ACADEMIA JAVA

Luis Wonen Olvera Vasquez
Examen Semana 3

Lugar:Ciudad de México
Fecha: 09/12/2022

1. Realiza un programa con pruebas unitarias

Pruebas unitarias: Las pruebas unitarias son una técnica de testeo mediante la cual se prueban módulos individuales o un componente individual de la aplicación de software para determinar si hay algún problema en el desarrollo. El objetivo es validar los componentes de la unidad código individual de la aplicación con su rendimiento. El propósito por tanto de las pruebas unitarias es probar la corrección del código aislado.

JUnit: es un marco de pruebas unitarias para Java. Es utilizada para realizar pruebas unitarias de una pequeña porción de código, aumentando así la productividad del desarrollador, la estabilidad del código y el tiempo dedicado a la depuración.

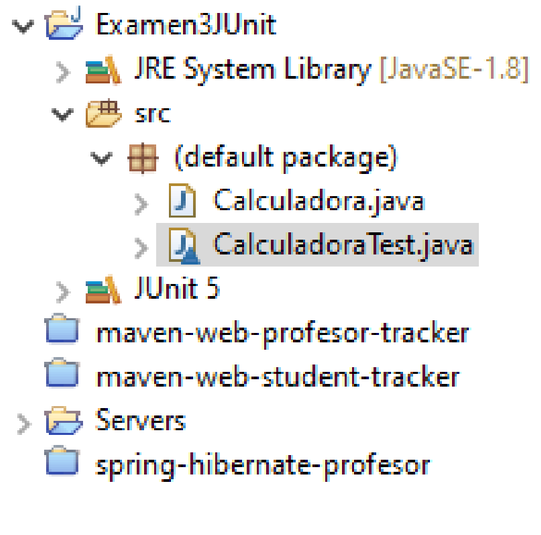
JUnit promueve la idea de probar y luego codificar, que consiste en configurar los datos de prueba para una pieza de código que se puede probar primero y luego implementar. Esta práctica se basa en probar un poco, codificar un poco, probar un poco, codificar un poco. Aumenta la productividad del programador y la estabilidad del código del programa, lo que a su vez reduce el estrés del programador y el tiempo dedicado a la depuración.

1.2 Realiza un programa con pruebas unitarias

Dentro de la carpeta Examenes/Semana 3 encontraremos el paquete Examen3JUnit. El objetivo del porgrama es hacer uso del Test de JUnit por lo tanto, se implementará una calculadora y sus respectivas operaciones.

Estas operaciones son las que se van a probar por lo tanto creamos la clase Calculadora.java en esta clase declaramos las variables, los constructores y los métodos correspondientes.

Además creamos la clase CalculadoraTest en donde con la anotación @Test nos indicara el contenido del test, y al ejecularlo nos marcará cuales fueron test exitosos y cuales test no lo fueron.



Calculadora.java | CalculadoraTest.java

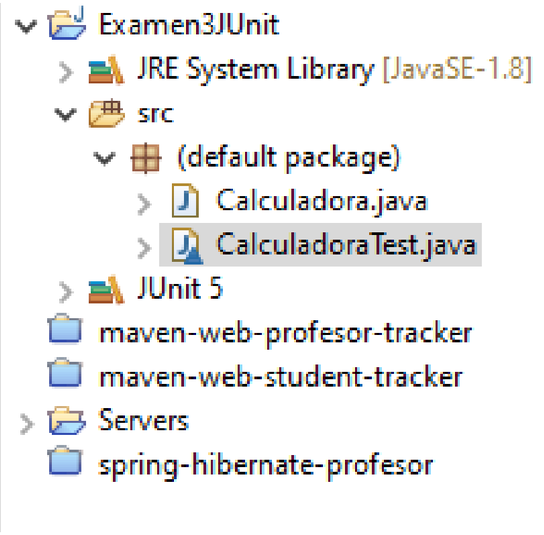
```java
public class Calculadora {

    //Creamos las variables de tipo entero que almacenaran al número
    int numero1;
    int numero2;
    int numero3;
    int numero4;

    public Calculadora(int numero1, int numero2, int numero3, int numero4) {
        super();
        this.numero1 = numero1;
        this.numero2 = numero2;
        this.numero3 = numero3;
        this.numero4 = numero4;
    }


    public Calculadora(int a, int b) {
        numero1 = a;
        numero2 = b;
    }
```

Calculadora.java | CalculadoraTest.java

```java

    public boolean operacion() {
        if(numero2 == numero1 && numero3 != numero4) {
            System.out.println("Se cumple la condición en la que el numero 1 y 2 son iguales mientras que numero 3 y
            boolean rest = true;
            return rest;
        }else {
            System.out.println("No se cumple la condición en la que el numero 1 y 2 son iguales mientras que numero
            boolean rest = true;
            return rest;
        }
    }
```

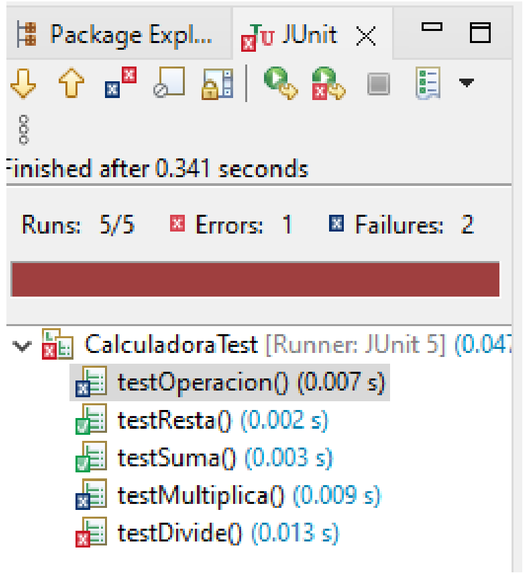
1.3 Realiza un programa con pruebas unitarias

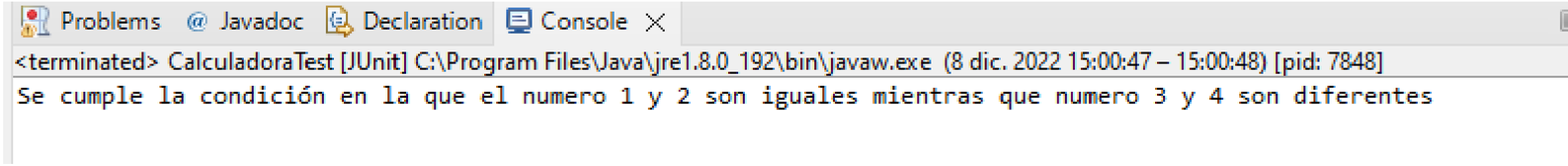
A continuación mostraremos la clase CalculadoraTest en donde se implementan los @Test, tambien mostraremos los resultados donde sabremos con ayuda del assertsEquals si los resultados que recibimos son los mismo que esperamos.



