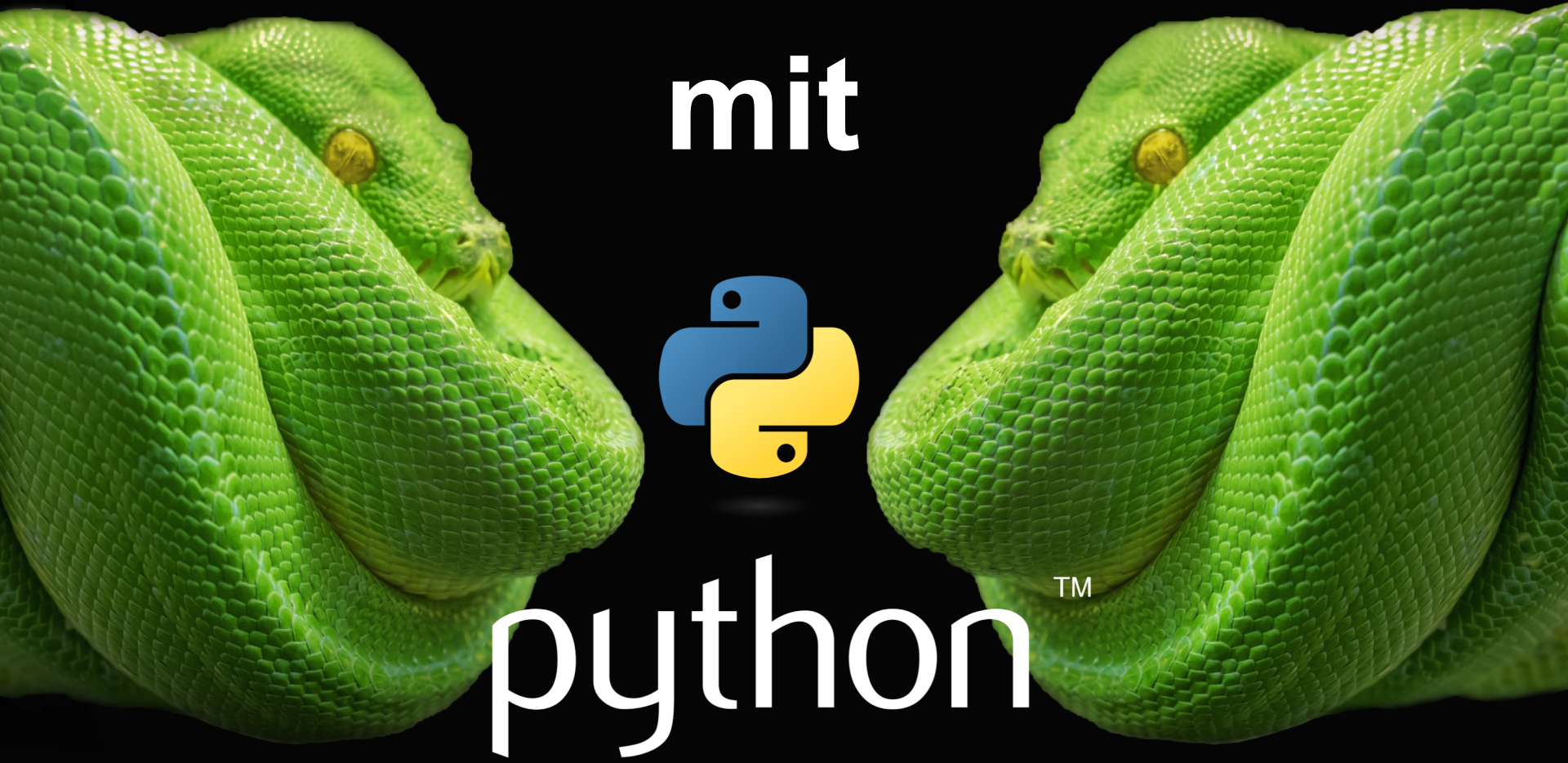
Design Patterns*

mit



python™

Dr.-Ing. Alexander Vondrous
Offenburg, den 9.12.2019

Foto: gettyimages

*Decorators

Design Pattern, Entwurfsmuster

Design Patterns beschreiben nützliche, wiederkehrende und flexible Programmierkonstrukte.

