UCAM UNIVERSIDAD CATÓLICA DE MURCIA

Modelado del Software

Aplicación del Patrón Visitor

2017/2018 Julio

t

Eduardo Fernández Campuzano – Práctica 2
UCAM– Email: efernandez984@alu.ucam.edu

Índice

- Descripción del patrón
- Propósito
- Motivación
- Participantes
- Usos
- Diagrama general
- Código Java

Descripción

- Es un patrón de comportamiento, es decir, enfatizan la colaboración entre objetos y se relacionan con la asignación de responsabilidades entre clases.
- Este, se utiliza para separar la lógica u operaciones que se pueden realizar sobre una estructura compleja. En ocasiones nos podemos encontrar con estructura de datos que requieren realizar operaciones sobre ella, pero estas operaciones pueden ser muy variadas e incluso se pueden desarrollar nuevas.

Propósito

Representar una operación que debe ser aplicada sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera. Separa el algoritmo de la estructura.

Motivación

Un compilador que representa los programas como árboles de derivación de la sintaxis abstracta, sobre los que ejecuta operaciones: comprobación de tipos, generación de código,… y además listados de código fuente, reestructuración de programas…

Participantes

- Visitor: Define una operación de visita para cada clase de un elemento concreto.
- ConcreteVisitor(ConcreteElement):
  - Implementa la interfaz “Visitor” y la operación “visit”.
- Element: define una operación “accept” con “visitor” como argumento.
- ConcreteElement(AssignmentNode): Implementa la operación “accept”
- ObjectStructure(compiler): Gestiona la estructura de objetos, y puede enumerar sus elementos.

Usos

- Se tienen muchas clases de objetos con diferentes interfaces y se quiere realizar operaciones sobre los objetos que dependen de las clases concretas.
- Es muy útil cuando las clases de la jerarquía no cambian, pero se añaden con frecuencia operaciones a la estructura.
- Si la jerarquía cambia no es aplicable, ya que cada vez que agrega nuevas clases que deben ser visitadas, hay que añadir una operación “visita” abstracta a la clase Visitante, y debe agregar una aplicación de dicha categoría a cada Visitante concreto que se haya escrito.

Ventajas

- Las clases visitadas por un mismo Visitor pueden no estar relacionadas entre si a través de la herencia.
- Con una instancia de un Visitor Concreto se puede recorrer la jerarquía
- Es fácil añadir nuevas operaciones a un programa

Inconvenientes

- Es difícil añadir nuevas clases de elementos ya que obliga a cambiar los visitantes.
- Recomendable aplicar el patrón Visitor solo usarlo en caso de que la jerarquía de clases sea estable.
- Nuevas operaciones requerirán recompilar todas las clases.

