

# JSP

## PARTE 3

### Controlador (Servlets)

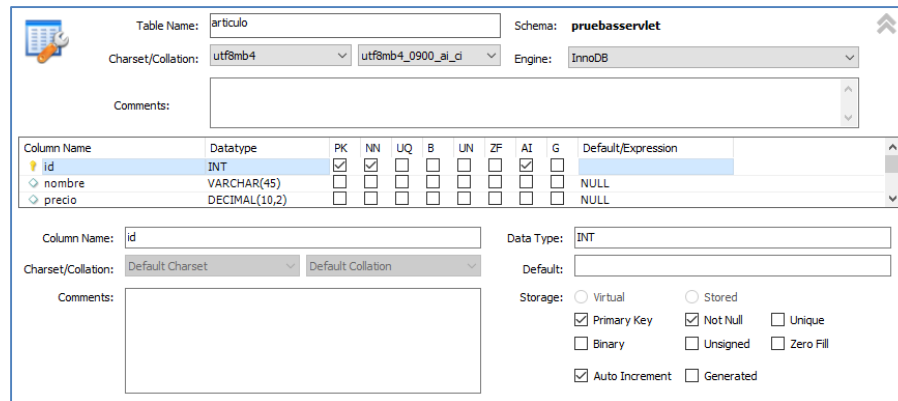
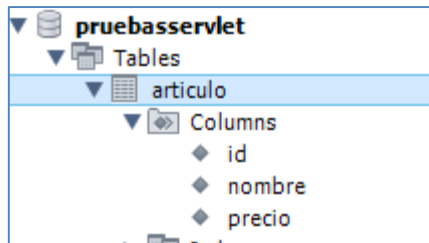
Con Apache Tomcat  
(Servidor Web y JSP)



## Antes de nada creamos un nuevo esquema en MySQL con una entidad

### Creación de esquema

- Creamos un esquema llamado **pruebasservlet**
- Dentro del esquema creamos una Tabla/Entidad llamada **articulo** con los siguientes atributos (id,nombre,precio)



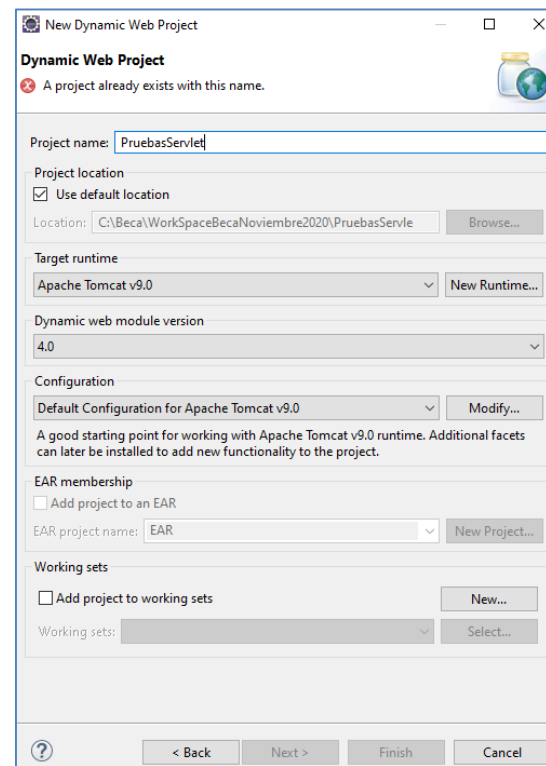
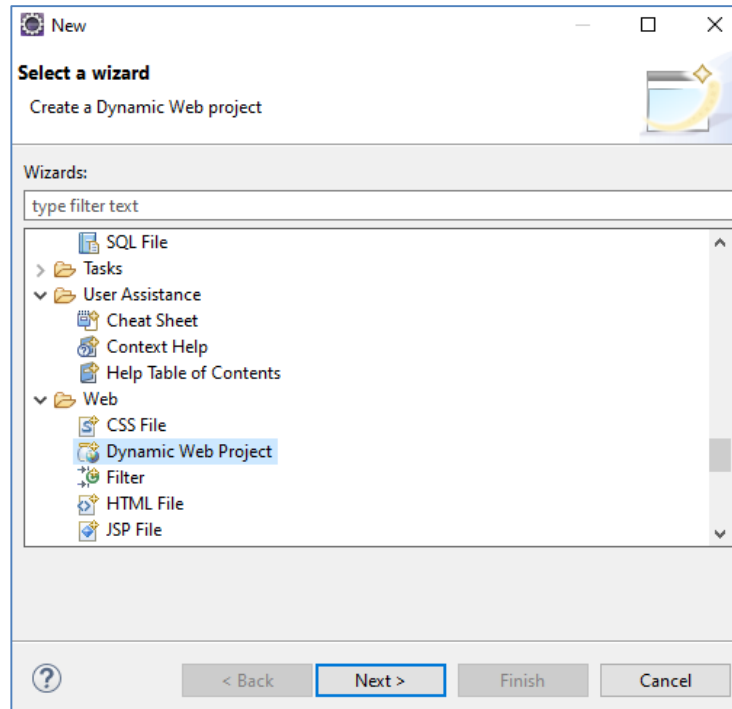


