

# CURSO STRUTS FRAMEWORK

# INTEGRACIÓN DE FRAMEWORKS

## STRUTS + SPRING + JPA

## CRUD



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 + Spring + JPA aplicando las operaciones CRUD (Cread-Read-Update-Delete) sobre la tabla de persona. Al finalizar deberemos observar lo siguiente:

**Personas con Struts 2**

[Agregar Persona](#)

idPersona	Nombre	Apellido Paterno	Apellido Materno	Email	Editar	Eliminar
1	Administrador	Esparza	Sistemas	mesparza@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
2	Carlos	Romero	Esparza	carlos.romero@gmail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
3	Karla	Lopez	Perez	klopez@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
7	Maria	Toledo	Ramírez	mtoledo@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>

localhost:8080/StrutsSpringJpaCRUD/agregarPersona.action



# REQUERIMIENTO DEL EJERCICIO

Mostrar el listado de personas utilizando Struts + Spring + 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 StrutsSpringJpaCRUD. Este proyecto integrará las 3 tecnologías: Struts + Spring + JPA.

Empecemos con nuestro ejercicio:



Experiencia y Conocimiento para tu vida

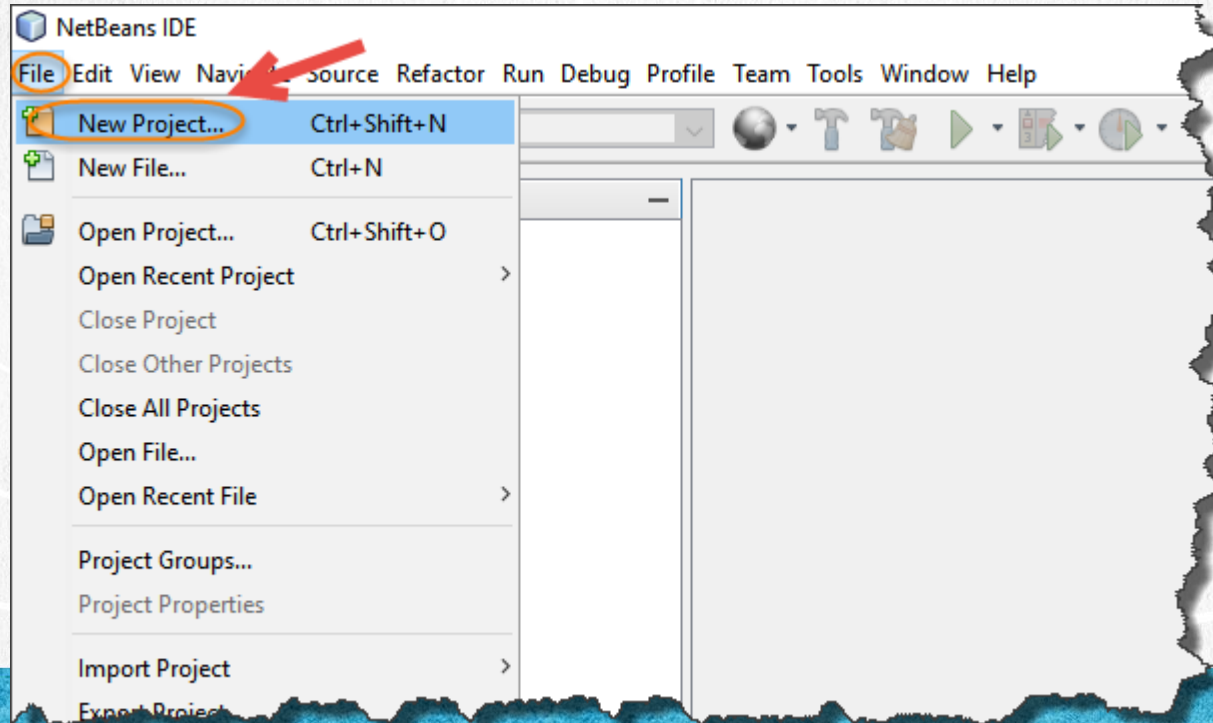
**CURSO STRUTS FRAMEWORK**

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



# 1. CREAR PROYECTO

Creamos nuestro ejercicio llamado StrutsSpringJpaCRUD:

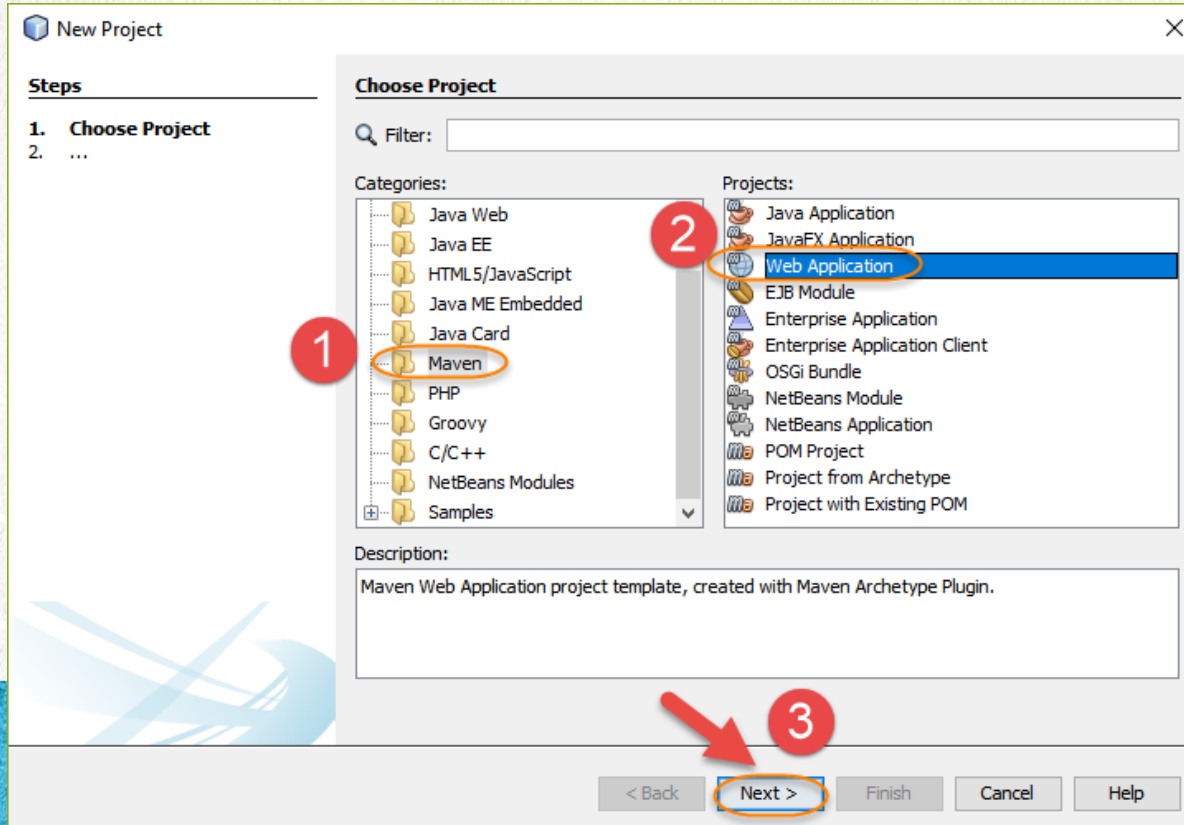


**CURSO STRUTS FRAMEWORK**

[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: StrutsSpringJpaCRUD

Project Location: C:\Cursos\Struts\03-IntegracionStrutsSpringJpaCRUD Browse...

Project Folder: s\Struts\03-IntegracionStrutsSpringJpaCRUD\StrutsSpringJpaCRUD

Artifact Id: StrutsSpringJpaCRUD

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)
  - Java EE Version: Java EE Web (highlighted with a red circle 2)
- Buttons:** < Back, Next >, **Finish** (highlighted with a red circle 3 and a red arrow), Cancel, Help

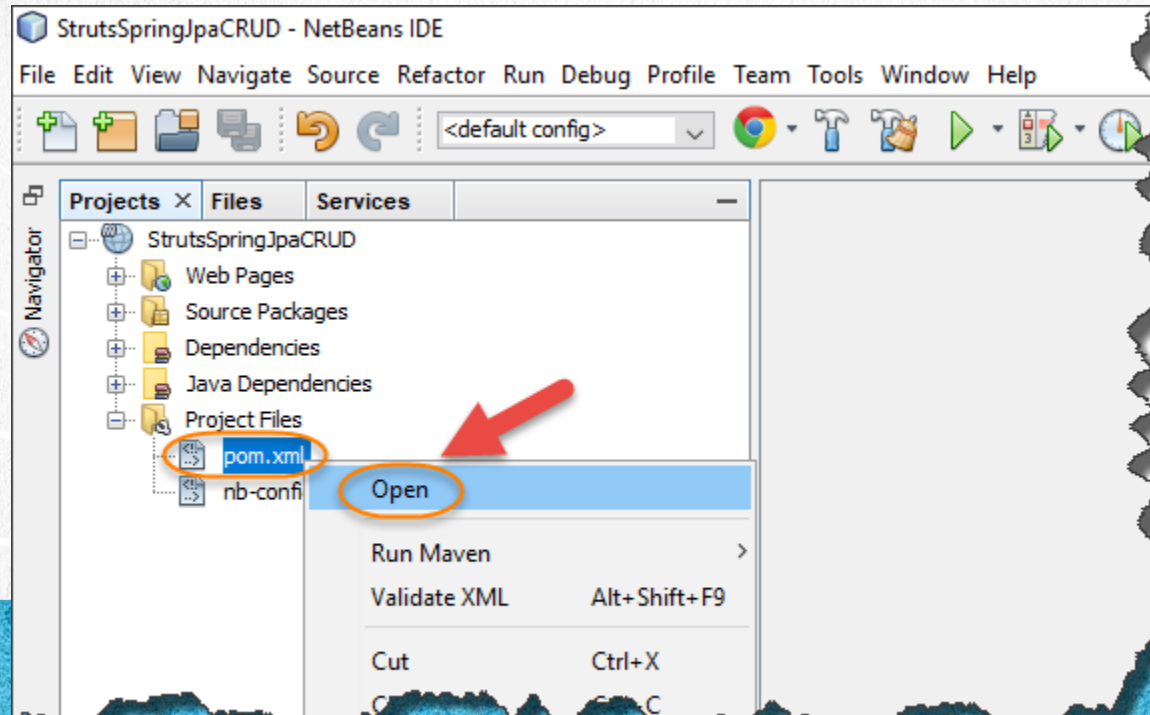
**CURSO STRUTS FRAMEWORK**

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



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

- Abrimos el archivo pom.xml de maven. Agregaremos todas las librerías necesarias para integrar las tecnologías descritas. No hay ningún cambio con el proyecto anterior básico, usamos las mismas librerías para crear este proyecto:



# 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>StrutsSpringJpaCRUD</artifactId>
  <version>1.0</version>
  <packaging>war</packaging>

  <name>StrutsSpringJpaCRUD</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <spring.version>5.0.4.RELEASE</spring.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-web-api</artifactId>
      <version>8.0</version>
      <scope>provided</scope>
    </dependency>
```



# PASO 3. MODIFICAMOS EL CÓDIGO

Archivo pom.xml:

Clic para ver el  
archivo

```
<dependency>
  <groupId>org.apache.struts</groupId>
  <artifactId>struts2-core</artifactId>
  <version>2.5.14.1</version>
</dependency>
<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>
```

# PASO 3. MODIFICAMOS EL CÓDIGO

## Archivo pom.xml:

Clic para ver el  
archivo

```
<!--Spring-->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-orm</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
  <version>${spring.version}</version>
</dependency>
```



# PASO 3. MODIFICAMOS EL CÓDIGO

Archivo pom.xml:

Clic para ver el  
archivo

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>${spring.version}</version>
</dependency>
<!-- Struts 2 y Spring integracion -->
<dependency>
  <groupId>org.apache.struts</groupId>
  <artifactId>struts2-spring-plugin</artifactId>
  <version>2.5.14.1</version>
</dependency>
<!-- MySql -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.42</version>
</dependency>
</dependencies>
```

**CURSO DE JAVA CON JDBC**

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

# PASO 3. MODIFICAMOS EL CÓDIGO

## Archivo pom.xml:

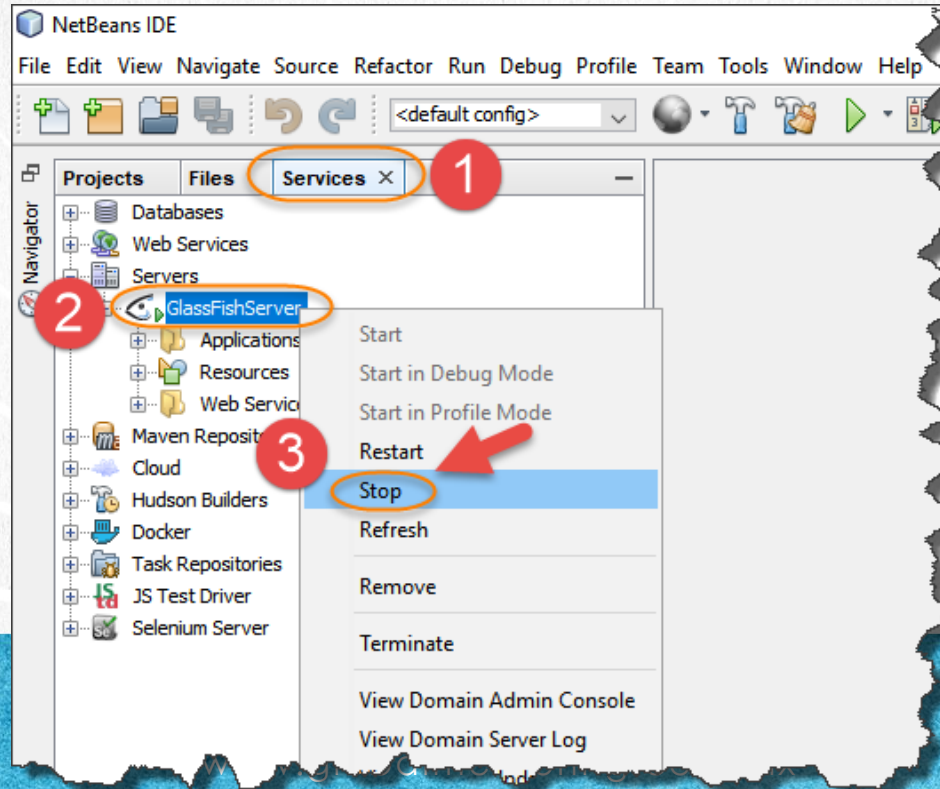
Clic para ver el  
archivo

```
<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>
```



