TEC | Tecnológico de Costa Rica

Escuela de Computación, Campus Tecnológico Local de San José
IC6821 Diseño de Software
Práctica Patrones Estructurales
Trabajo, discusión y programación en grupos
Valor 3.5%

Práctica de patrones estructurales - Agustín Brenes, Jesús Chavarría y Eduardo Saborío

Todos los equipos deben aportar la selección y justificación de escogencia del patrón que da solución a todos los casos planteados.

Los equipos PARES completan la práctica específicamente con los casos IMPARES y los equipos IMPARES con los casos PARES, por lo que cada equipo debe completar este documento con 4 ejercicios completos, pero se evaluará también su aporte en la solución de los casos que no deben seguir trabajando. Es decir, en la primera parte deben indicar la selección y justificación de todos los casos, aunque luego sólo se dediquen a trabajar en 4 específicos.

Primera etapa – Discusión en equipos.

Los equipos trabajan de forma asincrónica durante la primera lección para analizar todos los casos, proponer y justificar el patrón que mejor se ajusta para dar solución al problema. En el muro de publicaciones de su equipo privado en Teams, dejan la imagen de esta tabla antes de empezar la segunda lección:

Caso-Contexto	Propuesta	Justificación
1-Validación de un XML	Bridge	Se utiliza el Bridge para "Inyectar" la funcionalidad de escribir archivos en XML de manera independiente a la clase que llama a la función y sin la necesidad que se conozca su implementación.
3-Manejo de tipos de moneda	Adapter	Para este caso se va usar el patrón Adapter, ya que se tienen varios tipos de moneda (colones, dólares) y se quiere agregar uno nuevo (euros), pero la aplicación en si solo va a manejar registros en colones, por lo que el Adapter sería el mejor patrón a usar.
5-Mecanismo de Seguridad	Proxy	Se requiere controlar el acceso al objeto, lo cual calza con el propósito del Proxy. Se postpone la creación de objeto para primero verificar si el

		usuario tiene los privilegios o no.
7-Listas de Reproducción	Flyweight	Se utiliza el Flyweight ya que hay datos compartidos en dos casos. Las listas de reproducción comparten referencias a las canciones y las canciones comparten secciones. Por lo que el flyweight ayuda a mejorar el uso de memoria en ambos casos y no tener mucha información repetida.

En la segunda lección se discute en una plenaria las decisiones de los equipos y se revelan los patrones que dan solución a los casos. Entonces, cada equipo completa el documento para cada caso, esto es para cada caso asignado colocar el nombre del patrón que cree que se ajusta y sus razones que fundamentan su decisión, presenta el modelo básico u original del patrón y su versión para el caso en cuestión determinando las responsabilidades de cada uno de los elementos que le conforman y su participación en la solución del problema. No olviden los métodos mínimos necesarios que exige el patrón y todos aquellos que se requieran para poder implementar la solución propuesta.

Segunda etapa – Programación de casos asignados y entrega del documento.

El equipo programa en un proyecto usando Java o cualquier otro lenguaje de su predilección la implementación de los casos asignados, entregándolos en su carpeta de Archivos a nivel de equipo en una carpeta llamada PracticaPE_EquipoX.

Por último, se programa la solución y se adjuntan screenshots del código y de los resultados de ejecución para completar este documento y se sube junto con el proyecto programado en la carpeta de Teams del equipo.

Esta entrega queda asignada para el **lunes 17 de junio del 2021** a MEDIANOCHE.

En este documento, debe incorporar una sección de las referencias bibliográficas digitales consultadas por el equipo para su resolución.

Plantilla para responder los casos completos:

Problema No. 1
Contexto: Se requiere implementar un mecanismo que analice y valide la construcción correcta de un archivo con formato XML.
Patrón Estructural Propuesto: Composite

TEC | Tecnológico de Costa Rica

Escuela de Computación, Campus Tecnológico Local de San José
IC6821 Diseño de Software
Práctica Patrones Estructurales
Trabajo, discusión y programación en grupos
Valor 3.5%

Justificación de selección

Un archivo XML puede ser visualizado como un Componente hecho de Composites (<headers> que tienen más <headers>) o de Leaves (<headers> únicos).

