

Third lecture

- Review
- Visitor pattern
- Demeter/Java programming
 - Adaptive behavior
 - Debugging class dictionaries
- Traversal strategies

10/17/00

AOO/Demeter

1

Agenda for Adaptive Object-Oriented Software/Demeter

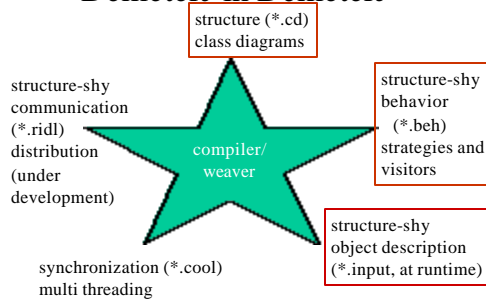
- ✓ Demeter/Java
- ✓ Java
- ✓ Java environment
- ✓ UML
- requirements
- domain analysis
- ✓ design
- ✓ implementation
- Adaptive Programming
- Aspect-Oriented Progr.
- principles
- heuristics
- patterns
- idioms
- theorems
- algorithms
- strategy graphs
- class graphs
- object graphs
- state graphs
- use cases
- interfaces
- Demeter Method
- iterative development
- spiral model
- ✓ traversals
- ✓ visitors
- packages

10/17/00

AOO/Demeter

2

DemeterJ in DemeterJ



10/17/00

AOO/Demeter

3

Design Patterns for Collaborative Behavior

- Navigate composite object structure
 - Iterator
 - Traversal
- Define task-specific behavior
 - Visitor

Visitor Pattern

- Intent: Modify the behavior of a group of collaborating classes without changing the classes.
- Examples:
 - Inventory: accumulate list of equipment
 - compute price
- In the following: Only Java, no Demeter

Visitor Pattern

```
abstract class EquipmentVisitor{
    abstract void visitCard(Card e);
    abstract void visitChassis(Chassis e);
    ...
}
```

Visitor Pattern

```
class PricingVisitor extends EquipmentVisitor{
  private Currency total;
  void VisitCard(Card e){total.add(e.NetPrice());}
  void VisitChassis(Chassis e)
    {total.add(e.DiscountPrice());}
  ...
}
```

Visitor Design Pattern

```
class InventoryVisitor extends EquipmentVisitor {
  Inventory inventory;
  void VisitCard(Card e) {inventory.accumulate(e);}
  void VisitChassis(Chassis e){
    inventory.accumulate(e);}
  ...
}
```

Visitor

```
class Card extends Equipment {
  void Accept(EquipmentVisitor v)
    {v.VisitCard(this);}
  ...
}
```

Visitor

```
class Chassis {
  void Accept(EquipmentVisitor v){
    v.VisitChassis(this);
    Enumeration e = parts.elements();
    while (e.hasMoreElements())
      ((Equipment) e.nextElement()).
        Accept(v);
  }
}
```

Visitor/Applicability

- Collaborative tasks
- Add behavior to composite structure

Visitor

```
AbstractVisitor: ConcreteVisitor1 |
                  ConcreteVisitor2.
  VisitConcreteA(ConcreteA);
  VisitConcreteB(ConcreteB);

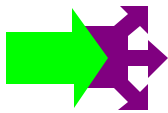
ConcreteA = ...
  Accept(Visitor v) {v.VisitConcreteA(this);
                    /* further traversal */ }
```

Visitor/Participants

- AbstractVisitor - empty visit operations
- ConcreteVisitor - task specific visit operation
- ConcreteElement - accept operation

Visitor

- Pros
 - Add behavior to composite class structure without cluttering class definitions
- Cons
 - Explicit traversal, hard-coding structure
 - Auxiliary structure, many empty methods



Visitor

- hard changes: modify class structure
 - add new concrete element class: need to update all visitors
 - rename parts
- easy changes: add a new visitor

Improve Visitor by

- Selective Visitor
 - before, after, around, not just accept
 - no empty methods
- Structure-shy Traversal
 - strategies



DemeterJ programming

- Now we focus on the structure and syntax of the Demeter/Java PL.
- It reuses Java unchanged. Java code between (@ and @) or {{ and }}.
- Provides syntax to define traversal strategies and visitors and how to glue the three together.