CalculadoraTest.java

```java
import static org.junit.jupiter.api.Assertions.*;

class CalculadoraTest {

    @Test
    void testOperacion() {
        Calculadora calcu = new Calculadora(1,1,2,3);
        boolean resultado = calcu.operacion();
        assertEquals("TRUE",resultado);
    }

    @Test
    void testSuma() {
        Calculadora calcu = new Calculadora(1000,10);
        int resultado = calcu.suma();
        assertEquals(1010,resultado);
    }

    @Test
    void testResta() {
        Calculadora calcu = new Calculadora(1000,0);
        int resultado = calcu.resta();
        assertEquals(1000,resultado);
    }

    @Test
    void testMultiplica() {
        Calculadora calcu = new Calculadora(-15,2);
```

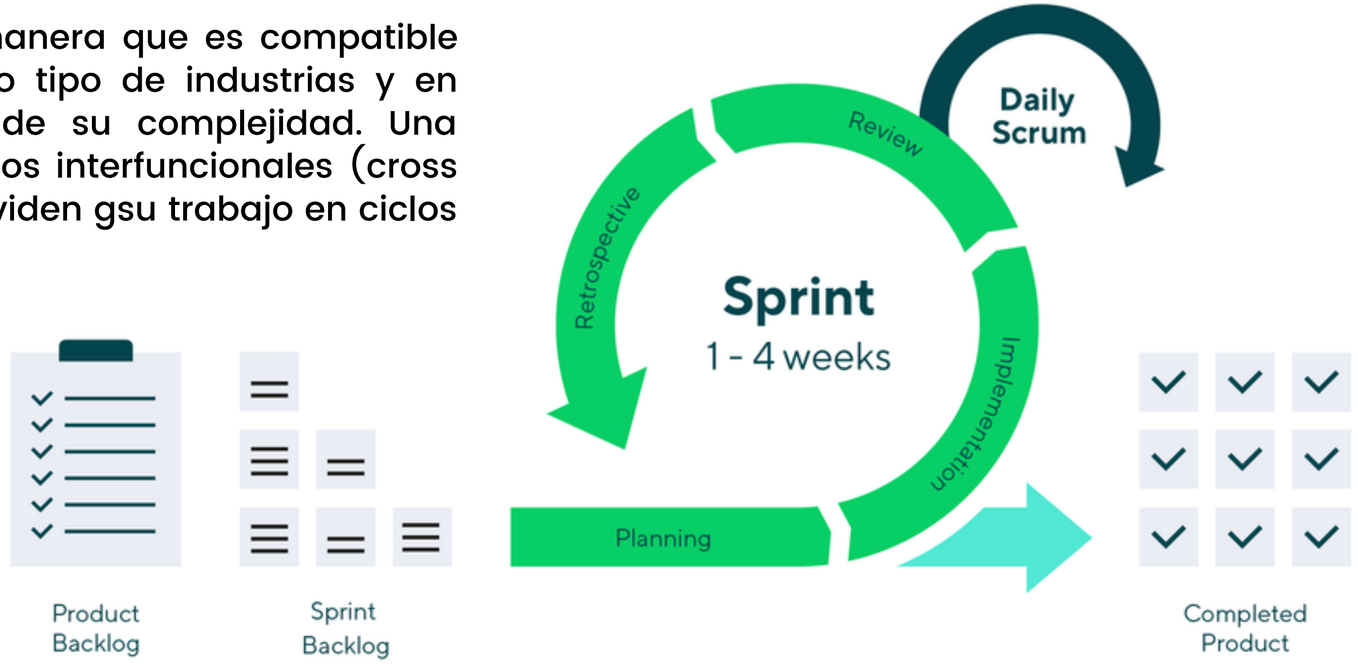




2. Cómo implementar SCRUM

Scrum es uno de los métodos ágiles más populares. Es un framework adaptable, iterativo, rápido, flexible y eficaz, diseñado para ofrecer un valor considerable en forma rápida a lo largo del proyecto. Scrum garantiza transparencia en la comunicación y crea un ambiente de responsabilidad colectiva y de progreso continuo.

El framework de Scrum, está estructurado de tal manera que es compatible con el desarrollo de productos y servicios en todo tipo de industrias y en cualquier tipo de proyecto, independientemente de su complejidad. Una fortaleza clave de Scrum radica en el uso de equipos interfuncionales (cross functional), autoorganizados y empoderados que dividen gsu trabajo en ciclos de trabajo cortos y concentrados llamados Sprints.



2.1 Cómo implementar SCRUM

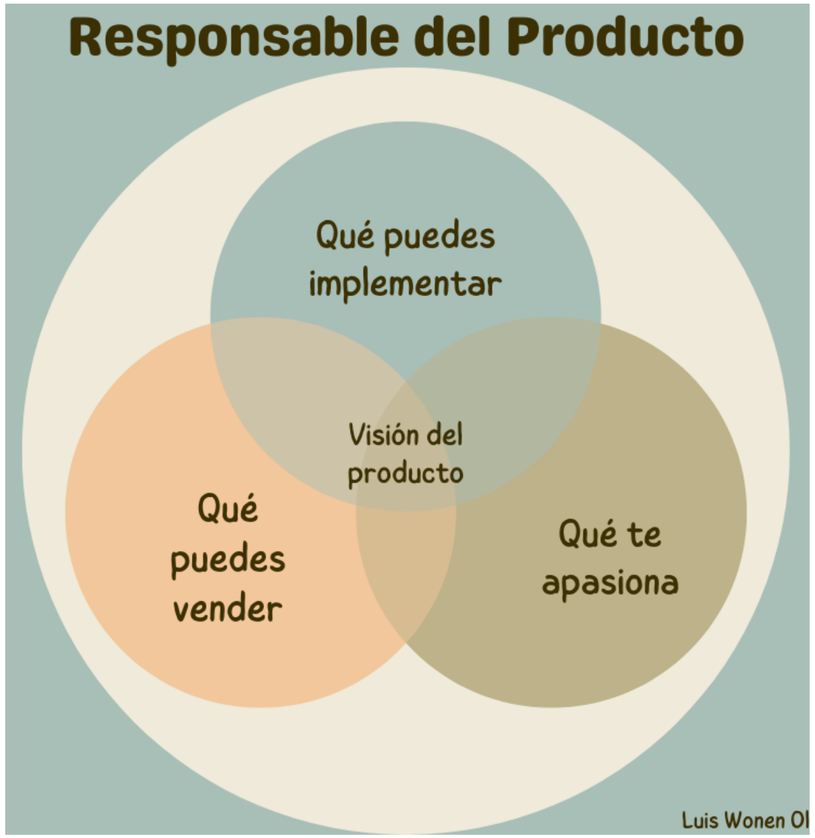
1.- Elige un responsable del producto: Este individuo es quien posee la visión de lo que vas a hacer, producir o lograr. Debe equilibrar múltiples atributos del producto, por lo que toma en cuenta los riesgos y recompensas, qué es posible, qué puede hacerse y qué es lo que le apasiona con lo anterior el responsable del producto debe tener la visión del producto.

