

A New Architecture

Reconciling Refactorings and Transformations

Balša Šarenac, Stéphane Ducasse and Nicolas Anquetil



Definitions

- Transformation
- Refactoring
- Precondition
- Atomic refactoring
- Composite refactoring

Imagine a world with only refactorings

- Then you have to do your transformations by hand
 - Override an existing method

Imagine a world with only transformations

- Then you can **break** your systems with just adding a method (as in VSCode :))

**We need both refactorings
and transformations**

Example

There are 10 methods calling privateTransform

Select a strategy

- ☐ Don't remove, but show me those senders
- ☐ Remove, then browse senders
- ☐ Remove it

Cancel Accept

Code example

RBInlineMethodRefactoring >> preconditions

▪ ▪ ▪ ▪

```
self isOverridden ifTrue: [
```

```
  self refactoringWarning:
```

```
    ('<1p>>><2s> is overridden. Do you want to inline it  
    anyway?')
```

```
      expandMacrosWith: self classOfTheMethodToInline
```

```
      with: self inlineSelector) ] ].
```

▪ ▪ ▪ ▪ ▪

Transformations AND Refactorings

- G. De Souza Santos defined Transformations and CompositeTransformations
- But a lot of code duplication
- Difficult to understand when using what
- What about preconditions?

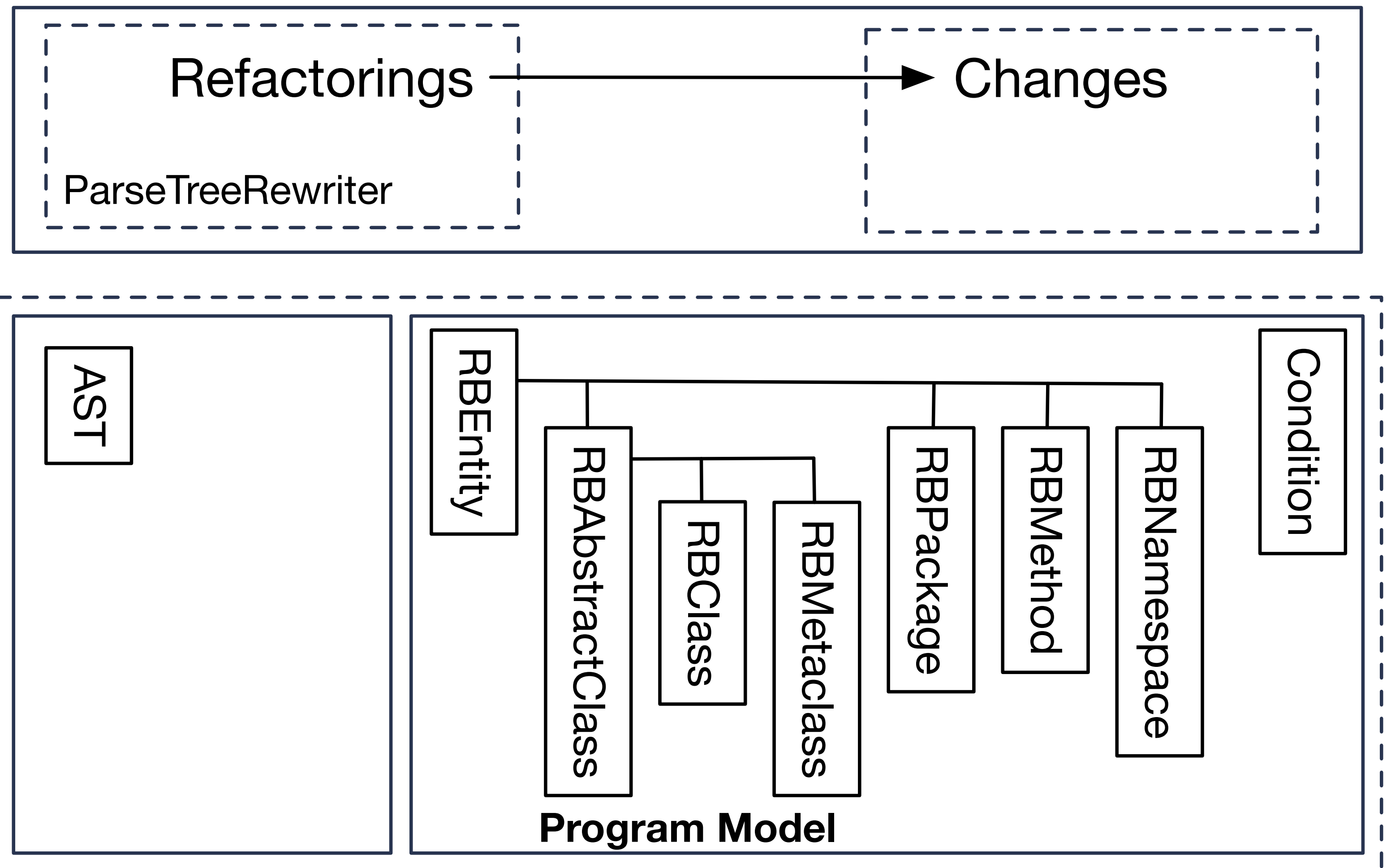
Goals of our **engineering** effort

- Modernise our engine
 - Reduce code duplication
 - Cleaner code
 - More tests
 - Assess refactorings (clear/correct preconditions / semantics)
 - Usability issues

Goals of our **scientific** work

- Reuse of transformations and refactorings to form new ones
- Understand composition issues (ongoing)
- Our ultimate goal is
 - Support you to write **your own** transformation
 - Domain specific refactoring definitions

A solid basis



- A refactoring reasons on a **program model**
- **Check preconditions** on such a program model
- Produces first class **changes** that can be **previewed**
- **Then and only then** actual modifications are done

A kind atomic approach

- A refactoring reasons on a program model
- Check preconditions on such a program model
- Produces first class changes that can be previewed
- **Then if something fails you can not apply the modifications**

About Preconditions

**Do you think that transformations
needs preconditions?**

“Transformation-based Refactorings” [IWST22] shows that there are different kinds of preconditions



Different kind of Preconditions

See IWSST 22

- Applicability
- Breaking change
- Skipping
- Others

applicabilityPreconditions

RBAddMethodTransformation >> applicabilityPreconditions

- ▪ ▪ class should exist ▪ ▪ ▪
- ▪ ▪ method should be parsable ▪ ▪ ▪

breakingChangesPrecondition

RBAddMethodRefactoring >> applicabilityPreconditions

. . . method shouldn't be overridden . . .

