

A ~~V~~ibrant ~~V~~isit

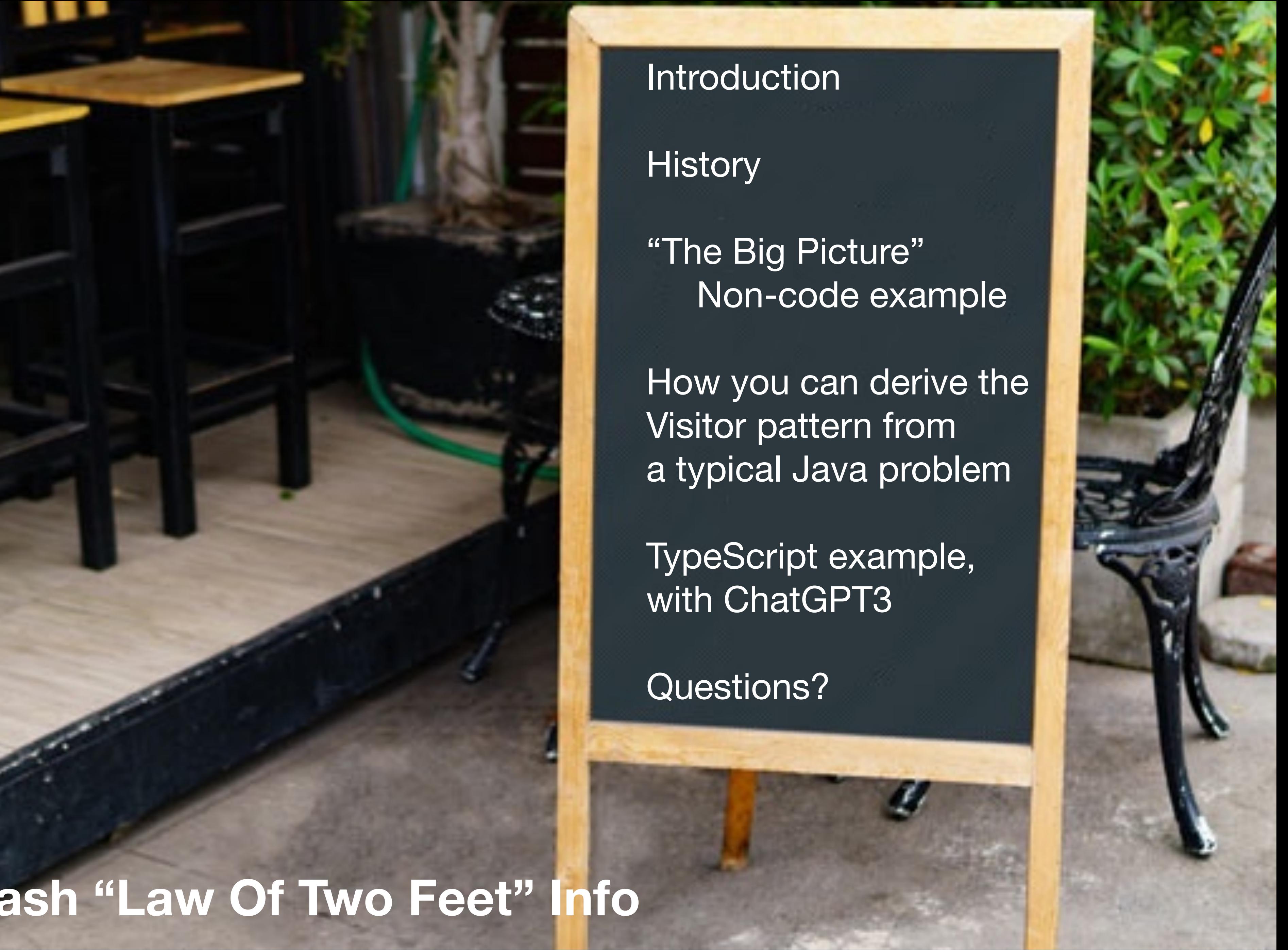
With The ~~V~~aluable

And ~~V~~ersatile

~~V~~isitor

Design Pattern





Introduction

History

“The Big Picture”

Non-code example

How you can derive the  
Visitor pattern from  
a typical Java problem

TypeScript example,  
with ChatGPT3

Questions?

CodeMash “Law Of Two Feet” Info

Kelly Morrison, Ph.D. Computer Engineering  
Solution Architect - Daugherty Business Solutions





A long, long time ago...

(1994)

# Design Patterns

Elements of Reusable  
Object-Oriented Software

Erich Gamma  
Richard Helm  
Ralph Johnson  
John Vlissides



Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch



ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



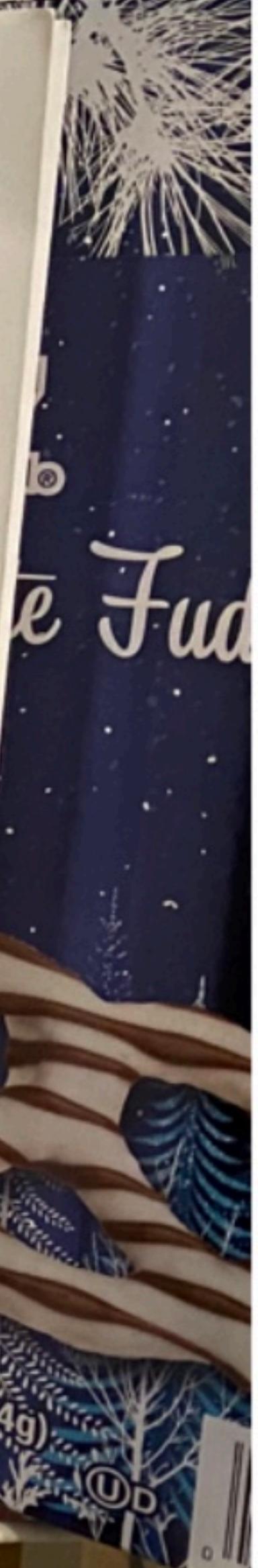
# PATTERN HATCHING

## Design Patterns Applied



JOHN VLASSIDES  
Foreword by James O. Coplien

SOFTWARE PATTERNS SERIES



- the Visitor pattern when an object structure contains many classes and you want to perform operations on them
- many distinct and unrelated operations need to be performed on one object structure, and you want to avoid “polluting” their class in one class. When the object structure is shared by many applications, it's probably easier just to define new operations over the structure. Changing the structure rarely changes the interface to all visitors.
- the classes defining the object structure rarely change, so defining new operations over the structure is costly. If the object structure defines the operations in the classes requires redefining the visitor to put operations in just those applications that need them.