JSP

Servlets

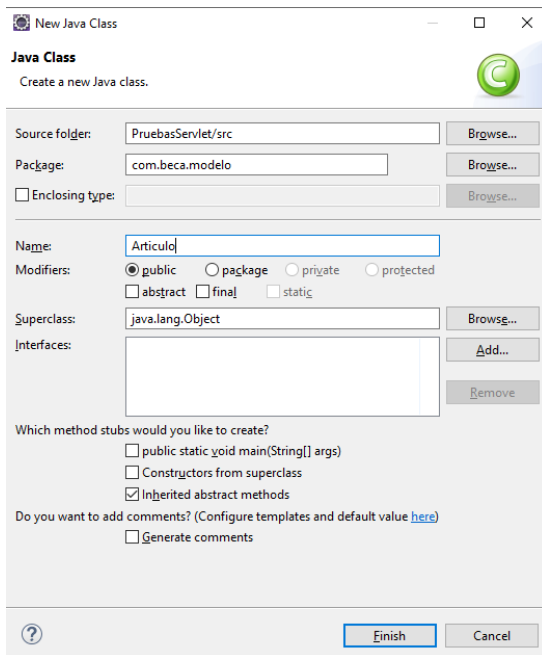
## Creamos un proyecto Web Dinámico

### Creación de proyecto Web Dinámico



## Creamos una clase Java Bean para el Modelo

### Servlets para MVC



New Java Class

Create a new Java class.

Source folder: PruebasServlet/src Browse...

Package: com.beca.modelo Browse...

☐ Enclosing type: Browse...

Name: Articulo

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

Finish Cancel

```
package com.beca.modelo;

public class Articulo {

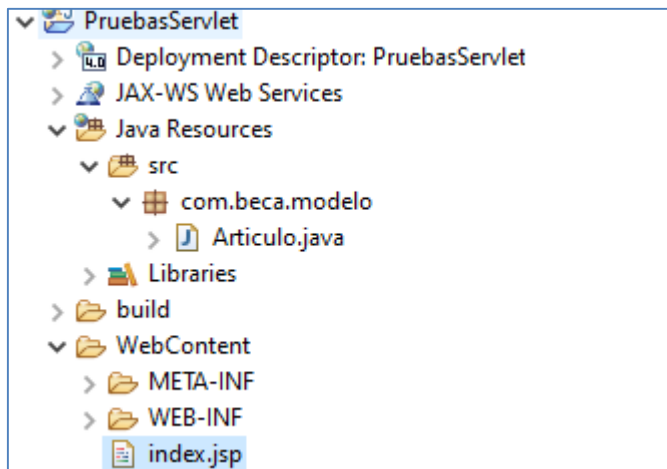
    Integer id;
    String nombre;
    Double precio;

    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public Double getPrecio() {
        return precio;
    }
    public void setPrecio(Double precio) {
        this.precio = precio;
    }
}
```



