



**Learning Object-Oriented  
Programming and Design with TDD**

# Learning From Real Examples

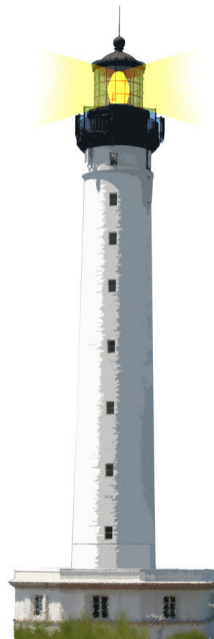
Stéphane Ducasse

<http://stephane.ducasse.free.fr>



<http://www.pharo.org>

**W7S05**

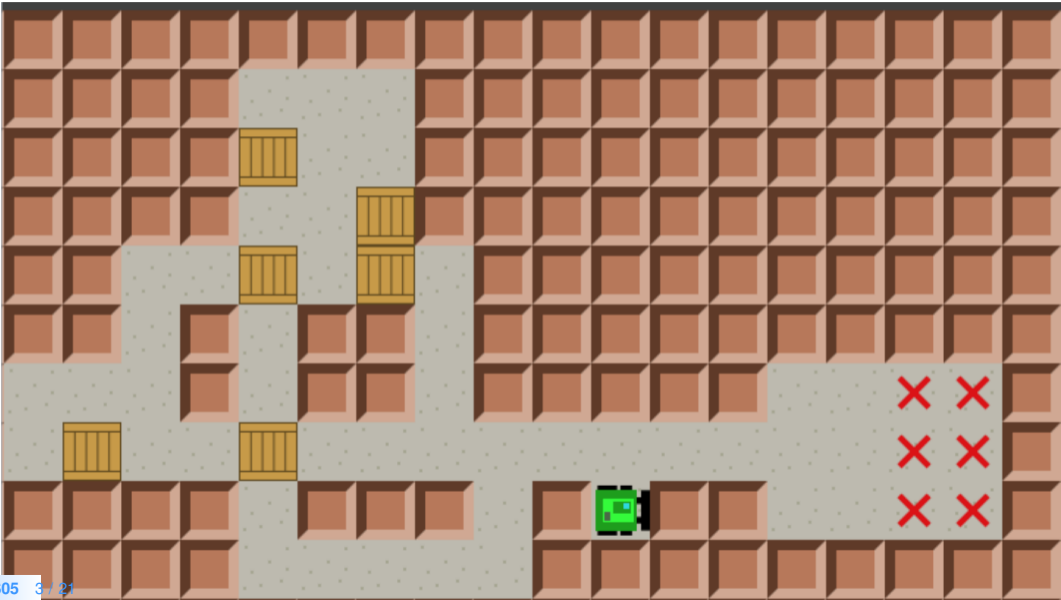


# What You Will Learn

- Thinking about design
- Comparing design
- Techniques also applicable to code/design review



# Studying a Sokoban Implementation



# Looking at the Model

- Block
  - EmptyBlock
  - Wall
- GameModel
- GameState
- Maze
- MazeTemplate
- MoveResult
  - Move
    - Push
  - NoMove

We will come back later...



# Views

- AthensSokobanWindow
- Ⓢ GameView
- Ⓢ AthensGameView
- Ⓢ NiceView
- Ⓢ PacmanView
- Ⓢ SimpleColorView
- Ⓢ TranscriptGameView

# Views are Decoupled from Model

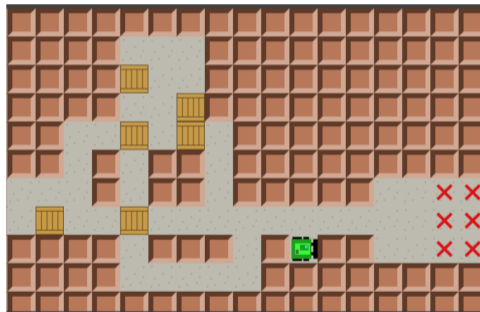
Good decomposition View/UIs

- Support multiple views
- Logic of displaying is encapsulated in different objects



# Let us Guess the Model

- Wall
- Floor
- Box
- Robot
- Target
- Board



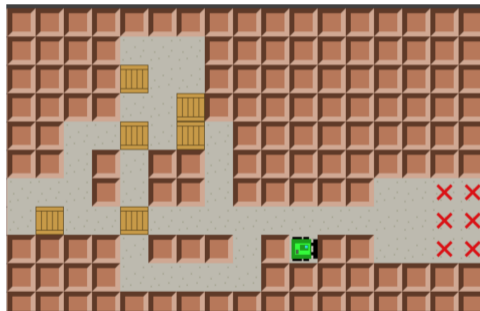
# Comparing Models

- Wall
- Floor
- Box
- Robot
- Target

Instead we got

- Block
  - EmptyBlock
  - Wall

Let us study that...





