Context-oriented Programming

- COP is a technique for modularizing cross-cutting concerns that allows for dynamic adaptation at runtime.
- Partial Methods defined in Layers can add/modify behavior in other classes and be combined (layering).
- Our approach: Classes act as Layers.
- This poster: Rewrite design patterns with COP to overcome shortcomings compared to traditional implementations.

### Method Lookup

Assumption: OperationCounterVisitor is the only active layer.

### Language Features

- `def Cls.foo() { super(); }`: super instead of proceed.
- `def Cls.bar();`: affected object identifier.
- `def *.foo() { super(); }`: wildcard class.
- `def Cls.bar();`: layer object identifier.

### Full(C) = \[ \sum_{i=0}^{\mid C \mid} (\text{super}^i (L) \mid C) \]

### Method Composition

- Layer composition stack: `L \mid C`: projection of `L` by `C` contains only methods in `L` targeting `C`.
- \(|C|\): number of superclasses of `C`.
- \(\mid S \mid\): layer composition stack (\(\mid S \mid\) is a `d` element).
- \(<C>\): list with only `C`.

### Visitor Design Pattern

- Simple Object Interaction: No double dispatch required. "visit" method belongs to obj.
- Abstract class `AbstrVisitor`:
  ```python
def PlusExpr.visit() { left.visit(); right.visit(); }
```