CSE 331: Design Patterns (3)

Observer

Problem: Multiple unrelated things need to happen when some state changes.

Solution: Class that manages the state allows other classes to register "observers".

State manager class cannot call the observers directly because the set of observers may change over time.

```
interface LibraryObserver {
  public void libraryChanged();
}
class Library {
  private Set<Item> items;
  private List<LibraryObserver> observers;
  public void registerObserver(LibraryObserver o) {
    this.observers.add(o);
  }
  public void addItem(Item item) {
    if (!items.contains(item)) {
      items.add(item);
      for (LibraryObserver o : observers) {
        o.libraryChanged();
      } } } }
```

```
Library l = new Library();
l.registerObserver(() ->
  System.out.println("got update"));
l.addItem(new Book("Effective Java"));
// prints "got update"
```

Visitor

Problem: Many operations want to traverse a complex-but-relatively-unchanging data structure.

Solution: Define each *operation* as a class that "visits" the data structure.

```
interface Expression {
  public void accept(ExpressionVisitor v);
}
interface ExpressionVisitor {
  public void visitConst(ConstantExpr e);
  public void visitVariable(VariableExpr e);
  public void visitAdd(AddExpr e);
  public void visitMult(MultExpr e);
}
class ConstantExpr implements Expression {
  public final int constant;
  ...
  public void accept(ExpressionVisitor v) {
    v.visitConst(this);
  }
}
class VariableExpr implements Expression {
  public final String variableName;
  ...
  public void accept(ExpressionVisitor v) {
    v.visitVariable(this);
  }
}
```

```
class AddExpr implements Expression {
  private Expression lhs;
  private Expression rhs;
  ...
  public void accept(ExpressionVisitor v) {
    lhs.accept(v);
    rhs.accept(v);
    v.visitAdd(this);
  }
}
class MultExpr implements Expression {
  private Expression lhs;
  private Expression rhs;
  ...
  public void accept(ExpressionVisitor v) {
    lhs.accept(v);
    rhs.accept(v);
    v.visitMult(this);
  }
}
```

```
class ExpressionPrinter
    implements ExpressionVisitor {
  public void visitConst(ConstantExpr e) {
    System.out.println("at constant " + e.constant);
  }
  public void visitVariable(VariableExpr e) {
    System.out.println("at variable " +
        e.variableName);      }
  public void visitAdd(AddExpr e) {
    System.out.println("add");
  }
  public void visitMult(MultExpr e) {
    System.out.println("mult");
  }
}
```

```
Expression e = new AddExpr(new MultExpr(new ConstantExpr(2), new VariableExpr("x")),
                           new ConstantExpr(1));
e.accept(new ExpressionPrinter());
```