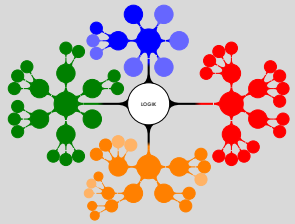
Tema 1

Desarrollo de software con patrones

Patron “Interceptor”

Asignatura *Desarrollo de Software* fecha 24 febrero 2016

Manuel I. Capel
Departamento de Lenguajes y Sistemas Informaticos
Universidad de Granada



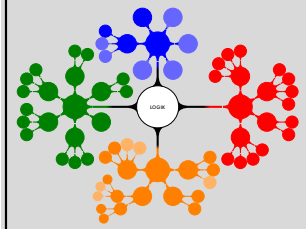
Contexto

Se pretende desarrollar un sistema de control automático de la velocidad para automóviles.
Inicialmente el sistema funcionará en “modo manual”: el conductor acelera hasta conseguir una velocidad de “crucero”.
La velocidad de crucero ha de mantenerse siempre hasta que el conductor frene o apague el modo automático.

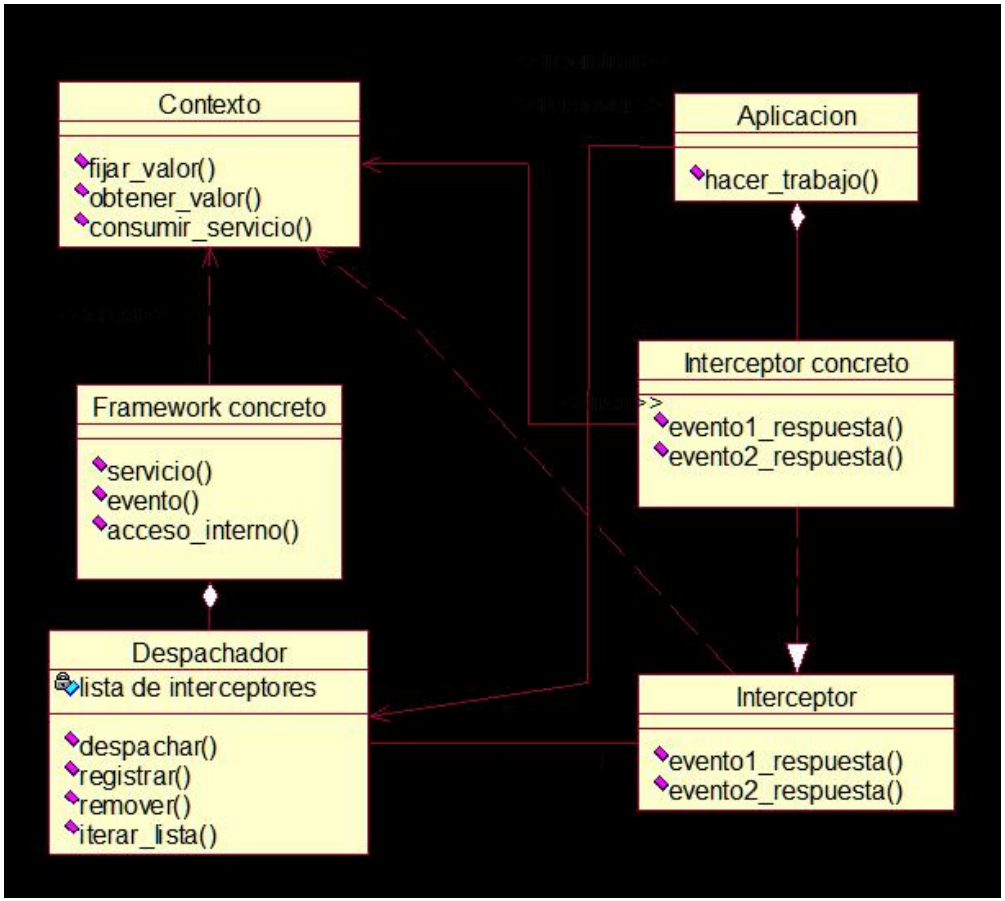
- *Acelerar*: incrementa la velocidad continuamente
- *Apagado*: se desactiva el sistema de control automático de velocidad
- *Reiniciar*: vuelve a la última velocidad y la mantiene
- *Modo Automático*: se memoriza y mantiene la velocidad de crucero actual

