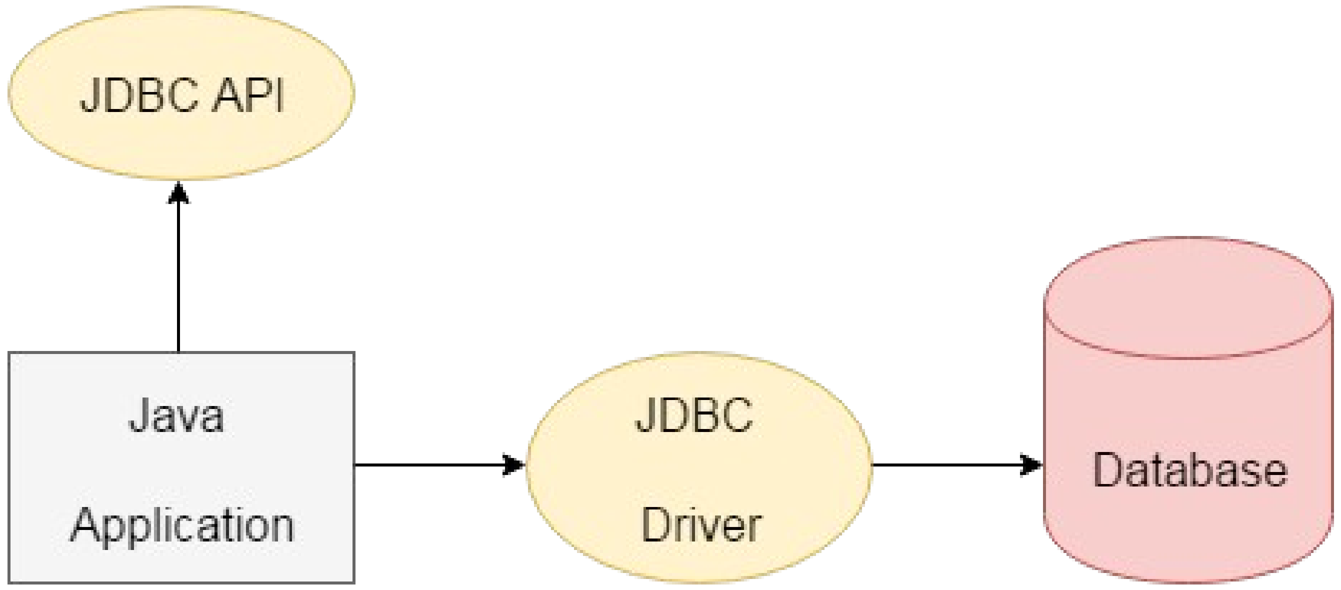
JDBC

- JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database.
- We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database.

JDBC



- The current version of JDBC is 4.3. It is the stable release since 21st September, 2017. It is based on the X/Open SQL Call Level Interface. The java.sql package contains classes and interfaces for JDBC API. A list of popular interfaces of JDBC API are given below:

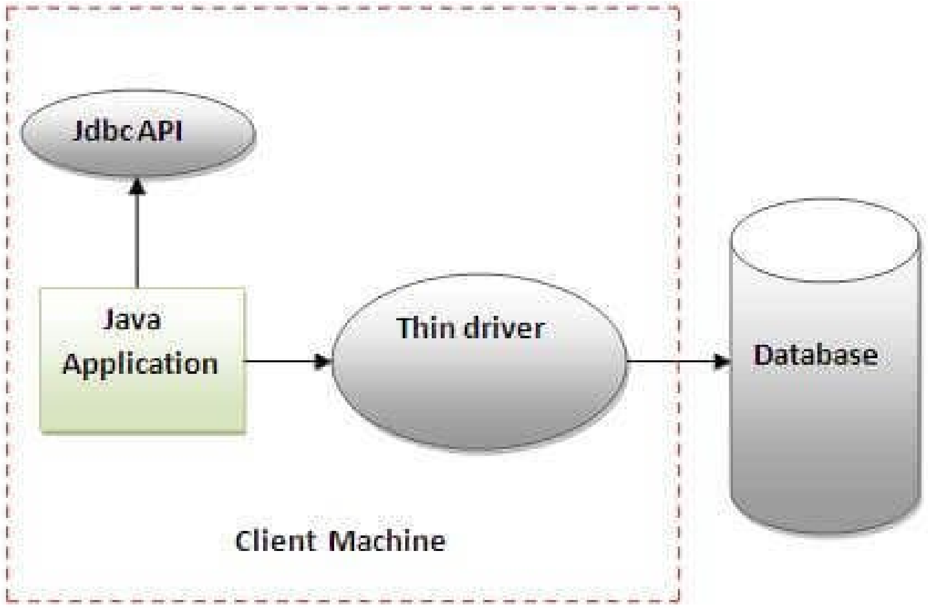
  Driver   Connection   Statement   PreparedStatement
  CallableStatement   ResultSet   ResultSetMetaData
  DatabaseMetaData   RowSet

