Introduction to JDBC

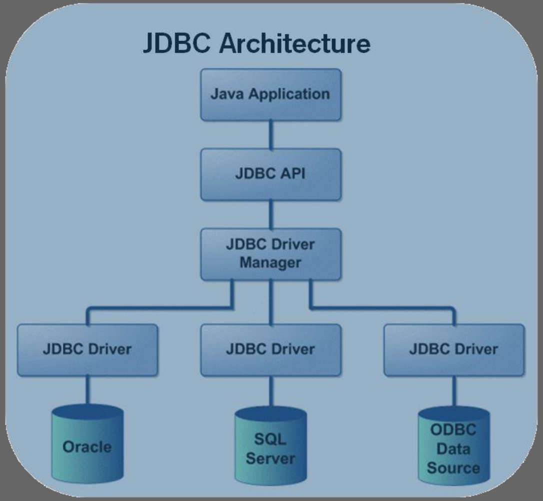
Professor:

Bladimir Bacca Cortes Ph.D.

Perception and Intelligent Systems Research Group

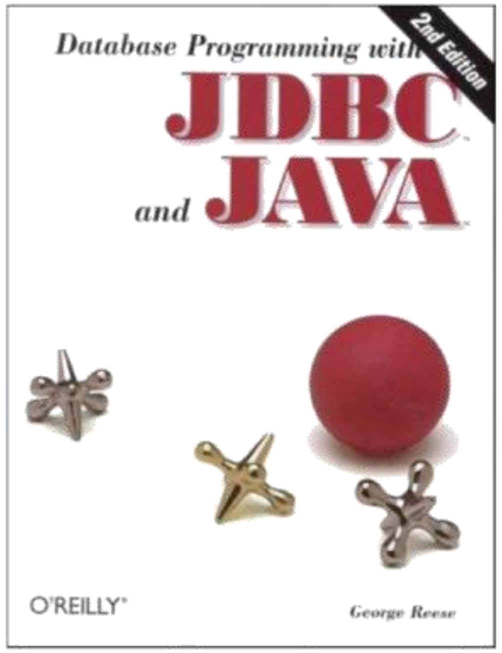
Email:

Bladimir.bacca@correounivalle.edu.co



Contents

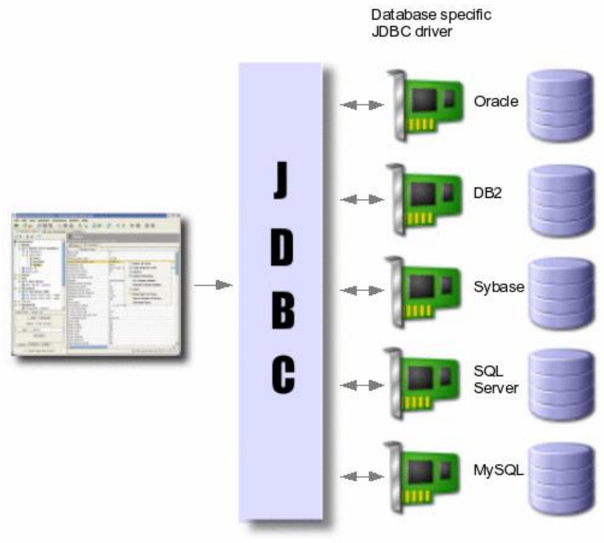
- Introduction to JDBC and its architecture
- Database drivers
  - Types
  - Framework
- Class hierarchy
- SQL data types and its relationship with Java data types.
- Connection and program basic steps
- Basic operations:
  - Data retrieving (SELECT)
  - Data updating (UDATE)
- Advanced concepts on JDBC
  - Basic transactions
  - Prepared SQL statements
  - Batch processing
  - Updateable result sets

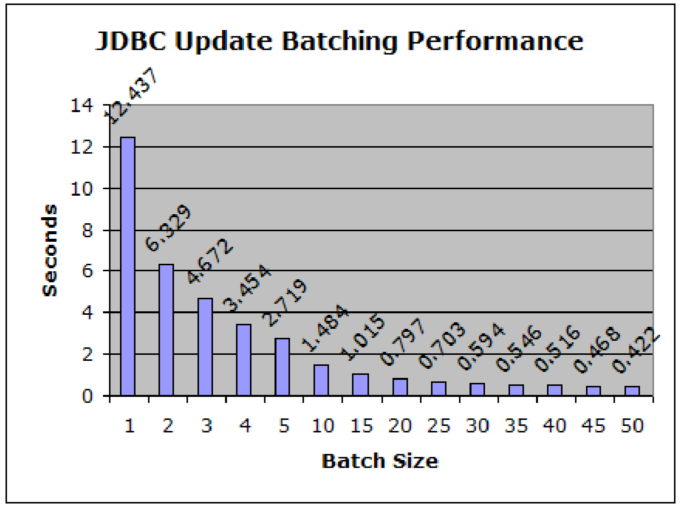


Team[oR] 2001
[x] java

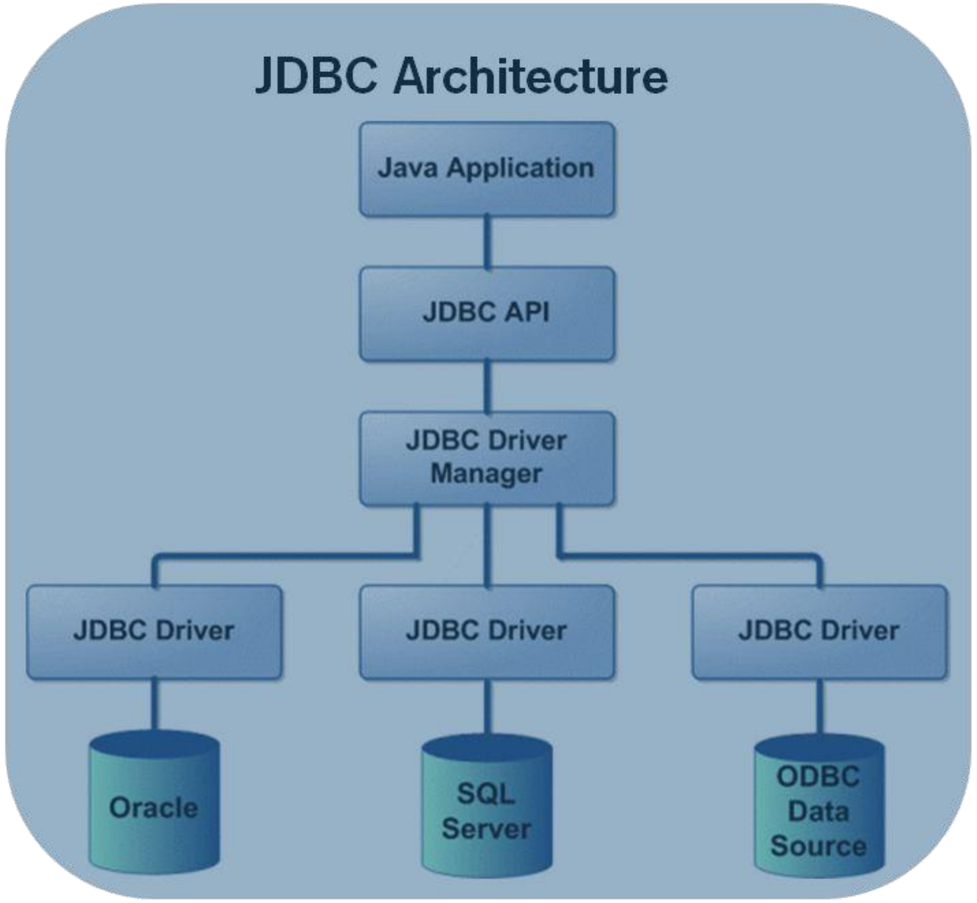
Introduction to JDBC and its Architecture

- **Requirements**
  - Database engine compatible with SQL2
  - Connectivity driver
  - Java 1.4.2 and up.
- **Properties**
  - It allows to handle **SQL sentences** through **JDBC classes**
  - **Transactions** support (Commit / Rollback)
  - **Batch** support
  - **On-line modification** of retrieved sentences.
  - Database **independence**
  - One-to-one relationship between Java **objects** and **relational data**.



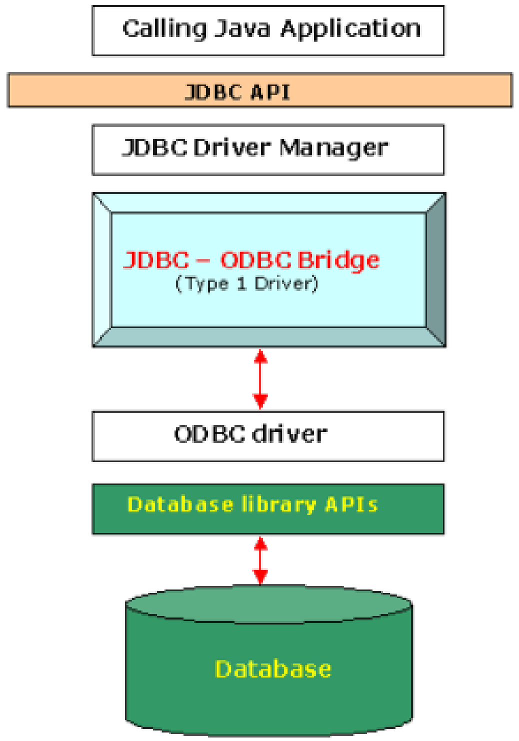


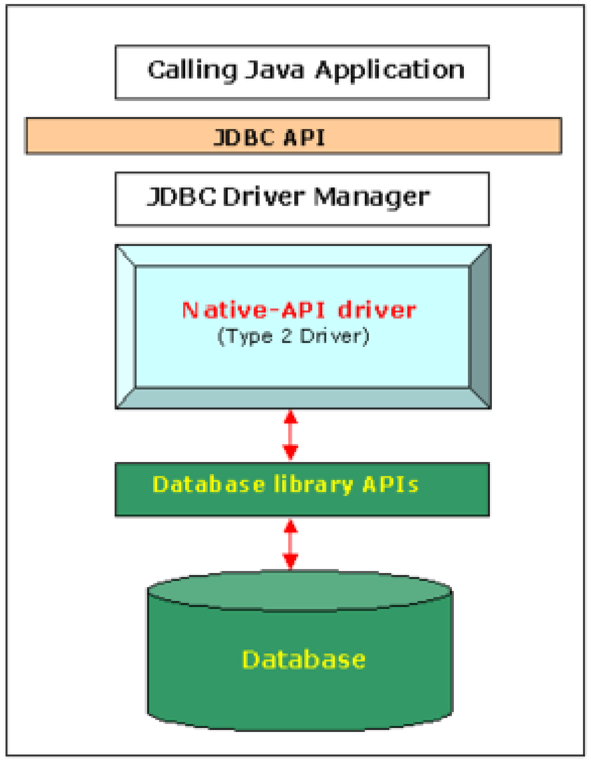
Introduction to JDBC and its Architecture



