

Adaptive Object-Oriented Software Development

The Demeter Method

Introduction

- Software engineering
- Programming languages
- Hands-on, practical, useful

Summary of Course

- How to design and implement flexible object-oriented software using the principles of adaptiveness.
- How to design and implement flexible 100% pure Java software

Who is in Attendance?

- Software developers who have some experience with object-oriented programming.
- Should know concepts of OOP, like classes, methods, and late binding.

Agenda for Adaptive Object-Oriented Software/Demeter

Demeter/Java

Java

Java environment

UML

XML

requirements

domain analysis

design

implementation

Adaptive Programming

Aspect-Oriented Progr.

principles

heuristics

patterns

idioms

theorems

algorithms

Demeter Method

iterative development

spiral model

strategy graphs

class graphs

object graphs

state graphs

use cases

interfaces

traversals

visitors

packages

Agenda

- UML class diagrams. Perspective: analysis, design, implementation
- Class graphs as special cases of UML class diagrams
- Textual representation of class graphs
- Default implementation of UML class diagrams as class graphs

Agenda

- UML interaction diagrams
 - sequence diagrams
 - collaboration diagrams
 - improving interaction diagrams to make them specification languages

Agenda

- Class graphs in textual form (construction, alternation)
- object graphs (graphical and textual form)
- object graphs defined by class graphs
- add repetition classes and optional parts
- translating class graphs to Java

Agenda

- annotating class graph with syntax: class dictionary
- printing objects and language defined by class dictionary
- grammars, parsing, ambiguous grammars, undecidable problem
- LL(1) grammars, robustness of sentences
- etc.

Overview

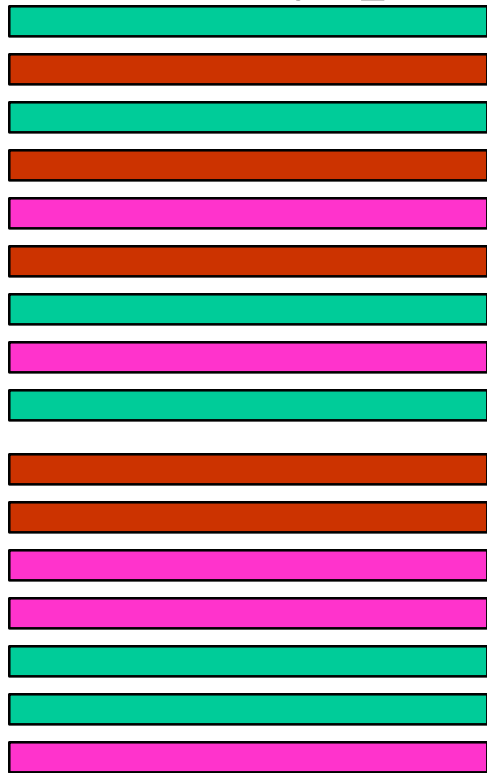
- good separation of concerns is the goal
- concerns should be cleanly localized
- programs should look like designs

Adaptive Programming

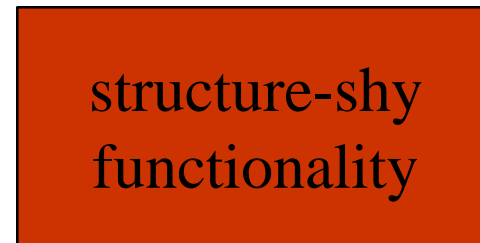
- Programs adapt to interesting context changes
- Structure-shy behavior
- Succinct representation of traversals
- Programming in terms of graph constraints

Cross-cutting of components and aspects

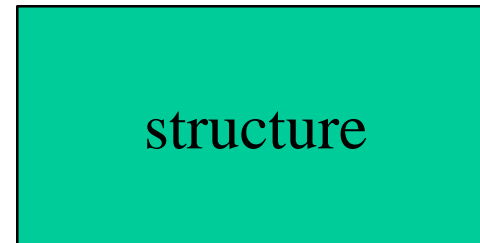
ordinary program



better program



Components



Aspect 1



Aspect 2

Aspect-Oriented Programming

components and aspect descriptions

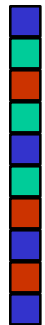
weaver
(compile-
time)



High-level view,
implementation may
be different



Source Code
(tangled code)



Examples of Aspects

- Synchronization of methods across classes
- Remote invocation (using Java RMI)
- Quality of Service (QoS)
- Failure handling
- External use (e.g., being a Java bean)
- Replication, Migration
- etc.

Connections

- explain adaptive programming in terms of patterns
- Aspect-Oriented Programming (AOP) is a generalization of Adaptive Programming (AP)
- correspondence:
adaptive program : object-oriented program
= sentence : object graph

Vocabulary

- Graph, nodes, edges, labels
- Class graph, construction, alternation
- Object graph, satisfying class graph
- UML class diagram
- Grammar, printing, parsing

Vocabulary

- Traversals, visitors
- Strategy graphs, path set

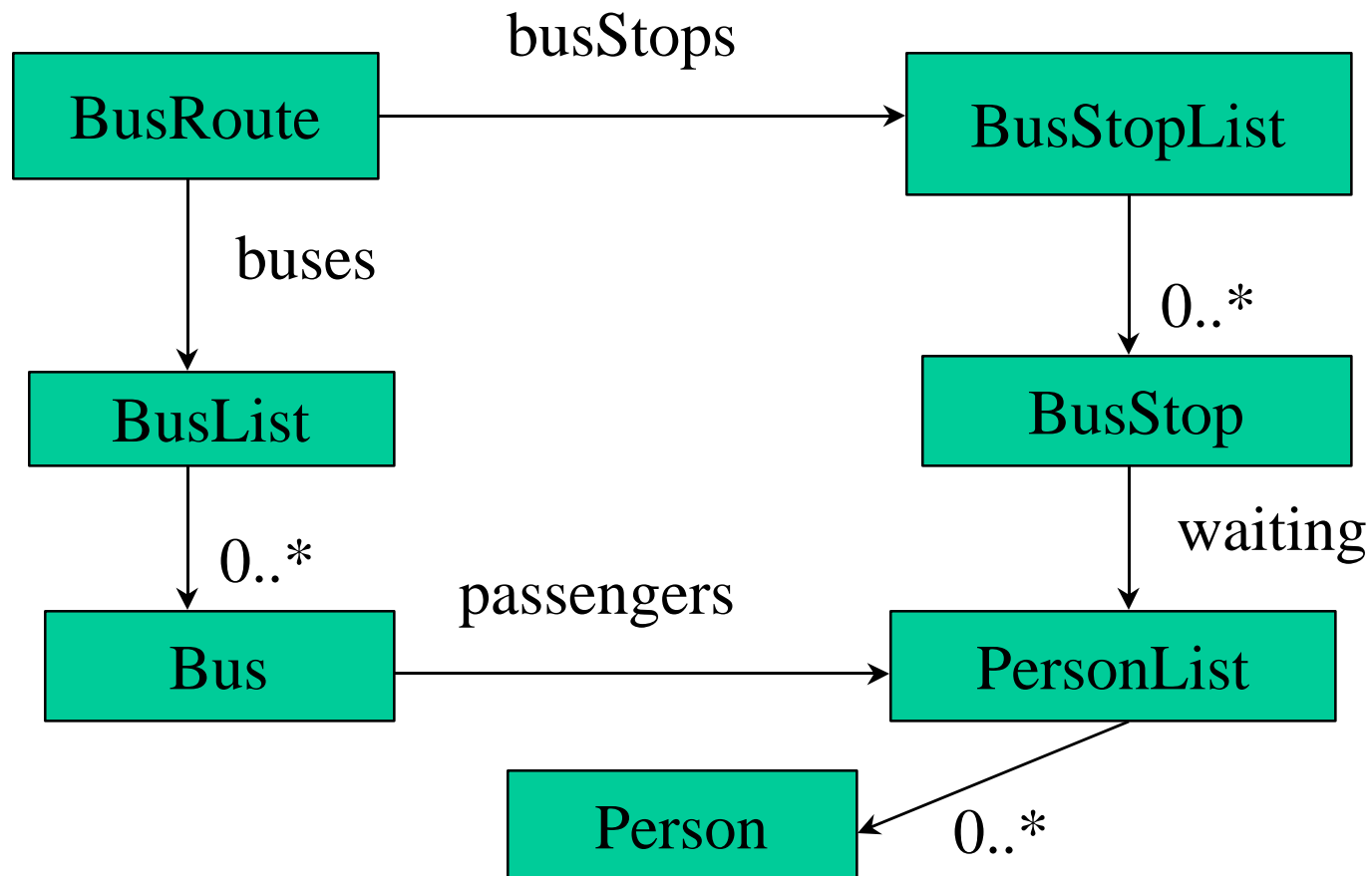
Overview this lecture

- Basic UML class diagrams
- Traversals/Collaborating classes
- Traversal strategy graphs
- Adaptive programming
- Tools for adaptive programming
- Demeter/Java and AP/Studio

