Desarrollo Avanzado de Aplicaciones I

Introducción a JDBC 1

idat

i d a t

inicio desarrollo aplicación término

idat

Inicio

Logro de aprendizaje - Introducción

idat

Logro de Aprendizaje

“Al finalizar la sesión, el participante podrá implementar una conexión a una Base de Datos MySQL y utilizar funciones CRUD a través de sentencias SQL en la Base de Datos.

idat

Introducción

idat

- Revisión Rápida de Temas de Sesión Anterior
- Revisión de Ejercicios de Sesión Anterior
- Inquietudes y/o Preguntas
- Agenda de Sesión
  - Introducción a JDBC
  - La Clase DriverManager
  - La Interface Connection
  - La Interface Statement
  - La Interface ResultSet
  - La Clase SQLException
  - Ejemplos y Ejercicios

Desarrollo

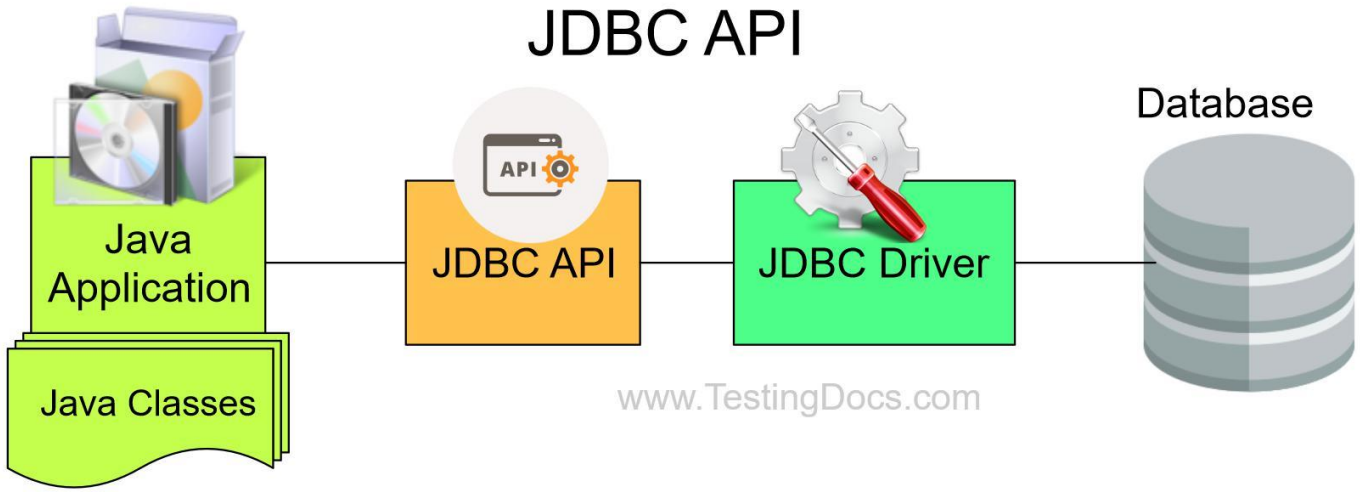
Desarrollo del Contenido de la Sesión

idat

JDBC: ¿Qué es?