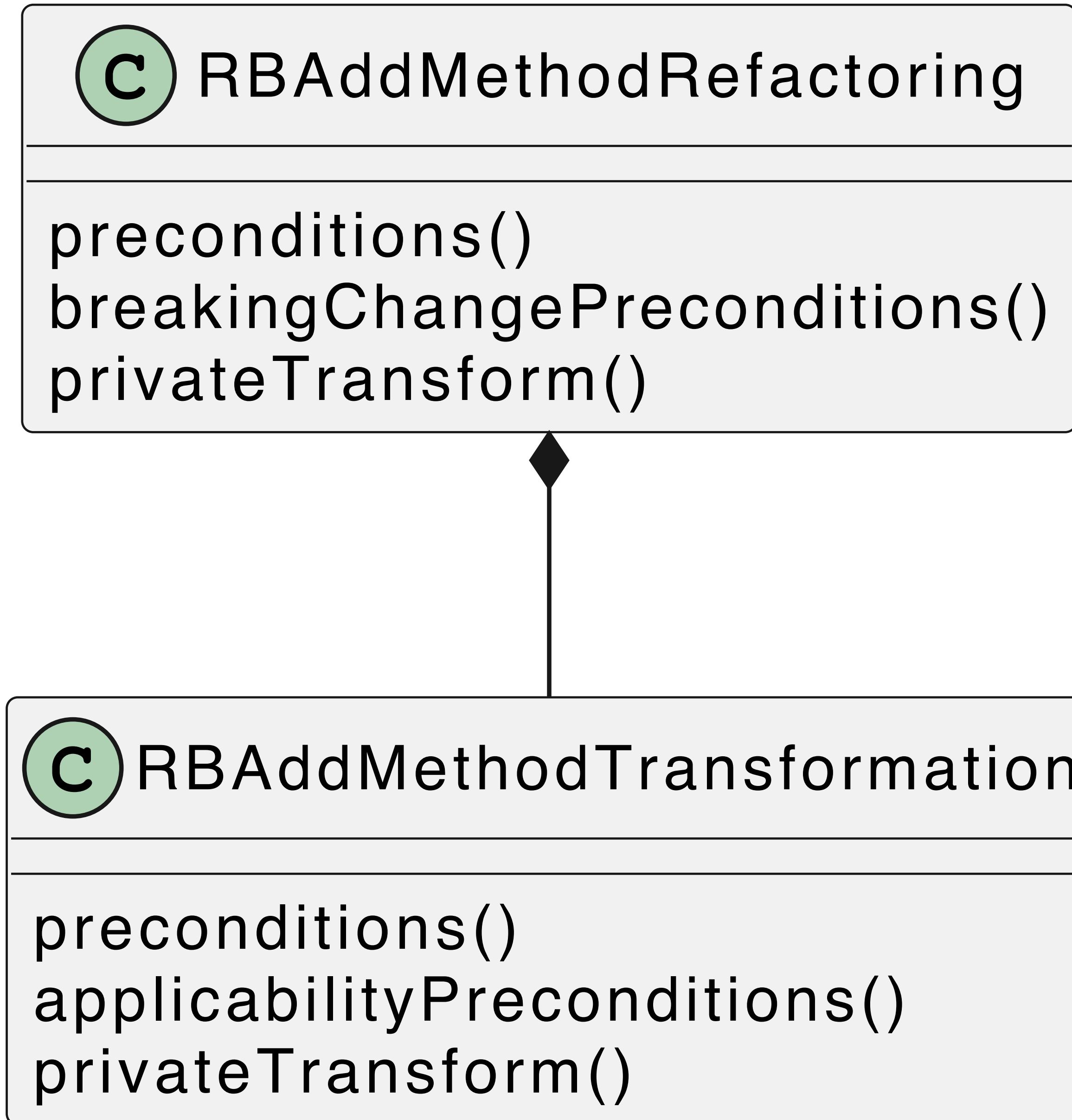
About Reuse

Clear separation

- Refactorings have breaking changes preconditions
- Transformations have applicability preconditions

Example

Add method



Example

Add method

RAddMethodRefactoring >> **preconditions**

`transformation` checkPreconditions.

`^ self` breakingChangePreconditions

RAddMethodRefactoring >> **privateTransform**

`transformation` privateTransform

RAddMethodTransformation >> **privateTransform**

`self` definingClass

compile: `sourceCode`

classified: `protocol`

Refactorings are decorators for transformations

RBAddMethodRefactoring —> RBAddMethodTransformation

- Refactoring uses Transformation to check applicability conditions
- Refactoring checks breaking change conditions
- Refactoring uses Transformation to make changes
- [Refactoring does cleanups and fixes if needed]

