



Tecnologías de Programación

Paradigma Orientado a Objetos

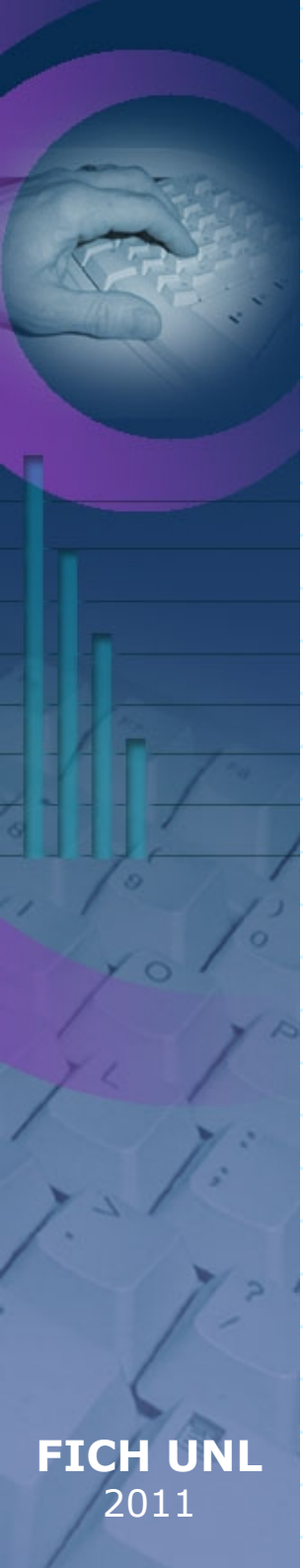
V – Patrones de Diseño



Singleton

El Patrón de Diseño Singleton (en español, Instancia Única) se utiliza para garantizar que una clase sólo tenga **una única instancia** y para facilitar un **punto de acceso global a la misma**.

```
public class Singleton {  
  
    private static Singleton instanciaUnica;  
  
    // Constructor  
    private Singleton() {}  
  
    public static Singleton getInstance() {  
  
        if (instanciaUnica == null) {  
  
            instanciaUnica = new Singleton();  
  
        }  
  
        return instanciaUnica;  
  
    }  
}
```



```