- A list of popular classes of JDBC API are given below:
- DriverManager class
- Blob class
- Clob class
- Types class

- We can use JDBC API to handle database using Java program and can perform the following activities:
- Connect to the database
- Execute queries and update statements to the database
- Retrieve the result received from the database.

- JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:
- JDBC-ODBC bridge driver
- Native-API driver (partially java driver)
- Network Protocol driver (fully java driver)
- Thin driver (fully java driver)

- The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.



- Thin Driver
- Advantage:
    - Better performance than all other drivers.
    - No software is required at client side or server side.
- Disadvantage:
    - Drivers depend on the Database.

Java Database Connectivity

- There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:
- Register the Driver class (optional)
- Create connection
- Create statement
- Execute queries
- Close connection

- Create the connection object
- The getConnection() method of DriverManager class is used to establish connection with the database.
- Syntax of getConnection() method

```
public static Connection
    getConnection(String url) throws SQLException

public static Connection
getConnection(String url,String name,String password)
    throws SQLException
```

- Example to establish connection with a MySQL database

```
String userName = "student";
String password = "Student10*";
String url =
"jdbc:mysql://localhost/stest";
conn = DriverManager.getConnection(url,
   userName, password);
```

- <u>Create the Statement object</u>
- The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.
- Syntax of createStatement() method

  public Statement createStatement()throws SQLException
- Example:

```
Statement stmt=con.createStatement();
```

- Execute the query

- The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

- Syntax of executeQuery() method

```
public ResultSet executeQuery(String sql) throws
    SQLException
```

- Example to execute query

```
ResultSet rs =
   stmt.executeQuery("select * from emp");

while (rs.next()){
   System.out.println(rs.getInt(1) + " " +
      rs.getString(2));
}
```

- <u>Close the connection object</u>
- By closing connection object, statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.
- Syntax of close() method

