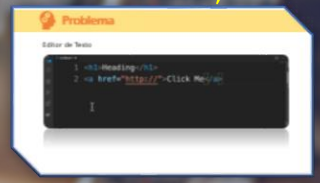
PATRONES DE DISEÑO

Patrones de Comportamiento - Visitor

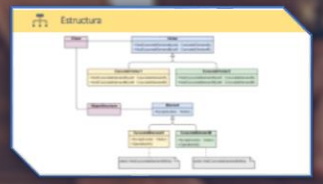
PATRON DE COMPORTAMIENTO - VISITOR

Propósito



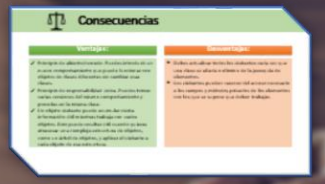
Problema

Solución

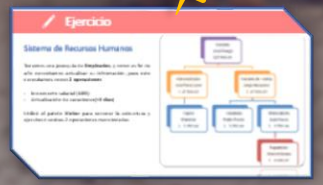


Estructura

Aplicabilidad



Consecuencias



Ejercicio

Tarea

Propósito

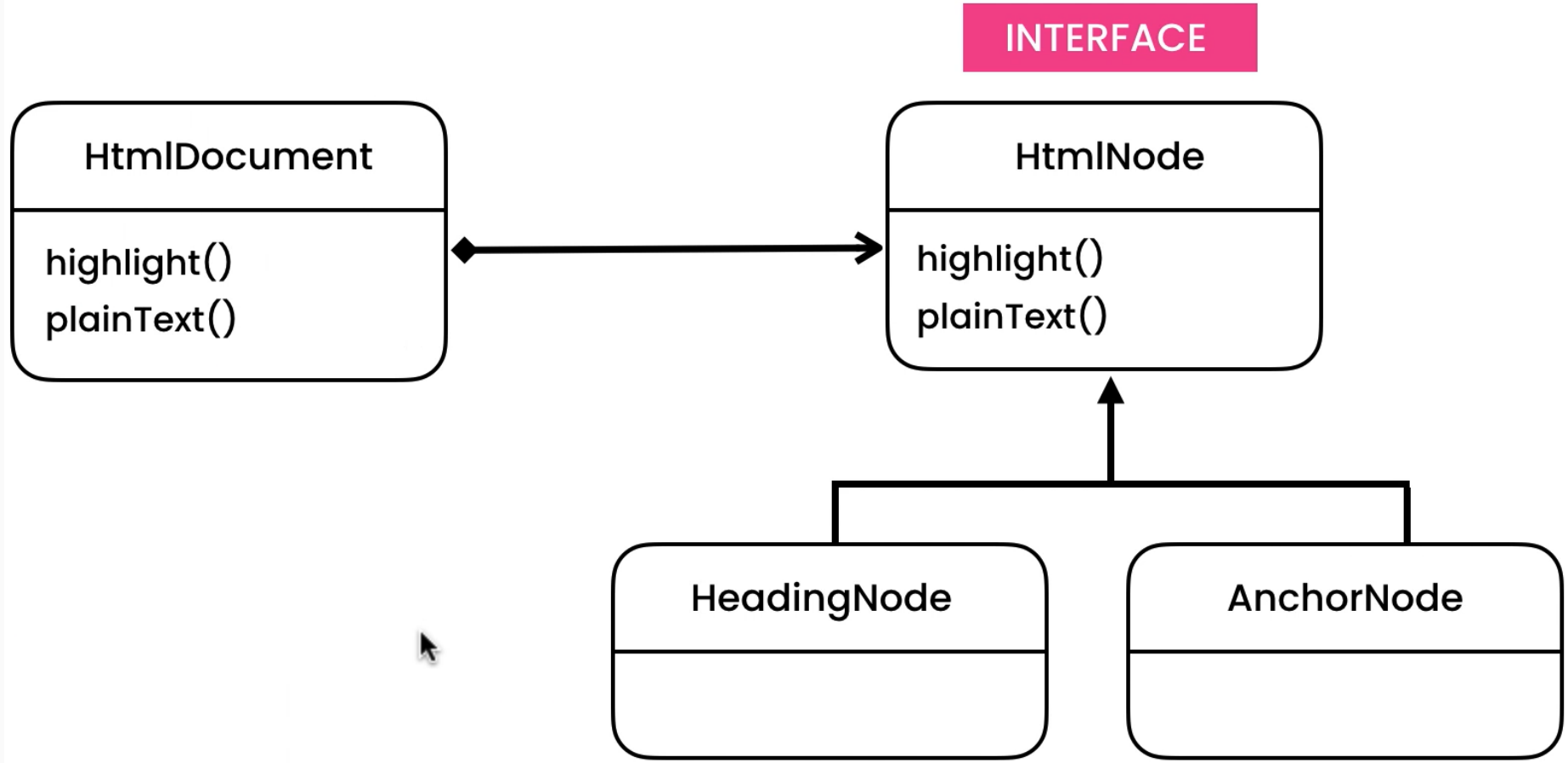
Visitor, es un patrón de diseño de comportamiento que "*Representa una operación sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera*".