Erzeugungsmuster
Beschreiben Objekterzeugung für höhere Flexibilität und Wiederverwendbarkeit

- Factory Pattern
- Singleton
- ...

Verhaltensmuster
Beschreiben Verantwortung und Verhalten zwischen Objekten

- Observer
- Command
- ...

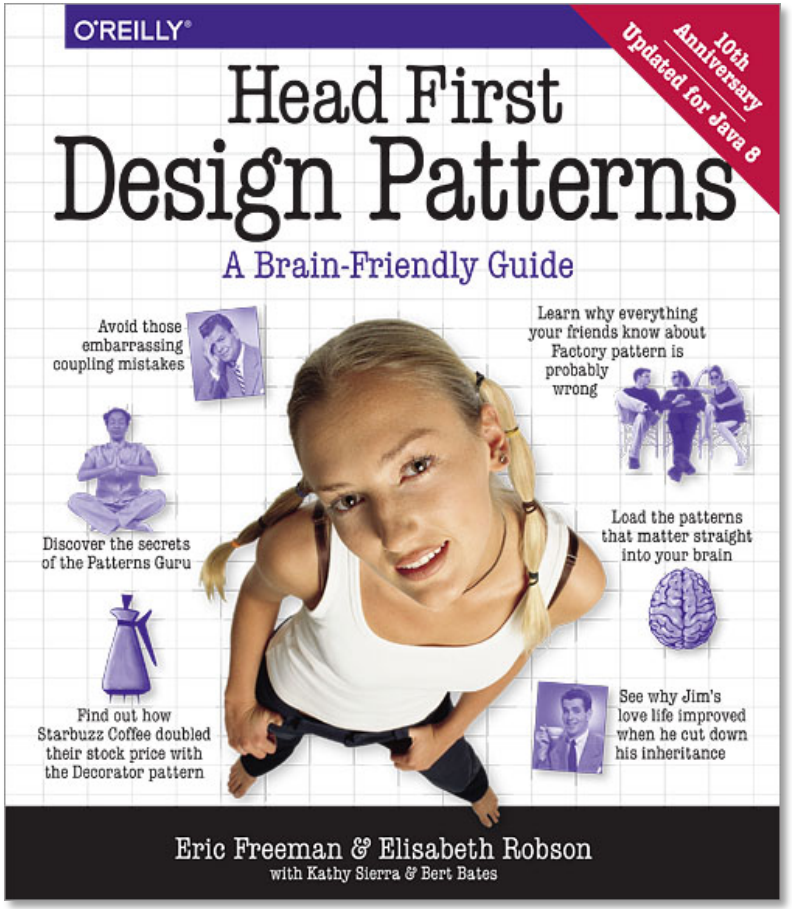
Strukturmuster
Beschreiben wie Objekte in größere Strukturen zusammengefasst werden ohne Flexibilität zu verlieren.

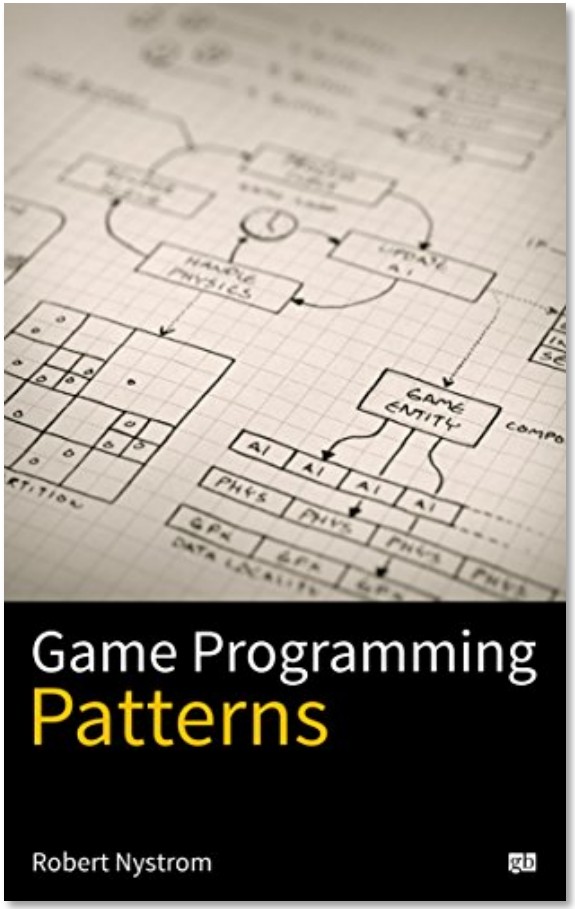
- Adapter
- Decorator
- ...

