Hierarchical Layer-based Class Extensions in Squeak/Smalltalk

Matthias Springer Hidehiko Masuhara Robert Hirschfeld

Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology
Hasso Plattner Institute, University of Potsdam

March 14, 2016

Overview

Introduction

- **Class Addition:** Add new method to class
- **Class Refinement:** Change (overwrite) existing method of class
- Use Cases
  - Convenience methods (e.g., `2.hours + 30.minutes`)
  - Bug fixing (*monkey patching*)
  - Multi-dimensional separation of concerns
    ($\rightarrow$ modular understandability)
  - Adding new operations to existing classes
    (c.f. *expression problem*, alternative to Visitor design pattern)
- Popular in Ruby and Smalltalk

Matriona Module System

- Module system for Squeak/Smalltalk
- Supports class nesting and class parameterization

[0] Picture copyright: S. Faric, CC BY 2.0 License

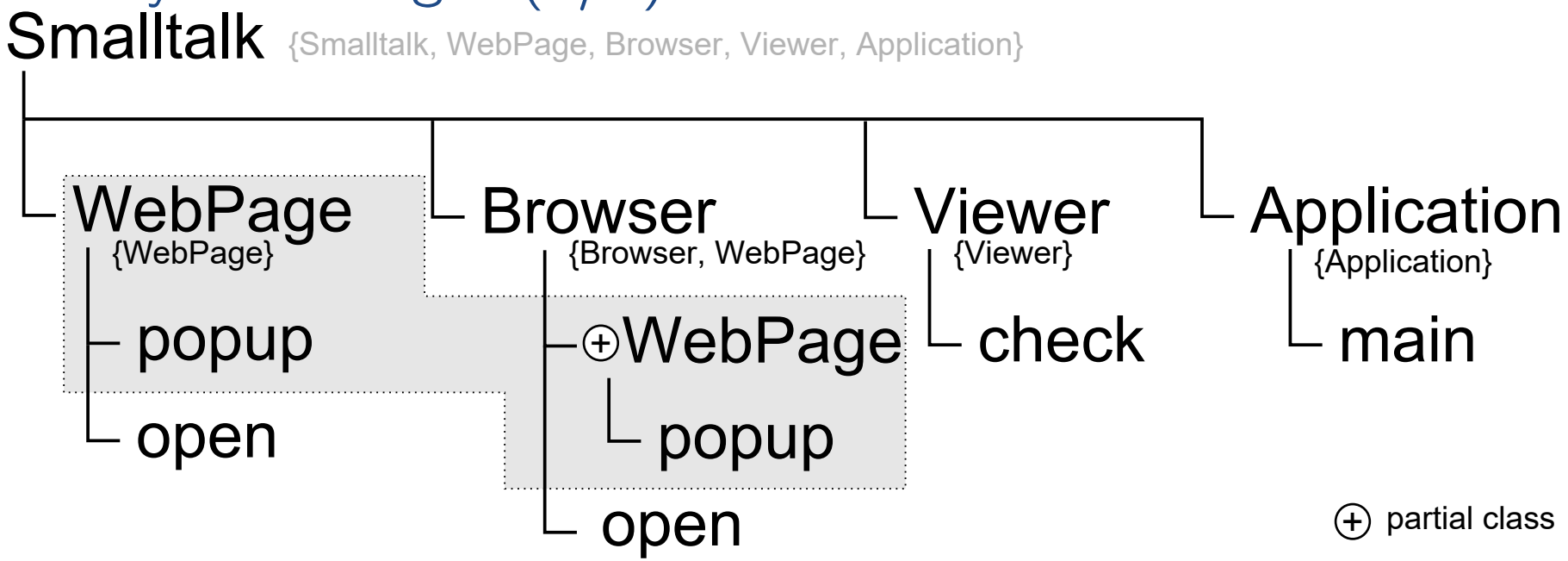
Overview

High-level Idea

Class extensions are . . .

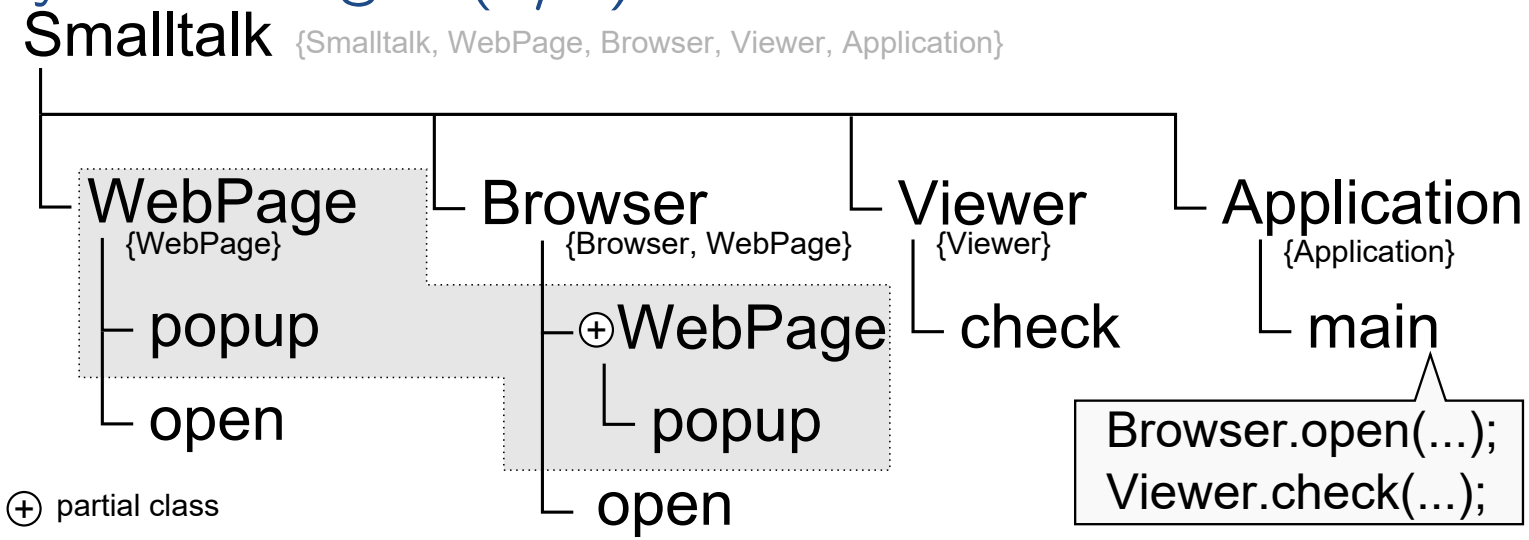
- class members (like methods and nested classes)
- subject to *local rebinding*: avoid breaking unrelated (*black box*) code
- active in a certain scope (*locality of changes*)
  - Explicit activation: class extension specifies in which parts of the program it should be active
  - Import activation: other class *requests* a class extension (mixins)
  - Hierarchical activation: class extension is active in all nested classes
- layered: multiple class extensions can be active at the same time
- (de)activated similar to layers in context-oriented programming (**layer activation stack**)

Locality of Changes (1/3)



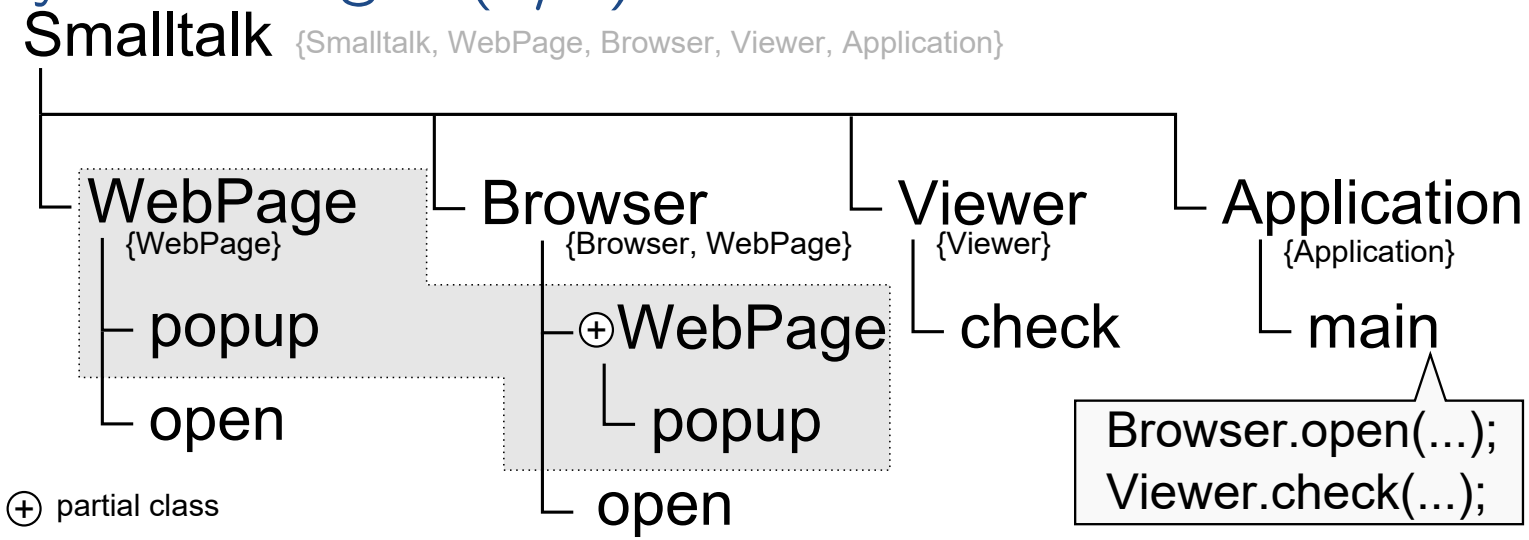
- Every class extension belogs to a *partial class*
- Class is activated if one of its methods is executing (e.g., `Browser.open` → `Browser`)
- *Scope* of class `Browser`: determines how long `Browser`'s class extensions will remain active (*deactivation* only)
- **Intuition:** *scope*(C) is the set of classes which are known to be compatible with C's extensions

Locality of Changes (2/3)

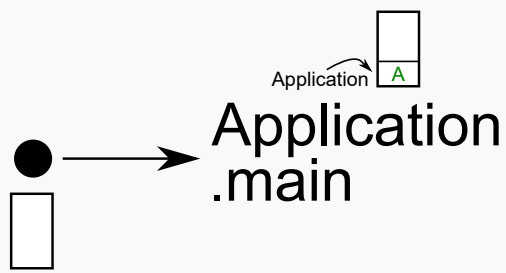


Example: `Application` calls `Browser` and `Viewer`

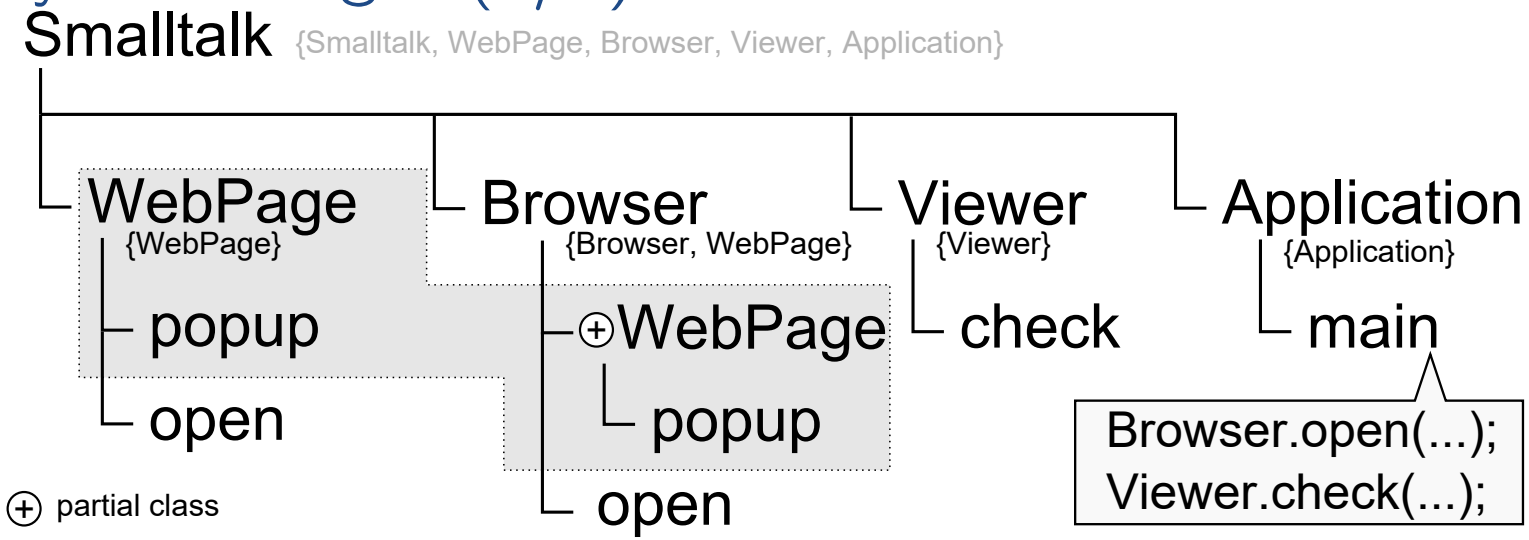
Locality of Changes (2/3)



