Ministerio de Educación Superior, Ciencia y Tecnología (Mescyt)

Centro de Tecnología y Educación Permanente TEP

Pontificia Universidad Católica Madre y Maestra (PUCMM)

Diplomado en Programación en Lenguaje JAVA

Laboratorio 01

Módulo III

Programación Web Java

Facilitador: Ing. Eudris Cabrera

Fecha : 23 Julio del 2016

Santiago de los Caballeros, República Dominicana

Introducción

¿Qué es una aplicación web?

'Por definición, se trata de algo más que un 'sitio web'.

Se trata de una aplicación cliente / servidor que utiliza un navegador Web como su programa cliente, y por consiguiente constituye un servicio interactivo mediante la conexión con los servidores a través de Internet (o Intranet).

Una aplicación web presenta contenido adaptado dinámicamente en función de parámetros de la petición, los comportamientos de los usuarios seguidos, y consideraciones de seguridad.

Una aplicación Web Java puede ser representada como una jerarquía de directorios y archivos, que a su vez contiene:

- Componentes Web (Servlets, JavaServer Pages, entre otros)
- Recursos estáticos (páginas html e imágenes).
- Clases Java.
- Librerías (Archivos Jars).
- Un archivo descriptor de despliegue (web.xml).

Una aplicación web de Java se puede implementar como un archivo **".war"**.
El archivo **".war"** es un archivo zip que contiene todo el contenido de la aplicación web correspondiente.

Las aplicaciones java web normalmente no se ejecutan directamente en el servidor, sino que se ejecutan dentro de un **contenedor** en el servidor. El contenedor proporciona un entorno de ejecución para aplicaciones web en Java.

El contenedor es para aplicaciones web en Java lo que la JVM (Java Virtual Machine) es para las aplicaciones Java que se ejecutan locales. El contenedor en sí se ejecuta en la JVM.

En general, Java distingue dos contenedores: El **contenedor web** y el **contenedor Java EE**.

Un contenedor Web apoya la ejecución de Servlets Java y Java Server Pages.

Un contenedor compatible con Java EE provee funcionalidades adicionales, tales como, gestor de ejecución de los Enterprise JavaBeans, interfaz de conexión entre el servidor Java EE y aplicaciones clientes.

Contenedores web típicos en el mundo Java son **Tomcat** o **Jetty**.

La especificación **Servlet** es el fundamento de las aplicaciones web Java. Muchos frameworks de alto nivel incluyendo **JAX-RS**, **JavaServer Faces (JSF)**, y **Spring MVC** están basados en **servlets** y **JavaServer Pages**.

Los frameworks web populares en Java son: Spring, Java Server Faces y Vaadin.
Estos frameworks web por lo general se ejecutan en un contenedor web.

Guía del laboratorio

Este laboratorio está enfocado en las tecnologías Servlet/JSP, elementos básicos en el desarrollo de aplicaciones Web con Java.

Requerimientos

Java 1.7+ (Recomendado Java 8)
Netbeans 8.0.1+
Apache Maven 3.2.1+
Tomcat 7.0.54 ó superior (Recomendado Apache Tomcat 8.0.27.0 ó superior)
Mysql Server 5.0+

Escenario

Se desea crear un proyecto en forma de **task list** para llevar un registro de nuestras tareas.
Nuestro objetivo principal es aplicar los conceptos de **jsp** and **servlets** desarrollando una pequeña aplicacion web java.

Creación y configuración de la base de datos.

Crear una base de datos en mysql llamada **`taskapp_diplomado.`**

```
create database taskapp_diplomado;
```

Ejecutar el archivo **taskapp_diplomado.sql** situado en el directorio ***programacion-web-java/labs/lab01/sql*** dentro del repositorio del diplomado.

Paso I

Crear una aplicación **Maven** del tipo **Web Application**, ir a la opción: **New Project -> Maven -> Web Application.**

Información del proyecto

Nombre : **taskapp-web**

artifactId: valor por defecto

groupId: ***org.diplomado.pucmm.mescyt***

version: ***valor por defecto "1.0-SNAPSHOT"***

package: ***org.diplomado.pucmm.mescyt.taskapp.web***

Modificar archivo pom.xml

Remover la dependencia **javaee-web-api,** agregada por defecto y agregar las siguientes dependencias.