DemeterJ programming

- for simple use: prepare program.cd, program.beh, program.input
- call demjava new (on UNIX and Windows): generates program.prj : be careful: program.prj~
- demjava test
- demjava clean if you (and the system) are confused. Regenerates and recompiles.

DemeterJ programming

- Style for calling Java program: always use a class Main which contains a static public void main function.
- `java -classpath gen/classes :... demjava.jar ... Main < program.input`
(Windows: `/->\ :->;`)
- `demjava test` will do the right thing

10/17/00

AOO/Demeter

19

DemeterJ programming

- `program.cd`
 - start it with a package name: package X; add package name to `program.prj` file. Call `java X.Main < program.input`
 - import classes: `import java.util.*`
 - terminate the start class with `*EOF*` (make sure class is not recursive)
 - See DemeterJ class dictionary for syntax (see DemeterJ AP Studio resource page)

10/17/00

AOO/Demeter

20

DemeterJ programming

- can also use look-ahead definitions as in Java Compiler Compiler. See Java class dictionary off DemeterJ resource page. Also Java Compiler Compiler home page.
- use look-ahead definitions only when absolutely needed. Most of the time you don't need them. I avoid them.

10/17/00

AOO/Demeter

21

DemeterJ programming

- Continue with programming of behavior

10/17/00

AOO/Demeter

22

Behavior

- Study different method kinds which DemeterJ supports.

10/17/00

AOO/Demeter

23

DemeterJ 96/97

- Methods
 - verbatim
 - traversal
 - before, after
- Strategies: bypassing ... to {A,B,C}



10/17/00

AOO/Demeter

24

DemeterJ 96/97

- Node and edge methods in visitors
 - define same code for multiple nodes and edges
- Attach methods to node sets
- Support for external Java classes
- Parsing support

10/17/00

AOO/Demeter

25

DemeterJ 97/98

- AP-Studio: visualize class graphs/traversals
- Adaptive methods
- Around methods
- Return, init methods
 - make visitors self-contained
- General strategies
- Traversals as objects



10/17/00

AOO/Demeter

26

DemeterJ 97/98

- Generic visitors: DisplayVisitor, etc.
- Derived edges
 - computed parts, instead of stored parts
 - traversal invokes computation
- Synchronization aspect
 - besides *.cd, *.beh also *.cool files
- LL(k) support for parsing

10/17/00

AOO/Demeter

27

DemeterJ 98/99



- Remote invocation aspect Ridl
- New front end: Uniform interface for all platforms using *.prj file.
- Weaver: support for compile-time weaving of Java code. Generator now produces *.wvr files
- demjava new: produces sample file

10/17/00

AOO/Demeter

28

DemeterJ 98/99



- Traversal graphs now printed to output directory
 - <traversal-name>_<source-class>.trv
- Is in form of a subgraph; may lose information. But accurate in many cases.
- New terminals: Line and Word
 - Line: everything up to next newline
 - Word: everything up to next whitespace

10/17/00

AOO/Demeter

29

DemeterJ 98/99



- Start and finish methods in visitor classes
- No more stars: *common* -> common
- DJ

10/17/00

AOO/Demeter

30

Behavior

- verbatim: Java. Typical pattern:
 - instantiate visitors
 - establish communication between visitors
 - call traversal with visitors
 - return result based on result in visitors



Verbatim methods

```
int count(...) throws Exception1, ...
    (@
    ...
    @)

int count(...) throws Exception1, ...
    {{
    ...
    }}
```

Behavior

- traversal method
 - combines strategy and visitor class and gives combination a name
 - allows subclassing of visitor arguments
 - traversal(V) {to Z;}