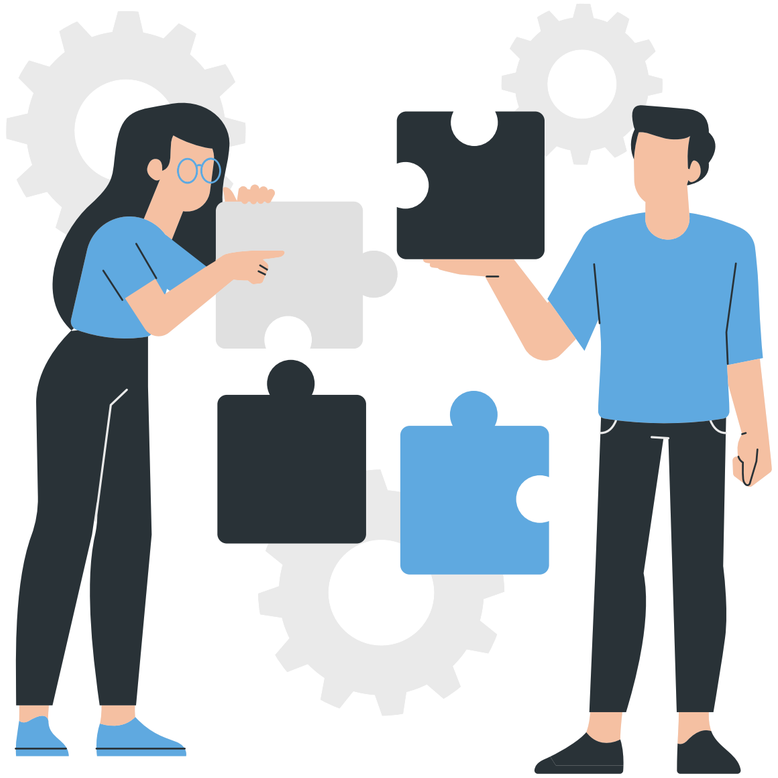
2.-Selecciona un equipo: ¿Quiénes serán las personas que harán efectivamente el trabajo? Este equipo debe contar con todas las habilidades necesarias para tomar la visión de los responsables del producto y hacerla realidad. Los equipos deben ser pequeños, de tres a nueve personas por regla general.

3.-Elige un Scrum Master. Ésta es la persona que capacitará al resto del equipo en el enfoque Scrum y que ayudará al equipo a eliminar todo lo que lo atrasa.

4.-Crea y prioriza una bitácora del producto: La bitácora es una lista que contiene todas las necesidades para lograr el producto. Esta bitácora debe contener todo lo que el equipo podría hacer con su debido orden de prioridad.

La bitácora del producto es única y el responsable del .producto es quien prioriza las decisiones en cuanto al producto, de la misma forma el responsable del producto debe consultar a todos los necesitados como al equipo para cerciorarse de que representa lo que la gente necesita y lo que se puede hacer.





2.2 Cómo implementar SCRUM

5.-Afina y estima la bitácora del producto: Una de las actividades más importantes a realizar es la estimación del esfuerzo necesario para realizar los elementos de la bitácora, para esto el equipo de trabajo debe examinar el elemento realizando preguntas como:

- ¿Hay suficiente información para llevar a cabo el elemento?
- ¿Éste es lo bastante pequeño para estimarse?
- ¿Existe una definición de terminado?
- ¿Esto crea valor visible?

6.- Planeación del sprint: El sprint consiste en reuniones de una extensión fija preferentemente inferior a un mes, esto es planeado por el equipo, scrum master y responsable del producto con el propósito de mostrar el avance en el proyecto, además de las mejoras, modificaciones que se tengan.

Otro propósito del sprint es conocer la velocidad del equipo (Número de puntos que acumulo el sprint) ya que es deber del scrum master y el equipo tratar de aumentar la velocidad del equipo. Durante esta reunión todos deben acordar asimismo una meta de sprint, que todos han de cumplir en este sprint

2.3 Cómo implementar SCRUM

7.- Vuelve visible el trabajo: El Dashboard (tabla de scrum), contiene tres columnas: Pendiente, En proceso y Terminado. Esto nos sirve para representar el estado de los elementos de esta forma sabremos cuándo se encuentran pendientes, en proceso o terminados.

Otra manera de volver visible el trabajo es crear un diagrama de finalización. En un eje aparece el número de puntos que el equipo introdujo en el sprint y en el otro el número de días. Cada día, el Scrum Master suma el número de puntos completados y los grafica en el diagrama de finalización. Idealmente, habrá una pendiente descendente que conduzca a cero puntos para el último día del sprint.

8.- Parada diaria o Scrum diario: Éste es el pulso de Scrum. Cada día, a la misma hora, durante no más de quince minutos, el equipo y el Scrum Master se reúnen y contestan tres preguntas:

- ¿Qué hiciste ayer para ayudar al equipo a terminar el sprint?
- ¿Qué harás hoy para ayudar al equipo a terminar el sprint?
- ¿Algún obstáculo te impide o impide al equipo cumplir la meta del sprint?

Lo que esto hace es ayudar al equipo a saber exactamente dónde se encuentra todo en el curso de un sprint.

2.4 Cómo implementar SCRUM

9.- Revisión del sprint o demostración del sprint: Ésta es la reunión en la que el equipo muestra lo que hizo durante el sprint. Todos pueden asistir, no sólo el responsable del producto, el Scrum Master y el equipo, sino también los demás interesados, la dirección, clientes, quien sea.

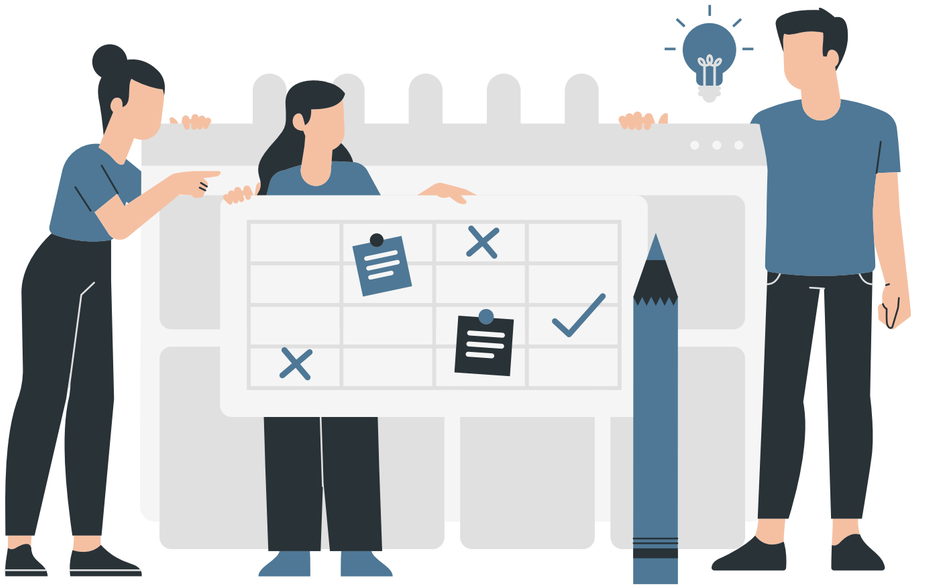
Ésta es una reunión abierta en la que el equipo hace una demostración de lo que pudo llevar a Terminado durante el sprint. El equipo debe mostrar únicamente lo que satisface la definición de Terminado, lo total y completamente concluido y que puede entregarse sin trabajo adicional. Esto puede no ser un producto terminado, pero sí una función concluida de uno de ellos.

10. Retrospectiva del sprint: Una vez que el equipo ha mostrado lo que logró en el sprint más reciente la cosa “terminada” y en posibilidad de enviarse a los clientes en busca de realimentación, podemos pensar en qué marchó bien, qué pudo haber marchado mejor y qué puede mejorar en el siguiente sprint, lo anterior nos lleva a formularnos la siguiente pregunta ¿Cuál es la mejora en el proceso que como equipo pueden implementar de inmediato?

2.5 Cómo implementar SCRUM

Es crucial que la gente, como equipo, asuma la responsabilidad de su proceso y de sus resultados y busque soluciones también como equipo. Al mismo tiempo, debe tener fortaleza para tocar los temas que le incomodan de un modo orientado a la solución, no acusatorio. Y el resto del equipo ha de tener la madurez de oír la realimentación, aceptarla y buscar una solución, no ponerse a la defensiva.