# Checking API

Type checks and disguised type checks

- often violate the "Don't Ask, Tell" principle
- favor hardcoded conditional design

For example:

- isEmptyBlock
- isWall

Let us check the way this API is used



# Too Many ifs....

```
GameView >> drawBlock: aBlock on: aCanvas
(aBlock isWall) ifTrue: [ self drawWall: aCanvas ].
(aBlock isEmptyBlock) ifTrue: [
  (aBlock hasPlayer) ifTrue: [
    (aBlock hasTarget) ifTrue: [ self drawTargetAndPlayer: aCanvas ].
    (aBlock hasTarget) ifFalse: [ self drawPlayer: aCanvas ]].
  (aBlock hasPlayer) ifFalse: [
    (aBlock hasBox) ifTrue: [
      (aBlock hasTarget) ifTrue: [ self drawTargetAndBox: aCanvas ].
      (aBlock hasTarget) ifFalse: [ self drawBox: aCanvas ]].
    (aBlock hasBox) ifFalse: [
      (aBlock hasTarget) ifTrue: [ self drawTarget: aCanvas ].
      (aBlock hasTarget) ifFalse: [ self drawEmptyBlock: aCanvas ]].
  ]
```



# Analysis....

- Changing a block is difficult
- Reuse logic (in another game) is impossible
- Logic is complex



# Why?

- The model only defines
  - EmptyBlock and
  - Wall
- There is no Player, no Target, no Box abstraction
- Too much logic is put in EmptyBlock



# A Better Model

- Block
  - Box
  - BoxOnTarget
  - EmptyBlock
  - Player
  - PlayerOnTarget
  - Wall



# A Possible Solution

```
GameView >> drawBlock: aBlock on: aCanvas  
  (aBlock isWall) ifTrue: [ self drawWall: aCanvas ].  
  (aBlock isEmptyBlock) ifTrue: [  
    ...
```

Becomes

```
AthensGameView >> drawBlock: aBlock on: aCanvas  
  aBlock drawOn: aCanvas view: self
```

```
Wall >> drawOn: aCanvas view: aView  
  aView drawWall: aCanvas
```

```
EmptyBlock >> drawOn: aCanvas view: aView  
  aView drawEmptyBlock: aCanvas
```



# A Possible Solution: Telling instead of Asking

```
AthensGameView >> drawBlock: aBlock on: aCanvas  
aBlock drawOn: aCanvas view: self
```

Tells the block that it should be drawn...

```
Wall >> drawOn: aCanvas view: aView  
aView drawWall: aCanvas
```

The wall tell the canvas that it should be drawn as a wall



# A Possible Solution: Telling instead of Asking

## Analysis

- The canvas has still the knowledge how to draw but does not ask the object about its kind.
- The canvas just **tells** a block to draw itself
- The current block **tells** the canvas to draw it accordingly

(see Lecture on double dispatch)





# Back to the Model

What are:

- MoveResult
  - Move
    - Push
  - NoMove
- Kind of Command objects
- Good to support Undo



# Studying the API

```
MoveResult >> isMove  
^ false
```

```
MoveResult >> isPush  
^ false
```

```
MoveResult >> isNoMove  
^ true
```

```
GameState >> moveBy: aDirection  
| moveResult |  
moveResult := maze moveBy: aDirection.  
(moveResult isMove) ifTrue: [ moves := moves + 1 ].  
(moveResult isPush) ifTrue: [  
    pushes := pushes + 1.  
    moves := moves + 1 ].  
self addMoveResult: moveResult.
```

# Do not ask tell

```
GameState >> moveBy: aDirection  
| moveResult |  
moveResult := maze moveBy: aDirection.  
moveResult updateGameState: self.  
self addMoveResult: moveResult.
```

```
Move >> updateGameState: aGameState  
aGameState incrementMoves
```

```
Push >> updateGameState: aGameState  
super updateGameState: aGameState.  
aGameState increasePushes
```

```
NoMove >> updateGameState: aGameState  
self
```



# Conclusion

- Messages act as a dispatcher
- Avoid conditional when possible
- Tell do not ask objects



A course by Stéphane Ducasse  
<http://stephane.ducasse.free.fr>

Reusing some parts of the Pharo Mocc by

Damien Cassou, Stéphane Ducasse, Luc Fabresse  
<http://mocc.pharo.org>



Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France  
<https://creativecommons.org/licenses/by-nc-nd/3.0/fr/>