JAVA PROGRAMMING

CONTENTS

Contents

- JDBC Architecture
- JDBC Drivers
- JDBC API – Loading a driver, Connecting to the DB
- Creating and Executing JDBC Statements
- Handling SQL Exceptions
- ResultSet Interface
- An Example Application for JDBC
- JSP and Servlets

Database

- A database is a systematic collection of data.
- They support electronic storage and manipulation of data.
- Databases make data management easy
- **Collection of Tables is DB which stores inter-related data**

Some Terminologies

- **Table** – Collection of Records
- **Record** – Collection of Attributes

Types in DB

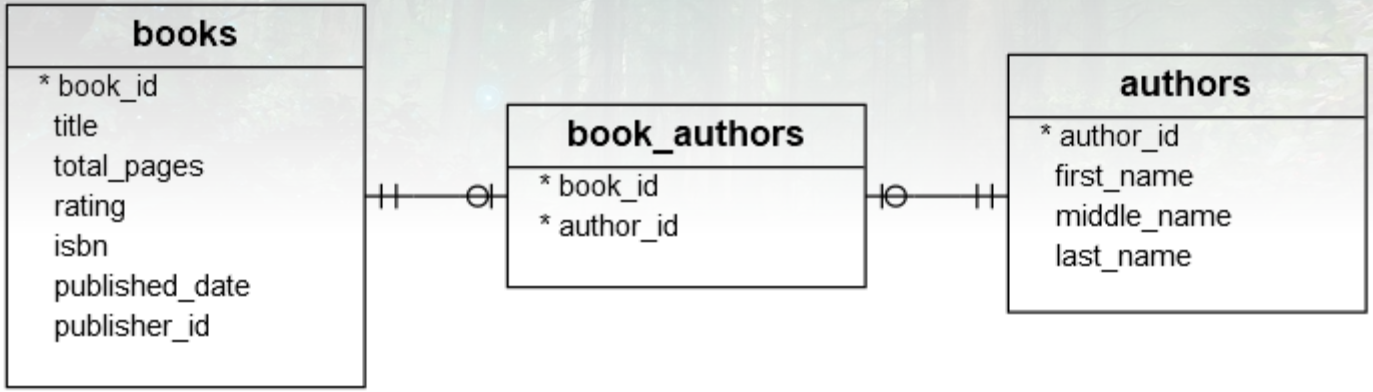
MySQL
mongoDB
amazon REDSHIFT
SQL
IBM
Google BigQuery
druid
ORACLE
cassandra
snowflake

Example for Table

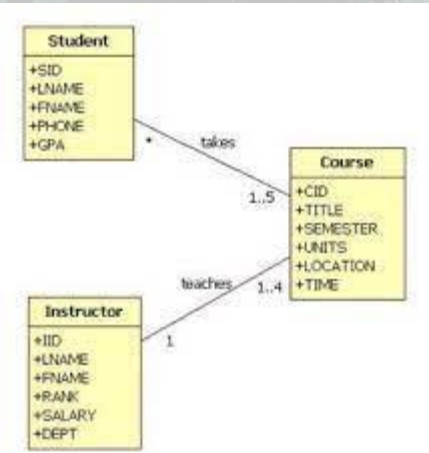
Table: customers

customer_id	first_name	last_name	phone	country
1	John	Doe	817-646-8833	USA
2	Robert	Luna	412-862-0502	USA
3	David	Robinson	208-340-7906	UK
4	John	Reinhardt	307-242-6285	UK
5	Betty	Taylor	806-749-2958	UAE

Example for Books Database



Example for College Database



FACULTY	
ID	NAME
1	MR. ALPHA
2	MR. BETA
3	MR. GAMMA
4	MR. DELTA
5	MR. EPSILON

STUDENTS	
ID	NAME
1	ALICE
2	BOB

CLASSES	
ID	NAME
1	CALCULUS
2	BIOLOGY

DEGREES	
ID	DEPARTMENT
1	MATHEMATICS

ADVISES		
ID	TEACHER ID	STUDENT ID
1	2	1
2	5	2

TEACHES		
ID	TEACHER ID	CLASS ID
1	2	1
2	3	1
3	5	2

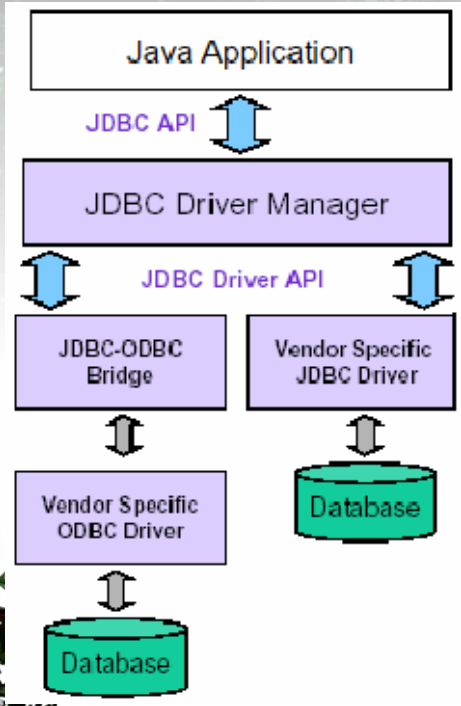
ENROLLMENTS		
ID	STUDENT ID	DEGREE ID
1	1	1

REGISTRATIONS		
ID	STUDENT ID	CLASS ID
1	1	1
2	2	2

JDBC

- JDBC stands for Java Database Connectivity.
- JDBC is a Java API to connect and execute the query with the database.
- It is a part of JavaSE (Java Standard Edition).
- JDBC API uses JDBC drivers to connect with the database.
- It provides a standard library for Java programs to connect to a database and send it commands using SQL
- It generalizes common database access functions into a set of common classes and methods
- Abstracts vendor specific details into a code library making the connectivity to multiple databases transparent to user

JDBC Architecture



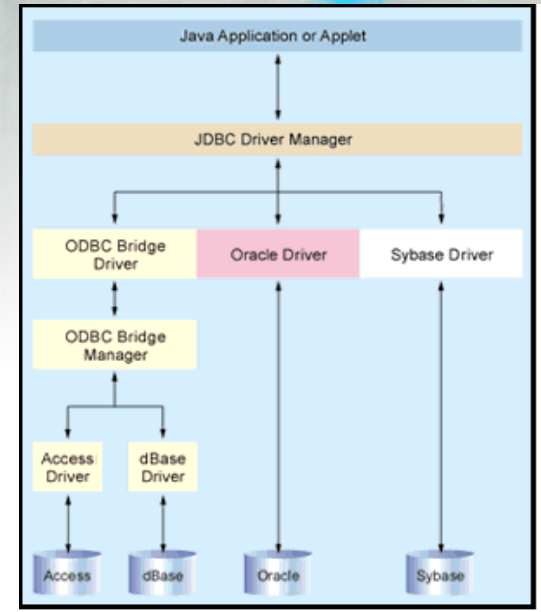


Figure 1. Anatomy of Data Access. The Driver Manager provides a consistent layer between your Java app and back-end database. JDBC works natively (such as with the Oracle driver in this example) or with any ODBC datasource.

