DESING PATTERNS

ITERATOR: Este patrón de diseño permite recorrer estructuras de datos por medio de recorrido secuencial.

Implementacion propia: En este ejemplo se crea un inventario de una tienda de computación donde cada detalle (que contiene producto y cantidad) se añade en una estructura, en este caso para facilitar la implementación se usó ArrayList como estructura. Para recorrer la estructura usamos el patrón de diseño Iterator.

InventarioDetalle define la plantilla para crear un detalle del inventario. InventarioCollection crea la colección de inventario e implementa el método createIterator perteneciente la clase Iterador importada que se implementa sobre la interfaz Collection. InventarioIterator presenta la forma de realizar el recorrido de la estructura usando iterador e imprimir los datos.

```
package iteratorExample;

/**
 *
 * @author Anderson Nuñez
 */
import java.util.ArrayList;
import java.util.Iterator;

class InventarioDetalle {

    private String detalle;
    private int cantidad;

    public InventarioDetalle (String detalle, int cantidad) {
        this.detalle = detalle;
        this.cantidad = cantidad;
    }

    public String getDetalle() {
        return detalle;
    }

    public void setDetalle(String detalle) {
        this.detalle = detalle;
    }

    public int getCantidad() {
        return cantidad;
    }

    public void setCantidad(int cantidad) {
        this.cantidad = cantidad;
    }
    public String getInventarioDetalle(){
        return "| " + getCantidad()+" - "+getDetalle() + " |";
    }
}

interface Collection {
    public Iterator createIterator();
}
/*
interface Iterator {
    boolean hasNext();
    Object next();
}*/

class InventarioCollection implements Collection{
    private ArrayList<InventarioDetalle> inventario;

    public InventarioCollection() {
        this.inventario = new ArrayList<InventarioDetalle>();
        addItem(2, "Nvidea Rtx");
        addItem(4, "Memoria Ram DDR4 8GB 2400 MHz");
        addItem(5, "Memoria Ram DDR4 16GB 2666 MHz");
        addItem(3, "MSI monitor 1440p 144Hz");
        addItem(6, "AMD RYZEN 5 3500 u");
    }
```

```
    public void addItem(int cantidad, String detalle){
        InventarioDetalle inventarioDetalle =
                new InventarioDetalle(detalle, cantidad);
        this.inventario.add(inventarioDetalle);
    }

    @Override
    public Iterator createIterator() {
        return this.inventario.iterator();
    }

}

class InventarioIterator{
    ArrayList<InventarioDetalle> inventario;
    Collection iteratorInventario;

    public InventarioIterator(Collection iteratorInventario) {
        this.iteratorInventario = iteratorInventario;
    }

    public void printInventario(){
        Iterator iterator = this.iteratorInventario.createIterator();
        System.out.println("---------------Presentacion de Inventario--------------");
        while(iterator.hasNext()){
            InventarioDetalle inventarioDetalle = (InventarioDetalle) iterator.next();
            System.out.println(inventarioDetalle.getInventarioDetalle());
        }
    }

}


public class InventarioIteratorExample {
    public static void main(String[] args) {
        InventarioCollection inventario = new InventarioCollection();
        InventarioIterator inventarioIterator = new InventarioIterator(inventario);
        inventarioIterator.printInventario();
    }
}
```

El resultado de ejecución es el siguiente:

```
 ------------------Presentacion de Inventario------------------
 | 2 - Nvidea Rtx |
 | 4 - Memoria Ram DDR4 8GB 2400 MHz |
 | 5 - Memoria Ram DDR4 16GB 2666 MHz |
 | 3 - MSI monitor 1440p 144Hz |
 | 6 - AMD RYZEN 5 3500 u |
 BUILD SUCCESSFUL (total time: 0 seconds)
```

Ejemplo tomado de refactoring.guru

En este ejemplo se itera sobre perfiles de redes sociales, creados previamente, profile iterator define la interfaz sobre la cual se nombran los metodos principales de este patron de diseño: hasNext(), getNext() y reset(), estos métodos se definen en la clase FacebookIterator y LinkedInIterator, estas dos clases definen la forma de realizar el recorrido de la estructura de perfiles. SocialNetworks define la interfaz en común de las dos redes sociales, nombrando métodos que son definidos en las clases Facebook y LinkedIn, métodos usados para la creación de amigos y colaboradores. Profiles crea los perfiles de redes sociales. Spammer se usa para enviar mensajes a todos los amigos o colaboradores. Finalmente se implementan todas las clases explicadas previamente sobre Demo.

