

## Design Patterns, cont.

## Announcements

- HW7, Due Friday November 20
  - Scroll down Details if errors on instructor tests
- Grades and feedback for HW0-5 in Homework Server
- Exam1-2, Quiz1-7 in LMS
- Rainbow grades coming up

2

## Design Patterns So Far

- Creational patterns: **Factories, Prototype, Singleton, Interning**
  - Problem: constructors in Java (and other OO languages) are inflexible
    1. Can't return a subtype of the type they belong to. "Factory" patterns address the issue: Factory method (e.g. `createBicycle()`), Factory class/object, Prototype
    2. Always return a **fresh new object**, can't reuse. "Sharing" patterns address the issue: Singleton, Interning

Fall 15 CSCI 2600, A Milanova

3

## Design Patterns Summary so Far

- Wrappers: **Adapter, Decorator, Proxy**
  - Structural patterns: when we want to change interface or functionality of an existing class, or restrict access to an object
- **Composite**
  - A structural pattern: expresses whole-part structures, gives uniform interface to client
- Started patterns for traversal of composites: **Interpreter, Procedural and Visitor**

Fall 15 CSCI 2600, A Milanova

4

## Outline of Today's Class

- Behavioral patterns for traversing composites
  - Interpreter
  - Procedural
  - Visitor
- Behavioral pattern for traversal of containers
  - Iterator
- A Design Exercise

5

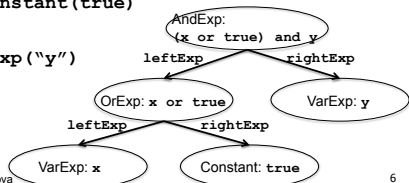
## Composite Objects

- Expression **(x or true)** and **y**

```
new AndExp (
```

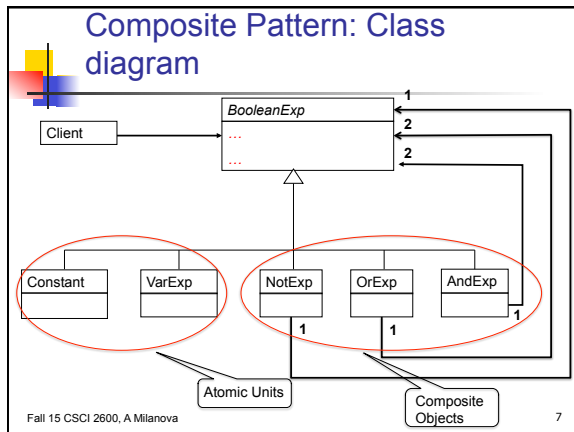
```
    new OrExp (
        new VarExp("x"),
        new Constant(true)
    ),
    new VarExp("y")
)
```

We have a **hierarchical structure**: AndExp is top, OrExp and VarExp are below in the hierarchy, etc.



Fall 15 CSCI 2600, A Milanova

6



### Traversing Composites

- Question: How to perform operations on composite objects (on all parts of the component)?
- The Interpreter, Procedural and Visitor patterns address this question

Fall 15 CSCI 2600, A Milanova 8

### Operations on Boolean Expressions

- Need to write code for each Operation/Object pair

	VarExp	Constant	AndExp	OrExp	NotExp
evaluate					
pretty-print					

Operations

- Question: do we group together (in a class) the code for a particular object or the code for a particular operation?

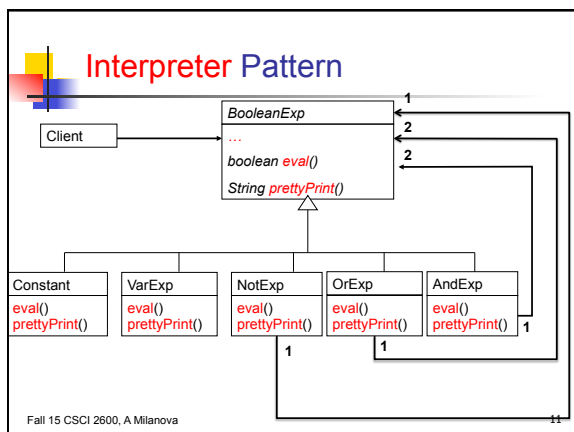
Fall 15 CSCI 2600, A Milanova 9

### Interpreter and Procedural Patterns

- Interpreter:** groups code for same **object**, spreads apart code for similar operations
- Procedural:** groups code for similar **operations**, spreads apart code for same object

