a Die + a DieHandle:

Practicing dispatch more

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

- See how conditionals can be turned into extensible design using messages
- Set the basis for more complex situation such as the Visitor Design Pattern
Remember Die and DieHandle

We create a die handle, add some die to it, and roll it.

```plaintext
| handle |
handle := DieHandle new
   addDie: (Die withFaces: 6);
   addDie: (Die withFaces: 10);
   yourself.
handle roll
```
Remember summing DieHandles

We add dieHandles together as in role playing games

DieHandleTest >> testSumming
| handle |
handle := 2 D20 + 3 D10.
self assert: handle diceNumber equals: 5
New requirement: N. 1

We want to add two dices together and get a DieHandle

(Die withFaces: 6) + (Die withFaces: 6)
DieTest >> testAddTwoDice

| hd |
hd := (Die withFaces: 6) + (Die withFaces: 6).
self assert: hd dice size equals: 2.
New requirement: N. 2

We want to be able to add a dice to a dice handle and the inverse

\[(\text{Die with Faces: 6}) + 2 \text{D20}\]

\[2 \text{D20} + (\text{Die with Faces: 6})\]
DieTest >> testAddingADieAndHandle
| hd |
hd := (Die faces: 6)
+ (DieHandle new
   addDie: 6;
   yourself).
self assert: hd dice size equals: 2
Possible solution with conditions

Die >> + aDieOrADieHandle
| selfAsDieHandle |
selfAsDieHandle := DieHandle new addDie: self.
^ selfAsDieHandle + aDieOrADieHandle

We are on class Die so we

- systematically create a dieHandle with the receiver and
- sum it with the argument
Possible solution with conditions

DieHandle >> + aDieOrADieHandle

^ (aDieOrADieHandle class = DieHandle)
    ifTrue: [ | handle |
        handle := self class new.
        self dice do: [ :each | handle addDie: each ].
        aDieOrADieHandle dice do: [ :each | handle addDie: each ].
        handle ]
    ifFalse: [ | handle |
        handle := self class new.
        self dice do: [ :each | handle addDie: each ].
        handle addDie: aDieOrADieHandle.
        handle ]
Limits of this approach

- Works for two cases but does not really scale!
- Each time you have a new case you have to change this method
- If we have different objects that should interact with different operations e.g.,
  - different kinds of text objects: list, figures, paragraph, section, title, text, reference...
  - different operations: rendering text, HTML, LaTeX
Hints

- Sending a message is making a choice
  - the system **selects the correct** method for a given receiver and executes it
- To select a method based on the receiver AND the argument, we have to send a message to the argument
Sketch of the solution

- When we add two elements (die or dieHandle) together,
- We tell the argument to add itself to the receiver

We are explicit about the receiver since we know it:

- When the receiver is a die, we tell the argument to add itself to a die
- When the receiver is a die handle, we tell the argument add itself to a die handle

Let us do it now!
First adding two dice

A first try

Die >> + aDie

^ DieHandle new
   addDie: self;
   addDie: aDie;
   yourself
Limits

Die >> + aDie
^ DieHandle new
  addDie: self;
  addDie: aDie;
  yourself

But aDie can be:

- A dice
- A die handle

For example as in

(Die withFaces: 6) + 2 D20
Introducing sumWithDie:

Adding two dice is useful, let us keep it and rename it:

```smalltalk
Die >> sumWithDie: aDie

^ DieHandle new
   addDie: self;
   addDie: aDie; yourself
```

Now we just say to the argument that we want to add a die

```smalltalk
Die >> + aDicable
^ aDicable sumWithDie: self
```
Adding Two Dice and Ready for More

```
+ aDicable
   ^ aDicable sumWithDie: self

: aDie(6) + : aDie(10)

Die

+ aDicable
   sumWithDie: aDie

 sumWithDie: aDie

^ DieHandle new
   addDie: self;
   addDie: aDie;
   yourself

Message 1

Message 2
```
Handling DieHandle as Argument

For example:

(Die withFaces: 6) + 2 D20

We just have to define a different `sumWithDie:` method on DieHandle

```smalltalk
DieHandle >> sumWithDie: aDie
    | handle |
    handle := self class new.
    self dice do: [ :each | handle addDie: each ].
    handle addDie: aDie.
    ^ handle
```
**Handling DieHandle as Argument**

```plaintext
+ aDicable^
  ^ aDicable sumWithDie: self
Die
  + aDicable
  sumWithDie: aDie
  ^ DieHandle new
  addDie: self;
  addDie: aDie;
  yourself
sumWithDie: aDie
  ^ DieHandle new
  addDie: self;
  addDie: aDie;
  yourself

DieHandle
  sumWithDie: aDie

<table>
<thead>
<tr>
<th>handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle := self class new.</td>
</tr>
<tr>
<td>self dice do: [:each</td>
</tr>
<tr>
<td>handle addDie: aDie.</td>
</tr>
<tr>
<td>^ handle</td>
</tr>
</tbody>
</table>

: aDie(6) + 2 D20
```
Sending a Message is Making a Choice

Message 1

Message 2

Message A

Message B
Sending a Message is Making a Choice

We get two messages/choices:

- One message for +
- One message for sumWithDie:
DieHandle as a receiver

Our solution should support:

- 2 D20 + (Die withFaces: 6)
- 2 D20 + 2D3
DieHandle as a receiver

We apply the same principle: We send a message to the argument telling to add itself with an handle

```
DieHandle >> + aDicable
  ^ aDicable sumWithHandle: self
```

Then we have to support:

- Summing two die handles
- Summing one die handle and a die
**Summing an handle with another one**

```smalltalk
DieHandle >> sumWithHandle: aDieHandle
    | handle |
handle := self class new.
self dice do: [:each | handle addDie: each ].
aDieHandle dice do: [:each | handle addDie: each ].
^ handle
```
Now the argument can be a die

Since the argument can be a die, we define `sumWithHandle`: also on `Die`.

```plaintext
Die >> sumWithHandle: aDieHandle
| handle |
handle := DieHandle new.
aDieHandle dice do: [ :each | handle addDie: each ].
handle addDie: self
^ handle
```
Double Dispatch between Die and DieHandle

Die:
+ aDicable
 ^ aDicable sumWithDie: aDie
   sumWithDieHandle: aDieHandle

sumWithDie: aDie
^ DieHandle new
  addDie: self;
  addDie: aDie;
  yourself

sumWithHandle: aDieHandle
  hd |
  hd := DieHandle new.
  aDieHandle dice do: [ :each | hd addDie: each ].
  hd addDie: self
  ^ hd

DieHandle:
+ aDicable
 ^ aDicable sumWithHandle: self

sumWithDie: aDie
  I handle |
  handle := self class new.
  self dice do: [ :each | handle addDie: each ].
  handle addDie: aDie.
  ^ handle

sumWithHandle: aDieHandle
  hd |
  hd := self class new.
  self dice do: [ :each | hd addDie: each ].
  aDieHandle dice do: [ :each | hd addDie: each ].
  ^ hd
Stepping back

- We applied two times a simple principle
  - Sending a message is making a choice/selecting the right method
- So sending a message to the argument is a way to select again between a couple of methods.
Conclusion

- Powerful
- Modular (compiler with 70 nodes scales without problems)
- Just sending an extra message to an argument and using late binding once again
- Basis for advanced design such as the Visitor Design Pattern