1: Basic UML class diagrams

- Graph with nodes and directed edges and labels for nodes and edges
- Nodes: classes, edges: relationships
- labels: class kind, edge kind, cardinality

UML Class Diagram

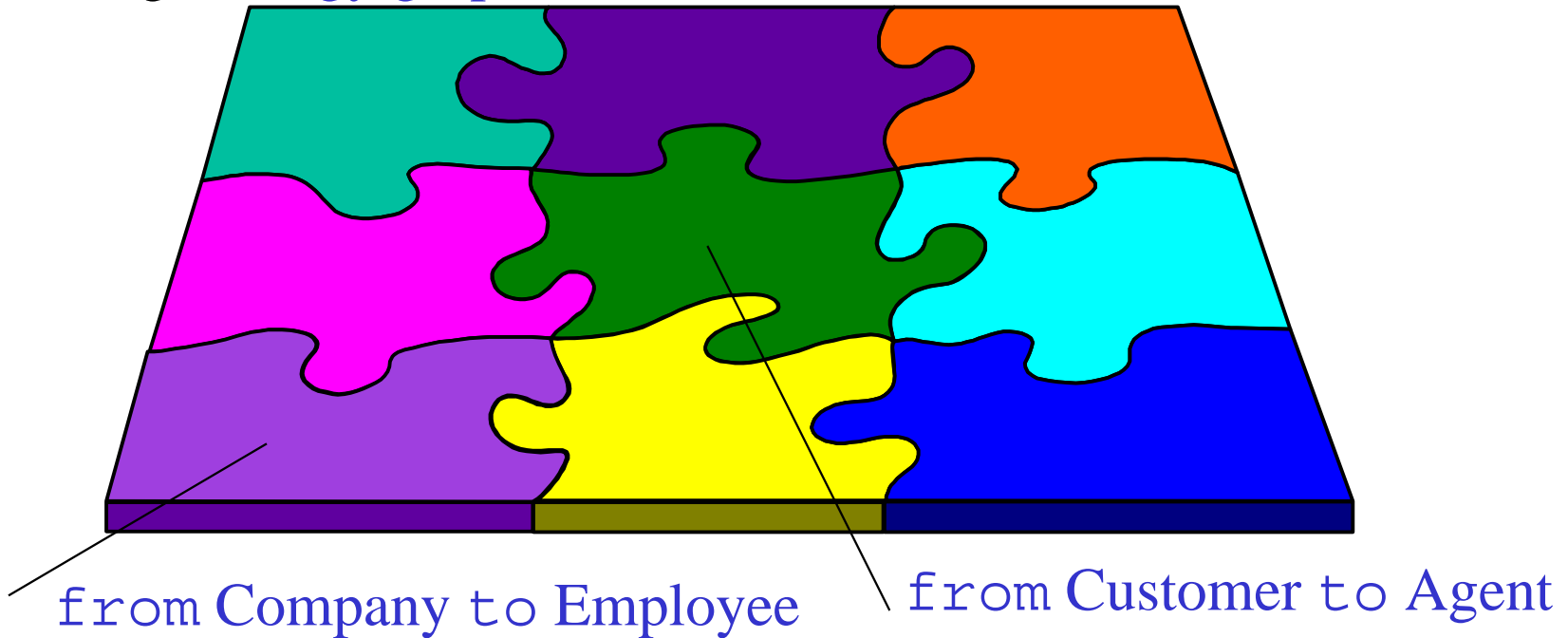


2: Traversals / Collaborating classes

- To process objects we need to traverse them
- Traversal can be specified by a group of collaborating classes

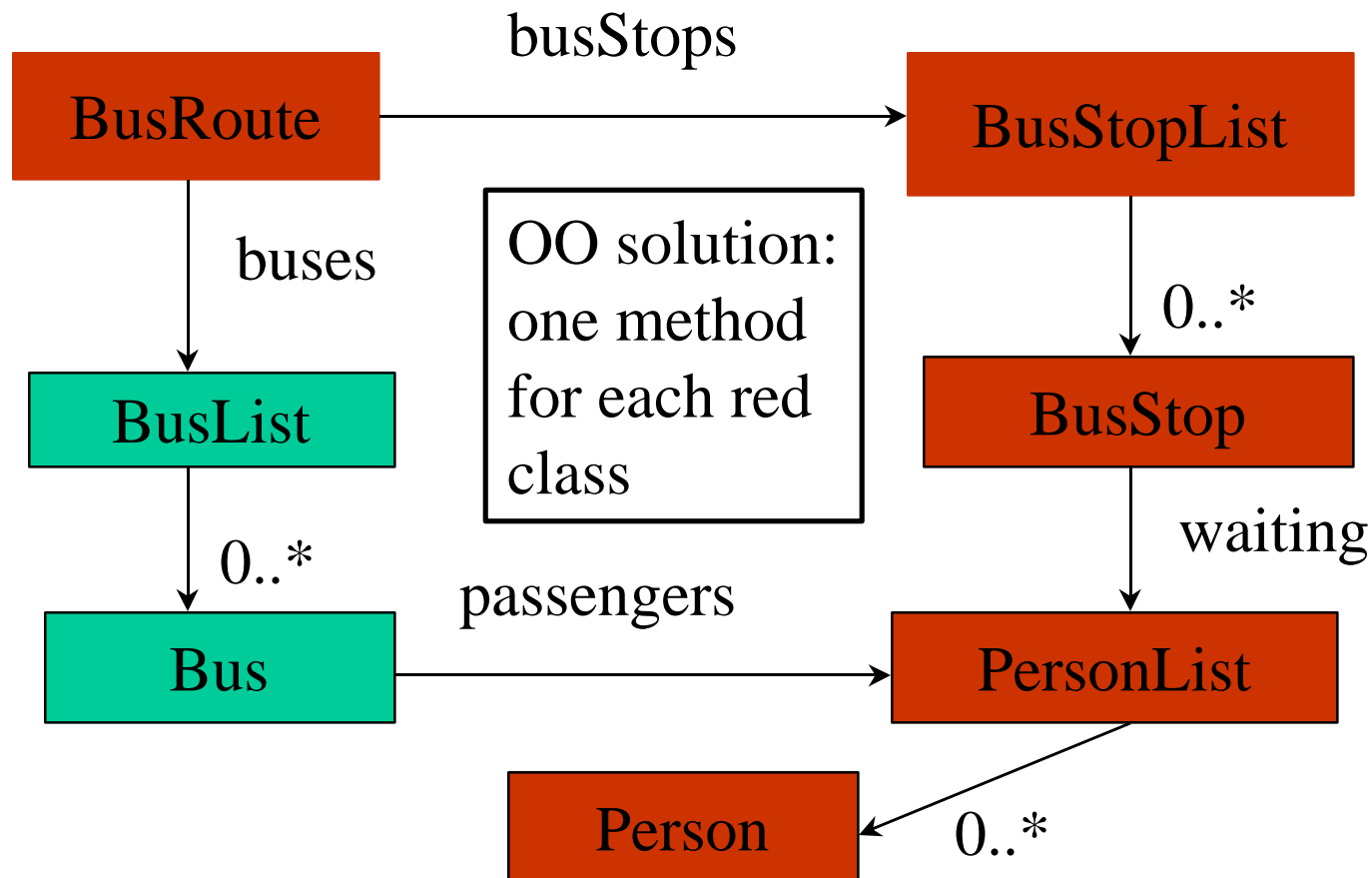
Collaborating Classes

use connectivity in class graph to define them succinctly
using *strategy graphs*



