

# **Java & JEE Training**

**Day 24 – JDBC Contd. &  
Introduction to Enterprise Java**

**MindsMapped Consulting**

# **Java & JEE Training**

**JDBC – Java Database Connectivity**

**MindsMapped Consulting**

# Introduction

---

- Data stored in variables and arrays is temporary
  - It's lost when a local variable goes out of scope or when the program terminates
- For long-term retention of data, computers use **files**.
- Computers store files on **secondary storage devices**
  - hard disks, optical disks, flash drives and magnetic tapes.
- Data maintained in files is **persistent data** because it exists beyond the duration of program execution.

# Data Hierarchy

---

Hierarchy of data	Example
Database	<div>Personnel file Department file Payroll file</div> (Project database)
Files	<div>098 - 40 - 1370 Fiske, Steven 01-05-1985 549 - 77 - 1001 Buckley, Bill 02-17-1979 005 - 10 - 6321 Johns, Francine 10-07-1997</div> (Personnel file)
Records	<div>098 - 40 - 1370 Fiske, Steven 01-05-1985</div> (Record containing SSN, last and first name, hire date)
Fields	<div>Fiske</div> (Last name field)
Characters (Bytes)	<div>1000100</div> (Letter F in ASCII)

# Databases

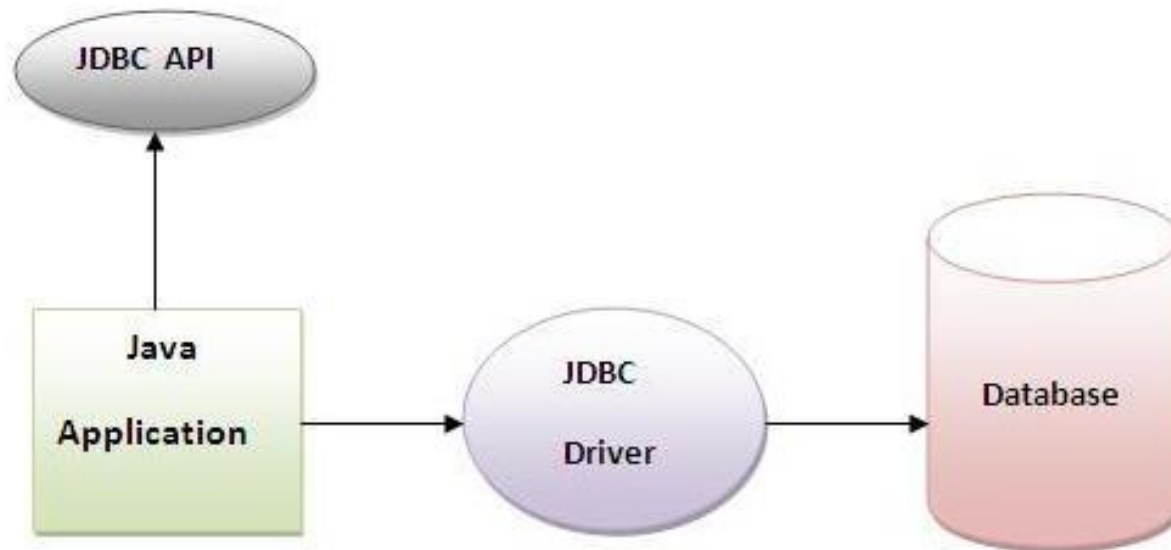
---

- There are many ways to organize records in a file. The most common is called a **sequential file**, in which records are stored in order by the record-key field.
- A group of related files is called a **database**.
- A collection of programs designed to create and manage databases is called a **database management system (DBMS)**.

# JDBC – Java Database Connectivity

---

- Java JDBC is a Java API to connect and execute query with the database. JDBC API uses JDBC drivers to connect with the database.
- Before JDBC, **ODBC API** was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).



# JDBC Driver

---

- JDBC Driver is a software component that enables java application to interact with the database.

## 5 Steps to connect to the database in java

---

1. Register the driver class
2. Creating connection
3. Creating statement
4. Executing queries
5. Closing connection

# 5 Steps to connect to the database in java

---

## 1. Register the driver class

```
Class.forName("oracle.jdbc.driver.OracleDriver")  
(throws ClassNotFoundException)
```

## 2. Creating connection –

```
Connection con=DriverManager.getConnection(  
"jdbc:oracle:thin:@localhost:1521:xe","system","password");  
Connection con=DriverManager.getConnection(  
"jdbc:oracle:thin:@localhost:1521:xe");  
(throws SQLException)
```

## 3. Creating statement:

```
Statement stmt=con.createStatement();  
(throws SQLException)
```

## 4. Executing queries

```
ResultSet rs=stmt.executeQuery("select * from emp");  
while(rs.next()){  
    System.out.println(rs.getInt(1)+" "+rs.getString(2));  
}  
(throws SQLException)
```

## 5. Closing connection

```
con.close(); // (throws SQLException)
```

# Working with Databases... Instruction.

---

- In our class, we will work with Oracle 11g XE (Express Edition).
- It is available for download for free from Oracle site:  
<http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html>
- Download the zip file, extract it, and run the setup file for installing Oracle database.
- Oracle database will get installed and use default port 1521.
- Then follow instructions here to complete the initial setup.

