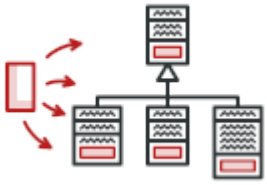

Visitor in Java

Visitor is a behavioral design pattern that allows adding new behaviors to existing class hierarchy without altering any existing code.

Read why Visitors can't be simply replaced with method overloading in our article **Visitor and Double Dispatch**.

Learn more about Visitor →

Navigation

Complexity: ★★★

Popularity: ★☆☆

Here are some examples of pattern in core Java libraries:

- `javax.lang.model.element.AnnotationValue` and `AnnotationValueVisitor`
- `javax.lang.model.element.Element` and `ElementVisitor`
- `javax.lang.model.type.TypeMirror` and `TypeVisitor`
- `java.nio.file.FileVisitor` and `SimpleFileVisitor`
- `javax.faces.component.visit.VisitContext` and `VisitCallback`

Exporting shapes into XML

In this example, we would want to export a set of geometric shapes into XML. The catch is that we don't want to change the code of shapes directly or at least keep it to the minimum.

In the end, the Visitor pattern establishes an infrastructure that allows us to add any behaviors to the shapes hierarchy without changing the existing code of those classes.

shapes

shapes/Shape.java: Common shape interface

```
package refactoring_guru.visitor.example.shapes;

import refactoring_guru.visitor.example.visitor.Visitor;

public interface Shape {
    void move(int x, int y);
    void draw();
    String accept(Visitor visitor);
}
```

shapes/Dot.java: A dot

```
package refactoring_guru.visitor.example.shapes;

import refactoring_guru.visitor.example.visitor.Visitor;

public class Dot implements Shape {
    private int id;
    private int x;
    private int y;
```

```
    public Dot(int id, int x, int y) {
        this.id = id;
        this.x = x;
        this.y = y;
    }

    @Override
    public void move(int x, int y) {
        // move shape
    }

    @Override
    public void draw() {
        // draw shape
    }

    @Override
    public String accept(Visitor visitor) {
        return visitor.visitDot(this);
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public int getId() {
        return id;
    }
}
```

shapes/Circle.java: A circle

```
package refactoring_guru.visitor.example.shapes;

import refactoring_guru.visitor.example.visitor.Visitor;

public class Circle extends Dot {
    private int radius;

    public Circle(int id, int x, int y, int radius) {
        super(id, x, y);
        this.radius = radius;
    }

    @Override
```

```
    }

    public int getRadius() {
        return radius;
    }
}
```

shapes/Rectangle.java: A rectangle

```
package refactoring_guru.visitor.example.shapes;

import refactoring_guru.visitor.example.visitor.Visitor;

public class Rectangle implements Shape {
    private int id;
    private int x;
    private int y;
    private int width;
    private int height;

    public Rectangle(int id, int x, int y, int width, int height) {
        this.id = id;
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }

    @Override
    public String accept(Visitor visitor) {
        return visitor.visitRectangle(this);
    }

    @Override
    public void move(int x, int y) {
        // move shape
    }

    @Override
    public void draw() {
        // draw shape
    }

    public int getId() {
        return id;
    }

    public int getX() {
        return x;
    }
```

```
        return y;
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }
}
```

shapes/CompoundShape.java: A compound shape

```
package refactoring_guru.visitor.example.shapes;

import refactoring_guru.visitor.example.visitor.Visitor;

import java.util.ArrayList;
import java.util.List;

public class CompoundShape implements Shape {
    public int id;
    public List<Shape> children = new ArrayList<>();

    public CompoundShape(int id) {
        this.id = id;
    }

    @Override
    public void move(int x, int y) {
        // move shape
    }

    @Override
    public void draw() {
        // draw shape
    }

    public int getId() {
        return id;
    }

    @Override
    public String accept(Visitor visitor) {
        return visitor.visitCompoundGraphic(this);
    }

    public void add(Shape shape) {
        children.add(shape);
```

