Universidad Tecnológica de Pereira

Mision TIC 2022

El futuro digital es de todos

MinTIC

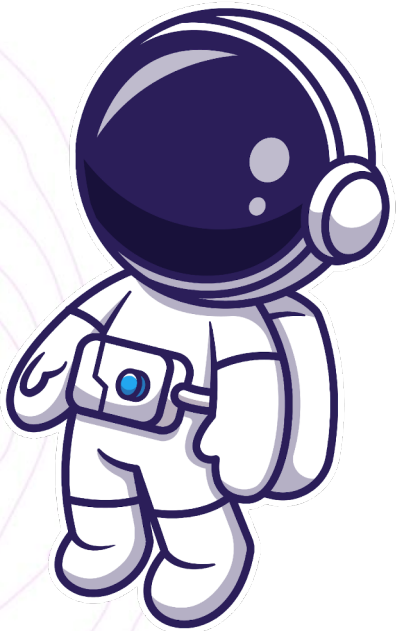
Unidad 4

11 - JDBC

Hechos QUE CONECTAN

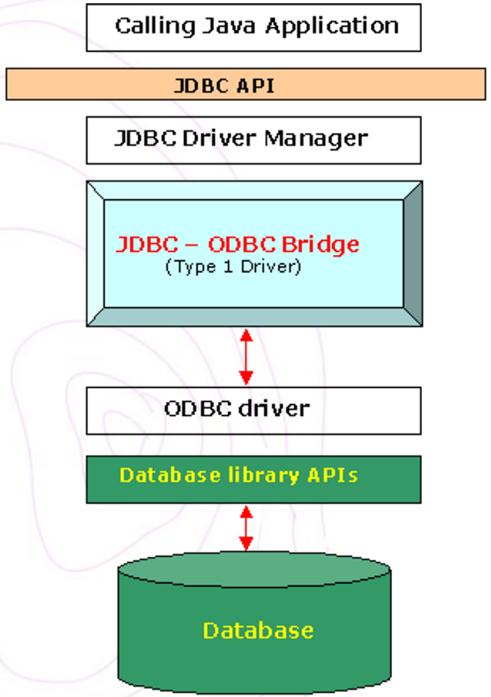
Persistencia en Java

- Se llama “persistencia” de los objetos a su capacidad para guardarse y recuperarse desde un medio de almacenamiento.
  - Archivos
  - Bases de Datos Relacionales
    - **JDBC - Java DataBase Connectivity**
    - JPA - Java Persistence API



JDBC - Java Database Connectivity

Es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.



API JDBC

Clase	Descripción
`java.sql.Connection`	Representa una conexión a la base de datos. Abstrae el detalle de cómo comunicarse con el servidor de base de datos.
`java.sql.DriverManager`	Administra los drivers JDBC usados por la aplicación. En conjunto con el URL y la autenticación apropiada, puede proveer aplicaciones con objetos Connection válidos.
`javax.sql.DataSource`	Abstrae los detalles (URL, autenticación) de cómo obtener una conexión a la base de datos. Es el método preferido para obtener objetos Connection.
`java.sql.Statement`	Provee métodos para que los desarrolladores puedan ejecutar sentencias SQL.
`java.sql.ResultSet`	Representa los resultados de una sentencia SQL. Estos objetos son retornados usualmente por métodos del objeto Statement.

java.sql.DriverManager

Permite al desarrollador conseguir un objeto Connection la cual se puede usar para ejecutar actividades en la base de datos.

Para establecer la conexión a la base de datos, se debe usar el método `getConnection()` dándole el URL JDBC, también con el usuario y contraseña para acceder a la base de datos si aplica.

El URL debe seguir la sintaxis requerida por la implementación particular de la base de datos.

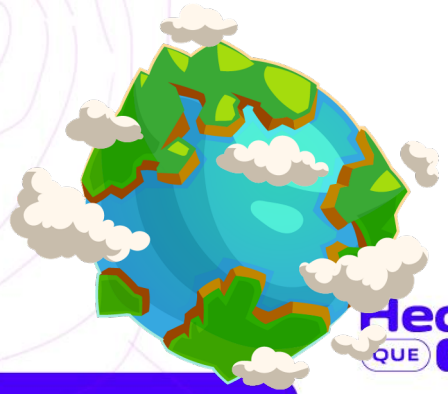
```
String url = "jdbc:<SGBD>:<database>";
String user = "username";
String password = "password";
Connection conn = null;

try {
  conn = DriverManager.getConnection(url, user, password);
  ...
} catch (SQLException e) {
  // Manejo de los errores
}
```

Cadenas de Conexión JDBC

DBMS	URL
IBM DB2	`jdbc:db2://<HOST>:<PORT>/<DB>`
MySQL	`jdbc:mysql://<HOST>:<PORT>/<DB>`
Postgresql	`jdbc:postgresql://<HOST>:<PORT>/<DB>`
Ms SQL Server	`jdbc:sqlserver://<HOST>\<DB>:<PORT>`
Oracle	`jdbc:oracle:thin:@<HOST>:<PORT>:<SID>`
JDBC-ODBC Bridge	`jdbc:odbc:<DB>`
SQLite	`jdbc:sqlite:<FILE_PATH>` `jdbc:sqlite::memory:`