Diagrama general

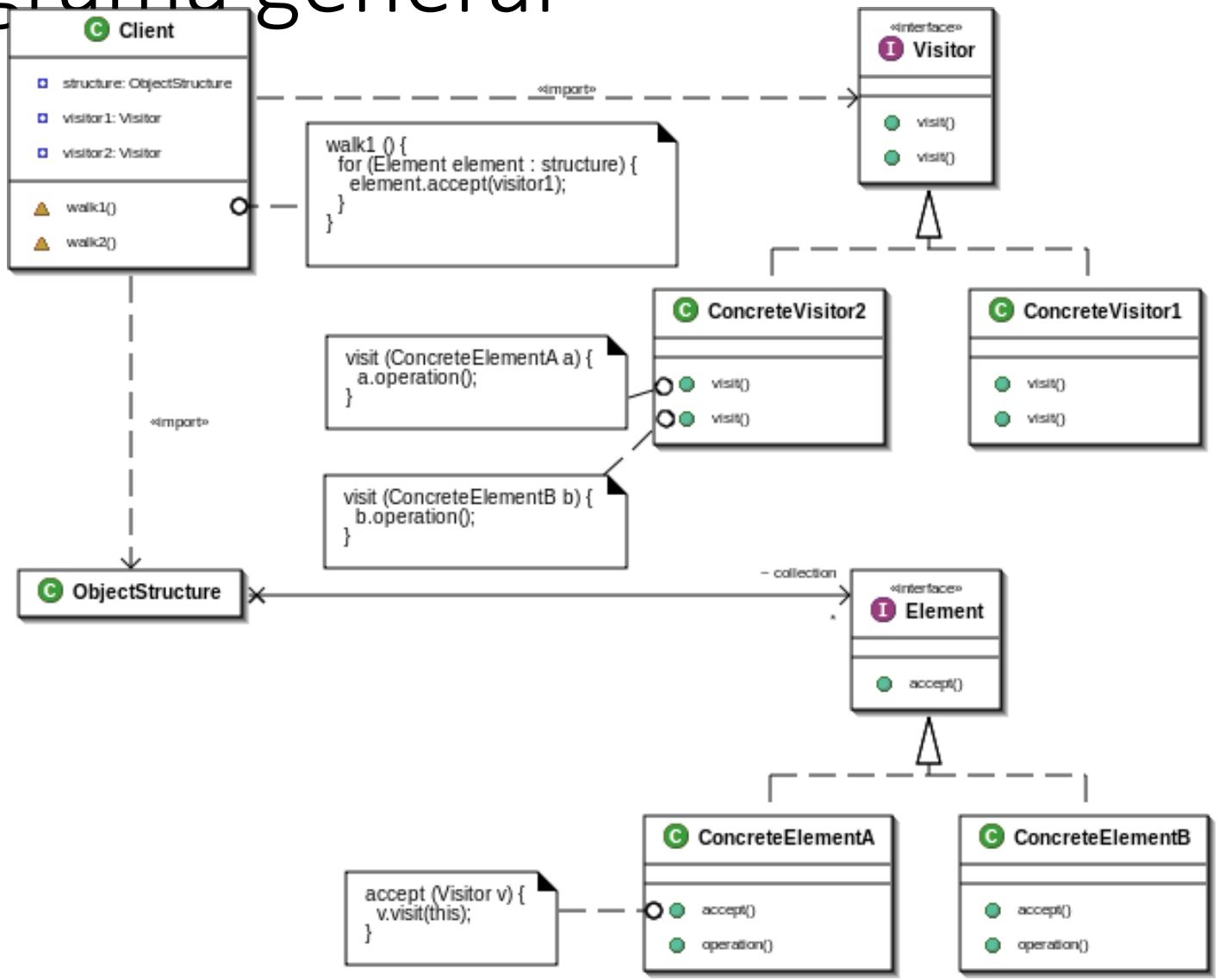