Clase ProfileIterator

```
package refactoring_guru.iterator.example.iterators;

import refactoring_guru.iterator.example.profile.Profile;

public interface ProfileIterator {
    boolean hasNext();

    Profile getNext();
```

```
    void reset();
}
```

Clase FacebookIterator

```
package refactoring_guru.iterator.example.iterators;

import refactoring_guru.iterator.example.profile.Profile;
import refactoring_guru.iterator.example.social_networks.Facebook;

import java.util.ArrayList;
import java.util.List;

public class FacebookIterator implements ProfileIterator {
    private Facebook facebook;
    private String type;
    private String email;
    private int currentPosition = 0;
    private List<String> emails = new ArrayList<>();
    private List<Profile> profiles = new ArrayList<>();

    public FacebookIterator(Facebook facebook, String type, String email) {
        this.facebook = facebook;
        this.type = type;
        this.email = email;
    }

    private void lazyLoad() {
        if (emails.size() == 0) {
            List<String> profiles =
facebook.requestProfileFriendsFromFacebook(this.email, this.type);
            for (String profile : profiles) {
                this.emails.add(profile);
                this.profiles.add(null);
            }
        }
    }

    @Override
    public boolean hasNext() {
        lazyLoad();
        return currentPosition < emails.size();
    }

    @Override
    public Profile getNext() {
        if (!hasNext()) {
            return null;
        }

        String friendEmail = emails.get(currentPosition);
        Profile friendProfile = profiles.get(currentPosition);
        if (friendProfile == null) {
            friendProfile = facebook.requestProfileFromFacebook(friendEmail);
            profiles.set(currentPosition, friendProfile);
        }
        currentPosition++;
        return friendProfile;
    }

    @Override
    public void reset() {
        currentPosition = 0;
    }
}
```

Clase LinkedInIterator

```
package refactoring_guru.iterator.example.iterators;
```

```java
import refactoring_guru.iterator.example.profile.Profile;
import refactoring_guru.iterator.example.social_networks.LinkedIn;

import java.util.ArrayList;
import java.util.List;

public class LinkedInIterator implements ProfileIterator {
    private LinkedIn linkedIn;
    private String type;
    private String email;
    private int currentPosition = 0;
    private List<String> emails = new ArrayList<>();
    private List<Profile> contacts = new ArrayList<>();

    public LinkedInIterator(LinkedIn linkedIn, String type, String email) {
        this.linkedIn = linkedIn;
        this.type = type;
        this.email = email;
    }

    private void lazyLoad() {
        if (emails.size() == 0) {
            List<String> profiles =
linkedIn.requestRelatedContactsFromLinkedInAPI(this.email, this.type);
            for (String profile : profiles) {
                this.emails.add(profile);
                this.contacts.add(null);
            }
        }
    }

    @Override
    public boolean hasNext() {
        lazyLoad();
        return currentPosition < emails.size();
    }

    @Override
    public Profile getNext() {
        if (!hasNext()) {
            return null;
        }

        String friendEmail = emails.get(currentPosition);
        Profile friendContact = contacts.get(currentPosition);
        if (friendContact == null) {
            friendContact = linkedIn.requestContactInfoFromLinkedInAPI(friendEmail);
            contacts.set(currentPosition, friendContact);
        }
        currentPosition++;
        return friendContact;
    }

    @Override
    public void reset() {
        currentPosition = 0;
    }
}
```

