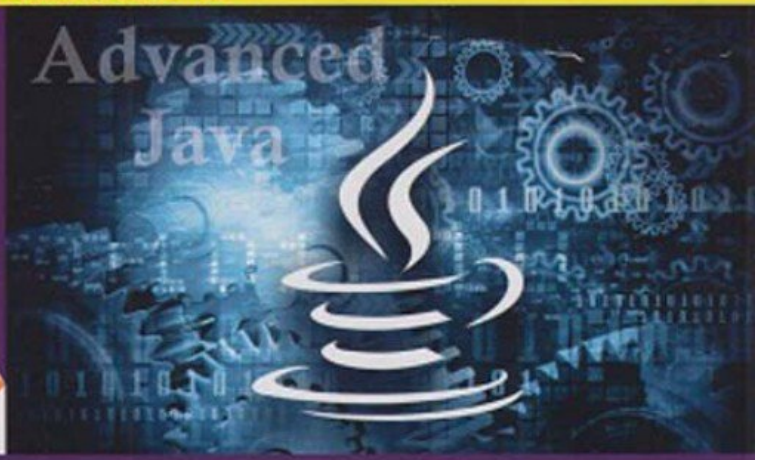
NEW SYLLABUS
CBCS PATTERN

T.Y. B.B.A. (C.A.)
SEMESTER - VI



Advanced Java

Dr. Ms. Manisha Bharambe

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

SPPU New Syllabus

A Book Of

ADVANCED JAVA

For T.Y.B.B.A. (C.A.) : Semester - VI

[Course Code CA - 603 : Credits - 03]

CBCS Pattern

As Per New Syllabus, Effective from June 2021

Dr. Ms. Manisha Bharambe
M.Sc. (Comp. Sci.), M.Phil, Ph.D. (Comp. Sci.)
Associate Professor, Computer Science Department
MES Abasaheb Garware College, Pune

Price ₹ 270.00

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

N6189

ADVANCED JAVA **ISBN 978-93-5451-523-1**

First Edition : March 2022

Published By :
NIRALI PRAKASHAN
Abhyudaya Pragati, 1312, Shivaji Nagar,
Off J.M. Road, Pune – 411005
Tel - (020) 25512336/37/39
Email : niralipune@pragationline.com

Polyplate

Printed By :
YOGIRAJ PRINTERS AND BINDERS
Survey No. 10/1A, Ghule Industrial Estate
Nanded Gaon Road
Nanded, Pune - 411041

DISTRIBUTION CENTRES

PUNE

Nirali Prakashan
(For orders outside Pune)
S. No. 28/27, Dhayari Narhe Road, Near Asian College
Pune 411041, Maharashtra
Tel : (020) 24690204; Mobile : 9657703143
Email : bookorder@pragationline.com

Nirali Prakashan
(For orders within Pune)
119, Budhwar Peth, Jogeshwari Mandir Lane
Pune 411002, Maharashtra
Tel : (020) 2445 2044; Mobile : 9657703145
Email : niralilocal@pragationline.com

MUMBAI

Nirali Prakashan
Rasdhara Co-op. Hsg. Society Ltd., 'D' Wing Ground Floor, 385 S.V.P. Road
Girgaum, Mumbai 400004, Maharashtra
Mobile : 7045821020, Tel : (022) 2385 6339 / 2386 9976
Email : niralimumbai@pragationline.com

DISTRIBUTION BRANCHES

DELHI
Nirali Prakashan
Room No. 2 Ground Floor
4575/15 Omkar Tower, Agarwal Road
Darya Ganj, New Delhi 110002
Mobile : 9555778814/9818561840
Email : delhi@niralibooks.com

BENGALURU
Nirali Prakashan
Maitri Ground Floor, Jaya Apartments,
No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003
Karnataka; Mob : 9686821074
Email : bengaluru@niralibooks.com

NAGPUR
Nirali Prakashan
Above Maratha Mandir, Shop No. 3,
First Floor, Rani Jhanshi Square,
Sitabuldi Nagpur 440012 (MAH)
Tel : (0712) 254 7129
Email : nagpur@niralibooks.com

KOLHAPUR
Nirali Prakashan
New Mahadvar Road, Kedar Plaza,
1st Floor Opp. IDBI Bank
Kolhapur 416 012 Maharashtra
Mob : 9850046155
Email : kolhapur@niralibooks.com

JALGAON
Nirali Prakashan
34, V. V. Golani Market, Navi Peth,
Jalgaon 425001, Maharashtra
Tel : (0257) 222 0395
Mob : 94234 91860
Email : jalgaon@niralibooks.com

SOLAPUR
Nirali Prakashan
R-158/2, Avanti Nagar, Near Golden
Gate, Pune Naka Chowk
Solapur 413001, Maharashtra
Mobile 9890918687
Email : solapur@niralibooks.com

marketing@pragationline.com | www.pragationline.com

Also find us on www.facebook.com/niralibooks

Preface ...

We take this opportunity to present this book entitled as **"Advanced Java"** to the students of Sixth Semester – T.Y.B.B.A (C.A.). The object of this book is to present the subject matter in a most concise and simple manner. The book is written strictly according to the New Syllabus (CBCS Pattern-2019).

The book has its own unique features. It covers theory of JDBC, Multithreading, Networking, Servlet and JSP and Spring and Hibernate.

We sincerely thank Shri. Dineshbhai Furia and Shri. Jignesh Furia of Nirali Prakashan, for the confidence reposed in us and giving us this opportunity to reach out to the students of BBA (CA) studies.

We have given our best inputs for this book. Any suggestions towards the improvement of this book and sincere comments are most welcome on niralipune@pragationline.com.

Authors

■■■

Syllabus ...

1. JDBC **[08 Hrs]**

- 1.1 Introduction
- 1.2 JDBC Architecture
- 1.3 JDBC Process
- 1.4 Working with ResultSet Interface

2. Multithreading **[12 Hrs]**

- 2.1 Introduction to Multithreading
- 2.2 Thread creation: Thread Class, Runnable Interface
- 2.3 Life cycle of Thread
- 2.4 Thread Priority
- 2.5 Execution of Thread Application
- 2.6 Synchronization and Interthread Communication

3. Networking **[05 Hrs]**

- 3.1 Overview of Networking.
- 3.2 Networking Basics: Port Number, Protocols and classes
- 3.3 Sockets, Reading from and Writing to a Socket

4. Servlet and JSP **[12 Hrs]**

Servlet

- 4.1 Introduction to Servlet
- 4.2 Types of Servlet: Generic Servlet and Http Servlet
- 4.3 Life Cycle of Servlet
- 4.4 Session Tracking
- 4.5 Servlet with Database

JSP

- 4.6 Introduction to JSP
- 4.7 JSP Life Cycle
- 4.8 Components of JSP
- 4.9 JSP with Database

5. Spring and Hibernate **[1 Hrs]**

Spring:

- 5.1 Introduction
- 5.2 Applications and Benefits of Spring
- 5.3 Architecture and Environment Setup
- 5.4 Hello World Example
- 5.5 Core Spring - IoC Containers, Spring Bean Definition, Scope, Lifecycle

Hibernate:

- 5.6 Architecture and Environment
- 5.7 Configuration, Sessions, Persistent Class
- 5.8 Mapping Files, Mapping Types
- 5.9 Examples

■■■

Contents ...

■■■

1... JDBC

Learning Objectives ...

Students will be able:

- To learn about JDBC Architecture.
- To understand the JDBC Process.
- To know the working with ResultSet Interface.

1.1 INTRODUCTION

- Java interacts with database using a common database application programming interface called as JDBC (Java DataBase Connectivity).
- JDBC is a Java API to connect and execute the query with the database. JDBC API uses JDBC drivers (JDBC-ODBC Bridge Driver, Native Driver etc.) to connect with the database.
- The Java JDBC API is part of the core JavaSE (Java Standard Edition) SDK, making JDBC available to all Java applications that want to use it. The API (Application Programming Interface) is a document that contains description of all the features of a product or software.
- JDBC is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases such as Oracle, PostgreSQL, Microsoft Access, DB2, Sybase, MySQL etc.
- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.
  1. Making a connection to a database.
  2. Creating SQL or MySQL statements.
  3. Executing SQL or MySQL queries in the database.
  4. Viewing and modifying the resulting records.
- Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database.

- JDBC API enables the java application to interact with different database. This is shown in Fig. 1.1.

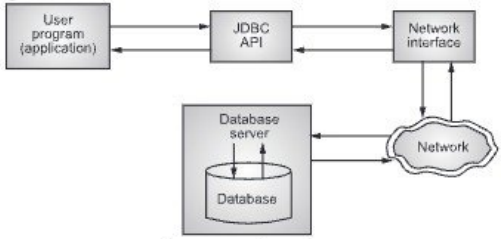


Fig. 1.1: Overview of JDBC

1.2 JDBC ARCHITECTURE

- The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers:
  1. **JDBC API:** This provides the application-to-JDBC Manager connection.
  2. **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.
- The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
- The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.
- Fig. 1.2 shows the location of the driver manager with respect to the JDBC drivers and the Java application.

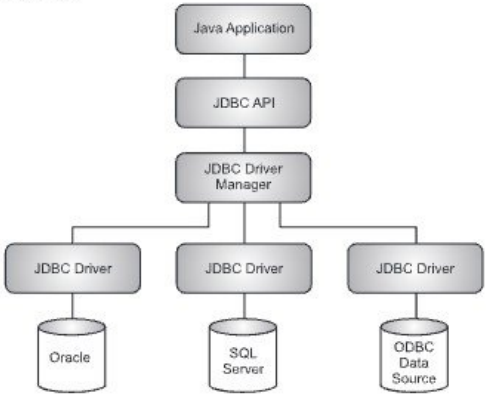


Fig. 1.2: JDBC Architecture

Common JDBC Components:

- The JDBC API provides the following interfaces and classes:
  1. **DriverManager** class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain sub protocol under JDBC will be used to establish a database Connection.
  2. **Driver Interface** handles the communications with the database server. We will interact directly with Driver objects very rarely, instead, we use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
  3. **Connection Interface** with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
  4. **Statement** we use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
  5. **ResultSet Objects** hold data retrieved from a database after we execute an SQL query using Statement objects. It acts as an iterator to allow us to move through its data.
  6. **SQLException** class handles any errors that occur in a database application.

1.2.1 Design of JDBC

- There are many databases available in the market like MySQL, Sybase, DB2, Microsoft Access etc. In order to extending Java, Sun Microsystems wanted to create some API which can be common for all databases available in the market.
- The first problem for designing such API was language barrier i.e., each database vender defines distinct way to communicate with application program. So the low level code written to communicate with an Oracle may be applicable for DB2.
- These JDBC drivers require, it should translate SQL queries in to a language understood by JDBC API.
- JDBC drivers created by DBMS Manufacturers have to:
  1. Established a connection between database and java Component.
  2. Translate SQL command in to a form that can be understood by both database and java components.
  3. Returns any kind of error message that conforms to the JDBC specification to the JDBC driver.
  4. At the end it should close the connection between DBMS and Java Component.
- JDBC is an Application Programming Interface (API) used to connect Java application with database. Design of JDBC defines the components of JDBC, which is used for connecting to the database.
- JDBC is an API that establishes a connection between a standard database and Java application that intends to use that database. JDBC helps us to connect to a database and execute SQL statements against a database.

- Fig. 1.3 shows components for JDBC design. These components are explained below:
  1. **Application:** Application in JDBC like Java Applet or a Servlet that communicates with a data source. Application is the data processing application that intends to access the data from the different databases.
  2. **JDBC API:** JDBC API provides classes, methods, and interfaces that allow Java programs to execute SQL statements and retrieve results from the database. JDBC API provides various interfaces and methods to establish easy connection with different databases. JDBC API provides connection from Application to JDBC Manager. The JDBC API ensures that a stable connection is established between the data storage unit and the JDBC application.
  3. **Driver Manager:** The Driver Manager plays an important role in the JDBC design. JDBC Driver manager loads the database-specific driver into an application in order to establish the connection with the database. The JDBC driver manager makes sure that the correct driver is being used to access the databases. It is also capable of handling multiple drivers connected to multiple databases simultaneously.
  4. **JDBC Drivers:** JDBC drivers help us to communicate with a data source (database) through JDBC. We need a JDBC driver that can intelligently interact with the respective data source.

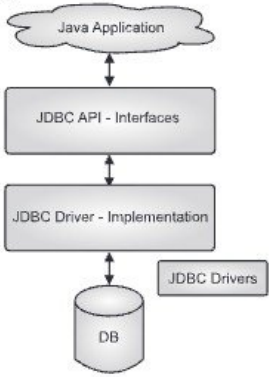


Fig. 1.3: Design of JDBC

1.3 JDBC DRIVERS

- JDBC Driver is a software component that enables Java application to interact with the database. JDBC Driver translates the standard JDBC API calls to the DBMS specific API calls.
- JDBC drivers implement the defined interfaces in the JDBC API, for interacting with the database server.

- For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.
- Fig. 1.4 provides a high-level overview of the various types of drivers.
- The Types 1 and 2 Drivers rely heavily on additional software, (typically C/C++ DLLs) installed on the client computer to provide database connectivity. Java and JDBC use these components to interact with the database.
- The Types 3 and 4 Drivers are pure Java implementations and require no additional software to be installed on the client, except for the JDBC driver.

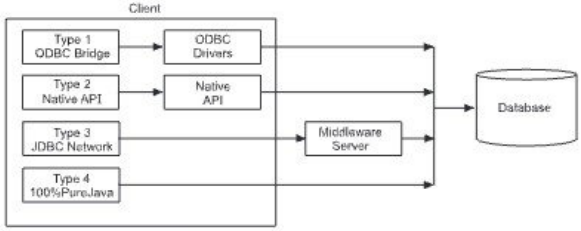


Fig. 1.4: Types JDBC Driver

- Fig. 1.5 shows the communication path between JDBC to database.

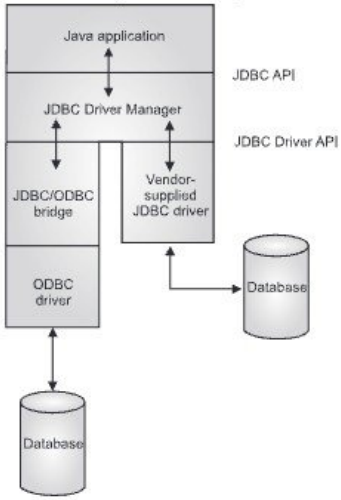


Fig. 1.5

1.3.1 Types of Drivers

- The JDBC API encapsulates relational database access into a hierarchy of Java classes and interfaces.
- JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates.
- Sun has divided the implementation types into four categories of Drivers, Types 1, 2, 3, and 4, which is explained below:

Type 1: JDBC-ODBC Bridge Driver:

- It is also referred as JDBC - ODBC bridge plus ODBC driver. It is a JavaSoft product.
- In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine.
- Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.
- When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.
- Fig. 1.6 shows Java JDBC-ODBC Bridge driver.

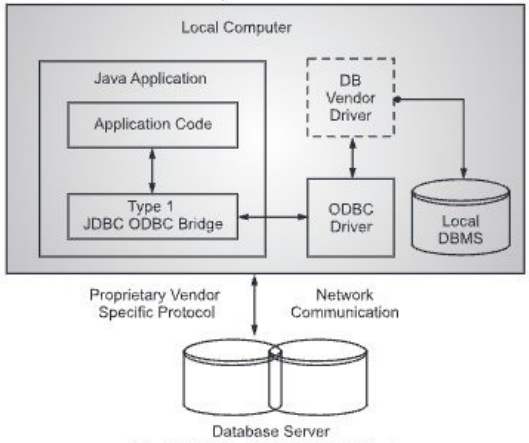


Fig. 1.6: JDBC - ODBC Bridge Driver

- The JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver.

Advantages:

1. Easy to use.
2. Can be easily connected to any database if that database has installed ODBC driver.

Disadvantages:

1. Performance degraded because JDBC method call is converted into the ODBC function calls.
2. The ODBC driver needs to be installed on the client machine.
3. As it is not written fully in java, so these drivers are not portable.
4. They are not much suitable for Web applications.

Type 2: JDBC-Native API Driver:

- It is also referred as Native-API partly-Java driver. This driver converts JDBC calls into calls on the client API for Oracle, Sybase, Db2, informix or other DBMS.
- In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.
- If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but they are fast as compare to Type 1 driver as Type 2 driver has no ODBC's overhead.
- Fig. 1.7 shows JDBC-Native API driver.

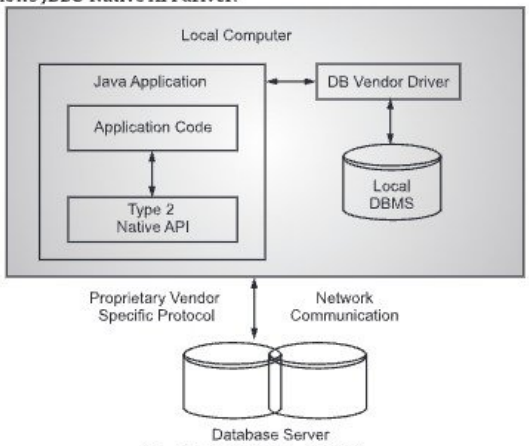


Fig. 1.7: JDBC-Native API Driver

- The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

Advantages:

1. Faster as compared to Type 1 Driver as Type 2 drivers are database specific as well as they do not require translation.
2. Contains additional features.

Disadvantages:

1. Requires installation of native API on client machine so not suitable for web applications.
2. Type 2 drivers are also not written in java so they having portability issue.
3. Increased cost of application.
4. As we need to install database specific Native API on client machine, so if database change we need to change Native API.

Type 3: JDBC-Net Pure Java Driver:

- In a Type 3 driver, a Three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server.
- The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
- This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.
- Fig. 1.8 shows JDBC-Net Pure Java Driver.

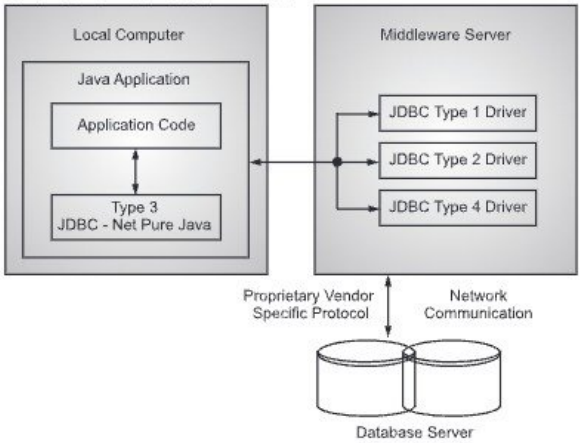


Fig. 1.8: JDBC-Net Pure Java Driver

- We can think of the application server as a JDBC "proxy", meaning that it makes calls for the client application.
- As a result, we need some knowledge of the application server's configuration in order to effectively use this driver type.
- The application server might use a Type 1, 2, or 4 drivers to communicate with the database, understanding the nuances will prove helpful.

Advantages:

1. It does not require any native library to be installed on the client.
2. It has database independency.
3. It provides facility to switch over from one database to another database.
4. They support several networking options, such as HTTP tunneling.
5. They are suitable for web applications.
6. They are most suitable driver type amongst all.

Disadvantages:

1. Slow due to increase number of network call.
2. They can be difficult to set up since; the architecture is complicated by the network interface.
3. It is costliest JDBC driver.

Type 4: Native Protocol Pure Java Drivers (100% Pure Java):

- In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection.
- This is the highest performance driver available for the database and is usually provided by the vendor itself.
- This kind of driver is extremely flexible; you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.
- Fig. 1.9 shows Native Protocol Pure Java driver.

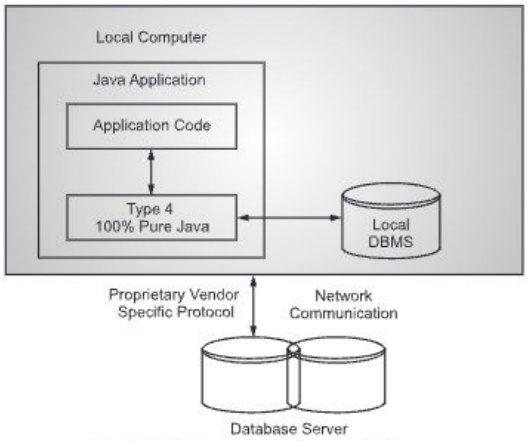


Fig. 1.9: Native Protocol Pure Java Drivers

- MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.

Advantages:

1. These are 100% pure java drivers so they are portable in nature.
2. Better performance than all other drivers.
3. No translation is requires.
4. No software is required at client side or server side.

Disadvantage:

1. Drivers are depending on the Database.

JDBC Packages:

- JDBC API is contained in two packages:
  1. **java.sql:** This package provides basic classes which are used to connect with database and used to load and store data in database.
  2. **javax.sql:** It extends the java.sql and provides classes that are used to interact with Java Naming and Directory Interface **(JNDI)** and manages connection pooling among other advanced JDBC features.

1.4 JDBC PROCESS

- Steps to create connectivity to the database MS-ACCESS/PostgreSQL/Mysql:
  - **Step 1 :** Load the driver and establish a database connection to the MS-Access/ PostgreSQL/mysql server.
  - **Step 2 :** Create an instance of the Statement object.
  - **Step 3 :** Execute a statement to get a ResultSet object.
  - **Step 4 :** Process the ResultSet object.
  - **Step 5 :** Close the database connection.

Step 1: Establishing a Database Connection:

(a) To **connect to the PostgreSQL database**, you need to provide account information such as username, password, and the connection string.

For example, method connects to a PostgreSQL database EMP and returns a Connection object:

```
Class.forName("postgresql.Driver");
public Connection connect() throws SQLException
{
    return DriverManager.getConnection(url, user, password);
}
```

The URL, user and password are as follows:

```
private final String url = "jdbc:postgresql://localhost/EMP";
private final String user = "postgres";
private final String password = "postgres";
```

(b) To **connect to the MS-ACCESS database**, you need to provide account information such as username, password, and the connection string.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con = DriverManager.getConnection("jdbc:odbc:EMP", "root", "xyz");
```

(c) **To connect to the Mysql database**:

```
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3400
/EMP","root","root");
```

Step 2: JDBC provides you with three kinds of Statement objects:

1. **Statement:** You use the Statement to implement a simple SQL statement that has no parameters.
2. **PreparedStatement:** It is the subclass of the Statement class. It gives you the ability to add the parameters to the SQL statements.
3. **CallableStatement:** It extends the PreparedStatement class used to execute a stored procedure that may have parameters.

Step 3: Executing the Query:

To execute a query, you use one of the following methods of the Statement object:

1. **execute:** Returns true if the first object of the query is a ResultSet object. You can get the ResultSet by calling the method getResultSet.
2. **executeQuery:** Returns only one ResultSet object.
3. **executeUpdate:** Returns the number of rows affected by the statement. You use this method for the INSERT, DELETE or UPDATE statement.

Step 4: Process the ResultSet Object:

We access the data in a ResultSet object through a cursor. This cursor is a pointer that points to one row of data in the ResultSet object. Initially, the cursor is positioned before the first row. We call various methods such as next (to move the cursor forward by one row) defined in the ResultSet object to move the cursor.

Step 5: Close the Connection:

When we are finished using a Connection, Statement or ResultSet object, call its close() method to close connection.

1.4.1 Executing SQL Statements

- The next step after establishment of proper database connection is creating SQL statement, using them to work with database.
- One of the following three statements is used to execute database queries.

1. **Statement**: It is used to execute query immediately without compilation of the query.

- The createStatement() method is used to create a statement object.
- The statement object contains the executeQuery() method to which you can pass query as an argument.
- The executeQuery() method executes a query that you passed as an argument and returns a ResultSet.
- The statement object also contains execute() and executeUpdate() methods.

- The execute() is used when there may be multiple results returned and executeUpdate() method is used when query contains update or delete operation that you want to perform on database.
- The executeUpdate() returns an integer value specifying the number of rows that are affected by query.
- **Use of executeQuery() Method: (using MS-ACCESS database)**

```
private Connection con;
statement st;
ResultSet rs;
try
{
    class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    con = DriverManager.getConnection("jdbc:odbc:EMP", "root", "xyz");
    String query = "Select * from employee";
    st=con.createStatement();
    rs=st.executeQuery(query);
    con.close();
}
```

- **Use of executeQuery() Method: (using Mysql database)**
  1. **Driver class:** The driver class for the mysql database is com.mysql.jdbc.Driver.
  2. **Connection URL:** The connection URL for the mysql database is jdbc:mysql://localhost:3400/EMP where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3400 is the port number and EMP is the database name. We may use any database, in such cases, we need to replace the EMP with our database name.
  3. **Username:** The default username for the mysql database is root.
  4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

```
private Connection con;
statement st;
ResultSet rs;
try
{
    Class.forName("com.mysql.jdbc.Driver");
    Connection con=DriverManager.getConnection( "jdbc:mysql://localhost
    :3400/EMP","root","root");
    String query="Select * from employee";
           st=con.createStatement();
           rs=st.executeQuery(query);
           con.close();
}
```

- **Use of executeQuery() Method: (using PostgreSQL database)**

```
private Connection con;
statement st;
ResultSet rs;
try
{
   Class.forName("postgresql.Driver");
      System.out.println("Driver Loaded ...");
      Connection con = DriverManager.getConnection
      ("jdbc:postgresql:EMP","postgres","");
      String query = "Select * from employee";
      st=con.createStatement();
         rs=st.executeQuery(query);
         con.close();
}
```

- **Use of execute() Method:**

```
String query = "drop table";
st.execute(query);
```

- **Use of executeUpdate() Method:**

```
String query = "Update employee SET ename='Deepak' WHERE eid='10' ";
rs=st.executeUpdate(query);
```

Program 1.1: Program for statement class.

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
public class SQLJDBC
{
   public static void main( String args[] )
   {
      Connection c = null;
      Statement stmt = null;
      try
      {
         class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
         c = DriverManager.getConnection("jdbc:odbc:EMP", "root", "xyz");
         //user name password as per your system
         System.out.println("Opened database successfully");
         stmt = c.createStatement();
```

```
            String sql = "CREATE TABLE Employee" +
              "(ID INT PRIMARY KEY NOT NULL," +
              "NAME TEXT NOT NULL," +
              "AGE INT NOT NULL," +
              "ADDRESS CHAR(50), "+
              "SALARY REAL)";
            stmt.executeUpdate(sql);
            stmt.close();
            c.close();
         } catch ( Exception e )
         {
            System.err.println(e.getClass().getName()+": "+ e.getMessage());
            System.exit(0);
         }
         System.out.println("Table created successfully");
      }
   }
```

Output:

```
Opened database successfully
Table created successfully
```

2. PreparedStatement:

- SQL queries are precompiled and executed using PreparedStatement.
- PreparedStatement is used when we want to execute SQL queries repeatedly but with different value.

```
select * from employee where eid=?
```

- In above example we want to retrieve employee information by specifying the employee id.
  Here, preparedStatement() method of connection object is used to precompile query which we pass as an argument to this method.
- In the following example we have given "?" at the place of eid, which is later on get replaced by actual eid using setxxx() method which requires two parameters, first is an integer that specifies the position of the question mark placeholder and second the value that replaces the question mark.
- Next there a executeQuery() method, here you need not to pass any parameter because the query that you want to execute is already associated with prepared statement object.

```
private Connection con;
PreparedStatement pst;
ResultSet rs;
   try {
         class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
            con = DriverManager.getConnection
                                        ("jdbc:odbc:EMP", "root", "xyz");
            String query = "Select * from employee where eid=?";
            pst=con.preparedStatement(query);
            pst.setString(1, "10");
            rs=pst.executeQuery();
            pst.close();
            con.close();
        }
```

Program 1.2: Program for PreparedStatement.

```
import java.sql.*;
public class jdbcConn
{
   public static void main(String[] args) throws Exception
   {
      Connection con;
      Statement stmt = null;

      class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      con = DriverManager.getConnection("jdbc:odbc:EMP", "root", "xyz");
      System.out.println("Opened database successfully");
      PreparedStatement updateemp = con.prepareStatement
                                        ("insert into emp values(?,?,?)");
      updateemp.setInt(1, 23);
      updateemp.setString(2, "Deepak");
      updateemp.setString(3, "Programmer");
      updateemp.executeUpdate();

      Statement stmt = con.createStatement();
      String query = "select * from emp";
      ResultSet rs =  stmt.executeQuery(query);
      System.out.println("Id Name    Job");

      while (rs.next())
      {
         int id = rs.getInt("id");
         String name = rs.getString("name");
         String job = rs.getString("job");
         System.out.println(id + "   " + name+"    "+job);
      }
   }
}
```

Output:

```
Opened database successfully
id   Name      Job
23  Deepak   Programmer
```

3. Callable Statement:

- This statement is used to call stored procedure from java component.
- Callable statement object uses three types of parameters when calling a stored procedure, i.e. IN, OUT, INOUT.
- The IN parameter is used to pass input value to the procedure, for which again we use setxxx() method, where xxx is data type.
- The OUT parameter is used to store any value return by procedure. For this OUT parameter should be registered using registerOutParameter() method.
- The INOUT parameter is a single parameter that is used for both, passing as well as retrieving information from procedure.
- The prepareCall() method is used to create an object of CallableStatement.