JDBC Drivers

There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver –

  ODBC Bridge
- Native Driver

  Native API, Partially Java
- Network Protocol Driver

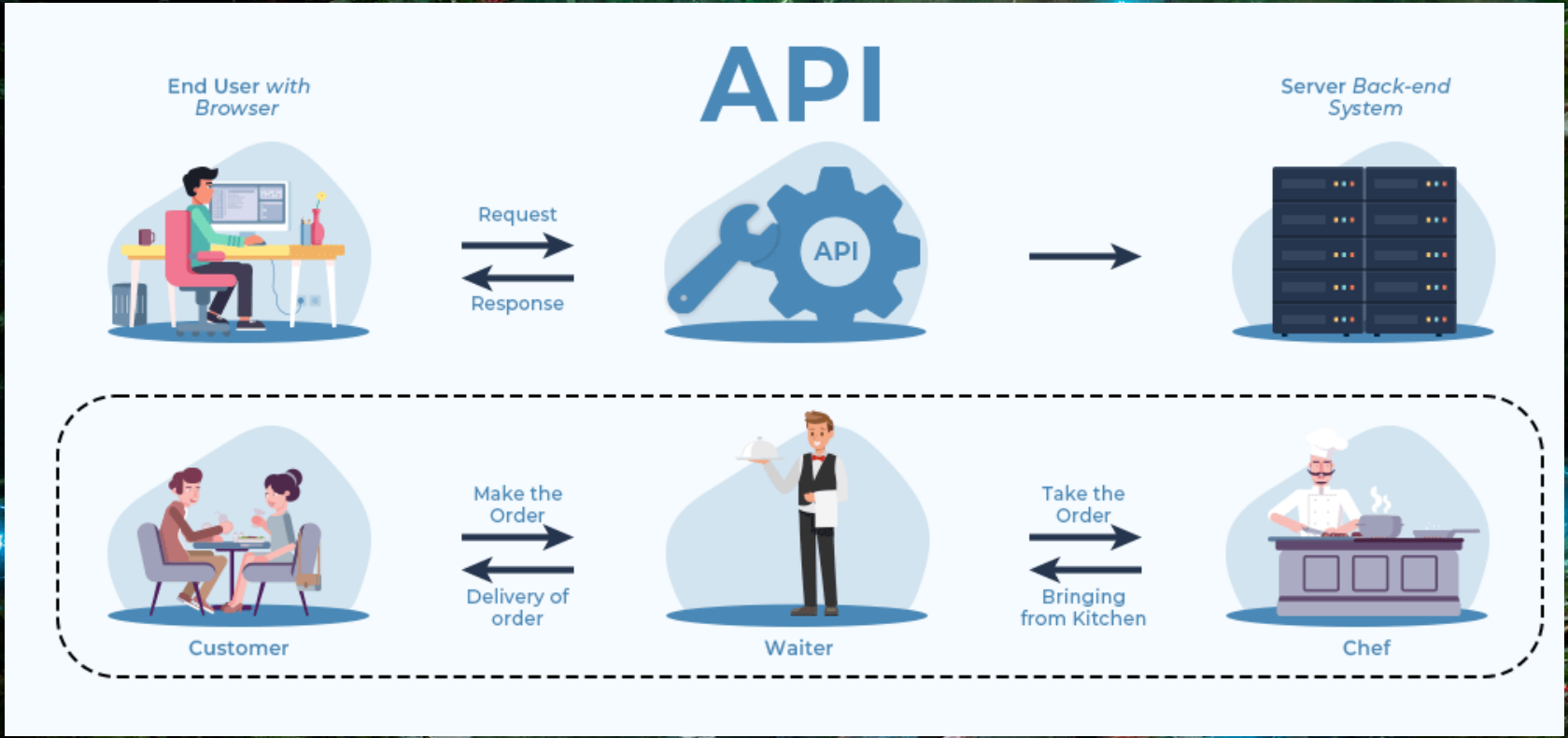
  Middleware, Networking
- Thin Driver

  Type IV(pure java driver) - 100% Java

Application Programming Interface (API)

- It is a set of routines, protocols and tools for building software and applications.
- API stands for application program interface.
- A programmer writing an application program can make a request to the Operating System using API

