

Nested Class Modularity in Squeak/Smalltalk

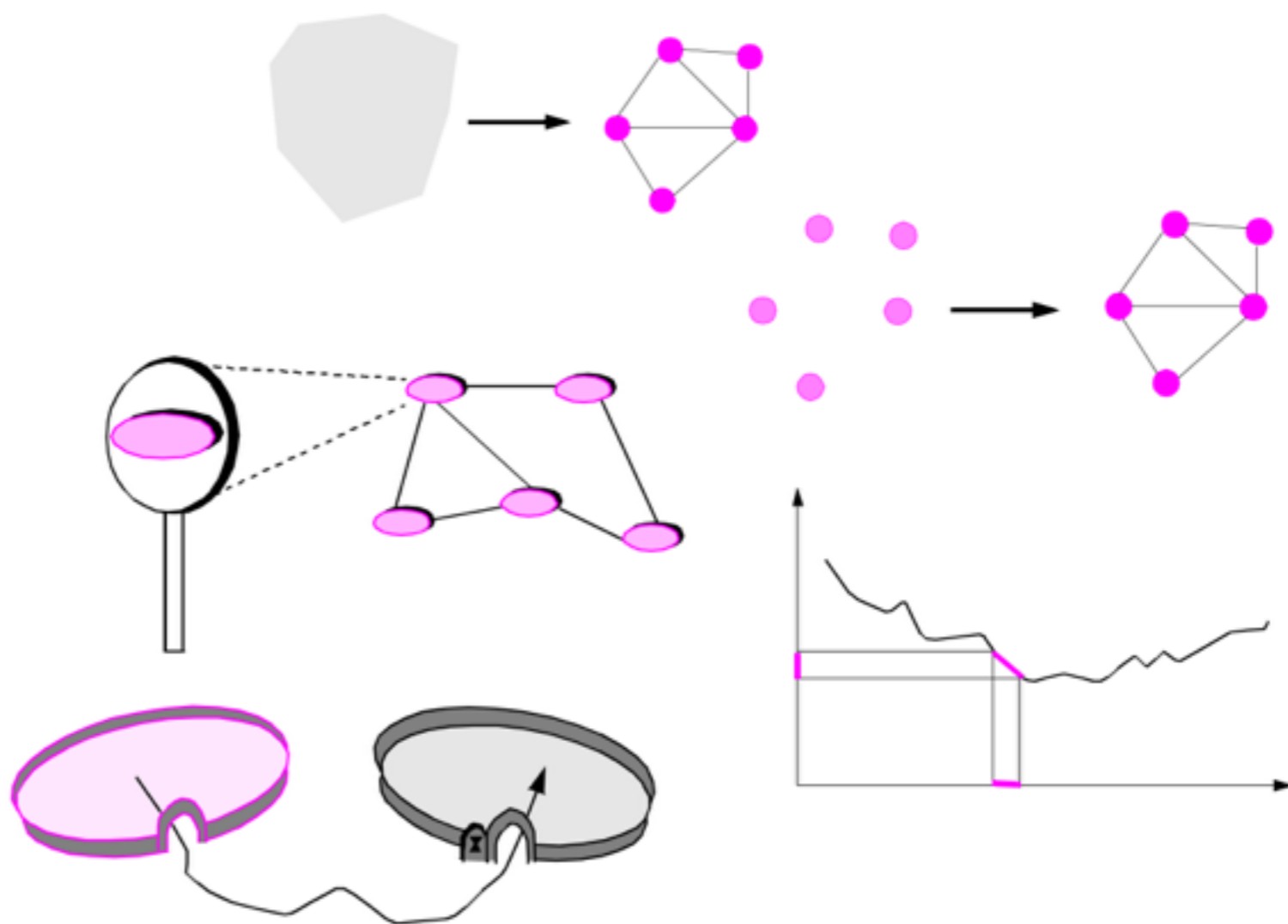
Matthias Springer
Software Architecture Group, Hasso Plattner Institute
Master's Thesis Disputation

August 21, 2015

What is Modularity?

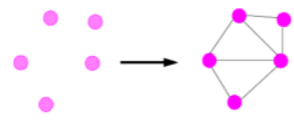
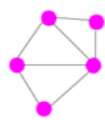
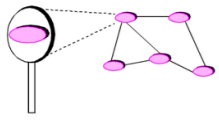
According to Bertrand Meyer (*Object-oriented Software Construction*)

- Decomposability
- Composability
- Understandability
- Continuity
- Protection



Modularity in Squeak

- Classes as modular units
- *Problems*
 - Duplicate Class Names
 - Dependency/Version Management
 - Hierarchical Decomposition



Duplicate Class Names

BroBreakout

- BroBall
- BroBlock
- BroBoundary
- BroBreakout
- BroExplosion
- **BroLevelBuilder**
- BroLevelStatistics
- BroLevelStatisticsItem
- BroLevelView
- BroLevelWorld
- BroMenuLabel
- BroMenuView
- *BroPowerup*
- BroPowerupAccelerate
- BroPowerupBall
- BroPowerupDecelerate
- BroPowerupEnlarge
- BroPowerupShrink
- BroRacket
- *BroView*
- BroWelcomeView

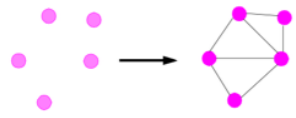
SpaceCleanup-Items

- ScuBucket
- *ScuDestructibleItem*
- ScuFloor
- *ScuItem*
- ScuMonster
- *ScuMovingItem*
- ScuPickUpItem
- ScuPlayer
- ScuPortal
- ScuSlime
- ScuWall
- ScuWater

same class names

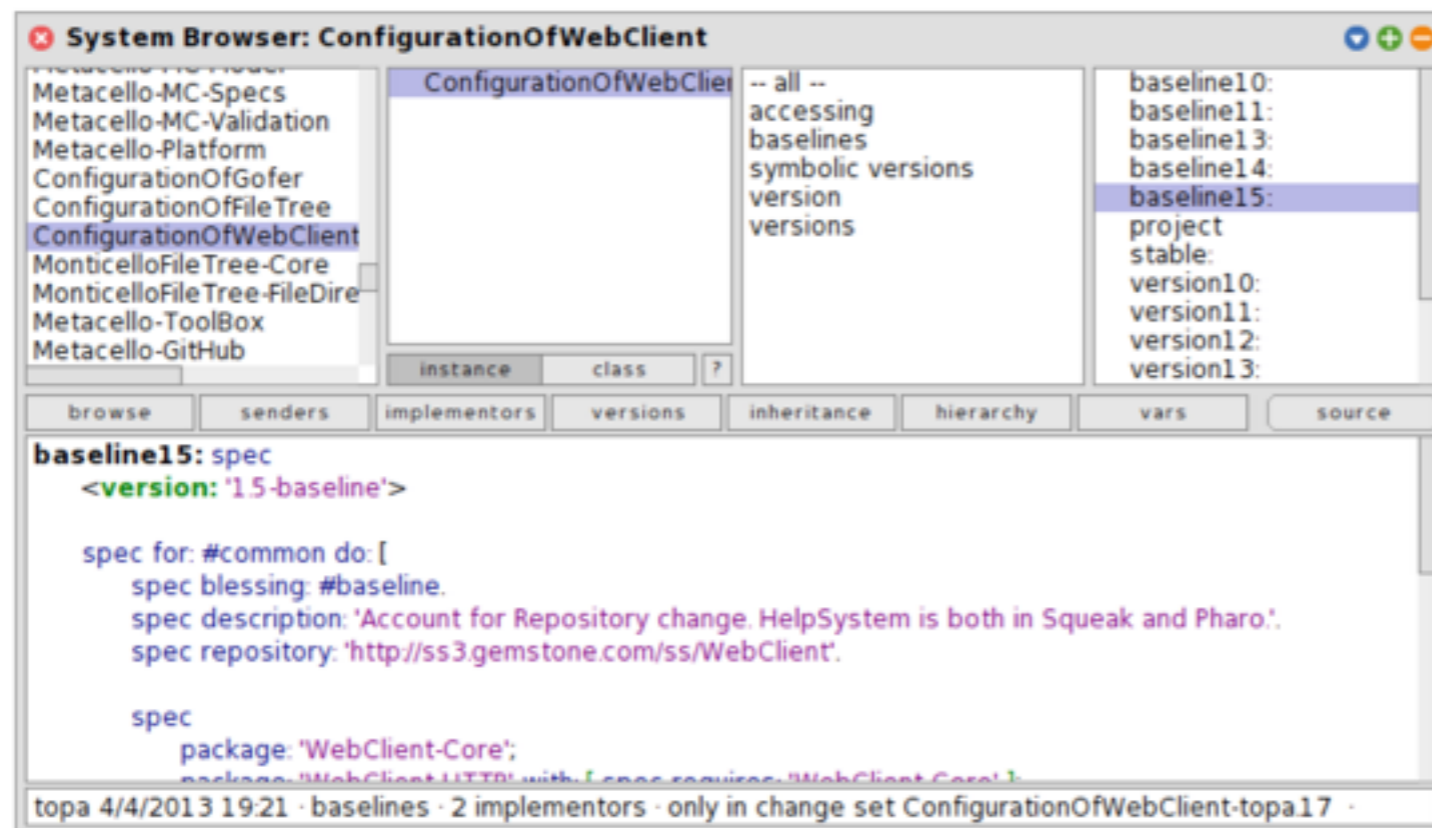
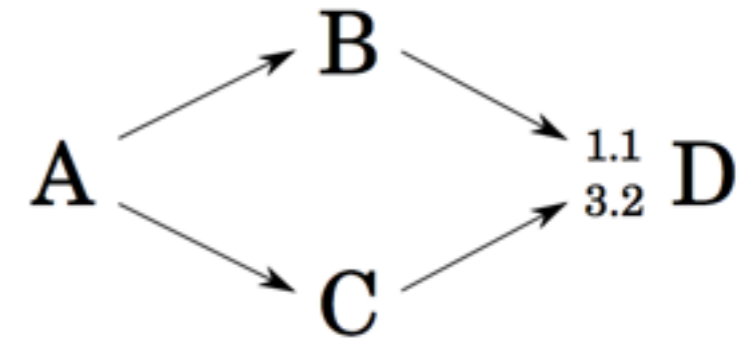
SpaceCleanup-Level