Database Drivers

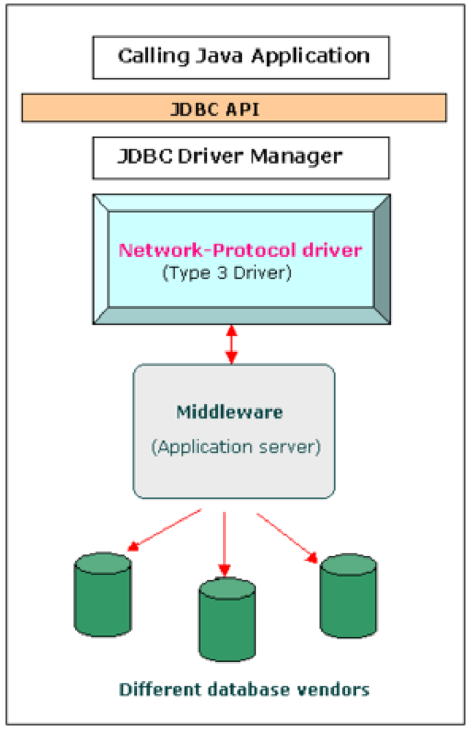
- **Note**: Each database engine needs a specific JDBC driver.
- **Types of drivers**:
  - **Type 1**: They are basically **connection bridges** between different database technologies, e.g. JDBC – ODBC. Additional software is needed in client side.
  - **Type 2**: They are **native code based drivers** (C, C++), it is called from Java but it is not platform independent.

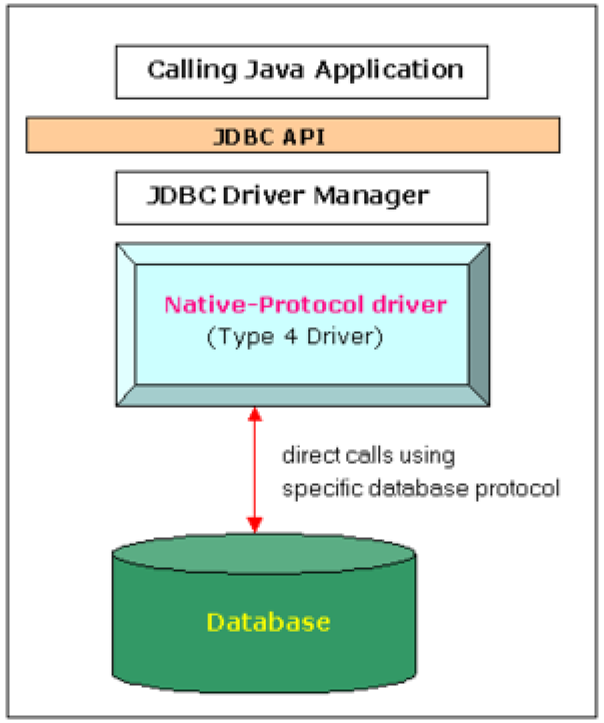




Database Drivers

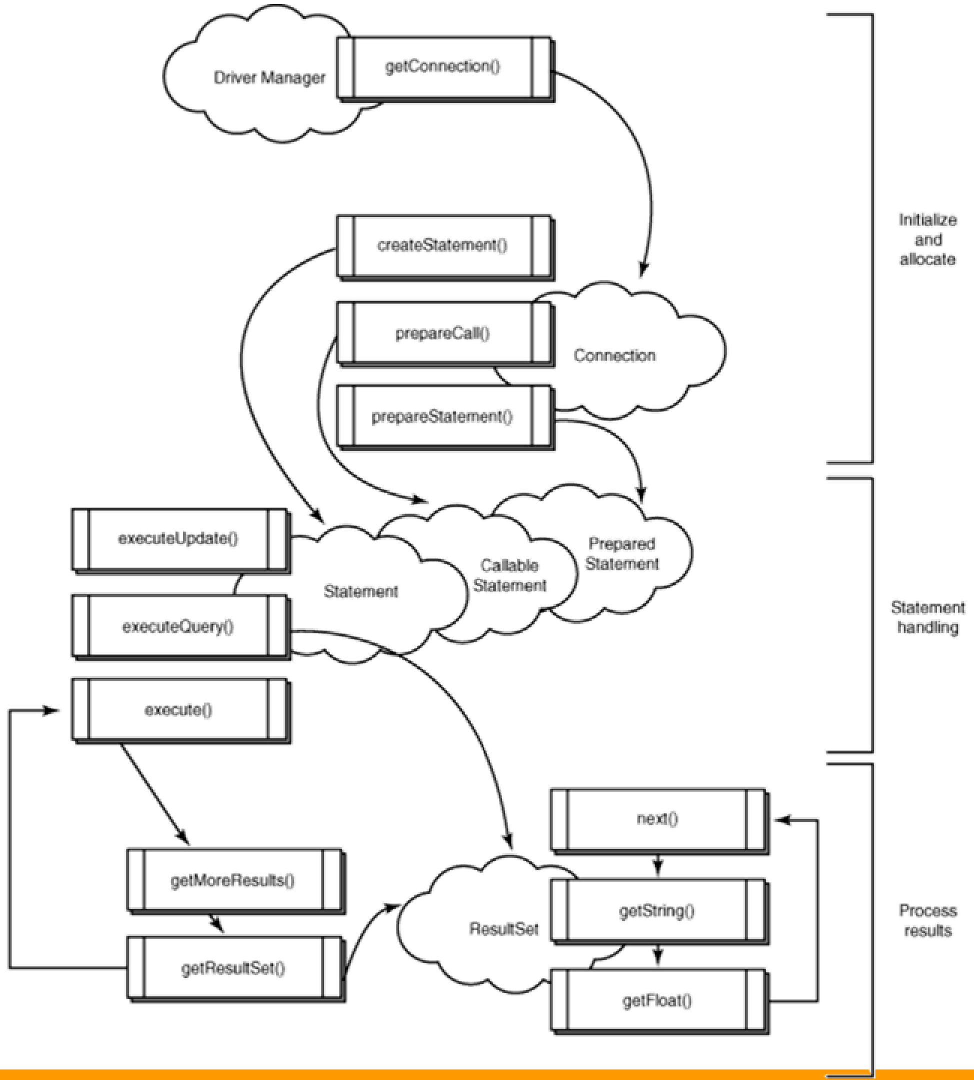
- **Note**: Each database engine needs a specific JDBC driver.
- **Types of drivers**:
  - **Type 3**: They use **basic socket connections**, but without implementing communication protocols which depend on database engines.
  - **Type 4**: They are most used, since they implement the **database engine communication** protocols. This drivers use TCP/IP links.





JDBC Class Hierarchy

- **Main classes** to connect and operate databases using JDBC:
  - **Java.sql.Driver** – It instantiates the JDBC driver to offer an entry point to it.
  - **Java.sql.DriverManager** – It holds and manages the database engine connection.
  - **Java.sql.Connection** – It works as file-handle in file systems offering an entry point to any SQL request.
  - **Java.sql.Statement** – It creates a SQL statement to be sent to the database engine.
  - **Java.sql.ResultSet** – It is an object related with the relational data obtained from the database engine.
  - **Java.sql.SQLException** – Exception that involves any SQL error in the SQL statement sent to the database engine.

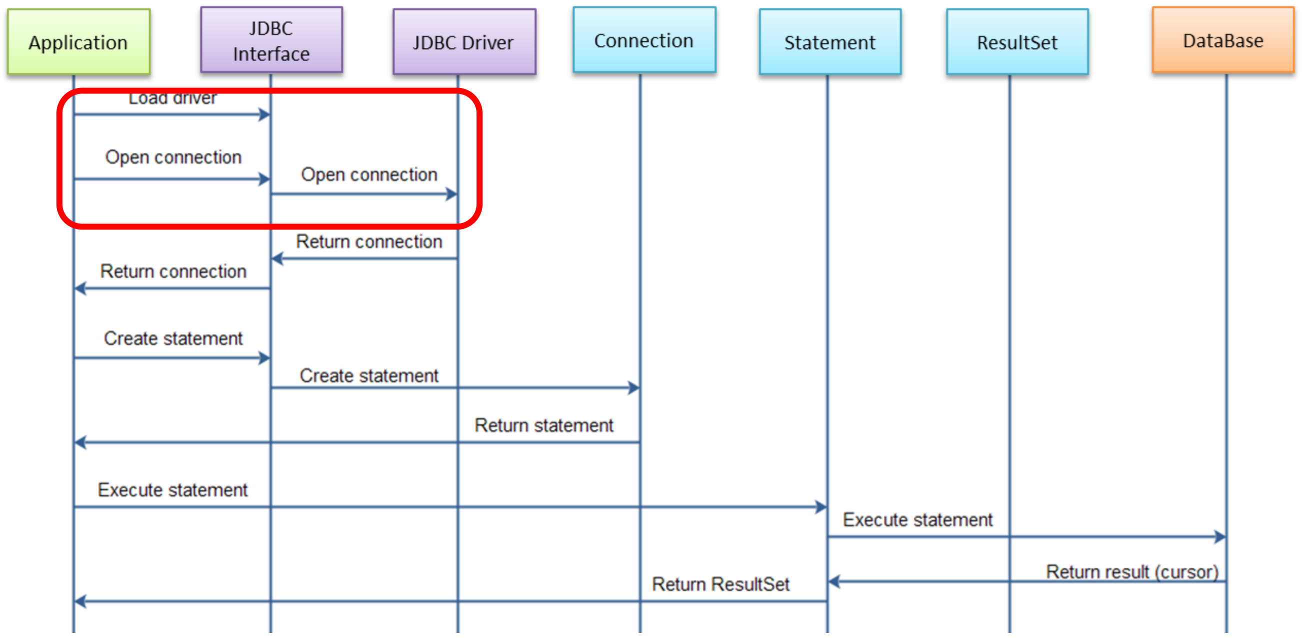


SQL Data Types and its Relationship with Java Data Types

JDBC Data Type	Java Data Type	JDBC Data Type	Java Data Type
CHAR	String	**BINARY**	byte[]
VARCHAR	String	**VARBINARY**	byte[]
LONGVARCHAR	String	**LONGVARBINARY**	byte[]
NUMERIC	java.math.BigDecimal	**DATE**	java.sql.Date
DECIMAL	java.math.BigDecimal	**TIME**	java.sql.Time
BIT	boolean	**TIMESTAMP**	java.sql.Timestamp
TINYINT	byte	**CLOB**	Clob
SMALLINT	short	**BLOB**	Blob
INTEGER	int	**ARRAY**	Array
BIGINT	long	**DISTINCT**	mapping of underlying type
REAL	float	**STRUCT**	Struct
FLOAT	double	**REF**	Ref
DOUBLE	double	**JAVA_OBJECT**	underlying Java class