public class Nodo extends Componente {

    private Vector<Componente> cElementos; // Hijos.

    private static Nodo oNodoRaiz;

    // CONSTRUCTORES
    private Nodo(String nombre) {
        super(nombre);
        this.cElementos = new Vector<Componente>();
    }

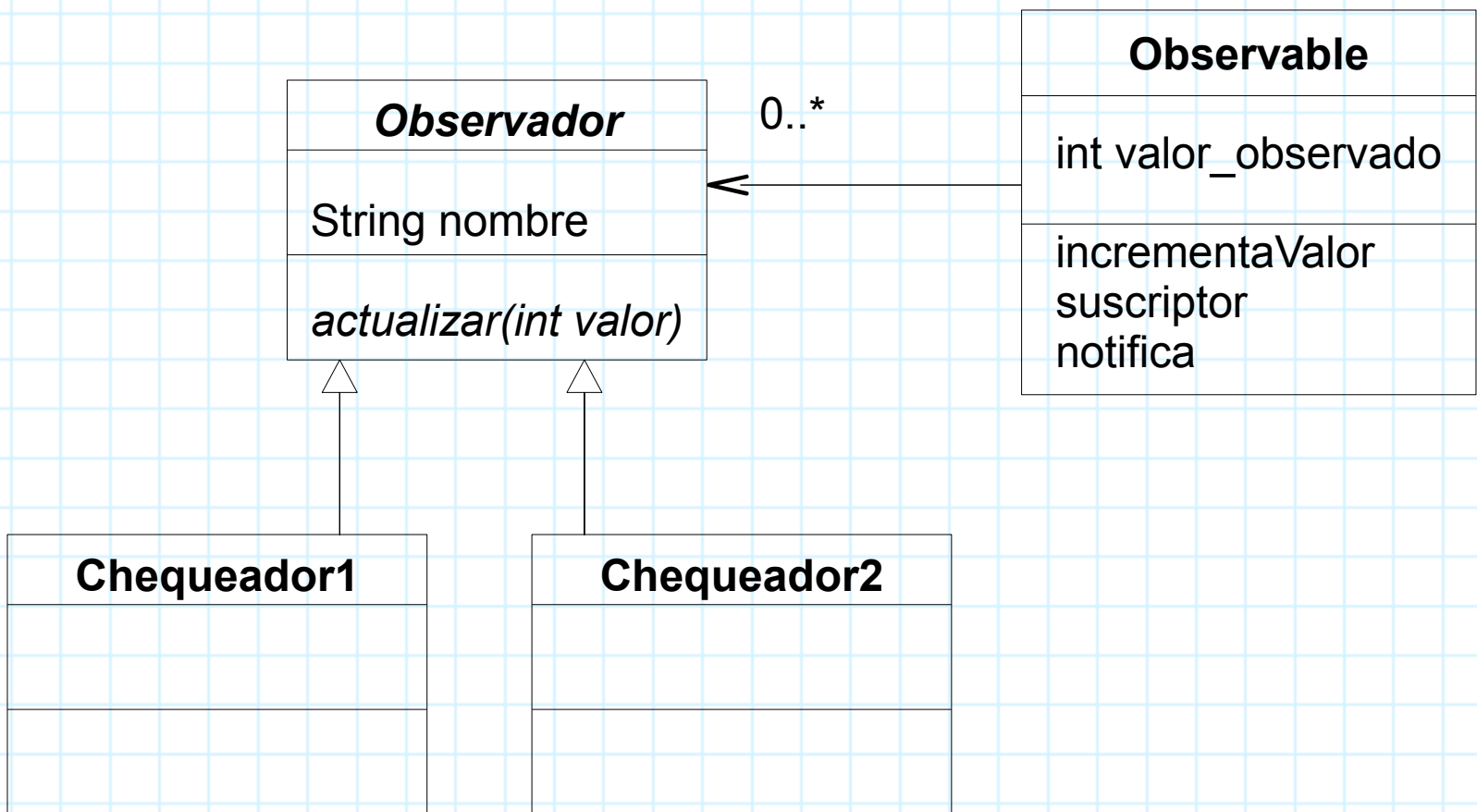
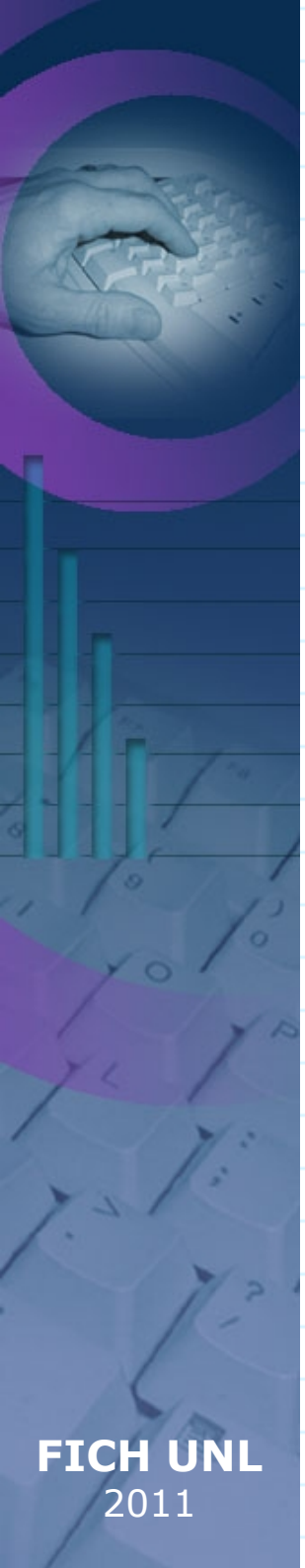
    public Nodo(String nombre, Componente oPadre) {
        super(nombre, oPadre);
        this.cElementos = new Vector<Componente>();
    }

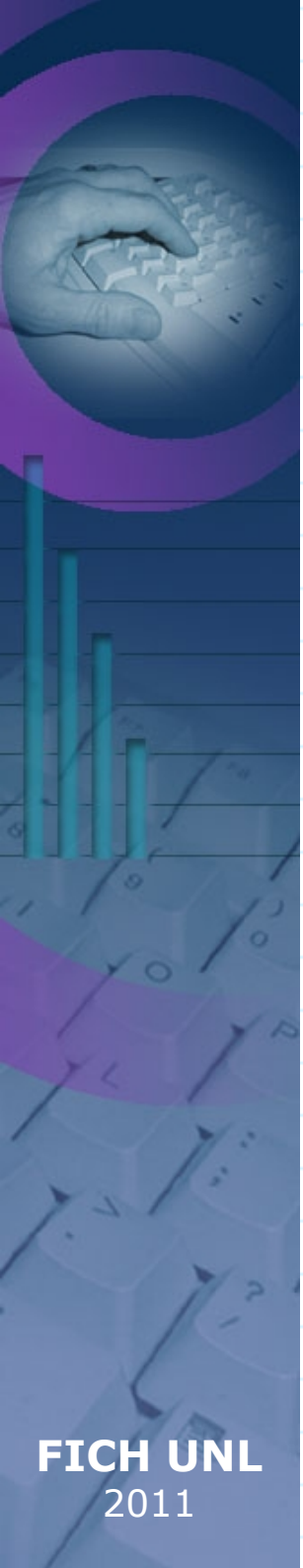
    public static Nodo getRaiz(String nombre) {
        if (oNodoRaiz == null) {
            oNodoRaiz = new Nodo(nombre);
        }
        return oNodoRaiz;
    }
}
```