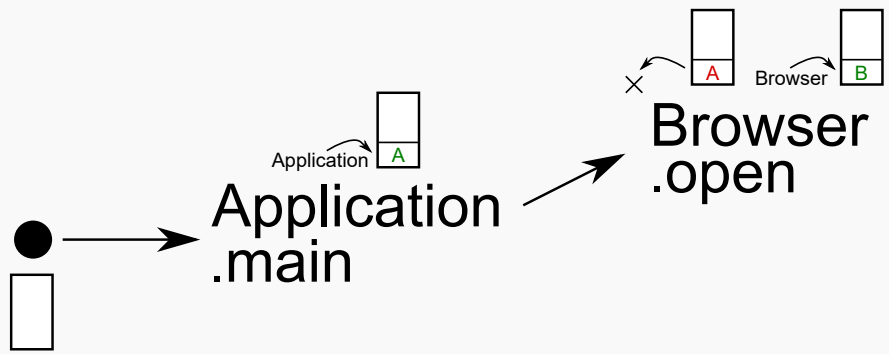
Example: `Application` calls `Browser` and `Viewer`



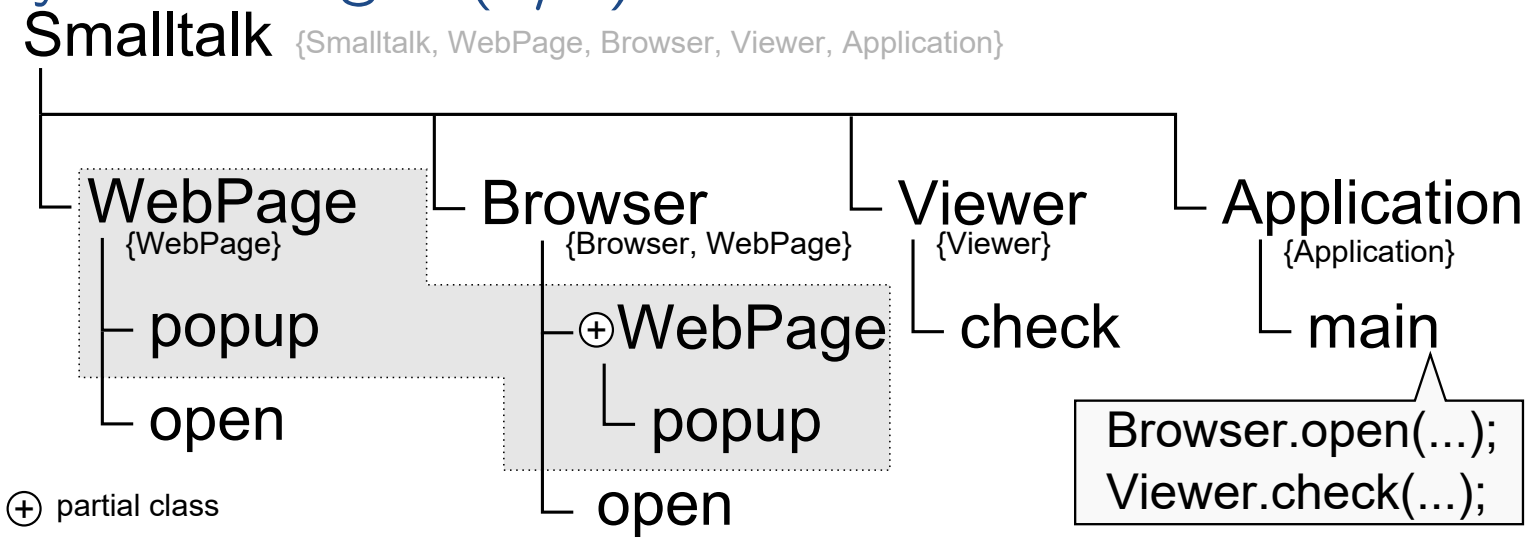
Locality of Changes (2/3)



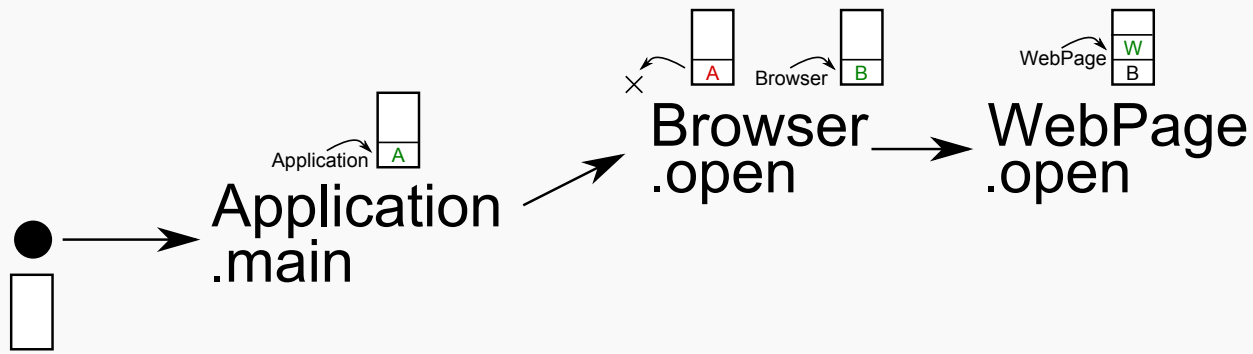
Example: `Application` calls `Browser` and `Viewer`



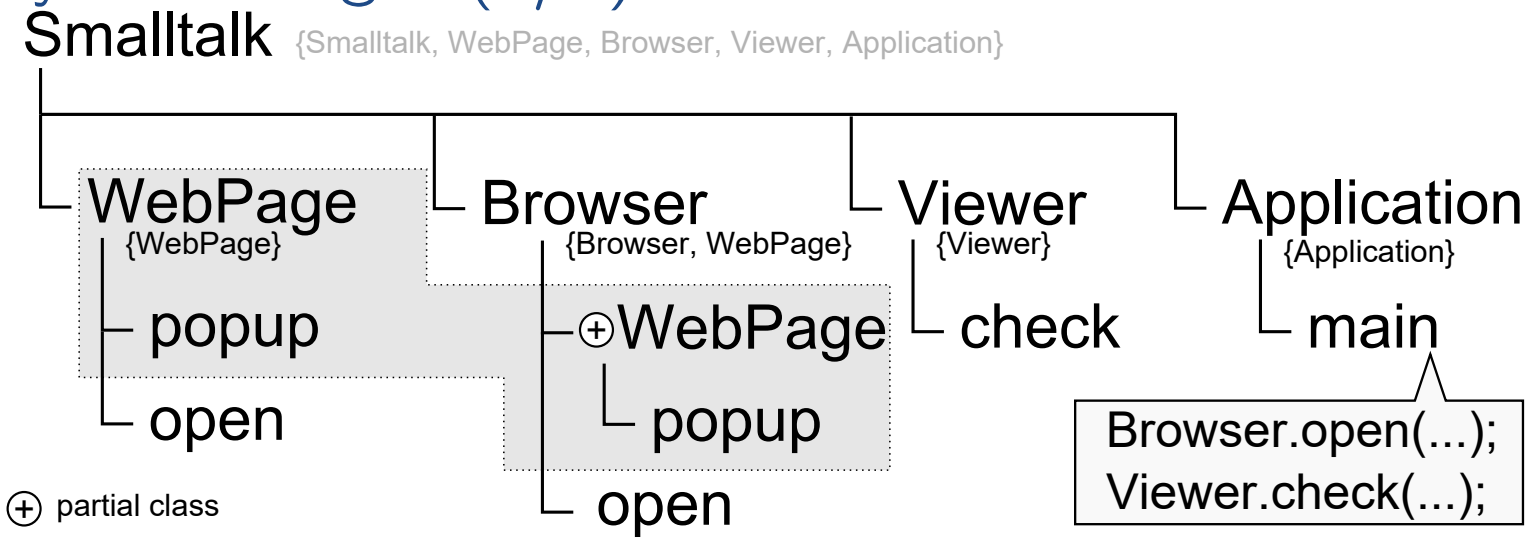
Locality of Changes (2/3)



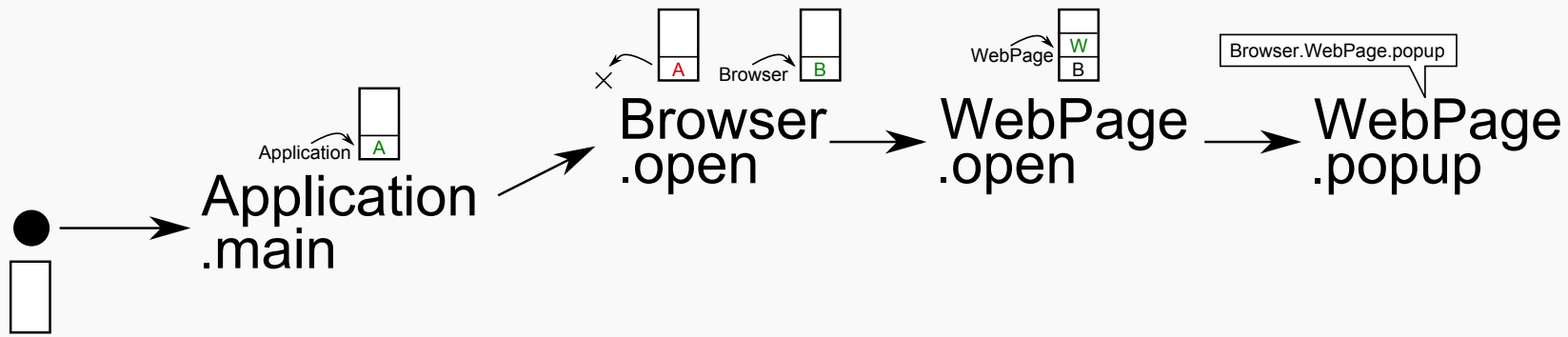
Example: `Application` calls `Browser` and `Viewer`



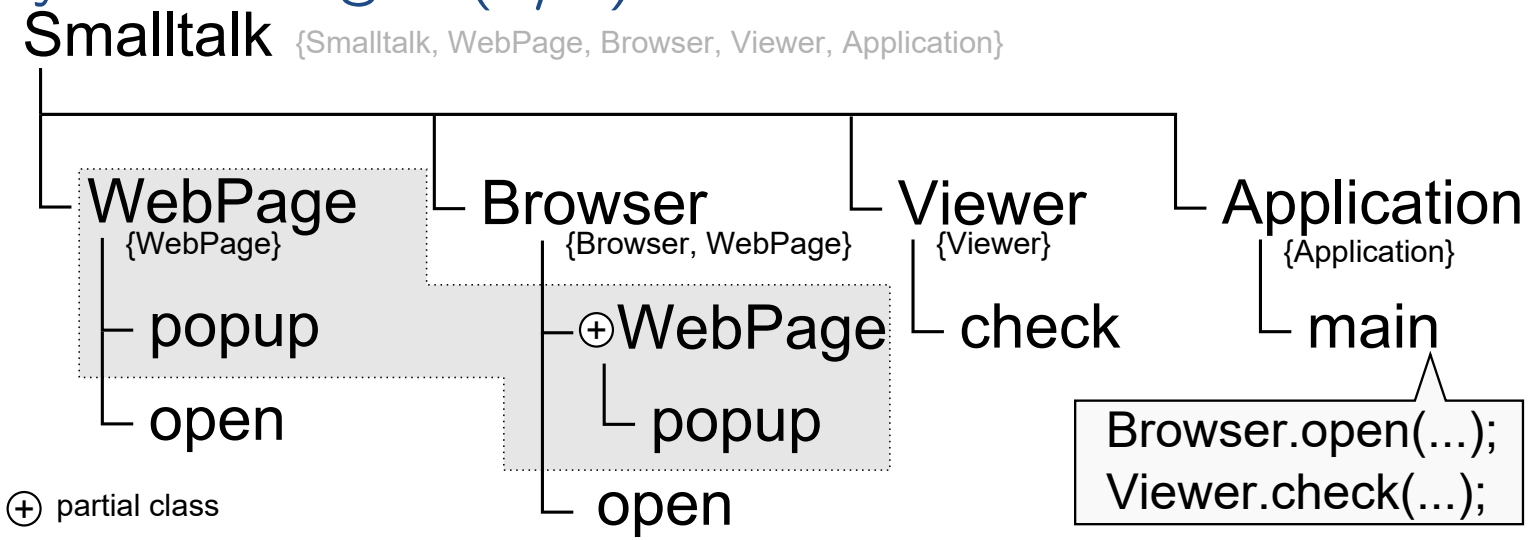
Locality of Changes (2/3)



