

# Learning Object-Oriented Programming, Design and TDD with Pharos

Stéphane Ducasse

March 11, 2019

Copyright 2017 by Stéphane Ducasse.

The contents of this book are protected under the Creative Commons Attribution-ShareAlike 3.0 Unported license.

You are **free**:

- to **Share**: to copy, distribute and transmit the work,
- to **Remix**: to adapt the work,

Under the following conditions:

**Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

**Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page:  
<http://creativecommons.org/licenses/by-sa/3.0/>

Any of the above conditions can be waived if you get permission from the copyright holder. Nothing in this license impairs or restricts the author's moral rights.



Your fair dealing and other rights are in no way affected by the above. This is a human-readable summary of the Legal Code (the full license):  
<http://creativecommons.org/licenses/by-sa/3.0/legalcode>

# Contents

<b>Illustrations</b>	<b>ii</b>
<b>1 Die DSL</b>	<b>1</b>
1.1 Rolling a dice handle . . . . .	2
1.2 Role playing syntax . . . . .	2
1.3 Adding DieHandles . . . . .	2
<b>Bibliography</b>	<b>5</b>

# Illustrations



# Die DSL

Here are the possible solutions of the implementation we asked for the DSL Chapter ??.

## Define class Die

```
[Object subclass: #Die
  instanceVariableNames: 'faces'
  classVariableNames: ''
  package: 'Dice'

Die >> initialize
  super initialize.
  faces := 6
```

## Rolling a die

```
[Die >> roll
  ^ faces atRandom
```

## Define class DieHandle

```
[Object subclass: #DieHandle
  instanceVariableNames: 'dice'
  classVariableNames: ''
  package: 'Dice'

DieHandle >> initialize
  super initialize.
  dice := OrderedCollection new.
```

## Die addition

```
[ DieHandle >> addDie: aDie
  dice add: aDie
```

### 1.1 Rolling a dice handle

```
[ DieHandleTest >> testRoll
  | handle |
  handle := DieHandle new
    addDie: (Die withFaces: 6);
    addDie: (Die withFaces: 10);
    yourself.
  1000 timesRepeat: [ handle roll between: 2 and: 16 ]

[ DieHandle >> roll

  | res |
  res := 0.
  dice do: [ :each | res := res + each roll ].
  ^ res
```

### 1.2 Role playing syntax

```
[ Integer >> D20
  | handle |
  handle := DieHandle new.
  self timesRepeat: [ handle addDie: (Die withFaces: 20)].
  ^ handle

[ Integer >> D: anInteger

  | handle |
  handle := DieHandle new.
  self timesRepeat: [ handle addDie: (Die withFaces: anInteger)].
  ^ handle
```

### 1.3 Adding DieHandles

```
[ DieHandle >> + aDieHandle
  "Returns a new handle that represents the addition of the receiver
  and the argument."
  | handle |
  handle := self class new.
  self dice do: [ :each | handle addDie: each ].
  aDieHandle dice do: [ :each | handle addDie: each ].
  ^ handle
```

This definition only works if the method `dice` defined below has been defined

```
[DieHandle >> dice  
  ^ dice
```

Indeed the first expression `self dice do:` could be rewritten as `dice do:` because `dice` is an instance variable of the class `DieHandle`. Now the expression `aDieHandle dice do:` cannot. Why? Because in Pharo you cannot access the state of another object directly. Here 2 `D20` is one handle and 3 `D10` another one. The first one cannot access the `dice` of the second one directly (while it can access its own). Therefore there is a need to define a message that provide access to the `dice`.





# Bibliography