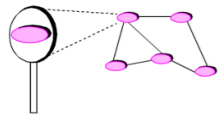
- ScuLevel
- **ScuLevelBuilder**
- ScuGridPatternLevelBuilder
- ScuRandomLevelBuilder
- ScuTile



Dependency Management

- Library version conflict
- Transitive dependency version conflict
- Overhead of dependency management systems

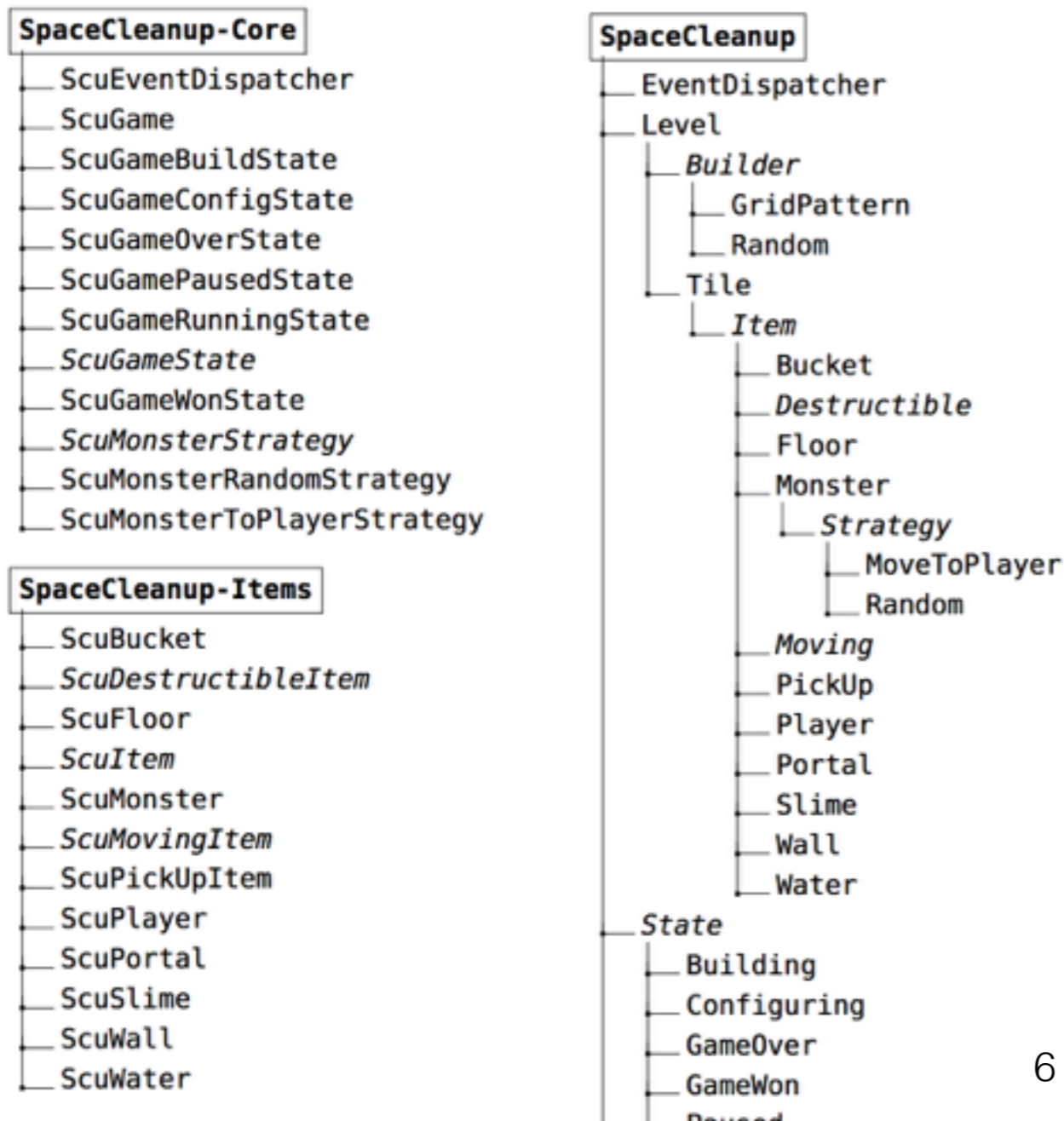




Hierarchical Decomposition

- “group together what belongs together”
(Effective Java, 2nd edition)

“On the Criteria To Be Used in Decomposing Systems into Modules”
(David L. Parnas)

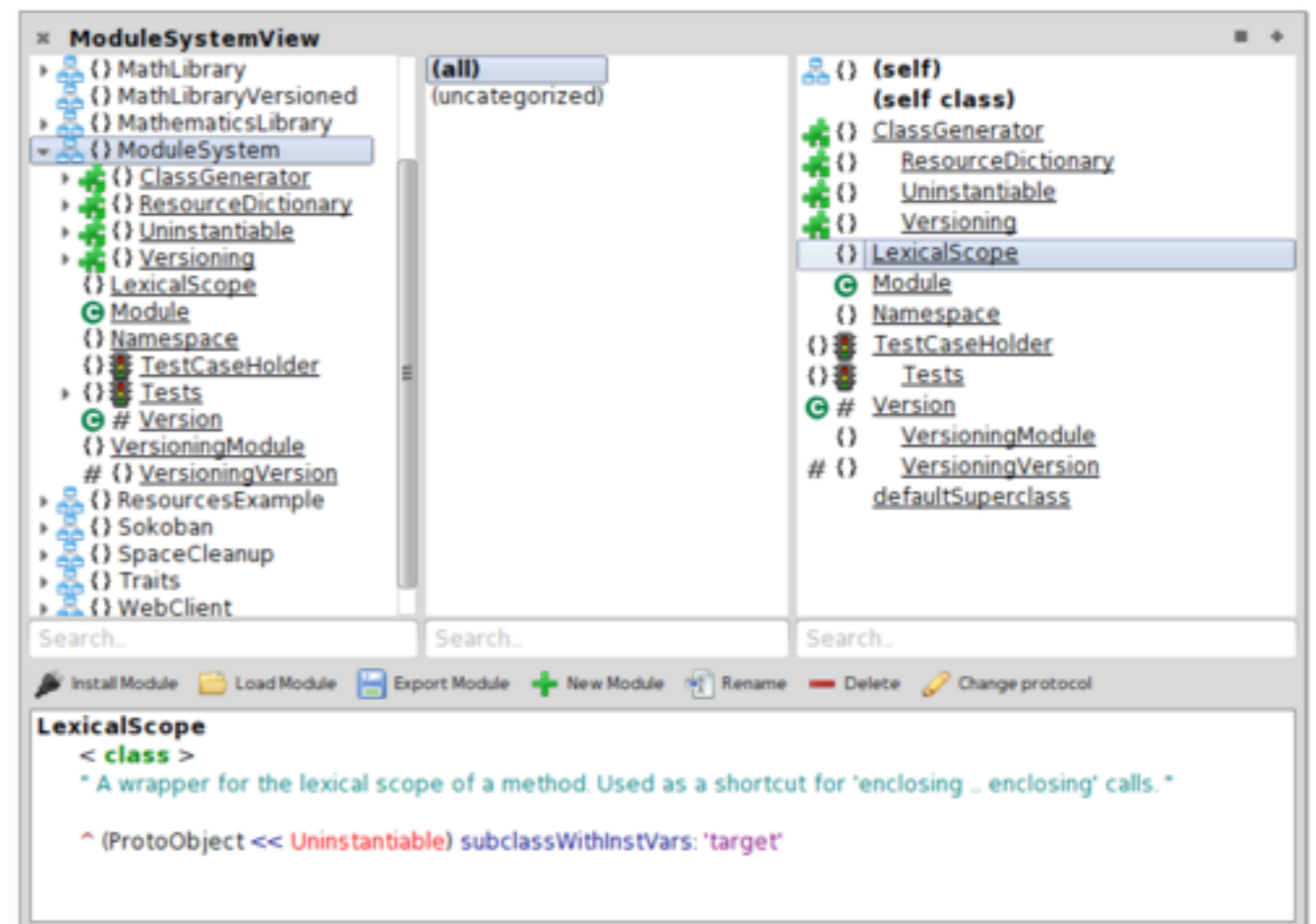


Concept

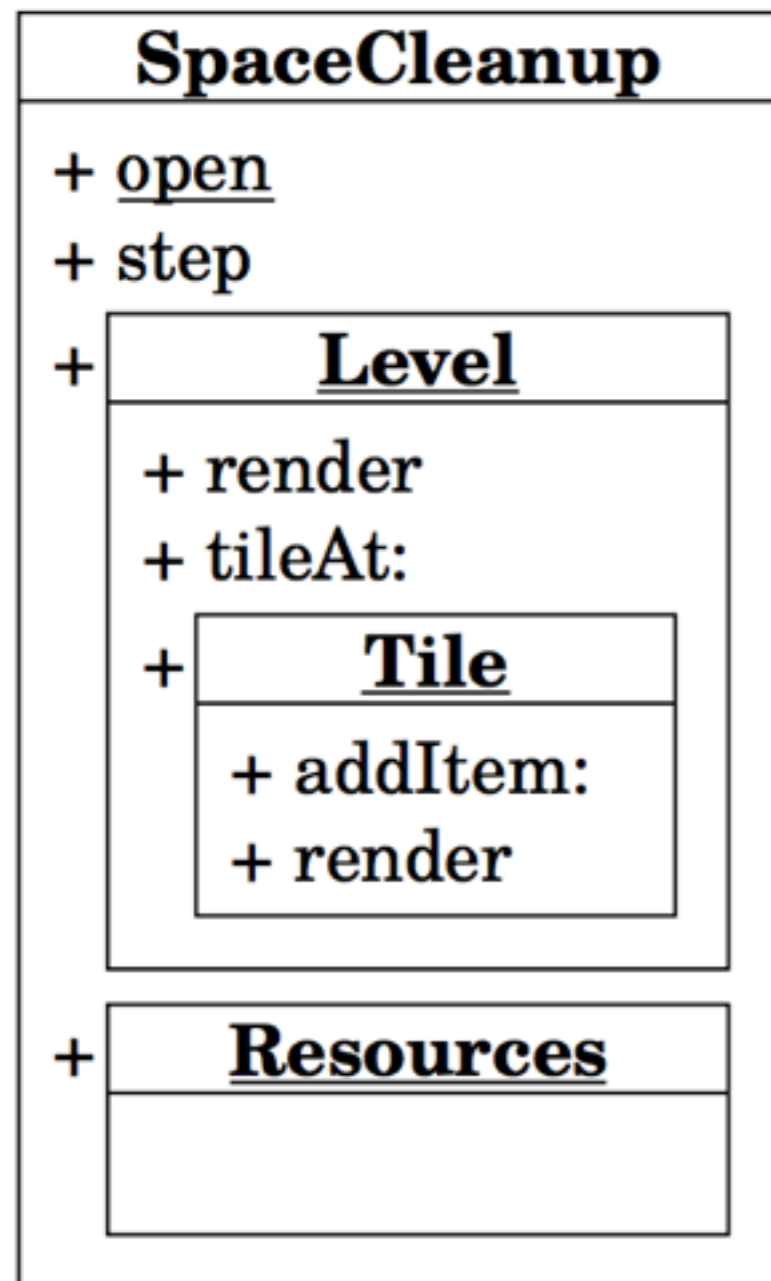
Matriona



- Module system based on class nesting for Squeak
- Matryoshka ~ Matriona
- No VM changes, minor changes Smalltalk compiler
- GUI based on Vivide