Nebenläufigkeitsmuster
Beschreiben nebenläufiges Zusammenspiel

- Barrier
- Thread pool
- ...

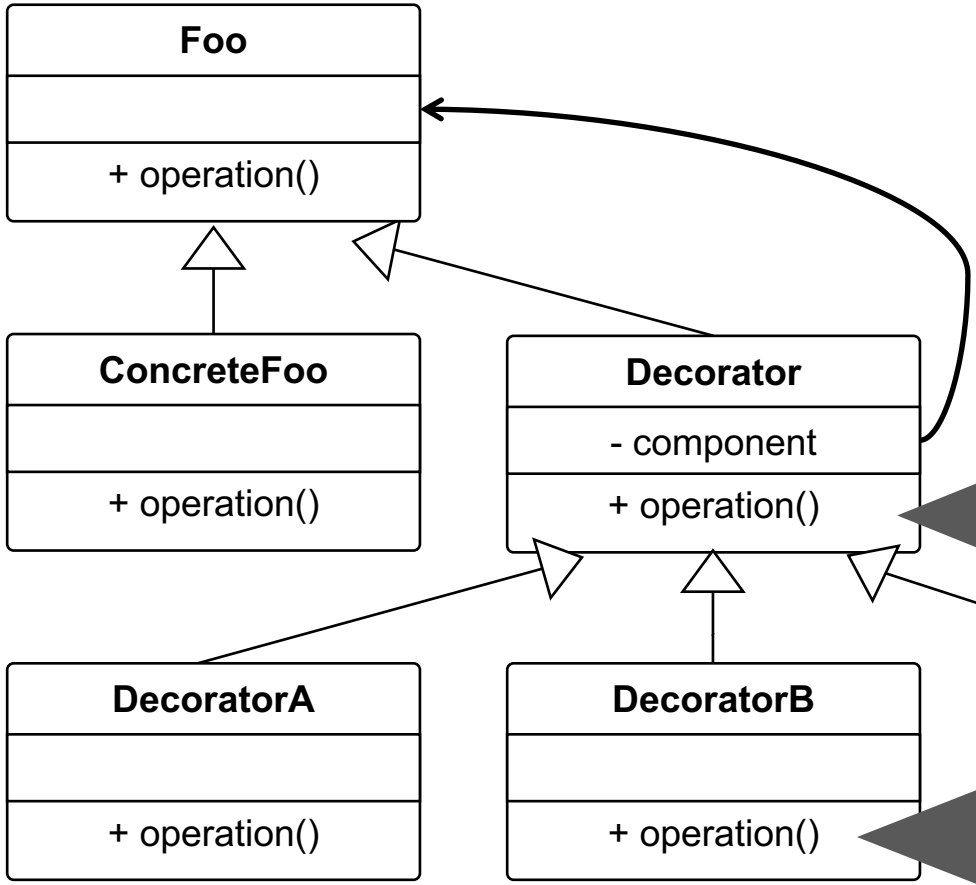
Literatur


O'REILLY
10th Anniversary
Updated for Java 8
Head First
Design Patterns
A Brain-Friendly Guide
Avoid those embarrassing coupling mistakes
Learn why everything your friends know about Factory pattern is probably wrong
Discover the secrets of the Patterns Guru
Load the patterns that matter straight into your brain
Find out how Starbuzz Coffee doubled their stock price with the Decorator pattern
See why Jim's love life improved when he cut down his inheritance
Eric Freeman & Elisabeth Robson
with Kathy Sierra & Bert Bates


