistics
- Forecast
- Expandable

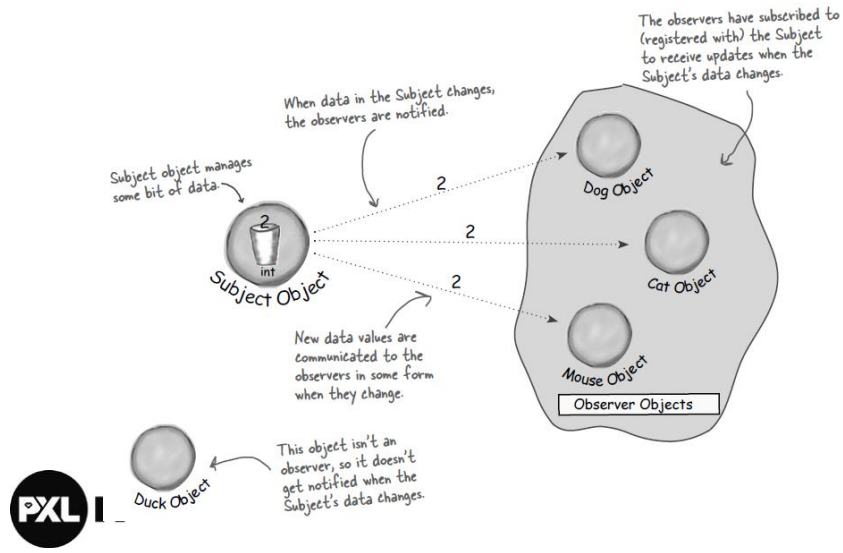
34

First draft

```
public class WeatherData
{
    // instance variable declarations

    public void MeasurementsChanged()
    {
        float temp = GetTemperature();
        float humidity = GetHumidity();
        float pressure = GetPressure();

        currentConditionsDisplay.update(temp, humidity, pressure);
        statisticsDisplay.update(temp, humidity, pressure);
        forecastDisplay.update(temp, humidity, pressure);
    }

    // other methods
}
```

35

Publishers + Subscribers = Observer



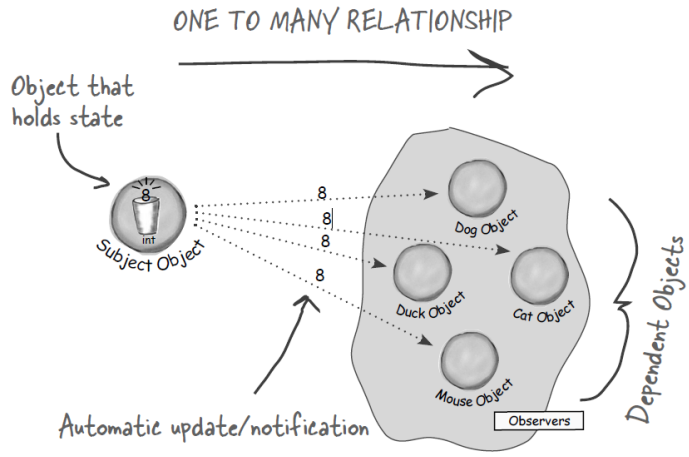
36

The Observer pattern

defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.

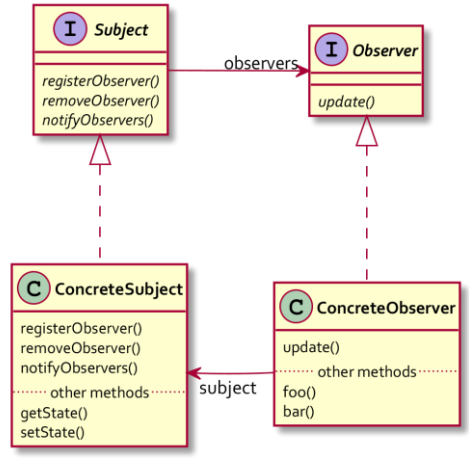
37

The Observer pattern



38

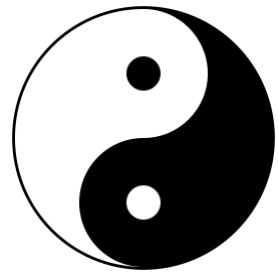
The Observer pattern



39

Design Principle 4

Strive for loosely coupled designs between objects that interact.



PXL IT

40

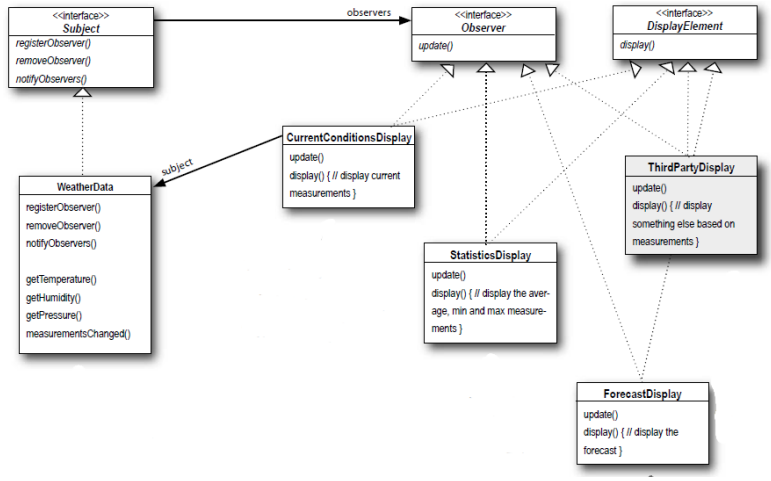
Loose Coupling

- Subject knows that an observer implements an interface, that's all
- Add observers anytime
- Subject unmodified when adding new observers
- Reuse of subjects or observers independently
- Changes in subject does not affect observers and vice versa

PXL IT

41

First Implementation



Have I seen this before?

- Events (C#), Observable (Java)
- Whole new programming paradigm: RX
  - https://github.com/ReactiveX/RxJava
  - https://github.com/Reactive-Extensions/Rx.NET
- …

Language integration

- Java
  - Observer, Observable
  - Push vs Pull style
  - Observable:
    - setChanged: state has changed
    - notifyObservers() or notifyObservers(Object arg)
  - Observer: update(Observable o, Object arg)
  - Critique pp. 71

44

Language integration

- C#
  - Events and delegates
  - http://stackoverflow.com/questions/1249517/super-simple-example-of-c-sharp-observer-observable-with-delegates

45