Behavior

- adaptive method
 - handles visitor instantiation and returns
 - 4 kinds of adaptive methods
 - traversal method/visitor class names
 - traversal method/inlined visitor
 - strategy/visitor class names
 - strategy/inlined visitor

Adaptive Methods

- strategy/visitor class names: often used
 - no need to reuse visitor objects?
 - no need to reuse strategy/visitor combination?
 - int countInhRels() via Alternat to Vertex (CountingVisitor);
 - return value of first visitor class
 - works only if visitors are not nested
 - encourages self-contained visitor classes

Adaptive Methods

- strategy/inlined visitor: often used
 - no need to reuse visitor objects?
 - no need to reuse visitor class?
 - int countInhRels() via Alternat to Vertex
 - { (@ int total; @)
 - init (@ total = 0; @); ...
 - return int (@ total @)}
 - PROPAGATION PATTERN STYLE in book

Adaptive Methods

- traversal method/visitor class names
 - no need to reuse visitor objects?
 - `int countInhRels() = allInh(CountingVisitor);`
 - `A f() = tm(V1,V2,V3);`
 - return value of first visitor class
 - works only if visitors are not communicating
 - encourages self-contained visitor classes

10/17/00

AOO/Demeter

37

Adaptive Methods

- traversal method/inlined visitor
 - no need to reuse visitor objects?
 - no need to reuse visitor class?
 - `int countInhRels() allInh`
 - `{ (@ int total; @)`
 - `init (@ total = 0; @); ...`
 - `return int (@ total @);`
 - on-the-fly visitor definition, no reuse intended

10/17/00

AOO/Demeter

38

Visitor class methods

- Goal: make visitors self-contained
 - wrapper methods: *before, after, around*
 - for nodes and construction edges
 - constructor *init, return*
 - *start, finish*

10/17/00

AOO/Demeter

39

Around methods

- Conditional traversal
- around A
 - `(@ ... subtraversal.apply(); ... @)`
- sometimes multiple subtraversals: see DemeterJ implementation.

10/17/00

AOO/Demeter

40

Visitor use in Adaptive Methods

- In *.cd
 - `visitor className = ...`
 - `visitors ... endvisitors`
- makes visitor class inherit from UniversalVisitor
- if you define visitor classes that do not inherit from UniversalVisitor, you need to provide *start, finish, return*

10/17/00

AOO/Demeter

41

Around Methods

- CondAVisitor {
 - around A (@
 - if (flag)
 - `{... subtraversal.apply(); ...}`
 - else {...} @)

10/17/00

AOO/Demeter

42

Around Methods

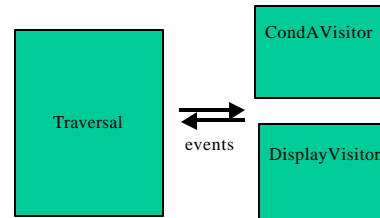
- CondAVis = <flag> boolean.
- void condADisplay(boolean c)
(@
 ConDAVis cv = new CondAVis(c);
 DisplayVisitor dv = new DisplayVisitor();
 this.t(cv, dv);
@)

10/17/00

AOO/Demeter

43

Event View



Conditional forwarding of events

10/17/00

AOO/Demeter

44

Around Methods for Control

- OnlyOnceVisitor { // prevents infinite loops
 around *
 (@ if (!host.get_visited()) {
 host.set_visited(true);
 subtraversal.apply();
 @})
• traverse(OnlyOnceVisitor, PrintVisitor);

10/17/00

AOO/Demeter

45

OnlyOnceVisitor

- Many uses: should be in visitor library?
 void safePrint() =
 trav(OnlyOnceVisitor, PrintVisitor);
- Several implementations:
 - intrusive: hosts must store visited flag.
 - unintrusive: hosts are not changed, use hash table instead.

10/17/00

AOO/Demeter

46

Around Methods for Control

- Control multiple visitors
- traversal.allSal(CondTravVisitor,
 CountingVisitor, SummingVisitor);
- CondTravVisitor {
 (@ boolean inclManagers; @)
 around Manager
 (@ if(inclManagers) subtraversal.apply(); @)}
- comp.allSal(new CondTravVisitor(false), ...)

