

POA

Felipe Moreno, Augusto Pecho

July 4, 2016

Programación Orientada a Aspectos

Introducción

- Objetivos

Conceptos Basicos

- Aspect

- JoinPoint

- Advice

- PointCut

- Introduction

- Target

- Weaving

Diferencias con otros paradigmas

Programación Orientada a Objetos

Herramientas de desarrollo

JoinPoint

PointCut

Advices

Implementación

Introducción

La Programación Orientada a Aspectos es un paradigma de programación, se basa en subject-oriented programming, metaobject protocols y adaptative programming. Fue desarrollado por Gregor Kiczales y sus colegas de Xerox PARC los cuales tuvieron un concepto explícito de AOP el cual tuvo su primera implementación denominada AspectJ que es una extensión para Java en el 2001. El equipo de desarrollo de IBM diseñaron un lenguaje Hyper/J a su vez en el 2001.

Objetivos

El principal objetivo de la POA es la separación de las funcionalidades dentro del sistema:

- Por un lado funcionalidades comunes utilizadas a lo largo de la aplicación.
- Por otro lado, las funcionalidades propias de cada módulo.

Cada funcionalidad común se encapsulará en una entidad.

Conceptos Basicos

Aspect

es la combinación de el pointcut y un advice.

JoinPoint

un conjunto de joinPoints se denomina pintcut. es una especificación de donde, en el programa, el aspecto debe ser ejecutado.

Advice

es un código adicional que quieres añadir al modelo, es decir, código adicional (puede ser un log) que se aplica con una acción.

PointCut

Es una determinada linea en el programa donde sucede una acción.

Introduction

permite añadir métodos o atributos a clases ya existentes.

Target

es el objeto o instancia de una clase que funciona como advice.

Weaving

toma la instrucciones de las clases y aspectos y crean nuevas clases con los aspectos definidos, las instrucciones son conocidas como advice, usa los pointcuts y joinpoints.

Diferencias con otros paradigmas

Programación Orientada a Objetos

En la programación orientada a objetos los sistemas se modelan como un conjunto de objetos que interactúan entre sí, sin embargo, falla al modelar los conceptos que se entrecruzan.

Entonces, la diferencia radica en que mientras la programación orientada a aspectos se enfoca en los conceptos que se entrecruzan, la programación orientada a objetos se enfoca en los conceptos comunes.

Herramientas de desarrollo

- **AspectC++** es un compilador que permite desarrollar aspectos en C++.
- **AspectJ** es una extensión Java del proyecto Eclipse para ayudar en el desarrollo orientado a aspectos.
- **Aspect**, un módulo Perl disponible en CPAN para la Programación Orientada a Aspectos (en inglés).
- **PHP-AOP (AOP.io)** es una lib que proporciona todo el paradigma de la POA en PHP.
- **phpAspect** es una extensión PHP para implementar el paradigma de la POA, que, mediante árboles de decisión XML, realiza el weaving del software para ser ejecutado como PHP estándar.

- **FLOW3** es un framework MVC de PHP incluye un módulo para poder realizar Programación orientada a Aspectos en nuevos desarrollos.
- **AOP** con SpringFramework 2.5 es un Framework de Java que permite programar en el paradigma de Aspectos utilizando Anotación Java.
- **Aspyct** AOP es un módulo de Python que permite incluir Programación orientada a Aspectos a programas ya existentes escritos en Python o a nuevos desarrollos.

JoinPoint

- Un punto de unión es un punto bien definido en el flujo del programa
 - Ejecutar algún código ("advice") cada vez que se alcanza un JoinPoint.
 - Evitar saturar el código con indicadores explícitos diciendo "Este es un punto de unión".
 - AspectJ proporciona una sintaxis para indicar que estos puntos se unen "desde fuera" al código real.
- Un JoinPoint es un punto en el flujo del programa donde "sucede algo"
- Ejemplos

- Llamada a un método
- Excepción
- Acceso a una variable
- Instancias de un objeto
- Referencia a un objeto

PointCut

- Las definiciones de PointCut consisten en un lado izquierdo y un lado derecho, separados por dos puntos.
- El lado izquierdo está formado por el nombre del PointCut y sus parámetros (es decir, los datos disponibles cuando los eventos ocurra)
- El lado derecho consiste en el propio PointCut.

```
1 pointcut callSayHello():  
2     call(* HelloAspectJDemo.sayHello());
```

– El nombre de este PointCut es 'callSayHello'.

