Aplicación en capas

Aplicación en capas **1**
¿Qué aprenderás? 2
Introducción 2
Análisis de la solución 3
Implementación del sistema 5
POJOS 10
API JDBC 11

¡Comencemos!

¿Qué aprenderás?

- Analizar la solución
- Implementar el sistema.

Introducción

Como programadores java EE, programadores java, o como le quieran llamar al puesto, necesitamos conocer todas estas capas y su implementación, ya sea para la creación de nuevos proyectos o dar mantenimientos a los ya existentes, por lo cual a continuación se empezará a desarrollar un pequeño sistema de gestión de cursos.

Si bien veremos el código paso a paso, el objetivo de la actividad es no solo programar como autómatas, sino que pensar un poco más allá y poder plasmar la comunicación de los componentes de forma estructurada en términos de la arquitectura, para generar un sistema homogéneo y perfectamente interconectado.

El sistema que se desarrollará, es simplemente una aplicación en donde un usuario puede ver una lista de cursos dictados por una institución de educación para luego poder elegir uno e inscribirse en él.

Análisis de la solución

En ingeniería de software, al momento de plasmar los requerimientos de un sistema se utiliza un artefacto llamado casos de uso. No detallaremos casos de uso a niveles de una documentación exhaustiva, pero es bueno saber que vamos a programar. Los siguientes pasos corresponden al caso de uso e inscripción a un curso, que servirá de ejemplo para la futura construcción.

- El usuario solicita una página de inscripción
- El sistema la arma y se la entrega al usuario
- El usuario selecciona un curso y sus datos para que al momento de confirmar, se genere una transacción a la base de datos.

En este trabajo vamos a implementar la inscripción a un curso, y pensaremos en términos de clases y objetos para modelar la solución. Para modelar las entidades debemos hacernos esta pregunta: ¿Qué entidades están involucradas en la inscripción a un curso?

Pensando un poco:

- Queremos inscribirnos a un curso, *curso* es una entidad.
- Si queremos inscribirnos al curso, debemos dar nuestros datos personales y además cómo pagar la inscripción: tenemos la entidad inscripcion.
- Actualmente existen varios métodos de pago, por lo cual tenemos la última entidad que sería la forma de pago.

Estas tres entidades se convertirán en el modelo de datos, con tres tablas:

Object Type **TABLE** Object **CURSO**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
CURSO	ID_CURSO	VARCHAR2	50	-	-	1	-	-	-
	DESCRIPCION	VARCHAR2	100	-	-	-	✓	-	-
	PRECIO	NUMBER	22	-	-	-	✓	-	-
									1 - 3

Imagen 1. Tabla Curso.
Fuente: Desafío Latam

Results Explain **Describe** Saved SQL History

Object Type **TABLE** Object **FORMA_PAGO**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
FORMA_PAGO	ID_FORMA_PAGO	VARCHAR2	100	-	-	1	-	-	-
	DESCRIPCION	VARCHAR2	100	-	-	-	✓	-	-
	RECARGA	VARCHAR2	100	-	-	-	✓	-	-
									1 - 3

Imagen 2. Tabla Forma de pago.
Fuente: Desafío Latam

Results Explain **Describe** Saved SQL History

Object Type **TABLE** Object **INSCRIPCION**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
INSCRIPCION	ID_INSCRIPCION	NUMBER	22	-	-	1	-	-	-
	NOMBRE	VARCHAR2	100	-	-	-	✓	-	-
	TELEFONO	NUMBER	22	-	-	-	✓	-	-
	ID_CURSO	VARCHAR2	50	-	-	-	✓	-	-
	ID_FORMA_PAGO	VARCHAR2	50	-	-	-	✓	-	-
									1 - 5