10/17/00

AOO/Demeter

47

Implementation of around methods

- Uses reflection
- third design
 - demjava generated class for each around method
 - inner classes (also generated class for each around method)
 - uses reflection, uses only an object for each around method

10/17/00

AOO/Demeter

48

Visitor programming camps

- One visitor camp, with subclassing
 - Includes the propagation pattern camp
 - DemeterJ implementors
- Multiple visitors camp, with subclassing
 - Multiple visitors allow to modularize behavior
 - AverageVisitor contains SumVisitor and CountVisitor.
 - parameterization would help

10/17/00

AOO/Demeter

49

Visitors in class dictionary

- Why are visitor classes in class dictionary?
 - They are classes.
 - We want to traverse them, print them, etc.
- Why mark them as visitors?
 - inherit from UniversalVisitor
- UniversalVisitor (abstract class)
 - empty methods for nodes and constr. edges

10/17/00

AOO/Demeter

50

Multiple visitors example

- Print edges in class dictionary
- traversal toAllElements(
EdgeDistinctionVisitor,
ClassNameTranspVisitor,
EdgeVisitor)
via ClassDef to {Subclass, Part};

10/17/00

AOO/Demeter

51

Highlights of class graph

```
ClassGraph = List(ClassDef).
ClassDef = ClassName ClassParts.
ClassParts : ConstructionClass | AlternationClass
common <parts> List(Part).
Part = PartName ClassName.
ConstructionClass = .
AlternationClass = List(Subclass).
Subclass = ClassName.
ClassName = <v>Ident. PartName = <v>Ident.
```

10/17/00

AOO/Demeter

52

Even simpler class graph

```
ClassGraph = List(ClassDef).
ClassDef = ClassName List(Part) List(Subclass).
Part = ClassName.
Subclass = ClassName.
ClassName = <v>Ident.
List(S) ~ {S}.
```

10/17/00

AOO/Demeter

53

Multiple visitors example

- EdgeDistinctionVisitor =
- ClassNameTranspVisitor =
<cn> ClassName.
- EdgeVisitor =
<cntv> ClassNameTranspVisitor.

10/17/00

AOO/Demeter

54

Multiple visitors

- EdgeVisitor {
 before {Part, Subclass}
 (@ // print pairs ...
 this.dig_out().get_name() ...
 @)}

10/17/00

AOO/Demeter

55

dig_out

```
EdgeVisitor {  
  ClassName dig_out() to ClassName  
  {beforeClassName(@ return_val = host; @)}  
} // no data member name revealed  
// but we might need a more complex strategy
```

It is useful to traverse visitors. Alternative would be:
ClassName dig_out()
{return this.get_x().get_y().get_z().get_className();}
VIOLATES LAW OF DEMETER, hard to maintain

10/17/00

AOO/Demeter

56

Multiple visitors

- EdgeDistinctionVisitor{
 before Part
 (@ /* construction edge */ @)
 before Subclass
 (@ /* alternation edge */ @)
 }

10/17/00

AOO/Demeter

57

Multiple visitors

- ClassNameTranspVisitor{
 before ClassDef
 (@ this.set_cn(...); @)
 }

10/17/00

AOO/Demeter

58

Two visitors

- independent
 - CountVisitor, PrintVisitor
- dependent
 - nested
 - AverageVisitor contains CountVisitor, SumVisitor
 - Visitor communication

10/17/00

AOO/Demeter

59

Universal traversals

- to *: can use any number of visitors
 - generates a new traversal method (collection of Java methods) for each use

10/17/00

AOO/Demeter

60

UniversalVisitor

- notparsed visitor UniversalVisitor
: PrintVisitor | CopyVisitor | DisplayVisitor
...

Expanded class dictionary file

- gen/program.xcd
 - expanded parameterization
 - added inheritance edges
 - no flattening
 - converted repetition classes
 - filled part names
 - added generic visitors

End of viewgraphs