Servlet

```
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
</dependency>
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
</dependency>
```

Mysql

```
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.36</version>
</dependency>
```

Paso II

Crear la siguiente estructura de paquetes

org.diplomado.pucmm.mescyt.taskapp.web.controller	Para las clases que tienen que ver con la navegación.
org.diplomado.pucmm.mescyt.taskapp.web.utilidades	Paquete para agrupar cualquier clase que sea un utilitario.
org.diplomado.pucmm.mescyt.taskapp.web.entidades	Clase que representan el dominio o negocio (POJO).
org.diplomado.pucmm.mescyt.taskapp.web.servicios	Para contener las clases que tienen que ver con acceso a datos.
org.diplomado.pucmm.mescyt.taskapp.web.filtros	Filtros para el manejo de acceso y autenticación
org.diplomado.pucmm.mescyt.taskapp.web.sesiones	Contener las clases para iniciar o concluir una sesión.

Paso III:

Crear clase que representen las tablas que tenemos en la base de datos.

Crear clase **Task** en el paquete **entidades** y crear métodos setter y getter.

Paso IV:

Configurar datasource en el archivo **META-INF/context.xml**

Datasource (Fuente de datos) es un nombre dado a la conexión establecida para una base de datos desde un servidor. Se utiliza comúnmente cuando se crea una consulta a la base de datos.

Según **Sun MicroSystem** un objeto **DataSource** representa una fábrica (factory) para las conexiones a la fuente de datos. Es una alternativa a la instalación de DriverManager, un objeto de origen de datos (DataSource) es el medio preferido para conseguir una conexión. Un objeto que implementa la interfaz **DataSource** típicamente se ha registrado en un servicio de nombres basado en la API **Java Naming and Directory Interface (JNDI)**.

La interfaz de origen de datos (Datasource) se lleva a cabo por un proveedor del controlador.

Hay tres tipos de implementaciones:

Implementación básica - produce un objeto de conexión estándar.
Implementación agrupación de conexiones (Connection pooling).
Produce un objeto Connection que participará automáticamente en la agrupación de conexiones. Esta aplicación funciona con un administrador de conexión a nivel de un la capa media (middle-tier).

Implementación transacción distribuida
Produce un objeto de conexión que puede utilizarse para transacciones distribuidas y casi siempre participa en la agrupación de conexiones.

Esta aplicación funciona con un gestor de transacciones de nivel medio y casi siempre con un administrador de conexión común.

```
 <Resource auth="Container" driverClassName="com.mysql.jdbc.Driver" global="jdbc/TaskApp"
maxActive="100" maxIdle="20" maxWait="10000" minIdle="5" name="jdbc/TaskApp"
password="rootweb" type="javax.sql.DataSource"
url="jdbc:mysql://localhost:3306/taskapp_diplomado" username="root"/>
```

jdbc/TaskApp es el nombre que le hemos dado a un nuestro datasource
url es el string de conexión a la base de datos
username usuario de la base de datos
password clave de la base de datos
minIdle, maxIdle, maxActive está relacionado con los tiempos que puede durar activa una conexión antes de pasar a inactivo.
driverClassName nombre del driver

Paso V:

1- Crear excepción personalizada para manejar las excepciones de nuestra aplicación.

Crear una nueva clase y llamarle **TaskAppException**, en el paquete de utilidades, luego debemos extender de Exception.

Ahora necesitamos sobrecargar los **constructores** disponible en la clase **Exception**.

Click derecho en el cuerpo de la clase y seleccionar la opción **Constructor**.
Marcar todos los constructores disponibles.

2- Crear servicios para tener acceso a la data de nuestras entidades

a) Crear clase **ServicioPersistenciaBase** que va a contener una serie de métodos que podemos reutilizar en los demás servicios.

