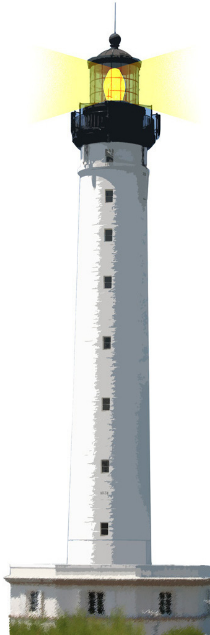
Advanced Object-Oriented Design

A double dispatch starter

S. Ducasse

http://www.pharo.org

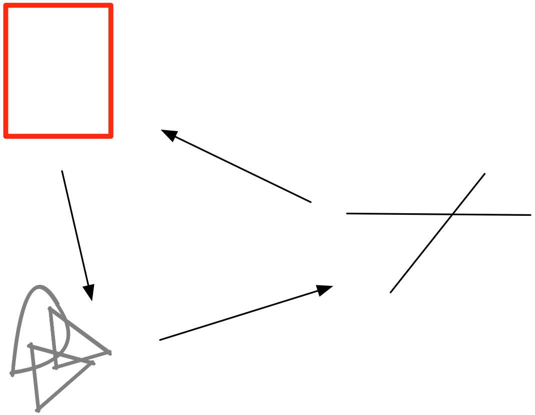


Goals

- In the quest of dispatch
- No conditionals!

```
>>> (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
  ^ ...
```

Hints

- The solution does not contain an explicit condition
- Remember sending a message is making a choice: selecting the right method
- When we execute the method `vs:` we know the receiver
- What if we introduce another method to make another choice?

Paper playAgainstStone:

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

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

Paper playAgainstStone

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

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

Paper playAgainstStone

Works for

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

But not for

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

- How to fix this?
- Easy!

Other playAgainstStone

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

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

In total

```
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

Scissors now

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

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

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

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

Paper now

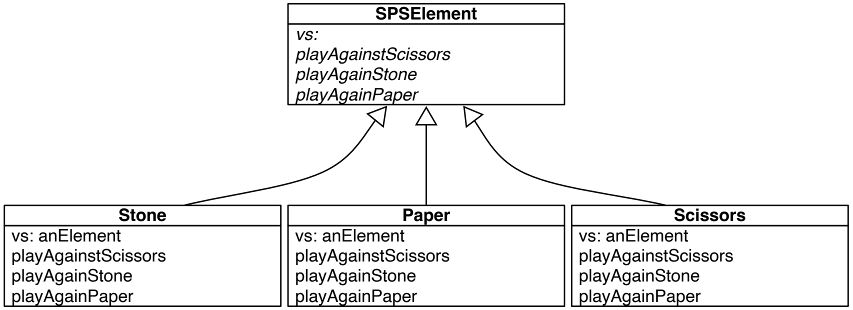
```
Paper >> vs: anotherTool
  ^ anotherTool playAgainstPaper
```

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

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

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

Solution Overview



Remark

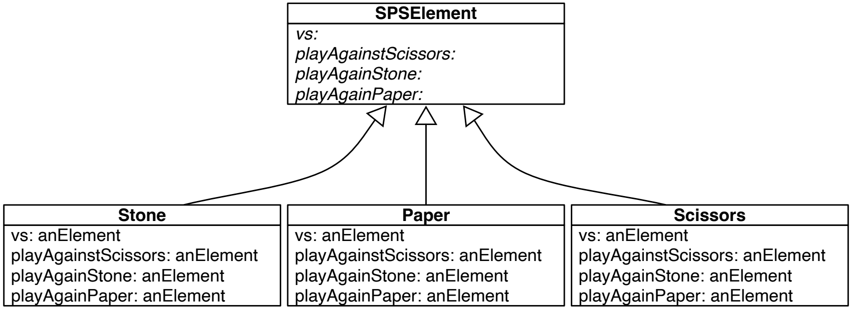
- In this toy example we do not need to pass the argument during the double dispatch
- But in general this is important as we want to do something with the first receiver (as in Visitor DP)

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

will just be

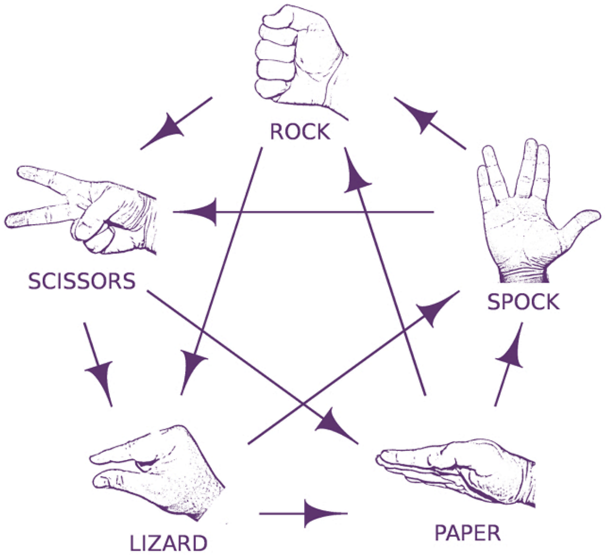
```
Scissors >> playAgainstPaper: aScissors
  ^ #scissors
```

Remark


SPSElement
vs:
playAgainstScissors:
playAgainStone:
playAgainPaper:
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...



Conclusion

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

A course by

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