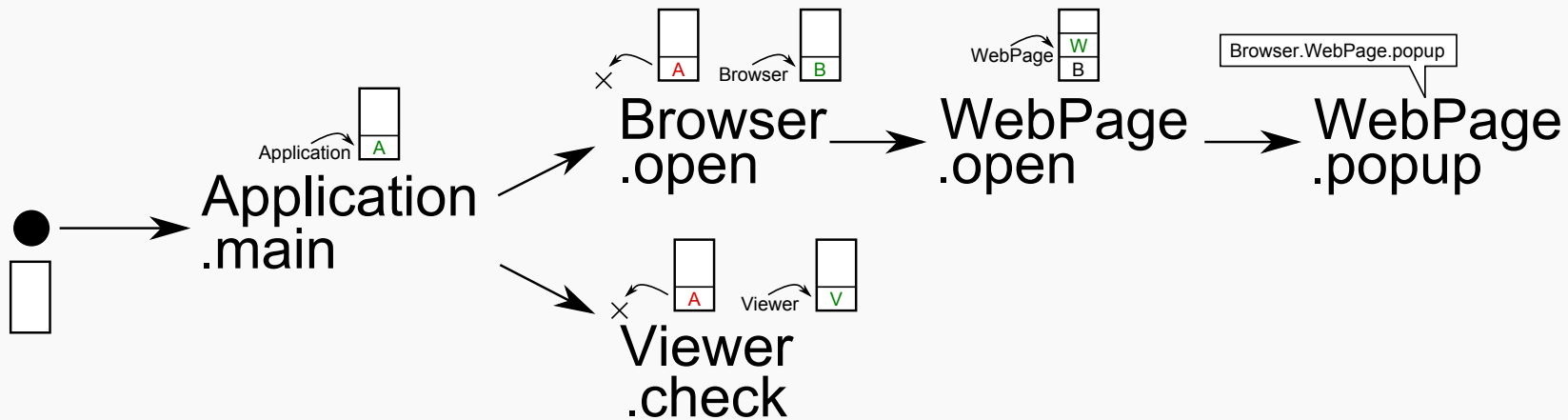
Example: `Application` calls `Browser` and `Viewer`



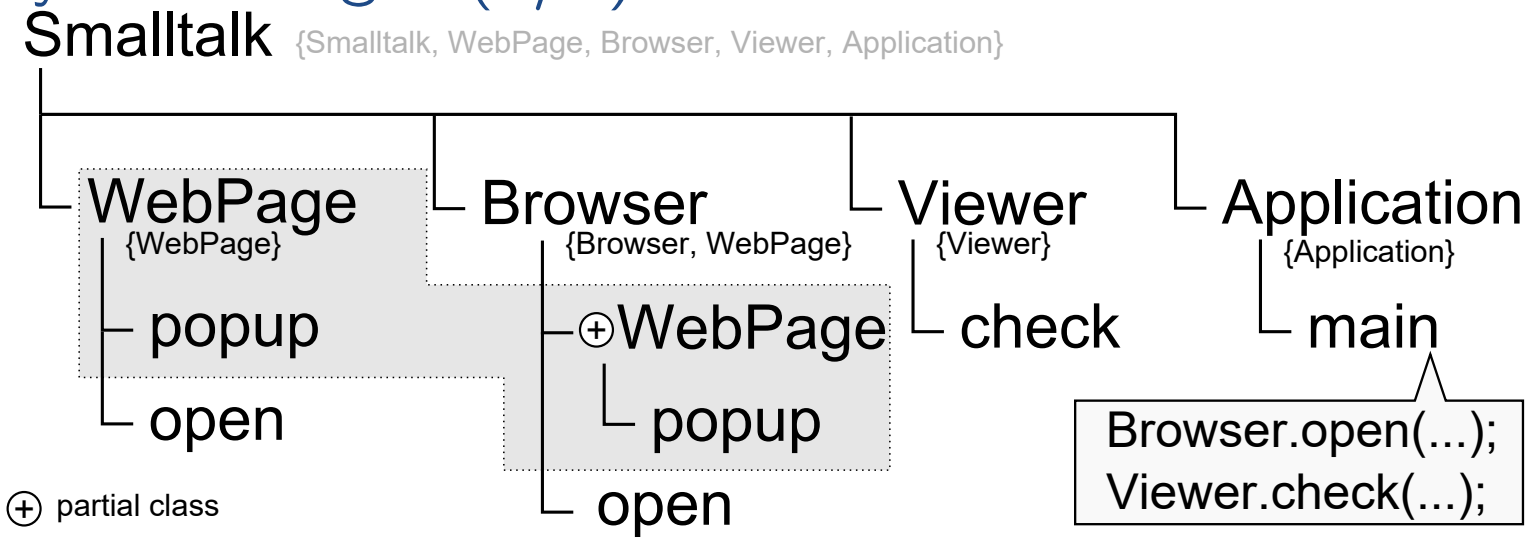
Locality of Changes (2/3)



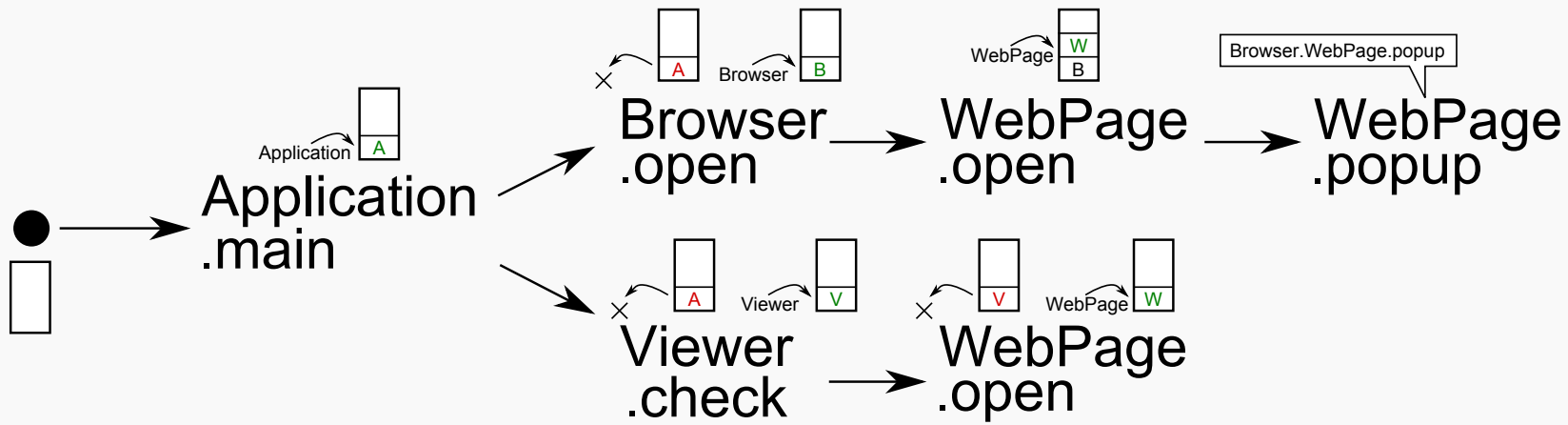
Example: `Application` calls `Browser` and `Viewer`



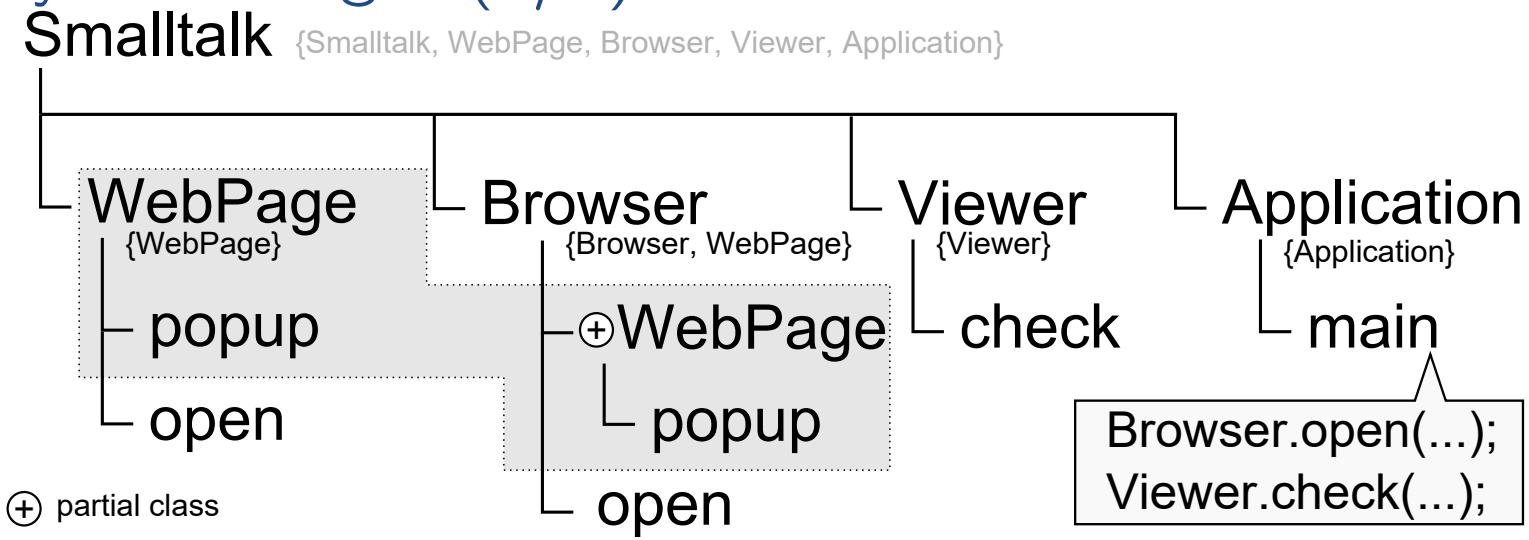
Locality of Changes (2/3)



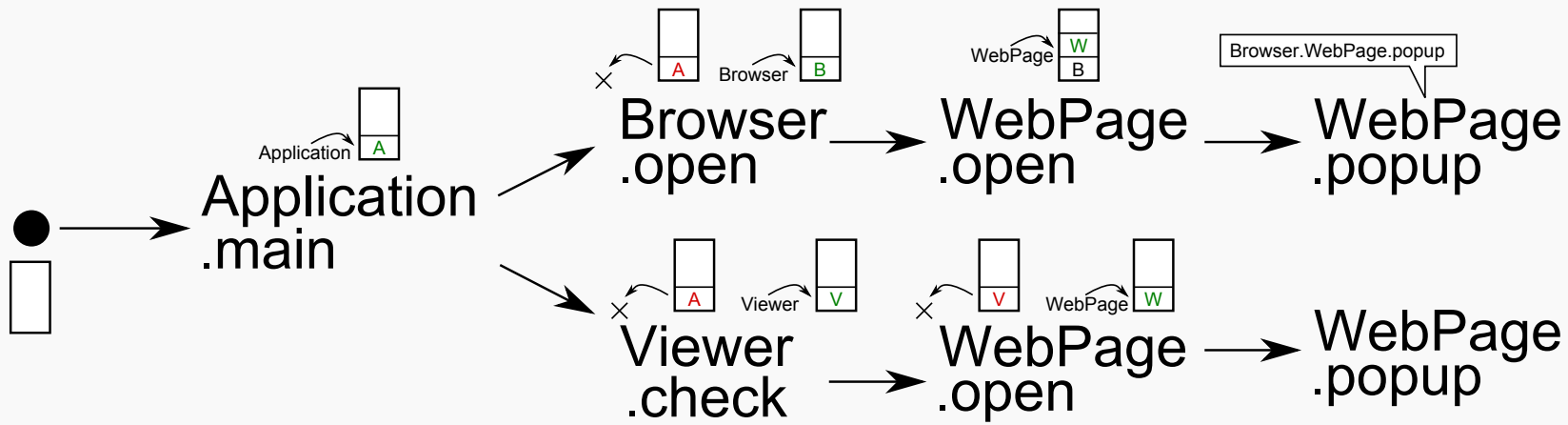
Example: `Application` calls `Browser` and `Viewer`



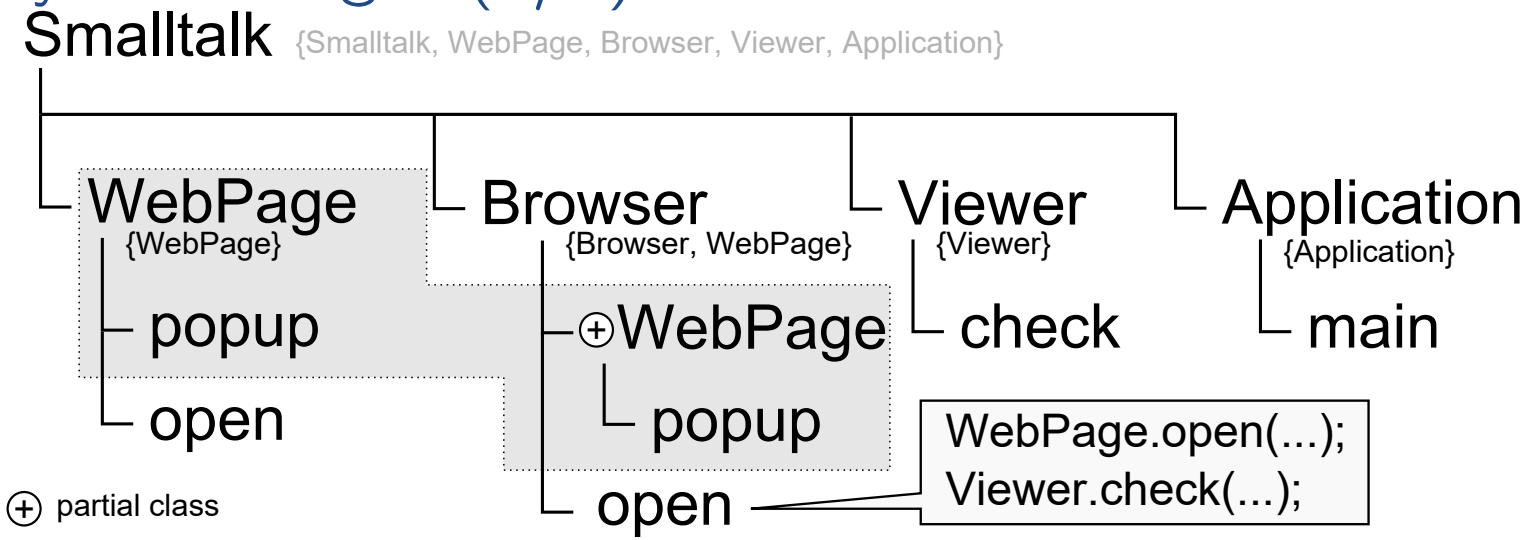
Locality of Changes (2/3)