## Creamos una nueva página index.jsp e instanciamos el Bean del Modelo

### Probando el Bean del Modelo

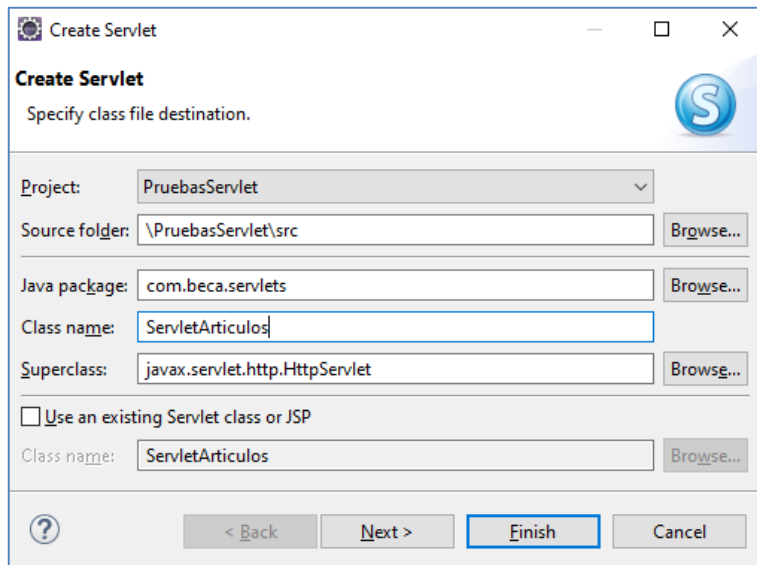


```
index.jsp
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2   pageEncoding="ISO-8859-1" isELIgnored="false"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="ISO-8859-1">
7 <title>Insert title here</title>
8 </head>
9 <body>
10
11 <jsp:useBean id="articulo1" class="com.beca.modelo.Articulo"/>
12 <jsp:setProperty name="articulo1" property="id" value="1"/>
13 <jsp:setProperty name="articulo1" property="nombre" value="El Artículo1"/>
14 <jsp:setProperty name="articulo1" property="precio" value="20.41"/>
15
16
17
18 <b>Artículo Id :</b>${articulo1.id}<br>
19 <b>Nombre :</b>${articulo1.nombre}<br>
20 <b>Precio :</b>${articulo1.precio}<br>
21
22 </body>
23 </html>
```

## Creación de un Servlet

### Servlets para MVC

- Una vez configurado lo anterior, vamos a crear nuestro Servlet que haga de controlador de las operaciones realizadas sobre la entidad **Articulo** (Botón derecho en el proyecto y **New Servlet**, le llamamos **ServletArticulos**)



**Create Servlet**

Specify class file destination.

Project: PruebasServlet

Source folder: \PruebasServlet\src [Browse...](#)

Java package: com.beca.servlets [Browse...](#)

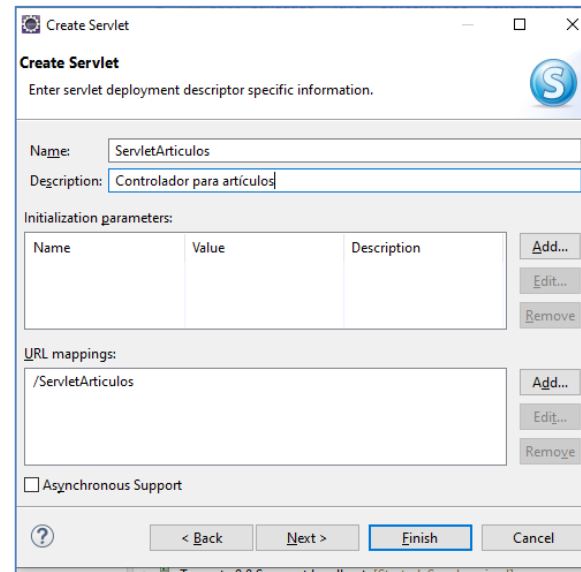
Class name: ServletArticulos

Superclass: javax.servlet.http.HttpServlet [Browse...](#)

☐ Use an existing Servlet class or JSP

Class name: ServletArticulos [Browse...](#)

[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)



**Create Servlet**

Enter servlet deployment descriptor specific information.

Name: ServletArticulos

Description: Controlador para artículos

Initialization parameters:

Name	Value	Description

[Add...](#) [Edit...](#) [Remove](#)

URL mappings:

/ServletArticulos [Add...](#) [Edit...](#) [Remove](#)