Imagen 3. Tabla Inscripción.
Fuente: Desafío Latam

Para interactuar con la base de datos, se facilita el script de creación de las tablas involucradas.

```
    create table curso (id_curso varchar2(50) primary key, descripcion
varchar2(100),precio number(22));
    create table forma_pago (id_forma_pago varchar2(50), descripcion
varchar2(100),recargo varchar2(100));
    create table inscripcion(id_inscripcion number(22) primary key,
nombre varchar2(100),telefono number(22), id_curso varchar2(50),
id_forma_pago varchar2(50),
    foreign key (id_curso)  references curso(id_curso));


    CREATE SEQUENCE inscripcion_sec MINVALUE 1 START WITH 1 INCREMENT BY
1 NOCACHE;

    insert into curso values('1','Java Enterprice Edition', 800.000);
    insert into curso values('2','Java Standar Edition', 600.000);
    insert into curso values('3','JavaScript ECMA 6', 500.000);

    insert into forma_pago values ('1598','Tarjeta Credito','10%' );
```

```
    insert into forma_pago values ('1547','Tarjeta Debito','15%' );
    insert into forma_pago values ('3578','Efectivo','5%' );
```

En base a este pequeño modelo de datos, se generará una estructura de packages en donde las distintas clases interactúan para dar soporte al requerimiento.

Implementación del sistema

Basta de teoría, vamos a empezar a implementar el sistema, ya que es la única forma de entender la arquitectura en capas y el MVC. Cómo será un ejemplo complejo, hay algunas entidades que serán programadas en java puro. Crear un proyecto *Dynamic Web Project* de nombre *Patrones_Diseño_MantenedorCursos*. Este proyecto es incremental, lo que significa que en esta sesión quedará solo con una capa programada, y el resto sigue en la otra sesión así que no pierdan el código. La estructura del proyecto es la siguiente:

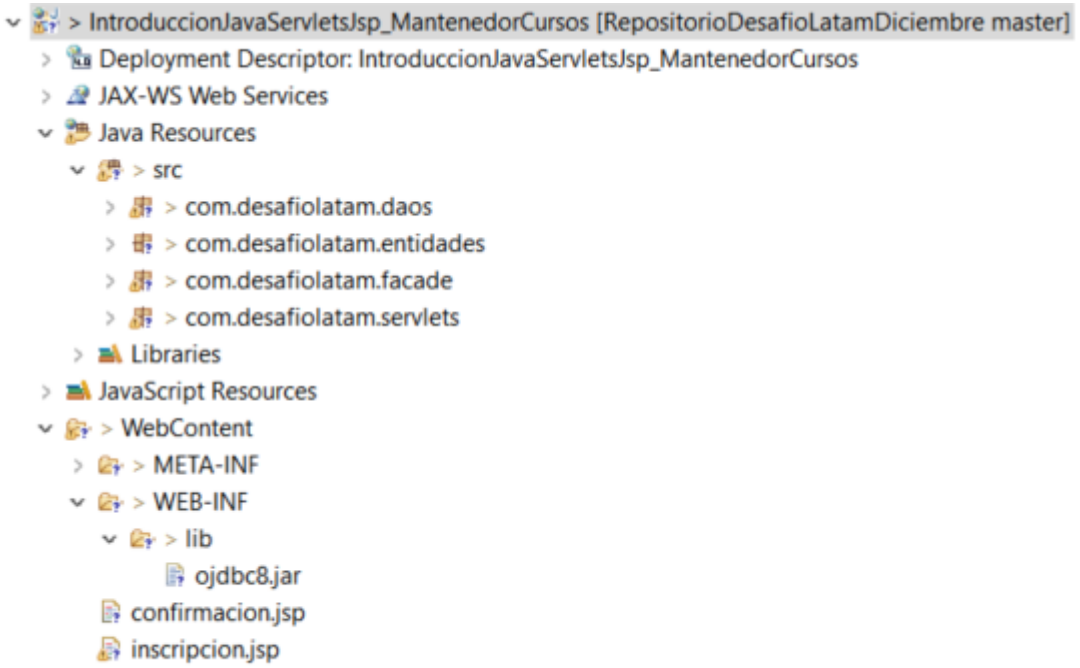

Imagen 4. Estructura de carpetas.
Fuente: Desafío Latam

Cuando se empieza a programar una solución empresarial, recomendamos empezar con las entidades de negocio, ya que son el corazón del sistema. En este caso analizamos el caso y obtuvimos 3 entidades, el curso, la forma de pago y la inscripción.

Si bien utilizaremos lo aprendido hasta acá (páginas jsp, etiquetas html y servlets) debemos programar clases java que mapean las entidades del mundo real de una inscripción. A medida que avancemos en el proceso iremos conociendo nuevos integrantes de una aplicación java JEE, por lo que nos detendremos a explicar qué son y cuál es su función.