Al final de la reunión, el equipo y el Scrum Master deben acordar una mejora al proceso que implementarán en el siguiente sprint. Esa mejora al proceso, también llamada kaizen, debe incorporarse en la bitácora del sprint siguiente, con pruebas de aceptación. De esta manera, el equipo podrá ver fácilmente si en verdad implementó la mejora y qué efecto tuvo ésta en la velocidad.



11.-Comienza de inmediato el ciclo del siguiente sprint, tomando en cuenta la experiencia del equipo con los impedimentos y mejoras del proceso.

3. Spring Batch

Spring Batch: Es un marco de trabajo por lotes ligero y completo diseñado para permitir el desarrollo de aplicaciones por lotes sólidas vitales para las operaciones diarias de los sistemas empresariales.

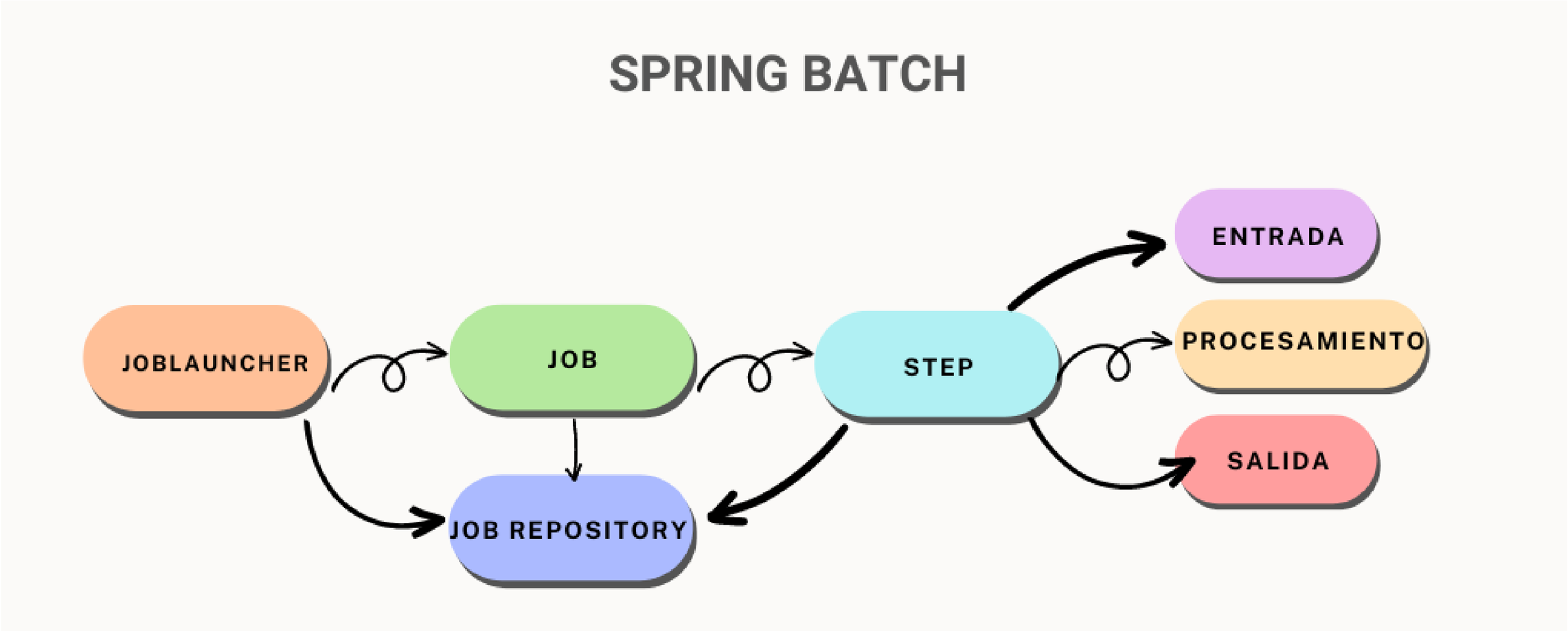
Spring Batch proporciona funciones reutilizables que son esenciales en el procesamiento de grandes volúmenes de registros, incluido el registro/rastreo, la gestión de transacciones, las estadísticas de procesamiento de trabajos, el reinicio de trabajos, la omisión y la gestión de recursos.

También proporciona funciones y servicios técnicos más avanzados que permitirán trabajos por lotes de alto rendimiento y volumen extremadamente alto a través de técnicas de optimización y partición. Los trabajos por lotes de gran volumen, tanto simples como complejos, pueden aprovechar el marco de una manera altamente escalable para procesar volúmenes significativos de información.

JobRepository lo podemos visualizar como se almacena la información (tablas). Joblauncher dispara el procesos batch, puede ser disparado por un proceso automatizado que se realiza a un tiempo especifico o por ser una persona.

Después de disparar puede existir muchos Job como actualizar, insertar, etc. Un proceso batch realiza muchos Job. Cada Job puede estar compuesto de 1 o muchos Step. El step, job y joblauncher persisten en el job repository (base de datos). Un Step esta compuesto de 3 componentes, una entrada, un procesamiento y una salida.

3.1 Spring Batch



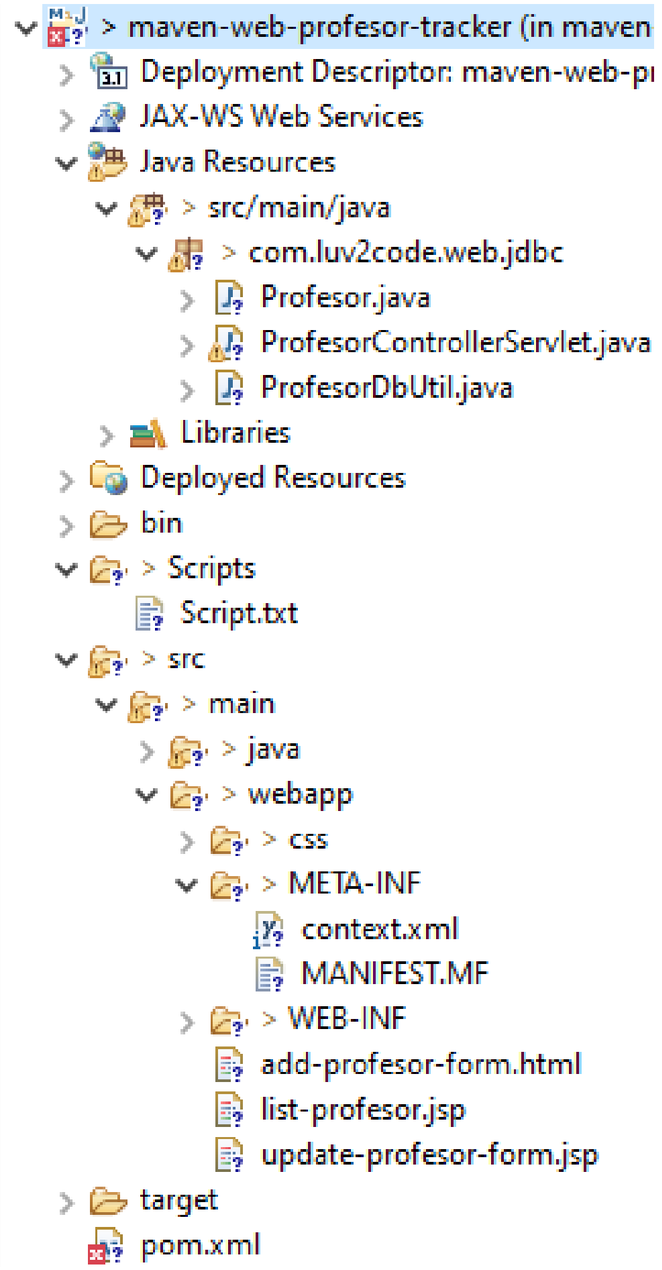
4. CRUD con Maven, JSP y Servlets

Dentro de la carpeta Examenes/Semana 3 encontramos la carpeta maven-web-profesor-tracker, este programa tiene el proposito de hacer un CRUD con el uso de Maven, JSP y Servlet.