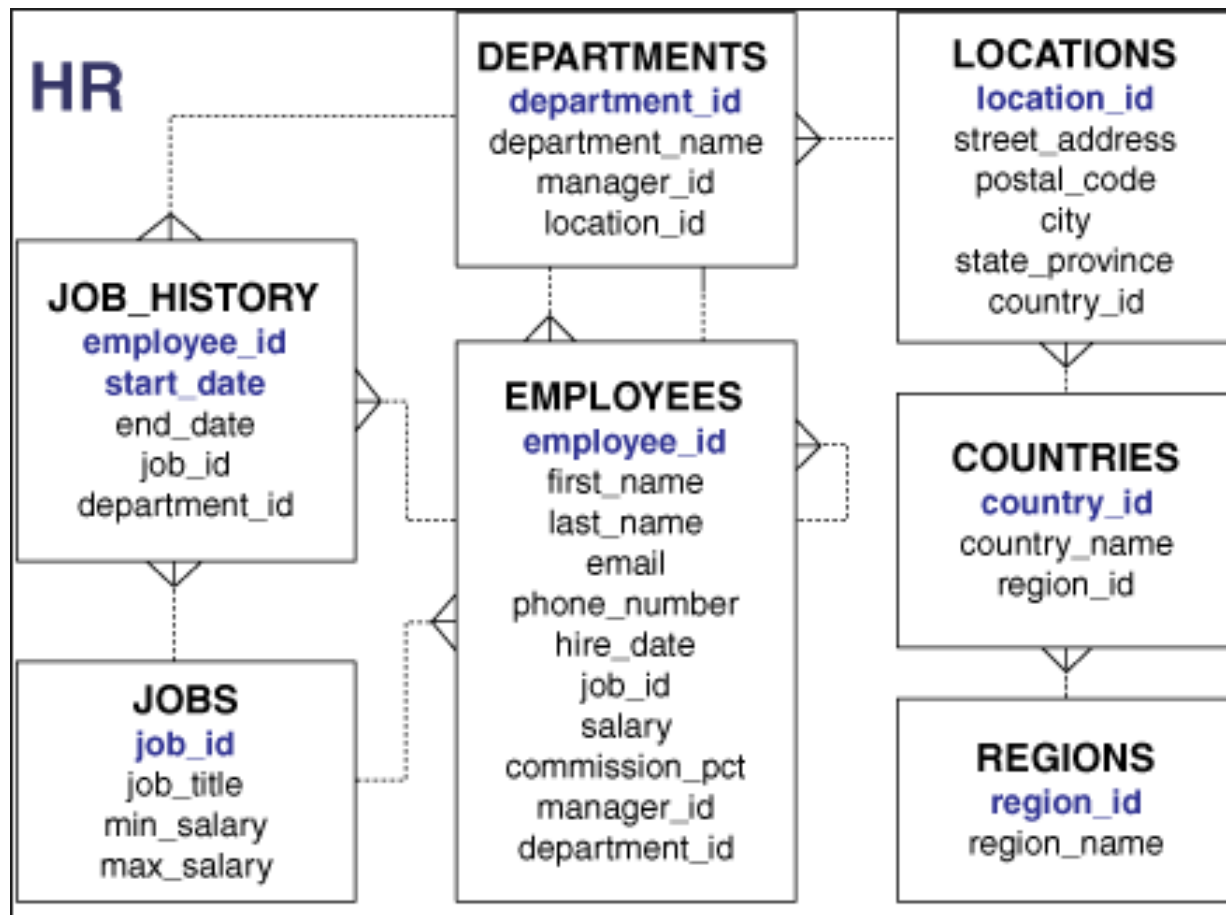
[https://docs.oracle.com/cd/E17781\\_01/admin.112/e18585/toc.htm](https://docs.oracle.com/cd/E17781_01/admin.112/e18585/toc.htm)

- We will be working with the HR user/schema in all our examples.
  - Unlock HR schema: ALTER USER HR ACCOUNT UNLOCK
  - Specify a password for HR: ALTER USER HR IDENTIFIED BY <PASSWORD>

# HR USER / SCHEMA

---

- In Oracle, User and Schema mean the same.



# Demo of HR Schema

---

- Let us look at the Employees and Departments Tables.
- Basic CRUD operations syntax reference for SQL:  
<http://www.orafaq.com/wiki/CRUD>

# Ojdbc6.jar

---

- Copy ojdbc6.jar from C:\oracle\app\oracle\product\11.2.0\server\jdbc\lib to your JRE\lib\ext folder.
- This has the JDBC driver you will need.
- Other way of doing is by adding the path to the jar file to your classpath
  - Temporarily: Set classpath in commandline.
  - Permanent: Add path of jar file to classpath environment variable.

# JDBC Example... Connecting to the database.

---

```
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object
Statement stmt=con.createStatement();

//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));

//step5 close the connection object
con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

# DriverManager class

---

- The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

`public static Connection getConnection(String url):`

is used to establish the connection with the specified url.

`public static Connection getConnection(String url,String userName,String password):`

is used to establish the connection with the specified url, username and password.

# Connection interface

---

- A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData.
- The Connection interface provide many methods for transaction management like commit(), rollback() etc.
- ***By default, connection commits the changes after executing queries.***
- Commonly used methods:

**1) public Statement createStatement():** creates a statement object that can be used to execute SQL queries.

**2) public Statement createStatement(int resultSetType,int resultSetConcurrency):** Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

**3) public void setAutoCommit(boolean status):** is used to set the commit status.By default it is true.

**4) public void commit():** saves the changes made since the previous commit/rollback permanent.

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

**6) public void close():** closes the connection and Releases a JDBC resources immediately.

# Statement interface

---

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

- 1) public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.
- 2) public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc.
- 3) public boolean execute(String sql):** is used to execute queries that may return multiple results.
- 4) public int[] executeBatch():** is used to execute batch of commands.

## Example: Insert, Update and Delete using Statement.

---

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe"
,"system","oracle");
Statement stmt=con.createStatement();

//stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");
//int result=stmt.executeUpdate("update emp765 set
name='Vimal',salary=10000 where id=33");
int result=stmt.executeUpdate("delete from emp765 where id=33");
System.out.println(result+" records affected");
con.close();
}}
```

# ResultSet interface

---

- The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.
- **By default, ResultSet object can be moved forward only and it is not updatable.**
- But we can make this object to move forward and backward direction by passing either TYPE\_SCROLL\_INSENSITIVE or TYPE\_SCROLL\_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

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

# Commonly used methods of ResultSet interface

---

<b>1) public boolean next():</b>	is used to move the cursor to the one row next from the current position.
<b>2) public boolean previous():</b>	is used to move the cursor to the one row previous from the current position.
<b>3) public boolean first():</b>	is used to move the cursor to the first row in result set object.
<b>4) public boolean last():</b>	is used to move the cursor to the last row in result set object.
<b>5) public boolean absolute(int row):</b>	is used to move the cursor to the specified row number in the ResultSet object.
<b>6) public boolean relative(int row):</b>	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
<b>7) public int getInt(int columnIndex):</b>	is used to return the data of specified column index of the current row as int.
<b>8) public int getInt(String columnName):</b>	is used to return the data of specified column name of the current row as int.
<b>9) public String getString(int columnIndex):</b>	is used to return the data of specified column index of the current row as String.
<b>10) public String getString(String columnName):</b>	is used to return the data of specified column name of the current row as String.