For example:

```
private Connection con;
CallableStatement cst;
ResultSet rs;
try
{
   class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   con = DriverManager.getConnection("jdbc:odbc:EMP", "root", "xyz");
   String query = "{ call EMPSALARY (?) } ";
   cst = con. prepareCall(query);
   cst.registerOutParameter(1, java.sql.Types. VARCHAR);
   rs=cst.execute ();
   EMPSALARY=cst.getString(1);
   cst.close();
   con.close();
}
```

Program 1.3: Program for Callable Statement.

```
import java.sql.*;
public class Sumval
{
   public static void main(String[] args) throws Exception
   {
      Connection con;
      Statement stmt = null;

      class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      con = DriverManager.getConnection("jdbc:odbc:EMP", "root", "xyz");
```

```
        CallableStatement stmt = con.prepareCall("{?= call sum(?,?)}");
        stmt.setInt(2,50);
        stmt.setInt(3,40);
        stmt.registerOutParameter(1,Types.INTEGER);
        stmt.execute();
        System.out.println(stmt.getInt(1));
    }
}
```

Output:

```
90
```

Closing a Database Connection:

- In JDBC you use a try-with-resources statement to close ResultSet, Statement, and Connection objects automatically.

Program 1.4: JDBC program to create a STUDENT(STUDNO, STUDNAME, STUDPER) table, fetch the record and display the all in the record of students.

```
import java.sql.*;    //Step 1 : import package
import java.io.*;
class Stud
{
   public static void main(String a1[])
   {
      try
      {
         class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
         con = DriverManager.getConnection
                                        ("jdbc:odbc:EMP", "root", "xyz");
         // Step 3 create connection
         System.out.println("Connection Established...");
         Statement st = con.createStatement();
         // Step 4 create statement object
         System.out.println("Create statement...");
         ResultSet rs = st.executeQuery("select * from student");
         //Step 5 create resultset object
         System.out.println("Retrieved data");
         System.out.println("Student no \t student Name \t Student Per");
         while(rs.next())          //Step 6 Retrive Data
         {
            System.out.print(" "+rs.getInt(1));
            System.out.print("\t\t"+rs.getString(2));
            System.out.print("\t\t"+rs.getFloat(3));
            System.out.println();
         }//while
         con.close();     //Step 7 close the connection
      }//try
```

```
        catch(Exception e)
        {
           System.out.println("Error"+e);
        }
     }//main
}//end of class
```

Output:

```
Driver Loaded...
Connection Established...
Create statement...
Retrieved data
Student no    student Name      Student Per
   10           amit               91.0
   20           shubham            75.0
   30           aditya             77.0
   40           omkar              82.0
   90           vandana            87.0
   100          shruti             97.0
```

1.5 WORKING WITH RESULTSET INTERFACE (SCROLLABLE AND UPDATABLE)

- All query that we fire on database return some result if there is no problem in query. Even if there is no item matching with the values specified in query, whatever result return by query should be store somewhere. Those results are getting stored in ResultSet in Java.
- When result is returned from database, database has a choice to construct it:
  1. **A read-only:** As the name suggested here ResultSet enables you to read through all of the values from the first to last. Once you reach the end, you cannot reread the values.
  2. **A scrollable:** This type you can reread ResultSet again and again. With it you can jump to any known row or just move back and forth through the current list.
  3. **An updateable**: This form is a live representation of the underlying database that enables you to insert, update, and delete rows. This form must also be scrollable.
- To get the type of ResultSet we can use the getType() and getConcurrency() methods. The getType() method returns one of three types:
  1. **TYPE_FORWARD_ONLY**: The result set is the read–only version that enables you to move in forward direction only.

2. **TYPE_SCROLL_SENSITIVE**: This ResultSet is Scrollable means you can move in any direction can directly access any row directly. Also this ResultSet is dynamic in nature means if any changes are made while ResultSet is open the values are reflected in ResultSet.
3. **TYPE_SCROLL_INSENSITIVE:** This ResultSet is Scrollable means you can move in any direction can directly access any row directly. But when others make changes to the underlying data source, they will not change the order of values you see in your return results.

- The methods of the ResultSet interface are given below:
  1. `public void beforeFirst() throws SQLException` method moves the cursor just before the first row.
  2. `public void afterLast() throws SQLException` method moves the cursor just after the last row.
  3. `public boolean first() throws SQLException` method moves the cursor to the first row.
  4. `public void last() throws SQLException` method moves the cursor to the last row.
  5. `public boolean absolute(int row) throws SQLException` method moves the cursor to the specified row.
  6. `public boolean relative(int row) throws SQLException` method moves the cursor the given number of rows forward or backward, from where it is currently pointing.
  7. `public boolean previous() throws SQLException` method moves the cursor to the previous row. This method returns false if the previous row is off the result set.
  8. `public boolean next() throws SQLException` method moves the cursor to the next row. This method returns false if there are no more rows in the result set.
  9. `public int getRow() throws SQLException` method returns the row number that the cursor is pointing to.
  10. `public void moveToInsertRow() throws SQLException` method moves the cursor to a special row in the result set that can be used to insert a new row into the database. The current cursor location is remembered.
  11. `public void moveToCurrentRow() throws SQLException` method moves the cursor back to the current row if the cursor is currently at the insert row; otherwise, this method does nothing.
  12. `public int getInt(String columnName) throws SQLException` method returns the int in the current row in the column named columnName.
  13. `public int getInt(int columnIndex) throws SQLException` method returns the int in the current row in the specified column index. The column index starts at 1, meaning the first column of a row is 1, the second column of a row is 2, and so on.

14. `public void updateString(int columnIndex, String s) throws SQLException` method changes the String in the specified column to the value of s.
15. `public void updateString(String columnName, String s) throws SQLException` method is similar to the previous method, except that the column is specified by its name instead of its index.
16. `public void deleteRow()` method deletes the current row from the database.
17. `public void refreshRow()` method refreshes the data in the ResultSet to reflect any recent changes in the database.
18. `public void cancelRowUpdates()` methods cancels any updates made on the current row.
19. `public void insertRow()` method inserts a row into the database. This method can only be invoked when the cursor is pointing to the insert row.

1.6 METADATA

- JDBC MetaData is the collective information about the data structure and property of a column available in table.
- The metadata of any table tells us the name of the columns, datatype used in column and constraint used to enter the value of data into column of the table.
- With the following interfaces we can retrieve some basic Metadata:

DatabaseMetaData:

- Applications sometime need to know something about the underlying database. For example, an application might want to know the version of JDBC, even the maximum number of open connections such as information can be retrieved using DatabaseMetaData.
- It provides data about a database's structure, such as the table names, primary and foreign keys, and data types etc. In addition, you can use a DatabaseMetaData object to probe the database to determine its attributes.
- The DatabaseMetaData object has many methods and properties that provide a lot of information about a database.

Creating DatabaseMetaData Objects:

- A Connection object represents a database connection and also instantiates a DatabaseMetaData object with the getMetaData() method.
- The DatabaseMetaData object holds information for the database to which the Connection object is connected.
- The following code shows how to create a DatabaseMetaData object:

```
DatabaseMetaData dmd = conn.getMetaData();
```

Methods of DatabaseMetaData:

(i) `getTables()` method retrieve information about the tables.
(ii) `getColumns()` method retrieve information about table columns in a database.
(iii) `getTypeInfo()` method determines what data types your target database supports.

(iv) `getPrimaryKeys()` and `getImportedKeys()` methods provide the primary and foreign-key information.

(v) `getProcedures()` and `getProcedureColumns()` methods provides information about the stored procedures in a database.

ResultSetMetaData:

- Represents the information about a particular result. Here, you can find information such as column–header names, the numbers of columns present in the return results, and whether the values are read–only.
- A ResultSetMetaData object is useful when you want to create a generic method with which to process ResultSets.
- By using metadata you can determine the data types of the ResultSet columns and call the correct getXXX() method to retrieve the data.
- The ResultSetMetaData interface provides descriptive information about the columns in a ResultSet such as the number of columns it contains or each column's data type.

Creating ResultSetMetaData Objects:

- Object or ResultSetMetaData can be created same as that of DatabaseMetaData, using getMetaData()method.

```
ResultSetMetaData rsmd = rs.getMetaData();
```

Methods of ResultSetMetaData:

1. `getTableName()` method returns the name of the table you queried.
2. `getColumnTypeName(int n)` method provides the name of the column's data type as defined by the database. The parameter n is the column position within the ResultSet.
3. `getColumnType(int n)` method returns the JDBC data type for a column at position n of the ResultSet.
4. `getColumnName(int n)` method returns the name of a column at position n in the ResultSet.
5. `getColumnCount()` method returns the number of columns in the ResultSet. Returns an int.

ParameterMetaData:

- An object that can be used to get information about the types and properties for each parameter marker in a PreparedStatement object.
- For some queries and driver implementations, the data that would be returned by a ParameterMetaData object may not be available until the PreparedStatement has been executed.

RowSetMetaData:

- An object that contains information about the columns in a RowSet object. This interface is an extension of the ResultSetMetaData interface with methods for setting the values in a RowSetMetaData object.

- When a RowSetReader object reads data into a RowSet object, it creates a RowSetMetaData object and initializes it using the methods in the RowSetMetaData interface. Then the reader passes the RowSetMetaData object to the RowSet.
- Metadata means data about data. We can get further information from the data. If you have to get metadata of a table like total number of columns, column name, column type etc.

Java DatabaseMetaData Interface:

- DatabaseMetaData interface provides methods to get MetaData 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 of DatabaseMetaData interface:
  1. **public String getDriverName() throws SQLException:** It returns the name of the JDBC driver.
  2. **public String getDriverVersion() throws SQLException:** It returns the version number of the JDBC driver.
  3. **public String getUserName() throws SQLException:** It returns the username of the database.
  4. **public String getDatabaseProductName() throws SQLException:** It returns the product name of the database.
  5. **public String getDatabaseProductVersion() throws SQLException:** It returns the product version of the database.
  6. **public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types) throws SQLException:** It returns the description of the tables of the specified catalogue. The table type can be TABLE, VIEW, ALIAS, SYSTEM TABLE, SYNONYM etc.

Program 1.5: DatabaseMetaData Program

```
import java.sql.*;
class Dbmd
{
   public static void main(String args[])
   {
      try
      {
         class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
         con = DriverManager.getConnection
                                        ("jdbc:odbc:EMP","root", "xyz");
         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);}
      }
   }
```

Output:

```
Driver Name: Oracle JDBC Driver
Driver Version: 10.2.0.1.0XE
Database Product Name: Oracle
Database Product Version: Oracle Database 10g Express Edition
Release 10.2.0.1.0 -Production
```

Additional Programs

Program 1.6: Program to insert data into table student (using command Line) and search a record and display the information of students.

```
import java.sql.*;
import java.io.*;
class Studinfo
{
   public static void main(String a[])
   {
      try
      {
         int sno = Integer.parseInt(a[0]);
         String sname = a[1];
         Float sper = Float.parseFloat(a[2]);
         int no = Integer.parseInt(a[3]);
         System.out.println("Student No = "+sno);
         System.out.println("Student Name = "+sname);
         System.out.println("Student Percentage = "+sper);
         class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
         con = DriverManager.getConnection("jdbc:odbc:EMP", "root", "xyz");
         System.out.println("Connection Established");
         PreparedStatement ps = con.prepareStatement
         ("insert into student values(?,?,?)");
         ps.setInt(1,sno);
         ps.setString(2,sname);
         ps.setFloat(3,sper);
```

```
            int i = ps.executeUpdate();
            System.out.println("Value ="+i);
            System.out.println("Insert Data Successfully ...");
            Statement st = con.createStatement();
            System.out.println("Statement Created...");
            ResultSet rs = st.executeQuery("select * from student");
            System.out.println("Student No \t Student Name \t Student Per");
            while(rs.next())
            {
               System.out.print(""+rs.getInt(1));
               System.out.print("\t\t"+rs.getString(2));
               System.out.print("\t\t"+rs.getFloat(3));
               System.out.println();
            } //while
            //searching using Constant
            Statement st1 = con.createStatement();
            System.out.println("Statement Created...");
            ResultSet rs1 = st.executeQuery
                                    ("select * from student where sno=20");
            System.out.println("FOUND...");
            System.out.println("Student No \t Student Name \t Student Per");
            while(rs1.next())
            {
               System.out.print(""+rs1.getInt(1));
               System.out.print("\t\t"+rs1.getString(2));
               System.out.print("\t\t"+rs1.getFloat(3));
               System.out.println();
            } //while
         }//try
         catch(Exception e)
         {
            System.out.println("Error "+e);
         }//catch
      }//end of main
   } //end of class
```

Output:

```
80    manisha  90.0
Student No = 20
Student Name = Shubham
Student Percentage = 75.0
Connection Established
Value =1
Insert Data Successfully ...
Statement Created...
```

```
Student No    Student Name      Student Per
 10             amit              91.0
 20             shubham           75.0
 30             aditya            77.0
 40             omkar             82.0
 90             vandana           87.0
 100            shruti            97.0
 80             manisha           90.0
Statement Created...
       FOUND...
Student No     Student Name      Student Per
    20           shubham           75.0
```

Program 1.7: A JDBC program to create a EMP (EMPNO, EMPNAME, EMPSALARY) table get the information from user and insert into EMPLOYEE table and search the record by EMPNO and EMPNAME and display the information of EMPLOYEE.

```
import java.sql.*;
import java.io.*;
class Emp
{
   public static void main(String a[])
   {
      try
      {
         int eno,esal,lim;
         String ename;
         class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
         con = DriverManager.getConnection("jdbc:odbc:EMP", "root", "xyz");
         System.out.println("Connection Established...");
         BufferedReader br = new BufferedReader
                                    (new InputStreamReader(System.in));
         System.out.println("How many record want to be insert?");
         lim = Integer.parseInt(br.readLine());
         for(int i=0;i<lim;i++)
         {
            System.out.println("Enter Employee No =");
            eno = Integer.parseInt(br.readLine());

            System.out.println("Enter Employee Name =");
            ename = br.readLine();

            System.out.println("Enter Employee Salary =");
            esal = Integer.parseInt(br.readLine());
```

```
    //insert
    PreparedStatement ps = con.prepareStatement
                                ("insert into emp values(?,?,?)");
    ps.setInt(1,eno);
    ps.setString(2,ename);
    ps.setInt(3,esal);
    int j = ps.executeUpdate();
    System.out.println("j = "+j);
}//for */
//Display (select )
Statement st = con.createStatement();
System.out.println("Statement Created...");

ResultSet rs = st.executeQuery("select * from emp");
System.out.println("*** Employee Record *** \n");
System.out.println("Emp No \t Emp Name \t Emp Salary");
while(rs.next())
{
    System.out.print(" "+rs.getInt(1));
    System.out.print("\t"+rs.getString(2));
    System.out.print("\t\t "+rs.getInt(3));
    System.out.println();
}
//Search by Employee Number
System.out.println("Enter Employee No to be Search =");
int no = Integer.parseInt(br.readLine());

Statement st1 = con.createStatement();
System.out.println("Statement Created...");

ResultSet rs1 = st1.executeQuery
                    ("select * from emp where eno = + '"+no+"'");
System.out.println("*** Employee Record *** \n");
System.out.println("Emp No \t Emp Name \t\t Emp Salary");
while(rs1.next())
{
    System.out.print(" "+rs1.getInt(1));
    System.out.print("\t"+rs1.getString(2));
    System.out.print("\t\t "+rs1.getInt(3));
    System.out.println();
}
//Search by Employee Name
System.out.println("Enter Employee Name to be Search =");
String name = br.readLine();
```

```
            Statement st2 = con.createStatement();
            System.out.println("Statement Created...");

            ResultSet rs2 = st2.executeQuery
                            ("select * from emp where ename = '"+name+"'");
            System.out.println("*** Employee Record *** \n");
            System.out.println("Emp No \t Emp Name \t\t Emp Salary");
            while(rs2.next())
            {
                System.out.print(" "+rs2.getInt(1));
                System.out.print("\t"+rs2.getString(2));
                System.out.print("\t\t "+rs2.getInt(3));
                System.out.println();
            }
        }//try
        catch(Exception e)
        {
            System.out.println("Error "+e);
        }
    }
}
```

Output:

```
Driver Loaded ...
Connection Established...
How many record want to be insert?
3
Enter Employee No =
101
Enter Employee Name =
Omkar
Enter Employee Salary =
80000
j = 1
Enter Employee No =
102
Enter Employee Name =
Ashu
Enter Employee Salary =
60000
j = 1
Enter Employee No =
103
```

```
Enter Employee Name =
Rahul
Enter Employee Salary =
50000
j = 1
Statement Created...
*** Employee Record ***

Emp No    Emp Name    Emp Salary
 101        Omkar       80000
 102        Ashu        60000
 103        Rahul       50000
Enter Employee No to be Search =
102
Statement Created...
*** Employee Record ***

Emp No    Emp Name    Emp Salary
 102        Ashu     60000

Enter Employee Name to be Search =
Omkar
Statement Created...
*** Employee Record ***
Emp No    Emp Name    Emp Salary
 101        Omkar       80000
```

Program 1.8: Program to create the table student with the fields roll number, name, percentage using PostgreSQL. Write a menu driven program to perform the following operations on student table.

(a) Insert
(b) Modify
(c) Delete
(d) Search
(e) View All
(f) Exit

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

public class Jdbcstudent
{
   public static void main(String[] args)
   {
      int choice,choice1;
```

```
Connection con=null;
Statement sm=null;
ResultSet rs=null;
BufferedReader br=new BufferedReader
                                    (new InputStreamReader(System.in));
try
{
   class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  con = DriverManager.getConnection("jdbc:odbc:EMP", "root", "xyz");
                          // Step 3 create connectin
   System.out.println("Connection Established...");
   sm=con.createStatement();
   do
   {
       System.out.println("1.Insert");
       System.out.println("2.Modify");
       System.out.println("1.Delete");
       System.out.println("4.Search");
       System.out.println("5.View All");
       System.out.println("6.Exit");
       System.out.println("Enter Your Choice:\t");
       choice=Integer.parseInt(br.readLine());
       switch (choice)
       {
          case 1:
          System.out.println("Enter Roll No:\t");
          int r=Integer.parseInt(br.readLine());
          System.out.println("Enter Name:\t");
          String name=br.readLine();
          System.out.println("Enter Percentage:\t");
          float f=Float.parseFloat(br.readLine());
          int result=sm.executeUpdate("insert into
                Student values("+r+",'"+name+"',"+f+");");
          if(result>0)
             System.out.println("Insert Successfully");
          else
             System.out.println("error");
          break;
          case 2:
          System.out.println("Enter Roll No:\t");
          r=Integer.parseInt(br.readLine());
          System.out.println("1.Name");
```

```
System.out.println("2.Percentage");
System.out.println("Enter Your Choice:\t");
choice1=Integer.parseInt(br.readLine());
switch (choice1)
{
   case 1:
   System.out.println("Enter Name:\t");
   name=br.readLine();
   result=sm.executeUpdate("update Student set
   sname='"+name+"'where sno="+r);
   if(result>0)
        System.out.println("Update Successfully");
   else
        System.out.println("error");
   break;
   case 2:
   System.out.println("Enter Percentage:\t");
   f=Float.parseFloat(br.readLine());
   result=sm.executeUpdate("update Student
   set per="+f+"where sno="+r);
   if(result>0)
      System.out.println("Update Successfully");
   else
      System.out.println("error");
   break;
}
break;
case 3:
System.out.println("Enter Roll No:\t");
r=Integer.parseInt(br.readLine());
result=sm.executeUpdate("delete from
Student where sno="+r+";");
if(result>0)
   System.out.println("Deleted Successfully");
else
   System.out.println("error");
break;
case 4:
System.out.println("Enter Roll No:\t");
r=Integer.parseInt(br.readLine());
rs=sm.executeQuery
              ("select * from Student where sno="+r+";");
ResultSetMetaData rsmd=rs.getMetaData();
int col=rsmd.getColumnCount();
```

```
                    for(int i=1;i<=col;i++)
                    System.out.print(rsmd.getColumnLabel(i) +"\t");
                    System.out.println();
                    while(rs.next())
                    {
                       for(int i=1;i<=col;i++)
                       System.out.print(rs.getString(i)+"\t");
                       System.out.println();
                    }
                    break;
                    case 5:
                    rs=sm.executeQuery("select * from Student;");
                    rsmd=rs.getMetaData();
                    col=rsmd.getColumnCount();

                    for(int i=1;i<=col;i++)
                    System.out.print(rsmd.getColumnLabel(i)+"\t");
                    System.out.println();
                    while(rs.next())
                    {
                       for(int i=1;i<=col;i++)
                       System.out.print(rs.getString(i)+"\t");
                       System.out.println();
                    }
                    break;
                    case 6:System.exit(0);
                 }
              }while(true);
           }
           catch(Exception e)
           {
              System.out.println(e);
           }
        }
     }
```

Output:

```
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
```

```
Enter Your Choice:
1
Enter Roll No:
1
Enter Name:
AAA
Enter Percentage:
56
Insert Successfully
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
Enter Your Choice:
1
Enter Roll No:
2
Enter Name:
BBB
Enter Percentage:
89
Insert Successfully
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
Enter Your Choice:
1
Enter Roll No:
3
Enter Name:
ZZZ
Enter Percentage:
92
Insert Successfully
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
```

```
Enter Your Choice:
5
sno     sname      sper
1       AAA        56
2       BBB        89
3       ZZZ        92
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
3
Enter Roll No:
1
Deleted Successfully
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
Enter Your Choice:
5
sno     sname     sper
2       BBB       89
3       ZZZ       92
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
Enter Your Choice:
1
Enter Roll No:
1
Enter Name:
AAA
```

```
Enter Percentage:
75
Insert Successfully
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
Enter Your Choice:
5
sno     sname     sper
2       BBB       89
3       ZZZ       92
1       AAA       75
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
Enter Your Choice:
2
Enter Roll No:
1
1.Name
2.Percentage
Enter Your Choice:
1
Enter Name:
XXX
Update Successfully
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
Enter Your Choice:
5
```

```
sno     sname     sper
2       BBB       89
3       ZZZ       92
1       XXX       75
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
Enter Your Choice:
4
Enter Roll No:
3
sno     sname     sper
3       ZZZ       92
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
Enter Your Choice:
6
```

SUMMARY

- JDBC stands for Java Database Connectivity which is a standard java API for database independent connectivity between Java & a wide range of databases.
- The JDBC API is available in two packages:
  (i) java.sql, and (ii) javax.sql
- There are four types of JDBC drivers known as:
  (i) JDBC-ODBC bridge plus ODBC driver, also called Type 1.
  (ii) Native-API, partly Java driver, also called Type 2.
  (iii) JDBC-Net, pure Java driver, also called Type 3.
  (iv) Native-protocol, pure Java driver, also called Type 4.
- DriverManager.getConnection() methods
  - getConnection(String url)
  - getConnection(String url, Properties prop)
  - getConnection(String url, String user, String password)

- There are three types of statements: Statement, PreparedStatement, CallableStatement
- The ResultSet interface provides methods for retrieving and manipulating the results of executed queries.
- Metadata means data about data. We can get further information from the data. If you have to get metadata of a table like total number of columns, column name, column type etc. This interface is useful because it provides methods to get metadata from the ResultSet object.
- A ResultSetMetaData object is useful when you want to create a generic method with which to process ResultSets.
- By using metadata you can determine the data types of the ResultSet columns and call the correct getXXX() method to retrieve the data.
- The ResultSetMetaData interface provides descriptive information about the columns in a ResultSet such as the number of columns it contains or each column's data type.

Check Your Understanding

1. Which packages contain the JDBC classes?
   (a) java.jdbc and javax.jdbc (b) java.jdbc and java.jdbc.sql
   (c) java.sql and javax.sql (d) java.rdb and javax.rdb
2. Which type of driver provides JDBC access via one or more ODBC drivers?
   (a) Type 1 driver (b) Type 2 driver
   (c) Type 3 driver (d) Type 4 driver
3. Which type of Statement can execute parameterized queries?
   (a) PreparedStatement
   (b) ParameterizedStatement
   (c) ParameterizedStatement and CallableStatement
   (d) All kinds of Statements (i.e. which implement a sub interface of Statement)
4. Which of the following statements is false as far as different type of statements is concern in JDBC?
   (a) Regular Statement (b) Prepared Statement
   (c) Callable Statement (d) Interim Statement
5. The JDBC-ODBC Bridge is
   (a) Three tiered (b) Multithreaded
   (c) Best for any platform (d) All of the above
6. Which driver is called as thin-driver in JDBC?
   (a) Type-4 driver (b) Type-1 driver
   (c) Type-3 driver (d) Type-2 driver

7. Which method is used to perform DML statements in JDBC?
   (a) execute() (b) executeQuery()
   (c) executeUpdate() (d) executeResult()
8. The JDBC API is what allows access to a data source from a Java middle tier?
   (a) True (b) False
9. Which method is used for an SQL statement that is executed frequently?
   (a) prepareStatement (b) prepareCall
   (c) createStatement (d) None of the above
10. The JDBC-ODBC Bridge is _____.
   (a) Multithreaded (b) Single threaded
   (c) Both of the above (d) None of the above
11. In the following JDBC drivers which is known as fully java driver?
   (a) Native-API driver (b) Network Protocol driver
   (c) Thin driver (d) Both (b) and (c)
12. Which of the following JDBC drivers is known as a partially java driver?
   (a) JDBC-ODBC Bridge driver (b) Native-API driver
   (c) Network Protocol driver (d) Thin driver
13. Which of the following is not a valid type of ResultSet?
   (a) ResultSet.TYPE_FORWARD_ONLY
   (b) ResultSet.TYPE_SCROLL_INSENSITIVE
   (c) ResultSet.TYPE_SCROLL_SENSITIVE
   (d) ResultSet.TYPE_BACKWARD_ONLY
14. Which of the following manages a list of database drivers in JDBC?
   (a) DriverManager (b) JDBC driver
   (c) Connection (d) Statement

Answers

1. (c)	2. (a)	3. (a)	4. (d)	5. (b)	6. (a)	7. (c)	8. (a)	9. (a)	10. (a)
11. (d)	12. (b)	13. (d)	14. (a)						

Practice Questions

Q.I Answer the following questions in short:

1. What is the use of SetAutoCommit()?
2. Write short note on: JDBC API Packages.
3. What is database?
4. What is meant by JDBC driver?
5. What is the use of JDBC API?
6. What is ResultSetMetaData?
7. What is DatabaseMetaData?

Q.II Answer the following questions:

1. Differentiate between Statement and PreparedStatement interface.
2. Explain various types of JDBC drivers. Discuss advantages and disadvantages of each.
3. What is a difference between a connection and statement?
4. Explain how to create the URL.
5. Explain the significance of each:
   (i) ResultSet
   (ii) ResultSetMetaData
   (iii) DatabaseMetaData
6. Which to use and when?
   (i) execute()
   (ii) executeUpdate()
   (iii) executeQuery()
   (iv) Statement
   (v) CallabaleStatement
   (vi) PreparedStatement.
7. By considering PreparedStatement interface, explain following methods of it
   (i) setString()
   (ii) setInt()
8. On what record is the cursor initially positioned in the ResultSet?
9. Write a small program to open a connection to a database.
10. Describe JDBC-ODBC Bridge in detail.
11. Explain architecture of JDBC in detail.
12. How to connect Java application to PostgresSQL? Explain with example.

Q.III Define Term:

1. JDBC Driver
2. MetaData
3. Statement
4. ResultSet
5. Callable Statement
6. Prepared Statement

■■■

2... Multithreading

Learning Objectives ...

Students will be able:

- To understand Concepts in Thread.
- To learn Life Cycle of Thread and Implementation of Thread.
- To study Thread Priorities and Execution of Thread Application.
- To understand Synchronization and Inter-thread Communication Concepts.

2.1 INTRODUCTION

- Multithreading is one of the most important feature of Java. Multithreading in Java is a process of executing multiple threads simultaneously. Java provides built-in support for multi-threaded programming.
- Java is a multi-threaded programming language which means we can develop multi-threaded program using Java.
- A multi-threaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources especially when your computer has multiple CPUs.
- By definition multitasking is when multiple processes share common processing resources such as a CPU.
- Multithreading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel.
- The OS (Operating System) divides processing time not only among different applications, but also among each thread within an application.
- Multithreading enables us to write in a way where multiple activities can proceed concurrently in the same program.

2.2 WHAT ARE THREADS?

- A task is completed by a program with the sequence of steps, called as process. Each specific task in a process is called thread.
- When there are multiple threads execute simultaneously, it is called multithreading.
- A thread is a program's path of execution. A thread is a line of execution. A thread is a lightweight sub process, a smallest unit of processing.
- Threads are independent, if there occurs exception in one thread, it doesn't affect other threads. It shares a common memory area.

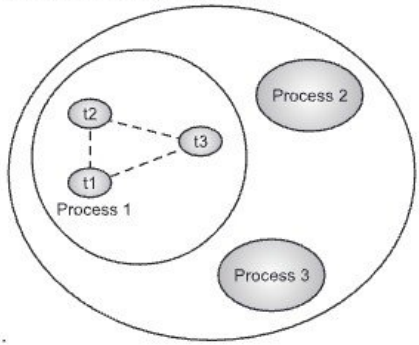
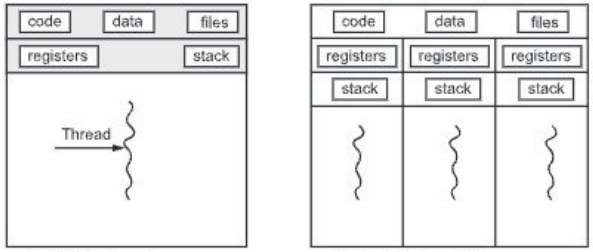


Fig. 2.1: Thread Process

- As shown in the Fig. 2.1, thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS and one process can have multiple threads.
- Conceptually, 'a thread is a basic unit of CPU utilization'. A thread is similar to the sequential programs.
- A single-thread has a beginning, a sequence and an end. A thread comprises a thread ID, a program counter, a register set and a stack.
- A thread shares the code section, data section and other operating system resources [like open files] with other threads belonging to the same process.
- Multi-threaded application has many threads executing concurrently. This is the facility given by Java virtual machine.
- Fig. 2.2 (a) illustrates single-threaded process and Fig. 2.2 (b) illustrates multi-threaded process.



(a) Single-thread Process (b) Multi-threaded Process

Fig. 2.2

2.2.1 Benefits of Multi-threaded Programming

- Following are the benefits of multi-threaded programming:
  1. **Resource sharing:** A thread shares memory and resources by default. The code sharing allows an application to have several thread activity within the same address space. So it is beneficial for memory.
  2. **Responsiveness:** Multi-threaded application gives more response to the users. For example, a web browser could allow us to login even if it is downloading some other image with different thread.
  3. **Economy:** If we allocate memory and resources for each process then it is costly. But here, thread shares resources of the same process, so it becomes economical. For example, in Solaris, creating a process is about 30 times slower than creating a thread.
  4. **Utilization of multiprocessor architecture:** Multithreading increases the usage of multi-CPU architecture.
- In short, we can say that:
  1. Multithreading requires less overhead.
  2. Threads are light weight.
  3. They shares same address space.
  4. Inexpensive inter-process communication.

2.2.2 Disadvantages of Multithreading

- Various disadvantages of multithreading are:
  1. Programming and debugging is more complex.
  2. Programmer may face race conditions or deadlocks.
  3. Thread consumes more processor time.
  4. Operating system spends more time in managing and switching between the threads.

2.3 LIFE CYCLE OF THREAD

- A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. This is also known as life cycle of a thread.
- The life cycle of the thread in java is controlled by JVM. Fig. 2.3 shows the different states of threads.

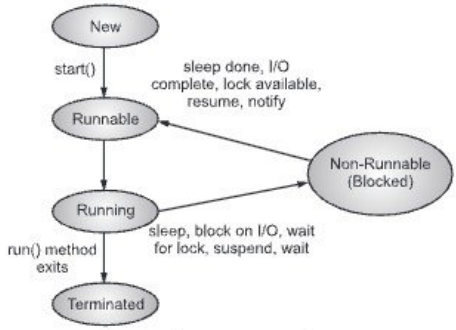


Fig. 2.3: Different states of thread

- Above mentioned stages are explained here:
  1. **New:** A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread. The thread is in new state if you create an instance of Thread class but before the invocation of start() method.
  2. **Runnable:** The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
  3. **Running:** The thread is in running state if the thread scheduler has selected it.
  4. **Non-Runnable (Blocked):** This is the state when the thread is still alive, but is currently not eligible to run.
  5. **Terminated:** A thread is in terminated or dead state when its run() method exits.

2.4 CREATING THREADS

- The Java programming language allows us to create a program that contains one or more parts that can run simultaneously at the same time.
- This type of program is known as a multi-threading program. Each part of this program is called a thread.
- We can create thread in java with two different ways, (See Fig. 2.4):
  1. Extending the thread class.
  2. Implements runnable interface.

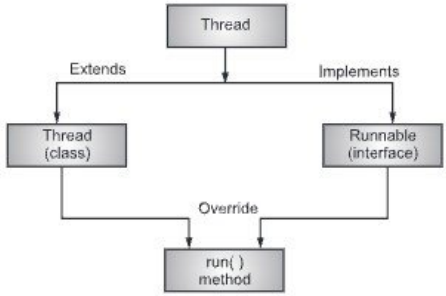


Fig. 2.4: Thread

2.4.1 Using a Thread Class

- The one way to create a thread is to create a new class that extends Thread class using the following two simple steps.
- This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

Step 1: You will need to override run( ) method available in Thread class. This method provides entry point for the thread and you will put your complete business logic inside this method. Following is simple syntax of run( ) method:

```
public void run()
```

Step 2: Once, Thread object is created, you can start it by calling start( ) method, which executes a call to run( ) method. Following is simple syntax of start() method:

```
void start();
```

Methods of Thread Class:

1. `public void start()`: Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
2. `public void run()`: If the Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3. `public final void setName(String name)`: Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4. `public final void setPriority(int priority)`: Sets the priority of this Thread object. The possible values are between 1 and 10.
5. `public final void setDaemon(boolean on)`: A parameter of true denotes this Thread as a daemon thread.

6. `public final void join(long millisec)`: The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
7. `public void interrupt()`: Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8. `public final boolean isAlive()`: Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

- The previous methods are invoked on a particular Thread object. The following methods in the Thread class are static. Invoking one of the static methods performs the operation on the currently running thread.
  1. `public static void yield()`: Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled.
  2. `public static void sleep(long millisec)`: Causes the currently running thread to block for at least the specified number of milliseconds.
  3. `public static boolean holdsLock(Object x)`: Returns true if the current thread holds the lock on the given Object.
  4. `public static Thread currentThread()`: Returns a reference to the currently running thread, which is the thread that invokes this method.

Program 2.1: Program for creating a thread.

```
public class DemoThread extends Thread
{
    public static void main(String[] args)
    {
        Thread thread1 = new Thread("My Thread1");
        Thread thread2 = new Thread("My Thread2");
        thread1.start();
        thread2.start();
        System.out.println("Thread names are following:");
        System.out.println(thread1.getName());
        System.out.println(thread2.getName());
    }
    public void run()
    {
    }
}
```

Output:

```
Thread names are following:
My Thread1
My Thread2
```

Program 2.2: Program for thread creation.

```
class ThreadDemo extends Thread
{
   private Thread t;
   private String threadName;
   ThreadDemo(String name)
   {
      threadName = name;
      System.out.println("Creating " +  threadName );
   }
   public void run()
   {
      System.out.println("Running " +  threadName );
      try
      {
         for(int i = 4; i > 0; i--)
      {
      System.out.println("Thread: " + threadName + ", " + i);
      // Let the thread sleep for a while.
      Thread.sleep(50);
      }
   } catch (InterruptedException e) {
      System.out.println("Thread " +  threadName + " interrupted.");
   }
   System.out.println("Thread " +  threadName + " exiting.");
   }
   public void start ()
   {
      System.out.println("Starting " +  threadName );
     if(t == null)
     {
         t = new Thread (this, threadName);
         t.start ();
     }
   }
}
```

```
public class TestThread
{
   public static void main(String args[])
   {
      ThreadDemo T1 = new ThreadDemo( "Thread-1");
      T1.start();
      ThreadDemo T2 = new ThreadDemo( "Thread-2");
      T2.start();
   }
}
```

Output:

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Running Thread-2
Thread: Thread-1, 4
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-2, 1
Thread: Thread-1, 1
Thread Thread-2 exiting.
Thread Thread-1 exiting.
```

2.4.2 main Thread

- When a Java program starts up, one thread begins running immediately. This is usually called the main thread.
- The main thread creates automatically when program is started.
- The main thread is important for two reasons:
  1. It is the thread from which other "child" threads will be spawned.
  2. Often, it must be the last thread to finish execution because it performs various shutdown actions.
- In every program there will be at least only thread running and it is called the main thread.

Program 2.3: Simple java main thread program.

```
class MainthreadDemo
{
   public static void main (String agrs[])
   {
     System.out.println("main thread");
   }  // end main
} //end class
```

Output:

```
main thread
```

- When this program is executed, it creates main thread. This reference is available in java-run-time system. This is represented in Fig. 2.5.

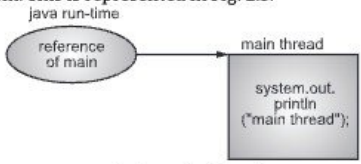


Fig. 2.5: Main thread

- Although the main thread is created automatically when your program is started, it can be controlled through a Thread object.
- To do so, we must obtain a reference to it by calling the method currentThread(), which is a public static member of Thread.
- Its general **form/syntax** is: `static Thread currentThread()`
- This method returns a reference to the thread in which it is called. Once you have a reference to the main thread, you can control it just like any other thread.

Program 2.4: Program for how to get reference of a thread. This is just to check whether a thread is available in a simple java program or not.

```
class Mythread
{
   public static void main(String args[])
   {
      Thread d = Thread.currentThread();     //to find reference
      System.out.println("current thread is" + d);
   }
} // end class
```

Output:

```
current thread isThread[main,5,main]
```

- In the above Program 2.4, we obtained the reference by calling currentThread() method. This reference is stored in a variable of type Thread.
- Here, it displays three things:
  1. name of Thread.
  2. its priority.
  3. group of Thread.
- In java, default name of main Thread is main. By default its priority is 5. Last main is the name of group to which this thread belongs. This is represented in Fig. 2.6.

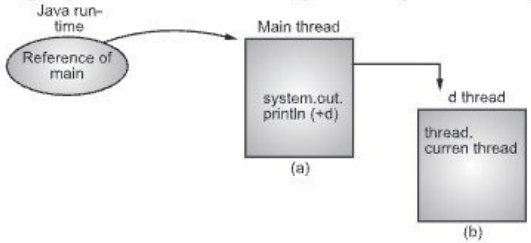


Fig. 2.6: (a) main thread, (b) created thread (when object is created)

Program 2.5: Illustrates the usage of setName() and getPriority() methods.

```
import java.lang.*;
class MytestThread
{
     public static void main(String args[])
     {
         Thread thread = Thread.currentThread();
         System.out.println("current Thread is " + thread);
         thread.setName("mythread");
         System.out.println("New name " + thread);
         int priority = thread.getPriority();
         thread.setPriority(10);
         System.out.println("The priority is : " + thread.getPriority());
     }
}
```

Output:

```
current Thread is Thread[main,5,main]
New name Thread[mythread,5,main]
The priority is : 10
```

2.4.3 Runnable Interface

- This is another easiest way of creating threads and it contains abstract method. If the class is intended to be executed as a thread then we can achieve this by implementing Runnable interface.
- We will need to follow three basic steps:

Step 1: As a first step you need to implement a run() method provided by Runnable interface. This method provides entry point for the thread and you will put your complete business logic inside this method. Following is simple syntax of run() method:

```
public void run( )
```

Step 2: At second step you will instantiate a Thread object using the following constructor:

```
Thread(Runnable threadObj, String threadName);
```

Where, threadObj is an instance of a class that implements the Runnable interface and threadName is the name given to the new thread.

Step 3: Once Thread object is created, you can start it by calling start( ) method, which executes a call to run( ) method. Following is simple syntax of start() method:

```
void start( );
```

Program 2.6: Program creates a new thread and starts it running.

```
class RunnableDemo implements Runnable
{
   private Thread t;
   private String threadName;
   RunnableDemo(String name)
   {
      threadName = name;
      System.out.println("Creating " +  threadName );
   }
   public void run()
   {
      System.out.println("Running" +  threadName );
   try
   {
      for(int i = 4; i > 0; i--)
      {
         System.out.println("Thread: " + threadName + ", " + i);
         // Let the thread sleep for a while.
         Thread.sleep(50);
      }
   }
```

```
        catch (InterruptedException e)
        {
            System.out.println("Thread " +  threadName + " interrupted.");
         }
            System.out.println("Thread " +  threadName + " exiting.");
         }
        public void start ()
        {
            System.out.println("Starting " +  threadName );
        if(t == null)
        {
            t = new Thread (this, threadName);
            t.start ();
        }
    }
    }
    public class TestMyThread
    {
        public static void main(String args[])
        {
            RunnableDemo R1 = new RunnableDemo( "Thread-1");
            R1.start();
            RunnableDemo R2 = new RunnableDemo( "Thread-2");
            R2.start();
        }
    }
```

Output:

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-2
Running Thread-1
Thread: Thread-2, 4
Thread: Thread-1, 4
Thread: Thread-2, 3
```

```
Thread: Thread-1, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-2 exiting.
Thread Thread-1 exiting.
```

2.5 THREAD PRIORITIES

- Thread priorities are the integers which decide how one thread should be treated with respect to the others.
- Thread priority decides when to switch from one running thread to another, process is called context switching.
- Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.
- Java thread priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).
- Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and very much platform dependent.
- To set the priority of the thread setPriority() method is used which is a method of the class Thread Class.

Program 2.7: Program of usage of Runnable interface with priority.

```
class TestMultiPriority extends Thread
{
   public void run()
   {
      System.out.println("Running thread name
                                  is: "+Thread.currentThread().getName());
      System.out.println("Running thread priority
                              is: "+Thread.currentThread().getPriority());
   }
   public static void main(String args[])
   {
      TestMultiPriority m1=new TestMultiPriority();
      TestMultiPriority m2=new TestMultiPriority();
```

```
        m1.setPriority(Thread.MIN_PRIORITY);
        m2.setPriority(Thread.MAX_PRIORITY);
        m1.start();
        m2.start();
    }
}
```

Output:

```
Running thread name is: Thread-1
Running thread priority is: 10
Running thread name is: Thread-0
Running thread priority is: 1
```

2.6 EXECUTION OF THREAD APPLICATION : RUNNING MULTIPLE THREADS

- The first way is to create a class which extends thread class and then create the instance of that class. This extended class must override method run( ).
- It must also call start( ) method. The Thread class is defined in package java.lang; so we have to import it.
- The following code segment tells the definition:

```
import java.lang.*;
public class Coun extends Thread
{
    public void run()
    {
        ________________
        ________________
        ________________
    }
}
```

- This will create a new class Coun and overrides method run( ). The program 2.8 shows how to write Thread program.

Program 2.8: Program for multiple threads.

```
import java.lang.*;
class Cons extends Thread
{
    //constructor
    Cons()
    {
        start();  //starts the thread
    }
```

```
    public void run()
    {
    try
    {
       for (int k=1;k<=5;++k)
       {
          System.out.println("mythread" + k);
          Thread.sleep(500);
       }
    }
    catch(InterruptedException ob)
    {
     }
     System.out.println("Thread exists");
    } // end run
} //end Cons
class Mainthread
{
    public static void main(String args[])
    {
       Cons c = new Cons();
       try
       {
          for (int i=1;i<=5;++i)
          {
             System.out.println("Main Thread" +i);
             Thread.sleep(1000);
          } //end for
       } //end try
       catch (InterruptedException ob)
       {
       }
       System.out.println("main Thread Exists");
    } //end main
} // end mainthread
```

Output:

```
mythread1
Main Thread1
mythread2
```

```
mythread3
Main Thread2
mythread4
mythread5
Main Thread3
Thread exists
Main Thread4
Main Thread5
main Thread Exists
```

- This output may change PC to PC. In this program, two threads main thread and mythread (coun) runs simultaneously.
- In the above example, mythread suspends threads for 500 millisecond by calling its sleep method. The mainthread first get control of CPU and count as 1 and then suspended for 1000 milliseconds.
- The Program 2.9 illustrates the calling of the start method from main method and not from constructor. In a constructor, we will use a super method. It gets one argument as string which is name of a thread. In this program, we will create three different objects.

Program 2.9: Program to use of super in threading.

```
class mythread extends Thread
{
   mythread(String name)
   {
      super(name);
   }
   public void run()
   {
      try
      {
         for(int k=5;k>0;k--)
         {
            System.out.println(getName() + k);
            Thread.sleep(500);
         }
      }
```

```
        catch(InterruptedException e)
        {
            System.out.println("Thread interrupted");
        }
        System.out.println("Thread exists");
      } //end run()
} //end mythread
//Main class for thread object
class Mainthread1
{
   public static void main(String args[])
   {
       mythread ob1 = new mythread ("First");
       mythread ob2 = new mythread ("Second");
       mythread ob3 = new mythread ("Third");
       ob1.start();
       ob2.start();
       ob3.start();
       try
       {
           for(int k=5;k>0;k--)
           {
               System.out.println("main thread" +k);
               Thread.sleep(500);
            }
       }
       catch(InterruptedException e)
       {
          System.out.println("interrupted thread");
       }
          System.out.println("main thread exists");
      } //end main
} //end class mainthread
```

Output:

```
main thread5
Third5
First5
Second5
```

```
Third4
First4
Second4
main thread4
Third3
Second3
First3
main thread3
Third2
First2
Second2
main thread2
Third1
Second1
First1
main thread1
Thread exists
main thread exists
Thread exists
Thread exists
```

- In above Program 2.9, the constructor is parameterized which takes one argument of type string. This argument is the name of thread.
- First, second and third are the names of thread. So in this program, four threads are running simultaneously. This is shown in Fig. 2.7.

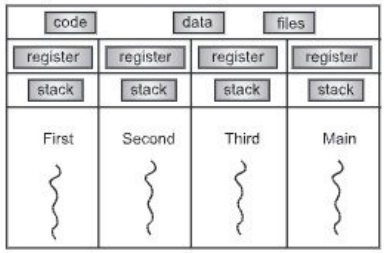


Fig. 2.7: Four threads with CPU

2.7 SYNCHRONIZATION

- In this section we will study synchronization and inter-thread communication in Java.

2.7.1 Synchronization

- Multithreading introduces asynchronous behavior to the programs. If a thread is writing some data another thread may be reading the same data at that time. This may bring inconsistency.
- When two or more threads wants to access a shared resource, then it must ensure that the resource will be used by only one thread at an instant of time. The mechanism of this process is called synchronization.
- Synchronization is the concept of the monitor or semaphore. Monitor works as mutex and restrict to one thread to own a monitor at a given time.
- As the thread acquires the lock, all the threads that want to acquire the monitor will be suspended.
- As the first thread exits from the monitor, one thread will acquire monitor from the waiting list of threads.
- Java programming language provides a very handy way of creating threads and synchronizing their task by using synchronized blocks.
- We keep shared resources within this block. Following is the general form of the synchronized statement:

```
synchronized(objectidentifier) {
   // Access shared variables and other shared resources
}
```

- Here, the objectidentifier is a reference to an object whose lock associates with the monitor that the synchronized statement represents.
- Synchronization in java is the capability to control the access of multiple threads to any shared resource.
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Program 2.10: Program for synchronization.

```
class mythread extends Thread
{
   String msg[]={"Java", "Supports", "Multithreading", "Concept"};
   mythread(String name)
   {
      super(name);
   }
   public void run()
   {
      display(getName());
      System.out.println("Exit from "+getName());
   }
```

```
    synchronized void display(String name ) //Synchrinized method
    {
      for(int i=0;i<msg.length;i++)
      {
         System.out.println(name+msg[i]);
      }
    }
}
/* Main class */
class MySynchro
{
    public static void main(String args[])
    {
       mythread t1=new mythread("Thread 1: ");
       mythread t2=new mythread("Thread 2: ");
       t1.start();
       t2.start();
       System.out.println("Main thread exited");
    }
}
```

Output:

```
Main thread exited
Thread 1: Java
Thread 2: Java
Thread 1: Supports
Thread 1: Multithreading
Thread 1: Concept
Thread 2: Supports
Thread 2: Multithreading
Thread 2: Concept
Exit from Thread 1:
Exit from Thread 2:
```

2.7.2 Inter-thread Communication

- Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other.
- Inter-thread communication is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods like wait( ), notify( ), notifyAll( ) of Object class, let's see in detail.

1. `wait()`: It is used to tell the calling thread to give up the lock and go to sleep until some other thread enters the same lock and calls notify().
   Syntax: `public final void wait(long timeout)`
2. `notify()`: It wakes up the first thread that called wait( ) on the same object.
   Syntax: `public final void notify()`
3. `notifyAll()`: It wakes up all the threads that called wait( ) on the same object. Here, the highest priority thread will run first.
   Syntax: `public final void notifyAll()`

Program 2.11: Program shows how two thread can communicate using wait() and notify() method. We can create a complex system using the same concept.

```
class Chat
{
   boolean flag = false;
   public synchronized void Question(String msg)
   {
      if (flag)
      {
        try
        {
           wait();
        }
        catch (InterruptedException e)
        {
           e.printStackTrace();
        }
     }
     System.out.println(msg);
     flag = true;
     notify();
 }
 public synchronized void Answer(String msg)
 {
     if (!flag)
     {
       try
       {
          wait();
       }
```

```
            catch (InterruptedException e)
            {
               e.printStackTrace();
            }
         }
         System.out.println(msg);
         flag = false;
         notify();
         }
   }
   class T1 implements Runnable
   {
      Chat m;
      String[] s1 = { "Hi", "How are you?", "I am also doing fine!" };
      public T1(Chat m1)
      {
         this.m = m1;
         new Thread(this, "Question").start();
      }
      public void run()
      {
         for (int i = 0; i < s1.length; i++)
         {
            m.Question(s1[i]);
         }
      }
   }
   class T2 implements Runnable
   {
      Chat m;
      String[] s2 = { "Hi", "I am good, what about you?", "Great!" };
      public T2(Chat m2)
      {
         this.m = m2;
         new Thread(this, "Answer").start();
      }
```

```
        public void run()
        {
            for (int i = 0; i < s2.length; i++)
            {
                m.Answer(s2[i]);
            }
        }
    }
    public class MyDemoThread
    {
        public static void main(String[] args)
        {
            Chat m = new Chat();
            new T1(m);
          new T2(m);
        }
    }
```

Output:

```
Hi
Hi
How are you?
I am good, what about you?
I am also doing fine!
Great!
```

Additional Programs

Program 2.12: Program to display the 100, 99, 98..... 1 using thread.

```
class MyThredDemo1
{
    public static void main(String args[])
    {
        Thread t = Thread.currentThread();
        System.out.println("Current thread is: " + t);
        t.setName("Demo Thread");
        System.out.println("After changing the name thread is: " + t);
        try
        {
            for(int n = 100; n> 0; n--)
```

```
            {
              System.out.println(n);
              Thread.sleep(1000);
            }
          }
          catch (InterruptedException e)
          {
             System.out.println("Thread interrupted");
          }
       }
    }
```

Output:

```
Current thread is: Thread[main,5,main]
After changing the name thread is: Thread[Demo Thread,5,main]
100
99
98
97
96
95
94
93
92
91
90
89
88
87
86
85
84
83
82
81
80
79
78
```

```
77
76
75
74
73
72
71
70
69
68
67
66
65
64
63
62
61
60
59
58
57
56
55
54
53
52
51
50
49
48
47
46
45
44
43
42
41
40
```

```
39
38
37
36
35
34
33
32
31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
```

Program 2.13: Program to use of sleep() method.

```
class NewThread implements Runnable
{
    Thread t;
    NewThread()
    {
        t = new Thread(this, "Demo Thread");
        System.out.println("Child Thread: " + t);
        t.start();
    }
    public void run()
    {
        try
        {
            for (int i = 0; i < 5;  i++)
            {
                System.out.println("Child Thread: " + i);
                Thread.sleep(1000);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println("Child Thread Interrupted");
        }
        System.out.println("Exiting child thread");
    }
}
class DemoMyThread2
{
    public static void main(String args[])
    {
        new NewThread();
        try
        {
            for (int i = 0;  i < 5; i++)
            {
```

```
                System.out.println("Main Thread: " + i);
                Thread.sleep(500);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println("Main Thread Interrupted");
        }
        System.out.println("Exiting main thread");
    }
}
```

Output:

```
Child Thread: Thread[Demo Thread,5,main]
Child Thread: 0
Main Thread: 0
Main Thread: 1
Child Thread: 1
Main Thread: 2
Main Thread: 3
Child Thread: 2
Main Thread: 4
Exiting main thread
Child Thread: 3
Child Thread: 4
Exiting child thread
```

Program 2.14: Program to use super().

```
class NewThread extends Thread
{
    NewThread()
    {
        super( "Demo Thread");
        System.out.println("Child Thread : " + this);
        start();
    }
    public void run()
    {
```

```
        try
        {
            for (int i = 0; i < 5;  i++)
            {
                System.out.println("Child Thread: " + i);
                Thread.sleep(1000);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println("Child Thread Interrupted");
        }
        System.out.println("Exiting child thread");
    }
}
class DemoMyThread3
{
    public static void main(String args[])
    {
        new NewThread();
        try
        {
            for (int i = 0;  i < 5; i++)
            {
                System.out.println("Main Thread: " + i);
                Thread.sleep(500);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println("Main Thread Interrupted");
        }
        System.out.println("Exiting main thread");
    }
}
```

Output:

```
Child Thread: Thread[Demo Thread,5,main]
Main Thread: 0
Child Thread: 0
Main Thread: 1
Child Thread: 1
Main Thread: 2
Main Thread: 3
Child Thread: 2
Main Thread: 4
Exiting main thread
Child Thread: 3
Child Thread: 4
Exiting child thread
```

Program 2.15: Java program to display "BYE CORONA..." message n times using Runnable Interface.

```
import java.io.*;
public class DemoMyThread4 implements Runnable
{
   int i, no;
   DemoMyThread4(int n)
   {
      no = n;
   }
   public void run()
   {
      for(i = 1; i<=no; i++)
      {
         System.out.println("BYE CORONA...");
         try
         {
            Thread.sleep(50);
         }
         catch(Exception e)
         {
            System.out.println(e);
         }
      }
   }
```

```
    public static void main(String args[])
    {
      try
      {
        int n;
        System.out.println("\nHow many time you want? ");
        BufferedReader br = new BufferedReader(new
                                          InputStreamReader(System.in));
        String str = br.readLine();
        n = Integer.parseInt(str);
        Thread t = new Thread(new DemoMyThread4(n));
        t.start();
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
    }
}
```

Output:

```
How many time you want?
5
BYE CORONA...
BYE CORONA...
BYE CORONA...
BYE CORONA...
BYE CORONA...
```

Program 2.16: Program to define a thread for printing text on output screen for 'n' number of times. Create 3 threads and run them. Pass the text 'n' parameters to the thread constructor.

Example:

(a) First thread prints "COVID19" 10 times.

(b) Second thread prints "LOCKDOWN2020" 20 times

(c) Third thread prints "VACCINATED2021" 30 times

```
import java.io.*;
import java.lang.String.*;
class TestPrint extends Thread
{
   String msg="";
   int n;
   TestPrint(String msg, int n)
   {
      this.msg=msg;
      this.n=n;
   }
   public void run()
   {
      try
      {
         for(int i=1;i<=n;i++)
         {
            System.out.println(msg+" "+i+" times");
         }
         System.out.println("\n ");
      }
      catch(Exception e)
      {
      }
   }
}
public class DemoMyThread5
{
   public static void main(String args[])
   {
      int n=Integer.parseInt(args[0]);
      TestPrint t1=new TestPrint("COVID19",n);
      t1.start();
      TestPrint t2=new TestPrint("LOCKDOWN2020",n+10);
      t2.start();
      TestPrint t3=new TestPrint("VACCINATED2021",n+20);
      t3.start();
   }
}
```

Output:

```
VACCINATED2021 1 times
LOCKDOWN2020 1 times
COVID19 1 times
LOCKDOWN2020 2 times
VACCINATED2021 2 times
LOCKDOWN2020 3 times
COVID19 2 times
LOCKDOWN2020 4 times
VACCINATED2021 3 times
LOCKDOWN2020 5 times
COVID19 3 times
LOCKDOWN2020 6 times
VACCINATED2021 4 times
LOCKDOWN2020 7 times
COVID19 4 times
LOCKDOWN2020 8 times
VACCINATED2021 5 times
LOCKDOWN2020 9 times
COVID19 5 times
LOCKDOWN2020 10 times
VACCINATED2021 6 times
LOCKDOWN2020 11 times
COVID19 6 times
LOCKDOWN2020 12 times
VACCINATED2021 7 times
LOCKDOWN2020 13 times
COVID19 7 times
LOCKDOWN2020 14 times
VACCINATED2021 8 times
LOCKDOWN2020 15 times
LOCKDOWN2020 16 times
VACCINATED2021 9 times
VACCINATED2021 10 times
LOCKDOWN2020 17 times
VACCINATED2021 11 times
```

```
VACCINATED2021 12 times
VACCINATED2021 13 times
VACCINATED2021 14 times
VACCINATED2021 15 times
VACCINATED2021 16 times
VACCINATED2021 17 times
VACCINATED2021 18 times
VACCINATED2021 19 times
VACCINATED2021 20 times
VACCINATED2021 21 times
VACCINATED2021 22 times
VACCINATED2021 23 times
VACCINATED2021 24 times
VACCINATED2021 25 times
VACCINATED2021 26 times
VACCINATED2021 27 times
```

Program 2.17: Program to use the method to suspend thread.