Example: `Application` calls `Browser` and `Viewer`

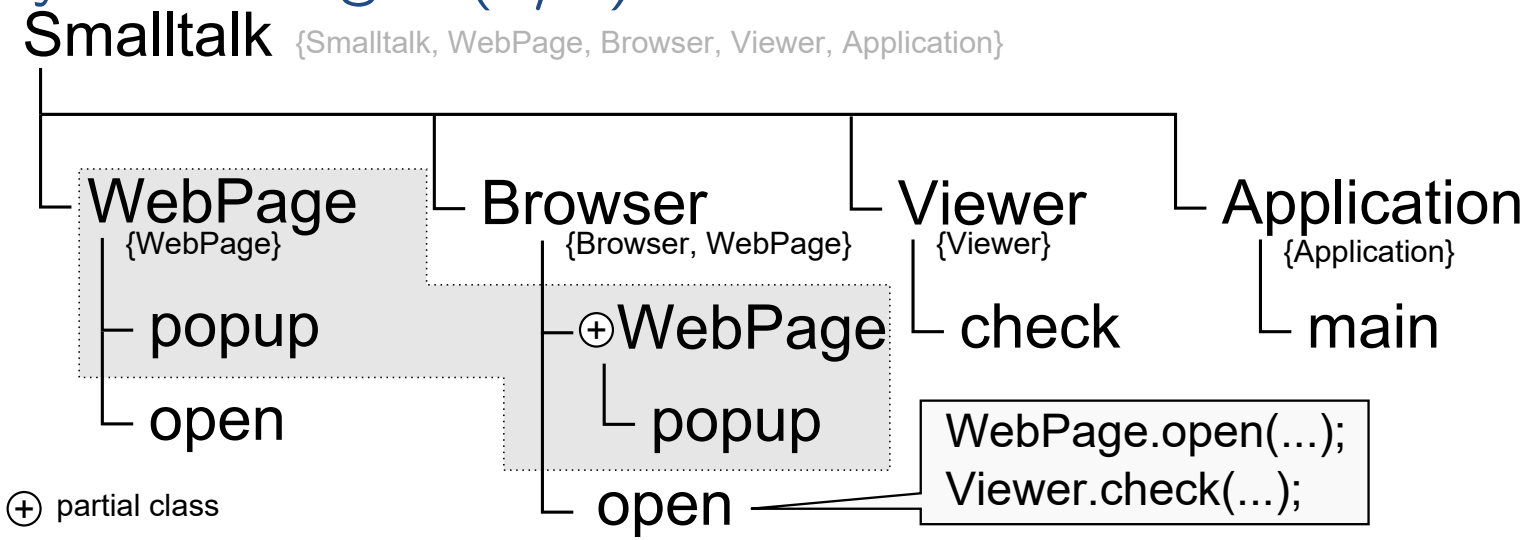


Locality of Changes (3/3)

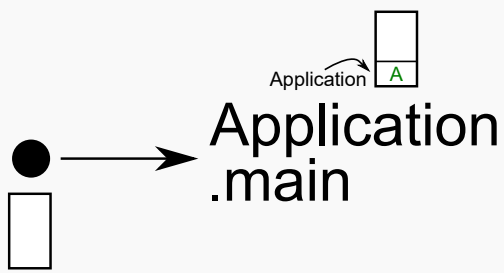


Example: `Application` calls `Browser` calls `Viewer`

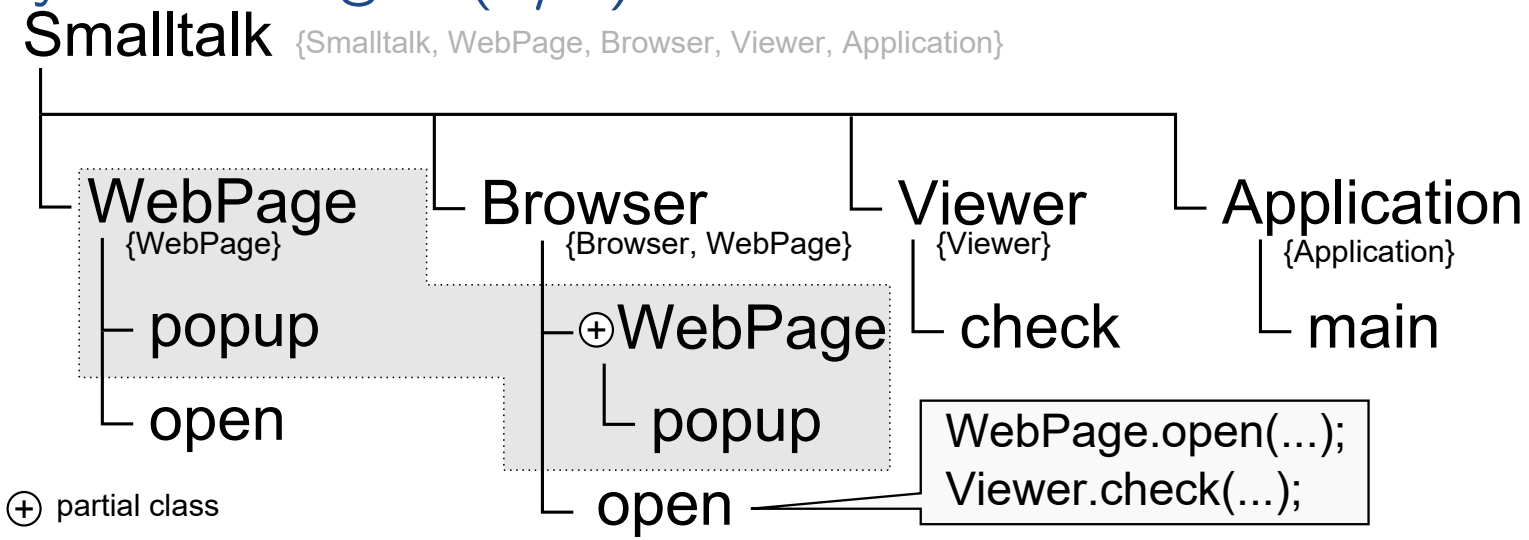
Locality of Changes (3/3)



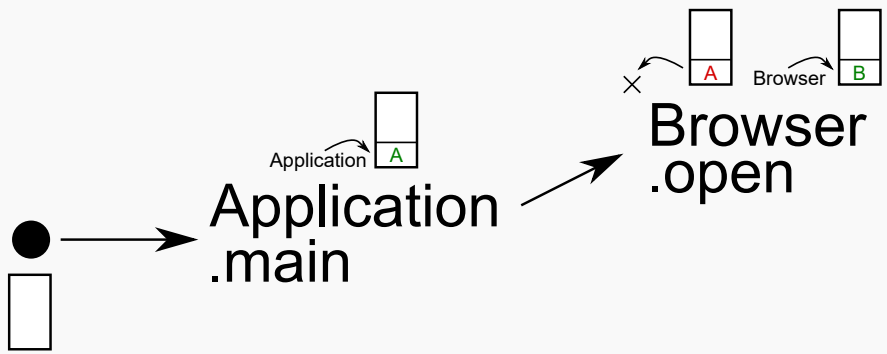
Example: `Application` calls `Browser` calls `Viewer`



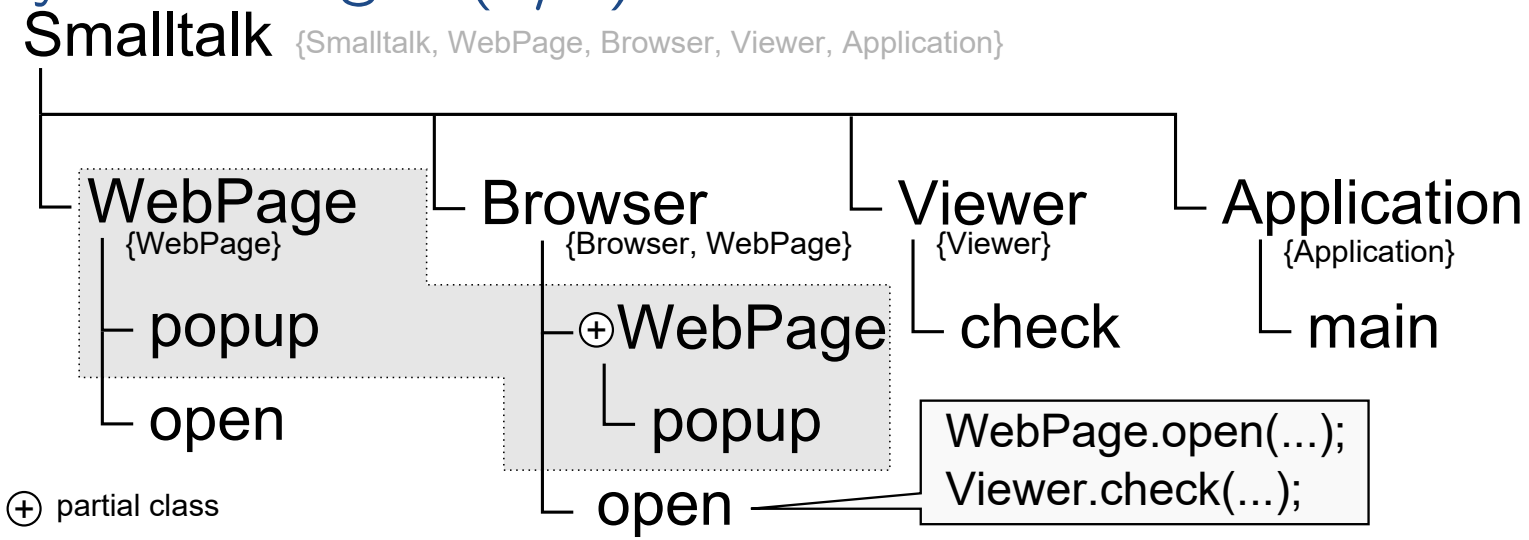
Locality of Changes (3/3)



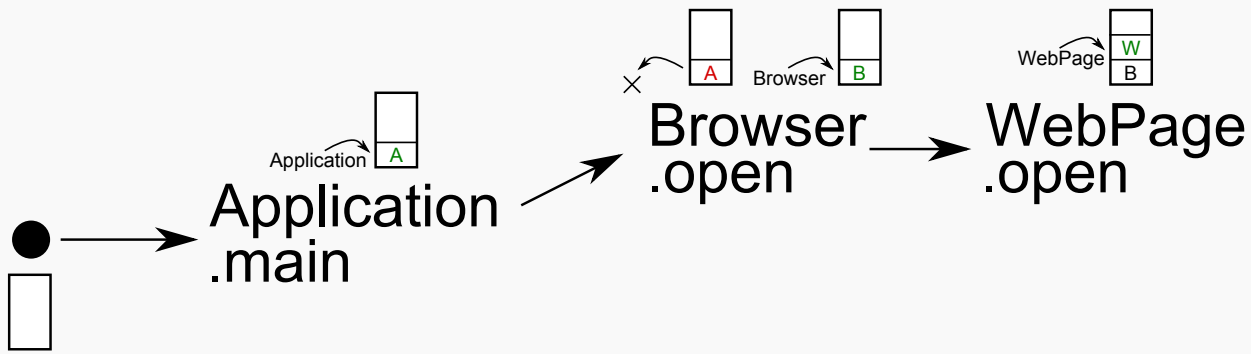
Example: `Application` calls `Browser` calls `Viewer`