	VarExp	AndExp
evaluate		
pretty-print		

Fall 15 CSCI 2600, A Milanova (based on slide by Michael Ernst) 10



### Interpreter Pattern

```

abstract class BooleanExp {
    boolean eval(Context c);
}

class Constant extends BooleanExp {
    private boolean const;
    Constant(boolean const) { this.const=const; }
    boolean eval(Context c) { return const; }
}

class VarExp extends BooleanExp {
    String varname;
    VarExp(String var) { varname = var; }
    boolean eval(Context c) { return c.lookup(varname); }
}

```

Fall 15 CSCI 2600, A Milanova 12

## Interpreter Pattern

```
class AndExp extends BooleanExp {
    private BooleanExp leftExp;
    private BooleanExp rightExp;
    AndExp(BooleanExp left, BooleanExp right) {
        leftExp = left;
        rightExp = right;
    }
    boolean eval(Context c) {
        return leftExp.eval(c) && rightExp.eval(c);
    }
}

// analogous definitions for OrExp and NotExp
```

Fall 15 CSCI 2600, A. Milanova

13

## Procedural Pattern

```
// Classes for expressions don't have eval!

class Evaluate {
    Context c;
    boolean evalConstExp(Constant c) {
        c.value(); // returns value of constant
    }
    boolean evalAndExp(AndExp e) {
        BooleanExp leftExp = e.leftExp();
        BooleanExp rightExp = e.rightExp();

        //Problem: How to invoke the right
        //implementation for leftExp and rightExp?
    }
    // also, evalVarExp, evalOrExp, evalNotExp
}
```

14

## Procedural Pattern

```
class Evaluate {
    Context c;
    ...
    boolean evalExp(BooleanExp e) {
        if (e instanceof VarExp)
            return evalVarExp((VarExp) e);
        else if (e instanceof Constant)
            return evalConstExp((Constant) e);
        else if (e instanceof OrExp)
            return evalOrExp((OrExp) e);
        else ...
    }
}
```

What is the problem with this code?

Fall 15 CSCI 2600, A. Milanova

15

## Visitor Pattern, a Variant of the Procedural Pattern

- Visitor helps traverse a hierarchical structure
- Nodes (objects in the hierarchy) **accept** visitors
- Visitors **visit** nodes (objects)

```
class SomeBooleanExp extends BooleanExp {
    void accept(Visitor v) {
        for each child of this node {
            child.accept(v);
        }
        v.visit(this);
    }
}

class Visitor {
    void visit(SomeBooleanExp e) { do work on e }
}
```

*n.accept(v) traverses the structure rooted at n, performing v's operation on every element*

Fall 15 CSCI 2600, A. Milanova (based on slide by Michael Ernst)

16

## Visitor Pattern

```
class VarExp extends BooleanExp {
    void accept(Visitor v) {
        v.visit(this);
    }
}

class AndExp extends BooleanExp {
    BooleanExp leftExp;
    BooleanExp rightExp;
    void accept(Visitor v) {
        leftExp.accept(v);
        rightExp.accept(v);
        v.visit(this);
    }
}

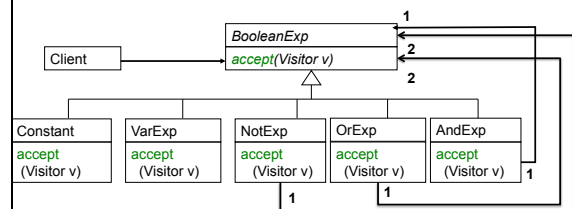
class Evaluate implements Visitor {
    // state, needed to
    // evaluate
    void visit(VarExp e) {
        //evaluate Var exp
    }
    void visit(AndExp e) {
        //evaluate And exp
    }
    ...
}

class PrettyPrint implements Visitor {
    ...
}
```

Fall 15 CSCI 2600, A. Milanova

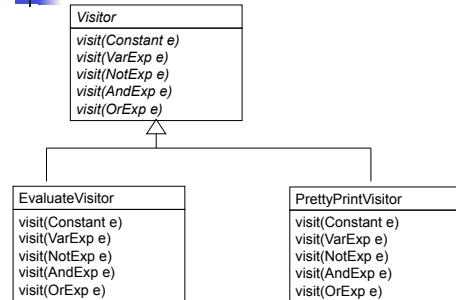
17

## The Visitor Pattern



18

## The Visitor Pattern



Fall 15 CSCI 2600, A Milanova

19

## Question

```

class VarExp extends BooleanExp {
    void accept(Visitor v) {
        v.visit(this);
    }
}
class Constant extends BooleanExp {
    void accept(Visitor v) {
        v.visit(this);
    }
}