Collaborating Classes

find all persons waiting at any bus stop on a bus route



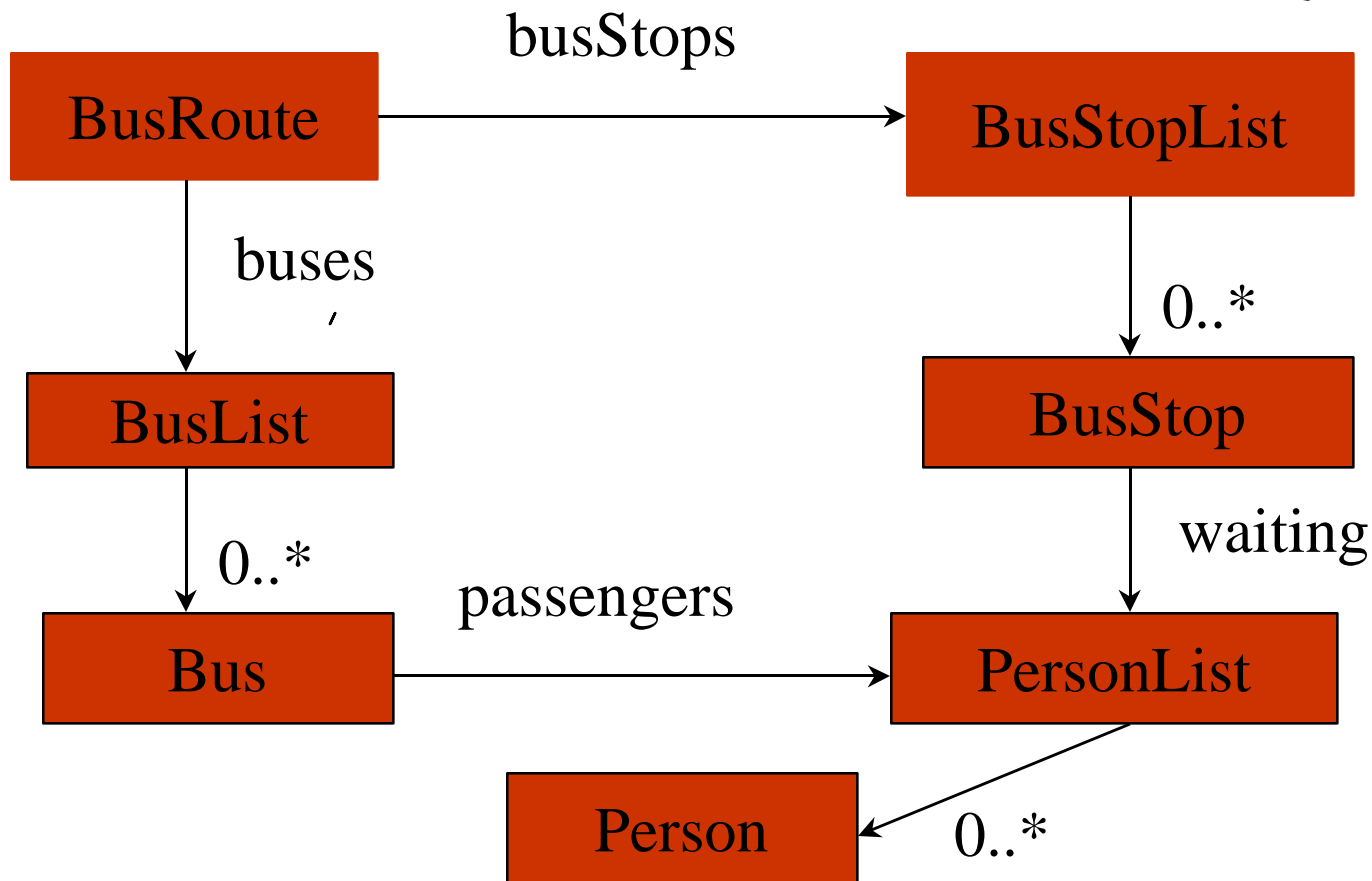
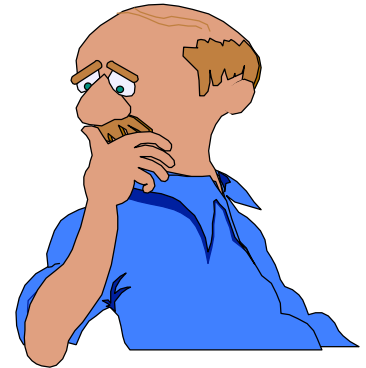
3: Traversal Strategy Graphs

- Want to define traversals succinctly
- Use graph to express abstraction of class diagram
- Express traversal intent: useful for documentation of object-oriented programs

find all persons waiting at any bus stop on a bus route

Traversal Strategy

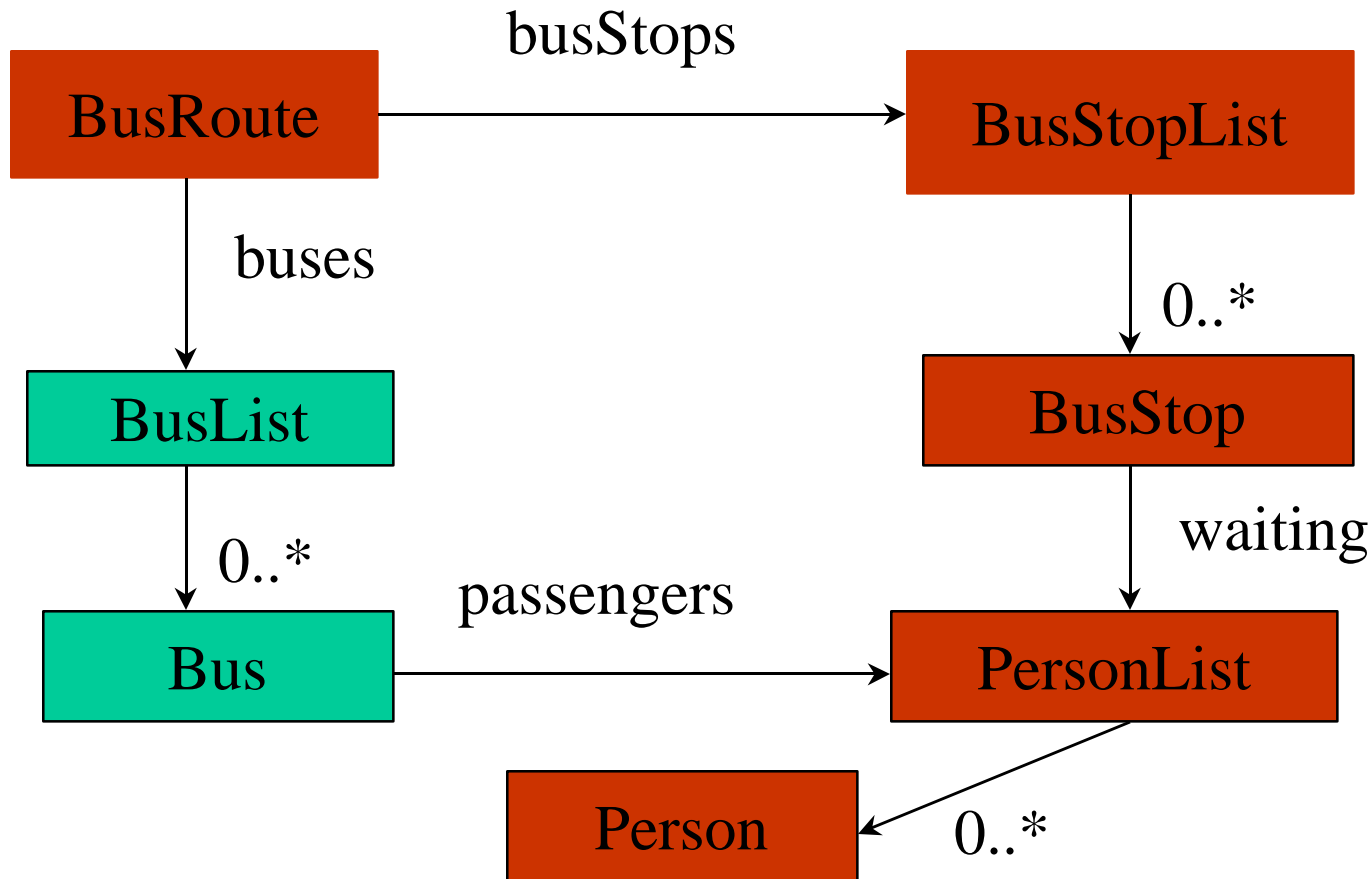
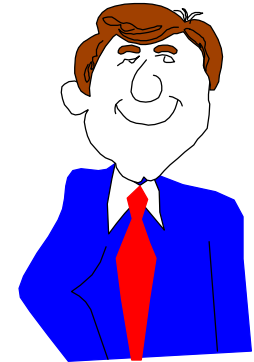
first try: from BusRoute to Person