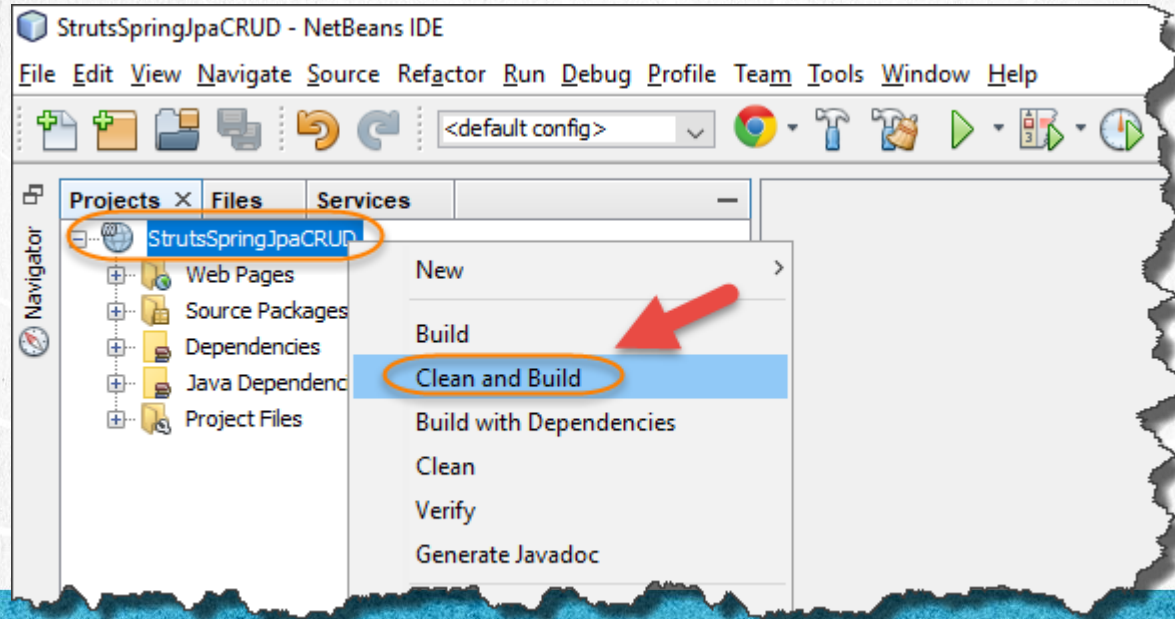
## 4. DETENEMOS GLASSFISH SI ESTUVIERA INICIADO

- 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:



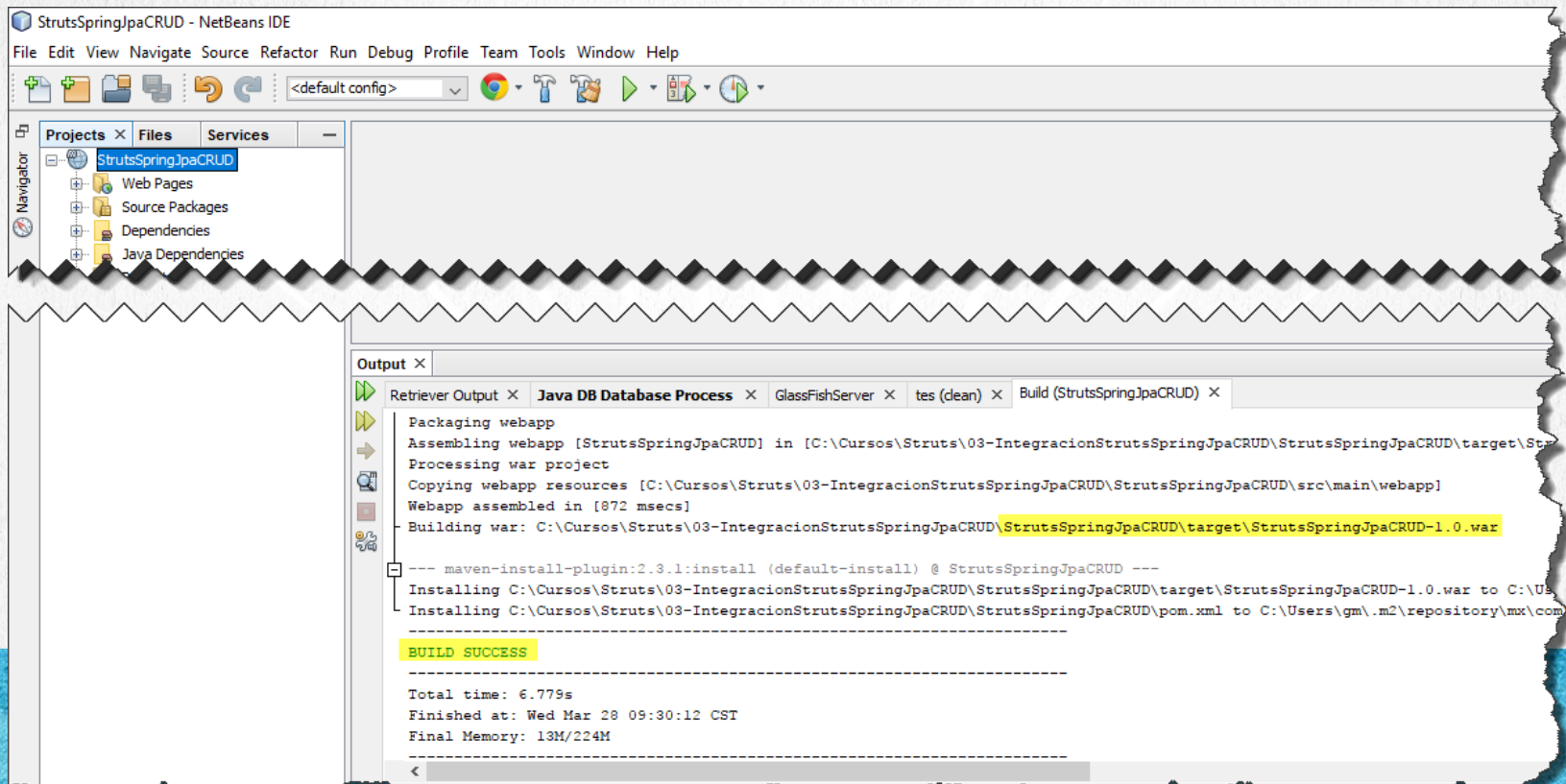
**CURSO STRUTS FRAMEWORK**

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



# 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).

No hay ningún cambio en este archivo comparado con el ejercicio anterior básico.

Veamos como queda nuestro archivo persistence.xml.



Experiencia y Conocimiento para tu vida

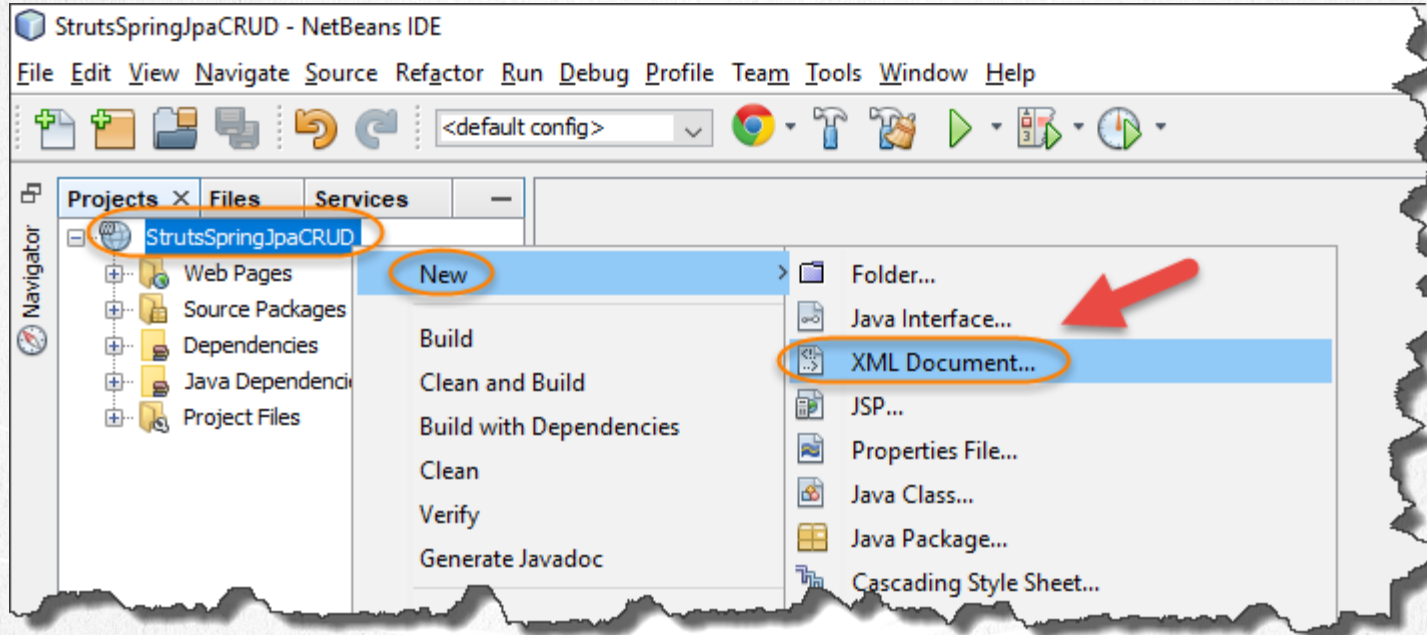
**CURSO STRUTS FRAMEWORK**

[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:





## 6. CREAMOS UN ARCHIVO XML

- El nombre del archivo, 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:

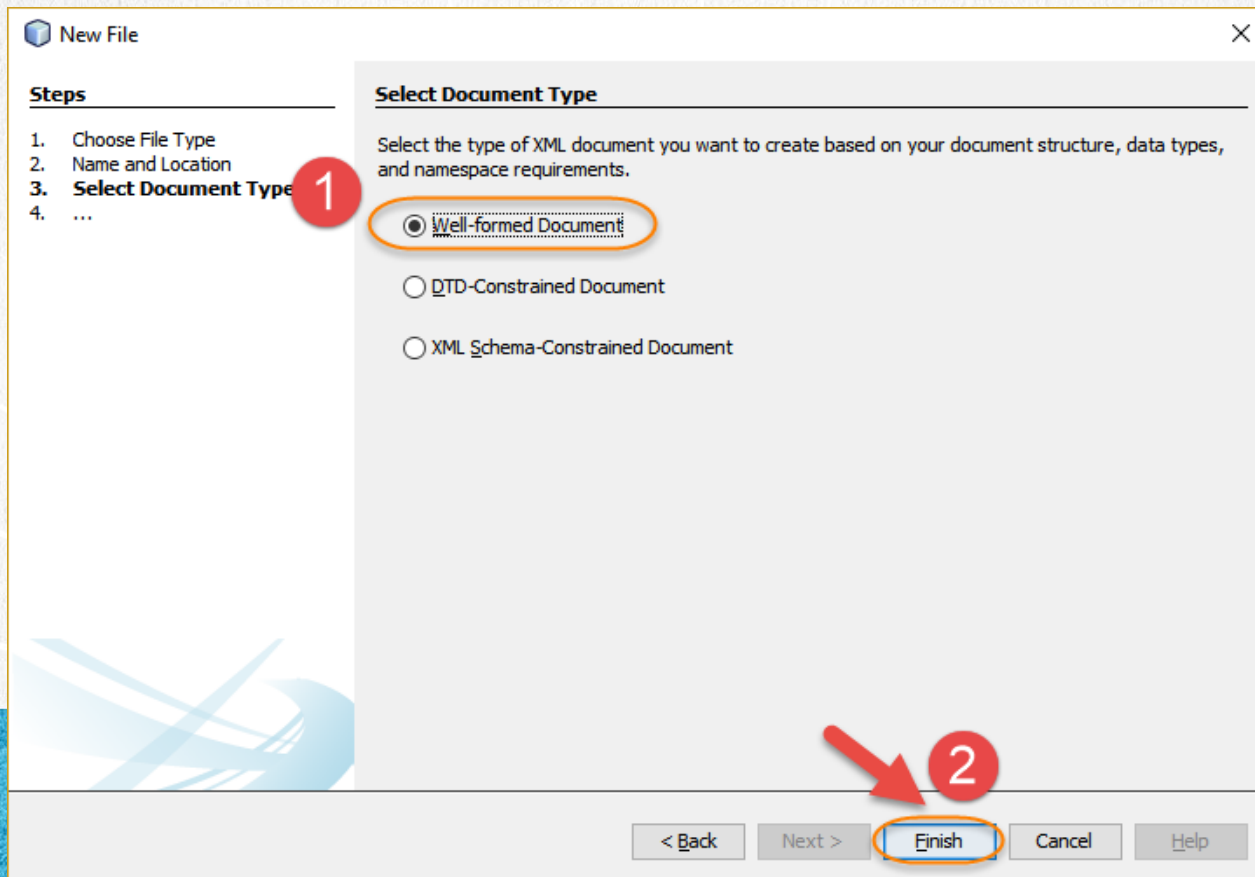
Project:

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 applicationContext.xml

Este archivo es el que nos permite configurar el framework de Spring.

Este archivo no tiene cambios respecto al ejercicio anterior básico.

Veamos como queda nuestro archivo applicationContext.xml



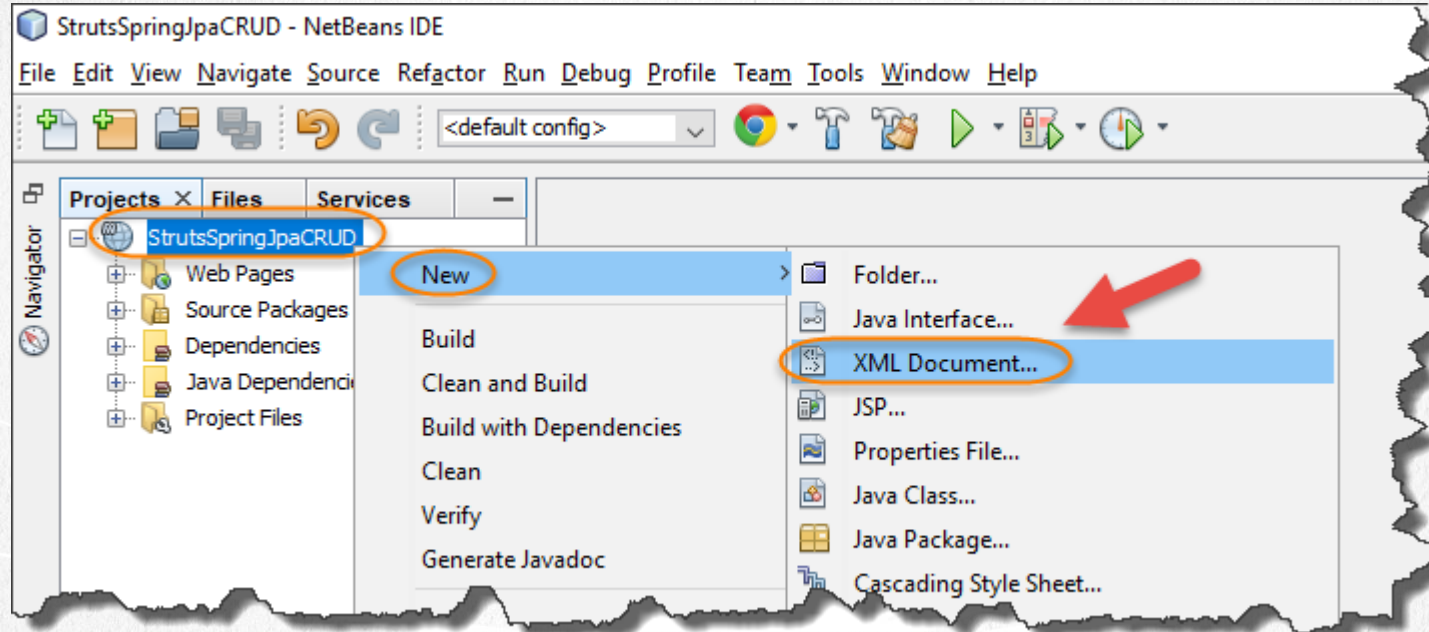
Experiencia y Conocimiento para tu vida

**CURSO STRUTS FRAMEWORK**

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

## 8. CREAMOS UN ARCHIVO XML

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





## 8. CREAMOS UN ARCHIVO XML

- El nombre del archivo, 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:

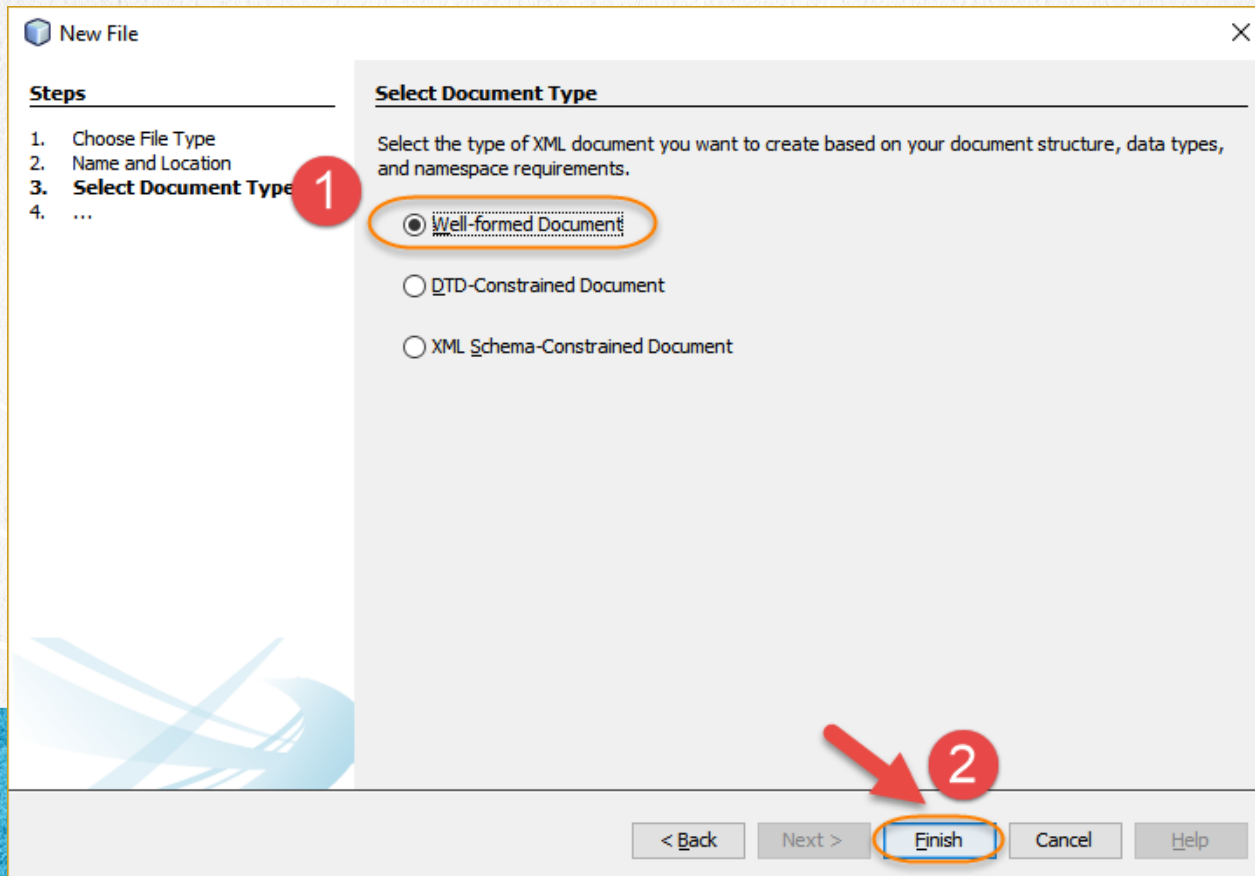
Folder:

Created File:



## 8. CREAMOS UN ARCHIVO XML

- Seleccionamos el tipo indicado y damos click en finalizar.



# PASO 9. MODIFICAMOS EL CÓDIGO

## Archivo applicationContext.xml:

Clic para ver el  
archivo

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:p="http://www.springframework.org/schema/p"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:jee="http://www.springframework.org/schema/jee"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/jee
    http://www.springframework.org/schema/jee/spring-jee.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd">

  <context:component-scan base-package="mx.com.gm.capaservicio" />
  <context:component-scan base-package="mx.com.gm.capadatos" />
```

# PASO 9. MODIFICAMOS EL CÓDIGO

Archivo applicationContext.xml:

Clic para ver el  
archivo

```
<!-- Obtiene el entity manager inyectado en la fabrica de Spring -->
<bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor" />

<bean id="transactionManager" class="org.springframework.transaction.jta.JtaTransactionManager" />

<!--Nombre que mapea con la Unidad de Persistencia en el archivo web.xml-->
<jee:jndi-lookup id="entityManagerFactory" jndi-name="persistence/PersistenceUnit" />

<!-- Detecta @Transactional -->
<tx:annotation-driven transaction-manager="transactionManager" />
</beans>
```



# 10. 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.

Además, también nos permite integrar el framework de Spring con nuestra aplicación web por medio de la configuración de un listener de Spring.

Nuestro archivo web.xml también nos permite configurar el nombre JNDI para la conexión a base de datos que utilizaremos con JPA vía JTA.

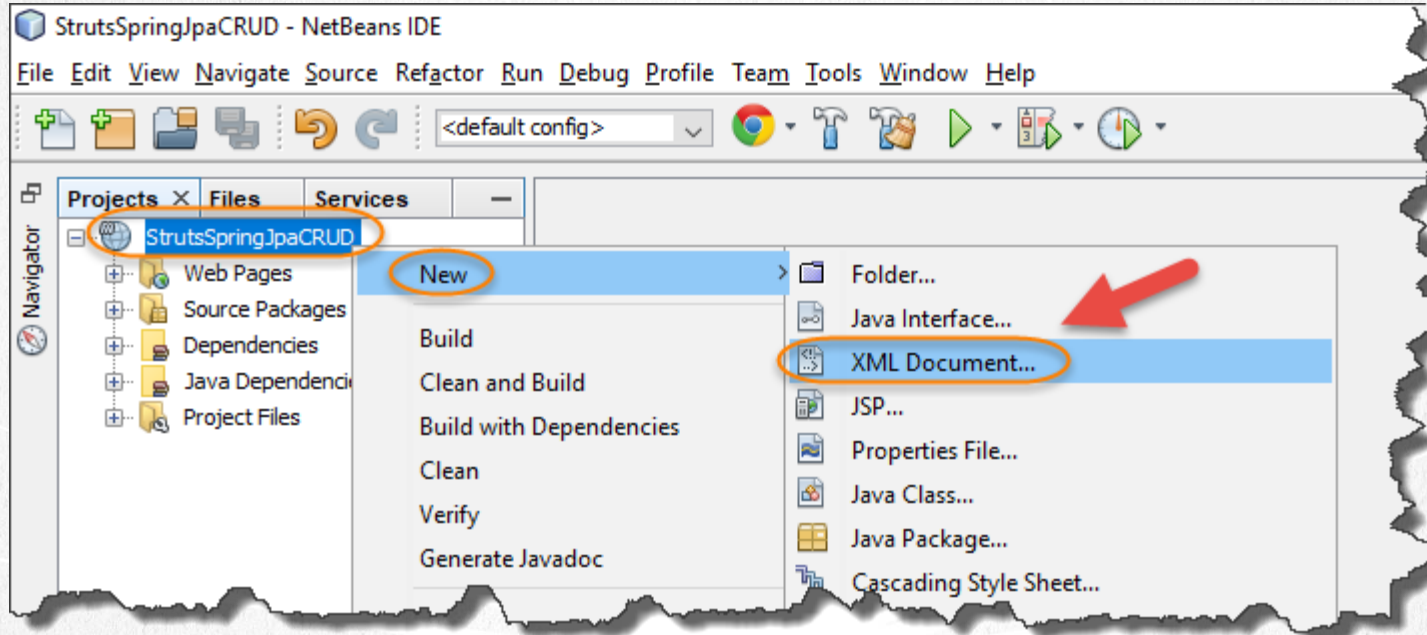
Normalmente deberíamos usar la última versión del namespace de JavaEE, pero por problemas con compatibilidad con Spring, usaremos la versión 3.1 del namespace.

Este archivo no tiene ninguna modificación comparado con el ejercicio básico anterior.

Veamos como queda nuestro archivo web.xml.

# 10. CREAMOS UN ARCHIVO XML

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





# 10. 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:

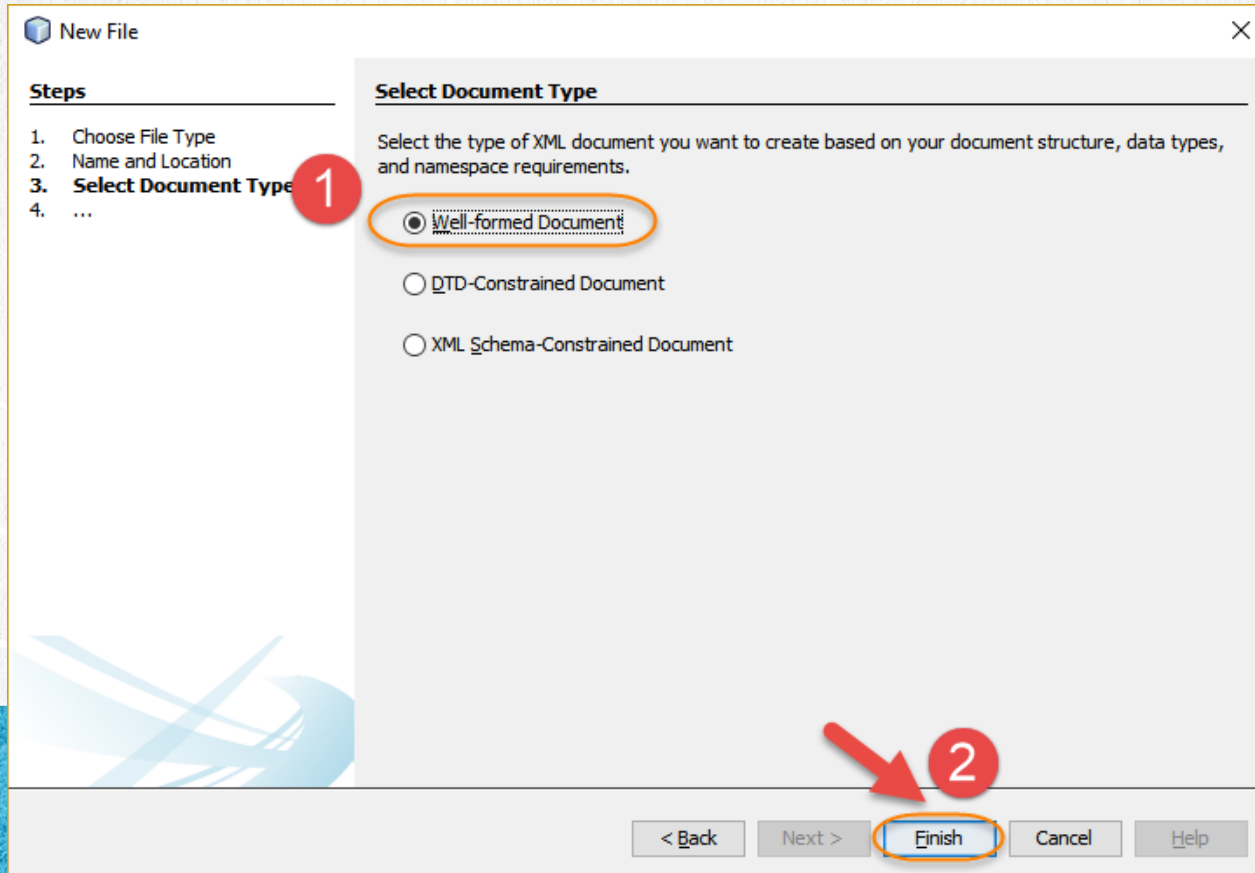
Folder:

Created File:



# 10. CREAMOS UN ARCHIVO XML

- Seleccionamos el tipo indicado y damos click en finalizar.



# PASO 11. MODIFICAMOS EL CÓDIGO

## Archivo web.xml:

Clic para ver el  
archivo

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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_3_1.xsd"
version="3.1">
  <!-- 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>

  <!-- integracion con Spring Framework-->
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

  <!-- nombre utilizado en el archivo applicationContext.xml de Spring y JPA-->
  <persistence-unit-ref>
    <persistence-unit-ref-name>persistence/PersistenceUnit</persistence-unit-ref-name>
    <persistence-unit-name>PersistenceUnit</persistence-unit-name>
  </persistence-unit-ref>
</web-app>
```

## 12. 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.

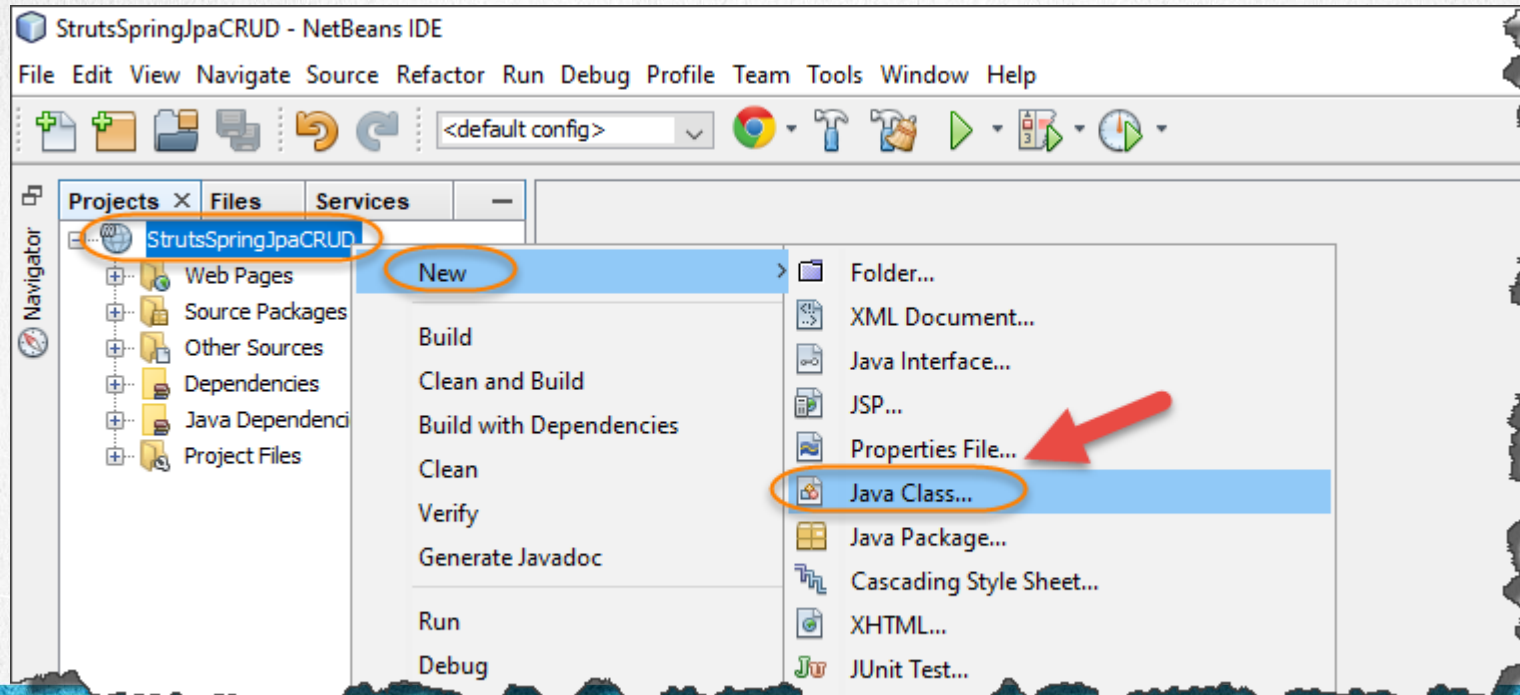
Esta clase no tiene ninguna diferencia con el ejercicio anterior básico.