Clase SocialNetworks

```java
package refactoring_guru.iterator.example.social_networks;

import refactoring_guru.iterator.example.iterators.ProfileIterator;

public interface SocialNetwork {
    ProfileIterator createFriendsIterator(String profileEmail);

    ProfileIterator createCoworkersIterator(String profileEmail);
}
```

Clase Facebook

```
package refactoring_guru.iterator.example.social_networks;

import refactoring_guru.iterator.example.iterators.FacebookIterator;
import refactoring_guru.iterator.example.iterators.ProfileIterator;
import refactoring_guru.iterator.example.profile.Profile;

import java.util.ArrayList;
import java.util.List;

public class Facebook implements SocialNetwork {
    private List<Profile> profiles;

    public Facebook(List<Profile> cache) {
        if (cache != null) {
            this.profiles = cache;
        } else {
            this.profiles = new ArrayList<>();
        }
    }

    public Profile requestProfileFromFacebook(String profileEmail) {
        // Here would be a POST request to one of the Facebook API endpoints.
        // Instead, we emulates long network connection, which you would expect
        // in the real life...
        simulateNetworkLatency();
        System.out.println("Facebook: Loading profile '" + profileEmail + "' over the
network...");

        // ...and return test data.
        return findProfile(profileEmail);
    }

    public List<String> requestProfileFriendsFromFacebook(String profileEmail, String
contactType) {
        // Here would be a POST request to one of the Facebook API endpoints.
        // Instead, we emulates long network connection, which you would expect
        // in the real life...
        simulateNetworkLatency();
        System.out.println("Facebook: Loading '" + contactType + "' list of '" +
profileEmail + "' over the network...");

        // ...and return test data.
        Profile profile = findProfile(profileEmail);
        if (profile != null) {
            return profile.getContacts(contactType);
        }
        return null;
    }

    private Profile findProfile(String profileEmail) {
        for (Profile profile : profiles) {
            if (profile.getEmail().equals(profileEmail)) {
                return profile;
            }
        }
        return null;
    }

    private void simulateNetworkLatency() {
        try {
            Thread.sleep(2500);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }

    @Override
    public ProfileIterator createFriendsIterator(String profileEmail) {
        return new FacebookIterator(this, "friends", profileEmail);
    }

    @Override
```

```
    public ProfileIterator createCoworkersIterator(String profileEmail) {
        return new FacebookIterator(this, "coworkers", profileEmail);
    }

}
```

Clase LinkedIn

```
package refactoring_guru.iterator.example.social_networks;

import refactoring_guru.iterator.example.iterators.LinkedInIterator;
import refactoring_guru.iterator.example.iterators.ProfileIterator;
import refactoring_guru.iterator.example.profile.Profile;

import java.util.ArrayList;
import java.util.List;

public class LinkedIn implements SocialNetwork {
    private List<Profile> contacts;

    public LinkedIn(List<Profile> cache) {
        if (cache != null) {
            this.contacts = cache;
        } else {
            this.contacts = new ArrayList<>();
        }
    }

    public Profile requestContactInfoFromLinkedInAPI(String profileEmail) {
        // Here would be a POST request to one of the LinkedIn API endpoints.
        // Instead, we emulates long network connection, which you would expect
        // in the real life...
        simulateNetworkLatency();
        System.out.println("LinkedIn: Loading profile '" + profileEmail + "' over the
network...");

        // ...and return test data.
        return findContact(profileEmail);
    }

    public List<String> requestRelatedContactsFromLinkedInAPI(String profileEmail, String
contactType) {
        // Here would be a POST request to one of the LinkedIn API endpoints.
        // Instead, we emulates long network connection, which you would expect
        // in the real life.
        simulateNetworkLatency();
        System.out.println("LinkedIn: Loading '" + contactType + "' list of '" +
profileEmail + "' over the network...");

        // ...and return test data.
        Profile profile = findContact(profileEmail);
        if (profile != null) {
            return profile.getContacts(contactType);
        }
        return null;
    }

    private Profile findContact(String profileEmail) {
        for (Profile profile : contacts) {
            if (profile.getEmail().equals(profileEmail)) {
                return profile;
            }
        }
        return null;
    }

    private void simulateNetworkLatency() {
        try {
            Thread.sleep(2500);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }
```

```
    @Override
    public ProfileIterator createFriendsIterator(String profileEmail) {
        return new LinkedInIterator(this, "friends", profileEmail);
    }

    @Override
    public ProfileIterator createCoworkersIterator(String profileEmail) {
        return new LinkedInIterator(this, "coworkers", profileEmail);
    }
}
```

