

# CURSO STRUTS FRAMEWORK

# INTEGRACIÓN DE FRAMEWORKS

# STRUTS + EJB + JPA

# BÁSICO



Por el experto: Ing. Ubaldo Acosta



Experiencia y Conocimiento para tu vida

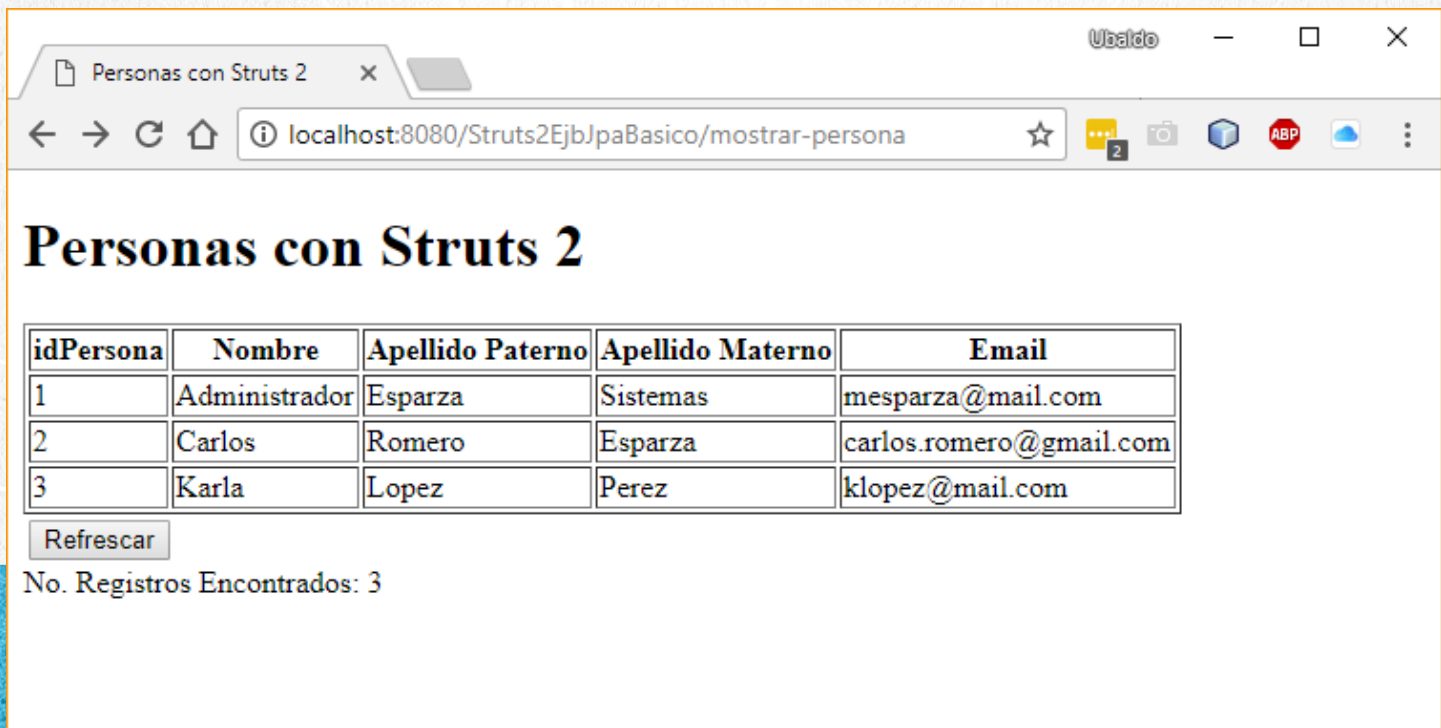


**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

# OBJETIVO DEL EJERCICIO

Crear una aplicación para poner en práctica la integración de los frameworks Struts + EJB+ JPA (Java EE). Al finalizar deberemos observar lo siguiente:



The screenshot shows a web browser window with the title "Personas con Struts 2". The address bar displays "localhost:8080/Struts2EjbJpaBasico/mostrar-persona". The page content includes a heading "Personas con Struts 2" and a table with 5 columns: "idPersona", "Nombre", "Apellido Paterno", "Apellido Materno", and "Email". The table contains 3 rows of data. Below the table is a "Refrescar" button and a text label "No. Registros Encontrados: 3".

idPersona	Nombre	Apellido Paterno	Apellido Materno	Email
1	Administrador	Esparza	Sistemas	mesparza@mail.com
2	Carlos	Romero	Esparza	carlos.romero@gmail.com
3	Karla	Lopez	Perez	klopez@mail.com

Refrescar

No. Registros Encontrados: 3



# REQUERIMIENTO DEL EJERCICIO

Mostrar el listado de personas utilizando Struts + EJB + JPA. Para ello necesitamos integrar las 3 tecnologías descritas.



**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

# 1. CREAR PROYECTO

Vamos a utilizar Maven para crear el proyecto Java Web. El proyecto se llamará Struts2EjbJpaBasico. Este proyecto integrará las 3 tecnologías: Struts + EJB+ JPA.

Empecemos con nuestro ejercicio:



Experiencia y Conocimiento para tu vida

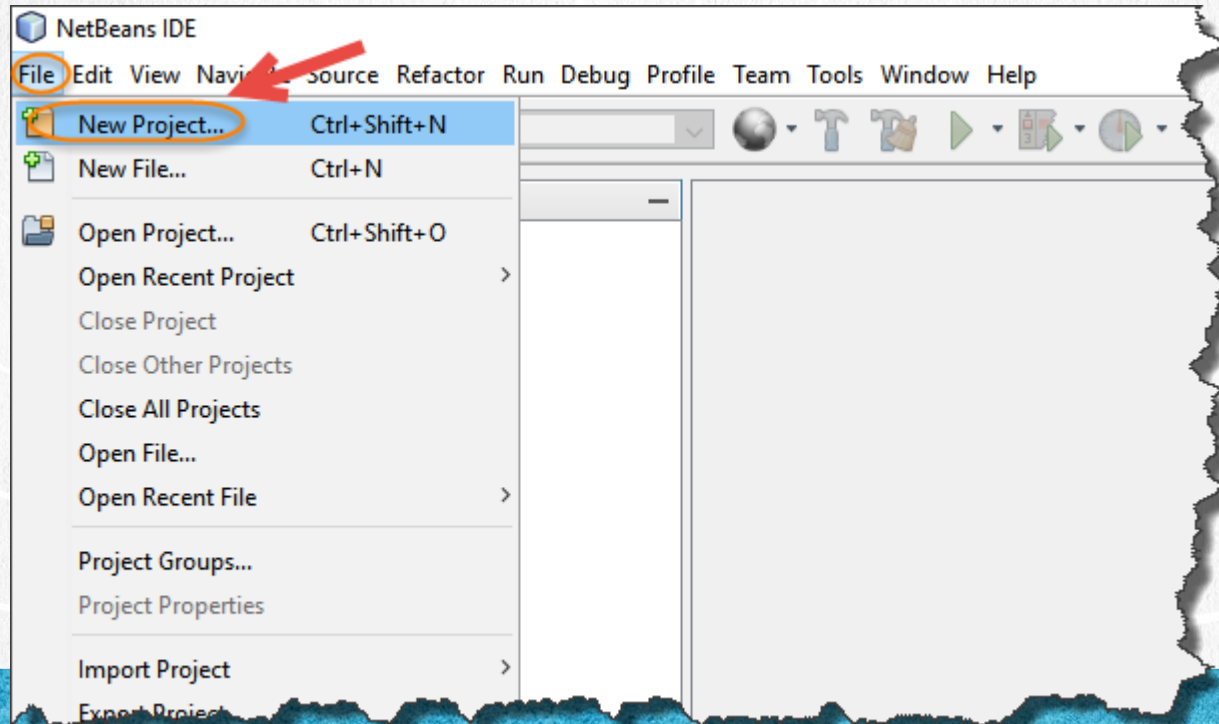
**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)



# 1. CREAR PROYECTO

Creamos nuestro ejercicio llamado Struts2EjbJpaBasico:

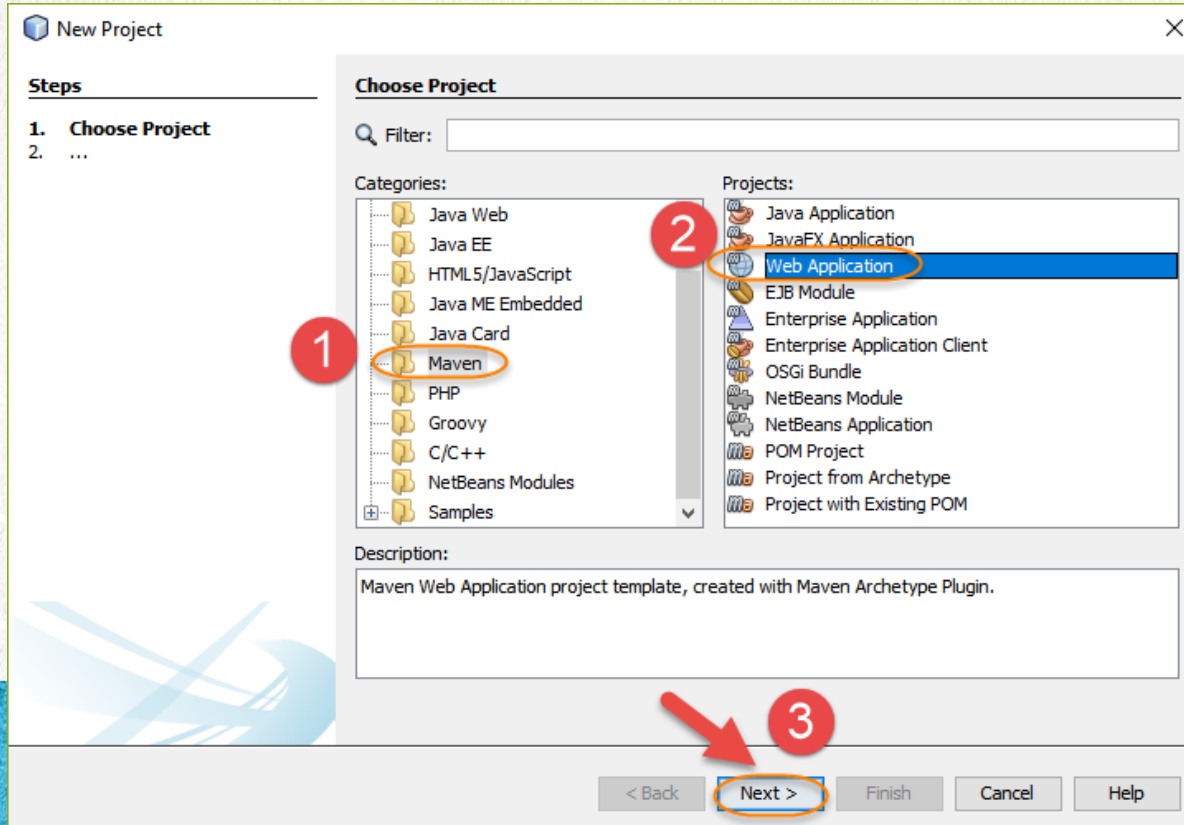


**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

# 1. CREAR PROYECTO

- Creamos un nuevo proyecto Java Maven de tipo Web Application:



# 1. CREAR PROYECTO

- Creamos un nuevo proyecto Java Maven:

**New Web Application**

**Steps**

1. Choose Project
2. **Name and Location**
3. Settings

**Name and Location**

Project Name: Struts2EjbJpaBasico

Project Location: C:\Cursos\Struts\02-IntegracionStrutsEjbJpaBasico [Browse...](#)

Project Folder: irsos\Struts\02-IntegracionStrutsEjbJpaBasico\Struts2EjbJpaBasico

Artifact Id: Struts2EjbJpaBasico

Group Id: mx.com.gm

Version: 1.0

Package:  (Optional)

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



# 1. CREAR PROYECTO

- Creamos un nuevo proyecto Java Maven:

The screenshot shows the 'New Web Application' wizard with the following details:

- Steps:**
  1. Choose Project
  2. Name and Location
  3. **Settings**
- Settings:**
  - Server:** GlassFishServer (highlighted with a red circle 1 and an orange oval)
  - Java EE Version:** Java EE Web (highlighted with a red circle 2 and an orange oval)
- Buttons:** < Back, Next >, **Finish** (highlighted with a red circle 3 and a red arrow), Cancel, Help

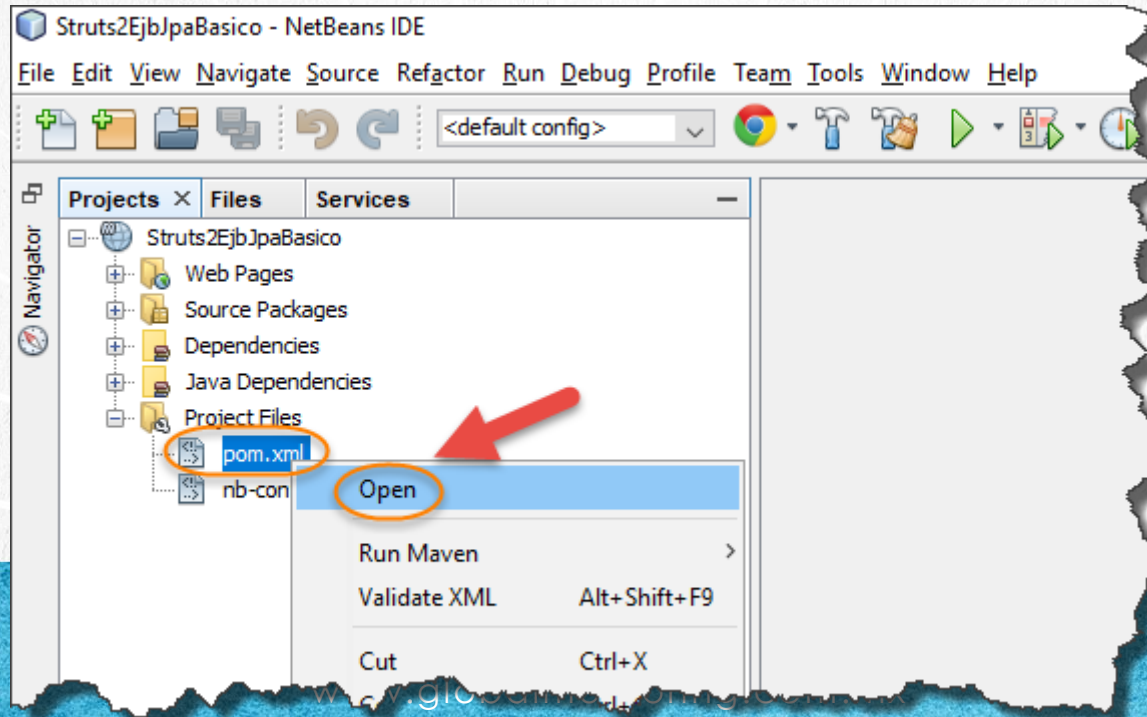
**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)