Diagrama original del patrón:

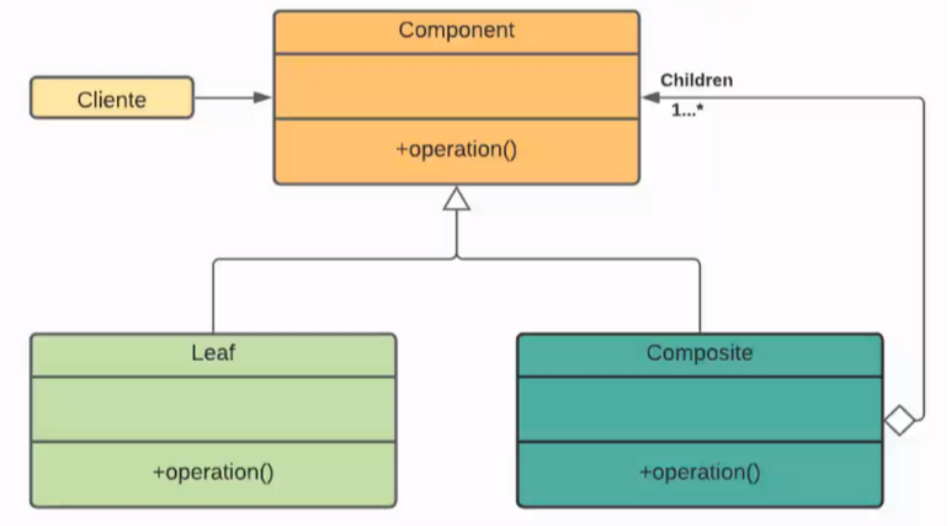
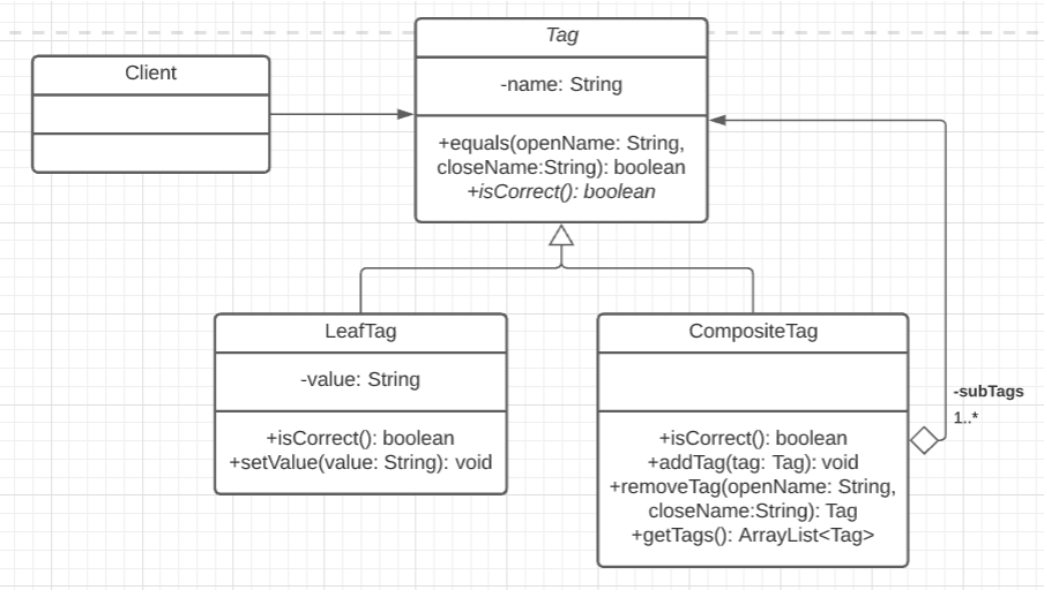


Diagrama del patrón en el contexto del problema:



Rol de cada elemento del patrón en la solución propuesta:

- Client: cliente que llama a la lógica que valida el XML
- Tag: clase abstracta de tipo “Component” en el patrón composite. Esta tiene el nombre del tag y tiene el método abstracto “isCorrect” que será implementado por sus clases hijas.
- LeafTag: clase hija de Tag, de tipo Leaf en el patrón Composite. Esta revisa si esta correcta con el método “isCorrect” implementado.

- CompositeTag: clase hija de Tag, de tipo Composite en el patrón Composite. Esta revisa si esta correcta con el método "isCorrect" implementado al revisar que cada Tag hijo que tenga esté correcto también.

- NOTA: el método "isCorrect" se revisa de manera iterativa desde lo más adentro de la estructura encontrada hasta el tag más externo. Finalmente, si el primer Tag (el que encapsula a los demás) es correcto, la estructura del XML es correcta. La lógica de validación la se pudo desarrollar, pero no fue posible leer bien el XML al final.

Screenshots de la programación del patrón y de la ejecución.

```
public abstract class Tag {
    protected String openName;
    protected String closeName;

    public Tag(String openName, String closeName) {
        //super();
        this.openName = openName;
        this.closeName = closeName;
    }

    public boolean equals(String openName, String closeName) {
        return this.openName.equals(openName) && this.closeName.equals(closeName);
    }

    public abstract boolean isCorrect();

}
```

```
public class LeafTag extends Tag {
    String value = "";

    public LeafTag(String openName, String closeName) {
        super(openName, closeName);
    }

    public void setValue(String value) {
        this.value = value;
    }

    @Override
    public boolean isCorrect() {
        return this.closeName.equals("/" + this.openName);
    }

}
```

