

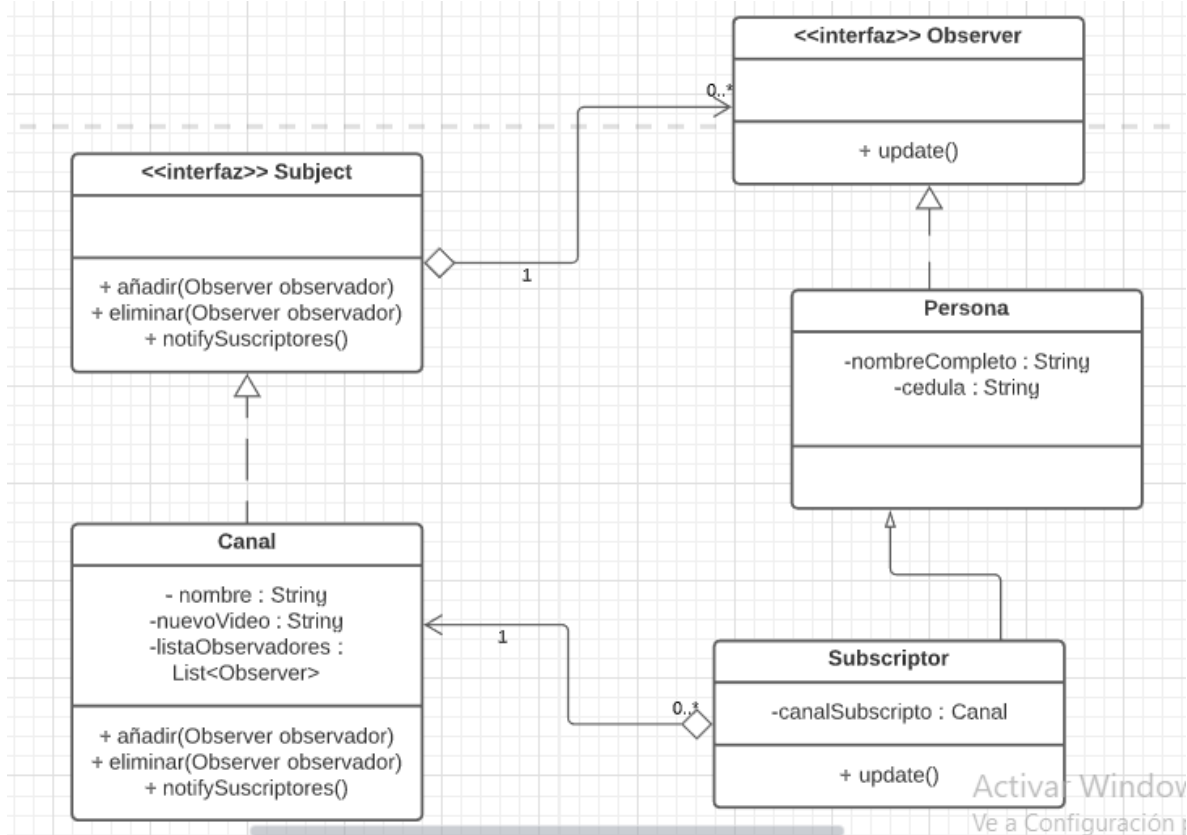


| | | |
|--|-------------------------------|-------------------------------|
|  | VICERRECTORADO DOCENTE | Código: GUIA-PRL-001 |
| | CONSEJO ACADÉMICO | Aprobación: 2016/04/06 |
| Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación | | |

| | | | | | |
|---|---|--|--|--|--|
|  | | | PRÁCTICA DE LABORATORIO | | |
| CARRERA: COMPUTACION | | | ASIGNATURA: PROGRAMACION APLICADA | | |
| NRO. PRÁCTICA: | 3 | TÍTULO PRÁCTICA: PATRON COMPORTAMIENTO OBSERVER | | | |
| OBJETIVO ALCANZADO: Identificar los cambios importantes de Java Diseñar e Implementar las nuevas técnicas de programación Entender los patrones de Java | | | | | |
| ACTIVIDADES DESARROLLADAS | | | | | |
| <p>1. Revisar la teoría y conceptos de Patrones de Diseño de Java.</p> <p>PATRON DE COMPORTAMIENTO OBSERVER.</p> <p>Este patrón sirve para notificar a varios objetos que su estado ha cambiado. Esto se basa en que una clase (Subject u Observable) que puede ser observada por una o varias otras clases (Observers). Para esto se deben crear 2 interfaces:</p> <ul style="list-style-type: none"> • Interfaz Observable o Subject.- en esta interfaz se deben declarar 3 métodos, y la debe implementar la clase Observable: <ul style="list-style-type: none"> ❖ Añadir(Objeto objeto) .- Esta sirve para añadir objetos a una lista y tener constancia de que objetos están interesados en recibir un mensaje cuando el estado del objeto de esta clase cambia. ❖ Eliminar(Objeto objeto) .- Para eliminar objetos que recibían mensajes cuando el estado del objeto de esta clase cambiaba. ❖ Notificar() .- Para notificar a la lista de objetos que hubo un cambio de estado del objeto de la clase que los contiene. • Interfaz Observer.- En esta interfaz se deben declarar 1 método en la clase que observa el cambio de estado de un objeto de una clase: <ul style="list-style-type: none"> ❖ Update() .- Mediante este método se le notifica al objeto observador que el objeto observable cambio su estado. <p>Este patrón nació de una problemática: los objetos observadores quieren saber cuándo otro objeto cambia de estado, entonces los objetos observadores tenían que preguntarle al objeto observable si ya cambio de estado a cada momento determinado. Para evitar este número indeterminado de preguntas, se creó la solución que el objeto observable, les informe a los objetos observadores cuando su estado ha cambiado. Lo cual soluciona el problema de una manera muy simple, ya que maneja los objetos que quieren ser notificados cuando existe un cambio de estado del objeto observable.</p> | | | | | |


2. Diseñar e implementar cada estudiante un patrón de diseño y verificar su funcionamiento. A continuación, se detalla el patrón a implementar:

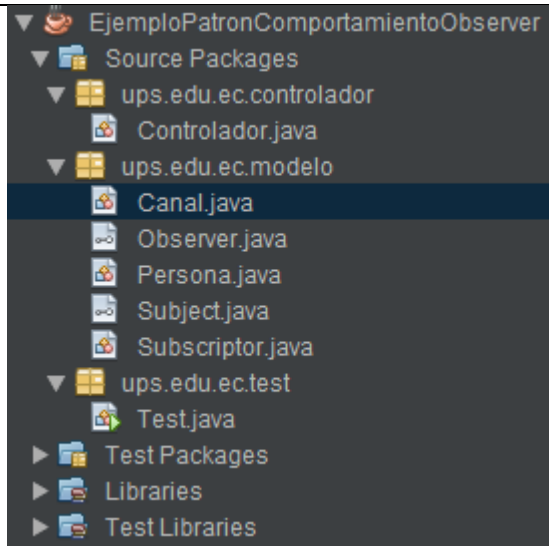
Para este patrón de comportamiento se creó un ejemplo entre un canal y un subscriptor. A continuación, se detalla el diagrama UML:



Como podemos observar, el objeto Observer también guarda el objeto Subject para poder utilizarlo en el método Update(), y poder conseguir los datos del Objeto Subject.