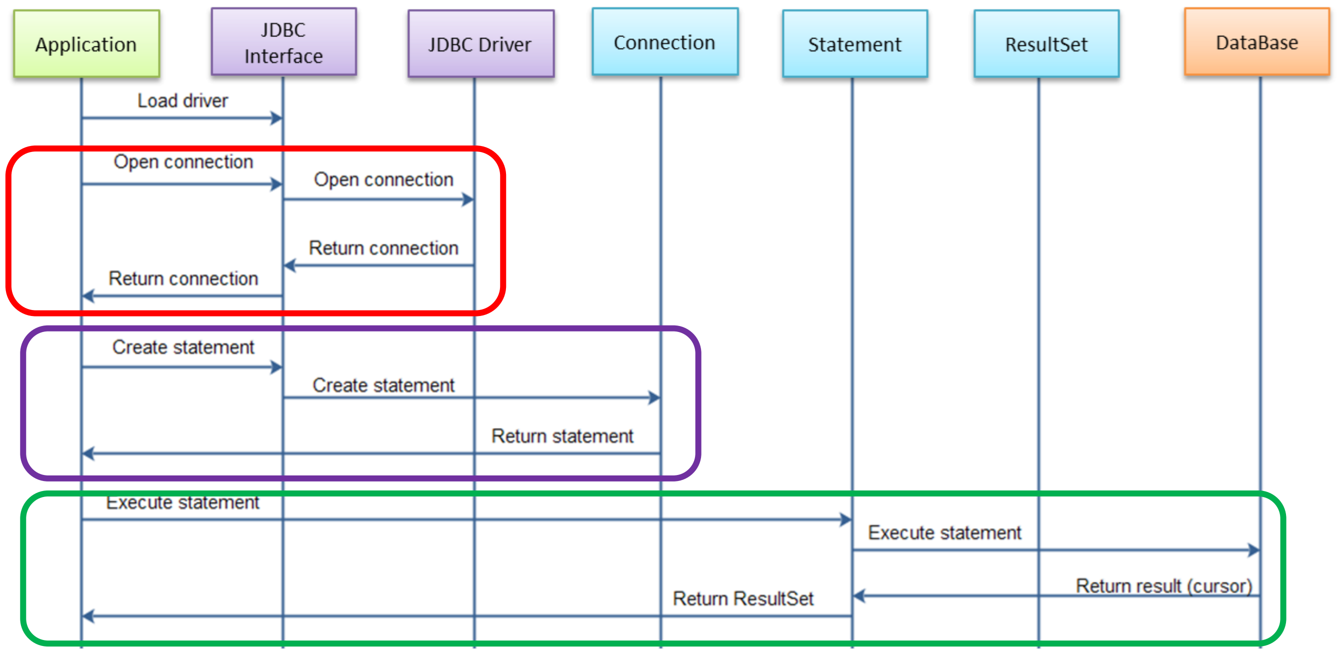
Connection and Program Basic Steps

- **Basic declarations**
  - *String url = “jdbc:mysql://libertad.univalle.edu.co/intDB”;*
  - **Format**: jdbc:sub-protocol:sub-name
    - **Sub-protocol** is used to identify the JDBC driver to use.
    - **Sub-name** is used to define where the database is placed and what is its name.
  - *String driver = "com.mysql.jdbc.Driver";*
  - **Installation**:
    - Download a type 4 driver depending on the database engine.
    - Un-compress it.
    - Copy the JAR file to: JAVA_HOME/jre/lib/ext
    - Updating the CLASSPATH environment variable.



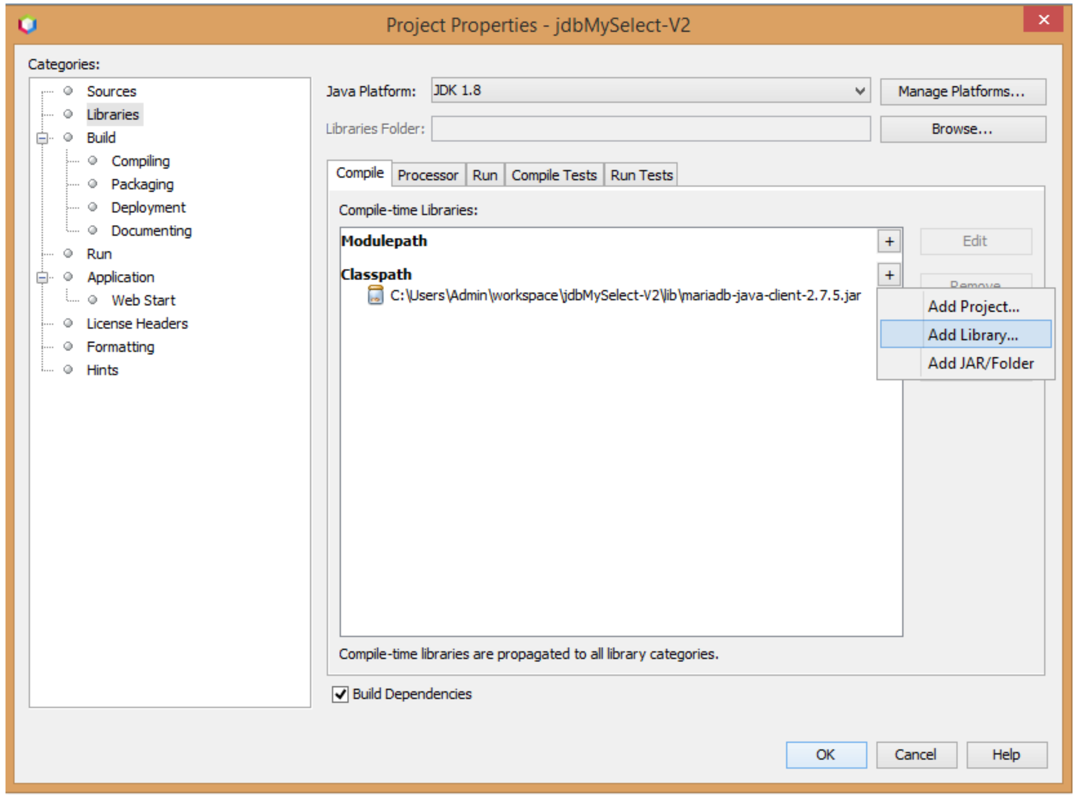
Connection and Program Basic Steps

- **Getting an instance of JDBC driver**
  - *Class.forName(driver).newInstance();*
- **Getting a connection from DriverManager**
  - *conn = DriverManager.getConnection(urlConexion, "evlady", "sol123");*
  - It provides: The URL connection, The database user and its password.
- **Creating a SQL statement**: *Statement stm = conn.createStatement();*
- **Getting results**: *ResultSet rs = stm.executeQuery("select COUNT(id) from ir_captura_disp");*



Adding JDBC Driver to Eclipse

- **Procedure to add the JDBC driver to your Eclipse Project**:
  - Create a **Library** folder into your eclipse project.
  - Copy the JDBC driver JAR into this Library folder.
  - Go to your Netbeans project properties.
  - Select the Library category.
  - Select on the “Add Library” option.
  - Select the JAR placed into the Library folder.


Project Properties - jdbMySelect-V2

Categories:
- Sources
- Libraries
- Build
  - Compiling
  - Packaging
  - Deployment
  - Documenting
- Run
- Application
  - Web Start
- License Headers
- Formatting
- Hints

Java Platform: JDK 1.8 | Manage Platforms...

Libraries Folder: | Browse...

Compile | Processor | Run | Compile Tests | Run Tests

Compile-time Libraries:

Modulepath

Classpath

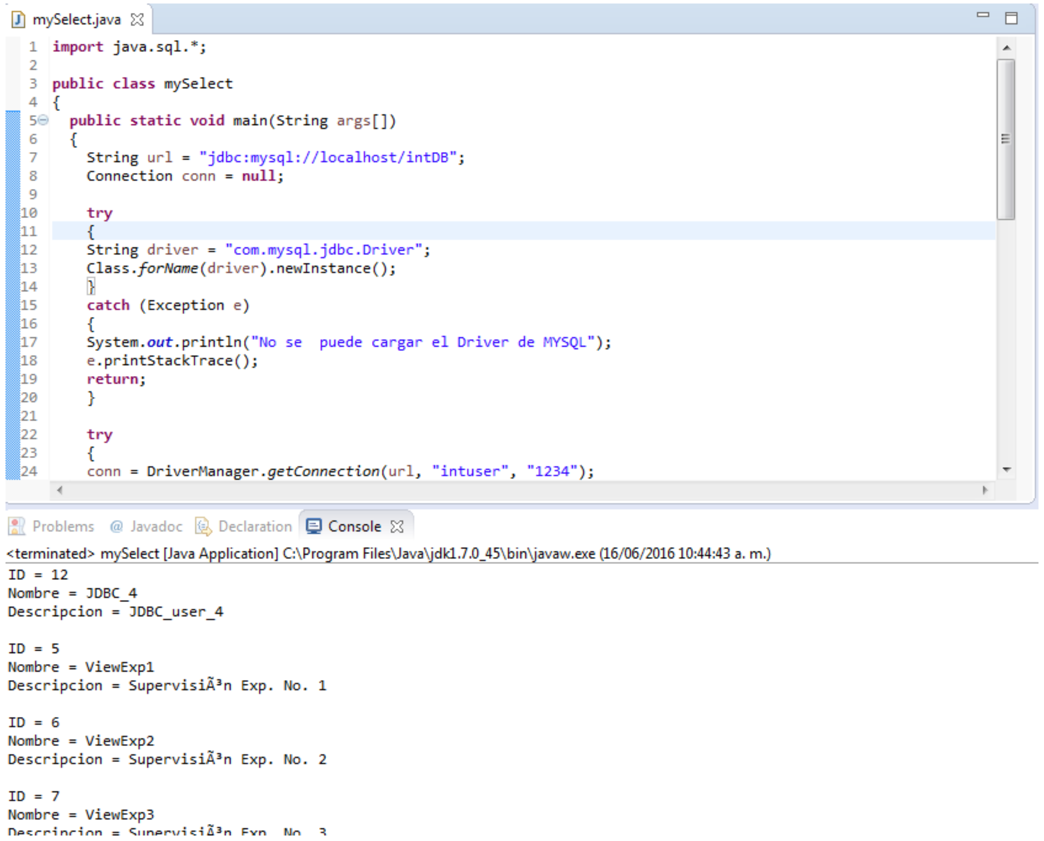
C:\Users\Admin\workspace\jdbMySelect-V2\lib\mariadb-java-client-2.7.5.jar

Compile-time libraries are propagated to all library categories.

