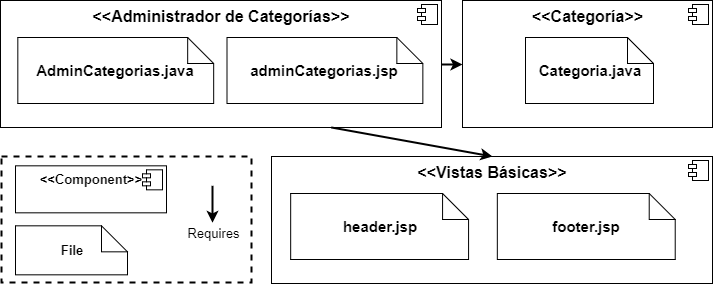
### Ejemplo base

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 POO.

### Programación orientada a objetos

La programación orientada a objetos (POO) se ha convertido en el paradigma principal de programación desde mediados de la década de los 90, cuando los lenguajes de programación que la soportaban se volvieron ampliamente disponibles. En la POO los problemas del mundo real se descomponen en objetos, los cuales abstraen el comportamiento y los datos en una sola unidad. La reutilización de software se basa en la POO, ya que esta proporciona un diseño y lenguaje que fomenta la modularidad, la encapsulación, la herencia y el polimorfismo.

Aunque la POO ha tenido éxito en el modelado y la implementación de sistemas de software complejos, también ha mostrado varias limitaciones con respecto, por ejemplo, al desarrollo de software basado en componentes. En particular, cuando se usa la POO para implementar los componentes de dominio de una línea de productos de software, los programadores se ven confrontados a problemas de mantenimiento del código de las aplicaciones debido a que con la POO es difícil separar las aplicaciones en módulos o componentes independientes. Esto implica que se gaste mucho tiempo en tareas de mantenimiento y de aumentación.

#### Tipos de requisitos que se pueden implementar con programación orientada a objetos

La POO es útil para implementar tanto requisitos funcionales como no funcionales. Sin embargo, como se discutió anteriormente, la modularización en la POO requiere mucho esfuerzo y en algunos casos los componentes que se implementan usando este paradigma no se pueden desacoplar completamente.

#### Proceso

La implementación de LPS usando la POO se hace bajo los enfoques de agrupación de funcionalidades en clases y de repartición de las clases por capas. Para implementar estos dos enfoques se toma como base el diagrama de componentes de la LPS. En este caso los componentes se implementarán mediante clases.

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 objetos en Java

**Herramientas:** para implementar objetos en Java se requiere:

* Un entorno de desarrollo web, por ejemplo **Eclipse**.
* 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.
* **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.

**Desarrollo:**

1. Utilizando Eclipse creamos un nuevo proyecto tipo “Dynamic Web Project” llamado “PooEShopping” y utilizamos Apache Tomcat como ejecutor. Este será el proyecto donde almacenaremos todos los componentes reutilizables.
2. Importamos el archivo .jar de la librería JSTL en la ruta WebContent/WEB-INF/lib/

**Iniciamos la implementación del archivo “Categoria.java”.**

1. Creamos el paquete org.pl.eshopping.models
2. Creamos la clase Categoria.java con el siguiente contenido y la ponemos en el paquete creado anteriormente:

|  |
| --- |
| 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 “AdminCategorias.java”.**

1. Creamos el paquete org.pl.eshopping.controllers
2. Creamos la clase AdminCategorias.java con el siguiente contenido y la ponemos en el paquete creado anteriormente:

|  |
| --- |
| package org.pl.eshopping.controllers;  import java.io.IOException;  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 “admincategorias.jsp”.**

1. Nos ubicamos en la ruta WebContent/ y 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. Nos ubicamos en la ruta WebContent/ y 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. Nos ubicamos en la ruta WebContent/ y 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. Todo este proceso se debe realizar de manera manual.

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> |

1. Modificamos la clase AdminCategorias.java y agregamos la siguiente importación

|  |
| --- |
| import org.pl.eshopping.models.Categoria; |

1. Modificamos la clase AdminCategorias.java y reemplazamos la siguiente línea

|  |
| --- |
| List<Categoria> categorias = null; |

Por:

|  |
| --- |
| List<Categoria> categorias = Categoria.categorias; |

**Finalmente:**

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

