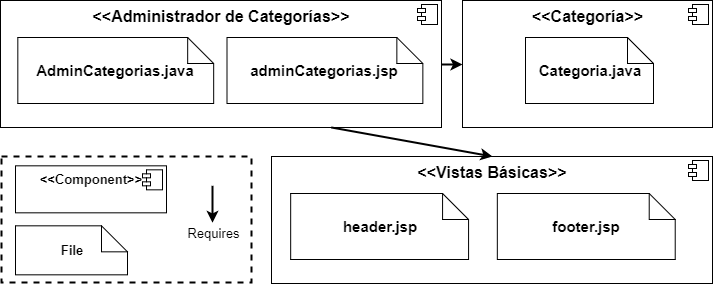
### Ejemplo base

Para trabajar las diferentes metodologías de implementación de componentes según los diferentes paradigmas de programación, utilizaremos el siguiente ejemplo.

**“Administración de categorías” y todas sus dependencias**.

Se basa en que, en una línea de tiendas virtuales, los productos deben pertenecer a categorías (“ropa hombre”, “electrodomesticos”, “televisores”, “etc”). Ahora, veamos el diagrama de componentes de esta “parte” de la línea de tiendas virtuales que presentamos en la Figura 1.



**Figura 1.** Parte del diagrama de componentes, correspondiente al componente

“Administrador de categorías” y sus dependencias.

### A continuación, describiremos el proceso de implementación de estos componentes, utilizando una arquitectura MVC, o en su defecto, una por capas que se acople al paradigma de programación DOP.

### Programación orientada a características y a deltas

La programación orientada a características (FOP por su sigla en inglés) es una metodología de programación, que pretende que la mejor manera de eliminar la redundancia y de aumentar la eficiencia de los programas computacionales es mediante la creación de un gran número de características de grano fino. Luego, esas pequeñas características son unidas entre sí para conformar las funciones, métodos o procedimientos (a las cuales nos referiremos como funciones de ahora en adelante) que se encargan de dotar de funcionalidad a un sistema (Lopez-Herrejon, 2005).

A diferencia de la POO, la FOP se focaliza en las características de los sistemas y no en los objetos que los componen. En otros términos, la FOP puede ser vista como un paradigma de desarrollo orientado a módulos pequeños, bien definidos y de propósito general con los cuales se conformarán las posibles funciones de los sistemas.

La implementación de componentes reutilizables basada en FOP, propone un diseño basado en capas, en la cual un programa o aplicación corresponde a una superposición de diferentes capas funcionales donde cada capa añade una funcionalidad al sistema.

Una tecnología que permite trabajar bajo este enfoque es DeltaJ (Koscielny *et al.*, 2014). DeltaJ se crea como herramienta para soportar programación orientada a deltas o DOP por su sigla en inglés. DOP es una vertiente de FOP que permite suavizar las restricciones de FOP y proporcionar un lenguaje de programación expresivo y flexible para LPS (Schaefer *et al.*, 2010). Esta tecnología permite que cada capa sea vista como una unidad atómica llamada delta; un delta por lo general no es funcional por sí mismo, pero unido con otro delta adquiere funcionalidad. Es responsabilidad del desarrollador determinar el nivel de detalle de cada delta teniendo en cuenta que entre más específicos o atómicos sean los deltas, mayor cantidad de deltas tendrá la aplicación y también se posibilitará un mayor reuso. Por el contrario, entre más grandes sean los deltas, menos cantidad de deltas tendrá la aplicación, por lo que cada delta podrá ser funcional por sí mismo pero se reducirá la posibilidad de reuso.

#### Proceso

En DOP todos los archivos Java se definen como deltas (tal como veremos a continuación).

En el caso de nuestra línea de tiendas virtuales, los componentes tipo vista se implementan como archivos con lenguaje HTML o correspondientes.

Finalmente, en este caso y para tener un panorama más detallado de estos tipos de aplicaciones, implementaremos a continuación todos los componentes de la Figura 1.