Problema

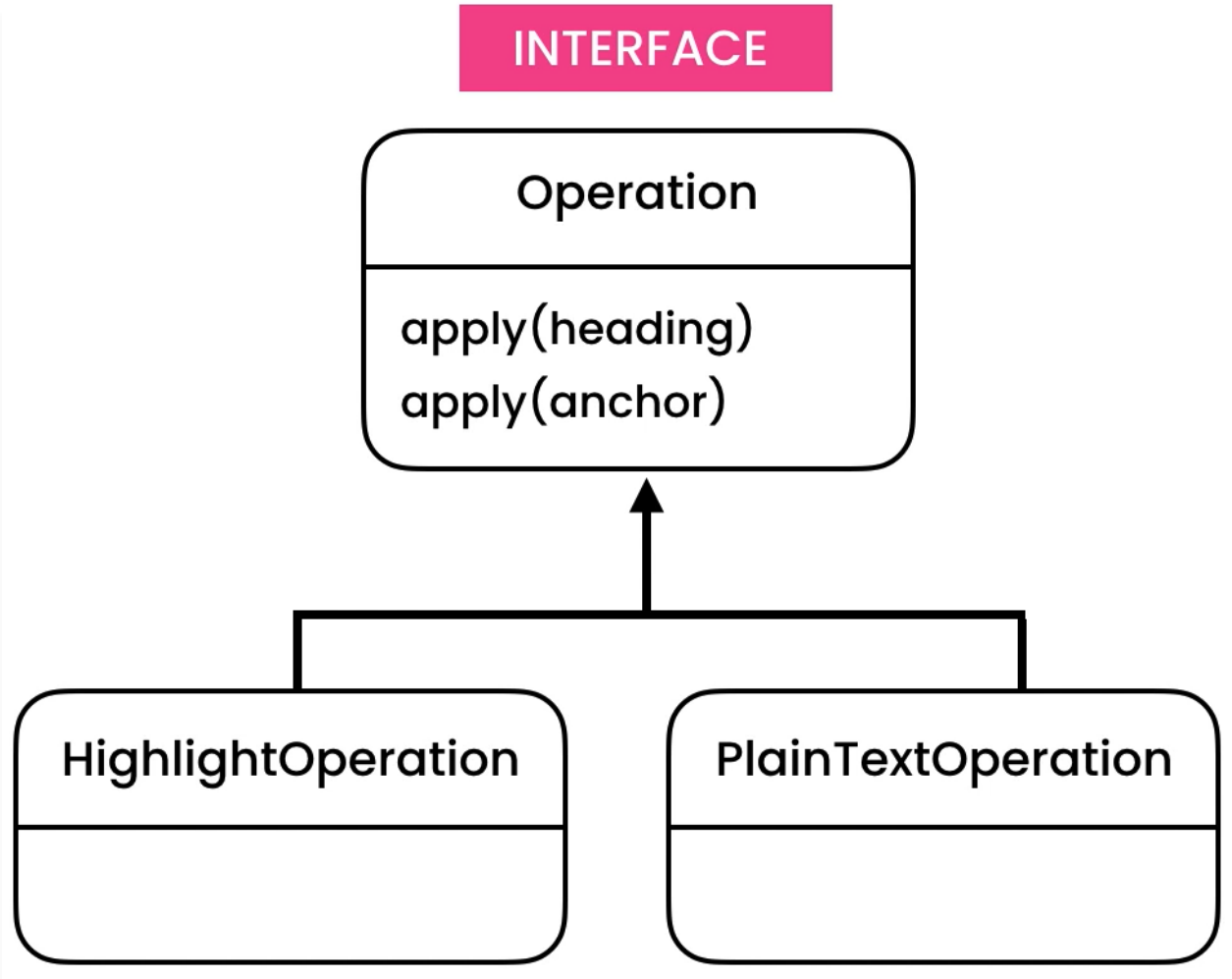
Editor de Texto

Demo

