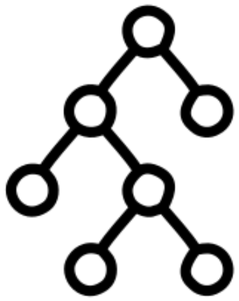
PATRÓN COMPOSITE

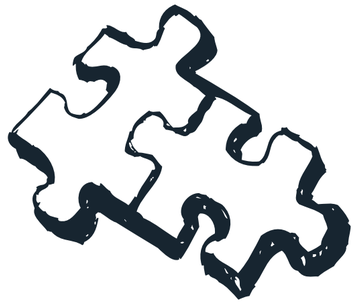
GEANCARLO RIVERA HERNÁNDEZ C06516
JULIO ALEJANDRO RODRÍGUEZ SALGUERA C16717

PROBLEMA

- El patrón composite es útil cuando nuestros elementos tienen una estructura jerárquica entre ellos
- El patrón permite almacenar grupos de elementos complejos y simples en un mismo objeto, lo que se puede representar como una estructura de árbol



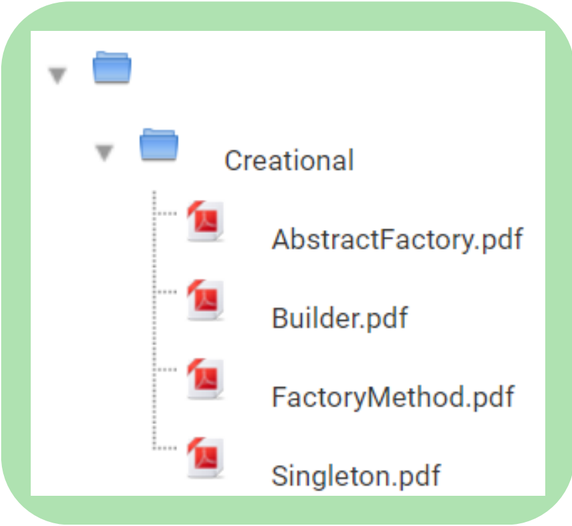
- Captura la esencia de la composición recursiva en términos orientados a objetos
- Permite simplificar el diseño y volver más conciso el código de una clase compleja
- Se puede tratar objetos individuales y compuestos de forma uniforme