find all persons waiting at any bus stop on a bus route

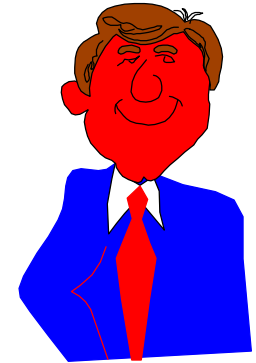
Traversal Strategy

from BusRoute through BusStop to Person

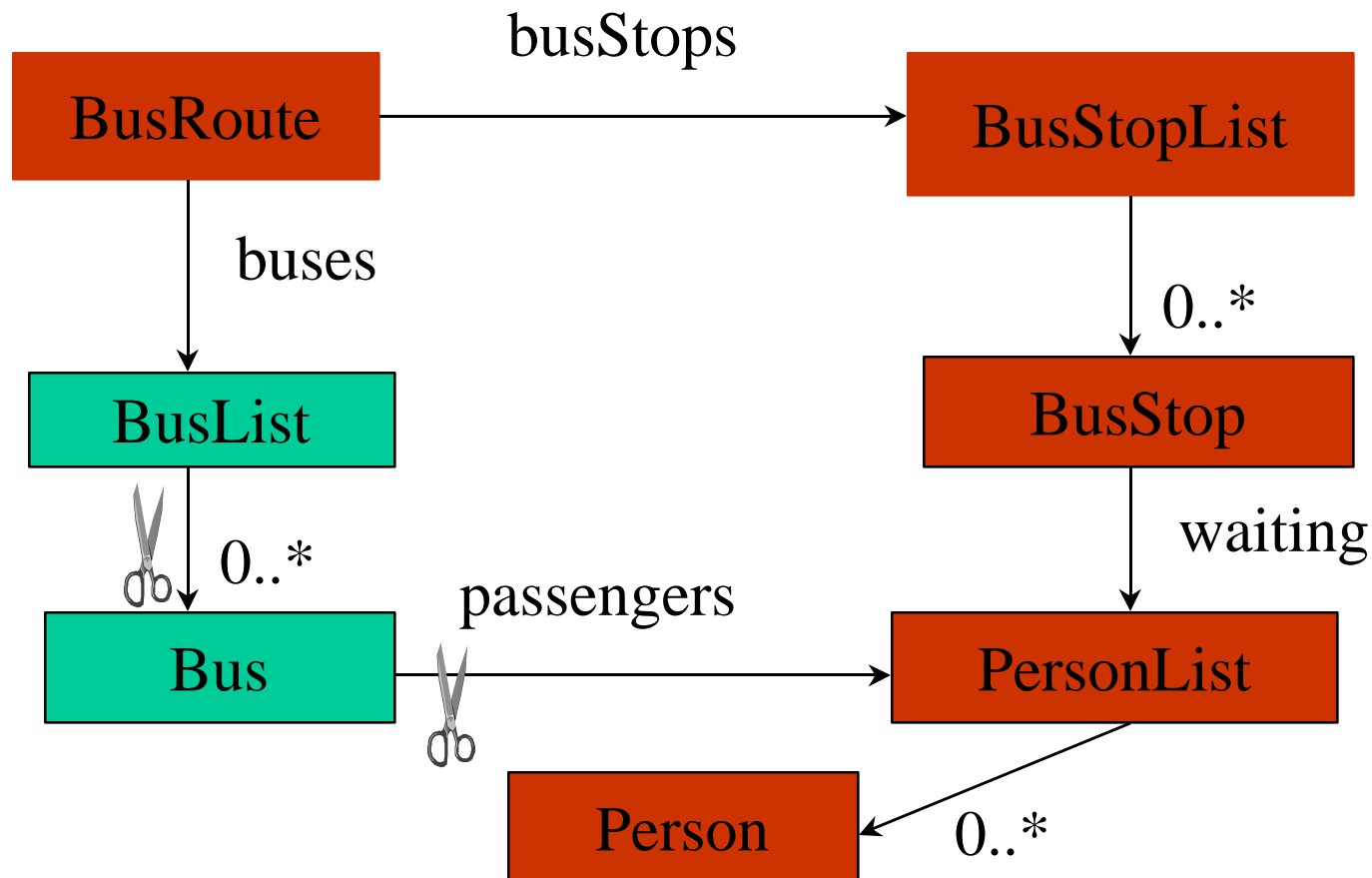


find all persons waiting at any bus stop on a bus route

Traversal Strategy



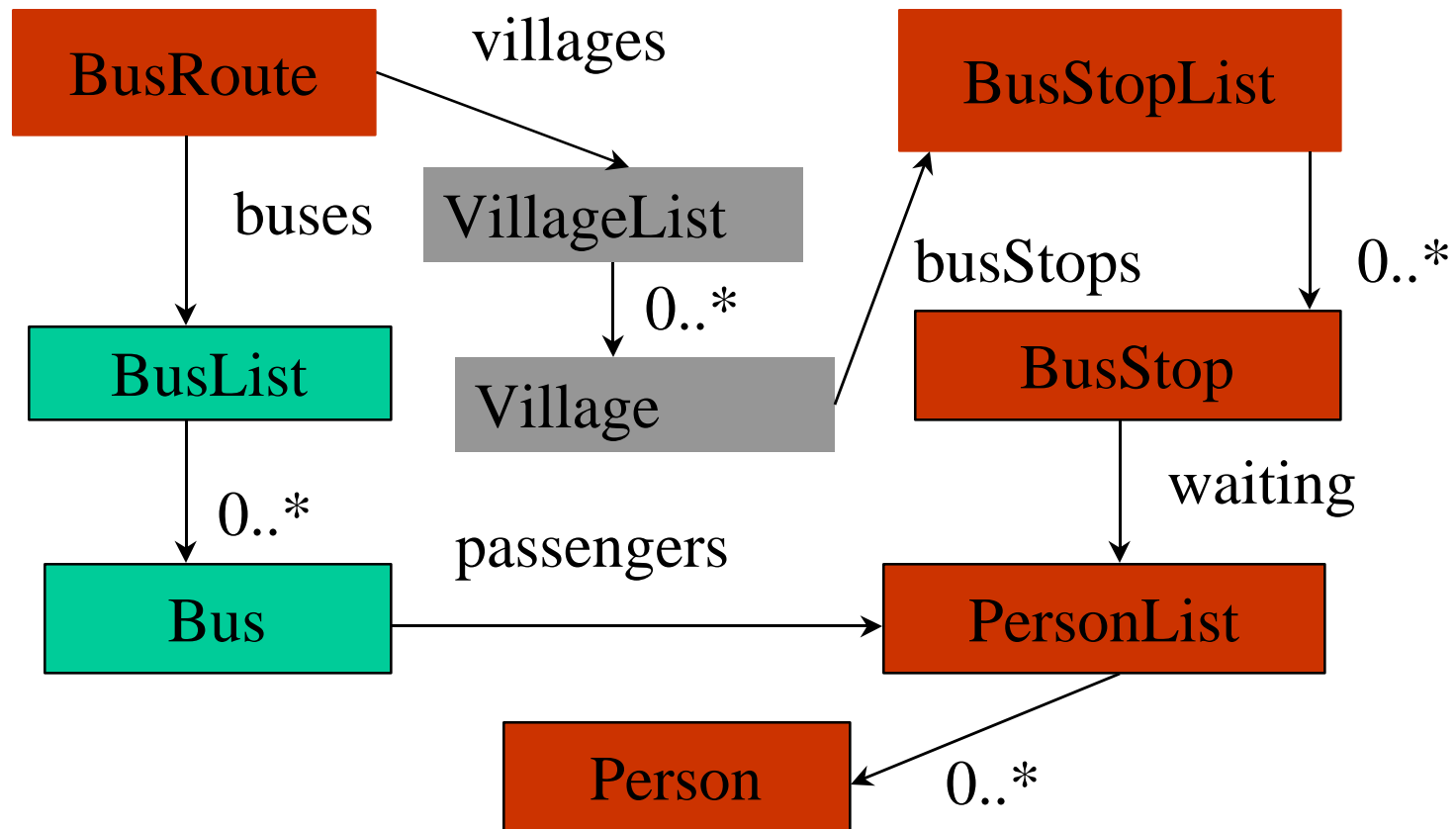
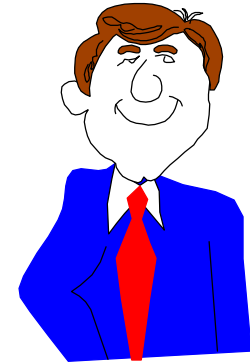
Altern.: *from BusRoute bypassing Bus to Person*



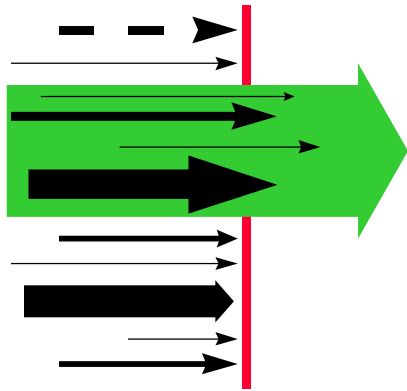
find all persons waiting at any bus stop on a bus route

