CHAPTER 1-JAVA DATABASE CONNECTIVITY (JDBC)

1.1 Objective

- Understand What is JDBC
- Understand JDBC Architecture
- Understand JDBC Drivers
- Understand Important Interfaces and Classes in JDBC API
- Understand JDBC steps using Type 4 Driver
- Understand Exception Handling in JDBC
- Understand Inserting Data
- Understand Reading Data
- Understand Updating Data
- Understand Deleting Data

1.2 Content

1.2.1 What is JDBC

By now you must have learned how to write business logic for an IT application using Object Oriented approach. As you belong to Java Stream, you must have used Java Language to write business logic. Also by now you must have learned how to store and retrieve information for an IT application using SQL.

While developing an IT application you will come across scenarios where you will have to store / persist data available in a program (Object states) for a longer duration, so that data can be retrieved at a later point of time if needed for processing. This is where JDBC comes handy. It helps java program to persist data into Database. JDBC is a Java API which enables Java Programs to connect to Databases, to store, search, modify and delete data. JDBC API is an interface based, industry standard for connectivity between Java programs and SQL based Databases. The JDBC standards are defined using Java Interfaces defined in java.sql package. We will explore more on these later in this chapter.

A programmer will be able to achieve 3 things using JDBC API.

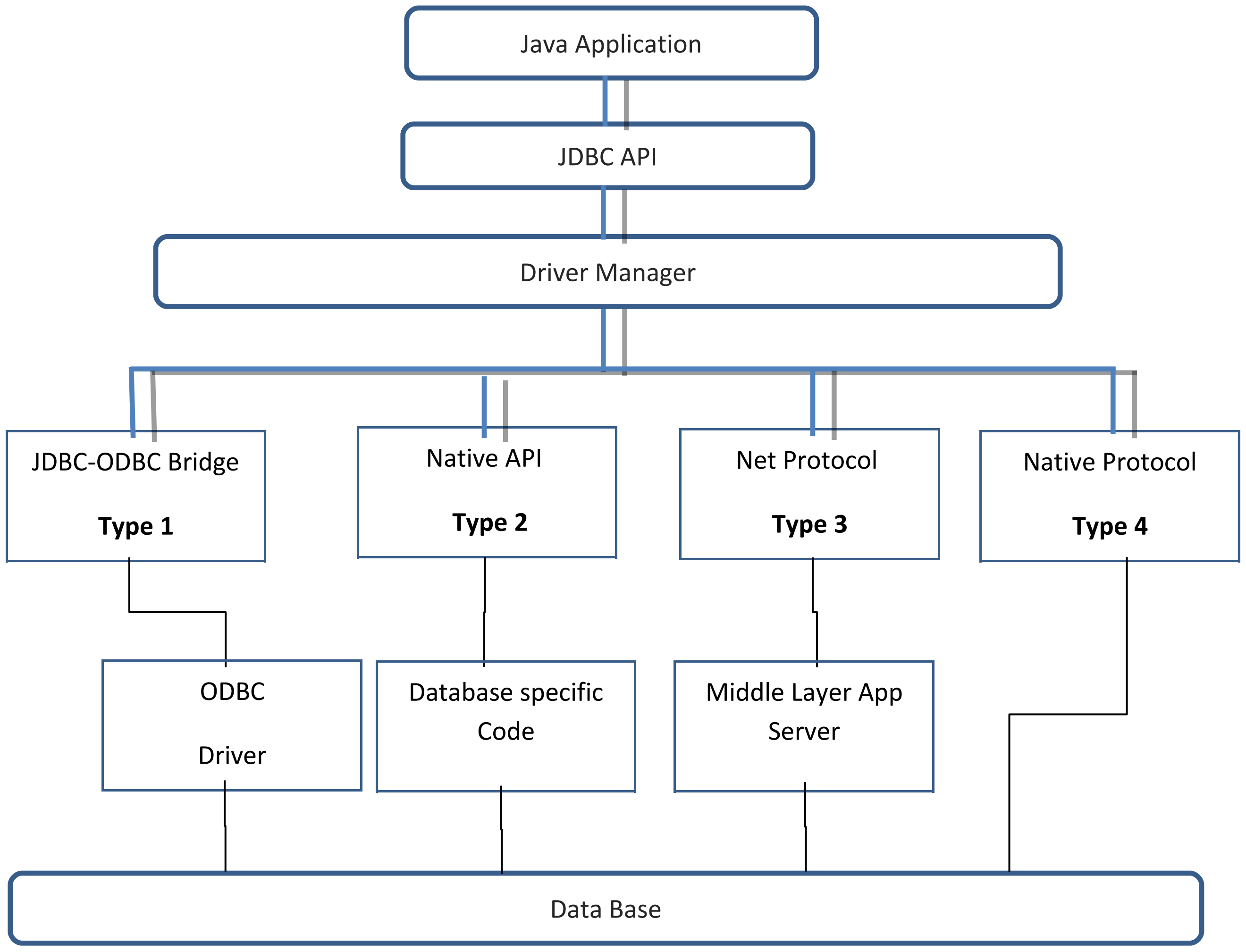
a. Establish Connection with Database
b. Send SQL commands to the Database

c. Process the results sent by the Database

We will learn all the above things in this chapter. The latest JDBC version available is JDBC 4.0.

1.2.2 JDBC Architecture

The figure below shows the JDBC architecture. Your application would use JDBC API if it wants to communicate with the Database. Using JDBC API, the application developer, will instruct DriverManager to manage driver classes. As of now we should only understand that DriverManager is a java program available in JDBC API which we will use to manage drivers.



JDBC Drivers are programs which can communicate to the database in the language, that database understands. There is lot of database available in the market, e.g, Oracle, MSSQLServer, MySql, Postgres and DB2 etc. Each database has own way of interpreting SQL. Drivers are capable of interpreting the java instruction and converting them to the database specific instructions. It also helps processing the results given by the database and converts to a format that java application can understand. There are majorly **four** types of drivers available. DriverManager is a java class capable of managing drivers for a

Java program. Let us understand them. Note any one of driver type can be used while developing the application. We will be using Type 4 driver for all our examples.

1.2.3 JDBC Drivers

Type1 Driver - **JDBC-ODBC Driver**: ODBC drivers are programs written in C/C++ which are used by .Net based applications to communicate with Databases. JDBC-ODBC type of drivers is java programmers which internally invoke ODBC programs. So your java application will instruct java programs in JDBC-ODBC bridge driver which then converts the instruction into ODBC instructions.

Disadvantage of using this type of driver is, performance is degraded, as all JDBC calls are converted to ODBC instructions and the response also goes in reverse. The second disadvantage is to use this type of driver we should have ODBC drivers installed on your system.

Type2 Driver -**The Native-API Driver:** JDBC calls from your application are directly converted into database specific calls. Type 2 Driver communicates directly with the database. Hence in the machine where your application will run would require some database specific program to be installed.

Disadvantage: Database specific programs to be installed on the machine before this type of driver can be used.

Type3 Driver - **Net-protocol:** In this case the JDBC requests coming from the application are passed through Middle tier server. The middle tier server analyzes the type database the request is for, and then converts it to the database specific instruction. By using this kind of driver the client need not worry about the type of database the instructions are sent to. The middle tier takes care converting the instructions to database specific instructions.

Disadvantage: Database specific coding is to be done in the middle ware. Database specific programs are to be installed in middle layer.

Type4 Driver - **Native-Protocol:** This type of driver converts the JDBC calls to vendor specific database instructions. Vendor Specific here means database vendors (e.g Oracle, MySQL, DB2 etc). These kind of drivers are implemented completely in Java and usually these are provided by the database vendors only. Performance of this type of driver is quite good.

Note: These drivers are supposed to be provided by the vendor only. To connect to Oracle using Type 4 driver we have to get the jar file from Oracle. To connect to MSSQL Server, using Type 4 we have to get java programs/drivers from Microsoft so on and so forth.

1.2.4 Important Interfaces and Classes in JDBC API

Let me list down few interfaces and Classes that will be used frequently during JDBC programming.

a. Driver Manager

DriverManager is a class which is available in java.sql package. This class is used to manage set of drivers. One of the important static methods that will be used is getConnection (String url, String user, String password). This method expects a url which would identify the database to connect, also userId and Password to connect to the database.

b. Connection

Connection is an interface whose implementation object represents a session with the database. As Connection is an interface we need to know that there has to be a class which implements this interface whose object will be created at Run time. This class is going to be implementation dependent and would differ from Database to Database. Well as JDBC is a standard we need not worry about the implementation class at all. As an application developer we just need to know the interfaces and methods declared with their use. We should not be bothered how they are implemented. Same is the case for rest of the interfaces below.

c. Statement

This interface represents a SQL statement that needs to be executed. It has many important methods for executing different types of SQL we will learn few of them. At run time during execution the Query to be executed at database is compiled first of all to check if the query is syntactically correct. If it is correct it executes the query to get the result from the database. In case query is syntactically incorrect the method throws an Exception at Run time.

d. **Result Set**

ResultSet interface represents the results that we get by executing a Query. The result of executing a query is like a table which can have multiple rows, each row having multiple fields. A ResultSet object also maintains a cursor pointer which points to current record. There are methods available in ResultSet to navigate through the multiple rows that we received as result of executing the query. This interface has lots of methods which can help us retrieve data of a field in a specific format.

e. **Prepared Statement**

PreparedStatement interface which stores a SQL statement which is pre compiled. Using this query is not complied every time before execution. We will see the advantage of the same in sometime.

Note: The above interfaces will be used and discussed again in detail with code samples.

1.2.5 Important Steps to Perform JDBC Operation

Let us now learn the basic steps that should be followed to perform any JDBC operation. JDBC operations that all can be performed are:

a. Store the data of a bean or an entity in the database(Create)

b. Fetch the data to a bean or a List of bean from the database(Read)

c. Modify data for an existing entity bean or an entity in the database(Update)

d. Remove the data of an existing bean or entity from the database (Delete)

No matter which ever operation we need to perform, the basic steps would still remain the same.

i. Load the Driver

ii. Create a Url String

iii. Use DriverManger to create a connection

iv. Use Connection reference to create Statement

v. Use Statement to execute Query

vi. Optionally process ResultSet

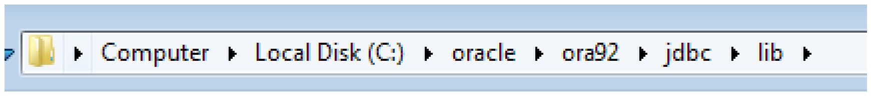
vii. Releasing resources

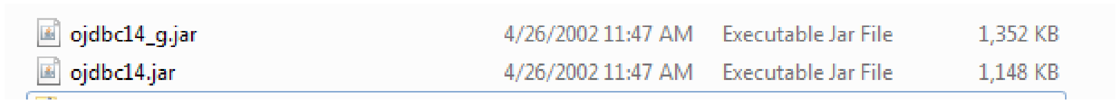
Let us explore them:

ι. **Load the Driver:** This is the first step of any JDBC program. Once the driver to be used has been decided, we have find out the java class for Driver and load it in the memory. As the driver will help communicate with the database. This can be done using the below instruction Class.forName("Name of the Driver to be loaded include package name"). This step will make sure that the driver class with the given name is loaded in the memory. The actual code for us where are using Oracle database and using Driver 4 type will be: Class.forName("oracle.jdbc.OracleDriver");. If you read the java documentation you will come to know that this method throws an Exception ClassNotFoundException. The method will throw the Exception if you have specified a class name which is not available in the classpath. This is enough to start with but if you are interested to understand what happens in the code read the below paragraph.

"Class" is a class available in "java.lang" package. Class has a static method forName. The job of forName method is to load the class code into memory in the form of an object whose name is provided as the parameter. In our case we are loading the Oracle Type 4 driver. To find out the actual driver class read the below paragraph.