## VISITOR

Object Behavioral

### Intent

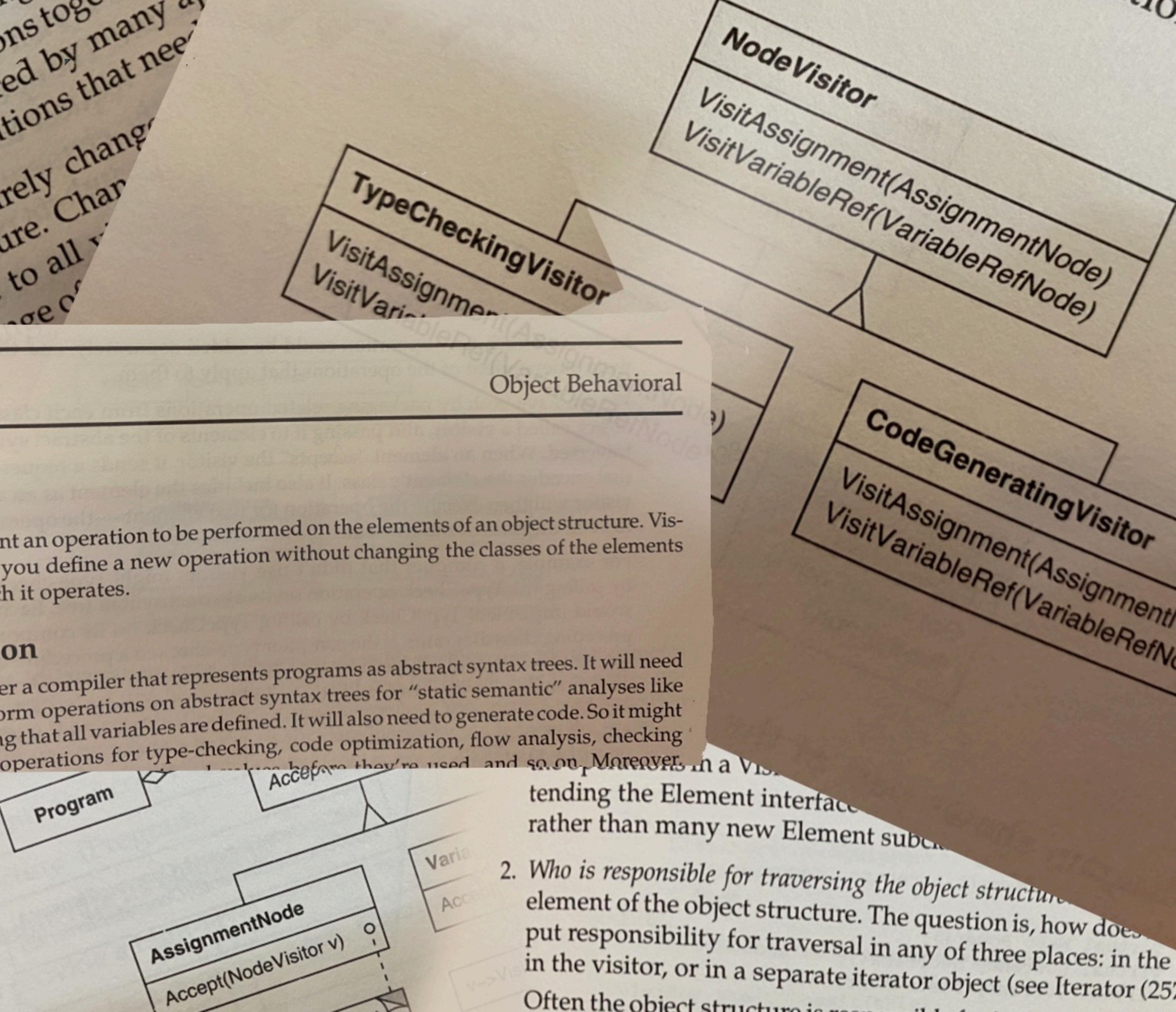
Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

### Motivation

Consider a compiler that represents programs as abstract syntax trees. It will need to perform operations on abstract syntax trees for “static semantic” analyses like checking that all variables are defined. It will also need to generate code. So it might define operations for type-checking, code optimization, flow analysis, checking for errors, accepting them when they're used, and so on. Moreover,

tending the Element interface is much easier than maintaining many new Element subclasses.

2. Who is responsible for traversing the object structure? That is, who is responsible for visiting each element of the object structure. The question is, how does the visitor know what to do with each element? Put another way, who is responsible for putting responsibility for traversal in any of three places: in the visitor, or in a separate iterator object (see Iterator (257)), or in the object structure itself?



# THE VISITOR PATTERN IS REGARDED

A collage of three women's faces in various expressions (smiling, neutral, serious) and a calculator screen displaying a right-angled triangle diagram and trigonometric values.

AS ONE OF THE MOST  
DIFFICULT DESIGN PATTERNS

$$y = ax^2 + bx + c$$
$$(x_1, x_2)$$

	30°	45°	60°
sin	$\frac{1}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$
cos	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{1}{2}$

•••



AA

google.com



Inbox(1369) - EarthLink Mail

DALL·E 2

what is the most complex design...

Google

what is the most complex design pattern



All

Images

Videos

Shopping

News

More

Tools

About 618,000,000 results (0.58 seconds)

“What is the most complicated design pattern that you have ever worked with?” And, we mostly agreed that the most complicated one is “**Visitor Pattern**”. Mar 15, 2021

<https://medium.com/geekculture/design-pattern-visitor...> :

[Design Pattern] Visitor Pattern — The Most Complicated ...



About featured snippets

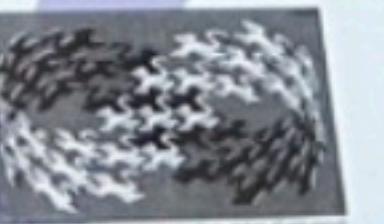


Feedback

# Design Patterns

Elements of Reusable  
Object-Oriented Software

Erich Gamma  
Richard Helm  
Ralph Johnson  
John Vlissides



Foreword by Grady Booch



**32 GAL / 121,13 L**

Heavy-Duty Outdoor Trash Can  
Poubelle robuste pour l'extérieur  
Basurero de exterior para  
trabajo pesado

