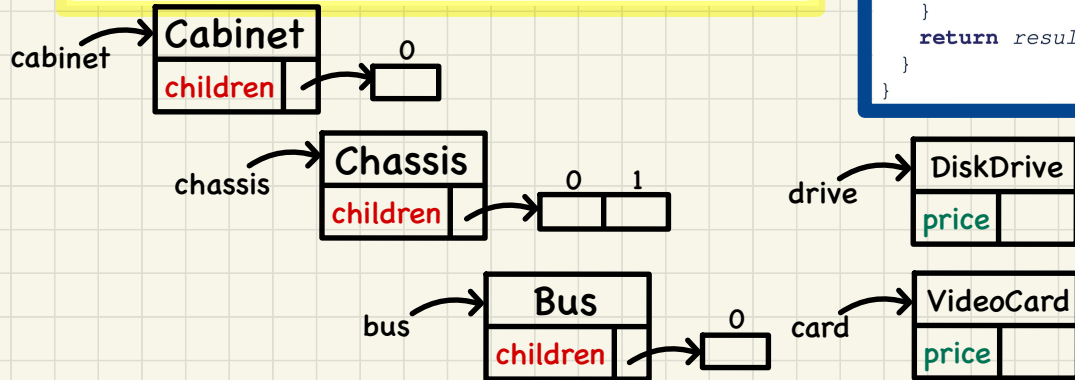


Composite Pattern: Testing

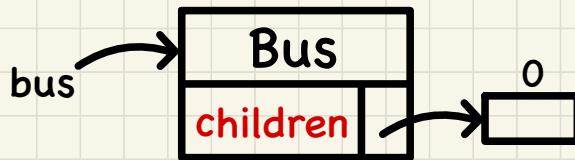
@Test

```
public void test_equipment() {  
    Equipment card, drive;  
    Bus bus;  
    Cabinet cabinet;  
    Chassis chassis;  
  
    card = new VideoCard("16Mbs Token Ring", 200);  
    drive = new DiskDrive("500 GB harddrive", 500);  
    bus = new Bus("MCA Bus");  
    chassis = new Chassis("PC Chassis");  
    cabinet = new Cabinet("PC Cabinet");  
    bus.add(card);  
    chassis.add(bus);  
    chassis.add(drive);  
    cabinet.add(chassis);  
  
    assertEquals(700.00, cabinet.price(), 0.1);  
}
```