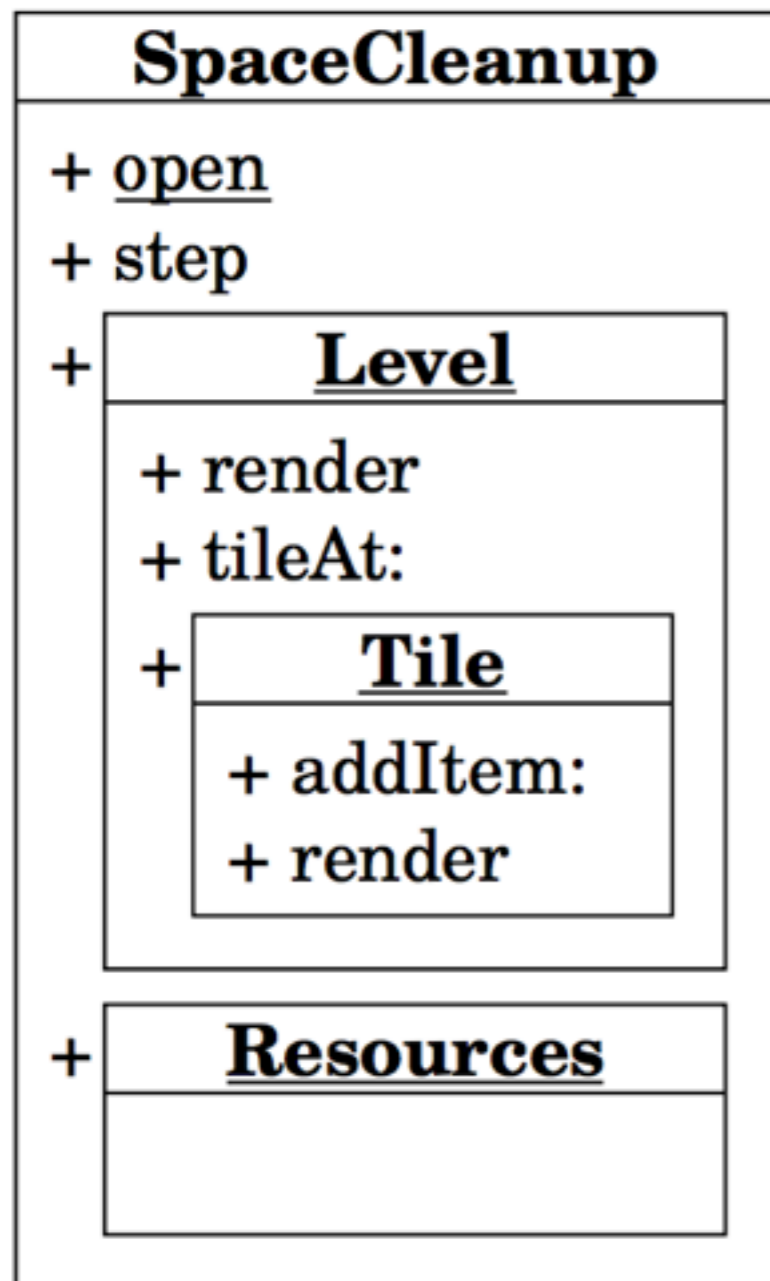


Nested Classes in Matriona

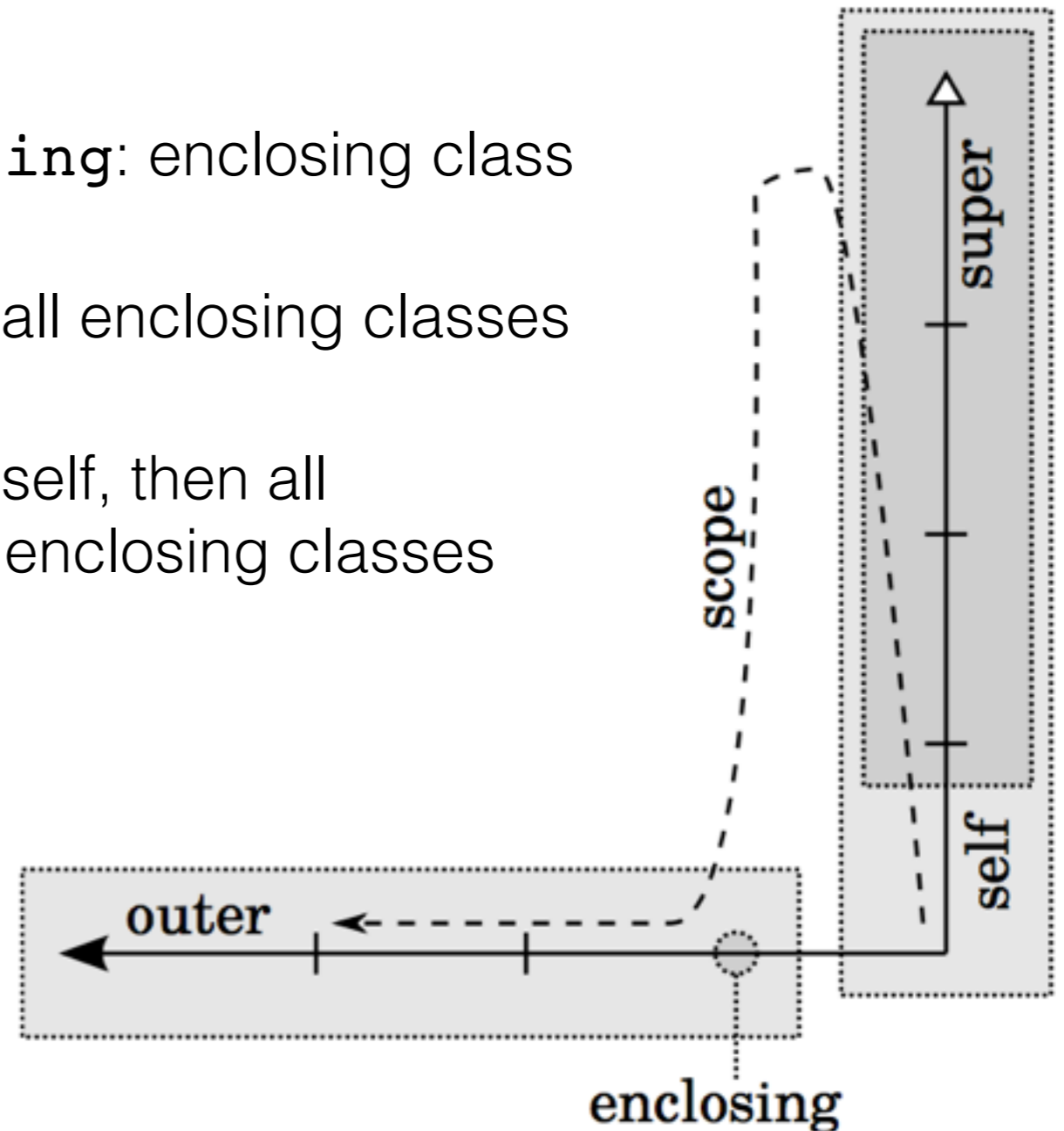


- Nested classes belong to *enclosing class* (like class instance variables)
- Access using message sends
- Nested classes are methods returning the class object