Veamos como queda nuestra clase Persona.java



# 12. CREAR UNA CLASE JAVA

- Creamos la clase Persona.java:



**CURSO STRUTS FRAMEWORK**

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

# 12. 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:

< Back   Next >   **Finish**   Cancel   Help

# PASO 13. 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 13. 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 13. 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 13. 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 13. 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;
}
}
```

# 14. 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.

Esta interface no tiene ninguna diferencia con el ejercicio básico anterior.

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



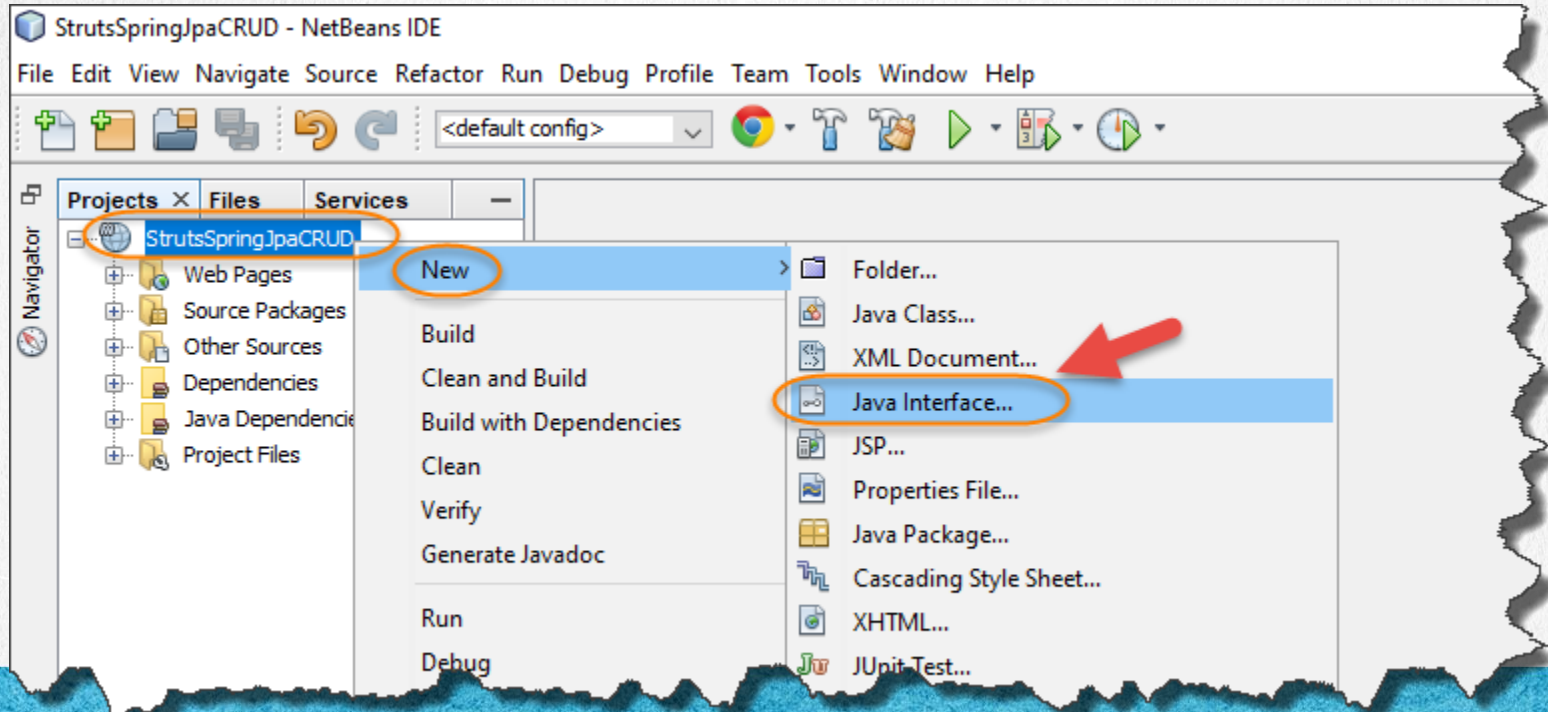
Experiencia y Conocimiento para tu vida

**CURSO STRUTS FRAMEWORK**

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

# 14. CREAR UNA INTERFACE JAVA

- Creamos la interface PersonaDao.java:



**CURSO STRUTS FRAMEWORK**

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



# 14. 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 15. 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);

}
```

# 16. 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 Spring y 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.

Esta es la misma clase que utilizamos en el ejercicio básico anterior.

Veamos como queda la clase PersonaDaoImpl.java



Experiencia y Conocimiento para tu vida

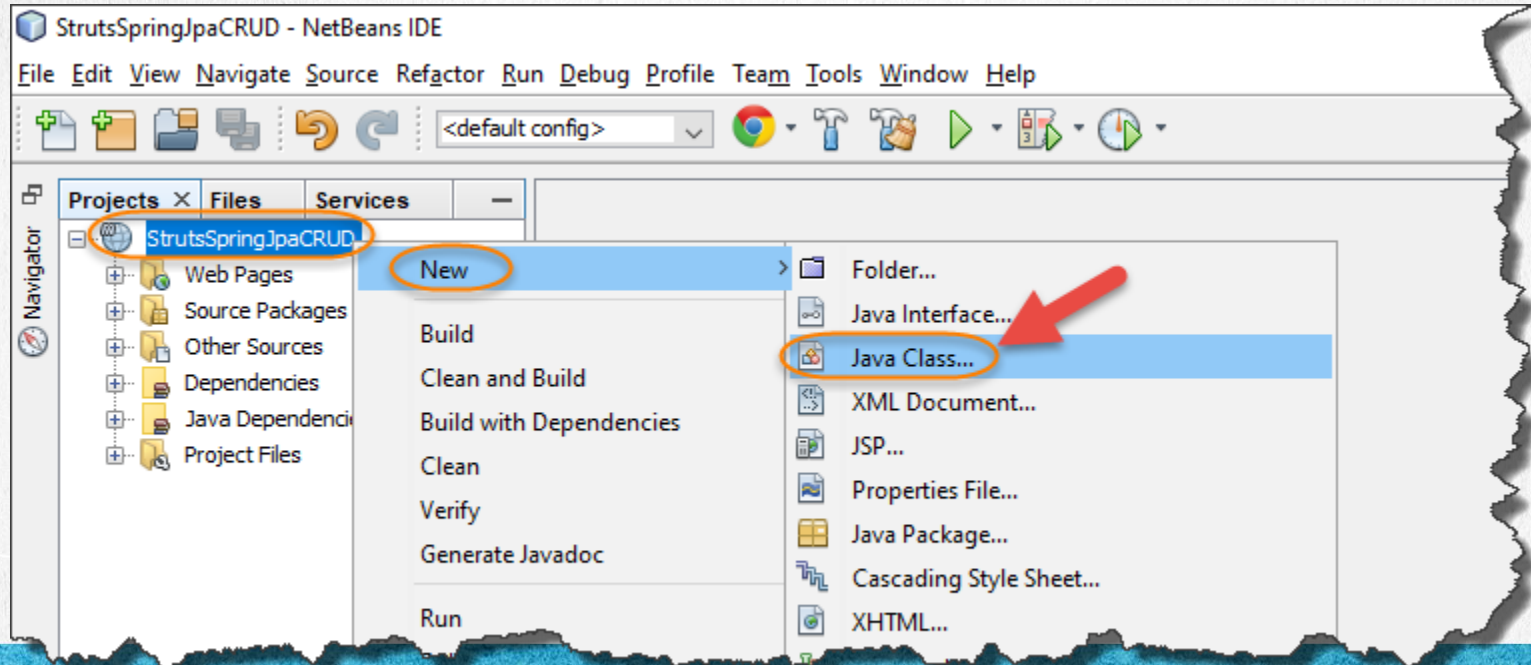
**CURSO STRUTS FRAMEWORK**

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



# 16. CREAR UNA CLASE JAVA

- Creamos la clase PersonaDaoImpl.java:



**CURSO STRUTS FRAMEWORK**

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

# 16. 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:

< Back   Next >   **Finish**   Cancel   Help

# PASO 17. MODIFICAMOS EL CÓDIGO

## Archivo PersonaDaoImpl.java:

Clic para ver el archivo

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

import java.util.List;
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;
import org.springframework.stereotype.Repository;

@Repository
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 17. 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 17. MODIFICAMOS EL CÓDIGO

## Archivo PersonaDaoImpl.java:

Clic para ver el archivo

```
@Override
public long contadorPersonas() {
    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();
}
}
```

# 18. 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.

Esta es la misma interface que utilizamos en el ejercicio básico anterior.

Veamos como queda esta interface `PersonaService.java`:



Experiencia y Conocimiento para tu vida

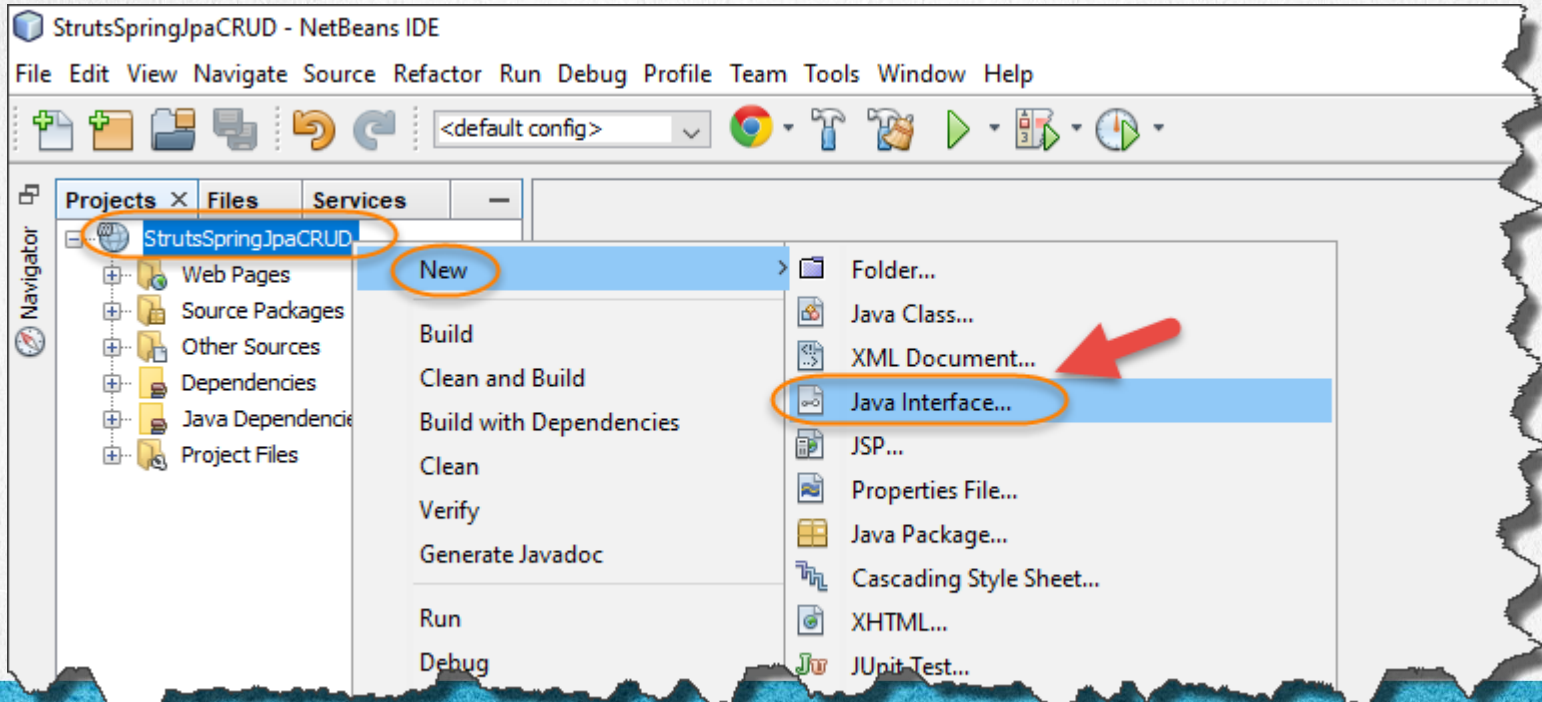
**CURSO STRUTS FRAMEWORK**

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



# 18. CREAR UNA INTERFACE JAVA

- Creamos la interface PersonaService.java:



**CURSO STRUTS FRAMEWORK**

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

# 18. 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:

# PASO 19. 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();

}
```



## 20. 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 Spring para manejar de manera automática el concepto de transacciones.

Esta es la misma clase que utilizamos en el ejercicio básico anterior.

Veamos como queda la clase PersonaServiceImpl.java



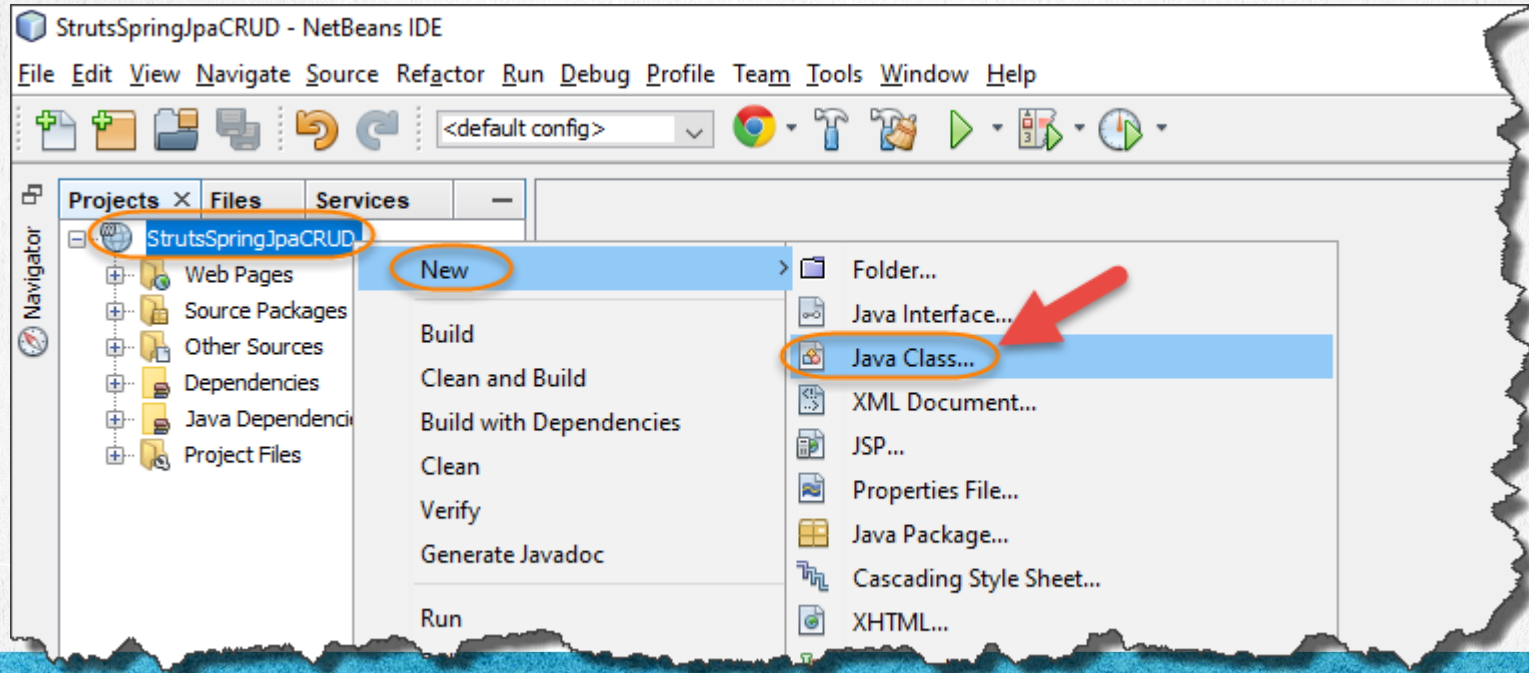
Experiencia y Conocimiento para tu vida

**CURSO STRUTS FRAMEWORK**

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

## 20. CREAR UNA CLASE JAVA

- Creamos la clase PersonaServiceImpl.java:



**CURSO STRUTS FRAMEWORK**

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



## 20. CREAR UNA CLASE JAVA

- Creamos la clase PersonaServiceImpl.java:

**New Java Class**

**Steps**

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

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File:



# PASO 21. MODIFICAMOS EL CÓDIGO

## Archivo PersonaServiceImpl.java:

Clic para ver el archivo

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

import java.util.List;
import mx.com.gm.capadatos.PersonaDao;
import mx.com.gm.capadatos.domain.Persona;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service("personaService")
@Transactional
public class PersonaServiceImpl implements PersonaService {

    @Autowired
    private PersonaDao personaDao;

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

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

# PASO 21. 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 22. CREAR UNA CLASE JAVA

La clase `PersonasAction.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 Spring y el plug-in de integración entre Struts y Spring que agregamos al archivo `pom.xml`

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`.

Esta clase sí tiene cambios respecto al ejercicio básico, ya que esta clase `Action` es precisamente la que soportará todos los cambios en la vista que queremos aplicar. Desde cambios en los nombres de nuestras clases, hasta nuevas funciones que utilizaremos para poder aplicar las operaciones CRUD (Create-Read-Update-Delete), es decir las operaciones de: listar, agregar, modificar y eliminar registros.



# PASO 22. CREAR UNA CLASE JAVA

En la clase `PersonasAction.java` hemos agregado varios métodos de tipo `execute` y no solo uno. De tal manera que por cada acción que ejecutemos desde el navegador, esta clase de tipo `Action` será la encargada de procesar dichas acciones.

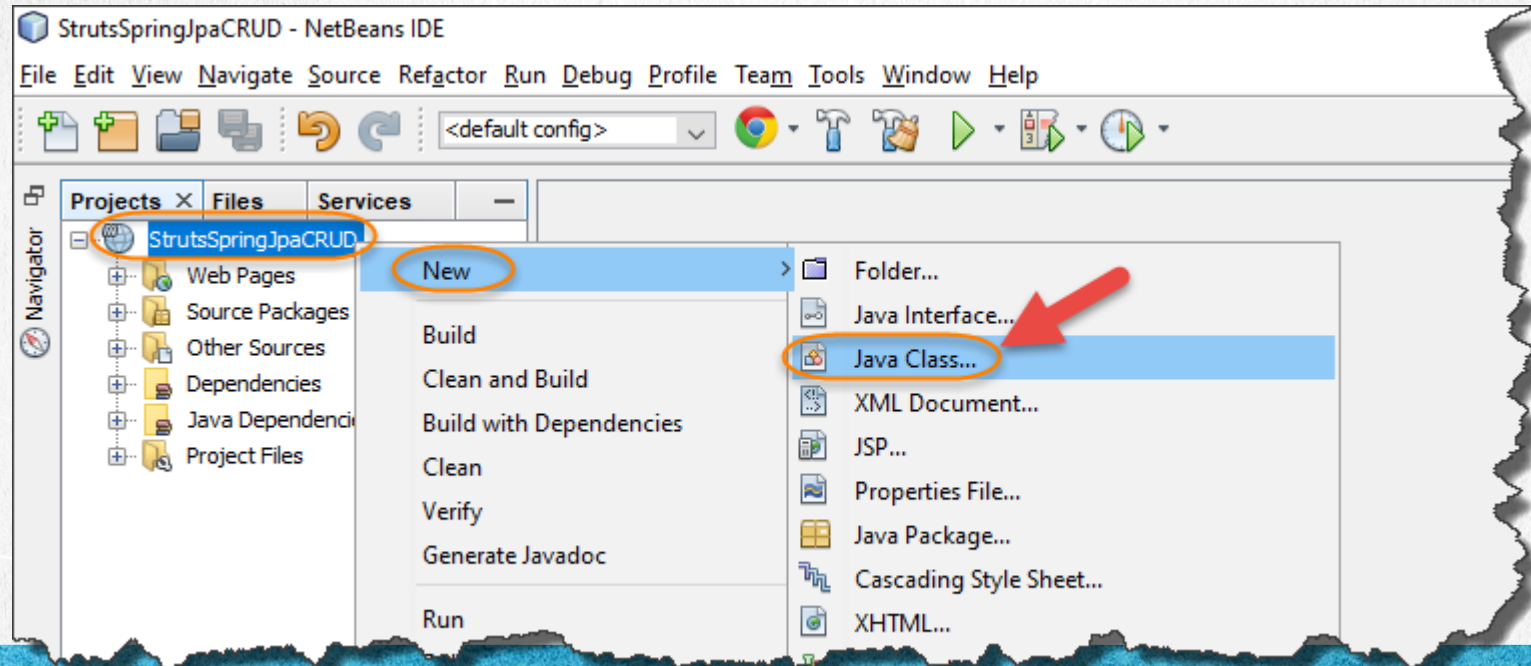
De esta forma estamos demostrando que una clase de tipo `Action` puede tener múltiples métodos de tipo `execute`, pero que pueden tener cualquier nombre, y con ayuda de la anotación `@Action` es que definimos el nombre del path que procesará el método indicado con esta anotación. Por lo que no es necesario que el nombre del path y el nombre del método coincidan.

Otra característica que vemos en estos métodos es que en algunos casos fue necesario especificar la vista que se ejecuta después de haber terminado de ejecutar dicho método. Para ello hicimos de la anotación `@Result` y se especifican algunos atributos dependiendo del tipo de vista seleccionada, como puede ser el nombre del resultado, la ubicación de la vista y en algunos casos el tipo de resultado como es `redirect`, para que vuelva a ejecutar un path el método `Action` y no solamente se ejecute el JSP, ya que en los casos de agregar, modificar o eliminar un registro del listado, necesitamos volver a desplegar el nuevo listado actualizado, y para ello no basta con llamar directamente al JSP de listado, sino que debemos de pasar por el método `listar` de la clase `PersonasAction` y así se refresque el listado de personas.

Veamos cómo queda nuestra clase `PersonasAction`.

# PASO 22. CREAR UNA CLASE JAVA

- Creamos la clase PersonasAction.java:



**CURSO STRUTS FRAMEWORK**

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



# PASO 22. CREAR UNA CLASE JAVA

- Creamos la clase PersonasAction.java:

**New Java Class**

**Steps**

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

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File:



# PASO 23. MODIFICAMOS EL CÓDIGO

## Archivo PersonasAction.java:

Clic para ver el archivo

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

import com.opensymphony.xwork2.ActionSupport;
import java.util.List;
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;
import org.apache.struts2.convention.annotation.Action;
import org.apache.struts2.convention.annotation.Result;
import org.springframework.beans.factory.annotation.Autowired;

public class PersonasAction extends ActionSupport {

    private List<Persona> personas;

    private Persona persona;

    @Autowired
    private PersonaService personaService;

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

    @Action(value = "/listar", results = {
        @Result(name = "personas", location = "/WEB-INF/content/personas.jsp")})
    public String listar() {
        this.personas = personaService.listarPersonas();
        return "personas";
    }
}
```

# PASO 23. MODIFICAMOS EL CÓDIGO

## Archivo PersonasAction.java:

Clic para ver el archivo

```
@Action(value = "/agregarPersona", results = {
    @Result(name = "persona", location = "/WEB-INF/content/persona.jsp")})
public String agregar() {
    //Creamos un nuevo objeto de tipo persona
    persona = new Persona();
    return "persona";
}

@Action(value = "/editarPersona", results = {
    @Result(name = "persona", location = "/WEB-INF/content/persona.jsp")})
public String editar() {
    persona = personaService.recuperarPersona(persona);
    return "persona";
}

@Action(value = "/eliminarPersona", results = {
    @Result(name = "success", location = "listar", type = "redirect")})
public String eliminar() {
    //Recuperamos el objeto persona, ya que solo tenemos el idPersona
    log.info("Metodo eliminar persona antes de recuperar:" + persona);
    persona = personaService.recuperarPersona(persona); //revisar si se puede quitar el == y sigue funcionando, por referencia
    log.info("Metodo eliminar persona despues de recuperar:" + persona);
    personaService.eliminarPersona(persona);
    return SUCCESS;
}
```

# PASO 23. MODIFICAMOS EL CÓDIGO

## Archivo PersonasAction.java:

Clic para ver el archivo

```
//No basta con mandar al JSP, sino a la accion de listar
//por ello redireccionamos a la accion listar
@Action(value = "/guardarPersona", results = {
    @Result(name = "success", location = "listar", type = "redirect")})
public String guardar() {
    //Diferenciamos la accion de agregar o editar con el idPersona
    if (persona.getIdPersona() == null) {
        personaService.agregarPersona(persona);
    } else {
        personaService.modificarPersona(persona);
    }
    return SUCCESS;
}

public Persona getPersona() {
    return persona;
}

public void setPersona(Persona persona) {
    this.persona = persona;
}

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

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



# PASO 24. CREAR EL ARCHIVO DE PROPIEDADES

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

Veamos como queda este archivo `PersonasAction.properties`

Este archivo también se modificó conforme al ejercicio básico anterior, para soportar el nuevo listado y las operaciones de agregar y modificar.



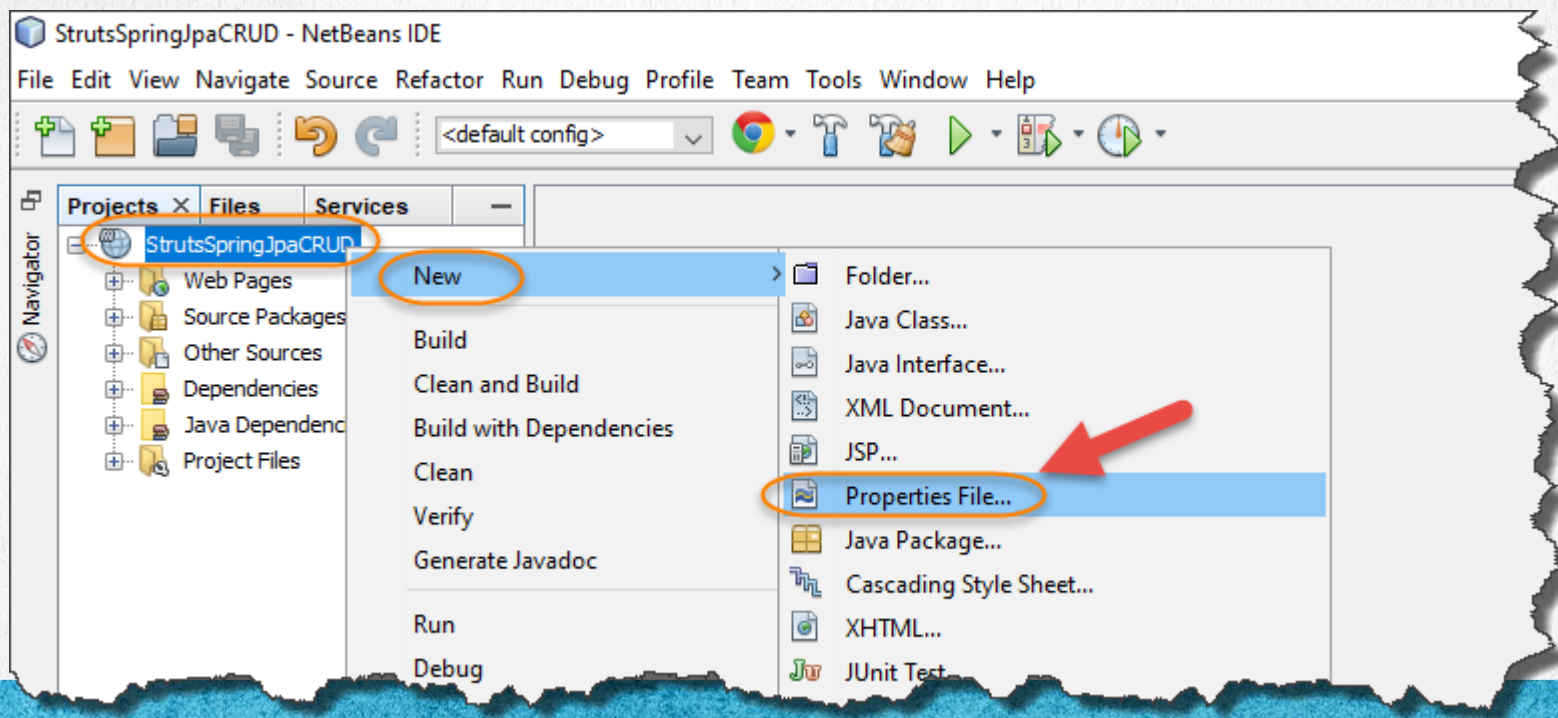
Experiencia y Conocimiento para tu vida

**CURSO STRUTS FRAMEWORK**

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

# PASO 24. CREAR EL ARCHIVO DE PROPIEDADES

- Creamos el archivo PersonasAction.properties como sigue:



**CURSO STRUTS FRAMEWORK**

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



# PASO 24. 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 25. MODIFICAMOS EL CÓDIGO

## Archivo PersonasAction.properties:

Clic para ver el archivo