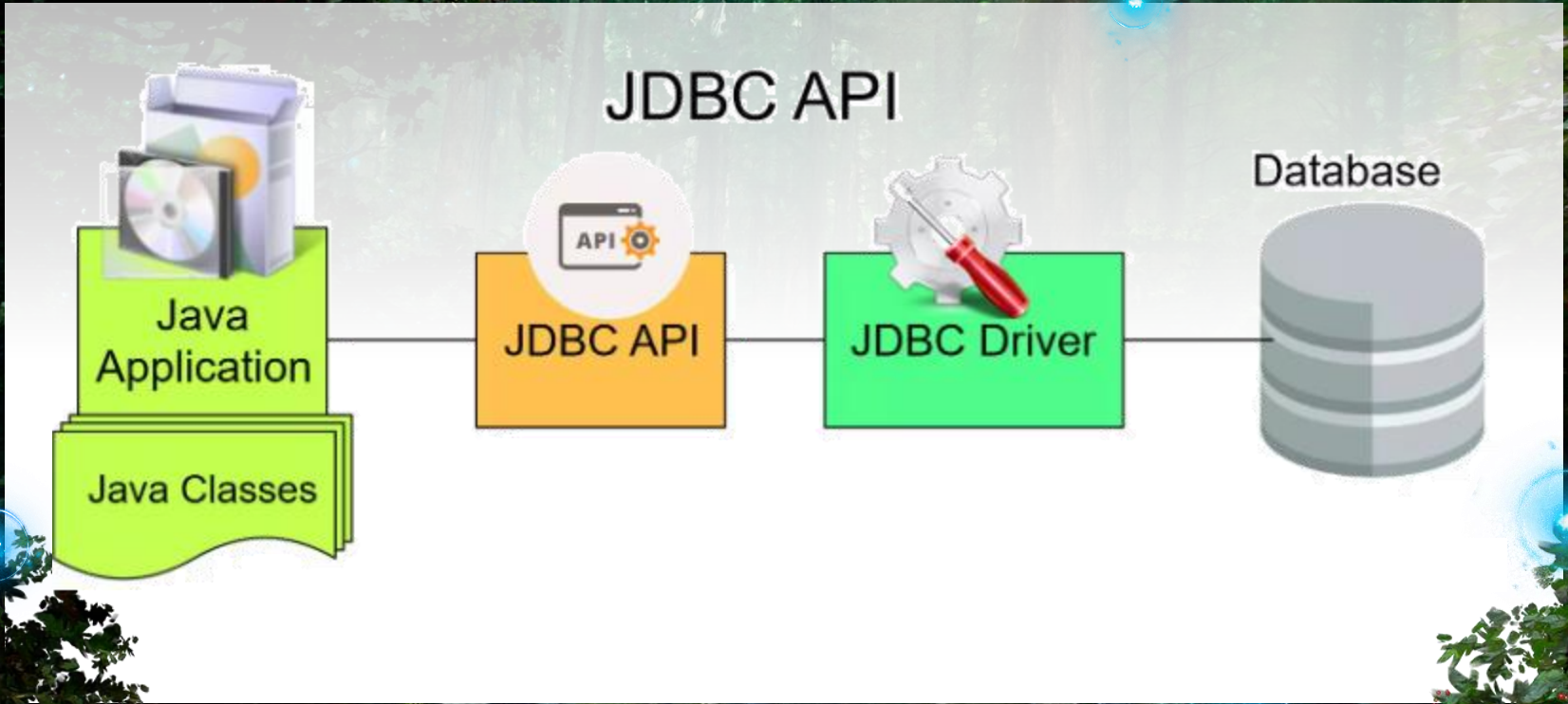
Application Programming Interface (API)



JDBC API

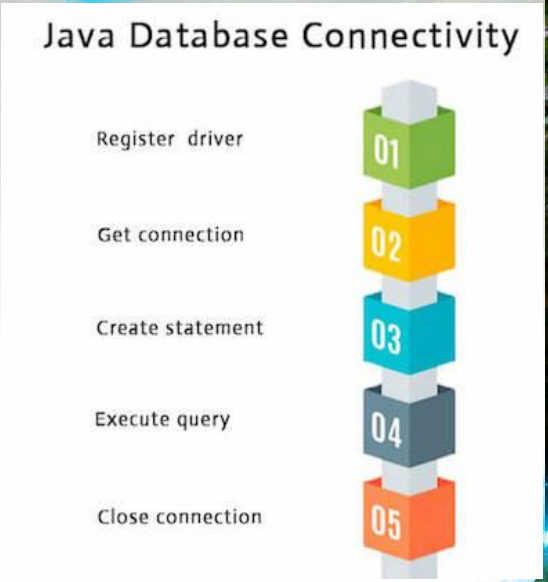
- Way to establish connection to database
- Approach to initiating queries
- Method to create stored procedures
- Data structure of the query result

JDBC API


JDBC API
Java Application
Java Classes
API
JDBC API
JDBC Driver
Database

Steps for Creating the Database with Java

1. Import the necessary Packages (**import java.sql.*;**)
2. Load the drivers using the forName() method
3. Register the drivers using DriverManager
4. Establish a connection using the Connection class object
5. Create a statement for Query
6. Execute the query
7. Close the connections



Importing the Package

import java.sql.*;

Import is a keyword
SQL – Structured Query Language is used to DB

Load the Driver

load the driver's class file into memory at the runtime

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Class is a keyword
forName is a method to load the driver

Connection Object Creation

Connection Object Creation

Connection con = DriverManager.getConnection(url,user,password)

user: Username from which your SQL command prompt can be accessed.
password: password from which the SQL command prompt can be accessed.
con: It is a reference to the Connection interface.
Url: Uniform Resource Locator
String **url** = **“jdbc:odbc:thin:@localhost:8080”**

Create a Statement

- Once a connection is established you can interact with the database.
- The JDBCStatement, CallableStatement, and PreparedStatement interfaces define the methods that enable you to send SQL commands and receive data from user database.

Use of JDBC Statement is as follows:

- **Statement st = con.createStatement();**

Execute the Query

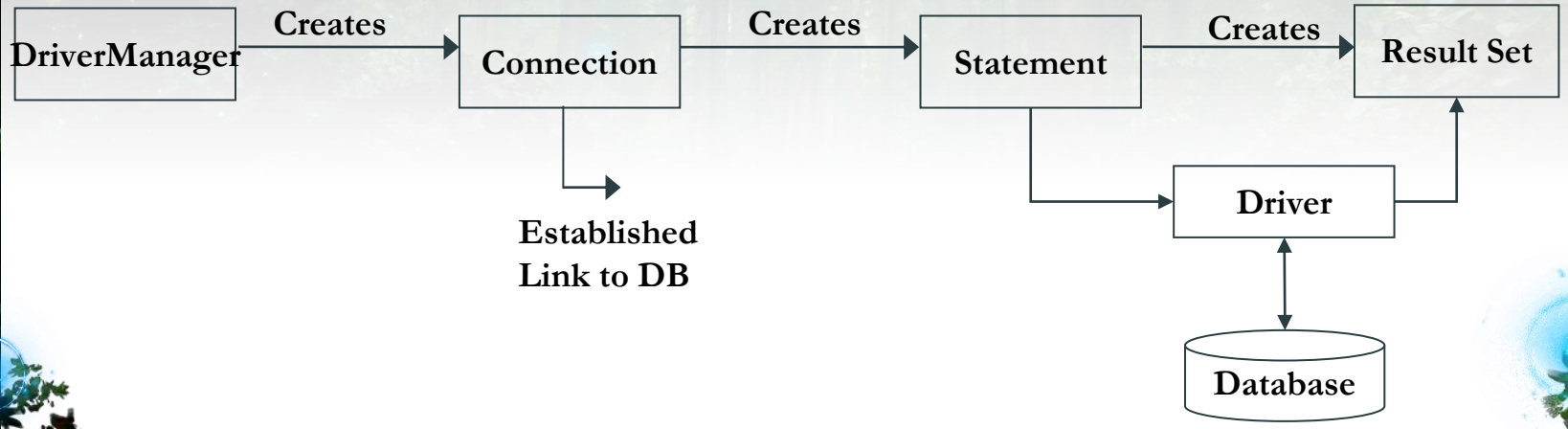
Execute the query

- The query here is an SQL Query.