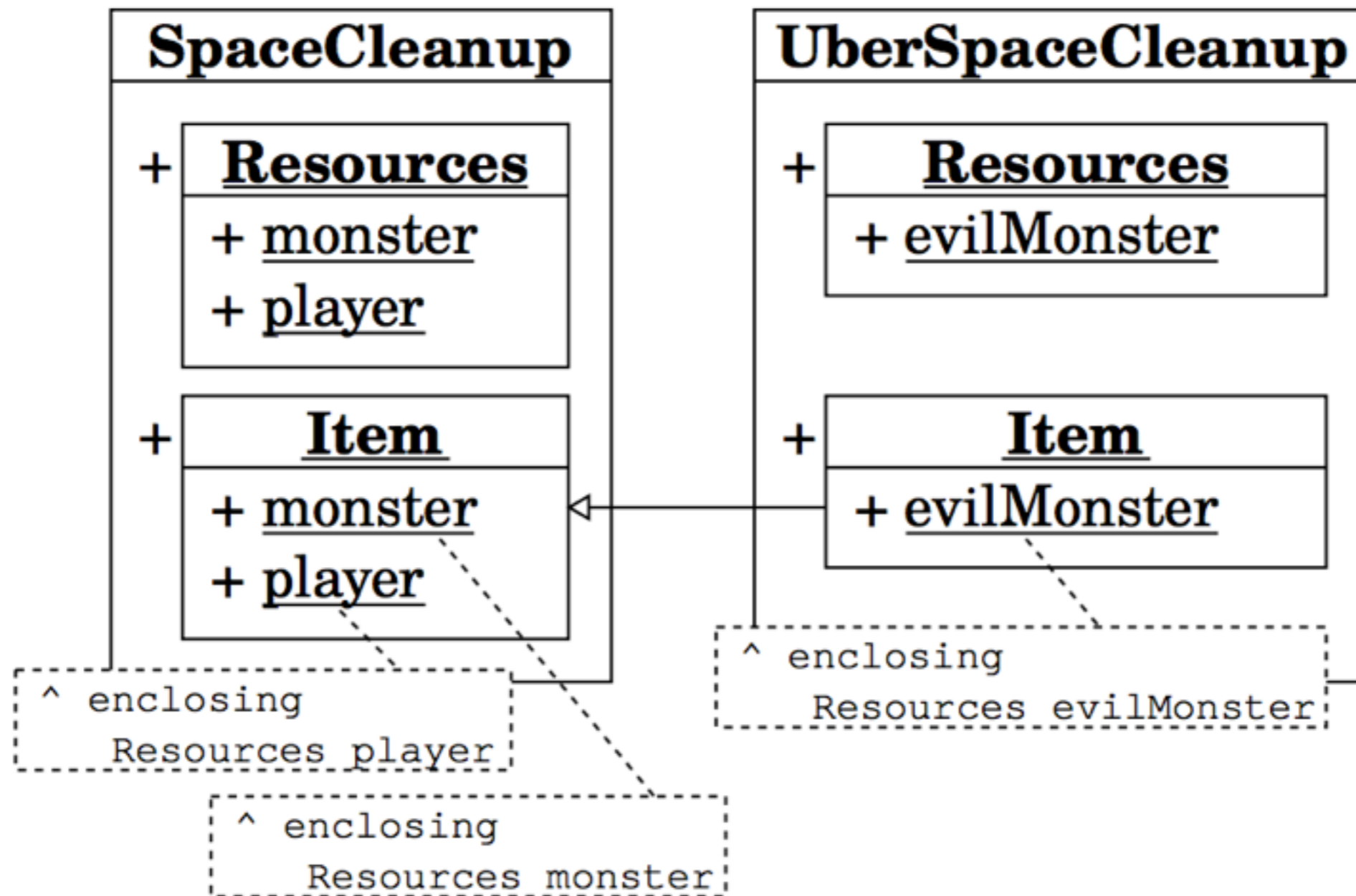
Accessing the Lexical Scope



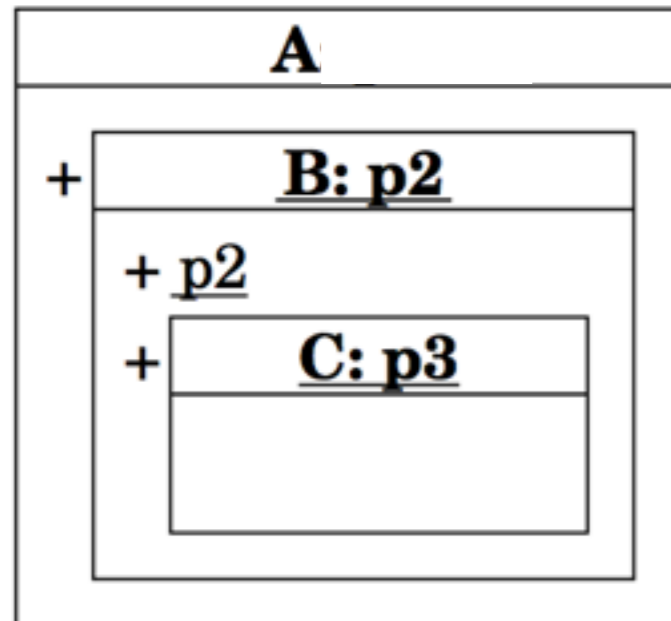
- enclosing: enclosing class
- outer: all enclosing classes
- scope: self, then all enclosing classes



Example: Lexical Scope



Parameterized Classes



(A B: 1) C: 2

scope p2

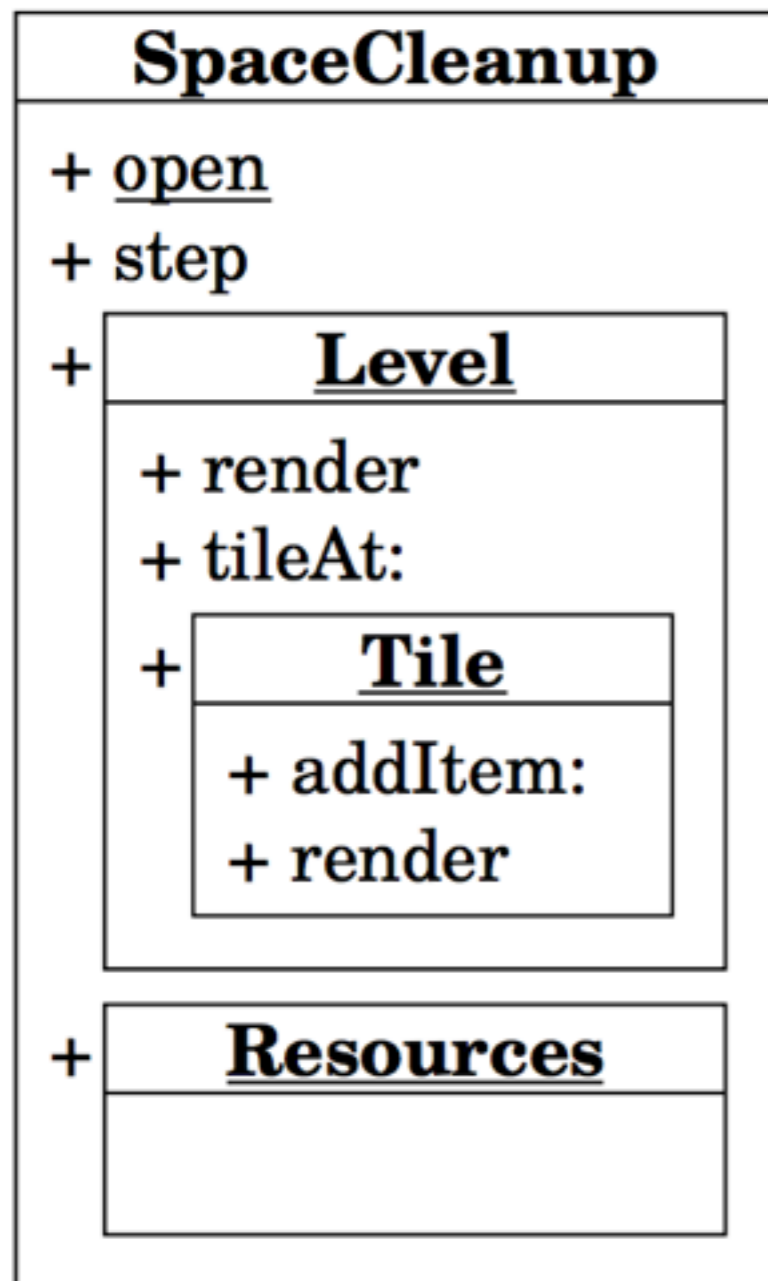
scope p3

Use Cases:

- External Configuration
- Mixins

Implementation

Notation



SpaceCleanup class>>Level

< class >

^ Morph subclass

SpaceCleanup class>>Level>>render

...

Notation

NewClass

< class >

^ Object

subclassWithInstVars: 'foo bar'

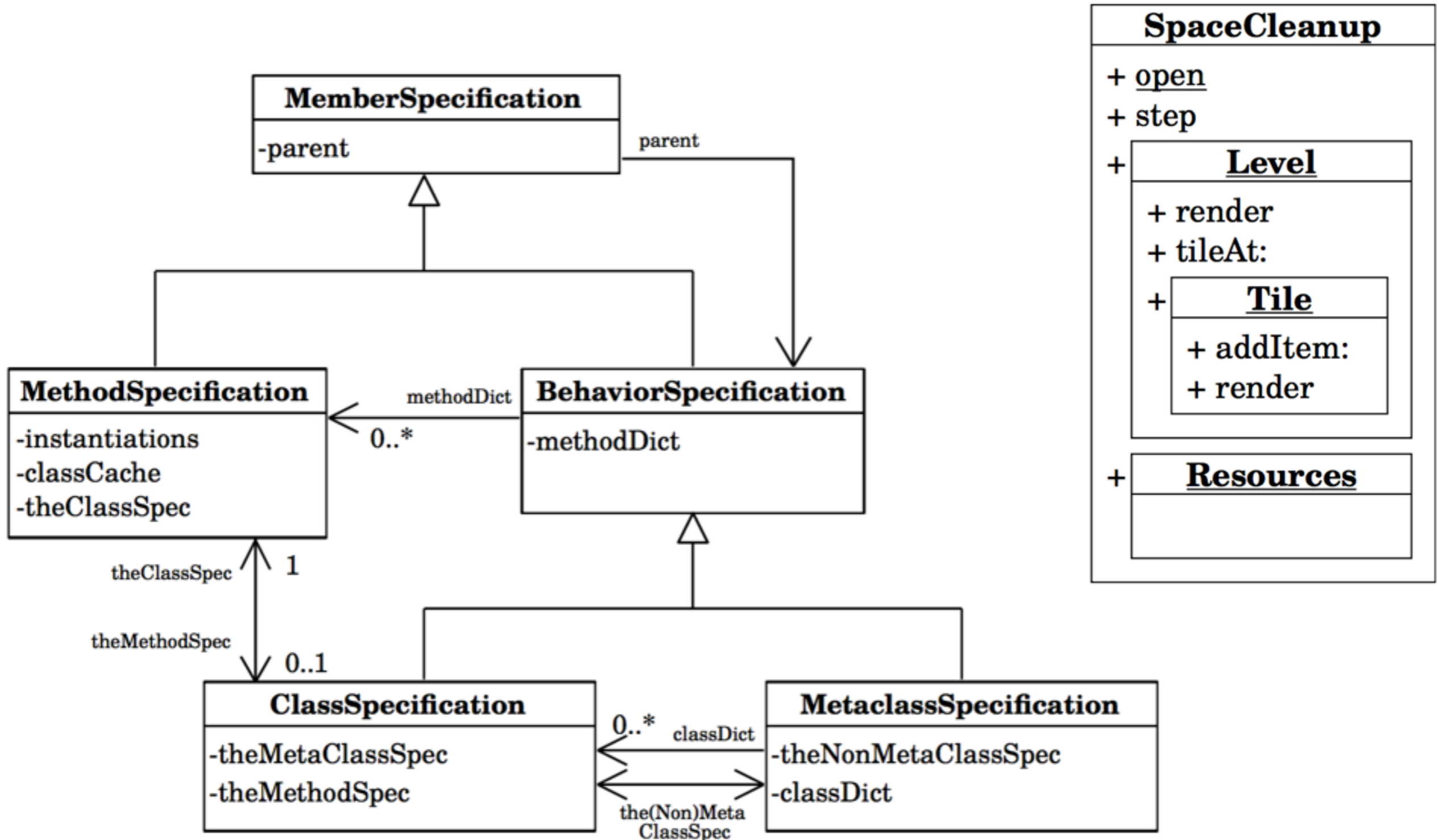
classVars: 'Bar'

classInstVars: 'Foo'