22.37 in x 24.82 in x 32 in  
56.82 cm x 62.53 cm x 81.28 cm

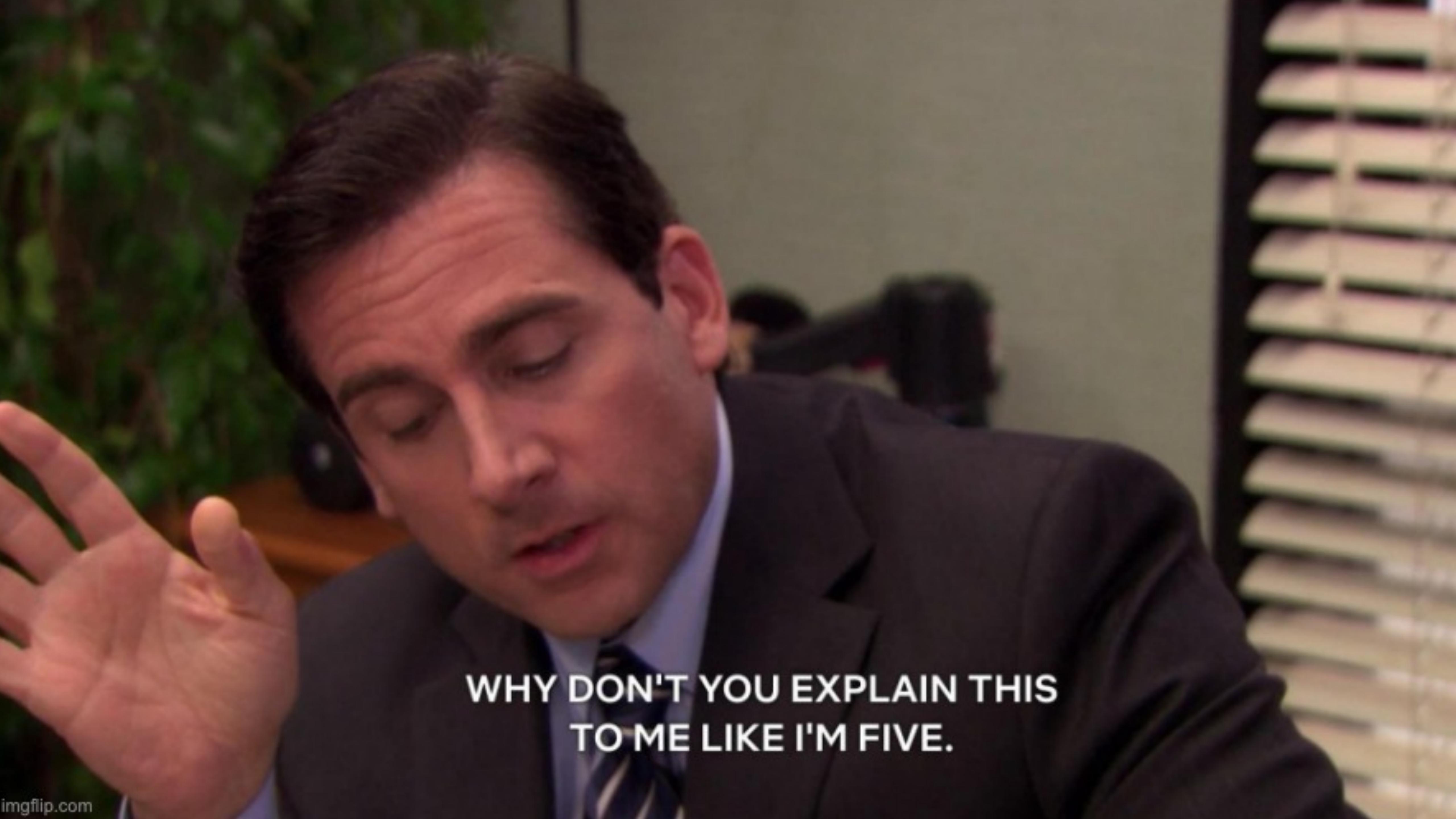
LIFETIME WARRANTY  
GARANTIE À VIE  
GARANTÍA DE POR VIDA

Stainless Steel  
Acier inoxydable  
Acero inoxidable  
Inoxidable acero

Made in U.S.A.  
Fabriqué au Canada  
Fabricado en U.S.A.  
Fabricado en U.S.A.

Model # 0227199

0227199

A close-up shot of Jim Halpert's face. He has a confused expression, with his eyebrows furrowed and his mouth slightly open. He is wearing a dark suit jacket, a light blue shirt, and a striped tie. His hands are clasped together in front of him.

WHY DON'T YOU EXPLAIN THIS  
TO ME LIKE I'M FIVE.



**Imagine that you sing  
birthday greetings.**

**You've been hired to sing  
at four conference rooms  
at the “**Lougle**”  
headquarters.**

## Today's visit to Lougle

Conference Room H

- Happy B'day to Bob

Conference Room K

- Happy B'day to Carol

Break Room, 4<sup>th</sup> floor

- Welcome, Ted

Loading Dock

- Farewell, Alice



Lougle

diskoffee

# **“Welcome to Loughle!**

**We’re excited to have you here to sing “Happy Birthday” to Bob, Carol, Ted, and Alice.”**





**“Here’s a  
map of the  
building.”**

**Have fun!”**

This isn't  
good.

# What's wrong?

- We shouldn't force a casual visitor to have to figure out where to go in an enormous building
- Security risk - what if the visitor wanders into a secure, or confidential, or dangerous area?
- What if the appointment changed? Maybe Carol's party got moved to a different room because the air conditioning was broken in Conference Room H?

**What's a  
better way?**



**The company is responsible for  
getting the visitor to the places  
she needs to go.**

**E.g., “Conference Room H”**

The visitor only knows that  
she'll be taken to the places  
she needs to be so that she  
can do her job there.

**That's the Visitor pattern in a nutshell.**

**It removes the burden of navigating a structure from visitors to the structure.**

**"ONCE MORE,  
WITH FEELING"  
CODE**

**Let's imagine we're a  
newspaper publisher  
for the web...**

✓  **newspaper** ~/Downloads/newspaper

>  .idea

✓  src

  ✓  main

    ✓  java

      ✓  org.officespace.publisher

        ✓  newspaper

          c FinancialArticle

          c Footer

          c Header

          c Headline

          c Newspaper

          c SportsArticle

          c Publisher

 resources

>  test

>  target

 newspaper.iml

 pom.xml

```
public static Newspaper createNewspaper() {  
    return Newspaper.builder()  
        .title("The Savannah Morning News")  
        .header(Header.builder().title("The News For CodeMash 2023 - Thursday January 11").build())  
        .headlines(Arrays.asList(  
            Headline.builder().text("Design Patterns Are Still Cool").build(),  
            Headline.builder().text("Whatever Happened To Ruby On Rails?").build()  
        ))  
        .sportsArticles(Arrays.asList(  
            SportsArticle.builder().title("UGA Wins Another Natty").build(),  
            SportsArticle.builder().title("TCU Had A Great Run").build()  
        ))  
        .financialArticles(Arrays.asList(  
            FinancialArticle.builder().title("NASDAQ Down Again").build(),  
            FinancialArticle.builder().title("401(k)s Not Looking Good").build()  
        ))  
        .footer(Footer.builder().copyright("An OfficeSpace Production (c) 2023").build())  
        .build();  
}
```

A man with dark hair and glasses, wearing a light blue shirt, a maroon vest, and a yellow patterned tie, is looking towards the right side of the frame with a slight smile. He appears to be in an office environment with other people and equipment visible in the background.