EJEMPLO

Secciones y carpetas

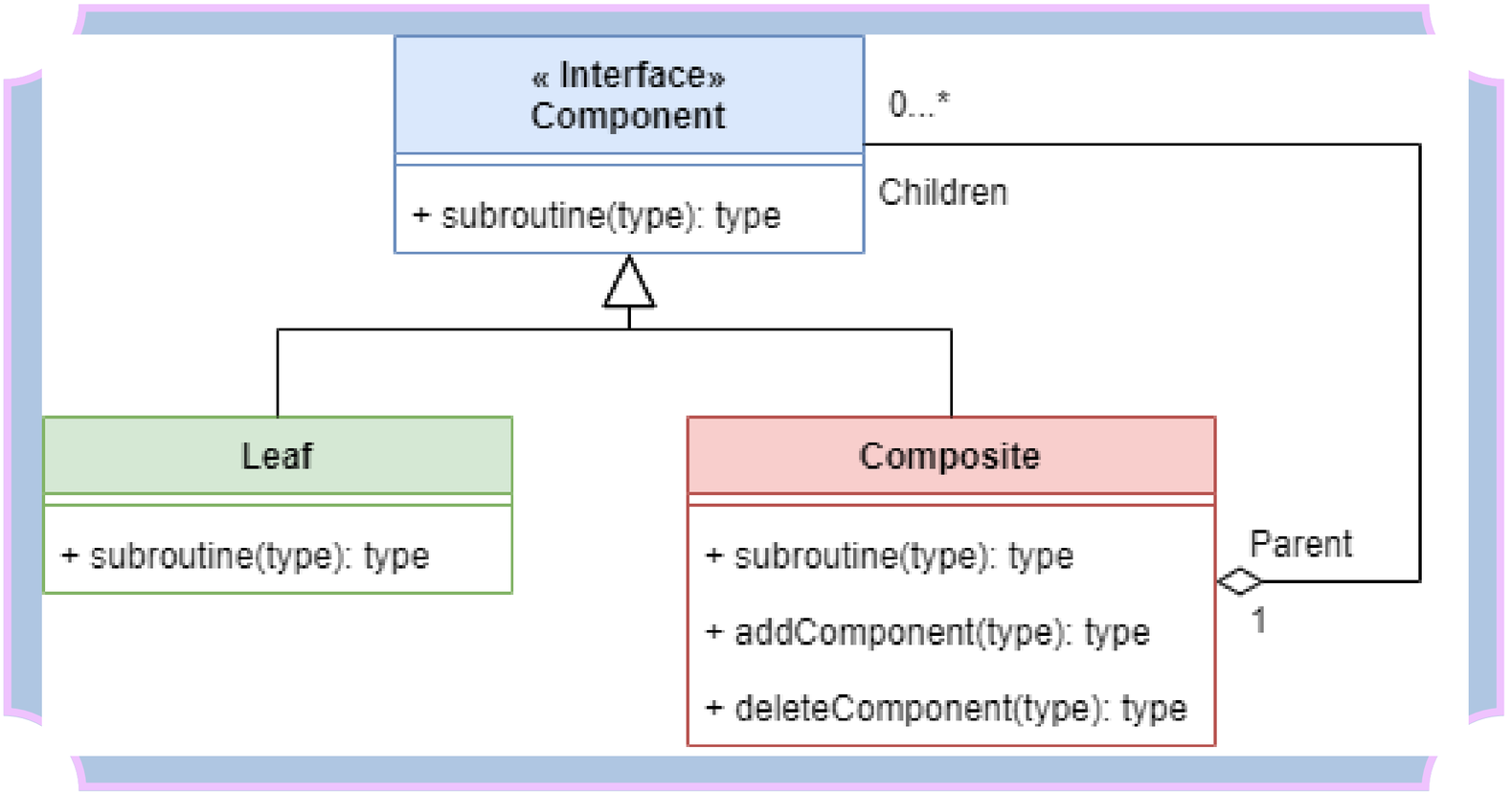
- En la plataforma virtual "Mediación Virtual" se utiliza el patrón en las secciones de los cursos.
- Las secciones separan los recursos de forma física y lógica según temas, fechas o importancia.
- Las secciones pueden contener una cantidad indefinida de recursos, como archivos o evaluaciones.
- Las acciones realizadas sobre la sección pueden ser transmitidas a todos los recursos que esta contiene.
- Las secciones también pueden contener carpetas de recursos.
- Si la sección es eliminada, la carpeta eliminará recursivamente todos sus contenidos.



SOLUCIÓN

Existen 4 componentes principales en este patrón:

- **Component**
- **Composite**
- **Leaf**
- **Client**



SOLUCIÓN

Component

- La clase Component declara la interfaz de la cual heredarán los objetos en la composición (Leaf y Composite).
- Component es responsable de implementar el comportamiento predeterminado para la interfaz común a todas las clases.
- Leaf y Composite heredan de Component y utilizan las interfaces declaradas para dirigirse a un objeto simple o compuesto.
- Component se relaciona con todos los demás componentes ya que todos heredan de él y utilizan las interfaces que Component declara.

SOLUCIÓN

Composite

- La clase Composite define el comportamiento de los componentes que tienen hijos.
- Composite puede almacenar una cantidad indefinida de componentes secundarios que hereden de Component.
- Composite tiene la capacidad de almacenar objetos de su mismo tipo, formando una composición recursiva.
- Composite implementa las operaciones o subrutinas relacionadas a los hijos definidas en la interfaz Component.

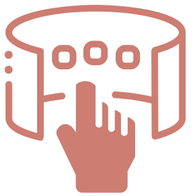
SOLUCIÓN

Leaf

- La clase Leaf representa los objetos hoja en la composición.
- Hereda de Component y puede ser contenido en un Composite.
- Permite que el Client haga un llamado individual a un método de un Leaf.
- Define el comportamiento de objetos primitivos y puede implementar su propia versión de las subrutinas.
- Representa la clase más baja de la jerarquía y no puede contener otras clases que hereden de esta.