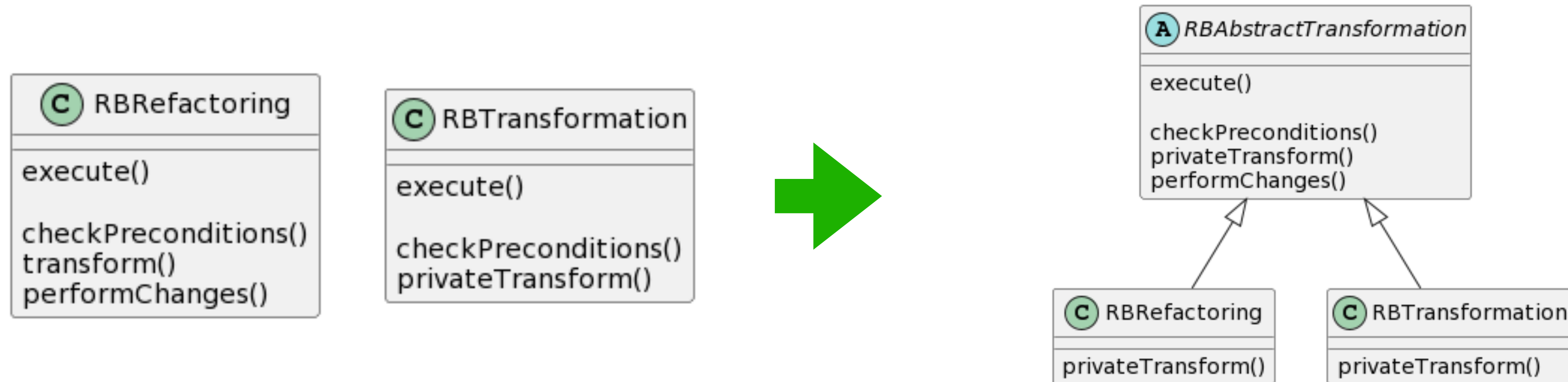
State of situation

- We are in the process of converting all the implementation to this design

About engineering

Realigning transformations and refactorings

- Better API
- Partial instantiation of refactorings to support better interaction
- Moving more responsibilities to refactorings