Estos son los paquetes y clases creados en este proyecto:

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



A continuación, el ejemplo de como se ejecuta el código en consola:

```

run:
Lionel Hay un nuevo video de: Codigos :Primer video
Andres Hay un nuevo video de: Codigos :Primer video
Mateo Hay un nuevo video de: Codigos :Primer video
Pizarro Hay un nuevo video de: Codigos :Primer video
Claudio Hay un nuevo video de: Codigos :Primer video

-----

Mateo Hay un nuevo video de: Codigos :Segundo video
Pizarro Hay un nuevo video de: Codigos :Segundo video
Claudio Hay un nuevo video de: Codigos :Segundo video
Canal{nombre=Codigos, nuevoVideo=Segundo video, listaSuscriptores=[Persona{nombre=Mateo}Subscriber{nombre=, Persona
BUILD SUCCESSFUL (total time: 0 seconds)

```


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

Ventajas

- Puedes introducir nuevas clases suscriptoras sin tener que cambiar el código de la notificadora (y viceversa si hay una interfaz notificadora).
- Puedes establecer relaciones entre objetos durante el tiempo de ejecución.
- Permite agregar un número ilimitado de Observadores que desean enterarse del cambio.

Desventajas

- Los suscriptores son notificados en un orden aleatorio.
- Si existe una gran cantidad de objetos observadores, tardaría mucho tiempo en notificarlo a todos.
- Las informaciones de cambios transmitidas pueden ser irrelevantes para ciertos observadores.

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

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

Clase Controlador:

/*

* To change this license header, choose License Headers in Project Properties.

* To change this template file, choose Tools | Templates

* and open the template in the editor.

*/

package ups.edu.ec.controlador;

import java.util.ArrayList;

import java.util.List;

/**

*

* @author Adolfo

* @param <T>

*/


public class Controlador<T> {

private List<T> listaGenerica;

public Controlador() {

listaGenerica = new ArrayList<>();

}

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

```
public void create(T objeto){
    listaGenerica.add(objeto);
}
```

```
public void update(T objetoAntiguo, T objetoNuevo){
    int index = listaGenerica.indexOf(objetoAntiguo);
    listaGenerica.set(index, objetoNuevo);
}
```

```
}
```

Interface Subject:

```
/*
```

```
* To change this license header, choose License Headers in Project Properties.
```

```
* To change this template file, choose Tools | Templates
```

```
* and open the template in the editor.
```

```
*/
```

```
package ups.edu.ec.modelo;
```

```
/**
```


```
*
```

```
* @author Adolfo
```

```
*/
```

```
public interface Subject {
```

```
    void añadirSubscriptor(Observer sub);
```

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

```
void eliminarSubscriptor(Observer sub);
```

```
void notifySubscriptores();
```

```
}
```

Clase Canal:

```
/*
```

```
* To change this license header, choose License Headers in Project Properties.
```

```
* To change this template file, choose Tools | Templates
```

```
* and open the template in the editor.
```

```
*/
```

```
package ups.edu.ec.modelo;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import java.util.Objects;
```

```
/**
```

```
*
```

```
* @author Adolfo
```

```
*/
```

```
public class Canal implements Subject {
```

```
    private String nombre;
```

```
    private String nuevoVideo;
```

```
    private List<Observer> listaSubscriptores;
```

```
    public Canal(String nombre) {
```

```
this.nombre = nombre;

listaSubscriptores = new ArrayList<>();

}

public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}


public String getNuevoVideo() {
    return nuevoVideo;
}

public void setNuevoVideo(String nuevoVideo) {
    this.nuevoVideo = nuevoVideo;
    this.notifySubscriptores();
}

@Override
public int hashCode() {
    int hash = 3;
    hash = 59 * hash + Objects.hashCode(this.nombre);
    return hash;
}

@Override
```

```
public boolean equals(Object obj) {  
    if (this == obj) {  
        return true;  
    }  
    if (obj == null) {  
        return false;  
    }  
    if (getClass() != obj.getClass()) {  
        return false;  
    }  
    final Canal other = (Canal) obj;  
    if (!Objects.equals(this.nombre, other.nombre)) {  
        return false;  
    }  
    return true;  
}  
  
@Override  
public String toString() {  
    return "Canal{" + "nombre=" + nombre + ", nuevoVideo=" + nuevoVideo +  
        ", listaSubscriptores=" + listaSubscriptores + '}';  
}  
  
@Override  
public void añadirSubscriptor(Observer sub) {  
    listaSubscriptores.add(sub);  
}
```