Client

- El Client manipula los objetos de la composición a través de la interfaz Component.
- Si hace un llamado a un objeto Leaf, este realizará la acción individualmente.
- Si hace un llamado a un objeto Composite, se hará un llamado recursivo a todos los objetos que heredan de Component que contiene.
- Es el encargado de utilizar la estructura de objetos compuestos para realizar sus acciones y tareas específicas.



EJEMPLO DE CÓDIGO

Component

```java
import java.util.List;
import java.util.ArrayList;
import java.util.Arrays;

public interface Component {
  void showComponent();
  void addComponent(Component component);
  void removeComponent(Component component);
}
```

Leaf

```java
public class Document implements Component {

  public void showComponent() {
    System.out.println("Showing a Document");
  }

  public void addComponent(Component component) {
  }

  public void removeComponent(Component component) {
  }
}
```

EJEMPLO DE CÓDIGO

Composite

```java
import java.util.List;
import java.util.ArrayList;
import java.util.Arrays;

public class Section implements Component {

  private String title;

  private List<Component> childResource;

  public Section(String title) {
    this.title = title;
    this.childResource = new ArrayList<>();
  }

  public void showComponent() {
    System.out.println("Showing the resources of the section " + this.title + "\n~~~");
    childResource.forEach(Component::showComponent);
    System.out.println("~~~");
  }

  public void addComponent(Component component) {
    this.childResource.add(component);
  }

  public void removeComponent(Component component) {
    this.childResource.remove(component);
  }
}
```

Main

```java
class Main {
  public static void main(String[] args) {
    Component folder = new Folder("Design Patterns");

    Component document1 = new Document();
    Component document2 = new Document();

    folder.addComponent(document1);
    folder.addComponent(document2);

    Component section = new Section("Software Design");

    Component evaluation1 = new Evaluation();
    Component evaluation2 = new Evaluation();

    section.addComponent(evaluation1);
    section.addComponent(folder);
    section.addComponent(evaluation2);

    section.showComponent();
  }
}
```

CONSECUENCIAS

Crea **jerarquías** de clases con objetos primitivos y compuestos

Permite tratar estructuras compuestas y objetos individuales de manera **uniforme**

Hace que sea más **fácil** agregar nuevos tipos de componentes

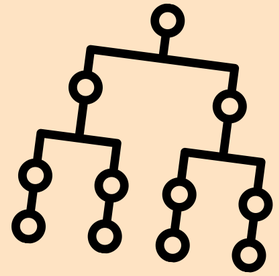
CONSECUENCIAS

Permite la creación de diseños muy generales y flexibles
Sin embargo, una desventaja de esta flexibilidad es que puede resultar difícil restringir la composición a ciertos componentes específicos

Algunas restricciones comúnes:

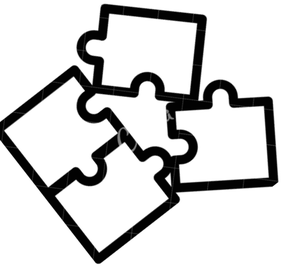
- **Restricción de cantidad de componentes en la composición**
- **Restricciones del tipo de componentes que pueden formar parte de la composición**
- **Restricciones de la estructura de la composición**

IMPLEMENTACIÓN



Consideraciones al implementar este patrón

- Tener referencias explícitas a las clases padre facilita navegación y gestión de la estructura
- Los hijos puedan tener varios padres para compartir componentes de la estructura, pero no se recomienda, pues esto puede causar problemas y ambigüedades en la estructura.
- La clase Component debe definir tantas operaciones comunes como sea posible para las clases Composite y Leaf.
- No es recomendable que la clase Component implemente una lista de componentes ya que implica un costo de espacio adicional para cada hoja, incluso si una hoja no tiene hijos.