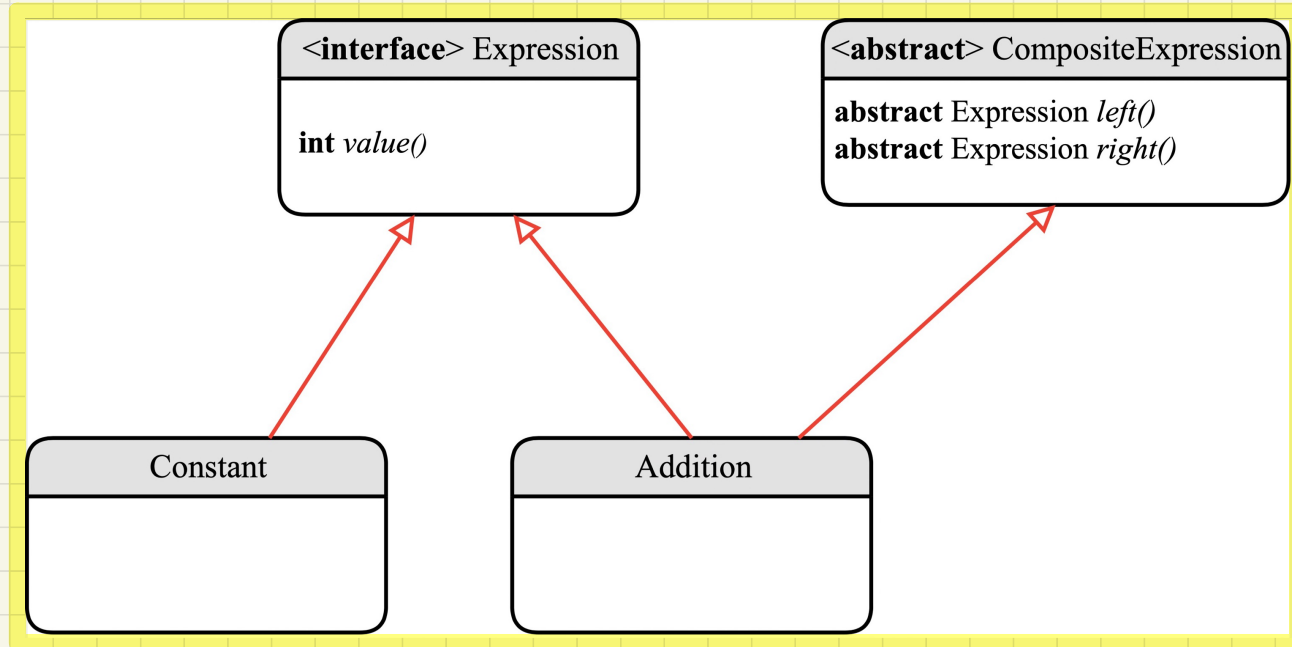


```
public abstract class BaseEquipment implements Equipment {  
    private String name;  
    private double price;  
    public BaseEquipment(String name, double price) {  
        this.name = name; this.price = price;  
    }  
    public String name() { return this.name; }  
    public double price() { return this.price; }  
}
```

```
public abstract class CompositeEquipment  
    extends Composite<Equipment>  
    implements Equipment  
{  
    private String name;  
    public CompositeEquipment(String name) {  
        this.name = name;  
        this.children = new ArrayList<>();  
    }  
    public String name() { return this.name; }  
    public double price() {  
        double result = 0.0;  
        for(Equipment child : this.children) {  
            result = result + child.price(); /* dynamic binding */  
        }  
        return result;  
    }  
}
```



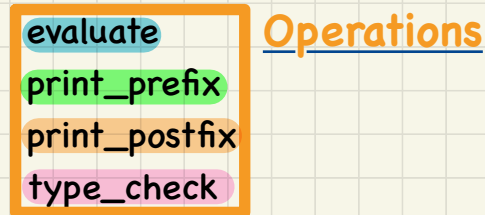
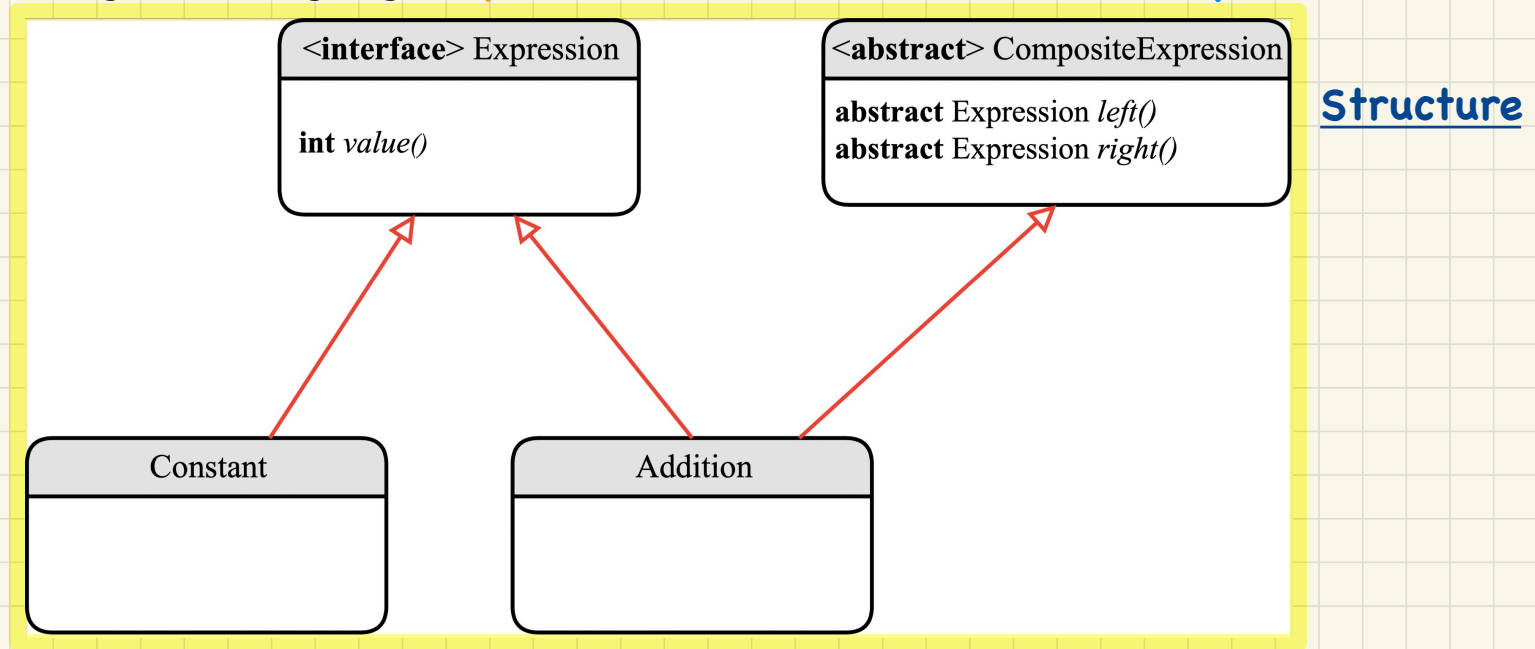
Design of Language Structure: Composite Pattern