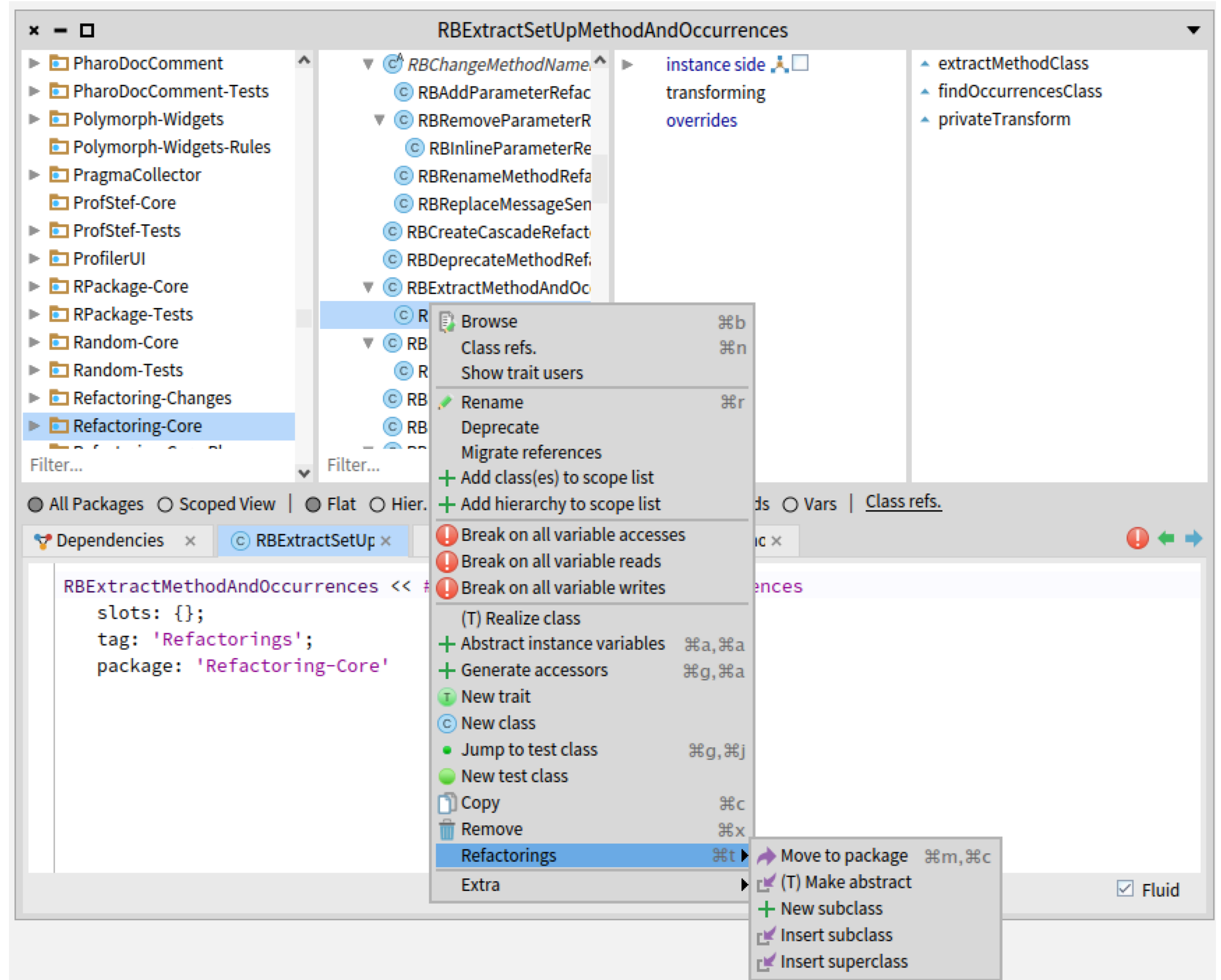


Revisit preconditions

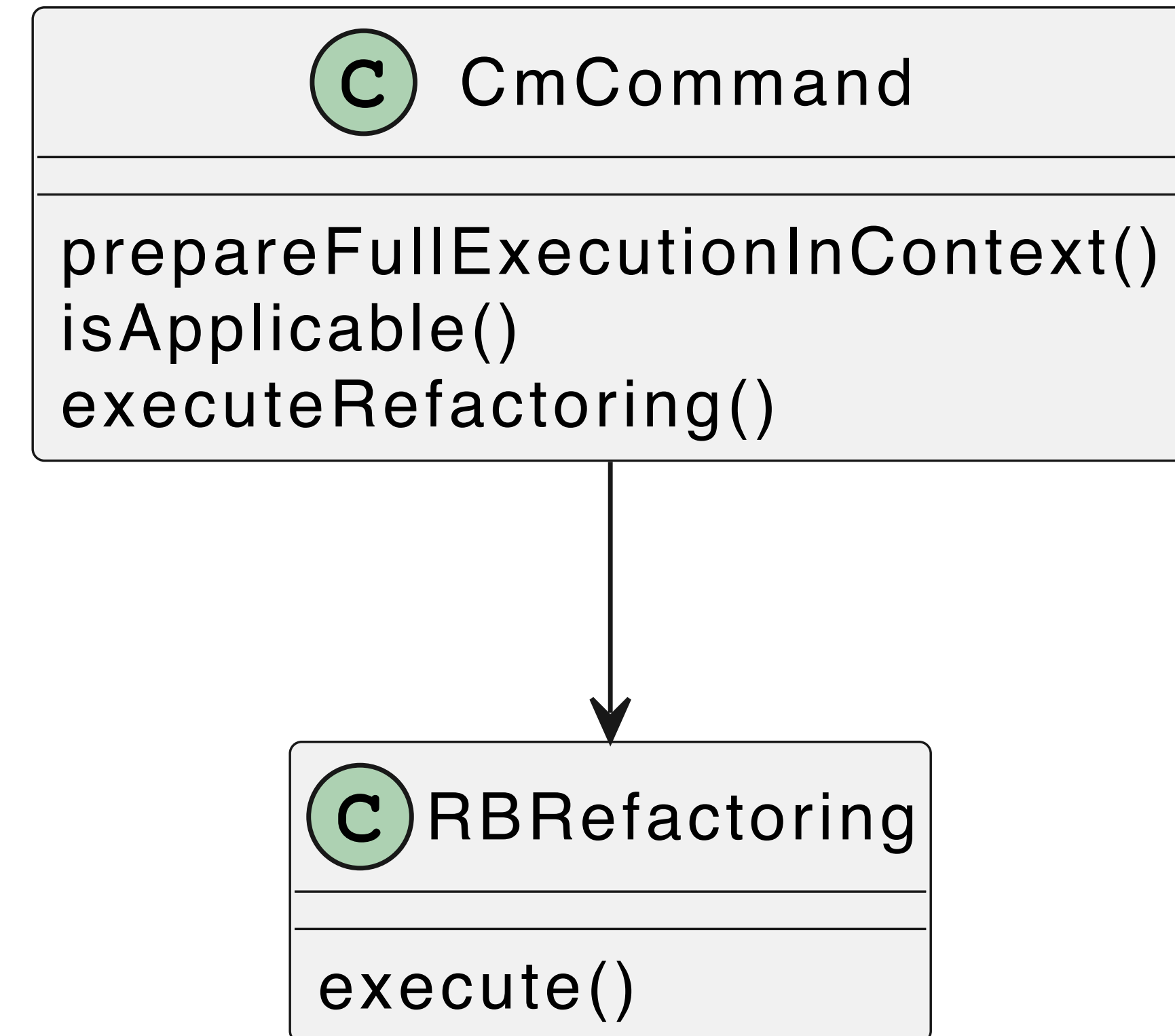
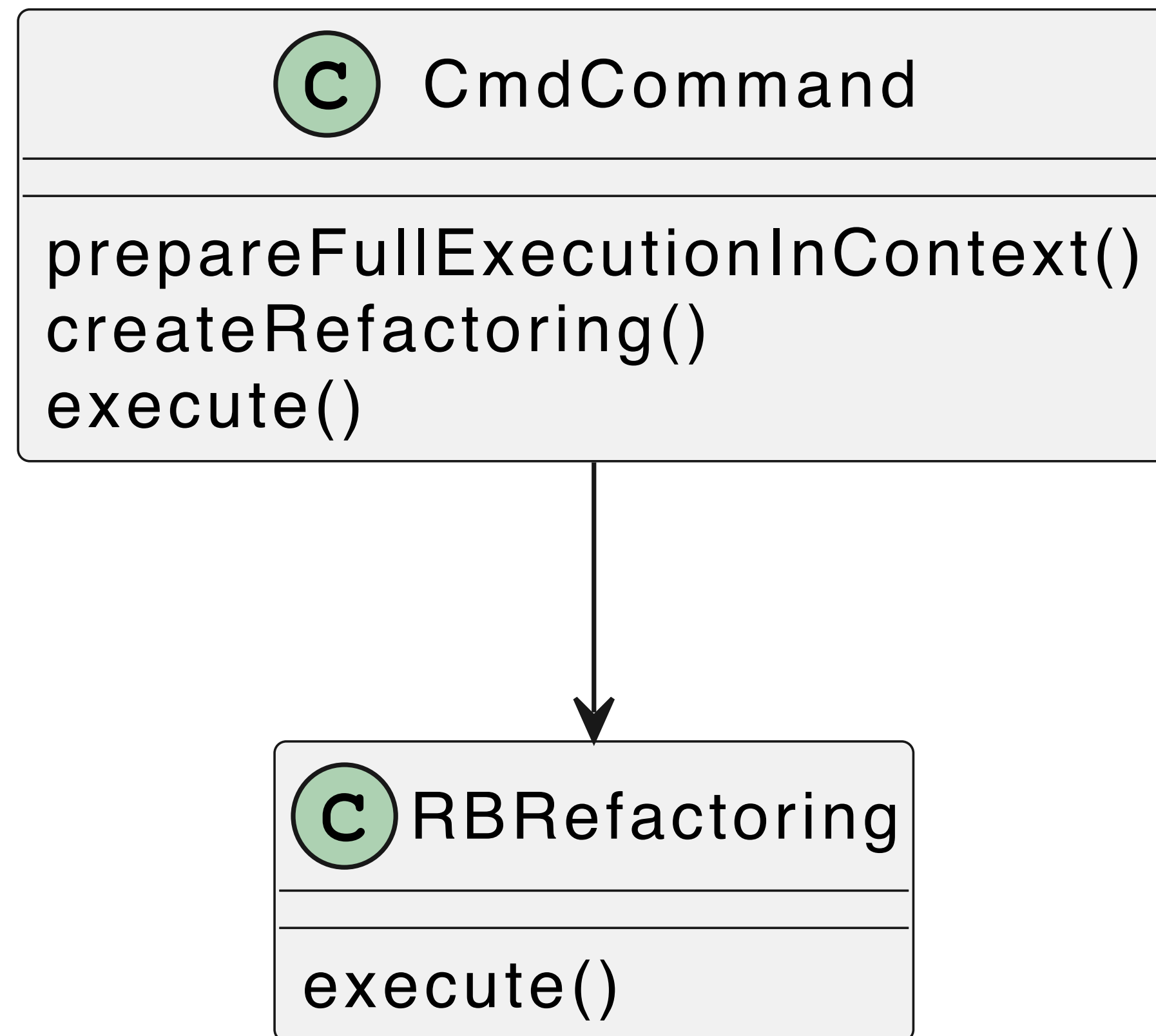
- Some preconditions were obscure / wrong
- Clearly identify breaking and applicability preconditions
- Adding a lot of comments
- Fixing, enhancing tests

