Tarea No. 15. Patrón Front Controller y CDI

Barrera Pérez Carlos Tonatihu
Profesor: José Asunción Enríquez Zárate
Web Application Development
Grupo: 3CM9

28 de abril de 2019

1. Desarrollo

1.1. Patrón front controller

El patrón de diseño front controller significa que todas las peticiones que vienen por un recurso en una aplicación serán manejadas por un solo manejador y después serán enviadas al respectivo manejador dependiendo del tipo de solicitud. El front controller puede usar auxiliares para lograr el mecanismo de envío.

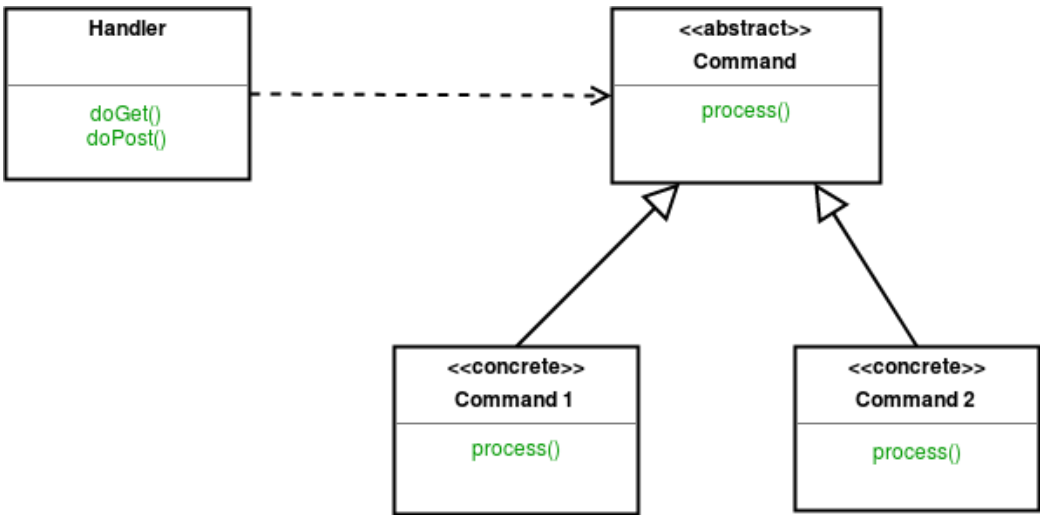

Figura 1: Patrón Front Controller

1.1.1. Elementos del patrón

- Controlador. El controlador es el punto de contacto inicial para el manejo de las peticiones en el sistema. El controlador delega a un auxiliar para completar la autenticación y autorización de un usuario o para inicializar la recuperación de contactos.
- Vista. Una vista representa y despliega información al cliente. La vista recupera información del modelo. Los auxiliares apoyan a las vistas mediante el encapsulado y adaptación del subyacente modelo de datos para su uso en la visualización.
- Despachador. Es el responsable del manejo de las vistas y la navegación, manejando la elección de la siguiente vista a presentar al usuario, y proveyendo el mecanismo para el control de vectores de este recurso.

- Auxiliar. Es el responsable de ayudar a la vista o el controlador a completar su proceso. Así, los auxiliares tienen numerosas responsabilidades, incluyendo el recolectado de datos requeridos por la vista y ordenando este modelo intermedio, en cuyo caso el auxiliar es algunas veces referido como un bean de valor.

1.1.2. Ventajas y desventajas

Las ventajas que presenta este patrón son las siguientes:

- Control centralizado. Este patrón maneja todas las peticiones a la aplicación. Esta implementación centralizada del control evita el uso de múltiples controladores, es deseable para hacer cumplir políticas de toda la aplicación, como el seguimiento y la seguridad de los usuarios.
- Seguridad de hilos. Un nuevo objeto de mando surge cada que se recibe una solicitud y estos objetos no están diseñados para ser seguros para subprocesos. Por lo tanto, sera seguro en las clases de mando. Aunque la seguridad no está garantizada cuando se reúnen los problemas de subprocesos, los códigos que actúan con el mando aun son seguros para subprocesos.

Sin embargo, se presentan las siguientes desventajas:

- No es posible escalar la aplicación utilizando este patrón.
- El desempeño es mejor si se lidia solo con una única petición.

1.1.3. Ejemplo

```
1  class TeacherView {
2      public void display() {
3          System.out.println("Teacher View");
4      }
5  }
6
7  class StudentView {
8      public void display() {
9          System.out.println("Student View");
10     }
11 }
12
```

```
13 class Dispatching {
14     private StudentView studentView;
15     private TeacherView teacherView;
16 
17     public Dispatching() {
18         studentView = new StudentView();
19         teacherView = new TeacherView();
20     }
21 
22     public void dispatch(String request) {
23         if(request.equalsIgnoreCase("Student")) {
24             studentView.display();
25         } else {
26             teacherView.display();
27         }
28     }
29 }
30 
31 class FrontController {
32     private Dispatching Dispatching;
33 
34     public FrontController() {
35         Dispatching = new Dispatching();
36     }
37 
38     private boolean isAuthenticUser() {
39         System.out.println("Authentication successfull.");
40         return true;
41     }
42 
43     private void trackRequest(String request) {
44         System.out.println("Requested View: " + request);
45     }
46 
47     public void dispatchRequest(String request) {
48         trackRequest(request);
49 
50         if(isAuthenticUser()) {
51             Dispatching.dispatch(request);
52         }
53     }
54 }
55 
```

```
56 class FrontControllerPattern {
57     public static void main(String[] args) {
58         FrontController frontController = new
    FrontController();
59         frontController.dispatchRequest("Teacher");
60         frontController.dispatchRequest("Student");
61     }
62 }
```

Al ejecutar este código el resultado es el siguiente:

```
Requested View: Teacher
Authentication successfull.
Teacher View
Requested View: Student
Authentication successful.
Student View
```

1.2. CDI

Contexts and Dependency Injection para JavaEE (CDI) 1.0 fue introducida como parte de la plataforma Java EE 6, y rápidamente se convirtió en una de los mas importantes y populares componentes de la plataforma.

CDI define un poderoso conjunto de servicios complementarios que ayudan a mejorar la estructura del código de la aplicación.

- Un ciclo de vida bien definido para objetos con estado vinculados a contextos de ciclo de vida, donde el conjunto de contextos es extensible.
- Un mecanismo de inyección de dependencias sofisticado y seguro de tipos que incluye la capacidad de seleccionar las dependencias en tiempo de desarrollo o de despliegue sin una configuración detallada.
- Compatibilidad con la modularidad de Java EE y la arquitectura de componentes de Java EE. La estructura modular de una aplicación java EE se tiene en cuenta al resolver las dependencias entre los componentes de java EE.
- Integración con EL, permitiendo que cualquier objeto contextual sea usado directamente dentro de una pagina JSP o JSF.
- Capacidad de decorar objetos inyectados.

- Capacidad de asociar interceptores a objetos vía enlaces de interceptor seguros.
- Un modelo de notificación de eventos.

2. Conclusiones

Es evidente la gran cantidad de temas que se tocan cuando se trata de trabajar con Java, en este caso se presentan el CDI y el patrón front controller el cual no es exclusivo de Java pero que su implementación en este lenguaje de programación puede llegar a ser más intuitiva que en otros lenguajes. Además, es claro que estos dos conceptos no son exclusivos del desarrollo de aplicaciones web ya que se pueden implementar en desarrollo de aplicaciones móviles o en aplicaciones nativas. Es por esto que el conocerlos nos brinda más herramientas a la hora de enfrentar un problema y proponer una solución a este.