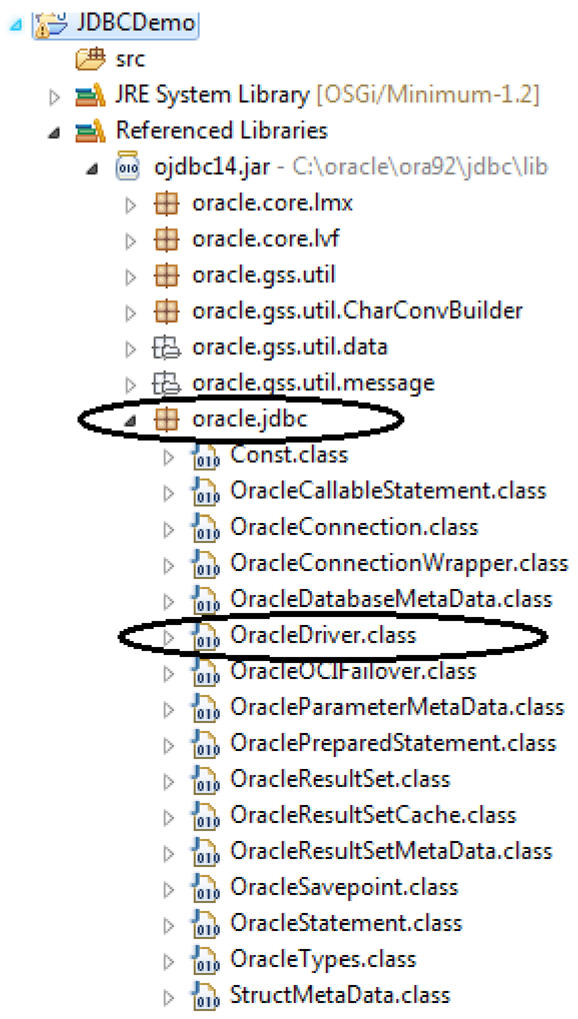
As highlighted earlier we are using Type4 driver. Hence the driver should be provided by the vendor. We are using Oracle Database to store the data. Hence Oracle should provide us the driver class. The driver class is usually provided in a jar file. The jar file that we should be using is "ojdbc14.jar". This jar file is usually found in the oracle client installation folder. Check with your faculty where Oracle client is installed in your machine. For my system it is installed in C:\Oracle folder. Usually it is found with in jdbc\lib folder within Oracle client folder. I am using Oracle 9.2 version of client installed. So the path is as given below.





ojdbc14_g.jar	4/26/2002 11:47 AM	Executable Jar File	1,352 KB
ojdbc14.jar	4/26/2002 11:47 AM	Executable Jar File	1,148 KB

You will find a lot of jar file available. ojdbc14.jar is the latest one. Hence you may use the same. This jar is supposed to be added to the class path of the project. After adding the jar file to the project you may explore the jar file. Please refer the image below.



```
JDBCDemo
  src
  JRE System Library [OSGi/Minimum-1.2]
  Referenced Libraries
    ojdbc14.jar - C:\oracle\ora92\jdbc\lib
      oracle.core.lmx
      oracle.core.lvf
      oracle.gss.util
      oracle.gss.util.CharConvBuilder
      oracle.gss.util.data
      oracle.gss.util.message
      oracle.jdbc
        Const.class
        OracleCallableStatement.class
        OracleConnection.class
        OracleConnectionWrapper.class
        OracleDatabaseMetaData.class
        OracleDriver.class
        OracleOCIFailover.class
        OracleParameterMetaData.class
        OraclePreparedStatement.class
        OracleResultSet.class
        OracleResultSetCache.class
        OracleResultSetMetaData.class
        OracleSavepoint.class
        OracleStatement.class
        OracleTypes.class
        StructMetaData.class
```

Depending upon which database you will use, find out the correct jar file and also the driver class within the jar file. This method throws ClassNotFoundException and should be handled either by a try catch block or by throwing the same.

ii. **Create the url String:** Depending on the database to be used also the type of driver to be used the url may vary. Find out the correct url before proceeding to the next step. For Oracle type4 the url will be as given below:
"jdbc:oracle:thin:@host:port:service" .

Note first green colored part is not going to change as long as we use Oracle database, but second part which is brown colored has to be changed based on which oracle instance you use. Host, port, service would depend on which database instance you are trying to connect. Get the details from your faculty.

ιιι. **Use DriverManager to create Connection:** Use the below java code to create a connection.

```
Connection con=DriverManager.getConnection(“url String”, “userId”, “password”);
```

Url String refers to the url created in step ii. Use the UserId and Password that you have used to connect to the database during SQL sessions. This method throws SQLException and should be handled by throwing or catching it.

ιϖ. **Use connection to create Statement:** Using the connection object reference created above you can create a statement object. The statement object can be used to execute Queries.

```
Statement st=Connection.createStatement();
```

This method throws SQLException and should be handled by throwing or catching it.

ϖ. Execute Query using statement:

Using statement object, you may either invoke executeQuery or executeUpdate method. Both of them expect a query in String format.

executeQuery method to be used for executing DRL’s

executeUpdate method to be used for executing DML’s, which includes insert, update and delete operations.

This method above throws SQLException and should be handled by throwing or catching it.

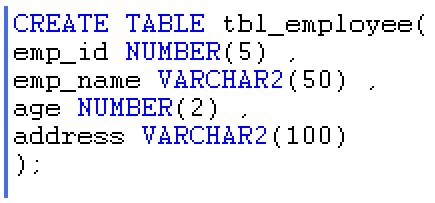
ϖι. **Optionally process the ResultSet:** Only when in the step iv. Above you run executeQuery the method always returns a resultSet Object. As highlighted earlier, it stores the result of executing a query. What needs to be done is looping through the resultset to process data of each record.

This method throws SQLException and should be handled by throwing or catching it.

ϖιι. **Releasing Resources:** You have opened the resources like connection database, Statement, ResultSet. These are shared resources. Once you are done with using these resources. Releasing resources should always be done in finally block, to make sure that resources are released even if there were exceptions.

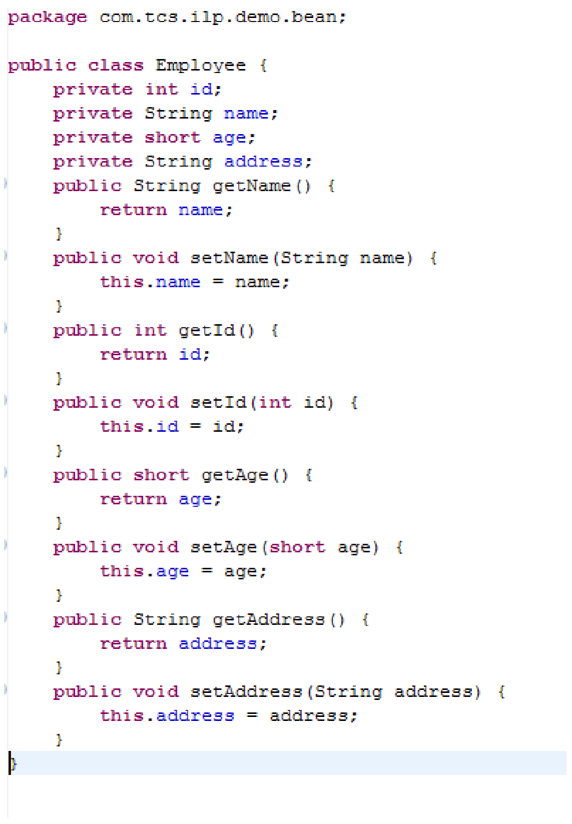
1.2.6 Inserting Data using JDBC

For the rest of the chapter we will learn how to insert, read, update and delete data from the database using JDBC. Let us refer to a sample table for all those examples. The table tbl_employee with the fields emp_id, emp_name, age, address would be a simple one to start with. The earlier chapters in Information Management module will help you understand the table structure better. You can refer back if you find the code given below complicated.



```
CREATE TABLE tbl_employee(
emp_id NUMBER(5) ,
emp_name VARCHAR2(50) ,
age NUMBER(2) ,
address VARCHAR2(100)
);
```

Also in the java layer we need to create a class Employee bean. This class should have attributes of the Employee with the above context. So the class would something like this.



```
package com.tcs.ilp.demo.bean;

public class Employee {
    private int id;
    private String name;
    private short age;
    private String address;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public short getAge() {
        return age;
    }
    public void setAge(short age) {
        this.age = age;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
}
```

Looking at the code you need to think how an Employee Object would look like, what attributes it would have, the states each employee object can have. Well these object states are to be persisted. So let us start with the very first operation insertion operation.

We have already created the bean class com.tcs.ilp.demo.Employee. Let us now create a class called as EmployeeDAO in the package com.tcs.ilp.demo.jdbc. A new kind of class name, isn't it? Yes DAO means Data Access Object. DAO class usually should have code related to accessing data from the database. Hence EmployeeDAO should have JDBC code related to DAO operations of Employee bean, like inserting Employee data, retrieving Employee data etc.

So when we talk about adding a record to the employee table, could you think about the method name and signature it should have? If you are able to do it great, else read further.

`public void addEmployee(Employee emp)` That's how it should be isn't it? The method signature tells that any one invoking addEmployee method should provide an Employee Object. The method should be able to add the employee data into the table.

The method skeleton would look something like this now.

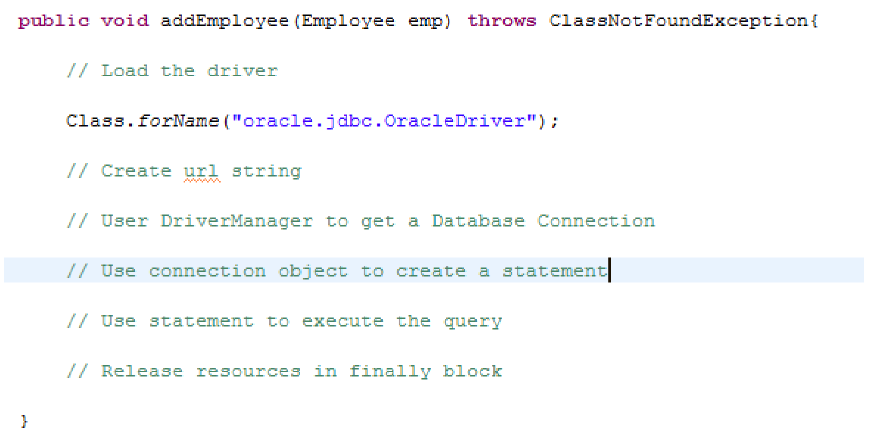
```
public void addEmployee(Employee emp){

    // Load the driver

    // Create url string

    // User DriverManager to get a Database Connection

    // Use connection object to create a statement

    // Use statement to execute the query

    // Release resources in finally block

}
```

Now your task is to learn what code is to be written in each and every step. The comments would guide you to proceed further.

First Step First, loading the driver.