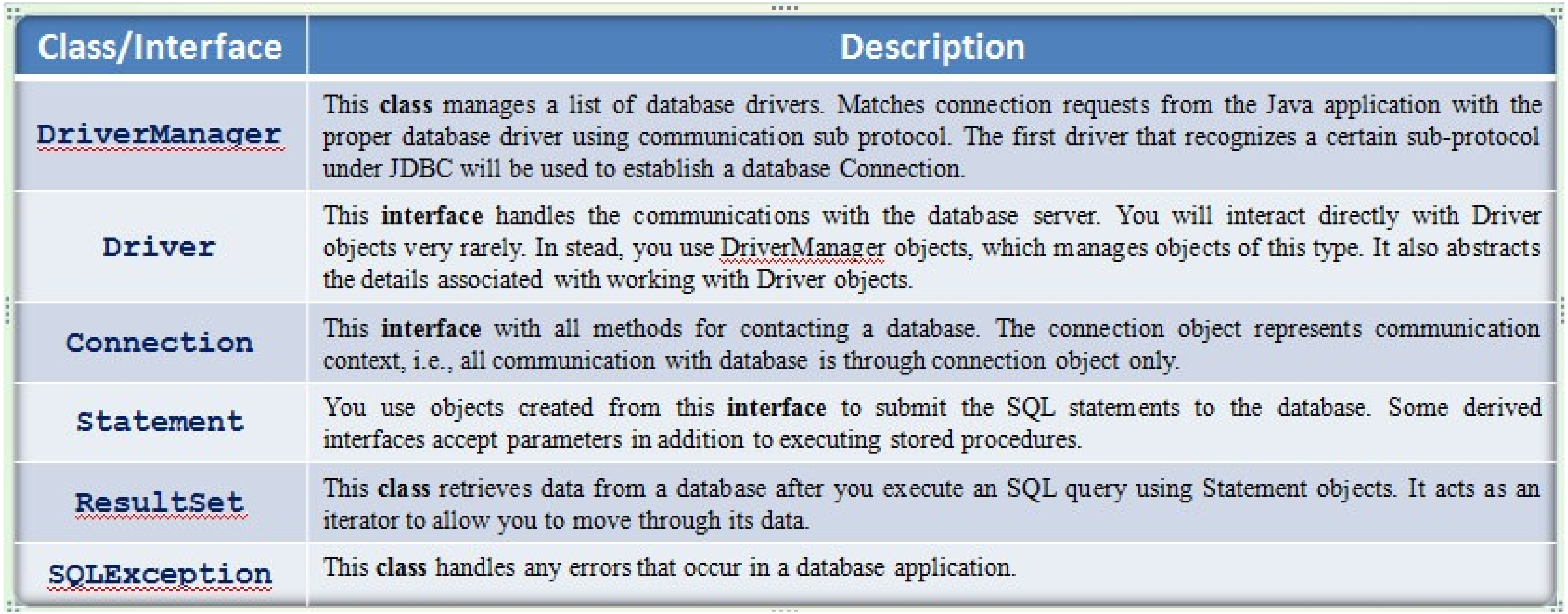
```
public void close()throws SQLException
```

- Example:

```
con.close();
```

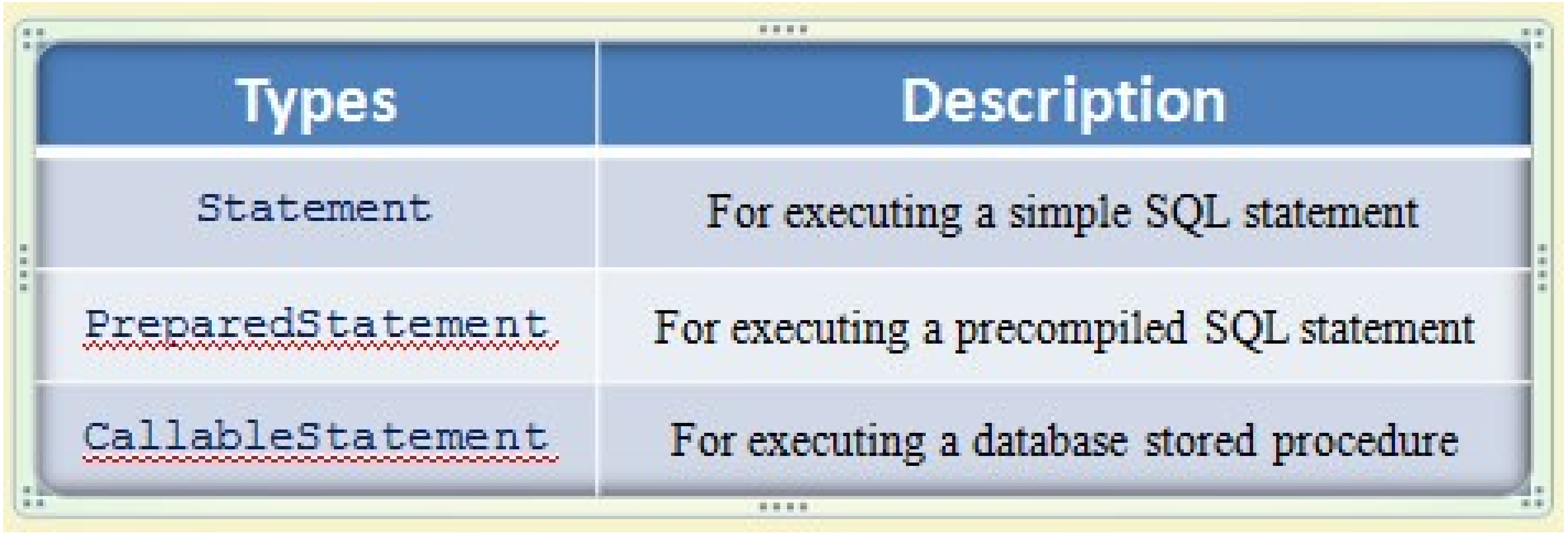
- Note: Since Java 7, JDBC has ability to use try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement.