Some of the Queries

- The query for updating/inserting a table in a database.
- The query for retrieving data.
- The executeQuery() method of the Statement interface is used to execute queries of retrieving values from the database.

Recall Steps



Recall Steps

Driver Manager: Loads database drivers and manages connections between the application and the driver
Driver: Translates API calls into operations for specific database
Connection: Session between application and data source
Statement: SQL statement to perform query or update
Metadata: Information about returned data, database, & driver
Result Set: Logical set of columns and rows of data returned by executing a statement

Example for Table Creation

```
import java.sql.*;
public class Acs
{
 public static void main(String args[])
   {
      String url = "jdbc:odbc:Dharma";
      Connection con;
      Statement stmt;
      String createString;

  createString = "create table Alex" +
                 "(COF_NAME VARCHAR(32), " +
                 "SUP_ID INTEGER, " +
                 "PRICE FLOAT, " +
                 "SALES INTEGER, " +
                 "TOTAL INTEGER)";
```

```
try
    {
     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }catch(java.lang.ClassNotFoundException e)
     {
      System.err.print("ClassNotFoundException: ");
      System.err.println(e.getMessage());
    }
```

Example for Table Creation

```
try
 {
  con = DriverManager.getConnection(url, "", "");
  stmt = con.createStatement();
  stmt.executeUpdate(createString);
  stmt.close();
  con.close();
  System.out.println("Table created.");
 } catch(SQLException ex)
   {
    System.err.println("SQLException: " + ex.getMessage());
   }

}
}
```

```
F:\pack>javac Acs.java

F:\pack>java Acs
Table created.
```

Example for Insertion

```
import java.sql.*;
public class InsertTable
{
 public static void main(String args[])
   {
      String url = "jdbc:odbc:Dharma";
      Connection con;
      Statement stmt;
      String createString="";

  try
  {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}catch(java.lang.ClassNotFoundException e)
   {
    System.err.print("ClassNotFoundException: ");
    System.err.println(e.getMessage());
   }
```

```
F:\pack>javac InsertTable.java

F:\pack>java InsertTable
Row(s) created.
```

Example for Insertion

```
try
{
 con = DriverManager.getConnection(url, "", "");
 stmt = con.createStatement();
 String sql = "INSERT INTO Bank VALUES ('ram', '44444', '100000000', 'Canara')";
 stmt.executeUpdate(sql);
 sql = "INSERT INTO Bank VALUES ('lakshan', '55555', '900000000', 'SBI')";
stmt.executeUpdate(sql);
 stmt.close();
 con.close();
 System.out.println("Row(s) created.");
}catch(SQLException ex)
 {
  System.err.println("SQLException: " + ex.getMessage());
 }
}
}
```

Bank

AccHolderN	AccNo	Balance	BankName
ram1	44444	100000000	Canara
ram2	44444	100000000	Canara
ram	44444	100000000	Canara
lakshan	55555	900000000	SBI
ram	44444	100000000	Canara
lakshan	55555	900000000	SBI

Example for Update

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

  String url = "jdbc:odbc:Dharma";
  Connection con;
  Statement stmt;
  String sqlString;

  try
  {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  }catch(java.lang.ClassNotFoundException e)
   {
    System.err.print("ClassNotFoundException: ");
    System.err.println(e.getMessage());
   }
```

```
F:\pack>javac Update.java

F:\pack>java Update
Row(s) updated.
```

Bank

AccHolderN	AccNo	Balance	BankName
ram1	56789	100000000	Canara
ram2	44444	100000000	Canara
ram	44444	100000000	Canara
lakshan	55555	900000000	SBI
ram1	56789	100000000	Canarasdakjfb
lakshan1	293482394	900000000	SBIsahdfkjsd
ram	44444	100000000	Canara
lakshan	55555	900000000	SBI
Sayan	1234	1000000	ICICI
UTPAL	5678	900000	AXIS

Example for Update

```
try
{

 con = DriverManager.getConnection(url, "", "");
 stmt = con.createStatement();
 sqlString ="UPDATE Bank SET AccNo=56789 WHERE AccHolderName ='ram1' ";
 stmt.executeUpdate(sqlString);
 stmt.close();
 con.close();
 System.out.println("Row(s) updated.");

}catch(SQLException ex)
 {
  System.err.println("SQLException: " + ex.getMessage());
 }
}
}
```

Example for Delete

```
import java.sql.*;
public class Delete
{
 public static void main(String args[])
 {
  String url = "jdbc:odbc:Dharma";
  Connection con;
  Statement stmt;
  String sqlString;

  try
  {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  }catch(java.lang.ClassNotFoundException e)
   {
    System.err.print("ClassNotFoundException: ");
    System.err.println(e.getMessage());
   }
  try
  {
   con = DriverManager.getConnection(url, "", "");
   stmt = con.createStatement();
   sqlString ="delete from Bank where AccNo='1234'";
   stmt.executeUpdate(sqlString);
   stmt.close();
   con.close();
   System.out.println("Row(s) deleted.");
  }

 catch(SQLException ex)
   {
    System.err.println("SQLException: " + ex.getMessage());
   }
 }
}
```

Example for Delete

```
F:\pack>javac Delete.java

F:\pack>java Delete
Row(s) deleted.
```

Bank

AccHolderN	AccNo	Balance	BankName
ram1	56789	100000000	Canara
ram2	44444	100000000	Canara
ram	44444	100000000	Canara
lakshan	55555	900000000	SBI
ram1	56789	100000000	Canarasdakjfb
lakshan1	293482394	900000000	SBIsahdfkjsd
ram	44444	100000000	Canara
lakshan	55555	900000000	SBI
UTPAL	5678	900000	AXIS
Dummy	1234	10000	ICICI

Bank

AccHolderN	AccNo	Balance	BankName
ram1	56789	100000000	Canara
ram2	44444	100000000	Canara
ram	44444	100000000	Canara
lakshan	55555	900000000	SBI
ram1	56789	100000000	Canarasdakjfb
lakshan1	293482394	900000000	SBIsahdfkjsd
ram	44444	100000000	Canara
lakshan	55555	900000000	SBI
UTPAL	5678	900000	AXIS

ResultSet Interface (Selection)

- Java **ResultSet** interface is a part of the **java.sql** package.
- It is one of the core components of the JDBC Framework.
- ResultSet Object is used to access query results retrieved from the relational databases.
- ResultSet maintains cursor/pointer which points to a single row of the query results.

ResultSet Interface Example for Employee