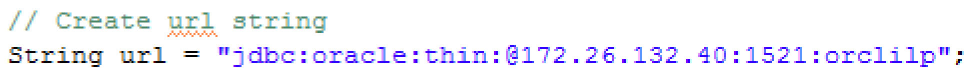
```
Class.forName("oracle.jdbc.OracleDriver");
```

As you know this method throws a compile time exception ClassNotFoundException, hence either you have to handle it by catching or you may throw it. Let us throw it. The method would look like:



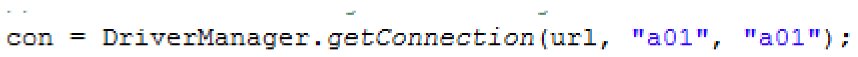
```
public void addEmployee(Employee emp) throws ClassNotFoundException{

    // Load the driver

    Class.forName("oracle.jdbc.OracleDriver");

    // Create url string

    // User DriverManager to get a Database Connection

    // Use connection object to create a statement

    // Use statement to execute the query

    // Release resources in finally block

}
```

Look at the signature of the method addEmployee, it throws ClassNotFoundException. Next step very easy defining url String. Find out your database ip, port, and service name and then form the url. For this example it is:



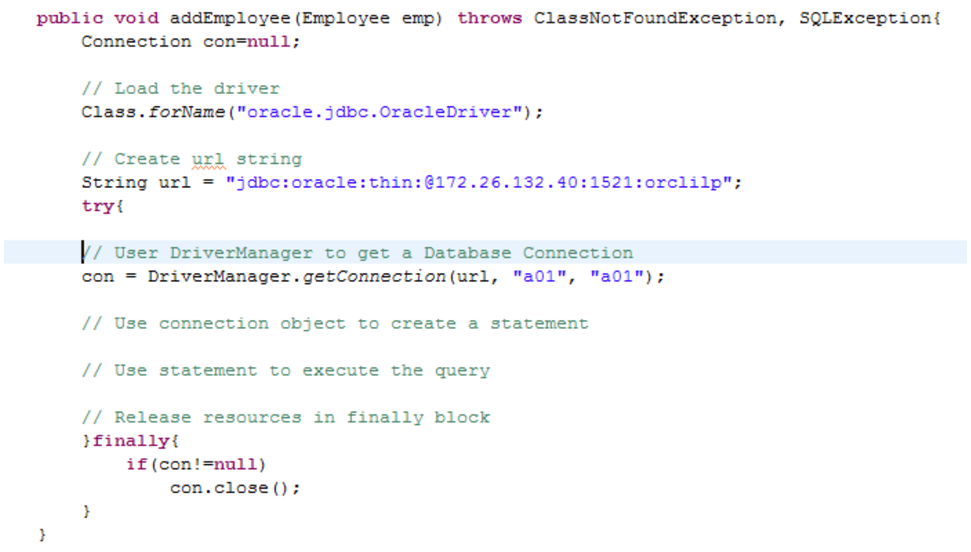
```
// Create url string
String url = "jdbc:oracle:thin:@172.26.132.40:1521:orclilp";
```

The next step is to get Database connection using DriverManager class.   Note to declare the variable con of java.sql.Connection type. You need url, userid and password. Please have it handy.



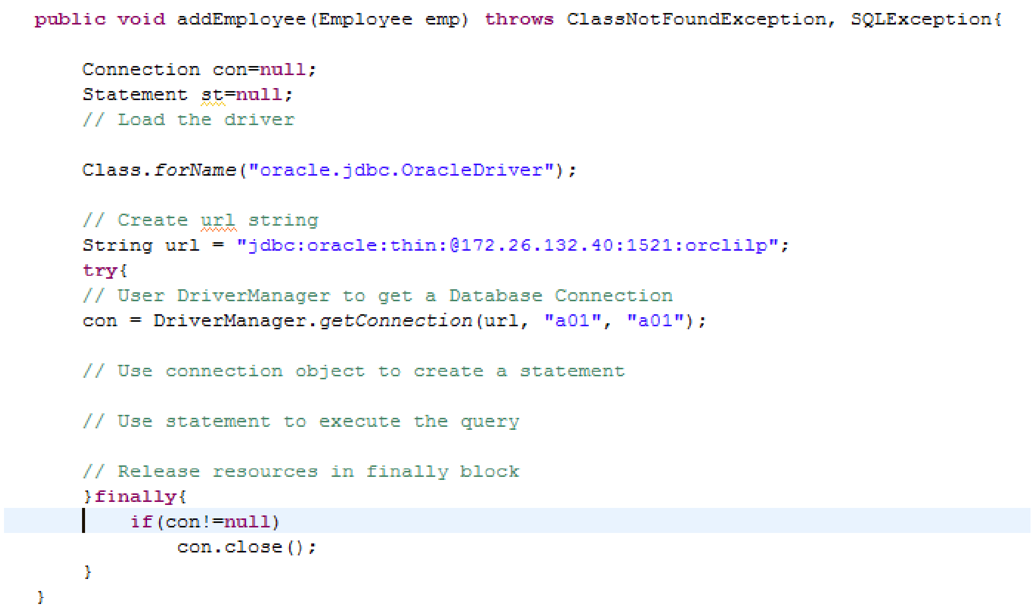
```
con = DriverManager.getConnection(url, "a01", "a01");
```

Now point to note down is the exception it would throw, the SQLException. All method invocation related to JDBC throw this exception. Either we can handle it or we will throw it. As told earlier let me throw it. The method would look like:



```
public void addEmployee(Employee emp) throws ClassNotFoundException, SQLException{
    Connection con=null;

    // Load the driver
    Class.forName("oracle.jdbc.OracleDriver");

    // Create url string
    String url = "jdbc:oracle:thin:@172.26.132.40:1521:orclilp";
    try{

    // User DriverManager to get a Database Connection
    con = DriverManager.getConnection(url, "a01", "a01");

    // Use connection object to create a statement

    // Use statement to execute the query

    // Release resources in finally block
    }finally{
        if(con!=null)
            con.close();
    }
}
```

As we have opened a database resource, the Connection, it is important to release the resource once we are done with the job. Hence let us surround the code of getting the connection with try and finally block. The code would now look like:



```
public void addEmployee(Employee emp) throws ClassNotFoundException, SQLException{

    Connection con=null;
    Statement st=null;
    // Load the driver

    Class.forName("oracle.jdbc.OracleDriver");

    // Create url string
    String url = "jdbc:oracle:thin:@172.26.132.40:1521:orclilp";
    try{
    // User DriverManager to get a Database Connection
    con = DriverManager.getConnection(url, "a01", "a01");

    // Use connection object to create a statement

    // Use statement to execute the query

    // Release resources in finally block
    }finally{
        if(con!=null)
            con.close();
    }
}
```

Going forward we will put all the jdbc related code in try block and any jdbc resource we are using should be released in the finally block.

The next step would be creating a statement using connection. The statement object too is a jdbc resource hence we should release it as well.

```java
public void addEmployee(Employee emp) throws ClassNotFoundException,
        SQLException {
    Connection con = null;
    Statement st = null;
    // Load the driver
    Class.forName("oracle.jdbc.OracleDriver");

    // Create url string
    String url = "jdbc:oracle:thin:@172.26.132.40:1521:orclilp";
    try {
        // User DriverManager to get a Database Connection
        con = DriverManager.getConnection(url, "a01", "a01");

        // Use connection object to create a statement
        st = con.createStatement();

        // Use statement to execute the query
    } finally {
        if(st!=null){
            st.close();
        }
        if (con != null)
            con.close();
    }
    // Release resources in finally block
}
```

The next statement would be executing query. The query that we want to execute is inserting a employee record in the database. So the sql code for inserting would be:

```sql
INSERT INTO TBL_EMPLOYEE VALUES ( 1, 'deepak jena', 22, 'Gandhinagar')
```

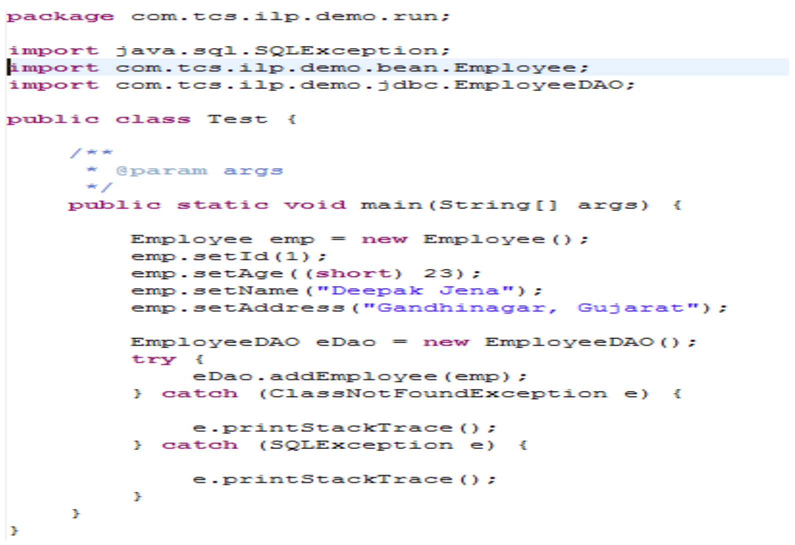
execute Update method of Statement object expects a String. Hence the above query has to be sent as a string but values of each of the field should come from variable as we cannot hardcode. As you know the method expects an employee bean whose state is to be persisted. So now the java code would be

```java
st.executeUpdate("insert into employee values(" + emp.getId()+ "'"+emp.getName()+"'" + emp.getAge()+"'" +emp.getAddress()+"'");
```

Now the final code would look like:

```java
public void addEmployee(Employee emp) throws ClassNotFoundException,
        SQLException {
    Connection con = null;
    Statement st = null;
    // Load the driver
    Class.forName("oracle.jdbc.OracleDriver");

    // Create url string
    String url = "jdbc:oracle:thin:@172.26.132.40:1521:orclilp";
    try {
        // User DriverManager to get a Database Connection
        con = DriverManager.getConnection(url, "a01", "a01");

        // Use connection object to create a statement
        st = con.createStatement();

        // Use statement to execute the query
        st.executeUpdate("insert into tbl_employee values(" + emp.getId()+ ",'"+emp.getName()+"'," + emp.getAge()+",'" +emp.getAddress()+"')");

        // Release resources in finally block
    } finally {
        if(st!=null){
            st.close();
        }
        if (con != null)
            con.close();
    }

}
```

The method to add a customer is ready to be tested. To test it, create a Test class, which would have the main method. Within the main method create an Employee bean and set the states of the bean. Then create EmployeeDAO object and then try invoking the addEmployee method EmployeeDAO object by passing the Employee Object. Run the class with main method. Check the data is inserted from your sql window. You may now try to add as many employee objects as you want.



```
package com.tcs.ilp.demo.run;

import java.sql.SQLException;
import com.tcs.ilp.demo.bean.Employee;
import com.tcs.ilp.demo.jdbc.EmployeeDAO;

public class Test {

    /**
     * @param args
     */
    public static void main(String[] args) {

        Employee emp = new Employee();
        emp.setId(1);
        emp.setAge((short) 23);
        emp.setName("Deepak Jena");
        emp.setAddress("Gandhinagar, Gujarat");

        EmployeeDAO eDao = new EmployeeDAO();
        try {
            eDao.addEmployee(emp);
        } catch (ClassNotFoundException e) {

            e.printStackTrace();
        } catch (SQLException e) {

            e.printStackTrace();
        }
    }
}
```

Now try may other scenarios given below to see how the code behaves. Try finding out the error you get when you make the following changes and run the code.

a. Give a wrong driver e.g oracle.jdbc1.OracleDriver

b. Create a wrong url string.

c. Try with wrong user id and password, in the step where you try to getConnection using DriverManager class.

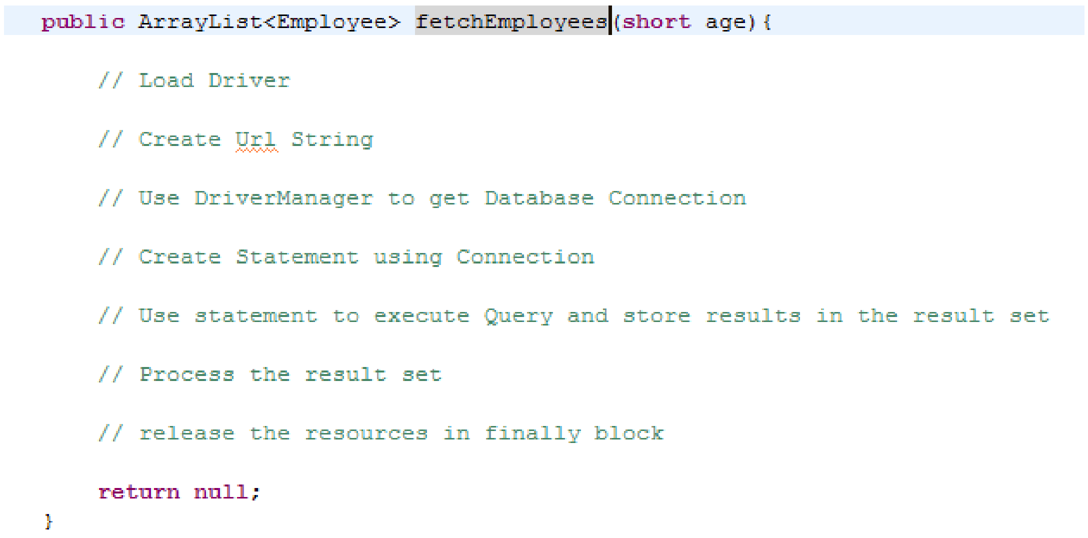
d. Try with a wrong query string, e.g; use invalid table name in the insert query

Before we proceed for the next section please add the below employee records in the database using JDBC.