• Comman JDBC Components



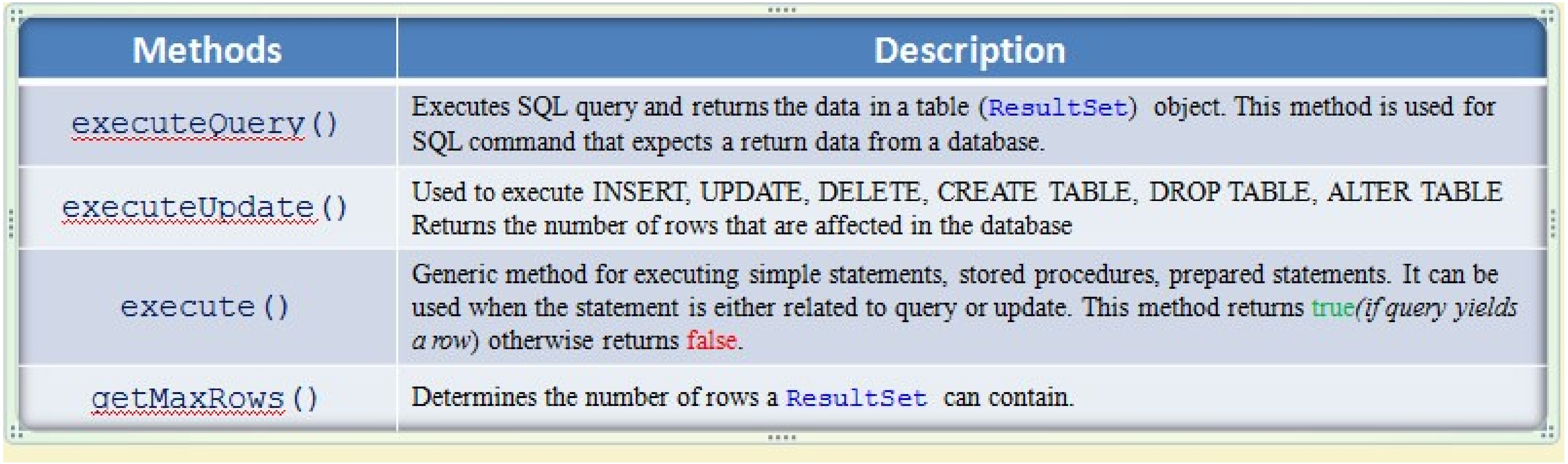
Class/Interface	Description
DriverManager	This **class** manages a list of database drivers. Matches connection requests from the Java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain sub-protocol under JDBC will be used to establish a database Connection.
Driver	This **interface** handles the communications with the database server. You will interact directly with Driver objects very rarely. In stead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
Connection	This **interface** with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
Statement	You use objects created from this **interface** to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
ResultSet	This **class** retrieves data from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
SQLException	This **class** handles any errors that occur in a database application.

- <u>Three Types of Statements</u>



Types	Description
Statement	For executing a simple SQL statement
PreparedStatement	For executing a precompiled SQL statement
CallableStatement	For executing a database stored procedure

Useful Methods in Statement class



Methods	Description
executeQuery()	Executes SQL query and returns the data in a table (ResultSet) object. This method is used for SQL command that expects a return data from a database.
executeUpdate()	Used to execute INSERT, UPDATE, DELETE, CREATE TABLE, DROP TABLE, ALTER TABLE Returns the number of rows that are affected in the database
execute()	Generic method for executing simple statements, stored procedures, prepared statements. It can be used when the statement is either related to query or update. This method returns true(*if query yields a row*) otherwise returns false.
getMaxRows()	Determines the number of rows a ResultSet can contain.

ResultSet

A ResultSet provides access to a table of data generated by executing a Statement.

Only one ResultSet per Statement can be open at once.