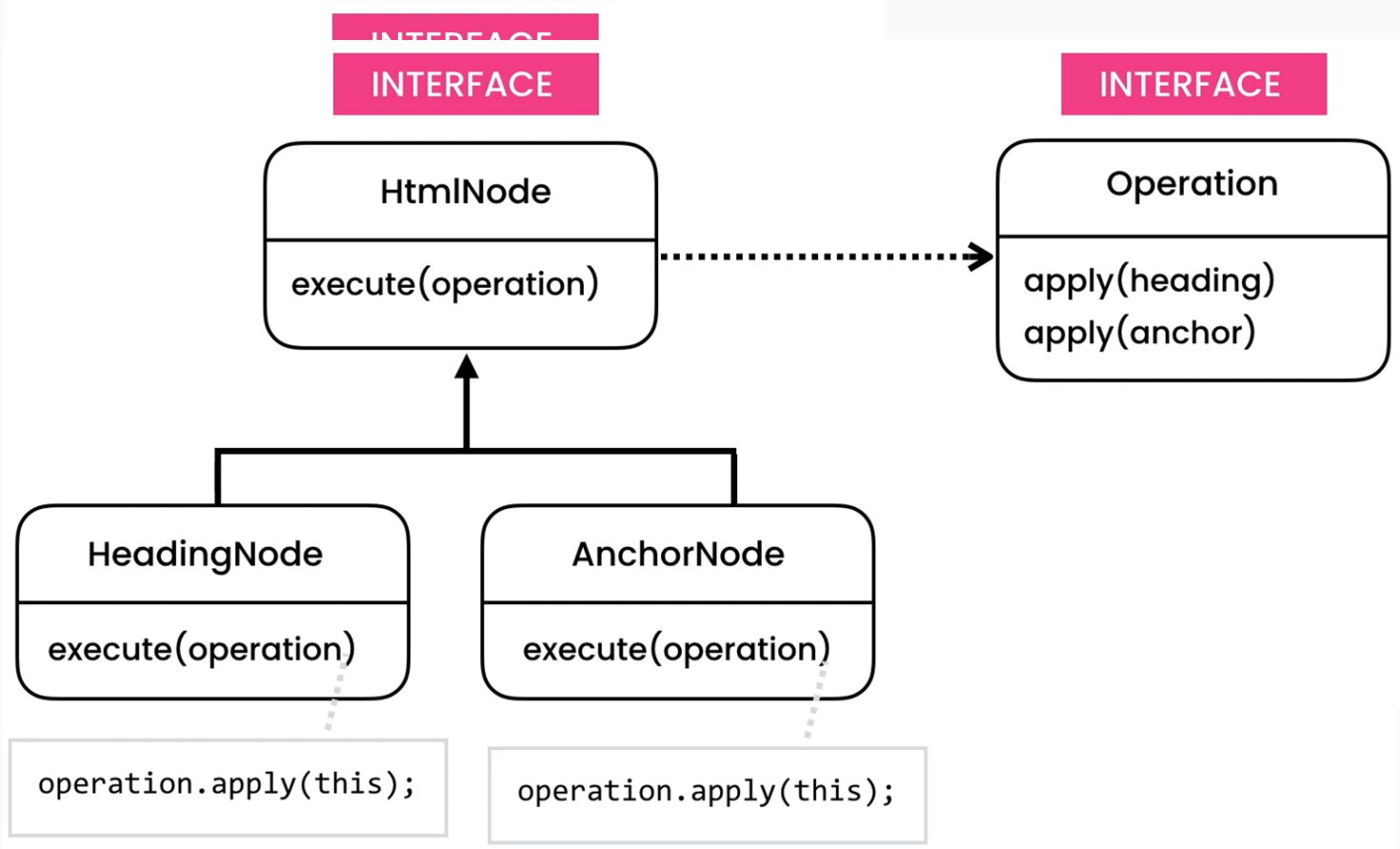
REVISEMOS EN CÓDIGO LA IMPLEMENTACIÓN DEL PROBLEMA