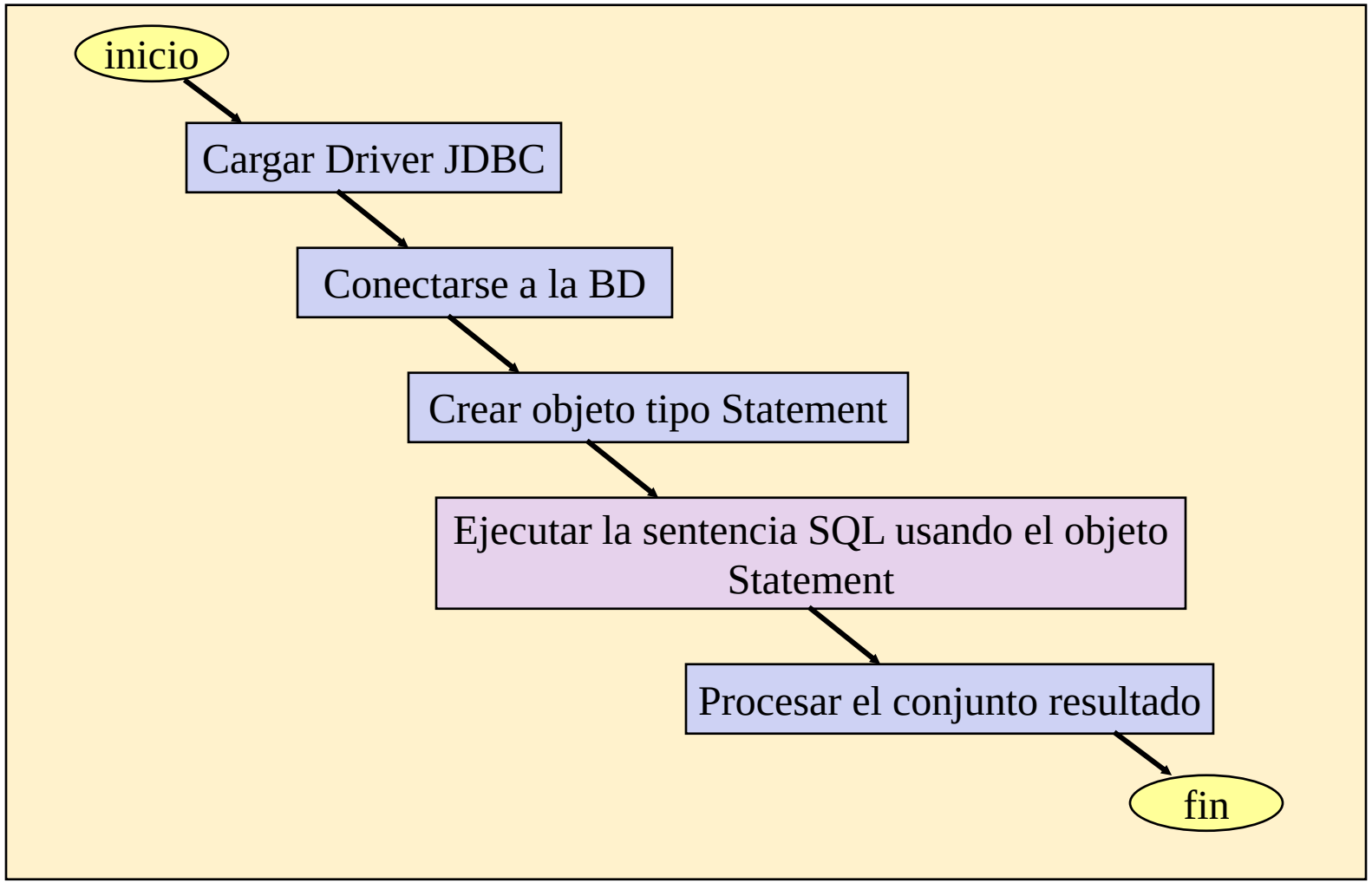
idat

- ✓ JDBC son las siglas en inglés de Java Database Connectivity y es el estándar de conectividad de bases de datos de Java, que permite acceder a diversos gestores de bases de datos en forma transparente.
- ✓ JDBC es una API que tiene una colección de clases e interfaces Java que permite acceder a una base de datos para realizar operaciones SQL. Para acceder a las bases de datos, se debe utilizar un controlador JDBC.



