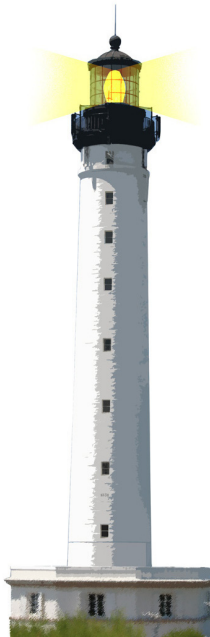
Learning Object-Oriented
Programming and Design with TDD

A different way to model the world

Stéphane Ducasse
http://stephane.ducasse.free.fr



http://www.pharo.org

W3S04

Model of a the World

- There is not one single model of the world
- There are multiple ways to capture a model of the world
  - Data driven (often used with relational database)
  - Object-Oriented Design
  - Procedural modeling
  - Reactive programming
- A program models the world

Object-Orientation

- Is a paradigm, not a technology
- Reflects, simulates the real world
- Organized in terms of decentralized organizations
- Tries to
  - handle complexity
  - enhance reusability
  - minimize maintenance cost

Comparing

- Procedural
- Object-Oriented Design

Structured/Procedural Programming Point of View

- Focuses upon structures and procedures
- Data is shared between procedures
- Data accessible from procedures (and client procedures too)
- Procedures know about the structure of data

Structured/Procedural Programming Point of View

- No late binding (can be simulated with pointer tables)
- Requires large number of procedures and procedure names
- Single namespace for procedures
- No decoupling between messages and methods: just procedures accessing data

Let us Compare

Problem: compute the total area of a set of geometric shapes

```
myPicture := Picture new.
myPicture add: (Square x: 3 y: 3 width: 3).
myPicture add: (Rectangle x: 5 y: with:5 height: 3)
myPicture add: (Circle x: 12 y: 3 radius: 3).

myPicture area
```

Procedural Way: Centralized Way (in Java)

```
double pictureArea() {
  double total = 0;
    for (Shape shape : shapes) {
      switch (shape.kind()) {
      case SQUARE:
        Square square = (Square) shape;
        total += square.width * square.width; break;
      case RECTANGLE:
        Rectangle rectangle = (Rectangle) shape;
        total += rectangle.width * rectangle.height; break;
      case CIRCLE:
        Circle circle = (Circle) shape;
        total += java.lang.Math.PI * circle.radius * circle.radius / 2; break;
       } }
    return total; }
```

Procedural Way: a Centralized Way (in Pharo)

```
pictureArea
  | total |
  total := 0.
  self shapes do [ :aShape |
    aShape kind == #SQUARE
        ifTrue: [ total := total + aShape width * aShape width ]
       ifFalse: [
        aShape kind == #RECTANGLE
          ifTrue: [
            total := aShape width * aShape height ]
          ifFalse: [
            total := total + (Float pi * shape radius squared / 2) ]
  ^ total
```

Procedural Way: Drawbacks

- All the logic is defined in a single place
  - monolithic
- No reuse of the main function pictureArea
- What if we want to add a new shape?
  - need to recompile the area procedure
  - need to check for the new shape

The OO Way: Delegate to Other Entities

```
Picture >> area
  | total |
  total := 0.
  self shapes do [ :aShape |
    total := total + aShape area ].
  ^ total
```

```
Square >> area
 ^ self side squared
```

```
Rectangle >> area
  ^ self width * self height
```

```
Circle >> area
 ^  (Float pi * self radius squared / 2)
```

OOP Advantages

- Adding a new shape
  - add a class with the area message
  - create objects of this class
- Reuse of the Picture >> area
- Reuse of the definition of the shapes
- Decentralised view of computation
- Each shape class represents its data/logic internally

There is a catch

To be able to reuse the code in Picture >> area and add new shape

- It is important that all the shapes can all answer the message area
- Polymorphism: different objects answering the same messages with different execution

What is OOP?

- An application is a collection of interacting entities (objects).
- Objects are characterized by **behavior** and **state**.
- Objects are described by **methods**, **data** are stored in **private** variables.
- Objects communicates by exchanging messages.
- Message passing late bound the selection of the method to be executed in response to messages.
- Ideally everything is an object

Objects

- Unique identity
- Private state
- React to message by selecting method to be executed
- Methods to perform computation
- Objects expose polymorphic interface to be able to be substitued for other objects

OOP Cornerstone: Encapsulation/Composition

Encapsulation

- Hide and control the internal representation of an object. This will ease further evolution
- Clients do not access object internals

Composition

- An object can be composed of several simpler other objects

OOP Cornerstone: Distribution of responsibility

Distribution of responsibility and delegation

- Computing a problem is the results of many objects performing (sub) tasks.

Late binding and message passing

- The receiver of a message determines which method will be executed on it.
  - What to perform? the message
  - How to perform? the method

Polymorphism

- Objects exhibiting the same interface can be substitued
- Class hierarchy defines families of **polymorphic** (kind of substituable) objects

OOP Cornerstone: Reuse via abstraction extension

Inheritance structures abstractions as conceptual hierarchies

- OrderedCollection **is a kind of** Collection
- Array **is a kind of** Collection

Inheritance supports reuse and extensions in subclasses

What you should know

- OOP describes programs as collaborating entities
- Objects encapsulate data and expose API of behavior
- Late binding selects the method to be executed in reaction to a message
- Classes reuse (extend, modify their superclass behavior)
- Good design promote polyrmorphism

A course by Stéphane Ducasse

http://stephane.ducasse.free.fr

Reusing some parts of the Pharo Mooc by

Damien Cassou, Stéphane Ducasse, Luc Fabresse

http://mooc.pharo.org