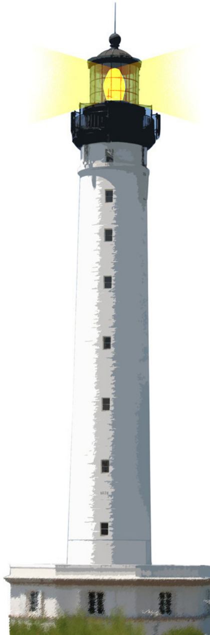
A double dispatch starter

S. Ducasse, G. Polito, P. Tesone, and L. Fabresse



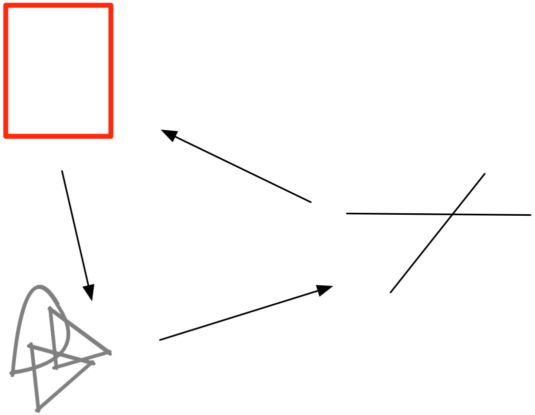
http://www.pharo.org

Goals

- In the quest of dispatch
- No conditionals!
- implementing:

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

Goals



Stone Paper Scissors: one Test

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

The inverse too

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

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

Let us start

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

```
Stone >> vs: anotherTool
    ^ ...
```

Hint 0

- The solution does not contain an explicit condition (No if, no checks)
- Remember sending a message is making a choice: it selects the right method

Hint 1: 3 classes

- Stone
- Paper
- Scissors

More hints

- When we execute the method vs: we know the receiver of the message
- So we have already half of the solution
- What if we introduce another method playAgainstStone to make another choice?

Defining Paper » playAgainstStone

```
Stone >> vs: anotherTool
    ^ ... playAgainstStone
```

```
Paper >> playAgainstStone
    ^ ...
```

Defining Paper » playAgainstStone

```
Stone >> vs: anotherTool
    ^ anotherTool playAgainstStone
```

```
Paper >> playAgainstStone
    ^ ...
```

Paper playAgainstStone defined

```
Stone >> vs: anotherTool
    ^ anotherTool playAgainstStone
```

```
Paper >> playAgainstStone
>>      ^ #paper
```

Stone new vs: Scissor new

Works for

```
>>> Stone new vs: Paper new
#paper
```

But not for

```
>>> Stone new vs: Scissor new
#stone
```

- How to fix this?
- Easy!

Supporting aScissor as argument

```
Stone >> vs: aScissor
    ^ aScissor playAgainstStone
```

- So we should implement playAgainstStone on Scissor

```
Scissors >> playAgainstStone
    ^ ...
```

Other playAgainstStone definitions

```
Scissors >> playAgainstStone
    ^ #stone
```

```
Stone >> playAgainstStone
    ^ #draw
```

Complete code for Stone as receiver

```
Stone >> vs: anotherTool
    ^ anotherTool playAgainstStone
```

```
Paper >> playAgainstStone
    ^ #paper
```

```
Scissors >> playAgainstStone
    ^ #stone
```

```
Stone >> playAgainstStone
    ^ #draw
```

Stepping back

- We know that a method is executed on a class (here Stone)
- We **send another message to the argument** to select another method (here playAgainstStone)
- Two messages to be able to select a method based on its receiver AND argument

Full Scissors code

```
Scissors >> vs: anotherTool
    ^ anotherTool playAgainstScissors
```

```
Scissors >> playAgainstScissors
    ^ #draw
```

```
Paper >> playAgainstScissors
    ^ #scissors
```

```
Stone >> playAgainstScissors
    ^ #stone
```