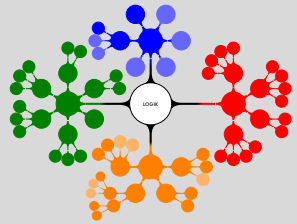
Applet con simulador del SCACV

`http://lsi.ugr.es/ist/Documentos/practica1/supuesto_control_automatico-2.htm`



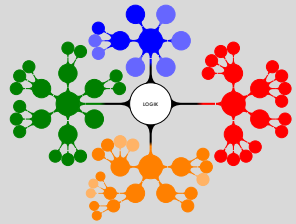
Patrón Interceptor General





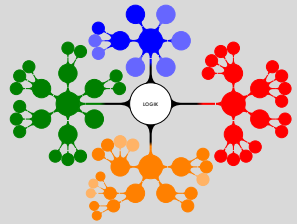
Entidades del diseño

- Activación automática en el `Cliente` llamando a su método *enviarPeticion(double)*
- Cada instancia de *Filtro* entre en ejecución cuando se envía la petición desde `Cliente`
- Cada cadena de filtros, al menos, posee una instancia de una clase *objetivo*, que se ejecuta en última posición

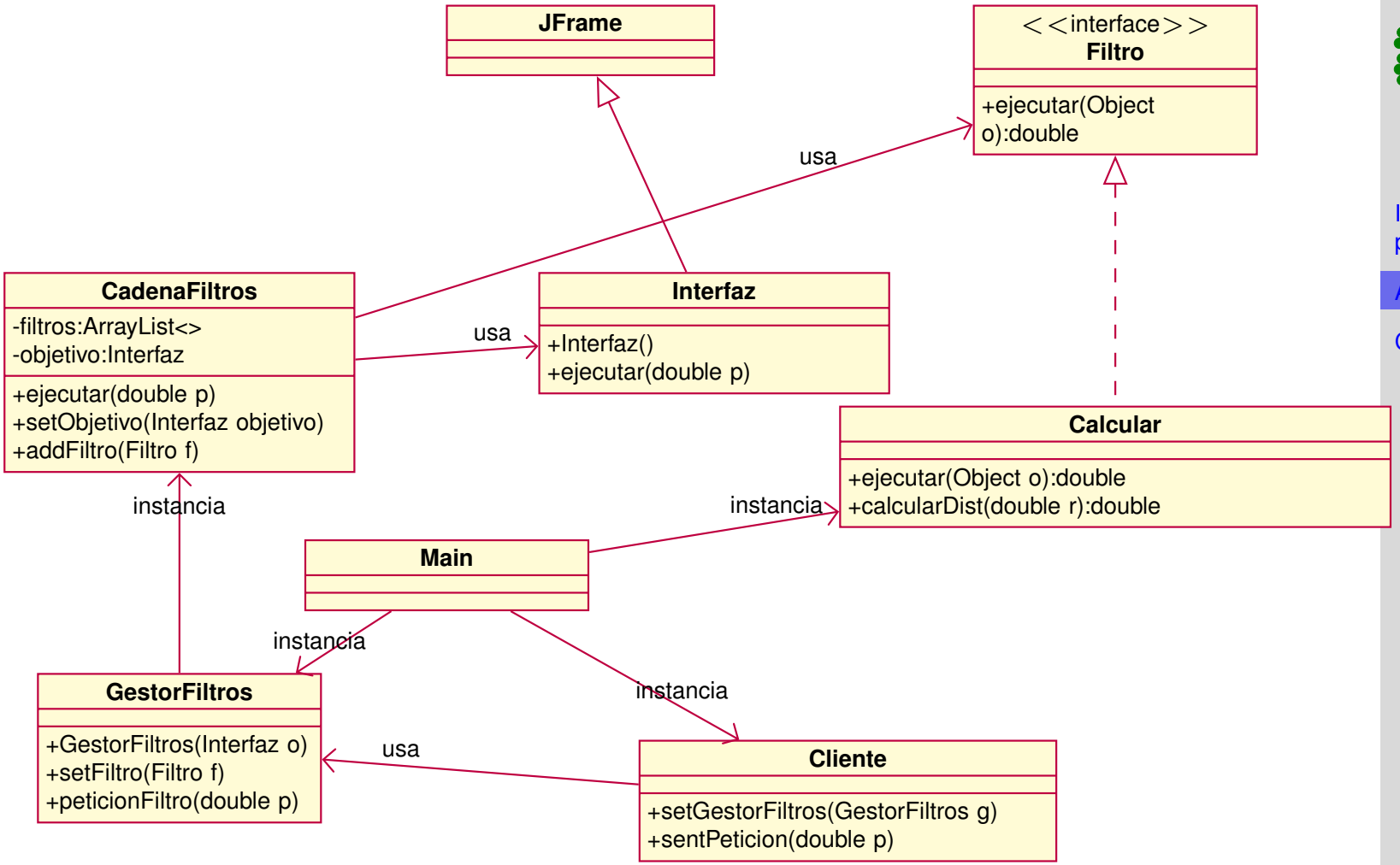


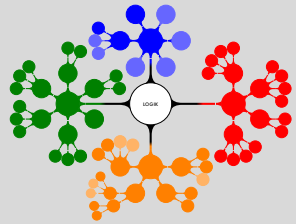
Entidades de diseño–II

- Clases de representación del patrón “Interceptor”:
  - `Interfaz Filtro`
  - `CadenaFiltros`
- *Objetivo*: se trata de un objeto que representa a una aplicación ya instalada en el marco de trabajo
  - Coincide con la clase `Interfaz`
- `GestorFiltros`
- `Cliente`
- Un programa de demostración (inclye `Main()`