Por lo anterior ocuparemos Apache Tomcat v9.0 y para la base de datos ocuparemos MySQL y para manejar la base de datos ocuparemos MySQL Workbech además de IDE Eclipse en su versión EE.

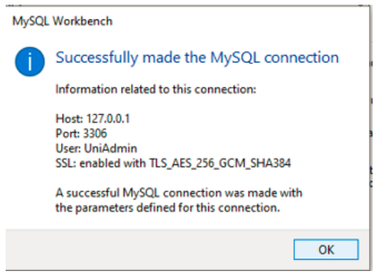
El programa consiste en realizar un CRUD a un profesor por lo que podremos agregar, eliminar, editar y eliminar profesores. Los campos que contemplaremos son: Nombre, Apellido, Email y Especialidad.

El primer paso es crear las clases necesarias Profesor.java, ProfesorControllerServlet.java y ProfesorDbUtil.java. Además de crear los los html y jsp correspondientes en la carpeta WEB-INF.

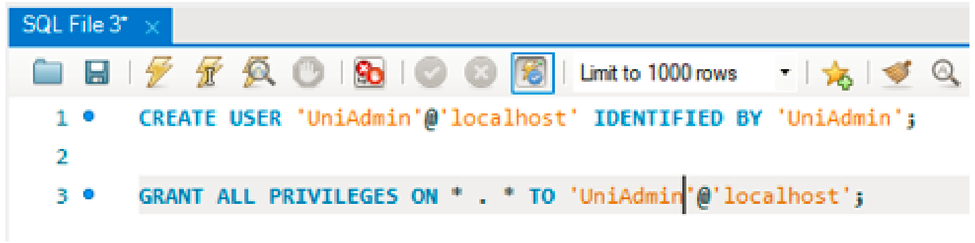


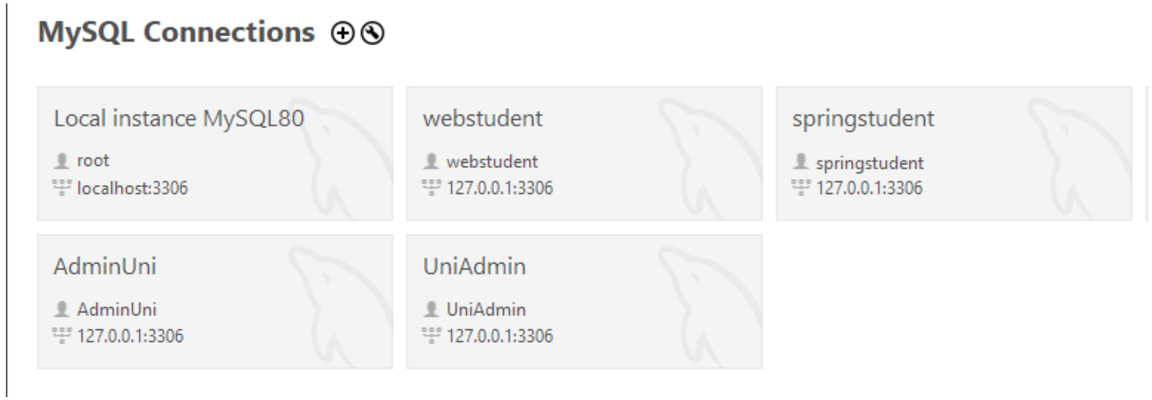
4.1 CRUD con Maven, JSP y Servlets

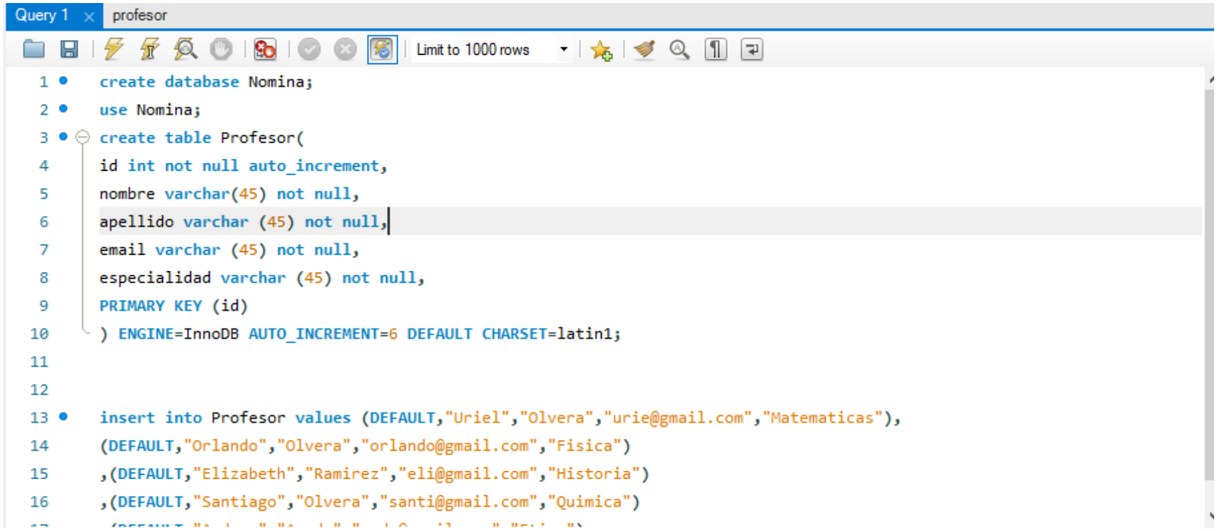
A continuación usaremos el MySQL Workbech para crear un usuario en este caso UniAdmin y aplicarle los permisos necesarios. Después entramos al UniAdmin y asignamos una contraseña y probamos la conexión.



Creamos una base datos Nomina en la cual tendrá una tabla llamada Profesor que contiene elementos como: id auto incrementable, nombre, apellido, email y especialidad.







4.2 CRUD con Maven, JSP y Servlets

Dentro del proyecto maven tenemos un archivo context.xml dentro del cual se realiza la coneccion a la base de datos en este documento pondremos el nombre de la base de datos, nombre del usuario y su contraseña.

También tendremos el web.xml el cual contiene la ruta que nos lleva a la página de bienvenida.