About the (T)

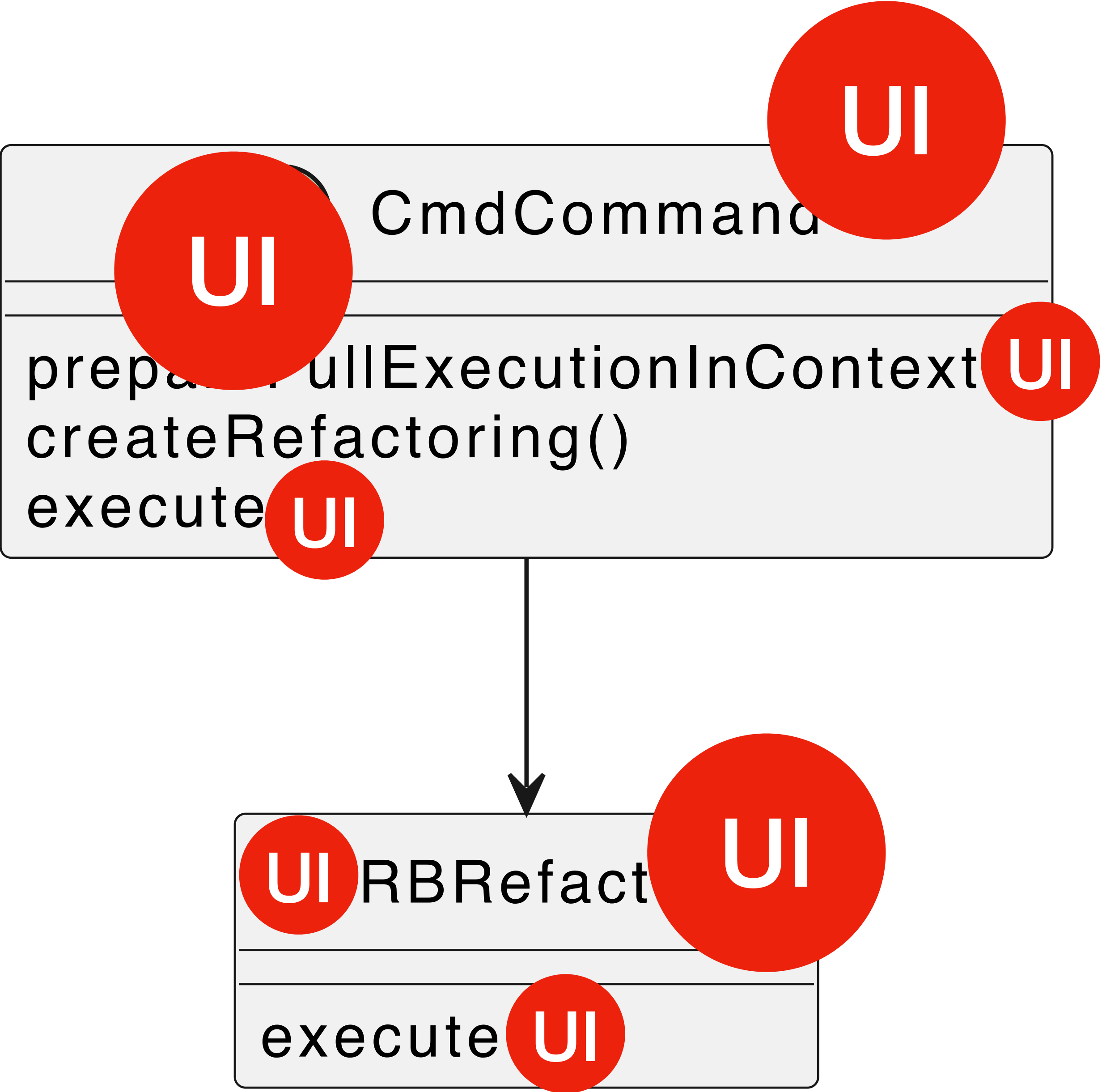
- You are warn when you use a Transformation