Game Programming
Patterns
Robert Nystrom

Decorator

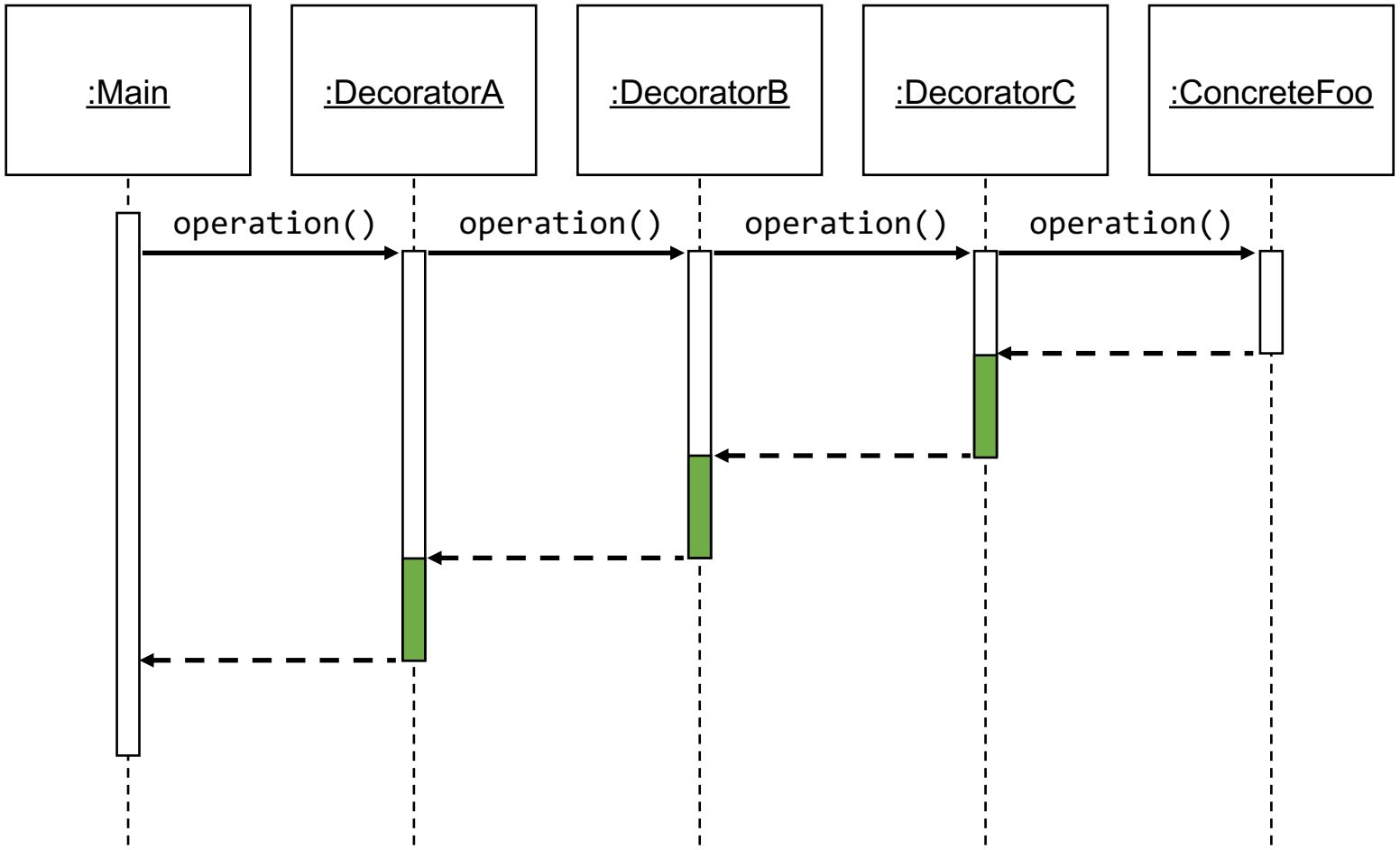
Sie möchten zur Laufzeit einem Objekt neue Aufgaben/Kompetenzen geben.
Sie möchten nicht alle möglichen Unterklassen Implementieren.




```
operation() {
  component.operation()
}
```