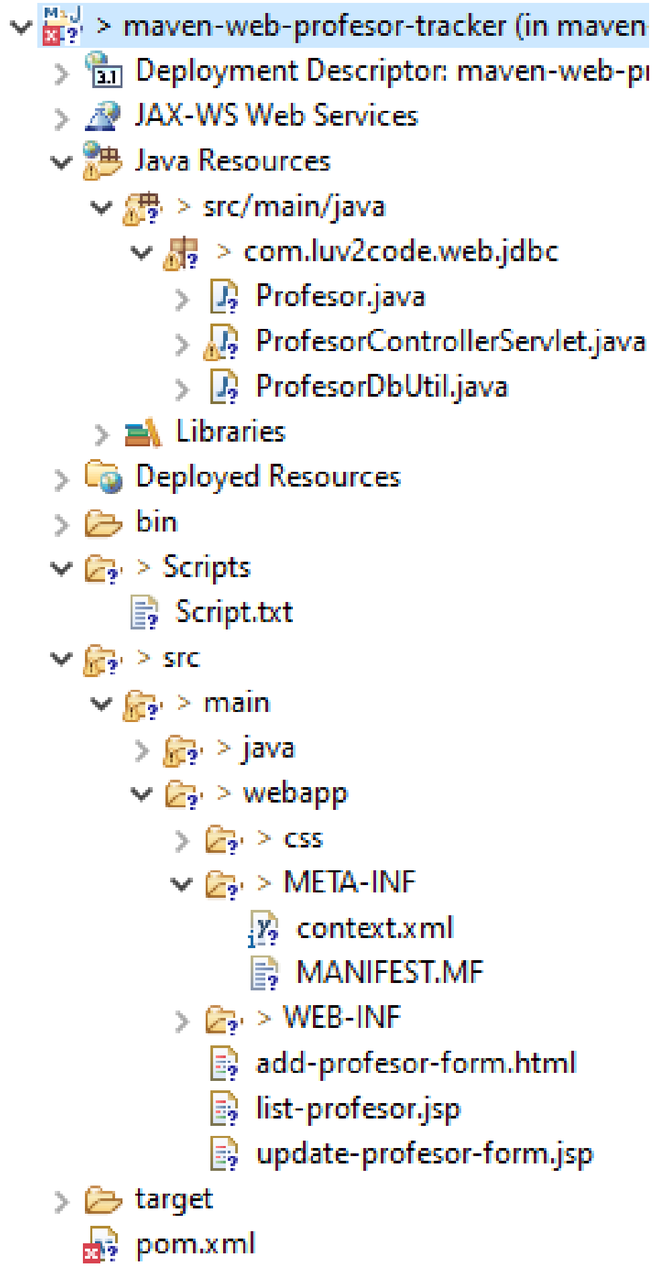
context.xml

maven-web-profesor-tracker/src/main/webapp/META-INF/context.xml

```
  <Resource name="jdbc/Nomina"
            auth="Container" type="javax.sql.DataSource"
               maxActive="20" maxIdle="5" maxWait="10000"
               username="UniAdmin" password="UniAdmin"
               driverClassName="com.mysql.cj.jdbc.Driver"
               url="jdbc:mysql://localhost:3306/Nomina?useSSL=false&amp;serverTimezone=UTC"/>
</Context>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  <display-name>Universidad</display-name>
  <welcome-file-list>
    <welcome-file>ProfesorControllerServlet</welcome-file>
  </welcome-file-list>
</web-app>
```



4.3 CRUD con Maven, JSP y Servlets

Dentro del proyecto maven tenemos un archivo Profesor.java que contiene la clase profesor, los contructores que ocuparemos además de los getters y setters. También tenemos el ProfesorControllerServlet representa el cerebro de nuestro programa tenemos diversos métodos como añadir profesor, cargar profesor, actualizar profesror, borrar profesor.

Por último tenemos el ProfesorDbUtil que contiene la conexión con la base de datos, además de todo lo concerniente a ella como lo son los script sql para actualizar, crear, eliminar y modificar.

Profesor.java

```java
package com.luv2code.web.jdbc;

public class Profesor {

    private int id;
    private String nombre;
    private String apellido;
    private String email;
    private String especialidad;



    public Profesor(int id, Strir
        super();
        this.id = id;
        this.nombre = nombre;
        this.apellido = apellido;
        this.email = email;
        this.especialidad = espec
    }

    public Profesor(String nombre
        super();
        this.nombre = nombre;
```

ProfesorControllerServlet.java

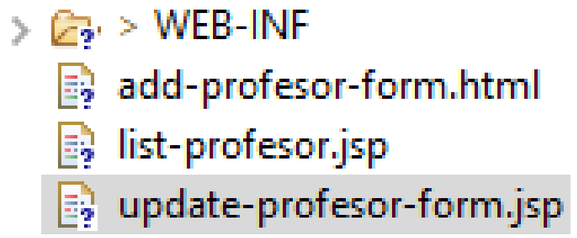
```java
package com.luv2code.web.jdbc;

import java.io.IOException;

/**
 * Servlet implementation class ProfesorControllerServlet
 */
@WebServlet("/ProfesorControllerServlet")
public class ProfesorControllerServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private ProfesorDbUtil profesorDbUtil;

    @Resource(name="jdbc/Nomina") //SE COMENTO PARA HACER USO DE JNDI
    private DataSource dataSource;
```

ProfesorDbUtil.java

```java
package com.luv2code.web.jdbc;

import java.sql.Connection;

public class ProfesorDbUtil {

    private DataSource dataSource;

    public ProfesorDbUtil(DataSource theDataSource) {
        dataSource = theDataSource;
    }

    public List<Profesor> getProfesor() throws Excepti

        List<Profesor> profesor = new ArrayList<>();

        Connection myConn = null;
        Statement myStmt = null;
        ResultSet myRs = null;
```

4.4 CRUD con Maven, JSP y Servlets

Por último tenemos los JSP y el html. Estos son para la vista del usuario, tenemos tres por que son 3 las vistas principales del porgrama, el lis-profesor corresponde a la vista principal, el add-profesor-form corresponde cuando se da click en el botón agregar profesor y el pudate-profesor-form representa la interface cuando queremos actualizar a un profesor.



list-profesor.jsp

```
<c:forEach var="tempProfesor" items="${LISTA_PROFESOR}">

    <!-- set up a link for each student -->
    <c:url var="tempLink" value="ProfesorControllerServlet">
        <c:param name="command" value="LOAD" />
        <c:param name="profesorId" value="${tempProfesor.id}" />
    </c:url>

    <!--  set up a link to delete a student -->
    <c:url var="deleteLink" value="ProfesorControllerServlet">
        <c:param name="command" value="DELETE" />
        <c:param name="profesorId" value="${tempProfesor.id}" />
    </c:url>

    <tr>
        <td> ${tempProfesor.nombre} </td>
        <td> ${tempProfesor.apellido} </td>
        <td> ${tempProfesor.email} </td>
        <td> ${tempProfesor.especialidad} </td>
        <td>
            <a href="${tempLink}">Actualizar</a>
             |
            <a href="${deleteLink}"
            onclick="if (!(confirm('¿Desa borrar el profesor?'))) r
            Borrar</a>
```