☑ Build Dependencies

Basic Operations – SELECT

- **Data retrieving:**
  - Open project: **jdbcMySelect-V2**
  - **Observations**:
    - Check if the database name, user and password are correct.
    - It is important noting how results are retrieved using *rs.next()*.
    - For each field in a table, there is a specialized method which gets data according with its data type.
    - It is important to finish the connection with the database engine before ending an application.

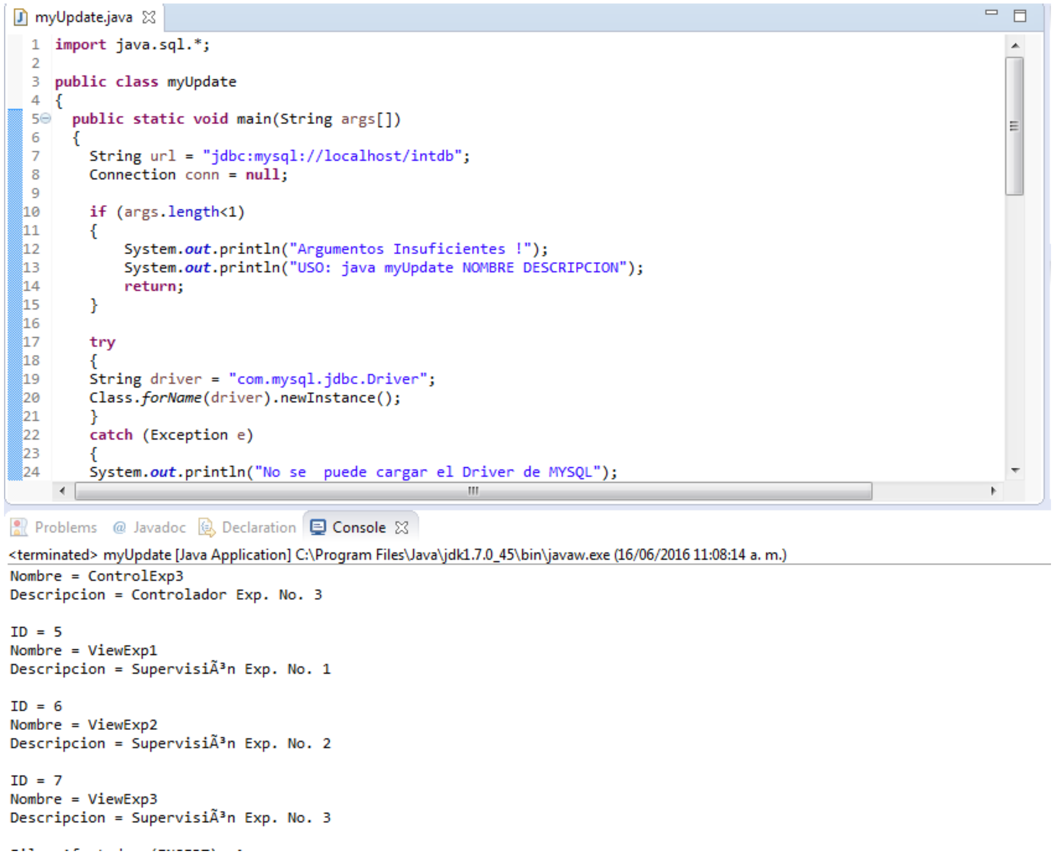


```java
import java.sql.*;

public class mySelect
{
  public static void main(String args[])
  {
    String url = "jdbc:mysql://localhost/intDB";
    Connection conn = null;

    try
    {
    String driver = "com.mysql.jdbc.Driver";
    Class.forName(driver).newInstance();
    }
    catch (Exception e)
    {
    System.out.println("No se  puede cargar el Driver de MYSQL");
    e.printStackTrace();
    return;
    }

    try
    {
    conn = DriverManager.getConnection(url, "intuser", "1234");
```

```
<terminated> mySelect [Java Application] C:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (16/06/2016 10:44:43 a. m.)
ID = 12
Nombre = JDBC_4
Descripcion = JDBC_user_4

ID = 5
Nombre = ViewExp1
Descripcion = SupervisiÃ³n Exp. No. 1

ID = 6
Nombre = ViewExp2
Descripcion = SupervisiÃ³n Exp. No. 2

ID = 7
Nombre = ViewExp3
Descripcion = SupervisiÃ³n Exp. No. 3
```

Basic Operations – UPDATE

- **Data updating:**
  - Open project: **jdbcMyUpdate-V2**
  - **Observations**:
    - In this case, *stm.executeUpdate()* was used instead of *stm.executeQuery().*
    - There is a returned value corresponding to the number of affected rows.
  - **Important note:**
    - All SQL queries requesting information to any DB are performed using **executeQuery()** method.
    - All SQL queries modifying data in any DB are performed using **executeUpdate().**
      - *INSERT*
      - *UDPATE*
      - *DELETE*


myUpdate.java

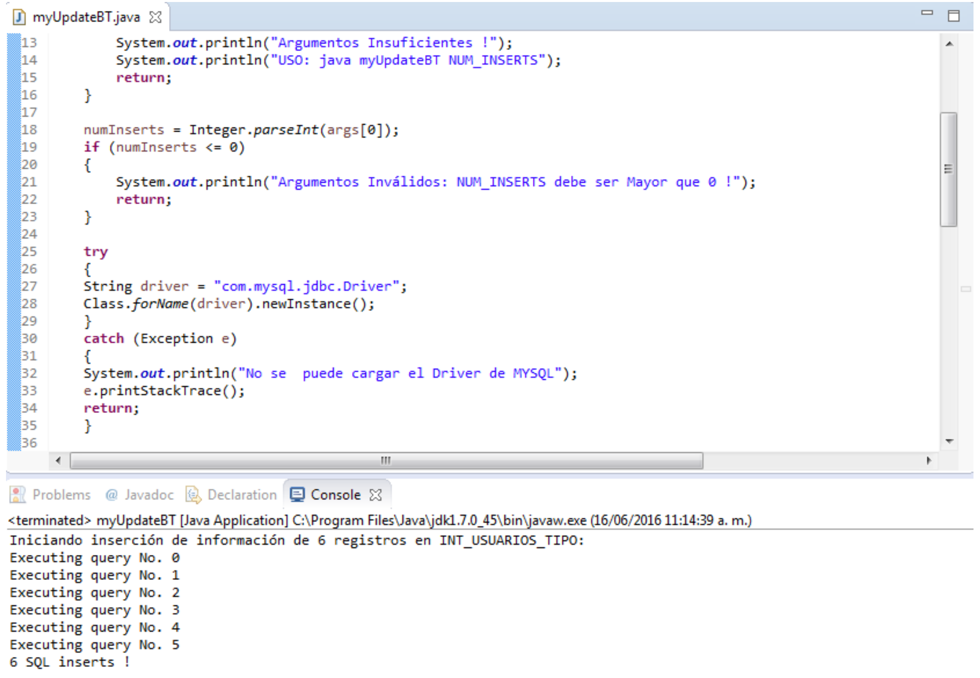
```java
import java.sql.*;

public class myUpdate
{
  public static void main(String args[])
  {
    String url = "jdbc:mysql://localhost/intdb";
    Connection conn = null;

    if (args.length<1)
    {
        System.out.println("Argumentos Insuficientes !");
        System.out.println("USO: java myUpdate NOMBRE DESCRIPCION");
        return;
    }

    try
    {
    String driver = "com.mysql.jdbc.Driver";
    Class.forName(driver).newInstance();
    }
    catch (Exception e)
    {
    System.out.println("No se  puede cargar el Driver de MYSQL");
```

```
<terminated> myUpdate [Java Application] C:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (16/06/2016 11:08:14 a. m.)
Nombre = ControlExp3
Descripcion = Controlador Exp. No. 3

ID = 5
Nombre = ViewExp1
Descripcion = SupervisiÃ³n Exp. No. 1

ID = 6
Nombre = ViewExp2
Descripcion = SupervisiÃ³n Exp. No. 2

ID = 7
Nombre = ViewExp3
Descripcion = SupervisiÃ³n Exp. No. 3
```

Advanced Concepts on JDBC – Basic Transactions

- By default, JDBC sends queries as soon as **executeUpdate()** or **executeQuery()** returns. This is called ***auto-commit***.
- However, it can be deactivated using the ***Connection*** class method ***setAutoCommit()***.
- Open project: **jdbcUpdateBasicTransactions-V2**

- **Observations**:
  - In this framework, the standard procedure is as follows:
    - Each SQL sentence is created (***createStatement()***),
    - The ***executeUpdate()*** or ***executeQuery()*** method is called
    - The SQL sentence is closed.
  - Finally, the ***Connection*** class method ***commit()*** is issued.
  - At this point, all SQL sentences are sent to the database engine.



```java
            System.out.println("Argumentos Insuficientes !");
            System.out.println("USO: java myUpdateBT NUM_INSERTS");
            return;
        }

        numInserts = Integer.parseInt(args[0]);
        if (numInserts <= 0)
        {
            System.out.println("Argumentos Inválidos: NUM_INSERTS debe ser Mayor que 0 !");
            return;
        }

        try
        {
        String driver = "com.mysql.jdbc.Driver";
        Class.forName(driver).newInstance();
        }
        catch (Exception e)
        {
        System.out.println("No se  puede cargar el Driver de MYSQL");
        e.printStackTrace();
        return;
        }

