

Prueba de Prácticas

Nombre:	
---------	--

22-06-2015

Enunciado del problema:

Se trata de programar un módulo del SCAV. Utilizando el patrón “Interceptor”¹ desarrollar un diagrama de clases y programar un proyecto Java de Eclipse para calcular la velocidad inicial de un vehículo a partir de un dato de entrada: *revoluciones del eje* y que posteriormente instale un manejador de eventos que *reaccione* cuando se pulsen cualquiera de los 2 botones: “Encender” (el motor del vehículo) y “Acelerar”. Para que el ejercicio sea considerado correcto hay que realizarlo de acuerdo con los siguientes requerimientos:

- Programar una clase anónima (`WindowAdapter()`) para terminar bien la ejecución de la clase **Interfaz**:


```
this.addWindowListener (new WindowAdapter(){
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
});
```
- Hay que programar los botones “Encender” y “Acelerar” y la etiqueta “APAGADO” / “ACELERANDO” dentro de un objeto panel de botones:


```
import java.awt.*;import javax.swing.BoxLayout;
import javax.swing.JPanel;import javax.swing.border.*;
public class PanelBotones extends JPanel {
    private javax.swing.JButton BotonAcelerar , BotonEncender;
    private javax.swing.JLabel EtiqMostrarEstado;
    public PanelBotones(){ ...}; // constructor
    synchronized private void
        BotonAcelerarActionPerformed(java.awt.event.ActionEvent evt){...};
    synchronized private void
        BotonEncenderActionPerformed(java.awt.event.ActionEvent evt){...};
}
```
- Utilizar de manera preferente elementos gráficos de la biblioteca `javax.swing.*`; (no usar AWT).
- Funcionamiento de los botones:
 - Inicialmente la etiqueta del panel principal mostrará el texto “APAGADO” y las etiquetas de los botones, el nombre de cada uno (ver figura 1(a))
 - El botón “Encender” será de selección de tipo conmutador `JToggleButton` cambiando de color y de texto (“Encender” / “Apagar”) cuando se pulsa (ver figura 1(b))
 - La pulsación del botón “Acelerando” cambia el texto de la etiqueta del panel principal a “ACELERANDO”, pero sólo si el motor está encendido; si no, no hace caso a la pulsación del usuario.
 - Si ahora se pulsa el botón que muestra ahora la etiqueta “Apagar”, la etiqueta del panel principal volverá a mostrar el texto inicial “APAGADO”.

¹ver descripción de este patrón al final del enunciado

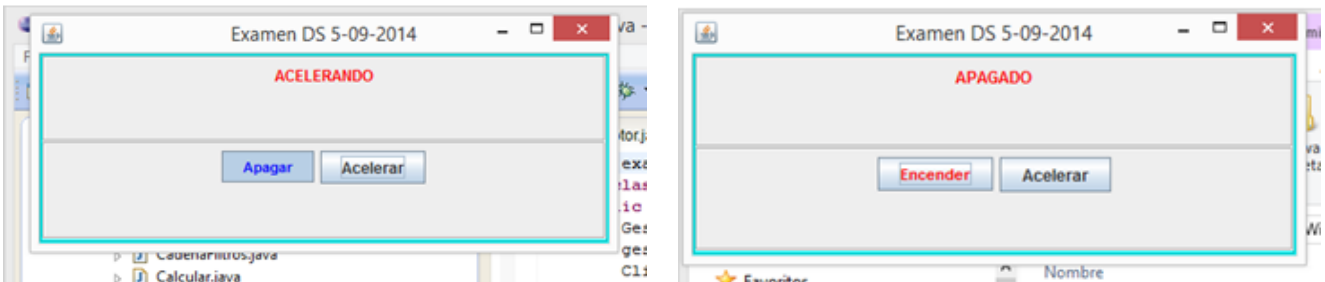


Tabla 1: Se muestran 2 instantes de la ejecución de la interfaz: (a) inicialmente; (b) después de pulsar el botón “Encender”

Descripción del patrón de diseño “Interceptor”:

Se trata de un patrón arquitectónico que permite la inclusión de nuevos servicios dentro de un marco de trabajo pre-existente de forma transparente para una aplicación *objetivo* (la que ya existe) y también para el sistema completo, que incluye a la aplicación anterior y a los servicios. Los nuevos servicios se activarán automáticamente desde la clase **Cliente** cuando se llama a su método `enviarPetición()` (`cliente.enviarPetición(500)`; en este ejercicio). Cada clase *filtro* se programa y su instancia se ejecuta cuando se produce y envía la petición desde el cliente, y antes de pasar tal petición a la clase *objetivo* (en nuestro caso esta clase será la clase **Interfaz**). Las clases que hay que programar para implementar correctamente el patrón “Interceptor” son las siguientes:

1. Interfaz **Filtro** que ha de implementar la clase *filtro específico*, en nuestro caso esta clase será la clase **Calcular** (velocidad).
2. **CadenaFiltros** que proporcionará varios filtros a la instancia de **GestorFiltros** y se encargará de ejecutar los filtros siguiendo el orden en el que fueron introducidos. Su código es similar a lo siguiente:

```
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;
    }
}
```

3. **Objetivo**: se trata de un objeto (que representa a una aplicación ya instalada en el marco de trabajo) que se encarga de procesar la petición que se envía desde el cliente. En nuestro caso será la clase **Interfaz**

4. **GestorFiltros**: crea la cadena de *filtros* y posee métodos para insertar los filtros en la cadena y provocar que cada uno ejecute la petición del cliente y también el *objetivo*.
5. Una instancia de **Cliente** se encarga de enviar la petición a la instancia de **Objetivo**

6. Por último, la aplicación que se ha de programar tendrá una clase principal simple de demostración del funcionamiento del patrón “Interceptor”, con un código similar a:

```
package examen1;
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
    }
}
```

7. Como *clases-filtro* se pueden programar 2 servicios para calcular la distancia recorrida (equivalente) al número de vueltas del eje y la velocidad actual, respectivamente:

```
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){
        double revoluciones= (double) o;
        double distancia= (revoluciones-revolAnt)*2*RADIO*3.1416;
        return distancia;
    }
}
```