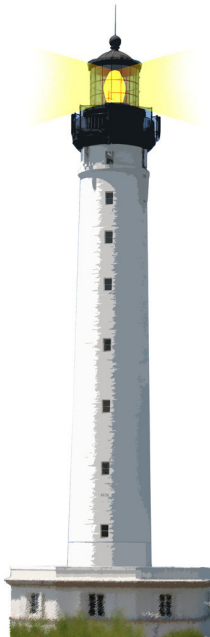
Object-Oriented
Design Lecture

Stone Paper Scissors

Stéphane Ducasse

http://stephane.ducasse.free.fr
http://car.mines-douai.fr/luc



Pharo

http://www.pharo.org

Objectives

- Another look at double dispatch
- Basis for Visitor Design pattern
- Avoid hardcoding conditionals

```
(Stone new play: Paper new)
>>> #paper
```

Stone Paper Scissors via Tests

```
StonePaperScissorsTest >> testPaperIsWinning
  self assert: (Stone new play: Paper new) equals: #paper
```

Stone Paper Scissors via Tests

```
StonePaperScissorsTest >> testPaperIsWinning
  self assert: (Stone new play: Paper new) equals: #paper
```

```
StonePaperScissorsTest >> testStoneAgainsStone
  self assert: (Stone new play: Stone new) equals: #draw
```

```
StonePaperScissorsTest >> testStoneIsWinning
  self assert: (Stone new play: Scissors new) equals: #stone
```

Let us start

```
StonePaperScissorsTest >> testPaperIsWinning
  self assert: (Stone new play: Paper new) equals: #paper
```

```
Stone >> play: anotherTool
  ^ ...
```

Paper playAgainstStone:

```
StonePaperScissorsTest >> testPaperIsWinning
  self assert: (Stone new play: Paper new) equals: #paper
```

```
Stone >> play: anotherTool
  ^ anotherTool playAgainstStone: self
```

```
Paper >> playAgainstStone: aStone
  ...
```

Paper playAgainstStone:

```
StonePaperScissorsTest >> testPaperIsWinning
  self assert: (Stone new play: Paper new) equals: #paper
```

```
Stone >> play: anotherTool
  ^ anotherTool playAgainstStone: self
```

```
Paper >> playAgainstStone: aStone
  ^ #paper
```

Other playAgainstStone:

```
Scissors >> playAgainstStone: aStone
  ^ #stone
```

```
Stone >> playAgainstStone: aStone
  ^ #draw
```

Scissors now

```
StonePaperScissorsTest >> testScissorsIsWinning
  self assert: (Scissors new play: Paper new) = #scissors
```

```
Scissors >> play: anotherTool
  ^ anotherTool playAgainstScissors: self
```

```
Scissors >> playAganstScissors: aScissors
  ^ #draw
```

```
Paper >> playAgainstScissors: aScissors
  ^ #scissors
```

```
Stone >> playAgainstScissors: aScissors
  ^ #stone
```

Paper now

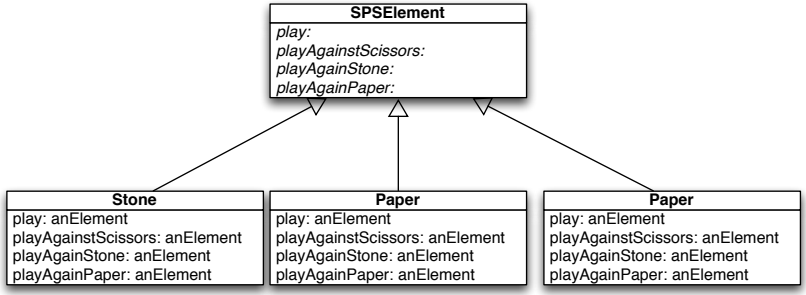
```
Paper >> play: anotherTool
  ^ anotherTool playAgainstPaper: self
```

```
Scissors >> playAgainstPaper: aPaper
  ^ #scissors
```

```
Paper >> playAgainstPaper: aPaper
  ^ #draw
```

```
Stone >> playAgainstPaper: aPaper
  ^ #paper
```

Overview


SPSElement
play:
playAgainstScissors:
playAgainStone:
playAgainPaper:
Stone
play: anElement
playAgainstScissors: anElement
playAgainStone: anElement
playAgainPaper: anElement
Paper
play: anElement
playAgainstScissors: anElement
playAgainStone: anElement
playAgainPaper: anElement
Paper
play: anElement
playAgainstScissors: anElement
playAgainStone: anElement
playAgainPaper: anElement

Remark

In this example we do not need to pass the argument during the double dispatch

```
Scissors >> playAgainstPaper: aPaper
  ^ #scissors
```

```
Scissors >> playAgainstPaper
  ^ #scissors
```

Thinking more...

When we return a token or a number we should check to do something after. So passing blocks may be better.

```
Paper new competeWith: Paper new
  onDraw: [ Game incrementDraw ]
  onReceiverWin: [ ]
  onReceiverLose: [ ]
```

Conclusion

- Powerful
- Modular
- Just sending an extra message to an argument and using late binding

A course by

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

and

Luc Fabresse
http://car.mines-douai.fr/luc