Continuando con el ejemplo, decíamos que debemos empezar por las entidades de negocio, las cuales los denominaremos DTO. Es necesario programar un DTO por cada tabla de base de datos (por el momento no usaremos ninguna base de datos, pero aun así se puede mapear las entidades para etapas posteriores). Para una mejor comprensión del sistema a desarrollar se genera el siguiente diagrama de clases con la estructura del sistema:

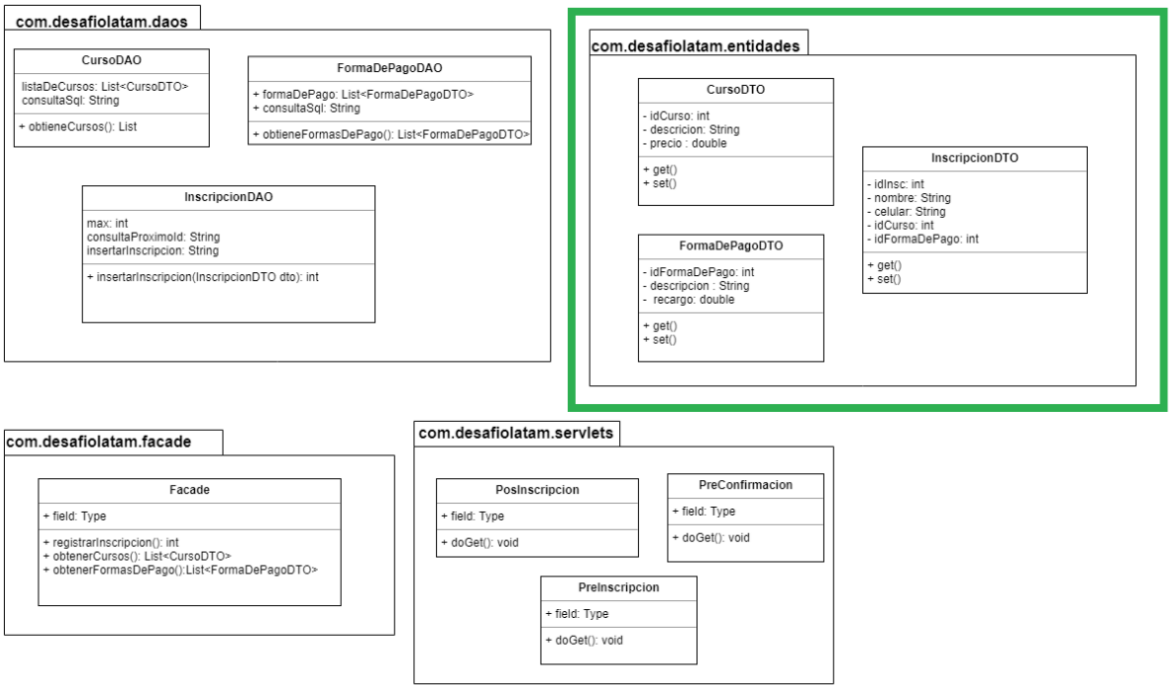

Imagen 5. Diagrama de clases: Entidades.
Fuente: Desafío Latam

Entonces generamos las tres clases de entidades del sistema, *Curso*, *Forma de pago* e *inscripción*. Es importante notar que son clases java puras sin framework ni utilidades especiales, estas clases son conocidas como POJOS.

Clase que representa la forma de pago

```
package com.desafiolatam.entidades;

public class FormaDePagoDTO {

  private int idFormaDePago;

  private String descripcion;

  private String recargo;

  public int getIdFormaDePago() {
        return idFormaDePago;
  }

  public void setIdFormaDePago(int idFormaDePago) {
        this.idFormaDePago = idFormaDePago;
  }

  public String getDescripcion() {
        return descripcion;
  }

  public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
  }

  public String getRecargo() {
        return recargo;
  }

  public void setRecargo(String recargo) {
        this.recargo = recargo;
  }
}
```