Java

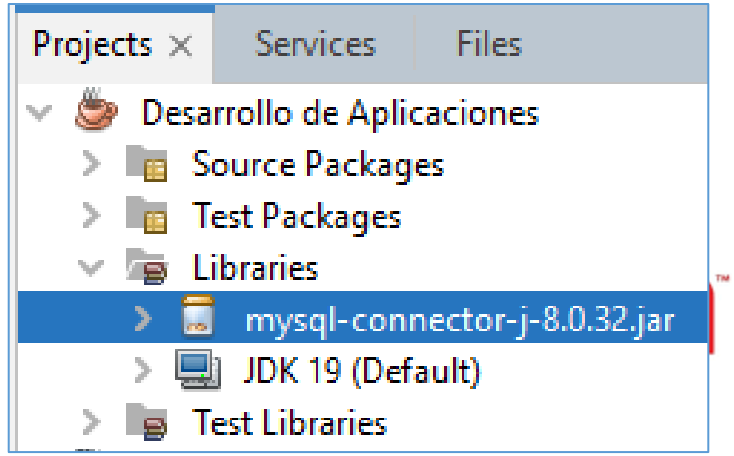
JDBC: Secuencia de Uso



La Clase Driver Manager

idat

- ✓ La clase DriverManager implementa la capa de gestión de JDBC. Guarda la lista de los drivers que están disponibles y establece la conexión entre la base de datos y el driver apropiado.
- ✓ Los drivers pueden ser cargados dinámicamente en tiempo de ejecución, utilizando el método estático "forName" de la clase "Class":
  - ➢ Class.forName("com.mysql.cj.jdbc.Driver");
- ✓ O pueden ser cargados directamente a través de la sección "Libraries" del proyecto.
- ✓ Su método principal es "getConnection", el cual se utiliza para establecer una conexión con la base de datos.



La Clase Driver Manager: Ejemplo

```
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try {
    conn = DriverManager.getConnection("jdbc:mysql://localhost/sakila?" +
            "user=root&password=admin");
    stmt = conn.createStatement();
    rs = stmt.executeQuery( string: "SELECT first_name, last_name FROM actor LIMIT 10");
    while (rs.next()) {
        System.out.println(rs.getString( string: "first_name") + " " +
                rs.getString( string: "last_name"));
    }
}
```

La Interface Connection

- ✓ La Interface “Connection” representa una conexión con una base de datos.
- ✓ Una sesión de conexión incluye la ejecución de sentencias SQL y los resultados que son devueltos después de la conexión.
- ✓ Una aplicación puede tener una o más conexiones con una única base de datos, o puede tener varias conexiones con varias bases de datos diferentes.
- ✓ La sesión de conexión se establece a partir de la llamada al método DriverManager.getConnection, el cual recibe una cadena de caracteres o parámetros con la información de URL, user y password.
- ✓ Este método intenta localizar un driver que pueda conectar con la base de datos representada por la URL, de la lista de drivers registrados.

La Interface Connection (cont.)

- ✓ Ejemplo de conexión:
  - ➢ Connection conn = DriverManager.getConnection ("jdbc:mysql://localhost/sakila?user=root&password=admin");
  - ➢ Donde: jdbc:mysql://localhost/sakila es la URL.
- ➢ Una URL JDBC suministra una forma de identificar una base de datos para que el driver apropiado pueda reconocerla y establecer la conexión con ella.
- ➢ Los proveedores de drivers son los que determinan actualmente que URL JDBC identifica su driver particular. Los desarrolladores solo usan la URL JDBC suministrada con el driver.

La Interface Connection: Ejemplo

idat

```
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try {
    conn = DriverManager.getConnection("jdbc:mysql://localhost/sakila?" +
            "user=root&password=admin");
    stmt = conn.createStatement();
    rs = stmt.executeQuery( string: "SELECT first_name, last_name FROM actor LIMIT 10");
    while (rs.next()) {
        System.out.println(rs.getString( string: "first_name") + " " +
                rs.getString( string: "last_name"));
    }
}
```

La Interface Statement

idat

- ✓ La Interface “Statement” se usa para enviar sentencias SQL a la base de datos. Actúa como contenedor para la ejecución de las sentencias SQL en una conexión dada.
- ✓ Hay tres tipos de interfaces “Statement”:
  - ➢ Statement, se usa para ejecutar una sentencia SQL simple sin parámetros. Suministra métodos básicos para ejecutar sentencias y devolver resultados.
  - ➢ PreparedStatement que hereda de Statement, se usa para ejecutar sentencias SQL precompiladas con o sin parámetros IN. Añade métodos para trabajar con los parámetros IN.
  - ➢ CallableStatement que hereda de PreparedStatement, se usa para ejecutar un procedimiento almacenado de base de datos. Añade métodos para trabajar con parámetros OUT.

La Interface Statement (cont.)