INTERFACE
HtmlDocument
highlight()
plainText()
HtmlNode
highlight()
plainText()
HeadingNode
AnchorNode

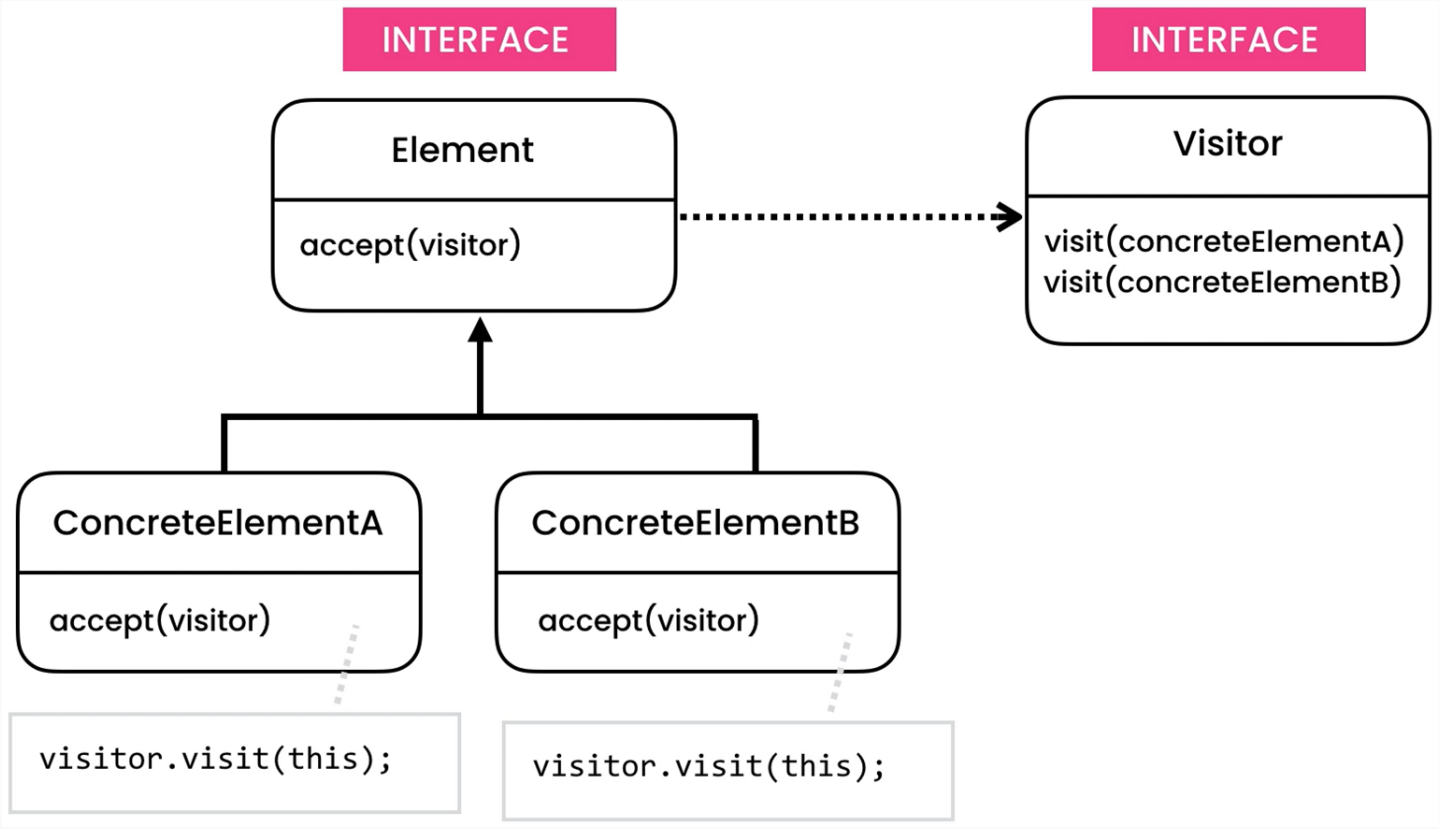
Solución


INTERFACE
Operation
apply(heading)
apply(anchor)
HighlightOperation
PlainTextOperation

Solución


INTERFACE
INTERFACE
HtmlNode
execute(operation)
Operation
apply(heading)
apply(anchor)
HeadingNode
execute(operation)
AnchorNode
execute(operation)
operation.apply(this);
operation.apply(this);

