Pemrograman Berorientasi Objek

Java Database Connectivity

Wa Ode Zuhayeni Madjida, SST, M.T

Outline

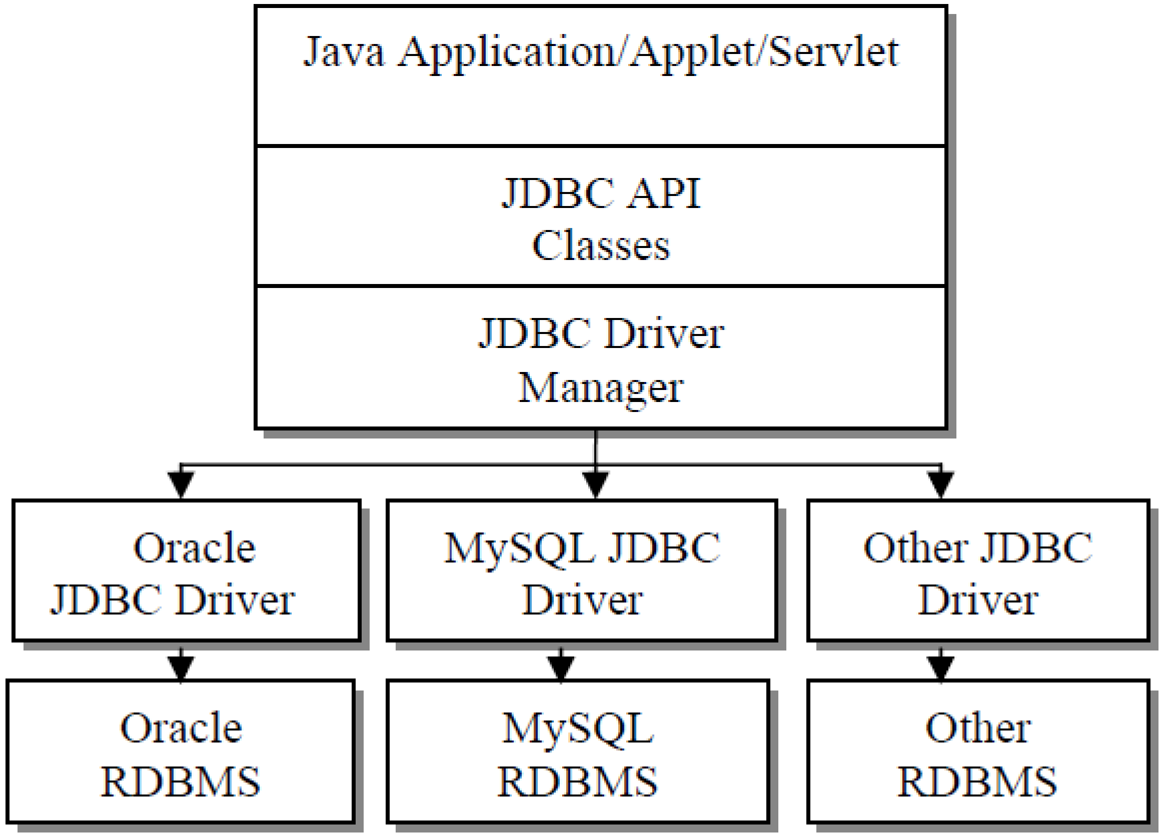
Introduction

- Data can be stored in normal text or binary files.
- As data grows, and application need to manipulate data in complex ways, storing data in such approach may not be sufficient.
- One of the best alternatives to files is to store data in databases.
- Databases allow us to manipulate and store data in an organized way, and hence transform data into meaningful information.

Java Database Connectivity (JDBC)

- Different database vendors provide different products to implement data access and manipulation mechanisms for storing and retrieving data stored from databases.
- Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database.
- JDBC APIs are independent of any vendor-specific implementations of DBMS