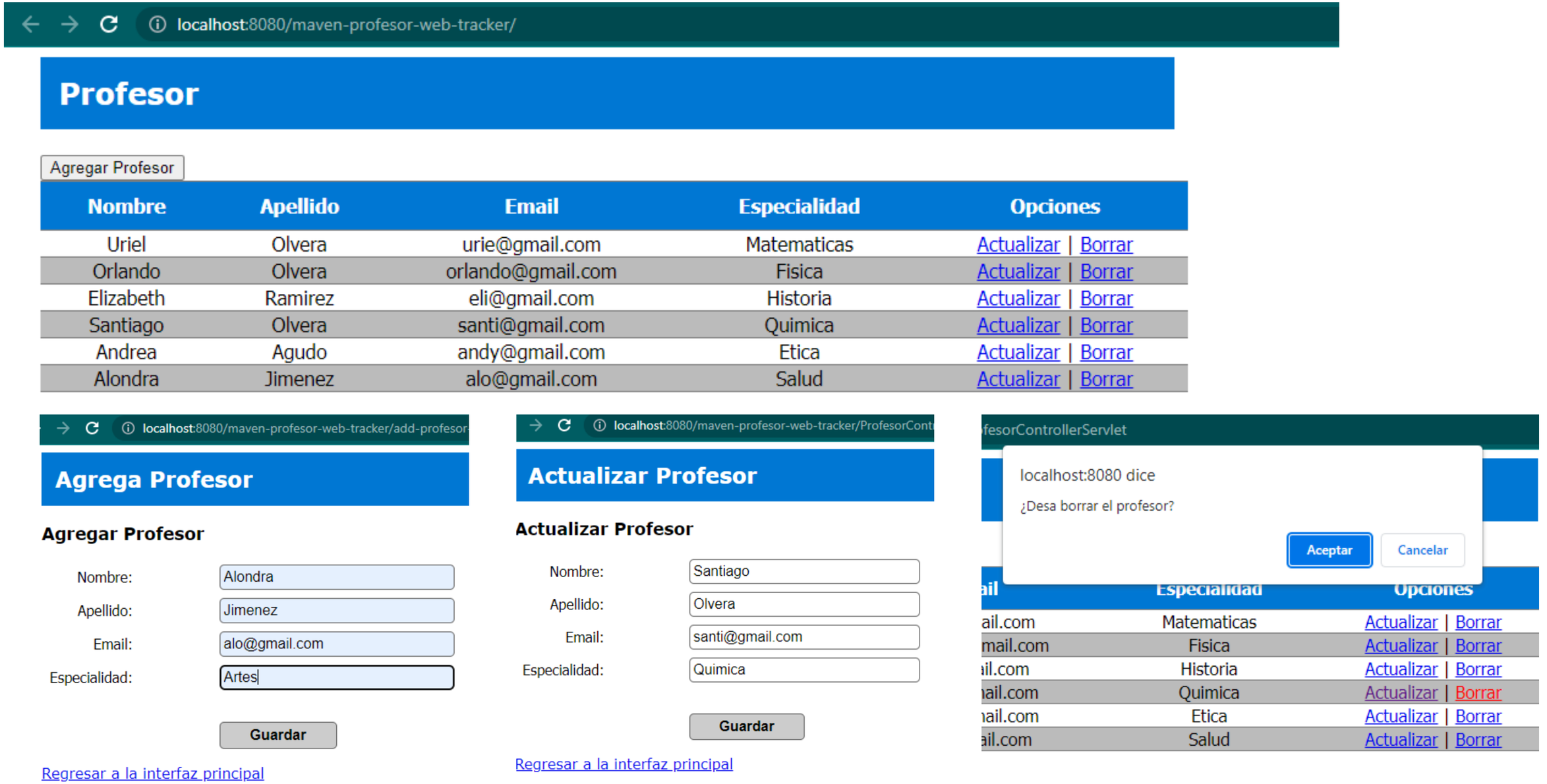
update-profesor-form.jsp

```
<!DOCTYPE html>
<html>

<head>
    <title>Actualizar Profesor</title>

    <link type="text/css" rel="stylesheet" href="css/
    <link type="text/css" rel="stylesheet" href="css/
</head>

<body>
    <div id="wrapper">
        <div id="header">
            <h2>Actualizar Profesor</h2>
        </div>
    </div>

    <div id="container">
        <h3>Actualizar Profesor</h3>

        <form action="ProfesorControllerServlet" meth

            <input type="hidden" name="command" value

            <input type="hidden" name="profesorId" va

            <table>
```

add-profesor-form.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Agregar Profesor</title>

    <link type="text/css" rel="stylesheet" href="css/sty
    <link type="text/css" rel="stylesheet" href="css/add
</head>

<body>
    <div id="wrapper">
        <div id="header">
            <h2>Agrega Profesor</h2>
        </div>
    </div>

    <div id="container">
        <h3>Agregar Profesor</h3>

        <form action="ProfesorControllerServlet" method="

            <input type="hidden" name="command" value="Al

            <table>
                <tbody>
                    <tr>
                        <td><label>Nombre:</label></td>
```

4.5 CRUD con Maven, JSP y Servlets



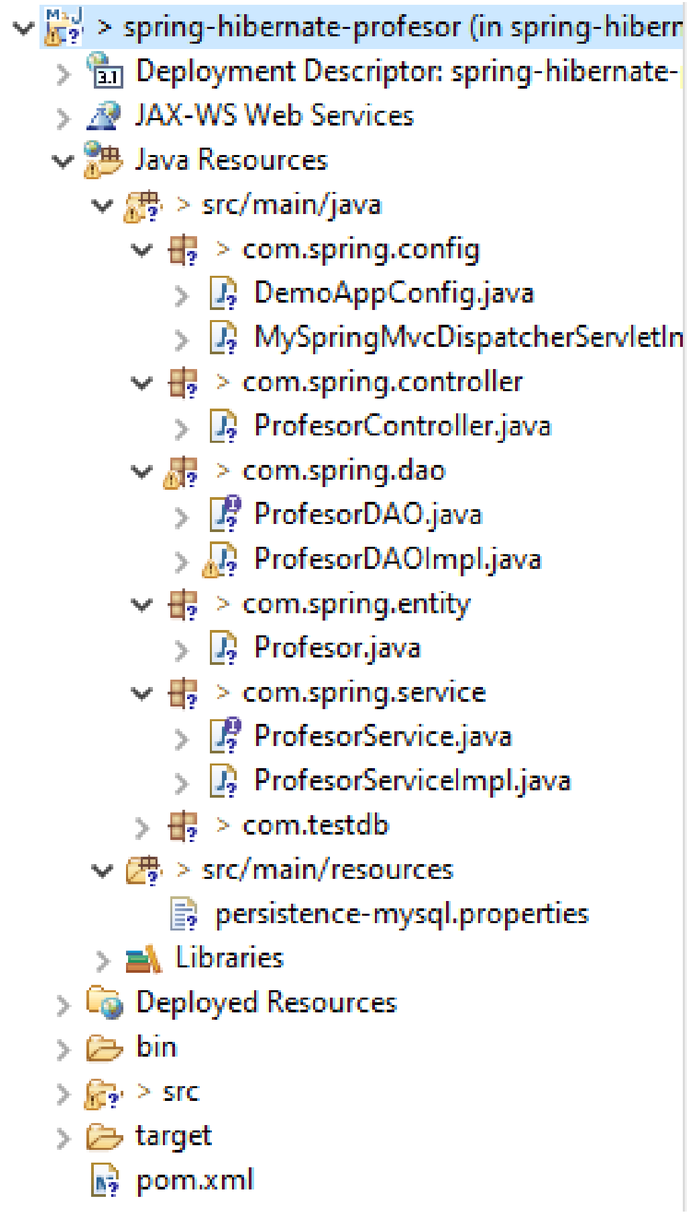
5. CRUD con Spring

Dentro de la carpeta Examenes/Semana 3 encontramos la carpeta spring-hibernate-profesor, este programa tiene el proposito de hacer un CRUD con el uso de spring.

Por lo anterior ocuparemos Apache Tomcat v9.0 y para la base de datos ocuparemos MySQL y para manejar la base de datos ocuparemos MySQL Workbech además de IDE Eclipse en su versión EE.

El programa consiste en realizar un CRUD a un profesor por lo que podremos agregar, eliminar, editar y eliminar profesores. Los campos que contemplaremos son: Nombre, Apellido, Email y Especialidad.

Observamos que existen diversos paquetes ya que dividimos el controller, service y el DAO.



5.1 CRUD con Spring

Dentro del paquete com.spring.config tenemos DemoAppConfig en esta parte tenemos el @ComponentScan que nos proporciona las direcciones de los paquetes además del @PropertySource que proporciona la dirección del persistence-mysql-properties que contiene los detalles de la conexión a la base de datos.

En este caso ocuparemos la misma base de datos que el proyecto pasado, por lo que haremos referencia a la diapositiva 14. Como podemos ver de la misma forma en el paquete entity tenemos a la clase Profesor.java que tiene diversas anotaciones como son @Entity, @Colum, @Table, entre otras, además del contructor vacio y métodos que ocuparemos.

Un aspecto a considerar es que con el uso de anotaciones se ahorran lineas de código.

DemoAppConfig.java