Hello, Peter, what's happening.  
I'm gonna need you to publish the  
newspaper to the Web.  
So, if you could do that by 5, that  
would be great.

```
public static String createHtmlVersion(final Newspaper newspaper) {
    StringBuilder html = new StringBuilder("<html><head></head><body>");
    html.append(toH1(newspaper.getTitle()));
    html.append(toH1(newspaper.getHeader().getTitle()));
    for (Headline headline : newspaper.getHeadlines()) {
        html.append(toH2(headline.getText()));
    }
    for (SportsArticle sportsArticle : newspaper.getSportsArticles()) {
        html.append(toH3(sportsArticle.getTitle()));
    }
    for (FinancialArticle financialArticle : newspaper.getFinancialArticles()) {
        html.append(toH3(financialArticle.getTitle()));
    }
    html.append(toH1(newspaper.getFooter().getCopyright()));
    html.append("</body></html>");
    return html.toString();
}

public static void main(final String[] args) {
    final String html = createHtmlVersion(createNewspaper());
    System.out.println(html);
}
```

A man with dark hair and glasses, wearing a light blue shirt, a maroon vest, and a yellow patterned tie, is looking towards the right. He appears to be in an office environment with other people and equipment in the background.

Hello, Peter, what's happening.  
The web version looks great, but  
I'm gonna need you to create a  
PDF version.  
So, if you could do that by 5, that  
would be great.

**CTRL-C**

**CTRL-V**

```
public static String createPDFVersion(final Newspaper newspaper) {  
    StringBuilder html = new StringBuilder(createPdfHeader());  
    html.append(createPdfSection(newspaper.getTitle()));  
    html.append(createPdfSection(newspaper.getHeader().getTitle()));  
    for (Headline headline : newspaper.getHeadlines()) {  
        html.append(createPdfSection(headline.getText()));  
    }  
    for (SportsArticle sportsArticle : newspaper.getSportsArticles()) {  
        html.append(createPdfSection(sportsArticle.getTitle()));  
    }  
    for (FinancialArticle financialArticle : newspaper.getFinancialArticles()) {  
        html.append(createPdfSection(financialArticle.getTitle()));  
    }  
    html.append(createPdfSection(newspaper.getFooter().getCopyright()));  
    html.append(createPdfFooter());  
    return html.toString();  
}  
  
public static void main(final String[] args) {  
    final String pdf = createPDFVersion(createNewspaper());  
    System.out.println(pdf);  
}
```

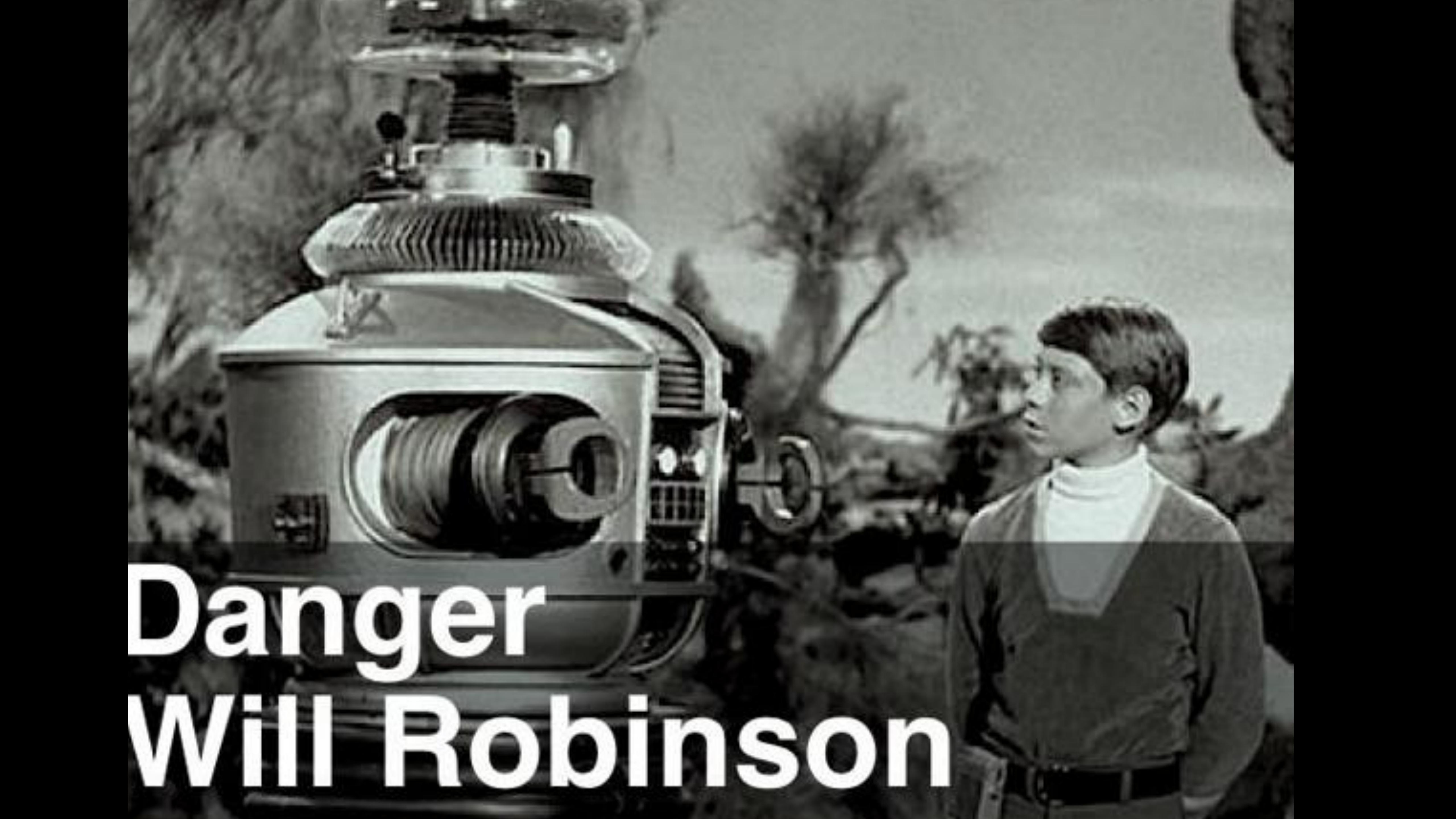
A close-up shot of a man with dark hair and glasses, wearing a light blue shirt, a yellow patterned tie, and a maroon vest. He has a surprised or shocked expression on his face. The background is blurred, showing what appears to be an office environment.

Hello, Peter, what's happening.  
Yeah, I'm gonna need you to create  
an ePub version for the Kindle.  
So, if you could do that by 5, that  
would be great.