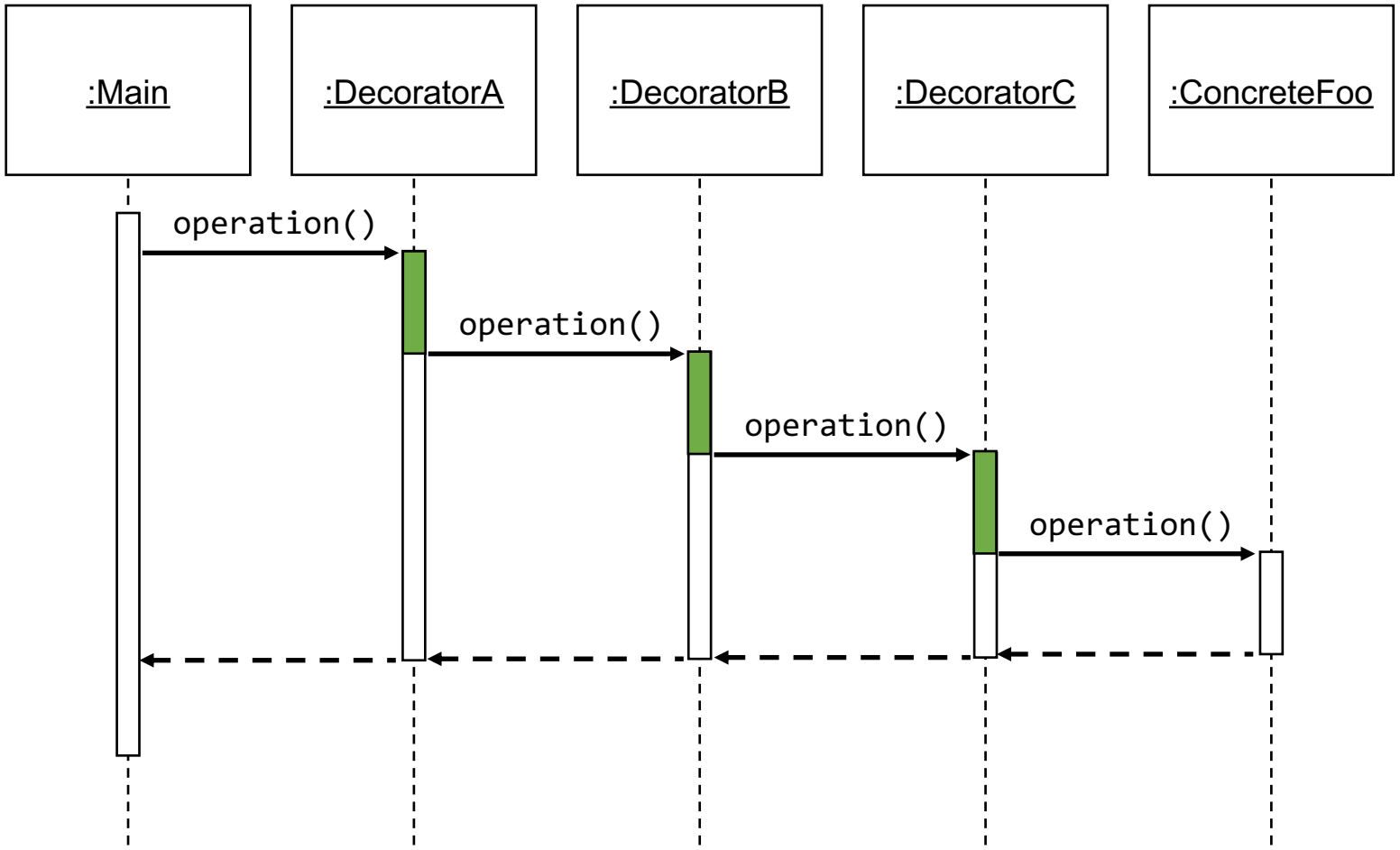
```
operation() {
  // Pre operation code
  component.operation()
  // Post operation code
}
```