Cmd to Cm2.0



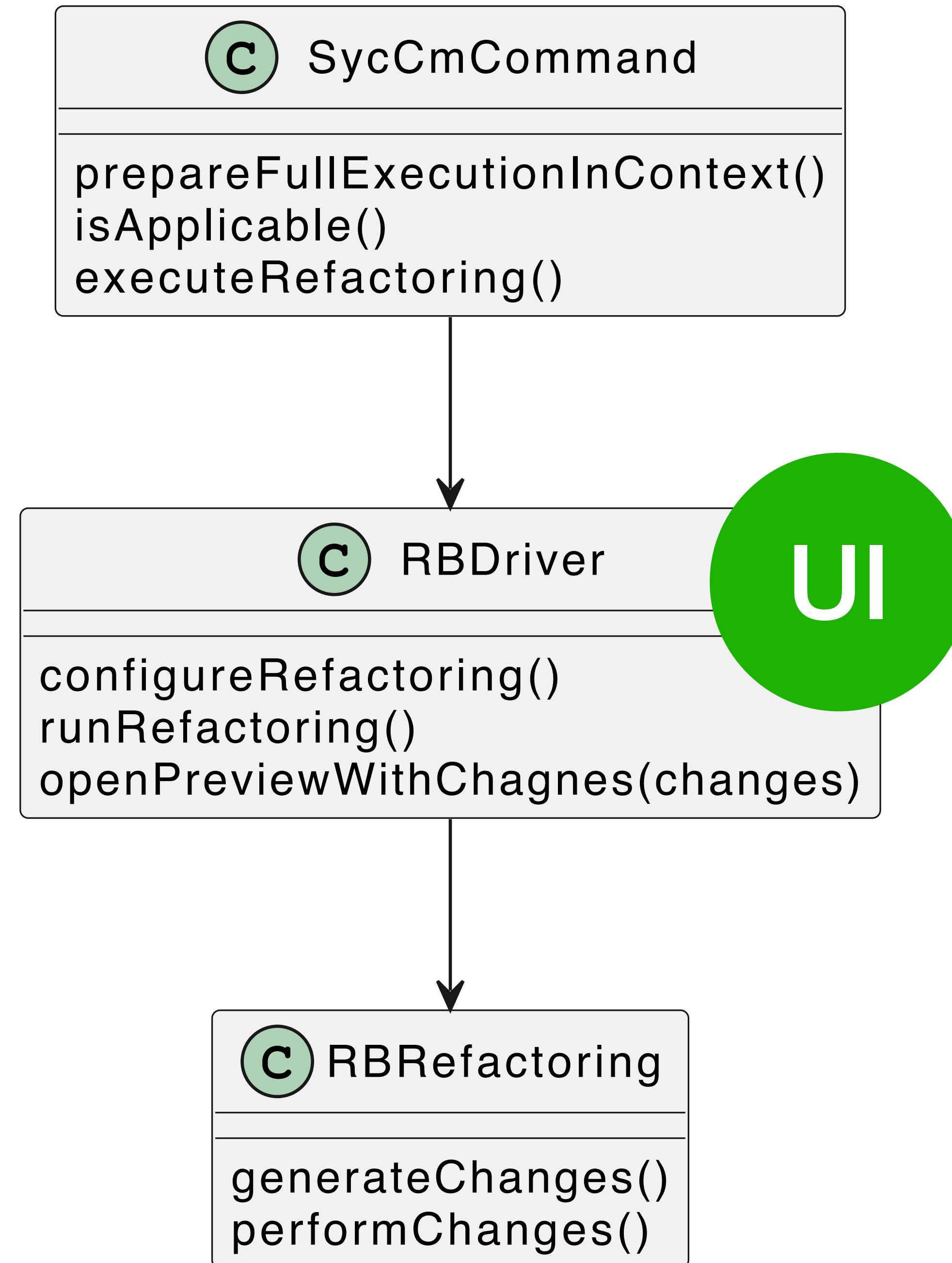
About UI



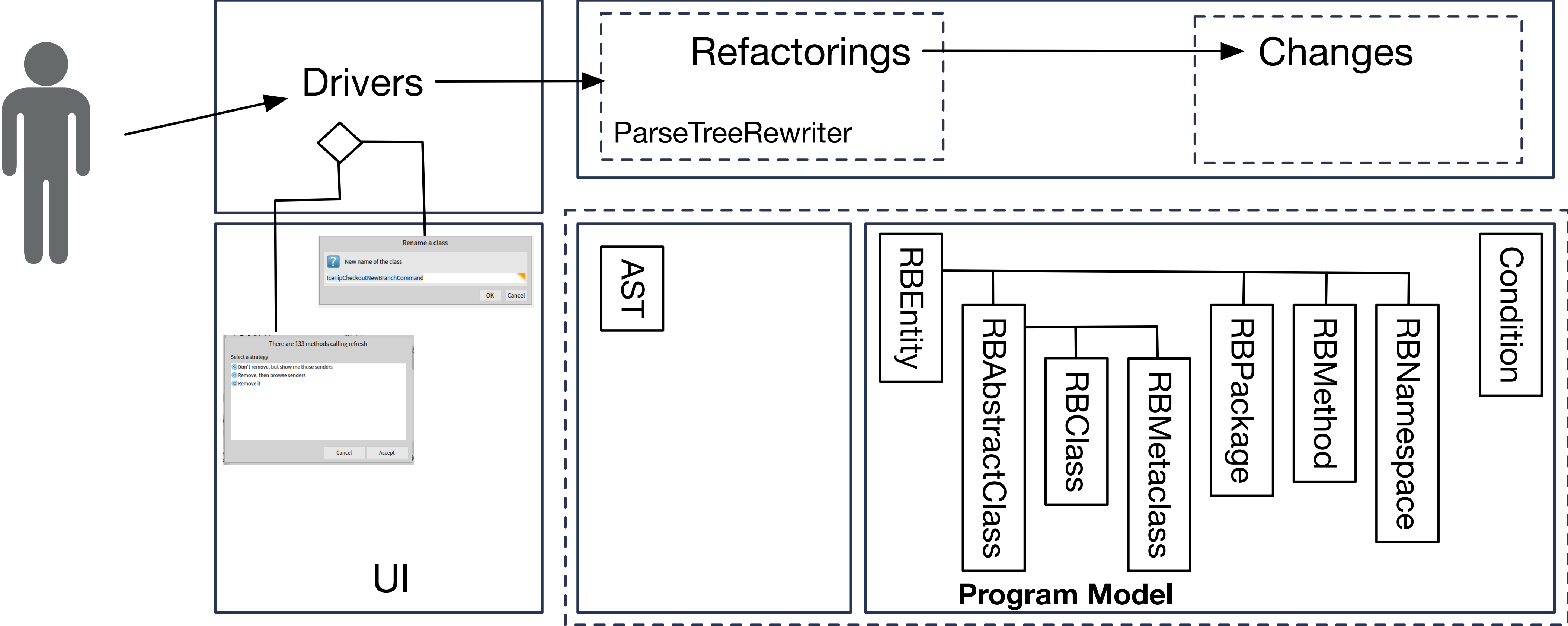
Preconditions should not raise UI!