## 2. ABRIMOS EL ARCHIVO POM.XML DE MAVEN

- El archivo pom.xml de maven administra las librerías Java que vamos a utilizar. Agregaremos todas las librerías necesarias para integrar las tecnologías descritas:



# PASO 3. MODIFICAMOS EL CÓDIGO

## Archivo pom.xml:

Clic para ver el  
archivo

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>mx.com.gm</groupId>
  <artifactId>Struts2EjbJpaBasico</artifactId>
  <version>1.0</version>
  <packaging>war</packaging>

  <name>Struts2EjbJpaBasico</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-web-api</artifactId>
      <version>8.0</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.apache.struts</groupId>
      <artifactId>struts2-core</artifactId>
      <version>2.5.14.1</version>
    </dependency>
```



# PASO 3. MODIFICAMOS EL CÓDIGO

## Archivo pom.xml:

Clic para ver el  
archivo

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.10.0</version>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.10.0</version>
</dependency>
<dependency>
  <groupId>org.apache.struts</groupId>
  <artifactId>struts2-convention-plugin</artifactId>
  <version>2.5.14.1</version>
</dependency>
<dependency>
  <groupId>org.ow2.asm</groupId>
  <artifactId>asm</artifactId>
  <version>6.0</version>
</dependency>
<!-- MySql -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.42</version>
</dependency>
```

# PASO 3. MODIFICAMOS EL CÓDIGO

## Archivo pom.xml:

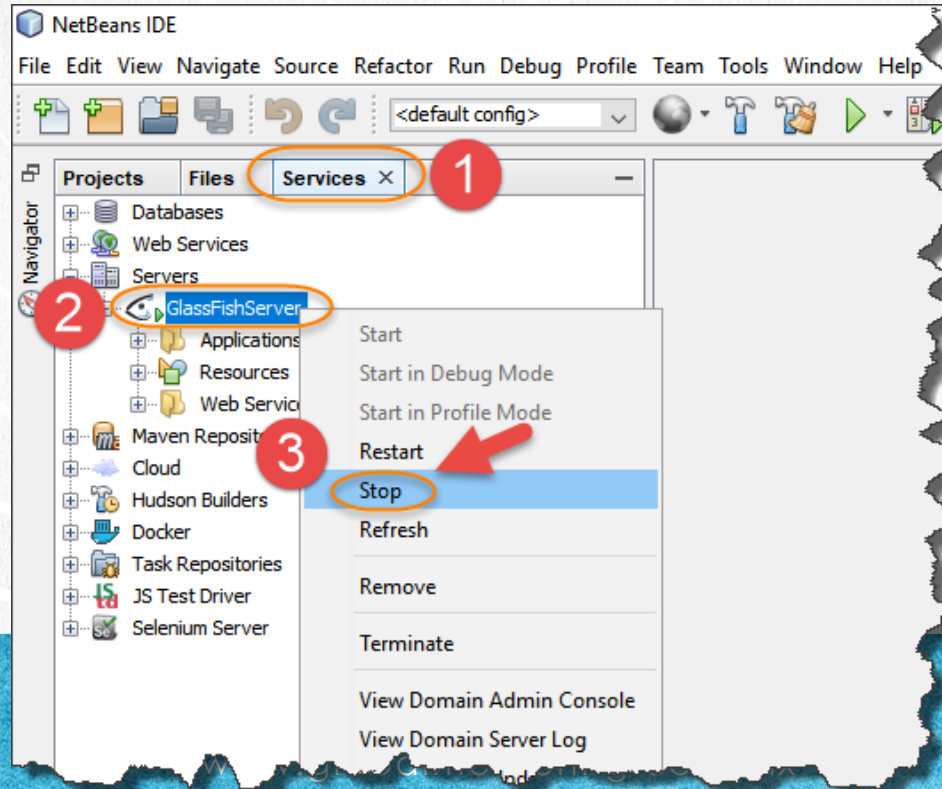
Clic para ver el  
archivo

```
<!--CDI para Struts 2 y Java EE-->
<dependency>
  <groupId>org.apache.struts</groupId>
  <artifactId>struts2-cdi-plugin</artifactId>
  <version>2.5.14.1</version>
</dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <version>2.3</version>
      <configuration>
        <failOnMissingWebXml>>false</failOnMissingWebXml>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.7.0</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```



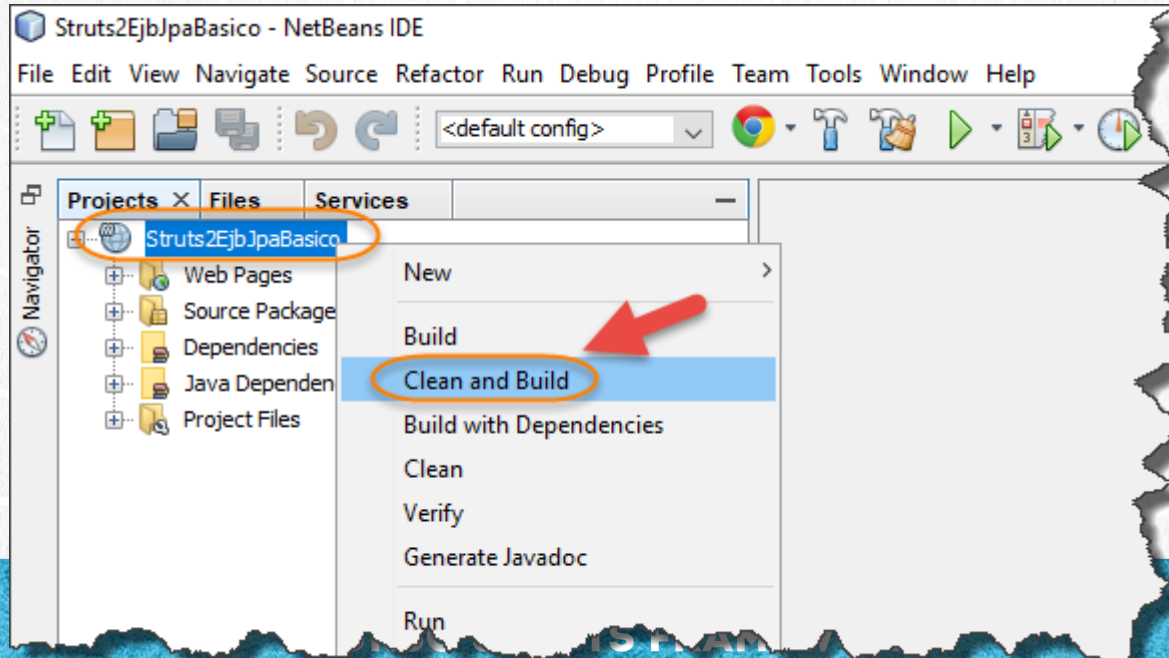
## 4. DETENEMOS GLASSFISH SI ESTUVIERA INICIADO

- Antes de hacer Clean & Build del proyecto para que descargue las librerías si fuera necesario, verificamos que el servidor de Glassfish no esté iniciado ya que puede haber problemas para hacer el proceso de Clean & build si el servidor está iniciado.



## 5. HACEMOS CLEAN & BUILD

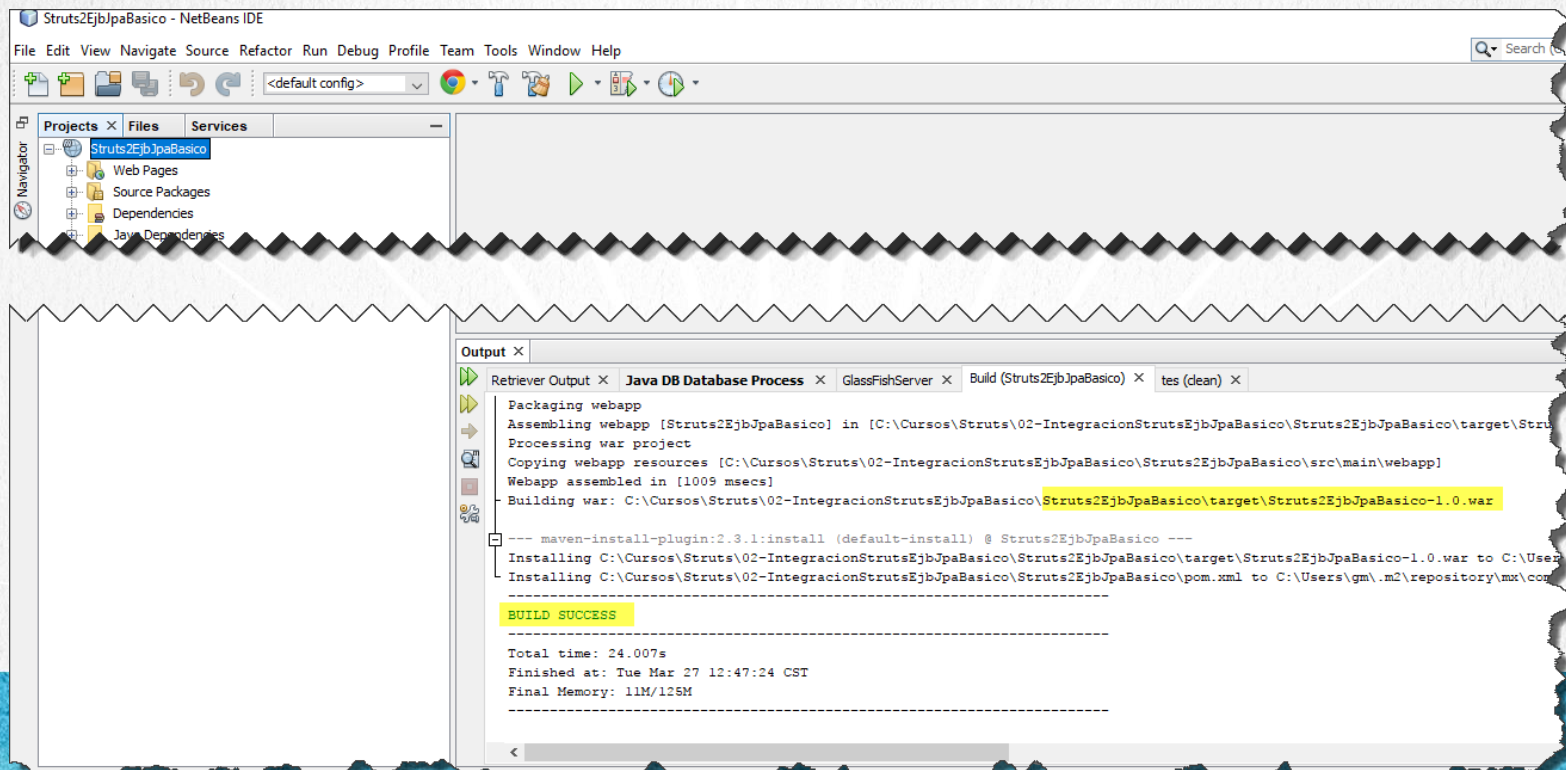
• Para que se descarguen las nuevas librerías si fuera necesario, hacemos Clean & Build al proyecto. Si por alguna razón este proceso falla, se debe desactivar cualquier software como antivirus, Windows defender o firewall durante este proceso para que no se impida la descarga de archivos .jar de Java. Una vez terminado se pueden volver a activar estos servicios. Este proceso puede demorar varios minutos dependiendo de su velocidad de internet:





# 5. HACEMOS CLEAN & BUILD

- Si ya no fue necesario descargar ninguna librería debido a que podría ya tener todas descargadas, el proceso es más rápido. Al final deberemos observar lo siguiente:



## 6. CREAMOS UN ARCHIVO XML

Vamos a crear a continuación el archivo persistence.xml

Este archivo es el que nos permite configurar la tecnología de JPA (Java Persistence API).

Hasta este punto no hay ninguna diferencia con la integración con Spring, ya que estamos utilizando la misma tecnología para la capa de datos que es JPA.

Veamos como queda nuestro archivo persistence.xml.



Experiencia y Conocimiento para tu vida

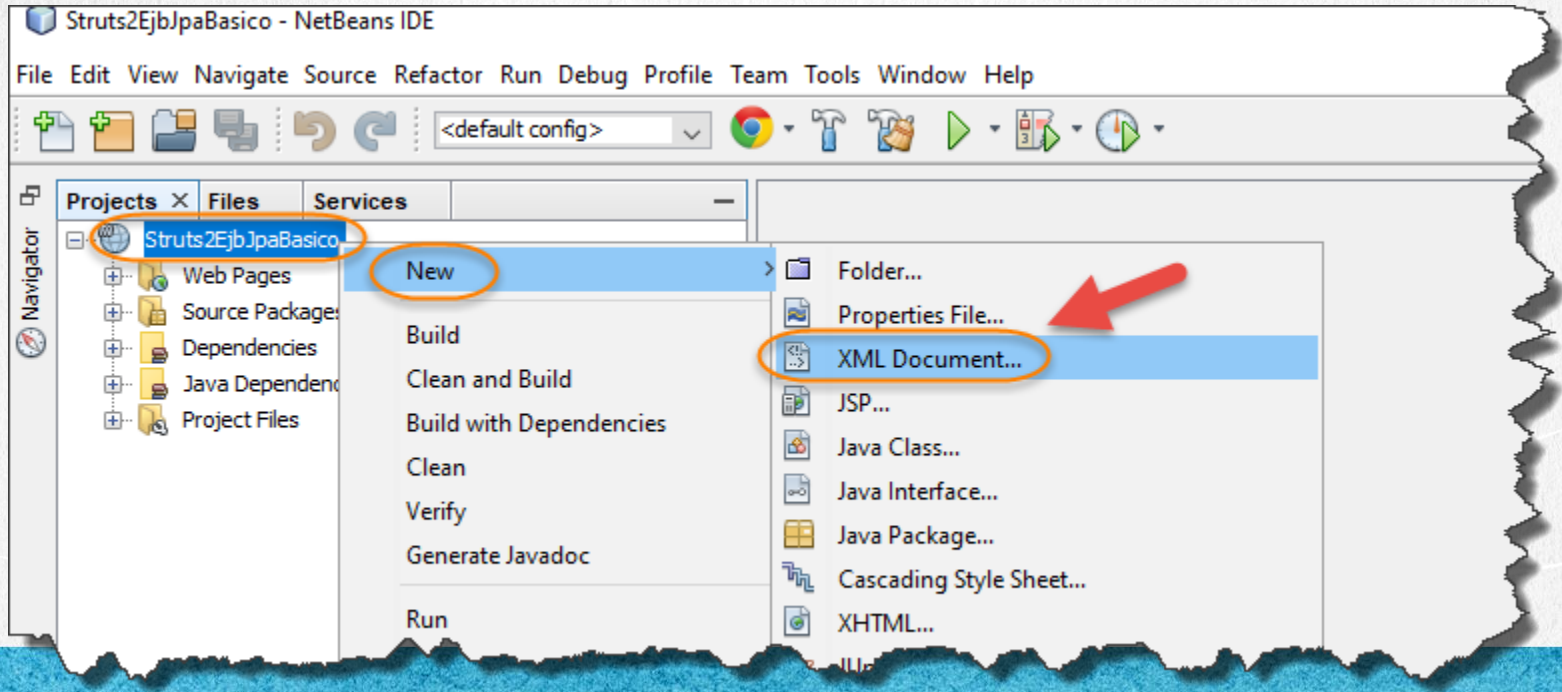
**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)



## 6. CREAMOS UN ARCHIVO XML

- Creamos el archivo persistence.xml y lo agregamos a la carpeta siguiente según se muestra:



**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)



## 6. CREAMOS UN ARCHIVO XML

- El nombre del archivo es web, no es necesario agregar la extensión, la agrega en automático el IDE ya que es un documento de tipo XML. Por último proporcionamos la ruta:

**New XML Document**

**Steps**

1. Choose File Type
2. **Name and Location**
3. Select Document Type
4. ...

**Name and Location**

File Name:

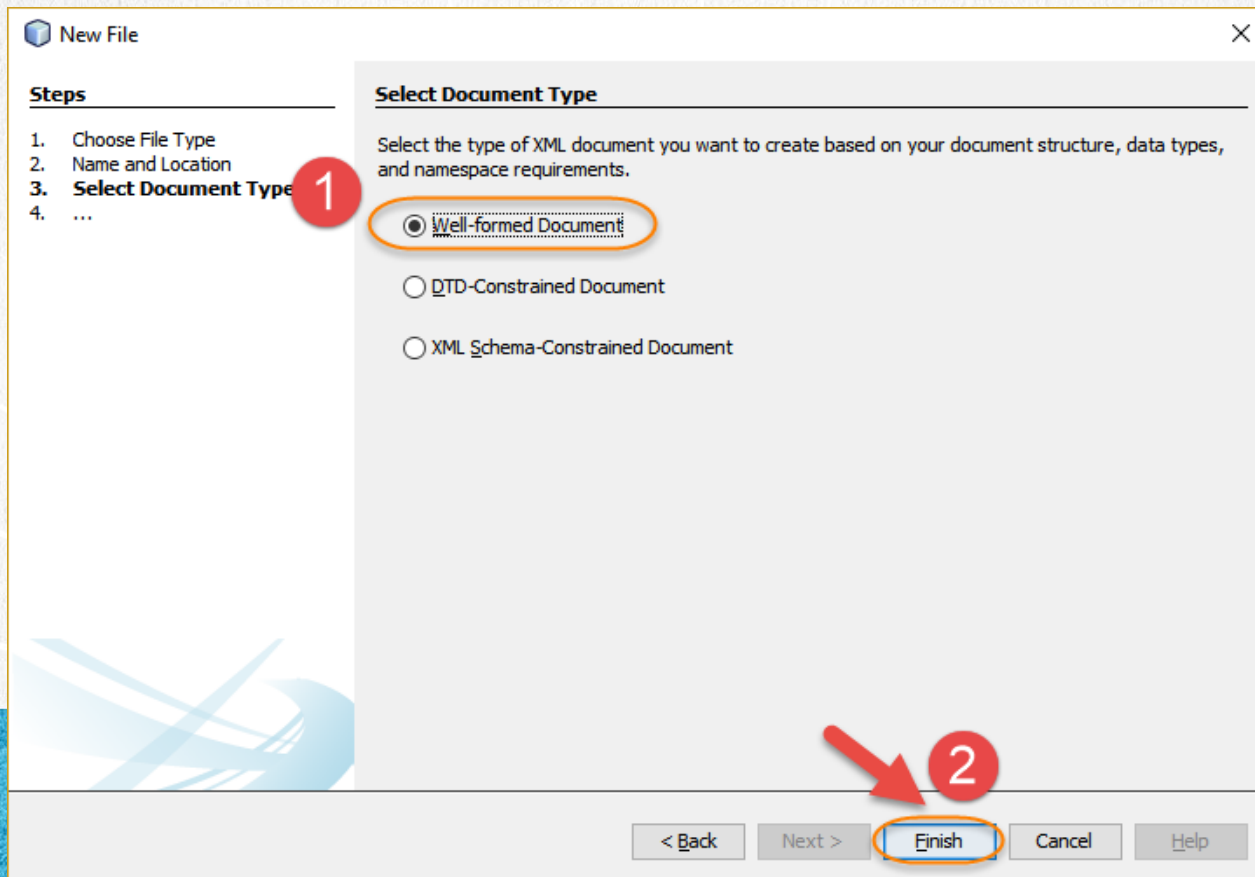
Project:

Folder:

Created File:

## 6. CREAMOS UN ARCHIVO XML

- Seleccionamos el tipo indicado y damos click en finalizar.



# PASO 7. MODIFICAMOS EL CÓDIGO

## Archivo persistence.xml:

Clic para ver el  
archivo

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
  version="2.2">
  <persistence-unit name="PersistenceUnit" transaction-type="JTA">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <jta-data-source>jdbc/PersonaDb</jta-data-source>
    <properties>
      <property name="eclipselink.logging.level" value="FINE"/>
      <property name="eclipselink.logging.parameters" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```



## 8. CREAMOS UN ARCHIVO XML

Vamos a crear a continuación el archivo web.xml

Este archivo es el que nos permite unir una aplicación Java Web con el framework de Struts, configurando el filtro de Struts en el archivo web.xml.

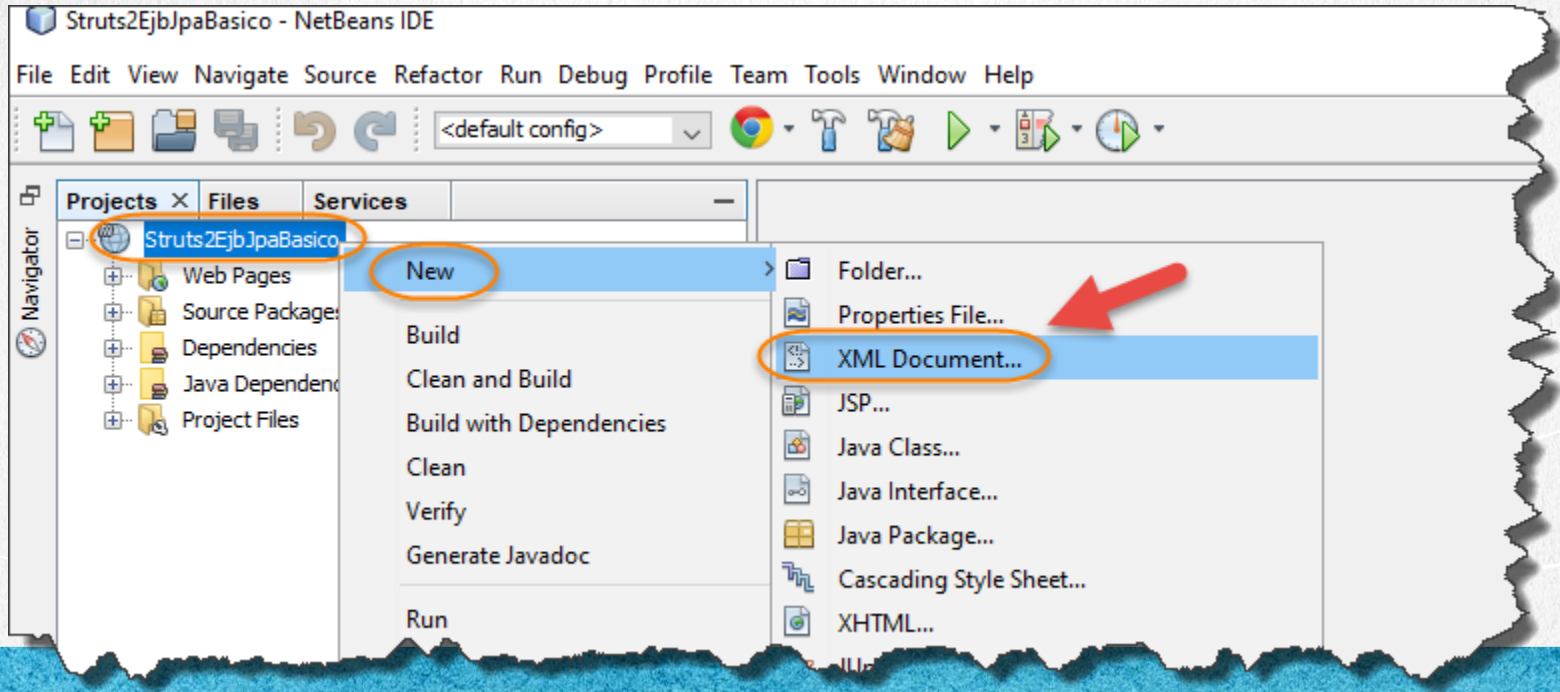
A diferencia de la integración con Spring, no es necesario agregar ninguna configuración más, debido a que vamos a utilizar las tecnologías de Java EE, las cuales ya están integradas en nuestra aplicación por utilizar el API de Java EE.

En este caso si podemos utilizar la última versión del namespace del archivo web.xml

Veamos como queda nuestro archivo web.xml.

## 8. CREAMOS UN ARCHIVO XML

- Creamos el archivo web.xml y lo agregamos a la carpeta WEB-INF según se muestra:



**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)



## 8. CREAMOS UN ARCHIVO XML

- El nombre del archivo es web, no es necesario agregar la extensión, la agrega en automático el IDE ya que es un documento de tipo XML. Por último proporcionamos la ruta:

**New XML Document**

**Steps**

1. Choose File Type
- 2. Name and Location**
3. Select Document Type
4. ...

**Name and Location**

File Name:

Project:

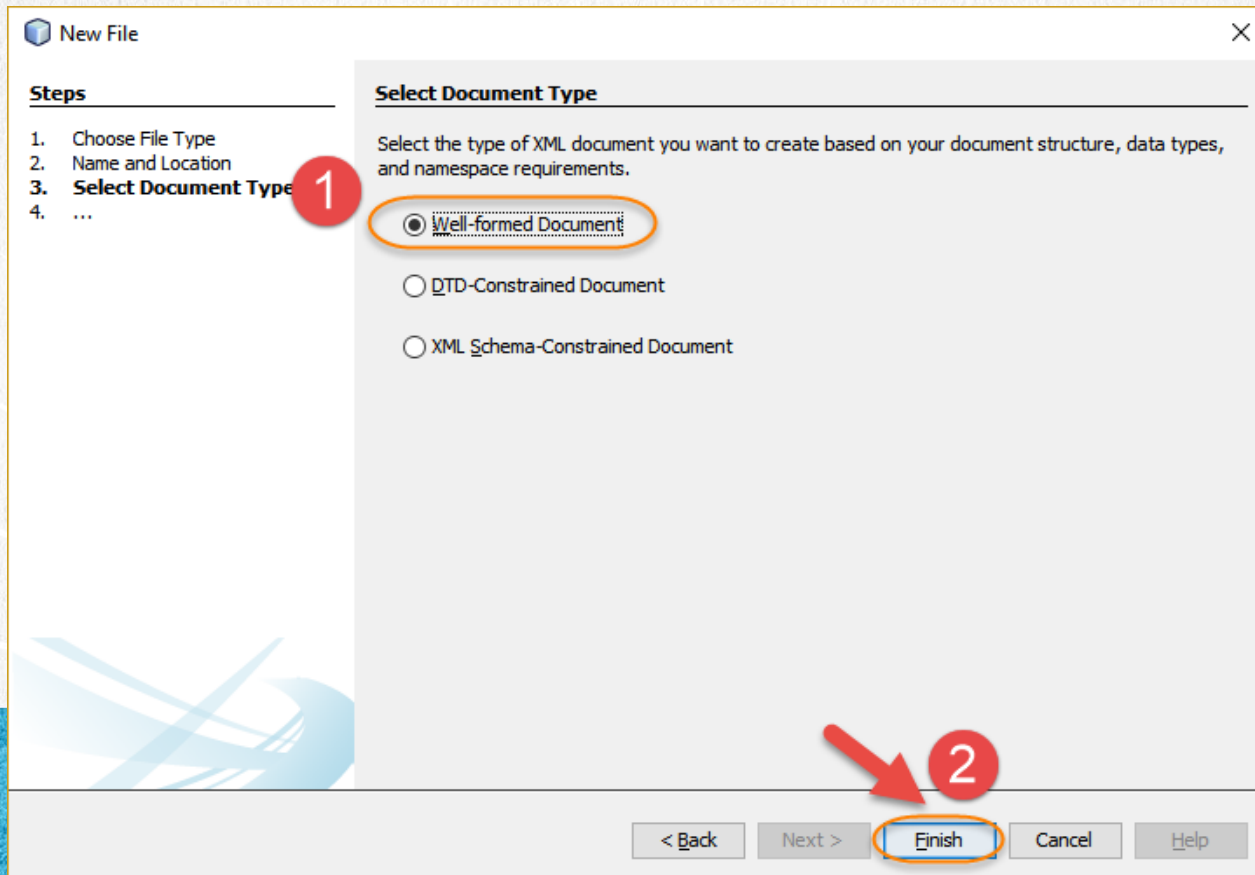
Folder:

Created File:



## 8. CREAMOS UN ARCHIVO XML

- Seleccionamos el tipo indicado y damos click en finalizar.



# PASO 9. MODIFICAMOS EL CÓDIGO

## Archivo web.xml:

Clic para ver el  
archivo

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="4.0"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd">

  <!-- Integracion con Struts Framework-->
  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

</web-app>
```

# 10. CREAR UNA CLASE JAVA

La clase de entidad Persona.java que vamos a crear a continuación es la clase que va a ser utilizada por la tecnología JPA para representar un registro de la tabla de persona de la base de datos.

Vamos a utilizar anotaciones de JPA donde sea necesario para personalizar la clase Persona.java y así pueda representar exactamente a los registros de la tabla persona de la base de datos.

A este tipo de clase también se les conoce como clase de dominio.