Decorator



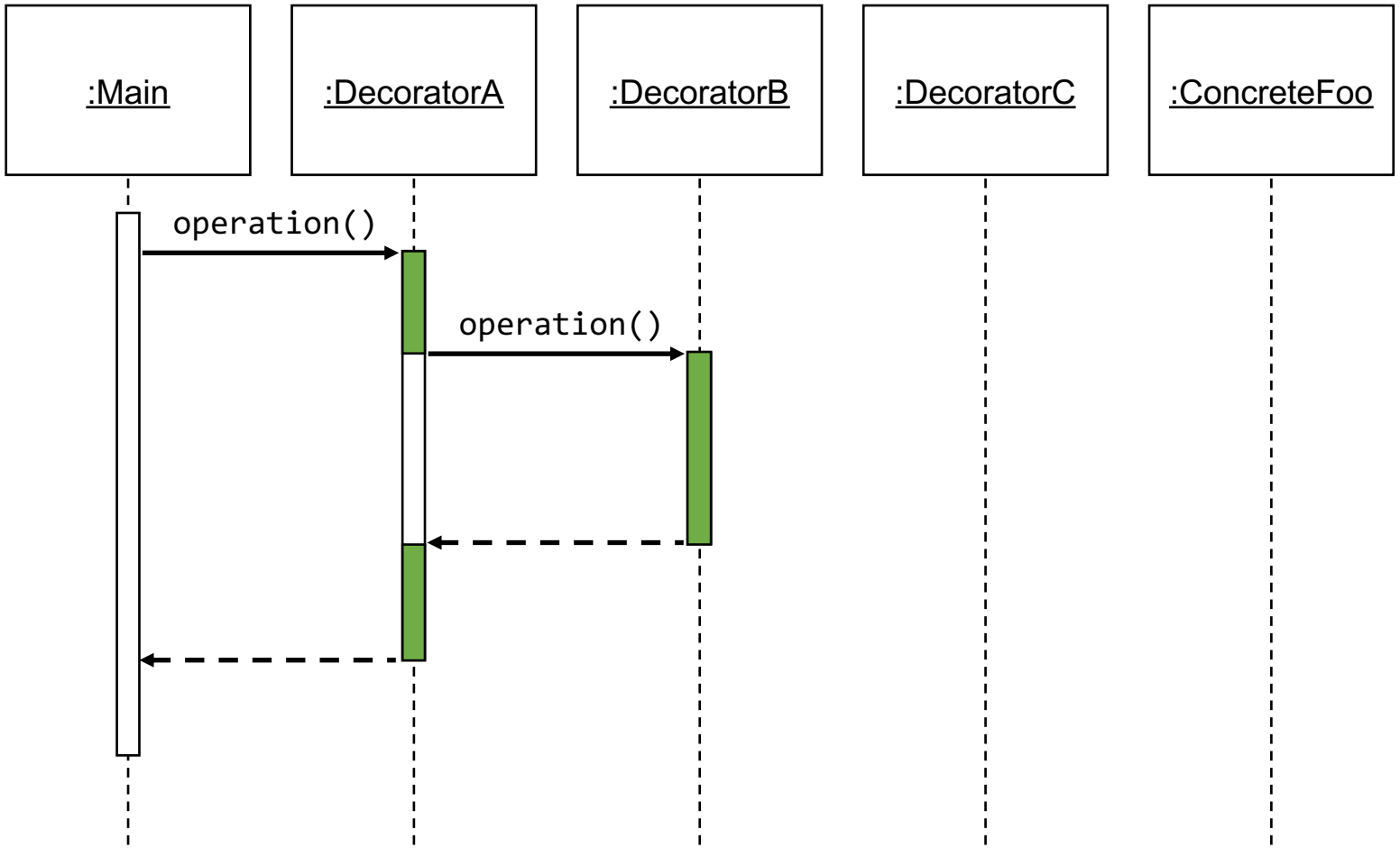
Q: Erfolgt die Erweiterung Vor (Pre) oder Nach (Post) operation()? A: Post

Decorator



Q: Erfolgt die Erweiterung Vor (Pre) oder Nach (Post) operation()? A: Pre

Decorator



Dekoratoren besitzen den vollen Zugriff auf Argumente und die Ausführung.