## Example of Scrollable ResultSet (retrieve data of 3<sup>rd</sup> row)

---

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sy
stem","oracle");
Statement
stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CO
NCUR_UPDATABLE);
ResultSet rs=stmt.executeQuery("select * from emp765");

//getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();
}}
```

# PreparedStatement interface

---

- The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

e.g. `String sql="insert into emp values(?,?,?)";`

- Protects against SQLInjection attacks.

Method	Description
<code>public void setInt(int paramIndex, int value)</code>	sets the integer value to the given parameter index.
<code>public void setString(int paramIndex, String value)</code>	sets the String value to the given parameter index.
<code>public void setFloat(int paramIndex, float value)</code>	sets the float value to the given parameter index.
<code>public void setDouble(int paramIndex, double value)</code>	sets the double value to the given parameter index.
<code>public int executeUpdate()</code>	executes the query. It is used for create, drop, insert, update, delete etc.
<code>public ResultSet executeQuery()</code>	executes the select query. It returns an instance of ResultSet.

## Example of PreparedStatement interface that inserts the record

---

- create table emp(id number(10),name varchar2(50));

```
import java.sql.*;
class InsertPrepared{
    public static void main(String args[]){
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");

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

            PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
            stmt.setInt(1,101);//1 specifies the first parameter in the query
            stmt.setString(2,"Ratan");

            int i=stmt.executeUpdate();
            System.out.println(i+" records inserted");

            con.close();

        }catch(Exception e){ System.out.println(e);}
    }
}
```

## Example of PreparedStatement interface that updates the record

---

```
PreparedStatement stmt=con.prepareStatement("update emp set  
name=? where id=?");  
stmt.setString(1,"Pawan");//the first parameter in the query (name)  
stmt.setInt(2,101);  
  
int i=stmt.executeUpdate();  
System.out.println(i+" records updated");
```

## Example of PreparedStatement interface that deletes the record

---

```
PreparedStatement stmt=con.prepareStatement("delete from emp where id=?");  
stmt.setInt(1,101);
```

```
int i=stmt.executeUpdate();  
System.out.println(i+" records deleted");
```

## Example of PreparedStatement interface that retrieve the records of a table

---

```
PreparedStatement stmt=con.prepareStatement("select * from emp");
ResultSet rs=stmt.executeQuery();
while(rs.next()){
    System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

# ResultSetMetaData Interface

---

- If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

Method	Description
<code>public int getColumnCount()throws SQLException</code>	it returns the total number of columns in the ResultSet object.
<code>public String getColumnName(int index)throws SQLException</code>	it returns the column name of the specified column index.
<code>public String getColumnName(int index)throws SQLException</code>	it returns the column type name for the specified index.
<code>public String getTableName(int index)throws SQLException</code>	it returns the table name for the specified column index.

## ResultSetMetaData example

---

```
import java.sql.*;
class Rsmd{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("select * from emp");
ResultSet rs=ps.executeQuery();
ResultSetMetaData rsmd=rs.getMetaData();

System.out.println("Total columns: "+rsmd.getColumnCount());
System.out.println("Column Name of 1st column: "+rsmd.getColumnName(1));
System.out.println("Column Type Name of 1st column:
"+rsmd.getColumnTypeName(1));

con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

# DatabaseMetaData interface

---

- DatabaseMetaData interface provides methods to get meta data of a database such as database product name, database product version, driver name, name of total number of tables, name of total number of views etc.
- Commonly used methods:
  - **public String getDriverName()throws SQLException:** it returns the name of the JDBC driver.
  - **public String getDriverVersion()throws SQLException:** it returns the version number of the JDBC driver.
  - **public String getUsername()throws SQLException:** it returns the username of the database.
  - **public String getDatabaseProductName()throws SQLException:** it returns the product name of the database.
  - **public String getDatabaseProductVersion()throws SQLException:** it returns the product version of the database.
  - **public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)throws SQLException:** it returns the description of the tables of the specified catalog. The table type can be TABLE, VIEW, ALIAS, SYSTEM TABLE, SYNONYM etc.

## Example of DatabaseMetaData interface

---

```
import java.sql.*;
class Dbmd{
public static void main(String args[]){
try{
    Class.forName("oracle.jdbc.driver.OracleDriver");

    Connection con=DriverManager.getConnection(
        "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
    DatabaseMetaData dbmd=con.getMetaData();

    System.out.println("Driver Name: "+dbmd.getDriverName());
    System.out.println("Driver Version: "+dbmd.getDriverVersion());
    System.out.println("UserName: "+dbmd.getUserName());
    System.out.println("Database Product Name: "+dbmd.getDatabaseProductName());
    System.out.println("Database Product Version:
        "+dbmd.getDatabaseProductVersion());

    con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

# DatabaseMetaData interface that prints all table names

---

```
import java.sql.*;
class Dbmd2{
public static void main(String args[]){
try{
    Class.forName("oracle.jdbc.driver.OracleDriver");

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

    DatabaseMetaData dbmd=con.getMetaData();
    String table[]={"TABLE"};
    ResultSet rs=dbmd.getTables(null,null,null,table);

    while(rs.next()){
        System.out.println(rs.getString(3));
    }

    con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

## DatabaseMetaData interface that prints total number of views

---

```
import java.sql.*;
class Dbmd3{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");

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

DatabaseMetaData dbmd=con.getMetaData();
String table[]={"VIEW"};
ResultSet rs=dbmd.getTables(null,null,null,table);

while(rs.next()){
System.out.println(rs.getString(3));
}

con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

# Example of storing image into Database

---

```
CREATE TABLE "IMGTABLE"
(  "NAME" VARCHAR2(4000),
  "PHOTO" BLOB
)

import java.sql.*;
import java.io.*;
public class InsertImage {
    public static void main(String[] args) {
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

            PreparedStatement ps=con.prepareStatement("insert into imgtable values(?,?)");
            ps.setString(1,"Pawan");

            FileInputStream fin=new FileInputStream("d:\\g.jpg");
            ps.setBinaryStream(2,fin,fin.available());
            int i=ps.executeUpdate();
            System.out.println(i+" records affected");

            con.close();
        }catch (Exception e) {e.printStackTrace();}
    }
}
```