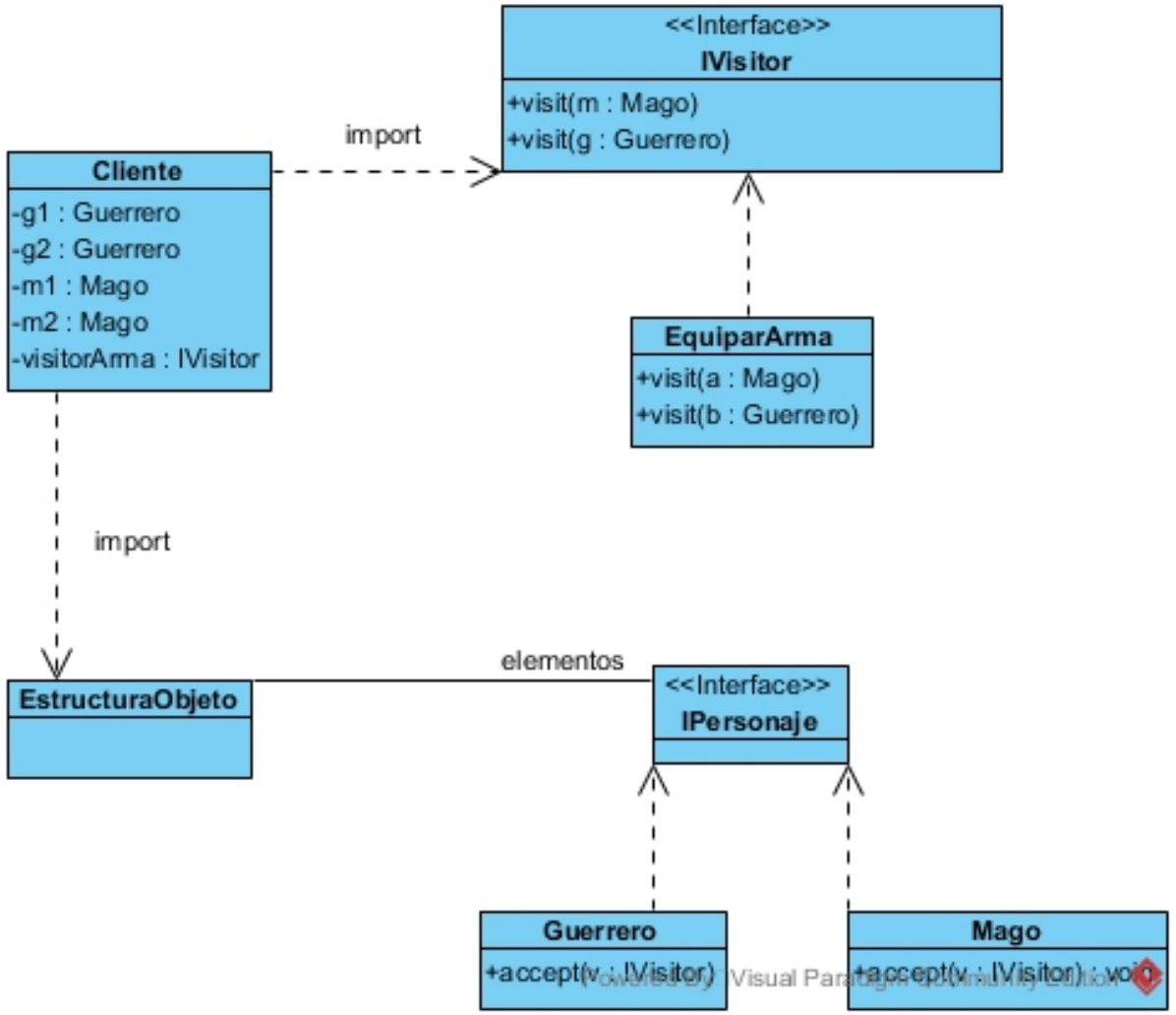
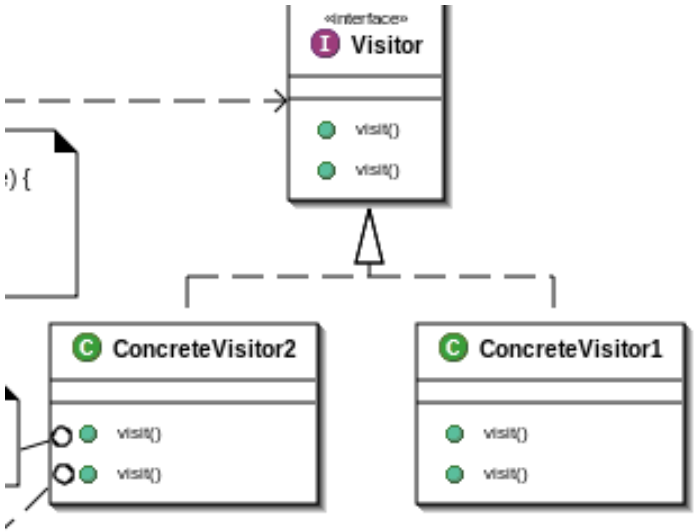