Crear método **getConeccion()** para manejar las conexiones con la base de datos. Cuando estamos trabajando desde un contenedor web o un servidor de aplicaciones debemos usar un datasource en lugar de llamar la clase **DriverManager**.

En pasos anteriores creamos el datasource **jdbc/TaskApp** y este es el momento de usarlo.

Para acceder a dicho recurso debemos crear una instancia de la interfaz **Context** y usar el método **lookup**.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("java:/comp/env/jdbc/TaskApp");
```

java:comp/env es el nodo en el árbol JNDI, donde puedes encontrar propiedades para una aplicación web, o un EJB.

En el código anterior la interfaz DataSource contiene un método que nos permite obtener la conexión.

```
Connection con = ds.getConnection();
```

Debemos validar si la conexión devuelve valores nulos y lanzar una excepción en caso que sea así.

```
 if (con == null) {
       throw new TaskListException("No pudo establecer conexión con la base de datos");
 }
```

Poniendo todo junto nuestro método **getConeccion()** quedará de la siguiente manera:

```
protected Connection getConeccion() throws TaskListException {
       try {
              Context ctx = new InitialContext();
              DataSource ds = (DataSource) ctx.lookup("java:/comp/env/jdbc/TaskApp");

              Connection con = ds.getConnection();

              if (con == null) {
               throw new TaskAppException("No pudo establecer conexion con la base de
datos");
              }
              return con;
       } catch (NamingException | SQLException ex) {
```

```
                ogger.getLogger(ServicioPersistenciaBase.class.getName()).log(Level.SEVERE,
null, ex);
                throw new TaskListException(ex);
            }
        }
```

b) Crear clase **ServicioTask** y extender de **ServicioPersistenciaBase**

Crear los métodos siguientes :

- List<Task> consultarTaskTodas()
- Optional<Task> consultarTaskPorId(int id)
- boolean guardarTask(Task task)
- boolean modificarTask(Task task)
- boolean borrarTask(int id)

Para el método **List<Task> consultarTaskTodas()** debemos hacer una consulta a la base de datos que nos devuelva todos los registros y luego encapsular cada registro en un tipo de dato Task y devolverlo en una lista de objetos del tipo **Task**.

```
public List<Task> consultarTaskTodas(){
    List<Task> lista = new ArrayList<>();

     try(Connection con = getConeccion()){
                try(PreparedStatement stmt = con.prepareStatement("select * from task order by
id desc")){
                try(ResultSet rs = stmt.executeQuery()){
                      while(rs.next()){
                       Task task = new Task();
                       task.setId(rs.getInt("id"));
                       task.setDescripcion(rs.getString("descripcion"));
                       task.setNombre(rs.getString("nombre"));
                       task.setPrioridad(rs.getString("prioridad"));
                       task.setFinalizado(rs.getBoolean("finalizado"));

                       lista.add(task);
                       }
                    }
                  }
        } catch (SQLException | TaskAppException ex) {
```

```
                Logger.getLogger(ServicioTask.class.getName()).log(Level.SEVERE, null, ex);
        }
        return lista;
      }
```

Para el método **Optional<Task> consultarTaskPorId(int id)** debemos hacer una consulta a la base de datos que dado un id devuelva el registro correspondiente y luego encapsular el registro en un tipo de dato Task.

En esta ocasión vamos a utilizar la clase **Optional**.

La clase **Optional** ayuda a eliminar los problemas de NullPointerException.
Optional<T> sirve como un contenedor para una referencia de objeto (nulo, o objeto real).
Piensa en Optional como un stream 0 ó 1 elemento. Este garantiza que la referencia de Optional no retornará nulo.

Puedes investigar más sobre **Optional** en la siguiente presentación
https://github.com/ecabrerar/java-8-mas-alla-de-las-expresiones-lambda