# Retrieving image from Oracle database

---

```
import java.sql.*;
import java.io.*;
public class RetrieveImage {
    public static void main(String[] args) {
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

            PreparedStatement ps=con.prepareStatement("select * from imgtable");
            ResultSet rs=ps.executeQuery();
            if(rs.next()){//now on 1st row

                Blob b=rs.getBlob(2);//2 means 2nd column data
                byte barr[]=b.getBytes(1,(int)b.length());//1 means first image

                FileOutputStream fout=new FileOutputStream("d:\\sonoo.jpg");
                fout.write(barr);

                fout.close();
            }//end of if
            System.out.println("ok");

            con.close();
        }catch (Exception e) {e.printStackTrace(); }
    }
}
```

# Storing file in a database

---

```
import java.io.*;
import java.sql.*;

public class StoreFile {
    public static void main(String[] args) {
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

            PreparedStatement ps=con.prepareStatement(
                "insert into filetable values(?,?)");

            File f=new File("d:\\myfile.txt");
            FileReader fr=new FileReader(f);

            ps.setInt(1,101);
            ps.setCharacterStream(2,fr,(int)f.length());
            int i=ps.executeUpdate();
            System.out.println(i+" records affected");

            con.close();

        }catch (Exception e) {e.printStackTrace();}
    }
}
```

```
CREATE TABLE "FILETABLE"
(
    "ID" NUMBER,
    "NAME" CLOB
)
/
```

# Retrieve file from database

---

```
import java.io.*;
import java.sql.*;

public class RetrieveFile {
    public static void main(String[] args) {
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

            PreparedStatement ps=con.prepareStatement("select * from filetable");
            ResultSet rs=ps.executeQuery();
            rs.next();//now on 1st row
            Clob c=rs.getClob(2);
            Reader r=c.getCharacterStream();
            FileWriter fw=new FileWriter("d:\\retrivefile.txt");
            int i;
            while((i=r.read())!=-1)
                fw.write((char)i);

            fw.close();
            con.close();

            System.out.println("success");
        }catch (Exception e) {e.printStackTrace(); }
    }
}
```

# CallableStatement Interface

---

- CallableStatement interface is used to call the **stored procedures and functions**.
- We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.
- Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

## How to get the instance of CallableStatement?

---

```
CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");
```

## Stored Procedure example

---

```
create table myuser(id number(10), name varchar2(200));
```

```
create or replace procedure "INSERTR"  
(id IN NUMBER, name IN VARCHAR2)  
is  
begin  
    insert into myuser values(id,name);  
end;
```

## Calling procedure from JDBC Example

---

```
import java.sql.*;
public class Proc {
    public static void main(String[] args) throws Exception{

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

        CallableStatement stmt=con.prepareCall("{call insertR(?,?)}");
        stmt.setInt(1,1011);
        stmt.setString(2,"Amit");
        stmt.execute();

        System.out.println("success");
    }
}
```

## Function example...

---

```
create or replace function sum4  
(n1 in number,n2 in number)  
return number  
is  
temp number(8);  
begin  
    temp :=n1+n2;  
    return temp;  
end;
```

# Calling function from JDBC Example

---

```
import java.sql.*;

public class FuncSum {
    public static void main(String[] args) throws Exception{

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

        CallableStatement stmt=con.prepareCall("{?= call sum4(?,?)}");
        stmt.setInt(2,10);
        stmt.setInt(3,43);
        stmt.registerOutParameter(1,Types.INTEGER);
        stmt.execute();

        System.out.println(stmt.getInt(1));

    }
}
```