JDBC Architectural Stack



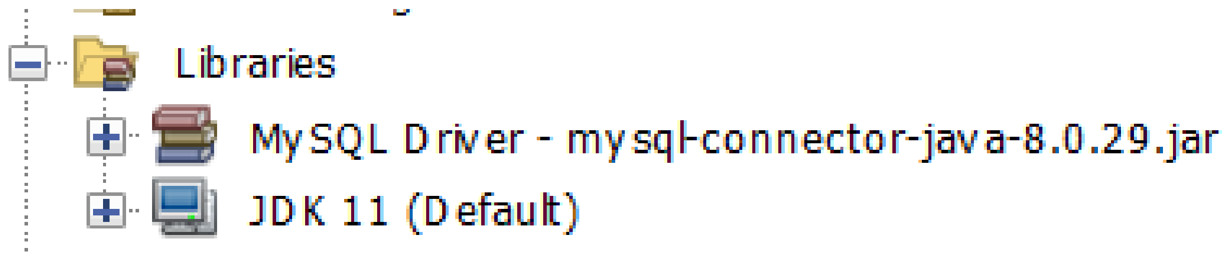
JDBC APIs

- Developing a typical database application using JDBC involve performing the following steps:
    1. Establishing a connection
        a. Loading a driver
        b. Connecting to a DBMS
    2. Data manipulation
        a. Creating a Statement object
        b. Formatting an SQL statement
        c. Executing the statement (CRUD operations)
        d. Closing the Statement and Connection

JDBC APIs – Establishing a Connection

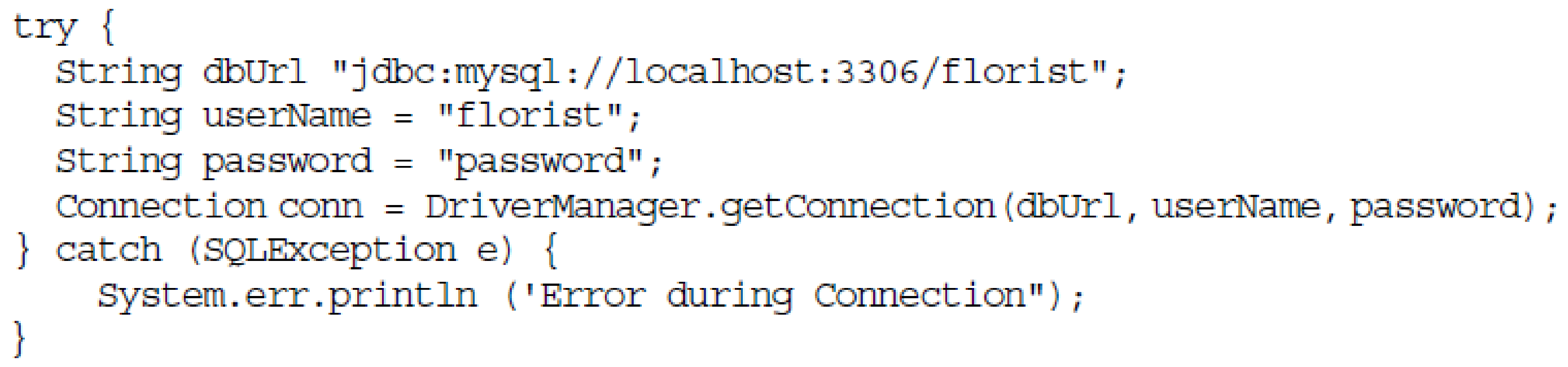
Loading the Driver

- In order to establish a connection with a DBMS, the required JDBC driver needs to be first loaded.
- For example, if you are using the MySQL database, you may copy the jar file containing the driver class (e.g.: mysql-connector-java-8.0.29-bin.jar)



JDBC APIs – Establishing a Connection

Connecting to the DBMS



```
try {
  String dbUrl "jdbc:mysql://localhost:3306/florist";
  String userName = "florist";
  String password = "password";
  Connection conn = DriverManager.getConnection(dbUrl, userName, password);
} catch (SQLException e) {
    System.err.println ('Error during Connection");
}
```