```
class NewThread implements Runnable
{
   String thName;
   Thread t;
   NewThread(String name)
   {
      thName = name;
      t = new Thread(this);
      System.out.println("Thread started: " + t);
      t.start();
   }
   public void run()
   {
      try
      {
         for (int i = 0; i <10;  i++)
         {
            System.out.println(t.getName()  + "    "  + i);
            Thread.sleep(200);
         }
      }
```

```
        catch (InterruptedException e)
        {
            System.out.println(t + " Interrupted");
        }
        System.out.println(thName + " Exiting");
    }
}
class DemoThread6
{
    public static void main(String args[])
    {
        NewThread th1 = new NewThread("First");
        NewThread th2 = new NewThread("Second");
        try
        {
            Thread.sleep(600);
            th1.t.suspend();
            System.out.println("Thread 1 suspended");
            Thread.sleep(600);
            th1.t.resume();
            System.out.println("Thread 1 resumed");
            th2.t.suspend();
            System.out.println("Thread 2 suspended");
            Thread.sleep(600);
            th2.t.resume();
            System.out.println("Thread 2 resumed");
        }
        catch (InterruptedException e)
        {
            System.out.println("Main Thread Interrupted");
        }
        try
        {
            System.out.println("Waiting for threads to terminate");
            th1.t.join();
            th2.t.join();
        }
```

```
        catch (InterruptedException e)
        {
            System.out.println("Main Thread Interrupted");
        }
        System.out.println("Exiting main thread");
    }
}
```

Output:

```
Thread started: Thread[Thread-0,5,main]
Thread started: Thread[Thread-1,5,main]
Thread-0   0
Thread-1   0
Thread-0   1
Thread-1   1
Thread-0   2
Thread-1   2
Thread 1 suspended
Thread-1   3
Thread-1   4
Thread-1   5
Thread 1 resumed
Thread-0   3
Thread 2 suspended
Thread-0   4
Thread-0   5
Thread-0   6
Thread 2 resumed
Thread-1   6
Waiting for threads to terminate
Thread-1   7
Thread-0   7
Thread-0   8
Thread-1   8
Thread-0   9
Thread-1   9
Second Exiting
First Exiting
Exiting main thread
```

Program 2.18: Program to use the synchronized() method.

```
import java.io.*;
import java.lang.*;
class A
{
   synchronized void first(B b)
   {
      String threadName = Thread.currentThread().getName();
      System.out.println(threadName + "Executing A.first()");
      System.out.println(threadName + "Trying to execute B.last()");
      b.last();
   }
   synchronized void last()
   {
      System.out.println("Inside A.last()");
   }
}
class B
{
   synchronized void first(A a)
   {
      String threadName = Thread.currentThread().getName();
      System.out.println(threadName + "Executing B.first()");
      System.out.println(threadName + "Trying to execute A.last()");
      a.last();
   }
   synchronized void last()
   {
      System.out.println("Inside B.last()");
   }
}
class Deadlock  implements Runnable
{
   A a = new A();
   B b = new B();
```

```
    Deadlock()
    {
        Thread.currentThread().setName("Main Thread");
        Thread t = new Thread(this, "Child Thread");
                t.start();
                a.first(b);
                System.out.println("Back in main thread");
    }
    public void run()
    {
        b.first(a);
        System.out.println("Back in child thread");
    }
    public static void main(String args[]) throws IOException
    {
        new Deadlock();
    }
}
```

Output:

```
Main ThreadExecuting A.first()
Main ThreadTrying to execute B.last()
Child ThreadExecuting B.first()
Child ThreadTrying to execute A.last()
-
-
-
```

Program 2.19: Program to use of wait(), notify() methods.

```
import java.io.*;
import java.lang.*;
class Shared
{
    int a;
    boolean valueChanged = false;
    synchronized int get_data()
    {
        if (!valueChanged)
```

```
        try
        {
            wait();
        }
        catch (InterruptedException e)
        {
            System.out.println(" Interrupted");
        }
        System.out.println("Read: " + a);
        valueChanged = false;
        notify();
        return a;
    }
    synchronized void put_data(int n)
    {
        if (valueChanged)
        try
        {
            wait();
        }
        catch (InterruptedException e)
        {
            System.out.println( "Interrupted");
        }
        this.a = n;
        valueChanged = true;
        System.out.println("Written: " + a);
        notify();
    }
}//Shared End
class Producer implements Runnable
{
    Shared ob;
    Producer (Shared ob)
    {
        this.ob = ob;
        new Thread(this, "Producer").start();
    }
```

```
    public void run()
    {
        int j = 0;
        while(true)
        {
            ob.put_data(j++);
        }
    }
}//Producer End
class Consumer implements Runnable
{
    Shared ob;
    Consumer (Shared ob)
    {
        this.ob = ob;
        new Thread(this, "Producer").start();
    }
    public void run()
    {
        while(true)
        {
            ob.get_data();
        }
    }
}//Consumer End
class DemoThread8
{
    public static void main(String args[])throws IOException
    {
        Shared ob = new Shared();
        new Producer(ob);
        new Consumer(ob);
        System.out.println("Press Cntl+c to stop");
    }
}
```

Output:

```
Press Cntl+c to stop
Written: 0
Read: 0
Written: 1
Read: 1
Written: 2
Read: 2
Written: 3
Read: 3
Written: 4
Read: 4
Written: 5
Read: 5
Written: 6
Read: 6
Written: 7
Read: 7
Written: 8
Read: 8
Written: 9
Read: 9
Written: 10
Read: 10
```

Program 2.20: Program to use of join() methods.

```
import java.io.*;
class Maths_op
{
   int a = 5;
   void add_op(int b)
   {
      try
      {
         Thread.sleep(500);
      }
```

```
        catch (InterruptedException e)
        {
            System.out.println(" Interrupted Thread");
        }
        a += b;
        System.out.println("a = " + a);
    }
}
class NewThread implements Runnable
{
    Thread t;
    Maths_op ob1;
    int op2;
    NewThread(Maths_op ob, int p)
    {
        ob1 = ob;
        op2 = p;
        t = new Thread(this);
        t.start();
    }
    public void run()
    {
        synchronized (ob1)
        {
            ob1.add_op(op2);
        }
    }
}
class DemoThread9
{
    public static void main(String args[])throws IOException
    {
        Maths_op ob = new Maths_op();
        NewThread th1 = new NewThread(ob, 100);
        NewThread th2 = new NewThread( ob, 200);
        NewThread th3 = new NewThread( ob, 300);
```

```
        try
        {
            th1.t.join();
            th2.t.join();
            th3.t.join();
        }
        catch (InterruptedException e)
        {
            System.out.println("Main Thread Interrupted");
        }
    }
}
```

Output:

```
a = 105
a = 405
a = 605
```

Program 2.21: Program to use of priority thread.

```
class NewThread implements Runnable
{
    Thread t;
    NewThread(String thname, int p)
    {
        t = new Thread(this);
        t.setName(thname);
        t.setPriority(p);
    }
    void start()
    {
        t.start();
    }
    public void run()
    {
        try
        {
            for (int i = 0; i <10;  i++)
            {
```

```
                System.out.println(t.getName()  +"  " + i);
            }
        }
        catch (Exception e)
        {
            System.out.println(" Interrupted");
        }
        System.out.println(t.getName()  + " Exiting");
    }
}
class DemoThread10
{
    public static void main(String args[])
    {
        NewThread th1 = new NewThread("COVISHIELD", Thread.NORM_PRIORITY + 2);
        NewThread th2 = new NewThread("COVAXIN", Thread.NORM_PRIORITY - 2);
        th1.start();
        th2.start();
        try
        {
            Thread.sleep(1000);
        }
        catch (InterruptedException e)
        {
            System.out.println("Main Thread Interrupted");
        }
        try
        {
            System.out.println("Waiting for threads to terminate");
            th1.t.join();
            th2.t.join();
        }
        catch (InterruptedException e)
        {
            System.out.println("Main Thread Interrupted");
        }
        System.out.println("Exiting main thread");
    }
}
```

Output:

```
COVISHIELD  0
COVISHIELD  1
COVISHIELD  2
COVISHIELD  3
COVISHIELD  4
COVISHIELD  5
COVISHIELD  6
COVISHIELD  7
COVISHIELD  8
COVISHIELD  9
COVISHIELD Exiting
COVAXIN  0
COVAXIN  1
COVAXIN  2
COVAXIN  3
COVAXIN  4
COVAXIN  5
COVAXIN  6
COVAXIN  7
COVAXIN  8
COVAXIN  9
COVAXIN Exiting
Waiting for threads to terminate
Exiting main thread
```

Summary

- Multithreading is a specialized form of multitasking. Multitasking can be of two types. These are Process based multi tasking and Thread based multitasking.
- Java runtime system depends on threads for many things. In a single threaded environment, when a thread is blocked when it is waiting for a resource. Because of this, the entire program stops running. The benefits of multithreading is that, one thread can pause without stopping the other parts of your program. All other threads continue to run. This helps in improving the efficiency by preventing the waste of CPU cycles.
- When Java starts up, one thread begins running immediately. This thread is called a main thread of a program. This main thread is the one when your program begins its execution.

- During the life time of a thread, it goes into several states mentioned below:
  - Newborn state
  - Runnable state
  - Running state
  - Block state
  - Dead state.
- Thread is created by instantiating an object of type Thread. Java defines two ways for creating a thread.
  - By extending Thread class.
  - By implementing the Runnable interface.
- In Java, Each thread is assigned a priority. Thread priorities are used by the thread scheduler to decide when each thread should be allowed to run.
- To set the threads priority, use setPriority() method. This method is a member of Thread class.
- Thread class also defines several priority constants
  - MIN_PRIORITY = 1
  - NORM_PRIORITY = 5 //default priority
  - MAX_PRIORITY = 10
- When two or more threads wants to access a shared resource, then it must ensure that the resource will be used by only one thread at an instant of time. The mechanism of this process is called synchronization.
- Inter-thread communication is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods like wait( ), notify( ), notifyAll( ) of Object class.

Check Your Understanding

1. In java, Thread can be created by _____.
   (a) extending Thread class (b) Implementing Runnable interface
   (c) both (a) and (b) (d) None of the above
2. To start the thread _____ method is override of the Thread class.
   (a) run() (b) start()
   (c) init() (d) go()
3. A thread becomes non-runnable when?
   (a) Its stop method is invoked (b) Its sleep method is invoked
   (c) Its finish method is invoked (d) Its init method is invoked

4. A method used to temporarily release time for other threads.
   (a) yield() (b) set()
   (c) release() (d) start()
5. A process can have _____.
   (a) only one thread (b) one or multiple thread
   (c) multiple sub-threads (d) multiple sub-threads and thread
6. The process of executing multiple threads simultaneously is termed as _____.
   (a) Multiprocessing (b) Multipurposing
   (c) Multithreading (d) All of the above
7. Which of the following statement is not correct about thread?
   (a) threads use a shared memory area
   (b) context-switching between the threads takes less time than process
   (c) At least one process is required for each thread.
   (d) thread is a heavyweight sub-process
8. This method returns a reference to the currently executing thread object.
   (a) currentThread() (b) isAlive()
   (c) yeild() (d) resume()
9. Which of the following is not a state of thread life cycle?
   (a) New (b) Running
   (c) Terminated (d) Establish
10. Which of this method is used to implement runnable interface?
    (a) run (b) join
    (c) resume (d) interrupt
11. Which of this method can be used to make the main thread to be executed last among all the threads?
    (a) sleep (b) run
    (c) suspend (d) interrupt
12. This method causes the currently executing thread object to pause and allow other threads to execute temporarily.
    (a) join (b) yield
    (c) resume (d) interrupt
13. What is the default priority of a thread?
    (a) 0 (b) 1
    (c) 5 (d) 10

Answers

1. (c)	2. (a)	3. (b)	4. (a)	5. (b)	6. (c)	7. (d)	8. (a)	9. (d)	10. (a)
11. (a)	12. (b)	13. (c)							

Practice Questions

Q.I Answer the following questions in short:

1. What is a thread?
2. What are two different ways used to implement threading in java programs?
3. Explain the concept of thread synchronization?
4. Explain the following methods.
   (a) Sleep()
   (b) resume()
   (c) getPriority()
   (d) getName()
   (e) wait()
   (f) notify()

Q.II Answer the following questions:

1. Explain thread life cycle in detail.
2. Explain the thread priorities with an example.
3. What is synchronization? Why it is used?

Q.III Define Term:

1. Multithreading
2. Thread
3. Synchronization

■■■

3... Networking

Learning Objectives ...

Students will be able:

- To understand the Basics of Networking.
- To know about java.net Package.
- To study about Networking Classes and Interfaces.
- To implement TCP/IP based Server and Client.

3.1 OVERVIEW OF NETWORKING

- Communication in the present day is vital part of our existence.
- In today's computing world, there are many expensive resources such as LASER printers, Fax machines and so on that needs to be shared.
- To facilitate this, robust network have come into existence. Networks allow expensive resources to be shared.
- Any programming environment must provide standard techniques to permit its applications to communicate across a network.
- Java programming environment provides simple yet robust techniques that permit Java applications to communicate across networks and share valuable resources.
- Java communication is built on standard client/server architecture. A server must have a port number on which some software is listener/talker.
- This software is consistently listening for client requests.
- Both the client and server on the networks must be uniquely identified. This is where TCP/IP comes into the picture.
- Using TCP/IP each computer on the network whether a server or a client can be uniquely identified using a number system, called the unique IP address of computer on a network.
- HTTP is Hyper Text Transfer Protocol used for web browsing.
- Computer networks consist of connections between computers and devices.
- Protocol is set of rules and regulations for computer communication.

- Communication on internet is done using either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP).
- When we write a java program communication over network, we are working on application layer.
- The classes defined in java.net package allow us to write a program, which can communicate over network without considering underlying protocol.
- But to decide which classes the program should use, we should understand how these protocol works.
- TCP is a connection based protocol (like telephone call).
- It guarantees that data sent from one end to other actually reaches to other end in the same order.
- HTTP, FTP, Telnet are the examples of applications, which need reliable communication channel.
- The order in which data is sent and received over the network is critical in case of these applications.
- UDP is connectionless protocol.

3.2 NETWORKING BASICS

3.2.1 Sockets

- Most networking java programs communicate using sockets.
- A network socket identifies an endpoint in the network.
- Sockets are the foundation of today's networking world as it allows a single computer to serve any clients. This can be accomplished with the help of a 'port'.
- Port is a numbered socket on a particular machine.
- An IP port on a specific machine is called socket.
- Anything that understands the standard protocol can plug in to the socket and can communicate.
- With electrical socket, it does not matter whether you plug in a lamp or television or music system as long as they work with standard voltage.
- Connection oriented sockets make use of TCP and connection less sockets use UDP.
- TCP based sockets guarantee arrival of data and in correct order, whereas UDP based sockets do not.
- When speaking about network sockets TCP/IP packets and IP addresses immediately come to mind.
- Socket communication takes place via a protocol.
- IP (Internet Protocol) is a low level routing protocol that breaks data into small packets and sends them to an address across a network.
- Transmission Control Protocol (TCP) is a higher level protocol that manages to string together these packets, sorting and retransmitting them as necessary, to robustly and reliably transmit data to an address across a network.

- A third protocol User Datagram Protocol (UDP) does a similar job as TCP and can be used directly to support fast, connectionless transport of packets from source to destination.

3.2.2 Port Numbers

- A computer has a single physical connection to the network. All data destined for a particular computer arrives through that connection. However, the data may be intended for different applications running on the computer. So how does the computer know to which application to forward the data? An answer is through the use of 'ports'.
- In connection-based communication such as TCP, a server application binds a socket to a specific port number. This has the effect of registering the server with the system to receive all data destined for that port. A client can then communicate with the server at the server's port, as illustrated here:

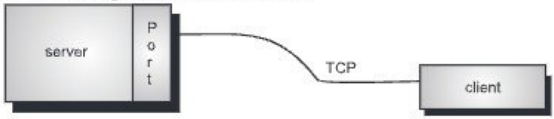


Fig. 3.1: Client Communicating with Server Port

- The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer.
- In datagram-based communication such as UDP, the datagram packet contains the port number of its destination and UDP routes the packet to the appropriate application, as illustrated in this figure:

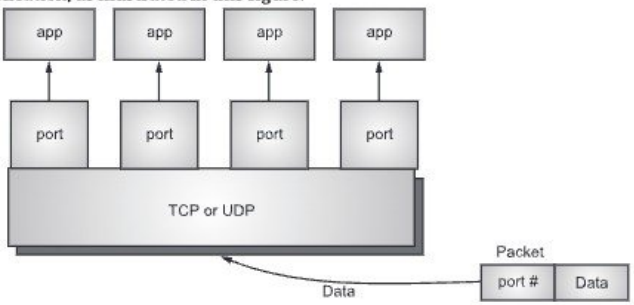


Fig. 3.2: UDP Packet Routing

- Port number is a 16-digit number and so ranges from 0 to 65535.
- The port numbers ranging from 0 - 1023 are restricted; they are reserved for use by well-known services such as HTTP and FTP and other system services. These ports are called well-known ports for the use of well-known services like HTTP, FTP etc. Your applications should not use the well known port numbers.

3.2.3 Protocols

1. Internet Protocol (IP)

- The Internet Protocol (IP) is a Layer 3 protocol, (network layer) that is used to transmit data packets over the Internet. It is undoubtedly the most widely used networking protocol in the world and has spread prolifically.
- Regardless of what type of networking hardware is used, it will almost certainly support IP networking.
- IP acts as a bridge between networks of different types, forming a worldwide network of computers and smaller sub networks.
- The Internet Protocol is a packet-switching network protocol. Information is exchanged between two hosts in the form of IP packets, also known as IP datagram's.
- Each datagram is treated as a discrete unit, not related to any other previously sent packet.
- Instead, a series of datagram's are sent and higher-level protocols at the transport layer provide connection services.

IP Address

- The addressing of IP datagram's is an important issue, as applications require a way to deliver packets to specific machines and to identify the sender. Each host machine under the Internet Protocol has a unique address, the IP address.
- The IP address is a four-byte (32-bit) address, which is usually expressed in dotted decimal format (For example, 192.163.0.6). Although a physical address will normally be issued to a machine, it will not be very useful as the machine goes onto a network.
- Even if somehow every machine could be located by its physical address, if the address changed for any reason (such as installation of a new networking connection or reassignment of the network interface by the administrator), then the machine would no longer be locatable.
- Instead, a new type of address is introduced, that is not bound to a particular physical location. IP address is a numerical number that uniquely identifies a machine on the Internet.
- Typically, one machine has a single IP address, but it can have multiple addresses. A machine could, for example, have more than one network card, or could be assigned multiple IP addresses, (known as **virtual addresses**) so that it can appear to the outside world as many different machines.
- Machines connected to the Internet can send data to that IP address and routers and gateways ensure delivery of the message. In normal programming, only the IP address is needed, the physical address is neither useful nor accessible in Java.

Host Name:

- Numerical address values serve the purposes of computers but they are not designed keeping people in mind. It is almost impossible for a human to remember thousands of 32-bit IP addresses in dotted decimal format.

- A much simpler addressing mechanism is to associate an easy-to-remember textual name with an IP address. This text name is known as the hostname.
- For example, companies on the Internet usually choose a .com address, such as www.microsoft.com or java.sun.com.

2. Transmission Control Protocol

- The Transmission Control Protocol (TCP) is a Layer 4 protocol, (transport layer) that provides guaranteed delivery and ordering of bytes.
- TCP uses the Internet Protocol to send TCP segments, which contain additional information that allows it to order packets and resend them if they are lost.
- TCP also adds an extra layer of abstraction, by using a communications port.
- A communications port is a numerical value (usually in the range 0–65,535) that can be used to distinguish one application or service from another. An IP address can be thought of as the location of a block of apartments and the port as the apartment number.
- One host machine can have many applications connected to one or more ports. An application could connect to a Web server running on a particular host and also to an e-mail server to check for new mail. Ports make all of this possible.
- TCPs main advantage is that it guarantees delivery and ordering of data, providing a simpler programming interface. However, this simplicity comes at a cost, reducing network performance. For faster communication, the User Datagram Protocol may be used.

3. User Datagram Protocol

- User Datagram Protocol (UDP) is a Layer 4 protocol, (transport layer) that applications can use to send packets of data across the Internet, (as opposed to TCP, which sends a sequence of bytes).
- Raw access to IP datagram is not very useful, as there is no easy way to determine which application a packet is for. Like TCP, UDP supports a port number, so it can be used to send datagram's to specific applications and services. Unlike TCP, UDP does not guarantee delivery of packets, or that they will arrive in the correct order.
- The additional error checking of TCP adds overhead and delays, so UDP might be seen to offer better performance. It is sufficient to realize that error-free transmission comes at a cost and UDP can be used as an alternative.

3.2.4 Client Server Architecture

- The term client/server is often mentioned in the context of networking.
- A server is an entity that has some resources that can be shared.
- A client is simply another entity that wants to gain access to those resources.
- It is not always true that "Client sockets run on client sites and server sockets run on server sites".
- Servers can both be servers and clients and same is with clients.

- In a network architecture each computer on network is either a client or server.
- Powerful computers are used as servers.
- PCs or workstations on which user applications are executed work as clients.
- Clients rely on servers for resources like files, devices, processing power.
- Although programs within single computer can implement client/server model, the idea is more applicable to network.
- Client/server model provides convenient way to interconnect programs distributed across different locations.
- Examples of client/servers:
  - Web server and clients are web browsers (use HTTP for communication).
  - Mail server and clients are e-mailing programs (use SMTP for communication).
  - DNS server and clients are any computer systems requesting a web site (converts URL into numeric IP address).
- In a networking environment, a socket on the server allows a client to plug in and access a server's resources. Server socket allows a computer to single handedly serve different clients, different kinds of information. This can be achieved by the introduction of a port, which is a numbered socket on a particular machine.
- To manage multiple client connection, the servers listening process must be multi-threaded.

Reserved Ports

- Basically, TCP/IP reserves the lower 1,024 ports for specific protocols.
- Port number 21 is for FTP, 23 is for TELNET, 25 is for e-mail, 79 is for finger, 80 is for HTTP, 119 is for net news and so on. It is up to each protocol to determine how a client should interact with the port.
- HTTP is the protocol that web browsers and servers use to transfer hypertext pages and images.
- **Client side:** HTTP is quite a simple protocol for a basic web page-browsing web server. When a client requests a web page from an HTTP server, an action (hit) it simply prints the name of the web page in a special format to predefined port and reads back the contents of the web page.
- **Server side:** HTTP server also responds with a status code number to tell the client whether the request can be fulfilled and why.
- The HTTP protocol is much more complicated that this example shows, but this is an actual transaction that we could have with any web server near us.
- If we are planning an application or program that or program incorporates e-mail, we will probably want to work at a higher level and use a library that encapsulates the protocol details.

Proxy Servers:

- A proxy server is software that runs on server that speaks the language of clients. This software hides the actual server from clients that communicate with it.
- It is like a machine that acts as a proxy for application protocol. This is required when client has restrictions on which servers they can connect to.
- This has two advantages: Direct access to internal machine is never established and the proxy server can control the transaction.
- Thus, a client would connect to a proxy server, which did not have such restrictions and the proxy server would in turn service the client while communicating back to back with a server on which the resources required by the client side.
- Proxy server is a bridge between a client application and real server.
- It checks each request to real server, to see it, it can satisfy the request Otherwise, forwards it to real server.
- Main goals of proxy server are:
  - Improve performance: Proxy server saves the results of all the requests. So, after some time it can improve the performance for group of users
  - Proxy server is on the same network as the user and so if data is available on proxy server, it can be fetched efficiently.
  - Filter requests: It can also be used to filter the requests. This helps in preventing access users to certain web-sites
- Thus, proxy server sits between LAN and external network and provides security and efficiency.
- Firewalls are alternatives used now-a-days.
- A proxy server has the additional ability to filter clients request or cache the results of those requests for future use. A caching proxy HTTP server can help reduce the bandwidth demands and a local networks connection to the internet.
- When a popular web site is being hit by hundreds of uses, a proxy server can get the contents of web server's popular pages once and serve these in its cache, thus saving expensive internet data transfers while providing fast access to the same pages on its clients.
- Most proxy servers also keep a record of networking events, to allow network administrators to track unusual communications and their origin in this way, employees visiting inappropriate sites or grouting off during work can be easily monitored.
- This might sound worrisome and introduce some very serious legal and privacy issues, but there are some security concerns addressed by logging, such as identifying disloyal employees who are visiting job-search sites or sending information to competitors.

Internet Addressing:

- The internet is the world's largest network and it is growing bigger every day. Every computer on the internet has a unique IP address.
- An internet address is the number that uniquely indentifies each computer on the internet just like any other network.
- Internet addresses consist of 32 bit values, organized into 4 groups of 8 bits each separated by dot[.].
- These addresses are represented in a format called dotted-decimal notation.
- In this notation each group of 8 bits, called as octet, is represented as decimal equivalent.
- Along with IP address, we also need to specify port for accessing data from a computer on internet e.g. port number for HTTP protocol is 80.
- Port number is specified using a colon after IP address.
- If port number is not specified default port for the protocol is used (e.g. port 80 for HTTP).
- The first few bits of IP address define which class of network lettered as A, B, C, and D or E, most internet users are on a class C network.
- The internet would not be a very friendly place to navigate if everyone had to refer to their addresses as a series of numbers separated by a period.
- For example, it is hard to imagine seeing http://192.168.90.1 at the bottom of an advertisement.
- A clearing house exists for a parallel hierarchy of names bound to each of these unique numbers. It is called the Domain Naming Service (DNS).
- Just as the four numbers of an IP address describes a network hierarchy from left to right, the name of an internet address, called its domain name, describes a machines location in a name space, from right to left.
- Under java such addresses are represented by the java.net.InetAddress class. This class can fill a variety of tasks, from resolving an IP address to looking up the hostname.

3.3 java.net - NETWORKING CLASSES AND INTERFACES

- The package **java.net** provides different classes for implementing networking applications.
- Using the socket classes, you can communicate with any server on the internet or you can implement your own internet server.
- A number of classes are provided to make it convenient to use Universal Resource Locators (URLs) to retrieve data on the internet.
- The java.net package also allows you to do low level networking with Datagram Packet Objects, which may be sent or received over the network through a Datagram Socket object.

3.3.1 Classes

- Java supports both TCP and UDP protocol families.
- URL and URLConnection classes provide a quick and easy way to access content using uniform resource locators.
- The URLEncoder class provides a way to convert text for use as arguments for CGI scripts.
- Server Socket, Datagram Socket and InetAddress are the classes that provide access to plain bare-bone networking facilities. They are the building blocks for implementing new protocols, talking to pre-existing servers and the like.
- The Content Handler and URLStreamHandler are the abstract classes used to extend the capabilities of URL class.
- The following table lists all of the classes in the package along with a brief description of what functionality each provides.

Class	Purpose
`URL`	Represents a Uniform Resource Locator.
`URLConnection`	Retrieves content addressed by URL objects.
`Socket`	Provides a TCP (connected, ordered stream) socket.
`Sever Socket`	Provides a server (listening) TCP socket.
`Datagram Socket`	Provides a UDP (connectionless datagram) socket.
`Datagram Packet`	Represents a datagram to be sent using a Datagram Socket.
`InetAddress`	Represents a host name and its corresponding IPnumber or numbers.
`URLEncoder`	Encodes text in the x-www-form-urlencoded format.
`URLStreamHandler`	Subclasses implement communications streams for different URL protocols.
`ContentHandler`	Subclasses know how to turn MIME objects into corresponding Java objects.
`SocketImpl`	Subclasses provide access to TCP/IP facilities.

3.3.2 Interfaces

- The java.net package defines three interfaces. These primarily are used behind the scenes by the other networking classes rather than by user classes.
- Unless you are porting Java to a new platform or are extending it to use a new socket protocol, you probably will have no need to implement these interfaces in a class.

- They are included here for completeness:
  1. **SocketImplFactory:** This interface defines a method that returns a SocketImpl instance appropriate to the underlying operating system.
  2. **URLStreamHandlerFactory:** Classes that implement this interface provide a mapping from protocols such as HTTP or FTP into the corresponding URLStreamHandler subclasses.
  3. **ContentHandlerFactory:** The URLStreamHandler class uses this interface to obtain ContentHandler objects for different content types. The interface specifies one method, createContentHandler(), which takes the MIME type for which a handler is desired as a String.

InetAddress

- This class is used to encapsulate both the numerical IP address and the domain name for that address.
- A host on the Internet can be represented either in dotted decimal format as an IP address or as a hostname such as yahoo.com.
- In Java, such addresses are represented by the java.net.InetAddress class.
- This class can fill a variety of tasks, from resolving an IP address to looking up the hostname.

Factory Methods and Instance Methods

- The InetAddress class is used to represent IP addresses within a Java networking application.
- This class has no visible (public) constructors. Instead, there are static methods that return InetAddress instances.
- Three commonly used InetAddress methods are shown here:
  1. `Static InetAddress[] getAllByName (String hostname) throws java.net.UnknownHostException, java.lang.SecurityException:` Returns, as a static method, an array of InetAddress instances representing the specified hostname. While most machines will have a single IP address, there are some situations in which one hostname can be mapped to many machines and/or a hostname can map too many addresses on one machine (virtual addresses). If the host cannot be resolved, or if resolving the host conflicts with the security manager, an exception will be thrown.
  2. `Static InetAddress get Byname (String hostname) throws java.net. UnknownHostException, java.lang.SecurityException:` Returns a netAddress instance representing the specified hostname, which may be represented as either as a text hostname (e.g., davidreilly.com) or as an IP address in dotted decimal format. If the host cannot be resolved, or resolving the host conflicts with the security manager, an exception will be thrown.

3. `Static InetAddress getLocalHost() throws java.net.UnknownHost Exception, java.lang.SecurityException`: Returns, as a static method, the IP address of the local host machine. If the IP address cannot be determined, or doing so conflicts with the security manager, then an exception will be thrown.

Instance Methods

- InetAddress class can contain several other methods which can be used on the objects returned by the methods just discussed.
  1. `Boolean equals(Object obj)`: Compares two IP addresses and returns "true" if there is a match.
  2. `Byte [] getAddress()`: Returns the IP address in byte format (as opposed to dotted decimal notation). The bytes are returned in network byte order, with the highest byte as bytearray[0].
  3. `String getHostAddress()`: Returns the IP address of the InetAddress in dotted decimal format.
  4. `String getHostName() throws java.lang.SecurityManager`: Returns the hostname of the InetAddress.
  5. `Boolean isMulticastAddress()`: Returns "true" if the InetAddress is a multicast address, also known as a Class D address.
  6. `String toString()`: Returns a string representation of the InetAddress. Readers are advised to use the getHostName() and getHostAddress() methods, to control which type of data is being requested.

Using InetAddress to determine localhost

- The first and most simple, example of InetAddress is to find out the IP address of the current machine.
- If a direct connection to the Internet exists, a meaningful result will be obtained, but dial-up users and those without any internet connection, (such as in an intranet environment) may get the loopback address of 127.0.0.1.
- The short example program given below shows how it is possible to determine the address.

Program 3.1: Program for LocalHostDemo.

