



Tecnologías de Programación

Paradigma Orientado a Objetos

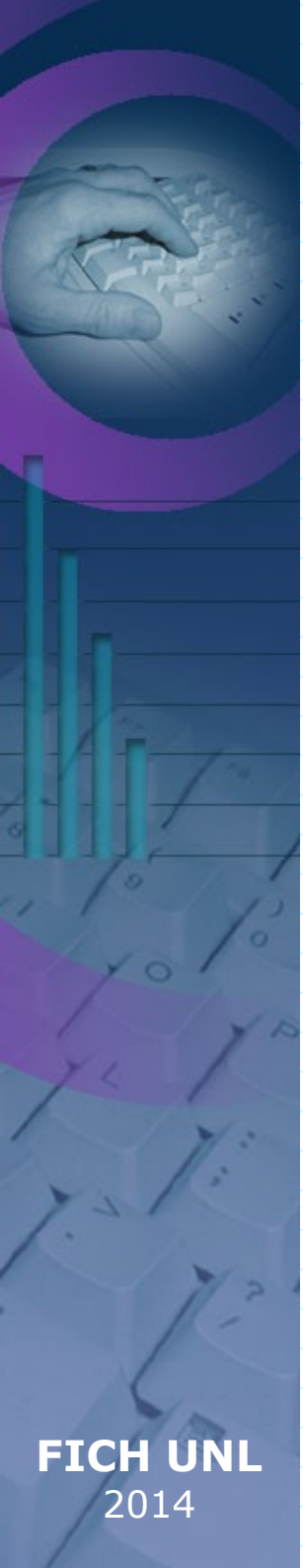
V – Patrones de Diseño



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>();
    }

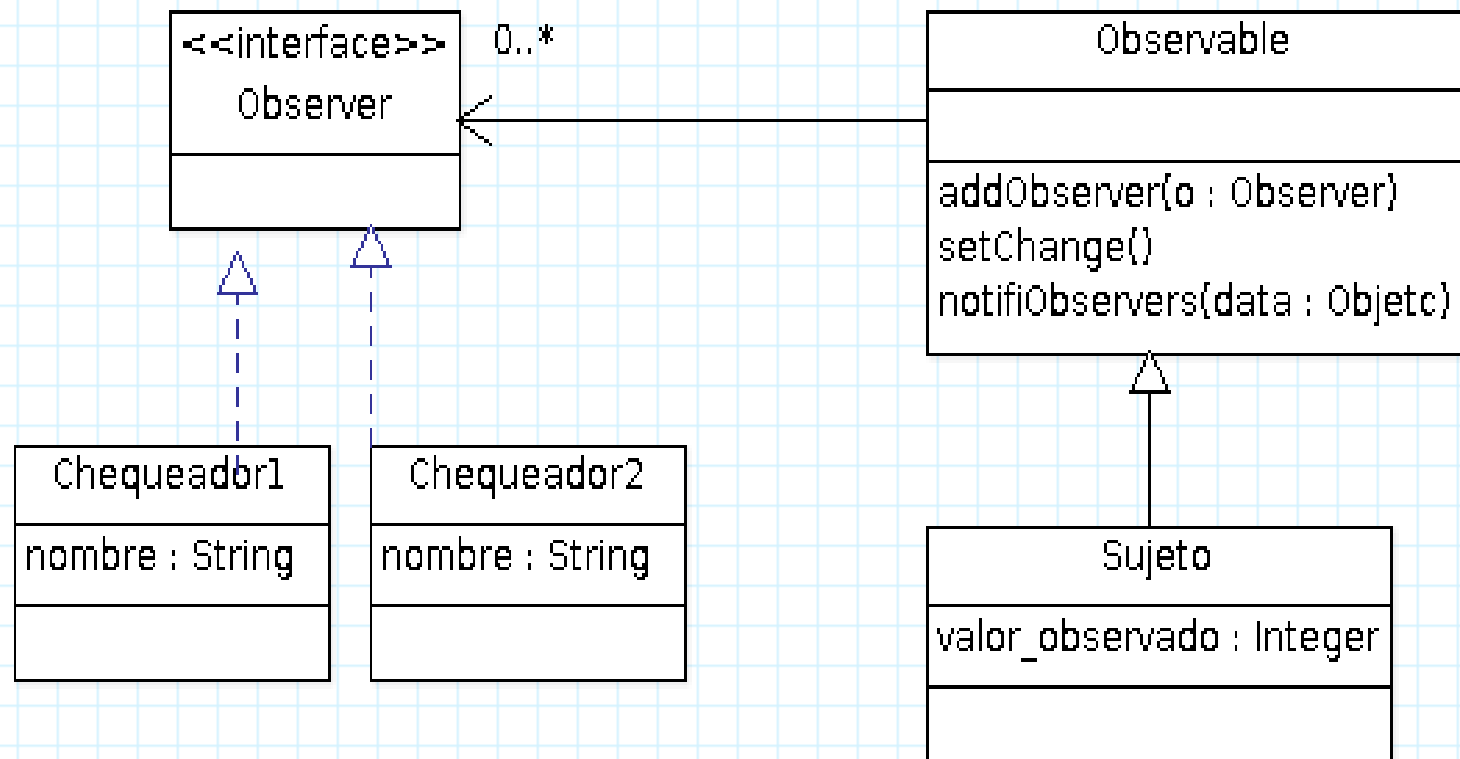
    public Nodo(String nombre, Componente oPadre) {
        super(nombre, oPadre);
        this.cElementos = new Vector<Componente>();
    }

    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.