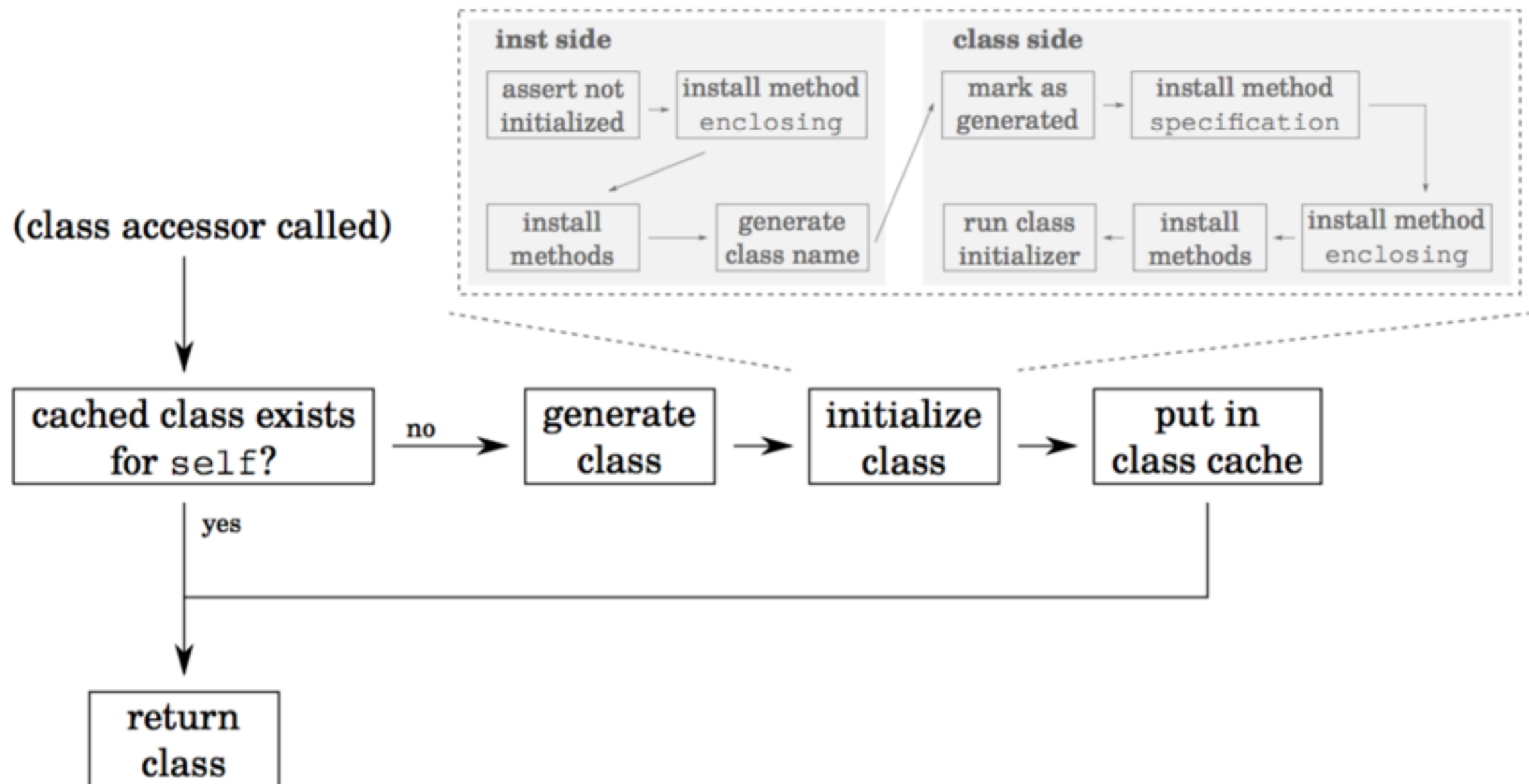
Notation

- *Class generator method*: method which returns the target class for model instantiation
- *Class definition*: target class is uninitialized
- *Class extension*: target class is already initialized