***Types class defines many constants such as INTEGER, VARCHAR, FLOAT, DOUBLE, BLOB, CLOB etc.***

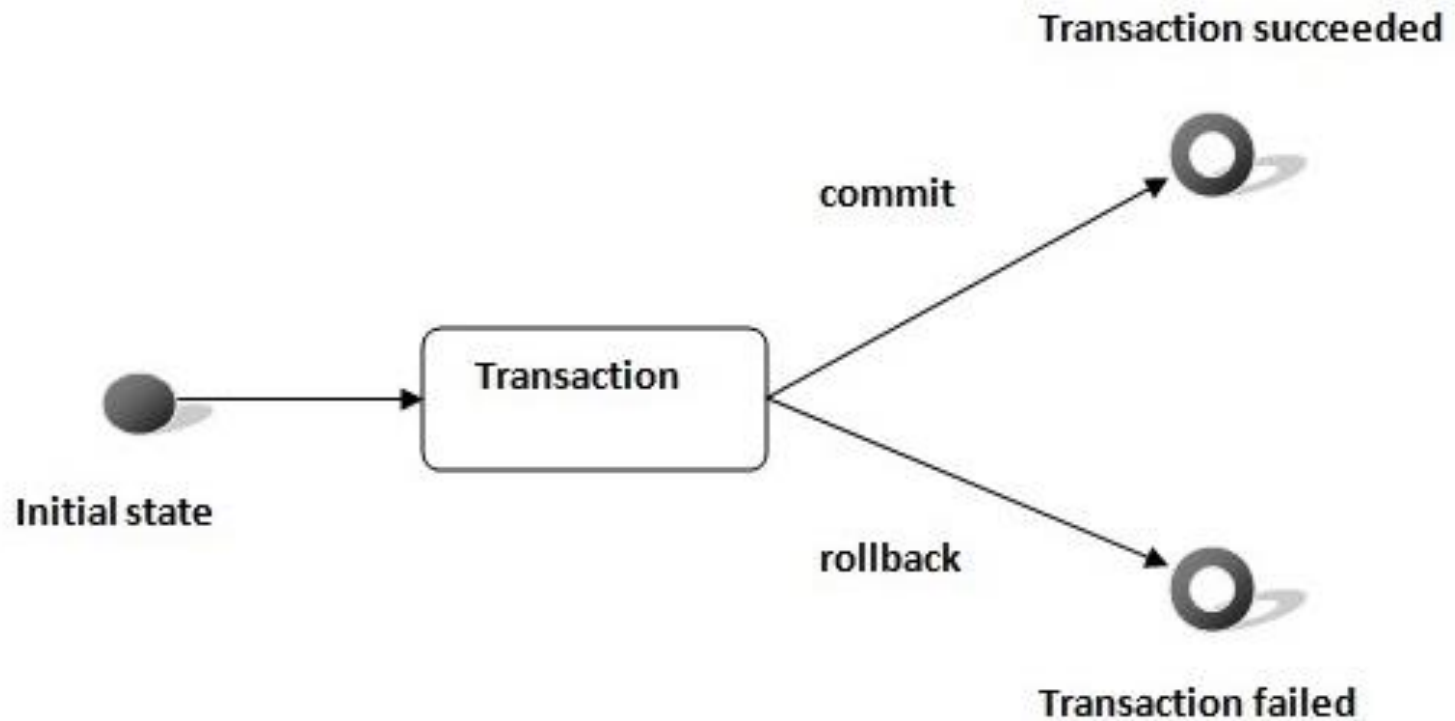
# Transaction Management in JDBC

---

- Transaction represents **a single unit of work**.
- ACID properties:
  1. **Atomicity** means either all successful or none.
  2. **Consistency** ensures bringing the database from one consistent state to another consistent state.
  3. **Isolation** ensures that transaction is isolated from other transaction.
  4. **Durability** means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

# Commit and rollback

---



# Transaction management using JDBC

---

In JDBC, **Connection interface** provides methods to manage transaction.

Method	Description
void setAutoCommit(boolean status)	It is true by default means each transaction is committed by default.
void commit()	commits the transaction.
void rollback()	cancels the transaction.

# Example of transaction management

---

```
import java.sql.*;
class FetchRecords{
public static void main(String args[])throws Exception{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection
    con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe"
    ,"system","oracle");
    con.setAutoCommit(false);

    Statement stmt=con.createStatement();
    stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");
    stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");

    con.commit();
    con.close();
}}
```

# Batch Processing in JDBC

---

- Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast.
- The `java.sql.Statement` and `java.sql.PreparedStatement` interfaces provide methods for batch processing.

Method	Description
<code>void addBatch(String query)</code>	It adds query into batch.
<code>int[] executeBatch()</code>	It executes the batch of queries.

# Batch Processing example using Statement

---

```
import java.sql.*;
class FetchRecords{
public static void main(String args[])throws Exception{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection
    con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system",
    "oracle");
    con.setAutoCommit(false);

    Statement stmt=con.createStatement();
    stmt.addBatch("insert into myuser values(190,'abhi',40000)");
    stmt.addBatch("insert into myuser values(191,'umesh',50000)");

    stmt.executeBatch();//executing the batch

    con.commit();
    con.close();
}}
```

## Exercise

---

Write a batch processing program using PreparedStatement to update multiple values entered by the user.

## JDBC RowSet (Introduced in JDK 5)

---

- Wrapper of ResultSet.
- Easy and flexible to use
- It is Scrollable and Updatable by default

# RowSet example

---

```
public class RowSetExample {
    public static void main(String[] args) throws Exception {
        Class.forName("oracle.jdbc.driver.OracleDriver");

        //Creating and Executing RowSet
        JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
        rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
        rowSet.setUsername("system");
        rowSet.setPassword("oracle");

        rowSet.setCommand("select * from emp400");
        rowSet.execute();

        while (rowSet.next()) {
            // Generating cursor Moved event
            System.out.println("Id: " + rowSet.getString(1));
            System.out.println("Name: " + rowSet.getString(2));
            System.out.println("Salary: " + rowSet.getString(3));
        }
    }
}
```

