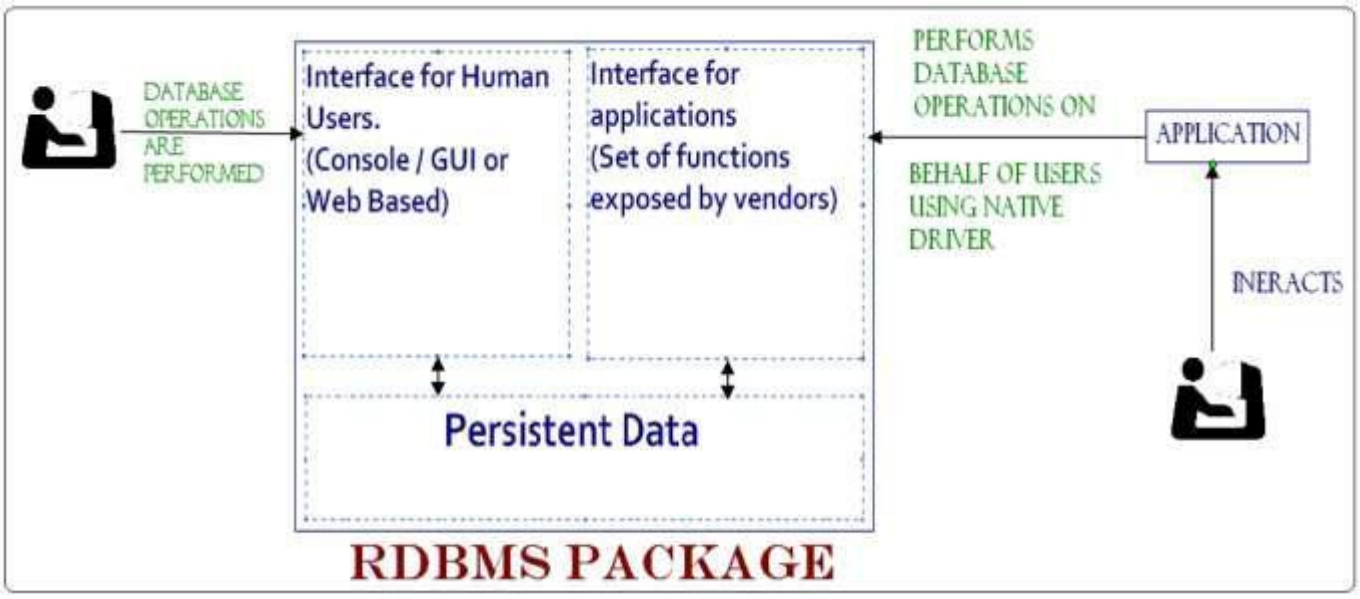
**JDBC**

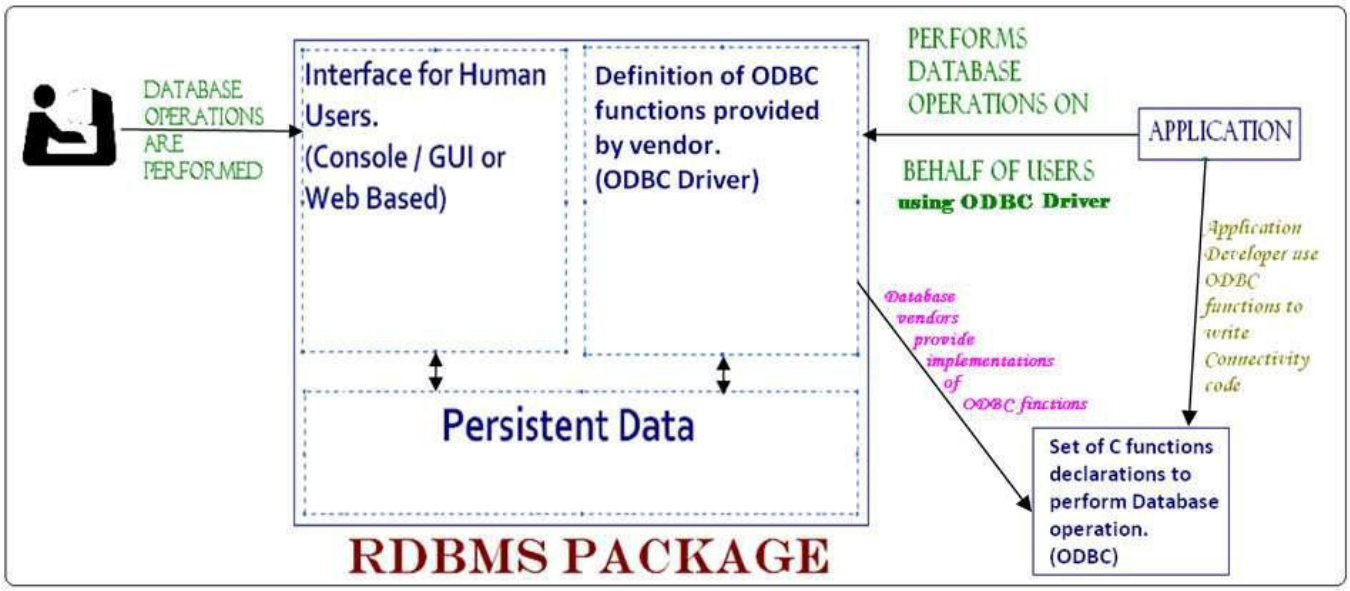
Is the java API that is used to connect Java Application to the database. This API contains classes and interfaces which are used by Java Programmers to connect Java Applications to the databases. Before understanding JDBC we must understand its need.



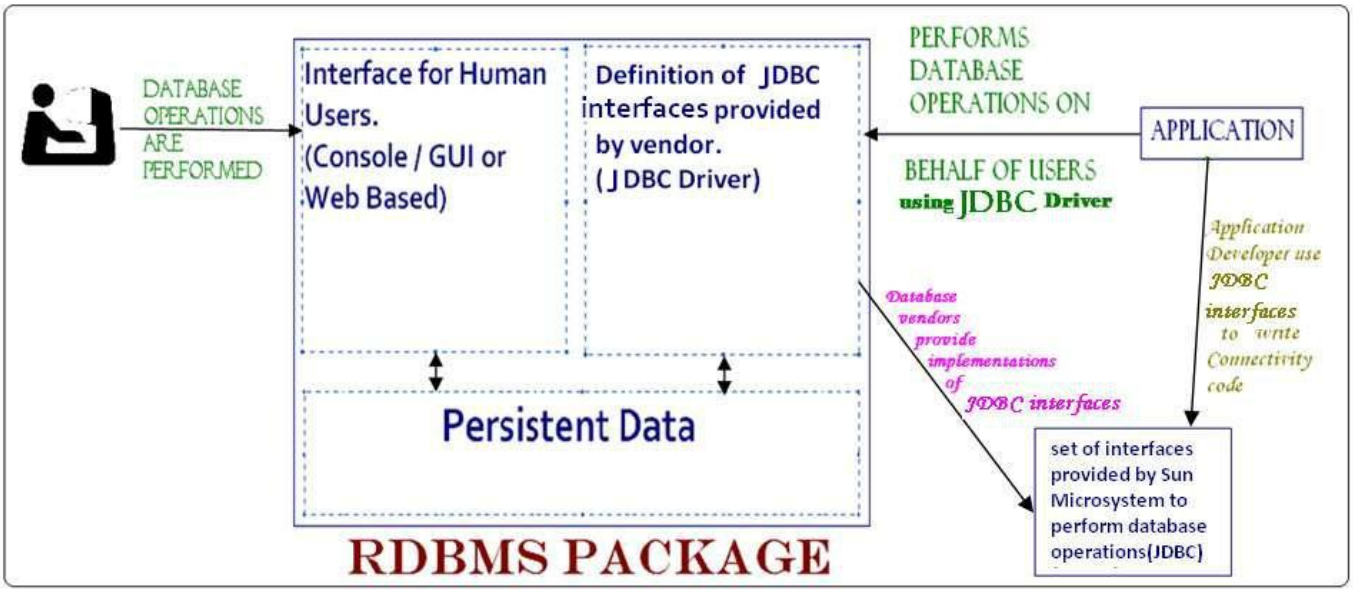

Using native driver in an application for communicating to a database had following problems:- 1. Application Programmers were to learn different API"s for different database packages.

2. Each time database package was changed in an application was required to be

modified. ODBC was introduced to solve above mentioned problems.

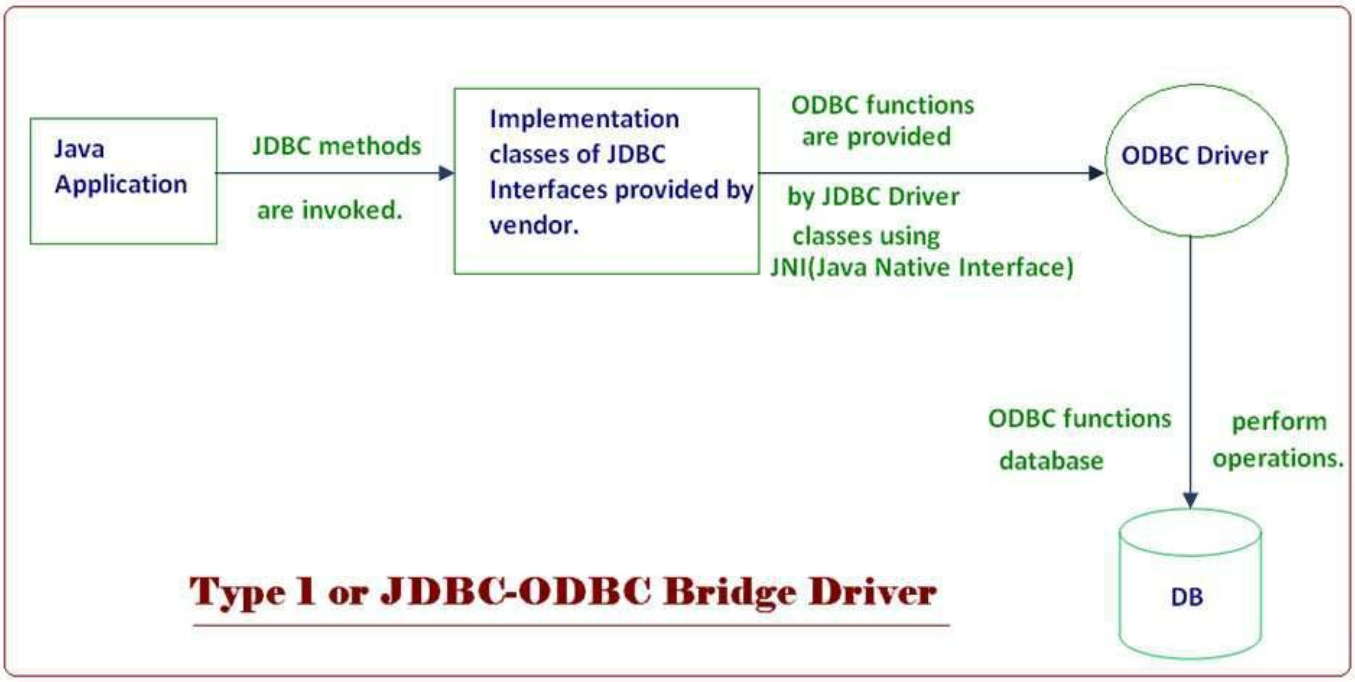



**Disadvantage of using ODBC** was that Application Programmers need to invoke C functions from their applications.




Different database vendors implemented JDBC interfaces in different ways. Depending upon the implementation we have 4 types of JDBC drivers.

1. **Type 1 or JBC-ODBC Bridge Driver** – In Type 1 JDBC driver, driver classes provided by database vendors invokes ODBC functions using JNI.




***Advantage of Type 1 driver –***

**a.** This is the simplest driver from implementation Sun Microsystem provides implementation of Type 1 JDBC driver with Core Java library.

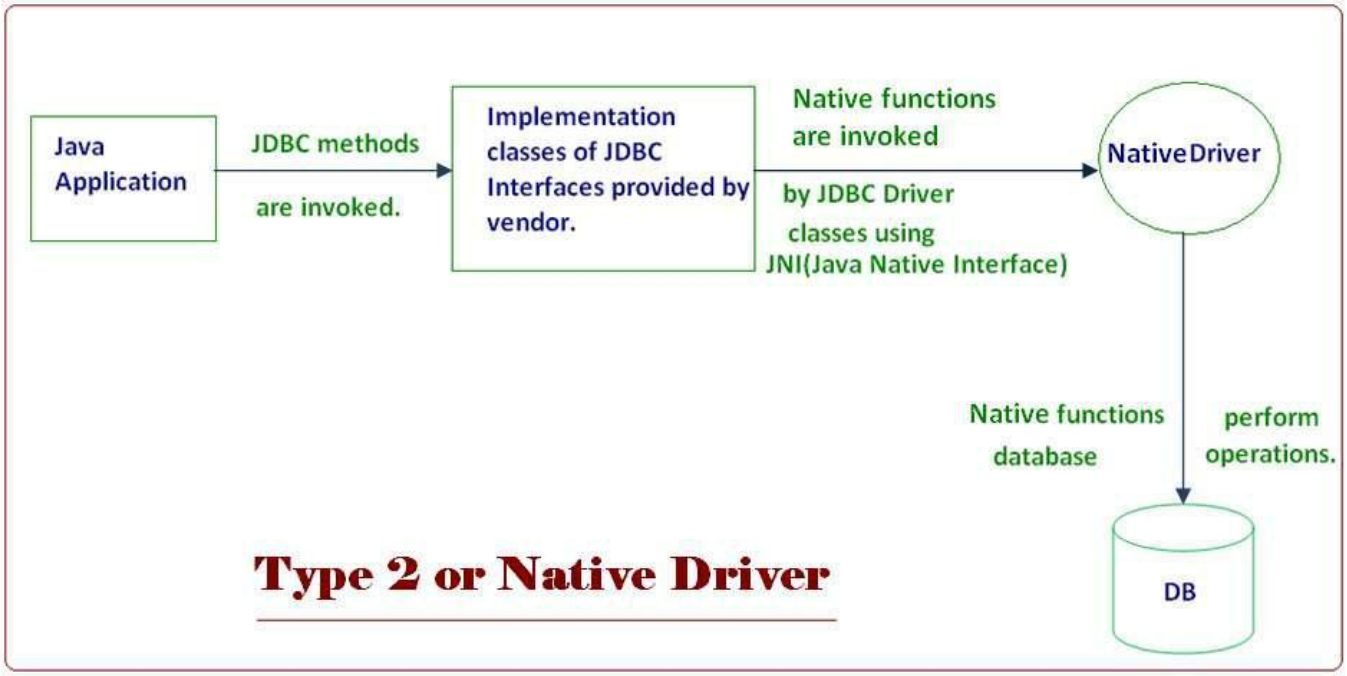
**b.** Single implementation of Type1 driver can be used with all databases.

***Disadvantage of Type 1 driver –***

**a.** Major disadvantage of Type1 JDBC Driver is the degradation of performance because each database operation requires multiple calls and conversion.

**b.** ODBC Driver needs to be installed on each machine on which application is to be executed.

### 2. Type 2 or Native Driver




In Type 2 Driver, driver classes provided by Vendor act as Java Wrapper of Native Driver. ***Advantage of Type 2 driver –***

**a.** ODBC driver is not required.

**b.** Better performance is obtained as compared to Type 1 Driver.

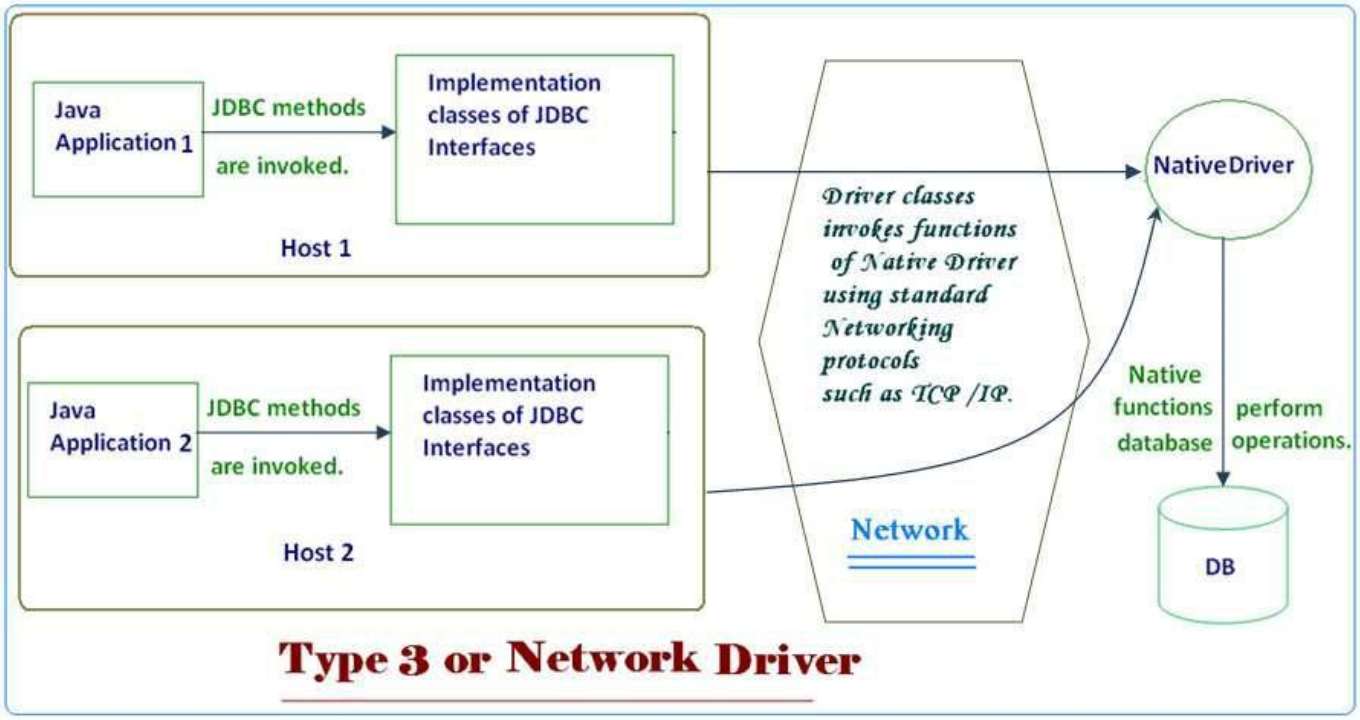
***Disadvantage of Type 2 driver –***

**a.** Native Driver needs to be installed on each machine on which application is to be executed.

**b.** For each database different driver implementation is required.

### 3. Type 3 or Network Driver.

a.



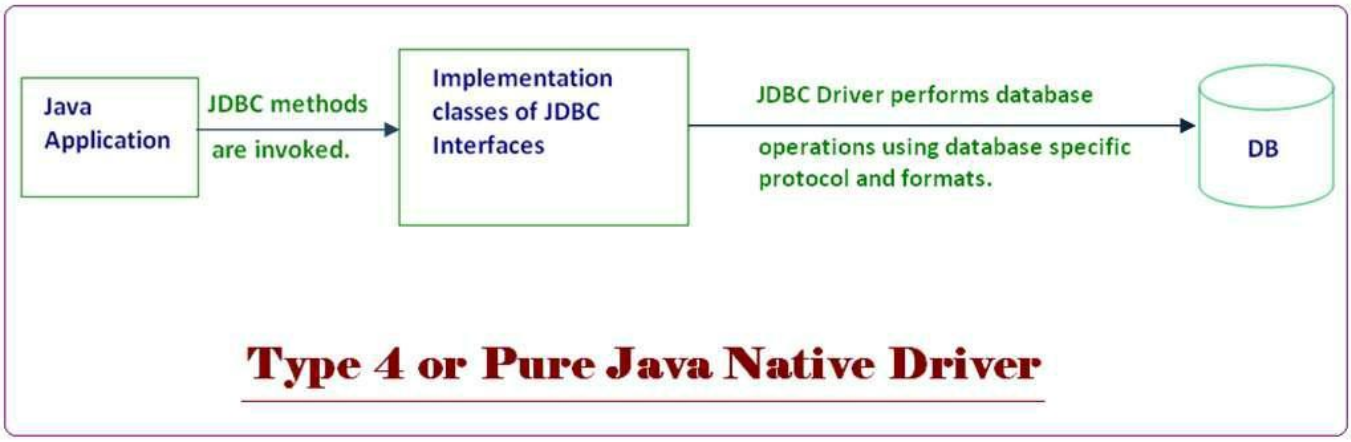

***Advantage of Type 3 driver –***

Native Driver need to be installed only on a single machine on the network.

***Disadvantage of Type 3 driver –***

Performance is degraded because of additional networking overhead.

4. **Type 4 or Pure Java Native Driver.**




***Advantage of Type 4 driver –***

**a.** ODBC & Native Driver is not required.

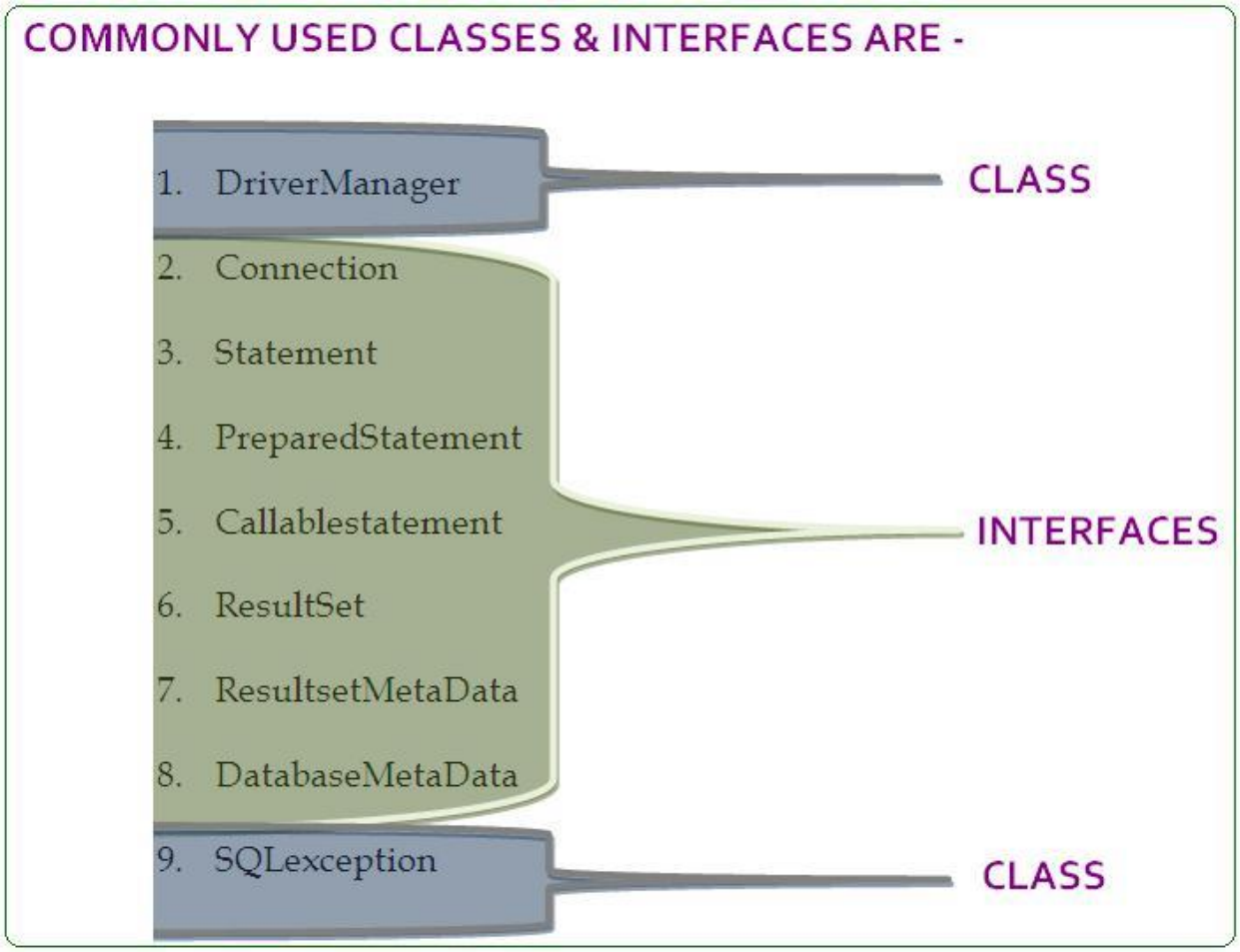
**b.** Better performance is obtained as compared to other drivers.

***Disadvantage of Type 4 driver –***

For each database different implementation of the driver is required.

**Dated : 18.07.10**

java.Sql package contains classes & interfaces of JDBC API.




- **DriverManager** is a utility class that acts as a factory of connections.
- **Connection Interface** provides the abstraction of a database connection and act as a factory of statements.
- **Statement** provides the facility of executing sql queries and act as a factory of Resultset.
- **PreparedStatement** provides the facility of executing parameterized query.
- **CallableStatement** provides the facility of invoking stored procedures & functions.
- **ResultSet** is used to store the result of a **SELECT QUERY** & act as a factory of Resultset MetaData.
- **ResultsetMetaData** provides the facility of obtaining information about the result contained in ResultSet.
- **DatabaseMetaData** is used to obtain information about the database.
- **SQLException** is the superclass of all database related exception.

Following steps are required to connect a Java application to a database:-

1. Driver class is registered with the **DriverManager**.
2. Using the **DriverManager, Connection** object is created.
3. Using the **Connection** object, **Statement** object is created.
4. Using the statement queries executed.
5. **Connection** is closed.

Implementation of steps:-

Each driver implementation provides a **MetaData** class that describes connection implementation for the driver. This class needs to be registered with the DriverManager.

All Driver classes contain registration code in their static block i.e. in order to register the driver class it simply needs to be loaded.

In case of **Type 1 driver sun.jdbc.odbc.JdbcOdbcDriver** class need to be loaded.

*Example –*

**1st Step**

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

**2nd step**

getConnection() method of DriverManager class is used to create a Connection object. *Syntax*-

**public static Connection getConnnection(String url) throws SQLException;**

**public static Connection getConnnection(String url, string username, String password) throws SQLException;**

**URL** represents the information that is used by the driver to establish a database connection. Different drivers require different information in different formats.

*In case of Type1, url has following format:-*

**"jdbc:odbc:DatasourceName"**

Let a DSN named myDb be created.

Then, connection con = DriverManager.getConnection("jdbc:odbc:myDb");

**3rd Step-**

Connection Interface provide createStatement () factory method for creating statement object. *Syntax* –

**public Statement createStatement();**

Example:-

*Statement stmt = con.createStatement();*

**4th Step –**

Statement interface provides following methods for executing queries:-

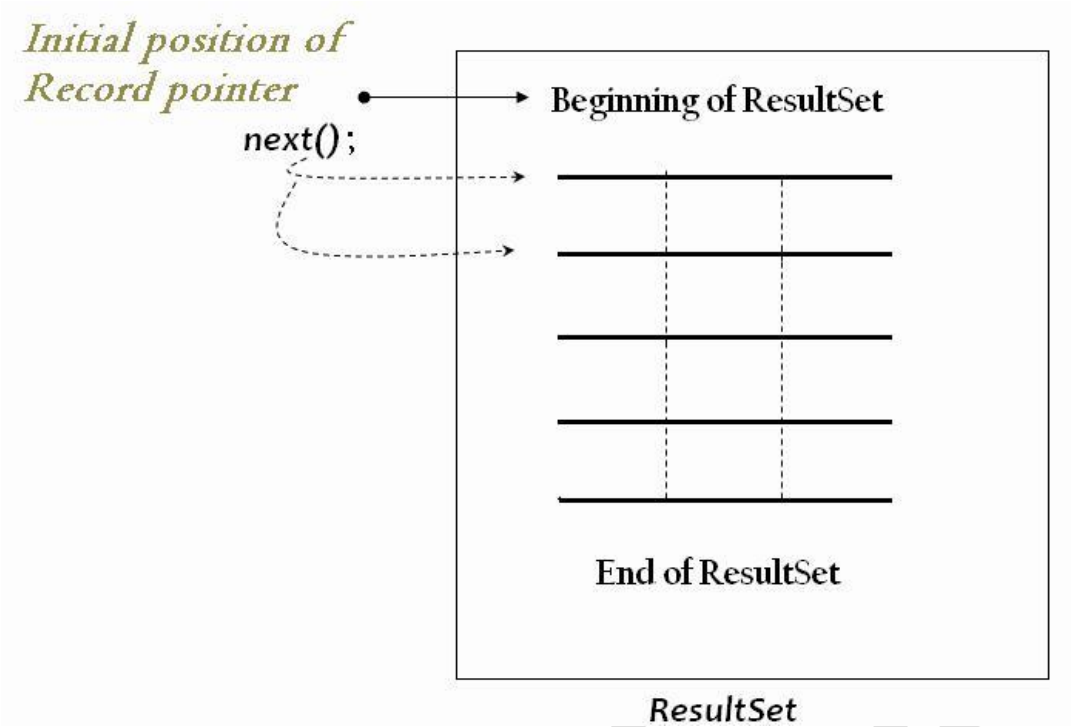
**public Resultset executeQuery(String selectQuery) throws SQLException;**

**public int executeUpdate(String NonSelectDMLQuery) throws SQLException;**

**public void execute (String NonDMLQuery) throws SQLException**

If select query is executed data is to be obtained from the ResultSet. Obtaining data from the ResultSet is a 2 step process:-

1. Record pointer is to be placed on the desired record.
2. Value of individual fields of the record is read.




**next()** method of Resultset is used to advance the record pointer by one record.

**public boolean next();**

Resultset interface provides various methods to read the value of individual fields of current record.

*General signature of these methods is:-*

public type getType(int fieldindex) throws SQLException;

*Actual Methods:-*

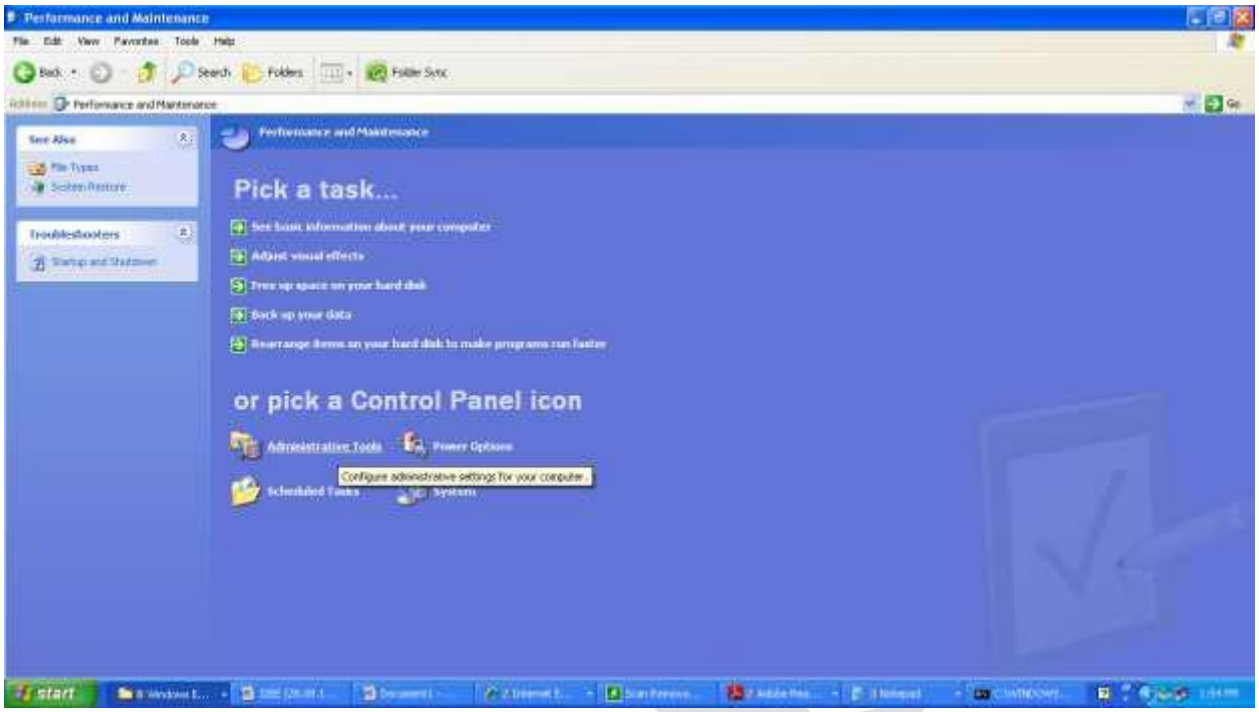
**public String getString(int index) throws SQLException;**
**public int getInt(int index) throws SQLException;**
**public float getFloat(int index) throws SQLException;**
etc.

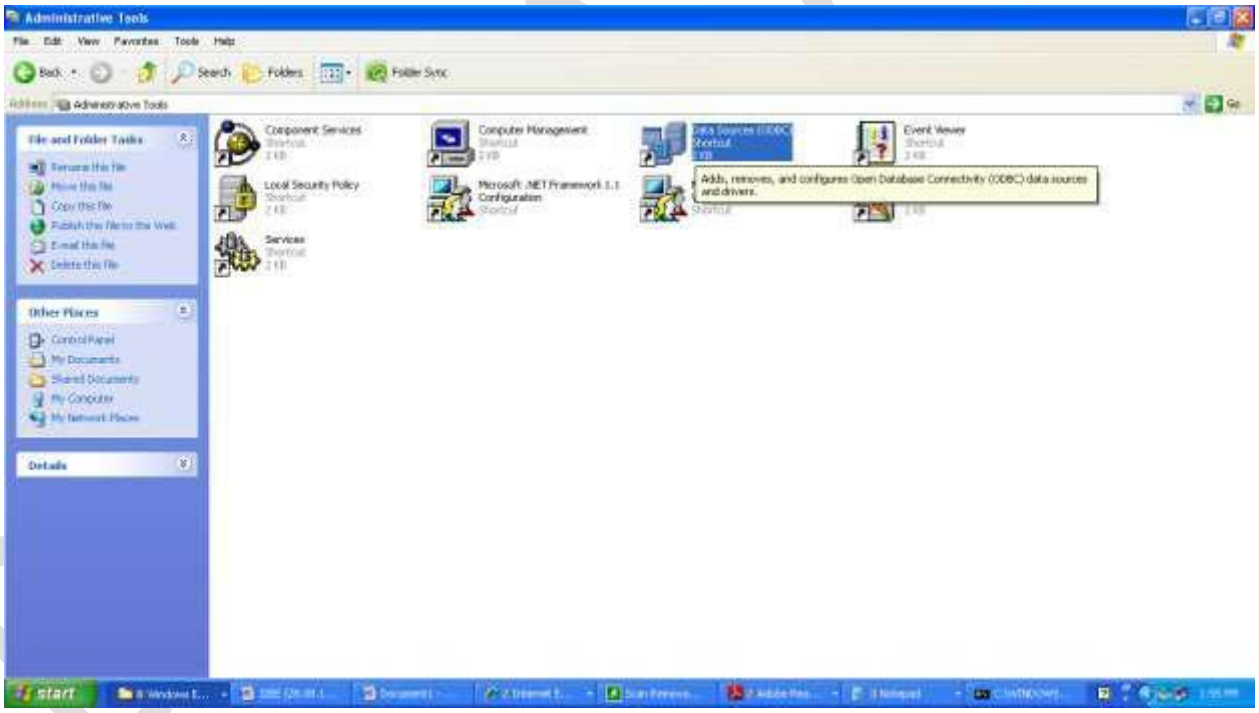
**Last step –**

Close() method of Connection interface is used to close the connection.

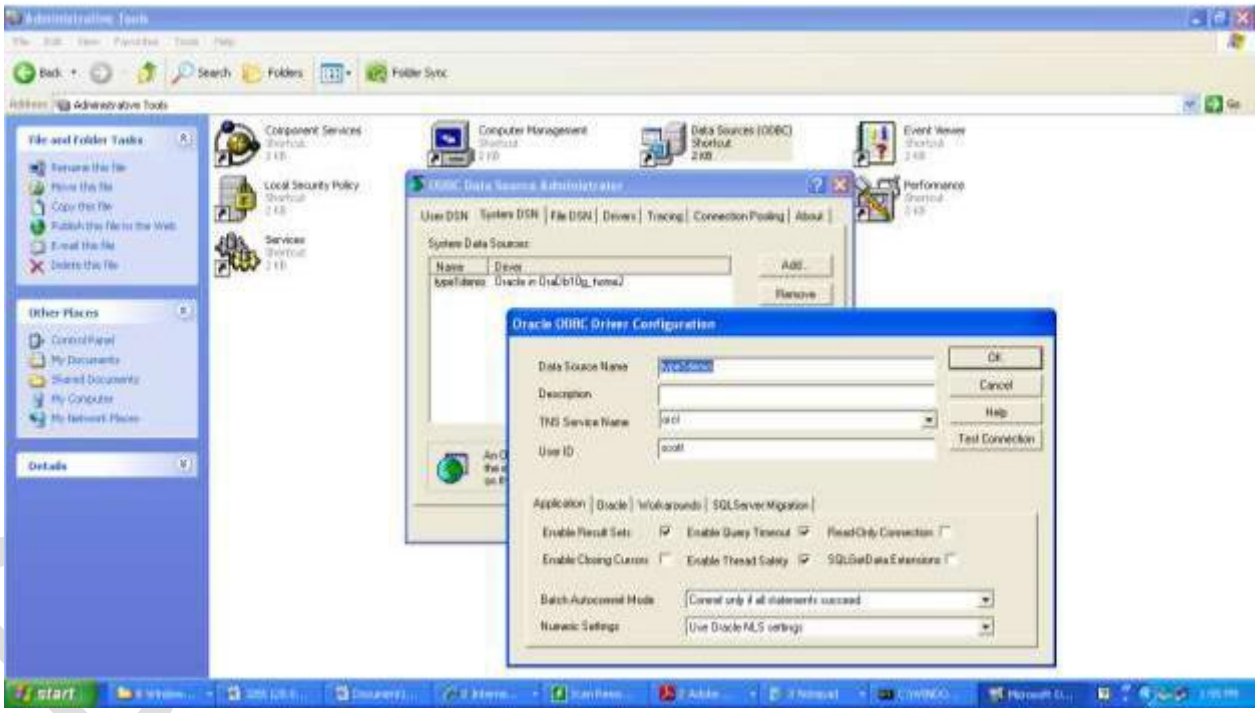
public void close() throws SQLException;

