Unit IV | Enterprise Java Beans | Java Database Connectivity (JDBC)

Enterprise Java Bean: Preparing a Class to be a JavaBeans, Creating a JavaBeans, JavaBeans Properties, Types of beans, Stateful Session bean, Stateless Session bean, Entity bean

Java Database Connectivity (JDBC): Merging Data from Multiple Tables: Joining, Manipulating, Databases with JDBC, Prepared Statements, Transaction Processing, Stored Procedures.

Enterprise Java Bean

What is Enterprise Java Bean (EJB)?

Enterprise Java Beans (EJB) is one of the important Java APIs for Enterprise Software Development. It is a server-side technology that defines the business logic of an application.

EJB provides an architecture to develop and deploy component based enterprise applications considering robustness, high scalability, and high performance.

Enterprise Java Beans is mainly used for web related software development providing Java Servlet Lifecycle (JSL) management, transaction procedure and other web services.

Business logic can be the basic functionality or may have database management.

To run EJB application we need an application server (EJB Container) such as Jboss, Glassfish, Weblogic, Websphere etc.

To access the enterprise java beans in some application we need as service named as Java Naming and Directory Interface (JNDI).

Properties of Enterprise Java Beans

- Life cycle management
- Security
- Transaction management
- Object pooling

Types of Enterprise Java Beans

Enterprise Java Beans can be of three types

- Session Bean
- Entity Bean
- Message Bean

Session Bean

Session bean contains business logic that can be invoked by local, remote or web service client. Such beans contain the business logic that do not require data persistence.

There are two types of session beans:

- Stateless session bean
- Stateful session bean

Stateful session bean performs business task with the help of a state. Stateful session bean can be used to access various method calls by storing the information in an instance variable. Some of the applications require information to be stored across separate method calls. In a shopping site, the items chosen by a customer must be stored as data is an example of stateful session bean.

Stateless session bean implement business logic without managing data in instance variables. They simply take some argument, process the data and return the result.

Entity Bean

It summarizes the state that can be remained in the database. It is deprecated. Now, it is replaced with JPA (Java Persistent API).

There are two types of entity bean:

- Bean Managed Persistence
- Container Managed Persistence

In a bean managed

persistence type of entity bean, the programmer has to write the code for database calls. It persists across multiple sessions and multiple clients.

Container managed persistence are enterprise bean that persists across database. In container managed persistence the container take care of database calls.

Message Driven Bean

Like Session Bean, it contains the business logic but it is invoked by passing messages.

Advantages of Enterprise Java Beans

- EJB repository contains system-level services to enterprise beans, the bean developer can focus on solving business problems. Rather than the bean developer, the EJB container is responsible for system-level services such as transaction management and security authorization.
- The beans rather than the clients contain the application's business logic, the client developer can focus on the presentation of the client.
- Enterprise Java Beans are portable elements, the application assembler can build new applications from the beans that already exists

Disadvantages of Enterprise Java Beans

- Requires application server

- Complex to understand and develop EJB applications.

Sample Code of Session Bean

First define an interface having abstract method to define the functionality then implement those methods in Bean class. Use @Remote annotation to mark an interface as remote interface to allow the remote access. Use @Stateless annotation to mark the bean class as stateless enterprise bean.

```
import javax.ejb.Remote;
@Remote
public interface BasicSessionBeanRemote {
  double squareRoot(double num);}
}
```

```
import javax.ejb.Stateless;
@Stateless
public class LibrarySessionBean implements BasicSessionBeanRemote {
   public  double squareRoot(int num) {
     return Math.sqrt(num);
  }
}
```