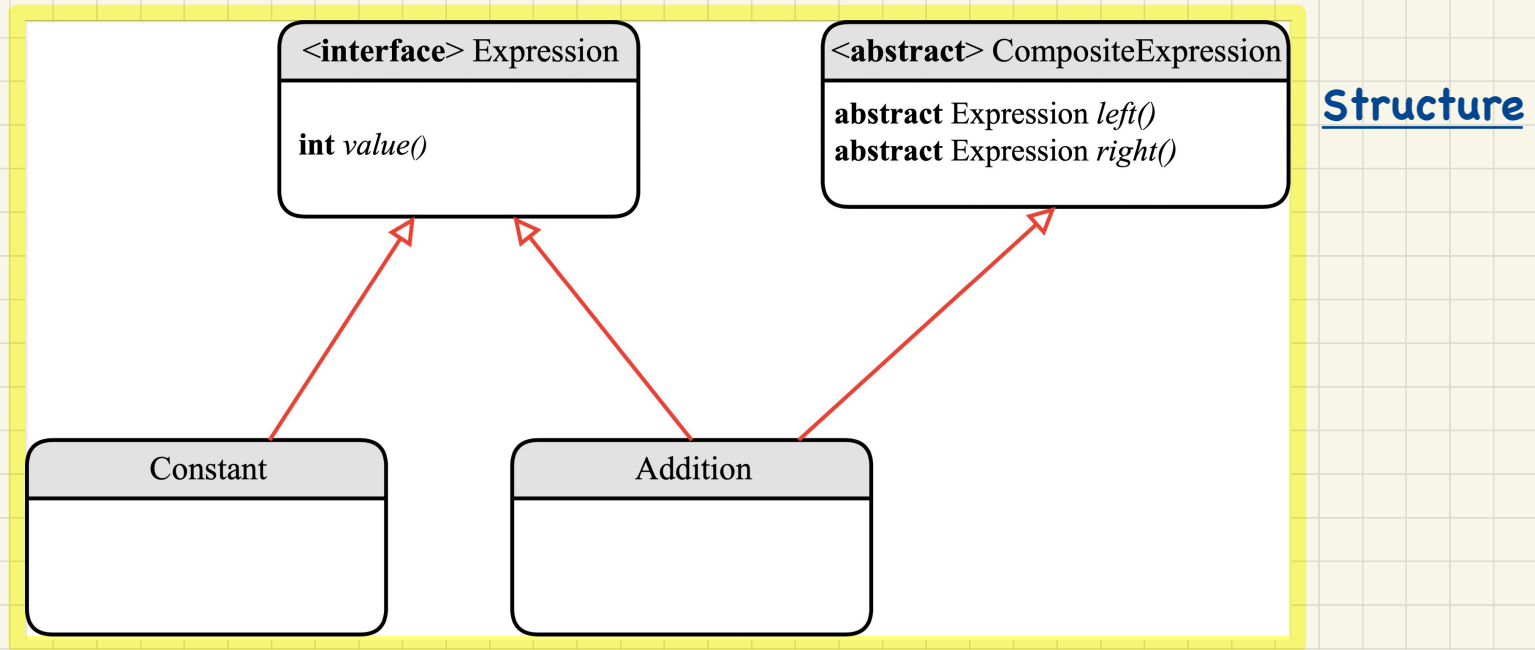
Q: How to construct a **composite object** representing "341 + 2"?