- El pointcut no tiene parámetros.
- El pointcut en sí mismo es '(* HelloAspectJDemo.sayHello ())'.
- El pointcut se refiere a cualquier momento en que se realiza la llamada al método 'HelloAspectJDemo.sayHello()'.

Advices

- BEFORE

```
1  before() : callSayHello() {  
2      System.out.println("Before execution");  
3  }
```

- AFTER

```
1  after() : callSayHello() {  
2      System.out.println("After execution");  
3  }
```

Implementación

- Se usó AspectJ Development Tool (AJDT) 2.2.4. Entorno de desarrollo Eclipse Mars.2 (versión 4.5.2). Base de Datos DB2 Advice Enterprise Edition versión 9.7.10.
- Creación de nuestro aspecto:

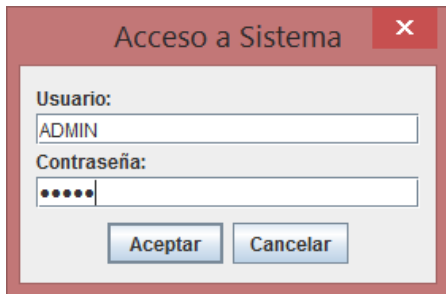
```
1  public aspect aopFile{  
2      .....  
3      .....  
4  }
```


- Estructura de los puntos de corte del programa:

```
1 pointcut checkLogeo(Ventana v):  
2     call(* *.checkLog(..)) && target(v);
```

```
1 pointcut addUser(Ventana v):  
2     call(* *.agrega_usuario()) && target(v);
```

```
1 pointcut showUsers(Ventana v):  
2     call(* *.muestra_usuarios()) && target(v  
    );
```



A screenshot of a Windows-style dialog box titled "Acceso a Sistema" (System Access). The dialog has a red title bar with a close button (X) in the top right corner. Inside the dialog, there are two input fields. The first is labeled "Usuario:" (User:) and contains the text "ADMIN". The second is labeled "Contraseña:" (Password:) and contains five dots, indicating a masked password. Below the input fields are two buttons: "Aceptar" (Accept) and "Cancelar" (Cancel).

Figure 1: Ingreso al Sistema

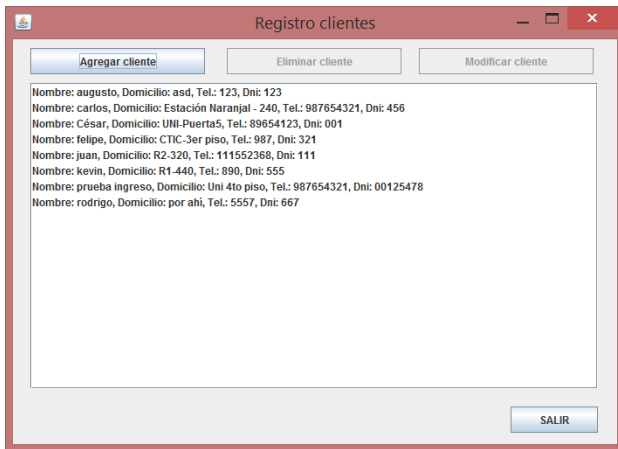


Figure 2: Datos del Sistema

A screenshot of a Windows-style dialog box titled "Introduzca datos". The dialog box has a red title bar with a close button (X) in the top right corner. The main area is light gray and contains four input fields with labels to their left: "Nombre:", "Dirección:", "Telefono:", and "Dni:". The "Nombre:" field contains the text "trap". Below the input fields are two buttons: "Aceptar" and "Cancelar".

Figure 3: Agregar Registro

```
1 pointcut modUser(Ventana v):  
2     call(* *.modifica_usuario()) && target(v  
    );
```

```
1 pointcut delUser(Ventana v):  
2     call(* *.elimina_usuario()) && target(v  
    );
```