```
import java.sql.*;
public class ResultSetExp {
        public static void main(String[] args) {
                String query = "select empid, firstname, lastname, dob from Emp";
                Connection conn = null;
                Statement stmt = null;
                try {
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                        conn = DriverManager.getConnection("jdbc:odbc:Sample", "", "");
                        stmt = conn.createStatement();
                        ResultSet rs = stmt.executeQuery(query);
                        while (rs.next()) {
                                Integer empId = rs.getInt(1);
                                String firstName = rs.getString(2);
                                String lastName = rs.getString(3);
                                Date dob = rs.getDate(4);
                                System.out.println("empId:" + empId);
                                System.out.println("firstName:" + firstName);
                                System.out.println("lastName:" + lastName);
                                System.out.println("dob:" + dob);
```

Emp

EmpID	FirstName	LastName	DOB
100	Alex	Krishna	9/7/1990
101	Mani	Narayan	9/20/1986
102	Dharma	Krishna	7/25/1984
103	Ram	Krishna	6/7/1991

ResultSet Interface Example for Employee

```
F:\pack>java ResultSetExp
empId:100
firstName:Alex
lastName:Krishna
dob:1990-09-07

empId:101
firstName:Mani
lastName:Narayan
dob:1986-09-20

empId:102
firstName:Dharma
lastName:Krishna
dob:1984-07-25

empId:103
firstName:Ram
lastName:Krishna
dob:1991-06-07
```

```
                System.out.println("empId:" + empId);
                System.out.println("firstName:" + firstName);
                System.out.println("lastName:" + lastName);
                System.out.println("dob:" + dob);
                System.out.println("");
            }
            rs.close();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                stmt.close();
                conn.close();
            } catch (Exception e) {}
        }
    }
}
```

Login Details Creation and Insertion (Exercise 1)

```
import java.sql.*;
class JDBCLogin {

  public static void main(String[] args) throws Exception {
    Connection conn = null;
    Statement stmt = null;

    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection("jdbc:odbc:Dharma", "", "");
      System.out.println("Creating login details table...");
      stmt = conn.createStatement();
      String sql = "CREATE TABLE Login_Details" +
                   "(UName VARCHAR(35)," + "Pwd VARCHAR(25))";
      stmt.executeUpdate(sql);
```

Login Details Creation and Insertion (Exercise 1)

```java
        System.out.println("Inserting data into table...");
        sql = "INSERT INTO login_details (uname, pwd) VALUES (?, ?)";
        PreparedStatement preparedStatement = conn.prepareStatement(sql);
        preparedStatement.setString(1, "Suhan".trim());
        preparedStatement.setString(2, "suhan123".trim());
        preparedStatement.executeUpdate();

        preparedStatement.setString(1, "Subi".trim());
        preparedStatement.setString(2, "subi123".trim());
        preparedStatement.executeUpdate();

        preparedStatement.setString(1, "Laksan".trim());
        preparedStatement.setString(2, "Laksh123".trim());
        preparedStatement.executeUpdate();
        System.out.println("Data inserted into table.");
        preparedStatement.close();
        stmt.close();
        conn.close();
    } catch(SQLException se) {
        se.printStackTrace();
    }
  }
}
```

```
F:\pack>javac JDBCLogin.java

F:\pack>java JDBCLogin
Connecting to database...
Creating login details table...
Inserting data into table...
Data inserted into table.
```

Login_Details

UName	Pwd
Suhan	suhan123
Subi	subi123
Laksan	Laksh123

Library Record Insertion and Update (Exercise 2)

```
import java.sql.*;
class LibraryRecord {
    public static void main(String[] args) throws Exception {
        Connection conn = null;
        Statement stmt = null;
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            System.out.println("Connecting to database...");
            conn = DriverManager.getConnection("jdbc:odbc:Dharma", "", "");

            // Insert a book record into the library table
            System.out.println("Inserting a book record into the library table...");
            stmt = conn.createStatement();
```

Library Record Insertion and Update (Exercise 2)

```
        String sql = "INSERT INTO Library (title, author, category, publication_date) VALUES ('The Great Gatsby', 'F. Scott Fitzgerald', 'Classic Literature',
'1925-04-10')";
        int rowsAffected = stmt.executeUpdate(sql);
        System.out.println(rowsAffected + " row(s) inserted.");

        // Update the publication date for a book record in the library table
        System.out.println("Updating the publication date for a book record in the library table...");
        sql = "UPDATE library SET publication_date = '1925-04-11' WHERE title = 'The Great Gatsby'";
        rowsAffected = stmt.executeUpdate(sql);
        System.out.println(rowsAffected + " row(s) updated.");
        stmt.close();
        conn.close();
    } catch(SQLException se) {
        se.printStackTrace();
    }
  }
}
```

```
F:\pack>javac LibraryRecord.java

F:\pack>java LibraryRecord
Connecting to database...
Inserting a book record into the library table...
1 row(s) inserted.
Updating the publication date for a book record in the library table...
1 row(s) updated.
```

Library

title	author	category	publication_
The Great Gatsby	F. Scott Fitzgerald	Classic Literature	1925-04-11

Student Database Details (INSERT,UPDATE, DELETE) (Exercise 3)

```
import java.sql.*;

class StudentDatabase {
    public static void main(String[] args) throws Exception {
        Connection conn = null;
        Statement stmt = null;

        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            System.out.println("Connecting to database...");
            conn = DriverManager.getConnection("jdbc:odbc:Dharma", "", "");

            System.out.println("Inserting a student record into the students table...");
            stmt = conn.createStatement();
            String sql = "INSERT INTO StudDB (student_id, first_name, last_name, major, gpa) VALUES (1001, 'Dhanush', 'Anand', 'Computer Science', 3.5)";
            int rowsAffected = stmt.executeUpdate(sql);
            sql = "INSERT INTO StudDB (student_id, first_name, last_name, major, gpa) VALUES (1002, 'Demo', 'Fun', 'CD', 2.5)";
            rowsAffected = stmt.executeUpdate(sql);
            sql = "INSERT INTO StudDB (student_id, first_name, last_name, major, gpa) VALUES (1003, 'Hai', 'Hello', 'Mech', 2.3)";
            rowsAffected = stmt.executeUpdate(sql);

            System.out.println(rowsAffected + " row(s) inserted.");
```

Student Database Details (INSERT,UPDATE, DELETE) (Exercise 3)

```
System.out.println(rowsAffected + " row(s) inserted.");

System.out.println("Updating the major for a student record in the students table...");
sql = "UPDATE StudDB SET major = 'Software Engineering' WHERE student_id = 1001";
rowsAffected = stmt.executeUpdate(sql);
System.out.println(rowsAffected + " row(s) updated.");