```
pform.titulo: Personas con Struts 2
pform.contador: No. Registros Encontrados
pform.enviar: Enviar
pform.detalle: Detalle Persona
pform.agregar: Agregar Persona
pform.listado: Regresar listado de Personas
pform.editar: Editar
pform.eliminar: Eliminar
p.idPersona: idPersona
p.nombre: Nombre
p.apePat: Apellido Paterno
p.apeMat: Apellido Materno
p.email: Email
```

# PASO 26. 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: [listar](#)



Experiencia y Conocimiento para tu vida

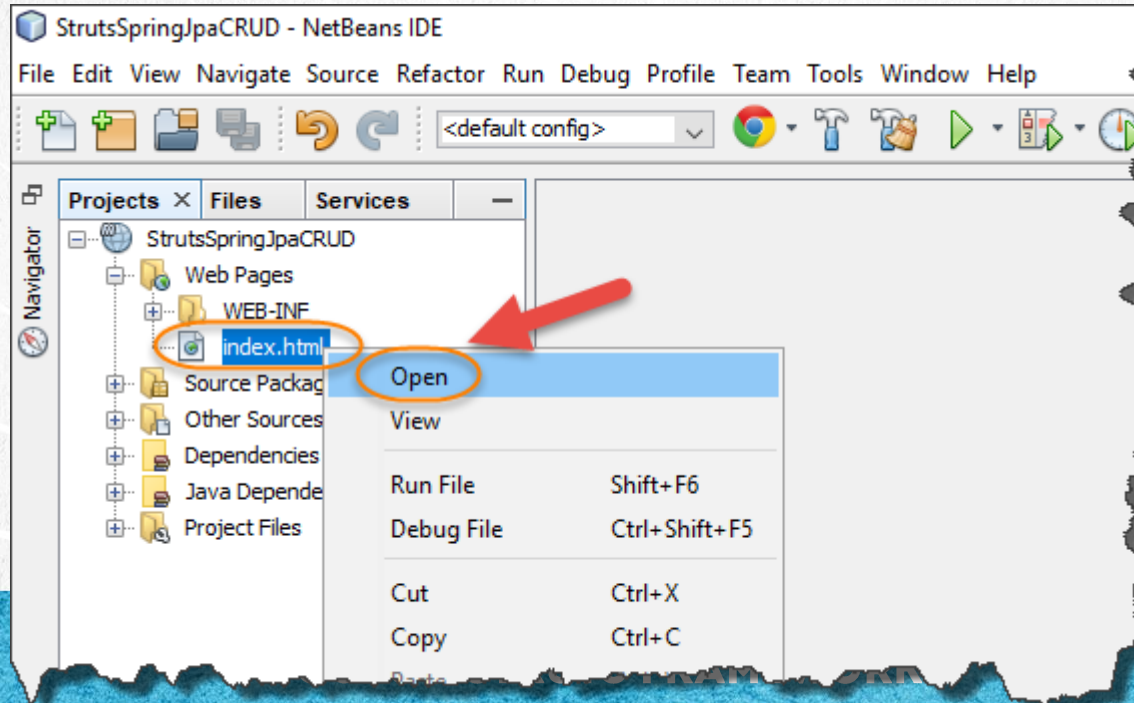
**CURSO STRUTS FRAMEWORK**

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



# PASO 26. 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 lo creamos:





# PASO 26. 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="listar">Ir a Personas</a>
  </body>
</html>
```

## PASO 27. CREAR EL ARCHIVO JSP

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

Este es el primer JSP que vamos a crear, el cual nos va a servir para mostrar el listado de objetos de tipo persona, y tendrá los links para agregar, modificar y eliminar un registro de tipo persona.

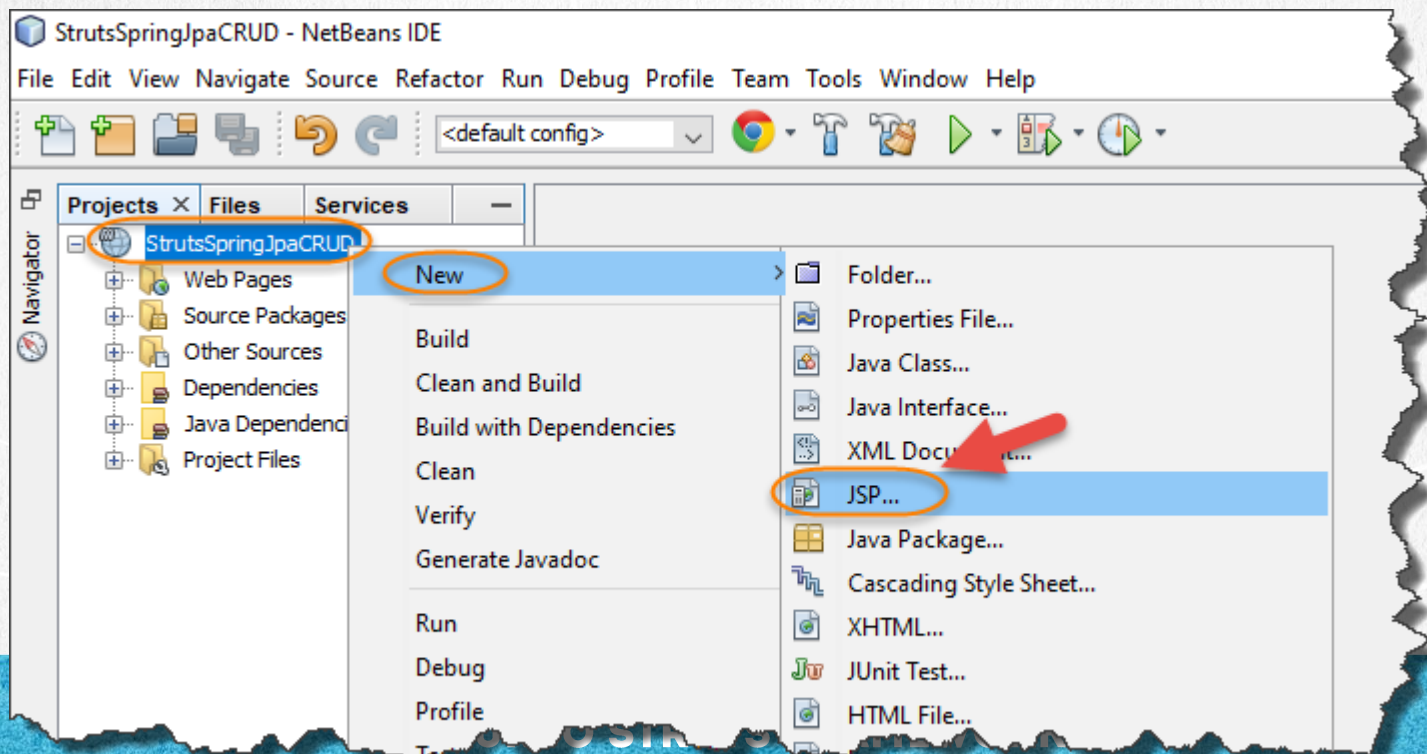
Además debemos depositar este JSP en la carpeta `/WEB-INF/content` según hemos visto en el tema de convenciones de Struts 2.

Este archivo también cambió conforme al ejercicio anterior básico, ya que hemos agregado el link de Agregar, así como las columnas de Editar y Eliminar en cada uno de los registros del listado. Veamos cómo quedó nuestro archivo `personas.jsp`



# PASO 27. CREAR EL ARCHIVO JSP

- Creamos el archivo personas.jsp:





# PASO 27. CREAR EL ARCHIVO JSP

- Creamos el archivo personas.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 28. MODIFICAMOS EL CÓDIGO

## Archivo personas.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="pform.titulo" /></title>
  </head>
  <body>
    <h1><s:text name="pform.titulo" /></h1>
    <a href="<s:url action="agregarPersona"/>"><s:text name="pform.agregar" /></a>
    <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>
            <th><s:text name="pform.editar" /></th>
            <th><s:text name="pform.eliminar" /></th>
          </tr>
```

# PASO 28. MODIFICAMOS EL CÓDIGO

## Archivo personas.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>
    <td>
      <s:url action="editarPersona" var="editarURL">
        <s:param name="persona.idPersona" value="%{idPersona}"></s:param>
      </s:url>
      <s:a href="%{editarURL}"><s:text name="pform.editar" /></s:a>
    </td>
    <td>
      <s:url action="eliminarPersona" var="eliminarURL">
        <s:param name="persona.idPersona" value="%{idPersona}"></s:param>
      </s:url>
      <s:a href="%{eliminarURL}"><s:text name="pform.eliminar" /></s:a>
    </td>
  </tr>
</s:iterator>
</table>
</div>
</s:if>
</body>
</html>
```



## PASO 29. CREAR EL ARCHIVO JSP

Ahora creamos el archivo: `persona.jsp`. En este caso este es un nuevo archivo JSP, el cual no hace un mapeo directo con una acción, ya que utilizaremos la misma clase `PersonasAction` para procesar los valores que utilizará la vista `persona.jsp`.

Este es el segundo JSP que crearemos, y lo utilizaremos básicamente para mostrar el detalle de un objeto de tipo `persona`. Y lo usaremos tanto para agregar o modificar un objeto de tipo `persona`.

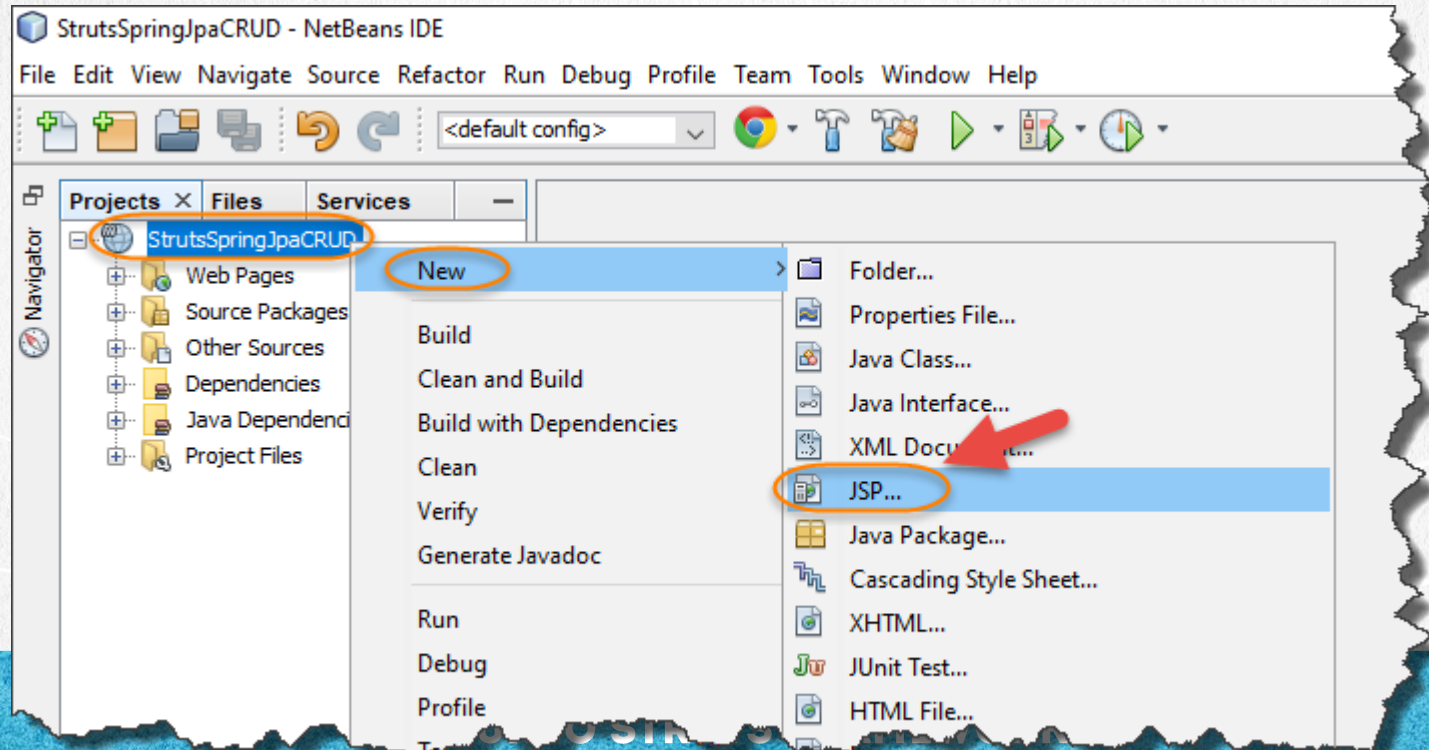
La única diferencia entre agregar y modificar una persona es el valores de `idPersona`, si es igual a nulo quiere decir que aún no tiene una representación en la base de datos y por lo tanto se hace un insert, es decir, se agrega a la base de datos. Por otro lado, si el valor es distinto a nulo, quiere decir que ya hay una representación en base de datos de ese objeto de tipo `Persona`, entonces estamos modificando el objeto (`update`).

Es importante observar el uso de `<s:hidden>` para incluir el campo de `idPersona`. Así que aunque no observemos este valor en el formulario, sí se está enviando al navegador como campo oculto, pudiendo tener un valor nulo si es un nuevo registro, o un valor distinto de nulo si es un registro que se está modificando.

Este archivo no es necesario depositarlo en la carpeta [/WEB-INF/content](#), sin embargo lo depositaremos allí para seguir manteniendo un orden en la organización de archivos JSPs.