- Preconditions had a lot of UI like:
 - Gather user input
 - Raise warnings
 - Show confirmation dialogs

New Tooling



New Architecture



Driver

- ▶ UI is Driver's responsibility now
 - Configures refactorings
 - Gathers user input
 - Displays errors and warnings
 - Displays any other relevant information (notifications, browsers, etc.)

Open questions

- Do we keep warning and exceptions
- Why not having failing reports that can be nicely displayed

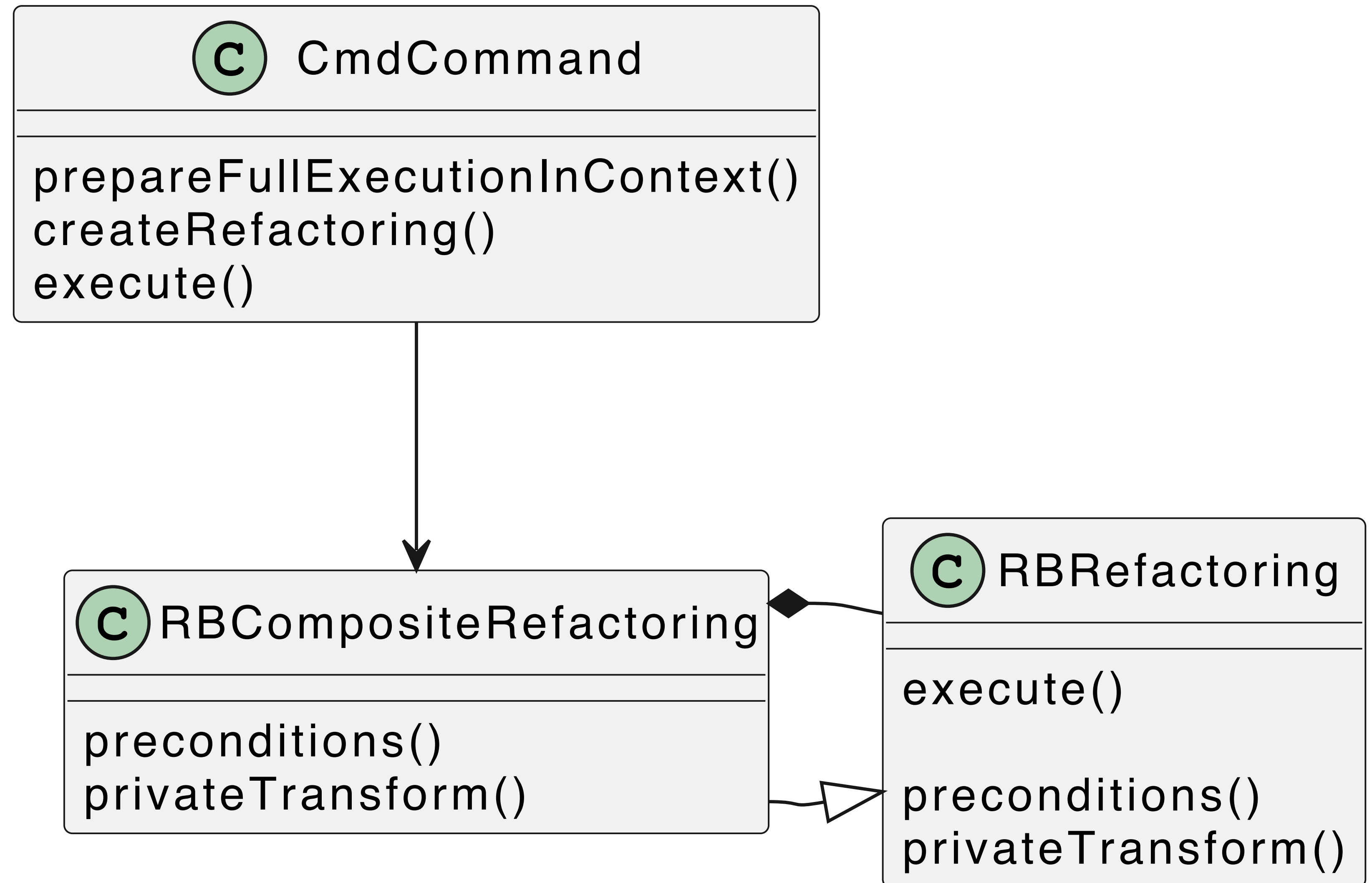
About Composition

(Started recently)

Existing Refactorings are monolithic

- G. De Souza Santos started to define more modular transformations (RBCompositeTransformation)
- We introduced
 - RBCompositeRefactoring
- Starting to play with composition semantics :)

Composite



Let us study RemoveInstanceVariables

```
refactoring := RBCompositeRefactoring new
  model: model;
  refactorings: (variables collect: [:each |
    RBRemoveInstanceVariableRefactoring
      model: model
      remove: each
      from: class]);
  yourself
```

RBCompositeRefactoring

- Execute in sequence refactorings
- P1, T1; P2, T2; ... Pn Tn

```
RBCompositeRefactoring >> privateTransform
```

```
    refactorings do: [ :each | each generateChanges ]
```


Different execution semantics

- Stop on failure (as RBCompositeRefactoring)
- Skip failed and proceed (as RBCompositeContinuingRefactoring)

RBCompositeContinuingRefactoring >> privateTransform

```
refactorings do:
    [ :each |
        [ each generateChanges] on: RBRefactoringError do: [ :ex | ]
    ]
```

Custom composite example

Can we remove both fooUnik and barUnikUnik?

```
X >> fooUnik  
    ^ 12
```

```
X >> barUnikUnik  
    ^ self fooUnik + 1
```