```

1. Why not move void accept(Visitor v) up into superclass BooleanExp?

```

class CounterVisitor implements Visitor {
    int count = 0;
    void visit(VarExp e) {
        //??
    }
    void visit(Constant e) {
        //??
    }
    void visit(AndExp e) {
        //??
    }
    ...
}

```

20

## Exercise: Write Count Visitor which counts #subexpressions in a BooleanExp object

```

class VarExp extends BooleanExp {
    void accept(Visitor v) {
        v.visit(this);
    }
}
class AndExp extends BooleanExp {
    BooleanExp leftExp;
    BooleanExp rightExp;
    void accept(Visitor v) {
        leftExp.accept(v);
        rightExp.accept(v);
        v.visit(this);
    }
}

```

```

class CounterVisitor implements Visitor {
    int count = 0;
    void visit(VarExp e) {
        //??
    }
    void visit(Constant e) {
        //??
    }
    void visit(AndExp e) {
        //??
    }
    ...
}

```

21

## Starting the Visitor

```

BooleanExp myExp =
    new AndExp(
        new OrExp(new VarExp("x"), new VarExp("y")),
        new VarExp("z")
    );
CounterVisitor v = new CounterVisitor();

myExp.accept(v);
int count = v.getCount();

```

Spring 15 CSCI 2600, A Milanova

22

## Exercise: Write Evaluate Visitor which evaluates a BooleanExp object

```

class VarExp extends BooleanExp {
    void accept(Visitor v) {
        v.visit(this);
    }
}
class AndExp extends BooleanExp {
    BooleanExp leftExp;
    BooleanExp rightExp;
    void accept(Visitor v) {
        leftExp.accept(v);
        rightExp.accept(v);
        v.visit(this);
    }
}

```

```

class EvaluateVisitor implements Visitor {
    // ??
    void visit(VarExp e) {
        // ??
    }
    void visit(Constant e) {
        // ??
    }
    void visit(AndExp e) {
        // ??
    }
    ...
}

```

23

## Exercise: Write a Visitor that Computes the Cost of a Bicycle Component (Note: Cost of a composite is sum of costs of components + assembly cost)

```

class Skewer extends BicycleComponent {
    void accept(Visitor v) {
        v.visit(this);
    }
}
class Wheel extends BicycleComponent {
    BicycleComponent skewer;
    BicycleComponent hub;
    ...
    void accept(Visitor v) {
        skewer.accept(v);
        hub.accept(v);
        v.visit(this);
    }
}

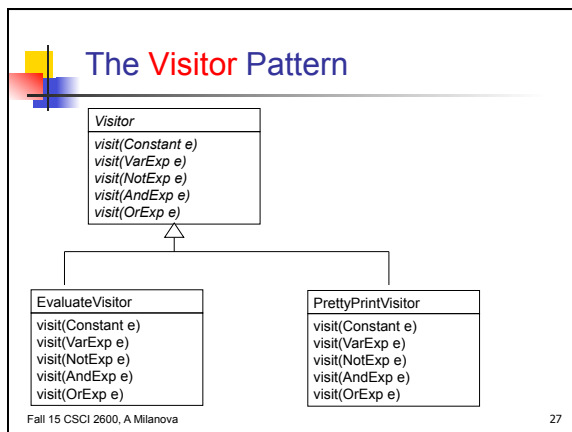
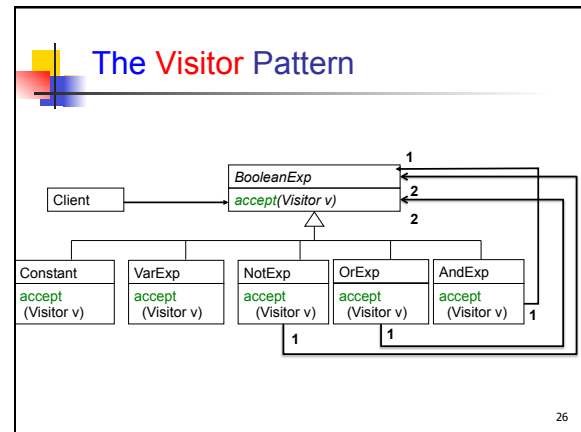
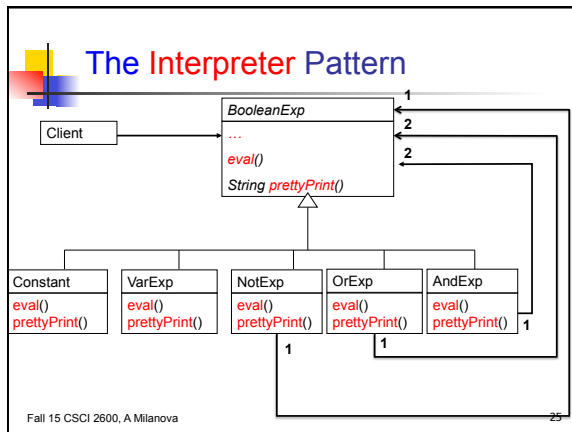
```

```

class CostVisitor implements Visitor {
    ...
    void visit(Skewer e) {
        //??
    }
    void visit(Hub e) {
        //??
    }
    void visit(Wheel e) {
        //??
    }
    ...
}