```

```
<terminated> myUpdateBT [Java Application] C:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (16/06/2016 11:14:39 a. m.)
Iniciando inserción de información de 6 registros en INT_USUARIOS_TIPO:
Executing query No. 0
Executing query No. 1
Executing query No. 2
Executing query No. 3
Executing query No. 4
Executing query No. 5
6 SQL inserts !
```

Advanced Concepts on JDBC – Prepared SQL Statements

- **Definition**: They are SQL statements stored in the database engine, and they can be called at any time.
- **Need**:
  - There are many queries used very often, where only some fields or selection conditions change.
  - In terms of database performance, these queries remains ready for execution, avoiding rebuild them in the database engine.
- **How to use it?**
  - PreparedStatement class

```
conn.setAutoCommit(false);
PreparedStatement stm = conn.prepareStatement(“INSERT INTO INT_PROCESO_VARS_DATA SET VALOR=?, TIEMPO=?, FECHA=?,
HORA=?, INT_PROCESO_VARS_ID=?’”);
```

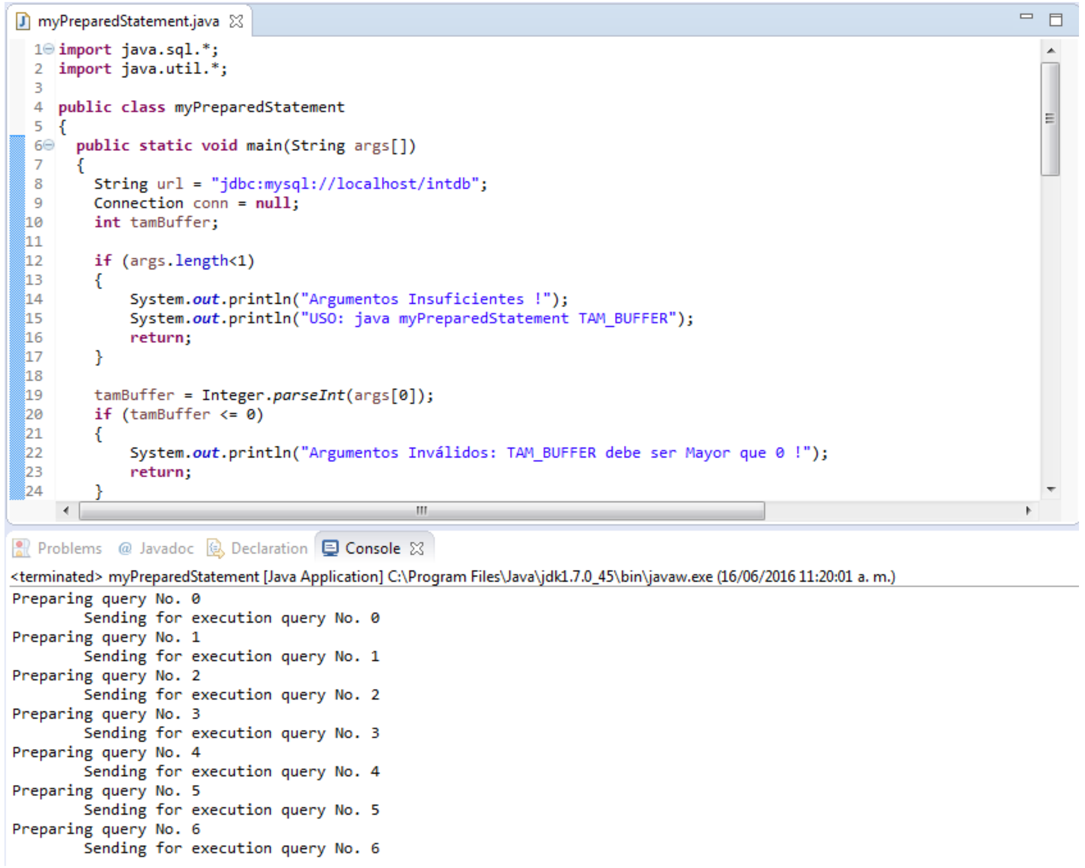
  - Afterwards, a loop can be used in order to introduce all values:

```
int i;
for(i=0; i<bufferData.length; i++)
{
 stm.setFloat(1, bufferData[i]);   stm.setFloat(2, bufferTime[i]);   stm.setDate(3, date.getDate());   stm.setTime(4, date.getTime());
stm.setInt(5, idVar);
 stm.execute();   stm.clearParameters();
}
conn.commit(); stm.close();
```

Advanced Concepts on JDBC – Prepared SQL Statements

- Open project: **jdbcMyPreparedStatement-V2**

- **Observations**:
  - Date and time objects are obtained from **java.sql.Date** and **java.sql.Time**.
  - In turn, these objects are initialized using the **GregorianCalendar** class from **java.utils**.


myPreparedStatement.java

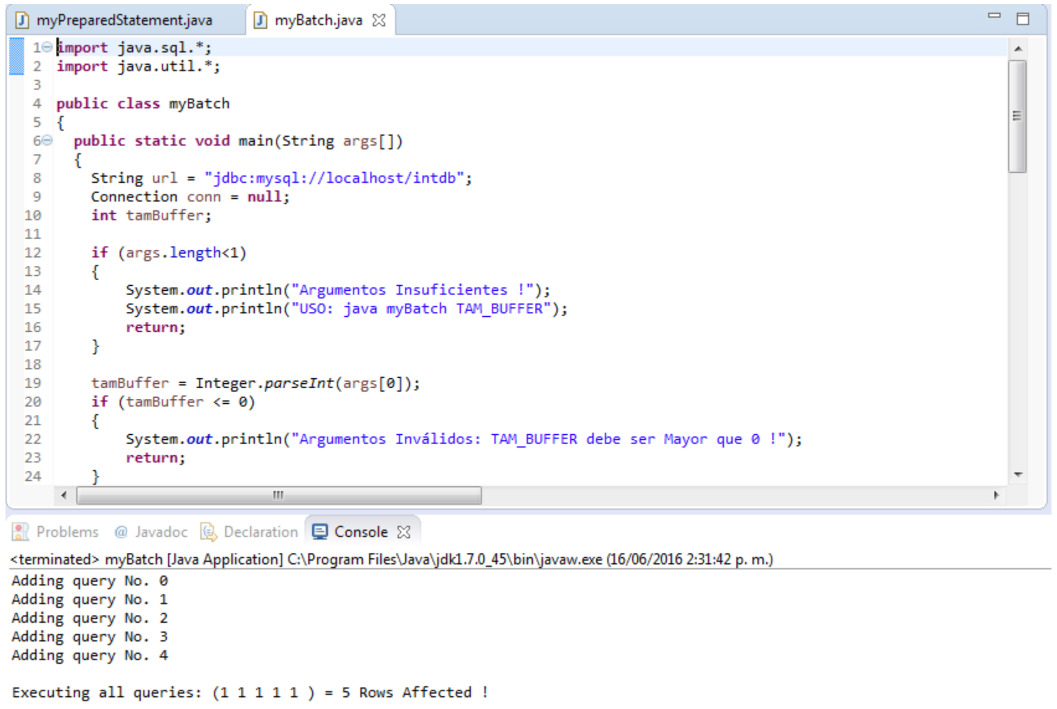
```
import java.sql.*;
import java.util.*;

public class myPreparedStatement
{
  public static void main(String args[])
  {
    String url = "jdbc:mysql://localhost/intdb";
    Connection conn = null;
    int tamBuffer;

    if (args.length<1)
    {
        System.out.println("Argumentos Insuficientes !");
        System.out.println("USO: java myPreparedStatement TAM_BUFFER");
        return;
    }

    tamBuffer = Integer.parseInt(args[0]);
    if (tamBuffer <= 0)
    {
        System.out.println("Argumentos Inválidos: TAM_BUFFER debe ser Mayor que 0 !");
        return;
    }
```

Problems @ Javadoc Declaration Console

```
<terminated> myPreparedStatement [Java Application] C:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (16/06/2016 11:20:01 a. m.)
Preparing query No. 0
        Sending for execution query No. 0
Preparing query No. 1
        Sending for execution query No. 1
Preparing query No. 2
        Sending for execution query No. 2
Preparing query No. 3
        Sending for execution query No. 3
Preparing query No. 4
        Sending for execution query No. 4
Preparing query No. 5
        Sending for execution query No. 5
Preparing query No. 6
        Sending for execution query No. 6
```

Advanced Concepts on JDBC – Batch Processing

- **Need and Properties**
  - A bunch of different database operations need to be executed independently of user interaction.
  - This kind of operation does not need manage results.
  - Normally, batch operations are related with CREATE, INSERT, UPDATE and DELETE.
  - It can be used with Prepared Statements also.
- Open project: **myBatch-V2**
- **Observations**:
  - **addBatch**() method is a tool for assigning a bunch of SQL statements to a JDBC Statement.
  - Once many SQL statements are grouped together, **executeBatch**() is called to execute them.



```java
import java.sql.*;
import java.util.*;

public class myBatch
{
  public static void main(String args[])
  {
    String url = "jdbc:mysql://localhost/intdb";
    Connection conn = null;
    int tamBuffer;

    if (args.length<1)
    {
        System.out.println("Argumentos Insuficientes !");
        System.out.println("USO: java myBatch TAM_BUFFER");
        return;
    }

    tamBuffer = Integer.parseInt(args[0]);
    if (tamBuffer <= 0)
    {
        System.out.println("Argumentos Inválidos: TAM_BUFFER debe ser Mayor que 0 !");
        return;
    }
```

```
<terminated> myBatch [Java Application] C:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (16/06/2016 2:31:42 p. m.)
Adding query No. 0
Adding query No. 1
Adding query No. 2
Adding query No. 3
Adding query No. 4

Executing all queries: (1 1 1 1 1 ) = 5 Rows Affected !
```

