30 Behavioral: Visitor Pattern — Extensible File-System Operations Assignment

Add new behaviours (compression, preview, …) to a stable file hierarchy **without touching the file classes**.  
 Each class below includes concise JavaDoc. Conclude with a brief **Reflection**.

.

├── analysis

│ └── visitor\_need.md ← why visitors beat “instanceof” chains

├── src/main/java

│ └── fs

│ ├── element

│ │ ├── File.java

│ │ ├── TextFile.java

│ │ └── ImageFile.java

│ ├── visitor

│ │ ├── FileVisitor.java

│ │ ├── CompressionVisitor.java

│ │ └── PreviewVisitor.java

│ ├── structure

│ │ └── FileSystem.java

│ └── VisitorDemo.java

├── src/test/java/fs

│ ├── CompressionVisitorTest.java

│ └── PreviewVisitorTest.java

├── reflection.md

└── README.md

#### **1 FileVisitor.java**

package fs.visitor;

import fs.element.ImageFile;

import fs.element.TextFile;

/\*\*

\* Visitor contract for operations applied to file elements.

\* Add one {@code visit(...)} overload per concrete element type.

\*/

public interface FileVisitor {

void visit(TextFile textFile);

void visit(ImageFile imageFile);

}

#### **2 Concrete visitors**

package fs.visitor;

import fs.element.\*;

/\*\* Compresses each file it encounters. \*/

public class CompressionVisitor implements FileVisitor {

@Override public void visit(TextFile t){ System.out.println("Compressing text: " + t.getName()); }

@Override public void visit(ImageFile i){ System.out.println("Compressing image: " + i.getName()); }

}

package fs.visitor;

import fs.element.\*;

/\*\* Generates quick previews for files. \*/

public class PreviewVisitor implements FileVisitor {

@Override public void visit(TextFile t){ System.out.println("Preview text: " + t.getName()); }

@Override public void visit(ImageFile i){ System.out.println("Preview image: " + i.getName()); }

}

#### **3 File element hierarchy**

package fs.element;

import fs.visitor.FileVisitor;

/\*\*

\* Element in the object-structure that can be “visited”.

\*/

public interface File {

void accept(FileVisitor v); // double-dispatch hook

String getName();

}

package fs.element;

import fs.visitor.FileVisitor;

/\*\* Plain-text document. \*/

public class TextFile implements File {

private final String name;

public TextFile(String name){ this.name = name; }

@Override public void accept(FileVisitor v){ v.visit(this); }

@Override public String getName(){ return name; }

}

package fs.element;

import fs.visitor.FileVisitor;

/\*\* Raster image file. \*/

public class ImageFile implements File {

private final String name;

public ImageFile(String name){ this.name = name; }

@Override public void accept(FileVisitor v){ v.visit(this); }

@Override public String getName(){ return name; }

}

#### **4 FileSystem.java (object structure)**

package fs.structure;

import fs.element.File;

import fs.visitor.FileVisitor;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Aggregates {@link File} elements and applies visitors to them.

\*/

public class FileSystem {

private final List<File> files = new ArrayList<>();

public void add(File f){ files.add(f); }

/\*\* Applies a visitor to every file in the structure. \*/

public void traverse(FileVisitor v){

for(File f: files) f.accept(v);

}

}

#### **5 Demo client**

package fs;

import fs.element.\*;

import fs.structure.FileSystem;

import fs.visitor.\*;

public class VisitorDemo {

public static void main(String[] args){

FileSystem fs = new FileSystem();

fs.add(new TextFile("report.txt"));

fs.add(new ImageFile("logo.png"));

System.out.println("== Compress ==");

fs.traverse(new CompressionVisitor());

System.out.println("\n== Preview ==");

fs.traverse(new PreviewVisitor());

}

}

Console

== Compress ==

Compressing text: report.txt

Compressing image: logo.png

== Preview ==

Preview text: report.txt

Preview image: logo.png

#### **6 Test sketches**

/\* CompressionVisitorTest.java \*/

FileSystem fs = new FileSystem();

fs.add(new TextFile("a.txt"));

ByteArrayOutputStream out = new ByteArrayOutputStream();

System.setOut(new PrintStream(out));

fs.traverse(new CompressionVisitor());

assertTrue(out.toString().contains("Compressing text: a.txt"));

## **reflection.md**

**Why Visitor?** File hierarchy is stable; new behaviours (hashing, virus-scan) arrive often. Visitor shifts those behaviours out of the elements, upholding Open/Closed for operations.

| **Advantage** | **Note** |
| --- | --- |
| Add ops, no element change | Drop in a new visitor; element code untouched. |
| Centralised logic | Related code collocated = easier maintenance. |
| Double dispatch | Gets compile-time type safety over instanceof chains. |

*Trade-offs*

* Harder to add a brand-new element class – every visitor needs a new visit() method.
* Extra ceremony: many small classes.

Apply Visitor when **operations change more frequently than data structures**, such as tooling on ASTs, file systems, UI widgets, or data-model audits.