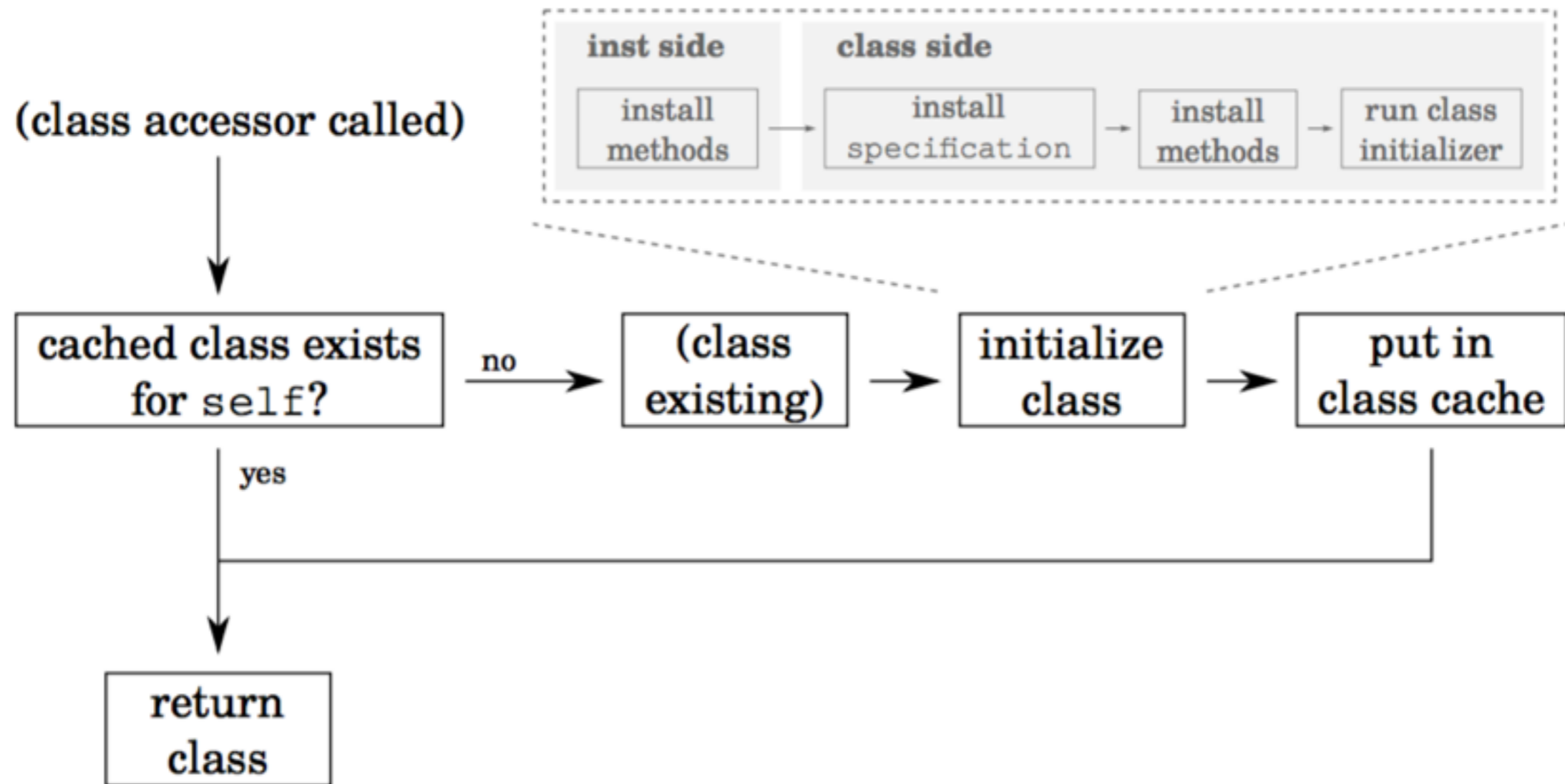
Meta Model



Class Definition

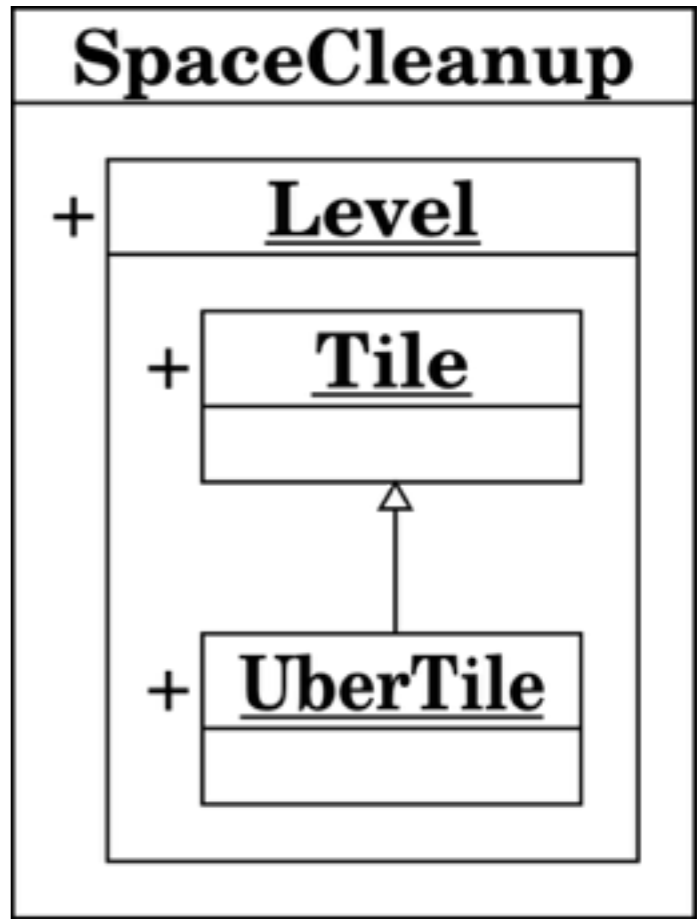
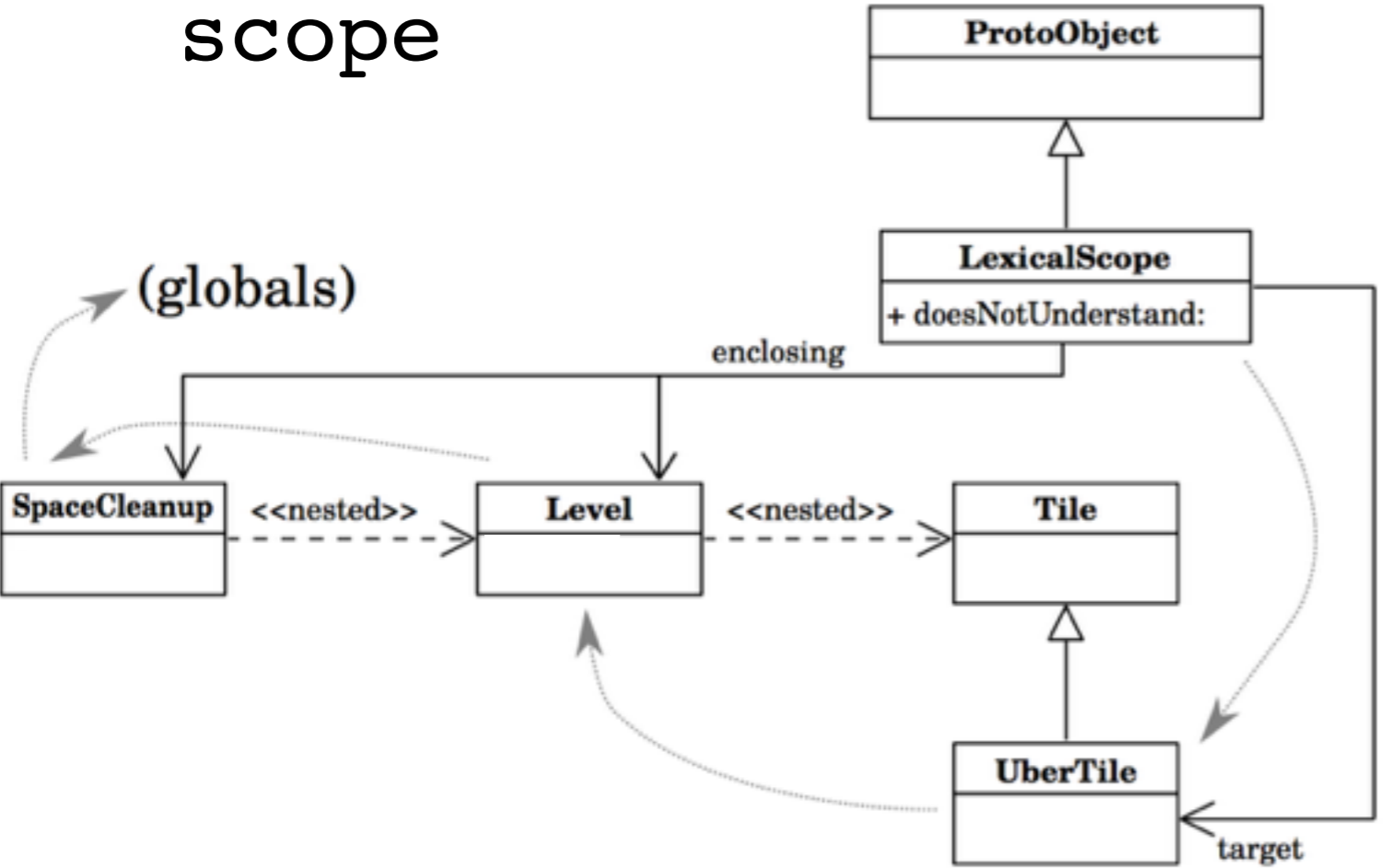


Class Extension



Keywords

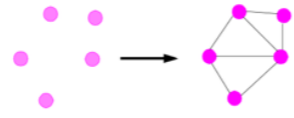
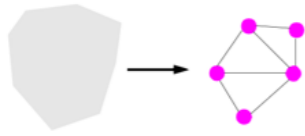
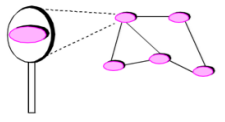
scope



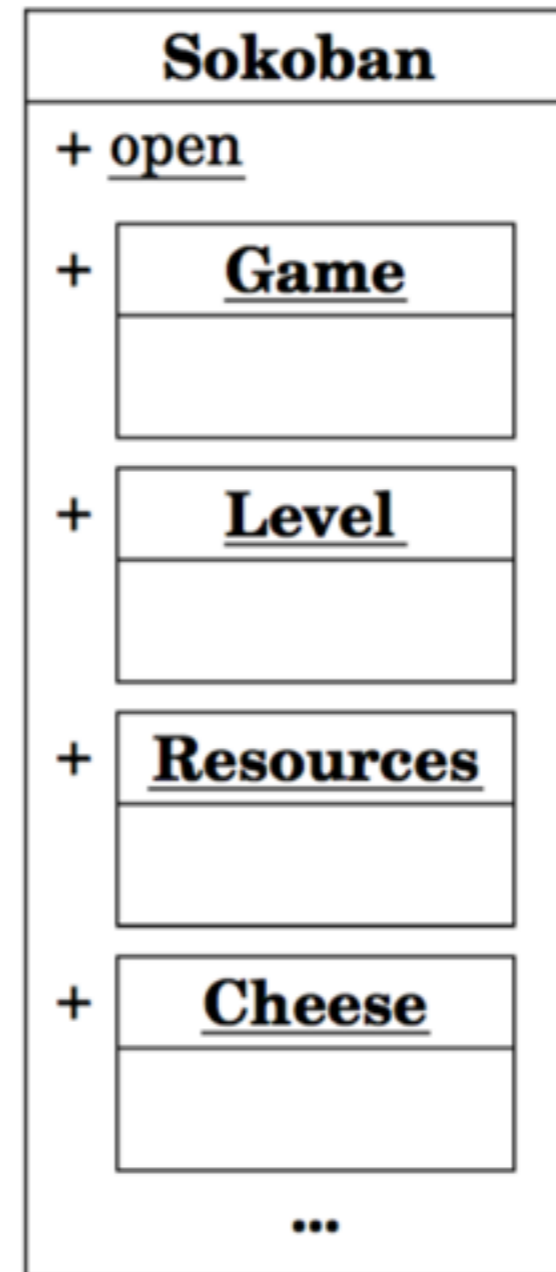
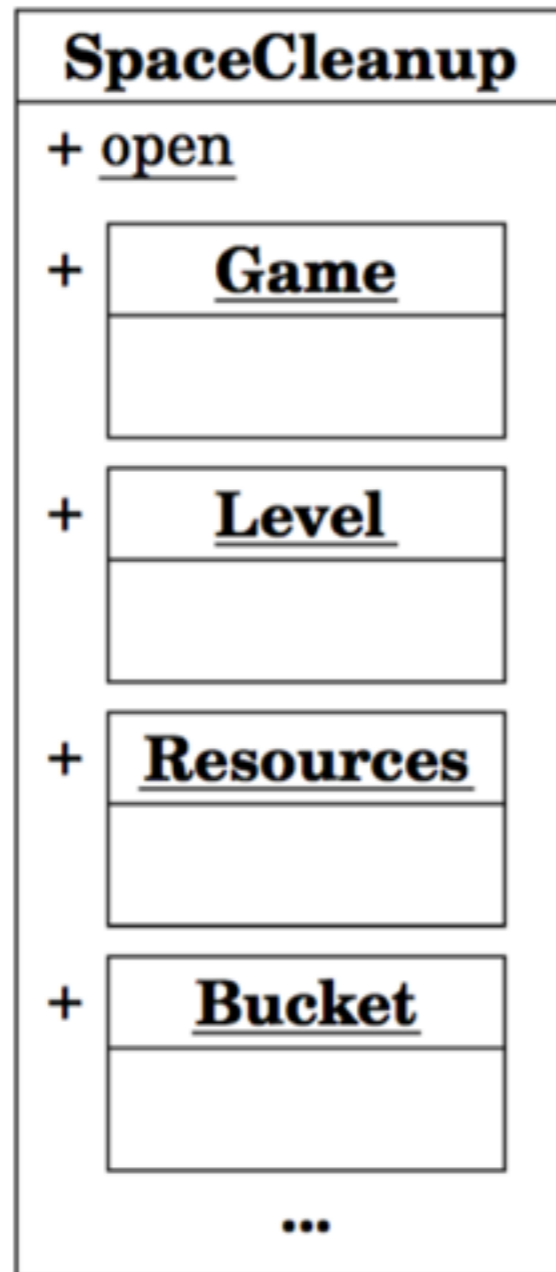
Implicit `LexicalScope` vs. Squeak Environments

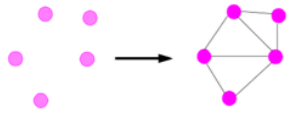
- *LexicalScope*: late-bound lookup
- *Squeak environments*: early-bound lookup
- Late-bound lookup makes it easier to react to structural changes/source code changes
- Parameterized classes cannot be early bound