System.out.println("Deleting a student record from the students table...");
sql = "DELETE FROM StudDB WHERE student_id = 1003";
rowsAffected = stmt.executeUpdate(sql);
System.out.println(rowsAffected + " row(s) deleted.");

System.out.println("Selecting all student records from the students table...");
sql = "SELECT * FROM StudDB";
ResultSet rs = stmt.executeQuery(sql);
```

Student Database Details (INSERT,UPDATE, DELETE) (Exercise 3)

```
        while(rs.next()) {
            int id  = rs.getInt("student_id");
            String firstName = rs.getString("first_name");
            String lastName = rs.getString("last_name");
            String major = rs.getString("major");
            double gpa = rs.getDouble("gpa");

            System.out.print("ID: " + id);
            System.out.print(", First Name: " + firstName);
            System.out.print(", Last Name: " + lastName);
            System.out.print(", Major: " + major);
            System.out.println(", GPA: " + gpa);
        }
        rs.close();
        stmt.close();
        conn.close();
    } catch(SQLException se) {
        se.printStackTrace();
    }
}
}
```

StudDB

student_id	first_name	last_name	major	gpa
1001	Dhanush	Anand	Software Engir	4
1002	Demo	Fun	CD	2

Student Database Details (INSERT,UPDATE, DELETE) (Exercise 3)

```
F:\pack>javac StudentDatabase.java

F:\pack>java StudentDatabase
Connecting to database...
Inserting a student record into the students table...
1 row(s) inserted.
Updating the major for a student record in the students table...
1 row(s) updated.
Deleting a student record from the students table...
1 row(s) deleted.
Selecting all student records from the students table...
ID: 1001, First Name: Dhanush, Last Name: Anand, Major: Software Engineering, GPA: 4.0
ID: 1002, First Name: Demo, Last Name: Fun, Major: CD, GPA: 2.0
```

Employee Searching By having ID (Exercise 4)

```
import java.sql.*;
class EmployeeSearch {
    public static void main(String[] args) throws Exception {
     Connection conn = null;
     Statement stmt = null;
     try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        System.out.println("Connecting to database...");
        conn = DriverManager.getConnection("jdbc:odbc:Dharma","", "");

        System.out.println("Searching for employee with ID 1001...");
        stmt = conn.createStatement();
        String sql = "SELECT * FROM Employees WHERE id =1";
        ResultSet rs = stmt.executeQuery(sql);
```

Employee Searching By having ID (Exercise 4)

```
        if(rs.next()) {
            int id  = rs.getInt("id");
            String first = rs.getString("first");
            String last = rs.getString("last");
            int age  = rs.getInt("age");
            System.out.print("ID: " + id);
            System.out.print(", First Name: " + first);
            System.out.print(", Last Name: " + last);
            System.out.println(", Age: " + age);
        } else {
            System.out.println("Employee not found.");
        }
        rs.close();
        stmt.close();
        conn.close();
    } catch(SQLException se) {
        se.printStackTrace();
    }
}
}
```

```
F:\pack>javac EmployeeSearch.java

F:\pack>java EmployeeSearch
Connecting to database...
Searching for employee with ID 1001...
ID: 1, First Name: Jack, Last Name: Smith, Age: 59
```

Employees

id	first	last	age
1	Jack	Smith	59
2	Jsey	Will	60
3	Jaamy	Sandry	61

Stock Update (Exercise 5)

```
import java.sql.*;
class StockUpdate {
    public static void main(String[] args)throws Exception {
        Connection conn = null;
        Statement stmt = null;
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            System.out.println("Connecting to database...");
            conn = DriverManager.getConnection("jdbc:odbc:Dharma", "", "");
            System.out.println("Updating new item arrival...");
            stmt = conn.createStatement();
            String sql = "UPDATE Stock SET Quantity = Quantity + 50 WHERE Item_Name = 'New_Item'";
            int rowsAffected = stmt.executeUpdate(sql);
            System.out.println(rowsAffected + " row(s) updated.");
            stmt.close();
            conn.close();
        } catch(SQLException se) {
            se.printStackTrace();
        }
    }
}
```

Stock

Quantity	Item_Name
150	New_Item
70	Old_Item

```
F:\pack>javac StockUpdate.java

F:\pack>java StockUpdate
Connecting to database...
Updating new item arrival...
1 row(s) updated.
```

SQL Exception Handling

- SQLException is available in the java.sql package.
- It extends the Exception class which means that user can use the methods available in the Exception class in the SQLException class as well
- Syntax error in the SQL statement may result in SQL Exception. When such an exception occurs, an object of the SQLException class will be passed to the catch block. By using the information in the SQLException object, user can catch that exception and continue the program

SQL Exception Handling Methods

Method Name	Description
getErrorCode()	It returns the error number
getMessage()	It returns the error message
getSQLState()	It returns the SQLState of the SQLException object. It can return null as well. For Database error, it will return XOPEN SQL State
getNextException()	It returns the next exception in the exception chain.
printStackTrace()	It prints the current exception and its backtrace to a standard error stream
setNextException(SQLEXception ex)	It is used to add another SQL exception in the chain

SQL Exception Handling Example

```
import java.sql.*;
public class Excep_Example {
public static void main(String[] args) throws ClassNotFoundException {
String update_query = "update employee_details set email='xyza@gmail.com' where empNum1 = 10011";
Class.forName("sun.jdbc.driver.JdbcOdbcDriver");
//Class.forName("oracle.jdbc.driver.OracleDriver");
try(Connection conn = DriverManager.getConnection("jdbc:odbc:Sample, "","")) 
{
Statement statemnt1 = conn.createStatement();
ResultSet rs1 =null;
statemnt1 = conn.createStatement();
System.out.println("Executing Update query using executeUpdate method");
int return_rows = statemnt1.executeUpdate(update_query);
System.out.println("No. of Affected Rows = "+ return_rows);
}
```

SQL Exception Handling Example

```
catch(SQLException sqe)
{
System.out.println("Error Code = " + sqe.getErrorCode());
System.out.println("SQL state = " + sqe.getSQLState());
System.out.println("Message = " + sqe.getMessage());
System.out.println("printTrace /n");
sqe.printStackTrace();
}
}
}
```

```
Console
<terminated> Exception_Example [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (29-Jul-2(
Executing Update query using executeUpdate method
Error Code = 904
SQL state = 42000
Message = ORA-00904: "EMPNUM1": invalid identifier

printTrace /n
java.sql.SQLSyntaxErrorException: ORA-00904: "EMPNUM1": invalid identifier

        at oracle.jdbc.driver.T4CTTIoer11.processError(T4CTTIoer11.java:494)
        at oracle.jdbc.driver.T4CTTIoer11.processError(T4CTTIoer11.java:446)
        at oracle.jdbc.driver.T4C8Oall.processError(T4C8Oall.java:1054)
        at oracle.jdbc.driver.T4CTTIfun.receive(T4CTTIfun.java:623)
        at oracle.jdbc.driver.T4CTTIfun.doRPC(T4CTTIfun.java:252)
        at oracle.jdbc.driver.T4C8Oall.doOALL(T4C8Oall.java:612)
        at oracle.jdbc.driver.T4CStatement.doOall8(T4CStatement.java:213)
        at oracle.jdbc.driver.T4CStatement.doOall8(T4CStatement.java:37)
        at oracle.jdbc.driver.T4CStatement.executeForRows(T4CStatement.java:896)
        at oracle.jdbc.driver.OracleStatement.doExecuteWithTimeout(OracleStatement.java:1119)
        at oracle.jdbc.driver.OracleStatement.executeUpdateInternal(OracleStatement.java:1661)
        at oracle.jdbc.driver.OracleStatement.executeLargeUpdate(OracleStatement.java:1626)
        at oracle.jdbc.driver.OracleStatement.executeUpdate(OracleStatement.java:1613)
        at oracle.jdbc.driver.OracleStatementWrapper.executeUpdate(OracleStatementWrapper.java:282)
        at com.STH.JDBC.Exception_Example.main(Exception_Example.java:24)
Caused by: Error : 904, Position : 60, Sql = update employee_details set email='martinL@gmail.com' where empNum1 = 10011, OriginalSq

        at oracle.jdbc.driver.T4CTTIoer11.processError(T4CTTIoer11.java:498)
        ... 14 more
```

Error code of the Exception

SQLState of the Exception

Message of the Exception

PrintTrace details of the Exception

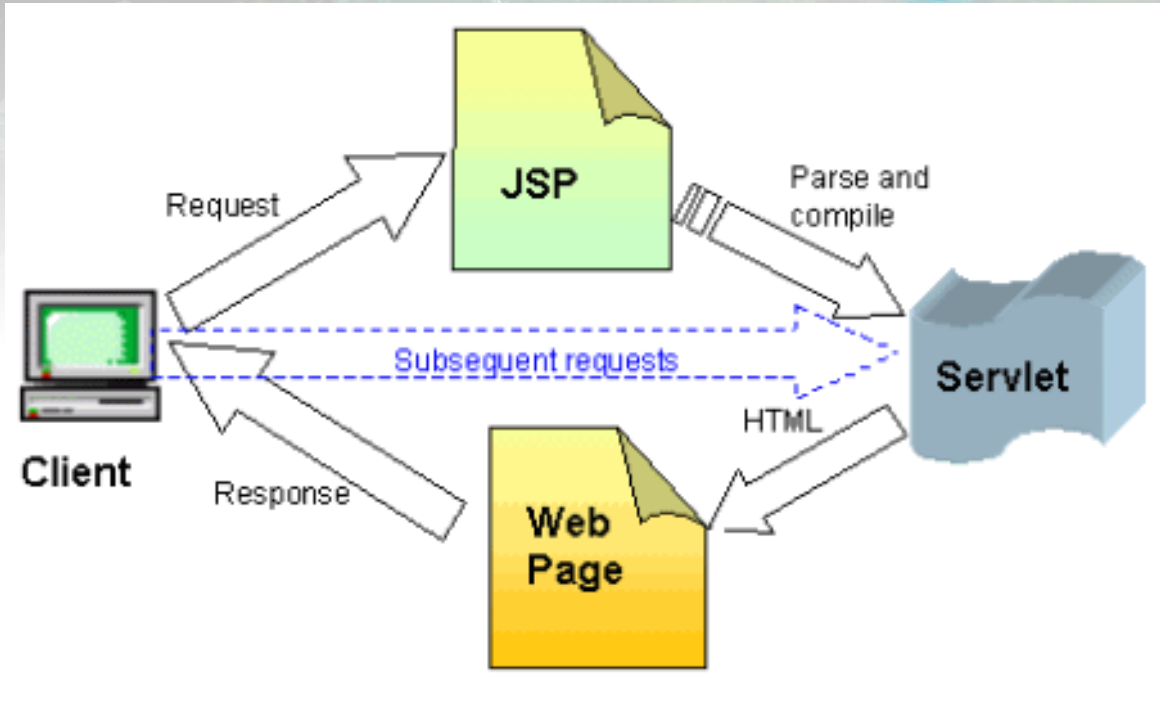
Java Server Pages (JSP)

- Java Server Pages (JSP) is a technology which is used to develop web pages by inserting Java code into the HTML pages by making special JSP tags. The JSP tags which allow java code to be included into it are **<% —-java code—-%>.**
- It can consist of either HTML or XML (combination of both is also possible) with JSP actions and commands.
- It can be used as HTML page, which can be used in forms and registration pages with the dynamic content into it.
- Dynamic content includes some fields like dropdown, checkboxes, etc. whose value will be fetched from the database.
- User can share information across pages using request and response objects.
- JSP can be used for separation of the view layer with the business logic in the web application.

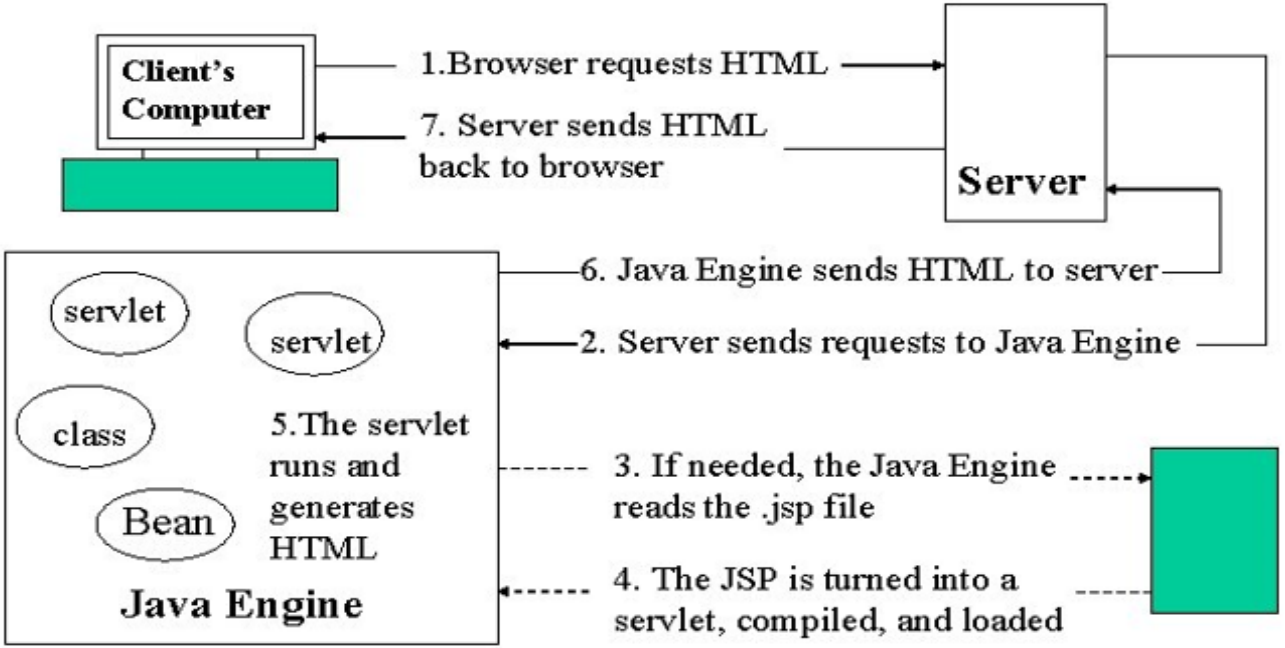
Why Java Server Pages (JSP)

- In Java server pages JSP, the execution is much faster compared to other dynamic languages.
- It is much better than Common Gateway Interface (CGI).
- Java server pages JSP are always compiled before its processed by the server as it reduces the effort of the server to create process.
- Java server pages JSP are built over Java Servlets API.

Java Server Pages (JSP)



Java Server Pages (JSP)



Java Server Pages (JSP) Example

hello.jsp