TEC | Tecnológico de Costa Rica

Escuela de Computación, Campus Tecnológico Local de San José
IC6821 Diseño de Software
Práctica Patrones Estructurales
Trabajo, discusión y programación en grupos
Valor 3.5%

```java
public class CompositeTag extends Tag {

    // agregar children, varios tags
    ArrayList<Tag> subTags;

    public CompositeTag(String openName, String closeName) {
        super(openName, closeName);
        subTags = new ArrayList<>();
    }

    public void addTag(Tag tag) {
        subTags.add(tag);
    }

    public Tag removeTag(String openName, String closeName) {
        for(int i = 0; i < subTags.size(); i++){
            if(subTags.get(i).equals(openName, closeName)){
                return subTags.remove(i);
            }
        }
        throw new Error("SubTag <" + openName + "><" + closeName + "> does not exist inside <" + this.openName + "><" + this.closeName + ">");
    }

    public ArrayList<Tag> getSubTags() {
        return subTags;
    }

    @Override
    public boolean isCorrect() {
        boolean correctNames = this.closeName.equals("/" + this.openName);
        if (!correctNames) {
            return false;
        }

        for (Tag t: subTags) {
            if (!t.isCorrect()) {
                return false;
            }
        }
        return true;
    }

}
```

Problema No. 3
Contexto: Se tiene una aplicación que maneja colones y dólares como tipos de moneda base en sus transacciones, pero a partir de este mes se va a incorporar el manejo de otras divisas como euros. La aplicación se conceptualizó para manejar sus registros en colones en su totalidad.
Patrón Estructural Propuesto Adapter
Justificación de selección: Se tiene en mente que ya existe la forma de realizar las transacciones en euros y en dólares, por lo que estas son sólamente incorporadas a este sistema que trabaja en colones. Por medio del patrón Adapter, el llamado a la transacción “processColones()” siempre es el mismo y acepta las transacciones en los tipos de moneda agregados.
Diagrama original del patrón:

Escuela de Computación, Campus Tecnológico Local de San José
IC6821 Diseño de Software
Práctica Patrones Estructurales
Trabajo, discusión y programación en grupos
Valor 3.5%

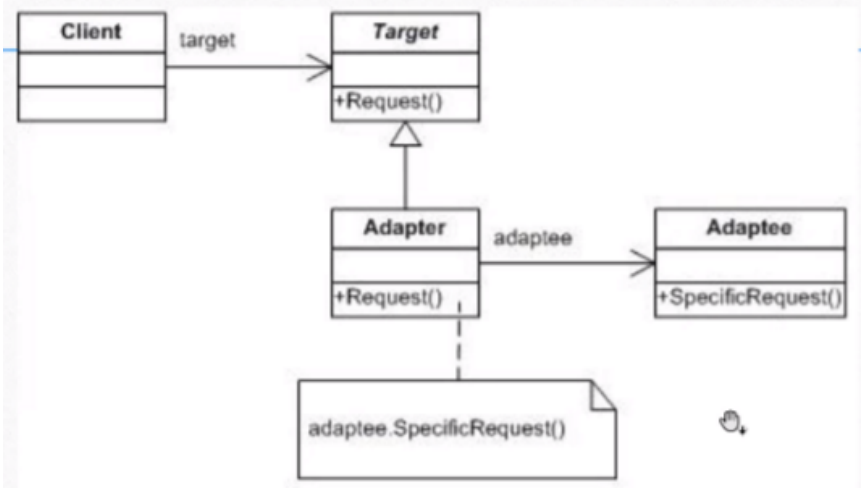
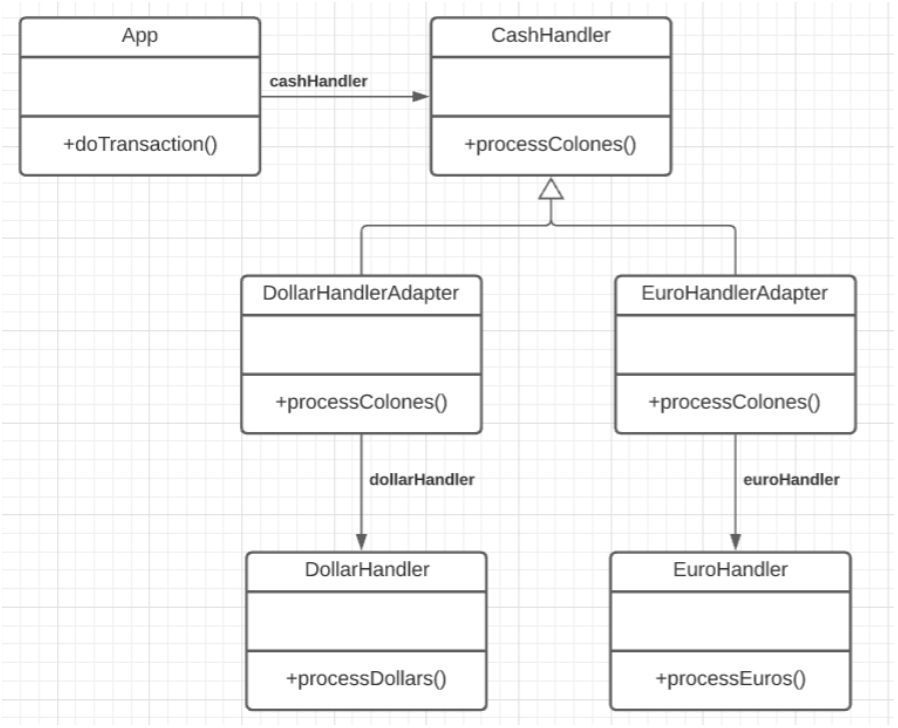