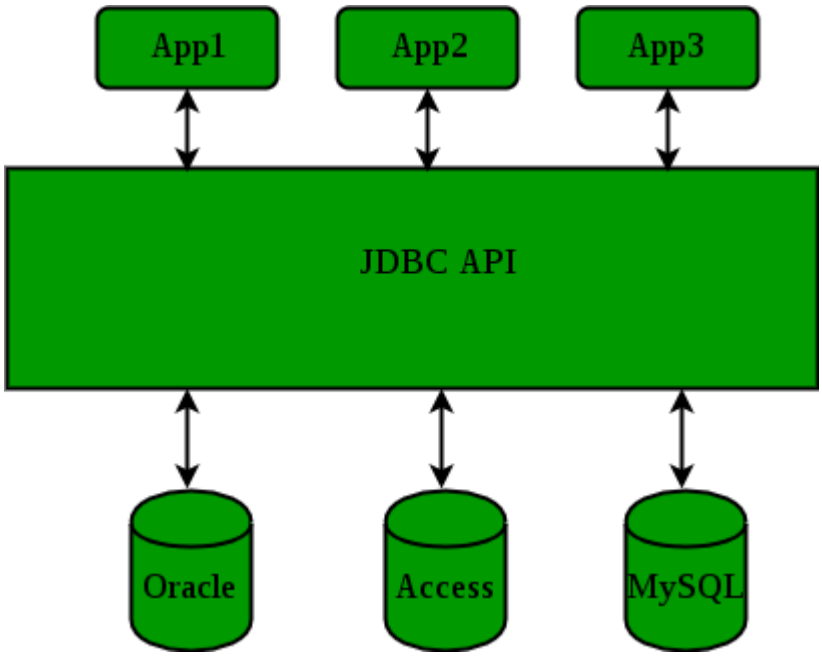
Java Database Connectivity (JDBC)

What is JDBC?

JDBC Stands for Java Database Connectivity.

Java is a front-end that can be connected to any back-end RDBMS using JDBC API (Application Programming Interface). Java provides **java.sql** and **javax.sql** packages with a set of classes and interfaces.

- DriverManager class
- SQLException class
- Connection interface
- Statement interface
- PreparedStatment interface
- CallableStatement interface
- ResultSet interface



What is driver?

Special classes provided by the RDBMS company bundled inside a JAR (Java Archive) file. It is also known as Java Connector.

Class name for MySQL is **com.mysql.cj.jdbc.Driver** class

Example

MySQL driver can be downloaded from https://dev.mysql.com/downloads/connector/j/

Types of JDBC Drivers

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand.

There are 4 types of JDBC drivers:

- Type-1 driver or JDBC-ODBC bridge driver
- Type-2 driver or Native-API driver
- Type-3 driver or Network Protocol driver
- Type-4 driver or Thin driver

Type-1 driver or JDBC-ODBC bridge driver

Type-1 driver or JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Type-1 driver is also called Universal driver because it can be used to connect to any of the databases.

- As a common driver is used in order to interact with different databases, the data transferred through this driver is not so secured.
- The ODBC bridge driver is needed to be installed in individual client machines.
- Type-1 driver isn't written in java, that's why it isn't a portable driver.
- This driver software is built-in with JDK so no need to install separately.
- It is a database independent driver.

Type-2 driver or Native-API driver

The Native API driver uses the client -side libraries of the database. This driver converts JDBC method calls into native calls of the database API. In order to interact with different database, this driver needs their local API, that's why data transfer is much more secure as compared to type-1 driver.

- Driver needs to be installed separately in individual client machines
- The Vendor client library needs to be installed on client machine.
- Type-2 driver isn't written in java, that's why it isn't a portable driver
- It is a database dependent driver.

Type-3 driver or Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. Here all the database connectivity drivers are present in a single server, hence no need of individual client-side installation.

- Type-3 drivers are fully written in Java, hence they are portable drivers.
- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- Network support is required on client machine.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.
- Switch facility to switch over from one database to another database.

Type-4 driver or Thin driver

Type-4 driver is also called native protocol driver. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.

- Does not require any native library and Middleware server, so no client-side or server-side installation.
- It is fully written in Java language, hence they are portable drivers.

How to insert data in database table using Scanner?

- Step 1
  - Create the database e.g. sectionc
- Step 2
  - Create a table e.g. student
    - rollno, int, primary
    - name, varchar, 50
    - branch, varchar, 20
- Input the data from user and create the SQL Statement
- Load the driver
- Establish the connection
- Prepare the statement
- Execute the statement
- Close the connection

```
import java.sql.*;
import java.util.Scanner;
public class SaveStudent {
    public static void main(String[] args) {
        try{
            Scanner sc=new Scanner(System.in);
            System.out.print("Roll No :");
            int rollno=sc.nextInt();
            System.out.println("Name : ");
            String name=sc.next();
            System.out.println("Branch : ");
            String branch=sc.next();
            String sql="INSERT INTO student
VALUES(?,?,?)";
            //Step 1: Load the driver
            DriverManager.registerDriver(new
com.mysql.cj.jdbc.Driver());
            //Step 2: Connect with database
            Connection
cn=DriverManager.getConnection("jdbc:mysql://localhost:33
06/sectionc","root","");
            //Step 3: Create the PreparedStatement
            PreparedStatement
ps=cn.prepareStatement(sql);
            ps.setInt(1,rollno);
            ps.setString(2,name);
            ps.setString(3,branch);
            ps.executeUpdate();
            cn.close();
            System.out.println("Record Saved");
        }catch (Exception ex){
            System.err.println("Error :
"+ex.getMessage());
        }
    }
}
```

Create a GUI Application to input the data of a student and save into database table.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
public class SaveStudentGUI extends Frame implements
ActionListener {
    Label lblRollno,lblName, lblBranch;
    TextField txtRollno, txtName, txtBranch;
    Button btnSave;
    SaveStudentGUI(){
        lblRollno=new Label("Rollno");
        lblBranch=new Label("Branch");
        lblName=new Label("Name");
        txtBranch=new TextField(10);
        txtName=new TextField(10);
        txtRollno=new TextField(10);
        btnSave=new Button("Save");
        btnSave.addActionListener(this);

        setLayout(new GridLayout(4,2));
        add(lblRollno);add(txtRollno);
        add(lblName);add(txtName);
        add(lblBranch);add(txtBranch);
        add(btnSave);


        setSize(300,300);
        setVisible(true);
    }

    public static void main(String[] args) {
        new SaveStudentGUI();
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        try{
            int rollno=Integer.parseInt(txtRollno.getText());
            String name=txtName.getText();
            String branch= txtBranch.getText();
            String sql="INSERT INTO student VALUES(?,?,?)";
            //Step 1: Load the driver
            DriverManager.registerDriver(new
com.mysql.cj.jdbc.Driver());
            //Step 2: Connect with database
            Connection
cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/se
ctionc","root","");
            //Step 3: Create the PreparedStatement
            PreparedStatement ps=cn.prepareStatement(sql);
            ps.setInt(1,rollno);
            ps.setString(2,name);
            ps.setString(3,branch);
```

```
            ps.executeUpdate();
            cn.close();
            JOptionPane.showMessageDialog(this,"Record
Saved");
        }catch (Exception ex){

JOptionPane.showMessageDialog(this,ex.getMessage());
        }
    }
}
```

How to view all records?

Use **executeQuery()** method of **PreparedStatement** to execute the SELECT statement

Store the result into ResultSet Interface

ResultSet rs=ps.executeQuery();

The ResultSet provides built-in method to move on the records

- boolean first()
- boolean last()
- boolean previous()
- boolean last()

Use getter method to read the data from current record

```
import java.sql.*;
class ReadlAllStudents {
    public static void main(String[] args) throws Exception {
        String sql="SELECT * FROM student";
        DriverManager.registerDriver(new
com.mysql.cj.jdbc.Driver());
        Connection
cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/se
ctionc","root","");
        PreparedStatement ps=cn.prepareStatement(sql);
        ResultSet rs=ps.executeQuery();
        while(rs.next()){

System.out.println(rs.getInt(1)+","+rs.getString(2));
        }
        cn.close();
    }
}
```

What is stored procedure?

A stored procedure is a prepared SQL code having some name like function name, that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter values that is passed.

Use **CREATE PROCEDURE** command to create a procedure and **CallableStatement** interface in Java to use it

How to create it in MySQL?

Every Relational Database Management Software can have their own syntax to create the procedures. In MySQL RDBMS, we need to first define the delimiter that will be used to terminate the stored procedure.

Write your procedure code between BEGIN and END.

Example

DELIMITER //

CREATE OR REPLACE PROCEDURE GetAllBooks()

BEGIN

SELECT * FROM book;

END //

DELIMITER ;

How to use it in Java?

To call the stored procedure in Java, use the **CallableStatement** interface. Create a reference of **CallableStatement** using **prepareCall()** method of Connection interface

```
import java.sql.*;
public class StoredProcedureTest {
    public static void main(String[] args) throws Exception {
        DriverManager.registerDriver(new
com.mysql.cj.jdbc.Driver());
        Connection
cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/li
brary","root","");
        CallableStatement cs=cn.prepareCall("call
GetAllBooks");
        ResultSet rs=cs.executeQuery();
        while(rs.next()){

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

```
        cn.close();
    }
}
```

What is transaction Processing?