☐ Asynchronous Support

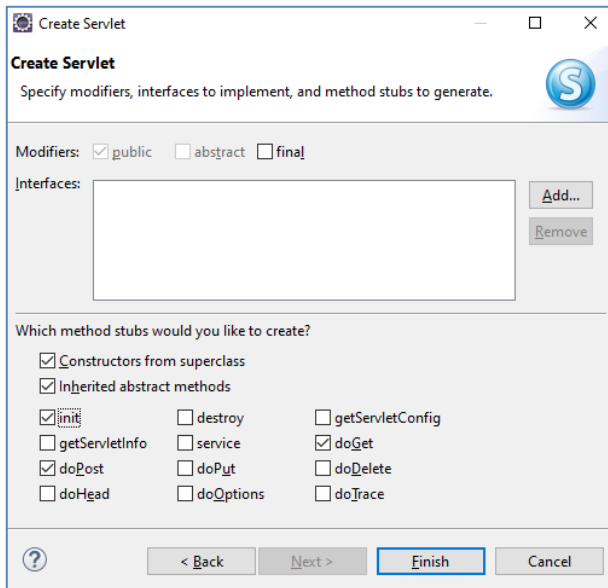
[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)



## Creación de un Servlet

### Servlets para MVC

- Una vez configurado lo anterior, vamos a crear nuestro Servlet que haga de controlador de las operaciones realizadas sobre la entidad **Articulo** (Botón derecho en el proyecto y **New Servlet**, le llamamos **ServletArticulos** (**Atención a los métodos**))





## Creación de un Servlet

### Servlets para MVC

```
ServletArticulos.java
1 package com.beca.servlets;
2
3 import java.io.IOException;
4
5 /**
6  * Servlet implementation class ServletArticulos
7  */
8 @WebServlet(description = "Controlador para artículos", urlPatterns = { "/ServletArticulos" })
9 public class ServletArticulos extends HttpServlet {
10     private static final long serialVersionUID = 1L;
11
12     /**
13      * @see HttpServlet#HttpServlet()
14      */
15     public ServletArticulos() {}
16
17     /**
18      * @see Servlet#init(ServletConfig)
19      */
20     public void init(ServletConfig config) throws ServletException {}
21
22     /**
23      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
24      */
25     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, I
26
27     /**
28      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
29      */
30     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, I
31 }
```

- **Línea 14** : anotaciones de Servlet (Ya no es necesario tener el fichero **web.xml**)
- **Línea 21** : constructor del Servlet
- **Línea 29** : se ejecuta sólo una vez, cuando el Servlet es creado, cada vez que el Servlet es instanciado. Nos sirve para inicializar parámetros del Servlet o clases adicionales que nos van a ayudar en la lógica del Servlet
- **Línea 36** : se ejecuta cada vez que se realiza una petición **GET** al Servlet
- **Línea 44** : se llama cada vez que se realiza una petición **POST** al Servlet
- Tanto las peticiones por **GET/POST** reciben los parámetros **request/response** al igual que vimos en las páginas JSP





## Creación de un Servlet

### Servlets para MVC

- El Servlet tiene 3 métodos importantes :

- **init** : donde se inicializa el servlet con los parámetros definidos en **web.xml**

```
public void init(ServletConfig config) throws ServletException {
```

- **doGet** : para procesar las peticiones **GET**

```
protected void doGet(HttpServletRequest request, HttpServletResponse response){
```

- **doPost** : para procesar las peticiones **POST**

```
protected void doPost(HttpServletRequest request, HttpServletResponse response){
```



## Creación de un Servlet

### Servlets para MVC

- Vamos a hacerlo más sencillo y vamos a trabajar siempre con las peticiones POST, para ello modificamos el método **doGet** como se muestra a continuación :

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    doPost(request, response);
}
```

De esta manera todas las peticiones **GET** que lleguen al Servlet serán redirigidas al método **doPost**



JSP

Servlets

## Creación de un Servlet

### Servlets para MVC

- Según indica la configuración que tenemos en las anotaciones del servlet ,nuestro servlet va a responder a la **URL**

```
urlPatterns = { "/ServletArticulos" }
```

Es decir, podemos probarlo accediendo a la **URL** <http://localhost:8080/PruebasServlet/ServletArticulos>



## Creación de un Servlet

### Servlets para MVC

- La idea es que el Servlet haga las funciones de un controlador, es decir, redirija la petición a la parte de lógica que sea conveniente y redirija la respuesta a la página JSP que sea conveniente, antes de redirigir la respuesta deberemos almacenar en el objeto **session** los Objetos que queremos se recuperen en la página JSP a donde redirigiremos la respuesta.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    HttpSession session = request.getSession(true);
}
```

- La sesión la controlamos así

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    HttpSession session = request.getSession(true);