Solución

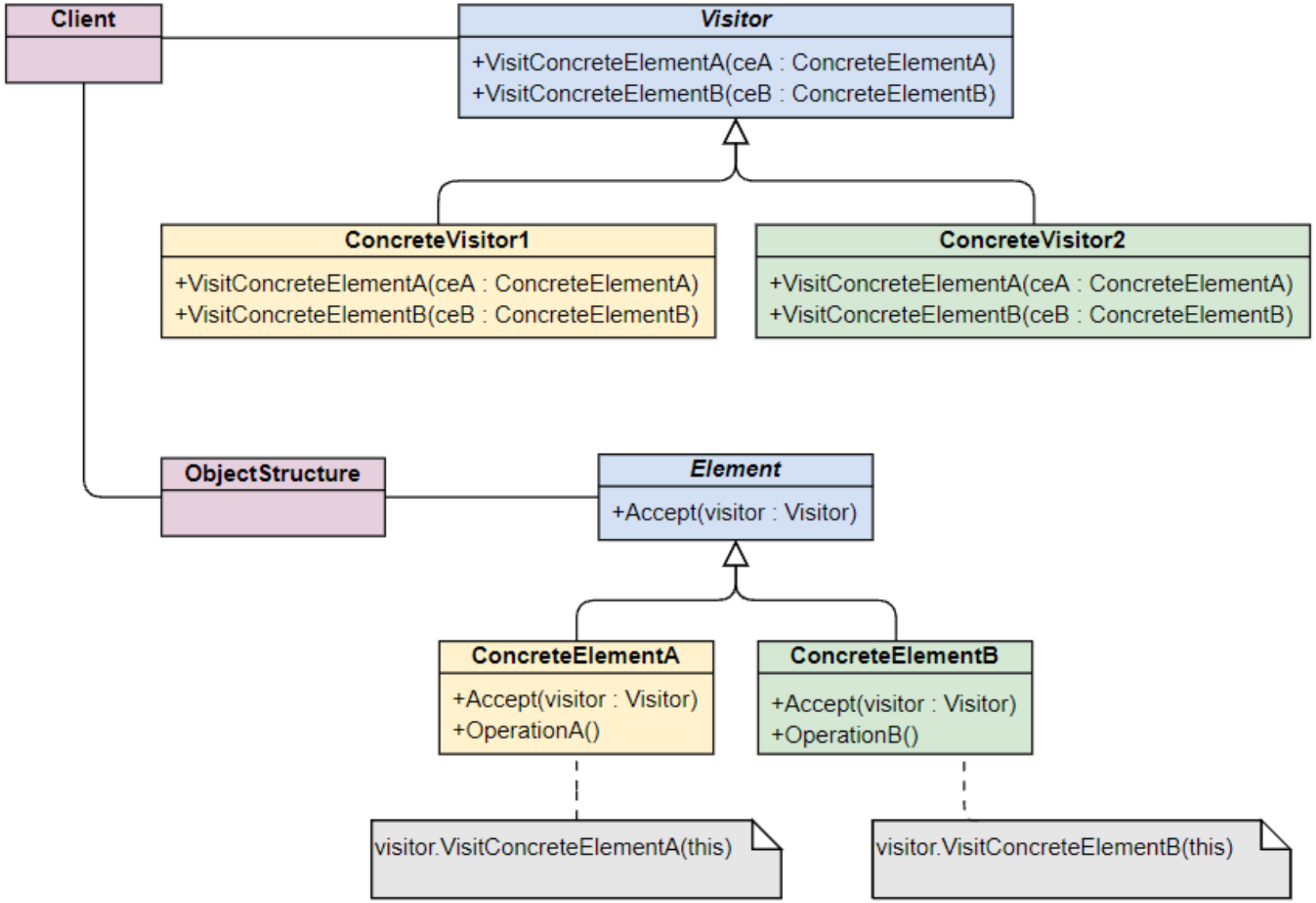

INTERFACE
INTERFACE
Element
accept(visitor)
Visitor
visit(concreteElementA)
visit(concreteElementB)
ConcreteElementA
accept(visitor)
ConcreteElementB
accept(visitor)
visitor.visit(this);
visitor.visit(this);