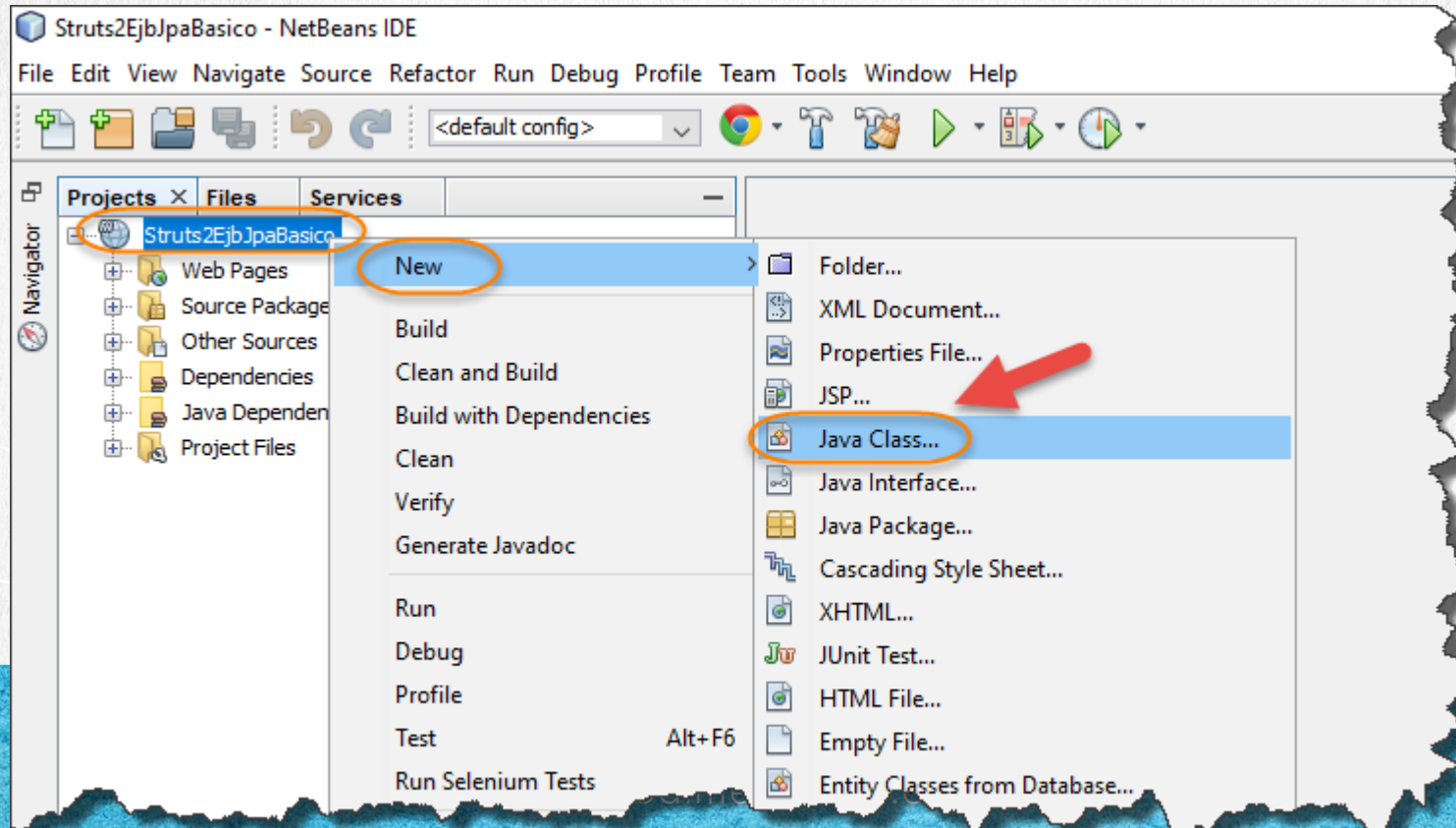
Esta clase no tiene ninguna diferencia con la integración con Spring, ya que utiliza la tecnología de JPA , así que es el mismo código.

Veamos como queda nuestra clase Persona.java



# 10. CREAR UNA CLASE JAVA

- Creamos la clase Persona.java:



# 10. CREAR UNA CLASE JAVA

- Creamos la clase Persona.java:

**New Java Class**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File:

# PASO 11. MODIFICAMOS EL CÓDIGO

## Archivo Persona.java:

Clic para ver el archivo

```
package mx.com.gm.capadatos.domain;

import java.io.Serializable;
import java.util.Objects;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Persona implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_persona")
    private Long idPersona;

    private String nombre;

    @Column(name = "apellido_paterno")
    private String apellidoPaterno;
```



# PASO 11. MODIFICAMOS EL CÓDIGO

## Archivo Persona.java:

Clic para ver el archivo

```
@Column(name = "apellido_materno")
private String apellidoMaterno;

private String email;

public Persona() {
}

public Persona(Long idPersona) {
    this.idPersona = idPersona;
}

public Long getIdPersona() {
    return idPersona;
}

public void setIdPersona(Long idPersona) {
    this.idPersona = idPersona;
}

public String getNombre() {
    return nombre;
}
```

# PASO 11. MODIFICAMOS EL CÓDIGO

## Archivo Persona.java:

Clic para ver el archivo

```
public void setNombre(String nombre) {  
    this.nombre = nombre;  
}  
  
public String getApellidoPaterno() {  
    return apellidoPaterno;  
}  
  
public void setApellidoPaterno(String apellidoPaterno) {  
    this.apellidoPaterno = apellidoPaterno;  
}  
  
public String getApellidoMaterno() {  
    return apellidoMaterno;  
}  
  
public void setApellidoMaterno(String apellidoMaterno) {  
    this.apellidoMaterno = apellidoMaterno;  
}  
  
public String getEmail() {  
    return email;  
}
```

# PASO 11. MODIFICAMOS EL CÓDIGO

## Archivo Persona.java:

Clic para ver el archivo

```
public void setEmail(String email) {
    this.email = email;
}

@Override
public String toString() {
    return "Persona [idPersona=" + idPersona + ", nombre=" + nombre
        + ", apePaterno=" + apellidoPaterno + ", apeMaterno=" + apellidoMaterno
        + ", email=" + email + "]";
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 79 * hash + Objects.hashCode(this.idPersona);
    return hash;
}
```



# PASO 11. MODIFICAMOS EL CÓDIGO

## Archivo Persona.java:

Clic para ver el archivo

```
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Persona other = (Persona) obj;
    if (!Objects.equals(this.idPersona, other.idPersona)) {
        return false;
    }
    return true;
}
}
```

## 12. CREAR UNA INTERFACE JAVA

Vamos a crear una interface Java. Recordemos que es una buena práctica programar utilizando interfaces para separar las capas de nuestra aplicación Java. Así que crearemos una interface y posteriormente su implementación.

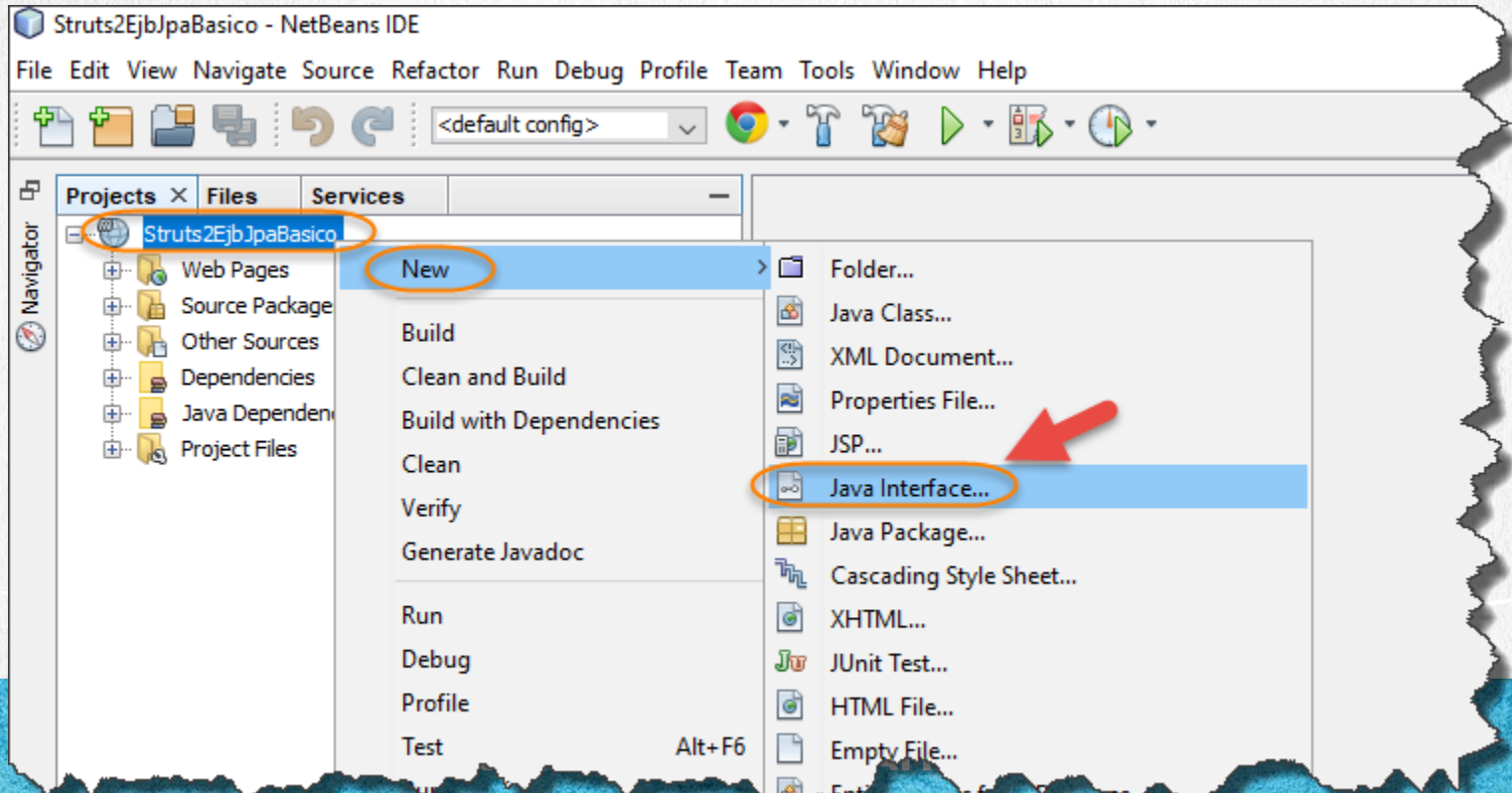
En esta interface aplicaremos el patrón de diseño DAO (Data Access Object), ya que es la interface que nos permitirá aplicar las operaciones sobre la clase de entidad de Persona, métodos como listar, agregar, modificar, eliminar objetos de tipo Persona.

Las interfaces que crearemos no tienen ninguna diferencia entre Spring y Java EE, ya que ambas tecnologías utilizan POJO's (Plain Old Java Objects) o clases simples de Java, en este caso interfaces simples de Java.

El nombre de la interface es PersonaDao.java, veamos como queda esta interface:

# 12. CREAR UNA INTERFACE JAVA

- Creamos la interface PersonaDao.java:





# 12. CREAR UNA INTERFACE JAVA

- Creamos la interface PersonaDao.java:

**New Java Interface**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File:

< Back   Next >   **Finish**   Cancel   Help

# PASO 13. MODIFICAMOS EL CÓDIGO

## Archivo PersonaDao.java:

Clic para ver el archivo

```
package mx.com.gm.capadatos;

import java.util.List;

import mx.com.gm.capadatos.domain.Persona;

public interface PersonaDao {

    void insertPersona(Persona persona);

    void updatePersona(Persona persona);

    void deletePersona(Persona persona);

    Persona findPersonaById(long idPersona);

    List<Persona> findAllPersonas();

    long contadorPersonas();

    Persona getPersonaByEmail(Persona persona);

}
```

# 14. CREAR UNA CLASE JAVA

Vamos a crear una clase Java llamada `PersonaDaoImpl.java` que implemente la interface `PersonaDao.java` recién creada.

Esta clase utilizará la tecnología de EJB, en lugar de Spring, y también utilizará la tecnología de JPA para realizar las operaciones sobre la base de datos y obtener la conexión a la base de datos, así como la clase de entidad `Persona.java` para poder comunicarse con la base de datos y realizar las operaciones descritas por la interface.

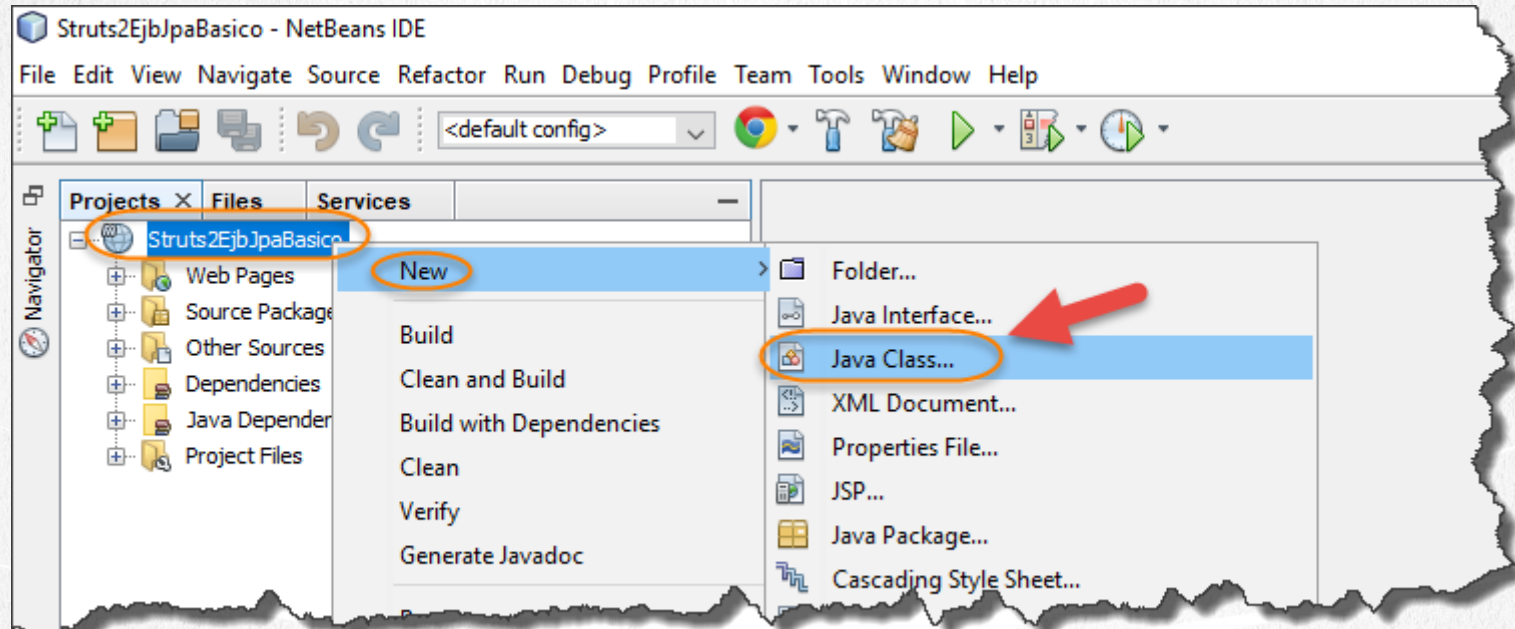
La única diferencia entre el EJB y Spring es la anotación que se utiliza para identificar la clase, por un lado Spring utiliza la anotación `@Repository` para indicar que es una clase que ejecuta operaciones en la capa de datos, y los EJB utiliza la anotación `@Stateless` para indicar que es un EJB de tipo `Stateless`, sin embargo todo lo demás es exactamente igual, por lo que podemos observar que ambas tecnologías son muy parecidas al día de hoy.