Robustness of Strategy

from BusRoute bypassing Bus to Person



Filter out noise in class diagram



- only three out of seven classes are mentioned in **traversal strategy!**

from **BusRoute** through **BusStop** to **Person**

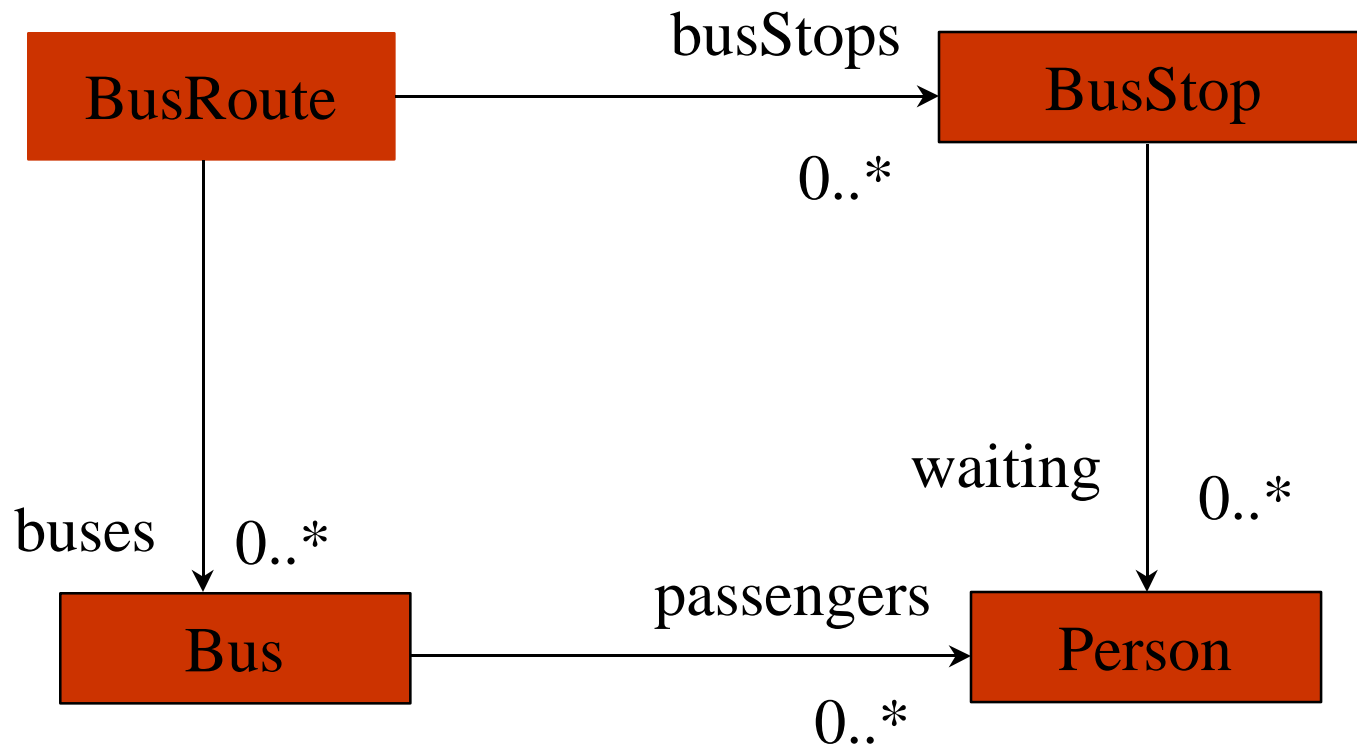
replaces traversal methods for the classes

BusRoute **VillageList** **Village** **BusStopList** **BusStop**
PersonList **Person**

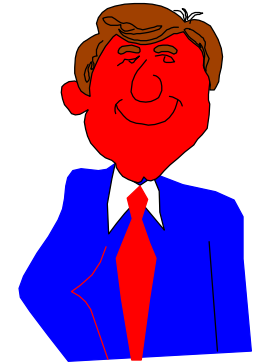
find all persons waiting at any bus stop on a bus route