```
 protected  Optional<Task> consultarTaskPorId(int id) throws TaskAppException {
    Optional<Task> opEntidad = Optional.empty();
       try (Connection con = getConeccion()) {
              try (PreparedStatement stmt = con.prepareStatement("select * from task where
id=?")) {
                                      stmt.setInt(1, id);
              try(ResultSet rs = stmt.executeQuery()){
                       rs.next();
                      Task task = new Task();
                      task.setId(rs.getInt("id"));
                      task.setDescripcion(rs.getString("descripcion"));
                      task.setNombre(rs.getString("nombre"));
                      task.setPrioridad(rs.getString("prioridad"));
                      task.setFinalizado(rs.getBoolean("finalizado"));

                      opEntidad = Optional.of(task);
                  }
                }
```

```
        } catch (SQLException | TaskAppException ex) {

Logger.getLogger(ServicioPersistenciaBase.class.getName()).info(MessageFormat.format("Error en
el SQl{0}", ex.getMessage()));
                    throw new TaskAppException(ex);
        }
        return opEntidad;
     }
```

Al momento de insertar el registro solamente nos interesa guardar los siguientes campos **nombre,descripción,prioridad**.

```
public boolean guardarTask(Task task) {
        boolean estado;
     try (Connection con = getConeccion()) {

              try (PreparedStatement stmt = con.prepareStatement("insert into task
(nombre,descripcion,prioridad)values(?,?,?)")) {
                    stmt.setString(1, task.getNombre());
                    stmt.setString(2, task.getDescripcion());
                    stmt.setString(3, task.getPrioridad());

                    estado = stmt.execute();

              }

     } catch (SQLException | TaskAppException ex) {
        Logger.getLogger(ServicioTask.class.getName()).log(Level.SEVERE, null, ex);
        estado = false;
    }
    return estado;
}
```

Para la modificación de un registro vamos a considerar los campos nombre,descripción,prioridad y finalizado.

```
public boolean modificarTask(Task task) {
        boolean estado;

        try (Connection con = getConeccion()) {

              try (PreparedStatement stmt = con.prepareStatement("update task set
nombre=?,descripcion=?,prioridad=?,finalizado=? where id=?")) {
```

```
                stmt.setString(1, task.getNombre());
                stmt.setString(2, task.getDescripcion());
                stmt.setString(3, task.getPrioridad());
                stmt.setBoolean(4, task.isFinalizado());
                stmt.setInt(5, task.getId());

                estado = stmt.execute();
                }

        } catch (SQLException | TaskAppException ex) {
                Logger.getLogger(ServicioTask.class.getName()).log(Level.SEVERE, null, ex);
                estado = false;
        }

        return estado;
        }
```

Borrar un registro dado un id previamente registrado

```
public boolean borrarTask(int id) {
        boolean estado;
        try (Connection con = getConeccion()) {
                        try (PreparedStatement stmt = con.prepareStatement("delete from task
        where id=?")) {
                        stmt.setInt(1, id);

                        estado = stmt.execute();
                        }
        } catch (SQLException | TaskAppException ex) {

Logger.getLogger(ServicioTask.class.getName()).info(MessageFormat.format("Error en el SQl{0}",
ex.getMessage()));
                        estado = false;
        }
        return estado;
}
```

Poniendo todo junto

ServicioPersistenciaBase:

```
package org.diplomado.pucmm.mescyt.tasklist.web.servicios;
```

```
import java.sql.Connection;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;
import org.diplomado.pucmm.mescyt.taskapp.web.utilidades.TaskAppException;

/**
 *
 * @author ecabrerar
 */
public class ServicioPersistenciaBase {

     protected Connection getConeccion() throws TaskAppException {

       try {
               Context ctx = new InitialContext();
               DataSource ds = (DataSource) ctx.lookup("java:/comp/env/jdbc/TaskApp");

               Connection con = ds.getConnection();

               if (con == null) {
               throw new TaskAppException("No pudo establecer conexion con la base de datos");
               }

               return con;
       } catch (NamingException | SQLException ex) {
               Logger.getLogger(ServicioPersistenciaBase.class.getName()).log(Level.SEVERE,
null, ex);
               throw new TaskAppException(ex);
       }
     }
}
```

ServicioTask:

```
package org.diplomado.pucmm.mescyt.tasklist.web.servicios;

import java.sql.Connection;
import java.sql.PreparedStatement;
```

```java
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.MessageFormat;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.diplomado.pucmm.mescyt.taskapp.web.entidades.Task;
import org.diplomado.pucmm.mescyt.taskapp.web.utilidades.TaskAppException;

/**
 *
 * @author ecabrerar
 */
public class ServicioTask extends ServicioPersistenciaBase {

public boolean guardarTask(Task task) {
        boolean estado;

    try (Connection con = getConeccion()) {
                try (PreparedStatement stmt = con.prepareStatement("insert into task
(nombre,descripcion,prioridad)values(?,?,?)")) {
                        stmt.setString(1, task.getNombre());
                        stmt.setString(2, task.getDescripcion());
                        stmt.setString(3, task.getPrioridad());

                        estado = stmt.execute();
                  }
        } catch (SQLException | TaskAppException ex) {
                        Logger.getLogger(ServicioTask.class.getName()).log(Level.SEVERE, null,
ex);
                           estado = false;
        }

        return estado;
}

public boolean modificarTask(Task task) {
        boolean estado;

    try (Connection con = getConeccion()) {
```

```
            try (PreparedStatement stmt = con.prepareStatement("update task set
nombre=?,descripcion=?,prioridad=?,finalizado=? where id=?")) {
                    stmt.setString(1, task.getNombre());
                    stmt.setString(2, task.getDescripcion());
                    stmt.setString(3, task.getPrioridad());
                    stmt.setBoolean(4, task.isFinalizado());
                    stmt.setInt(5, task.getId());

                    estado = stmt.execute();
          }
       } catch (SQLException | TaskAppException ex) {
                    Logger.getLogger(ServicioTask.class.getName()).log(Level.SEVERE, null,
ex);
            estado = false;
       }

       return estado;
}

public List<Task> consultarTaskTodas(){

     List<Task> lista = new ArrayList<>();

     try(Connection con = getConeccion()){
            try(PreparedStatement stmt = con.prepareStatement("select * from task order by
id desc")){

                    try(ResultSet rs = stmt.executeQuery()){
                          while(rs.next()){

                                  Task task = new Task();
                                  task.setId(rs.getInt("id"));
                                  task.setDescripcion(rs.getString("descripcion"));
                                  task.setNombre(rs.getString("nombre"));
                                  task.setPrioridad(rs.getString("prioridad"));
                                  task.setFinalizado(rs.getBoolean("finalizado"));

                                  lista.add(task);
                           }
                    }
                }

       } catch (SQLException | TaskAppException ex) {
```

```
                        Logger.getLogger(ServicioTask.class.getName()).log(Level.SEVERE, null,
ex);
        }

        return lista;
  }


 protected Optional<Task> consultarTaskPorId(int id) throws TaskAppException {
        Optional<Task> opEntidad = Optional.empty();

        try (Connection con = getConeccion()) {
                        try (PreparedStatement stmt = con.prepareStatement("select * from task
        where id=?")) {
                                stmt.setInt(1, id);
                                try(ResultSet rs = stmt.executeQuery()){
                                        rs.next();
                                        Task task = new Task();
                                        task.setId(rs.getInt("id"));
                                        task.setDescripcion(rs.getString("descripcion"));
                                        task.setNombre(rs.getString("nombre"));
                                        task.setPrioridad(rs.getString("prioridad"));
                                        task.setFinalizado(rs.getBoolean("finalizado"));

                                        opEntidad = Optional.of(task);
                                }
                           }
        } catch (SQLException | TaskAppException ex) {

Logger.getLogger(ServicioPersistenciaBase.class.getName()).info(MessageFormat.format("Error en
el SQl{0}", ex.getMessage()));
                        throw new TaskAppException(ex);
        }

        return opEntidad;

}

public boolean borrarTask(int id) {
        boolean estado;
        try (Connection con = getConeccion()) {
                        try (PreparedStatement stmt = con.prepareStatement("delete from task
        where id=?")) {
```

```
                    stmt.setInt(1, id);
                    estado = stmt.execute();
                    }
        } catch (SQLException | TaskAppException ex) {