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

@Override

```
public void eliminarSubscriber(Observer sub) {
    listaSubscriptores.remove(sub);
}
```

@Override

```
public void notifySubscriptores() {
    listaSubscriptores.stream().forEach(sub -> sub.update());
}
```

```
}
```

Interface Observer:

```
/*
```

```
* To change this license header, choose License Headers in Project Properties.
```

```
* To change this template file, choose Tools | Templates
```

```
* and open the template in the editor.
```

```
*/
```

```
package ups.edu.ec.modelo;
```

```
/**
```

```
*
```


```
* @author Adolfo
```

```
*/
```

```
public interface Observer {
```

```
    public void update();
```

```
}
```

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

Clase Persona:

/*

* To change this license header, choose License Headers in Project Properties.

* To change this template file, choose Tools | Templates

* and open the template in the editor.

*/

package ups.edu.ec.modelo;

/**

*

* @author User

*/

public class Persona implements Observer{

private String nombre;

public Persona(String nombre) {

 this.nombre = nombre;

}

public String getNombre() {


 return nombre;

}

public void setNombre(String nombre) {

 this.nombre = nombre;

}

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

@Override

```
public String toString() {
    return "Persona{" + "nombre=" + nombre + '}';
}
```

@Override

```
public void update() {

}
```

```
}
```

Clase Subscriptor:

```
/*
```

```
* To change this license header, choose License Headers in Project Properties.
```

```
* To change this template file, choose Tools | Templates
```

```
* and open the template in the editor.
```

```
*/
```

```
package ups.edu.ec.modelo;
```

```
/**
```

```
*
```


```
* @author Adolfo
```

```
*/
```

```
public class Subscriptor extends Persona {
```

```
    private Canal canalSuscripcion;
```

```
public Subscriptor(String nombre, Canal canalSuscripcion) {  
    super(nombre);  
    this.canalSuscripcion = canalSuscripcion;  
}  
  
public Canal getCanalSuscripcion() {  
    return canalSuscripcion;  
}  
  
public void setCanalSuscripcion(Canal canalSuscripcion) {  
    this.canalSuscripcion = canalSuscripcion;  
}  
  
@Override  
public String toString() {  
    return super.toString() + "Subscriptor{" + "nombre=";  
}  
  
@Override  
public void update() {  
    System.out.println(this.getNombre() + " Hay un nuevo video de: " +  
        canalSuscripcion.getNombre() + " :" + canalSuscripcion.getNuevoVideo());  
}  
  
}
```

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

| |
|--|
| |
| |
| |
| N. |
| |
| RESULTADO(S) OBTENIDO(S): <ul style="list-style-type: none"> Un entendimiento claro y preciso sobre un patrón de Comportamiento en Java, los cuales son de ayuda para futuras prácticas. Construcción de un ejemplo aplicando el patrón asignado por el docente, y por ende la demostración de dicho ejemplo. |
| CONCLUSIONES: <p>Esta practica es de mucha ayuda por varias razones: la primera nos ayuda a nosotros como estudiantes a leer sobre los distintos patrones que existen en Java y que nos ayudarán en el futuro para construir mejores aplicaciones. Segundo, aprendí que hay muchos patrones y que es necesario conocerlos para aplicarlos de la forma correcta y además es de ayuda para optimizar muchos procesos. Y por último, esto nos sirve para conocer las distintas problemáticas que existe al momento de programar.</p> |
| RECOMENDACIONES: <p>No existe ninguna recomendación, ya que todas las especificaciones por parte del docente fueron claras y específicas sobre esta tarea.</p> |

Nombre de estudiante: Adolfo Sebastián Jara Gavilanes.

Firma de estudiante:

