Java Data Base Connectivity

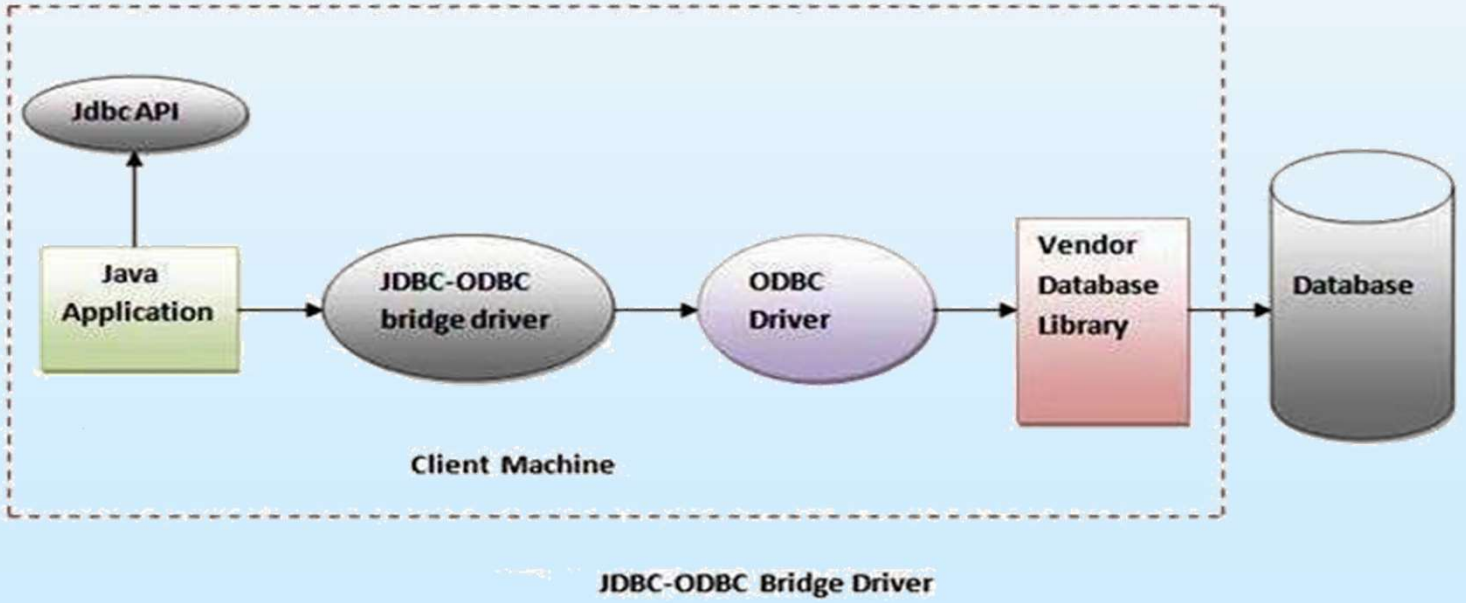
Introduction

- **Need for ODBC/JDBC**
  - Software applications are written in specific programming languages (Java, C, C# etc.)
  - Databases accept queries in database specific language (such as SQL)
  - Interface required to translate between the two
  - ODBC and JDBC are two interfaces
    - ODBC is a platform-, language- and operating system independent interface
      - provides a standard API for accessing SQL on Windows platform
    - JDBC is an API for Java
    - Java programs can use JDBC-to-ODBC bridge to talk to ODBC compliant database
      - a database driver provided by Sun Microsystems in the package sun.jdbc.odbc.JdbcOdbcDriver

Introduction

JDBC Driver

- A software component (a set of classes) that enables Java application to interact with the database
- JDBC-ODBC bridge driver uses ODBC driver to connect to the database
  - Converts JDBC method calls to ODBC function calls



JDBC-ODBC Bridge Driver

Creating JDBC Applications

Steps

1. Import the packages
2. Register the JDBC driver
3. Open a connection
4. Execute a query
5. Extract data from result set
6. Clean up the environment

Creating JDBC Applications

Import the packages

- JDBC classes are contained in two packages:
  - java.sql
    - connecting to a database directly from Java code
    - manipulating data in relational databases by running SQL statements and stored procedures
    - some advanced features such as using transactions

java.sql.Connection	java.sql.Savepoint
java.sql.Driver	java.sql.SQLException
java.sql.DriverManager	java.sql.Statement
java.sql.ResultSet	…

Creating JDBC Applications

- **Register the JDBC driver**
  - Initialize a driver to open a communication channel with the database
  - Done by using forName() method of the class Java.lang.Class
    - Returns the Class object associated with the class or interface in the given string argument

```
public static void forName (String className)
                throws ClassNotFoundException
```

Example: Class.forName ("com.mysql.jdbc.Driver");

Creating JDBC Applications

- **Open a connection**
  - DriverManager class acts as a mediator between user and interfaces
  - Use getConnection () method to create a connection object (which represents a physical connection with the database)

```
public static Connection getConnection (String url);

public static Connection getConnection (String url,
                String userName, String passWord);
```

Example:

```
Connection conn =
    DriverManager.getConnection (jdbc:mysql://localhost/emp, "", "");
```

Creating JDBC Applications

- **Execute a query**
  - Use an object of type Statement from the Connection Class for building and submitting an SQL statement to the database

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

ResultSet rs = stmt.executeQuery (
        "SELECT id, first, last, age FROM Employees");

int rows = stmt.executeUpdate ("DELETE * FROM Employees");

boolean ok = stmt.execute(Any SQL Query);
```

Creating JDBC Applications

- **Extract data from result set**
  - Used for fetching data from the database
  - Use appropriate methods to retrieve the data from the result set

```
int id = rs.getInt ("id");
int age = rs.getInt ("age");
String first = rs.getString ("first");
String last = rs.getString ("last");
```

Creating JDBC Applications

- **Clean up the environment**
  - Explicitly close all database resources, connections etc.