Diagrama del patrón en el contexto del problema:



Rol de cada elemento del patrón en la solución propuesta:

- App: aplicación que llama al procesador de transacciones.
- CashHandler: clase abstracta que define método processColones.
- DollarHandler: clase existente que procesa transacciones en dólares.
- EuroHandler: clase existente que procesa transacciones en euros.
- DollarHandlerAdapter: clase concreta que implementa método processColones llamando al processDollars del DollarHandler.
- EuroHandlerAdapter: clase concreta que implementa método processColones llamando al processEuros del EuroHandler.

TEC | Tecnológico de Costa Rica

Escuela de Computación, Campus Tecnológico Local de San José
IC6821 Diseño de Software
Práctica Patrones Estructurales
Trabajo, discusión y programación en grupos
Valor 3.5%

Screenshots de la programación del patrón y de la ejecución.

```java
public abstract class CashHandler {

    public CashHandler() {}

    public abstract void processColones(double amount);

}
```

```java
public class DollarHandler {

    public DollarHandler() {}

    public void processDollars(double amount) {
        System.out.println(amount + " dólares han sido depositados!");
    }

}
```

```java
public class EuroHandler {

    public EuroHandler() {}

    public void processEuros(double amount) {
        System.out.println(amount + " euros han sido depositados!");
    }

}
```

```java
public class DollarHandlerAdapter extends CashHandler {

    private DollarHandler dollarHandler;
    private final double tipoCambio = 615.43d;

    public DollarHandlerAdapter() {
        super();
        dollarHandler = new DollarHandler();
    }

    @Override
    public void processColones(double amount) {
        dollarHandler.processDollars(amount);
        double colones = amount * tipoCambio;
        System.out.println(colones + " colones han sido depositados!");
    }

}
```

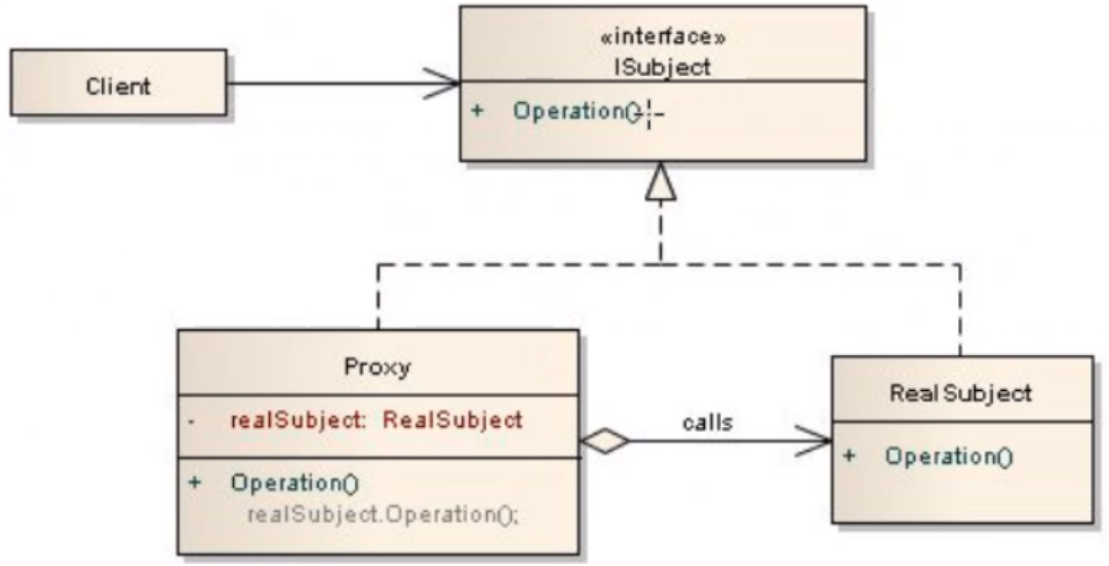
```java
public class EuroHandlerAdapter extends CashHandler {

    private EuroHandler euroHandler;
    private final double tipoCambio = 747.92d;

    public EuroHandlerAdapter(){
        super();
        euroHandler = new EuroHandler();
    }

    @Override
    public void processColones(double amount) {
        euroHandler.processEuros(amount);
        double colones = amount * tipoCambio;
        System.out.println(colones + " colones han sido depositados!");
    }

}
```

```
run:
1. Dolares
2. Euros
Digite el tipo de moneda que desea para la transacción:
1
Introduzca la cantidad que desea depositar:
123.54
123.54 dólares han sido depositados!
76030.2222 colones han sido depositados!
BUILD SUCCESSFUL (total time: 10 seconds)
```

```
run:
1. Dolares
2. Euros
Digite el tipo de moneda que desea para la transacción:
2
Introduzca la cantidad que desea depositar:
450
450.0 euros han sido depositados!
336564.0 colones han sido depositados!
BUILD SUCCESSFUL (total time: 14 seconds)
```