EmpId#	Name	Age	Address
1	Deepak Jena	23	Gandhingar, Gujarat
2	Nisha Amoli	32	Bhubaneswar, Odhisha
3	Kiran Jain	25	Trivandram, Kerela
4	Amol Agarwal	45	Jaipur, Rajasathan
5	Khatija Banu	23	Bhubaneswar, Odhisha
6	Amar Shorf	32	Indore, MP
7	Prasanna Jena	45	Bhubaneswar, Odhisha

1.2.7 Reading Data from the Database

We are done with inserting data of few employees into the database. There would be scenarios where we need to fetch data from the database to process them. In this section of the chapter we will focus on that scenario. Well now it will be easy most of the steps that we have learned above will remain same. There are few additional steps and difference in few steps. You will understand it by looking at the below skeleton code. The objective is to find out all the employees whose age is 32.



```
public ArrayList<Employee> fetchEmployees(short age){

    // Load Driver

    // Create Url String

    // Use DriverManager to get Database Connection

    // Create Statement using Connection

    // Use statement to execute Query and store results in the result set

    // Process the result set

    // release the resources in finally block

    return null;
}
```

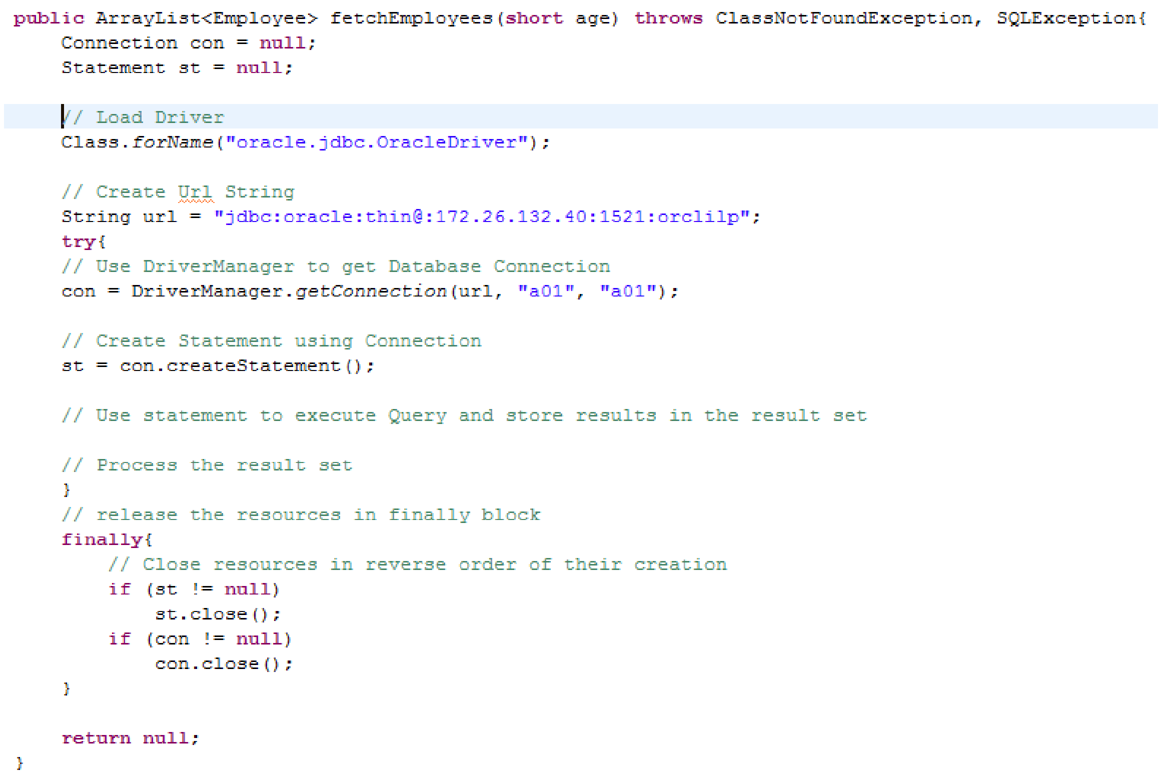
Look at the code and find the difference with the insert code. Take 2 mins studying both the skeleton code. If you were able to find the difference great, if not do not worry just read below to find the difference

a. Find out difference in signature

- Name of the method (addEmployee and fetchEmployees)
- Return type of the method (void and ArrayList<Employee>)
- Parameter passed (Eployee object and age)
- All the step till executing query is exactly the same but at execute query now we store result in ResultSet reference
- Next step is processing the ResultSet.

The point to note here is, ResultSet object is also a jdbc resource and should be released in finally block. The key thing to learn out here is accessing data from result set. So let's begin.

To start with let's have the code up to creating a statement as we saw in earlier example:



```java
public ArrayList<Employee> fetchEmployees(short age) throws ClassNotFoundException, SQLException{
    Connection con = null;
    Statement st = null;

    // Load Driver
    Class.forName("oracle.jdbc.OracleDriver");

    // Create Url String
    String url = "jdbc:oracle:thin@:172.26.132.40:1521:orclilp";
    try{
    // Use DriverManager to get Database Connection
    con = DriverManager.getConnection(url, "a01", "a01");

    // Create Statement using Connection
    st = con.createStatement();

    // Use statement to execute Query and store results in the result set

    // Process the result set
    }
    // release the resources in finally block
    finally{
        // Close resources in reverse order of their creation
        if (st != null)
            st.close();
        if (con != null)
            con.close();
    }

    return null;
}
```

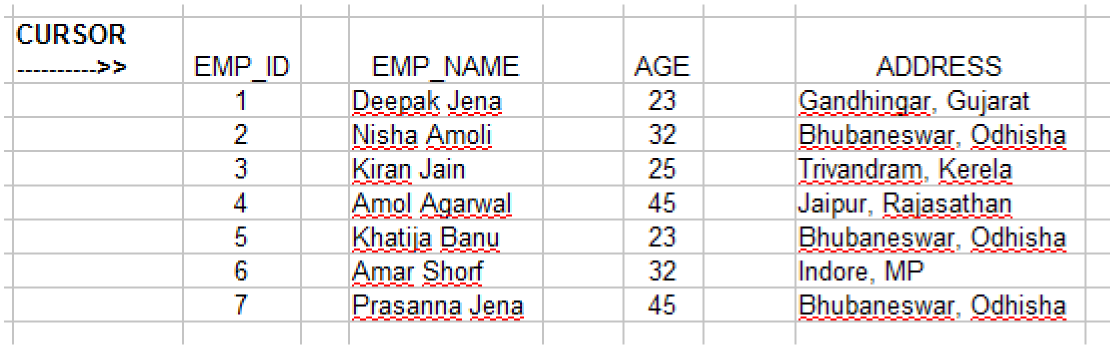
The next step would be executing the query. So what would be the query if we want to fetch all the employee data whose age is equal to 32.

```sql
SELECT EMP_ID, EMP_NAME, AGE, ADDRESS FROM TBL_EMPLOYEE WHERE AGE=32;
```

The value of age should be a variable; we should not hard code it. But as we know that there is a parameter to the method age, we should use it, isn’t it? So the query string in Java would look something like this.

```
public ArrayList<Employee> fetchEmployees(short age)
        throws ClassNotFoundException, SQLException {
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    // Load Driver
    Class.forName("oracle.jdbc.OracleDriver");
    // Create Url String
    String url = "jdbc:oracle:thin@:172.26.132.40:1521:orclilp";
    try {
        // Use DriverManager to get Database Connection
        con = DriverManager.getConnection(url, "a01", "a01");

        // Create Statement using Connection
        st = con.createStatement();

        // Use statement to execute Query and store results in the resultset
        rs = st
                .executeQuery("select EMP_ID, EMP_NAME, AGE, ADDRESS from TBL_EMPLOYEE where AGE="
                        + age);
        // Process the result set
    }
    // release the resources in finally block
    finally {
        // Close resources in reverse order of their creation
        if (rs != null)
            rs.close();
        if (st != null)
            st.close();
        if (con != null)
            con.close();
    }
    return null;
}
```

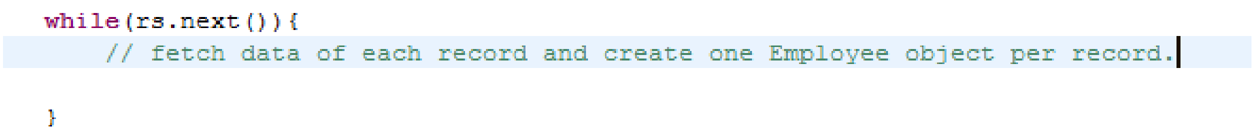
Now we have received the result from the database in a ResultSet. Visualize Result Set as a Table which has multiple rows and each row having few fields. This table is created dynamically based on the query we fired. The sample result set would look like this. If you have already inserted the data as instructed earlier, the resultset would look like this. Remember you will not be able to see the resultset. This is just a conceptual diagram for you to understand.



CURSOR ---------->>	EMP_ID	EMP_NAME	AGE	ADDRESS
	1	Deepak Jena	23	Gandhingar, Gujarat
	2	Nisha Amoli	32	Bhubaneswar, Odhisha
	3	Kiran Jain	25	Trivandram, Kerela
	4	Amol Agarwal	45	Jaipur, Rajasathan
	5	Khatija Banu	23	Bhubaneswar, Odhisha
	6	Amar Shorf	32	Indore, MP
	7	Prasanna Jena	45	Bhubaneswar, Odhisha

Do you see an arrow at the leftmost top corner? It is called as CURSOR. A cursor is nothing but a pointer which will point to a record within the resultset. When a resultset object is created, cursor always points to a position which is earlier to the first record. There are methods available in the resultset which will help us move the cursor record by record. We will explore one of them. When the cursor is pointing to any record we can fetch the data of each field in the record. Let’s see with the code. The method to use is rs.next(). This method when invoked moves the cursor to the next record and returns true or false. It returns true if the cursor moved to the next record. But if the cursor has already reached the last record and we invoke the method next(), it would return false value, to inform that there no more records available to traverse.

We will create a while loop to move from one record to the next of the resultset.



```
while(rs.next()){
    // fetch data of each record and create one Employee object per record.

}
```

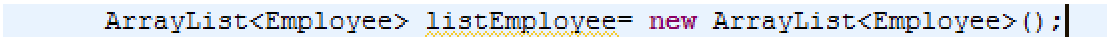
If we write the above code, the while loop is going to run for those many number of times, as the number of records present in the resultset object. For each record we need to fetch data from the fields and create an employee object. To achieve that, we need to explore other methods available in resultset object.

When a resultset is pointing to a record, we can use rs.getString(String fieldName), rs.getInteger(String fieldName) etc. So in our example there are four fields, first field emp_id is integer, second field emp_name which is string. These methods in resultset help us fetch the data from the field of a record and store it in appropriate java data type. Like data in emp_name field should be stored in String variable hence we should use rs.getString() method. Data available in age field should be stored in a short variable hence we should use rs.getShort() etc. Interesting to note, the return type of getString() method is String, the returntype of getShort() method is short and so on and so forth. Each of the method getString(), getShort(), getInt() expects a string parameter. The string parameter is the name of the field whose data is to be fetched.

Hence to fetch data of emp_name field we should use the following signature.

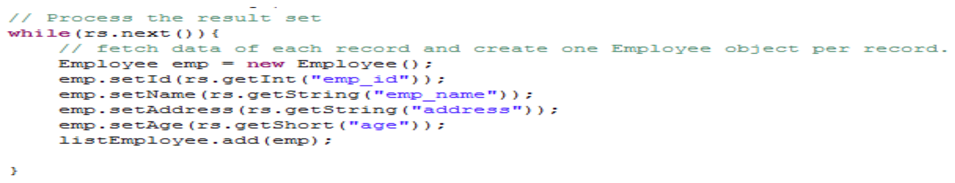
rs.getString("emp_name");

Now create an ArrayList to store all the Employee objects that would be created.



```
    ArrayList<Employee> listEmployee= new ArrayList<Employee>();
```

listEmployee will store the reference of all the Employee objects created.

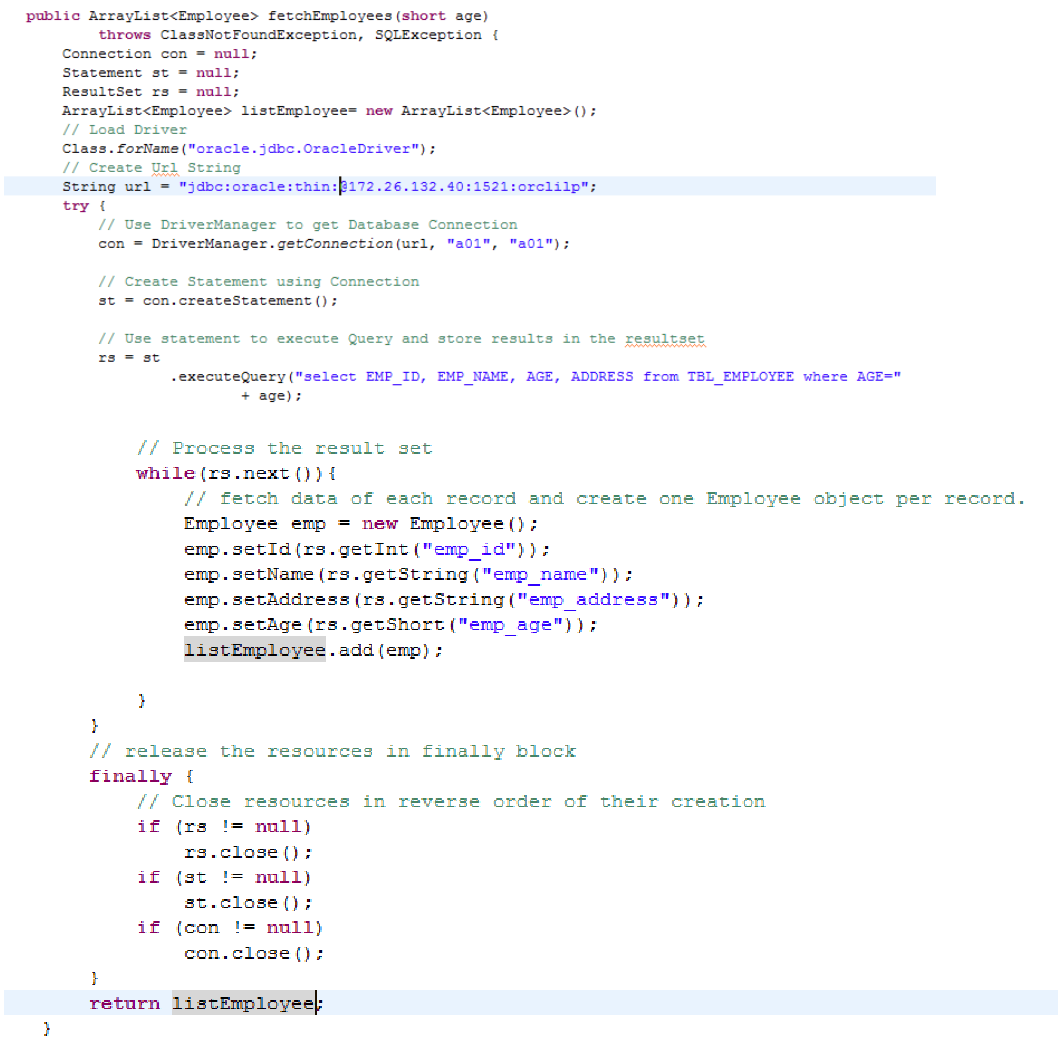


```
// Process the result set
while(rs.next()){
    // fetch data of each record and create one Employee object per record.
    Employee emp = new Employee();
    emp.setId(rs.getInt("emp_id"));
    emp.setName(rs.getString("emp_name"));
    emp.setAddress(rs.getString("address"));
    emp.setAge(rs.getShort("age"));
    listEmployee.add(emp);

}
```

The above code loops over all the records in the result set in the while loop.

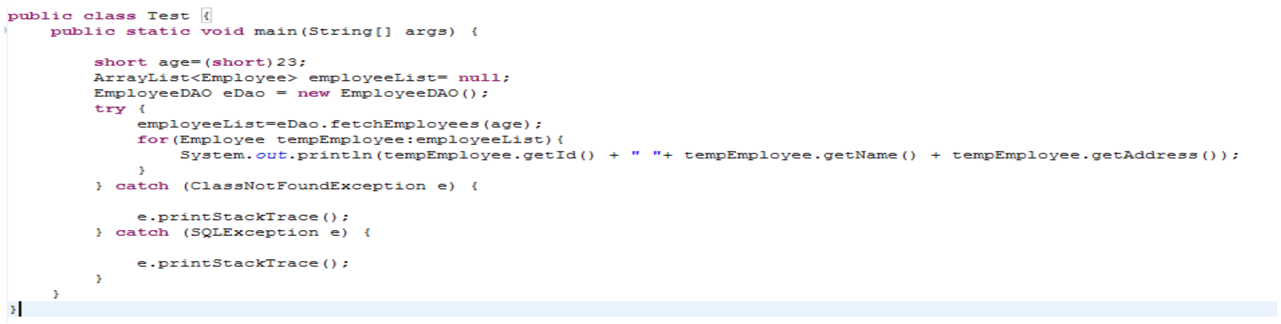
In each iteration a new Employee Object is created. The data fetched from emp_id field which is of integer type is stored in Employee objects id field, Data fetched from emp_name field which is of String type is stored in Employee objects name field. At the end, the employee object reference is added to

ArrayList employee. And finally we can return the ArrayList which contains all the employees which were selected by executing the query. Let's see how the final code would look like:



```
public ArrayList<Employee> fetchEmployees(short age)
        throws ClassNotFoundException, SQLException {
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    ArrayList<Employee> listEmployee= new ArrayList<Employee>();
    // Load Driver
    Class.forName("oracle.jdbc.OracleDriver");
    // Create Url String
    String url = "jdbc:oracle:thin:@172.26.132.40:1521:orclilp";
    try {
        // Use DriverManager to get Database Connection
        con = DriverManager.getConnection(url, "a01", "a01");

        // Create Statement using Connection
        st = con.createStatement();

        // Use statement to execute Query and store results in the resultset
        rs = st
                .executeQuery("select EMP_ID, EMP_NAME, AGE, ADDRESS from TBL_EMPLOYEE where AGE="
                        + age);

        // Process the result set
        while(rs.next()){
            // fetch data of each record and create one Employee object per record.
            Employee emp = new Employee();
            emp.setId(rs.getInt("emp_id"));
            emp.setName(rs.getString("emp_name"));
            emp.setAddress(rs.getString("emp_address"));
            emp.setAge(rs.getShort("emp_age"));
            listEmployee.add(emp);

        }
    }
    // release the resources in finally block
    finally {
        // Close resources in reverse order of their creation
        if (rs != null)
            rs.close();
        if (st != null)
            st.close();
        if (con != null)
            con.close();
    }
    return listEmployee;
}
```

We just created the DAO method to fetch employee data from the database. This can be tested. To test it, as earlier we have to create a Test class with main method. Within the main method create an object of EmployeeDAO and call fetchEmployees method. As per the signature the fetchEmplyee method expects age value which should be of short type. The test class would look something like this.



```
public class Test {
    public static void main(String[] args) {

        short age=(short)23;
        ArrayList<Employee> employeeList= null;
        EmployeeDAO eDao = new EmployeeDAO();
        try {
            employeeList=eDao.fetchEmployees(age);
            for(Employee tempEmployee:employeeList){
                System.out.println(tempEmployee.getId() + " "+ tempEmployee.getName() + tempEmployee.getAddress());
            }
        } catch (ClassNotFoundException e) {

            e.printStackTrace();
        } catch (SQLException e) {

            e.printStackTrace();
        }
    }
}
```

Now try may other scenarios given below to see how the code behaves. Try finding out the error you get when you make the following changes and run the code.

a. Give a wrong driver e.g oracle.jdbc1.OracleDriver

b. Create a wrong url string.

c. Try with wrong user id and password while creating getting the connection using DriverManager

d. Try with a wrong query string, i.e, try executing an invalid query string.

Objective is to study the compilation and runtime exceptions that you get.

1.2.8 Update existing Data stored

We will use and learn a different approach to update data in the database. We will use PreparedStatement instead of using Statement. Updating data in the database can still be achieved using Statement but we are doing it to learn how and why to use PreparedStatement. If we use Statement to update, all above steps would remain same except that while executing the statement we will use exectuteUpdate method and pass an update query string to the method.

But it is important to understand the difference between using of Statement and PreparedStatement.

Let us understand what had happened in case of Statement. We used statement for both the above cases. Let us analyze what happens in case of searching employees. Let's take a sample example. We want all the employees where EMP_NAME is matching a given name.

```
    rs = st.executeQuery("select EMP_ID, EMP_NAME, AGE, ADDRESS from TBL_EMPLOYEE where EMP_NAME='"+ name +"'");
```

Based on the value of the variable name, the query will be dynamically generated. Value of name could be Manish or Deepak or any other name. When statement is used the query generated dynamically is first compiled at database level to check the syntax of the query. For every execution the query is checked for syntax before execution. Due to this feature there is a possibility of hackers to do SQL Injection. SQL Injection is a technique in which hackers would try to send the value of the variable in such a way that a new query is formed.

As query gets complied every time before execution any value sent becomes part of the query. This is the biggest problem of statements. Hackers try to exploit this feature and try to send SQL code which will provide more information than intended.

For the above code if someone sends the value of name as Deepak, then the query after compilation would turn out to be

```
    String query= "select EMP_ID, EMP_NAME, AGE, ADDRESS from TBL_EMPLOYEE where EMP_NAME='deepak'";
```

But if a hacker is trying to hack , he would try to give different values than what a normal human being is going to give. He will also try to give the parameter value such that a new query would be created.

Just to take an example, he would pass the below value: `' OR '1'='1`

If you analyze properly the query that will be compiled finally will be:

```
"select EMP_ID, EMP_NAME, AGE, ADDRESS from TBL_EMPLOYEE where EMP_NAME='' OR '1' = '1'";
```

You see the hacker was able to inject some additional query which was not intended. This is what is known as SQL Injection. To avoid this we can use prepared statement.

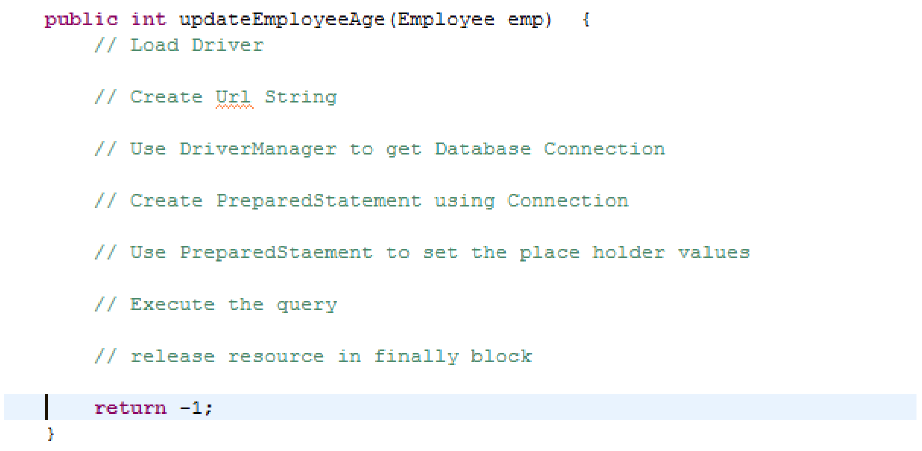
If we use prepared statement the query is already precompile and the values passed on later will only become part of the value. Hence the query will never get changed. Whatever query has been provided by the programmer, the same query is going to be executed. But any value passed by the hacker will only become part of the value expected.

