Visitor Pattern

Roland Backhouse

March 4, 2003

The Visitor Pattern

- An instance of a "design pattern" (Gamma, Helm, Johnson and Vlissides).
- Used in eg. type-checking, code optimization, flow analysis, pretty-printing, program restructuring, code instrumentation.
- Function: to traverse or "visit" the nodes of the abstract syntax tree performing operations appropriate to the particular application.
- Used in Triangle compiler for contextual analysis and code generation.
- Avoids burdening the classes in the `AbstractSyntaxTrees` package with the code for type checking, pretty printing etc. Instead, separate packages are implemented for each application.

AST

```
package Triangle.AbstractSyntaxTrees;
...
public abstract class AST {

   ...

  public abstract Object visit(Visitor v, Object o);

}
```

Note: The parameter of type `Object` is *not* used in the contextual analysis. The result object is used to return eg the types of expressions and identifier bindings.

IfCommand

```
package Triangle.AbstractSyntaxTrees;
  ...
public class IfCommand extends Command {

  public Object visit(Visitor v, Object o) {
    return v.visitIfCommand(this, o);
  }

  ...
}
```

Contextual Analysis

`Checker` implements the `Visitor` class.

```
/*
 * @(#)Checker.java                              2.0 1999/08/11
 *
 * Copyright (C) 1999 D.A. Watt and D.F. Brown
*/

import Triangle.AbstractSyntaxTrees.*;
...
public final class Checker implements Visitor {

...
}
```

Standard Environment

```
private void establishStdEnvironment () {

  // ERRORtypeDecl is the type declaration associated
  // with TypeDenoters that are not valid.
  // The identifier is the empty string.
  // As the empty string is invalid as an identifier in a
  // program this prevents the use of the ERROR type in a
  // program (inadvertently or otherwise.
  StdEnvironment.ERRORtypeDecl = declareStdType("");

  StdEnvironment.booleanTypeDecl = declareStdType("Boolean");
  StdEnvironment.integerTypeDecl = declareStdType("Integer");
  StdEnvironment.falseDecl =
     declareStdConst("false",StdEnvironment.booleanTypeDecl);
```

LetCommand (Checker)

```
public Object visitLetCommand(LetCommand ast, Object o) {
  idTable.openScope();
  ast.D.visit(this, null);
  ast.C.visit(this, null);
  idTable.closeScope();
  return null;
}
```

VarDeclaration (Checker)

```
public Object visitVarDeclaration(VarDeclaration ast, Object o) {
  TypeDeclaration binding = (TypeDeclaration)ast.T.visit(this,o);
  ast.T.D = binding;
  if (idTable.declaredAtThisLevel(ast.I.spelling))
     reporter.reportError ("Identifier \"%\" already declared",
                           ast.I.spelling, ast.I.position);

     idTable.enter(ast.I.spelling, ast);

  return null;
}
```

visitIfCommand (Checker)

```
public Object visitIfCommand(IfCommand ast, Object o) {
    TypeDenoter eType = (TypeDenoter) ast.E.visit(this, null);
    if (! eType.equals(StdEnvironment.booleanType))
      reporter.reportError("Boolean expression expected here",
                           "", ast.E.position);
    ast.C1.visit(this, null);
    ast.C2.visit(this, null);
    return null;
  }
```

visitIdentifier (Checker)

```
public Object visitIdentifierExpression(IdentifierExpression ast,
                                        Object o) {
   Declaration binding = (Declaration) ast.I.visit(this, null);
   TypeDeclaration idType = StdEnvironment.ERRORtypeDecl;
   if (binding == null)
       reportUndeclared(ast.I);
   if (binding instanceof VarDeclaration)
       idType = ((VarDeclaration)binding).T.D;
   else if (binding instanceof ConstDeclaration)
       idType = ((ConstDeclaration)binding).T;
   else reporter.reportError
           ("Identifier \"%\" is not a variable or constant",
            ast.I.spelling, ast.position);
   return idType;
}
```

check (Checker)

```
public void check(Program ast) {
  ast.visit(this, null);
}
```

Compiler

```
static boolean compileProgram (String sourceName,
                               String objectName)  {
   ...
   theAST = parser.parseProgram(); // 1st pass
   if (reporter.numErrors == 0) {
      System.out.println ("Contextual Analysis ...");
      checker.check(theAST);         // 2nd pass
   }
   ...
```