Use Cases

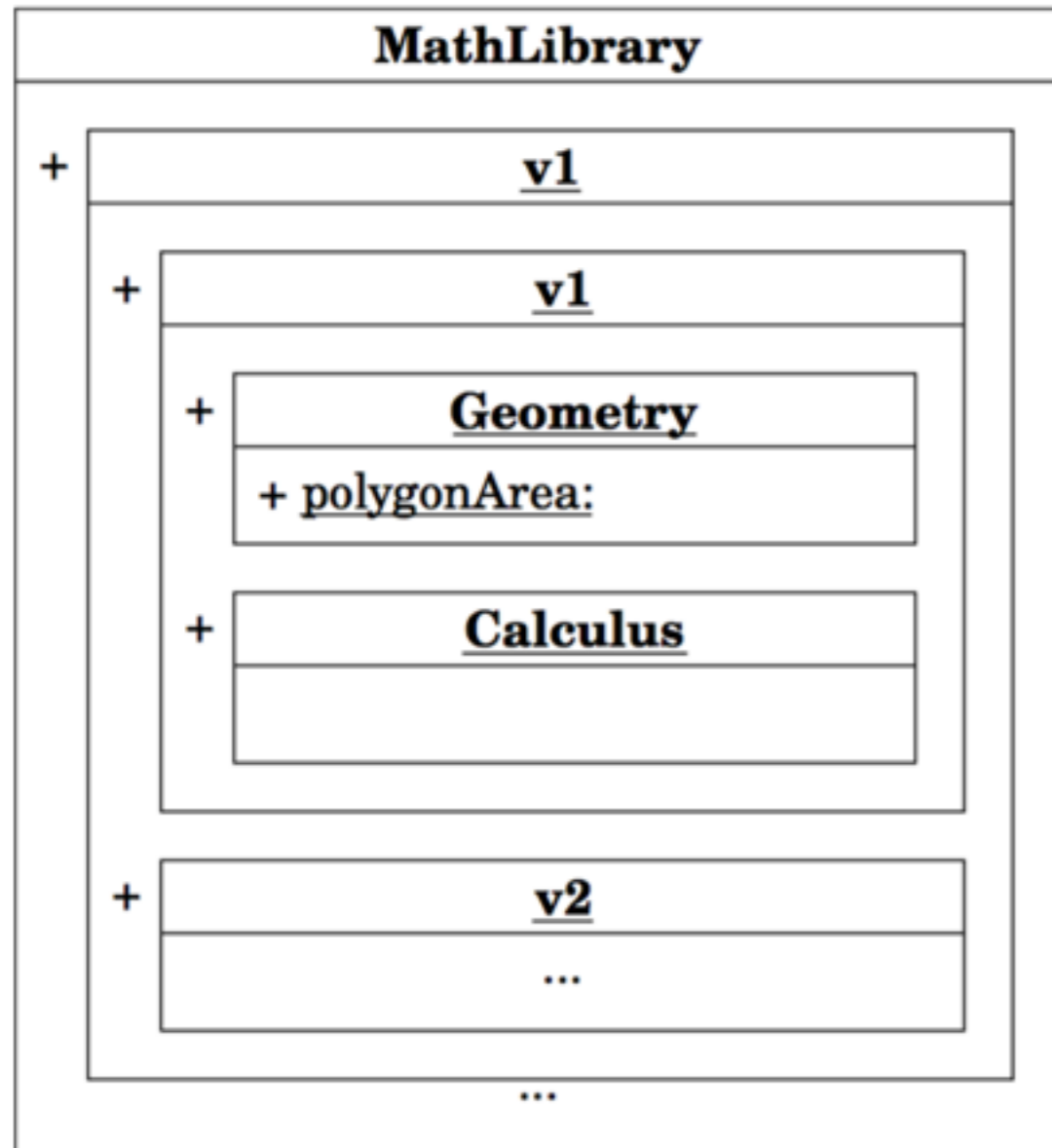


Duplicate Class Names



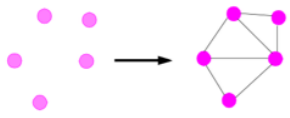


Versioning

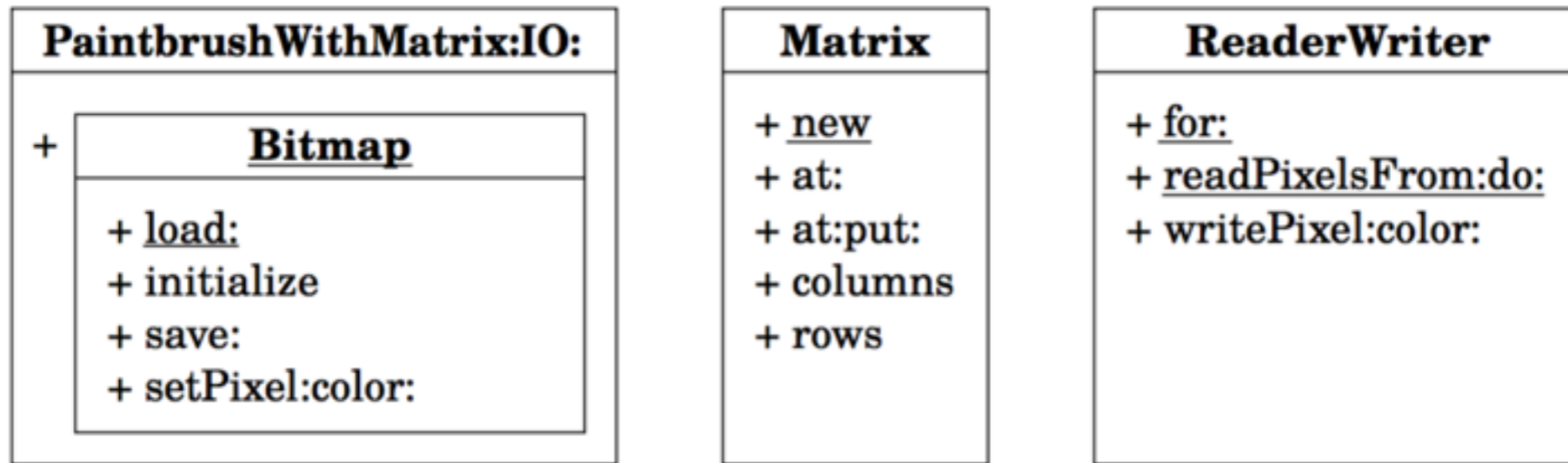


```
MyApp class »MathLib
    ^ MathLibrary v1 latest
```

```
MyApp class »...»...»Geometry
    ^ scope MathLib Geometry
```

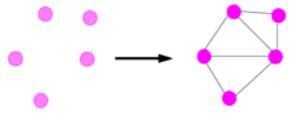


External Configuration

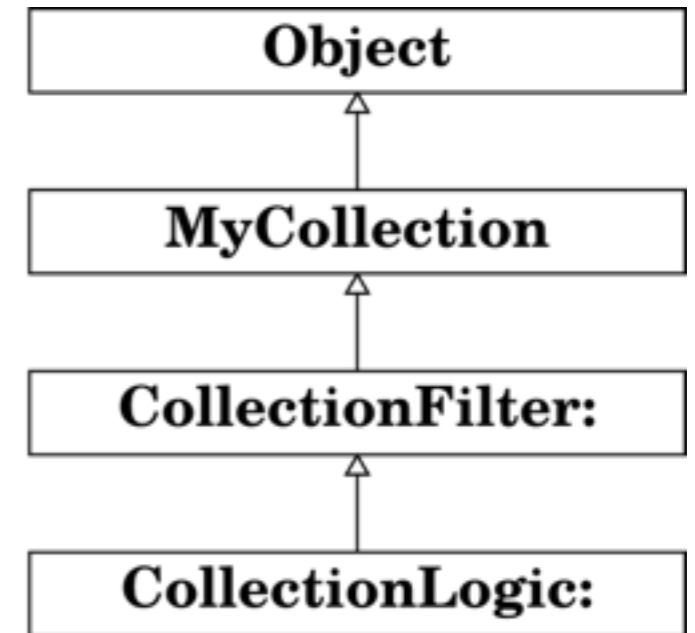
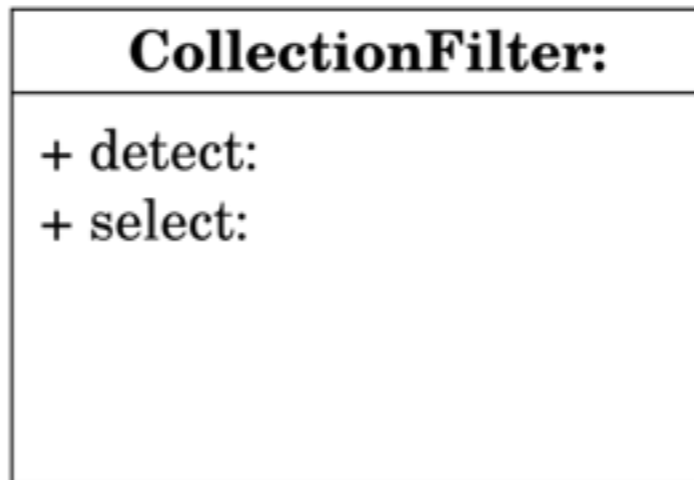
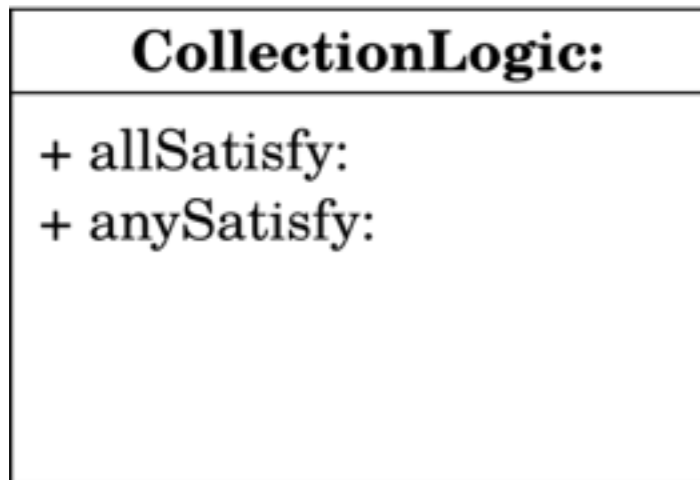


```
(PaintbrushWithMatrix: Matrix IO: ReaderWriter) class»Bitmap class»load: aFile
```

```
| instance |
instance := self new.
scope ReaderWriter
    readPixelsFrom: aFile
    do: [ :point :color | instance setPixel: point color: color ].
^ instance
```