A sample query that will be compiled before execution in case of prepared statement would look like this.

```
"select EMP_ID, EMP_NAME, AGE, ADDRESS from TBL_EMPLOYEE where EMP_NAME=?";
```

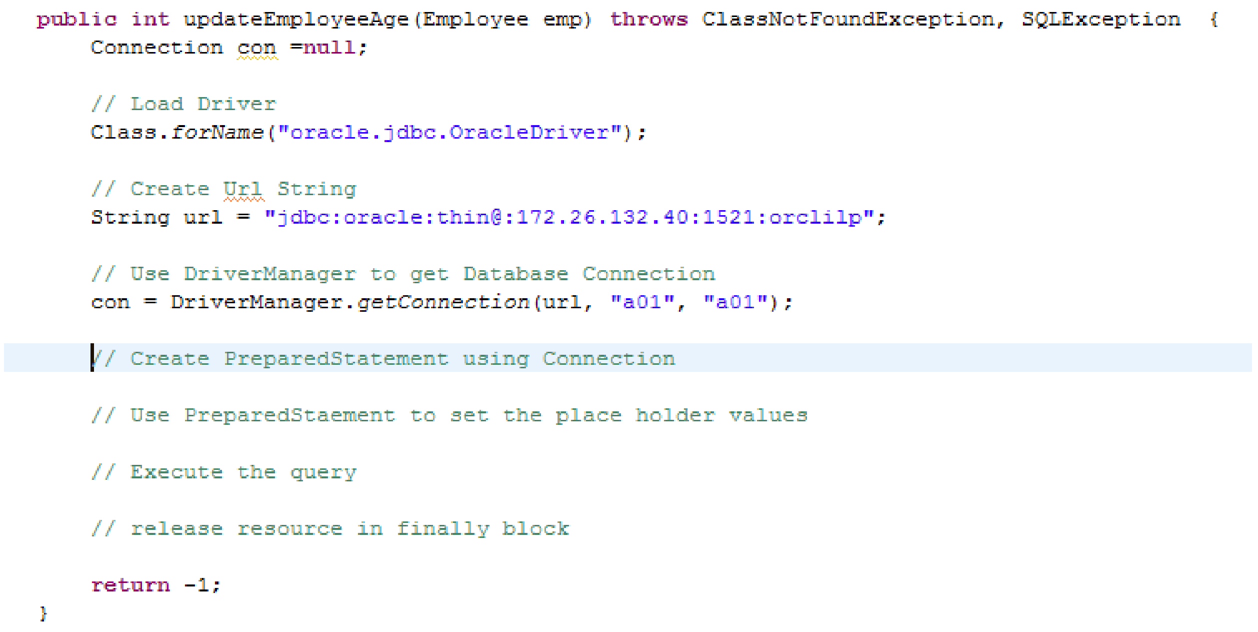
Now during execution if Deepak is passed as value, it will just become part of the value. Even if the hacker tries to inject any other query like what we saw earlier, `' OR '1'='1` . As the query is already compiled it will only become part of the value. Hence SQL injection is not possible when we use prepared statement. Let us now explore the steps to be used, when we want to use PreparedStatemetnt instead of Statement to execute the query. In the given method we would try to update the age of an employee to a new value based on the employeeId provided. Note the employee age and employee Id will be received as sate of Employee Object.

The pseudo code would look like what is given in the figure below:



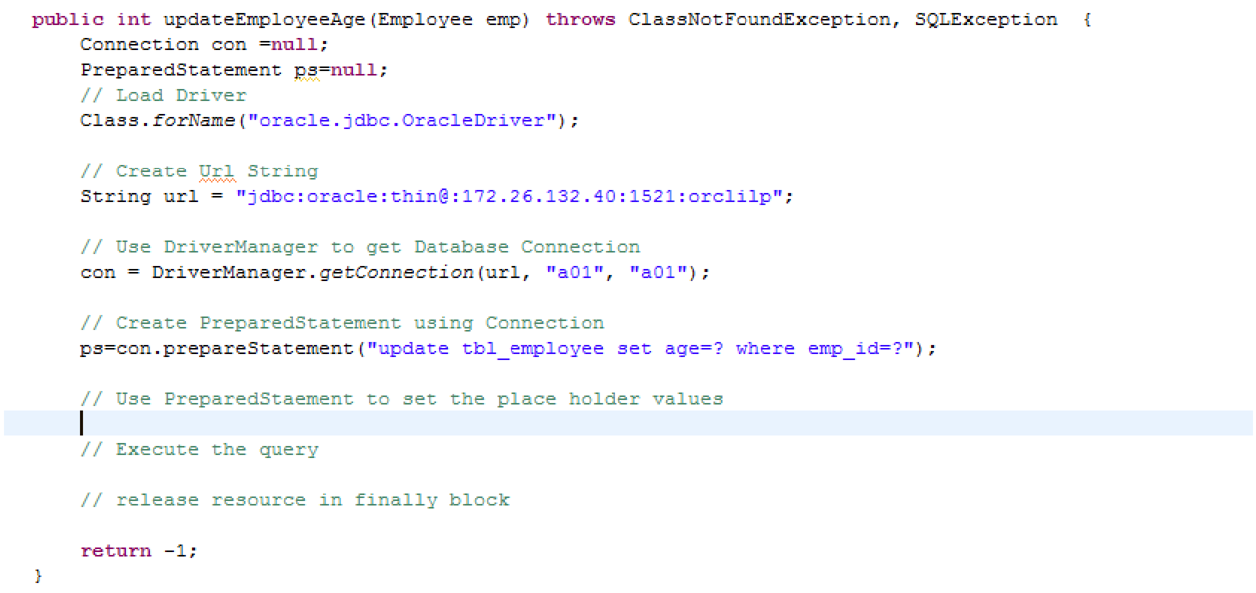
```
public int updateEmployeeAge(Employee emp)  {
    // Load Driver

    // Create Url String

    // Use DriverManager to get Database Connection

    // Create PreparedStatement using Connection

    // Use PreparedStaement to set the place holder values

    // Execute the query

    // release resource in finally block

    return -1;
}
```

The initial few steps till creating a connection would still remain same. Hence the code initially would look like:



```
public int updateEmployeeAge(Employee emp) throws ClassNotFoundException, SQLException  {
    Connection con =null;

    // Load Driver
    Class.forName("oracle.jdbc.OracleDriver");

    // Create Url String
    String url = "jdbc:oracle:thin@:172.26.132.40:1521:orclilp";

    // Use DriverManager to get Database Connection
    con = DriverManager.getConnection(url, "a01", "a01");

    // Create PreparedStatement using Connection

    // Use PreparedStaement to set the place holder values

    // Execute the query

    // release resource in finally block

    return -1;
}
```

Then next step would be preparing a Preparedstatement using the connection. The signature is con.prepareStatement(String query) to be prepared.



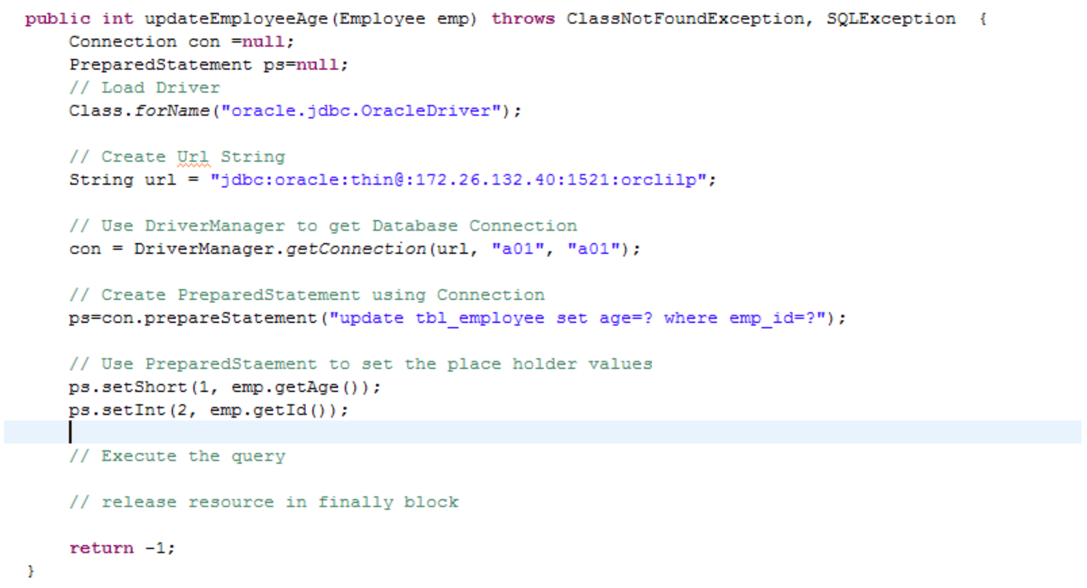
```java
public int updateEmployeeAge(Employee emp) throws ClassNotFoundException, SQLException  {
    Connection con =null;
    PreparedStatement ps=null;
    // Load Driver
    Class.forName("oracle.jdbc.OracleDriver");

    // Create Url String
    String url = "jdbc:oracle:thin@:172.26.132.40:1521:orclilp";

    // Use DriverManager to get Database Connection
    con = DriverManager.getConnection(url, "a01", "a01");

    // Create PreparedStatement using Connection
    ps=con.prepareStatement("update tbl_employee set age=? where emp_id=?");

    // Use PreparedStaement to set the place holder values

    // Execute the query

    // release resource in finally block

    return -1;
}
```

Now you see the difference where earlier we used variables in places where we needed dynamic data, here those are replaced by ‘?’. If your query will require multiple variables then we will put multiple question marks. These are called as place holders. The query will be compiled to check the syntax assuming the placeholders and before execution the values should be replaced with the appropriate values. There are methods available within preparedstatement object which help you set the values of right datatype at the right place.

For example ps.setString (int position, String value) can be used to set a String value to one of the placeholders whose position is to be mentioned as the first parameter.

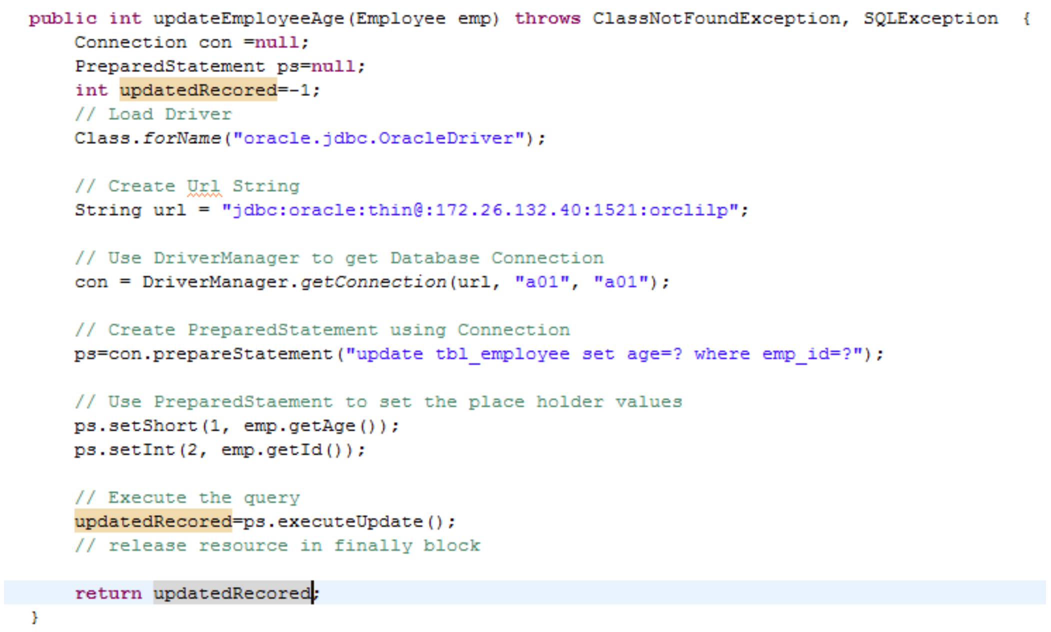
Similarly if the programmer needs to set integer he may use ps.setInt(int position, int value). What we need is to set a short value in the 1st position. Hence the value would look something like this.

The next step is setting all the parameter values. 1st parameter is age of short type and 2nd parameter is employee id of int type. So the code would now look like:



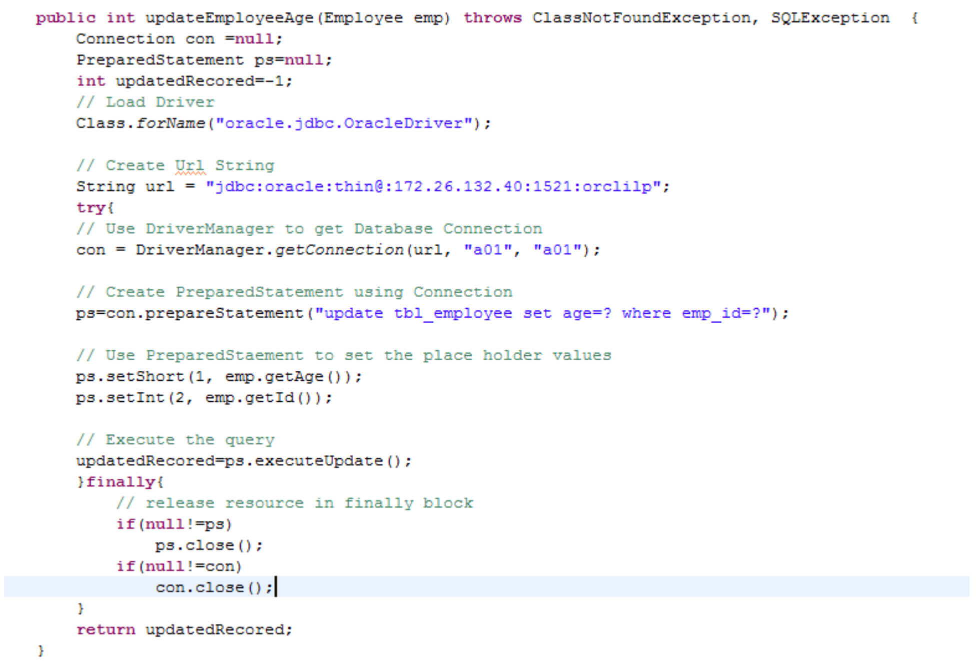
```java
public int updateEmployeeAge(Employee emp) throws ClassNotFoundException, SQLException  {
    Connection con =null;
    PreparedStatement ps=null;
    // Load Driver
    Class.forName("oracle.jdbc.OracleDriver");

    // Create Url String
    String url = "jdbc:oracle:thin@:172.26.132.40:1521:orclilp";

    // Use DriverManager to get Database Connection
    con = DriverManager.getConnection(url, "a01", "a01");

    // Create PreparedStatement using Connection
    ps=con.prepareStatement("update tbl_employee set age=? where emp_id=?");

    // Use PreparedStaement to set the place holder values
    ps.setShort(1, emp.getAge());
    ps.setInt(2, emp.getId());

    // Execute the query

    // release resource in finally block

    return -1;
}
```

Next step is executing the query using executeUpdate method and storing the number of records updated in a integer variable.



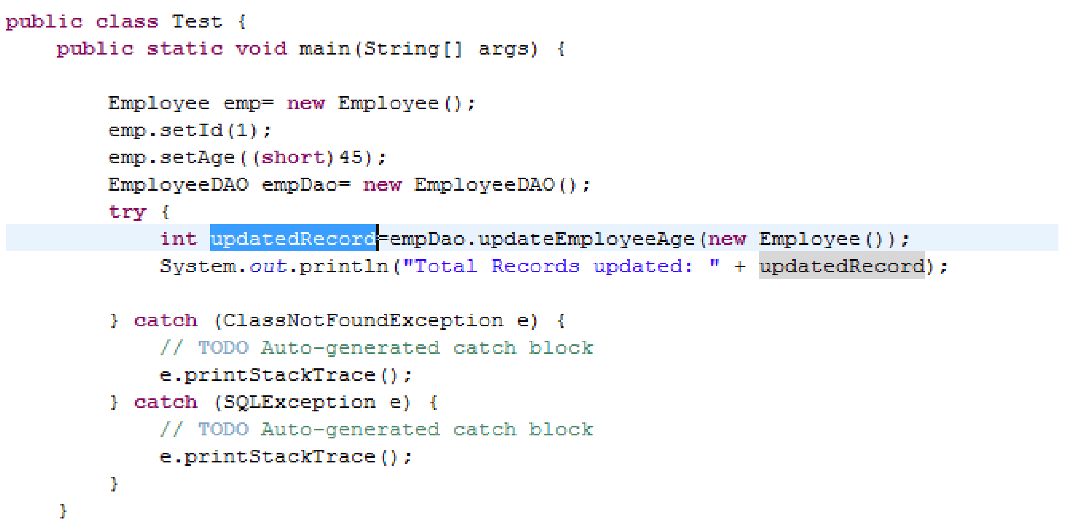
```java
public int updateEmployeeAge(Employee emp) throws ClassNotFoundException, SQLException  {
    Connection con =null;
    PreparedStatement ps=null;
    int updatedRecored=-1;
    // Load Driver
    Class.forName("oracle.jdbc.OracleDriver");

    // Create Url String
    String url = "jdbc:oracle:thin@:172.26.132.40:1521:orclilp";

    // Use DriverManager to get Database Connection
    con = DriverManager.getConnection(url, "a01", "a01");

    // Create PreparedStatement using Connection
    ps=con.prepareStatement("update tbl_employee set age=? where emp_id=?");

    // Use PreparedStaement to set the place holder values
    ps.setShort(1, emp.getAge());
    ps.setInt(2, emp.getId());

    // Execute the query
    updatedRecored=ps.executeUpdate();
    // release resource in finally block

    return updatedRecored;
}
```

Don’t forget to close all the resources in finally block. The resources over here are ps and con. The final code would look like:



```
public int updateEmployeeAge(Employee emp) throws ClassNotFoundException, SQLException  {
    Connection con =null;
    PreparedStatement ps=null;
    int updatedRecored=-1;
    // Load Driver
    Class.forName("oracle.jdbc.OracleDriver");

    // Create Url String
    String url = "jdbc:oracle:thin@:172.26.132.40:1521:orclilp";
    try{
    // Use DriverManager to get Database Connection
    con = DriverManager.getConnection(url, "a01", "a01");

    // Create PreparedStatement using Connection
    ps=con.prepareStatement("update tbl_employee set age=? where emp_id=?");

    // Use PreparedStaement to set the place holder values
    ps.setShort(1, emp.getAge());
    ps.setInt(2, emp.getId());

    // Execute the query
    updatedRecored=ps.executeUpdate();
    }finally{
        // release resource in finally block
        if(null!=ps)
            ps.close();
        if(null!=con)
            con.close();
    }
    return updatedRecored;
}
```

The method to update the age of an Employee using employee id is completed. The next thing is to write a Test class with main method to test the code. It would look like:



```
public class Test {
    public static void main(String[] args) {

        Employee emp= new Employee();
        emp.setId(1);
        emp.setAge((short)45);
        EmployeeDAO empDao= new EmployeeDAO();
        try {
            int updatedRecord=empDao.updateEmployeeAge(new Employee());
            System.out.println("Total Records updated: " + updatedRecord);

        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

After executing the code please verify the data in the database. Also try the below scenarios:

a. Give a wrong driver e.g oracle.jdbc1.OracleDriver

b. Create a wrong url string.

c. Try with wrong user id and password, in the step where you try to getConnection using DriverManager class.

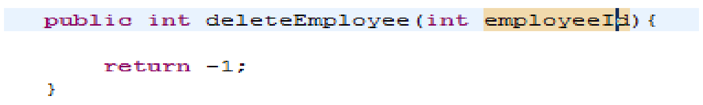
d. Try with a wrong query string, e.g; use invalid table name in the insert query

e. Try setting data in wrong position

f. Try setting data of wrong data type

1.2.9 Deleting existing Data

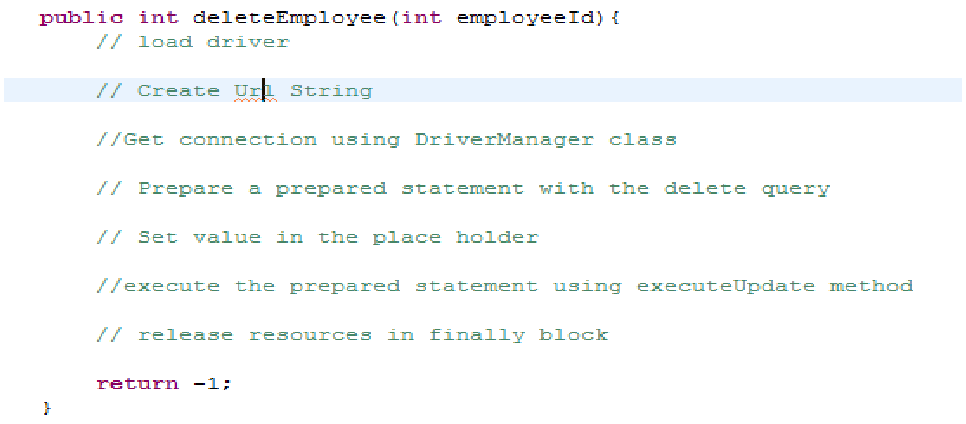
This is the last section of JDBC chapter. As we have learned most of the steps things are going to be pretty easy over here. We will use PreparedStatement to delete a record. Also we will delete data based on the employeeId given. To start with the signature of the method should be:



```
public int deleteEmployee(int employeeId){

    return -1;
}
```

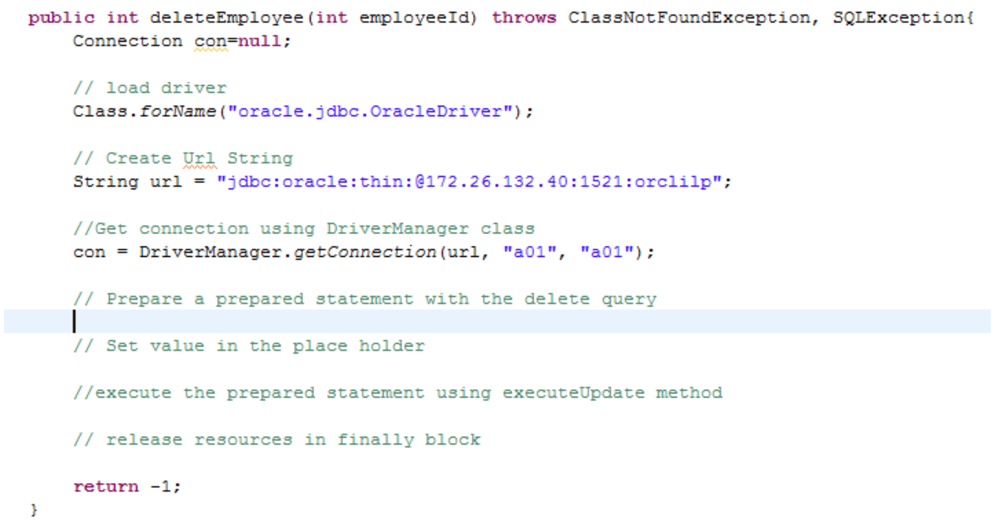
We understand that the method would expect an id and delete the record with the given employeeId.

The steps with in the method would be:



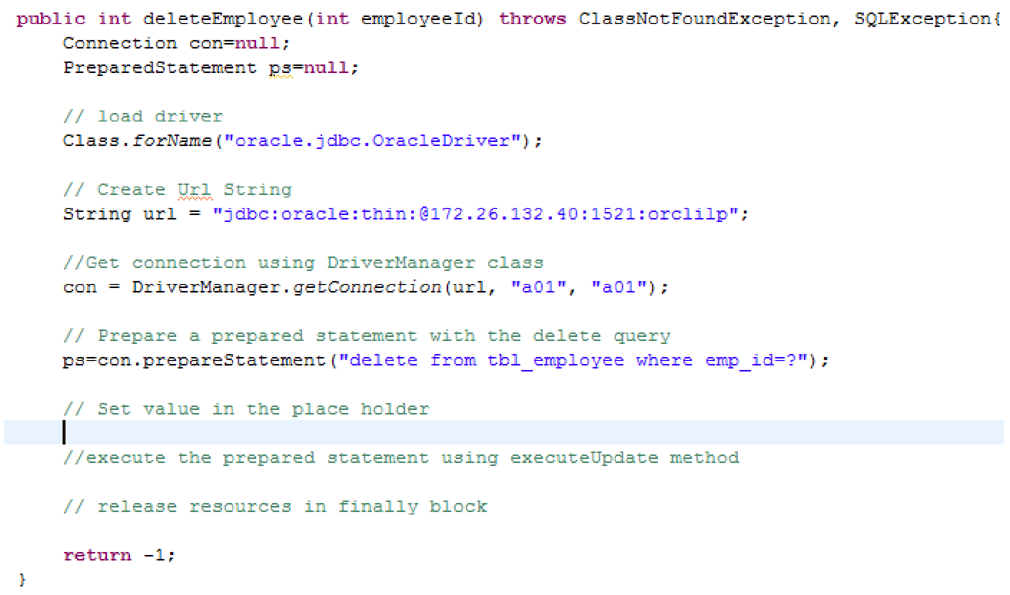
```
public int deleteEmployee(int employeeId){
    // load driver

    // Create Url String

    //Get connection using DriverManager class

    // Prepare a prepared statement with the delete query

    // Set value in the place holder

    //execute the prepared statement using executeUpdate method

    // release resources in finally block

    return -1;
}
```

All steps till creating connection would remain the same as in the above method. Hence it would look like:



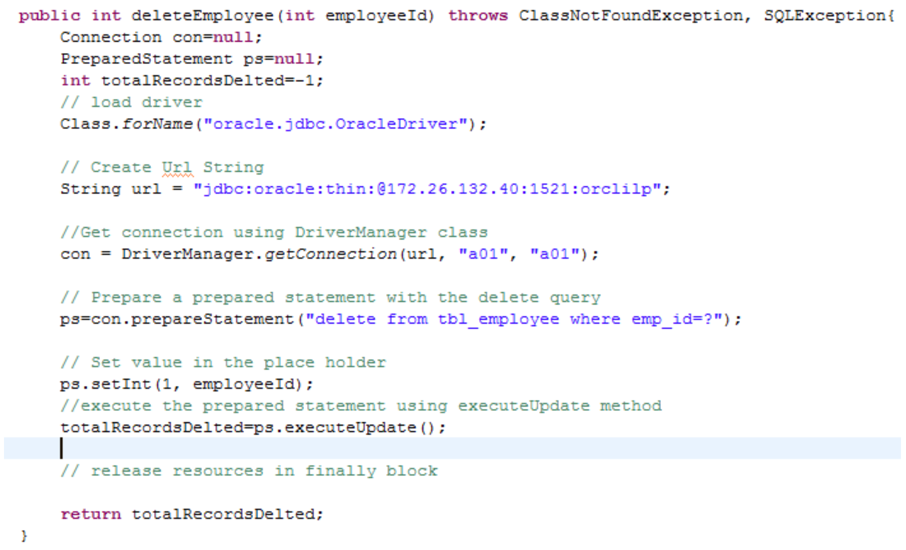
```
public int deleteEmployee(int employeeId) throws ClassNotFoundException, SQLException{
    Connection con=null;

    // load driver
    Class.forName("oracle.jdbc.OracleDriver");

    // Create Url String
    String url = "jdbc:oracle:thin:@172.26.132.40:1521:orclilp";

    //Get connection using DriverManager class
    con = DriverManager.getConnection(url, "a01", "a01");

    // Prepare a prepared statement with the delete query

    // Set value in the place holder

    //execute the prepared statement using executeUpdate method

    // release resources in finally block

    return -1;
}
```

Next step would be creating preparedstatement with the delete query.



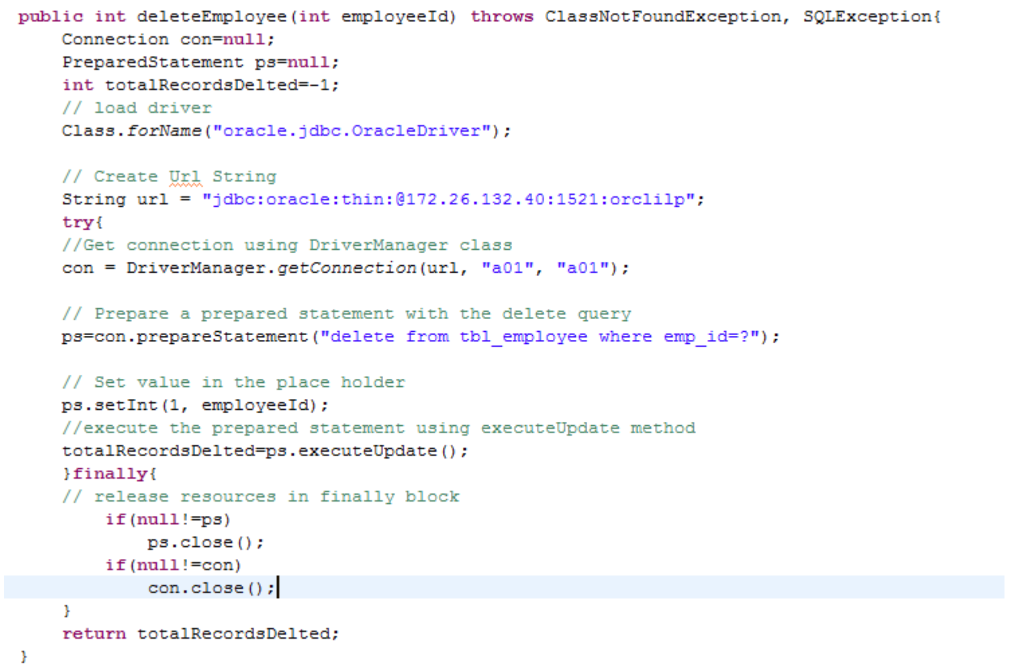
```
public int deleteEmployee(int employeeId) throws ClassNotFoundException, SQLException{
    Connection con=null;
    PreparedStatement ps=null;

    // load driver
    Class.forName("oracle.jdbc.OracleDriver");

    // Create Url String
    String url = "jdbc:oracle:thin:@172.26.132.40:1521:orclilp";

    //Get connection using DriverManager class
    con = DriverManager.getConnection(url, "a01", "a01");

    // Prepare a prepared statement with the delete query
    ps=con.prepareStatement("delete from tbl_employee where emp_id=?");

    // Set value in the place holder

    //execute the prepared statement using executeUpdate method

    // release resources in finally block

    return -1;
}
```

The next step is to set the employeeId in the parameter and then execute the sql statement.



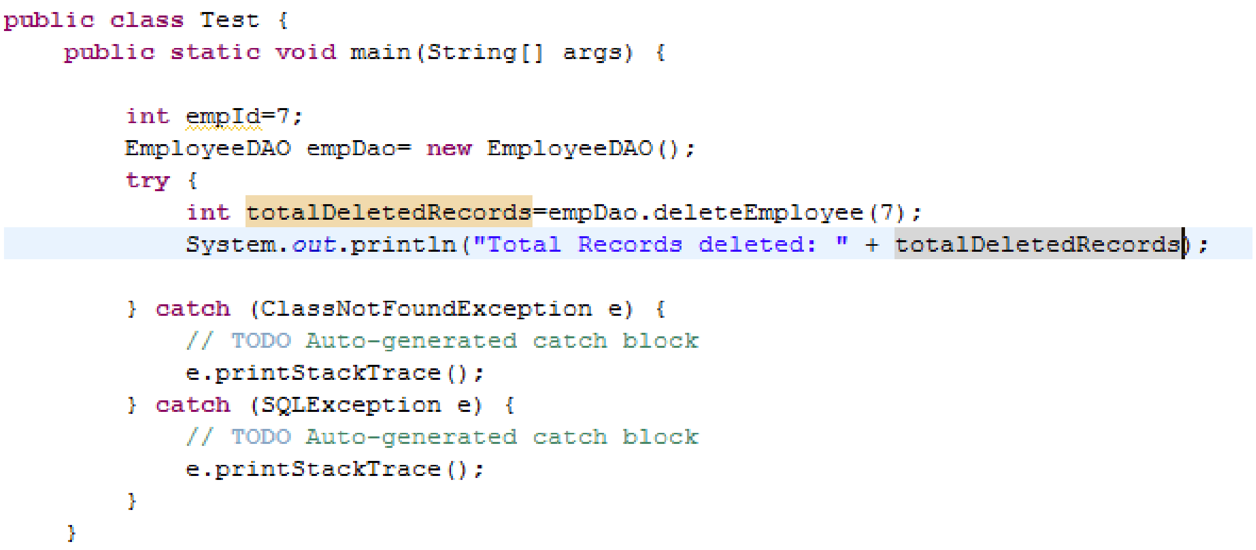
```
public int deleteEmployee(int employeeId) throws ClassNotFoundException, SQLException{
    Connection con=null;
    PreparedStatement ps=null;
    int totalRecordsDelted=-1;
    // load driver
    Class.forName("oracle.jdbc.OracleDriver");

    // Create Url String
    String url = "jdbc:oracle:thin:@172.26.132.40:1521:orclilp";

    //Get connection using DriverManager class
    con = DriverManager.getConnection(url, "a01", "a01");

    // Prepare a prepared statement with the delete query
    ps=con.prepareStatement("delete from tbl_employee where emp_id=?");

    // Set value in the place holder
    ps.setInt(1, employeeId);
    //execute the prepared statement using executeUpdate method
    totalRecordsDelted=ps.executeUpdate();

    // release resources in finally block

    return totalRecordsDelted;
}
```

Also of course the last step would be releasing all the resources. So the final code would look like:



```java
public int deleteEmployee(int employeeId) throws ClassNotFoundException, SQLException{
    Connection con=null;
    PreparedStatement ps=null;
    int totalRecordsDelted=-1;
    // load driver
    Class.forName("oracle.jdbc.OracleDriver");

    // Create Url String
    String url = "jdbc:oracle:thin:@172.26.132.40:1521:orclilp";
    try{
    //Get connection using DriverManager class
    con = DriverManager.getConnection(url, "a01", "a01");

    // Prepare a prepared statement with the delete query
    ps=con.prepareStatement("delete from tbl_employee where emp_id=?");

    // Set value in the place holder
    ps.setInt(1, employeeId);
    //execute the prepared statement using executeUpdate method
    totalRecordsDelted=ps.executeUpdate();
    }finally{
    // release resources in finally block
        if(null!=ps)
            ps.close();
        if(null!=con)
            con.close();
    }
    return totalRecordsDelted;
}
```

Now we are ready to test the delete method that we created right now. Let's create a Test class with main method to call the method. It would look like:



```java
public class Test {
    public static void main(String[] args) {

        int empId=7;
        EmployeeDAO empDao= new EmployeeDAO();
        try {
            int totalDeletedRecords=empDao.deleteEmployee(7);
            System.out.println("Total Records deleted: " + totalDeletedRecords);

        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

1.3 Reference Links:

http://www.java2novice.com/jdbc/

http://www.tutorialspoint.com/jdbc/

http://docs.oracle.com/javase/tutorial/jdbc/