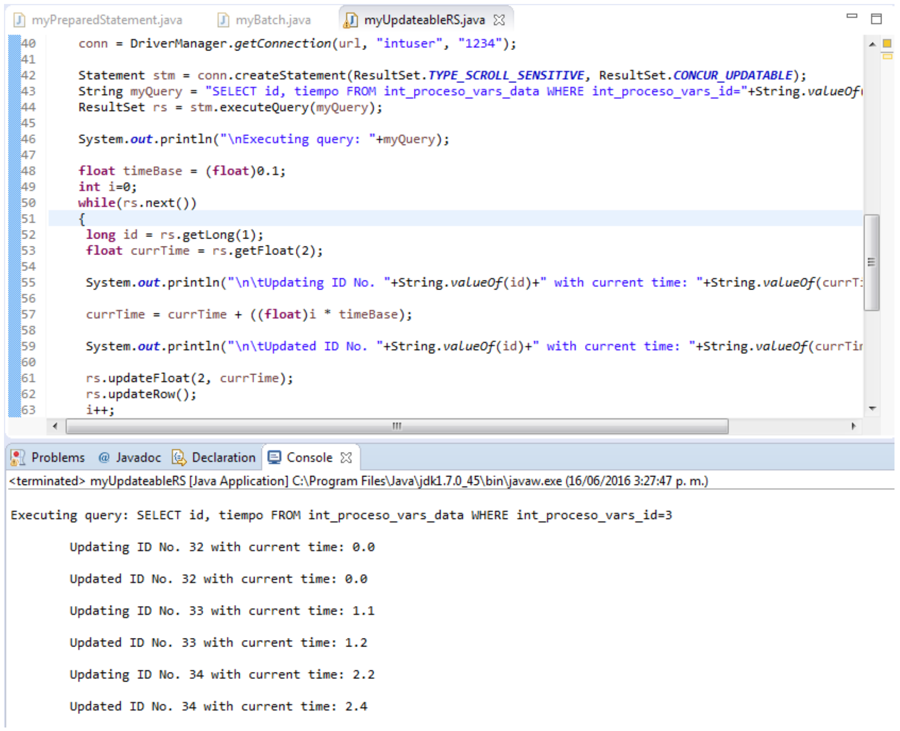
Advanced Concepts on JDBC – Updateable Result Sets

- **Need and properties**
  - Normally, at same time results are gathered from databases these data could be modified on-line.
  - In first place, the Statement must be created using specialized arguments of preparedStatement() or createStatement() methods:
    - ResultSet.TYPE_SCROLL_SENSITIVE which means that you can see changes made to the results if you scroll back to the modified row at a later time.
    - ResultSet.CONCUR_UPDATEABLE which means that each row of the ResultSet object can be modified.

    PreparedStatement stm = **conn.prepareStatement**("SELECT ID, BALANCE, INTERES FROM CUENTAS", **ResultSet**.TYPE_SCROLL_SENSITIVE, **ResultSet**.CONCUR_UPDATABLE);
  - **Caution**: It is applicable only from a single table and including the primary key columns.
  - At the end, updatable result sets builds hidden UPDATE SQL sentences for you.

Advanced Concepts on JDBC – Updateable Result Sets

- Open project: **myUpdateResulSets-V2**
- **Observations**:
  - JDBC introduces **updateXXX**() methods to match its **getXXX**() methods in order to enable you to update the **ResultSet** object.
  - In the same way **ResultSets** are updatable using **UPDATE**, this can be done with **DELETE** and **INSERT** sentences.
  - In case of **DELETE** sentences, **rs.deleteRow**() method is used in order to delete the current row.
  - In case of **INSERT** sentences, a new row is created using **rs.moveToInsertRow**() method, then using **rs.updateXXX**() methods values can be modified, afterwards **rs.insertRow**() method is called in order to make changes permanent, finally calling **rs.moveToCurrentRow**() returns you to the row you were on before the **INSERT** operation.



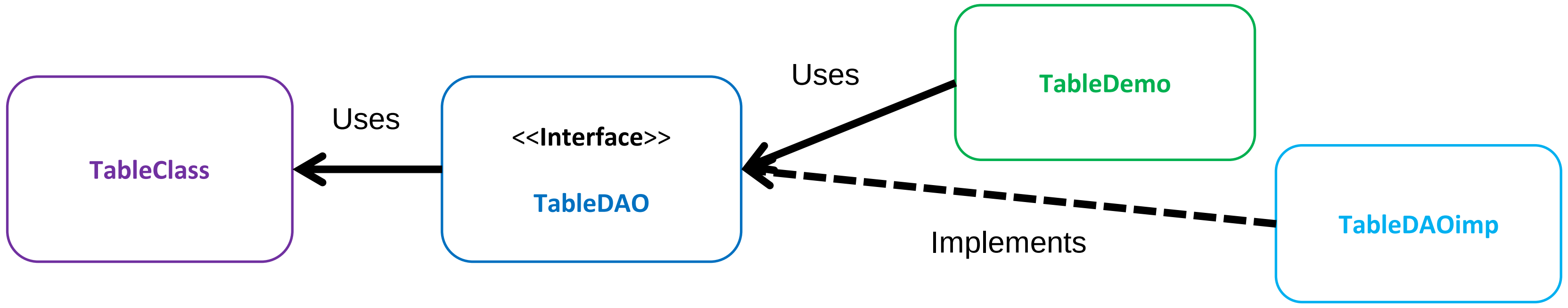
```java
    conn = DriverManager.getConnection(url, "intuser", "1234");

    Statement stm = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
    String myQuery = "SELECT id, tiempo FROM int_proceso_vars_data WHERE int_proceso_vars_id="+String.valueOf(
    ResultSet rs = stm.executeQuery(myQuery);

    System.out.println("\nExecuting query: "+myQuery);

    float timeBase = (float)0.1;
    int i=0;
    while(rs.next())
    {
     long id = rs.getLong(1);
     float currTime = rs.getFloat(2);

     System.out.println("\n\tUpdating ID No. "+String.valueOf(id)+" with current time: "+String.valueOf(currT

     currTime = currTime + ((float)i * timeBase);

     System.out.println("\n\tUpdated ID No. "+String.valueOf(id)+" with current time: "+String.valueOf(currTi

     rs.updateFloat(2, currTime);
     rs.updateRow();
     i++;
```

```
<terminated> myUpdateableRS [Java Application] C:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (16/06/2016 3:27:47 p. m.)

Executing query: SELECT id, tiempo FROM int_proceso_vars_data WHERE int_proceso_vars_id=3

        Updating ID No. 32 with current time: 0.0

        Updated ID No. 32 with current time: 0.0

        Updating ID No. 33 with current time: 1.1

        Updated ID No. 33 with current time: 1.2

        Updating ID No. 34 with current time: 2.2

        Updated ID No. 34 with current time: 2.4
```

DAO – Data Access Objects

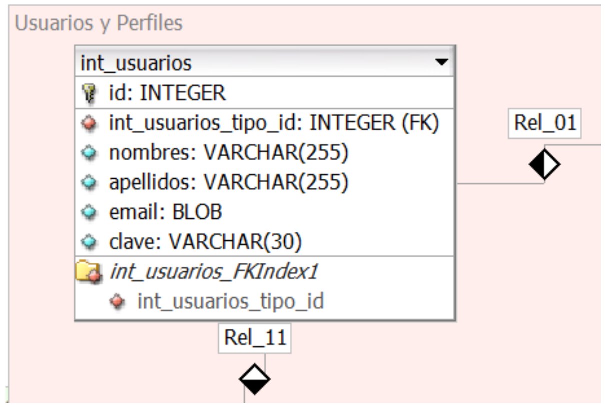
- **DAO** is a design pattern for software development.
- **Design patterns**: It represents the best practices used by experienced object-oriented software developers.
- Design patterns are solutions to general problems that software developers faced during software development.
- **DAO**: Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business services.

- **Components**:
  - **Model Object or Value Object** - This object is simple POJO (Plain Old Java Object) containing get/set methods to store data retrieved using DAO class.
  - **DAO Interface** - This interface defines the standard operations to be performed on a model object(s) (CRUDS)
  - **DAO concrete class** - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.

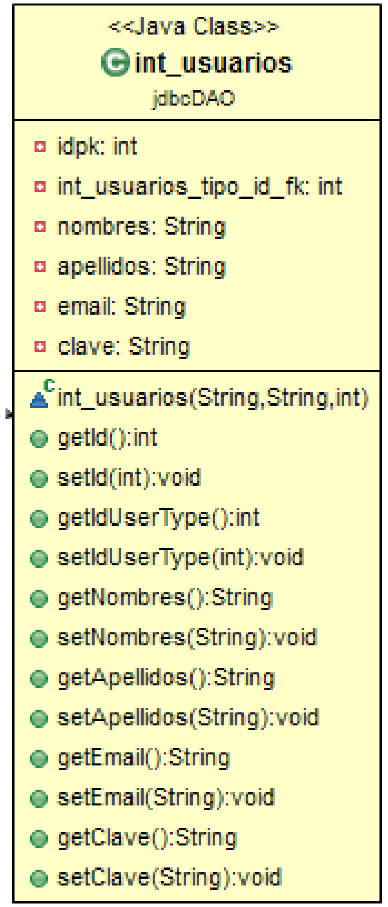


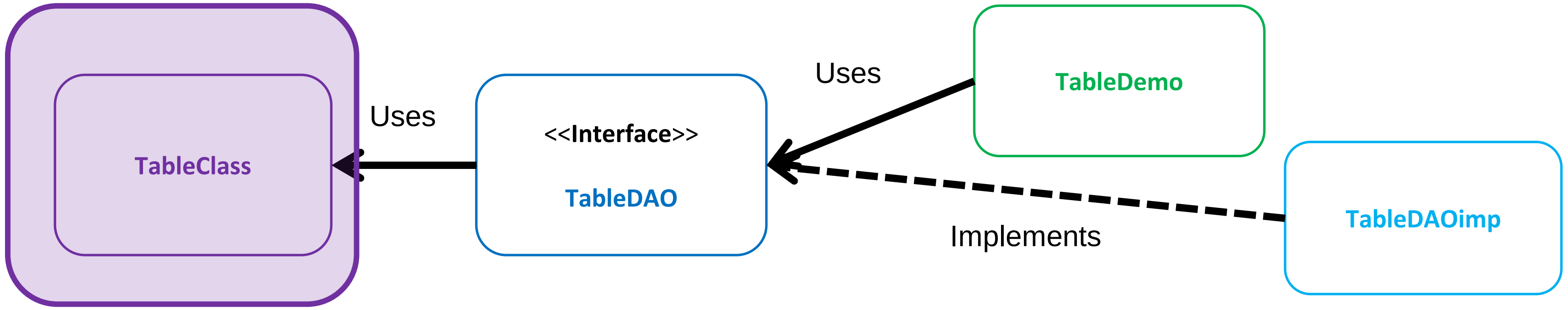
DAO – Data Access Objects

- **Example**: *int_usuarios* table.
- **Project**: **jdbcDAO-V2** on Netbeans.
- **Model Object or Value Object** - This object is simple POJO (Plain Old Java Object) containing get/set methods to store data retrieved using DAO class.