Mixins



```
MyCollection»do: aBlock
```

```
...
```

```
CollectionFilter: base
```

```
< class >
```

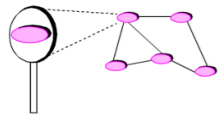
```
^ base subclass
```

```
(CollectionFilter: base)»detect: aBlock
```

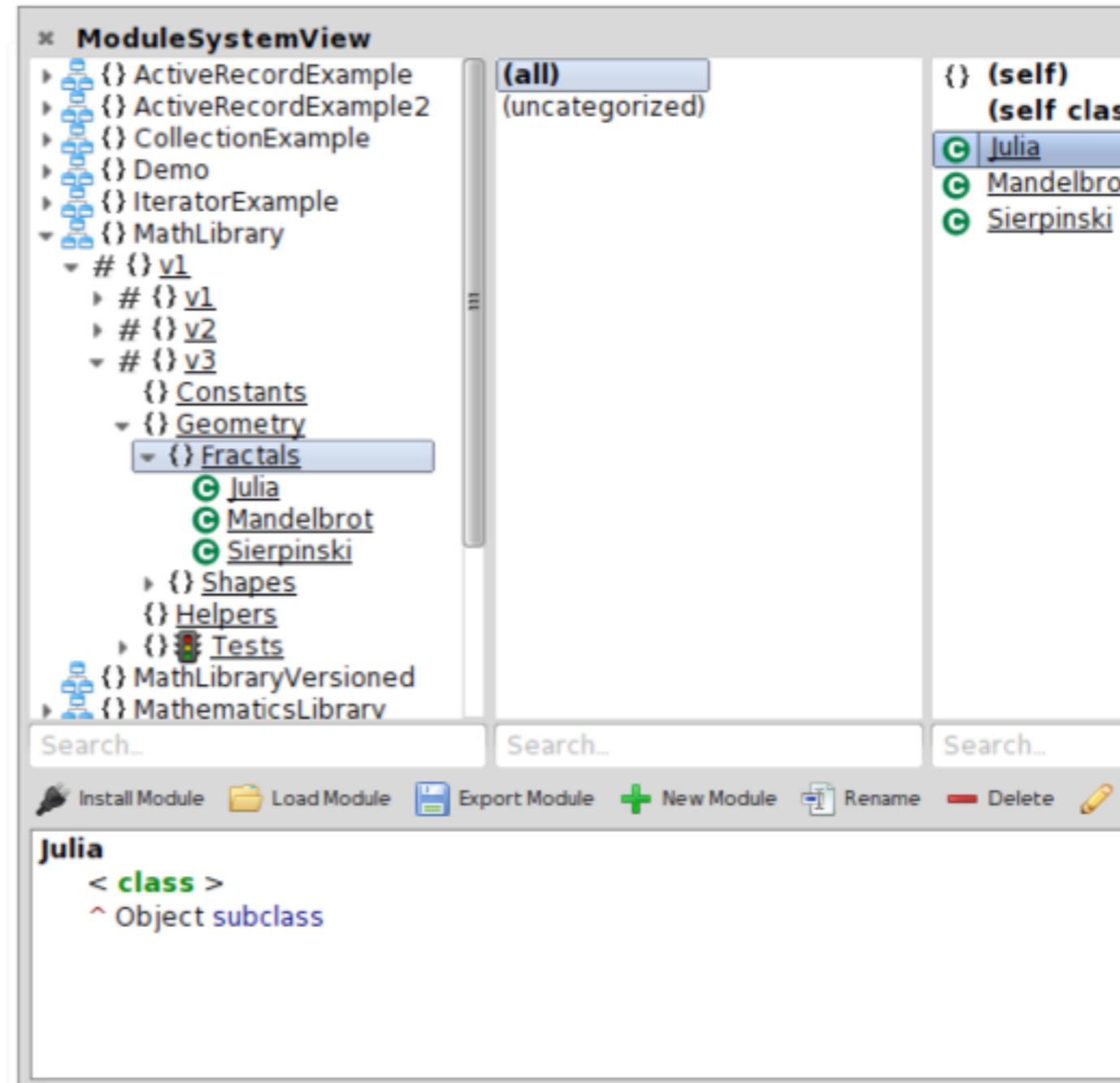
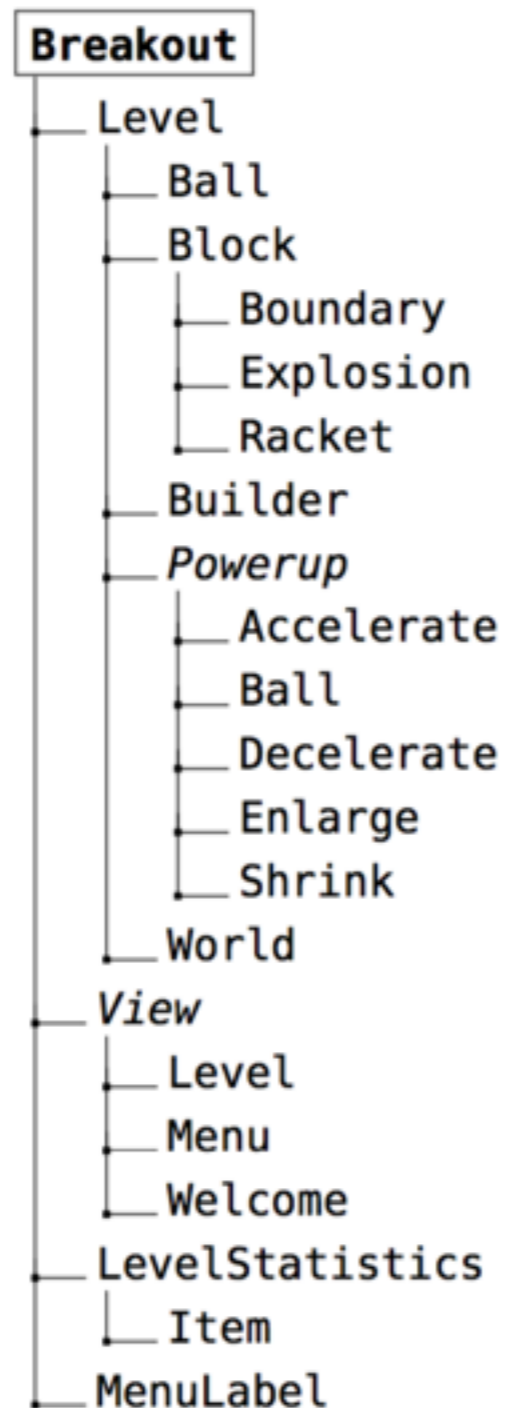
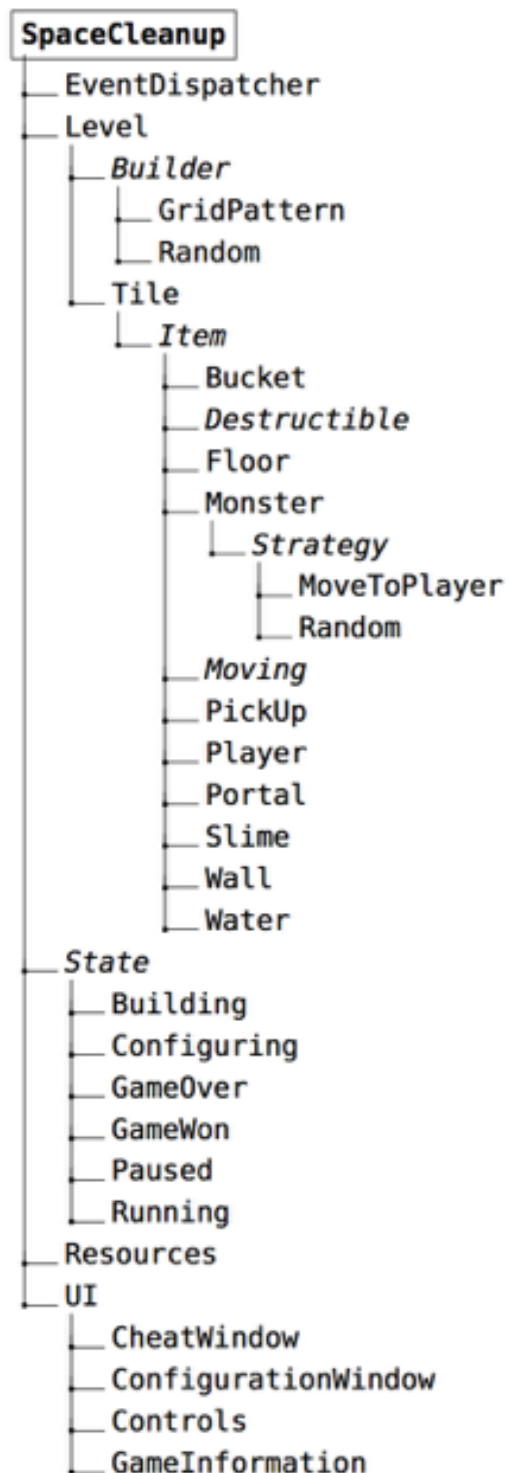
```
self do: [ :el | (aBlock value: el) ifTrue: [ ^ el ] ].
```

```
self error: 'element not found'
```

Mixin = abstract subclass /
class transformer function



Hierarchical Decomposition



Future Work

Future Work

- Performance (byte code transformation)
- Squeak integration + GUI (browser)
- Extension methods
- Migration of legacy code

Related Work

Related Work

- *Duplicate Class Names:*
Packages / Namespaces (VisualWorks, Java, Ruby),
Squeak environments, Newspeak modules
- *Class Nesting:*
Newspeak, BETA, Java, Ruby, Python
- *Dependency Management:*
Newspeak, Maven, RubyGems, pip, Metacello
- *Parameterized Classes / Mixins:*
Java generics, C++ templates, Newspeak, Ruby, Python,
Traits

Summary

- *Matriona*: a module system for Squeak based on class nesting
- “Design Principles Behind Smalltalk” (D. H. Ingalls)
 - *Personal Mastery*: entire system should be comprehensible by a single individual
 - *Factoring*: each independent component appears only once
 - *Modularity*: no component should depend on internal details of another component
 - *Good Design*: system should be built with a minimum set of unchangeable parts, which are as generic as possible