JDBC APIs – Data Manipulation

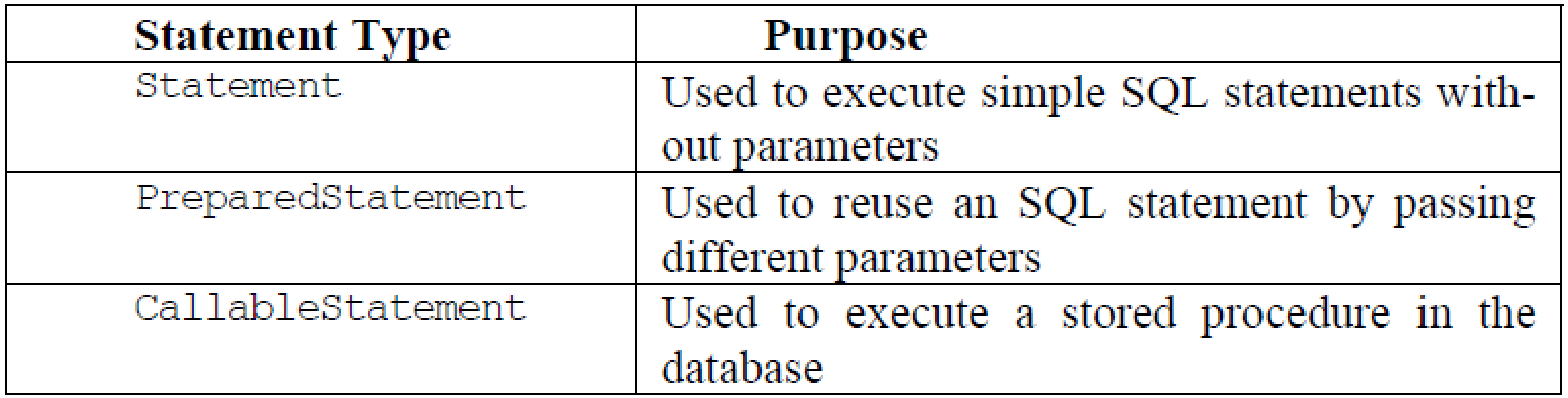
Creating Statement Objects

- After the connection to the database is established, the data can be retrieved, inserted, updated, or deleted from the database.
- This is done by using the Statement object of the java.sql package
- This object is used to send SQL statements to the database

JDBC APIs – Data Manipulation

Creating Statement Objects

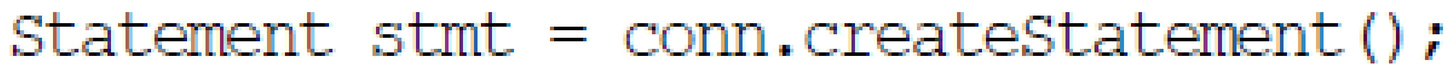
- JDBC Statement types:



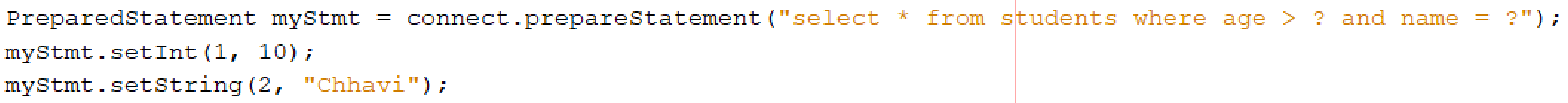
Statement Type	Purpose
`Statement`	Used to execute simple SQL statements without parameters
`PreparedStatement`	Used to reuse an SQL statement by passing different parameters
`CallableStatement`	Used to execute a stored procedure in the database

JDBC APIs – Data Manipulation

- Creating Statement objects:



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



```
PreparedStatement myStmt = connect.prepareStatement("select * from students where age > ? and name = ?");
myStmt.setInt(1, 10);
myStmt.setString(2, "Chhavi");
```

JDBC APIs – Data Manipulation

Excuting SQL Statements

- Execute SQL Statements:

```
ResultSet rSet = stmt.executeQuery("SELECT name, description " +
                                   "FROM ProductTable WHERE id='p001'");
```

- The ResultSet object that is returned is like a cursor to a Collection of results. To access the retrieved values, the cursor needs to be positioned at the first object.

```
rSet.next();
String name = rSet.getString();
String description = rSet.getString();
```

JDBC APIs – Data Manipulation

Excuting SQL Statements

- Statement execute methods:

Statement Execute Methods	Purpose	Return
executeQuery	For retrieving single result-set, e.g., using SELECT SQL statement.	ResultSet Returns a ResultSet object
executeUpdate	For modifying the database, e.g., INSERT, UPDATE, DELETE, CREATE TABLE and DROP TABLE SQL statements.	Int Returns the number of rows affected by the execution of the SQL statement
execute	Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false	booelan