Logger.getLogger(ServicioTask.class.getName()).info(MessageFormat.format("Error en el SQl{0}",
ex.getMessage()));
                estado = false;
        }

        return estado;
    }

}
```

Paso VI:

A continuación, vamos a crear una consulta simple usando JSP.

JavaServer Pages (JSP) es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.

Click derecho encima del proyecto, **New** > **JSP** y colocar el nombre **list**. Dejar la opción que está marcada por defecto.

En nuestra consulta, vamos a utilizar una tabla **HTML** (http://www.w3schools.com/html/html_tables.asp) para mostrar los datos.

Definiendo columnas a mostrar en la tabla

```
<tr>
<th>Id</th>
<th>Nombre</th>
<th>Descripcion</th>
<th>Prioridad</th>
</tr>
```

Luego, llamar el método del **ServicioTask** que nos permite consultar todos los registros.

```
<%
                ServicioTask servicioTask = new ServicioTask();
                List<Task> lista = servicioTask.consultarTaskTodas();


                %>
```

Nota:
Los scriptlets **JSP** nos permiten empotrar segmentos de código java dentro de una página JSP.

El código empotrado se inserta directamente en el servlet generado que se ejecuta cuando se pide la página.

Este **scriptlet** usa las variables declaradas en las directivas descritas anteriormente.

Los Scriptlets van encerradas entre etiquetas **<% y %>**.

```
<%
strMult = request.getParameter("MULTIPLIER");
socsec = request.getParameter("SOCSEC");
integerMult = new Integer(strMult);
multiplier = integerMult.intValue();
bonus = 100.00;
%>
```

Después de obtener todos los registros y guardarlo en una lista, debemos recorrerlo para mostrarlo en la tabla.

```
<%
 for(Task t : lista){
                %>
                <tr>
                <td><%=t.getId()%></td>
                <td><%=t.getDescripcion()%></td>
                <td><%=t.getNombre()%></td>
                <td><%=t.getPrioridad()%></td>

                </tr>
                <%
```

```
            }
        %>
```

Recuerde que todas las clases Java que son usadas dentro de un JSP requieren que usted haga la importación de la clase dentro del JSP.

Todo junto

```
<%@page import="org.diplomado.pucmm.mescyt.taskapp.web.entidades.Task"%>
<%@page import="java.util.List"%>
<%@page import="org.diplomado.pucmm.mescyt.tasklist.web.servicios.ServicioTask"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
        <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Consultas Task</title>
        </head>
        <body>

        <table style="width: 100%">
                <tr>
                <th>Id</th>
                <th>Nombre</th>
                <th>Descripcion</th>
                <th>Prioridad</th>

                </tr>
                <%
                ServicioTask servicioTask = new ServicioTask();
                List<Task> lista = servicioTask.consultarTaskTodas();

                for(Task t : lista){
                %>
                <tr>
                <td><%=t.getId()%></td>
                <td><%=t.getDescripcion()%></td>
                <td><%=t.getNombre()%></td>
                <td><%=t.getPrioridad()%></td>
               </tr>
                <%
```

```
            }
            %>
        </table>
        </body>
</html>
```

Para correr el ejemplo, clic derecho encima del proyecto, luego la opción **Run**. Asegurarse que el servidor tomcat esté encendido a la hora de darle a correr. **Url** para acceder al ejemplo http://localhost:8084/taskapp-web/list.jsp . Sustituir el puerto mostrado por el puerto definido en su máquina.

JSP permiten la utilización de código Java mediante scripts. Además, es posible utilizar algunas acciones JSP predefinidas mediante etiquetas.
El ejemplo anterior puede ser enriquecido mediante la utilización de Bibliotecas de Etiquetas (TagLibs o Tag Libraries).

Click derecho encima del proyecto, **New > JSP** y colocar el nombre list-jstl. Dejar la opción que está marcada por defecto.
En lugar de utilizar un for java tradicional, usaremos la etiqueta **JSTL forEach**. Esta etiqueta tiene un atributo llamado **items**, donde se coloca la fuente de datos y otro más llamado **var**, que es la variable donde se guarda cada valor al momento de recorrer el conjunto de datos.

Usamos la misma definición de tabla usado en el ejemplo anterior y sólo cambiaremos donde recorremos los datos.

