
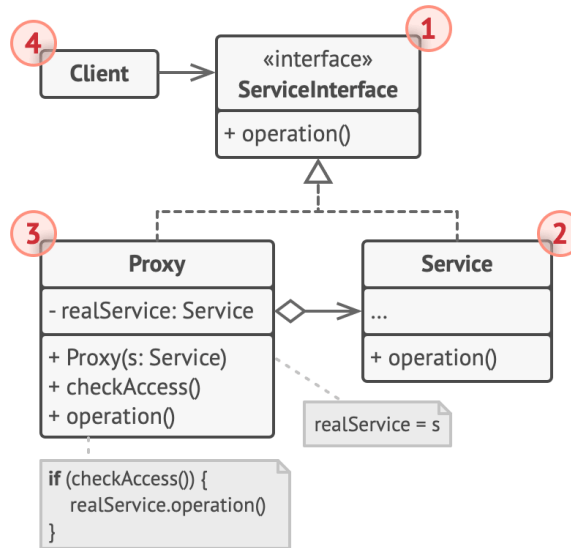
	<b>VICERRECTORADO DOCENTE</b>	<b>Código:</b> GUIA-PRL-001
	CONSEJO ACADÉMICO	<b>Aprobación:</b> 2016/04/06
<b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

		<b>PRÁCTICA DE LABORATORIO</b>	
<b>CARRERA:</b> COMPUTACIÓN		<b>ASIGNATURA:</b> Programación Aplicada	
<b>NRO. PRÁCTICA:</b>	4	<b>TÍTULO PRÁCTICA:</b> Patrones en Java	
<b>OBJETIVO ALCANZADO:</b> <p>En esta practica se identifico los cambios importantes de Java</p> <p>Diseñar e Implementar las nuevas técnicas de programación</p> <p>Entender los patrones de Java</p>			
<b>ACTIVIDADES DESARROLLADAS</b>			
<b>1. Revisar la teoría y conceptos de Patrones de Diseño de Java</b> <p>Los Patrones de diseño son herramientas para solucionar problemas de Diseño enfocados al desarrollo de software, estos patrones deben ser reusables permitiendo así que sean adaptados a diferentes problemáticas.</p> <p>Cuando hablamos de problemáticas nos referimos a condiciones o problemas reiterativos en el desarrollo de software donde la solución se encuentra identificada mediante la aplicación de una serie de pasos, adaptando el sistema a una estructura definida por un patrón, garantizando que esta solución pueda ser aplicada cuantas veces sea necesario en circunstancias similares, evitando así la búsqueda de otras soluciones cuando estas ya se han dado anteriormente</p>			
<b>2. Diseñar e implementar cada estudiante un patrón de diseño y verificar su funcionamiento. A continuación, se detalla el patrón a implementar:</b> <p style="text-align: center;">Patrón de diseño Proxy:</p> <ul style="list-style-type: none"> <li>Concepto:  <p>Proxy es un patrón de diseño estructural cuyo propósito es proporcionar un subrogado o intermediario de un objeto para controlar su acceso.</p> <p>Un proxy controla el acceso al objeto original, permitiéndonos hacer algo antes o después de que la solicitud llegue al objeto original.</p> </li> <li>Estructura</li> </ul>			



- La Interfaz de Servicio declara la interfaz del Servicio. El proxy debe seguir esta interfaz para poder camuflarse como objeto de servicio.
- Servicio es una clase que proporciona una lógica de negocio útil
- La clase Proxy tiene un campo de referencia que apunta a un objeto de servicio. Cuando el proxy finaliza su procesamiento, pasa la solicitud al objeto de servicio.
- El Cliente debe funcionar con servicios y proxis a través de la misma interfaz. De este modo puedes pasar un proxy a cualquier código que espere un objeto de servicio.
- Aplicaciones del patrón proxy
- Inicialización diferida (proxy virtual). Es cuando tienes un objeto de servicio muy pesado que utiliza muchos recursos del sistema al estar siempre funcionando, aunque solo lo necesites de vez en cuando.

*En lugar de crear el objeto cuando se lanza la aplicación, puedes retrasar la inicialización del objeto a un momento en que sea realmente necesario.*

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

- Control de acceso (proxy de protección). Es cuando quieres que únicamente clientes específicos sean capaces de utilizar el objeto de servicio, por ejemplo, cuando tus objetos son partes fundamentales de un sistema operativo y los clientes son varias aplicaciones lanzadas (incluyendo maliciosas).

*El proxy puede pasar la solicitud al objeto de servicio tan sólo si las credenciales del cliente cumplen ciertos criterios.*

- Ejecución local de un servicio remoto (proxy remoto). Es cuando el objeto de servicio se ubica en un servidor remoto.

*En este caso, el proxy pasa la solicitud del cliente por la red, gestionando todos los detalles desagradables de trabajar con la red.*

- Solicitudes de registro (proxy de registro). Es cuando quieres mantener un historial de solicitudes al objeto de servicio.

*El proxy puede registrar cada solicitud antes de pasarla al servicio.*

- Resultados de solicitudes en caché (proxy de caché). Es cuando necesitas guardar en caché resultados de solicitudes de clientes y gestionar el ciclo de vida de ese caché, especialmente si los resultados son muchos.

*El proxy puede implementar el caché para solicitudes recurrentes que siempre dan los mismos resultados. El proxy puede utilizar los parámetros de las solicitudes como claves de caché.*

- Implementación:

1. Si no hay una interfaz de servicio preexistente, crea una para que los objetos de proxy y de servicio sean intercambiables. No siempre resulta posible extraer la interfaz de la clase servicio, porque tienes que cambiar todos los clientes del servicio para utilizar esa interfaz. El plan B consiste en convertir el proxy en una subclase de la clase servicio, de forma que herede la interfaz del servicio.
2. Crea la clase proxy. Debe tener un campo para almacenar una referencia al servicio. Normalmente los proxis crean y gestionan el ciclo de vida completo de sus servicios. En raras ocasiones, el cliente pasa un servicio al proxy a través de un constructor.
3. Implementa los métodos del proxy según sus propósitos. En la mayoría de los casos, después de hacer cierta labor, el proxy debería delegar el trabajo a un objeto de servicio.