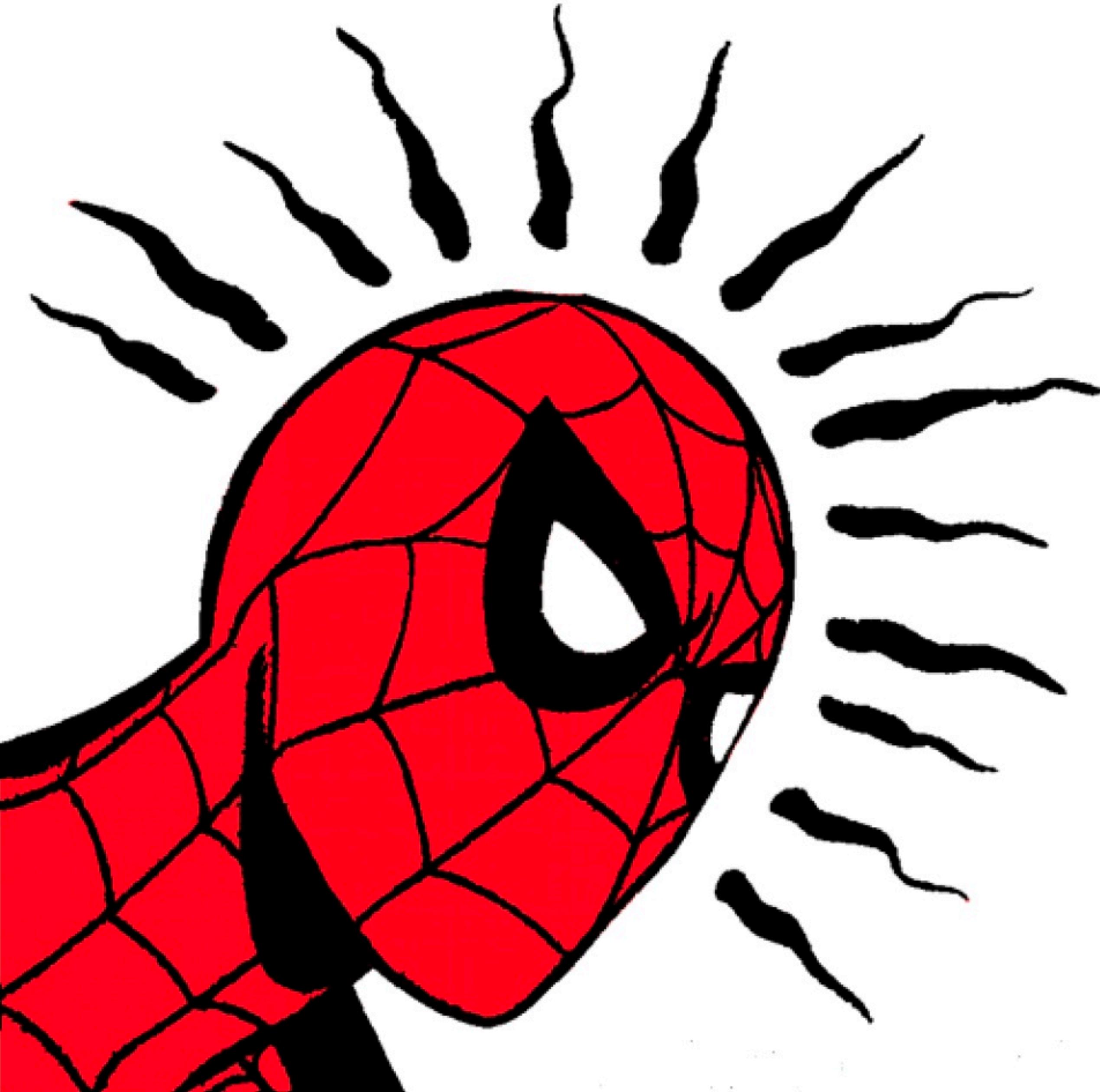
**CTRL-C**

**CTRL-V**

```
public static String createEpubVersion(final Newspaper newspaper) {  
    StringBuilder html = new StringBuilder(createEpubHeader());  
    html.append(createEpubFragment(newspaper.getTitle()));  
    html.append(createEpubFragment(newspaper.getHeader().getTitle()));  
    for (Headline headline : newspaper.getHeadlines()) {  
        html.append(createEpubFragment(headline.getText()));  
    }  
    for (SportsArticle sportsArticle : newspaper.getSportsArticles()) {  
        html.append(createEpubFragment(sportsArticle.getTitle()));  
    }  
    for (FinancialArticle financialArticle : newspaper.getFinancialArticles()) {  
        html.append(createEpubFragment(financialArticle.getTitle()));  
    }  
    html.append(createEpubFragment(newspaper.getFooter().getCopyright()));  
    html.append(createEpubFooter());  
    return html.toString();  
}  
  
public static void main(final String[] args) {  
    final String epub = createEpubVersion(createNewspaper());  
    System.out.println(epub);  
}
```



**Danger  
Will Robinson**



Your  
“developer sense”  
should be tingling



**STOP  
AND  
THINK**

# What's wrong with this code?

- **Duplication:** three functions that are all very similar
- We keep having to delve into **internal details** of the Newspaper object
- We have **navigation** code mixed with **business logic**
- It's **fragile**
  - What if the Newspaper internal structure changes?
  - What if we missed something (e.g., “Developer articles”)?