SQLite Database URL

Tipo	Formato URL
Incrustada en la aplicación (Embebida)	**jdbc:sqlite:<file_path>** jdbc:sqlite:sample.db jdbc:sqlite:C:/sqlite/db/sample.db
En memoria	**jdbc:sqlite::memory:**



Conectando JDBC a SQLITE

pom.xml

```
...
<dependencies>
  ...
  <dependency>
    <groupId>org.xerial</groupId>
    <artifactId>sqlite-jdbc</artifactId>
    <version>3.36.0.1</version>
  </dependency>
  ...
</dependencies>
...
```

```
String url = "jdbc:sqlite:sample.db";
Connection conn = null;
...
try {
  conn = DriverManager.getConnection(url);
  ...
} catch (SQLException e) {
  // Manejo de los errores
}
```

java.sql.Connection / java.sql.Statement

Los objetos `java.sql.Connection` representa una conexión actual a la base de datos. Una vez tenemos la instancia de este objeto, podemos crear una instancia de `java.sql.Statement`, el cual nos permitirá ejecutar sentencias SQL.

Los objetos Statement provee varios métodos para ejecutar SQL. Los dos más usados son:

- **executeQuery**: Toma una sentencia SELECT y retorna el resultado de la operación como un objeto ResultSet.
- **executeUpdate**: Toma una sentencia INSERT, UPDATE, o DELETE y retorna el número de filas afectadas por la operación.

Creando una sentencia

```
String url = "jdbc:sqlite:sample.db";
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try {
  conn = DriverManager.getConnection(url);
  stmt = conn.createStatement();
  ...
} catch (SQLException e) {
  // Manejo de los errores
}
```

java.sql.ResultSet

Encapsula los resultados de una consulta a la base de datos.

Los datos dentro de un objeto ResultSet pueden ser mejor visualizados como una tabla. La información puede ser recuperada de a una fila a la vez, y el objeto ResultSet mantendrá el control de la fila actual.

Para iterar por las filas, usamos el método **next()**, el cual mueve el apuntador de fila a la siguiente. Este método retorna true si existe una siguiente fila.

```
while (rs.next()) {
    //Lee los datos de la fila actual
}
```

Para obtener los datos de cada fila, el objeto ResultSet no da varios métodos get los cuales toman como parámetro el número de columna o el nombre de la columna.

- El método getString() es para obtener un dato String
- El método getInt() es para obtener un dato int
- El método getBoolean() es para obtener un dato boolean.

Es recomendado, sin embargo, usar los nombres de los campos en lugar de la posición en la fila.

Recuperando datos de una consulta

```
...
ResultSet rs = null;
try {
  conn = DriverManager.getConnection(url);
  stmt = conn.createStatement();
  rs = stmt.executeQuery("SELECT * FROM employees");
  while (rs.next()) {
    int id = rs.getInt("employee_id");
    String nombre = rs.getString("first_name");
    String apellido = rs.getString("last_name");
    // Hacer algo con los datos leídos
  }
  rs.close();
  ...
} catch (SQLException e) {
  // Manejo de los errores
}
```

Liberando recursos

Es un paso muy importante para evitar bloqueos después que la operación se haya completado. Debe ser hecho de forma explícita y es responsabilidad del programador.

Sin esta liberación, los recursos tomados podrían no se usados en el futuro. Para aplicaciones grandes, podría hacer que pierda rápidamente las conexiones disponibles.

Los recursos se liberan con el método close() disponible en objetos Connection, Statement, y ResultSet.

Se debe cerrar primero el objeto ResultSet, luego el Statement, y finalmente el objeto Connection.

Recuerda que también puedes usar el `try-with-resources`

```
...
try {
  conn = DriverManager.getConnection(url);
  ...
} catch (SQLException se) {
  System.err.println("Un error mientras intenta consulta");
} finally {
  try {
    if (rs != null){
      rs.close();
    }
    if (stmt != null) {
      stmt.close();
    }
    if (conn != null){
      conn.close();
    }
  } catch (SQLException e) {
  }
}
```

java.sql.PreparedStatement

Subclase de java.sql.Statement, la cual va a contener una consulta pre-compilada y puede estar esperando algún(os) parámetro(s).