Full Paper code

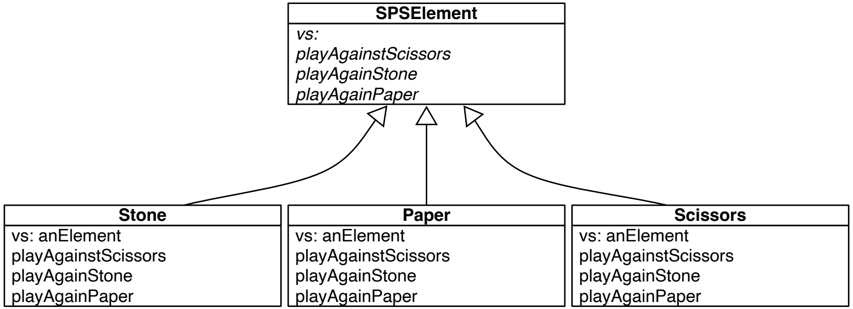
```
Paper >> vs: anotherTool
    ^ anotherTool playAgainstPaper
```

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

```
Paper >> playAgainstPaper
    ^ #draw
```

```
Stone >> playAgainstPaper
    ^ #paper
```

Solution overview



Double dispatch

- Two messages: vs: and one of playAgainstPaper, playAgainstStone or, playAgainstScissors
- First the system selects the correct vs:
- Second it selects the second method

Remark

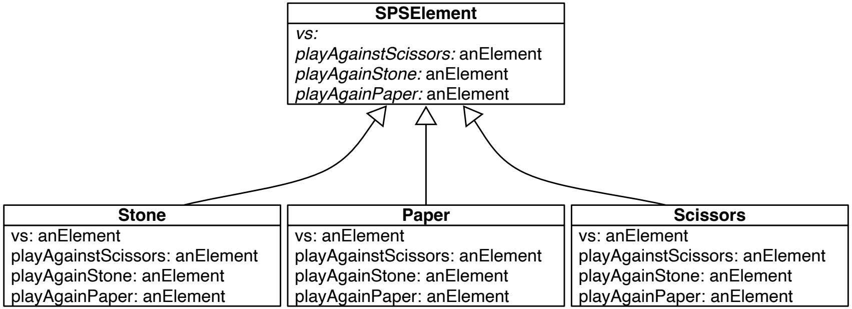
- In this toy example we do not need to pass the argument during the double dispatch
- But in general this is important as we want to do something with the first receiver (as in Visitor DP)

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

will just be

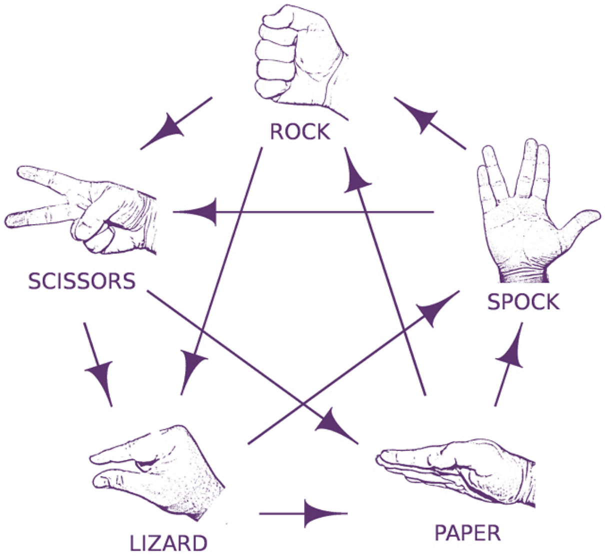
```
Scissors >> playAgainstPaper: aScissors
    ^ #scissors
```

With an argument


SPSElement
vs:
playAgainstScissors: anElement
playAgainStone: anElement
playAgainPaper: anElement
Stone
vs: anElement
playAgainstScissors: anElement
playAgainStone: anElement
playAgainPaper: anElement
Paper
vs: anElement
playAgainstScissors: anElement
playAgainStone: anElement
playAgainPaper: anElement
Scissors
vs: anElement
playAgainstScissors: anElement
playAgainStone: anElement
playAgainPaper: anElement

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

Extending it...



Extensible

- You can extend Stone, Paper, Scissors with Spock and Lizard **without changing any line** of existing code.
- Implement it!

Conclusion

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

A course by

S. Ducasse, G. Polito, and Pablo Tesone

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/`