A double dispatch starter

Stone Paper Scissors

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

- Exercise dispatch
- Do not use conditionals!
- Implement:
  
  > Stone new vs: Paper new
  
  #paper
Goals

ROCK ← PAPER

SCISSORS
Stone Paper Scissors: one Test

```small
StonePaperScissorsTest >> testPaperIsWinning

self assert: (Stone new vs: Paper new) equals: #paper
```
The inverse too

```smalltalk
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: anElement
  ^ ...

Hint 1

- 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 2: 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
- Introduce another method `playAgainstStone` to make another choice
Defining Paper » playAgainstStone

Stone >> vs: anElement
  ^ ... playAgainstStone

Paper >> playAgainstStone
  ^ ...

Defining Paper » playAgainstStone

Stone >> vs: anElement
  ^ anElement playAgainstStone

Paper >> playAgainstStone
  ^ ...

Paper playAgainstStone definition

Stone >> vs: anElement
^ anElement 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
...

• How to fix this?
• Easy!
Supporting aScissor as argument

Stone >> vs: aScissor
  ^ aScissor playAgainstStone

- So we should implement playAgainstStone on Scissor

Scissors >> playAgainstStone
  ^ ...

M6S1  14 / 27
Other playAgainstStone definitions

Scissors >> playAgainstStone
^ #stone

Stone >> playAgainstStone
^ #draw
Full code of Stone

Stone >> vs: anElement
  ^ anElement playAgainstStone

Paper >> playAgainstStone
  ^ #paper

Scissors >> playAgainstStone
  ^ #stone

Stone >> playAgainstStone
  ^ #draw
Stepping back

- While executing the method Stone»vs:, we know that the method is executed on Stone class.
- We send another message to the argument to select another method (here playAgainstStone).
- Conclusion: Two messages to be able to select a method based on its receiver AND argument.
Full code of Scissors

Scissors >> vs: anElement
  ^ anElement playAgainstScissors

Scissors >> playAgainstScissors
  ^ #draw

Paper >> playAgainstScissors
  ^ #scissors

Stone >> playAgainstScissors
  ^ #stone
Full code of Paper

Paper >> vs: anElement
   ^ anElement 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 Design Pattern)

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

will just be

```ruby
Scissors >> playAgainstPaper: aScissors
  ^ #scissors
```
With an argument

Paper >> vs: anotherTool
  ^ anotherTool playAgainstPaper: self
Extending it...
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
Produced as part of the course on http://www.fun-mooc.fr

Advanced Object-Oriented Design and Development with Pharo

A course by
S.Ducasse, L. Fabresse, G. Polito, and P. 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/

2023