```
import java.net.*;
public class LocalHostDemo
{
public static void main(String args[])
{
System.out.println ("Looking up local host");
try
{
// Get the local host
```

```
InetAddress localAddress = InetAddress.getLocalHost();
System.out.println ("IP address: "+ localAddress.getHostAddress());
}
catch (UnknownHostException uhe)
{
System.out.println ("Error - unable to resolve localhost");
}
}
}
```

How LocalHostDemo Works?

- The LocalHostDemo application starts by prompting the user that an IP address lookup will be performed, (this is important if there is any delay in determining the IP address).
- The networking operation must be enclosed within a try/catch block, because it is possible that no IP address will be found and an exception will be thrown.
- Using the static method InetAddress.getLocalHost(), we obtain an object representing an IP address.
- To display the address in dotted decimal notation, the `InetAddress.getHostAddress()` method is used.

Using InetAddress to find out about other addresses

- The previous example familiarized you with the InetAddress class.
- In this example, we use the static method, InetAddress.getByName() to return an InetAddress instance. We then display the IP address and hostname, using the same methods in LocalHostDemo.

Running NetworkResolverDemo

- This application requires as a parameter either a hostname or an IP address.

Program 3.2: Program for NetworkResolverDemo.

```
//java NetworkResolverDemo
import java.net.*;
public class ExampleByName {
public static void main (String[] args) {
try {
InetAddress address = InetAddress.getByName("www.yahoo.com");
System.out.println(address);
}
catch (UnknownHostException e) {
System.out.println("Could not find www.yahoo.com");
}
}
}
```

- On rare occasions, you will need to connect to a machine that does not have a hostname. In this case, you can pass a String containing the dotted quad form of the IP address to,

```
InetAddress.getByName(): InetAddress address =
                                        InetAddress.getByName("196.148.60.9");
```

Uses the IP address for www.yahoo.com instead of the name.

Program 3.3: Program that prints the address of 196.148.60.9.

```
import java.net.*;
public class ExampleByAddress
{
public static void main (String[] args)
{
try
{
InetAddress address = InetAddress.getByName("196.14860.9");
System.out.println(address);
}
catch (UnknownHostException e) {
System.out.println("Could not find 196.148.60.9");
}
}
}
```

- Some computers have more than one Internet address. Given a hostname, InetAddress.getAllByName() returns an array that contains all the addresses corresponding to that name. Its use is straightforward:

```
InetAddress[] addresses =  InetAddress.getAllByName("www.microsoft.com");
```

Like `InetAddress.getByName()`, `InetAddress.getAllByName()`

It can throw an `UnknownHostException`, so you need to enclose it in a `try` block or declare that your method throws `UnknownHostException`. Example below demonstrates by returning a complete list of the IP addresses for www.microsoft.com.

Program 3.4: Program that prints all the addresses of www.microsoft.com

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

```
try
{
    InetAddress[] addresses = InetAddress.getAllByName("www.microsoft.com");
    for (int i = 0; i < addresses.length; i++)
    {
        System.out.println(addresses[i]);
    }
}
catch (UnknownHostException e)
{
    System.out.println("Could not find www.microsoft.com");
}
}
}
```

3.4 IMPLEMENTING TCP/IP BASED SERVER AND CLIENT

- The Transmission Control Protocol (TCP) is a stream-based method of network communication that is far different from any discussed previously. This section discusses TCP streams and how they operate under Java.
- TCP provides an interface to network communications that is radically different from the User Datagram Protocol (UDP).
- The properties of TCP make it highly attractive to network programmers, as it simplifies network communication by removing many of the obstacles of UDP, such as ordering of packets and packet loss.
- While UDP is concerned with the transmission of packets of data, TCP focuses on establishing a network connection, through which a stream of bytes may be sent and received.
- In previous topic we saw that packets may be sent through a network using various paths and may arrive at different times.
- This benefits performance and robustness, as the loss of a single packet does not necessarily disrupt the transmission of other packets. Such a system creates extra work for programmers who need to guarantee delivery of data.
- TCP eliminates this extra work by guaranteeing delivery and order, providing for a reliable byte communication stream between client and server that supports two-way communication.
- It establishes a "virtual connection" between two machines, through which streams of data may be sent, (see Fig. 3.3).

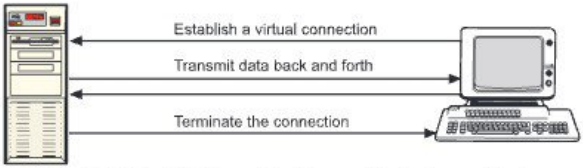


Fig. 3.3: TCP establishes a virtual connection to transmit data

- TCP uses a lower-level communications protocol, the Internet Protocol (IP), to establish the connection between machines.
- This connection provides an interface that allows streams of bytes to be sent and received and transparently converts the data into IP datagram packets.
- A common problem with UDP datagram is that they do not guarantee the arrival of packets at destination.
- TCP takes care of this problem. It provides guaranteed delivery of bytes of data.
- The virtual connection between two machines is represented by a socket.
- Sockets, allow data to be sent and received; there are substantial differences between a UDP socket and a TCP socket.
  - First, TCP sockets are connected to a single machine, where as UDP sockets may transmit or receive data from multiple machines.
  - Second, UDP sockets only send and receive packets of data, whereas TCP allows transmission of data through byte streams, (represented as an InputStream and OutputStream). They are converted into datagram packets for transmission over the network, without requiring the programmer to interfere.

3.4.1 Communication between Applications using Ports

- It is clear that there are significant differences between TCP and UDP, but there is also an important similarity between these two protocols.
- Both share the concept of a communications port, which distinguishes one application from another.
- When a TCP socket establishes a connection to another machine, it requires two very important pieces of information to connect to the remote end, the IP address of the machine and the port number.
- In addition, a local IP address and port number will be bound to it, so that the remote machine can identify which application established the connection.

3.4.2 Socket Operations

- TCP sockets can perform a variety of operations. They can:
  1. Establish a connection to a remote host.
  2. Send data to a remote host.
  3. Receive data from a remote host.
  4. Close a connection.

- In addition, there is a special type of socket that provides a service that will bind to a specific port number. This type of socket is normally used only in servers and can perform the following operations.

3.4.3 TCP and Client/Server Paradigm

- In network programming, (and often in other forms of communication, such as database programming), applications that use sockets are divided into two categories, the client and the server.
- You are probably familiar with the phrase client/server programming, although the exact meaning of the phrase may be unclear to you. This paradigm is the subject of the discussion below.

3.4.3.1 Client/Server Paradigm

- The client/server paradigm divides software into two categories, clients and servers.
- A client is software that initiates a connection and sends requests, whereas a server is software that listens for connections and processes requests.
- In the context of UDP programming, no actual connection is established and UDP applications may both initiate and receive requests on the same socket.
- In the context of TCP, where connections are established between machines, the client/server paradigm is much more relevant.

3.4.4 TCP Sockets

- Java offers good support for TCP sockets, in the form of two socket classes, java.net.Socket and java.net.ServerSocket. When writing client software that connects to an existing service, the Socket class should be used. When writing server software that binds to a local port in order to provide a service, the ServerSocket class should be employed.

3.4.4.1 Socket Class

- The Socket class represents client sockets and is a communication channel between two TCP communications ports belonging to one or two machines.
- A socket may connect to a port on the local system, avoiding the need for a second machine, but most network software will usually involve two machines.
- TCP sockets cannot communicate with more than two machines. If this functionality is required, a client application should establish multiple socket connections, one for each machine.
- **Constructors:** The constructor for the java.net.Socket class is,

```
Socket (String ipaddress, int portno)
```

- The easiest way to create a socket is to specify the hostname of the machine and the port of the service.

- For example, to connect to a Web server on port 80, the following code might be used:

```
try
{
// Connect to the specified host and port
Socket mySocket = new Socket ("www.awl.com", 80);
// ......
}
catch (Exception e)
{
System.err.println ("Err - " + e);
}
```

- However, a wide range of constructors is available, for different situations. Unless otherwise specified, all constructors are public.
  1. `Protected Socket()`: Creates an unconnected socket using the default implementation provided by the current socket factory. Developers should not normally use this method, as it does not allow a hostname or port to be specified.
  2. `Socket(InetAddress address int port) throws java.io.IO-Exception, java.lang.SecurityException`: Creates a socket connected to the specified IP address and port. If a connection cannot be established, or if connecting to that host violates a security restriction (such as when an applet tries to connect to a machine other than the machine from which it was loaded), an exception is thrown.
  3. `Socket(InetAddress address, int port, InetAddress local-Address int localPort) throws ava.io.IOException, java.-lang.SecurityException`: Creates a socket connected to the specified address and port and is bound to the specified local address and local port. By default, a free port is used, but this method allows you to specify a specific port number, as well as a specific address, in the case of multihomed hosts (i.e., a machine where the localhost is known by two or more IP addresses).
  4. `Protected Socket(SocketImpl implementation)`: Creates an unconnected socket using the specified socket implementation. Developers should not normally use this method, as it does not allow a hostname or port to be specified.
  5. `Socket(String host, int port) throws java.net.UnknownHost-Exception, java.io.IOException, java.lang.SecurityException`: Creates a socket connected to the specified host and port. This method allows a string to be specified, rather than an InetAddress. If the hostname could not be resolved, a connection could not be established, or a security restriction is violated, an exception is thrown.

6. `Socket(String host, int port, InetAddress localAddress, int localPort) throws java.net.UnknownHostException, java.io. IOException, java.lang.SecurityException`: Creates a socket connected to the specified host and port and bound to the specified local port and address. This allows a hostname to be specified as a string and not an InetAddress instance, as well as allowing a specific local address and port to be bound to. These local parameters are useful for multihomed hosts (i.e., a machine where the localhost is known by two or more IP addresses). If the hostname cannot be resolved, a connection cannot be established, or a security restriction is violated, an exception is thrown.

3.4.4.2 Using a Socket

- Sockets can perform a variety of tasks, such as reading information, sending data, closing a connection and setting socket options.
- In addition, the following methods are provided to obtain information about a socket, such as address and port locations.
- **Methods:**

1. `Void close() throws java.io.IOException`: Closes the socket connection. Closing a connect may or may not allow remaining data to be sent, depending on the value of the SO_LINGER socket option. Developers are advised to flush any output streams before closing a socket connection.
2. `InetAddress getInetAddress()`: Returns the address of the remote machine that is connected to the socket.
3. `InputStream getInputStream() throws java.io.IOException`: Returns an input stream, which reads from the application this socket is connected to.
4. `OutputStream getOutputStream() throws java.io.IOException`: Returns an output stream, which writes to the application that this socket is connected to.
5. `Boolean getKeepAlive() throws java.net.SocketException`: Returns the state of the SO_KEEPALIVE socket option.
6. `InetAddress getLocalAddress()`: Returns the local address associated with the socket, (useful in the case of multihomed machines).
7. `int getLocalPort()`: Returns the port number that the socket is bound to on the local machine.
8. `int getPort()`: Returns the port number of the remote service to which the socket is connected.
9. `int getReceiveBufferSize() throws java.net.SocketException`: Returns the receive buffer size used by the socket, determined by the value of the SO_RCVBUF socket option.
10. `int getSendBufferSize() throws java.net.SocketException`: Returns the send buffer size used by the socket, determined by the value of the SO_SNDBUF socket option.

11. `int getSoTimeout() throws java.net.SocketException`: Returns the value of the SO_TIMEOUT socket option, which controls how many milliseconds a read operation will block for. If a value of 0 is returned, the timer is disabled and a thread will block indefinitely, (until data is available or the stream is terminated).

3.4.4.3 Reading from and Writing to TCP Sockets

- Creating client software that uses TCP for communication is extremely easy in Java, no matter what operating system is being used.
- The Java Networking API provides a consistent, platform neutral interface that allows client applications to connect to remote services.
- Once, a socket is created, it is connected and ready to read/write by using the socket's input and output streams. These streams do not need to be created; they are provided by the Socket.getInputStream() and Socket.getOutputStream() methods.
- A filter can easily be connected to a socket stream, to make for simpler programming.
- The following code snippet demonstrates a simple TCP client that connects a BufferedReader to the socket input stream and a PrintStream to the socket output stream.

```
try
{
// Connect a socket to some host machine and port
Socket socket = new Socket (somehost, someport);
// Connect a buffered reader
BufferedReader reader = new BufferedReader (new InputStreamReader
                                                (socket.getInputStream()));
// Connect a print stream
PrintStream pstream = new PrintStream(socket.getOutputStream());
}
catch (Exception t)
{
System.err.println ("Error - " + t);
}
```

3.4.5 Creating a TCP Client

- Having discussed the functionality of the Socket class, we will now examine a complete TCP client.
- The client will look at here is a daytime client, which, as its name suggests, connects to a daytime server to read the current day and time.
- Establishing a socket connection and reading from it is a fairly simple process, requiring very little code. By default, the daytime service runs on port 13.
- Not every machine has a daytime server running, but a UNIX server would be a good system to run the client against.

Program 3.5: Program for DaytimeClient.

```
import java.net.*
import java.io.*;
public class DaytimeClient
{
public static final int SERVICE_PORT = 13;
public static void main(String args[])
{
// Check for hostname parameter
if (args.length != 1)
{
   System.out.println ("Syntax - DaytimeClient host");
   return;
}
// Get the hostname of server
String hostname = args[0];
try
{
   / Get a socket to the daytime service
   Socket daytime = new Socket (hostname, SERVICE_PORT);
   System.out.println ("Connection established");
   // Set the socket option just in case server stalls
   daytime.setSoTimeout ( 2000 );
   // Read from the server
   BufferedReader reader = new BufferedReader
                        (new InputStreamReader(daytime.getInputStream()));
   System.out.println ("Results: "+reader.readLine());
   // Close the connection
   daytime.close();
}
catch (IOException ioe)
{
   System.err.println ("Error " + ioe);
}
}
}
```

3.4.5.1 How Daytime Client Works?

- The daytime application is straightforward and uses concepts already discussed.
- A socket is created, an input stream is obtained and timeouts are enabled in the rare event that a server as simple as daytime fails during a connection.
- Rather than connecting a filtered stream, a buffered reader is connected to the socket input stream and the results are displayed to the user.
- Finally, the client terminates after closing the socket connection. This is about as simple a socket client as you can get, complexity comes from implementing network protocols, not from network-specific coding.

3.4.5.2 Running Daytime Client

- Running the application is easy. Simply specify the hostname of a machine running the daytime service as a command-line parameter and run it.
- If you use a nonstandard port for the daytime server (discussed later), remember to change the port number in the client and recompile.
- For example, to run the client against a server running on the local machine, the following command would be used:

```
java DaytimeClient localhost
```

3.4.6 ServerSocket Class

- A special type of socket, the server socket, is used to provide TCP services. Client sockets bind to any free port on the local machine and connect to a specific server port and host.
- The difference with server sockets is that they bind to a specific port on the local machine, so that remote clients may locate a service.
- Client socket connections will connect to only one machine, whereas server sockets are capable of fulfilling the requests of multiple clients.
- The way it works is simple, clients are aware of a service running on a particular port, (usually the port number is well known and used for particular protocols, but servers may run on nonstandard port numbers as well). They establish a connection and within the server, the connection is accepted.
- Multiple connections can be accepted at the same time, or a server may choose to accept only one connection at any given moment.
- Once accepted, the connection is represented as a normal socket in the form of a Socket object once you have mastered the Socket class it becomes almost as simple to write servers as it does clients.
- The only difference between a server and a client is that the server binds to a specific port, using a Server Socket object.
- This Server Socket object acts as a factory for client connections you do not need to create instances of the Socket class yourself.
- These connections are modeled as a normal socket, so you can connect input and output filter streams, (or even a reader and writer) to the connection.

3.4.6.1 Creating a ServerSocket

- Once, a server socket is created, it will be bound to a local port and ready to accept incoming connections.
- When clients attempt to connect, they are placed into a queue. Once, all free space in the queue is exhausted, further clients will be refused.
- **Constructors:** The simplest way to create a server socket is to bind to a local address, which is specified as the only parameter, using a constructor. For example, to provide a service on port 80 (usually used for Web servers), the following snippet of code would be used:

```
try
{
// Bind to port 80, to provide a TCP service (like HTTP)
ServerSocket myServer = new ServerSocket (80);
// ......
}
catch (IOException t)
{
System.err.println ("I/O error - " + t);
}
```

- This is the simplest form of the Server Socket constructor, but there are several others that allow additional customization. All of these constructors are marked as public.
  1. `ServerSocket(int port) throws java.io.IOException, java.lang.SecurityException`: Binds the server socket to the specified port number, so that remote clients may locate the TCP service. If a value of zero is passed, any free port will be used—however, clients will be unable to access the service unless notified somehow of the port number. By default, the queue size is set to 50, but an alternate constructor is provided that allows modification of this setting. If the port is already bound, or security restrictions (such as security policies or operating system restrictions on well-known ports) prevent access, an exception is thrown.
  2. `ServerSocket(int port, int numberOfClients) throws java.io. IOException, java.lang.SecurityException`: Binds the server socket to the specified port number and allocates sufficient space to the queue to support the specified number of client sockets.
  3. `ServerSocket(int port, int numberOfClients, InetAddress address) throws java.io.IOException, java.lang.Security-Exception`: Binds the server socket to the specified port number and allocates sufficient space to the queue to support the specified number of client sockets. This is an overloaded version of the ServerSocket(int port, int numberOfClients) constructor that allows a server

socket to bind to a specific IP address, in the case of a multihomed machine. For example, a machine may have two network cards, or may be configured to represent itself as several machines by using virtual IP addresses. Specifying a null value for the address will cause the server socket to accept requests on all local addresses. If the port is already bound or security restrictions prevent access, an exception is thrown.

3.4.6.2 Using a ServerSocket

- While the Socket class is fairly versatile and has many methods, the Server Socket class does not really do that much, other than accept connections and act as a factory for Socket objects that model the connection between client and server.
- The most important method is the accept() method, which accepts client connection requests, but there are several others that developers may find useful.
- **Methods:** All methods are public unless otherwise noted.
  1. `Socket accept() throws java.io.IOException, java.lang.-Security Exception:` Waits for a client to request a connection to the server socket and accepts it. This is a blocking I/O operation and will not return until a connection is made (unless the timeout socket option is set). When a connection is established, it will be returned as a Socket object. When accepting connections, each client request will be verified by the default security manager, which makes it possible to accept certain IP addresses and block others, causing an exception to be thrown. However, servers do not need to rely on the security manager to block or terminate connections—the identity of a client can be determined by calling the getInetAddress() method of the client socket.
  2. `Void close() throws java.io.IOException:` Closes the server socket, which unbinds the TCP port and allows other services to use it.
  3. `InetAddress getInetAddress():` Returns the address of the server socket, which may be different from the local address in the case of a multihomed machine (i.e., a machine whose localhost is known by two or more IP addresses).
  4. `int getLocalPort():` Returns the port number to which the server socket is bound.
  5. `int getSoTimeout() throws java.io.IOException:` Returns the value of the timeout socket option, which determines how many milliseconds an accept() operation can block for. If a value of zero is returned, the accept operation blocks indefinitely.

3.4.6.3 Accepting and Processing Requests from TCP Clients

- The most important function of a server socket is to accept client sockets. Once, a client socket is obtained, the server can perform all the "real work" of server programming, which involves reading from and writing to the socket to implement a network protocol. The exact data that is sent or received is dependent on the details of the protocol.

- For example, a mail server that provides access to stored messages would listen to commands and send back message contents.
- A TELNET server would listen for keystrokes and pass these to a log-in shell and send back output to the network client. Protocol-specific actions are less network and more programming-oriented.
- The following snippet shows how client sockets are accepted and how I/O streams may be connected to the client:

```
// Perform a blocking read operation, to read the next socket
// connection
Socket nextSocket = someServerSocket.accept();
// Connect a filter reader and writer to the stream
BufferedReader reader = new BufferedReader (new InputStreamReader
                                        (nextSocket.getInputStream()));
PrintWriter writer = new PrintWriter
                    (new OutputStreamWriter(nextSocket.getOutputStream()));
```

- From then on the server may conduct the tasks needed to process and respond to client requests, or may choose to leave this task for code executing in another thread.
- Remember that just like any other form of I/O operation in Java, code will block indefinitely while reading a response from a client, so to service multiple clients concurrently, threads must be used.
- In simple cases, however, multiple threads of execution may not be necessary, particularly if requests are responded to quickly and take little time to process.
- Creating fully-fledged client/server applications that implement popular Internet protocols involves a fair amount of effort, especially for those new to network programming.
- It also draws on other skills, such as multi-threaded programming, discussed in the next chapter. For now, we will focus on a simple, bare-bones TCP server that executes as a single-threaded application.

3.4.7 Creating a TCP Server

- One of the most enjoyable parts of networking is writing a network server.
- Clients send requests and respond to data sent back, but the server performs most of the real work.

Program 3.6: Program for DaytimeServer.

```
import java.net.*;
import java.io.*;
public class DaytimeServer
{
public static final int SERVICE_PORT = 13;
```

```
public static void main(String args[])
{
try
{
    // Bind to the service port, to grant clients
    // access to the TCP daytime service
    ServerSocket server = new ServerSocket (SERVICE_PORT);
   System.out.println ("Daytime service started");
   // Loop indefinitely, accepting clients
   for (;;)
   {
      // Get the next TCP client
      Socket nextClient = server.accept();
      // Display connection details
      System.out.println ("Received request from "
               + nextClient.getInetAddress() + ":" + nextClient.getPort());
      // Do not read, just write the message
      OutputStream out = nextClient.getOutputStream();
      PrintStream pout = new PrintStream (out);
      // Write the current date out to the user
      pout.print( new java.util.Date() );
      // Flush unsent bytes
      out.flush();
      // Close stream
      out.close();
      // Close the connection
      nextClient.close();
   }
}
catch (BindException t)
{
   System.err.println ("Service already running on port " + SERVICE_PORT);
}
catch (IOException t)
{
   System.err.println ("I/O error - " + t);
}
}
}
```

3.4.7.1 How Daytime Server Works?

- For a server, this is about as simple as it gets. The first step in this server is to create a Server Socket. If this port is already bound, a BindException will be thrown, as no two servers can share the same port. Otherwise, the server socket is created; the next step is to wait for connections.
- Since, daytime is a very simple protocol and our first example of a TCP server should be a simple one; we use here a single-threaded server.
- A for loop that loops indefinitely is commonly used in simple TCP servers, or a while loop whose expression always evaluates to true. Inside this loop, the first line you will find is the server.accept() method, which blocks until a client attempts to connect.
- This method returns a socket that represents the connection to the client. For logging, the IP address and port of the connection is sent to System.out. You will see this every time someone logs in and gets the time of day.
- Daytime is a response-only protocol, so we do not need to worry about reading any input. We obtain an OutputStream and then wrap it in a PrintStream to make it easier to work with.
- Determining the date and time using the java.util. Date class, we send it over the TCP stream to the client. Finally, we flush all data in the print stream and close the connection by calling close() on the socket.

3.4.7.2 Running DaytimeServer

- Running the server is very simple. The server has no command-line parameters.
- To run the server on the local machine, the following command would be used:

```
java Daytime Server
```

Program 3.7: Program to create TCP server and client.

Example of server

```
import java.net.*;
import java.io.*;
public class Server
{
   public static void main(String args[]) throws Exception
   {
      try
      {
         ServerSocket serverSocket = new ServerSocket(4444);
         Socket clientSocket = serverSocket.accept();
         PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
         true); // Second Boolean parameter tells whether to flush
         // after the new line character or not
```

```
            BufferedReader       in       =       new       BufferedReader(new
            InputStreamReader(clientSocket.getInputStream()));
            String inputLine = in.readLine();
            out.println("Server received " + inputLine););
            clientSocket.close();
        }
        Catch (Exception e)
        {
        }
    }
}
```

Example of client:

```
import java.net.*;
import java.io.*;
public class Client
{
    public static void main(String args[]) throws Exception
    {
        int ch;
        Socket socket = Socket("192.0.0.100", 4444);
        InputStream in = socket.getInputStream();
        OutputStream out = socket.getOutputStream();
        /* InputStream and OutputStream performs streaming of bytes */
        String str = "Trying to connect to server";
        Byte buf[] = str.getBytes(); // converts String object to byte array
        out.write(buf);
        while ((ch = in.read()) != -1)
        {  //read() method reads a int value from //stream
           System.out.print((char)ch);
        }
        socket.close();
    }
}
```

3.5 DATAGRAMS - DATAGRAM PACKET, DATAGRAM SERVER AND CLIENT

- The User Datagram Protocol (UDP) is a commonly used transport protocol employed by many types of applications.
- UDP is a connectionless transport protocol, meaning that it does not guarantee either packet delivery or that packets arrive in sequential order.
- Rather than reading from and writing to, an ordered sequence of bytes (using I/O streams), bytes of data are grouped together in discrete packets, which are sent over the network.

- UDP sockets can receive data from more than one host machine. If several machines must be communicated with, then UDP may be more convenient than other mechanisms such as TCP.
- Some network protocols specify UDP as the transport mechanism, requiring its use. Java supports the User Datagram Protocol in the form of two classes:

```
java.net.DatagramPacket
java.net.DatagramSocket
```

3.5.1 DatagramPacket Class

- The DatagramPacket class represents a data packet intended for transmission using the User Datagram Protocol (see Fig. 3.4).
- Packets are containers for a small sequence of bytes and include addressing information such as an IP address and a port.

Fig. 3.4: DatagramPacket representation of a UDP packet

- The meaning of the data stored in a Datagram Packet is determined by its context. When a Datagram Packet has been read from a UDP socket, the IP address of the packet represents the address of the sender, (likewise with the port number).
- However, when a Datagram Packet is used to send a UDP packet, the IP address stored in Datagram Packet represents the address of the recipient, (likewise with the port number). This reversal of meaning is important to remember, one would not want to send a packet back to oneself.

3.5.2 Creating a DatagramPacket

- There are two reasons to create a new DatagramPacket:
  1. To send data to a remote machine using UDP.
  2. To receive data sent by a remote machine using UDP.

- **Constructors:** The choice of which DatagramPacket constructor to use is determined by its intended purpose. Either constructor requires the specification of a byte array, which will be used to store the UDP packet contents and the length of the data packet.
- To create a DatagramPacket for receiving incoming UDP packets, the following constructor should be used:

```
Datagram Packet (byte [] buffer, int length)
```

- For example:

```
DatagramPacket packet = new DatagramPacket (new byte [256], 256);
```

- To send a DatagramPacket to a remote machine, it is preferable to use the following constructor:

```
DatagramPacket (byte [] buffer, int length, InetAddress
                                              dest_addr, int dest_port);
```

- For example:

```
InetAddress addr = InetAddress.getByName ("192.163.0.1");
DatagramPacket packet = new DatagramPacket (new byte [128], 128,
                                                        addr, 2000);
```

3.5.3 Using a DatagramPacket

- The DatagramPacket class provides some important methods that allow the remote address, remote port, data (as a byte array) and length of the packet to be retrieved.
- **Methods:**
  1. `InetAddress getAddress()`: Returns the IP address from which a DatagramPacket was sent, or (if the packet is going to be sent to a remote machine), the destination IP address.
  2. `Byte [] get Data()`: Returns the contents of the DatagramPacket, represented as an array of bytes.
  3. `int getLength()`: Returns the length of the data stored in a DatagramPacket. This can be less than the actual size of the data buffer.
  4. `int getPort()`: Returns the port number from which a DatagramPacket was sent, or (if the packet is going to be sent to a remote machine), the destination port number.
  5. `void setAddress(InetAddress addr)`: Assigns a new destination address to a DatagramPacket.
  6. `void setData(byte[] buffer)`: Assigns a new data buffer to the DatagramPacket. Remember to make the buffer long enough, to prevent data loss.
  7. `void setLength(int length)`: Assigns a new length to the DatagramPacket. Remember that the length must be less than or equal to the maximum size of the

data buffer, or an IllegalArgumentException will be thrown. When sending a smaller amount of data, you can adjust the length to fit—you do not need to resize the data buffer.

8. `void setPort(int port)`: Assigns a new destination port to a DatagramPacket.

3.5.4 DatagramSocket Class

- The DatagramSocket class provides access to a UDP socket, which allows UDP packets to be sent and received.
- A DatagramPacket is used to represent a UDP packet and must be created prior to receiving any packets.
- The same DatagramSocket can be used to receive packets as well as to send them. However, read operations are blocking, meaning that the application will continue to wait until a packet arrives.
- Since, UDP packets do not guarantee delivery; this can cause an application to stall if the sender does not resubmit packets.
- You can use multiple threads of execution; you can use nonblocking I/O to avoid this problem, as shown in the EchoClient example.

3.5.5 Creating a DatagramSocket

- A DatagramSocket can be used to both send and receive packets. Each DatagramSocket binds to a port on the local machine, which is used for addressing packets.
- The port number need not match the port number of the remote machine, but if the application is a UDP server, it will usually choose a specific port number.
- If the DatagramSocket is intended to be a client and does not need to bind to a specific port number, a blank constructor can be specified.
- **Constructors:** To create a client DatagramSocket, the constructor is used: `DatagramSocket() throws java.net.SocketException`. To create a server Datagram Socket, the following constructor is used, which takes as a parameter the port to which the UDP service will be bound: DatagramSocket(int port) throws java.net.SocketException. If a machine is known by several IP addresses (referred to as multihomed), you can specify the IP address and port to which a UDP service should be bound. It takes as parameters the port to which the UDP service will be bound, as well as the InetAddress of the service. This constructor is:

```
DatagramSocket (int port, InetAddress addr) throws
                                        java.net.SocketException
```

3.5.6 Using a DatagramSocket

- DatagramSocket is used to receive incoming UDP packets and to send outgoing UDP packets.
- It provides methods to send and receive packets, as well as to specify a timeout value when nonblocking I/O is being used, to inspect and modify maximum UDP packet sizes and to close the socket.

- **Methods:**
  1. `void close()`: Closes a socket and unbinds it from the local port.
  2. `void connect(InetAddress remote_addr int remote_port)`: Restricts access to the specified remote address and port. The designation is a misnomer, as UDP does not actually create a "connection" between one machine and another. However, if this method is used, it causes exceptions to be thrown if an attempt is made to send packets to, or read packets from, any other host and port than those specified.
  3. `void disconnect()`: Disconnects the DatagramSocket and removes any restrictions imposed on it by an earlier connect operation.
  4. `InetAddress getInetAddress()`: Returns the remote address to which the socket is connected, or null if no such connection exists.
  5. `int getPort()`: Returns the remote port to which the socket is connected, or –1 if no such connection exists.
  6. `InetAddress getLocalAddress()`: Returns the local address to which the socket is bound.
  7. `int getLocalPort()`: Returns the local port to which the socket is bound.
  8. `int getReceiveBufferSize() throws java.net.SocketException`: Returns the maximum buffer size used for incoming UDP packets.
  9. `int getSendBufferSize() throws java.net.SocketException`: Returns the maximum buffer size used for outgoing UDP packets.
  10. `int getSoTimeout() throws java.net.SocketException`: Returns the value of the timeout socket option. This value is used to determine the number of milliseconds a read operation will block before throwing a java.io.InterruptedIOException. By default, this value will be zero, indicating that blocking I/O will be used.
  11. `void receive(DatagramPacket packet) throws java.io.IO-Exception`: Reads a UDP packet and stores the contents in the specified packet. The address and port fields of the packet will be overwritten with the sender address and port fields and the length field of the packet will contain the length of the original packet, which can be less than the size of the packet's byte-array. If a timeout value has not been specified by using DatagramSocket.setSoTimeout(int duration), this method will block indefinitely. If a timeout value has been specified, a java.io.InterruptedIOException will be thrown if the time is exceeded.
  12. `void send(DatagramPacket packet) throws java.io.IOException`: Sends a UDP packet, represented by the specified packet parameter.
  13. `void setReceiveBufferSize(int length) throws java.net. SocketException`: Sets the maximum buffer size used for incoming UDP packets. Whether the specified length will be adhered to is dependent on the operating system.

14. `void setSendBufferSize(int length) throws java.net.Socket-Exception:` Sets the maximum buffer size used for outgoing UDP packets. Whether the specified length will be adhered to is dependent on the operating system.
15. `void setSoTimeout(int duration) throws java.net.Socket Exception:` Sets the value of the timeout socket option. This value is the number of milliseconds a read operation will block before throwing a java.io.InterruptedIOException.

3.5.7 Listening for UDP Packets

- Before an application can read UDP packets sent to it by remote machines, it must bind a socket to a local UDP port using DatagramSocket and create a DatagramPacket that will act as a container for the UDP packet's data.
- When an application wishes to read UDP packets, it calls the DatagramSocket.receive method, which copies a UDP packet into the specified DatagramPacket.
- The contents of the DatagramPacket are processed and the process is repeated as needed.
- The following code snippet illustrates this process:

```
DatagramPacket packet = new DatagramPacket (new byte[256], 256);
DatagramSocket socket = new DatagramSocket(2000);
boolean finished = false;
while (! finished )
{
socket.receive (packet);
// process the packet
}
socket.close();
```

- When processing the packet, the application must work directly with an array of bytes. If, however, your application is better suited to reading text, you can use classes from the Java I/O package to convert between a byte array and another type of stream or reader.

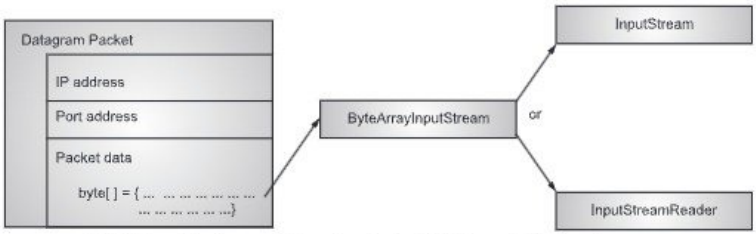


Fig. 3.5: Reading from a UDP packet is simplified by applying input streams

- By hooking a ByteArrayInputStream to the contents of a datagram and then to another type of InputStream or an InputStreamReader, you can access the contents of UDP packets relatively easily, (see Fig. 3.5).
- Many developers prefer to use Java I/O streams to process data, using a DataInputStream or a BufferedReader to access the contents of byte arrays.
- For example, to hook up a DataInputStream to the contents of a DatagramPacket, the following code can be used:

```
ByteArrayInputStream bin = new ByteArrayInputStream(packet.getData());
DataInputStream din = new DataInputStream (bin);
// Read the contents of the UDP packet
    .......
```

3.5.7.1 Sending UDP Packets

- The same interface (DatagramSocket) employed to receive UDP packets are also used to send them.
- When sending a packet, the application must create a DatagramPacket, set the address and port information and write the data intended for transmission to its byte array.
- If replying to a received packet, the address and port information will already be stored and only the data need be overwritten.
- Once, the packet is ready for transmission, the send method of DatagramSocket is invoked and a UDP packet is sent, (see Fig. 3.6).

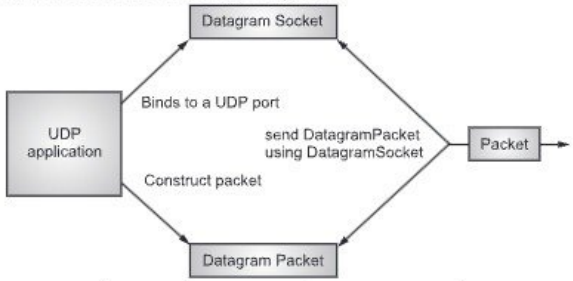


Fig. 3.6: Packets are sent using a DatagramSocket

- The following code illustrates following process:

```
DatagramSocket socket = new DatagramSocket(2000);
DatagramPacket packet = new DatagramPacket (new byte[256], 256);
packet.setAddress (InetAddress.getByName (somehost));
packet.setPort (2000);
boolean finished = false;
```

```
while !finished
{
// Write data to packet buffer
.........
socket.send (packet);
// Do something else, like read other packets, or check to
// see if no more packets to send
.........
}
socket.close();
```

3.5.7.2 Example of User Datagram Protocol

- To demonstrate how UDP packets are sent and received, we will compile and run two small examples.
- The first will bind to a local port, read a packet and display its contents and addressing information. The second example will send the packet read by the first.

Program 3.8: Program for PacketReceiveDemo.

```
import java.net.*;
import java.io.*;
public class PacketReceiveDemo
{
public static void main (String args[])
{
try
{
   System.out.println ("Binding to local port 2000");
   // Create a datagram socket, bound to the
   // specific port 2000
   DatagramSocket socket = new DatagramSocket(2000);
   System.out.println ("Bound to local port " + socket.getLocalPort());
   // Create a datagram packet, containing a
   // maximum buffer of 256 bytes
   DatagramPacket packet = new DatagramPacket( new byte[256], 256 );

   // Receive a packet - remember by default
   //this is a blocking operation
   socket.receive(packet);
   System.out.println ("Packet received!");
```

```
        // Display packet information
        InetAddress remote_addr = packet.getAddress();
        System.out.println ("Sent by: " + remote_addr.getHostAddress());
        System.out.println ("Sent from: " + packet.getPort());
        // Display packet contents, by reading
        // from byte array
        ByteArrayInputStream bin = new ByteArrayInputStream(packet.getData());
        // Display only up to the length of the
        // original UDP packet
        for (int i=0; i < packet.getLength(); i++)
        {
            int data = bin.read();
            if (data == -1)
            break;
            else
            System.out.print ((char)data);
        }
        socket.close();
    }
    catch (IOException t)
    {
        System.err.println ("Error - " +t);
    }
    }
    }
```

3.5.7.3 How PacketReceiveDemo Works?

- Most of the code is self-explanatory or is similar to code snippets shown earlier. However, readers may benefit from a closer examination.
- The application starts by binding to a specific port, 2000. Applications offering a service generally bind to a specific port.
- When acting as a receiver, your application should choose a specific port number, so that a sender can send UDP packets to this port.
- Next, the application prepares a DatagramPacket for storing UDP packets and creates a new buffer for storing packet data.

```
// Create a datagram socket, bound to the specific port 2000
DatagramSocket socket = new DatagramSocket(2000);
```

```
System.out.println ("Bound to local port " + socket.getLocalPort());
// Create a datagram packet, containing a maximum buffer of 256
// bytes
DatagramPacket packet = new DatagramPacket(new byte[256], 256);
```

- Now the application is ready to read a packet. The read operation is blocking, so until a packet arrives, the server will wait.
- When a packet is successfully delivered to the application, the addressing information for the packet is displayed so that it can be determined where it came from.

```
// Receive a packet - remember by default this is a blocking
// operation
socket.receive(packet);
// Display packet information
InetAddress remote_addr = packet.getAddress();
System.out.println ("Sent by: " + remote_addr.getHostAddress());
System.out.println ("Send from: " + packet.getPort());
```

- To provide easy access to the contents of the UDP packet, the application uses a ByteArrayInputStream to read from the packet. Reading one character at a time, the program displays the contents of the packet and then finishes.
- Note that Unicode characters, which are represented by more than just a single byte, cannot be written out in this fashion (readers and writers would be more appropriate if internationalization support is required).

```
// Display packet contents, by reading from byte array
ByteArrayInputStream bin = new ByteArrayInputStream
(packet.getData());
// Display only up to the length of the original UDP packet
for (int i=0; i < packet.getLength(); i++)
{
   int data = bin.read();
   if (data == -1)
   break;
   else
   System.out.print ((char) data);
}
```

Program 3.9: Program for PacketSendDemo.

```
import java.net.*;
import java.io.*;
public class PacketSendDemo
{
public static void main (String args[])
{
int argc = args.length;
// Check for valid number of parameters
if (argc != 1)
{
   System.out.println ("Syntax:");
   System.out.println ("java PacketSendDemo hostname");
   return;
}
String hostname = args[0];
try
{
   System.out.println ("Binding to a local port");
   // Create a datagram socket, bound to any available
   // local port
   DatagramSocket socket = new DatagramSocket();
   System.out.println ("Bound to local port " + socket.getLocalPort());
   // Create a message to send using a UDP packet
   ByteArrayOutputStream bout = new ByteArrayOutputStream();
   PrintStream pout = new PrintStream (bout);
   pout.print ("Greetings!");
   // Get the contents of our message as an array
   // of bytes
   byte[] barray = bout.toByteArray();
   // Create a datagram packet, containing our byte
   // array
   DatagramPacket packet = new DatagramPacket(barray, barray.length);
   System.out.println ("Looking up hostname " + hostname);
   // Lookup the specified hostname and get an
   // InetAddress
```

```
        InetAddress remote_addr = InetAddress.getByName(hostname);
        System.out.println ("Hostname resolved as" +
                                                remote_addr.getHostAddress());
        // Address packet to sender
        packet.setAddress (remote_addr);
        // Set port number to 2000
        packet.setPort (2000);
        // Send the packet - remember no guarantee of delivery
        socket.send(packet);
        System.out.println ("Packet sent!");
    }
    catch (UnknownHostException uhe)
    {
        System.err.println ("Can't find host " + hostname);
    }
    catch (IOException t)
    {
        System.err.println ("Error - " + t);
    }
    }
    }
```

3.5.7.4 How PacketSendDemo Works?

- The second example uses UDP to talk to the first example. This example acts as the sender, dispatching a UDP packet to the receiver, which contains an ASCII text-greeting message.
- Though it uses some similar classes (DatagramSocket, DatagramPacket), they are employed in a slightly different way.
- The application starts by binding a UDP socket to a local port, which will be used to send the data packet. Unlike the receiver demonstration, it does not matter which local port is being used.
- In fact, any free port is a candidate and you may find that running the application several times will result in different port numbers. After binding to a port, the port number is displayed to demonstrate this.

```
// Create a datagram socket, bound to any available local port
DatagramSocket socket = new DatagramSocket();
System.out.println ("Bound to local port " + socket.getLocalPort());
```

- Before sending any data, we need to create a DatagramPacket. First, a ByteArrayOutputStream is used to create a sequence of bytes.

- Once, this is complete, the array of bytes is passed to the DatagramPacket constructor.

```
// Create a message to send using a UDP packet
ByteArrayOutputStream bout = new ByteArrayOutputStream();
PrintStream pout = new PrintStream (bout);
pout.print ("Greetings!");
// Get the contents of our message as an array of bytes
byte[] barray = bout.toByteArray();
// Create a datagram packet, containing our byte array
DatagramPacket packet = new DatagramPacket(barray, barray.length);
```

- Note that the packet has some data it needs to be correctly addressed. As with a postal message, if it lacks correct address information it cannot be delivered. We start by obtaining an InetAddress for the remote machine and then display its IP address. This InetAddress is passed to the setAddress method of DatagramPacket, ensuring that it will arrive at the correct machine. However, we must go one step further and specify a port number. In this case, port 2000 is matched, as the receiver will be bound to that port.

```
System.out.println ("Looking up hostname" +hostname);
// Lookup the specified hostname and get an InetAddress
InetAddress remote_addr = InetAddress.getByName(hostname);
System.out.println ("Hostname resolved as" + remote_addr.getHostAddress());
// Address packet to sender
packet.setAddress (remote_addr);
// Set port number to 2000
packet.setPort (2000);
```

- The final step, after all this work, is to send the packet. This is the easiest step of all—simply invokes the send method of DatagramSocket. Again, remember: there is no guarantee of delivery, so it is possible for a packet to become lost in transit.
- A more robust application would try to read an acknowledgement and resend the message if it had become lost.

```
// Send the packet - remember no guarantee of delivery
    socket.send(packet);
```

3.5.7.5 Running the UDP Examples

- To run these examples, you will need to open two console windows. The first application to be run is the receiver, which will wait for a UDP packet.
- There are no parameters for the receiver, so to run it uses:

```
java PacketReceiveDemo
```

- In a second window, you then need to run the sender. This application could be run from any computer on a local network or the Internet (providing there isn't a firewall between the two hosts).
- If you'd like, you can also run it from the same machine. It takes a single parameter, the hostname of the remote machine:

```
java PacketSendDemo myhostname
```

3.6 URL CONNECTIONS

- Each and every day when we use the Internet to check our mail online, visit a web page or browse an FTP folder, we use our browser. And while there can be a great number of Internet browsers out there, each of them offering different functions and boasting a different design, one thing that unites all of them is the fact that they are built with a single purpose - to handle URLs.
- Uniform Resource Locator (URL) is the global address of documents and other resources on the World Wide Web.
- The Uniform Resource Locator was created in 1994 by Tim Berners-Lee and the URI working group of the Internet Engineering Task Force (IETF).
- A Uniform Resource Locator or Universal Resource Locator (URL) is a specific character string that constitutes a reference to an Internet resource.
- A URL is technically a type of Uniform Resource Identifier (URI).
- **Definition:** A URL is a formatted text string used by Web browsers, e-mail clients and other software to identify a network resource on the Internet. Network resources are files that can be plain Web pages, other text documents, graphics, or programs.
- The java.net.URL class represents a URL. There are constructors to create new URLs and methods to parse the different parts of a URL. However, the heart of the class is the methods that allow you to get an InputStream from a URL so you can read data from a server.
- The URL class is closely tied to protocol and content handlers.
- The objective is to separate the data being downloaded from the protocol used to download it.
- The protocol handler is responsible for communicating with the server that is moving bytes from the server to the client.
- It handles any necessary negotiation with the server and any headers. Its job is to return only the actual bytes of the data or file requested.
- The content handler takes those bytes and translates them into some kind of Java object such as an;

```
InputStream or ImageProducer
```

- When you construct a URL object, Java looks for a protocol handler that understands the protocol part of the URL such as "http" or "mailto". If no such handler is found, the constructor throws a MalformedURLException.
- The exact protocols that are supported vary from implementation to implementations though http and file are supported pretty much everywhere.

3.6.1 URL Format

- Each file available on the World Wide Web can be identified and accessed through its corresponding URL.
- Standing for Uniform Resource Locator, a URL represents the global web address of documents, including web pages or image files, and programs such as CGI applications or Java applets.
- Its main mission is to identify the location of a document or a program available on the web and specify the mechanism for accessing it through a web browser.
- URL strings consist of three parts (substrings):
  1. Network protocol.
  2. Host name or address.
  3. File or Resource location.
- These substrings are separated by special characters as follows:

  `protocol:// host / location`

  1. **URL Protocol:** The 'protocol' substring defines a network protocol to be used to access a resource. These strings are short names followed by the three characters '://' (a simple naming convention to denote a protocol definition). Typical URL protocols include http://, ftp://, and mailto://.
  2. **URL Host:** The 'host' substring identifies a computer or other network device. Hosts come from standard Internet databases such as DNS and can be names or IP addresses. For example, `computernetworkandcommunication.about.com` is the host for this Web page.
  3. **URL Location:** The 'location' substring contains a path to one specific network resource on the host. Resources are normally located in a host directory or folder. For example, `/pragati/Engineering/bldef-url.htm` is the location of this Web page including two subdirectories and the file name.
- When the location element is omitted such as in `http://computernetworkandcommunication.about.com/`, the URL conventionally points to the root directory of the host and often a home page (like 'home.htm').
- There are four constructors in the `java.net.URL` class. All can throw MalformedURLExceptions.
  1. `public URL(String u) throws MalformedURLException`
  2. `public URL(String protocol, String host, String file)`
     `throws MalformedURLException`
  3. `public URL(String protocol, String host, int port, String file)`
     `throws MalformedURLException`
  4. `public URL(URL context, String u) throws MalformedURLException`

1. Given a complete absolute URL like http://www.VIT.edu/schedule/timeTable.html, you construct a URL object for that URL like this:

```
URL u = null;
  try
  {
  u = new URL("http://www.VIT.edu/schedule/timeTable.html");
  }
  catch (MalformedURLException ex)
  {
  }
```

2. You can also construct the URL by passing its pieces to the constructor, like this:

```
URL u = null;
try {
  u = new URL("http", "www.VIT.edu", "/schedule/timeTable.html");
}
catch (MalformedURLException ex)
{
}
```

3. You do not normally need to specify a port for a URL. Most protocols have default ports. For instance, the http port is 80; but sometimes this does change and in that case you can use the third constructor:

```
URL u = null;
  try {
    u   = new URL("http", "www.VIT.edu", 80, "/schedule/timeTable.html");
  }
  catch (MalformedURLException ex) {
  }
```

   Many HTML files contain relative URLs.

4. The fourth constructor above creates URLs relative to a given URL. For example,

```
try {
  URL ul = new
  URL("http://www.Nirali.org/course/day3/03.html");
  URL u2 = new URL(ul, "09.html");
  }
catch (MalformedURLException ex) {
  }
```

- The java.net.URL class has five methods to split a URL into its component parts. These are:

```
public String getProtocol()
public String getHost()
public int   getPort()
public String getFile()
public String getRef()
```

- For example,

```
try {
    URL u = new URL("http://www.VIT.edu/schedule/timeTable.html");
    System.out.println("The protocol is " + u.getProtocol());
    System.out.println("The host is " + u.getHost());
    System.out.println("The port is " + u.getPort());
    System.out.println("The file is " + u.getFile());
    System.out.println("The anchor is " + u.getRef());
}
catch (MalformedURLException ex) {
}
```

- If a port is not explicitly specified in the URL, it's set to -1. This does not mean that the connection is attempted on port -1 (which does not exist) but rather that the default port is to be used.
- If the ref does not exist, it's just null, so watch out for Null PointerExceptions.

3.6.2 URLConnection

- URLConnection is the general purpose class for accessing the attributes of a remote resource.
- The abstract class `URLConnection` is the superclass of all classes that represent a communications link between the application and a URL.
- Instances of this class can be used both to read from and to write to the resource referenced by the URL.
- After you have successfully created a `URL` object, you can call the `URL` object's `openConnection` method to get `URLConnection` Object.
- For example, the following code opens a connection to the site pragati.com:

```
try {
    URL myURL = new URL("http://pragati.com");
    URLConnection myURLConnection = myURL.openConnection();
    myURLConnection.connect();
} catch (MalformedURLException e) {     // new URL() failed
    ..............
} catch (IOException e) { // openConnection() failed
    ..............
}
```

- A new `URLConnection` object is created every time by calling the `OpenConnection` method of the protocol handler for this URL.
- You are not always required to explicitly call the `connect` method to initiate the connection. Now that you have successfully connected to your URL, you can use the `URLConnection` object to perform actions such as reading from or writing to the connection.

- **Example:** In the following example, we create a `URLConnection` using the `openConnection()`, method of a URL object and then use it to examine the document's properties and content.

Program 3.10: Program for URLConnection.

```
import java.net.*;
import java.io.*;
import java.util.Date;
class UCDemo
{
   public static void main(String args[]) throws Exception
   {
   int c;
   URL hp = new URL("http://www.Nirali.com/java/");
   URLConnection hpCon = hp.openConnection();
   System.out.println("Date:" + new Date(hpCon.getDate()));
   System.out.println("Content-Type:" + hpCon.getContentType());
   System.out.println("Expires: " + hpCon.getExpiration());
   System.out.println("Last_Modified:" + new Date(hpCon.getLastModified()));
   int len = hpCon.getContentLength();
   System.out.println("Content-Length:" + len);
   if (len > 0)
   {
   System.out.println("=== Content of Books ===");
   InputStream input = hpCon.getinputStream();
   int i = len;
   while (((c = input.read()) != -1) && (-i > 0))
   {
   System.out.print((char) c);
   }
   input.close();
   } else {
   System.out.println("No Content of Books Available");
   }
   }
   }
```

SUMMARY

- Most networking java programs communicate using sockets.
- Connection oriented sockets make use of TCP and connection less sockets use UDP.
- Port number is a 16-digit number and so ranges from 0 to 65535. The port numbers ranging from 0 - 1023 are restricted.

- The IP address is a four-byte (32-bit) address, which is usually expressed in dotted decimal format (For example, 192.163.0.6). Although a physical address will normally be issued to a machine, it will not be very useful as the machine goes onto a network.
  - The Transmission Control Protocol (TCP) is a Layer 4 protocol, (transport layer) that provides guaranteed delivery and ordering of bytes.
  - TCP uses the Internet Protocol to send TCP segments, which contain additional information that allows it to order packets and resend them if they are lost.
  - User Datagram Protocol (UDP) is a Layer 4 protocol, (transport layer) that applications can use to send packets of data across the Internet, (as opposed to TCP, which sends a sequence of bytes).
- A proxy server is software that runs on server that speaks the language of clients. This software hides the actual server from clients that communicate with it.
- An internet address is the number that uniquely indentifies each computer on the internet just like any other network.
- Internet addresses consist of 32 bit values, organized into 4 groups of 8 bits each separated by dot[.].
- In java such addresses are represented by the **java.net.InetAddress** class.
- The package **java.net** provides different classes for implementing networking applications.
  - Using the static method InetAddress.getLocalHost(), we obtain an object representing an IP address.
  - To display the address in dotted decimal notation, the InetAddress.getHostAddress() method is used.
- Java offers good support for TCP sockets, in the form of two socket classes, `java.net.Socket and java.net.ServerSocket.`
- The User Datagram Protocol (UDP) is a commonly used transport protocol employed by many types of applications.
- UDP is a connectionless transport protocol, meaning that it does not guarantee either packet delivery or that packets arrive in sequential order.
- Some network protocols specify UDP as the transport mechanism, requiring its use. Java supports the User Datagram Protocol in the form of two classes:
  - java.net.DatagramPacket
  - java.net.DatagramSocket
- The DatagramPacket class provides some important methods that allow the remote address, remote port, data (as a byte array) and length of the packet to be retrieved.

Check Your Understanding

1. Which of these package contains classes and interfaces for networking?
   (a) java.io (b) java.util
   (c) java.net (d) javax.swing
2. Which of the following protocol follows connection less service?
   (a) TCP (b) TCP/IP
   (c) UDP (d) HTTP

3. Which of the following statement is TRUE?
   (a) With stream sockets there is no need to establish any connection and data flows between the processes are as continuous streams.
   (b) Stream sockets are said to provide a connection-less service and UDP protocol is used
   (c) Datagram sockets are said to provide a connection-oriented service and TCP protocol is used
   (d) With datagram sockets there is no need to establish any connection and data flows between the processes are as packets.
4. The server listens for a connection request from a client using which of the following statement?
   (a) Socket s = new Socket(ServerName, port);
   (b) Socket s = serverSocket.accept()
   (c) Socket s = serverSocket.getSocket()
   (d) Socket s = new Socket(ServerName);
5. What will be the output of the following Java program?