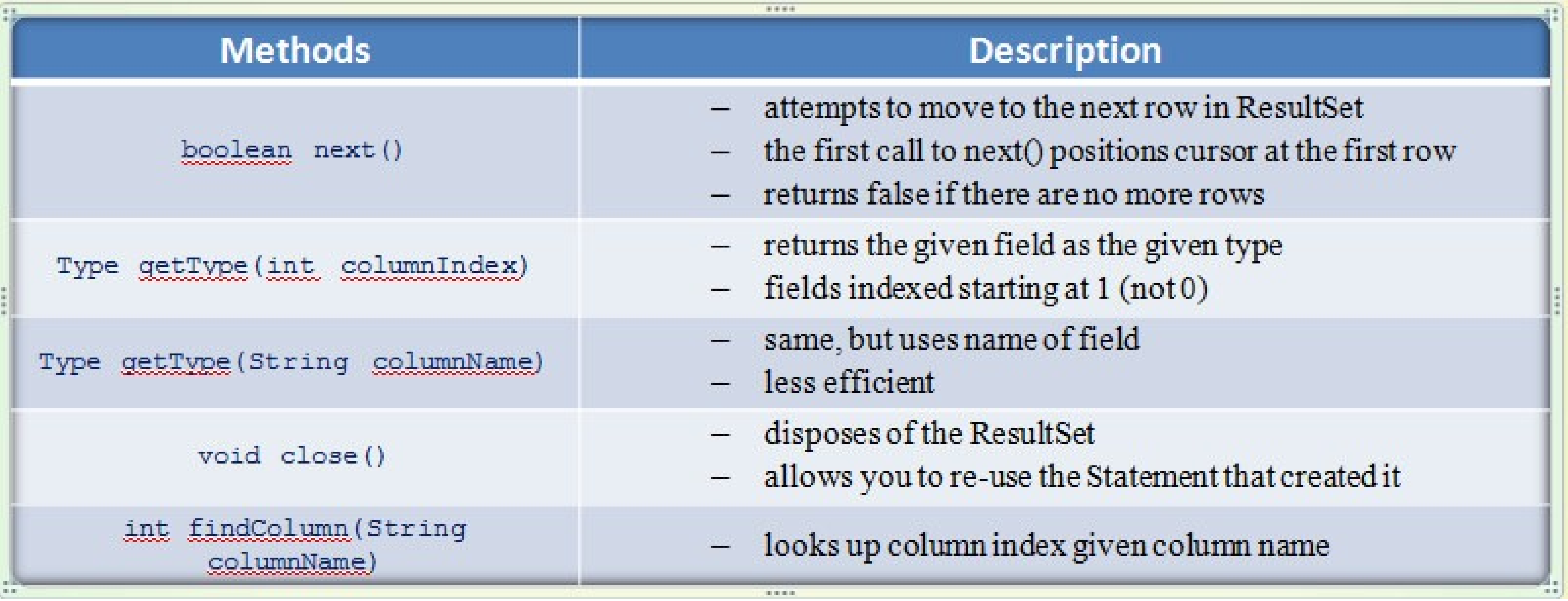
The table rows are retrieved in sequence.

A ResultSet maintains a cursor pointing to its current row of data.

The next() method moves the cursor to the next row.