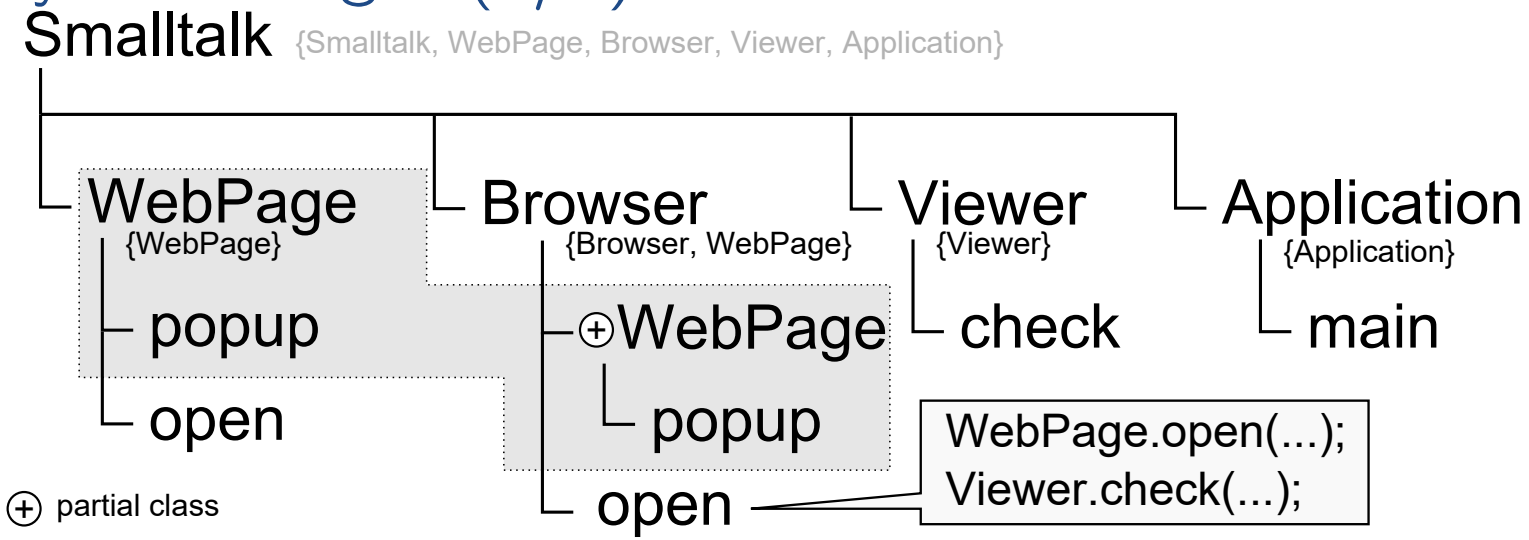
Locality of Changes (3/3)



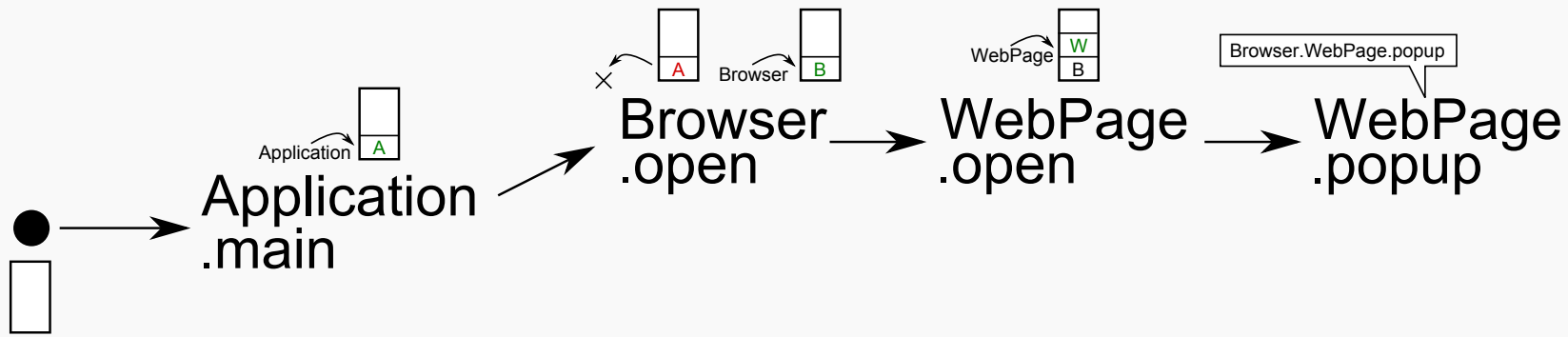
Example: `Application` calls `Browser` calls `Viewer`



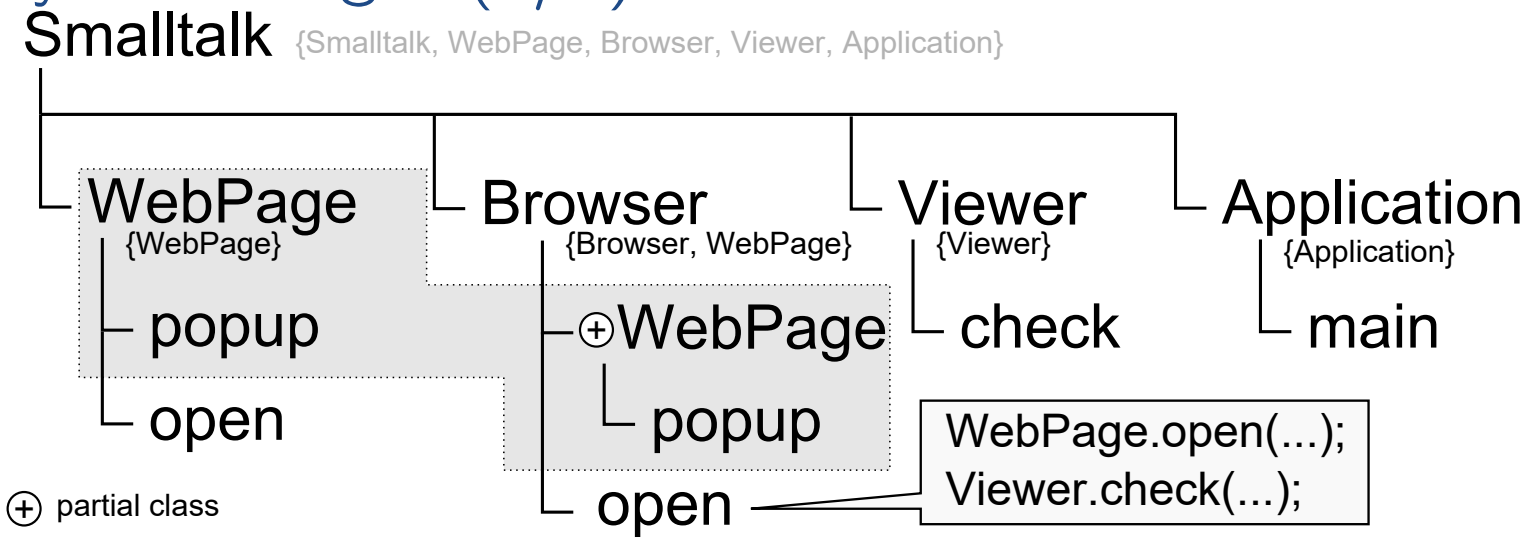
Locality of Changes (3/3)



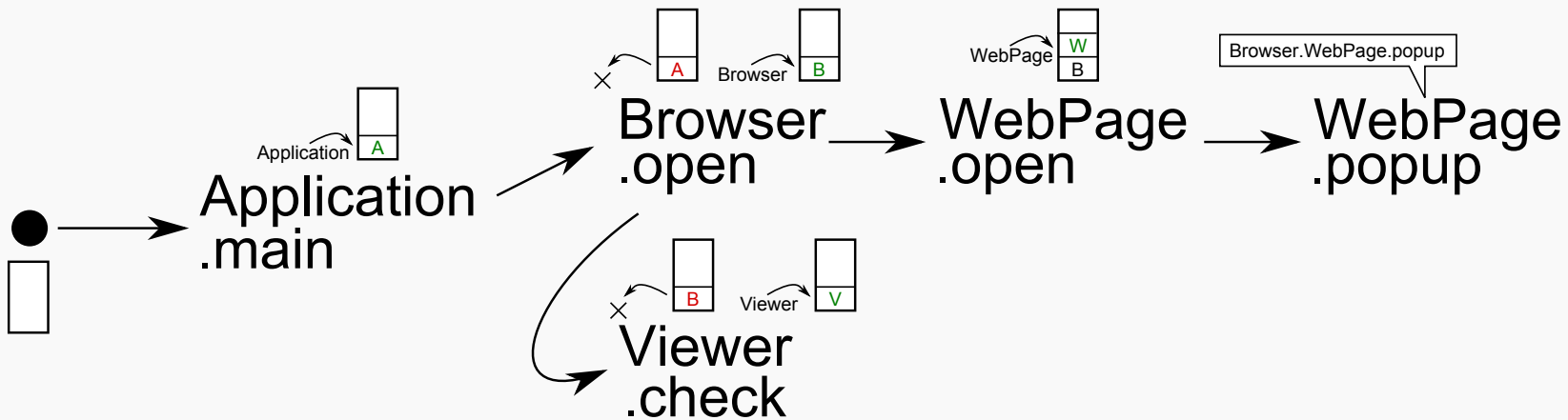
Example: `Application` calls `Browser` calls `Viewer`



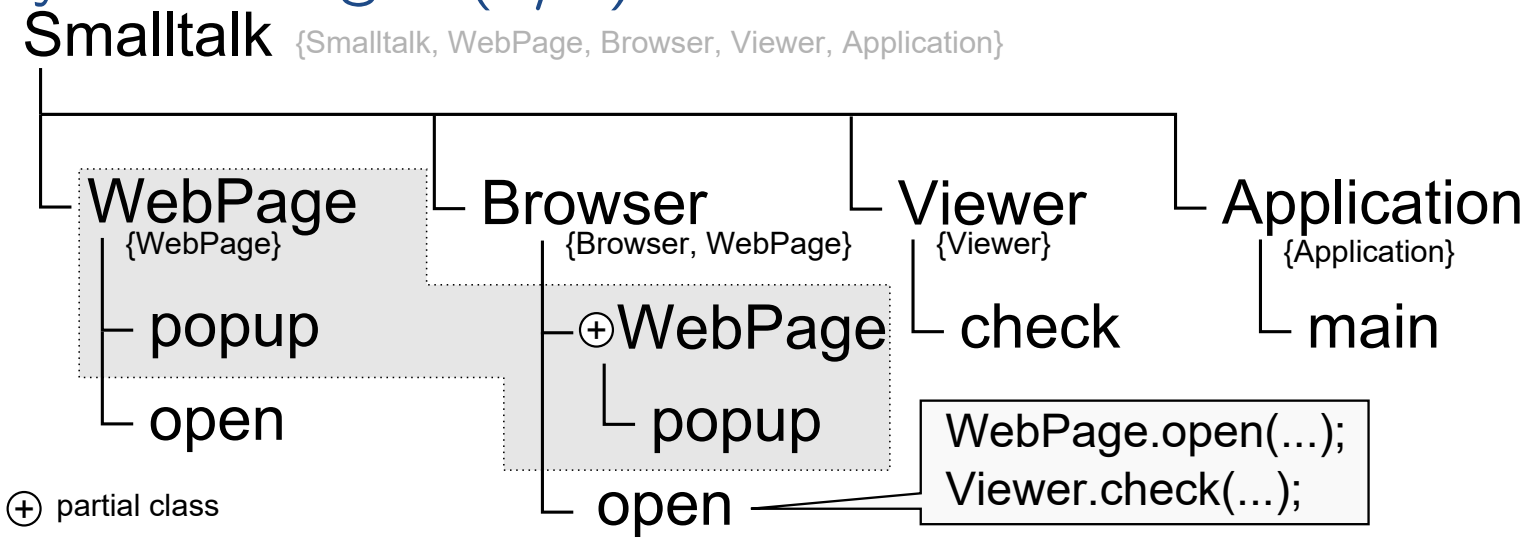
Locality of Changes (3/3)



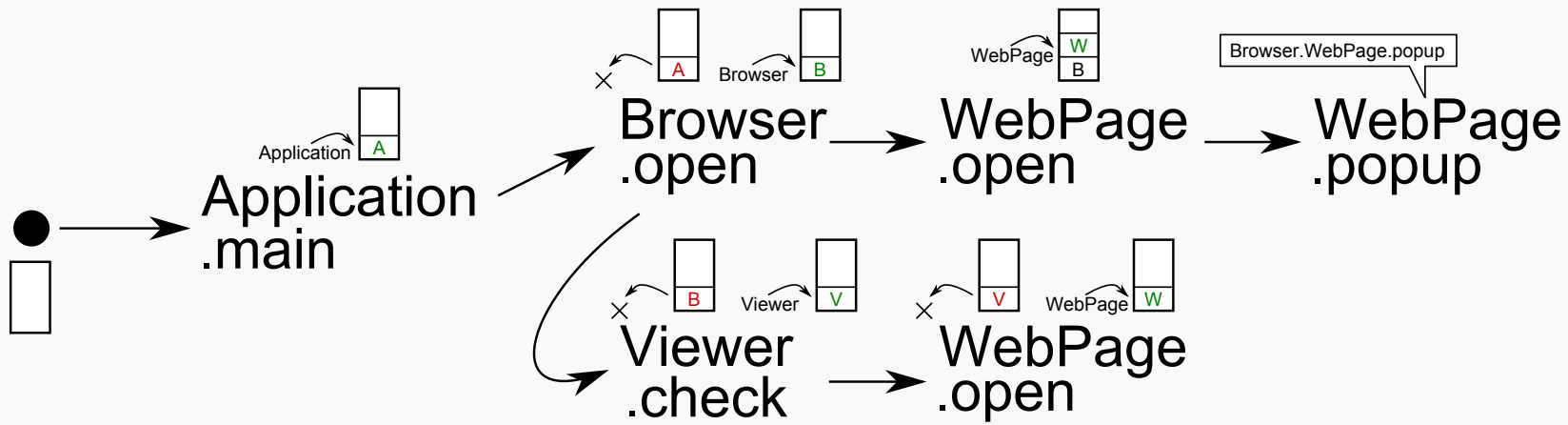
Example: `Application` calls `Browser` calls `Viewer`