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

4. Considera introducir un método de creación que decida si el cliente obtiene un proxy o un servicio real. Puede tratarse de un simple método estático en la clase proxy o de todo un método de fábrica.
5. Considera implementar la inicialización diferida para el objeto de servicio.

3. Probar y modificar el patrón de diseño a fin de generar cuales son las ventajas y desventajas.

Ventajas:

- Puedes controlar el objeto de servicio sin que los clientes lo sepan.
- Puedes gestionar el ciclo de vida del objeto de servicio cuando a los clientes no les importa.
- El proxy funciona incluso si el objeto de servicio no está listo o no está disponible.
- *Principio de abierto/cerrado*. Puedes introducir nuevos proxys sin cambiar el servicio o los clientes.

Desventajas:

- El código puede complicarse ya que debes introducir gran cantidad de clases nuevas.
- La respuesta del servicio puede retrasarse.

4. Realizar práctica codificando los códigos de los patrones y su estructura.

- Se creo una clase abstracta con los siguientes métodos:

```

public abstract class AbstractBanco {

    private double saldo;

    public double getSaldo() {
        return saldo;
    }

    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }

    public abstract void mostrarDineroActual();
    public abstract void depositarDinero(double deposito);
    public abstract void retirarDinero(double retiro);
}

```

- Se creo la clase BancoPichincha la cual hereda de la clase abstracta

```
/*  
public class BancoPichincha extends AbstractBanco{  
  
    @Override  
    public void mostrarDineroActual() {  
        System.out.println("-----Banco Pichincha-----\n");  
        System.out.println("Tu saldo Actual es: " + getSaldo());  
        System.out.println("-----");  
    }  
  
    @Override  
    public void depositarDinero(double deposito) {  
        System.out.println("-----Banco Pichincha-----\n");  
        System.out.println("El valor de tu deposito es: " + deposito);  
        setSaldo(getSaldo() + deposito);  
        System.out.println("Tu saldo Actual es: " + getSaldo());  
        System.out.println("-----");  
    }  
  
    @Override  
    public void retirarDinero(double retiro) {  
        System.out.println("-----Banco Pichincha-----\n");  
        System.out.println("El valor de tu retiro es: " + retiro);  
        setSaldo(getSaldo() - retiro);  
        System.out.println("Tu saldo Actual es: " + getSaldo());  
        System.out.println("-----");  
    }  
}
```

- Se creo la clase BancoProxy la cual será una clase intermediaria entre el usuario y la clase BancoPichincha cumpliendo así con la estructura del patrón proxy

```
public class BancoProxy extends AbstractBanco {

    Scanner leer = new Scanner(System.in);

    AbstractBanco banco;

    public BancoProxy() {
        this.banco = new BancoPichincha();
    }

    public boolean validar() {
        System.out.print("Ingrese la contraseña: -> ");
        var contraseña = leer.next();
        return contraseña.equals("12345");
    }

    @Override
    public void mostrarDineroActual() {
        if (validar()) {
            banco.mostrarDineroActual();
        } else {
            System.out.println("Contraseña incorrecta");
        }
    }

    @Override
    public void depositarDinero(double deposito) {
        banco.depositarDinero(deposito);
    }

    @Override
    public void retirarDinero(double retiro) {
        if (banco.getSaldo() >= retiro) {
            banco.retirarDinero(retiro);
        } else {
            System.out.println("-----SALDO INSUFICIENTE-----");
        }
    }
}
```

- Finalmente se probaron los resultados

```
run:
Ingrese la contraseña: -> 12345
-----Banco Pichincha-----

Tu saldo Actual es: 0.0
-----Banco Pichincha-----

El valor de tu deposito es: 500.26
Tu saldo Actual es: 500.26
-----Banco Pichincha-----

El valor de tu retiro es: 463.12
Tu saldo Actual es: 37.139999999999986
-----Banco Pichincha-----

BUILD SUCCESSFUL (total time: 4 seconds)
```

```
1  /*
2  * To change this license header, choose License Hea
3  * To change this template file, choose Tools | Temp
4  * and open the template in the editor.
5  */
6  package ec.edu.ups.vista;
7
8  import ec.edu.patronProxy.BancoProxy;
9
10 /**
11  *
12  * @author ariel
13  */
14 public class Main {
15
16     public static void main(String[] args) {
17
18         BancoProxy bancoProxy = new BancoProxy();
19         bancoProxy.mostrarDineroActual();
20         bancoProxy.depositarDinero(500.26);
21         bancoProxy.retirarDinero(463.12);
22
23     }
24
25 }
```

- Repositorio de la practica:

<https://github.com/VazquezAriel/Ejemplo-PatronProxy/tree/master>

## RESULTADO(S) OBTENIDO(S):

Realizar procesos de investigación sobre los patrones de diseño de Java

Entender los patrones y su utilización dentro de aplicaciones Java.

Entender las funcionalidades basadas en patrones.

## CONCLUSIONES:

En esta práctica se continuó probando las nuevas funcionalidades de Java se pudo conocer los diversos patrones de diseño que podemos implementar en nuestros proyectos

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

### RECOMENDACIONES:

Realizar el trabajo dentro del tiempo establecido.

**Nombre de estudiante:** Ariel Vazquez

**Firma de estudiante:**

