

abstract class Exp {

public abstract void accept (ExpVisitor v);

}

class IntExp extends Exp {

int n;

public void accept (ExpVisitor v)

{v.visitIntExp(this);}

}

class ArithExp extends Exp {

String op;

Exp left, right;

public void accept (ExpVisitor v)

{v.visitArithExp(this);}

}

abstract class ExpVisitor {

public abstract void visitIntExp (IntExp e);

public abstract void visitArithExp (ArithExp e);

}//(CONCRETE VISITOR)

class printInfixExpVisitor extends ExpVisitor {

public void visitIntExp (IntExp e) {

System.out.print(e.n);

}

public void visitArithExp (ArithExp e) {

System.out.print(“(“);

e.left.accept(this);

System.out.print(e.op);

e.right.accept(this);

System.out.print(“)“);

}

}

//(CONCRETE VISITOR)

class printprefixExpVisitor extends ExpVisitor {

public void visitIntExp (IntExp e) {

System.out.print(e.n);

}

public void visitArithExp (ArithExp e) {

System.out.print(“(“);

System.out.print(e.op);

e.left.accept(this);

e.right.accept(this);

System.out.print(“)“);

}

}

**(ITERATOR)**

import java.util.\*;

abstract class Shape {

abstract void Draw ();

}

class Circle extends Shape {

void Draw () { System.out.println("Circle");}

}

class Square extends Shape {

void Draw () { System.out.println("Square");}

}

class Line extends Shape {

void Draw () { System.out.println("Line");}

}

class Panel extends Shape {

private List<Shape> children = new ArrayList<Shape>();

void Draw () {

System.out.println("Panel");

for (Iterator<Shape> i = children.iterator(); i.hasNext(); ) {

Shape s = i.next();

s.Draw();

}

}

public void Add (Shape s) {

children.add (s);

}

public void Remove (Shape s) {

children.remove (s);

}

}

public class Test {

public static void main (String[] args) {

Panel root = new Panel ();

Circle circle = new Circle ();

Square square = new Square ();

Panel child = new Panel();

Line line = new Line();

root.Add (circle);

root.Add (square);

root.Add (child);

child.Add (line);

root.Draw();

}

}

**(OBSERVER)**

class Data {

private List<Observer> observers = new ArrayList<Observer>();

public void attach(Observer o) {

observers.add (o);

}

public void deattach(Observer o) {

observers.remove (o);

}

int a=50, b=30, c=20;

void set (int a, int b, int c) {

this.a=a; this.b=b; this.c=c;

//notify all observers

Iterator<Observer> i = observers.iterator();

while (i.hasNext()) {

Observer o = i.next();

o.update(this);

}

}

}

abstract class Observer {

abstract void update (Data d);

}

class PieChart extends Observer {

public void update (Data d) {

System.out.println("PieChart");

System.out.println("a="+d.a);

}

}

class ColumnChart extends Observer {

public void update (Data d) {

System.out.println("ColumnChart");

System.out.println("b="+d.b);

}

}

public class Test {

public static void main (String[] args) {

Data d = new Data();

PieChart pc = new PieChart();

ColumnChart cc = new ColumnChart();

d.attach(pc); d.attach(cc);

d.set(10,30,60);

}

}

**(SOME CLIENT)**

Data d = new Data();

PieChart pc = new PieChart();

ColumnChart cc = new ColumnChart();

d.attach(pc); d.attach(cc);

d.set(10,30,60);

* **Iterator**:
  + Aggregate and access elements sequentially
* **Observer**:
  + Observers update automatically when observed object changes
* **Visitor**:
  + Operations applied to elements of a heterogeneous object structure