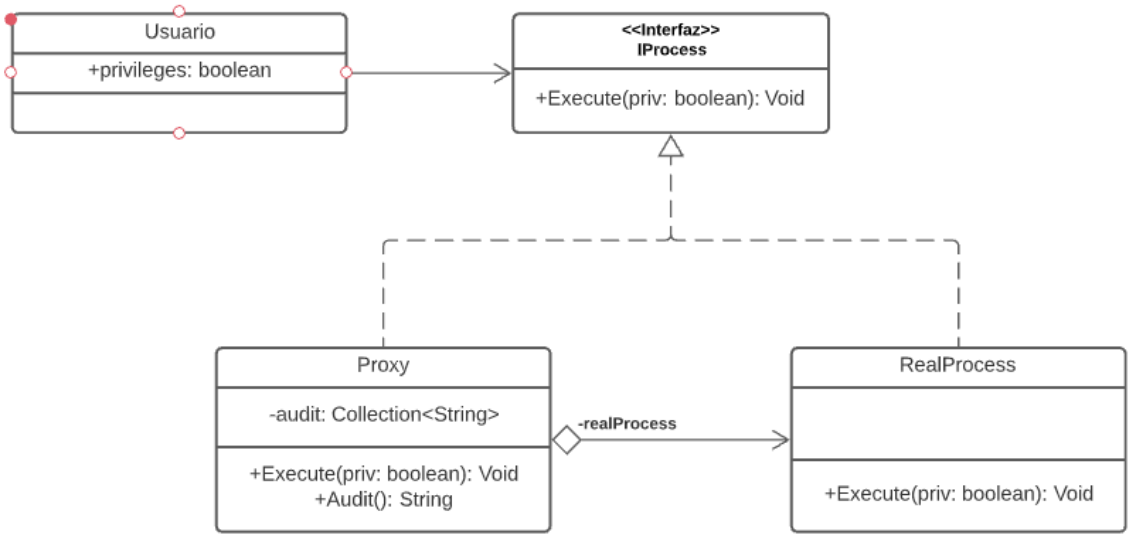
TEC | Tecnológico de Costa Rica

Escuela de Computación, Campus Tecnológico Local de San José
IC6821 Diseño de Software
Práctica Patrones Estructurales
Trabajo, discusión y programación en grupos
Valor 3.5%

Problema No. 5
Contexto: Se requiere un mecanismo de seguridad que intercepte las ejecuciones de ciertos procesos para verificar si el usuario cuenta con los privilegios necesarios evitando que usuarios no autorizados los ejecuten, además, una vez que el proceso es ejecutado, se auditará la ejecución y quedará un registro de la ejecución. Por lo tanto, las acciones de verificación y auditoría deben ser realizadas de forma transparente para el usuario conectado.
Patrón Estructural Propuesto Proxy Pattern
Justificación de selección Se requiere controlar el acceso al objeto, lo cual calza con el propósito del Proxy Pattern. Se pospone la creación del objeto para primero verificar si el usuario tiene los privilegios o no.
Diagrama original del patrón: 
Diagrama del patrón en el contexto del problema:

TEC | Tecnológico de Costa Rica

Escuela de Computación, Campus Tecnológico Local de San José
IC6821 Diseño de Software
Práctica Patrones Estructurales
Trabajo, discusión y programación en grupos
Valor 3.5%



Rol de cada elemento del patrón en la solución propuesta:

- Usuario: El usuario que ejecuta el proxy. Tiene un boolean que representa sus privilegios.
- IProcess: Interfaz que es implementada en el proceso y el proxy.
- Proxy: Objeto que agarra las ejecuciones del Usuario.
    - Execute: Ejecuta el método de RealProcess después de verificar los privilegios del usuario.
    - Audit: Muestra en pantalla las ejecuciones de los procesos.
- RealProcess: Representa el proceso a ejecutar a través del Proxy.

Screenshots de la programación del patrón y de la ejecución.