Clase que representa un curso

```
package com.desafiolatam.entidades;

public class CursoDTO {

  private int idCurso;

  private String descricion;

  private double precio;

  public int getIdCurso() {
        return idCurso;
  }
  public void setIdCurso(int idCurso) {
        this.idCurso = idCurso;
  }
  public String getDescricion() {
        return descricion;
  }
  public void setDescricion(String descricion) {
        this.descricion = descricion;
  }
  public double getPrecio() {
        return precio;
  }
  public void setPrecio(double precio) {
        this.precio = precio;
  }
}
```

Clase que representa la inscripción

```
package com.desafiolatam.entidades;
public class InscripcionDTO {
  private int idInsc;
  private String nombre;
  private String celular;
  private int idCurso;
  private int idFormaDePago;
  public int getIdInsc() {
        return idInsc;
  }
  public void setIdInsc(int idInsc) {
        this.idInsc = idInsc;
  }
  public String getNombre() {
        return nombre;
  }
  public void setNombre(String nombre) {
        this.nombre = nombre;
  }
  public String getCelular() {
        return celular;
  }
  public void setCelular(String celular) {
        this.celular = celular;
  }
  public int getIdCurso() {
        return idCurso;
  }
  public void setIdCurso(int idCurso) {
        this.idCurso = idCurso;
  }
  public int getIdFormaDePago() {
        return idFormaDePago;
  }
  public void setIdFormaDePago(int idFormaDePago) {
        this.idFormaDePago = idFormaDePago;
  }
}
```

POJOS

Básicamente los POJOS (Plain Old Java Object) es un concepto creado al momento de la aparición de frameworks como Spring que describen a las clases java puras con una estructura estándar en donde se declaran los atributos, los setters y getters correspondientes. Si bien tiene esta estructura no tienen lógica de negocio en ellos y solo se usan para la comunicación y el modelamiento del sistema.

Una característica importante de los POJOS es que no dependen de ningún framework, por lo cual permite que un sistema no esté ligado a nadie, aumentando la portabilidad del código.

Retomando el proyecto de cursos, ya tenemos tres clases POJOS que representan las entidades de nuestro negocio (revisar los 3 últimos códigos). Ahora es tiempo de programar las clases denominadas como DAOS.

La función de una clase DAO es ser el interlocutor entre el sistema y la base de datos. Para entender mejor la función de estas clases, diremos que en estos archivos van las consultas a la base de datos. A continuación se muestra la clase Curso Dado que tiene un método de nombre getCursos cuya única misión es retornar una lista con todos los cursos existentes.

Para poder utilizar la clase DAO es necesario generar una conexión a una base de datos que contendrá las tablas del sistema. Esta conexión la otorga la api jdbc.

API JDBC

Sigla de Java Database Connectivity, es la interfaz de programación estándar que permite que los sistemas java puedan comunicarse y gestionar las bases de datos. La api jdbc consta de un conjunto de interfaces y clases escritas en java que permiten la manipulación y administración de los datos de un sistema de base de datos.

Gracias a esta implementación, los programadores java pueden trabajar con distintas bases de datos, siempre y cuando exista un driver que sea el intermediario entre el sistema y el motor de datos.

En caso de querer más información está la documentación oficial. Sin embargo para realizar una conexión a alguna base de datos es bastante sencilla y necesita de solo algunos métodos de toda la implementación. Este trabajo será manejado utilizando el motor de base de datos Oracle en su versión 11g.

Para trabajar con Oracle se necesita el conector OJDBC6 provisto por la misma empresa. Este conector es un .jar que contiene las clases necesarias para generar una conexión con las bases de datos de oracle.

Si se necesita conectar el sistema a algún otro motor de base de datos como mysql, oracle, postgres, etc. Se necesita su correspondiente conector el cual varía según el producto, pero en lo que se refiere a las clases que permiten la conexión no cambian, ya que es una interfaz de conexión.

La siguiente imagen gráfica muestra la interoperabilidad de bases de datos que se pueden utilizar.