```
import java.net.*;
class networking
{
    public static void main(String[] args) throws UnknownHostException
    {
        InetAddress obj1 = InetAddress.getByName("sanfoundry.com");
        InetAddress obj2 = InetAddress.getByName("sanfoundry.com");
        boolean x = obj1.equals(obj2);
        System.out.print(x);
    }
}
```

   (a) 0 (b) 1
   (c) true (d) false
6. Which class is used to create servers that listen for either local client or remote client programs?
   (a) ServerSockets (b) httpServer
   (c) httpResponse (d) None of the above
7. The client requests a connection to a server using which of the following statement?
   (a) Socket s = new Socket(ServerName, port);
   (b) Socket s = serverSocket.accept();
   (c) Socket s = serverSocket.getSocket();
   (d) Socket s = new Socket(ServerName);

8. Which method of URL class represents a URL and it has complete set of methods to manipulate URL in Java?
   (a) java.net.URL (b) java.net.URLConnection
   (c) Both (a) and (b) (d) None of the above
9. Which of these class is used to encapsulate IP address and DNS?
   (a) DatagramPacket (b) URL
   (c) InetAddress (d) ContentHandler
10. In the socket programming, for an IP address, which can be used to find the host name and IP address of a client/ server?
    (a) The ServerSocket class (b) The Socket class
    (c) The InetAddress class (d) The Connection interface

Answers

1. (c)	2. (c)	3. (d)	4. (b)	5. (c)	6. (a)	7. (a)	8. (a)	9. (c)	10. (c)

Practice Questions

Q.I Answer the following questions in short:

1. What is Networking?
2. What are the advantages of networking?
3. What is Protocol? List out at least two protocols.
4. What is Internet?
5. Define the following terms:
6. Explain use of Socket class.
7. What is the difference between TCP/IP and UDP?
8. Explain the use of ServerSocket class.
9. Explain the use of DatagramSocket class.
10. Define protocol list difference between TCP and UDP.
11. What is Internet Addressing? Give the purpose of InetAddress class?
12. Describe the use of following:
    (i) Reserved sockets
    (ii) DNS (Domain Naming Service)
13. Explain the use of following methods of DatagramPacket class with example:
    (i) getPort()
    (ii) getData()
14. What is proxy server? Give the purpose of proxy server.

Q.II Answer the following questions:

1. Explain with suitable diagram the HTTP communication between Web server and web client.
2. What is Client Server Model?
3. Write short note on URL Programming.
4. Explain TCP/IP client sockets with suitable example.
5. Explain TCP/IP server sockets with suitable example.
6. Explain UDP Server sockets with suitable example.

7. Explain UDP Client sockets with suitable example.
8. Write a program that demonstrates TCP/IP based communication between client and server.
9. Write a program that demonstrates UDP based communication between client and server.
10. Write a program to demonstrate use of URL and URL Connection class for communication.
11. Write a program to retrieve and display port number and host name of URL.
12. What is client server model? Explain use of socket and server Socket class.
13. Write the use of InetAddress Class with suitable example.
14. Write a program to retrieve and display host name and port number using InetAddress Class.
15. Write the sequential steps for establishing a connection between client socket and server socket using TCP protocol.
16. State four main difference between TCP and UDP protocols.
17. Give the use of URL class along with syntax of constructor of URL class.
18. What is IP address and write names of classes that return IP address.
19. WAP to display protocol, hostname, Port number and file name of an URL. "www.msbte.com/main".
20. What is the use of DatagramPacket? Explain creation of sockets in case of UDP.
21. What is URL? Explain use of the URL connection class with its 4 methods.
22. What is Internet Addressing? Explain any four methods of InetAddress class.

Q.III Define Term:

1. Web Browser
2. Web server
3. Protocol
4. HTTP
5. Home Page
6. Fire wall
7. Proxy server
8. PORT
9. URL

■■■

4...

Servlet and JSP

Learning Objectives ...

Students will be able:

- To understand Concepts of Servlet.
- To learn Session Tracking using Servlet.
- To learn Session Tracking using Servlet.
- To study JSP Concepts.
- To Learn Life Cycle of Servlet and JSP.
- To study Components of JSP.

4.1 INTRODUCTION TO SERVLET

- A Servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP (HyperText Transfer Protocol).
- Java Servlets are programs that run on a web or application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
- Using Servlets, we can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- A user enters a Uniform Resource Locator (URL) into a browser. The browser generates an HTTP request to the appropriate web server. The web server maps this request to a specific file. That file is returned in an HTTP response to the browser. The HTTP header in the response indicates the type of the content.
- The Multipurpose Internet Mail Extensions (MIME) are used for this purpose. For example, ordinary ASCII text has a MIME type of text/plain. The Hypertext Markup Language (HTML) source code of a web page has a MIME type of text/html.
- Web applications are helper application that resides at web server and build dynamic web pages.
- A dynamic page could be anything like a page that randomly chooses picture to display or even a page that display current time.

- When a user issues a request for a URL that corresponds to a java servlets, the server hands the request off to the Servlet (program on server side) for processing.
- The Servlet dynamically produces a response to the request, typically an HTML web page and sends it back to the requesting web browser.
- Servlets provide a component-based, platform-independent method for building web based applications. Servlets provides an object-oriented and extensible middle tier for Web server based applications.
- Web servers generally cannot talk to databases – a web server alone cannot create web page content using data held in database.
- This information cannot make accessible to public through the web server. Here, Servlets helps to extend the functionality of a web server.
- A dynamic page could be anything like a page that randomly chooses picture to display or even a page that displays the current time.

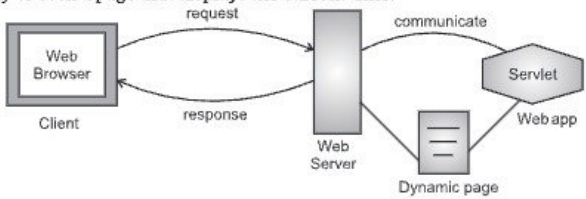


Fig. 4.1

- As Servlet Technology uses Java, web applications made using Servlet are secured, scalable and robust.

4.1.1 What is Servlet?

- A Java Servlet is a server side program that services HTTP requests and returns the result as HTTP responses.
- A servlet:
  - is a program written using java that runs in a server application to answer clients requests.
  - is supported by virtually all web servers and application servers.
  - solves the performance problem by executing all requests as thread in one process.
  - is a technology i.e. used to create web application.
  - is an API that provides many interfaces and classes including documentations.

4.1.2 Advantages of using Servlets

1. Less response time because each request runs in a separate thread.
2. Servlets are scalable and portable.
3. Servlets are robust and object oriented.

4. Servlets are platform independent.
5. Servlet technology is very secured because of it uses java language.
6. Servlets are managed by the Java Virtual Machine (JVM). As such, you do not need to worry about memory leak or garbage collection, which helps you write robust applications.

- Servlet API consists of two important packages that encapsulate all the important classes and interface, namely javax.servlet and javax.servlet.http.

4.1.3 How a Servlet Works?

- A Servlet is a java program that runs on web server. This program will start running when there is request from client side. A servlet is loaded by the servlet container the first time the Servlet is requested.
- When client sends a request to server, the user request is given to server, the server run Servlet and gives user request to servlet.
- Then Servlet perform operation and generate result and returns the result as response to the server, which in turn sends the response back to the user.
- After that, the Servlet stays in memory waiting for other requests - it will not be unloaded from the memory unless the Servlet container sees as hostage of memory.
- Each time the Servlet is requested, however, the servlet container compares the timestamp of the loaded Servlet with the Servlet class file. If the class file timestamp is more recent, the Servlet is reloaded into memory.
- This way, we do not need to restart the servlet container every time you update your servlet.
- Fig. 4.2 shows working of Servlet.

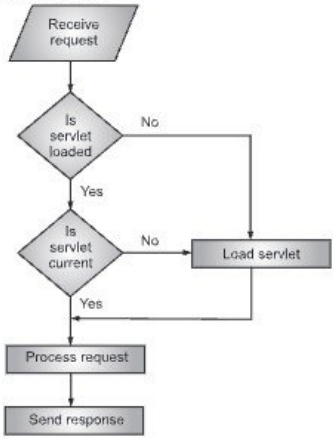


Fig. 4.2: Working of Servlet

4.1.4 Hierarchy of Servlet

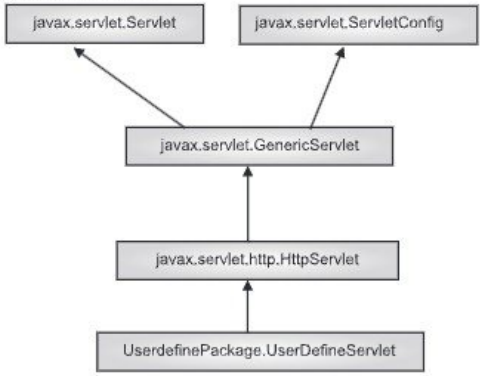


Fig. 4.3

- The root interface of the servlet class is Servlet interface hierarchy. All Servlets need to either directly or indirectly implement the Servlet interface.
- The GenericServlet class of the Servlet API implements the Servlet interface. In addition to the Servlet interface, the GenericServlet class implements the ServletConfig interface of the Servlet API and the Serializable interface of the standard java.io.package.
- The object of the ServletConfig interface is used by the Web container to pass the configuration information to a servlet when a Servlet is initialized.
- To develop a servlet that communicates using HTTP, we need to extend the HttpServlet class in our servlet. The HttpServlet class extends the GenericServlet class and provides built-in HTTP functionality.
- **javax.servlet.Servlet** Interface contains following methods:
  1. public void destroy()
  2. public ServletConfig getServletConfig()
  3. public String getServletInfo()
  4. public String getServletInfo()
- **javax.servlet.ServletConfig** Interface contains following methods:
  1. public String getInitParameter(String param)
  2. public Enumeration getInitParameterNames()
  3. public ServletContext getServletContext()

4.2 TYPES OF SERVLET

- There are two types of servlets: GenericServlet and HttpServlet. GenericServlet defines the generic or protocol independent servlet. HttpServlet is subclass of GenericServlet and provides some http specific functionality like doGet and doPost methods.

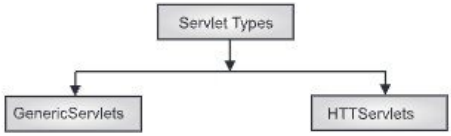


Fig. 4.4

1. **Generic Servlets:** It extends javax.servlet.GenericServlet. Generic Servlets are protocol independent. They contain no inherent HTTP support or any other transport protocol.
2. **HTTP Servlets:** It extends javax.servlet.HttpServlet. They have built-in HTTP protocol support and are more useful in a Sun Java System Web Server environment.

- For both Servlet types, you implement the constructor method init() and the destructor method destroy() to initialize or deallocate resources.
- All Servlets must implement a service() method, which is responsible for handling Servlet requests.
- For generic Servlets, simply override the service method to provide routines for handling requests.
- HTTP Servlets provide a service method that automatically routes the request to another method in the Servlet based on which HTTP transfer method is used.
- So, for HTTP servlets, override doPost() to process POST requests, doGet() to process GET requests, and so on.

1. Generic Servlets:

GenericServlet Class:

- GenericServlet is an abstract class that provides implementation of most of the basic servlet methods.

Methods of GenericServlet Class:

- `public void init(ServletConfig)`
- `public abstract void service(ServletRequest request, ServletResposne response)`
- `public void destroy()`
- `public ServletConfig getServletConfig()`
- `public String getServletInfo()`

- `public ServletContext getServletContext()`
- `public String getInitParameter(String name)`
- `public Enumeration getInitParameterNames()`
- `public String getServletName()`
- `public void log(String msg)`
- `public void log(String msg, Throwable t)`

2. HTTP Servlets:

- It extends `javax.servlet.HttpServlet`.
- It has built-in HTTP protocol support and are more useful in a Sun Java System Web Server environment.
- HTTP servlets provide a service method that automatically routes the request to another method in the servlet based on which HTTP transfer method is used.
- HTTP servlets, override `doPost()` to process POST requests, `doGet()` to process GET requests.

HttpServlet Class:

- HttpServlet is also an abstract class. This class gives implementation of various service() methods of Servlet interface.
- To create a servlet, we should create a class that extends HttpServlet abstract class. The Servlet class that we will create, must not override service() method. Our servlet class will override only the doGet() and/or doPost() methods.
- The service() method of HttpServlet class listens to the Http methods (GET, POST etc.) from request stream and invokes doGet() or doPost() methods based on Http Method type.

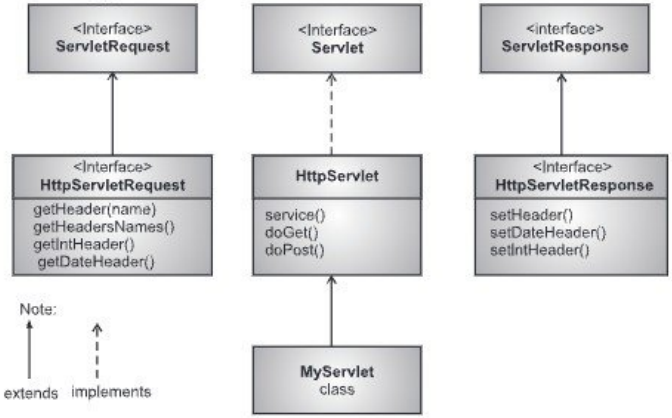


Fig. 4.5

- GenericServlet is protocol independent implementation of Servlet interface whereas HttpServlet is HTTP protocol specific implementation.
- Most of the times we use servlet for creating web application and that's why we extend HttpServlet class.
- HttpServlet class extends GenericServlet and also provides some other methods specific to HTTP protocol.

4.3 LIFE CYCLE OF SERVLET

- It can be defined as, the entire process from its creation till the destruction. The following are the paths followed by a servlet.
  - The servlet is initialized by calling the **init()** method.
  - The servlet calls **service()** method to process a client's request.
  - The servlet is terminated by calling the **destroy()** method.
  - Finally, servlet is garbage collected by the garbage collector of the JVM.

1. **init() Method:**
   - The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.
   - The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.
   - When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.
   - The init method definition looks like this:

```
public void init() throws ServletException
{
 // Initialization code...
}
```

2. **service() Method:**
   - The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client (browsers) and to write the formatted response back to the client.
   - Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

- Here, is the **signature/syntax** of this method:

```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException
{
}
```

- The service() method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate.

(i) doGet() Method:

- A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
   // Servlet code...
}
```

(ii) doPost() Method:

- A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
   // Servlet code...
}
```

3. destroy() Method:

- The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.
- After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

```
public void destroy()
{
   // Finalization code...
}
```

- The Fig. 4.6 depicts a typical servlet life cycle scenario.
  - First the HTTP requests coming to the server are delegated to the servlet container.

- The servlet container loads the servlet before invoking the service() method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.

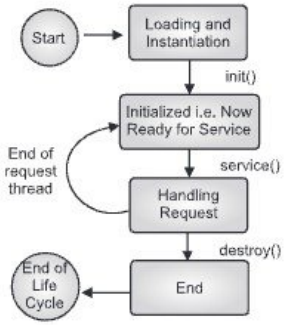


Fig. 4.6: Life cycle of Servlet

- Service method dispatches the requests such as DELETE, GET, HEAD, OPTIONS, POST, PUT, and TRACE.
- A GET request is dispatched to the doGet(HttpServletRequest request, HttpServletResponse response) method.
- A POST request is dispatched to the doPost(HttpServletRequest request, HttpServletResponse response) method. These are the two methods you will usually override.
- doGet and doPost typically do the same thing, so usually you do the real work in one, and have the other just call it.

```
public void doGet(HttpServletRequest request, HttpServletResponse
response)
{
   doPost(request, response);
}
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
{
```

- This method services a GET request.

Classes and Interfaces of javax.servlet:

Interfaces	Classes
Servlet	ServletInputStream
ServletContext	ServletOutputStream
ServletConfig	ServletRequestWrapper
ServletRequest	ServletResponseWrapper
ServletResponse	ServletRequestEvent
ServletContextListener	ServletContextEvent
RequestDispatcher	ServletRequestAttributeEvent
SingleThreadModel	ServletContextAttributeEvent
Filter	ServletException
FilterConfig	UnavailableException
FilterChain	GenericServlet

Classes and Interface of javax.servlet.http:

Classes	Interfaces
HttpServlet	HttpServletRequest
HttpServletResponse	HttpSessionAttributeListener
HttpSession	HttpSessionListener
Cookie	HttpSessionEvent

4.4 CREATING SERVLET

- Servlets are Java classes which service HTTP requests and implement the javax.servlet.Servlet interface.
- Web application developers typically write Servlets that extend javax.servlet.http.HttpServlet, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.
- To create a Servlet application you need to follow the below mentioned steps. These steps are common for the entire Web server. In our example we are using Apache Tomcat server.
- Apache Tomcat is an open source web server for testing Servlets and JSP technology. Download latest version of Tomcat Server and install it on your machine using link https://tomcat.apache.org/download-80.cgi

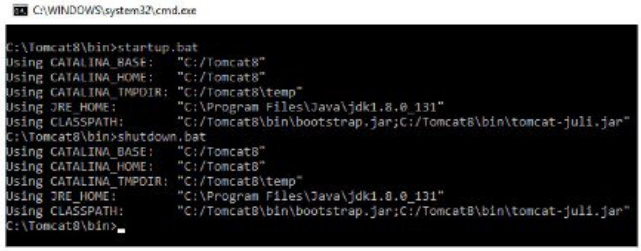
Steps of Tomcat Configuration:

Step 1 : Download and Setup JAVA variables globally.

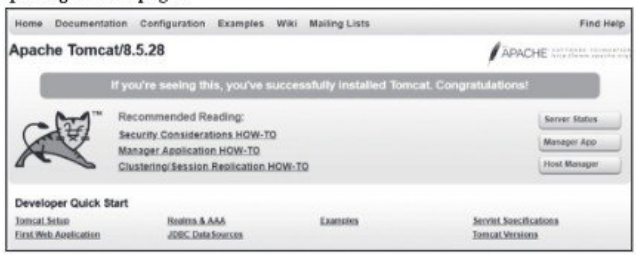
- Setup JAVA_HOME variable path as C:\Program Files\Java\jdk1.8.0_131 path may change in your computer.
- Or setup JRE_HOME variable path as C:\Program Files\Java\jre1.8.0_131 path may change in your computer.
- Setup CATALINA_HOME variable path as C:\Tomcat8 which is my Tomcat installation path.

Step 2 : After successful installation, go to BIN folder directly under Tomcat folder. You will find two batch files with names **startup.bat** and **shutdown.bat**.

- C:\Tomcat8\bin>startup.bat
- C:\Tomcat8\bin>shutdown.bat



Step 3 : Go to 'http://localhost:8080'. You should see the below page. It means you successfully installed Tomcat Server. You should start the server before opening the webpage.



Step 4 : Go to Tomcat/conf/tomcat-users.xml

Add Usernames, Passwords and Roles required. Without configuring these Usernames, You will be able to access Server Status, Manager App and Host Manager options on the Tomcat Home Page.

- <role rolename="manager-gui"/>
- <user username="admin" password="admin" roles="manager-gui"/>
- <role rolename="admin-gui"/>
- <user username="tomcat" password="tomcat" roles="admin-gui"/>

Sample Code for Hello World:

- Following is the sample source code structure of a Servlet example to write Hello World:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class HelloWorld extends HttpServlet
{
   private String message;
   public void init() throws ServletException
   {
      // Do required initialization
      message = "Hello World 2022...";
   }
   public void doGet(HttpServletRequest request, HttpServletResponse
                           response) throws ServletException, IOException
   {
      // Set response content type
      response.setContentType("text/html");
      // Actual logic goes here.
      PrintWriter out = response.getWriter();
      out.println("<h1>" + message + "</h1>");
   }
   public void destroy()
   {
      // do nothing.
   }
}
```

Compiling the Servlet:

- We will begin by building and testing a simple servlet. The basic steps are the following:

1. Create and compile the servlet source code. Then, copy the servlet's class file to the proper directory, and add the servlet's name and mappings to the proper **web.xml** file.
2. Start Tomcat.
3. Start a web browser and request the servlet.

- Let us save the code in Helloservlet.java file and put this file in /root/apache-tomcat-5.5.25/webapps/servlets-examples/WEB-INF/classes
- Assuming the environment is setup properly, go in ServletDemo directory and compile Helloservlet.java.
- If the Servlet depends on any other libraries, we have to include those JAR files on our CLASSPATH as well. We have included only servlet-api.jar JAR file because we are not using any other library in Helloservlet program.
- This command line uses the built-in javac compiler that comes with the Sun Microsystems Java Software Development Kit (JDK). For this command to work properly, we have to include the location of the Java SDK that we are using in the PATH environment variable.
- If everything goes fine, above compilation would produce Helloservlet.class file in the same directory. Next section would explain how a compiled servlet would be deployed in production.

Servlet Deployment:

- Create following entries in web.xml file located in /root/apache-tomcat-5.5.25/webapps/servlets-examples/web.xml.

```
<servlet>
    <servlet-name> Helloservlet </servlet-name>
    <servlet-class> Helloservlet </servlet-class>
</servlet>
    <servlet-mapping>
    <servlet-name> Helloservlet </servlet-name>
    <url-pattern>/servlet/Helloservlet </url-pattern>
</servlet-mapping>
```

- Above entries to be created inside <web-app>...</web-app> tags available in web.xml file. There could be various entries in this table already available, but never mind.
- We are almost done, now let us start tomcat server using:

```
[root@localhost bin]# sh startup.sh
Using CATALINA_BASE:       /root/apache-tomcat-5.5.25/
Using CATALINA_HOME:       /root/apache-tomcat-5.5.25/
Using CATALINA_TMPDIR:     /root/apache-tomcat-5.5.25//temp
Using JRE_HOME:            /root/jdk1.5.0_05/
[root@localhost bin]# sh shutdown.sh
Using CATALINA_BASE:       /root/apache-tomcat-5.5.25/
Using CATALINA_HOME:       /root/apache-tomcat-5.5.25/
Using CATALINA_TMPDIR:     /root/apache-tomcat-5.5.25//temp
Using JRE_HOME:            /root/jdk1.5.0_05/
```

- If everything goes fine, we would get following output:

- A number of Web servers that support Servlets are available in the market. Some web servers are freely downloadable and Tomcat is one of them.
- Apache Tomcat is an open source software implementation of the Java Servlet and JavaServer Pages technologies and can act as a standalone server for testing servlets and can be integrated with the Apache Web Server.

4.5 HANDLING GET AND POST REQUESTS (HTTP)

- HTTP (HyperText Transfer Protocol) is a protocol that clients and servers use on the web to communicate.
- It is similar to other internet protocols such as SMTP (Simple Mail Transfer Protocol) and FTP (File Transfer Protocol) but there is one fundamental difference. HTTP is a stateless protocol i.e. HTTP supports only one request per connection.
- This means that with HTTP the clients connect to the server to send one request and then disconnects. This mechanism allows more users to connect to a given server over a period of time.
- The client sends an HTTP request and the server answers (responses) with an HTML page to the client, using HTTP, (See Fig. 4.7).

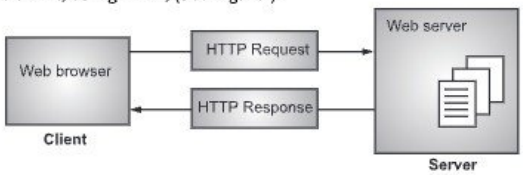


Fig. 4.7: HTML Page to the Client, using HTTP

- Fig. 4.8 diagram shows the position of Servelts in a Web Application.

HTTP Methods:

- HTTP request can be made using a variety of methods, but most often used are GET and POST.
- The method name tells the server the kind of request that is being made, and how the rest of the message will be formatted.

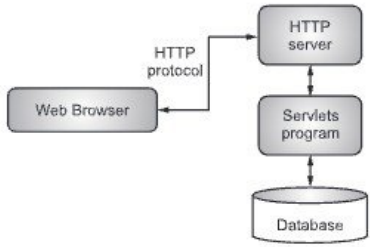


Fig. 4.8: Servlet in a Web Application

HTTP Methods:

1. OPTIONS: Request for communication options that are available on the request/ response chain.
2. GET: Request to retrieve information from server using a given URI (Uniform Resource Identifier).
3. HEAD: Identical to GET except that it does not return a message-body, only the headers and status line.
4. POST: Request for server to accept the entity enclosed in the body of HTTP method.
5. DELETE: Request for the Server to delete the resource.
6. CONNECT: Reserved for use with a proxy that can switch to being a tunnel.
7. PUT: This is same as POST, but POST is used to create, PUT can be used to create as well as update. It replaces all current representations of the target resource with the uploaded content.

1. GET Method:

- The GET method sends the encoded user information appended to the page request.
- The page and the encoded information are separated by the ? character as follows:

```
http://www.test.com/hello?key1=value1&key2=value2
```

- The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box.
- Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be in a request string.
- This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using doGet() method.
- The syntax of doGet is as follows:

```
public void doGet(HttpServletRequest req, HttpServletRequest resp)
throws IOException, ServletException
{ ..................................... }
```

2. Post Method:

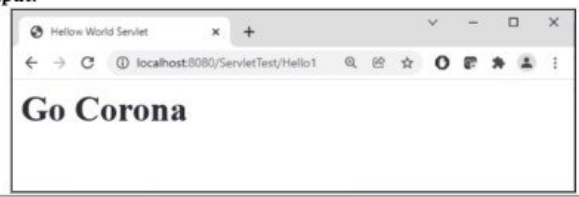
- A generally more reliable method of passing information to a backend program is the POST method.
- This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL it sends it as a separate message.
- This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using doPost() method.
- The syntax of doPost is as given below:

```
public void doPost (HttpServlet-Request req, HttpServletResponse resp)
throws ServletException, IOException
{ ..................................... }
```

Program 4.1: Program to use of doGet and display Hello Computer Servlet.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Hello1 extends HttpServlet
{
   public void doGet(HttpServletRequest req,
             HttpServletResponse resp) throws ServletException,IOException
   {
      PrintWriter out;
      //set the content type
      resp.setContentType("text/html");
      out = resp.getWriter();
      out.println("<HTML><HEAD><TITLE>");
      out.println("Hellow World Servlet");
      out.println("</TITLE></HEAD><BODY>");
      out.println("<H1>Go Corona</H1>");
      out.println("<BODY></HTML>");
      out.close();
   }
}
```

Output:



Difference between GET and POST Requests:

Sr. No.	GET Request	POST Request
1.	In case of Get request, only limited amount of data can be sent because data is sent in header.	In case of post request, large amount of data can be sent because data is sent in body.
2.	Get request is not secured because data is exposed in URL bar.	Post request is secured because data is not exposed in URL bar.
3.	Get request can be bookmarked.	Post request cannot be bookmarked.
4.	Get request is idempotent. It means second request will be ignored until response of first request is delivered.	Post request is non-idempotent.
5.	Get request is more efficient and used more than Post.	Post request is less efficient and used less than Get.

4.5.1 javax.servlet Package

- This contains various classes and interfaces which establish the framework in which servlet operate.

Interface	Description
`Servlet`	Declares life cycle methods for a Servlet.
`ServletConfig`	Allows Servlet to get initialization parameter.
`ServletRequest`	Used to read data from a client request.
`ServletReponse`	Used to write data to a client response.
`SingleThreadModel`	Indicates that Servlet is threadsafe.
`GenericServlet`	Implements Servlet and servletConfig interfaces.
`ServletInputStream`	Stream for reading request from a client.
`ServletOutputStream`	Stream for writing responses to a client.
`ServletException`	Indicates a Servlet error occurred.
`UnavailableException`	Indicates a Servlet is unavailable.

4.5.2 Servlet Interface

- It has three methods init(), service() and destroy(). These methods are called by the server during the life cycle of a Servlet.

- The list of methods in this interface is given below:

Method	Description
`destroy()`	Called at the time of unloading Servlet.
`ServletConfig getServlet Config()`	Returns ServletConfig object that contains any initialization parameters.
`getServletInfo()`	Returns a string describing the Servlet.
`init(ServletConfig sc())`	It is called at the initialization of Servlet.
`services(SevletRequest req, ServletResponse res)`	It is called to process a request from a client. It reads requests and sends response back.

4.5.3 ServletConfig Interface

- This interface allows a servlet to obtain configuration data when loaded.
- The following methods are used in this interface:

Method	Description
`ServletContext getServletContext()`	Returns a context for the Servlet.
`getInitParameter(String param)`	Returns value of initialization.
`getServletName()`	Returns name of the invoking Servlets.

4.5.4 ServletRequest Interface

- It allows getting the information about client request.
- The following methods are used in this interface:

Method	Description
`getAttribute(String attr)`	Returns value of the attribute.
`getContentLength()`	Returns size of the request. – 1 is returned if size is unavailable.
`getContentType()`	Returns type of request. null if no type available.
`getProtocol()`	Returns description of the protocol.
`ServletInputStreamgetInputStream()`	Returns a stream used to read data.
`getServerName()`	Returns name of server.
`getServerPort()`	Returns port number.
`getRemoteAddr()`	Returns string equivalent to client IP address.
`getRemoteHost()`	Return string equivalent to client host name.
`getScheme()`	Returns transmission scheme of URL e.g. http, ftp etc.

4.5.5 ServletResponse Interface

- This is used to return response back to the client.
- The methods of this interface are listed below:

Method	Description
`getCharacterEncoding()`	Returns character encoding for the response.
`ServletOutputStream getOutputStream()`	Returns a stream used to write binary data to the response.
`PrintWriter getWriter()`	Returns printwriter used to write character data to the response.
`setContentLength(int size)`	Sets the content length for the response to size.
`setContentType(String type)`	Sets the content type for the response to the type.

4.5.6 javax.servlet.http Package

- This contains number of interfaces and classes commonly used by the developer.
- The following table shows interfaces and classes of this package:

Interface	Description
`HttpServletRequest`	Reads data from HTTP request.
`HttpServletResponse`	Writes data to HTTP response.
`HttpSession`	Allows session data to be read and written.
`HttpSessionBindingListner`	Informs an object it is bound or unbound from a session.
Classes	**Description**
`Cookie`	Allows state information to be stored on a client machine.
`HttpServlet`	Provides methods to handle http request and response.
`HttpSessionEvent`	Encapsulates session-changed events.
`HttpSessionBindingEvent`	Tells that a listener is bound to or unbound from a session value.

HttpServletRequest Interface:

- This interface encapsulates information of the request made to a servlet. This interface gets information from client so that it can be used in service() method.
- The HTTP protocol specified header information to be accessed from service() method.

- The different methods of this interface are shown below:

Method	Description
`getMethod()`	Returns the HTTP Get, Post method.
`getQueryString()`	Returns query string that is part of HTTP request.
`getRemoteUser()`	Returns name of user making HTTP request.
`getRequestedSessionId()`	Returns session id with HTTP request.
`getSession(boolean)`	If boolean = false, it returns valid current session else creates a new session.
`isRequestedSessionIdValid()`	Checks whether the HTTP request associated with a session is valid.
`getCookies()`	Returns array of cookies.

HttpServletResponse Interface:

- This encapsulates the information of the response sent by a servlet to a client.
- This allows a service() method to manipulate HTTP protocol specified header information and return data to its client.
- There are various methods available which are shown below:

Method	Description
`addCookie`	Adds specified cookie to a response.
`encodeUrl(String)`	Encodes specified url by adding session to it.
`sendRedirect(String)`	Sends temporary redirect response to a client using specified redirect location url.
`sendError(int)`	Sends error Response to a client.

4.6 READING SERVLET PARAMETERS (FORM)

- The ServletRequest class includes the method which allows us to read the names and values of parameters those are included in a client request.
- This is shown by using two programs, one is for HTML code and other one is for accepting the values from the user.
- Servlets handles form data parsing automatically using the following methods depending on the situation:
  1. `getParameter()`: You call request.getParameter() method to get the value of a form parameter.
  2. `getParameterValues()`: Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
  3. `getParameterNames()`: Call this method if you want a complete list of all parameters in the current request.

1. GET Method Example Using URL Form:

- Here, is a simple program which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this input.