TEC | Tecnológico de Costa Rica

Escuela de Computación, Campus Tecnológico Local de San José
IC6821 Diseño de Software
Práctica Patrones Estructurales
Trabajo, discusión y programación en grupos
Valor 3.5%

```java
public class Usuario {
    private boolean privileges;
    private IProcess process;

    public Usuario(boolean privileges, IProcess process) {
        this.privileges = privileges;
        this.process = process;
    }

    public void setPrivileges(boolean privileges) {
        this.privileges = privileges;
    }

    public boolean getPrivileges() {
        return this.privileges;
    }

    public void callOp() {
        process.execute(privileges);
    }
}
```

```java
public interface IProcess {
    public void execute(boolean priv);
}
```

```java
public class RealProcess implements IProcess{
    public RealProcess() {}

   @Override
   public void execute(boolean priv) {
       System.out.println("Executing...");
       System.out.println("Executed!");
   }
}
```

Práctica Patrones Estructurales
Trabajo, discusión y programación en grupos
Valor 3.5%

```java
import java.util.ArrayList;

public class Proxy implements IProcess {
    private ArrayList<String> audit;
    private RealProcess realProcess;

    public Proxy(RealProcess realProcess) {
        this.audit = new ArrayList();
        this.realProcess = realProcess;
    }

    @Override
    public void execute(boolean priv){
        if(priv) {
            realProcess.execute(priv);
            audit.add("Operation executed succesfully");
            System.out.println("Permission granted!");
        } else {
            audit.add("Operation denied, user doesn't have the privileges");
            System.out.println("Permission denied!");
        }
    }

    public String audit(){
        String s = "";
        for(int i = 0; i < audit.size(); i++) {
            s += "Entry " + i + ": " + audit.get(i) + "\n";
        }
        return s;
    }
}
```

```java
public class Problema5_Estructurales {
    public static void main(String[] args) {
        RealProcess real = new RealProcess();
        Proxy proxy = new Proxy(real);

        Usuario userPrivileged = new Usuario(true, proxy);
        Usuario userForbidden = new Usuario(false, proxy);

        userPrivileged.callOp();
        userForbidden.callOp();
        userPrivileged.callOp();
        userPrivileged.callOp();
        userForbidden.callOp();

        System.out.println(proxy.audit());
    }
}
```

```
run:
Executing...
Executed!
Permission granted!
Permission denied!
Executing...
Executed!
Permission granted!
Executing...
Executed!
Permission granted!
Permission denied!
Entry 0: Operation executed succesfully
Entry 1: Operation denied, user doesn't have the privileges
Entry 2: Operation executed succesfully
Entry 3: Operation executed succesfully
Entry 4: Operation denied, user doesn't have the privileges
```

Problema No. 7
Contexto: Modelar una propuesta de aplicación que administre listas de reproducción, las cuales están conformados por canciones que son compartidas por todas las listas de reproducción. Además, las canciones tienen secciones compartidas para mejorar la cantidad de memoria utilizada.
Patrón Estructural Propuesto Flyweight
Justificación de selección Como se quiere crear Listas de reproducción donde entre ellas pueden compartir canciones, se plantea usar el patrón Flyweight para reducir redundancia y aumentar la eficiencia.
Diagrama original del patrón:

Escuela de Computación, Campus Tecnológico Local de San José
IC6821 Diseño de Software
Práctica Patrones Estructurales
Trabajo, discusión y programación en grupos
Valor 3.5%