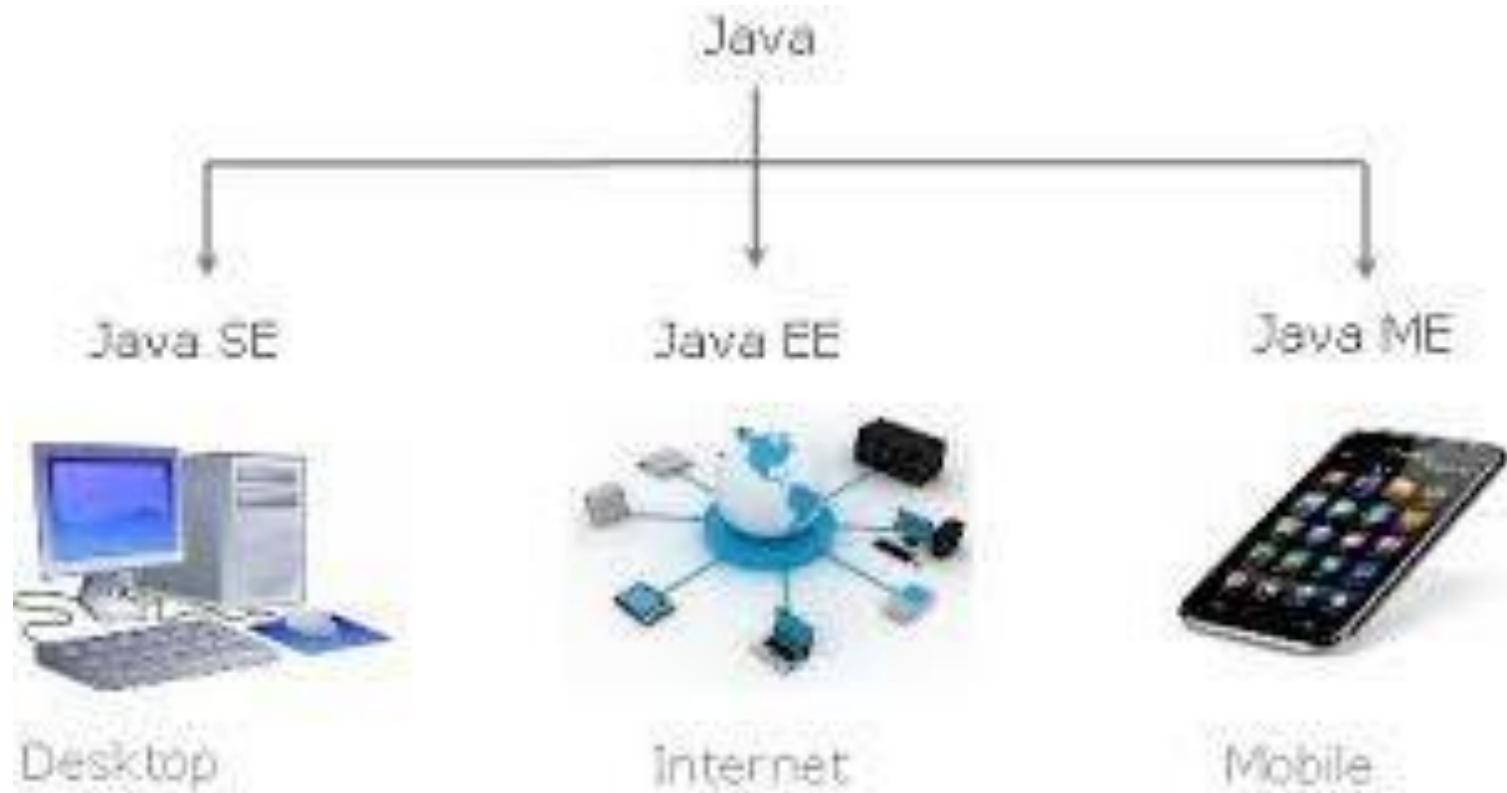
# **Java & JEE Training**

**Introduction to JEE**

**MindsMapped Consulting**

# Re-look at the Java Major Editions

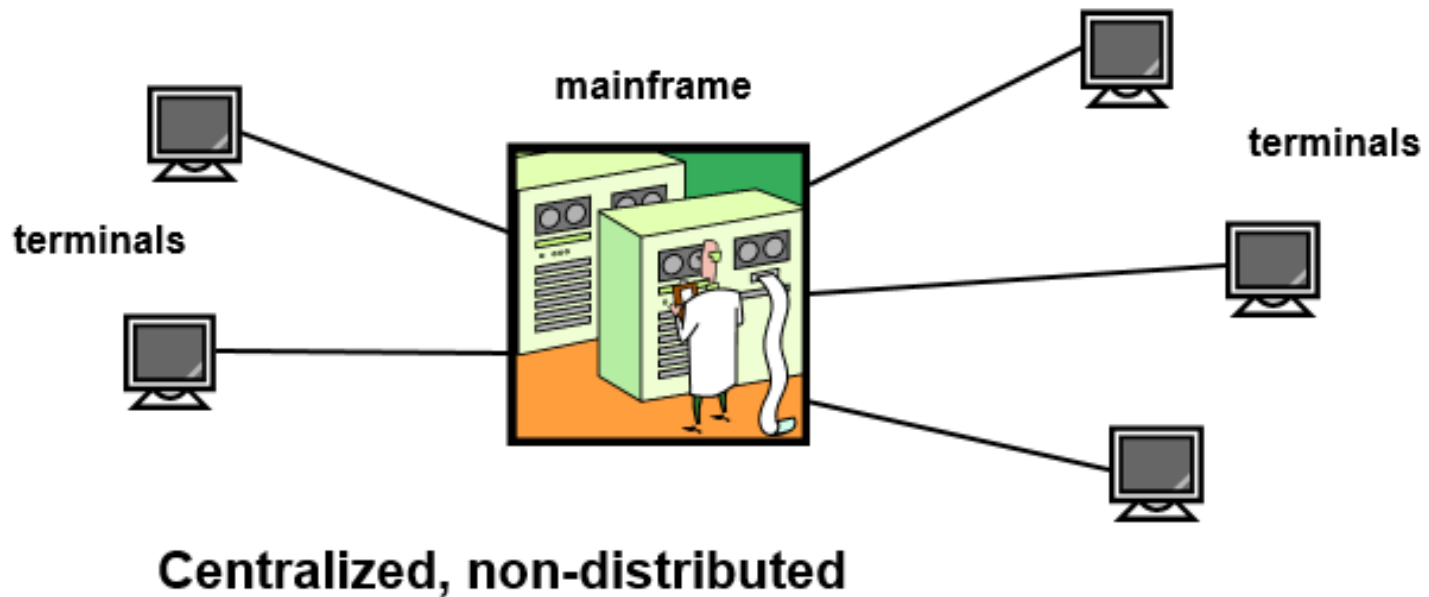
---



# Application Servers

---

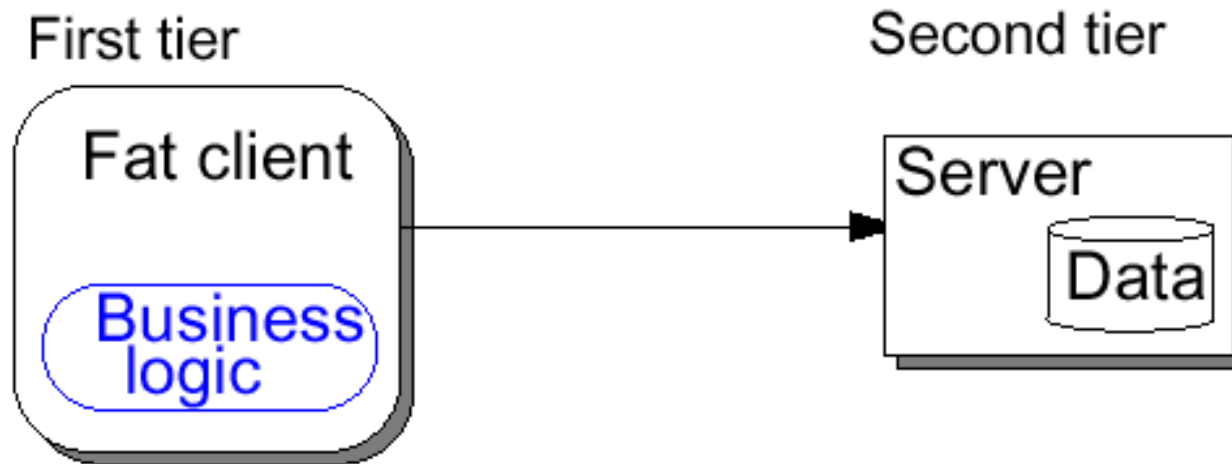
- *In the beginning, there was darkness and cold. Then, ...*