Data Definition Language (DDL) with JDBC

Creating a Table

- The executeUpdate() method of the Statement class is used to specify the CREATE sql statement.
- The exact syntax can differ from one DBMS to the other.

```
import java.sql.*;
public class CreateTable {
  public static void main (String[] args){
    Connection conn = null;
    try {
      String userName = "florist";
      String password = "password";
      String url = "jdbc:mysql://localhost:3306/florist";
      Class.forName("com.mysql.jdbc.Driver").newInstance ();
      conn = DriverManager.getConnection(url, userName, password);
      Statement stmt =conn.createStatement();
      String tableName = "'florist'.'ItemTable'";
      stmt.executeUpdate ("CREATE TABLE " + tableName +
                          " ('Id' varchar(255) NOT NULL, " +
                          "'name' varchar(255) default NULL, " +
                          "'description' varchar(255) default NULL, " +
                          "'quantity' int(5), PRIMARY KEY ('Id'));");
    } catch (Exception e) {
        System.err.println (e.getMessage());
    }
  }
}
```

Data Definition Language (DDL) with JDBC

Dropping a Table

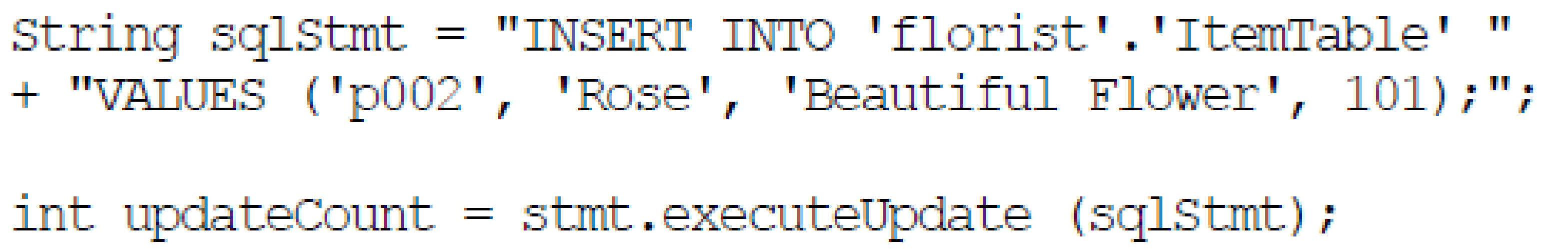
- The executeUpdate() method of the Statement class is used to specify the DROP sql statement.

```
import java.sql.*;
public class DropTable {
  public static void main (String[] args){
    Connection conn = null;
    try {
      String userName = "florist";
      String password = "password";
      String url = "jdbc:mysql://localhost:3306/florist";
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      conn = DriverManager.getConnection(url, userName, password);
      Statement stmt =conn.createStatement();
      String tableName = "'florist'.'ItemTable'";
      stmt.executeUpdate ("DROP TABLE " + tableName + ";");
    }catch (Exception e) {
       System.err.println (e.getMessage());
    }
  }
}
```

Data Manipulation Language (DML) with JDBC

Inserting Record

- Records can be inserted into database using the executeUpdate() method of the Statement class.
- The return value of the executeUpdate() method is the number of rows affected.



```
String sqlStmt = "INSERT INTO 'florist'.'ItemTable' "
+ "VALUES ('p002', 'Rose', 'Beautiful Flower', 101);";

int updateCount = stmt.executeUpdate (sqlStmt);
```

Data Manipulation Language (DML) with JDBC

Deleting Record

```
String sqlStmt = "DELETE FROM 'florist'.'ItemTable' "
                                        + "WHERE 'Id' = 'p001';";
int i = stmt.executeUpdate ( sqlStmt);
```

Data Manipulation Language (DML) with JDBC

Retrieving Record

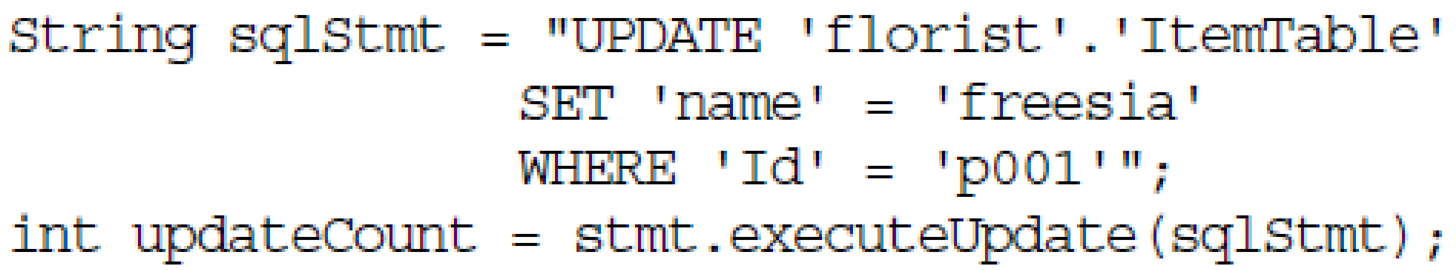
- SQL Query statements can be executed by using the executeQuery() method of the class.
- The return value is a Collection of ResultSet objects.

```
String sqlStmt = "SELECT 'Id', 'name' FROM 'florist'.'ItemTable'";
ResultSet rSet = stmt.executeQuery(sqlStmt);
while (rSet.next()) {
  System.out.println ("Id = " + rSet.getString("Id") + " Name = " +
                          rSet.getString("name"));
}
```