Q: How to extend the design to include **variables** and **subtractions**?

Design of Language **Operation**: How to Extend the **Composite** Pattern?



Design of a Language Application: Open-Closed Principle

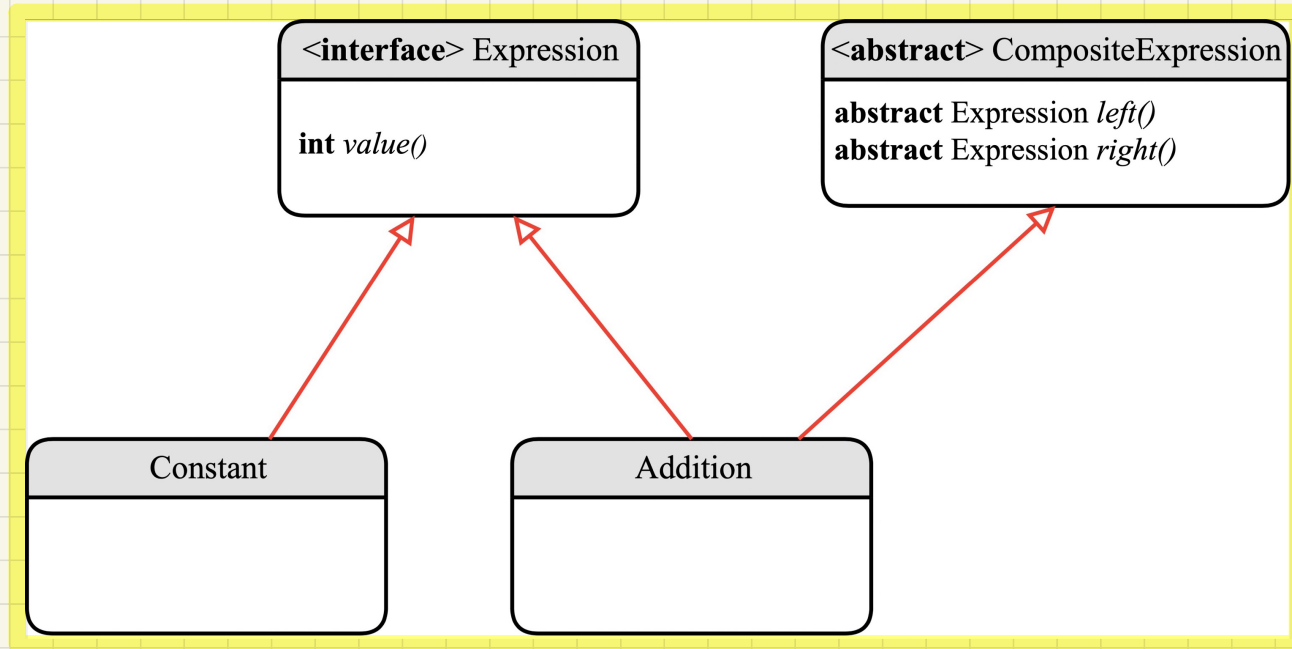


evaluate
print_prefix
print_postfix
type_check

Operations

	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

Design of a Language Application: Open-Closed Principle



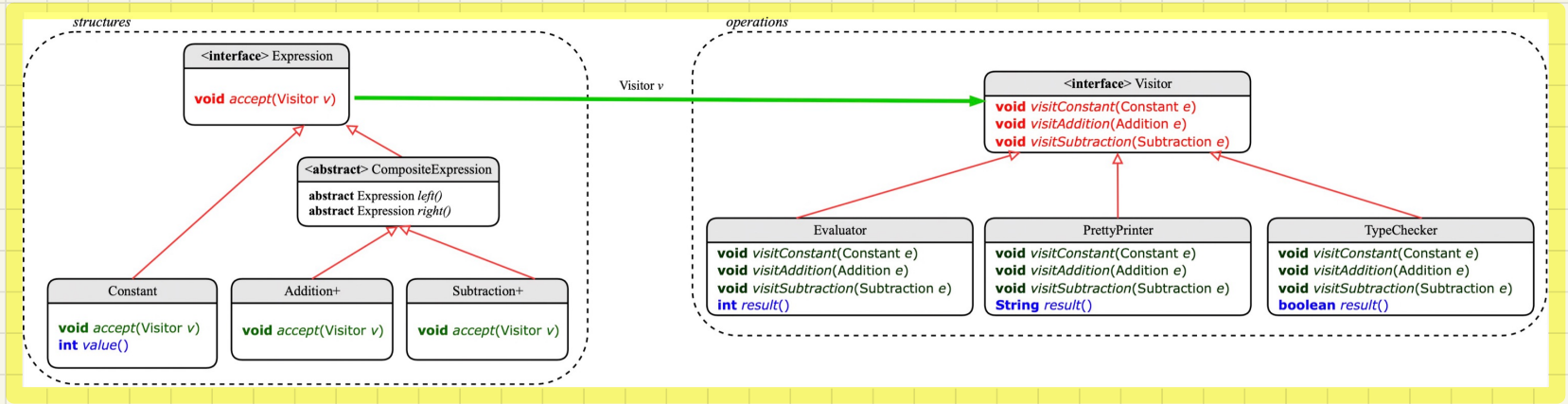
Structure

evaluate
print_prefix
print_postfix
type_check

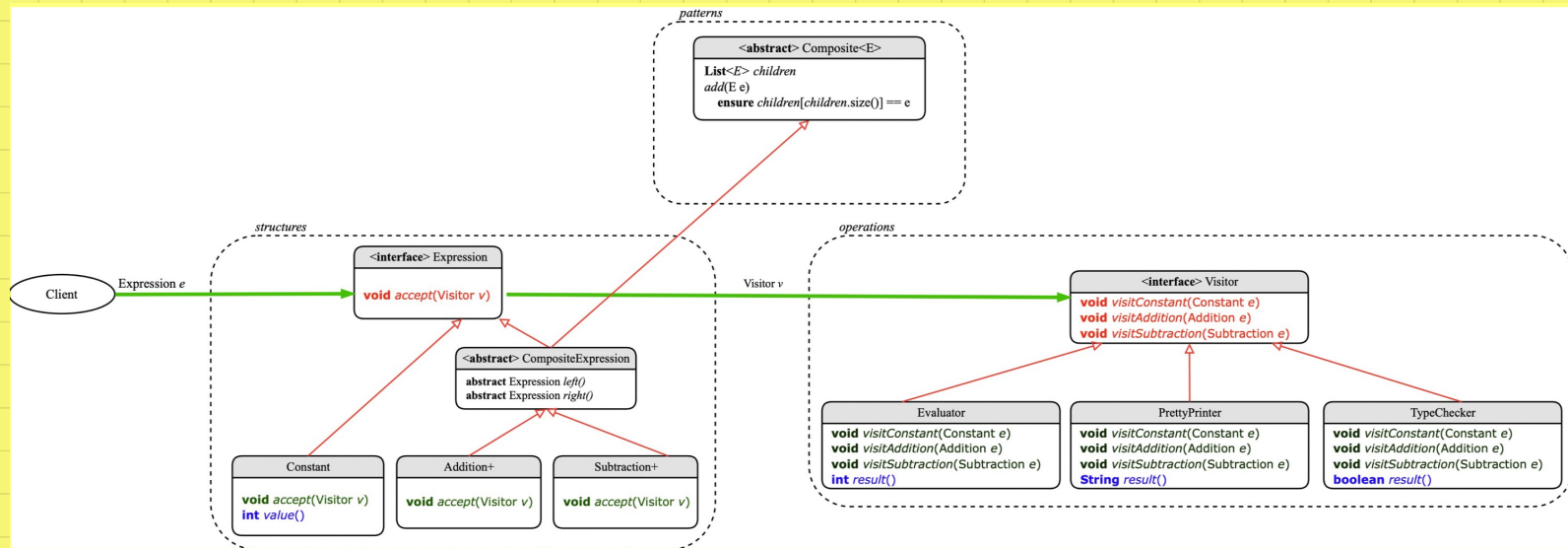
Operations

	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

Visitor Design Pattern: Architecture



Visitor Design Pattern: Architecture



```
1 @Test