Clase Profile

```
package refactoring_guru.iterator.example.profile;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Profile {
    private String name;
    private String email;
    private Map<String, List<String>> contacts = new HashMap<>();

    public Profile(String email, String name, String... contacts) {
        this.email = email;
        this.name = name;

        // Parse contact list from a set of "friend:email@gmail.com" pairs.
        for (String contact : contacts) {
            String[] parts = contact.split(":");
            String contactType = "friend", contactEmail;
            if (parts.length == 1) {
                contactEmail = parts[0];
            }
            else {
                contactType = parts[0];
                contactEmail = parts[1];
            }
            if (!this.contacts.containsKey(contactType)) {
                this.contacts.put(contactType, new ArrayList<>());
            }
            this.contacts.get(contactType).add(contactEmail);
        }
    }

    public String getEmail() {
        return email;
    }

    public String getName() {
        return name;
    }

    public List<String> getContacts(String contactType) {
        if (!this.contacts.containsKey(contactType)) {
            this.contacts.put(contactType, new ArrayList<>());
        }
        return contacts.get(contactType);
    }
}
```

Clase Spammer

```
package refactoring_guru.iterator.example.spammer;

import refactoring_guru.iterator.example.iterators.ProfileIterator;
import refactoring_guru.iterator.example.profile.Profile;
import refactoring_guru.iterator.example.social_networks.SocialNetwork;
```

```
public class SocialSpammer {
    public SocialNetwork network;
    public ProfileIterator iterator;

    public SocialSpammer(SocialNetwork network) {
        this.network = network;
    }

    public void sendSpamToFriends(String profileEmail, String message) {
        System.out.println("\nIterating over friends...\n");
        iterator = network.createFriendsIterator(profileEmail);
        while (iterator.hasNext()) {
            Profile profile = iterator.getNext();
            sendMessage(profile.getEmail(), message);
        }
    }

    public void sendSpamToCoworkers(String profileEmail, String message) {
        System.out.println("\nIterating over coworkers...\n");
        iterator = network.createCoworkersIterator(profileEmail);
        while (iterator.hasNext()) {
            Profile profile = iterator.getNext();
            sendMessage(profile.getEmail(), message);
        }
    }

    public void sendMessage(String email, String message) {
        System.out.println("Sent message to: '" + email + "'. Message body: '" + message
+ "'");
    }
}
```