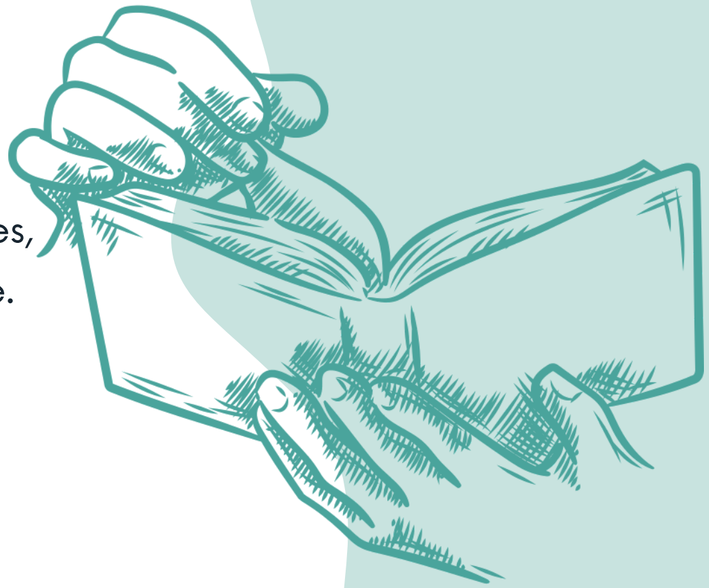
IMPLEMENTACIÓN

- Si el ordenamiento de los hijos es un factor en la implementación, se deben diseñar cuidadosamente las interfaces de acceso y gestión de los hijos para manejar su secuencia.
- Existen varias opciones a elegir la estructura de datos para almacenar componentes en un Composite. Elegir una depende del contexto en el que se trabaje.
- Se podría implementar una estructura tipo caché en la clase Composite para mejorar rendimiento si se necesita navegar o buscar composiciones con frecuencia.
- La clase Composite debería ser la responsable de eliminar a sus hijos cuando ésta se destruya.

RELACIÓN CON OTROS PATRONES

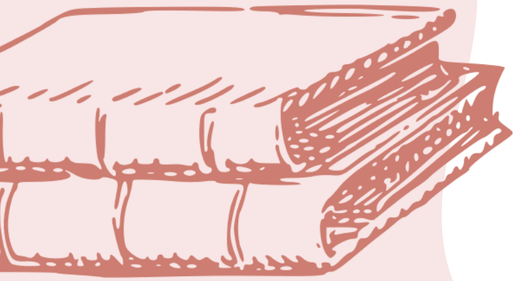
- DECORATOR
- VISITOR
- FLYWEIGHT
- ITERATOR

REFERENCIAS



Haythornwaite, C. (2002). Gamma, E., Helm, R., Johnson, R. & Vlissides, J. Design Patterns: Elements of Reusable Object Oriented Software. New York: Addison-Wesley, 1995. ADDISON-WESLEY.

El patrón Composite: ejemplos de soluciones para jerarquías parte-todo. (2020, 11 septiembre). IONOS Digital Guide. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/patron-composite/



GRACIAS

ACTIVIDAD: CLIENTE DICE

El grupo será dividido en subgrupos con identificador y debe seguir las instrucciones que el cliente indica, de lo contrario pierde

Ostras

Angel Chaves Chinchilla	Ostras 1
Cristopher Hernandez Calderon	Ostras 2
Gabriel González Flores	Ostras 3
Daniel Lizano Morales	Ostras 4
Jorge Loria	Ostras 5

Bambús

Fabio Andrés Sanabria Valerín	Bambús 1
Esteban Iglesias Vargas	Bambús 2
Juan Carlos Aguilar Torres	Bambús 3
Mauricio Delgado Leandro	Bambús 4
Francisco Mora Díaz	Bambús 5

Langostas

Carlos Antonio Sánchez Blanco	Langosta 1
Lizeth Corrales Cortés	Langosta 2
Ignacio Robles	Langosta 3
Camilo Suárez Sandí	Langosta 4
David Cerdas Alvarado	Langosta 5

Palmeras

Michelle Fonseca Carrillo	Palmeras 1
Julián Sedó Álvarez	Palmeras 2
Esteban Castañeda Blanco	Palmeras 3
Javier Pupo Acosta	Palmeras 4
Luis Diego Barrantes Rojas	Palmeras 5

¿LISTOS?