    ServletContext sc = getServletContext();
    RequestDispatcher rd = sc.getRequestDispatcher("/articulos/articulos2.jsp");
    rd.forward(request, response);
}
```

- La redirección la realizamos así

- Recordemos que podemos almacenar en la sesión cualquier Objeto que queramos y que el Objeto sesión es compartido en toda la aplicación



### Ejercicio redirección desde servlet

#### Servlets

- Redirigir la salida del método **doPost** del servlet a la página **articulos.jsp**
- Redirigir la salida del método **doPost** del servlet a la página **articulos.jsp**, antes de realizar la redirección almacenar un Objeto en el objeto **session** de tipo **List<String>** que tenga dos elementos en la lista **"ARTICULO1","ARTICULO2"**
- En la página **articulos.jsp** recuperar el objeto almacenado en el objeto **session** y mostrarlo por pantalla



## Creación de un Servlet

### Servlets para MVC

- Ahora ya tenemos nuestro Servlet preparado para canalizar todas las peticiones de nuestras páginas JSP
- Tendremos que sustituir dentro de nuestras páginas JSP , el valor del atributo **action** de nuestros formularios por el valor de la URL definida en la configuración del Servlet

```
<form method="POST" action="./ServletArticulos">
  <input type="text" name="nombre"/>
  <button type="submit">Aceptar</button>
</form>
```

- Cuando hagamos click en el botón, los datos del formulario serán enviados al **servlet ServletArticulos**



### Ejercicio obtención de parámetros en servlet y reenvío de los mismos modificados

lowupp.jsp

- Implementar un conjunto de Html y Jsp llamado **lowupp** : **lowupp.html**, **lowupp.jsp** (El formulario que enviará los datos estará en el Html), usar el fichero **lowupp.jsp** sólo para visualizar el resultado devuelto por la lógica del Servlet)
- Añadir un elemento **radio** con el nombre **operacion** que permita dos valores : **upper** y **lower**
- Modificar la lógica del servlet para que cuando reciba el parámetro **operacion** con el valor **upper** que devuelva el parámetro **nombre** en mayúsculas y cuando reciba el parámetro **operacion** con el valor **lower** que devuelva el parámetro **nombre** en minúsculas



## Configurar varios Servlets

### Servlets para MVC

- Podemos crear más Servlets para distribuir la lógica entre ellos
- Una manera de hacerlo sería definir un Servlet por entidad
- Así podríamos tener separada la lógica de cada entidad en un Servlet diferente.
- Todavía se puede hacer mejor, pero vamos a empezar de esta manera.





## Configurar varios Servlets

### ServletClientes y ServletArticulos

- Crear dos nuevas páginas : **clientes.html** y **articulos.html** con sus respectivos **.jsp** de respuesta (**clientes.jsp**, **articulos.jsp**)
- En **clientes.html** crear un formulario que envíe como parámetros un campo **nombre** y otro campo **apellidos** al servlet **ServletClientes**
- Que el **ServletClientes** cuando reciba los parámetros, los almacene en una clase Java llamada **Cliente** que tenga esos atributos y almacene esta clase **Clientes** en el Objeto **session** con el nombre **salida** y rediriga a la página **clientes.jsp** para visualizar el Objeto **salida** con sus atributos
- Hacer lo mismo con **articulos.html** pero con los campos **nombre** y **precio** y enviando la petición al servlet **ServletArticulos**, guardando los parámetros en un Objeto llamado **Articulo** y redirigiendo la salida a la página **articulos.jsp** que es la que mostrará el Objeto **Articulo** guardado en el Objeto **session**



## Configurar la conexión a Bases de datos MySQL

### Conexión BBDD MySQL

- La idea es usar el método **init** del **Servlet** para configurar la conexión a la BBDD
- Crearemos una clase en **com.beca.db.DataManager** que nos ayudará a gestionar las conexiones de la base de datos
- La configuración de la conexión a la BBDD la guardaremos en los parámetros de inicialización del Servlet (Lo vamos a hacer con anotaciones)
- Necesitaremos descargar los Drivers de MySQL para Java y dejarlos en la carpeta **WEB-INF\lib** (Como hicimos con los de JSTL)
- Para conectar con una BBDD en Java necesitaremos los siguientes parámetros
  - Usuario : root
  - Contraseña : root
  - Url : jdbc:mysql://localhost:3306/pruebasservlet
  - jdbcDriver : com.mysql.cj.jdbc.Driver



### Configurar la conexión a Bases de datos MySql

#### Conexión BBDD MySql

- Establecer los parámetros de configuración de la Base de datos en los parámetros de inicialización del Servlet

