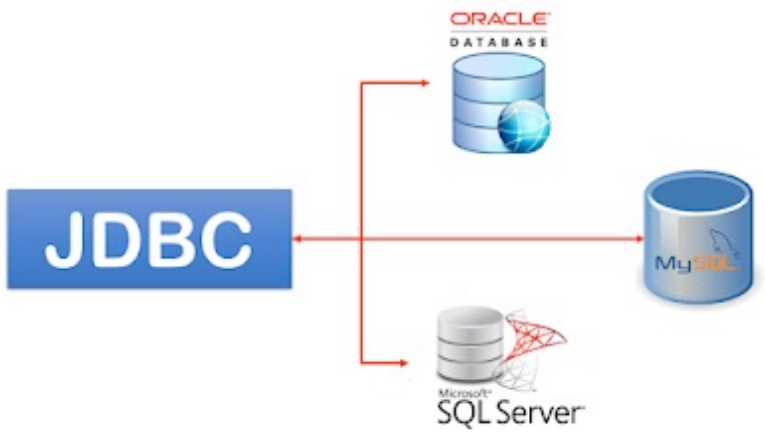
Diseño y Desarrollo de Sistemas de Información

Acceso a Bases de Datos desde Java (JDBC)

Java DataBase Connectivity (JDBC)

- Librería (API) estándar de acceso a base de datos desde el lenguaje de programación Java
- Está incluida en Java SE (*Standard Edition*)
- A día 22 de septiembre de 2023, la versión estable de Java SE incluye la versión 21 de JDK y la versión de 4.3 de JDBC



- Para conectarse a una base de datos específica, es necesario su *driver* JDBC

- Un *driver* se encuentra en un fichero *jar* que se añade a la aplicación como cualquier otra librería

- En el caso de proyectos **Maven**, usaremos el concepto de dependencia para actualizar, de forma sencilla, la versión del driver

- El funcionamiento de un programa que hace uso de JDBC sigue los siguientes pasos:

  - Importar las clases necesarias
  - Cargar el driver JDBC
  - Identificar el origen de los datos (la base de datos)
  - Establecer una conexión con la base de datos
  - Enviar consultas SQL y procesar el resultado
  - Liberar los recursos al terminar

- Los programas deben importar el paquete *java.sql*

```
import java.sql.*
```

Establecer una conexión

- Las BD con las que trabajaremos estarán en servidores Oracle y MariaDB. Las aplicaciones se desarrollarán en el lado del cliente y se comunicarán con los servidores a través de la red

- Un objeto de tipo *Connection* representa una conexión física entre el cliente y el servidor

- Para crear una conexión se usa la clase *DriverManager* y el método *getConnection()* que, en general, tiene como parámetros el nombre del driver, la dirección IP del servidor, el puerto de conexión, el nombre y la contraseña de un usuario de la BD. Los parámetros del método *getConnection()* dependerán del servidor de base de datos

```
Connection conn = DriverManager.getConnection(
"jdbc:oracle:thin:@172.17.20.39:1521:etsi", "DDSI_001","DDSI_001" );
```

```
Connection conn = DriverManager.getConnection(
"jdbc:mariadb://172.18.1.241:3306/DDSI_001", "DDSI_001","DDSI_001" );
```

Establecer una conexión

- En nuestro proyecto crearemos, en la capa **Modelo**, una clase **Conexion**, que tendrá un atributo de tipo **Connection**...

```
private Connection conn = null;
```

- ... y un constructor que recibe los datos de la conexión a la base de datos

```
public Conexion (String sgbd, String ip, String service_bd, String usuario,
      String password)  throws ClassNotFoundException, SQLException { ... }
```

- Con los datos de conexión, se hace la llamada a **DriverManager.getConnection()**

```
conn = DriverManager.getConnection( ... );
```

- A partir de ese momento, ***conn*** es el objeto encargado de interactuar con la base de datos. Habrá que programar un método "get" para devolver ese objeto y poder usarlo en el resto del proyecto

Cerrar una conexión

- Cuando se quiere terminar una conexión, es necesario cerrarla para liberar los recursos que ha usado

- El método para cerrar una conexión es *close()* y lo invoca un objeto de tipo Connection

```
if (conn != null)
        conn.close();
```

¿En qué capa se programa cada parte del proyecto?

- A modo de ejemplo, se muestran fragmentos de algunas de las clases

Aplicación

```
ControladorLogin cLogin = new ControladorLogin();
```

Controlador

```
public class ControladorLogin {

    private Conexion conexion = null;
    private VistaMensajes vMensaje = null;
    private VistaInfoBD vInfoBD = null;

public ControladorLogin() {
        vMensaje = new VistaMensajes();
        vInfoBD = new VistaInfoBD();
...
```

La clase **ControladorLogin** tiene los atributos de las capas **Modelo** y **Vista** que necesite

¿En qué capa se programa cada parte del proyecto?

Controlador

```
private Conexion conectarBD() {
        try {
            String server = "mariadb";
            String ip = "172.18.1.241";
            String bd = "DDSI_999";
            String u = "DDSI_999";
            String p = "DDSI_999";

            conexion = new Conexion(server, ip, bd, u, p);
            vMensaje.mensajeConsola("Conexión Correcta");

        } catch (SQLException sqle) {
            vMensaje.mensajeConsola("Error de conexión ", sqle.getMessage());
        } catch (ClassNotFoundException ex) {
            vMensaje.mensajeConsola("Error indeterminado ", ex.getMessage());
        }
```

Ejemplo de llamadas a métodos de la vista **VistaMensajes**

¿En qué capa se programa cada parte del proyecto?

Vista

```
public class VistaMensajes {

   public void mensajeConsola(String texto) {
      System.out.println ("****************************************");
      System.out.println (texto);
      System.out.println ("****************************************");
   }
```

Ejemplo de método de la clase **VistaMensajes**

Modelo

```
public class Conexion {

   private Connection conn = null;

   public Conexion (String sgbd, String ip, String service_bd, String usuario, String password)
         throws ClassNotFoundException, SQLException {
...
```

Ejemplo de cabecera del constructor de la clase **Conexion**

Gestión de errores

- En el proyecto es necesario controlar los posibles errores y mostrar información detallada cuando se produzcan
- En Java, la captura de errores se realiza mediante bloques *try-catch*

```
try {
// bloque de código que controla los errores
}
catch (ExcepcionTipo1 exOb) {
// gestor de excepciones para Excepción Tipo 1
}
....
catch (ExcepcionTipoN exOb) {
// gestor de excepciones para Excepción Tipo N
}
finally {
// código que se ejecutará siempre
}
```

Gestión de errores

- Ejemplo de captura de excepción *SQLException*

```
catch (SQLException se) {
  String mensaje = ("codigo: " + se.getErrorCode() +
                    " SQL: " + se.getSQLState()+
                    " Texto :" + se.getMessage());
  vMensaje.mensajeConsola("Se ha producido un error: ", mensaje);
}
```

Gestión de errores

- En lugar de capturar las excepciones en el propio método, también se pueden propagar al método de nivel superior

- Para ello, en el método en el que se puede producir la excepción, se indica el/los tipos de excepciones que pueden ser "lanzados", usando la instrucción *throws*

```
public Conexion() throws ClassNotFoundException, SQLException {
    ...
}
```

En nuestro proyecto, las excepciones se capturarán en la capa *Controlador.* En la capa *Modelo*, los métodos "lanzarán" las excepciones para que sean capturadas en la capa *Controlador*