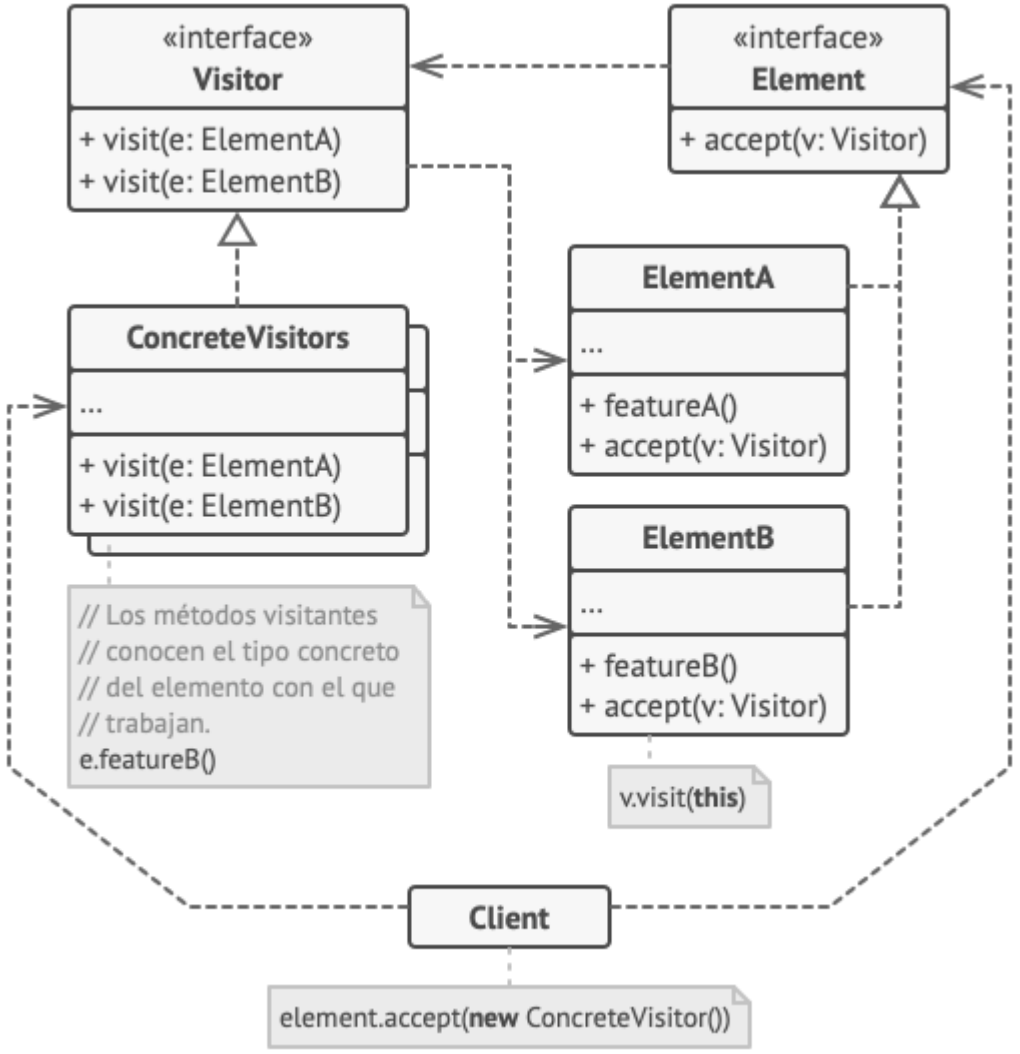
Solución

Demo

REVISEMOS EN CÓDIGO LA IMPLEMENTACIÓN DE LA SOLUCIÓN

Estructura


Client
Visitor
+VisitConcreteElementA(ceA : ConcreteElementA)
+VisitConcreteElementB(ceB : ConcreteElementB)
ConcreteVisitor1
+VisitConcreteElementA(ceA : ConcreteElementA)
+VisitConcreteElementB(ceB : ConcreteElementB)
ConcreteVisitor2
+VisitConcreteElementA(ceA : ConcreteElementA)
+VisitConcreteElementB(ceB : ConcreteElementB)
ObjectStructure
Element
+Accept(visitor : Visitor)
ConcreteElementA
+Accept(visitor : Visitor)
+OperationA()
ConcreteElementB
+Accept(visitor : Visitor)
+OperationB()
visitor.VisitConcreteElementA(this)
visitor.VisitConcreteElementB(this)



- Declara la interfaz visitante con un grupo de métodos “visitantes”
- Declara la interfaz de elemento.
- Implementa los métodos de aceptación en todas las clases de elemento concreto.
- Las clases de elemento sólo deben funcionar con visitantes a través de la interfaz visitante.
- Revisa el comportamiento que no pueda implementarse dentro de la jerarquía de elementos.
- Revisa si el visitante necesita acceso a algunos miembros privados de la clase elemento.
- El cliente debe crear objetos visitantes y pasarlos dentro de elementos a través de métodos de “aceptación”.

• Implementación