2 public void test_expression_evaluation() {
3     CompositeExpression add;
4     Expression c1, c2;
5     Visitor v;
6     c1 = new Constant(1); c2 = new Constant(2);
7     add = new Addition(c1, c2);
8     v = new Evaluator();
9     add.accept(v);
10    assertEquals(3, ((Evaluator) v).result());
11 }
```

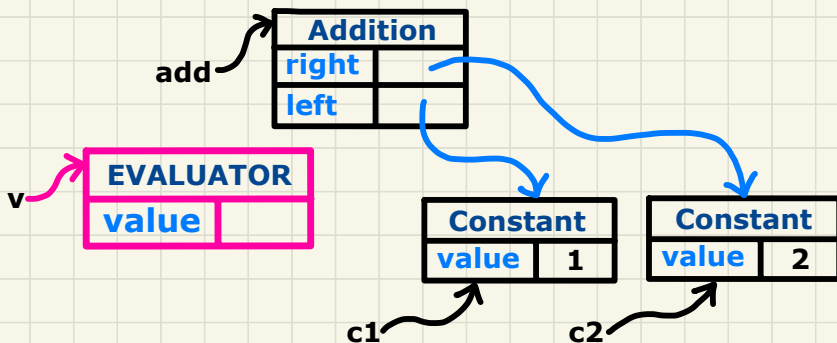
How to Use Visitors

Visitor Design Pattern: Implementation

```
1  @Test
2  public void test_expression_evaluation() {
3      CompositeExpression add;
4      Expression c1, c2;
5      Visitor v;
6      c1 = new Constant(1); c2 = new Constant(2);
7      add = new Addition(c1, c2);
8      v = new Evaluator();
9      add.accept(v);
10     assertEquals(3, ((Evaluator) v).result());
11 }
```