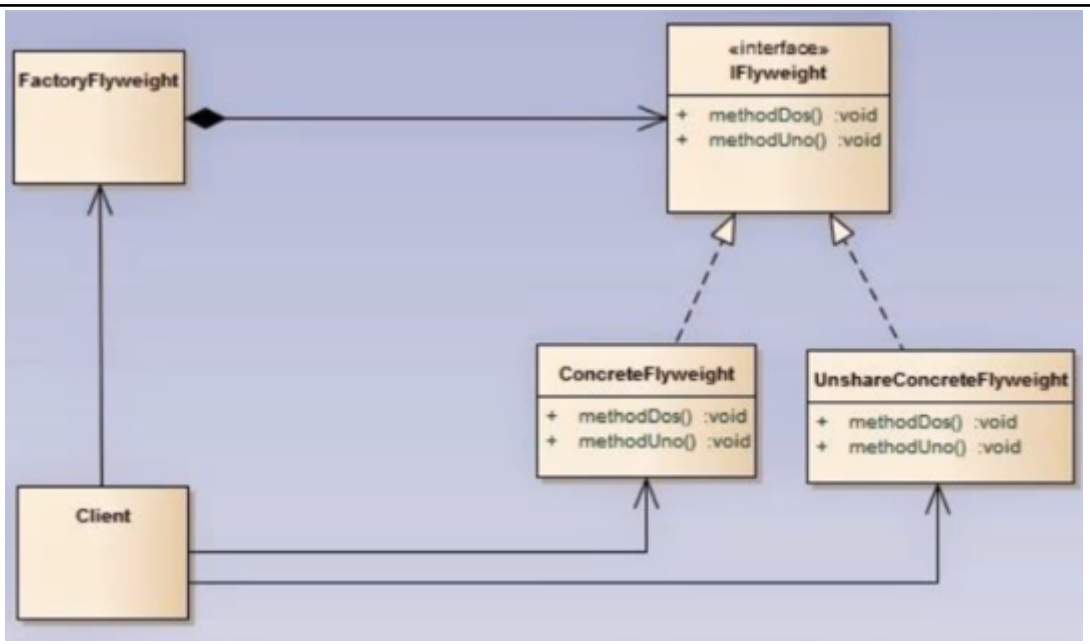
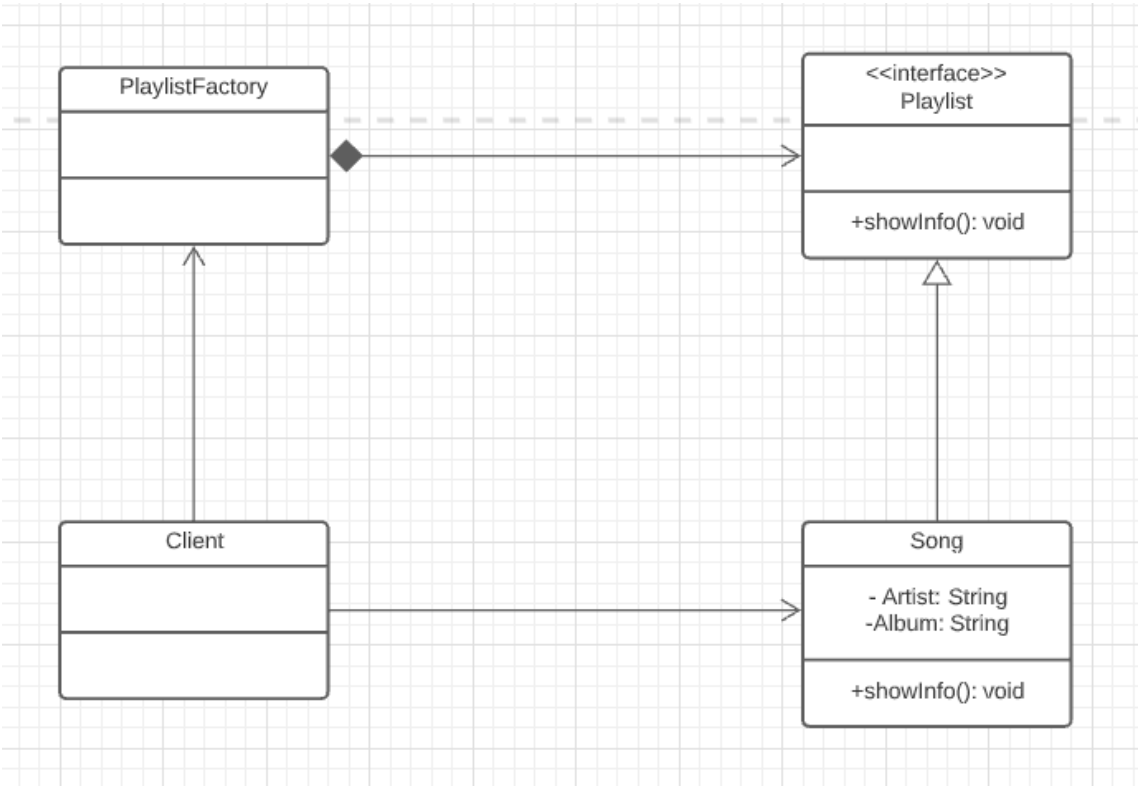


Diagrama del patrón en el contexto del problema:



Rol de cada elemento del patrón en la solución propuesta:

- FlyweightFactory: Crea y maneja objetos flyweight para asegurarse que se compartan adecuadamente

- Playlist: Interfaz que contiene el método para mostrar la información
- Client: Trabaja con una referencia a una Interfaz
- Song: Contiene los datos de las canciones