**What's the same?**

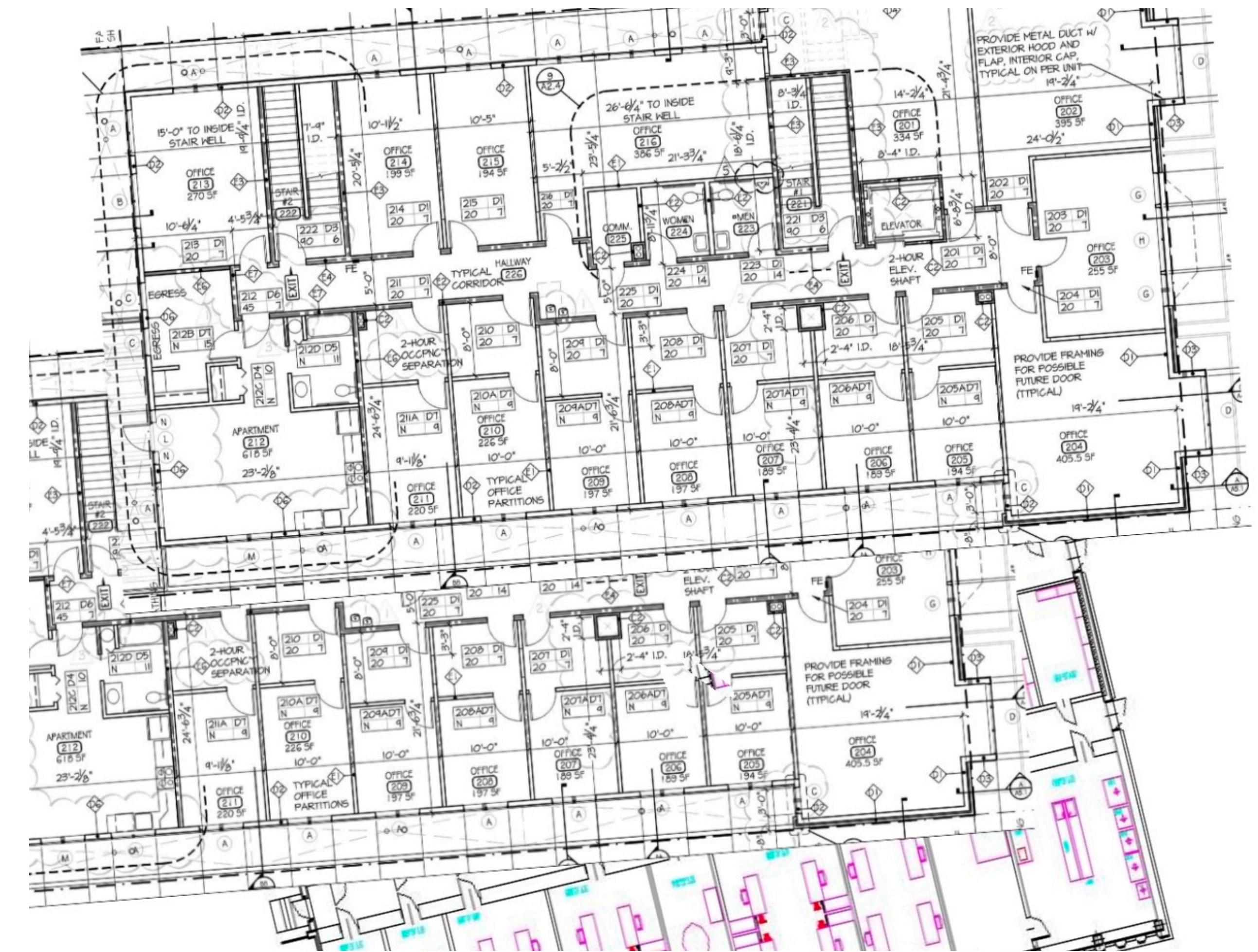
**What's different?**

```
public static String createPDFVersion(Final Newspaper newspaper) {  
    StringBuilder html = new StringBuilder(createPdfHeader());  
    html.append(createPdfSection(newspaper.getTitle()));  
    html.append(createPdfSection(newspaper.getHeader().getTitle()));  
    for (Headline headline : newspaper.getHeadlines()) {  
        html.append(createPdfSection(headline.getText()));  
    }  
    for (SportsArticle sportsArticle : newspaper.getSportsArticles()) {  
        html.append(createPdfSection(sportsArticle.getTitle()));  
    }  
    for (FinancialArticle financialArticle : newspaper.getFinancialArticles()) {  
        html.append(createPdfSection(financialArticle.getTitle()));  
    }  
    html.append(createPdfSection(newspaper.getFooter().getCopyright()));  
    html.append(createPdfFooter());  
    return html.toString();  
}
```

This prints  
the paper

This navigates the  
paper

# This is similar to a visitor having to navigate an unfamiliar building



**What if we  
separated those  
responsibilities?**

# We'd like to pull the PDF generation code out of the navigation code

```
public static String createPDFVersion(final Newspaper newspaper) {  
    // ...  
    newspaper.getTitle()  
    newspaper.getHeader().getTitle()  
    for (Headline headline : newspaper.getHeadlines()) {  
        headline.getText()  
    }  
    for (SportsArticle sportsArticle : newspaper.getSportsArticles()) {  
        sportsArticle.getTitle()  
    }  
    for (FinancialArticle financialArticle : newspaper.getFinancialArticles()) {  
        financialArticle.getTitle()  
    }  
    newspaper.getFooter().getCopyright()  
}
```

```
public static String createPDFVersion(final Newspaper newspaper) {  
    StringBuilder html = new StringBuilder(createPdfHeader());  
    html.append(createPdfSection("Headlines", newspaper.getHeadlines()));  
    html.append(createPdfSection("Sports Articles", newspaper.getSportsArticles()));  
    html.append(createPdfSection("Financial Articles", newspaper.getFinancialArticles()));  
    html.append(createPdfSection("Footer", newspaper.getFooter().getCopyright()));  
    html.append(createPdfFooter());  
    return html.toString();  
}
```

# The “intern” should do the navigation. The “visitor” should just be led around to do her job.

```
public static String createPDFVersion(final Newspaper newspaper) {  
    newspaper.getTitle()  
    newspaper.getHeader().getTitle()  
    for (Headline headline : newspaper.getHeadlines()) {  
        headline.getText()  
    }  
    for (SportsArticle sportsArticle : newspaper.getSportsArticles()) {  
        sportsArticle.getTitle()  
    }  
    for (FinancialArticle financialArticle : newspaper.getFinancialArticles()  
        financialArticle.getTitle()  
    }  
    newspaper.getFooter().getCopyright()  
}
```



```
public static String createPDFVersion(final Newspaper newspaper) {  
    StringBuilder html = new StringBuilder(createPdfHeader());  
    html.append(createPdfSection( ));  
    return html.toString();  
}
```