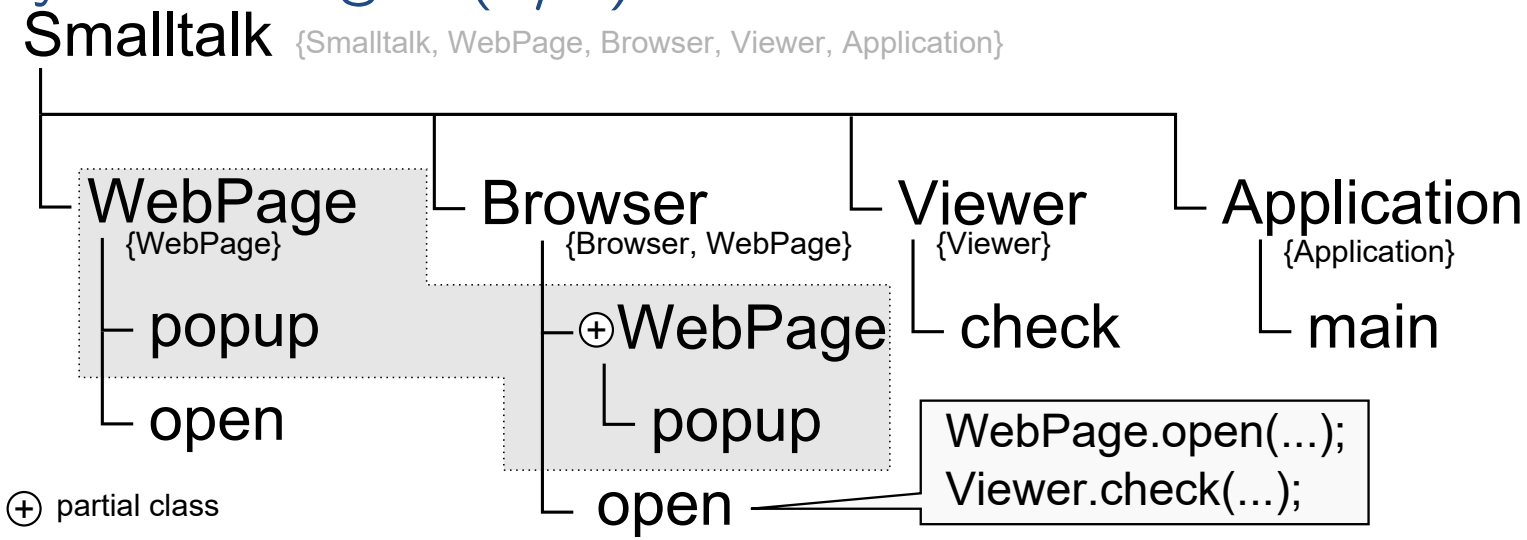
Locality of Changes (3/3)



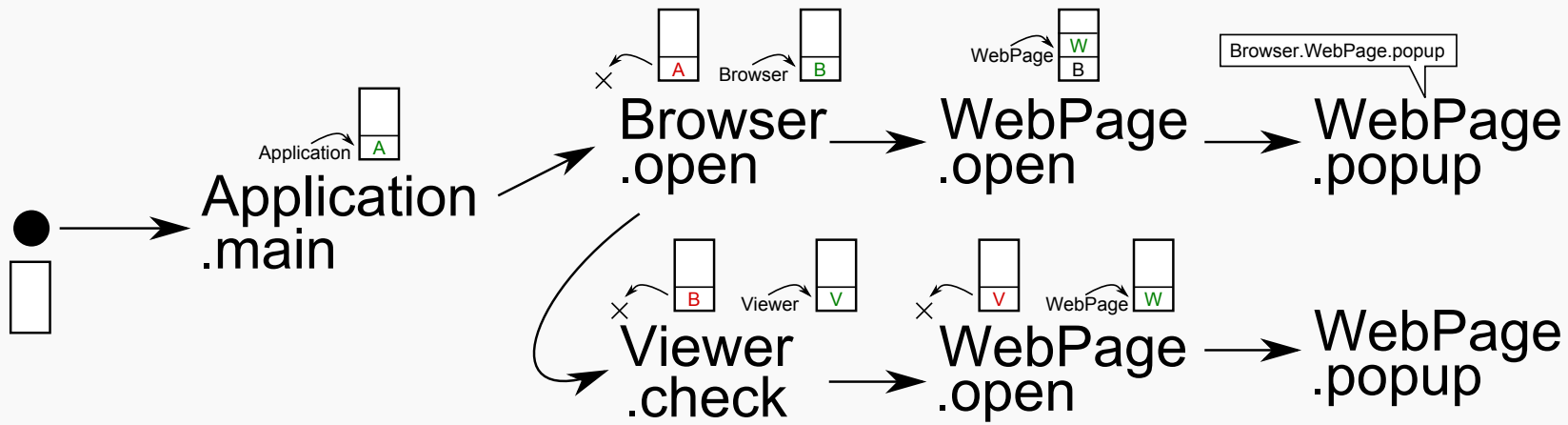
Example: `Application` calls `Browser` calls `Viewer`



Locality of Changes (3/3)



Example: `Application` calls `Browser` calls `Viewer`



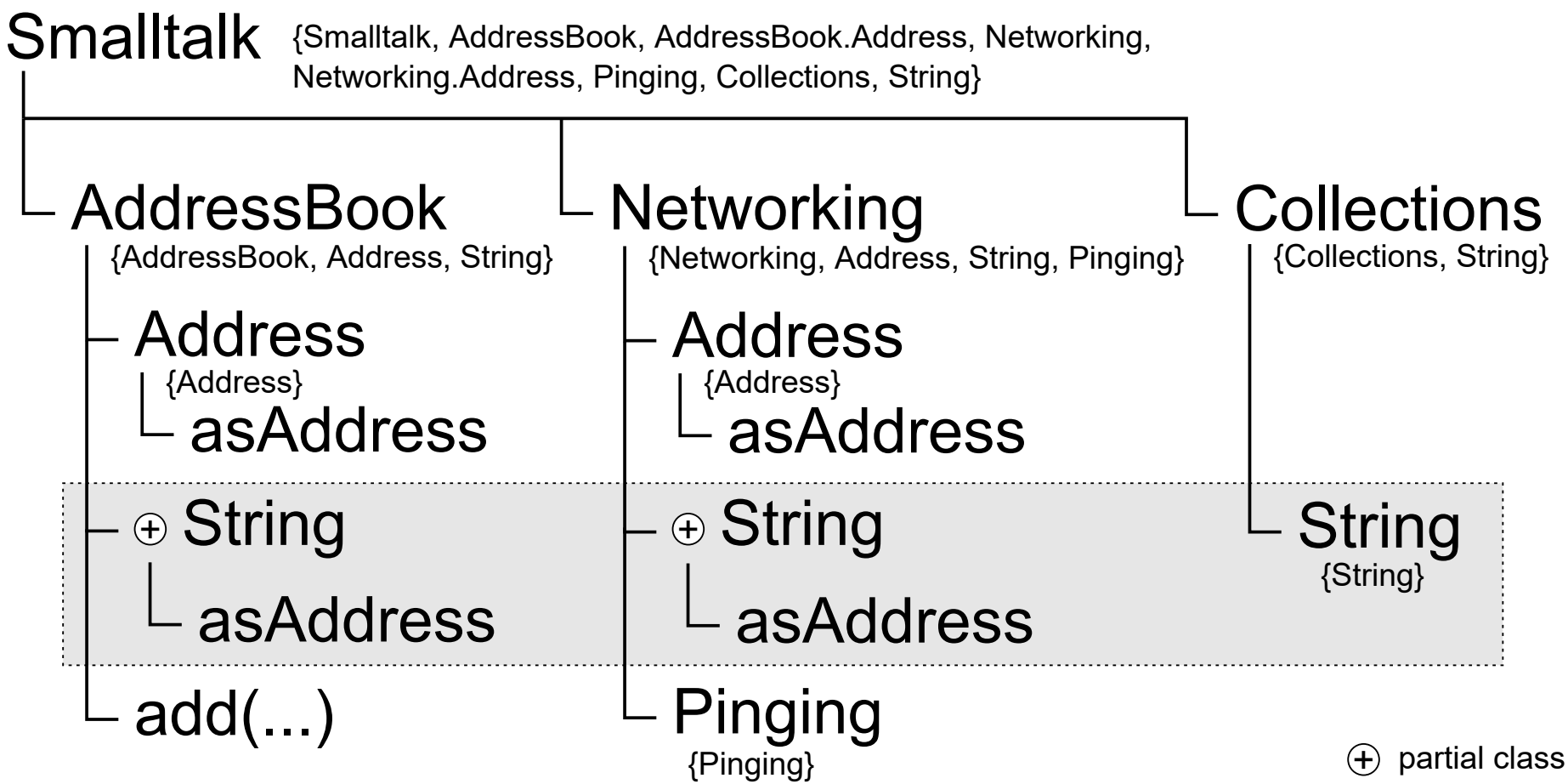
Scope of a Class

Class L remains active as long as a method within $scope(L)$ is executing.

Definition

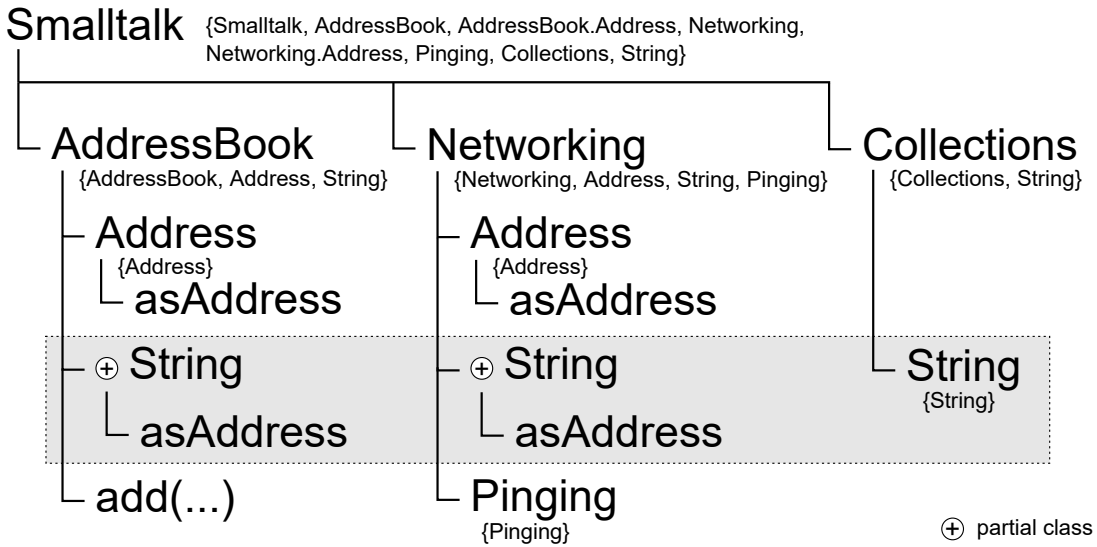
$$scope(L) = \{L\} \quad \text{(reflexivity)}$$
$$\cup \{target(P) \mid P \in partials(L)\} \quad \text{(local rebinding)}$$

Hierarchical Scoping



Hierarchical Scoping

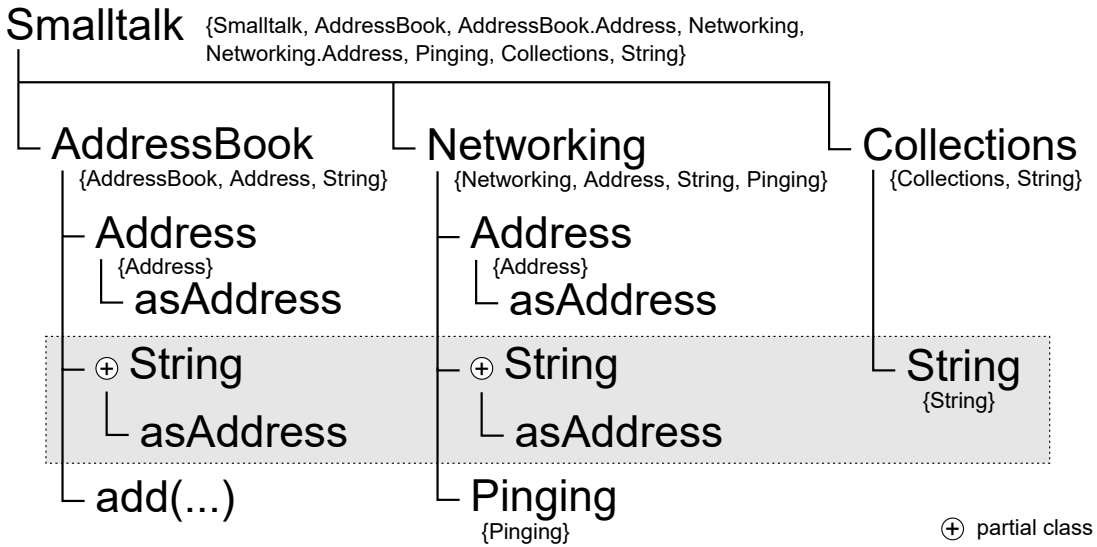
- **Activation:** Add current and all enclosing classes
- **Deactivation:** For all nested classes N in C: $scope(N) \subseteq scope(C)$