```
@WebServlet(description = "Controlador para articulos", urlPatterns = { "/ServletArticulos" }, initParams = {  
    @WebInitParam(name = "dbUrl", value = "jdbc:mysql://localhost:3306/pruebasservlet?serverTimezone=UTC"),  
    @WebInitParam(name = "dbPass", value = "root"), @WebInitParam(name = "dbUser", value = "root"),  
    @WebInitParam(name = "jdbcDriver", value = "com.mysql.cj.jdbc.Driver") })
```

- Configuramos la clase **com.beca.db.DataManager** con los atributos **dbUrl**, **dbUsuario** y **dbPass**
- Deberemos generar los métodos GET/SET para esos atributos



## Configurar la conexión a Bases de datos MySql

### Conexión BBDD MySql

```
public class DataManager {
    String dbUrl;
    String dbUsuario;
    String dbPass;

    public Connection getConnection() {
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(getDbUrl(), getDbUsuario(), getDbPass());

            conn.setAutoCommit(false);
        } catch (SQLException e) {
            System.out.println("Could not connect to DB: " + e.getMessage());
        }
        return conn;
    }

    public void closeConnection(Connection conn) {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
            }
        }
    }
}
```

```
public String getDbUrl() {
    return dbUrl;
}

public void setDbUrl(String dbUrl) {
    this.dbUrl = dbUrl;
}

public String getDbUsuario() {
    return dbUsuario;
}

public void setDbUsuario(String dbUsuario) {
    this.dbUsuario = dbUsuario;
}

public String getDbPass() {
    return dbPass;
}

public void setDbPass(String dbPass) {
    this.dbPass = dbPass;
}
```

- Declaramos una variable **dataManager** en nuestro Servlet para poder gestionar las conexiones con la BBDD

```
public class ServletArticulos extends HttpServlet {
    private static final long serialVersionUID = 1L;
    ServletContext sc;
    DataManager dataManager;
```



### Configurar la conexión a Bases de datos MySql

#### Conexión BBDD MySql

- En el método **init** del Servlet recuperamos los parámetros de inicialización del Servlet para configurar los parámetros de conexión de la BBDD en nuestra clase **DataManager**

```
public void init(ServletConfig config) throws ServletException {
    System.out.print("LLEGA AQUÍ");
    sc = config.getServletContext();

    super.init(config);
    sc = getServletContext();
    config = getServletConfig();

    /* Obtenemos los parámetros de conexión de la Base de datos */
    dataManager = new DataManager();
    dataManager.setDbUrl(config.getInitParameter("dbUrl"));
    dataManager.setDbUsuario(config.getInitParameter("dbUser"));
    dataManager.setDbPass(config.getInitParameter("dbPass"));

    ServletContext context = config.getServletContext();

    context.setAttribute("dataManager", dataManager);

    try {
        System.out.println(config.getInitParameter("jdbcDriver"));
        Class.forName(config.getInitParameter("jdbcDriver"));
    } catch (Exception e) {
        System.out.println(e.toString());
    }
}
```





## Configurar la conexión a Bases de datos MySQL

### Conexión BBDD MySQL

- En la clase **DataManager** hemos definido dos métodos uno para obtener una conexión a la BBDD y otro para cerrar una conexión de la BBDD

```
public Connection getConnection() {  
  
public void closeConnection(Connection conn) {
```

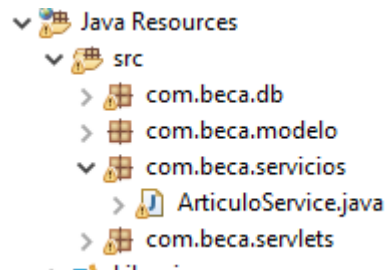
- Las conexiones a la BBDD las tenemos que cerrar nada más terminada la operación sobre la BBDD en la que estemos usando esa conexión



## Configurar la conexión a Bases de datos MySql

### Conexión BBDD MySql

- Ahora vamos a crear un servicio por cada entidad (En nuestro caso sólo tenemos la entidad Artículo) para realizar todas las operaciones en la BBDD, vamos a crear la más simple , listar todos los artículos de la tabla **articulos**
- Creamos la clase **com.beca.servicios.ArticuloService**





## Configurar la conexión a Bases de datos MySql

### Conexión BBDD MySql

- Como hemos dicho antes, sólo vamos a implementar el método **getArticulos** para que nos devuelva todos los registros de la tabla **articulos**

