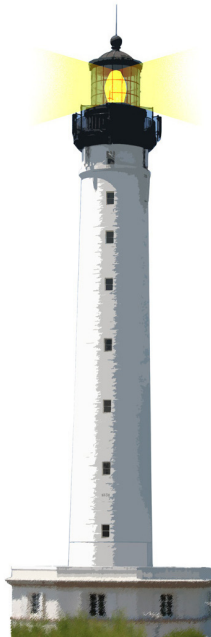
Double Dispatch

Adding numbers as a Kata

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



Outline

- Some **fun** exercises
- Thinking about them
- Discovering double dispatch
- **Chewing** double dispatch
- Stepping back

Adding Integer and Float primitives

Given the following primitives

primitive addi(i,j) returns the addition of two integers i + j

primitive addf(f1,f2) returns the addition of two floats f1 + f2

i.asFloat() converts an integer to a float

Adding Integer and Float

```
>>> 1 + 2
3
>>> 1.1 + 2
3.1
>>> 2 + 1.3
3.3
>>> 1.1 + 2.2
3.3
```

Implement `+`

But with not a single explicit conditional (no `if`)

A first hint

Sending a message is making a choice and classes support choice expressions

So

- Two classes Integer and Float

And

- Two classes Integer and Float
- Two methods +: one in each class

Let us see

```
Integer >> + aNumber
  "fill me up :)"



Float >> + aNumber



  "fill me up :)"
```

Another key hint

When you execute a method you know that the receiver is from the class of the method!

Let us get started

Imagine that we add one method sumWithInteger: anInteger

Adding sumWithInteger: anInteger

```
Integer >> + aNumber

  "fill me up :)"

Integer >> sumWithInteger: anInteger
  ...



Float >> + aNumber


  "fill me up :)"
```

Look like an easy definition

```
Integer >> sumWithInteger: anInteger
  ^ addi(self, anInteger)
```

Here we strongly assume that anInteger is of class Integer

How do we connect them?

```
Integer >> + aNumber
  ^ ...

Integer >> sumWithInteger: anInteger
  ^ addi(self, anInteger)


Float >> + aNumber

  "fill me up :)"
```

An hint: it should work for `1 + 2`

Now we can add 1+2

```
Integer >> + aNumber
  ^ aNumber sumWithInteger: self

Integer >> sumWithInteger: anInteger
  ^ addi(self, anInteger)


Float >> + aNumber

  "fill me up :)"
```

With folded constants: 1 + 2

```
Integer (1) >> + 2
  ^ 2 sumWithInteger: 1

Integer (2) >> sumWithInteger: 1
  ^ addi(2, 1)
```

What about 2 + 1.2?

```
Integer >> + aNumber
  ^ aNumber sumWithInteger: self

Integer >> sumWithInteger: anInteger
  ^ addi(self, anInteger)



Float >> + aNumber

Oops....?
```

Looks like we need sumWithInteger: anInteger on Float

What about 2 + 1.2?

```
Float >> sumWithInteger: anInteger
  "fill me up :)"
```

Looks easy

```
Float >> sumWithInteger: anInteger
  ^ addf(self, asFloat(anInteger))
```

Here we assume that the argument is instance of Integer

Now we support 2 + 1.2

```
Integer >> + aNumber
  ^ aNumber sumWithInteger: self

Integer >> sumWithInteger: anInteger
  ^ addi(self, anInteger)



Float >> + aNumber


Float >> sumWithInteger: anInteger
  ^ addf(self, asFloat(anInteger))
```

With folded constants: 2 + 1.2

```
> Integer (2) >> + 1.2
>    ^ 1.2 sumWithInteger: 2

  Integer >> sumWithInteger: anInteger
    ^ addi(self, anInteger)

  Float >> + aNumber


> Float (1.2) >> sumWithInteger: 2
>    ^ addf(1.2, asFloat(2))
```

What about 1.2 + 2.1?

```
Integer >> + aNumber
  ^ aNumber sumWithInteger: self
Integer >> sumWithInteger: anInteger
  ^ addi(self , anInteger)



Float >> + aNumber
  ^ ...

Float >> sumWithInteger: anInteger
  ^ addf(self, asFloat(anInteger))
```

We should define + on Float

Now we are supporting 1.2 + 2.1?

```
Integer >> + aNumber
  ^ aNumber sumWithInteger: self
Integer >> sumWithInteger: anInteger
  ^ addi(self , anInteger)



Float >> + aNumber
  ^ aNumber sumWithFloat: self

Float >> sumWithInteger: anInteger
  ^ addf(self, asFloat(anInteger))
```

Supporting 1.2+ 2

```
  Integer >> + aNumber
    ^ aNumber sumWithInteger: self
  Integer >> sumWithInteger: anInteger
    ^ addi(self , anInteger)

>  Integer >> sumWithFloat: aFloat
>    ^ addf(aFloat, asFloat(self))

  Float >> + aNumber
    ^ aNumber sumWithFloat: self
  Float >> sumWithInteger: anInteger
    ^ addf(self, asFloat(anInteger))

>  Float >> sumWithFloat: aFloat
>    ^ addf(self, aFloat)
```