Even better: interface class graph

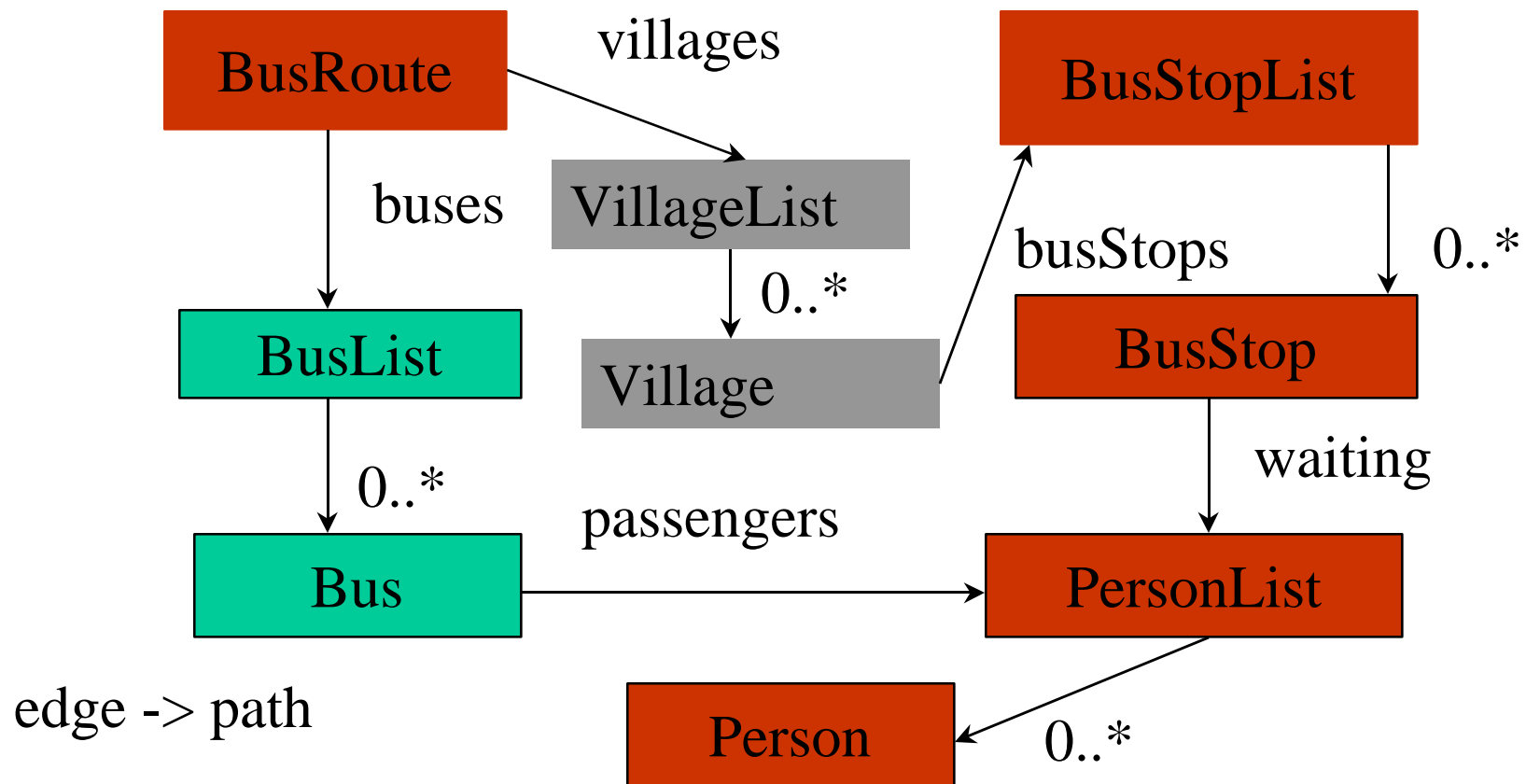
from BusRoute through BusStop to Person



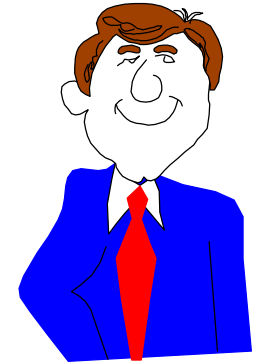
Map interface class graph to application class graph



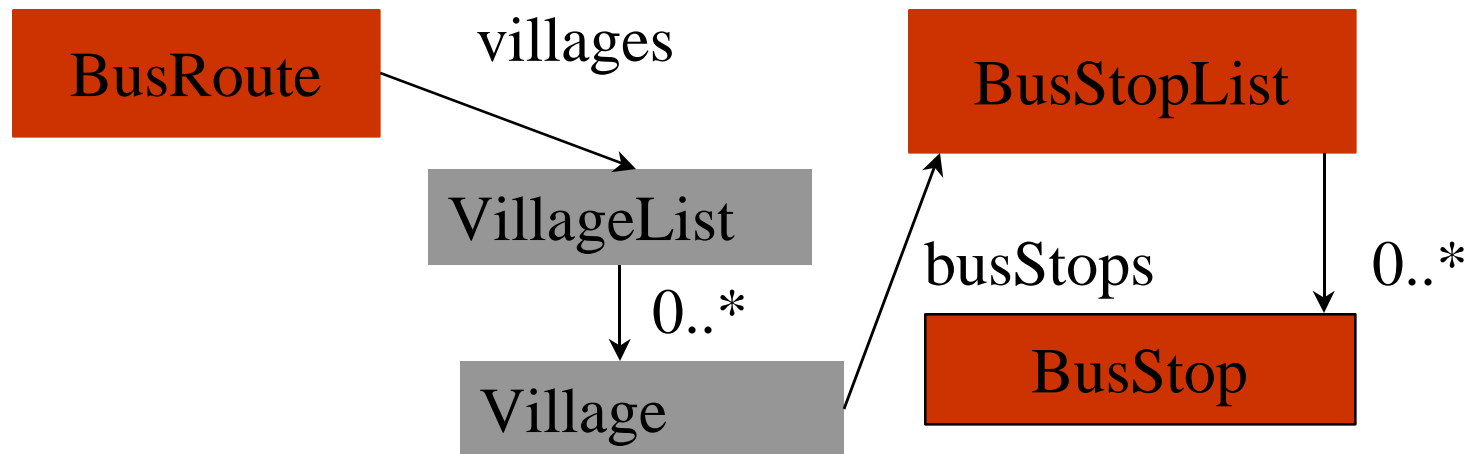
from BusRoute through BusStop to Person



Map interface class graph to application class graph



from `BusRoute` through `BusStop` to `Person`



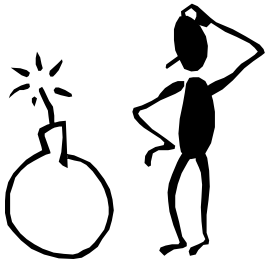
edge -> path

Benefits of interface class graph

- Shields program from details of application class graph
- Makes program more robust and simpler

Why Traversal Strategies?

- Law of Demeter: a method should talk only to its friends:
 - arguments and part objects (computed or stored)
 - and newly created objects



- Dilemma:
 - Small method problem of OO (if followed) or
 - Unmaintainable code (if not followed)



- Traversal strategies are the solution to this dilemma

4: Adaptive Programming

- How can we use strategies to program?
- Need to do useful work besides traversing: visitors
- Incremental behavior composition using visitors

Writing Adaptive Programs with Strategies

strategy: from BusRoute through BusStop to Person

```
BusRoute {
  traversal waitingPersons(PersonVisitor) {
    through BusStop to Person; } // from is implicit
  int printWaitingPersons() // traversal/visitor weaving instr.
    = waitingPersons(PrintPersonVisitor);
  PrintPersonVisitor {
    before Person (@ ... @) ... }
  PersonVisitor {init (@ r = 0; @) ... }
```