Clase Demo

```
package refactoring_guru.iterator.example;

import refactoring_guru.iterator.example.profile.Profile;
import refactoring_guru.iterator.example.social_networks.Facebook;
import refactoring_guru.iterator.example.social_networks.LinkedIn;
import refactoring_guru.iterator.example.social_networks.SocialNetwork;
import refactoring_guru.iterator.example.spammer.SocialSpammer;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/**
 * Demo class. Everything comes together here.
 */
public class Demo {
    public static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        System.out.println("Please specify social network to target spam tool
(default:Facebook):");
        System.out.println("1. Facebook");
        System.out.println("2. LinkedIn");
        String choice = scanner.nextLine();

        SocialNetwork network;
        if (choice.equals("2")) {
            network = new LinkedIn(createTestProfiles());
        }
        else {
            network = new Facebook(createTestProfiles());
        }

        SocialSpammer spammer = new SocialSpammer(network);
        spammer.sendSpamToFriends("anna.smith@bing.com",
                "Hey! This is Anna's friend Josh. Can you do me a favor and like this
post [link]?");
        spammer.sendSpamToCoworkers("anna.smith@bing.com",
```

```
                "Hey! This is Anna's boss Jason. Anna told me you would be interested in
[link].");
    }

    public static List<Profile> createTestProfiles() {
        List<Profile> data = new ArrayList<Profile>();
        data.add(new Profile("anna.smith@bing.com", "Anna Smith",
"friends:mad_max@ya.com", "friends:catwoman@yahoo.com", "coworkers:sam@amazon.com"));
        data.add(new Profile("mad_max@ya.com", "Maximilian",
"friends:anna.smith@bing.com", "coworkers:sam@amazon.com"));
        data.add(new Profile("bill@microsoft.eu", "Billie",
"coworkers:avanger@ukr.net"));
        data.add(new Profile("avanger@ukr.net", "John Day",
"coworkers:bill@microsoft.eu"));
        data.add(new Profile("sam@amazon.com", "Sam Kitting",
"coworkers:anna.smith@bing.com", "coworkers:mad_max@ya.com",
"friends:catwoman@yahoo.com"));
        data.add(new Profile("catwoman@yahoo.com", "Liza", "friends:anna.smith@bing.com",
"friends:sam@amazon.com"));
        return data;
    }
}
```

DECORATOR DESING PATTERN

Definición: Por medio de la herencia, este patrón permite añadir un comportamiento o funcionalidad a un objeto, este proceso se realiza de forma dinámica, es decir, que los comportamientos son añadidos al objeto en tiempo de ejecución y se aplica a todas las instancias de la clase (relacionada con el objeto).

Implementación propia:

En la siguiente implementación añadiremos en tiempo de ejecución funciones recursivas tales como Sumatoria y Factorial sobre el objeto de la clase MultiplicadorDos.

Clase FuncionRecursivas:

```
package decoratorImplemantation;

/**
 *
 * @author Anderson Nuñez
 */
public interface FuncionRecursivas {
    public void funcAion();
}
```

Clase MultiplicadorDos

```
package decoratorImplemantation;

/**
 *
 * @author Anderson Nuñez
 */
public class MultiplicadorDos implements FuncionRecursivas{

    @Override
    public void funcion() {

        System.out.println("-----------Tabla de multiplicar x2-----------");
        int resultado = recursion(10);

    }
    public int recursion(int cant) {
        if (cant == 0){
            return 0;
        }else{
            System.out.println(String.format("2x%-2d es: | %-2d |", cant, 2*cant));
            return 2*cant+recursion(cant-1);
```

```
        }
    }
}
```

Clase FunctionsDecorator

```
package decoratorImplemantation;

/**
 *
 * @author Anderson Nuñez
 */
public abstract class FunctionsDecorator implements FuncionRecursivas{
    private FuncionRecursivas funciones;

    public FunctionsDecorator(FuncionRecursivas funciones) {
        this.funciones = funciones;
    }
/*
    @Override
    public void funcion() {
        funciones.funcion();
    }*/

    protected FuncionRecursivas getFunction(){
        return funciones;
    }
}
```

Clase Sumatoria

```
package decoratorImplemantation;

/**
 *
 * @author Anderson Nuñez
 */
public class Sumatoria extends FunctionsDecorator{
    int result = 0;
    public Sumatoria(FuncionRecursivas funciones) {
        super(funciones);
    }

    @Override
    public void funcion(){
        System.out.println("-----------------Sumatorias----------------");
        recursive(10);
        getFunction().funcion();
    }
    public int recursive(int cant){
        if (cant==0){
            this.result = 0;
            System.out.println(String.format("Sumatoria desde cero hasta %2d es:
%3d",cant,result));
            return 0;
        }else {
            int result = cant + recursive(cant-1);
            System.out.println(String.format("Sumatoria desde cero hasta %2d es:
%3d",cant,result));
            return result;
        }
    }

}
```