```

24



## Visitor Pattern

- Must add definitions of **visit** in Visitor hierarchy and **accept** in Object hierarchy
- visit** may do many different things: evaluate, count nodes, pretty print, etc.
- It is easy to add operations (just add a new Visitor class!), but it is hard to add nodes (must modify entire hierarchy of Visitors!)

Spring 15 CSCI 2600, A. Milanova

## Visitor Pattern's Double (Dynamic) Dispatch

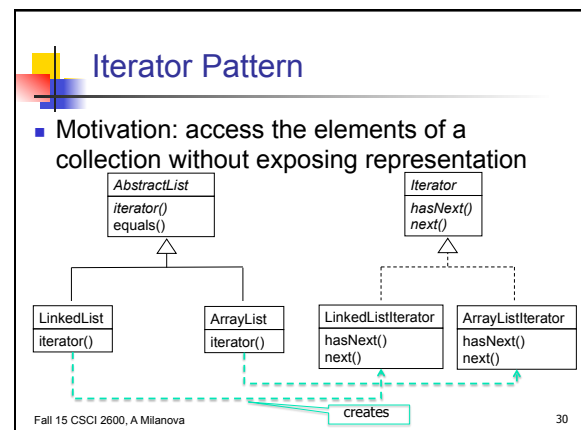
	VarExp	AndExp
evaluate		
pretty-print		

**myExp.accept(v): we want to choose the right operation**  
**myExp.accept(v)** // dynamically dispatch the right  
 // implementation of **accept**, e.g., **AndExp.accept**

```

class AndExp {
    void accept(Visitor v) {
        ...
        v.visit(this); // at compile-time, chooses the
    } // method family visit(AndExp). At
    // runtime, dispatches the right implementation of
    // visit(AndExp), e.g.,
    // EvaluateVisitor.visit(AndExp)
  }
  
```

29



## Iterator Pattern

- Visitor vs. Iterator?
- Visitor pattern is similar to Iterator but different because it has knowledge of structure, not just sequence

Fall 15 CSCI 2600, A. Milanova

31

## Design Patterns Summary so Far

- **Factory method, Factory object, Prototype**
  - Creational patterns: address problem that constructors can't return subtypes
- **Singleton, Interning**
  - Creational patterns: address problem that constructors always return a new instance of class
- **Wrappers: Adapter, Decorator, Proxy**
  - Structural patterns: when we want to change interface or functionality of an existing class, or restrict access to an object

Fall 15 CSCI 2600, A. Milanova

32

## Design Patterns Summary so Far

- **Composite**
  - A structural pattern: expresses whole-part structures, gives uniform interface to client
- **Interpreter, Procedural, Visitor**
  - Behavioral patterns: address the problem of how to traverse composite structures
- **Iterator**
- Next time: Refactoring and more patterns: **Strategy, State, Template Method**

Fall 15 CSCI 2600, A. Milanova

33

## A Design Exercise

- We are building a document editor --- a large rectangle that displays a document. Document can mix text, graphical shapes, etc. Surrounding the document are the menu, scrollbars, borders, etc.
- Structure, Formatting, Embellishing the UI, User commands, Spell checking

Fall 15 CSCI 2600, A. Milanova

34

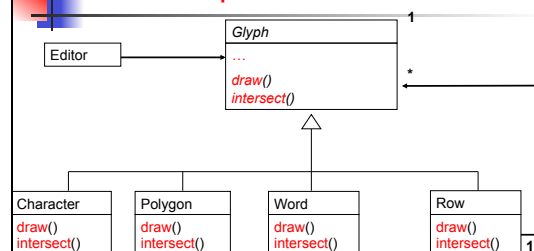
## Structure

- Hierarchical structure --- document is made of columns, a column is made up of rows, a row is made up of words, images, etc.
- Editor should treat text and graphics uniformly. Editor should treat simple and complex elements uniformly
- What design pattern?

Fall 15 CSCI 2600, A. Milanova

35

## The Composite Pattern



What's missing from this picture?