Veamos como queda la clase `PersonaDaoImpl.java` utilizando la tecnología de Java EE:



# 14. CREAR UNA CLASE JAVA

- Creamos la clase PersonaDaoImpl.java:



# 14. CREAR UNA CLASE JAVA

- Creamos la clase PersonaDaoImpl.java:

**New Java Class**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File: `isica\Struts2SpringJpaBasico\src\main\java\mx\com\gm\capadatos\PersonaDaoImpl.java`

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

# PASO 15. MODIFICAMOS EL CÓDIGO

## Archivo PersonaDaoImpl.java:

Clic para ver el archivo

```
package mx.com.gm.capadatos;

import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.CacheStoreMode;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;
import mx.com.gm.capadatos.domain.Persona;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

@Stateless
public class PersonaDaoImpl implements PersonaDao {

    Logger log = LogManager.getRootLogger();

    @PersistenceContext
    private EntityManager em;

    @Override
    public void insertPersona(Persona persona) {
        // Insertamos nuevo objeto
        em.persist(persona);
    }
}
```



# PASO 15. MODIFICAMOS EL CÓDIGO

## Archivo PersonaDaoImpl.java:

Clic para ver el archivo

```
@Override
public void updatePersona(Persona persona) {
    // Actualizamos al objeto
    em.merge(persona);
}

@Override
public void deletePersona(Persona persona) {
    em.remove(em.merge(persona));
}

@Override
public Persona findPersonaById(long idPersona) {
    return em.find(Persona.class, idPersona);
}

@Override
public List<Persona> findAllPersonas() {
    String jpql = "SELECT p FROM Persona p";
    Query query = em.createQuery(jpql);
    //Forzar a ir directamente a la base de datos para refrescar datos
    query.setHint("javax.persistence.cache.storeMode", CacheStoreMode.REFRESH);
    List<Persona> personas = query.getResultList();
    System.out.println("personas:" + personas);
    return personas;
}
```

# PASO 15. MODIFICAMOS EL CÓDIGO

## Archivo PersonaDaoImpl.java:

Clic para ver el archivo

```
@Override
public long contarPersonas() {
    String consulta = "select count(p) from Persona p";
    Query q = em.createQuery(consulta);
    long contador = (long) q.getSingleResult();
    return contador;
}

@Override
public Persona getPersonaByEmail(Persona persona) {
    String cadena = "%" + persona.getEmail() + "%"; //se usa en el like como caracteres especiales
    String consulta = "from Persona p where upper(p.email) like upper(:param1)";
    Query q = em.createQuery(consulta);
    q.setParameter("param1", cadena);
    return (Persona) q.getSingleResult();
}
```

# 16. CREAR UNA INTERFACE JAVA

Vamos a crear una interface Java `PersonaService.java`. Recordemos que es una buena práctica programar utilizando interfaces para separar las capas de nuestra aplicación Java. Así que crearemos una interface y posteriormente su implementación.

En este caso no hay diferencias entre la implementación de Spring y Java EE.

Veamos como queda esta interface `PersonaService.java`:



Experiencia y Conocimiento para tu vida

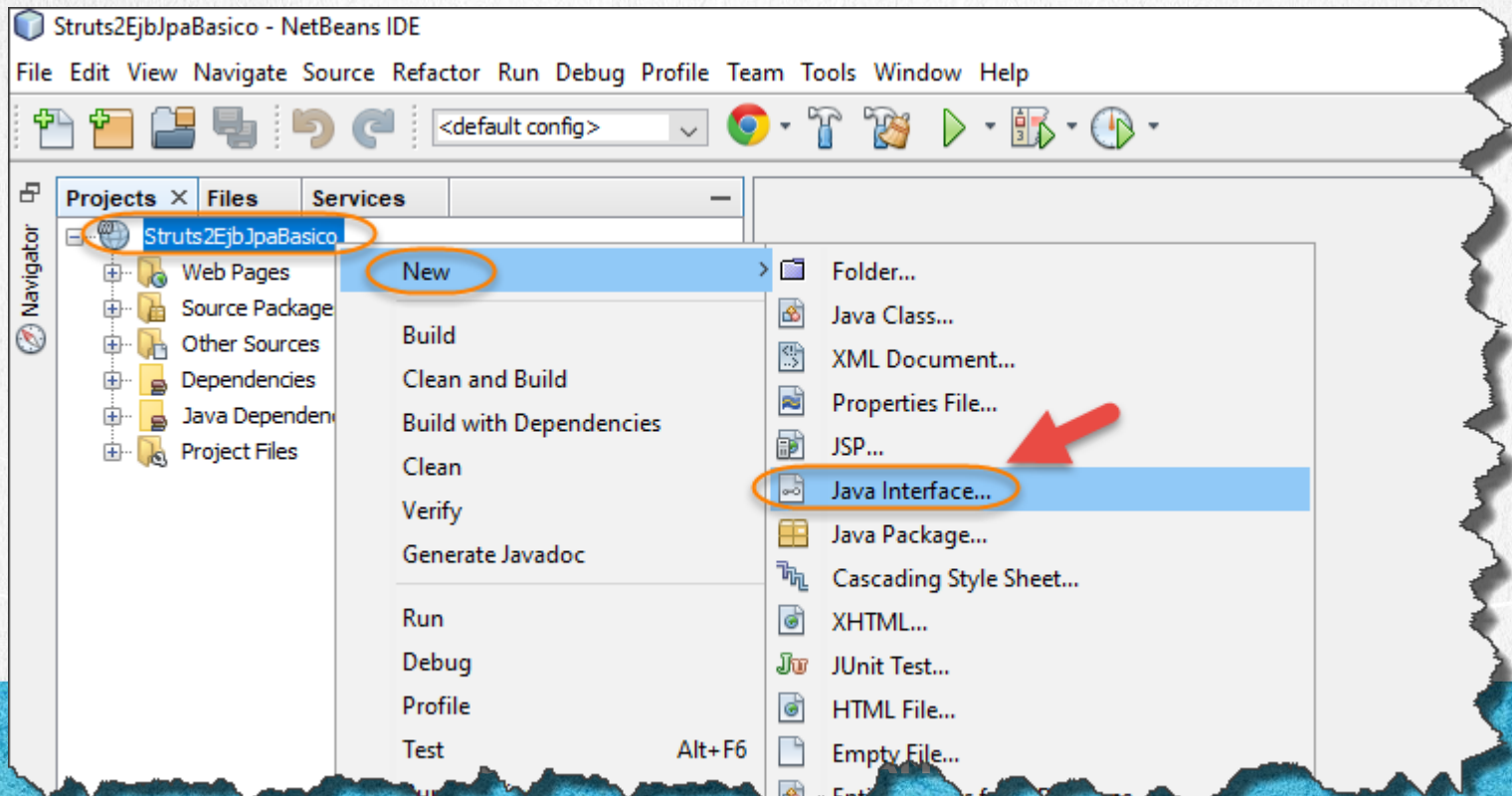
**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)



# 16. CREAR UNA INTERFACE JAVA

- Creamos la interface PersonaService.java:



# 16. CREAR UNA INTERFACE JAVA

- Creamos la interface PersonaService.java:

**New Java Interface**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File:

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

# PASO 17. MODIFICAMOS EL CÓDIGO

## Archivo PersonaService.java:

Clic para ver el archivo

```
package mx.com.gm.capaservicio;

import java.util.List;
import mx.com.gm.capadatos.domain.Persona;

public interface PersonaService {

    public List<Persona> listarPersonas();

    public Persona recuperarPersona(Persona persona);

    public void agregarPersona(Persona persona);

    public void modificarPersona(Persona persona);

    public void eliminarPersona(Persona persona);

    public long contarPersonas();

}
```



# 18. CREAR UNA CLASE JAVA

Vamos a crear una clase Java llamada `PersonaServiceImpl.java` que implemente la interface `PersonaService.java` recién creada.

Esta clase utilizará la tecnología de EJB para manejar de manera automática el concepto de transacciones.

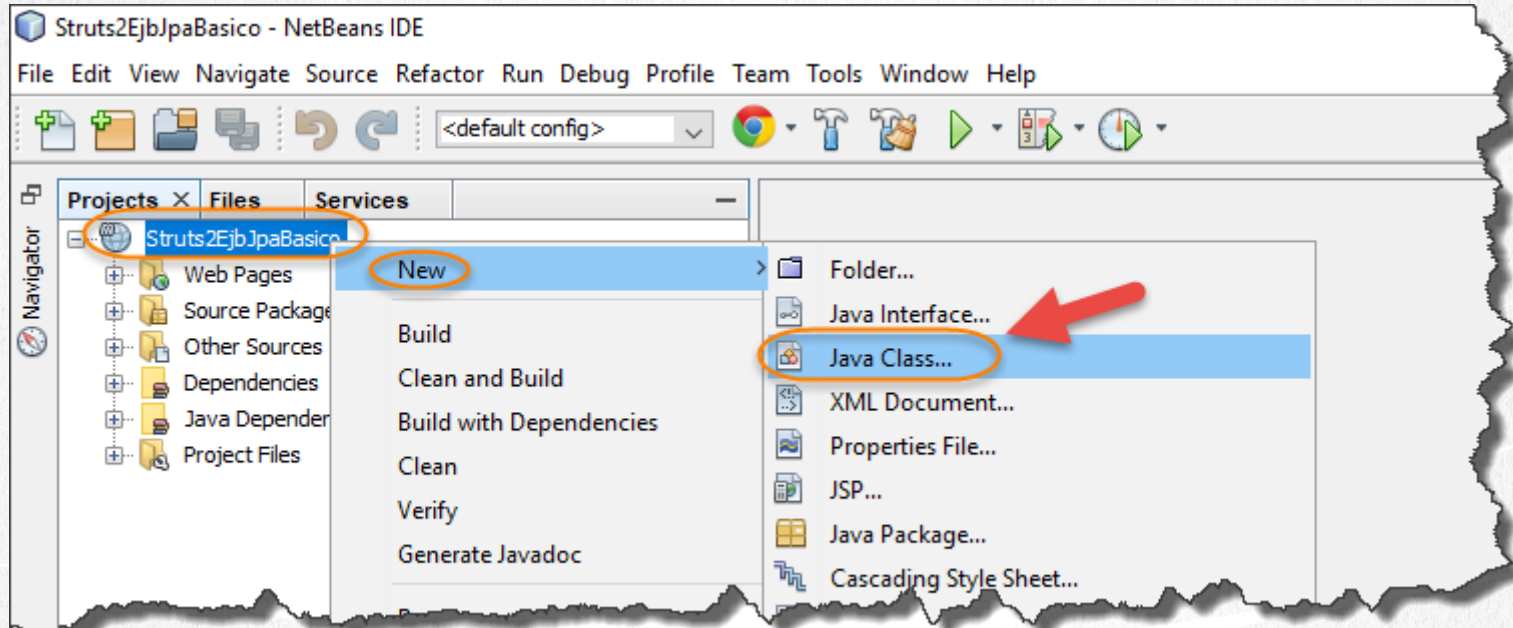
A diferencia de la clase de Spring, vamos a utilizar la anotación `@Stateless` en lugar de `@Service` que se usó con Spring.

Y para hacer la inyección de dependencias con la capa de datos, vamos a utilizar CDI (Context and Dependency Injection) por lo tanto usaremos la anotación `@Inject`, en lugar de `@Autowire` que usamos con Spring Framework.

Veamos como queda la clase `PersonaServiceImpl.java`

# 18. CREAR UNA CLASE JAVA

- Creamos la clase PersonaServiceImpl.java:





# 18. CREAR UNA CLASE JAVA

- Creamos la clase PersonaServiceImpl.java:

**New Java Interface**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File:

< Back   Next >   **Finish**   Cancel   Help



# PASO 19. MODIFICAMOS EL CÓDIGO

## Archivo PersonaServiceImpl.java:

[Clic para ver el archivo](#)

```
package mx.com.gm.capaservicio;

import java.util.List;
import javax.ejb.Stateless;
import javax.inject.Inject;
import mx.com.gm.capadatos.PersonaDao;
import mx.com.gm.capadatos.domain.Persona;

@Stateless
public class PersonaServiceImpl implements PersonaService {

    @Inject
    private PersonaDao personaDao;

    @Override
    public List<Persona> listarPersonas() {
        return personaDao.findAllPersonas();
    }

    @Override
    public Persona recuperarPersona(Persona persona) {
        return personaDao.findPersonaById(persona.getIdPersona());
    }
}
```

# PASO 19. MODIFICAMOS EL CÓDIGO

## Archivo PersonaServiceImpl.java:

Clic para ver el archivo

```
@Override
public void agregarPersona(Persona persona) {
    personaDao.insertPersona(persona);
}

@Override
public void modificarPersona(Persona persona) {
    personaDao.updatePersona(persona);
}

@Override
public void eliminarPersona(Persona persona) {
    personaDao.deletePersona(persona);
}

@Override
public long contarPersonas() {
    return personaDao.contadorPersonas();
}
}
```

# PASO 20. CREAR UNA CLASE JAVA

La clase `MostrarPersonaAction.java` que vamos a crear a continuación va a hacer las veces de Controlador (Action) y Modelo (Bean).

Vamos a extender de la clase `ActionSupport` y para sobrescribiremos el método `execute`.