visitor

visitor/Visitor.java: Common visitor interface

```
package refactoring_guru.visitor.example.visitor;

import refactoring_guru.visitor.example.shapes.Circle;
import refactoring_guru.visitor.example.shapes.CompoundShape;
import refactoring_guru.visitor.example.shapes.Dot;
import refactoring_guru.visitor.example.shapes.Rectangle;

public interface Visitor {
    String visitDot(Dot dot);

    String visitCircle(Circle circle);

    String visitRectangle(Rectangle rectangle);

    String visitCompoundGraphic(CompoundShape cg);
}
```

visitor/XMLExportVisitor.java: Concrete visitor, exports all shapes into XML

```
package refactoring_guru.visitor.example.visitor;

import refactoring_guru.visitor.example.shapes.*;

public class XMLExportVisitor implements Visitor {

    public String export(Shape... args) {
        StringBuilder sb = new StringBuilder();
        sb.append("<?xml version=\"1.0\" encoding=\"utf-8\"?>" + "\n");
        for (Shape shape : args) {
            sb.append(shape.accept(this)).append("\n");
        }
        return sb.toString();
    }

    public String visitDot(Dot d) {
        return "<dot>" + "\n" +
                "    <id>" + d.getId() + "</id>" + "\n" +
                "    <x>" + d.getX() + "</x>" + "\n" +
                "    <y>" + d.getY() + "</y>" + "\n" +
                "</dot>";
    }
```

```
                "    <id>" + c.getId() + "</id>" + "\n" +
                "    <x>" + c.getX() + "</x>" + "\n" +
                "    <y>" + c.getY() + "</y>" + "\n" +
                "    <radius>" + c.getRadius() + "</radius>" + "\n" +
                "</circle>";
    }

    public String visitRectangle(Rectangle r) {
        return "<rectangle>" + "\n" +
                "    <id>" + r.getId() + "</id>" + "\n" +
                "    <x>" + r.getX() + "</x>" + "\n" +
                "    <y>" + r.getY() + "</y>" + "\n" +
                "    <width>" + r.getWidth() + "</width>" + "\n" +
                "    <height>" + r.getHeight() + "</height>" + "\n" +
                "</rectangle>";
    }

    public String visitCompoundGraphic(CompoundShape cg) {
        return "<compound_graphic>" + "\n" +
                "   <id>" + cg.getId() + "</id>" + "\n" +
                _visitCompoundGraphic(cg) +
                "</compound_graphic>";
    }

    private String _visitCompoundGraphic(CompoundShape cg) {
        StringBuilder sb = new StringBuilder();
        for (Shape shape : cg.children) {
            String obj = shape.accept(this);
            // Proper indentation for sub-objects.
            obj = "    " + obj.replace("\n", "\n    ") + "\n";
            sb.append(obj);
        }
        return sb.toString();
    }

}
```

Demo.java: Client code

```
package refactoring_guru.visitor.example;

import refactoring_guru.visitor.example.shapes.*;
import refactoring_guru.visitor.example.visitor.XMLExportVisitor;

public class Demo {
    public static void main(String[] args) {
        Dot dot = new Dot(1, 10, 55);
        Circle circle = new Circle(2, 23, 15, 10);
        Rectangle rectangle = new Rectangle(3, 10, 17, 20, 30);
```

```
        compoundShape.add(circle);
        compoundShape.add(rectangle);

        CompoundShape c = new CompoundShape(5);
        c.add(dot);
        compoundShape.add(c);

        export(circle, compoundShape);
    }

    private static void export(Shape... shapes) {
        XMLExportVisitor exportVisitor = new XMLExportVisitor();
        System.out.println(exportVisitor.export(shapes));
    }
}
```

OutputDemo.txt: Execution result

```
<?xml version="1.0" encoding="utf-8"?>
<circle>
    <id>2</id>
    <x>23</x>
    <y>15</y>
    <radius>10</radius>
</circle>

<?xml version="1.0" encoding="utf-8"?>
<compound_graphic>
   <id>4</id>
    <dot>
        <id>1</id>
        <x>10</x>
        <y>55</y>
    </dot>
    <circle>
        <id>2</id>
        <x>23</x>
        <y>15</y>
        <radius>10</radius>
    </circle>
    <rectangle>
        <id>3</id>
        <x>10</x>
        <y>17</y>
        <width>20</width>
        <height>30</height>
    </rectangle>
    <compound_graphic>
       <id>5</id>
        <dot>
```