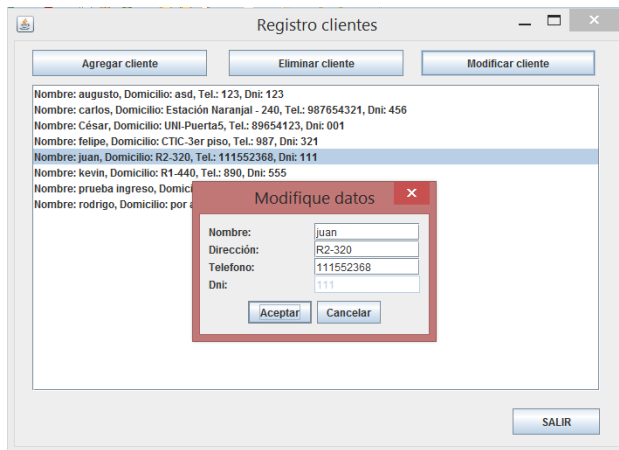


Figure 4: Modificar Registro

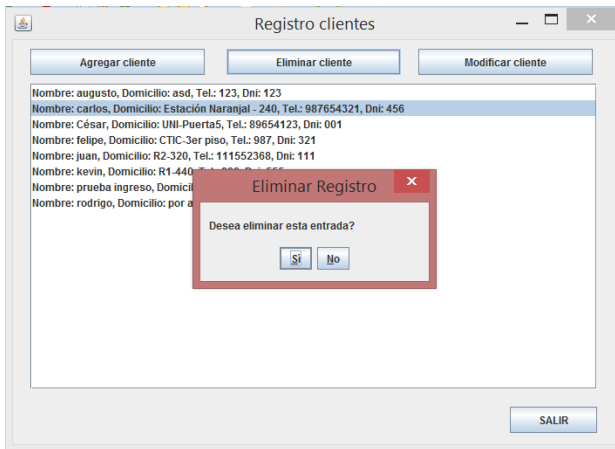


Figure 5: Eliminar Registro

- Uso de BEFORE y AFTER:

```
1  before(Ventana v): checkLogeo(v){
2      System.out.println("");
3      System.out.println("Verificando
        informacion...");
4      while(val==false){
5          if(JOptionPane.showConfirmDialog
            (v,v.panel,"Acceso a Sistema"
6              ,
                JOptionPane.OK_CANCEL_OPTION
                    ,JOptionPane.
                        PLAIN_MESSAGE)==
                            JOptionPane.OK_OPTION){
```



```
7         user = v.campoUser.  
            getText();  
8     pass = v.campoPass.getPassword();  
9     }  
10    else{  
11        System.exit(0);  
12    }  
13    try {  
14        val = con.check_log(user  
            , String.valueOf(pass  
            ));  
15    } catch (SQLException e) {  
16        e.printStackTrace();  
17    } catch (Exception e) {
```

```
18         e.printStackTrace();
19     }
20 }
21 JOptionPane.showMessageDialog(v, "Acceso
22     CORRECTO");
23 }
```

```
1  after(Ventana v): showUsers(v){
2      try {
3          con.showClient();
4      } catch (SQLException e) {
5          e.printStackTrace();
6      } catch (Exception e) {
7          e.printStackTrace();
8      }
9      clientes = con.get_list();
10     v.numPer = clientes.size();
11     if(clientes!=null){
12         for(int i=0; i<clientes.size();
            i++){
```

```
13         Cliente cl = (Cliente)
14             clientes.get(i);
15         System.out.println(cl.
16             toString());
17         v.modelo.add(i,cl.toString()
18             );
19     }
20 }
21 else{
22     JOptionPane.showMessageDialog(v,
23         "NO HAY USUARIOS REGISTRADOS
24     ");
25 }
```

- Creación de la Interfaz:

```
1  public class Ventana extends JFrame implements
    ActionListener, ListSelectionListener{
2      .....
3      .....
4      public void actionPerformed(ActionEvent e) {
5          //Evento de cierre de programa
6          if(e.getSource()==salir){
7              System.exit(0);
8          }
9
10         //Evento que gestiona el ingreso de
            datos
```

```
11         if (e.getSource()==cliente) {
12             this.agrega_usuario();
13         }
14
15         //Evento que gestiona la modificacion de
16         //datos
17         if (e.getSource()==modificar) {
18             this.modifica_usuario();
19         }
20
21         //Evento que gestiona la eliminacion de
22         //una entrada
23         if (e.getSource()==eliminar) {
24             this.elimina_usuario();
25         }
26     }
```

```
23         }  
24     } //fin de la clase 'actionPerformed'  
25         .....  
26         .....  
27 }
```

- Creación de la Clase de Prueba:

```
1  public class Test {  
2      public static void main(String[] args){  
3          Ventana window = new Ventana();  
4          window.setSize(710,510);  
5          window.setVisible(true);  
6          window.setTitle("Registro clientes");  
7          window.setDefaultCloseOperation(JFrame.  
            EXIT_ON_CLOSE);  
8      }  
9  }
```


References

- [1] <http://c2.com/cgi/wiki?AspectOrientedProgramming>
- [2] Instalación de AJDT para Eclipse: <http://o7planning.org/en/10115/install-aspectj-development-tools-into-eclipse>

Muchas Gracias