

Dependencias



- Mecanismo que permite alertar a unos objetos del comportamiento de otro
 - Usado en MCV
 - Cuando un objeto se modifica, puede avisar a otros objetos para que actúen en consecuencia
- Observable: objeto cuyo comportamiento queremos observar
- Observador: objeto que está pendiente del comportamiento de un observable

1

Interface

java.util.Observer



- Debe implementarla cualquier clase que desee ser observadora

```
public interface Observer {  
    public void update(Observable, Object)  
}
```

3

Clase

java.util.Observable



- Superclase de cualquier clase que desee ser observada
- Metodos de instancia

```
void addObserver(Observer)  
void deleteObserver(Observer)  
void deleteObservers()  
void setChanged()  
void notifyObservers()  
void notifyObservers(Object)
```

2

Comportamiento del modelo de dependencias



- Un objeto observable informa que ha cambiado con

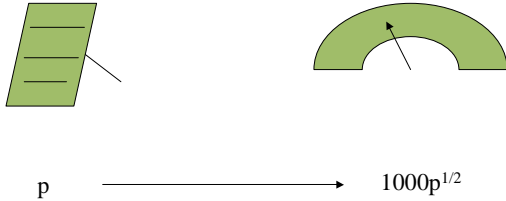
```
void setChanged()
```
- avisa a los observadores con uno de entre

```
void notifyObservers()  
void notifyObservers(Object)
```
- y los observadores reciben

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

4

Ejemplo. Pedal y aguja cuentarevoluciones



5

Ejemplo. El observable Pedal de acelerador



```
public class Pedal extends Observable {
    public Pedal() {
        fuerza = 0;
    }
    public void acelera(int x) {
        fuerza += x;
        setChanged();
        notifyObservers();
    }
    public void desacelera(int x) {
        fuerza -= x;
        setChanged();
        notifyObservers(new Integer(fuerza));
    }
    public int pedal() {
        return fuerza;
    }
}
```

7

Ejemplo. El observador Aguja cuentarevoluciones



```
import java.util.*;
public class AgujaRev implements Observer {
    private int rev;
    public AgujaRev() {
        rev = 0;
    }
    public int revoluciones() {
        return rev;
    }
    public void update(Observable ob, Object o) {
        if (ob!=null)
            rev = (int)(1000*Math.sqrt(((Pedal)ob).pedal()));
    }
}
```

6

Ejemplo. Todo junto



```
public class PrPedal {
    public static void main(String[] args) {
        Pedal p = new Pedal();
        AgujaRev v = new AgujaRev();
        // Hacemos v observador de p
        p.addObserver(v);
        p.acelera(5);
        System.out.println(v.revoluciones());
        p.desacelera(2);
        System.out.println(v.revoluciones());
    }
}
```

8

El modelo de dependencias



- La clase `Observable` tiene como variable de instancia una colección de observadores
- La ejecución de `notifyObservers()`
 - Si el objeto ha cambiado (`setChanged()`)
 - Itera sobre la colección enviando el mensaje
 - `update(this, null)`
- La ejecución de `notifyObservers(o)`
 - Si el objeto ha cambiado (`setChanged()`)
 - Itera sobre la colección enviando el mensaje
 - `update(this, o)`
- Ejercicio: Implementar dependencias

9

La clase Boton I



```
import java.util.*;
public class Boton extends Observable implements Observer {
    private boolean pulsado;
    public Boton() {
        pulsado = false;
    }
    public boolean estaPulsado() {
        return pulsado;
    }
    public void liberar() {
        pulsado = false;
    }
    public void pulsar() {
        if (!estaPulsado()) {
            pulsado = true;
            setChanged();
            notifyObservers();
        }
    }
    ...
}
```

11

Ejemplo. Botonera



- Una botonera está compuesta por una serie de botones.
- Cuando un botón se pulsa, cualquier otro botón que estuviera pulsado se debe liberar.
- Vamos a implementar dos clases
 - Botonera: la que contiene a los botones
 - Boton: implementa un botón.
 - Cada botón debe observar el comportamiento del resto de los botones de la misma botonera

10

La clase Boton II



```
...
public void update(Observable ob, Object o) {
    Boton b = (Boton)ob;
    // if (this!=b) // (1)
        liberar();
}
public String toString() {
    if (estaPulsado())
        return "#";
    else
        return "-";
}
}
```

12

La clase Botonera I



```
public class Botonera
{
    private Boton bs[];
    public Botonera(int n)
    {
        bs = new Boton[n];
        for (int i=0;i<n;i++)
            bs[i] = new Boton();
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                if (i!=j) // (2)
                    bs[i].addObserver(bs[j]);
    }
    ...
}
```

13

La clase UsaBotonera



```
public class UsaBotonera {
    public static void main(String [] args) {
        Botonera b = new Botonera(5);
        System.out.println("Inicialmente :"+b);
        b.pulsar(3);
        System.out.println("Se pulsa el 3:"+b);
        b.pulsar(1);
        System.out.println("Se pulsa el 1:"+b);
    }
}
```

15

La clase Botonera II



```
...
public void pulsar(int n) {
    bs[n].pulsar();
}

public String toString() {
    String r="";
    for(int i=0;i<bs.length;i++)
        r+=bs[i];
    return r;
}
}
```

14