

[Design Patterns Book \(/design-patte](#)

[/ Design Patterns \(/design_patterns\)](#)
[/ Behavioral patterns \(/design_patterns/behavioral_patterns\)](#)
[/ Visitor \(/design_patterns/visitor\)](#)

Visitor in *Java*

◀ [Back to **Visitor** description \(/design_patterns/visitor\)](#)

Visitor design pattern

1. Add an `accept(Visitor)` method to the "element" hierarchy
2. Create a "visitor" base class w/ a `visit()` method for every "element" type
3. Create a "visitor" derived class for each "operation" to do on "elements"
4. Client creates "visitor" objects and passes each to `accept()` calls

Java

```
interface Element {
    // 1. accept(Visitor) interface
    public void accept( Visitor v ); // first dispatch
}

class This implements Element {
    // 1. accept(Visitor) implementation
    public void accept( Visitor v ) {
        v.visit( this );
    }
    public String thiss() {
        return "This";
    }
}

class That implements Element {
    public void accept( Visitor v ) {
        v.visit( this );
    }
    public String that() {
        return "That";
    }
}

class TheOther implements Element {
    public void accept( Visitor v ) {
        v.visit( this );
    }
    public String theOther() {
        return "TheOther";
    }
}

// 2. Create a "visitor" base class with a visit() method
interface Visitor {
    public void visit( This e ); // second dispatch
    public void visit( That e );
    public void visit( TheOther e );
}

// 3. Create a "visitor" derived class for each "operatio
class UpVisitor implements Visitor {
    public void visit( This e ) {
        System.out.println( "do Up on " + e.thiss() );
    }
    public void visit( That e ) {
        System.out.println( "do Up on " + e.that() );
    }
    public void visit( TheOther e ) {
        System.out.println( "do Up on " + e.theOther() );
    }
}

class DownVisitor implements Visitor {
    public void visit( This e ) {
        System.out.println( "do Down on " + e.thiss() );
    }
    public void visit( That e ) {
        System.out.println( "do Down on " + e.that() );
    }
}
```

```

    }
    public void visit( TheOther e ) {
        System.out.println( "do Down on " + e.theOther() );
    }
}

class VisitorDemo {
    public static Element[] list = { new This(), new That()

    // 4. Client creates "visitor" objects and passes each
    public static void main( String[] args ) {
        UpVisitor    up    = new UpVisitor();
        DownVisitor  down  = new DownVisitor();
        for (int i=0; i < list.length; i++) {
            list[i].accept( up );
        }
        for (int i=0; i < list.length; i++) {
            list[i].accept( down );
        }
    }
}

```

<p>do Up on This do Up on That do Up on TheOther</p>	<p>do Down on This do Down on That do Down on TheOther</p>	Output
--	--	--------

Read next

This article is taken from our book **Design Patterns Explained Simply** (</design-patterns-simply>).

All of the design patterns are compiled there. The book is written in clear, simple language that makes it easy to read and understand (just like this article).

We distribute it in PDF & EPUB formats so you can get it onto your iPad, Kindle, or other portable device immediately after your purchase.



[\(/design-patterns-simply\)](/design-patterns-simply)

♥ [Learn more \(/design-patterns-simply\)](/design-patterns-simply)

Other Visitor examples

Java	Visitor in Java ← ☺	Visitor in Java (/design_patterns/visitor/java/4)	Visitor in Java (/design_patterns/visitor/java/3)	Visitor in Java: Double dispatch (within a single hierarchy) (/design_patterns/visitor/java/2)
C#	Visitor in C# (/design_patterns/visitor/c-sharp-dot-net)			
C++	Visitor in C++: Before and after (/design_patterns/visitor/cpp/1)	Visitor in C++: Recovering lost type information (/design_patterns/visitor/cpp/3)	Visitor in C++ (/design_patterns/visitor/cpp/2)	
PHP	Visitor in PHP (/design_patterns/visitor/php)			
Delphi	Visitor in Delphi (/design_patterns/visitor/delphi)			

[← Template Method](#) [\(/design_patterns/template_method\)](/design_patterns/template_method) |
 [▲ Visitor](#) [\(/design_patterns/visitor\)](/design_patterns/visitor)



This work is licensed under a Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 Unported License (<http://creativecommons.org/licenses/by-nc-nd/3.0/>)

[Design Patterns \(/design_patterns\)](/design_patterns)
[AntiPatterns \(/antipatterns\)](/antipatterns) [Refactoring \(/refactoring\)](/refactoring)

[My account \(/user\)](/user)
[Contacts \(/contact\)](/contact)

[UML \(/uml\)](#)

[About us \(/about-us\)](#)

 720

 1,088

 1.5k