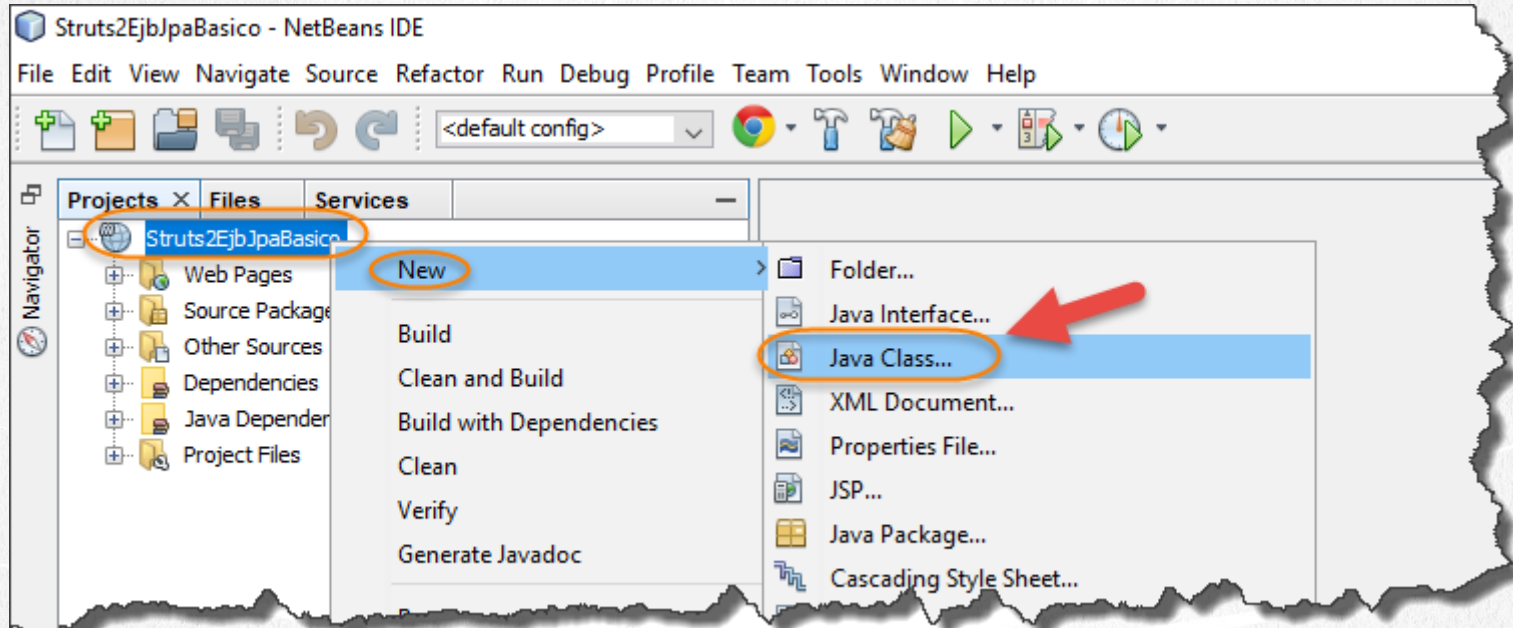
El modelo lo obtendremos con ayuda de la interface de servicio, el cual inyectaremos con ayuda de Java EE y el plug-in de integración entre Struts y CDI de Java EE que agregamos al archivo `pom.xml`. Así que para hacer la inyección de dependencia utilizaremos la anotación `@Inject`.

Recordemos que debemos respetar las convenciones de Struts2, así que esta clase debe estar dentro de un paquete que contenga la palabra: `struts`, `struts2`, `action` o `actions`, además debe terminar con la palabra `Action`.



# PASO 20. CREAR UNA CLASE JAVA

- Creamos la clase MostrarPersonaAction.java:



# PASO 20. CREAR UNA CLASE JAVA

- Creamos la clase `MostrarPersonaAction.java`:

**New Java Class**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name: `MostrarPersonaAction`

Project: `Struts2EjbJpaBasico`

Location: `Source Packages`

Package: `mx.com.gm.actions`

Created File: `asico\Struts2EjbJpaBasico\src\main\java\mx\com\gm\actions\MostrarPersonaAction.java`

< Back   Next >   **Finish**   Cancel   Help

# PASO 21. MODIFICAMOS EL CÓDIGO

## Archivo MostrarPersonaAction.java:

Clic para ver el archivo

```
package mx.com.gm.actions;

import com.opensymphony.xwork2.ActionSupport;
import java.util.List;
import javax.inject.Inject;
import mx.com.gm.capadatos.domain.Persona;
import mx.com.gm.capaservicio.PersonaService;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class MostrarPersonaAction extends ActionSupport {

    private long contadorPersonas;

    Logger log = LogManager.getLogger(MostrarPersonaAction.class);

    @Inject
    private PersonaService personaService;

    private List<Persona> personas;
```



# PASO 21. MODIFICAMOS EL CÓDIGO

Archivo MostrarPersonaAction.java:

Clic para ver el archivo

```
@Override
public String execute() {
    this.personas = personaService.listarPersonas();
    this.contadorPersonas = personaService.contarPersonas();
    return SUCCESS;
}

public List<Persona> getPersonas() {
    return personas;
}

public void setPersonas(List<Persona> personas) {
    this.personas = personas;
}

public long getContadorPersonas() {
    return contadorPersonas;
}

public void setContadorPersonas(long contadorPersonas) {
    this.contadorPersonas = contadorPersonas;
}
}
```

# PASO 22. CREAR EL ARCHIVO DE PROPIEDADES

Creamos un archivo `MostrarPersonaAction.properties`. Este archivo tiene los mensajes que utilizaremos en las páginas JSP de Struts.

Veamos como queda este archivo `MostrarPersonaAction.properties`



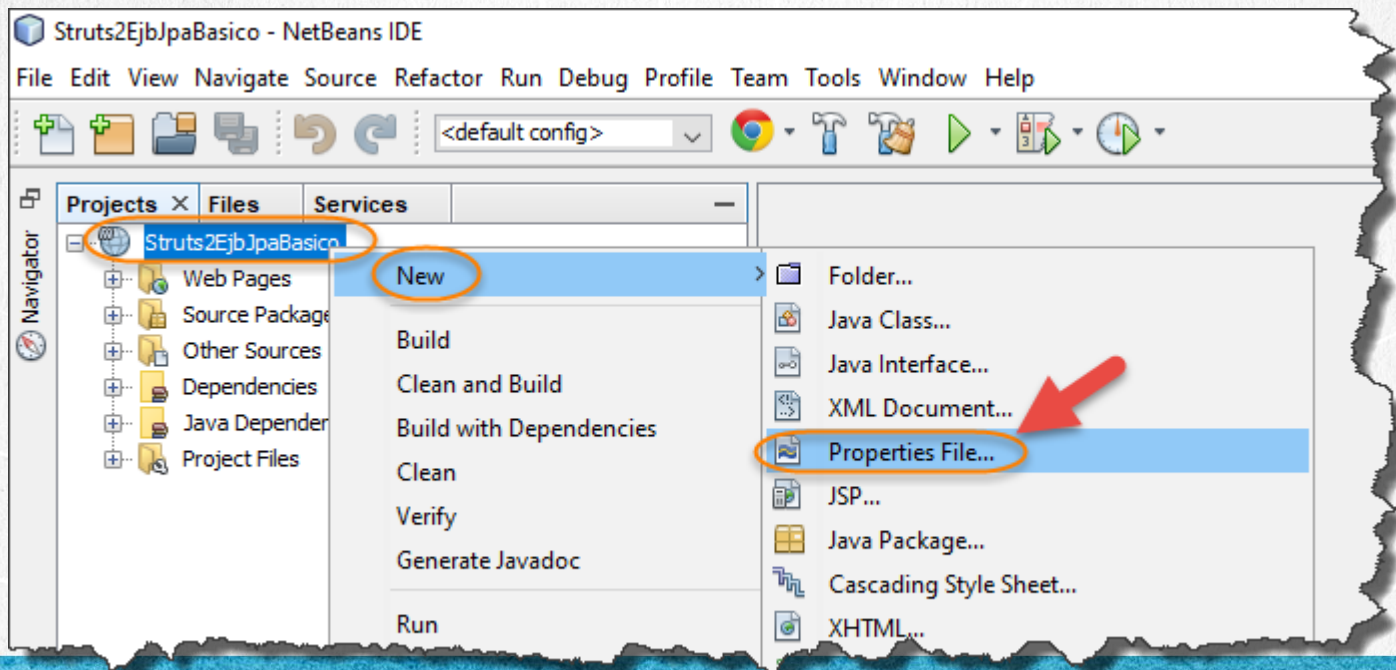
Experiencia y Conocimiento para tu vida

**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

# PASO 22. CREAR EL ARCHIVO DE PROPIEDADES

- Creamos el archivo `MostrarPersonaAction.properties` como sigue:



**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)



# PASO 22. CREAR EL ARCHIVO DE PROPIEDADES

- Depositamos el archivo en la carpeta de resources según se muestra:

**New Properties File**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

File Name:

Project:

Folder:

Created File:

**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

# PASO 23. MODIFICAMOS EL CÓDIGO

[Archivo MostrarPersonaAction.properties:](#)

Clic para ver el archivo

```
persona.titulo: Personas con Struts 2
persona.contador: No. Registros Encontrados
persona.boton: Refrescar
p.idPersona: idPersona
p.nombre: Nombre
p.apePat: Apellido Paterno
p.apeMat: Apellido Materno
p.email: Email
```



# PASO 24. MODIFICAMOS EL ARCHIVO INDEX.HTML

En automático el IDE agrega un archivo llamado index.html. Sin embargo si este archivo no se crea debemos agregarlo al proyecto a nivel raíz de Web Pages.

El archivo index.html realmente aún no forma parte del framework de Struts, sin embargo será el punto de entrada para que se ejecute el framework de Struts, ya que desde este archivo indicaremos cual es la acción que deseamos que se ejecute.

En este ejercicio el path que utilizaremos será: [mostrar-persona](#)



Experiencia y Conocimiento para tu vida

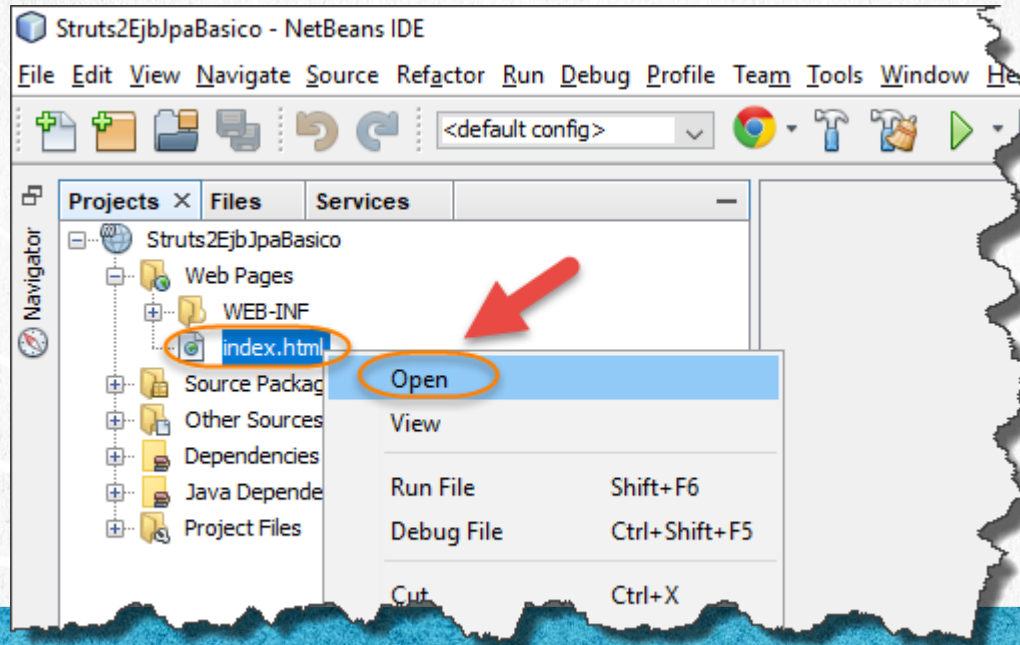
**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)



# PASO 24. MODIFICAMOS EL ARCHIVO INDEX.HTML

- Modificamos el archivo index.html. En caso de que este archivo no exista a nivel raíz de la carpeta Web Pages según se muestra:



**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

# PASO 24. MODIFICAMOS EL CÓDIGO

Archivo index.html:

Clic para ver el archivo

```
<!DOCTYPE html>
<html>
  <head>
    <title>Inicio</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <a href="mostrar-persona">Ir a mostrar Persona</a>
  </body>
</html>
```



## PASO 25. CREAR EL ARCHIVO JSP

Ahora creamos el archivo: mostrar-persona.jsp. Recordar que este nombre corresponde con el path que se va a utilizar para llamar la acción correspondiente (MostrarPersonaAction.java), así que separamos por un guión medio cada palabra de la clase de tipo Action.

Además debemos depositar este JSP en la carpeta [/WEB-INF/content](#) según hemos visto en el tema de convenciones de Struts 2.



Experiencia y Conocimiento para tu vida

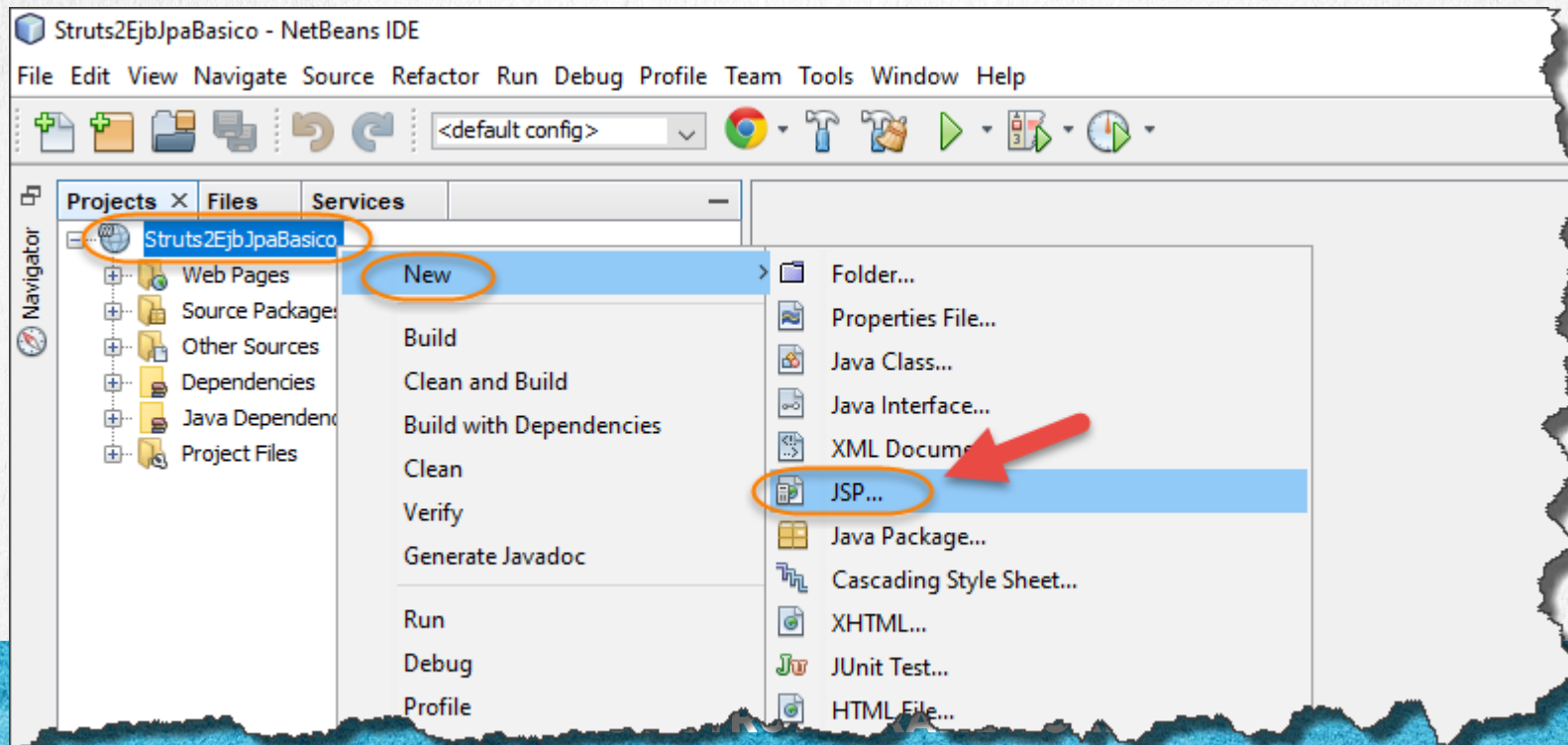
**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)