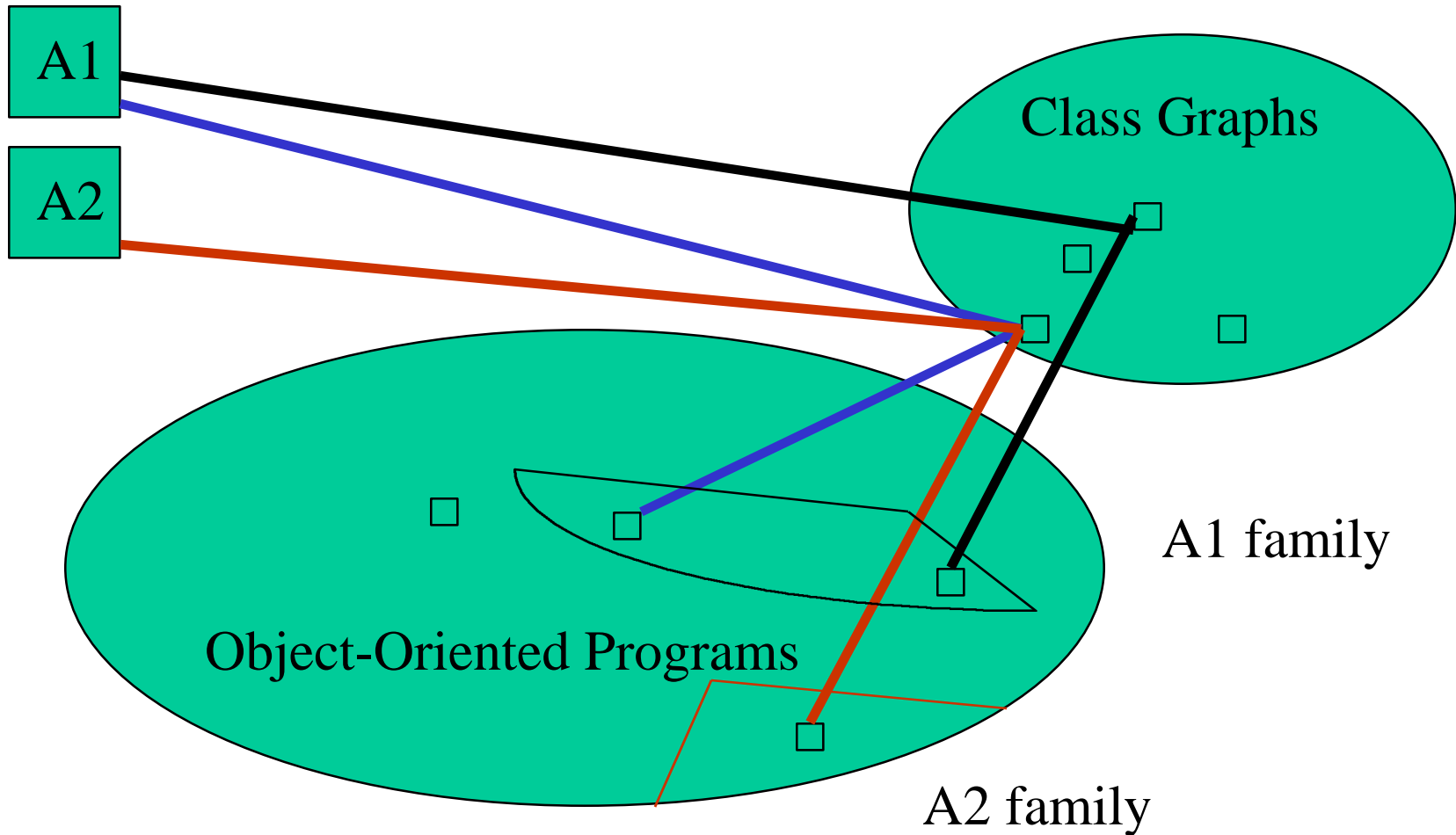
Extension of Java: keywords: traversal init
through bypassing to before after etc.

Taxi driver analogy

- Streets and intersections correspond to class graph
- Traversal strategy determines how the taxi will navigate through the streets
- You can take pictures before and after intersections
- You can veto sub traversals

Programming in Large Families

Two adaptive programs



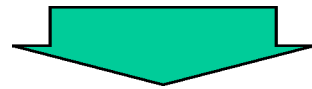
Adaptive Programming

Strategy Diagrams



are use-case based
abstractions of

Class Diagrams

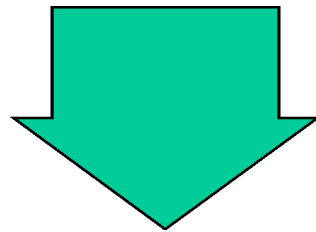


define family of

Object Diagrams

Adaptive Programming

Strategy Diagrams

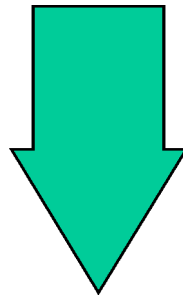


define traversals
of

Object Diagrams

Adaptive Programming

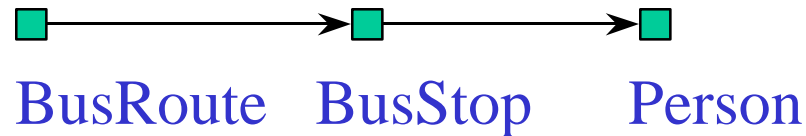
Strategy Diagrams



guide and
inform

Visitors

Strategy Diagrams



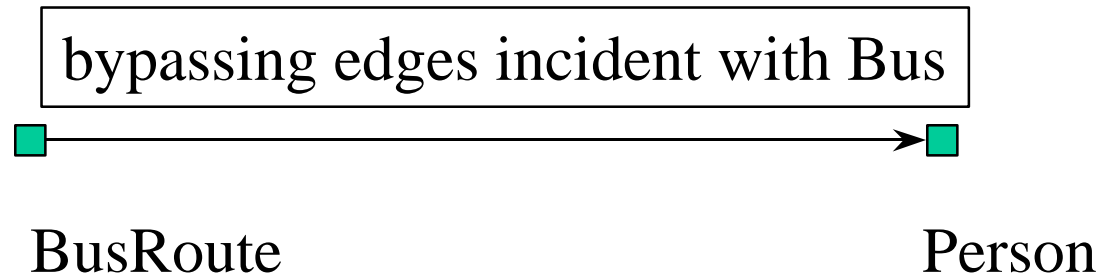
Nodes: positive information: Mark corner stones in class graph: Overall topology of collaborating classes. 3 nodes:

from BusRoute

through BusStop

to Person

Strategy Diagrams



Edges: negative information:
Delete edges from class diagram.

from BusRoute bypassing Bus to Person

5: Tools for Adaptive Programming

- free and commercial tools available

Free Tools on WWW

- Demeter/C++
- Demeter/Java
- Demeter/StKlos
- Dem/Perl5
- Dem/C++
- Dem/CLOS
- Demeter/Object Pascal



last four developed outside our group

Commercial Tools available on WWW



StructureBuilder from Tendril Software Inc.

next release will have support for traversals

www.tendril.com

Benefits of Demeter

- robustness to changes
- shorter programs
- design matches program
 - more understandable code
- partially automated evolution
- keep all benefits of OO technology
- improved productivity



Applicable to design and documentation
of your current systems.

Demeter/Java

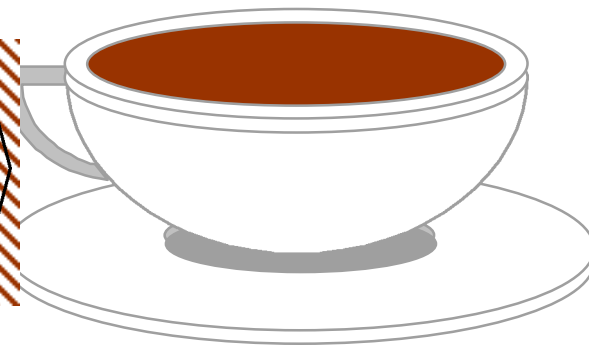
www.ccs.neu.edu/research/demeter

- class diagrams
- functionality
 - strategies
 - visitors
- etc.