**Let's start with  
the visitor**



## Today's visit to Lougle

Conference Room H

- Happy B'day to Bob

Conference Room K

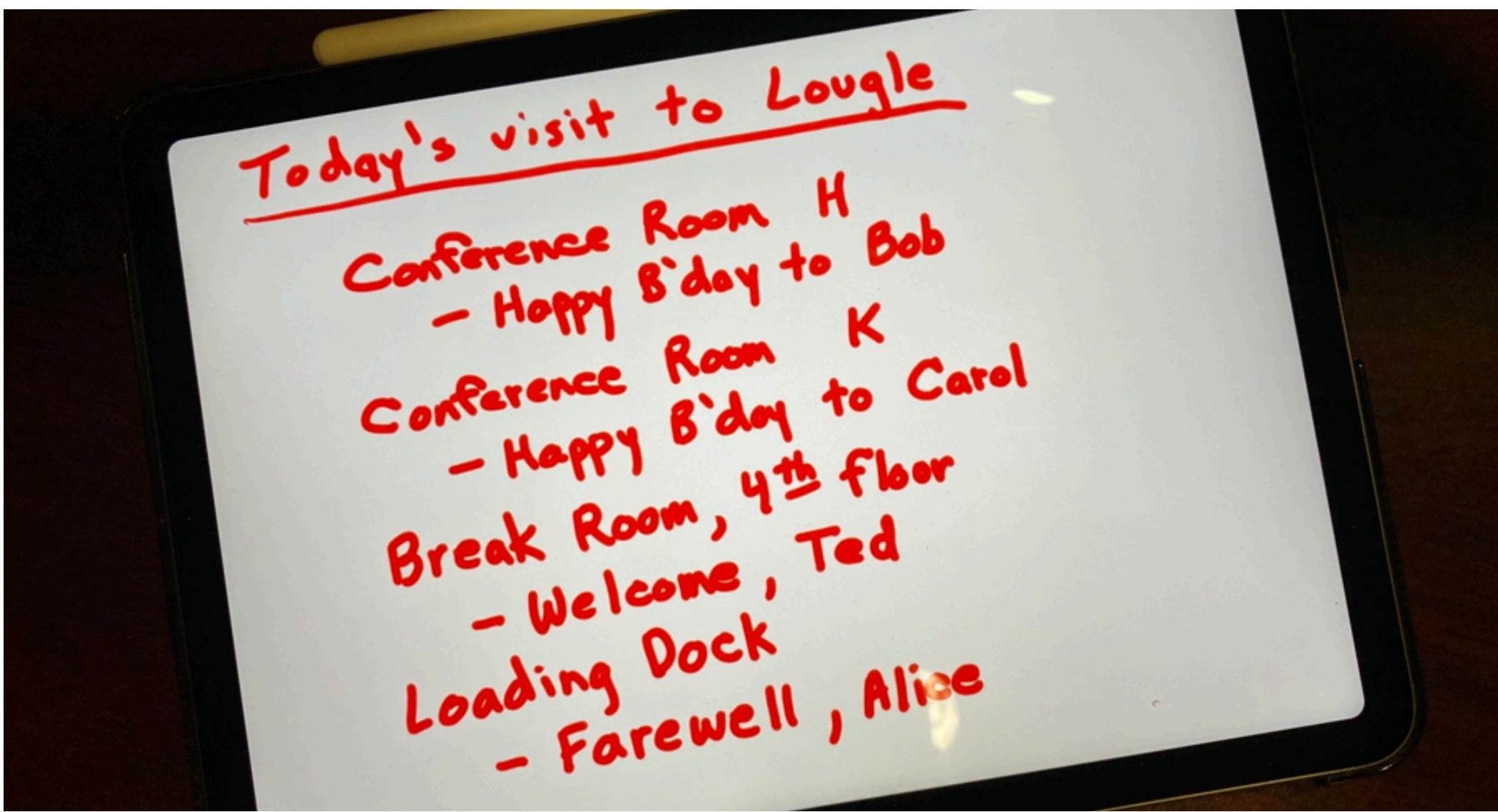
- Happy B'day to Carol

Break Room, 4<sup>th</sup> floor

- Welcome, Ted

Loading Dock

- Farewell, Alice



```
public class LougleVisitor {  
    public void whenIAmInConferenceRoomH() {  
        singHappyBirthdayTo("Bob");  
    }  
  
    public void whenIAmInConferenceRoomK() {  
        singHappyBirthdayTo("Carol");  
    }  
  
    public void whenIAmInTheFourthFloorBreakRoom() {  
        singWelcomeTo("Ted");  
    }  
  
    public void whenIAmOnTheLoadingDock() {  
        singleGoodbyeTo("Alice");  
    }  
}
```

```
public interface NewspaperVisitor {  
  
    public default void begin() {}  
  
    public default void doSomethingWithATitle(String title) {}  
  
    public default void doSomethingWithAHeader(Header header) {}  
  
    public default void doSomethingWithAHeadline(Headline headline) {}  
  
    public default void doSomethingWithASportsArticle(SportsArticle sportsArticle) {}  
  
    public default void doSomethingWithAFinancialArticle(FinancialArticle financialArticle) {}  
  
    public default void doSomethingWithAFooter(Footer footer) {}  
  
    public default void end() {}  
}
```

```
public class PDFVisitor implements NewspaperVisitor {  
  
    StringBuilder pdf = new StringBuilder(createPdfHeader());  
  
    @Override  
    public void doSomethingWithAHeader(Header header) {  
        pdf.append(createPdfSection(header.getTitle()));  
    }  
  
    @Override  
    public void doSomethingWithATitle(String title) {  
        pdf.append(createPdfSection(title));  
    }  
  
    @Override  
    public void doSomethingWithAHeadline(Headline headline) {  
        pdf.append(createPdfSection(headline.getText()));  
    }  
  
    @Override  
    public void doSomethingWithASportsArticle(final SportsArticle sportsArticle) {  
        pdf.append(createPdfSection(sportsArticle.getTitle()));  
    }  
  
    public String getPdf() {  
        return pdf.toString();  
    }  
}
```



Now, let's put the  
navigation code in  
the **Newspaper** class



# We want to move this navigation code into the `Newspaper` class

```
public static String createPDFVersion(final Newspaper newspaper) {  
    newspaper.getTitle()  
    newspaper.getHeader().getTitle()  
    for (Headline headline : newspaper.getHeadlines()) {  
        headline.getText()  
    }  
    for (SportsArticle sportsArticle : newspaper.getSportsArticles()) {  
        sportsArticle.getTitle()  
    }  
    for (FinancialArticle financialArticle : newspaper.getFinancialArticles()) {  
        financialArticle.getTitle()  
    }  
    newspaper.getFooter().getCopyright()  
}
```



```
public class Newspaper {  
    public void accept(NewspaperVisitor visitor) {  
        visitor.begin();  
        visitor.doSomethingWithATitle(title);  
        visitor.doSomethingWithAHeader(header);  
        for (Headline headline : headlines) {  
            visitor.doSomethingWithAHeadline(headline);  
        }  
        for (SportsArticle sportsArticle : sportsArticles) {  
            visitor.doSomethingWithASportsArticle(sportsArticle);  
        }  
        for (FinancialArticle financialArticle : financialArticles) {  
            visitor.doSomethingWithAFinancialArticle(financialArticle);  
        }  
        visitor.doSomethingWithAFooter(footer);  
        visitor.end();  
    }  
}
```



**Finally, let's  
hook it all up**

```
public static void main(final String[] args) {  
  
    Newspaper newspaper = createNewspaper();  
  
    PDFVisitor pdfVisitor = new PDFVisitor();  
    newspaper.accept(pdfVisitor);  
  
    System.out.println(pdfVisitor.getPdf());  
}
```

# What have we accomplished?

- The navigation code is now with the data structure itself, not with the users
- Users no longer need to know how to navigate the data structure
  - They just specify what they want to do for the items in it
- We can change the internal data structure without affecting users
- We can add new items to the data structure without breaking existing code
- We've separated the business logic from the data structure navigation

# What if we wanted to create an HTML page?

```
public class HTMLVisitor implements NewspaperVisitor {  
  
    private final StringBuilder html = new StringBuilder("<html><head></head><body>");  
  
    @Override  
    public void doSomethingWithAHeader(Header header) {  
        html.append(toH1(header.getTitle()));  
    }  
  
    @Override  
    public void doSomethingWithATitle(String title) {  
        html.append(toH1(title));  
    }  
  
    @Override  
    public void doSomethingWithAHeadline(Headline headline) {  
        html.append(toH2(headline.getText()));  
    }  
  
    public String getHTML() {  
        return html.append("</body></html>").toString();  
    }  
  
    private String toH1(final String text) {  
        return "<h1>" + text + "</h1>";  
    }  
}
```

```
public static void main(final String[] args) {  
      
    Newspaper newspaper = createNewspaper();  
      
    HTMLVisitor htmlVisitor = new HTMLVisitor();  
    newspaper.accept(htmlVisitor);  
    System.out.println(htmlVisitor.getHTML());  
      
    PDFVisitor pdfVisitor = new PDFVisitor();  
    newspaper.accept(pdfVisitor);  
    System.out.println(pdfVisitor.getPdf());  
}
```