```
public static List<Articulo> getArticulos(Connection con) {
    Articulo art;

    List<Articulo> listaArts = new ArrayList();
    if (con != null) {
        try {
            Statement s = con.createStatement();
            String sql = "select a.id,a.nombre,a.precio from articulo a";

            try {
                ResultSet rs = s.executeQuery(sql);
                while (rs.next()) {
                    art = new Articulo();
                    art.setId(rs.getInt("id"));
                    art.setNombre(rs.getString("nombre"));
                    art.setPrecio(rs.getDouble("precio"));
                    listaArts.add(art);
                }
            } finally {
                s.close();
            }
        } catch (SQLException e) {
            System.out.println("No puede recuperar la lista de artículos " + e.getMessage());
        } finally {
            if (con != null) {
                try {
                    con.close();
                } catch (SQLException e) {
                }
            }
        }
    }
    return listaArts;
}
```







## Configurar la conexión a Bases de datos MySQL

### Conexión BBDD MySQL

- Ahora podríamos invocar desde dentro de la lógica de nuestro controlador **ServletArticulos** a cualquiera de estos métodos definidos dentro del servicio **ArticuloService**
- El método **POST** de nuestro controlador **ServletArticulos** podría quedar de la siguiente manera, manteniendo los ejemplos anteriores y este nuevo de **listar todos los artículos**



## Configurar la conexión a Bases de datos MySql

### Conexión BBDD MySql

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    RequestDispatcher rd;
    String operacion = new String();

    HttpSession session = request.getSession(true);

    operacion = request.getParameter("operacion");
    System.out.println("PASO1"+operacion);
    if (operacion != null) {

        System.out.println(operacion);
        if ("lower".equalsIgnoreCase(operacion) || "upper".equalsIgnoreCase(operacion))
            this.lowupp(operacion, request, response);

        else if("articulosListar".equalsIgnoreCase(operacion))
            this.articulosListar(request, response);

        else {
            List<String> salida = new ArrayList<>();
            salida.add("ARTICULO1");
            salida.add("ARTICULO2");
            session.setAttribute("respuesta", salida);
            System.out.println("VAMOA A SLAI");
            rd = sc.getRequestDispatcher("/articulos/articulos.jsp");
            rd.forward(request, response);
        }
    }
}
```





### Configurar la conexión a Bases de datos MySql

#### Conexión BBDD MySql

```
void lowupp(String operacion, HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    /* Variables para devolución de resultados */

    String lowuppResultado = new String();

    HttpSession session = request.getSession(true);
    RequestDispatcher rd;

    switch (operacion) {

    case "lower":
        lowuppResultado = request.getParameter("nombre").toLowerCase();
        session.setAttribute("respuesta", lowuppResultado);
        break;
    case "upper":
        lowuppResultado = request.getParameter("nombre").toUpperCase();
        session.setAttribute("respuesta", lowuppResultado);
        break;

    }

    rd = sc.getRequestDispatcher("/lowupp/lowupp.jsp");
    rd.forward(request, response);
}
```



## Operaciones sobre Artículos

### Artículos

- Implementar el método **articulosListar** descrito en las transparencias anteriores
- Crear una página JSP con el siguiente formato, para recibir la respuesta de **articulosListar** llamada **/articulos/articulos.jsp**

ID	NOMBRE	PRECIO
1	ART1	12.21
2	ART2	11.0

- Implementar las operaciones **insert** , **update** y **delete** con las páginas que consideréis , para gestionar la respuesta de esas operaciones usar **/articulos/articulos.jsp**
- \*\* Usar el parámetro <<operación>> para distinguir el tipo de operación que estamos invocando



## Entidad cliente

### ServletClientes

- Crear una nueva entidad en la BBDD llamada **cliente** con los atributos (**dni, nombre, apellidos, direccion, teléfono**)
- Crear un Bean de Modelo para la entidad **cliente**
- Crear un controlador (Servlet) llamado **ServletClientes**
- Implementar todas las operaciones sobre la entidad **cliente** : **insert, select, update, delete**



JSP

Ejercicio

## Operación clienteCompraArticulo

clienteCompraArticulo

- Implementar la operación **clienteCompraArticulo**
- Definirla en el Servlet correcto
- Implementar las páginas JSP necesarias para validar la operación