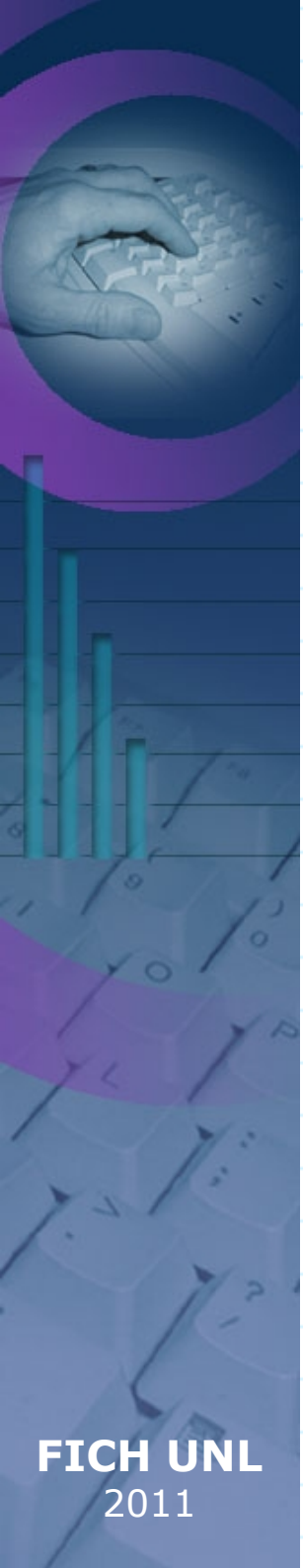
Observer

- El patrón Observador (en inglés: Observer) define una dependencia del tipo **uno-a-muchos** entre objetos, de manera que cuando uno de los objetos cambia su estado, el observador se encarga de notificar este cambio a todos los otros dependientes.
- El objetivo de este patrón es **desacoplar** la clase de los objetos clientes del objeto, aumentando la modularidad del lenguaje, así como evitar bucles de actualización.
- Las ideas básicas del patrón, que son bien sencillas: el objeto de datos, llamémoslo "Sujeto" a partir de ahora, contiene atributos mediante los cuales cualquier objeto **observador** o **vista** se puede suscribir a él pasándole una referencia a sí mismo. El **Sujeto** mantiene así una lista de las referencias a sus observadores.