Diagrama general



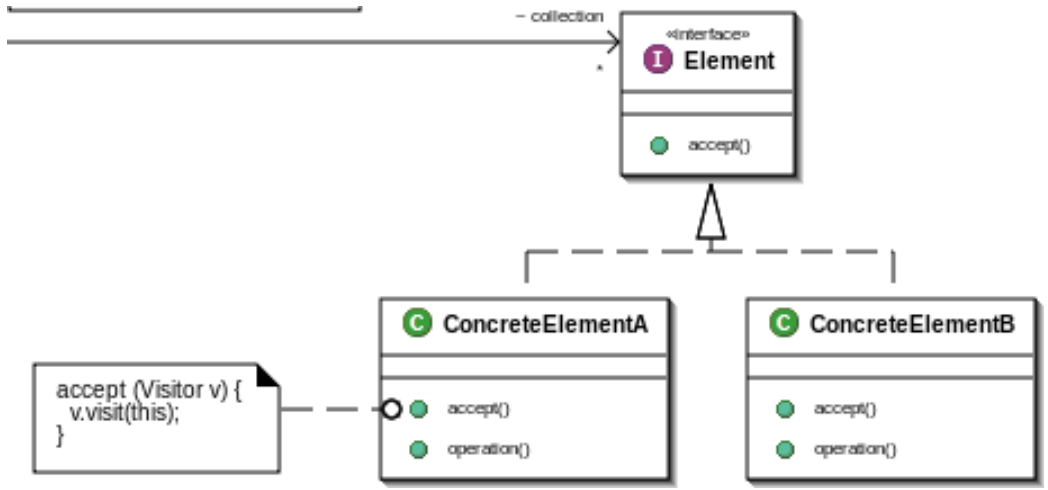
```java
import java.util.List;
public class Main
{
    public static void main(String[] args)
    {
        // Crear los elementos
        Guerrero g1 = new Guerrero();
        Guerrero g2 = new Guerrero();
        Mago m1 = new Mago();
        Mago m2 = new Mago();

        // Crear una lista para guardar los elementos
         List<IPersonaje> personajes = new ArrayList<IPersonaje>();

        // Agregar los elementos a una lista
        personajes.add(g1);
        personajes.add(g2);
        personajes.add(m1);
        personajes.add(m2);

        // Creamos el Visitor y le pasamos la lista
         IVisitor visitorArma = new EquiparArma();
         visitorArma.visit( personajes );

        // Comprobar el resultado:
        System.out.println( "Arma del guerrero g1: " + g1.getArma() );
        System.out.println( "Arma del guerrero g2: " + g2.getArma() );
        System.out.println( "Arma del mago m1: " + m1.getArma() );
        System.out.println( "Arma del mago m1: " + m2.getArma() );
    }
}
```

```java
package Visitor01;
import java.util.List;
 public interface IVisitor
{
    public void visit( Mago m );
    public void visit( Guerrero g );
    public void visit( List<IPersonaje> elementList );
}
```



```java
package Visitor01;
import java.util.List;
public class EquiparArma implements IVisitor
{
    @Override
     public void visit( Mago m )
    {
        m.setArma("DAGA");
    }
    // ------------------------------
    @Override
     public void visit( Guerrero g )
    {
        g.setArma("ESPADA");
    }
    // ------------------------------
    @Override
     public void visit( List<IPersonaje> personajes )
    {
        for( IPersonaje p : personajes )
        {
            p.accept(this);
        }
    }
}
```

```java
package Visitor01;
public interface IPersonaje
{
    public void accept( IVisitor visitor );
}
```



```java
package Visitor01;
public class Guerrero implements IPersonaje
{
    private String arma = "";
    // ------------------------------
    public Guerrero() {
    }
    // ------------------------------
    public String getArma()
    {
        return this.arma;
    }
    // ------------------------------
    public void setArma(String arma)
    {
        this.arma = arma;
    }
    // ------------------------------
    @Override
     public void accept( IVisitor visitor )
    {
         visitor.visit(this);
    }
}
```

```java
package Visitor01;
public class Mago implements IPersonaje
{
    private String arma = "";
    // ------------------------------
    public Mago() {
    }
    // ------------------------------
    public String getArma()
    {
        return this.arma;
    }
    // ------------------------------
    public void setArma(String arma)
    {
        this.arma = arma;
    }
    // ------------------------------
    @Override
     public void accept( IVisitor visitor )
    {
         visitor.visit(this);
    }
}
```

Enlace a Youtube

https://youtu.be/CYMXB0y_HW4