```
<?xml version="1.0" encoding="UTF-8" ?>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Insert title here</title>
</head>
<body>
<% System.out.println("Hello World!"); %>
</body>
</html>
```

Servlets

- Servlet is a Java technology that is managed by a container called a servlet engine.
- It generates dynamic content and interacts with the client through Request and Response.
- Servlet extends the functionality of a web server.
- Though servlets can respond to many types of requests, they generally implement web containers for hosting any website on web server.
- Therefore, it is qualified as a server-side servlet web API

JSP vs. Servlets

Servlet	JSP
Servlets are Java classes that handle HTTP requests and generate HTTP responses.	JSPs are text files that contain a mixture of HTML and Java code, which is then translated into a servlet by the web server.
Servlets are typically used for more complex tasks such as handling form submissions, connecting to databases, and managing sessions.	JSPs are typically used for simple tasks such as displaying dynamic data in an HTML template.
Servlets are more difficult to write and debug than JSPs, because they require more Java code.	JSPs are easier to write and debug than servlets, because they contain less Java code and more HTML.
Servlets are more efficient than JSPs, because they are compiled into Java bytecode and can be reused across multiple requests.	JSPs are less efficient than servlets, because they are interpreted at runtime and must be translated into a servlet before each request.

References

- https://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html
- https://cse.iitkgp.ac.in/~dsamanta/java/ch10.htm
- https://www.guru99.com/jsp-tutorial.html

Roadmap for Unit 5

1. JDBC Architecture
2. JDBC API
3. JDBC Commands
4. Insertion, Deletion, Updation and Selection
5. SQL Exceptions
6. JSP, Servlets

Roadmap for Java Programming

Unit I
Introduction, Looping and Operators

1

2

Unit – II
Classes and Objects, Constructors

Unit III
Inheritance, Interface, Exception Handling

3

4

Unit IV
Multithreading and Input/Output Files

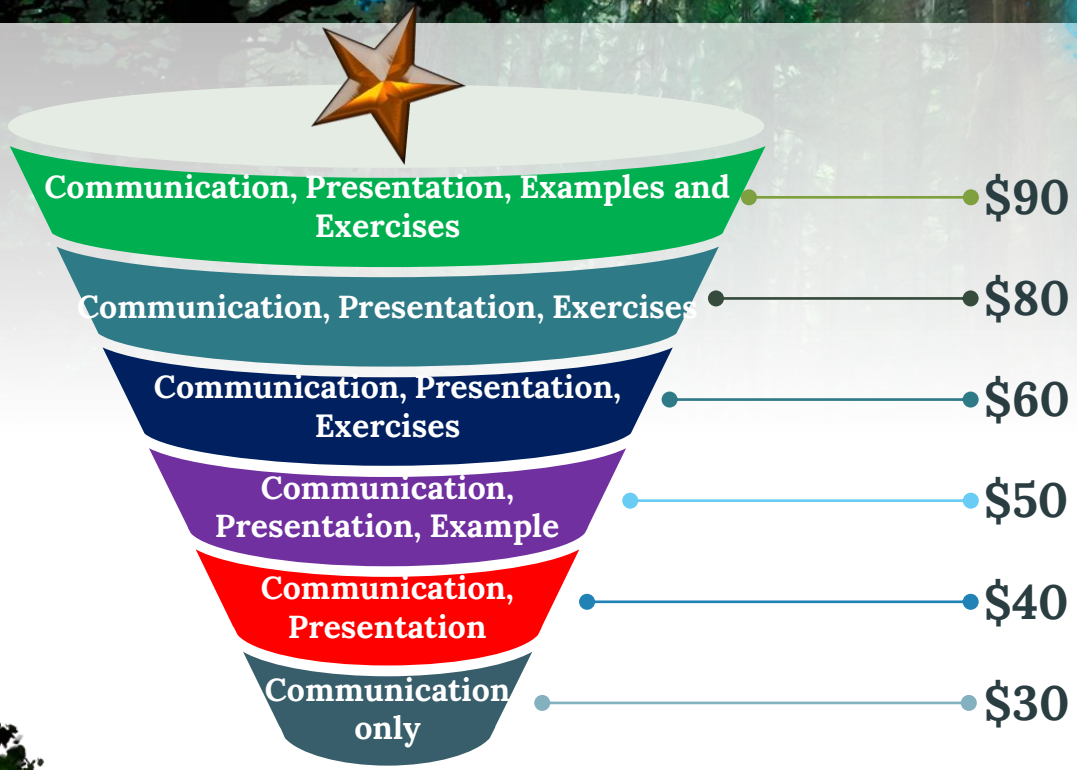
Unit V
JDBC, JSP

5

6

Revision

Feedback


Communication, Presentation, Examples and Exercises
$90
Communication, Presentation, Exercises
$80
Communication, Presentation, Exercises
$60
Communication, Presentation, Example
$50
Communication, Presentation
$40
Communication only
$30

Questions

THANK
YOU!