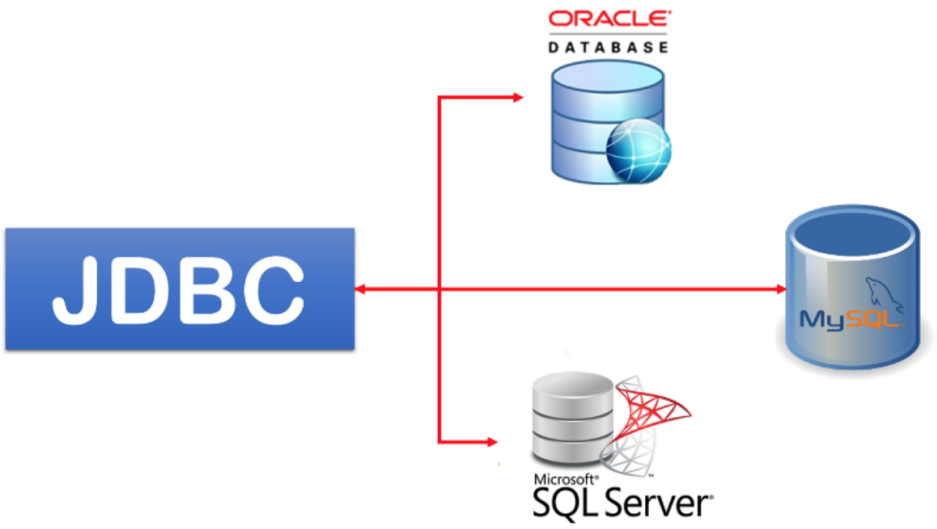

Imagen 6. Con jdbc es posible utilizar cualquier motor de BD.
Fuente: Desafío Latam

Después de la pequeña introducción, exponemos los pasos para generar una conexión.

1. **Descargar OJDBC6:** para descargar el driver nos dirigimos a:

- http://www.java2s.com/Code/Jar/o/Downloadojdbc6jar.htm

2. **Importar .jar al proyecto:** En eclipse importamos el jar al proyecto, de la siguiente forma:

- Pinchar con botón derecho sobre el proyecto y seleccionar la opción BUILD PATH.
- Ubicar la opción Java Build Path y seleccionarla.
- Seleccionar la pestaña LIBRARIES y seleccionar la opción Add External Jar.
- Seleccionar el jar elegido y aceptar. Se verá en la ventana el árbol de directorios con las librerías previamente seleccionadas, al guardar y salir ya se tienen los jars enlazados al proyecto.

3. **Registrar el driver jdbc:** Para poder acceder a la base de datos desde un sistema java es necesario generar una pequeña rutina que registrara el driver jdbc al sistema. Para hacer esto se utilizará el métodoforName() que corresponde a java.lang.class que carga directamente el driver.

   Las forma de utilizar esta variable es:

- **Class.forname('oracle.jdbc.oracleDriver');**

4. **Abrir la conexión:** Una vez que se tiene registrado el driver en el sistema, es tiempo de abrir la conexión con el método estático getConnection() de la clase java.sql.DriverManager. Esta clase retornará a java.sql.Connection.

5. **Quinto paso para establecer la conexión recibida:** En el cuarto paso se abrió una conexión mediante el método getConnection(). Este método retorna una conexión activa, la cual se usa para establecer al fin la conexión a la base de datos que se desee. Para establecer la conexión, se deben definir la url de la base de datos, el usuario de base de datos y la password de la misma. La sentencia para declarar estos parámetros es:getConnection(String URL, String user, String password).

En un caso real la declaración de getConnection sigue la siguiente estructura

```
Connection conn = DriverManager.getConnection
("jdbc:oracle:thin:@localhost:1521:xe", "Empresa_DB", "empresa");
```

El detalle de los parámetros es el siguiente:

- **localhost:1521**: Es la dirección del host en donde está la base de datos. En el caso de probar en nuestro computador la dirección es localhost y el puerto por defecto es el 8080.
- **xe**: Nombre de la base de datos Oracle.
- **local**: corresponde al nombre que se le dio a la instalación de oracle.
- **Empresa_DB**: Nombre del usuario dueño del esquema.
- **empresa**: Password del usuario.

Con estos pasos es posible generar una conexión correcta a una base de datos Oracle. En pasos posteriores se aplicará al sistema de ejemplo que se está desarrollando.

Continuando con el desarrollo del sistema de práctica, se continúa con la clase CursoDao en donde se genera la conexión a la base de datos.