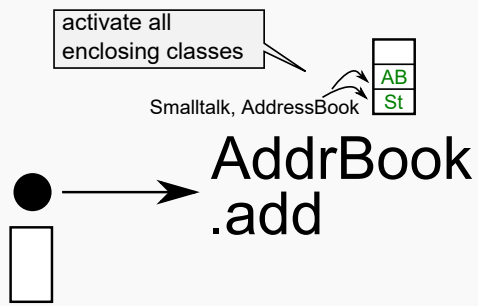
Example: Address book application with remote storage

Hierarchical Scoping

- **Activation:** Add current and all enclosing classes
- **Deactivation:** For all nested classes N in C: $scope(N) \subseteq scope(C)$

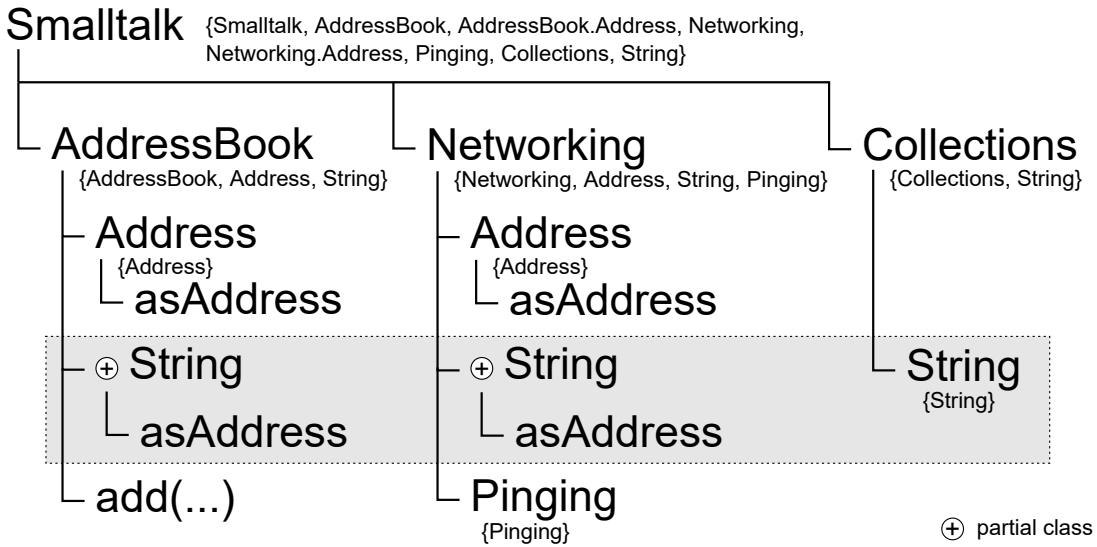


Example: Address book application with remote storage

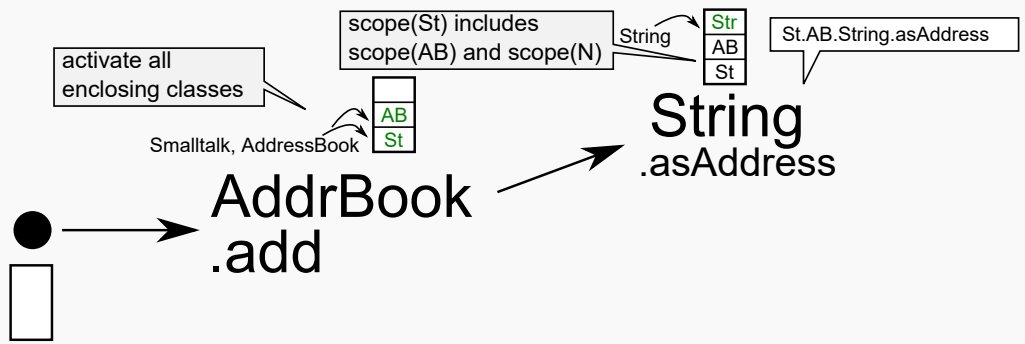


Hierarchical Scoping

- **Activation:** Add current and all enclosing classes
- **Deactivation:** For all nested classes N in C: $scope(N) \subseteq scope(C)$

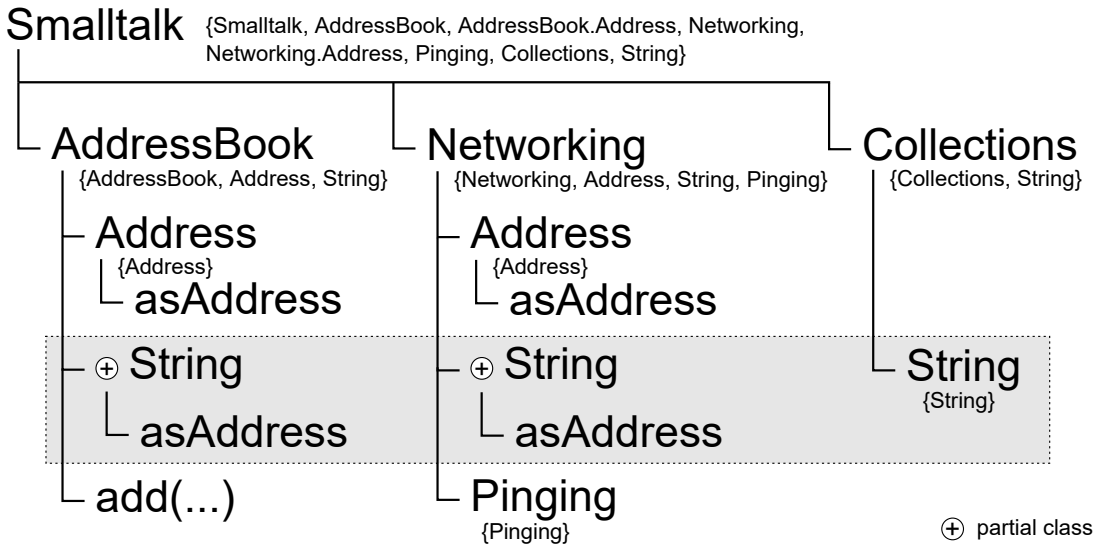


Example: Address book application with remote storage

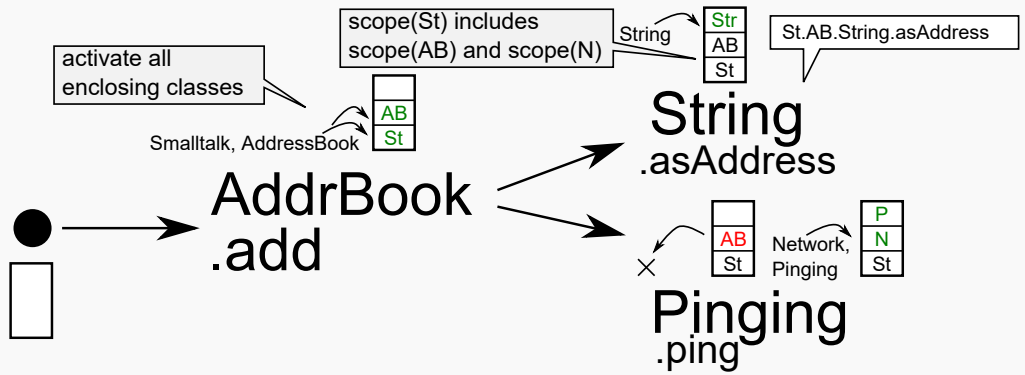


Hierarchical Scoping

- **Activation:** Add current and all enclosing classes
- **Deactivation:** For all nested classes N in C: $scope(N) \subseteq scope(C)$

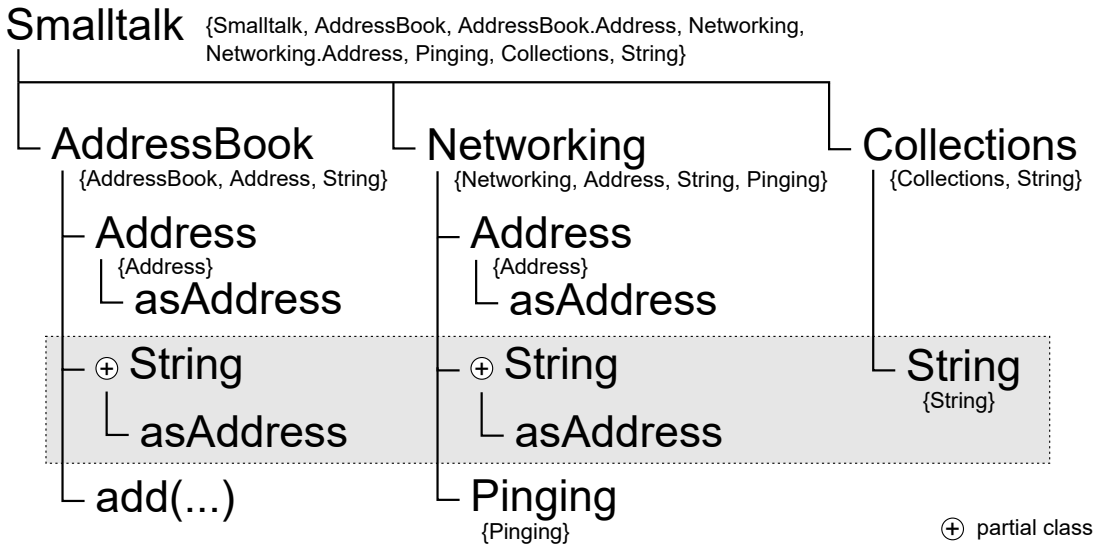


Example: Address book application with remote storage

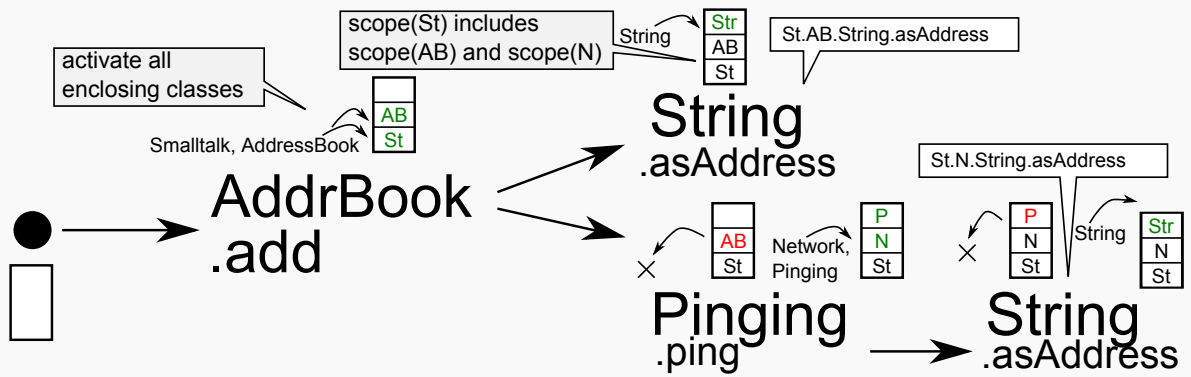


Hierarchical Scoping

- **Activation:** Add current and all enclosing classes
- **Deactivation:** For all nested classes N in C: $scope(N) \subseteq scope(C)$



Example: Address book application with remote storage



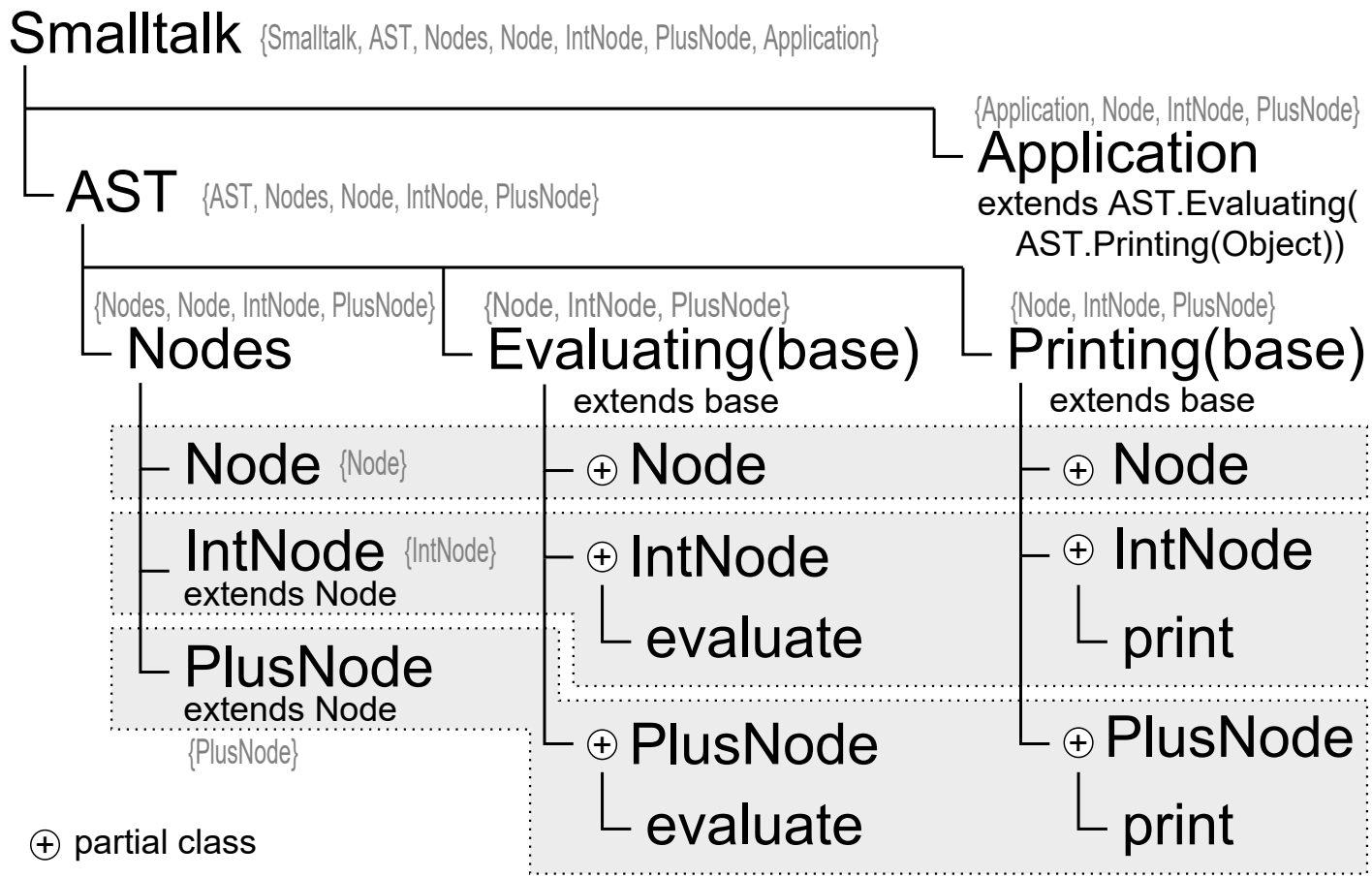
Scope of a Class

Class L remains active as long as a method within $scope(L)$ is executing.