# PASO 29. CREAR EL ARCHIVO JSP

- Creamos el archivo persona.jsp:



# PASO 29. CREAR EL ARCHIVO JSP

- Creamos el archivo 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 30. MODIFICAMOS EL CÓDIGO

## Archivo 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="pform.detalle" /></title>
  </head>
  <body>
    <h1><s:text name="pform.detalle" /></h1>
    <a href="<s:url action="listar"/>"><s:text name="pform.listado" /></a>

    <s:form action="guardarPersona">
      <s:hidden name="persona.idPersona" />
      <s:textfield name="persona.nombre" key="p.nombre" />
      <s:textfield name="persona.apellidoPaterno" key="p.apePat" />
      <s:textfield name="persona.apellidoMaterno" key="p.apeMat" />
      <s:textfield name="persona.email" key="p.email" />
      <s:submit action="guardarPersona" key="pform.enviar"/>
    </s:form>
  </body>
</html>
```

# PASO 31. 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.



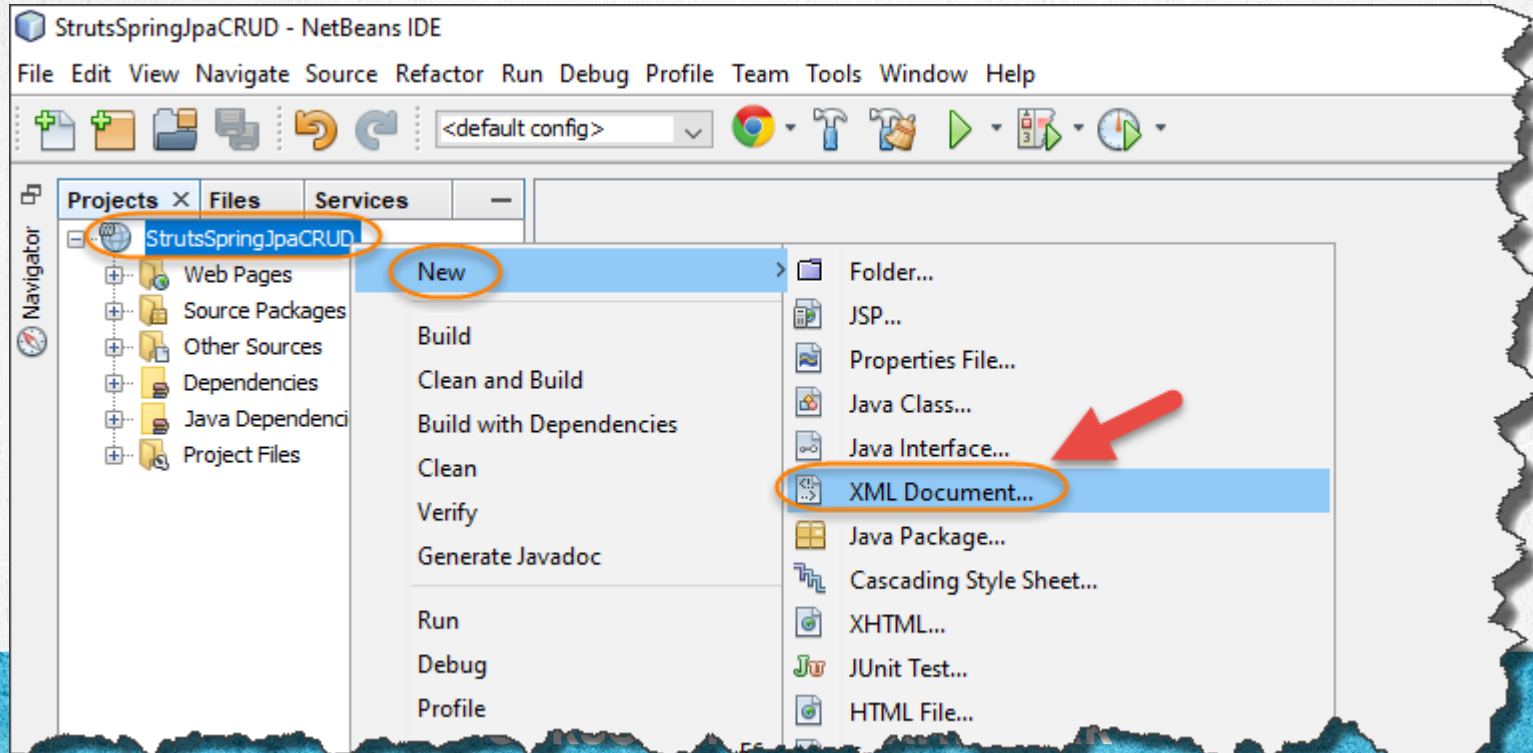
Experiencia y Conocimiento para tu vida

**CURSO STRUTS FRAMEWORK**

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

# PASO 31. CREAR EL ARCHIVO LOG4J2.XML

- Creamos el archivo log4j2.xml como sigue:





# PASO 31. 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:

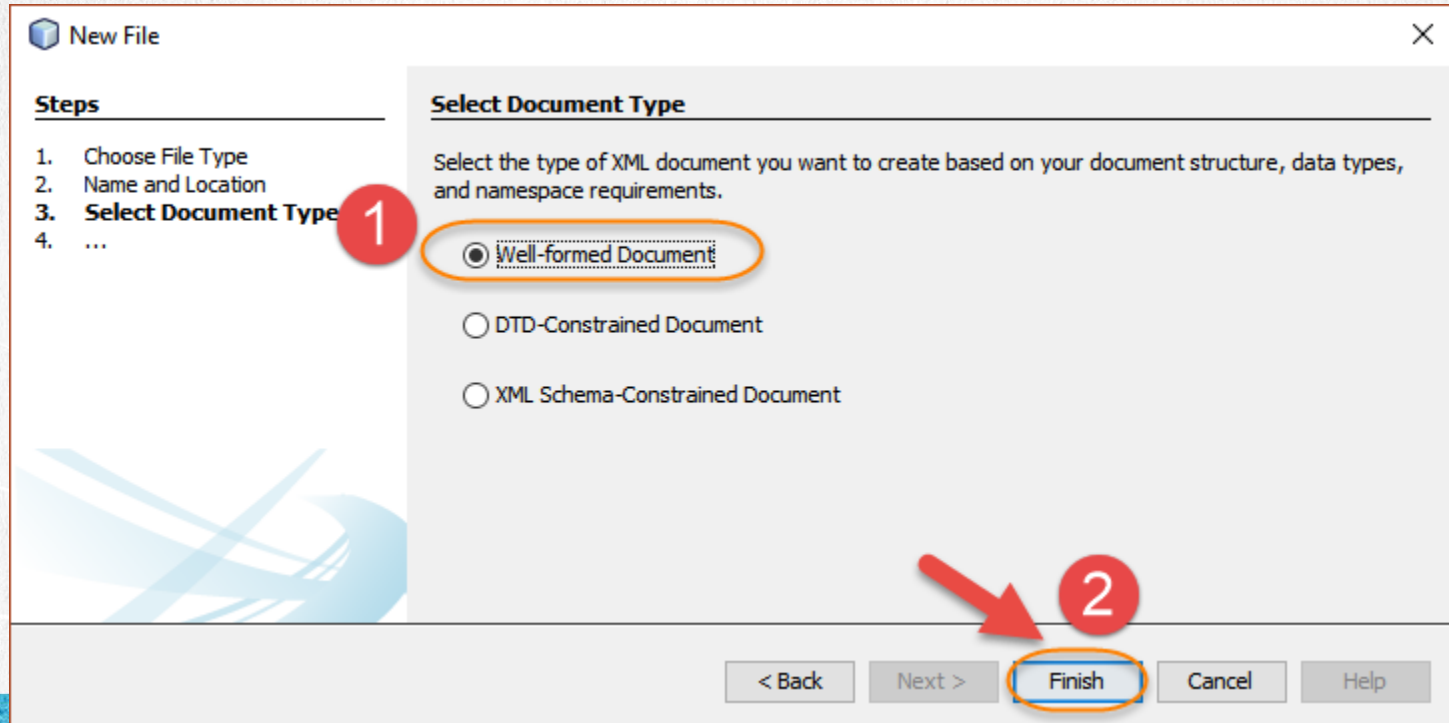
< Back **Next >** Finish Cancel Help

**CURSO STRUTS FRAMEWORK**

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

# PASO 31. CREAR EL ARCHIVO LOG4J2.XML

- Seleccionamos la opción mostrada:



**CURSO STRUTS FRAMEWORK**

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

# PASO 32. MODIFICAMOS EL CÓDIGO

Archivo log4j2.xml:

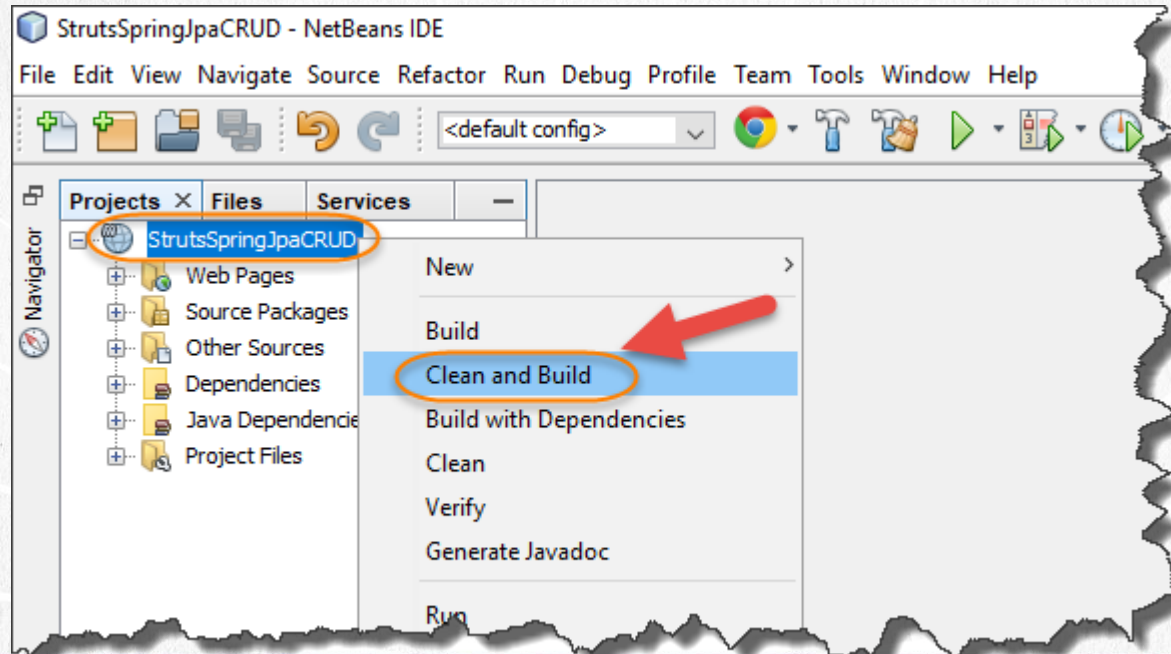
Clic para ver el archivo

```
<?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 33. HACEMOS CLEAN & BUILD

- Hacemos Clean & Build para asegurarnos que tenemos todo listo:

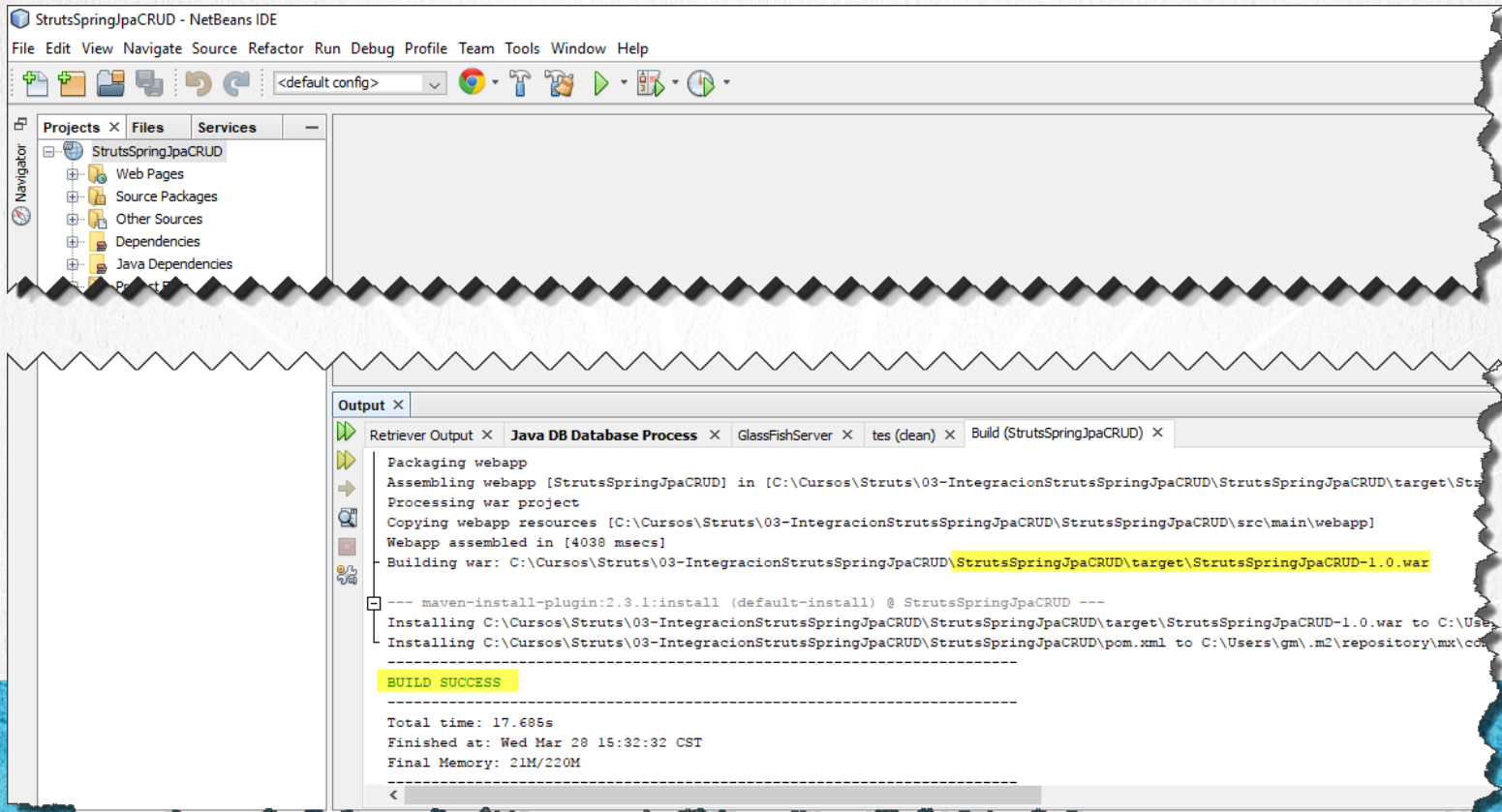


**CURSO STRUTS FRAMEWORK**

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

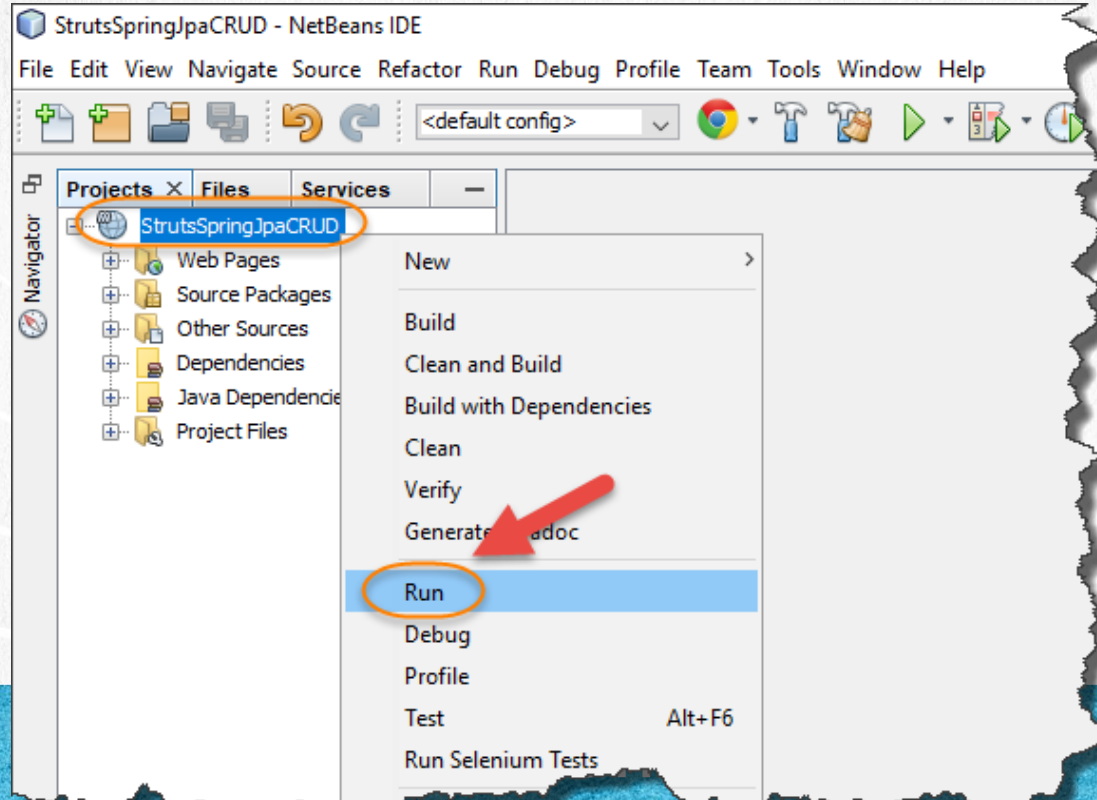
# PASO 33. EJECUTAMOS CLEAN & BUILD

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



# PASO 34. EJECUTAMOS LA APLICACIÓN

- Ejecutamos la aplicación StrutsSpringJpaCRUD como sigue:





# PASO 34. EJECUTAMOS LA APLICACIÓN

- Ejecutamos la aplicación como sigue:

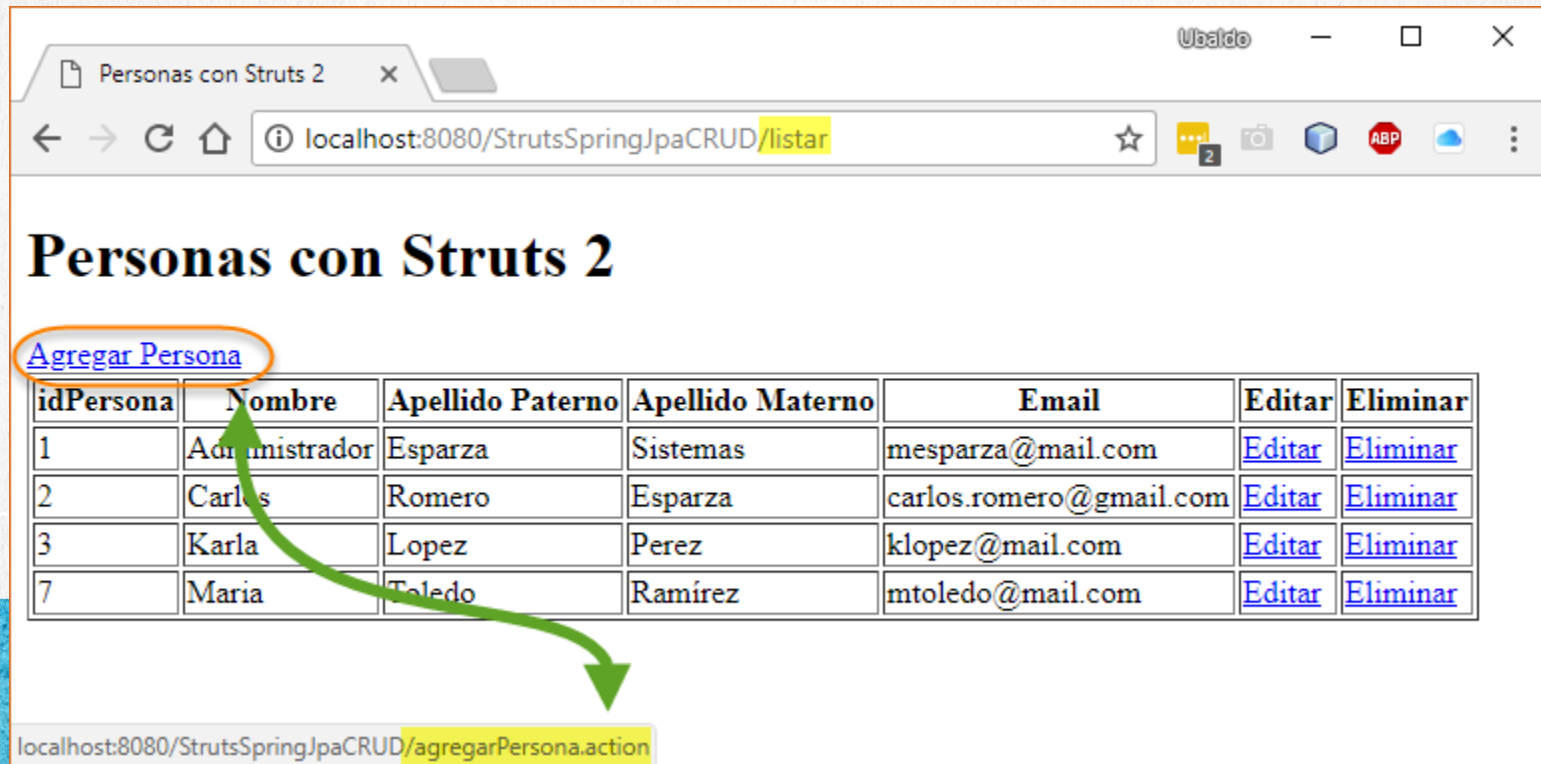


**CURSO STRUTS FRAMEWORK**

www.globalmentoring.com.mx

# PASO 34. 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. Si damos click en el link de Agregar Persona nos llevará a otra vista:



The screenshot shows a web browser window with the title "Personas con Struts 2". The address bar displays "localhost:8080/StrutsSpringJpaCRUD/listar". The main content area has the heading "Personas con Struts 2" and a link "Agregar Persona" circled in orange. Below the link is a table with columns: idPersona, Nombre, Apellido Paterno, Apellido Materno, Email, Editar, and Eliminar. A green arrow points from the "Agregar Persona" link to the URL "localhost:8080/StrutsSpringJpaCRUD/agregarPersona.action" at the bottom of the browser window.

idPersona	Nombre	Apellido Paterno	Apellido Materno	Email	Editar	Eliminar
1	Administrador	Esparza	Sistemas	mesparza@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
2	Carlos	Romero	Esparza	carlos.romero@gmail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
3	Karla	Lopez	Perez	klopez@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
7	Maria	Toledo	Ramírez	mtoledo@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>

localhost:8080/StrutsSpringJpaCRUD/agregarPersona.action

# PASO 34. EJECUTAMOS LA APLICACIÓN

- En esta pantalla podemos regresar al listado de personas, o enviar los nuevos datos para agregar un nuevo objeto de tipo Persona como se observa.

Ubaldo

Detalle Persona

localhost:8080/StrutsSpringJpaCRUD/agregarPersona.action

## Detalle Persona

[Regresar listado de Personas](#)

Nombre:

Apellido Paterno:

Apellido Materno:

Email:



# PASO 34. EJECUTAMOS LA APLICACIÓN

- Después de agregar un nuevo registro, ya deberemos verlo reflejado en nuestro listado. También podemos editar o eliminar un registro del listado. Podemos observar que en caso de dar click en Editar sobre algún registro se selecciona cual es el idPersona que queremos modificar:

Ubaldo

Personas con Struts 2

localhost:8080/StrutsSpringJpaCRUD/listar

## Personas con Struts 2

[Agregar Persona](#)

idPersona	Nombre	Apellido Paterno	Apellido Materno	Email	Editar	Eliminar
1	Administrador	Esparza	Sistemas	mesparza@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
2	Carlos	Romero	Esparza	carlos.romero@gmail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
3	Karla	Lopez	Perez	klopez@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
7	Maria	Toledo	Ramírez	mtoledo@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
8	Juan	Ortega	Morales	jortega@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>

localhost:8080/StrutsSpringJpaCRUD/editarPersona.action?persona.idPersona=8

# PASO 34. EJECUTAMOS LA APLICACIÓN

- En esta pantalla nuevamente vemos el detalle del objeto seleccionado, sin embargo como ya tiene un idPersona, en lugar de hacer un insert, se hará un update. Esta lógica está dentro del objeto PersonasAction. Guardamos los cambios indicados:

Detalle Persona

[Regresar listado de Personas](#)

Nombre:

Apellido Paterno:

Apellido Materno:

Email:

# PASO 34. EJECUTAMOS LA APLICACIÓN

- Después de modificar el registro seleccionado, ya deberemos verlo reflejado los cambios. También podemos eliminar un registro del listado. Si damos clic en Eliminar sobre algún registro se selecciona cual es el idPersona que queremos eliminar:

The screenshot shows a web browser window with the title 'Personas con Struts 2'. The address bar shows the URL 'localhost:8080/StrutsSpringJpaCRUD/listar'. The page content includes a link 'Agregar Persona' and a table with the following data:

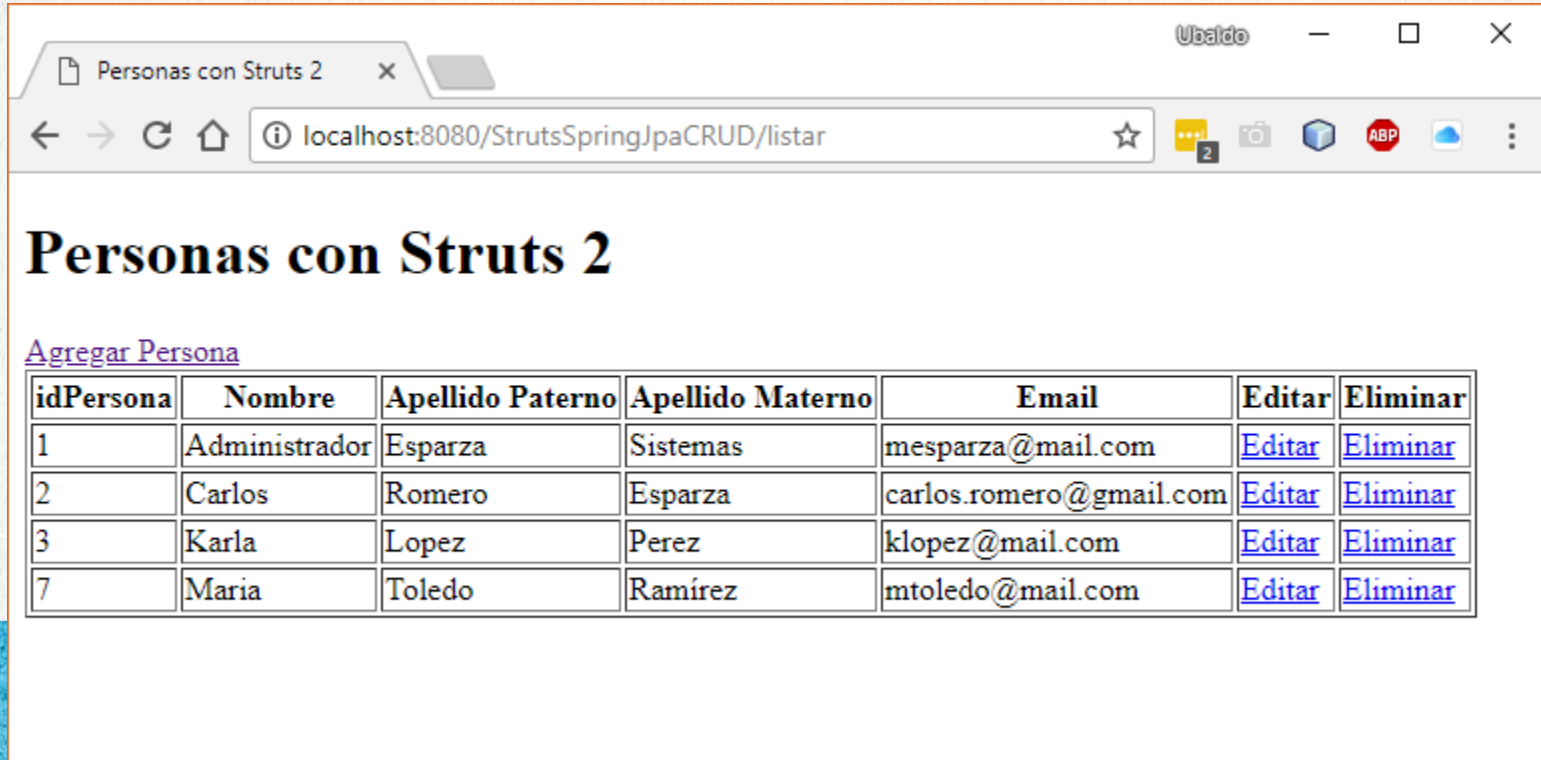
idPersona	Nombre	Apellido Paterno	Apellido Materno	Email	Editar	Eliminar
1	Administrador	Esparza	Sistemas	mesparza@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
2	Carlos	Romero	Esparza	carlos.romero@gmail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
3	Karla	Lopez	Perez	klopez@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
7	Maria	Toledo	Ramírez	mtoledo@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
8	Juan	Ortega	Caballero	jortegac@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>

Below the table, the browser's address bar shows the URL 'localhost:8080/StrutsSpringJpaCRUD/eliminarPersona.action?persona.idPersona=8'. A green arrow points from the 'Eliminar' button in the row for idPersona=8 to this URL.



# PASO 34. EJECUTAMOS LA APLICACIÓN

- Y con esto ya hemos ejecutado las 4 operaciones: Listar, Agregar, Modificar y Eliminar, o en inglés: CRUD (Create-Read-Update-Delete).



Ubaldo

Personas con Struts 2

localhost:8080/StrutsSpringJpaCRUD/listar

## Personas con Struts 2

[Agregar Persona](#)

idPersona	Nombre	Apellido Paterno	Apellido Materno	Email	Editar	Eliminar
1	Administrador	Esparza	Sistemas	mesparza@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
2	Carlos	Romero	Esparza	carlos.romero@gmail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
3	Karla	Lopez	Perez	klopez@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>
7	Maria	Toledo	Ramírez	mtoledo@mail.com	<a href="#">Editar</a>	<a href="#">Eliminar</a>

# 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 otros frameworks y tecnologías como Spring 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 apoyarte en cualquier momento de 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
- Spring Framework
- JPA (Java Persistence API)

En este ejercicio hemos aplicado las operaciones CRUD (Create-Read-Update-Delete) para la tabla de persona.

Es importante mencionar que según se indicó a lo largo del ejercicio, el cambio mayor se realizó en la capa de presentación, es decir, en la clase Action de Struts y en la vista los JSP's. Así que todo lo que se realizó en el ejercicio anterior respecto a la capa de servicio (negocio) y la capa de datos se reutilizó por completo, de esta manera hemos observado como al aplicar las mejores prácticas y arquitecturas, el impacto sobre nuestra aplicación se reduce y simplifica.



**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)