Usuarios y Perfiles

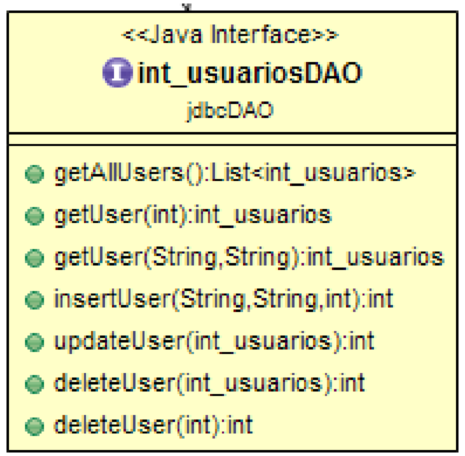
```
int_usuarios
id: INTEGER
int_usuarios_tipo_id: INTEGER (FK)
nombres: VARCHAR(255)
apellidos: VARCHAR(255)
email: BLOB
clave: VARCHAR(30)
int_usuarios_FKIndex1
  int_usuarios_tipo_id
```


```
<<Java Class>>
int_usuarios
jdbcDAO

idpk: int
int_usuarios_tipo_id_fk: int
nombres: String
apellidos: String
email: String
clave: String

int_usuarios(String,String,int)
getId():int
setId(int):void
getIdUserType():int
setIdUserType(int):void
getNombres():String
setNombres(String):void
getApellidos():String
setApellidos(String):void
getEmail():String
setEmail(String):void
getClave():String
setClave(String):void
```

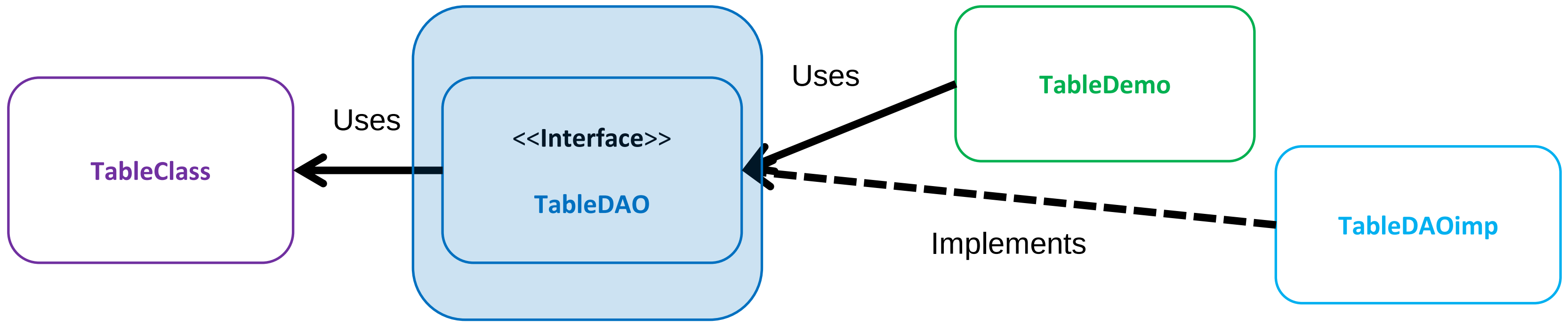


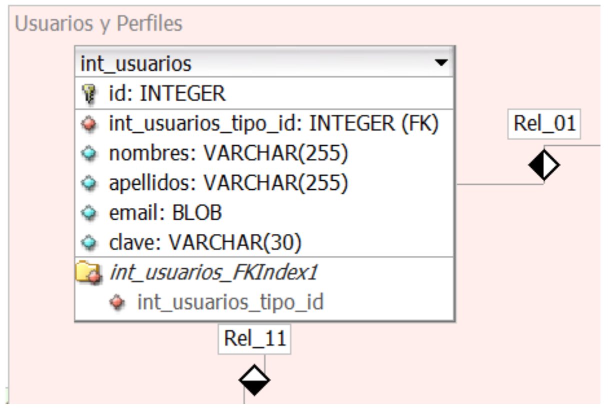
DAO – Data Access Objects

- **Example**: *int_usuarios* table.
- **Project**: **jdbcDAO-V2** on Netbeans.
- **DAO Interface** - This interface defines the standard operations to be performed on a model object(s) (CRUDS).



```
<<Java Interface>>
int_usuariosDAO
jdbcDAO

getAllUsers():List<int_usuarios>
getUser(int):int_usuarios
getUser(String,String):int_usuarios
insertUser(String,String,int):int
updateUser(int_usuarios):int
deleteUser(int_usuarios):int
deleteUser(int):int
```



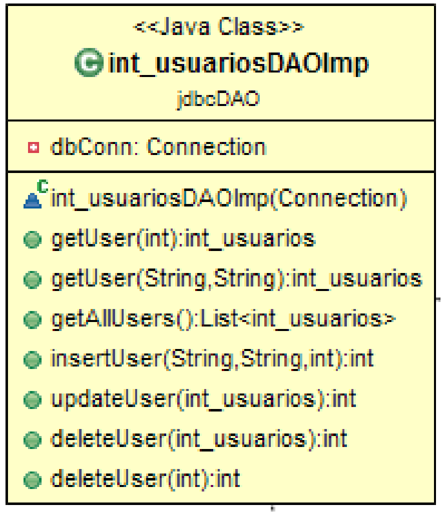


Usuarios y Perfiles

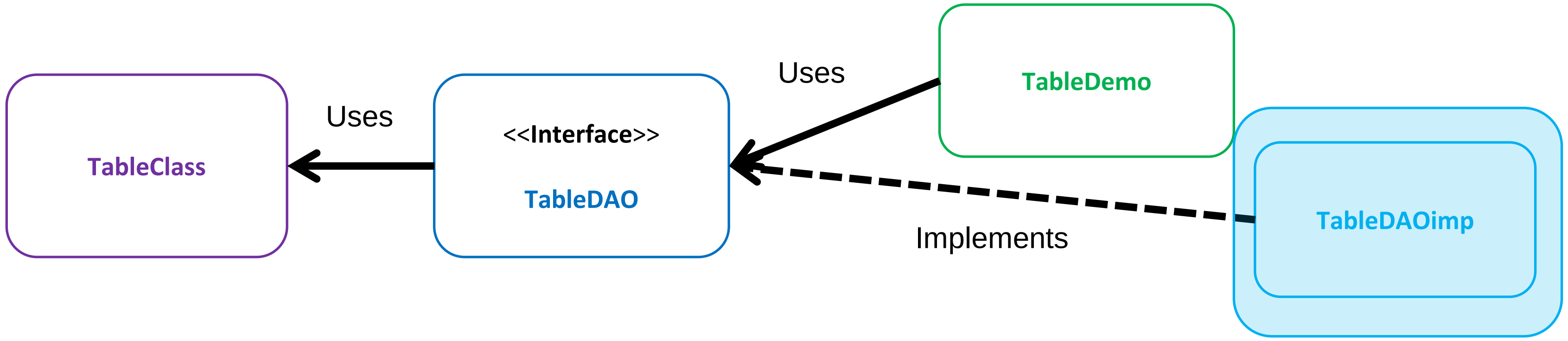
```
int_usuarios
id: INTEGER
int_usuarios_tipo_id: INTEGER (FK)
nombres: VARCHAR(255)
apellidos: VARCHAR(255)
email: BLOB
clave: VARCHAR(30)
int_usuarios_FKIndex1
  int_usuarios_tipo_id
```

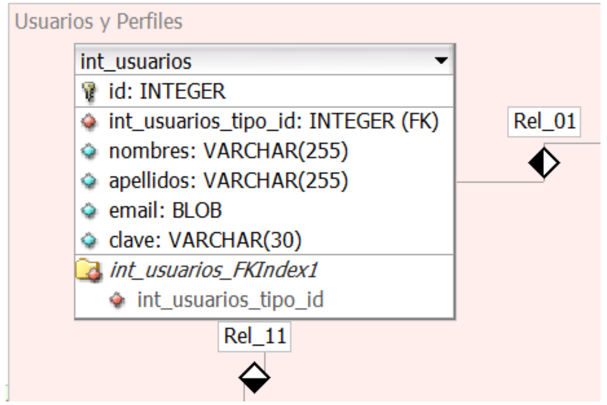
DAO – Data Access Objects

- **Example**: *int_usuarios* table.
- **Project**: **jdbcDAO-V2** on Netbeans.
- **DAO concrete class** - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism



```
<<Java Class>>
int_usuariosDAOImp
jdbcDAO
dbConn: Connection
int_usuariosDAOImp(Connection)
getUser(int):int_usuarios
getUser(String,String):int_usuarios
getAllUsers():List<int_usuarios>
insertUser(String,String,int):int
updateUser(int_usuarios):int
deleteUser(int_usuarios):int
deleteUser(int):int
```



