

```

public abstract class Expression4 {
    public abstract void accept(ExpressionVisitor visitor);
}
C1  public class Constant4 extends Expression4 {
C2
C3      int value;
C4
C5      public Constant4(int v) {
C6          value = v;
C7      }
C8
C9      public void accept(ExpressionVisitor visitor) {
C10         visitor.visitConstant(this);
}
A1  public class Addition4 extends Expression4 {
A3      Expression4 left, right;
A5      public Addition4(Expression4 l, Expression4 r) {
A6          left = l;
A7          right = r;
A8      }
A10     public void accept(ExpressionVisitor visitor) {
A11         visitor.visitAddition(this);
}
}
// Classes for Multiplication and Negation are similar...
public interface ExpressionVisitor {
    public void visitConstant(Constant4 c);
    public void visitAddition(Addition4 a);
    public void visitMultiplication(Multiplication4 m);
    public void visitNegation(Negation4 n);
}
public class ExpressionFormatter implements ExpressionVisitor {
    /** The formatted string being built */
    private String string;
    /** Public access to the string */
    public String getString() { return string; }
    /** Initialise with an empty string */
    public ExpressionFormatter() {
        string = "";
    }
    /** Get the string for a constant */
    public void visitConstant(Constant4 c) {
E1      string += c.value;

```

```

E2    }
E4    /** Build the string for a sum */
E5    public void visitAddition(Addition4 a) {
E6        string += "(";
E7        a.left.accept(this);
E8        string += "+";
E9        a.right.accept(this);
E10       string += ")";
}
...
}
static void demonstrateExpr() {
// illustration of using Expression4
    Expression4 x, y, c;
D1    ExpressionFormatter f = new ExpressionFormatter();
D3    x = new Constant4(1);
D4    y = new Constant4(2);
D5    c = new Addition4(x, y);
D6    d = new Multiplication4(c, new Constant4(42));
D8    c.accept(f);
D9    System.out.println(f.getString());
}

```