Screenshots de la programación del patrón y de la ejecución.

```
*/
package problema7;

import java.util.HashMap;
import java.util.Map;

/**
 *
 * @author Chuz2
 */
public class Factory {
    IPlaylist song;

    public void listarCanciones(String name[], String artist[], String album[]){
        for (int i = 0; i < artist.length; i++ ) {
            song = new Song(name[i], artist[i], album[i]);
            System.out.println("Datos de la canción" + song.showInfo());
        }
    }
}
```

```
*/
public interface IPlaylist {
    public String showInfo();
}
```

```
public class Song implements IPlaylist {
    public String name;
    public String artist;
    public String album;

    public Song (String name, String artist, String album){
        this.name = name;
        this.artist = artist;
        this.album = album;
    }

    @Override
    public String showInfo() {
        return "\nNombre: " + this.name + " Artista: " + this.artist + " Albúm: " + this.album;
    }
}
```

```
*/
public class Main {
    public static void main(String[] args){

        String[] name = {"Breathin", "Talk", "24K Magic", "The Diary of Jane"};
        String[] artist = {"Ariana Grande", "Coldplay", "Bruno Mars", "Breaking Benjamin"};
        String[] album = {"Sweetener", "X&Y", "24K Magic", "Phobia"};

        Factory factory = new Factory();
        factory.listarCanciones(name, artist, album);
    }


}
```

```
run:
Datos de la canción
Nombre: Breathin Artista: Ariana Grande Albúm: Sweetener
Datos de la canción
Nombre: Talk Artista: Coldplay Albúm: X&Y
Datos de la canción
Nombre: 24K Magic Artista: Bruno Mars Albúm: 24K Magic
Datos de la canción
Nombre: The Diary of Jane Artista: Breaking Benjamin Albúm: Phobia
BUILD SUCCESSFUL (total time: 2 seconds)
```

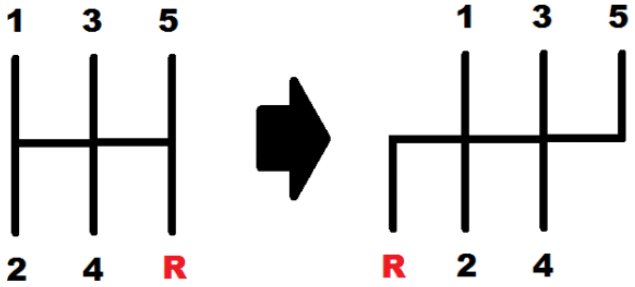
Ejercicio 1

Se requiere implementar un mecanismo que analice y valide la construcción correcta de un archivo con formato XML.

Ejercicio 2

Suponga que un vehículo tiene un sistema transmisión de cambios manuales, por lo que el conductor debe interactuar con el embrague, el acelerador y la palanca de cambios.

Suponga que este vehículo sufre una avería y tienen que cambiar la caja de cambios, haciendo que la marcha atrás, en lugar de estar en la posición inferior derecha, se encuentre en la posición inferior izquierda.



El problema de que cambie este elemento debería implicar que todos los conductores que utilicen el vehículo deberán cambiar aprender la nueva interfaz, pero en realidad,

el mecanismo para ser conducido sigue siendo el mismo. ¿Cómo podría, a través de un patrón estructural, lograr que los conductores no se preocupen si deben manipular el vehículo de manera diferente sólo porque la palanca de cambio está organizada de manera distinta?

Ejercicio 3

Se tiene una aplicación que maneja colones y dólares como tipos de moneda base en sus transacciones, pero a partir de este mes se va a incorporar el manejo de otras divisas como euros. La aplicación se conceptualizó para manejar sus registros en colones en su totalidad.

Ejercicio 4

Suponga un personaje de un videojuego que porta un arma que usa para eliminar a sus enemigos. Dicha arma, por ser de un tipo determinado, tiene una serie de propiedades como el radio de acción, nivel de ruido, número de balas que puede almacenar. Sin embargo, es posible que el personaje pueda incorporar elementos al arma que puedan cambiar estas propiedades como un silenciador o un cargador extra.

Ejercicio 5

Se requiere un mecanismo de seguridad que intercepte las ejecuciones de ciertos procesos para verificar si el usuario cuenta con los privilegios necesarias evitando que usuarios no autorizados los ejecuten, además, una vez que el proceso es ejecutado, se auditara la ejecución y quedará un registro de la ejecución. Por lo tanto, las acciones de verificación y auditoría deben ser realizadas de forma transparente para el usuario conectado.

Ejercicio 6

Se requiere utilizar un programa en el cual se cargarán varios autos. Al pedirle el detalle del mismo, muestra placa, color, dueño. Hay que tener en cuenta que debe ocupar poca memoria ya que es un proyecto masivo.

Ejercicio 7

Modelar una propuesta de aplicación que administre listas de reproducción, las cuales está conformados por canciones que son compartidas por todas las listas de reproducción. Además, las canciones tienen secciones compartidas para mejorar la cantidad de memoria utilizada.

Ejercicio 8

Modelar una aplicación que nos permite procesar un mensaje en capas, donde cada capa se encargará de procesar un mensaje a diferente nivel. Primero se convierte un Objeto en XML, seguido, se empaqueta en un mensaje SOAP para después encriptarlo, finalmente obtendremos un mensaje SOAP totalmente encriptado, el cual podrá ser enviado de forma segura a un destinatario. La idea sería que el orden de procesamiento de cada capa podrá cambiar de posición para obtener un resultado

diferente, de la misma manera, podrá ser agregados nuevos mecanismos en medio en un solo paso.