```
1  package com.spring.config;
2
3  import java.beans.PropertyVetoException;
26
27 @Configuration
28 @EnableWebMvc
29 @EnableTransactionManagement
30 @ComponentScan("com.spring")
31 @PropertySource({ "classpath:persistence-mysql.properties" })
32 public class DemoAppConfig implements WebMvcConfigurer {
```

ProfesorController.java

```
1  package com.spring.controller;
2
3  import java.util.List;
16
17 @Controller
18 @RequestMapping("/profesor")
19 public class ProfesorController {
20
21     // need to inject our profesor service
22     @Autowired
23     private ProfesorService profesorService;
24
25     @GetMapping("/list")
26     public String listProfesors(Model theModel) {
27
28         // get profesors from the service
29         List<Profesor> theProfesors = profesorService.getProfesors();
30
31         // add the profesors to the model
32         theModel.addAttribute("profesors", theProfesors);
33
34         return "list-profesors";
35     }
36
37     @GetMapping("/showFormForAdd")
38     public String showFormForAdd(Model theModel) {
39
40         // create model attribute to bind form data
41         Profesor theProfesor = new Profesor();
```

5.2 CRUD con Spring

Los paquetes nos sirven para distinguir quien se encargara del entity, controller, service y dao. Como hablamos anteriormente tenemos las clases Profesor.java, ProfesorController que funge como el cerebro del programa contiene anotaciones como @GetMapping, @PostMapping, @Controller, etc. En el paquete service y dao tenemos las clases que implementan las interfaces.

Dentro del paquete com.spring.service está la clase ProfesorServiceImpl, que implementa la interface ProfesorService, la clase que implementa se encarga de sobreescribir los métodos. En el paquete com.spring.dao la clase ProfesorDAO contiene los scripts sql para realizar las consultas a la base de datos, como Select, update, delete.

ProfesorController.java

```java
package com.spring.controller;

import java.util.List;

@Controller
@RequestMapping("/profesor")
public class ProfesorController {

    // need to inject our profesor service
    @Autowired
    private ProfesorService profesorService;

    @GetMapping("/list")
    public String listProfesors(Model theModel) {

        // get profesors from the service
        List<Profesor> theProfesors = profesorService.

        // add the profesors to the model
        theModel.addAttribute("profesors", theProfesor

        return "list-profesors";
    }

    @GetMapping("/showFormForAdd")
    public String showFormForAdd(Model theModel) {
```

ProfesorDAOImpl.java

```java
package com.spring.dao;

import java.util.List;

@Repository
public class ProfesorDAOImpl implements ProfesorDAO {

    // need to inject the session factory
    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public List<Profesor> getProfesors() {

        // get the current hibernate session
        Session currentSession = sessionFactory.getCurrentSession();

        // create a query  ... sort by last name
        Query<Profesor> theQuery =
                currentSession.createQuery("from Profesor order by apellido",
                                           Profesor.class);

        // execute query and get result list
        List<Profesor> profesor = theQuery.getResultList();

        // return the results
        return profesor;
    }
```

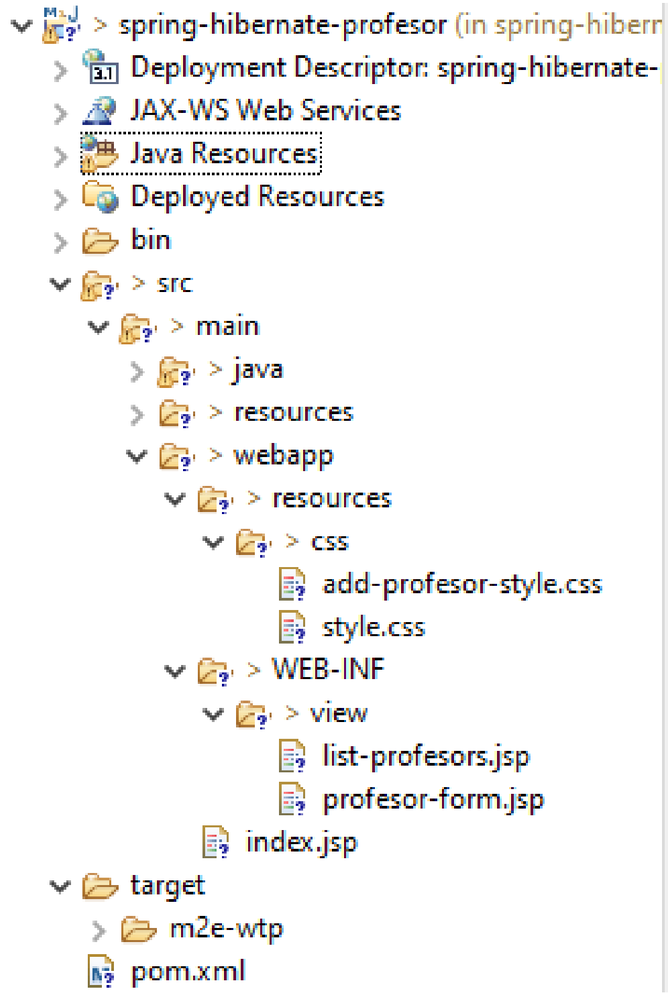
ProfesorServiceImpl.java

```java
package com.spring.service;

import java.util.List;

@Service
public class ProfesorServiceImpl implements ProfesorService {

    // need to inject profesor dao
    @Autowired
    private ProfesorDAO profesorDAO;

    @Override
    @Transactional
    public List<Profesor> getProfesors() {
        return profesorDAO.getProfesors();
    }

    @Override
    @Transactional
    public void saveProfesor(Profesor theProfesor) {

        profesorDAO.saveProfesor(theProfesor);
    }

    @Override
    @Transactional
    public Profesor getProfesor(int theId) {
```

5.3 CRUD con Spring

Para la vista tenemos profesor-form.jsp y list.profesor.jspdentro del list-profesor tenemos la interface principal, la que se nos muestra el ejecutar el programa, mientras que el profesor-form contiene la vista para la actualización del Profesor.

Además de lo anterior tenemos los documentos style.css y add-profesor-style.css que son para proporcionar el estilo a los jsp.



5.4 CRUD con Spring