Clase Factorial

```
package decoratorImplemantation;

/**
 *
```

```
 * @author Anderson Nuñez
 */
public class Factorial extends FunctionsDecorator{

    public Factorial(FuncionRecursivas funciones) {
        super(funciones);
    }

    @Override
    public void funcion() {
        System.out.println("---------Factorial de un numero----------");
        int resultado = recursive(10);
        getFunction().funcion();
    }

    public int recursive(int cant){
        if (cant==1) {
            System.out.println(String.format("Factorial de %2d  es: | %7d |", cant, 1));
            return 1;
        }else {
            int factorial = cant * recursive(cant-1);
            System.out.println(String.format("Factorial de %2d  es: | %7d |", cant,
factorial));
            return factorial;
        }
    }

}
```

Resultado de ejecución:

```
run:
Ejecutando Funciones Recursivas
-----------------Sumatorias----------------
Sumatoria desde cero hasta  0 es:   0
Sumatoria desde cero hasta  1 es:   1
Sumatoria desde cero hasta  2 es:   3
Sumatoria desde cero hasta  3 es:   6
Sumatoria desde cero hasta  4 es:  10
Sumatoria desde cero hasta  5 es:  15
Sumatoria desde cero hasta  6 es:  21
Sumatoria desde cero hasta  7 es:  28
Sumatoria desde cero hasta  8 es:  36
Sumatoria desde cero hasta  9 es:  45
Sumatoria desde cero hasta 10 es:  55
---------Factorial de un numero----------
Factorial de  1  es: |       1 |
Factorial de  2  es: |       2 |
Factorial de  3  es: |       6 |
Factorial de  4  es: |      24 |
Factorial de  5  es: |     120 |
Factorial de  6  es: |     720 |
Factorial de  7  es: |    5040 |
Factorial de  8  es: |   40320 |
Factorial de  9  es: |  362880 |
Factorial de 10  es: | 3628800 |
-----------Tabla de multiplicar x2-----------
2x10 es: | 20 |
2x9  es: | 18 |
2x8  es: | 16 |
2x7  es: | 14 |
2x6  es: | 12 |
2x5  es: | 10 |
2x4  es: | 8  |
2x3  es: | 6  |
2x2  es: | 4  |
2x1  es: | 2  |
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ejemplo de implementación:

Ejemplo tomado de https://refactoring.guru/es/design-patterns/decorator/java/example.

En este ejemplo en tiempo de ejecución se editara un archivo denominado OutputDemo.txt realizando procesos de escritura inicialmente, posterior a esto se codificara la información y se guardara la información codificada sobre el mismo archivo, adicional a esto de decodificara la información y se guardara nuevamente. Lo anterior usando el pratron de diseño decorator sobre FileDataSource.

decorators/DataSource.java: Una interfaz común de datos que define operaciones de leer y escribir

```
package refactoring_guru.decorator.example.decorators;

public interface DataSource {
    void writeData(String data);

    String readData();
}
```

decorators/FileDataSource.java: Escritor-lector de datos simple

```
package refactoring_guru.decorator.example.decorators;

import java.io.*;

public class FileDataSource implements DataSource {
    private String name;

    public FileDataSource(String name) {
        this.name = name;
    }

    @Override
    public void writeData(String data) {
        File file = new File(name);
        try (OutputStream fos = new FileOutputStream(file)) {
            fos.write(data.getBytes(), 0, data.length());
        } catch (IOException ex) {
            System.out.println(ex.getMessage());
        }
    }
```

```
    @Override
    public String readData() {
        char[] buffer = null;
        File file = new File(name);
        try (FileReader reader = new FileReader(file)) {
            buffer = new char[(int) file.length()];
            reader.read(buffer);
        } catch (IOException ex) {
            System.out.println(ex.getMessage());
        }
        return new String(buffer);
    }
}
```

decorators/DataSourceDecorator.java: Decorador abstracto base

```java
package refactoring_guru.decorator.example.decorators;

public class DataSourceDecorator implements DataSource {
    private DataSource wrappee;

    DataSourceDecorator(DataSource source) {
        this.wrappee = source;
    }

    @Override
    public void writeData(String data) {
        wrappee.writeData(data);
    }