**Ejemplo:**

#### Implementación de deltas en Java

**Herramientas.** Para implementar DOP en Java se requiere:

* **Eclipse:** entorno de desarrollo de software Web.
* **DeltaJ 1.5:** plugin para Eclipse, brinda las herramientas para utilizar el paradigma de programación orientada a deltas.
* **XText:** requisito de DeltaJ, este último implementa un compilador para los archivos “.deltaj”.
* **El componente *JSTL Library*** de Java EE; este componente extiende la funcionalidad de las JavaServer Pages (JSP) permitiendo utilizar unas etiquetas específicas para el desarrollo de páginas web dinámicas.
* Un servidor web escrito en Java que soporte el uso de servlets y JSPs, por ejemplo **Apache Tomcat.** El servidor web debe brindar soporte para aplicaciones JEE, y será usado dentro del proyecto para desplegar los componentes desarrollados.

**Desarrollo:**

1. Utilizando Eclipse creamos un nuevo proyecto tipo “DeltaJ Project” llamado “TiendaVirtual”. Este será el proyecto donde almacenaremos todos los componentes reutilizables. En Eclipse deberemos completar la siguiente información:

***SPL Name*:** TiendaVirtual  
***Features*:** Categoria, AdminCategorias  
***Deltas*:** dCategoria, dAdminCategorias, dModAdminCat

1. Modificamos el archivo TiendaVirtual.spl ubicado en la ruta spl-info/ TiendaVirtual.spl con el siguiente código:

|  |
| --- |
| SPL TiendaVirtual {  Features = {Categoria, AdminCategorias}  Deltas = {dCategoria, dAdminCategorias, dModAdminCat}  Constraints {  }  Partitions {  {dAdminCategorias} when (AdminCategorias);  {dCategoria, dModAdminCat} when (Categoria);  }  Products {  complete = { Categoria, AdminCategorias};  }  } |

Este archivo permite definir:  
 **- *Features*:** las características o componentes del proyecto.  
 **- *Deltas*:** los módulos deltas de la aplicación (generalmente uno por característica).

**- *Constraints*:** la cual permite declarar dependencias entre las características.

**- *Partitions*:** permite relacionar una característica con su correspondiente implementación (DeltaJ). Si una característica modifica otras caracteristicas se debe colocar la lista de dependencias completa.

**- *Products*:** permite generar productos mediante la selección de Features.

**Iniciamos la implementación del archivo “dCategoria.deltaj”.**

|  |
| --- |
| delta dCategoria {  adds{  package org.pl.eshopping.models;  import java.util.ArrayList;  import java.util.List;    public class Categoria {  public Integer id;  public String nombre;  public String descripcion;    public Categoria(Integer i, String n, String d){  this.id=i;  this.nombre=n;  this.descripcion=d;  }    public static List<Categoria> categorias = new ArrayList<Categoria>() {  {  add(new Categoria(102, "Ropa", "ropa en general"));  add(new Categoria(103, "Gorras", "gorras en general"));  add(new Categoria(104, "Televisores", "Tvs en general"));  }  };    public String getNombre() { return nombre; }  public void setNombre(String nombre) { this.nombre = nombre; }  public Integer getId() { return id; }  public void setId(Integer id) { this.id = id; }  public String getDescripcion() { return descripcion; }  public void setDescripcion(String descripcion) { this.descripcion = descripcion; }  }  }  } |

**Iniciamos la implementación del archivo “dAdminCategorias.deltaj”.**

|  |
| --- |
| delta dAdminCategorias {  adds{  package org.pl.eshopping.controllers;    import java.io.IOException;  import java.util.List;  import javax.servlet.RequestDispatcher;  import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    @WebServlet("/AdminCategorias")  public class AdminCategorias extends HttpServlet {             List<Categoria> categorias = null;    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {    RequestDispatcher view = request.getRequestDispatcher("admincategorias.jsp");  request.setAttribute("categorias", categorias);  view.forward(request, response);  }  }    }  } |

**Iniciamos la implementación del archivo “dModAdminCat.deltaj”.**

|  |
| --- |
| delta dModAdminCat {  modifies org.pl.eshopping.controllers.AdminCategorias{  adds import org.pl.eshopping.models.Categoria;  removes categorias;  adds List<Categoria> categorias = Categoria.categorias;  }  } |