Pythons 1st Class Citizen

Funktionen können Variablen zugewiesen werden
Funktionen können Funktionen zurückgeben
Funktionen können als Argument übergeben werden

```
def say_hello(name):
    print('Hello %s!' % name)


say_hello('Alex')
```

```
Hello Alex!
```

```
def hey(funktion, name):
    print('Hey')
    funktion(name)

hey(say_hello, 'Alex')
```

```
Hey
Hello Alex!
```

```
foo = hey
foo(say_hello, 'Alex')
```

Closures are simple

Closure Definition

„[...] Es beschreibt eine anonyme Funktion, die Zugriffe auf ihren Erstellungskontext enthält. [...]“

Quelle: Wikipedia [Stand: 16.11.19 13:55]

```
def mult_x(x):
    def multiplizierer(a):
        return a * x

    return multiplizierer


mult_21 = mult_x(21)
print(mult_21(2))
```

```
42
```

Vereinfachte Closure Definition

Die innere Funktion, welche von einer Funktion zurückgegeben wird ist ein **Closure**.

Oder

Die eingeschlossene Funktion ist ein **Closure**.

Wozu Closures?

- Um eine gängige Schnittstelle zu bewahren (Adapter Muster)
- Um Code-Vervielfältigung zu reduzieren
- Um Funktionsaufrufe zu verzögern
- ...

Decorators in Python

Beispiel

```
def verbose(funktion):
    def wrapper():
        print('Vor' + funktion.__name__)
        result = funktion()
        print('Nach' + funktion.__name__)
        return result
    return wrapper


def hallo():
    print('Hallo')


hallo = verbose(hallo)
hallo()
```

```
Vor hallo
Hallo
Nach hallo
```

Syntactic Sugar

```
@verbose
def greet():
    print('Howdy')


greet()
```

```
Vor greet
Howdy
Nach greet
```


Doughnut: maxpixel

Parameterized Decorators

Beispiel

```
@verbose(3)
def hallo():
    print('Hallo')

hallo()
```

```
Vor hallo
Start 345234
Hallo
Ende 345241
Nach hallo
Gesamtdauer: 7
```

```
@verbose(1)
def hallo():
    print('Hallo')

hallo()
```

```
Vor hallo
Hallo
Nach hallo
```

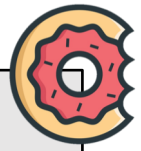
Parameterized Decorators

```
def verbose(level):
    def decorator(funktion):
        def wrapper():
            # if level > 2: start stopwatch
            print 'Vor ', funktion.__name__
            funktion()
            print 'Nach ', funktion.__name__
            # if level > 2: stop stopwatch & print
        return wrapper
    return decorator


def hallo():
    print('Hallo')

hallo = verbose(3)(hallo)
```

```
@verbose(3)
def hallo():
    print('Hallo')
```



Doughnut: maxpixel

Wozu?

Dekoratoren können

- Argumente
- die dekorierte Funktion
- Ergebnisse

Ändern und Inspizieren

Dekoratoren werden üblicherweise für folgendes genutzt:

- Teure Operation Zwischenspeichern
- Unzuverlässige Funktionen wiederholen
- sys.stdout umlenken und abfangen
- Zeitmessung
- Zeitüberschreitungen feststellen und behandeln
- Zugriffskontrolle

Beispiele

```
@login_required

@group_required('role')

@timeit

@app.route('/')

@online_environment

@log_level('all')

@log('terminal')

...
```