Fall 15 CSCI 2600, A. Milanova

36

## Formatting

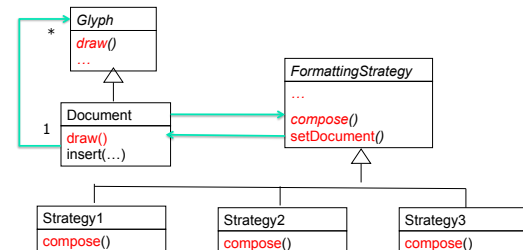
- Formatting displays the document
- Many different formatting strategies are possible
  - We would use different formatting strategies over the same hierarchical structure
- Each formatting strategy is complex

Fall 15 CSCI 2600, A. Milanova

37

## The Strategy Pattern

- Encapsulates an algorithm in an object



Fall 15 CSCI 2600, A. Milanova

38

## Embellishing the UI

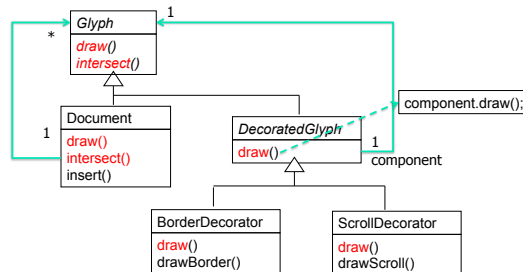
- We would like to embellish the document display. One embellishment adds a border around the document area, another one adds a horizontal scroll bar and a third one adds a vertical scroll bar
- We would like to do this dynamically --- one can create any combination of embellished documents
- What pattern?

Fall 15 CSCI 2600, A. Milanova

39

## The Decorator Pattern

- Adds functionality, preserves interface



Fall 15 CSCI 2600, A. Milanova

40

## User Commands

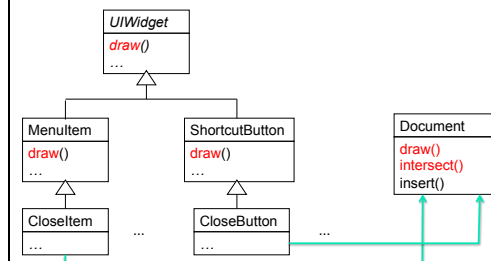
- Editor supports many user "commands": open and close, cut and paste, etc.
- There is different user interface for the same command
  - E.g., close document through a pull-down menu item, close button, key shortcut, other
- Supports undo and redo!

Fall 15 CSCI 2600, A. Milanova

41

## A Naïve Design

- What's wrong with this design?

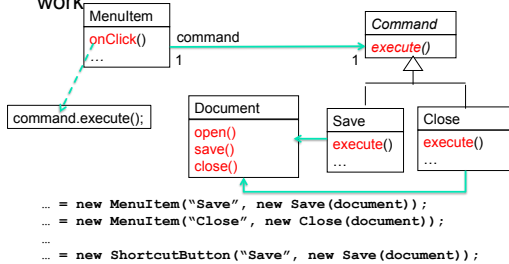


Fall 15 CSCI 2600, A. Milanova

42

## The Command Pattern

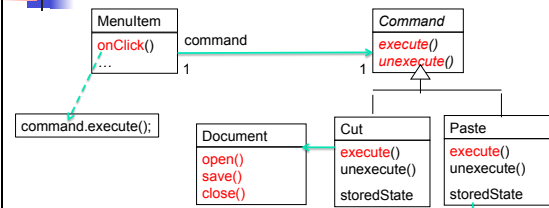
- Separates MenuItems from Commands that do the work



Fall 15 CSCI 2600, A. Milanova

43

## Easy to Add Undo/Redo!



- Editor maintains a history (e.g., a stack) of commands that have been executed

Fall 15 CSCI 2600, A. Milanova

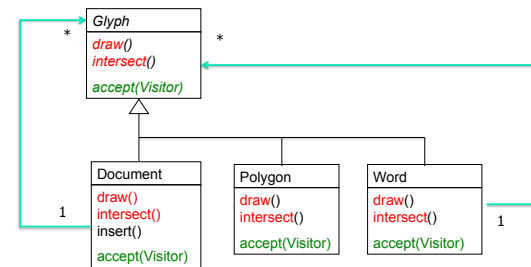
44

## Adding Spell Checking

- Requires traversal of document hierarchy
- We want to avoid writing this functionality into the document structure
- We would like to add other traversals in the future, e.g., search, word count, hyphenation
- What pattern?

45

## The Visitor Pattern



Fall 15 CSCI 2600, A. Milanova

46