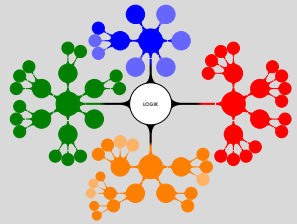
Diseño basado en el patrón “Interceptor” adaptado





“Interceptor” – Cadena de filtros

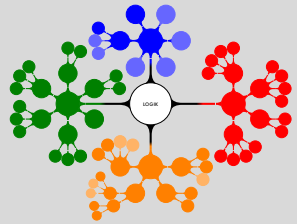
```
public class CadenaFiltros {
private// declarar: filtros es un ArrayList generico de
    elementos Filtro
private Interfaz objetivo;
public void addFiltro(Filtro filtro){
  filtros.add(filtro);
}
public void ejecutar(double peticion){
  for(Filtro filtro : filtros){
   System.out.println("Nueva velocidad (m/s) "+filtro.
       ejecutar(peticion));
  }
  objetivo.ejecutar(peticion);
}
public void setObjetivo(Interfaz objetivo){
   this.objetivo = objetivo;
}
}
```

“Interceptor” – Filtros individuales

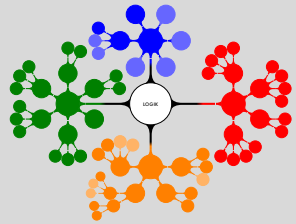
```
public class Calcular implements Filtro{
...
public double ejecutar(Object o) {
    double distancia= (double) o;
    double velocidad= distancia*3600/INTERVALO;
    revolAnt=revoluciones;
    return velocidad;
  }
}

public class CalcularDistancia implements Filtro{
...
 public double ejecutar(Object o){
```

“Interceptor”–Aplicación

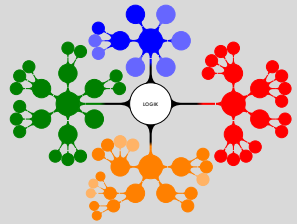
```
package Practica1;
public class DemoInterceptor {
 public static void main(String[] args) {
  GestorFiltros gestorFiltros = new GestorFiltros(new
      Interfaz());
  gestorFiltros.setFiltro(new Calcular());
  Cliente cliente = new Cliente();
  cliente.setGestorFiltros(gestorFiltros);
  cliente.enviarPeticion(500);//numero inicial de vueltas
      del eje
                }
}
```

Interfaz

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Interfaz extends JFrame{
   public Interfaz(){
           PanelBotones panel = new PanelBotones();
           setTitle("Practica-1.4");
           getContentPane().add(panel);
       panel.setPreferredSize(new Dimension(450,150));

       //terminar bien el programa
       this.addWindowListener (new WindowAdapter(){
         public void windowClosing(WindowEvent e){
          System.exit(0);
         }
       });
   }
   public void ejecutar(double peticion){
           this.pack();
           this.setVisible(true);
           System.out.println("Para un numero de vueltas
               iniciales del eje= "+peticion);
   }
}
```

PanelBotones

```
public class PanelBotones extends JPanel {
  private javax.swing.JButton BotonAcelerar;
  private javax.swing.JToggleButton BotonEncender;
  private javax.swing.JLabel EtiqMostrarEstado;
  public PanelBotones(){
     //Crear objetos-botones, etiquetas
     //Fijar el aspecto de los paneles con setLayout() y
         setBorder()
     //crear subpaneles: ... new JPanel()
         //Adaptar las etiquetas e incluirlas en los
             subpaneles
        //subpanel.add(EtiqMostrarEstado); this.add(
            subpanel2); ...
        //...lo mismo con los 2 botones
  }
  //Recogedor de eventos del boton Encender
  synchronized private void BotonEncenderActionPerformed(
      java.awt.event.ActionEvent evt){
    if (BotonEncender.isSelected()){
       //Programar el cambio de aspecto del botón
    }
    else{
         //Programar el cambio de aspecto del botón
    }
  }
  //Recogedor de eventos del boton Acelerar
```