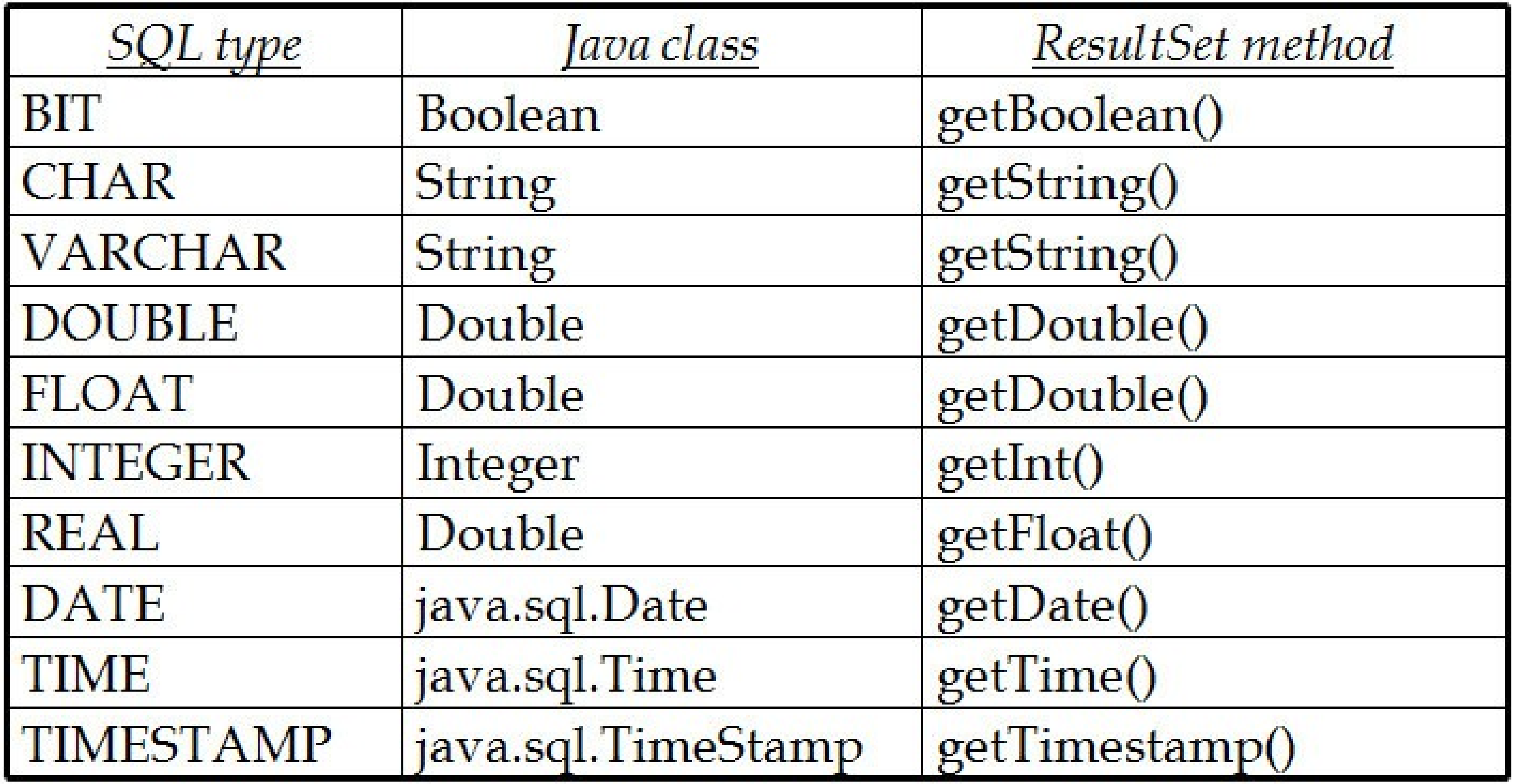
you can't rewind.

Useful ResultSet Methods



Methods	Description
`boolean next()`	– attempts to move to the next row in ResultSet – the first call to next() positions cursor at the first row – returns false if there are no more rows
`Type getType(int columnIndex)`	– returns the given field as the given type – fields indexed starting at 1 (not 0)
`Type getType(String columnName)`	– same, but uses name of field – less efficient
`void close()`	– disposes of the ResultSet – allows you to re-use the Statement that created it
`int findColumn(String columnName)`	– looks up column index given column name