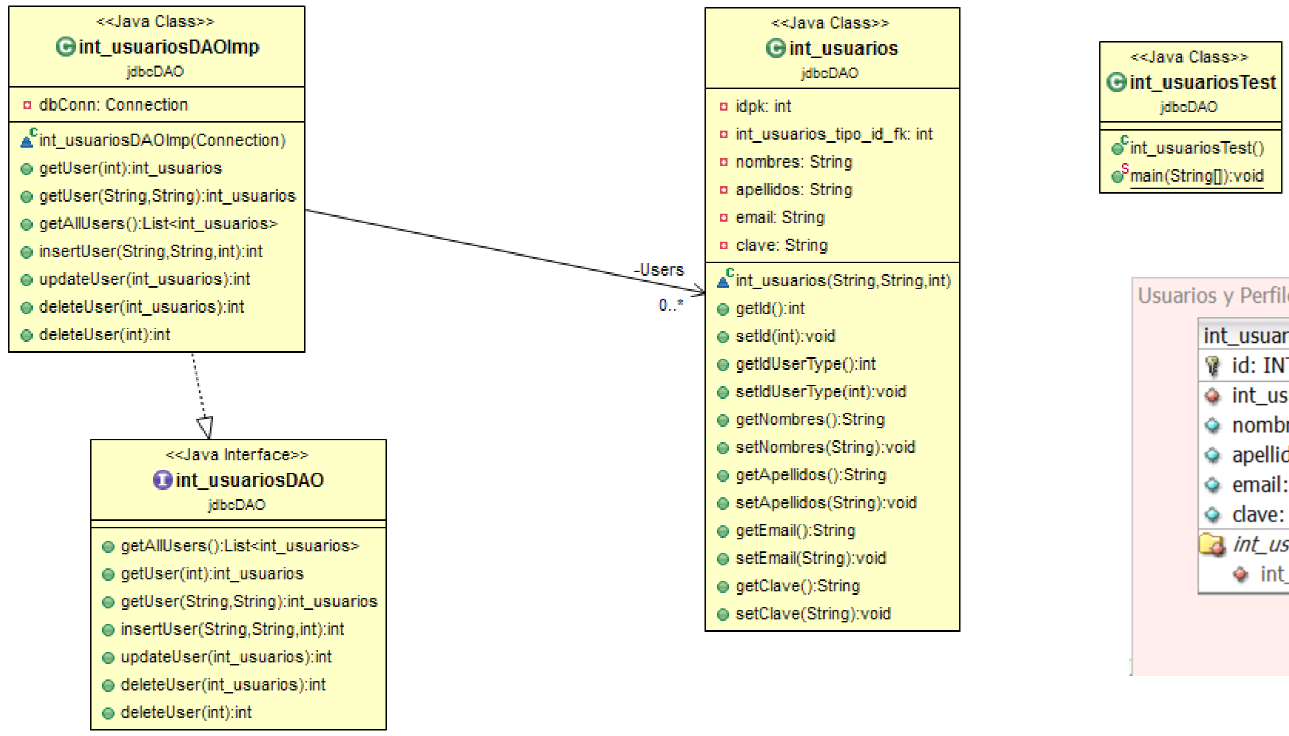


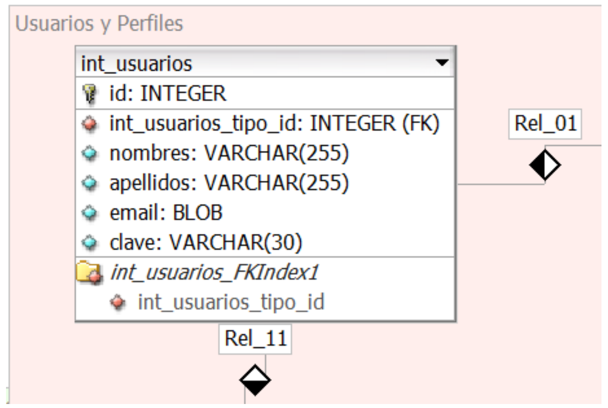
Usuarios y Perfiles

```
int_usuarios
id: INTEGER
int_usuarios_tipo_id: INTEGER (FK)
nombres: VARCHAR(255)
apellidos: VARCHAR(255)
email: BLOB
clave: VARCHAR(30)
int_usuarios_FKIndex1
int_usuarios_tipo_id
```

DAO – Data Access Objects

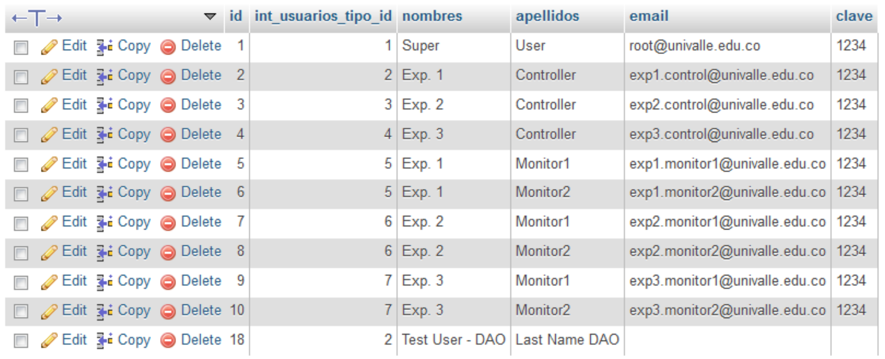
- **Example**: *int_usuarios* table.
- **Project**: **jdbcDAO-V2** on Netbeans.



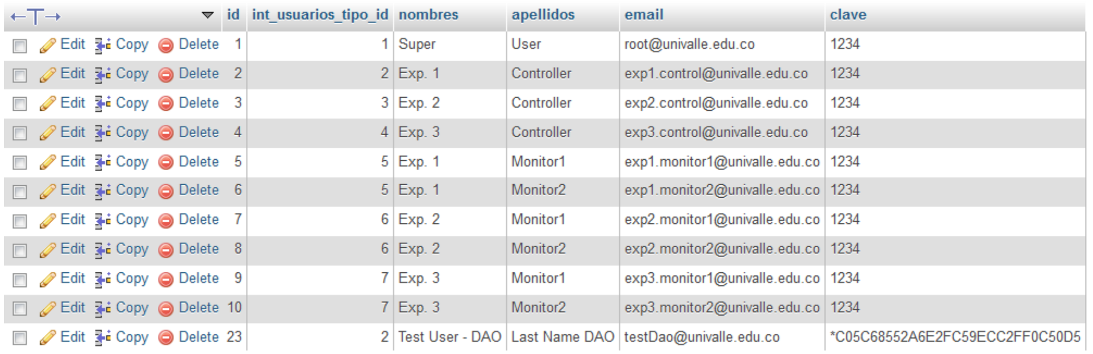


DAO – Data Access Objects

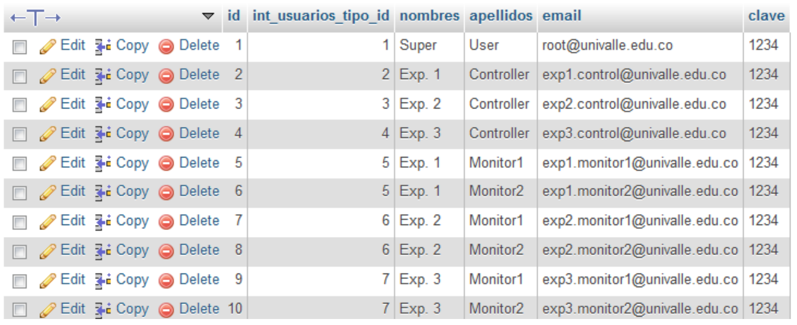
- **Example**: *int_usuarios* table.
- **Project**: **jdbcDAO-V2** on Netbeans.
- Run **int_usuariosTest.java**, then:
  - Press 1 and ENTER
  - Press 2 and ENTER, check the PHPMYADMIN to validate the information.
  - Press 3 and ENTER , check the PHPMYADMIN to validate the information.
  - Press 4 and ENTER , check the PHPMYADMIN to validate the information.



←T→	id	int_usuarios_tipo_id	nombres	apellidos	email	clave
Edit Copy Delete	1	1	Super	User	root@univalle.edu.co	1234
Edit Copy Delete	2	2	Exp. 1	Controller	exp1.control@univalle.edu.co	1234
Edit Copy Delete	3	3	Exp. 2	Controller	exp2.control@univalle.edu.co	1234
Edit Copy Delete	4	4	Exp. 3	Controller	exp3.control@univalle.edu.co	1234
Edit Copy Delete	5	5	Exp. 1	Monitor1	exp1.monitor1@univalle.edu.co	1234
Edit Copy Delete	6	5	Exp. 1	Monitor2	exp1.monitor2@univalle.edu.co	1234
Edit Copy Delete	7	6	Exp. 2	Monitor1	exp2.monitor1@univalle.edu.co	1234
Edit Copy Delete	8	6	Exp. 2	Monitor2	exp2.monitor2@univalle.edu.co	1234
Edit Copy Delete	9	7	Exp. 3	Monitor1	exp3.monitor1@univalle.edu.co	1234
Edit Copy Delete	10	7	Exp. 3	Monitor2	exp3.monitor2@univalle.edu.co	1234
Edit Copy Delete	18	2	Test User - DAO	Last Name DAO		



←T→	id	int_usuarios_tipo_id	nombres	apellidos	email	clave
Edit Copy Delete	1	1	Super	User	root@univalle.edu.co	1234
Edit Copy Delete	2	2	Exp. 1	Controller	exp1.control@univalle.edu.co	1234
Edit Copy Delete	3	3	Exp. 2	Controller	exp2.control@univalle.edu.co	1234
Edit Copy Delete	4	4	Exp. 3	Controller	exp3.control@univalle.edu.co	1234
Edit Copy Delete	5	5	Exp. 1	Monitor1	exp1.monitor1@univalle.edu.co	1234
Edit Copy Delete	6	5	Exp. 1	Monitor2	exp1.monitor2@univalle.edu.co	1234
Edit Copy Delete	7	6	Exp. 2	Monitor1	exp2.monitor1@univalle.edu.co	1234
Edit Copy Delete	8	6	Exp. 2	Monitor2	exp2.monitor2@univalle.edu.co	1234
Edit Copy Delete	9	7	Exp. 3	Monitor1	exp3.monitor1@univalle.edu.co	1234
Edit Copy Delete	10	7	Exp. 3	Monitor2	exp3.monitor2@univalle.edu.co	1234
Edit Copy Delete	23	2	Test User - DAO	Last Name DAO	testDao@univalle.edu.co	*C05C68552A6E2FC59ECC2FF0C50D5



←T→	id	int_usuarios_tipo_id	nombres	apellidos	email	clave
Edit Copy Delete	1	1	Super	User	root@univalle.edu.co	1234
Edit Copy Delete	2	2	Exp. 1	Controller	exp1.control@univalle.edu.co	1234
Edit Copy Delete	3	3	Exp. 2	Controller	exp2.control@univalle.edu.co	1234
Edit Copy Delete	4	4	Exp. 3	Controller	exp3.control@univalle.edu.co	1234
Edit Copy Delete	5	5	Exp. 1	Monitor1	exp1.monitor1@univalle.edu.co	1234
Edit Copy Delete	6	5	Exp. 1	Monitor2	exp1.monitor2@univalle.edu.co	1234
Edit Copy Delete	7	6	Exp. 2	Monitor1	exp2.monitor1@univalle.edu.co	1234
Edit Copy Delete	8	6	Exp. 2	Monitor2	exp2.monitor2@univalle.edu.co	1234
Edit Copy Delete	9	7	Exp. 3	Monitor1	exp3.monitor1@univalle.edu.co	1234
Edit Copy Delete	10	7	Exp. 3	Monitor2	exp3.monitor2@univalle.edu.co	1234

Questions?