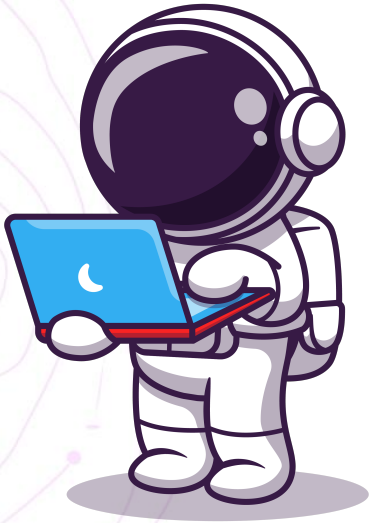
```
String sql = "INSERT INTO GRADES (GRADE, MIN_SALARY, MAX_SALARY) VALUES ( ? , ? , ? )";

PreparedStatement pstmt = conn.prepareStatement(sql);
```

Para darle los valores a los parámetros que se definen en esta clase, usamos los métodos setXXX(), donde XXX es el tipo de dato que va a asignar. **Los parámetros empiezan a contar desde 1**.

```
pstmt.setString(1, "G");
pstmt.setLong(2, 1);
pstmt.setLong(3, 99999);
pstmt.executeUpdate();
```

Vamos al código

- Crear un proyecto Maven para los ejercicios de la clase 11
  - Ctrl + Shift + P
  - *> Java: Create Java Project...*
  - *Maven*
  - *maven-archetype-quickstart*, *<versión más reciente>*
  - **group Id**: co.edu.utp.misiontic2022.c2
  - **artefact Id**: clase11-bookshop
- Realizar las acciones propuestas en el ejercicio



Descripción del Ejercicio

Este ejercicio se basa en las tablas para libros, existencias y operaciones de venta. El objetivo es desarrollar un programa de línea de comandos que sea capaz de realizar las siguientes operaciones:

- Imprimir en salida estándar los datos de un libro dado su ISBN.
- Imprimir en salida estándar los datos de todos los libros.
- Consultar el número de unidades en stock de un libro, dado su ISBN.
- Realizar una transacción de venta de un libro, dados su ISBN y el número de unidades a vender.

Se proporciona un esqueleto de la aplicación, con las siguientes clases

- DBManager
- BookShop
- Book

Descripción del Ejercicio

La clase BookShop está ya programada, y contiene diferentes métodos para la gestión en consola que permite ejecutar las operaciones.

Estos métodos invocan a los métodos de la clase DBManager que se deben programar en este ejercicio.

La clase Book también está programada, y se usa para encapsular los datos acerca de un libro.

Necesitarás configurar el driver de JDBC de SQLIte para que la máquina virtual sea capaz de encontrar este driver cuando intentes conectarte a la base de datos.

Ejercicio

1. Crea una nueva base de datos llamada **BookShop.db**
2. Crea las tablas que necesites para representar los libros en el catálogo de una librería.
3. Para cada libro, almacena el título, ISBN y año de publicación.
4. Crea una tabla aparte que almacene las unidades que tiene la librería en existencias de cada libro
5. Crea otra tabla en que se almacenen las operaciones de venta (fecha y hora, libro vendido, y número de unidades vendidas).

Ejercicio

6. Implementa el método **connect()** de la clase **DBManager** para que establezca una conexión con la base de datos y guarde dicha conexión en el atributo connection.
7. Implementa el método **searchBook** de la clase DBManager para que devuelva el libro cuyo ISBN se le indique.
8. Implementa el método **listBooks** de la clase DBManager para que devuelva un objeto List con los datos de todos los libros de la tabla de libros.

Ejercicio

9. Implementa el método **getStock(int id)** de la clase DBManager para que devuelva las existencias del libro cuyo identificador reciba.
10. Implementa el método **sellBook(int id, int units)** de la clase DBManager para que ejecute como transacción los pasos necesarios para vender un libro: anotación de la venta, comprobación de existencias y actualización de existencias. Haz lo que sea necesario para que las transacciones funcionen de forma consistente.

java.sql.CallableStatement

Subclase de java.sql.PreparedStatement, la cual va a ayudar a hacer las llamadas a funciones y procedimientos almacenados de la base de datos.

Para llamar un procedimiento almacenado, debemos crear el CallableStatement de la siguiente manera

```
var cstmt = conn.prepareCall("{ call myProcedure(?) }");
```

Al igual que el PreparedStatement, usamos los métodos setXXX() para asignar los parámetros. Los parámetros empiezan a contar desde 1.

```
cstmt.setString(1, "Hola");
cstmt.executeUpdate();
```

Si lo que queremos es hacer el llamado a una función almacenada que retorna un valor, entonces lo creamos:

```
cstmt = conn.prepareCall("{ ? = call myFunction(?) }");
```

Y debemos registrar qué parámetros serán de salida usando el método **registerOutParameter()**, antes de asignar valores a los parámetros y ejecutar el llamado.

```
cstmt.registerOutParameter(1, Type.INTEGER);
cstmt.setString(2, "Hola");

cstmt.excecuteUpdate();
```

Y para obtener el valor de los parámetros de salida, usamos el método getXXX()

```
int result = cstmt.getInt(1);
```

Para la próxima sesión...

- Terminar los ejercicios que no se terminaron… (si aplica)
- Revisar el funcionamiento de la aplicación que se encuenta en https://github.com/cesardiaz-utp/MisionTIC2022-Ciclo2-Unidad4-JDBC
- Ver videos: (Material complementario)
  - Curso JDBC