```
 <c:forEach items="${task_list}" var="task">
            <tr>
            <td>${task.getId()}</td>
            <td>${task.getDescripcion()}</td>
            <td>${task.getNombre()}</td>
            <td>${task.getPrioridad()}</td>

            </tr>
            </c:forEach>
```

Tomar en consideración que cuando usamos JSTL o pasamos valores como atributo, debemos usar la definición de variable usando los símbolos ${}.
Poniendo todo junto

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@page import="org.diplomado.pucmm.mescyt.taskapp.web.entidades.Task"%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
        <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Consultas Task</title>
        </head>
        <body>

        <table style="width: 100%">
                <tr>
                <th>Id</th>
                <th>Nombre</th>
                <th>Descripcion</th>
                <th>Prioridad</th>
                </tr>

                <c:forEach items="${task_list}" var="task">
                <tr>
                <td>${task.getId()}</td>
                <td>${task.getDescripcion()}</td>
                <td>${task.getNombre()}</td>
                <td>${task.getPrioridad()}</td>

                </tr>
                </c:forEach>
        </table>
        </body>
</html>
```

En este ejemplo estamos usando a JSP solamente para la parte visual, las decisiones, la tomaremos en un **Servlet** que usaremos como controlador.

Click derecho encima del paquete **org.diplomado.pucmm.mescyt.taskapp.web.controller**, **New > Servlet** y colocar el nombre **IndexServlet**, luego pulsar **Finish**.

```
@WebServlet(name = "IndexServlet", urlPatterns = {"/IndexServlet"})
public class IndexServlet extends HttpServlet {
 ....
}
```

Queremos que al momento de iniciar el Servlet se inicialice una instancia de la clase **ServicioTask**.
Debemos sobreescribir el método **init**.
Dentro del cuerpo de la clase Click **Derecho** y seleccionar la opción **Insert Code**, Luego **Override Method**. Expandir la opción **Generic Servlet** y seleccionar el método **init(ServletConfig config)**.

Se debe agregar el siguiente método

```
@Override
public void init(ServletConfig config) throws ServletException {
        super.init(config);

}
```

Definimos la instancia de la clase **ServicioTask** e inicializamos en el método init

```
        ServicioTask servicioTask;

        @Override
        public void init(ServletConfig config) throws ServletException {
        super.init(config);
```

```
        servicioTask = new ServicioTask();
        }
```

En el método **processRequest(HttpServletRequest request, HttpServletResponse response)**, borrar el contenido por defecto
y colocar el atributo **task_list**, donde se cargarán los datos.

```
request.setAttribute("task_list", servicioTask.consultarTaskTodas());
```

Queremos que cuando se llame el Servlet **IndexServlet** nos redireccione al archivo **list-jstl.jsp**

```
ServletContext servletContext = request.getServletContext();

        servletContext
                .getRequestDispatcher("/list-jstl.jsp")
                .forward(request, response);
```

El método **processRequest(HttpServletRequest request, HttpServletResponse response)** debe quedar así.

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

        request.setAttribute("task_list", servicioTask.consultarTaskTodas());

        ServletContext servletContext = request.getServletContext();

        servletContext
                .getRequestDispatcher("/list-jstl.jsp")
                .forward(request, response);

        }
```

Nota:

ServletContext es un objeto que contiene meta-información acerca de la aplicación web.

Pasar atributos

Asignación
context.setAttribute("someValue", "aValue");

Lectura
Object attribute = context.getAttribute("someValue");
${someValue}

Para correr el ejemplo, clic derecho encima del proyecto, luego la opción **Run**. Asegurarse que el servidor tomcat esté encendido a la hora de darle a correr. Url para acceder al ejemplo http://localhost:8084/taskapp-web/IndexServlet . Sustituir el puerto mostrado por el puerto definido en su máquina.