# Application Servers

---

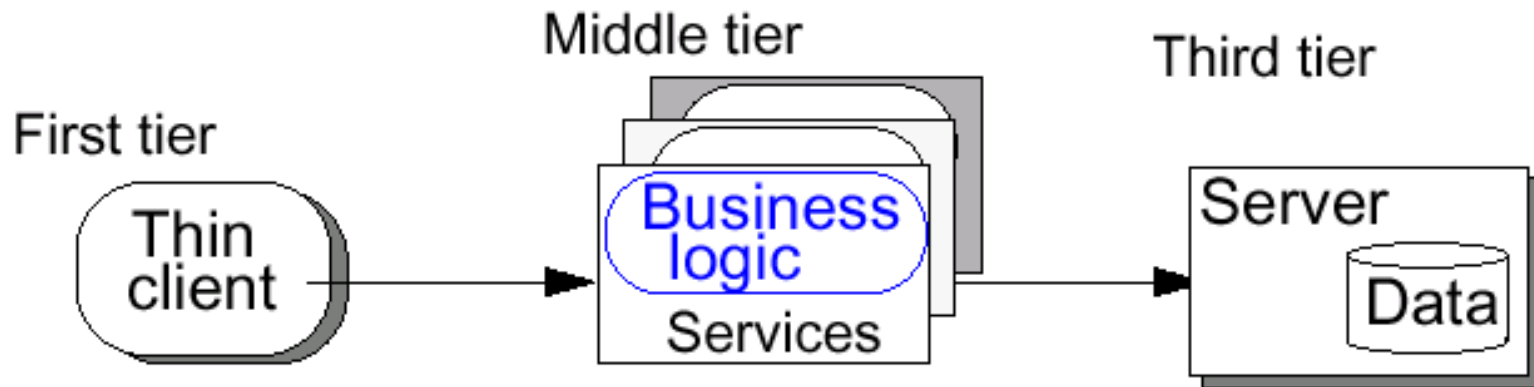
- In the 90's, systems should be *client-server*



# Application Servers

---

- Today, enterprise applications use the *multi-tier* model



# Application Servers

---

- “Multi-tier applications” have several independent components
- An *application server* provides the infrastructure and services to run such applications

# Application Servers

---

- Application server products can be separated into 3 categories:
  - JEE-based solutions
  - Non-JEE solutions (PHP, ColdFusion, Perl, etc.)
  - And the Microsoft solution (ASP/COM and now .NET with ASP.NET, VB.NET, C#, etc.)

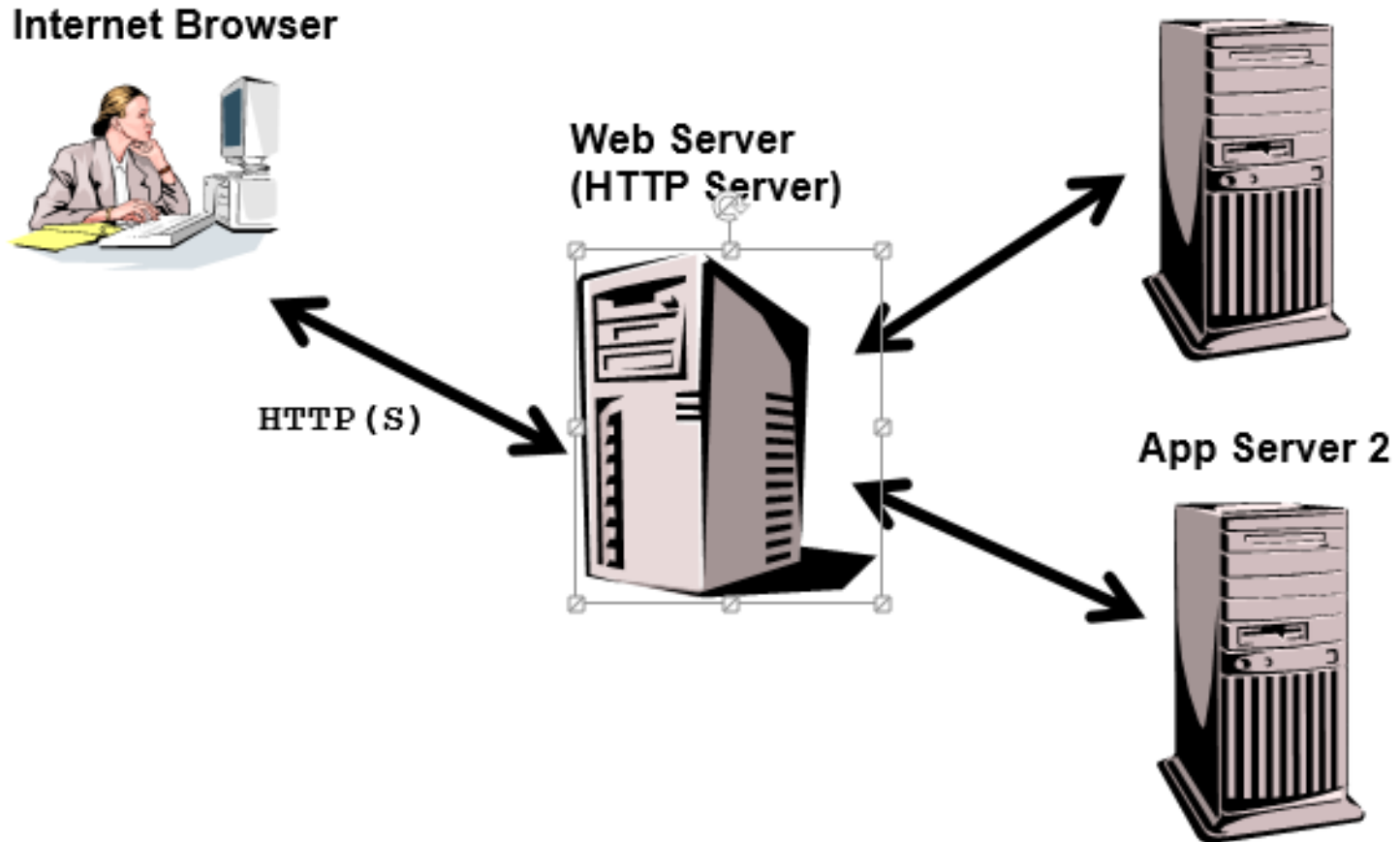
# J2EE Application Servers

---

- Major J2EE products:
  - BEA WebLogic
  - IBM WebSphere
  - Sun iPlanet Application Server
  - Oracle 9iAS
  - HP/Bluestone Total-e-Server
  - Borland AppServer
  - JBoss (free open source)