    @Override
    public String readData() {
        return wrappee.readData();
    }
}
```

decorators/EncryptionDecorator.java: Decorador de codificación

```java
package refactoring_guru.decorator.example.decorators;

import java.util.Base64;

public class EncryptionDecorator extends DataSourceDecorator {

    public EncryptionDecorator(DataSource source) {
        super(source);
    }

    @Override
    public void writeData(String data) {
        super.writeData(encode(data));
    }

    @Override
    public String readData() {
        return decode(super.readData());
    }

    private String encode(String data) {
        byte[] result = data.getBytes();
        for (int i = 0; i < result.length; i++) {
            result[i] += (byte) 1;
        }
        return Base64.getEncoder().encodeToString(result);
    }

    private String decode(String data) {
        byte[] result = Base64.getDecoder().decode(data);
        for (int i = 0; i < result.length; i++) {
            result[i] -= (byte) 1;
        }
        return new String(result);
    }
}
```

decorators/CompressionDecorator.java: Decorador de compresión

```java
package refactoring_guru.decorator.example.decorators;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Base64;
import java.util.zip.Deflater;
import java.util.zip.DeflaterOutputStream;
import java.util.zip.InflaterInputStream;

public class CompressionDecorator extends DataSourceDecorator {
    private int compLevel = 6;

    public CompressionDecorator(DataSource source) {
        super(source);
    }

    public int getCompressionLevel() {
        return compLevel;
    }

    public void setCompressionLevel(int value) {
        compLevel = value;
    }
```

```java
    @Override
    public void writeData(String data) {
        super.writeData(compress(data));
    }

    @Override
    public String readData() {
        return decompress(super.readData());
    }

    private String compress(String stringData) {
        byte[] data = stringData.getBytes();
        try {
            ByteArrayOutputStream bout = new ByteArrayOutputStream(512);
            DeflaterOutputStream dos = new DeflaterOutputStream(bout, new Deflater(compL
            dos.write(data);
            dos.close();
            bout.close();
            return Base64.getEncoder().encodeToString(bout.toByteArray());
        } catch (IOException ex) {
            return null;
        }
    }
```

```
    private String decompress(String stringData) {
        byte[] data = Base64.getDecoder().decode(stringData);
        try {
            InputStream in = new ByteArrayInputStream(data);
            InflaterInputStream iin = new InflaterInputStream(in);
            ByteArrayOutputStream bout = new ByteArrayOutputStream(512);
            int b;
            while ((b = iin.read()) != -1) {
                bout.write(b);
            }
            in.close();
            iin.close();
            bout.close();
            return new String(bout.toByteArray());
        } catch (IOException ex) {
            return null;
        }
    }
}
```

Demo.java: Código cliente

```
package refactoring_guru.decorator.example;

import refactoring_guru.decorator.example.decorators.*;

public class Demo {
    public static void main(String[] args) {
        String salaryRecords = "Name,Salary\nJohn Smith,100000\nSteven Jobs,912000";
        DataSourceDecorator encoded = new CompressionDecorator(
                                        new EncryptionDecorator(
                                            new FileDataSource("out/OutputDemo.txt")));
        encoded.writeData(salaryRecords);
        DataSource plain = new FileDataSource("out/OutputDemo.txt");

        System.out.println("- Input ----------------");
        System.out.println(salaryRecords);
        System.out.println("- Encoded --------------");
        System.out.println(plain.readData());
        System.out.println("- Decoded --------------");
        System.out.println(encoded.readData());
    }
}
```

OutputDemo.txt: Resultado de la ejecución

```
- Input ----------------
Name,Salary
John Smith,100000
Steven Jobs,912000
- Encoded --------------
Zkt7e1Q5eU8yUm1Qe0ZsdHJ2VXp6dDBKVnhrUHtUe0sxRUYxQkJIdjVLTVZ0dVI5Q2IwOXFISmVUMU5rcENQmdx
- Decoded --------------
Name,Salary
John Smith,100000
Steven Jobs,912000
```