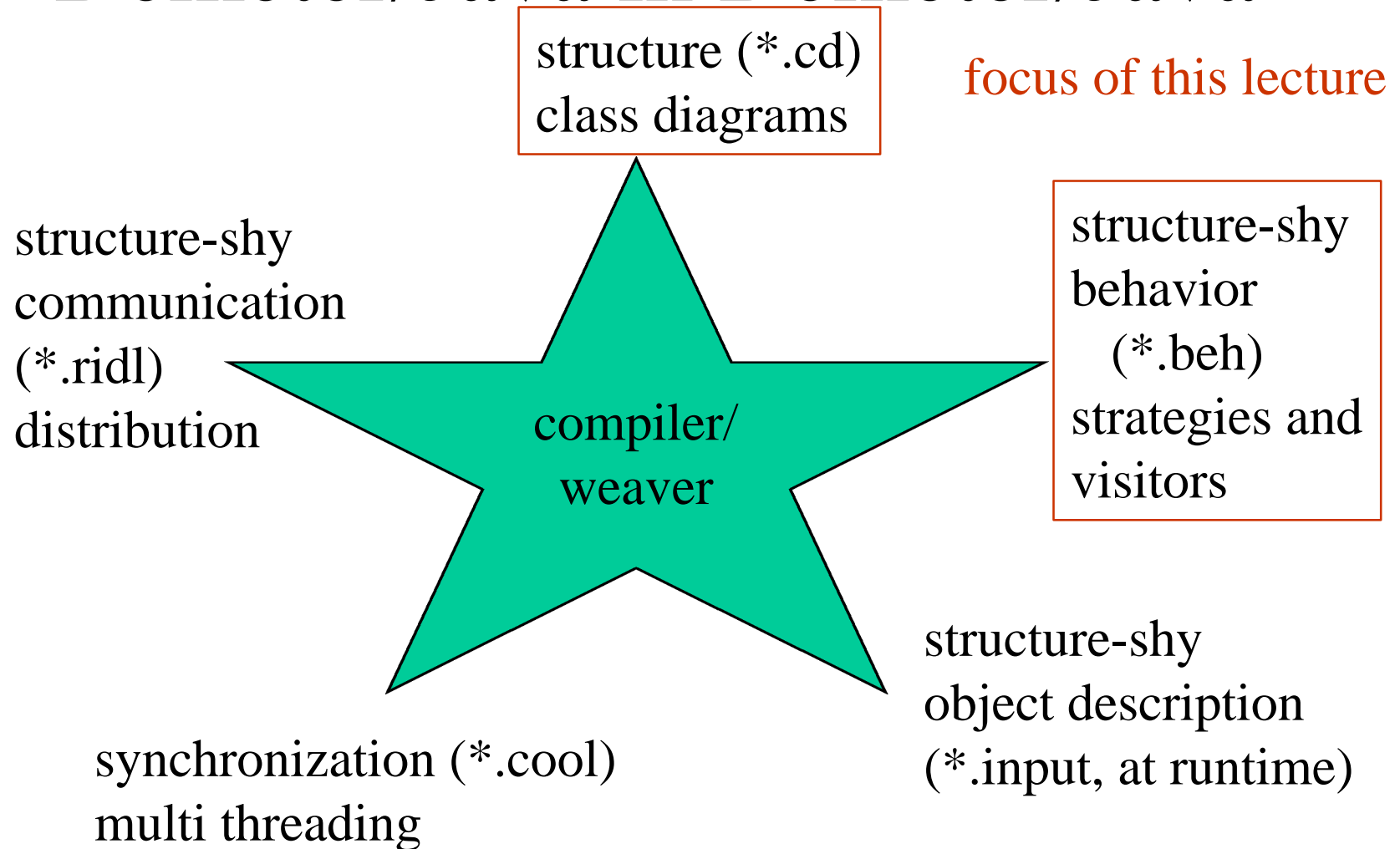
weaver



**Executable
Java code for your favorite
commercial Java Software
Development Environment**

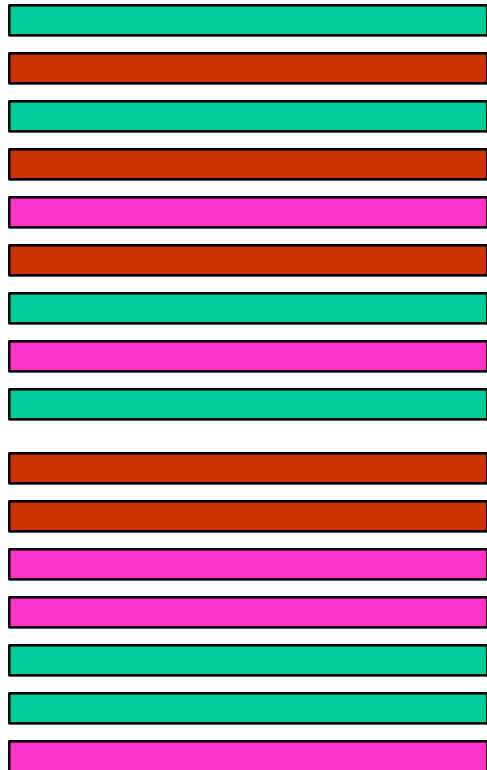


Demeter/Java in Demeter/Java

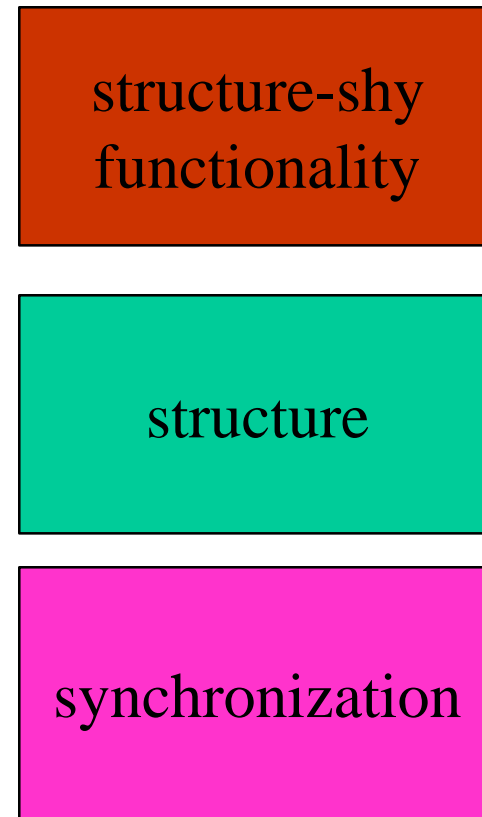


Cross-cutting in Demeter/Java

generated
Java program

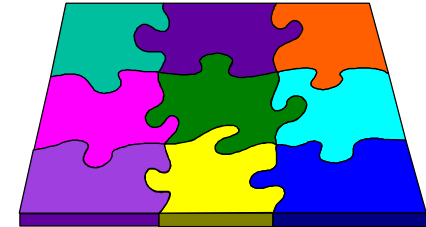


Demeter/Java program



replicated!

AP Studio



- visual development of traversal strategies relative to class diagram
- visual feedback about collaborating classes
- visual development of annotated UML class diagrams

Strengths of Demeter/Java

Theory

- Novel algorithms for strategies

- Formal semantics

- correctness theorems

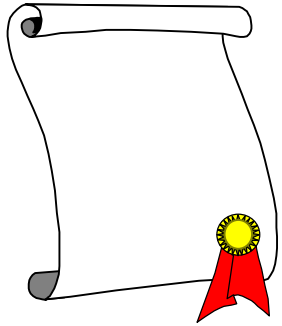
Practice

- Extensive feedback (8 years)

- Reflective implementation

Meeting the Needs

- Demeter/Java
 - Easier evolution of class diagrams (with strategy diagrams)
 - Easier evolution of behavior (with visitors)
 - Easier evolution of objects (with sentences)



Real Life

- Used in several commercial projects
- Implemented by several independent developers
- Used in several courses, both academic and commercial

Summary

- What has been learned: Simple UML class diagrams, strategies and adaptive programs
- How can you apply:
 - Demeter/Java takes adaptive programs as input
 - Document object-oriented programs with strategies
 - Design in terms of traversals and visitors

Where to get more information

- Adaptive Programming book
- UML Distilled
- Demeter/Java home page
- Course home page:

[www.ccs.neu.edu/research/demeter/
course/f98](http://www.ccs.neu.edu/research/demeter/course/f98)

Feedback

- Request feedback of training session