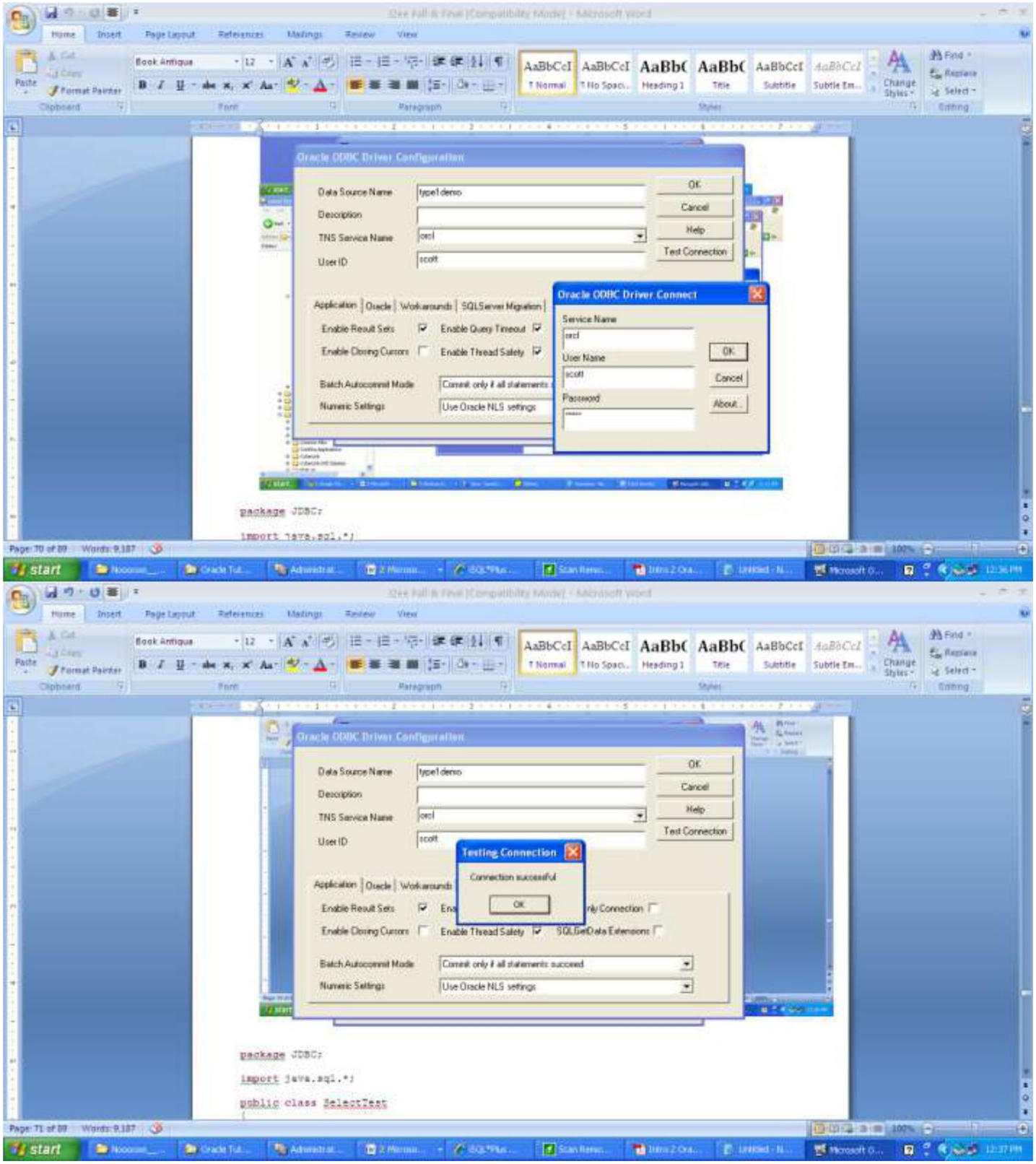
- **How to use Type1 driver**

- 
Performance and Maintenance
Pick a task...
or pick a Control Panel icon
Administrative Tools
Configure administrative settings for your computer.

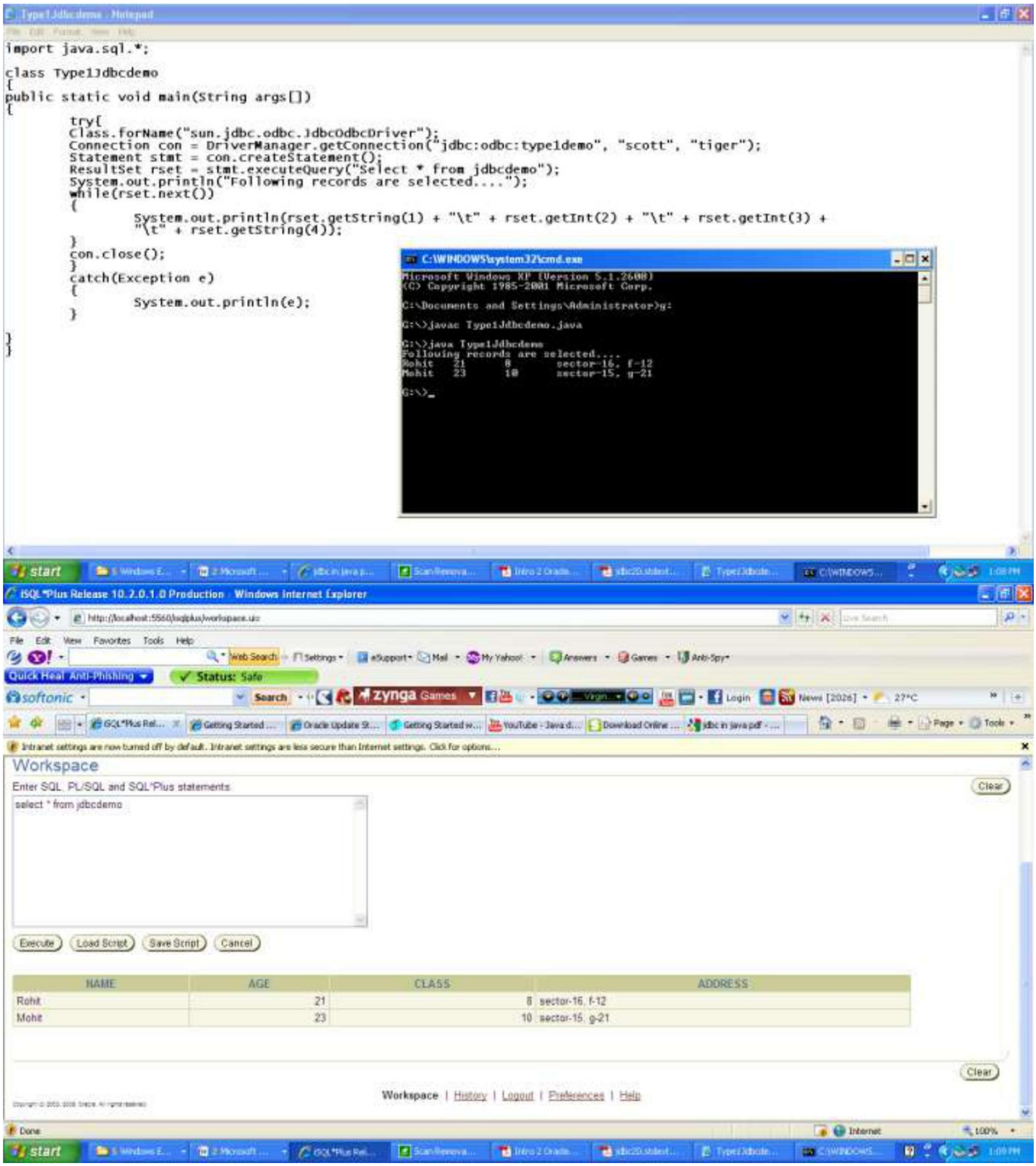

Administrative Tools
Component Services
Computer Management
Event Viewer
Local Security Policy
Services
Adds, removes, and configures Open Database Connectivity (ODBC) data sources and drivers.


Create New Data Source
Finish
Cancel


Oracle ODBC Driver Configuration
Data Source Name
Description
TNS Service Name
User ID
OK
Cancel
Help
Test Connection




Example of type1




```
package JDBC;

import java.sql.*;
```

```
public class SelectTest
{
	public static void main(String[] args)
	{
		try
		{
			Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
			Connection con = DriverManager.getConnection("jdbc:odbc:myDb", "scott",
"tiger");
			Statement stmt = con.createStatement();
			ResultSet rset = stmt.executeQuery("select * from
			emp");	System.out.println("Following records are
			selected..."); while(rset.next())
			{
				System.out.println(rset.getInt(1)+ "\t" + rset.getString(2)+
"\t" + rset.getString(3)+ "\t" + rset.getInt(4)+"\t" + rset.getString(5)+ "\t" +
rset.getInt(6)+ "\t" + rset.getInt(7)+ "\t" + rset.getInt(8));
			}
			con.close();
		}catch(Exception e)
		{
			System.out.println(e);
		}
	}
}
```

*Output –*