With folded constants: 1.2 + 2

```
  Integer >> + aNumber
    ^ aNumber sumWithInteger: self
  Integer >> sumWithInteger: anInteger
    ^ addi(self , anInteger)
>  Integer (2) >> sumWithFloat: 1.2
>    ^ addf(1.2, asFloat(2))

>  Float (1.2) >> + 2
>    ^ 2 sumWithFloat: 1.2
  Float >> sumWithInteger: anInteger
    ^ addf(self, asFloat(anInteger))
  Float >> sumWithFloat: aFloat
    ^ addf(self, aFloat)
```

Ok now relax

- Take a pen and follow the calls to the following expressions
- Follow with your fingers if necessary :)

```
1 + 2
1.1 + 2
2 + 1.3
1.1 + 2.2
```

Key point

```
Integer >> + aNumber
  ^ aNumber sumWithInteger: self
```

Two choices/messages:

- one for +: select one Integer or Float implementation
- one for sumWithInteger:, sumWithFloat:: select one Integer or Float implementation

Exercise 2: How to add Fraction?

```
f := Fraction num: 1 denum: 2.

>>> f num
1
>>> f denum
2
>>> f asFloat
0.5
```

```
(1/2) + 3
3 + 3.3
1.3 + (2/5)
(1/3) + (4/3)
```

Introducing Fraction

```
Fraction >> + aNumber
  ^ ...
```

It follows the same pattern.

Introducing Fraction

```
Fraction >> + aNumber
  ^ aNumber sumWithFraction: self
  ...
```

Introducing Fraction

```
Fraction >> + aNumber
  ^ aNumber sumWithFraction: self
Fraction >> sumWithFraction: aFrac
  ...
```

Supports (1/2) + (4/3)

```
Fraction >> + aNumber
  ^ aNumber sumWithFraction: self
Fraction >> sumWithFraction: aFrac
  ^ Fraction num: (self num * aFrac denum) + (aFrac num * self denum)
    denum: aFrac denum * self denum
...
```

Taking care of Integers and Floats

```
Fraction >> + aNumber
  ^ aNumber sumWithFraction: self
Fraction >> sumWithFraction: aFrac
  ^ Fraction num: (self num * aFrac denum) + (aFrac num * self denum)
    denum: aFrac denum * self denum

Integer >> sumWithFraction: aFrac
  ...
Float >> sumWithFraction: aFrac
  ...
```

Now supporting: (1/2) + 1 and (1/2) + 2.1

```
Fraction >> + aNumber
  ^ aNumber sumWithFraction: self
Fraction >> sumWithFraction: aFrac
  ^ Fraction num: (self num * aFrac denum) + (aFrac num * self denum)
    denum: aFrac denum * self denum
...
Integer >> sumWithFraction: aFrac
  ^ Fraction num: (self * aFrac denum) + aFrac num denum: aFrac denum
Float >> sumWithFraction: aFrac
  ^ addf(self, aFrac asFloat)
```

Full code for Fraction

```
Fraction >> + aNumber
  ^ aNumber sumWithFraction: self
Fraction >> sumWithFraction: aFrac
  ^ Fraction num: (self num * aFrac denum) + (aFrac num * self denum)
    denum: aFrac denum * self denum
Fraction >> sumWithInteger: anInteger
  ^ Fraction num: (self num + anInteger * aFrac denum) denum: aFrac denum
Fraction >> sumWithFloat: aFloat
  ^ addf(self aFloat, aFloat)
Integer >> sumWithFraction: aFrac
  ^ Fraction num: (self * aFrac denum) + aFrac num denum: aFrac denum
Float >> sumWithFraction: aFrac
  ^ addf(self, aFrac asFloat)
```

Ok now relax

- Take a pen and follow the calls to the following expressions
- Follow with your fingers if necessary :)

```
(1/2) + 3
3 + 3.3
1.3 + (2/5)
(1/3) + (4/3)
```

Key point

```
X >> + aNumber
  ^ aNumber sumWithX: self
```

Two choices/messages:

- one for +: select one Integer, Float, or Fraction implementation
- one for sumWithInteger:, ...: select one Integer, Float, or Fraction implementation

Stepping back

- We can add Fraction without changing any previous method
- Another example of "Sending a message is making a choice"

Different kinds of messages

- Primary operations
- Double dispatching methods

Double Dispatch

- Essence of Visitor Design Pattern (see Lecture)
- Double dispatch is a clear illustration of **Do not ask, Tell** OOP tenet
- Used really frequently for event, drawing, ...

When not using Double Dispatch

- No class to dispatch on
- We need an different instance of dispatch to!

What about overloading

- Double dispatch is **also** useful in statically-typed languages
- Overloading for double dispatch will not work in presence of inheritance: Will not select the expected method

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