# We can do some interesting things...

```
public class ArticleCounter implements NewspaperVisitor {  
  
    private int sportsArticleCount;  
  
    public void doSomethingWithASportsArticle(SportsArticle sportsArticle) {  
        ++sportsArticleCount;  
    }  
  
    public int getSportsArticleCount() {  
        return sportsArticleCount;  
    }  
}
```

# This could be the beginning of a statistics class

```
Newspaper newspaper = createNewspaper();  
  
ArticleCounter articleCounter = new ArticleCounter();  
newspaper.accept(articleCounter);  
System.out.println("There are " + articleCounter.getSportsArticleCount() + " sports articles.");
```

# We could use this for filtering

```
public class OnlyShowGeorgiaSportsArticles implements NewspaperVisitor {  
  
    @Override  
    public void doSomethingWithASportsArticle(SportsArticle sportsArticle) {  
        if (sportsArticle.getTitle().contains("Georgia")) {  
            showArticleOnFrontPage(sportsArticle);  
        } else {  
            /* ignore it */  
        }  
    }  
}
```

**That's the  
visitor pattern\***

The “Gang of Four” pattern is slightly different.

There, each object in the hierarchy implements the accept(...) method.

```
]@Builder  
]@Getter  
public class FinancialArticle {  
  
    private final String title;  
  
    public void accept(NewspaperVisitor newspaperVisitor) {  
        newspaperVisitor.doSomethingWithAFinancialArticle(this);  
    }  
}
```

```
@Builder  
@Getter  
public class SportsArticle {  
  
    private final String title;  
  
    public void accept(NewspaperVisitor newspaperVisitor) {  
        newspaperVisitor.doSomethingWithASportsArticle(this);  
    }  
}
```

```
public class Newspaper {  
  
    public void accept(NewspaperVisitor visitor) {  
        visitor.begin();  
        visitor.doSomethingWithATitle(title);  
        visitor.doSomethingWithAHeader(header);  
        for (Headline headline : headlines) {  
            visitor.doSomethingWithAHeadline(headline);  
        }  
        for (SportsArticle sportsArticle : sportsArticles) {  
            visitor.doSomethingWithASportsArticle(sportsArticle);  
        }  
        for (FinancialArticle financialArticle : financialArticles)  
            visitor.doSomethingWithAFinancialArticle(financialArti  
        }  
        visitor.doSomethingWithAFooter(footer);  
        visitor.end();  
    }  
}
```

```
public class Newspaper {  
  
    public void accept(NewspaperVisitor visitor) {  
        visitor.begin();  
        visitor.doSomethingWithATitle(title);  
        header.accept(visitor);  
        for (Headline headline : headlines) {  
            headline.accept(visitor);  
        }  
        for (SportsArticle sportsArticle : sportsArticles) {  
            sportsArticle.accept(visitor);  
        }  
        for (FinancialArticle financialArticle : financialArticles) {  
            financialArticle.accept(visitor);  
        }  
        footer.accept(visitor);  
        visitor.end();  
    }  
}
```

# TypeScript implementations

```
class Header {
|   private title: string;
}

class Headline {
|   private title: string;
}

class SportsArticle {
|   private title: string;
}

class FinancialArticle {
|   private title: string;
}

class Footer {
|   private title: string;
}

class Newspaper {
    private title: string;
    private header: Header;
    private headlines: Headline[];
    private sportsArticles: SportsArticle[];
    private financialArticles: FinancialArticle[];
    private footer: Footer;
}
```

**Let's use  
ChatGPT3!**



show me how to add the visitor design pattern to this TypeScript code

```
class Header {  
    private title: string;  
}  
  
class Headline {  
    private title: string;  
}  
  
class SportsArticle {  
    private title: string;  
}  
  
class FinancialArticle {  
    private title: string;  
}  
  
class Footer {  
    private title: string;  
}  
  
class Newspaper {  
    private title: string;  
    private header: Header;  
    private headlines: Headline[];  
    private sportsArticles: SportsArticle[];  
    private financialArticles: FinancialArticle[];  
    private footer: Footer;  
}
```

Here is an example of how you might add the Visitor pattern to the given TypeScript code:

Copy code

```
interface Visitor {
    visitHeader(header: Header): void;
    visitHeadline(headline: Headline): void;
    visitSportsArticle(sportsArticle: SportsArticle): void;
    visitFinancialArticle(financialArticle: FinancialArticle): void;
    visitFooter(footer: Footer): void;
}

class Header {
    private title: string;

    accept(visitor: Visitor): void {
        visitor.visitHeader(this);
    }
}

class Headline {
    private title: string;

    accept(visitor: Visitor): void {
        visitor.visitHeadline(this);
    }
}

class SportsArticle {
    private title: string;

    accept(visitor: Visitor): void {

```

```
class Footer {
    private title: string;

    accept(visitor: Visitor): void {
        visitor.visitFooter(this);
    }
}

class Newspaper {
    private title: string;
    private header: Header;
    private headlines: Headline[];
    private sportsArticles: SportsArticle[];
    private financialArticles: FinancialArticle[];
    private footer: Footer;

    accept(visitor: Visitor): void {
        this.header.accept(visitor);
        this.headlines.forEach(headline => headline.accept(visitor));
        this.sportsArticles.forEach(sportsArticle =>
            sportsArticle.accept(visitor));
        this.financialArticles.forEach(financialArticle =>
            financialArticle.accept(visitor));
        this.footer.accept(visitor);
    }
}
```

In this example, the `Header`, `Headline`, `SportsArticle`, `FinancialArticle`, and `Footer` classes have been given an `accept()` method that takes a `Visitor` as an argument. The `Newspaper` class also has an `accept()` method that calls the `accept()` method of each of its properties, including the arrays of `Headline`, `SportsArticle`, and `FinancialArticle`.

You can create a visitor class that implements the Visitor interface, and then call the accept method of the newspaper object, passing the visitor as a parameter, in this way the visitor will visit all the components of the newspaper.

 Copy code

```
class PrintVisitor implements Visitor {
    visitHeader(header: Header): void {
        console.log(`Header: ${header.title}`);
    }
    visitHeadline(headline: Headline): void {
        console.log(`Headline: ${headline.title}`);
    }
    visitSportsArticle(sportsArticle: SportsArticle): void {
        console.log(`Sports Article: ${sportsArticle.title}`);
    }
    visitFinancialArticle(financialArticle: FinancialArticle): void {
        console.log(`Financial Article: ${financialArticle.title}`);
    }
    visitFooter/footer(footer: Footer):
```

# I'M KELLY L. MORRISON

ANY QUESTIONS?



# Daugherty

BUSINESS SOLUTIONS

Work: [kelly.morrison@daugherty.com](mailto:kelly.morrison@daugherty.com)

Home: [kellyivymorrison@gmail.com](mailto:kellyivymorrison@gmail.com)

**Kelly Morrison**  
Manager / Application Architect at Daugherty  
Business Solutions



[LinkedIn](#)