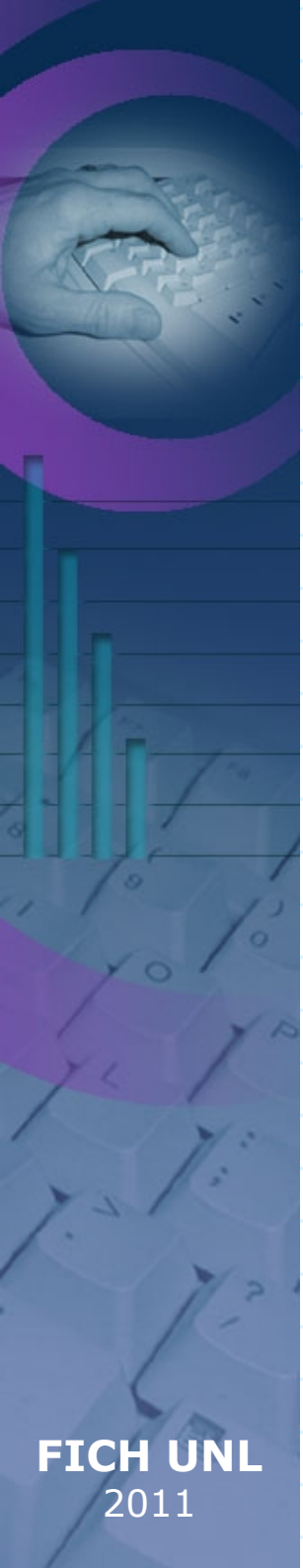


```
public class Observable {  
  
    private Vector<Observador> cObservadores;  
  
    private int valor_observador;  
  
    public Observable(int valor_observador) {  
        cObservadores = new Vector<Observador>();  
        this.valor_observador = valor_observador;  
    }  
  
    public void incrementaValor() {  
        valor_observador++;  
        System.out.println("Se incremento valor a: " + valor_observador);  
        this.notifica();  
    }  
  
    public void suscriptor(Observador obj) {  
        this.cObservadores.add(obj);  
    }  
  
    private void notifica() {  
        for(Observador oObservador : this.cObservadores) {  
            oObservador.actualizar(this.valor_observador);  
        }  
    }  
}
```



```
public abstract class Observador {  
  
    protected String nombre;  
  
    public Observador(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public abstract void actualizar(int valor);  
  
}
```

```
public class Chequeador1 extends Observador {  
  
    public Chequeador1(String nombre) {  
        super(nombre);  
    }  
  
    public void actualizar(int valor) {  
        if (valor == 2) {  
            System.out.println("Observador " +  
                               super.nombre +  
                               " Observable en valor 2");  
        }  
    }  
  
}
```

```
public class Principal {  
  
    public static void main(String[] args) {  
  
        Observable oSujeto = new Observable(0);  
  
        oSujeto.suscriptor(new Chequeador1("Observador A"));  
        oSujeto.suscriptor(new Chequeador2("Observador B"));  
  
        oSujeto.incrementaValor();  
        oSujeto.incrementaValor();  
        oSujeto.incrementaValor();  
        oSujeto.incrementaValor();  
        oSujeto.incrementaValor();  
    }  
}
```

CONSOLA

```
Se incremento valor a: 1  
Se incremento valor a: 2  
Observador Observador A Observable en valor 2  
Se incremento valor a: 3  
Observador Observador B Observable en valor 3  
Se incremento valor a: 4  
Se incremento valor a: 5
```