Aplicabilidad

Cuándo utilizar este patrón?

Utiliza el patrón Visitor cuando necesites realizar una operación sobre todos los elementos de una compleja estructura de objetos (por ejemplo, un árbol de objetos).

Utiliza el patrón Visitor para limpiar la lógica de negocio de comportamientos auxiliares.

Utiliza el patrón cuando un comportamiento solo tenga sentido en algunas clases de una jerarquía de clases, pero no en otras.

Consecuencias

Ventajas:

- ✓ Principio de abierto/cerrado. Puedes introducir un nuevo comportamiento que puede funcionar con objetos de clases diferentes sin cambiar esas clases.
- ✓ Principio de responsabilidad única. Puedes tomar varias versiones del mismo comportamiento y ponerlas en la misma clase.
- ✓ Un objeto visitante puede acumular cierta información útil mientras trabaja con varios objetos. Esto puede resultar útil cuando quieras atravesar una compleja estructura de objetos, como un árbol de objetos, y aplicar el visitante a cada objeto de esa estructura.

Desventajas:

- × Debes actualizar todos los visitantes cada vez que una clase se añada o elimine de la jerarquía de elementos.
- × Los visitantes pueden carecer del acceso necesario a los campos y métodos privados de los elementos con los que se supone que deben trabajar.