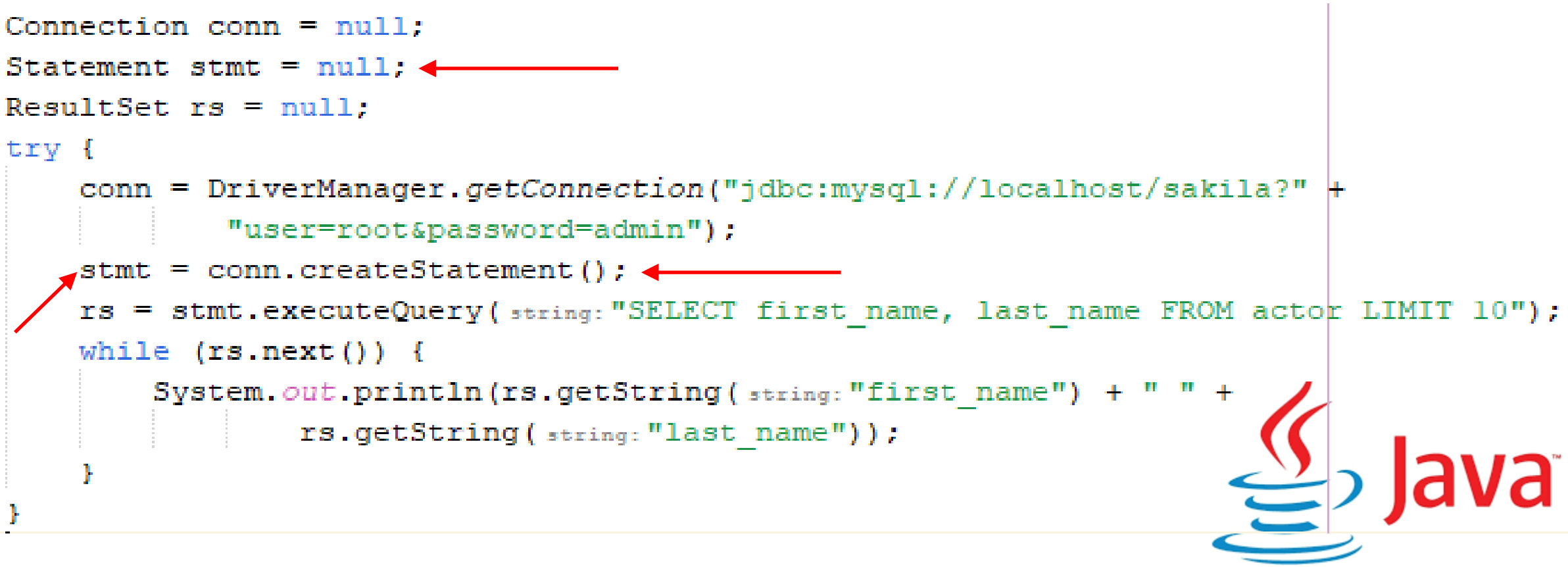
idat

- ✓ Para crear una “Statement” se usa el método “createStatement” de la interface “Connection”.
  - ➢ Statement stmt = conn.createStatement()
  - ➢ Donde: “conn” es una variable “Connection” previamente creada.
- ✓ La interface “Statement” nos suministra tres métodos diferentes para ejecutar sentencias SQL: “executeQuery”, “executeUpdate” y “execute”.
- ✓ El método “executeQuery” esta diseñado para sentencias que producen como resultado un único conjunto de resultados, tal como las sentencias SELECT.
- ✓ El método “executeUpdate” se usa para ejecutar sentencias INSERT, UPDATE ó DELETE, así como sentencias SQL DDL (Data Definition Language) como CREATE TABLE o DROP TABLE.

La Interface Statement (cont.)

- ✓ El valor devuelto de “executeUpdate” es un entero que indica el número de filas que han sido afectadas (referido como “update count”). Para sentencias tales como CREATE TABLE o DROP TABLE, que no operan sobre filas, el valor devuelto por “executeUpdate” es siempre cero.
- ✓ El método “execute” se usa para ejecutar sentencias que devuelven más de un conjunto de resultados o mas de un “update count” o una combinación de ambos.
- ✓ Cuando una conexión está en modo auto-commit, las sentencias ejecutadas son “confirmadas” o “rechazadas” cuando se completan. Una sentencia se considera completa cuando ha sido ejecutada (aplica para el “executeUpdate”) o cuando se han devuelto todos los resultados (aplica para el “executeQuery”).

