Context-Oriented Programming

Willian Paiva & Nathalie Craeye

January 30, 2017

Outline

The objective

Simplification and control

- Make it simpler to take the context in consideration.
- Better control over the method selection.
- Well define the entities.
- Tackle *crosscutting-concerns*.

Context and behavior variants

COP subdivides the Context into 3 categories:

Actor

Ex: Function or methods call, messages ...

Environment

Ex: GPS, battery, light sensor ...

System

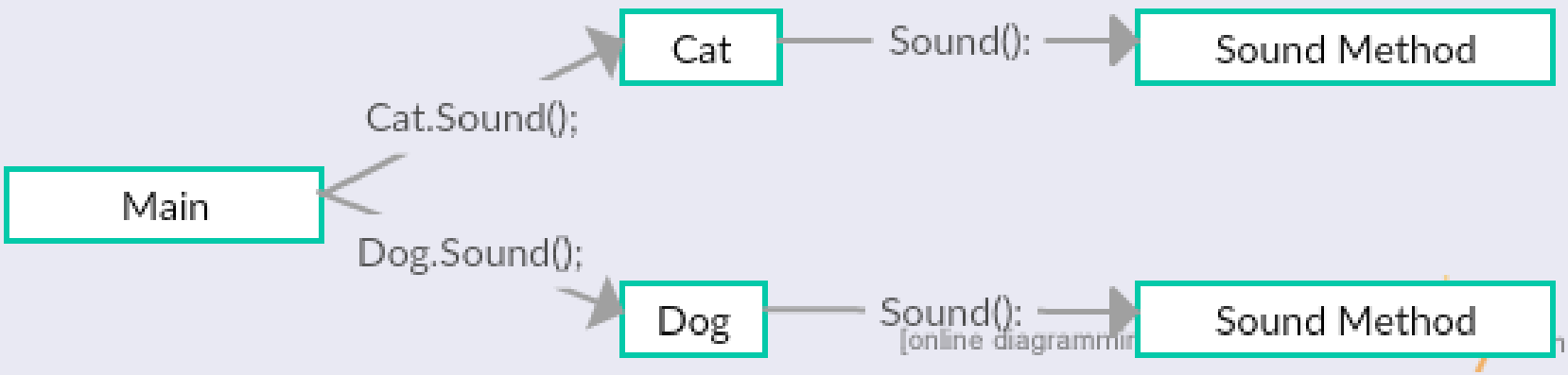
Ex: Methods, objects, subsystems ...

Multi-dimensional message dispatch

One dimension (Procedural programming)

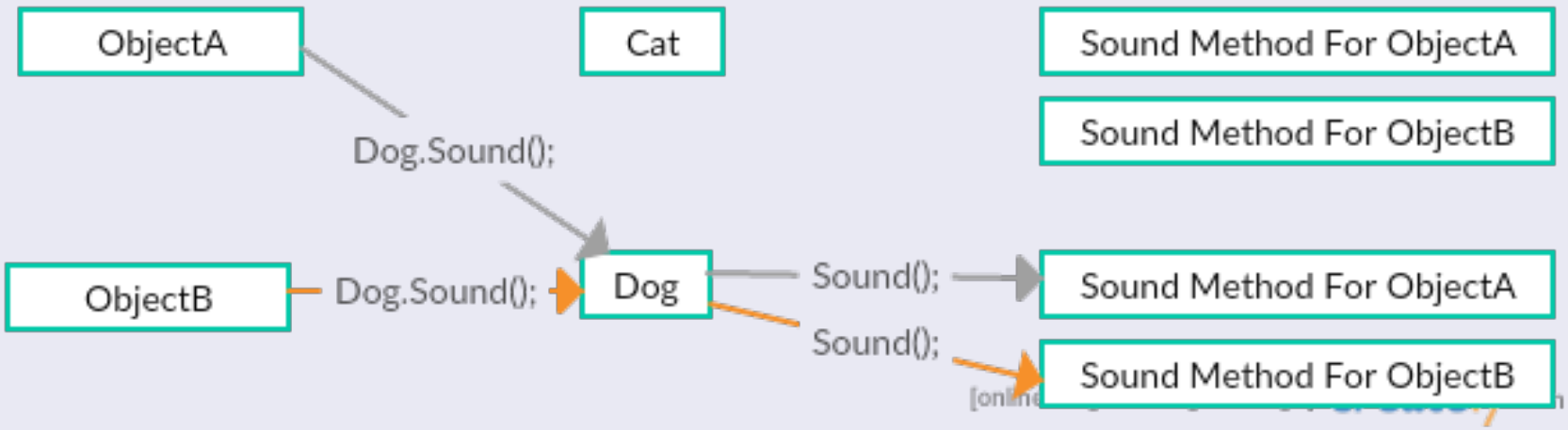
Multi-dimensional message dispatch

Two dimensions (Object-oriented programming)



Multi-dimensional message dispatch

Three dimensions (Subject-oriented programming)



Layers

Definition

- First-class entities
- Activation and deactivation
  - Arbitrary parts of the code
  - Conditional (environment)
- Scope
  - executes the code on the scope in or out the layer

Multi-dimensional message dispatch

Four dimensions (Context-oriented programming)

```
class Calculate{
    Calculate(){};
    void heavyCalc(){ someHeavyCalc();}
    Layer batWarning {
        void heavyCalc(){
            sendNotification(" Warning Low Battery level");
            proceed();
        }
    }
    Layer lowMemory {
        void heavyCalc(){
            sendNotification("Not enough memory to execute");
            throw new NotEnoughtMemoryException();
        }
    }
}
```

Multi-dimensional message dispatch

Four dimensions cont. . .

```
Calculate c = new Calculate();
if(systemMemory() < mimMem){
    with(lowMemory){
        c.heavyCalc();
    }
}else if(batteryLevel() < mimBat){
    with(batWarning){
        c.heavyCalc();
    }
}else{
    c.heavyCalc();
}
```

Decorator pattern

Common intents

- Can withdrawn responsibilities.(???)
- Add behavior or state to individual objects at run-time.

Cons

- With COP it is possible to activate and deactivate layers with a simpler syntax.

Pros

- By using the decorator pattern a class don't need to know/declare future *"decorations"*.

Aspect-oriented programming

Advantages over COP

- Decrease code scattering by a static way of programming.
- The class has no knowledge of the aspects.
- Provides similar functionalities with (*pointcut*, *after* and *before*).
- Integrated in many more languages like Java and C++.

Q&A