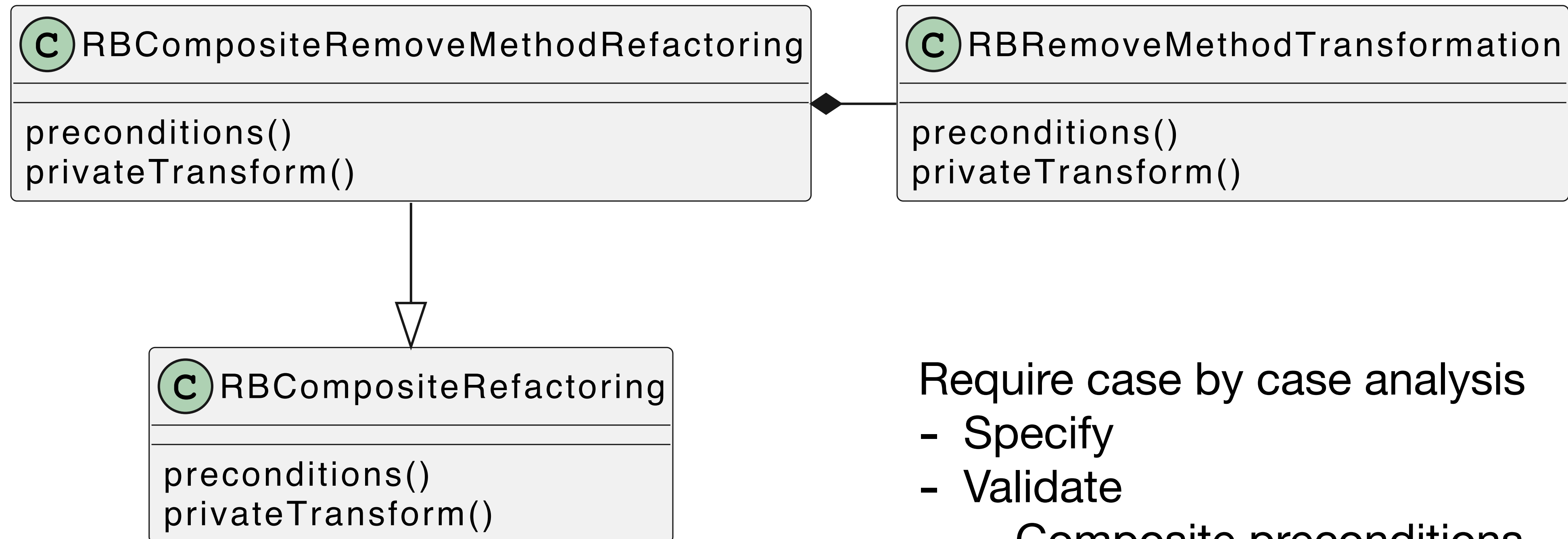
Custom composite need

removeMethods (fooUnik, barUnikUnik)

is not equals to

**removeMethod (fooUnik);
removeMethod (barUnikUnik)**

Custom composite



Require case by case analysis

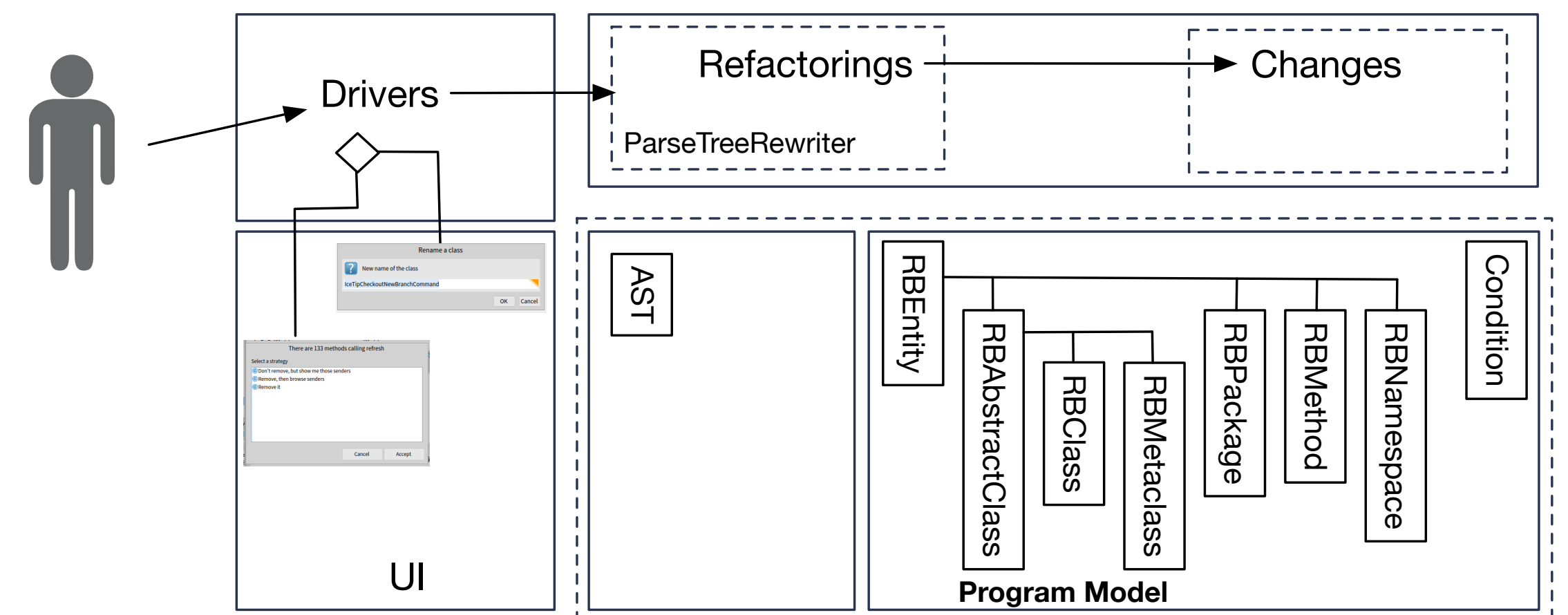
- Specify
- Validate
 - Composite preconditions
 - Component preconditions

Future work: a large effort

- Continue eliminating code duplication between refactorings and transformation
- Leverage composition of refactorings where possible
- Migrate to Commander2.0
- Migrate all UI to Driver
- A lot more to...

New architecture for the future :)

- Many many hidden improvements
- Driver for interactive application
- Clear roles for Transformations and Refactoring:
 - A refactoring is a decorator of a transformation
 - Better separation of concerns



Still some work but the path is clear