La Interface Statement: Ejemplo



```
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try {
    conn = DriverManager.getConnection("jdbc:mysql://localhost/sakila?" +
            "user=root&password=admin");
    stmt = conn.createStatement();
    rs = stmt.executeQuery( string: "SELECT first_name, last_name FROM actor LIMIT 10");
    while (rs.next()) {
        System.out.println(rs.getString( string: "first_name") + " " +
                rs.getString( string: "last_name"));
    }
}
```

La Interface ResultSet

- ✓ La sentencia SQL que será enviada a la base de datos como argumento de uno de los métodos de ejecución de la interface “Statement” devuelve un conjunto de resultados que los recibe un “ResultSet”:
  - ➢ ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Tabla1");
  - ➢ Donde: stmt es la variable “Statement” previamente creada.
- ✓ Un “ResultSet” contiene todas las filas que satisfacen las condiciones de una sentencia SQL y proporciona el acceso a los datos de estas filas mediante un conjunto de métodos “getTIPODATO” que permiten el acceso a las diferentes columnas de la filas.
- ✓ El método ResultSet.next se usa para moverse a la siguiente fila del conjunto de resultados, convirtiendo a ésta en la fila actual. El método devuelve verdadero mientras hayan mas filas por recorrer.

La Interface ResultSet (cont.)

- Los métodos "getTIPODATO" suministran los medios para recuperar los valores de las columnas de la fila actual. Puede usarse o bien el nombre de la columna o el número de la columna.
- Para los métodos "getTIPODATO", el driver JDBC intenta convertir los datos subyacentes a tipos de datos Java.
- Todos los métodos que ejecutan sentencias SQL cierran los objetos "ResultSet" previamente abiertos como resultado de las llamadas a "Statement".

TIPO DE DATO SQL	TIPO DE DATO EN JAVA
BIT	boolean
SMALLINT	short
INTEGER	int
FLOAT	double
BIGINT	long
REAL	float
DOUBLE	double
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
CHAR	java.lang.String
VARCHAR	java.lang.String
LONGVARCHAR	java.lang.String
DATE	java.sql.Date
TIM	java.sql.Time
TIMESTAMP	java.sql.Timestamp
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]

La Interface ResultSet : Ejemplo

idat

```
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try {
    conn = DriverManager.getConnection("jdbc:mysql://localhost/sakila?" +
            "user=root&password=admin");
    stmt = conn.createStatement();
    rs = stmt.executeQuery( string: "SELECT first_name, last_name FROM actor LIMIT 10");
    while (rs.next()) {
        System.out.println(rs.getString( string: "first_name") + " " +
            rs.getString( string: "last_name"));
    }
}
```

La Clase SQLException

- ✓ SQLException es una ampliación de java.lang.Exception y proporciona información adicional relacionada con las anomalías que se producen en un contexto de acceso a base de datos con el API JDBC.
- ✓ La Clase "SQLException" tiene los siguientes métodos:
  - ➢ getMessage(), devuelve la descripción del mensaje de error.
  - ➢ getSQLState(), devuelve un código SQL estándar definido por ISO/ANSI y el Open Group que identifica de forma unívoca el error.
  - ➢ getErrorCode(), es un código de error que lanza la base de datos. En este caso el código de error es diferente dependiendo del proveedor de base de datos que estemos utilizando.
  - ➢ getCause(), devuelve una lista de objetos que han provocado el error.
  - ➢ getNextException(), devuelve la cadena de excepciones que se ha producido, para navegar sobre ella y ver el detalle de esas excepciones.

La Clase SQLException: Ejemplo

idat

```
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try {
    conn = DriverManager.getConnection("jdbc:mysql://localhost/sakila?" +
            "user=root&password=admin");
    stmt = conn.createStatement();
    rs = stmt.executeQuery( string: "SELECT first_name, last_name FROM actor LIMIT 10");
    while (rs.next()) {
        System.out.println(rs.getString( string: "first_name") + " " +
                rs.getString( string: "last_name"));
    }
}
catch (SQLException ex) {
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}
```

Importar las Clases e Interfaces

- ✓ Para la utilización de las clases e interfaces de JDBC, se debe realizar los "import" respectivos:

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

Aplicación

Revisar ejemplos y realizar ejercicios prácticos

idat

Término

Indicaciones generales y/o Resumen de Sesión

idat

Resumen de Sesión

idat

- Introducción a JDBC
- La Clase DriverManager
- La Interface Connection
- La Interface Statement
- La Interface ResultSet
- La Clase SQLException
- Ejemplos y Ejercicios

GRACIAS