Definition

$$scope(L) = \{L\} \qquad \text{(reflexivity)}$$
$$\cup\, \{\sout{target(P) \mid P \in partials(L)}\} \qquad \text{(local rebinding)}$$
$$\cup\, \{C \mid C \in nested^*(target(P)) \wedge P \in partials(L)\}$$
$$\cup\, \{C \mid C \in scope(N) \wedge N \in nested(L)\} \qquad \text{(hierarch. scoping)}$$

Importing Class Extensions



- Scope of a class includes scope of superclass
- Activate a class `C` if a method is executing in its context (polymorphic receiver class's superclass hierarchy includes `C`)

Scope of a Class

Class L remains active as long as a method within $scope(L)$ is executing.

Definition

$$scope(L) = \{L\} \qquad \text{(reflexivity)}$$
$$\cup \{C \mid C \in nested^*(target(P)) \wedge P \in partials(L)\} \qquad \text{(loc. rebinding)}$$
$$\cup \{C \mid C \in scope(N) \wedge N \in nested(L)\} \qquad \text{(hierarch. scoping)}$$
$$\cup\, scope(superclass(L)) \qquad \text{(importing class extensions)}$$

Overview

Scope of a Class

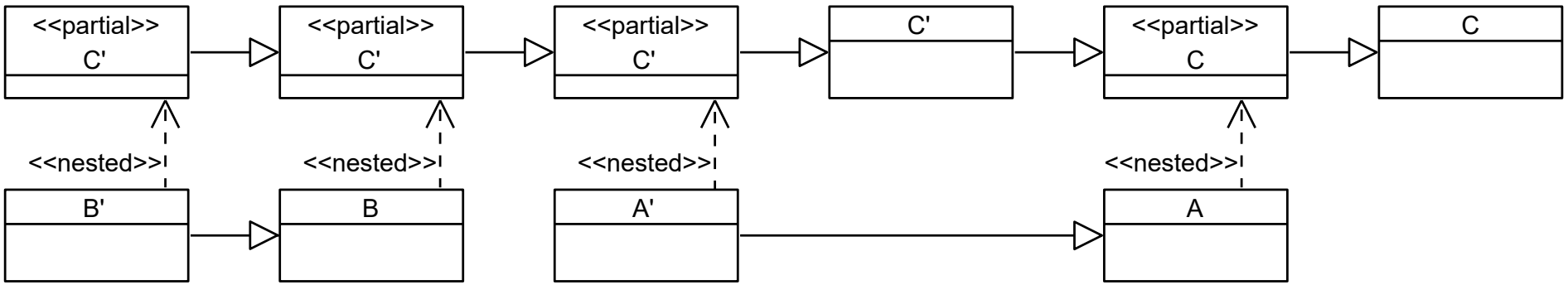
When calling or returning to a method on an object of class C, activate all enclosing classes of C and C.

Class L remains active as long as a method within $scope(L)$ is executing.

Definition

$$
\begin{aligned}
scope(L) = \{L\} & \qquad \text{(reflexivity)} \\
\cup \{C \mid C \in nested^*(target(P)) \wedge P \in partials(L)\} & \qquad \text{(loc. rebinding)} \\
\cup \{C \mid C \in scope(N) \wedge N \in nested(L)\} & \qquad \text{(hierarch. scoping)} \\
\cup scope(superclass(L)) & \qquad \text{(importing class extensions)}
\end{aligned}
$$

Effective Superclass Hierarchy



- Insert active partial classes in superclass hierarchy
- Activation order determines order of partial classes
- No `proceed` statement, use `super` instead

Overview

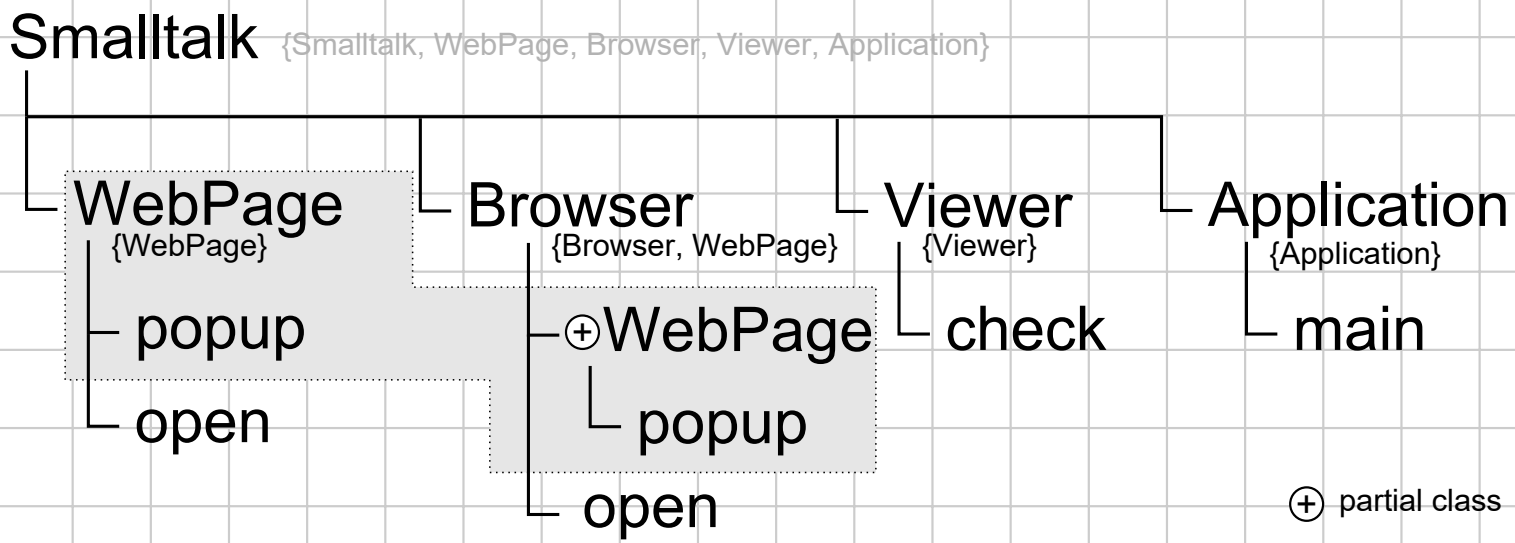
Conclusion

- **High-level idea:** Make sure that class extensions are not destructive by confining their scope to *compatible* classes
- **Scoping dimensions:** explicit scoping, class nesting hierarchy (*hierarchical scoping*), superclass hierarchy (*import scoping*)
- Future Work
  - Implementation details and performance optimizations
  - Blocks/anonymous functions
  - Formal semantics of the mechanism

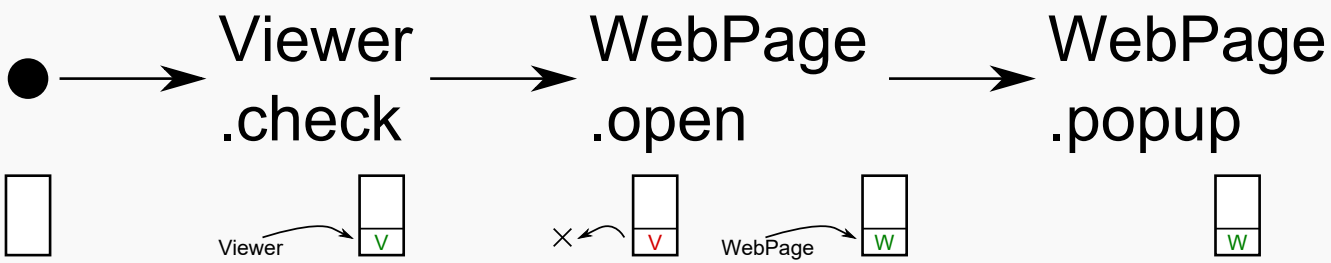
Appendix

Appendix

Locality of Changes (4/3)

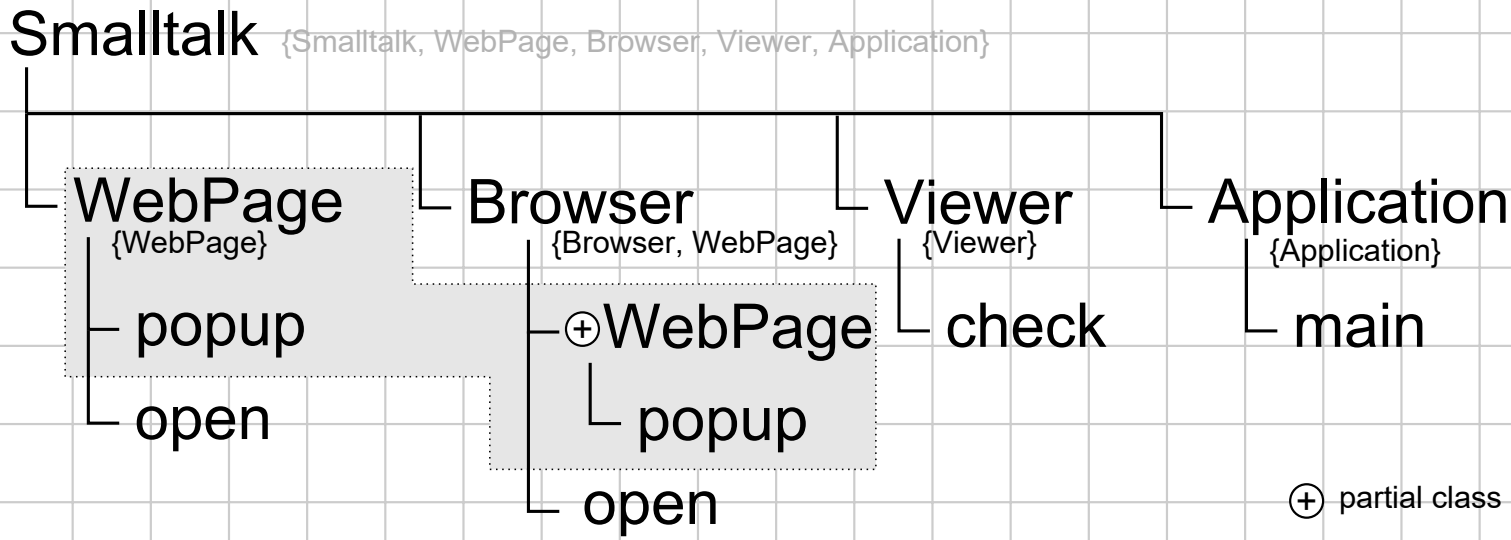


Example: Standalone usage of `Viewer`

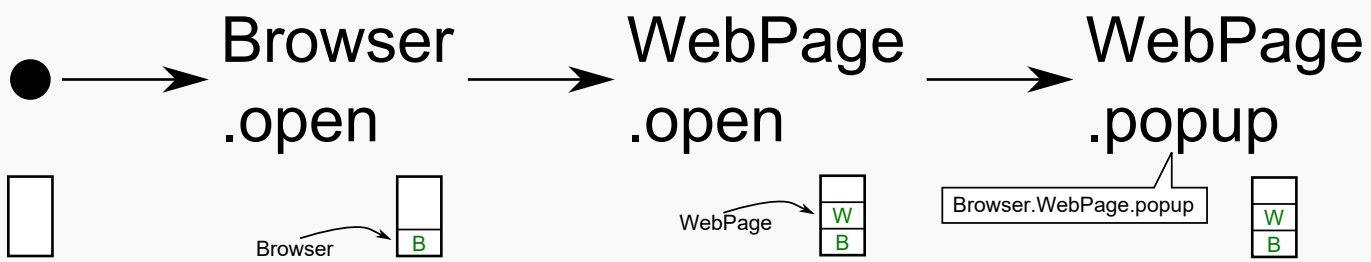


`WebPage.Browser` is never activated

Locality of Changes (5/3)



Example: Standalone usage of `Browser`



`Browser.WebPage.popup` is called