1. Luego de terminar la creación de los deltas, nos paramos en el archivo TiendaVirtual.spl y realizamos una pequeña modificación (por ejemplo, dejamos un espacio en alguna parte), guardamos el archivo y nos aparecerá una opción para generar un nuevo producto. Generaremos la única opción disponible lo cual crea una carpeta llamada src-gen con los deltas (ahora archivos .java) integrados.
2. DeltaJ no nos permite diseñar las vistas como deltas. Por lo tanto, debemos crear un directorio aparte el cual podemos llamar “Vistas” y crear los siguientes archivos.

**Iniciamos la implementación del archivo “admincategorias.jsp”.**

1. Creamos la vista admincategorias.jsp con el siguiente contenido:

|  |
| --- |
| <%@ include file="header.jsp" %>  <div class="main">  <div class="container" style="margin-top: 60px;">         <h1>Listado de Categorias</h1>  </div>  </div>  <div class="container">     <table>     <tr>         <th width="80">ID</th>         <th width="120">Nombre</th>         <th width="120">Descripcion</th>     </tr>     <c:forEach items="${categorias}" var="cat">         <tr>             <td>${cat.getId()}</td>             <td>${cat.getNombre()}</td>             <td>${cat.getDescripcion()}</td>         </tr>     </c:forEach>  </table>  </div>  <%@ include file="footer.jsp" %> |

**Iniciamos la implementación del archivo “header.jsp”.**

1. Creamos la vista header.jsp con el siguiente contenido:

|  |
| --- |
| <%@ page contentType="text/html" pageEncoding="UTF-8"%>  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  <html>  <head>   <meta charset="utf-8">   <title>${title}</title>   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">  </head>  <body>  <nav class="navbar navbar-inverse navbar-fixed-top">       <div class="container">         <div id="navbar">           <ul class="nav navbar-nav">             </ul>         </div>       </div>     </nav> |

**Iniciamos la implementación del archivo “footer.jsp”.**

1. Creamos la vista footer.jsp con el siguiente contenido:

|  |
| --- |
| <div class="container">  <hr />  <footer>  <p>© By MyCompany</p>  </footer>  </div>  </body> |

**Modificación:**

Ahora modificaremos los componentes anteriormente desarrollados para permitir que se conecten adecuadamente. En este caso y para este ejercicio solo debemos realizar modificaciones en las vistas; ya que DeltaJ nos permitió realizar modificaciones a los componentes tipo Java (tal como se vio en el delta dModAdminCat.deltaj).

1. Modificamos el header.jsp y dentro de las etiquetas <ul></ul> agregamos la siguiente línea.

|  |
| --- |
| <li><a href="<c:url value='AdminCategorias'/>">Admin Categorias</a></li> |

**Integración:**

Para ver los componentes integrados debemos:

1. Utilizando Eclipse creamos un nuevo proyecto tipo “Dynamic Web Project” llamado “DopEShopping” y utilizamos Apache Tomcat como ejecutor.
2. Importamos el archivo .jar de la librería JSTL en la ruta WebContent/WEB-INF/lib/
3. Incluimos las vistas anteriormente creadas en la ruta WebContent/
4. DeltaJ generó unos archivos .java en la carpeta src-gen; movemos esos archivos (Categoria.Java y AdminCategorias.java) a sus ubicaciones correspondientes (dentro de la carpeta src/). Para este caso sugerimos revisar muy bien la definición de los paquetes (cuando se realizan importaciones) ya que DeltaJ añade algunos directorios que pueden ocasionar errores.

**Finalmente:**

Corremos Apache Tomcat y nos debería salir algo como lo siguiente:



**Para más información y ejemplos revisa los siguientes enlaces:**

* <https://pdfs.semanticscholar.org/8e7b/58708691b44f62c01556a5a04dcb03788fc0.pdf>
* <https://www.isf.cs.tu-bs.de/cms/research/deltas/example.html>