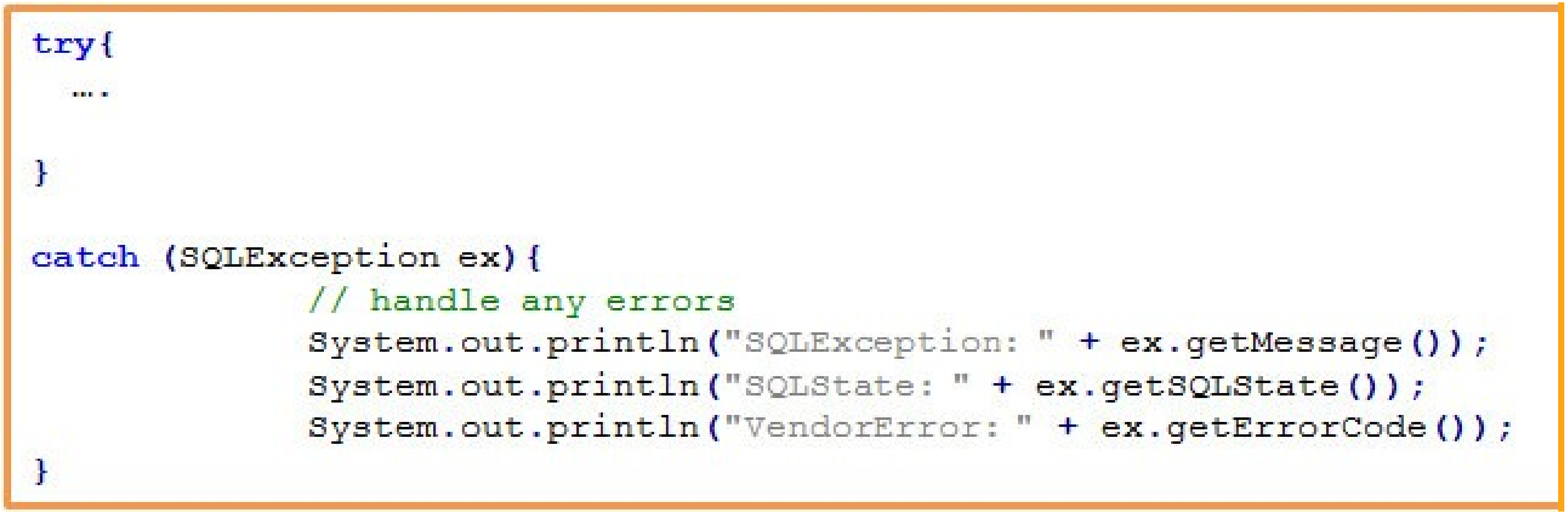
Matching with Java and SQL data types



SQL type	*Java class*	*ResultSet method*
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

JDBC exceptions

SQLException is an exception class which provides information on database access errors.



```
try{
  ….

}

catch (SQLException ex){
             // handle any errors
             System.out.println("SQLException: " + ex.getMessage());
             System.out.println("SQLState: " + ex.getSQLState());
             System.out.println("VendorError: " + ex.getErrorCode());
}
```

Creating Statements

- A Statement is an interface that represents a SQL statement. You execute Statement objects, and they generate ResultSet objects, which is a table of data representing a database result set. You need a Connection object to create a Statement object.

```
stmt = con.createStatement();
```

Executing Queries

- To execute a query, call an execute method from Statement such as the following:
- execute: Returns true if the first object that the query returns is a ResultSet object. Use this method if the query could return one or more ResultSet objects. Retrieve the ResultSet objects returned from the query by repeatedly calling Statement.getResultSet.
- executeQuery: Returns one ResultSet object.

Executing Queries (contd.)

- executeUpdate: Returns an integer representing the number of rows affected by the SQL statement. Use this method if you are using INSERT, DELETE, or UPDATE SQL statements.

```
ResultSet rs = stmt.executeQuery(query);
```

Processing ResultSet Objects

- You access the data in a ResultSet object through a cursor. Note that this cursor is not a database cursor. This cursor is a pointer that points to one row of data in the ResultSet object. Initially, the cursor is positioned before the first row. You call various methods defined in the ResultSet object to move the cursor.

Closing Connections

- When you are finished using a Statement, call the method Statement.close to immediately release the resources it is using. When you call this method, its ResultSet objects are closed.
- To ensure that the Statement object is closed at the end of the method, regardless of any SQLException objects thrown, wrap it in a finally block:

```
} finally {
    if (stmt != null) { stmt.close(); }
}
```

- The following statement is an try-with-resources statement, which declares one resource, stmt, that will be automatically closed when the try block terminates:

```
try (Statement stmt = con.createStatement()) {
    // ...
}
```

Programs on JDBC

- UserStudent.java
- ConnectDB.java
- CreateTable.java
- InsertRecords.java
- UpdateRecords
- UpdateRecords2.java
- DeleteRecords.java
- DropTable.java
- PreparedStmt.java
- AvgMarks.java