While performing insertion, deletion and updation operations in the database tables, the data must be processed completely or should not processed at all to manage the consistency in the database.

To manage the transactions, we need to use the transaction processing.

The Connection interface provides three methods to manage the transactions

- setAutoCommit()
- commit()
- rollback()

First set the auto commit as false so that the records must not be written on disc until we call the commit() method

If some exception occurs we need to call the rollback() method

```
import java.sql.*;
public class TransactionTest {
    public static void main(String[] args) {
        String sql="INSERT INTO book VALUES(?,?,?)";
        Connection cn=null;
        try {
            DriverManager.registerDriver(new
com.mysql.cj.jdbc.Driver());
            cn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/libra
ry", "root", "");
            PreparedStatement ps=cn.prepareStatement(sql);
            cn.setAutoCommit(false);
            try{
                ps.setInt(1,109);
                ps.setString(2,"RPA Development");
                ps.setFloat(3,345.90f);
                ps.executeUpdate();
                cn.commit();
                System.out.println("Transaction Complete");
            }catch (SQLException ex){
                System.out.println("Transaction Roll backed");
                cn.rollback();
            }
        }catch (Exception ex){
            System.err.println("Error : "+ex.getMessage());
        }
    }
}
```

How to merge data from two or more tables?

We can read the data from multiple tables once they have primary key and foreign key relationship using simple SELECT statement or SQL Joins

Example

We have two tables course and student.

The course table have columns as course code (ccode) and course name (cname)

1. ccode, char, 10, primary key
2. cname, varchar, 50, not null

The student table has rollno, student name (sname) and course code (ccode) where ccode is a foreign key

1. rollno, char, 12, primary key
2. sname, varchar, 50, not null
3. ccode, char, 10, foreign key

CREATE TABLE course(ccode char(10) primary key, cname varchar(50) not null);

CREATE TABLE student(rollno char(12) primary key, sname varchar(50) not null, ccode char(10), CONSTRAINT fk_ccode FOREIGN KEY (ccode) REFERENCES course(ccode));

Now insert few sample records

INSERT INTO course VALUES('101','B.Tech.');

INSERT INTO course VALUES('102','MBA');

INSERT INTO student VALUES('19B112233','Amit Kumar','101');

INSERT INTO student VALUES('19B112234','Sanjay Sharma','102');

Make the simple select query to retrieve the data from two tables or SQL JOIN

Simple SELECT statement

SELECT rollno, sname, course.cname FROM course,student WHERE course.ccode=student.ccode;

SQL Join

SELECT student.rollno, Student.sname, course.cname

FROM student

INNER JOIN course ON student.ccode=course.ccode;

Now write the Java code using Simple Query

```
import java.sql.*;
public class MultipleTableTest {
    public static void main(String[] args) throws Exception {
        DriverManager.registerDriver(new
com.mysql.cj.jdbc.Driver());
```

```
        Connection
cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/li
brary","root","");
        String sql="SELECT rollno, sname, course.cname FROM
course,student WHERE course.ccode=student.ccode";
        PreparedStatement ps=cn.prepareStatement(sql);
        ResultSet rs=ps.executeQuery();
        while(rs.next()){

System.out.println(rs.getString(1)+","+rs.getString(2)+","+rs.
getString(3));
        }
        cn.close();
    }
}
```

```
import java.sql.*;
public class MultipleTableTestJoin {
    public static void main(String[] args) throws Exception {
        DriverManager.registerDriver(new
com.mysql.cj.jdbc.Driver());
        Connection
cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/li
brary","root","");
        String sql="SELECT student.rollno, Student.sname,
course.cname FROM student INNER JOIN course ON
student.ccode=course.ccode";
        PreparedStatement ps=cn.prepareStatement(sql);
        ResultSet rs=ps.executeQuery();
        while(rs.next()){

System.out.println(rs.getString(1)+","+rs.getString(2)+","+rs.
getString(3));
        }
        cn.close();
    }
}
```