# Web Server and Application Server

---



# What is JEE?

---

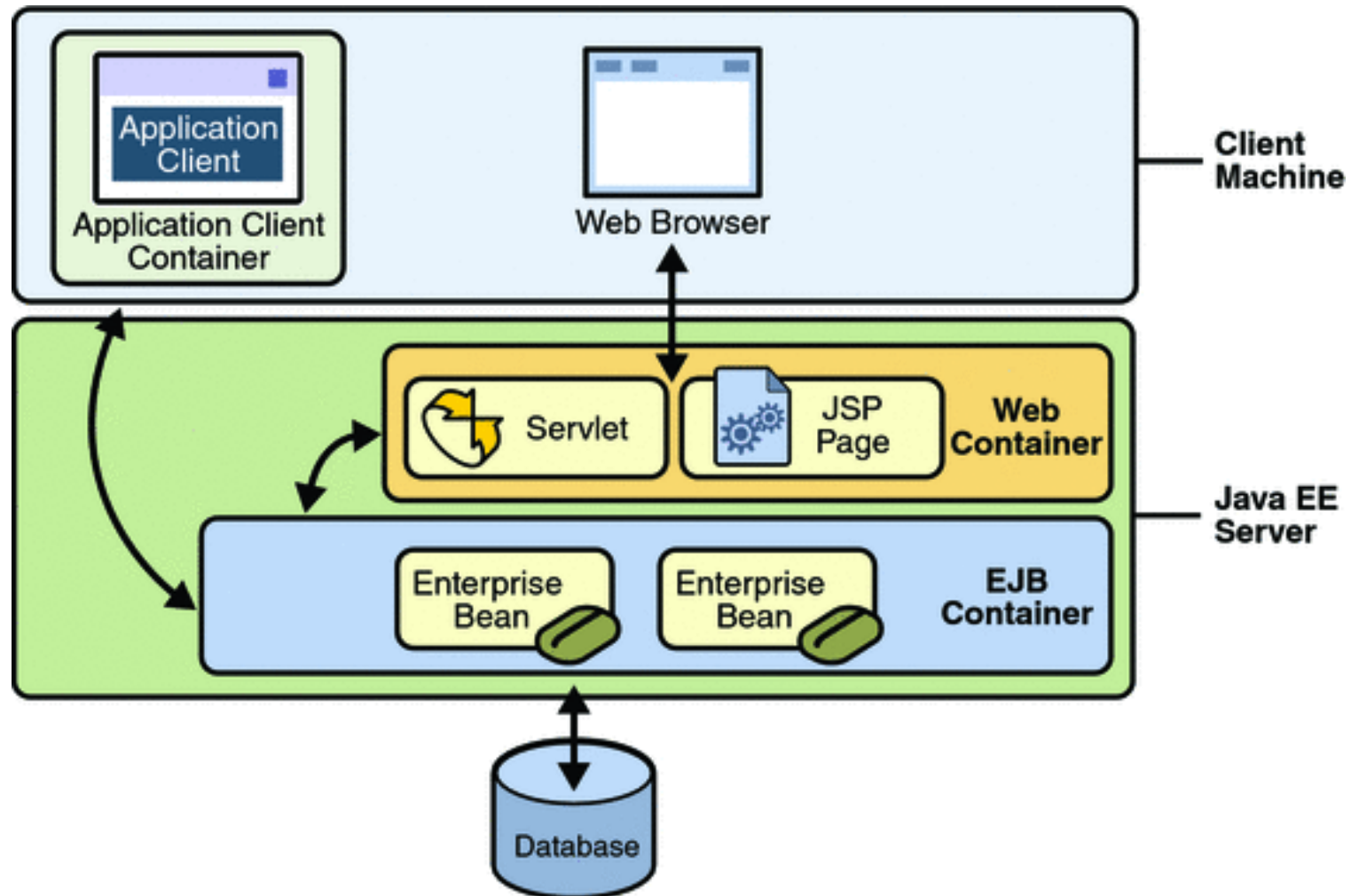
- It is a public specification that embodies several technologies
- Current version is JEE 6
- J2EE defines a model for developing multi-tier, web based, enterprise applications with distributed components
- Benefits:
  - High availability
  - Scalability
  - Integration with existing systems
  - Freedom to choose vendors of application servers, tools, components
  - Multi-platform

# Main technologies

---

- JavaServer Pages (JSP)
- Servlet
- Enterprise JavaBeans (EJB)
- Java Server Faces (JSF) – Not covered in this course.

# Enterprise Java – Multi-tier architecture



- Used for web pages with dynamic content
- Processes HTTP requests (non-blocking call-and-return)
- Accepts HTML tags, special JSP tags, and scriptlets of Java code
- Separates static content from presentation logic
- Can be created by web designer using HTML tools

# Servlet

---

- Used for web pages with dynamic content
- Processes HTTP requests (non-blocking call-and-return)
- Written in Java; uses print statements to render HTML
- Loaded into memory once and then called many times
- Provides APIs for session management

- EJBs are *distributed components* used to implement business logic (no UI)
- Developer concentrates on business logic
- Availability, scalability, security, interoperability and integrability handled by the J2EE server
- Client of EJBs can be JSPs, servlets, other EJBs and external applications
- Clients see *interfaces*