Visualizing Line 3 to Line 7

Executing Composite and Visitor Patterns at Runtime



Tracing `add.accept(v)`
Double Dispatch

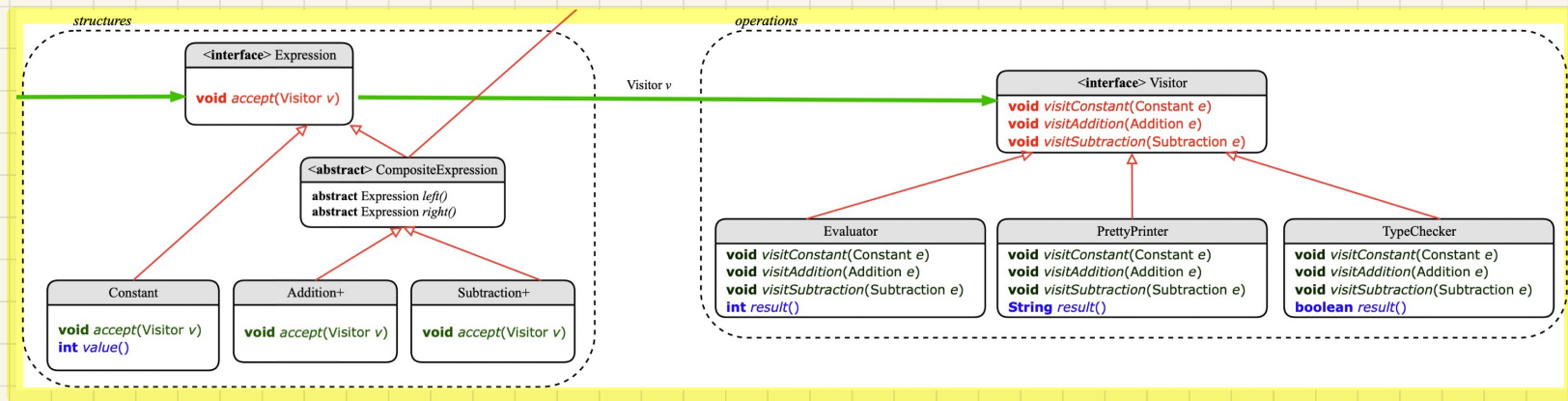
```
public class Constant implements Expression {  
    ...  
    public void accept(Visitor v) {  
        v.visitConstant(this);  
    }  
}
```

```
public class Addition extends CompositeExpression {  
    ...  
    public void accept(Visitor v) {  
        v.visitAddition(this);  
    }  
}
```

```
public interface Visitor {  
    public void visitConstant(Constant e);  
    public void visitAddition(Addition e);  
    public void visitSubtraction(Subtraction e);  
}
```

```
public class Evaluator implements Visitor {  
    private int result;  
    ...  
    public void visitConstant(Constant e) {  
        this.result = e.value();  
    }  
    public void visitAddition(Addition e) {  
        Evaluator evalL = new Evaluator();  
        Evaluator evalR = new Evaluator();  
        e.getLeft().accept(evalL);  
        e.getRight().accept(evalR);  
        this.result = evalL.result() + evalR.result();  
    }  
}
```

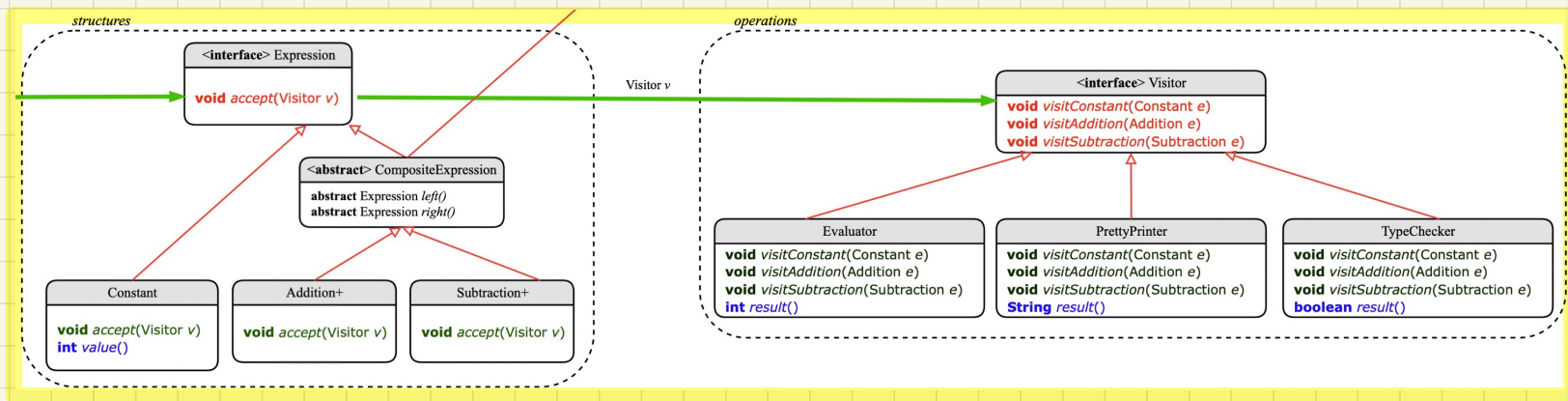
Visitor Pattern: Open-Closed and Single-Choice Principles



What if a **new language construct** is added?

If the **visitor pattern** is adopted, what should be **closed**?

Visitor Pattern: Open-Closed and Single-Choice Principles



What if a **new language operation** is added?

If the **visitor pattern** is adopted, what should be **open**?