```
rs.close();
stmt.close();
conn.close();
```

More about JDBC

- **Connection interface**
  - A Connection is a session between Java application and database
  - Provides methods for transaction management like commit(), rollback() etc.
  - By default, connection commits the changes after executing queries

More about JDBC

- **Connection interface** (Continued …)
  - Methods
    1. public Statement createStatement ()

       Creates a statement object that can be used to execute queries
    2. public void setAutoCommit (boolean status)

       Set commit status; default is true
    3. public void commit ()

       Saves changes made since the previous commit/rollback
    4. public void rollback ()

       Drops all changes made since the previous commit/rollback
    5. public void rollback (Savepoint savePointName)

       Drops all changes made after the given Savepoint object was set

More about JDBC

- **Connection interface** (Continued …)
  - Methods (Continued …)
    6. public Savepoint setSavepoint (String savePointName)

       Defines a new savepoint and returns a Savepoint object
    7. public releaseSavepoint (Savepoint savepointName)

       Deletes a savepoint
    8. public void close ()

       Closes the connection and releases the JDBC resource

More about JDBC

Statement interface

- Provides methods to execute queries with the database

1. public ResultSet executeQuery (String sql)

   Used to execute SELECT query; Returns an object of ResultSet

2. public int executeUpdate (String sql)

   Used to execute CREATE, DROP, INSERT, UPDATE, DELETE queries;

   Returns an integer specifying the number of rows affected

3. public boolean execute (String sql)

   Used to execute queries that may return multiple results

More about JDBC

- **ResultSet interface**
  - An object of ResultSet maintains a cursor pointing to a particular row of data (initially points to before the first row)
  - By default, it can be moved forward only (TYPE_FORWARD_ONLY) and is not updateable (CONCUR_READ_ONLY)
  - Possible to make it move forward or backward by passing TYPE_SCROLL_INSENSITVE or TYPE_SCROLL_SENSITIVE in createStatement (int, int)
  - Possible to make it updateable by using CONCUR_UPDATEABLE

  Example:

```
Statement stmt = con.createStatement (
                        ResultSet.TYPE_SCROLL_INSENSITVE,
                        ResultSet.CONCUR_UPDATABLE);
```

More about JDBC

- **ResultSet interface** (Continued …)
  - Methods
    1. public boolean next ()

       Move cursor to next row
    2. public boolean previous ()

       Move cursor to previous row
    3. public boolean first ()

       Move cursor to first row
    4. public boolean last ()

       Move cursor to last row
    5. public boolean beforeFirst ()

       Move cursor to the position before the first row
    6. public boolean afterLast ()

       Move cursor to the position after the last row

More about JDBC

- **ResultSet interface** (Continued …)
  - Methods (Continued)

7. public boolean absolute (int row)

   Move cursor to specified row number

8. public boolean relative (int row)

   Move cursor to the relative row (positive/negative)

9. public int getInt (int columnIndex)

   Return data in column columnIndex in the current row as int

10. public int getInt (String columnName)

    Return data in column columnName in the current row as int

11. public String getString (int columnIndex)

    Return data in column columnIndex in the current row as string

12. public String getString (String columnName)

    Return data in column columnName in the current row as string

Connecting to Databases

- **MySql database**
  - Driver class: com.mysql.jdbc.Driver
  - Connection URL: jdbc:mysql://host:portno/dbname
    - jdbc is API, mysql is database, host is server name on which mySql is running, portno is port number and dbname is the database name
  - Username: Default is root
  - Password: Given by user at the time of installing mySql database

Example:

```
Class.forName (com.mysql.jdbc.Driver)
Connection con = DriverManager.getConnection (
        "com.mysql://localhost:3306/emp", "root", "root");
```

Connecting to Databases

- **Oracle database**
  - Driver class: oracle.jdbc.driver.OracleDriver
  - Connection URL: jdbc:oracle:thin:@localhost:portno/xe
    - jdbc is API, oracle is database, localhost is server name on which oracle is running, portno is port number (default 1521) and xe is the Oracle Service name
  - Username: Default is system
  - Password: Given by user at the time of installing oracle database

Example:

```
Class.forName (oracle.jdbc.driver.OracleDriver)
Connection con = DriverManager.getConnection (
        "jdbc:oracle:thin:@localhost:1521:xe",
        "system", "oracle");
```

Connecting to Databases

- **Access database**
  - Example:

```
String database="student.mdb";
String url="jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
Connection c=DriverManager.getConnection (url);
```

The End