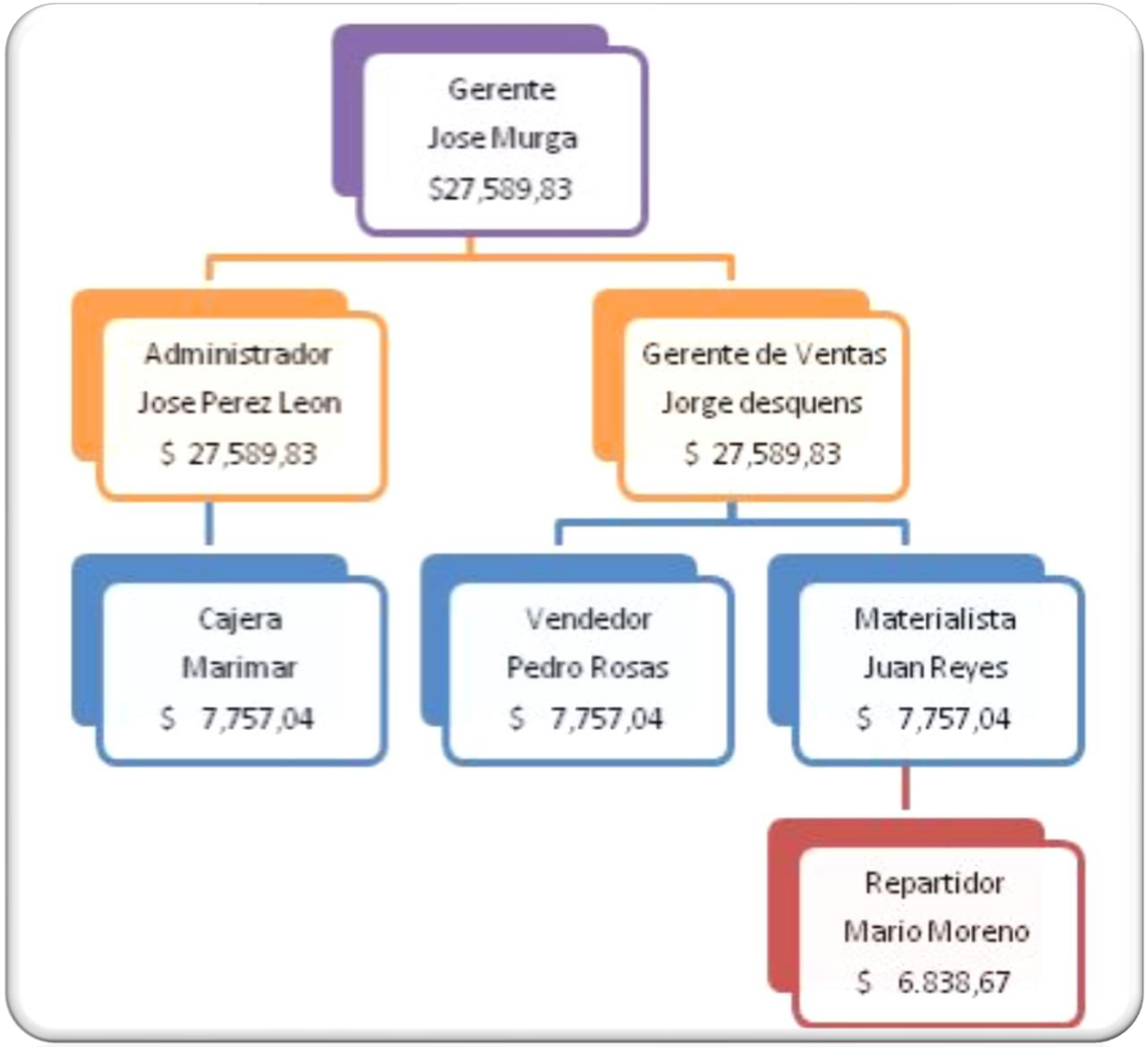
Ejercicio

Sistema de Recursos Humanos

Tenemos una jerarquía de **Empleados**, y como es fin de año necesitamos actualizar su información, para esto necesitamos correr **2 operaciones**

- Incremento salarial (**10%**)
- Actualización de vacaciones**(+3 días**)

Utilicé el patrón **Visitor** para recorrer la estructura y ejecutar nuestras 2 operaciones mencionadas



Demo

REVISEMOS EN CÓDIGO LA IMPLEMENTACIÓN LA PRÁCTICA

Tarea

Sistema pago impuestos para una tienda

Tenemos 3 tipos de **elementos** de consume que se venden en nuestra tienda y que pagan diferentes impuestos segun su tipo:

- Licor,
- Tabaco y
- Necesario (Canasta familiar).

Implemente el **calculo impuestos** para cada uno de estos elementos utilizando el patron **Visitor**

Muchas Gracias...

los espero la siguiente clase

DEMO

REVISEMOS EN CÓDIGO LA IMPLEMENTACIÓN DEL PROBLEMA