# PASO 25. CREAR EL ARCHIVO JSP

- Creamos el archivo mostrar-persona.jsp:



# PASO 25. CREAR EL ARCHIVO JSP

- Creamos el archivo mostrar-persona.jsp en la ruta mostrada:

New JSP

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

File Name:

Project:

Location:

Folder:

Created File:

Options:

☒ JSP File (Standard Syntax) ☐ Create as a JSP Segment

☐ JSP Document (XML Syntax)

Description:

# PASO 26. MODIFICAMOS EL CÓDIGO

[Archivo mostrar-persona.jsp:](#)

Clic para ver el archivo

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE html>
<html>
  <head>
    <title><s:text name="persona.titulo" /></title>
  </head>
  <body>
    <h1><s:text name="persona.titulo" /></h1>

    <s:if test="personas.size() > 0">
      <div>
        <table border="1">
          <tr>
            <th><s:text name="p.idPersona" /></th>
            <th><s:text name="p.nombre" /></th>
            <th><s:text name="p.apePat" /></th>
            <th><s:text name="p.apeMat" /></th>
            <th><s:text name="p.email" /></th>
          </tr>
```

**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)



# PASO 26. MODIFICAMOS EL CÓDIGO

Archivo mostrar-persona.jsp:

Clic para ver el archivo

```
<s:iterator value="personas">
  <tr>
    <td><s:property value="idPersona" /></td>
    <td><s:property value="nombre" /></td>
    <td><s:property value="apellidoPaterno" /></td>
    <td><s:property value="apellidoMaterno" /></td>
    <td><s:property value="email" /></td>
  </tr>
</s:iterator>
</table>
</div>
</s:if>

<s:form>
  <s:submit key="persona.boton" name="submit" />
</s:form>

<div><s:text name="persona.contador" />: <s:property value="contadorPersonas" /></div>
</body>
</html>
```

**CURSO STRUTS FRAMEWORK**

www.globalmentoring.com.mx

# PASO 27. CREAR EL ARCHIVO LOG4J2.XML

Creamos un archivo log4j2.xml. El API de log4j nos permite manejar el log o bitácora de una aplicación Java de manera más simple.

Este archivo lo depositamos en la ruta de recursos del proyecto maven. Si no se usa maven entonces el archivo se debe depositar a nivel raíz del src del código Java.



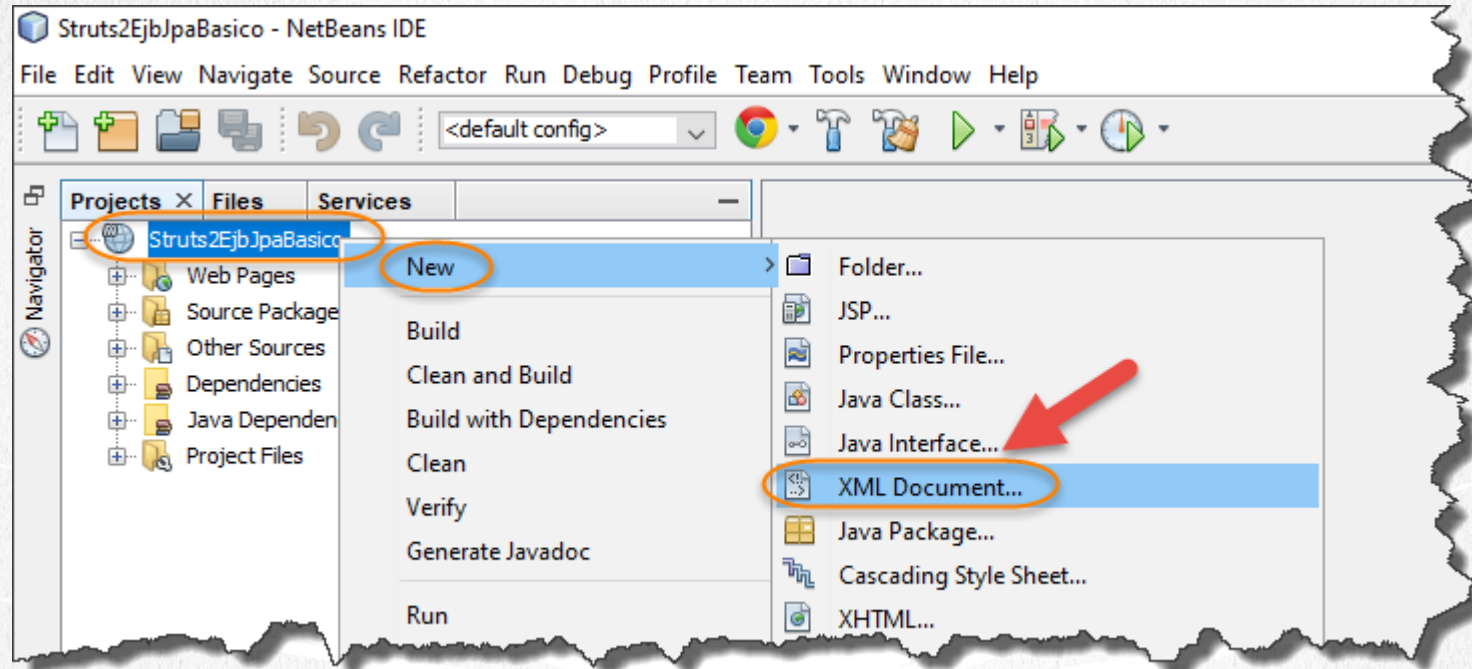
Experiencia y Conocimiento para tu vida

**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

# PASO 27. CREAR EL ARCHIVO LOG4J2.XML

- Creamos el archivo log4j2.xml como sigue:





# PASO 27. CREAR EL ARCHIVO LOG4J2.XML

- Depositamos el archivo en la carpeta de resources según se muestra:

**New XML Document**

**Steps**

1. Choose File Type
- 2. Name and Location**
3. Select Document Type
4. ...

**Name and Location**

File Name:

Project:

Folder:

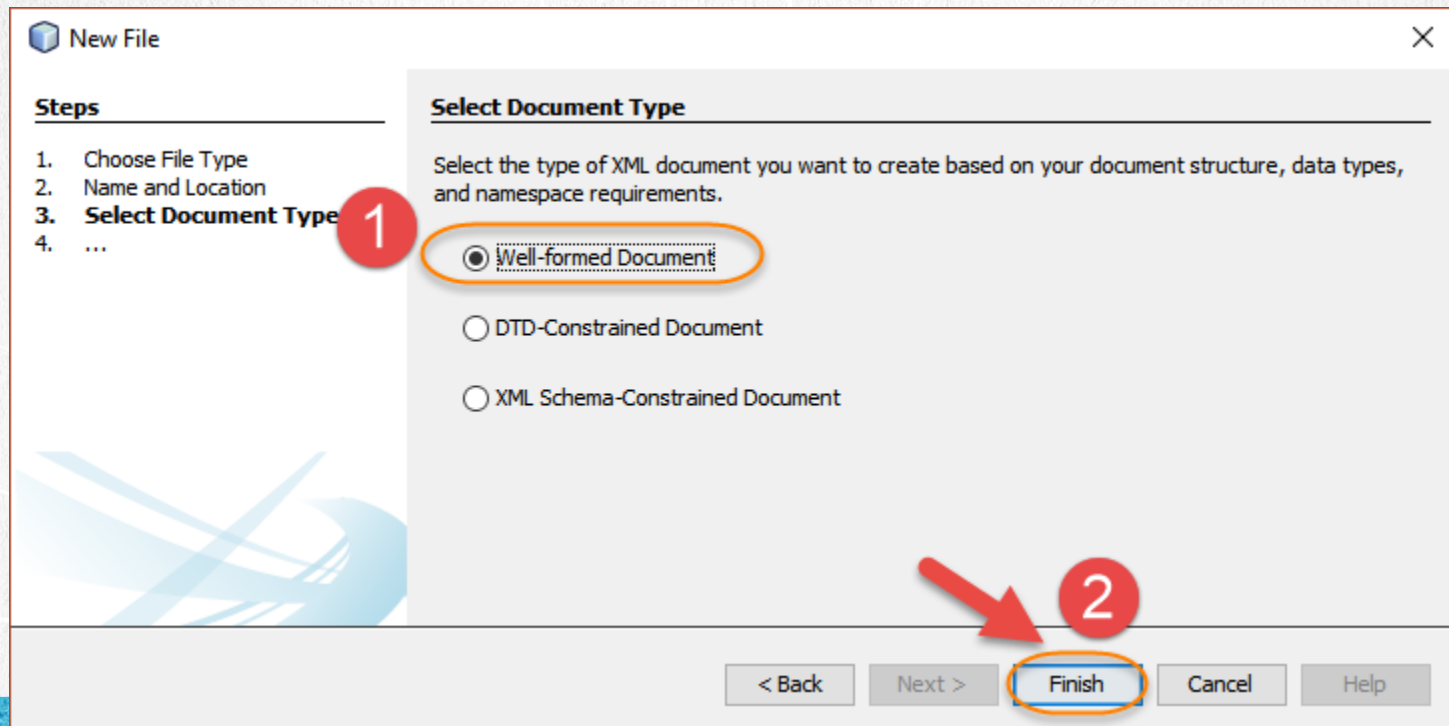
Created File:

**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

# PASO 27. CREAR EL ARCHIVO LOG4J2.XML

- Seleccionamos la opción mostrada:



**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

# PASO 28. MODIFICAMOS EL CÓDIGO

Archivo log4j2.xml:

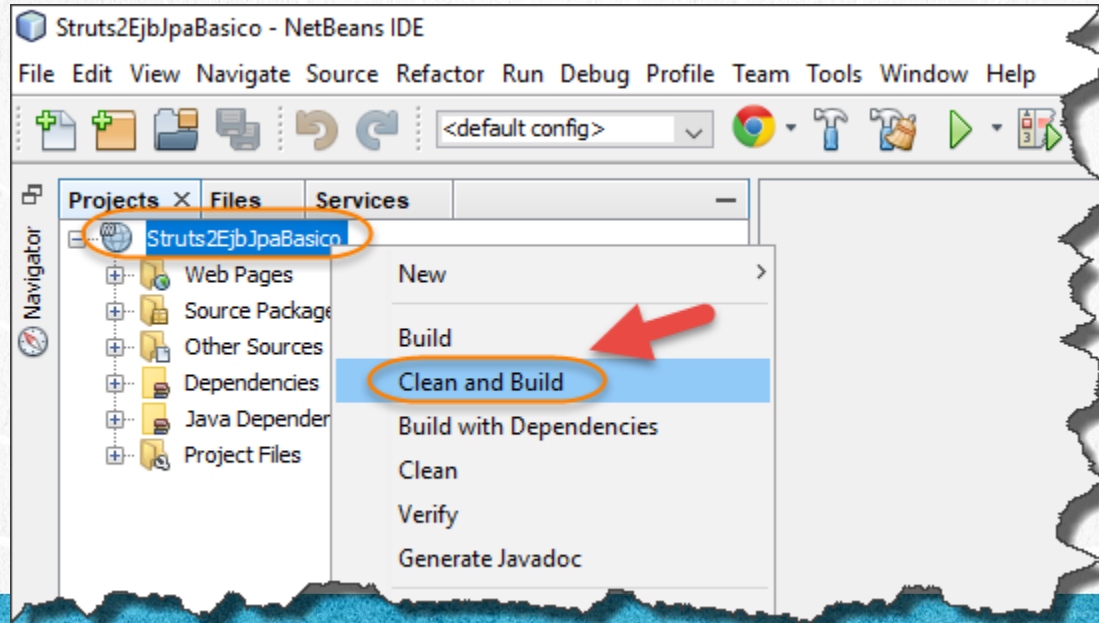
Clic para ver el archivo

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
  <Appenders>
    <Console name="STDOUT" target="SYSTEM_OUT">
      <PatternLayout pattern="%F:%L) - %m%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Logger name="com.opensymphony.xwork2" level="info"/>
    <Logger name="org.apache.struts2" level="info"/>
    <Root level="info">
      <AppenderRef ref="STDOUT"/>
    </Root>
  </Loggers>
</Configuration>
```



# PASO 29. HACEMOS CLEAN & BUILD

- Antes de ejecutar la aplicación, hacemos Clean & Build para asegurarnos que tenemos todo listo:

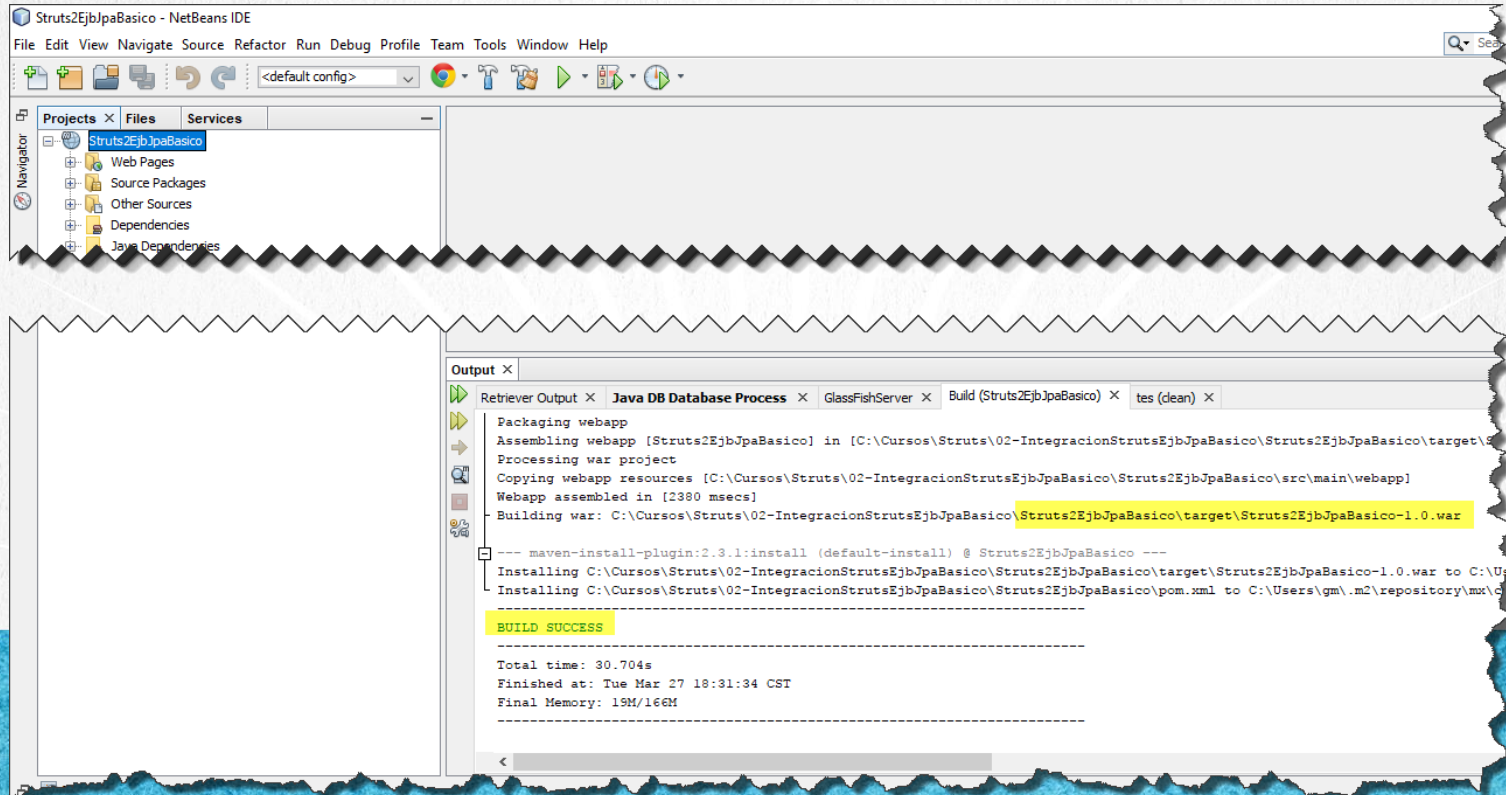


**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

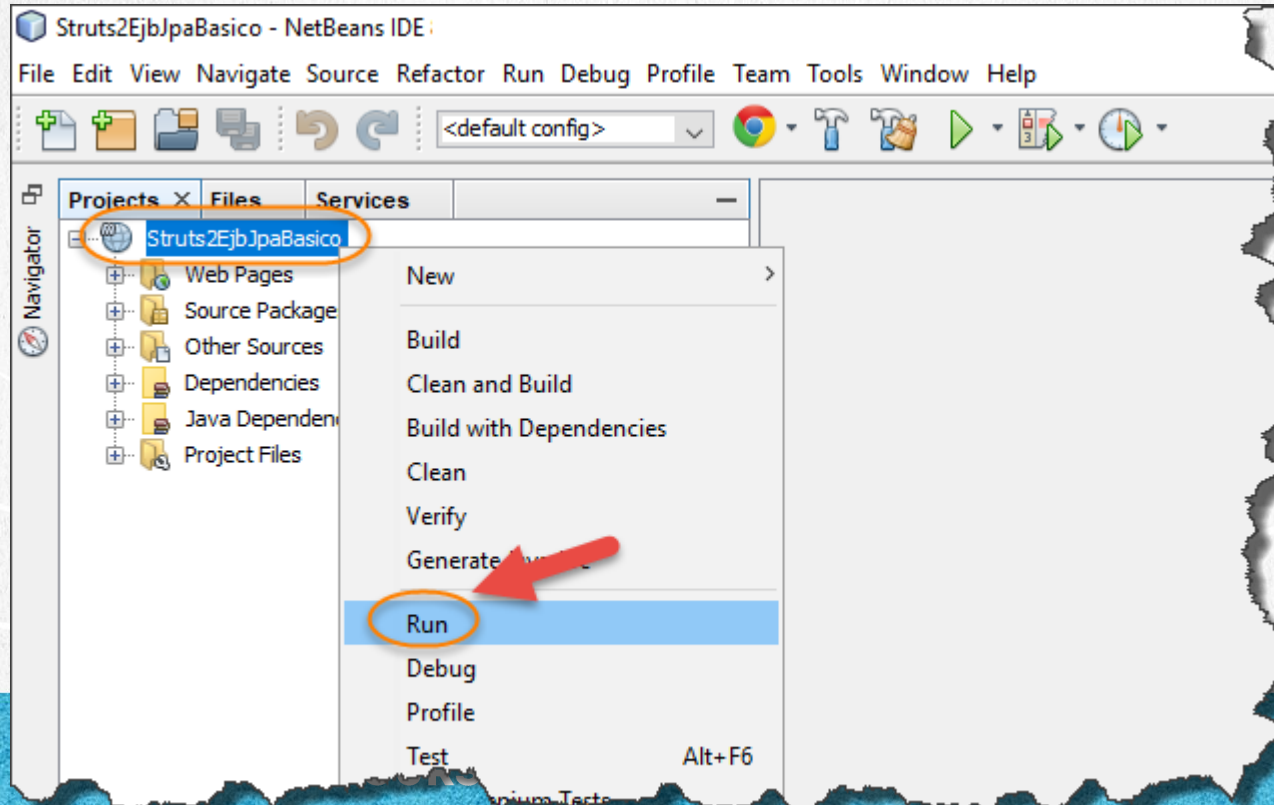
# PASO 29. EJECUTAMOS LA APLICACIÓN

- Observamos que el proceso se haya ejecutado con éxito (Build Success):



# PASO 30. EJECUTAMOS LA APLICACIÓN

- Ejecutamos la aplicación Struts2EjbJpaBasico como sigue:





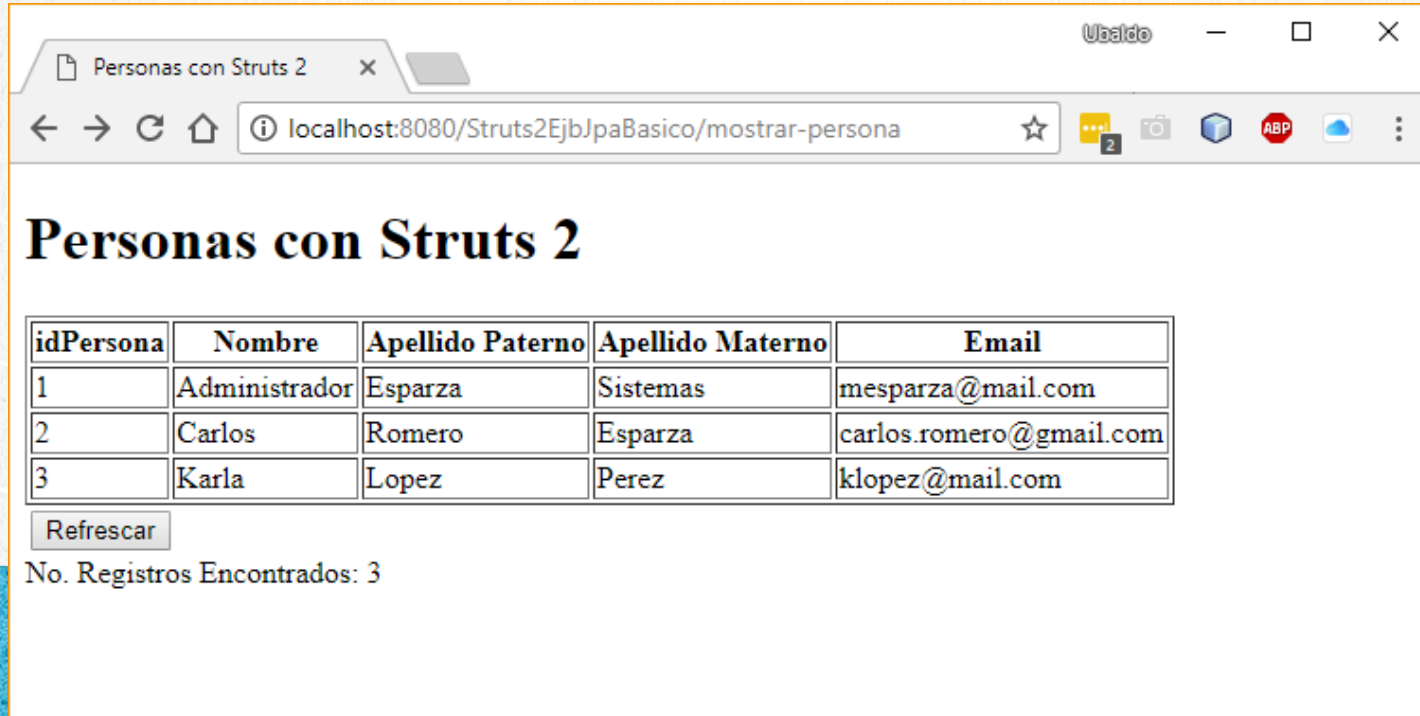
# PASO 30. EJECUTAMOS LA APLICACIÓN

- Ejecutamos la aplicación como sigue:



# PASO 30. EJECUTAMOS LA APLICACIÓN

- Deberemos observar el listado de personas como sigue. Los datos pueden variar dependiendo de la información que tengamos en la tabla de persona en la base de datos:



Personas con Struts 2

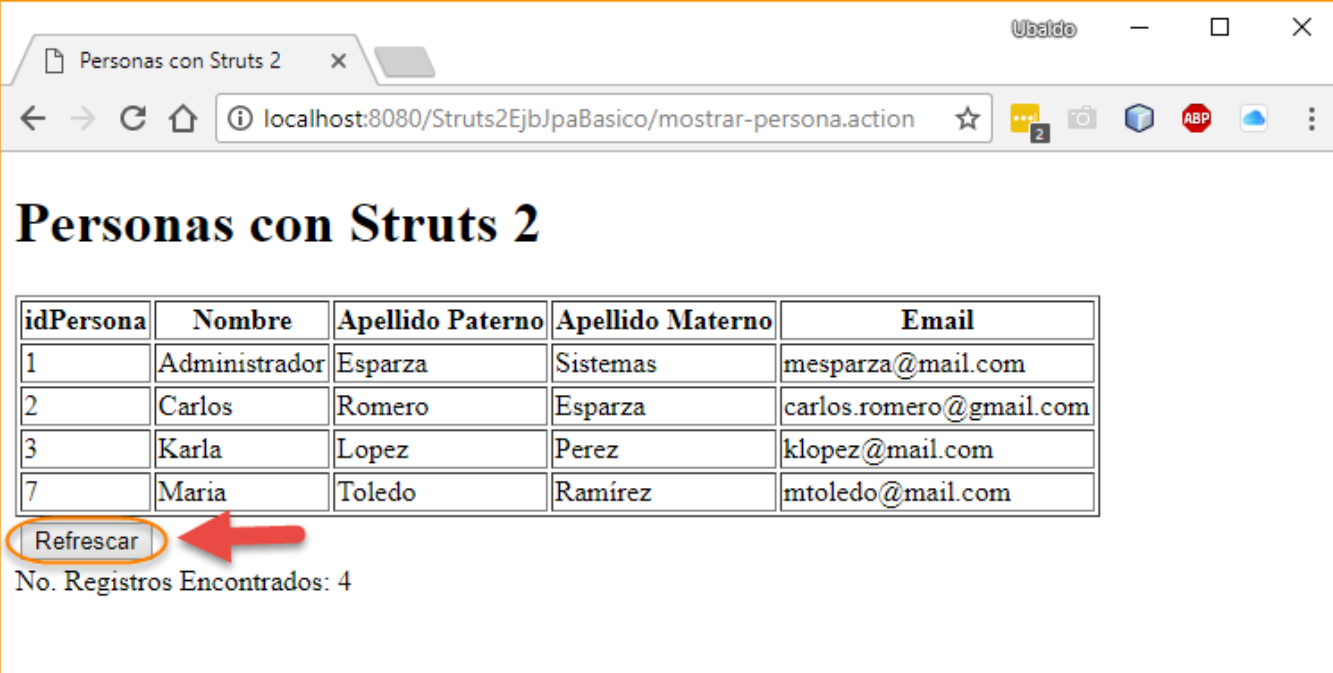
idPersona	Nombre	Apellido Paterno	Apellido Materno	Email
1	Administrador	Esparza	Sistemas	mesparza@mail.com
2	Carlos	Romero	Esparza	carlos.romero@gmail.com
3	Karla	Lopez	Perez	klopez@mail.com

Refrescar

No. Registros Encontrados: 3

# PASO 30. EJECUTAMOS LA APLICACIÓN

• Si agregamos más registros, modificamos o eliminamos registros de la tabla de persona, podemos dar click en el botón de refrescar para ver la nueva información. De momento esta información se puede agregar directamente en la base de datos o usando el software de MySql Workbench. Una vez agregada la información damos clic en el botón de Refrescar y veremos el nuevo listado con las modificaciones hechas:



**Personas con Struts 2**

idPersona	Nombre	Apellido Paterno	Apellido Materno	Email
1	Administrador	Esparza	Sistemas	mesparza@mail.com
2	Carlos	Romero	Esparza	carlos.romero@gmail.com
3	Karla	Lopez	Perez	klopez@mail.com
7	Maria	Toledo	Ramírez	mtoledo@mail.com

**Refrescar**

No. Registros Encontrados: 4



# RECOMENDACIONES FINALES

Si por alguna razón falla el ejercicio, se pueden hacer varias cosas para corregirlo:

- 1) Detener el servidor de Glassfish
- 2) Hacer un Clean & Build del proyecto para tener la versión más reciente compilada
- 3) Volver a hacer Run del proyecto (desplegar nuevamente el proyecto en el servidor)

Si lo anterior no funciona, pueden probar cargando el proyecto resuelto el cual es funcional 100% y descartar problemas de configuración en su ambiente o cualquier otro error de código.

La configuración por convenciones de Struts 2, es muy sensible, de tal manera que todo debe estar escrito tal como se especificó en el ejercicio, ya que cualquier cambio en los nombres provocará que no se ejecute correctamente el ejercicio.

La integración con otras tecnologías de Java EE como son EJB y JPA también es muy propenso a errores, así que puedes apoyarte del proyecto resuelto que te entregamos, el cual es 100% funcional, y así resolver problemas comunes en cualquier momento con esta documentación y los proyectos resueltos que te entregamos.

**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

# CONCLUSIÓN DEL EJERCICIO

Con este ejercicio hemos creado una aplicación que integra las 3 tecnologías como son:

- Struts 2
- EJB (Enterprise Java Beans de Java EE)
- JPA (Java Persistence API de Java EE)

El ejercicio tiene un listado de personas, pero es una arquitectura del mundo real, con capas lógicas, totalmente funcional y para poner en práctica la integración de las 3 tecnologías.

En los siguientes ejercicios aplicaremos las operaciones CRUD (Create-Read-Update-Delete) para la tabla de persona.



Experiencia y Conocimiento para tu vida

**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)



**CURSO ONLINE**

# **STRUTS 2 FRAMEWORK**

---

Por: Ing. Ubaldo Acosta



Experiencia y Conocimiento para tu vida



**CURSO STRUTS FRAMEWORK**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)