Data Manipulation Language (DML) with JDBC

Updating Record

- Records can be updated into database using the executeUpdate() method of the Statement class.

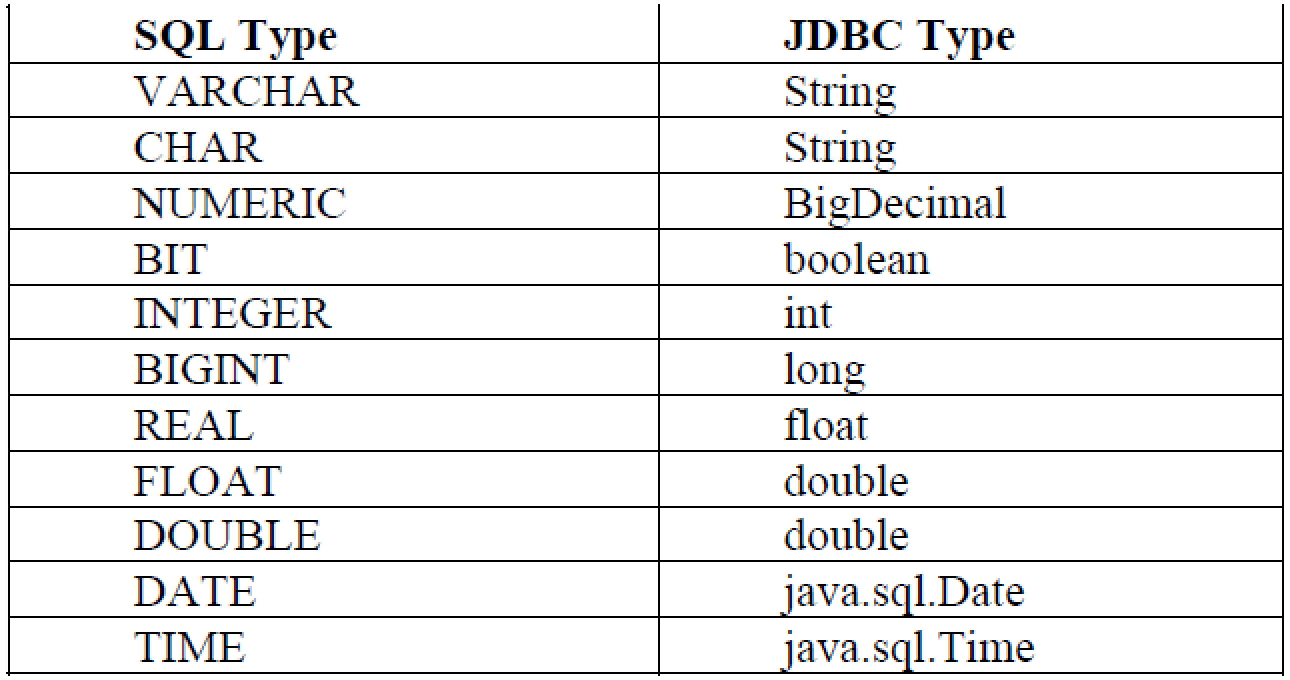


```
String sqlStmt = "UPDATE 'florist'.'ItemTable'
                  SET 'name' = 'freesia'
                  WHERE 'Id' = 'p001'";
int updateCount = stmt.executeUpdate(sqlStmt);
```

Data Manipulation Language (DML) with JDBC

Mapping of SQL Type to JDBC Type

- Corresponding to every JDBC type, there is a get method in the ResultSet class, for example, getString(), getInt(), getDate(), getFloat(), getTime(), and so on.



SQL Type	JDBC Type
VARCHAR	String
CHAR	String
NUMERIC	BigDecimal
BIT	boolean
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
DATE	java.sql.Date
TIME	java.sql.Time