Soporte Java para el patrón “Observador”

```
/*Superclase de cualquier clase que desee ser observada*/
```

```
Clase java.util.Observable
```

```
// METODOS DE INSTANCIA
```

```
addObserver(java.util.Observer oObserver)
```

```
deleteObserver(java.util.Observer oObserver)
```

```
/*Un objeto observable informa que ha cambiado*/
```

```
SetChange()
```

```
/*Avisa a los observadores*/
```

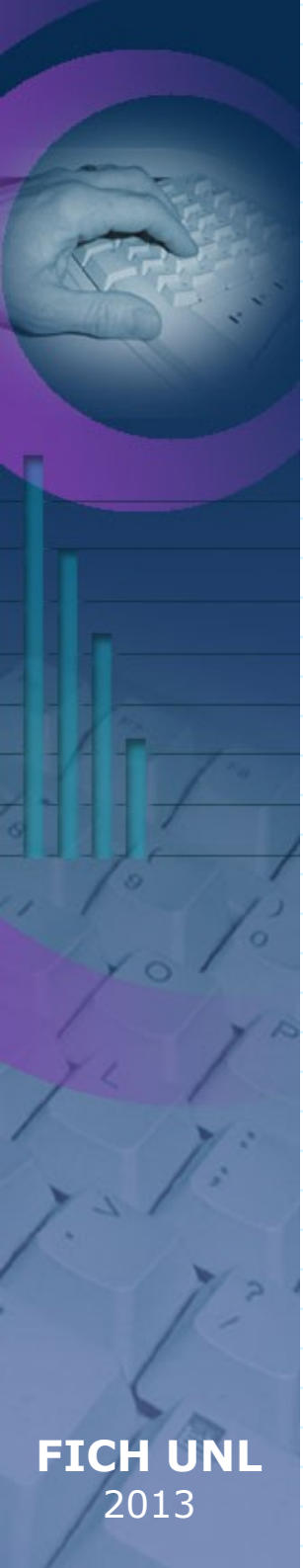
```
notifyObserver()
```

```
notifyObservers(Object data)
```

```
/*Debe implementarla cualquier clase que desea ser observadora*/
```

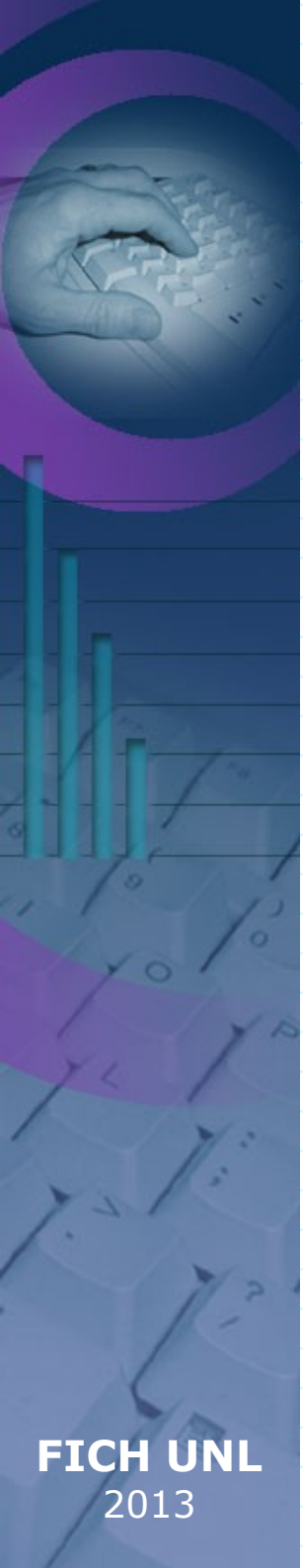
```
Interface java.util.Observer
```

```
void update (Observable o, Object data)
```



```
public class Sujeto extends Observable {  
  
    private Integer valor_observador;  
    public Sujeto(Integer valor_observador) {  
        super();  
        this.valor_observador = valor_observador;  
    }  
    public void incrementaValor() {  
        valor_observador++;  
        System.out.println("Valor: " + valor_observador);  
        this.setChanged();  
        this.notifyObservers(valor_observador);  
    }  
}
```

```
public class Chequeador1 implements Observer {  
  
    private String nombre;  
    public Chequeador1(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public void update(Observable arg0, Object arg1) {  
  
        // Voy a sensar los valores pares  
        Integer valor_observado = (Integer) arg1;  
        if (valor_observado%2 == 0)  
            System.out.println( "Observador " + nombre +  
                                " Valor censado PAR: " + arg1);  
    }  
}
```

```
public class Principal {  
    public static void main(String[] args) {  
        Sujeto oSujeto = new Sujeto(3);  
  
        Chequeador1 oCheck1 = new Chequeador1("PARES");  
        Chequeador2 oCheck2 = new Chequeador2("Multi3");  
  
        oSujeto.addObserver(oCheck1);  
        oSujeto.addObserver(oCheck2);  
  
        for ( Integer i = new Integer(1); i <= 8 ; i++ )  
            oSujeto.incrementaValor();  
    }  
}
```

CONSOLA

Se incremento valor a: 4

Observador PARES Valor censado PAR: 4

Se incremento valor a: 5

Se incremento valor a: 6

Observador Multi3 Valor censado MULTIPLO DE 3: 6

Observador PARES Valor censado PAR: 6

Se incremento valor a: 7

Se incremento valor a: 8

Observador PARES Valor censado PAR: 8

Se incremento valor a: 9

Observador Multi3 Valor censado MULTIPLO DE 3: 9

Se incremento valor a: 10

Observador PARES Valor censado PAR: 10

Se incremento valor a: 11