```
<html>
<body>
<form action="HelloForm" method="GET" target="_blank">
<input type="text" name="firstname"/>Firstname
<input type="text" name="lastname"/>Lastname
<br><br>
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

- Keep this HTML in a file HelloForm.htm and put it in <Tomcat-installation-directory>/webapps/ROOT directory. When we would access HelloForm.htm, here is the actual output of the above form.

- Here is a simple URL which will pass two values to HelloForm program using GET method.

```
http://localhost:8080/servlets-examples/servlet
/HelloForm?first_name=Arina&last_name=Willams
```

- Below is HelloForm.java servlet program to handle input given by web browser. We are going to use getParameter() method which makes it very easy to access passed information:

```
// Import required java libraries
import java.io.*;
importjavax.servlet.*;
importjavax.servlet.http.*;
// ExtendHttpServlet class
```

```
public class HelloForm extends HttpServlet
{
   public void doGet(HttpServletRequest request,
   HttpServletResponse response)throwsServletException, IOException
   {
      // Set response content type
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Using GET Method to Read Form Data";
      String docType = "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType + "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<ul>\n" + " <li><b>First Name</b>:"
        + request.getParameter("first_name") + "\n" +
        "<li><b>Last Name</b>:"
        + request.getParameter("last_name") + "\n" +
        "</ul>\n" + "</body></html>");
  }
}
```

- Assuming the environment is setup properly, compile HelloForm.java.
- The above compilation would produce HelloForm.class file. Next we would have to copy this class file in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes and create following entries in web.xml file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/.

```
<servlet>
    <servlet-name>HelloForm</servlet-name>
    <servlet-class>HelloForm</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloForm</servlet-name>
    <url-pattern>/servlet/HelloForm</url-pattern>
</servlet-mapping>
```

- Now type http://localhost:8080/HelloForm?first_name=Arina and last_name=Willams in our browser's Location:box and make sure we already started tomcat server, before firing above command in the browser. This would generate following result:

- Try to enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

2. POST Method Example Using Form:

- Let us do little modification in the above servlet, so that it can handle GET as well as POST methods. Below is HelloForm.java servlet program to handle input given by web browser using GET or POST methods.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class HelloForm1 extends HttpServlet
{
   // Method to handle post method request.
   public void doPost(HttpServletRequest request, HttpServletResponse
   response)
   throws ServletException, IOException
   {
      // Set response content type
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Using Post Method to Read Form Data";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
         "transitional//en\">\n";
         out.println(docType + "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body bgcolor=\"#f0f0f0\">\n" +
                "<h1 align=\"center\">" + title + "</h1>\n" +
```

```
                "<ul>\n" + "<li><b>First Name</b>: "
                + request.getParameter("first_name") + "\n" +
                "  <li><b>Last Name</b>: "
                + request.getParameter("last_name") + "\n" +
                "</ul>\n" + "</body></html>");
    }
}
```

- Now compile, deploy the above Servlet and test it using Hello.htm with the POST method as follows:

```
<html>
<body>
<form action="HelloForm1" method="POST" target="_blank">
<input type="text" name="firstname"/>Firstname
<input type="text" name="lastname"/>Lastname
<br><br>
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

- Here, is the actual output of the above form, Try to enter First and Last Name and then click submit button to see the result on your local machine where tomcat is running.

Output:

3. Passing Checkbox Data to Servlet Program:

- Checkboxes are used when more than one option is required to be selected.
- Here is example HTML code, CheckBox.htm, for a form with two checkboxes.

```
<html>
<body>
<form action="CheckBoxTest" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> Maths
<input type="checkbox" name="physics"/> Physics
<input type="checkbox" name="chemistry" checked="checked" />Chem
<br><br>
<input type="submit" value="Select Subject" />
</form>
</body>
</html>
```

Output:

- Below is CheckBox.java Servlet program to handle input given by web browser for checkbox button.

```
// Import required java libraries
import java.io.*;
importjavax.servlet.*;
importjavax.servlet.http.*;
// ExtendHttpServlet class
public class CheckBoxTest extends HttpServlet
{
   // Method to handle GET method request.
   public void doGet(HttpServletRequest request,
   HttpServletResponse response) throws ServletException, IOException
   {
      // Set response content type
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
```

```
            String title = "Reading Checkbox Data";
            String docType =
            "<!doctype html public \"-//w3c//dtd html 4.0" +
            "transitional//en\">\n";
            out.println(docType + "<html>\n" +
                  "<head><title>" + title + "</title></head>\n"
                  + "<body bgcolor=\"#f0f0f0\">\n"
                  + "<h1 align=\"center\">" + title + "</h1>\n"
                  + "<ul>\n" + "  <li><b>Maths Flag: </b>: "
                  + request.getParameter("maths") + "\n"
                  +" <li><b>Physics Flag: </b>: "
                  + request.getParameter("physics") + "\n"
                  + " <li><b>Chemistry Flag: </b>: "
                  + request.getParameter("chemistry") + "\n" + "</ul>\n"
                  + "</body></html>");
      }
      // Method to handle POST method request.
      public void doPost(HttpServletRequest request,
                                                HttpServletResponse response)
      throwsServletException, IOException
      {
          doGet(request, response);
      }
   }
```

Output:

4. **Reading all Form Parameters:**

HTML Code

```
<html>
<body>
<form action="ReadParams" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> Maths
<input type="checkbox" name="physics"/> Physics
<input type="checkbox" name="chemistry" checked="checked" /> Chem
<br><br>
<input type="submit" value="Select Subject" />
</form>
</body>
</html>
```

- Now calling servlet using above form would generate following result:

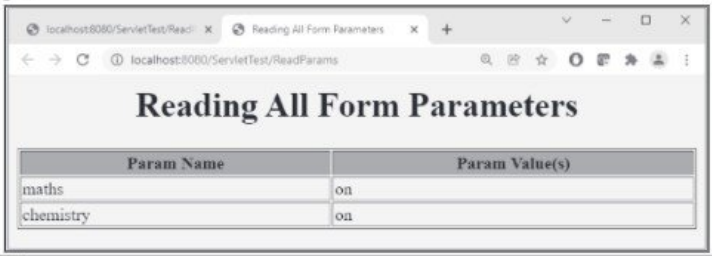
- We can try above servlet to read any other form's data which is having other objects like text box, radio button or drop down box etc.
- Following is the generic example which uses getParameterNames() method of HttpServletRequest to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order.
- Once, we have an Enumeration, we can loop down the Enumeration in the standard manner, using hasMoreElements() method to determine when to stop and using nextElement() method to get each parameter name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
// Extend HttpServlet class
```

```
public class ReadParams extends HttpServlet
{
   // Method to handle GET method request.
   public void doGet(HttpServletRequest request,
   HttpServletResponse response) throws ServletException, IOException
   {
      // Set response content type
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Reading All Form Parameters";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType + "<html>\n" +
         "<head><title>" + title + "</title></head>\n" +
         "<body bgcolor=\"#f0f0f0\">\n" +
         "<h1 align=\"center\">" + title + "</h1>\n" +
         "<table width=\"100%\" border=\"1\" align=\"center\">\n" +
         "<tr bgcolor=\"#949494\">\n" +
         "<th>Param Name</th><th>Param Value(s)</th>\n"+
         "</tr>\n");
      Enumeration paramNames = request.getParameterNames();
      while(paramNames.hasMoreElements())
      {
         String paramName = (String)paramNames.nextElement();
         out.print("<tr><td>" + paramName + "</td>\n<td>");
         String[] paramValues = request.getParameterValues(paramName);
         // Read single valued data
         if (paramValues.length == 1)
         {
            String paramValue = paramValues[0];
            if (paramValue.length() == 0)
            out.println("<i>No Value</i>");
            else
            out.println(paramValue);
         } else
```

```
            {
                // Read multiple valued data
                out.println("<ul>");
                for(int i=0; i < paramValues.length; i++)
                {
                    out.println("<li>" + paramValues[i]);
                }
                out.println("</ul>");
            }
        }
        out.println("</tr>\n</table>\n</body></html>");
    }
    // Method to handle POST method request.
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
    throws ServletException, IOException
    {
        doGet(request, response);
    }
}
```

Output:



4.7 SESSION MANAGEMENT

- Session management is a mechanism used by the Web container to store session information for a particular user.
- There are four different techniques used by Servlet application for session management. These techniques are Cookies, Hidden field, URL Rewriting and Session Object.

4.7.1 Concept of Session

- Session simply means a particular interval of time. Session is used to store everything that we can get from the client from all the requests the client makes.
- Fig. 4.9 shows how sessions work.

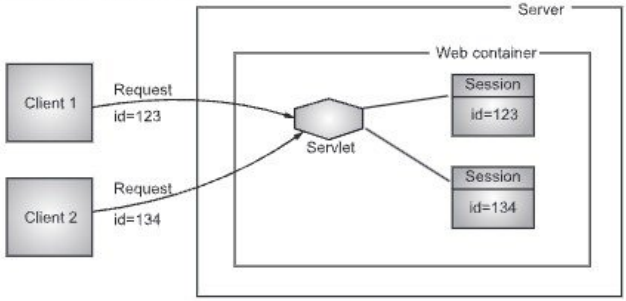


Fig. 4.9: Working of Session

4.7.2 Session Tracking

- Session tracking is a way to maintain state (data) of a user. It is also known as session management in Servlet.
- HTTP protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.
- HTTP is stateless that means each request is considered as the new request. It is shown in the Fig. 4.10.

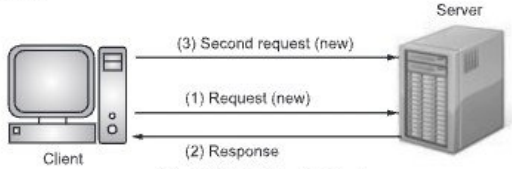


Fig. 4.10: Session tracking

- There are following techniques used in session tracking:
  1. User authorization.
  2. Cookies.
  3. Hidden Form Field.
  4. URL Rewriting.
  5. HttpSession.

4.7.2.1 User Authorization

- Users can be authorized to use the web application in different ways. Basic concept is that the user will provide username and password to login to the application. Based on that the user can be identified and the session can be maintained.
- One way to perform session tracking is to leverage the information that comes with user authorization.
- We can use the username to track a client session. Once a user has logged in, the browser remembers her username and resends the name and password as the user views new pages on the site.
- A Servlet can identify the user through her username and thereby track her session. For example, if the user adds an item to her virtual shopping cart, that fact can be remembered (in a shared class or external database, perhaps) and used later by another Servlet when the user goes to the check-out page.
- For example, a Servlet that utilizes user authorization might add an item to a user's shopping cart with code like the following:

```
String name = req.getRemoteUser();
if (name == null)
{
   // Explain that the server administrator should protect this page
}
else
{
   String[] items = req.getParameterValues("item");
   if (items != null)
   {
      for (int i = 0; i < items.length; i++)
      {
         addItemToCart(name, items[i]);
      }
   }
}
```

- Another servlet can then retrieve the items from a user's cart with code like this:

```
String name = req.getRemoteUser();
if (name == null)
{
   // Explain that the server administrator should protect this page
}
```

```
else
{
    String[] items = getItemsFromCart(name);
}
```

- The biggest advantage of using user authorization to perform session tracking is that it's easy to implement.
- Simply tell the server to protect a set of pages, and use getRemoteUser() to identify each client. Another advantage is that the technique works even when the user accesses the site from different machines. It also works even if the user strays from your site or exits her browser before coming back.
- The biggest disadvantage of user authorization is that it requires each user to register for an account and then log in each time she starts visiting your site.
- Most users will tolerate registering and logging in as a necessary evil when they are accessing sensitive information, but it's overkill for simple session tracking. We clearly need a better approach to support anonymous session tracking.
- Another small problem with user authorization is that a user cannot simultaneously maintain more than one session at the same site.

4.7.2.2 URL Rewriting

- URL rewriting is another way to support anonymous session tracking. With URL rewriting, every local URL the user might click on is dynamically modified or rewritten, to include extra information.
- The extra information can be in the form of extra path information, added parameters or some custom, server-specific URL change.
- Due to the limited space available in rewriting a URL, the extra information is usually limited to a unique session ID.
- With URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource.
- We can send parameter name/value pairs using the following format:

```
url?name1=value1&name2=value2&...
```

- A name and a value is separated using an equal sign (=) a parameter name/value pair is separated from another parameter name/value pair using the ampersand (&).
- When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server.
- From a Servlet, you can use the HttpServletRequest interface's getParameter method to obtain a parameter value.
- For instance, to obtain the value of the second parameter, we write the following:

```
request.getParameter(name2);
```

- The use of URL rewriting is easy and simple. When using this technique, however, we need to consider several things:
  1. The number of characters that can be passed in a URL is limited. Typically, a browser can pass up to 2,000 characters.
  2. The value that we pass can be seen in the URL. Sometimes, this is not desirable. For example, some people prefer their password not to appear on the URL.
  3. We need to encode certain characters, such as & and ? characters and white spaces, that you append to a URL.

URL Rewriting for Session Management:

- If the client has disabled cookie in the browser then session management using cookie won't work. In that case URL rewriting can be used as a backup.
- In URL rewriting, a token (parameter) is added at the end of the URL. The token consist of name/value pair separated by an equal (=) sign.
- Fig. 4.11 shows an example of URL rewriting.

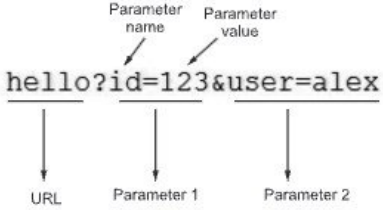


Fig. 4.11: URL Rewriting

- When the user clicks the URL links having parameters, the request goes to the Web Container with extra bit of information at the end of URL. The Web Container will fetch the extra part of the requested URL and uses it for session management.
- The getParameter() method is used to get the parameter value.

Advantage of URL Rewriting:

1. It will always work whether cookie is disabled or not, (browser independent).
2. Extra form submission is not required on each pages.

Disadvantage of URL Rewriting:

1. It will work only with links.
2. It can send only textual information.

4.7.2.3 Hidden Form Fields

- Hidden form field can also be used to store session information for a particular client. In case of hidden form field a hidden field is used to store client state.
- In this case user information is stored in hidden field value and retrieve from another Servlet.

- In short, in Hidden Form Field a hidden (invisible) textfield is used for maintaining the state of a user.
- In such case, we store the information in the hidden field and get it from another Servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.
- Let's see the code to store value in hidden field.

```
<input type="hidden" name="uname" value="Vimal Joshi">
```

Here, uname is the hidden field name and Vimal Joshi is the hidden field value.

- In Fig. 4.12 we are storing the name of the user in a hidden textfield and getting that value from another Servlet.

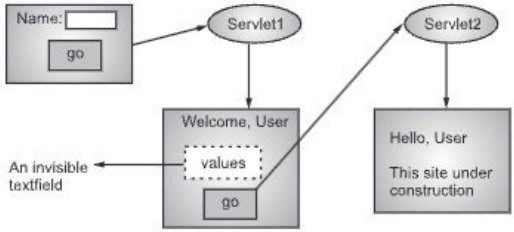


Fig. 4.12: Hidden Form Field

Advantage of Hidden Form Field:

1. It will always work whether cookie is disabled or not.

Disadvantages of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each page.
3. Only textual information can be used.

4.7.2.4 Cookies

- The third technique that we can use to manage user sessions is by using cookies. A cookie is a small piece of information that is passed back and forth in the HTTP request and response.
- Even though a cookie can be created on the client side using some scripting language such as JavaScript, it is usually created by a server resource, such as a servlet.
- The cookies sent by a Servlet to the client will be passed back to the server when the client requests another page from the same application.

How Cookie Works?

- By default, each request is considered as a new request. In cookies technique, we add cookie with response from the Servlet. So cookie is stored in the cache of the browser.
- After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

- Fig. 4.13 shows working of cookies.

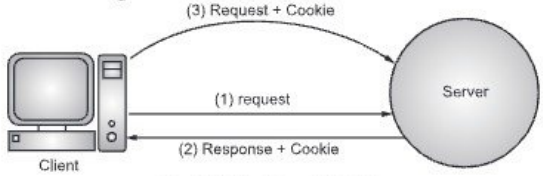


Fig. 4.13: Working of Cookies

- Types of cookies in servlets:
  1. **Non-persistent Cookie:** It is valid for single session only. It is removed each time when user closes the browser.
  2. **Persistent Cookie:** It is valid for multiple session. It is not removed each time when user closes the browser. It is removed only if user logout or sign out.
- In Servlet programming, a cookie is represented by the Cookie class in the javax.servlet.http package.
- We can create a cookie by calling the Cookie class constructor and passing two String objects i.e., the name and value of the cookie.
- For instance, the following code creates a cookie object called c1. The cookie has the name "myCookie" and a value of "secret":

```
Cookie c1 = new Cookie("myCookie", "secret");
```

- Then you can add the cookie to the HTTP response using the addCookie method of the HttpServletResponse interface:

```
response.addCookie(c1);
```

- The following example shows how you can create two cookies called userName and password and illustrates how those cookies are transferred back to the server. The servlet is called CookieServlet and its code is given below.
- When it is first invoked, the doGet method of the servlet is called. The method creates two cookies and adds both to the HttpServletResponse object, as follows:

```
Cookie c1 = new Cookie("userName", "tanmay");
Cookie c2 = new Cookie("password", "rashmi");
response.addCookie(c1);
response.addCookie(c2);
```

- Next, the doGet method sends an HTML form that the user can click to send another request to the servlet:

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<HTML>");
```

```
out.println("<HEAD>");
out.println("<TITLE>Cookie Test</TITLE>");
out.println("</HEAD>");
out.println("<BODY>");
out.println("Please click the button to see the cookies sent to you.");
out.println("<BR>");
out.println("<FORM METHOD=POST>");
out.println("<INPUT TYPE=SUBMIT VALUE=Submit>");
out.println("</FORM>");
out.println("</BODY>");
out.println("</HTML>");
```

- The form does not have any element other than a submit button. When the form is submitted, the doPost method is invoked.
- To retrieve cookies, you use the getCookies method of the HttpServletRequest interface. This method returns a Cookie array containing all cookies in the request. It is your responsibility to loop through the array to get the cookie you want, as follows:

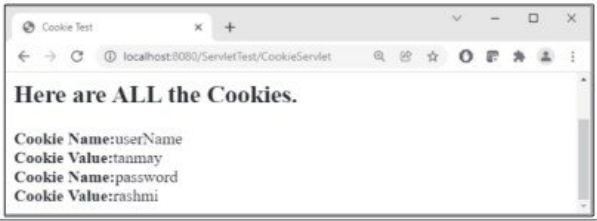
```
Cookie[] cookies = request.getCookies();
int length = cookies.length;
for (int i=0; i<length; i++)
{
   Cookie cookie = cookies[i];
   out.println("<B>Cookie Name:</B> " +cookie.getName() + "<BR>");
   out.println("<B>Cookie Value:</B> " +cookie.getValue() + "<BR>");
}
```

Program 4.2: Program to sending and receiving cookies.

```
importjavax.servlet.*;
importjavax.servlet.http.*;
import java.io.*;
importjava.util.*;
public class CookieServlet extends HttpServlet
{
   /*Process the HTTP Get request*/
   public void doGet(HttpServletRequest request, HttpServletResponse
                              response) throws ServletException, IOException
   {
      Cookie c1 = new Cookie("userName", "tanmay");
      Cookie c2 = new Cookie("password", "rashmi");
```

```
        response.addCookie(c1);
        response.addCookie(c2);
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Cookie Test</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("Please click the button to see the cookies sent to
                    you.");
        out.println("<BR>");
        out.println("<FORM METHOD=POST>");
        out.println("<INPUT TYPE=SUBMIT VALUE=Submit>");
        out.println("</FORM>");
        out.println("</BODY>");
        out.println("</HTML>");
    }
    /**Process the HTTP Post request*/
    public void doPost(HttpServletRequest request, HttpServletResponse
                                response) throws ServletException,IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Cookie Test</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<BR><BR><H2> Here are ALL the Cookies.</H2>");
        Cookie[] cookies = request.getCookies();
        int length = cookies.length;
        for (int i=0; i<length; i++)
        {
            Cookie cookie = cookies[i];
            out.println("<B>Cookie Name:</B>" + cookie.getName() + "<BR>");
            out.println("<B>Cookie Value:</B>" + cookie.getValue() + "<BR>");
        }
        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

Output:



Advantages of Cookies:

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantages of Cookies:

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Persisting Cookies:

- The cookies we created in the previous last as long as the browser is open. When the browser is closed, the cookies are deleted.
- We can choose to persist cookies so that they last longer.
- The javax.servlet.http.Cookie class has the setMaxAge method that sets the maximum age of the cookie in seconds.

4.7.2.5 HttpSession

- The HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from HttpSession object.
- Any servlet can have access to HttpSession object throughout the getSession() method of the HttpServletRequest object.
- An object of HttpSession can be used to perform two tasks:
  1. Bind objects.
  2. View and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

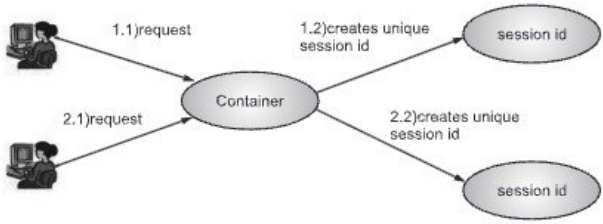


Fig. 4.14: HTTP Session

How HttpSession Works?

- Fig. 4.15 shows working of HttpSession.

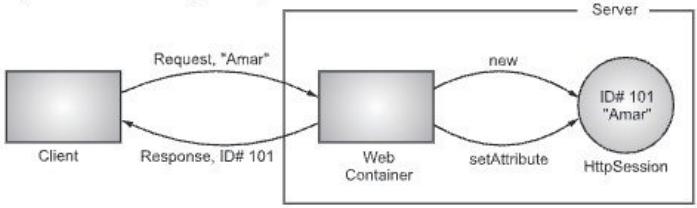
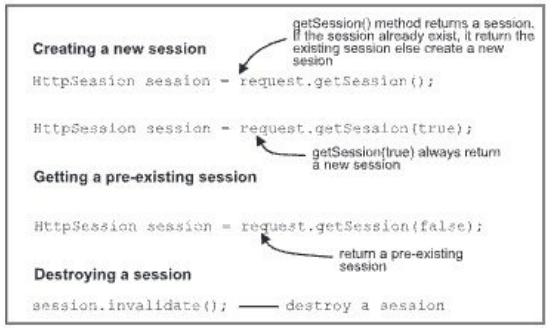


Fig. 4.15: Working of HttpSession

1. On client's first request, the Web Container generates a unique session ID and web container to identify where the request is coming from.
2. The Web Container uses this ID, finds the matching session with the ID and associates the session with the request.

HttpSession Interface:



Some Methods of HttpSession:

1. `long getCreationTime()` method returns the time when the session was created, measured in milliseconds since midnight January 1, 1970 GMT.
2. `String getId()` method returns a string containing the unique identifier assigned to the session.
3. `long getLastAccessedTime()` method returns the last time the client sent a request associated with the session.
4. `int getMaxInactiveInterval()` method returns the maximum time interval, in seconds.
5. `void invalidate()` method destroy the session.
6. `boolean isNew()` method returns true if the session is new else false.
7. `void setMaxInactiveInterval(int interval)` method specifies the time, in seconds, after servlet container will invalidate the session.

Program 4.3: Complete program demonstrating usage of HttpSession. All the files mentioned below are required for the example.

index.html

```
<html>
<body>
<form method="post" action=
                "http://localhost:8080/servlets-examples/servlet/Validate">
  User: <input type="text" name="user" /><br/>
  Password: <input type="text" name="pass" ><br/>
  <input type="submit" value="submit">
</form>
</body>
</html>
```

web.xml

```
    <servlet>
        <servlet-name>Validate</servlet-name>
        <servlet-class>Validate</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>Welcome</servlet-name>
        <servlet-class>Welcome</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Validate</servlet-name>
```

```
        <url-pattern>/servlet/Validate</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Welcome</servlet-name>
        <url-pattern>/servlet/Welcome</url-pattern>
    </servlet-mapping>
```

Validate.java

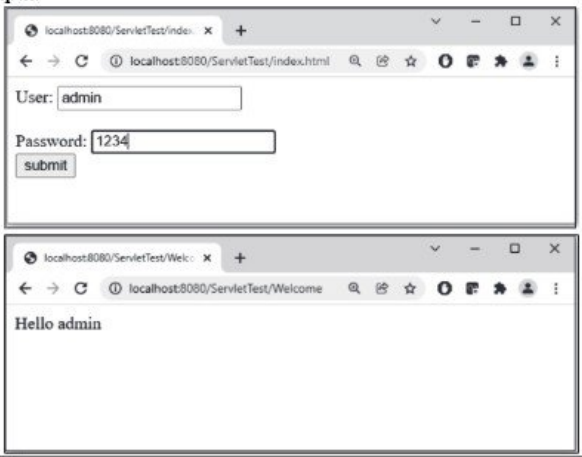
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Validate extends HttpServlet
 {
   protected void doPost(HttpServletRequest request,
                                                HttpServletResponse response)
   throws ServletException, IOException
   {
      response.setContentType("text/html");
      String name = request.getParameter("user");
      String pass = request.getParameter("pass");
      if(pass.equals("1234"))
      {
         //creating a session
         HttpSession session = request.getSession();
         session.setAttribute("user", name);
         response.sendRedirect("Welcome");
      }
   }
}
```

Welcome.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Welcome extends HttpServlet
{
   protected void doGet(HttpServletRequest request,
                                                HttpServletResponse response)
```

```
        throws ServletException, IOException
        {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            HttpSession session = request.getSession();
            String user = (String)session.getAttribute("user");
            out.println("Hello "+user);
        }
    }
```

Output:



4.8 SERVLET WITH DATABASE

- To start with basic concept, let us create a simple table and create few records in that table as follows:

Create database:

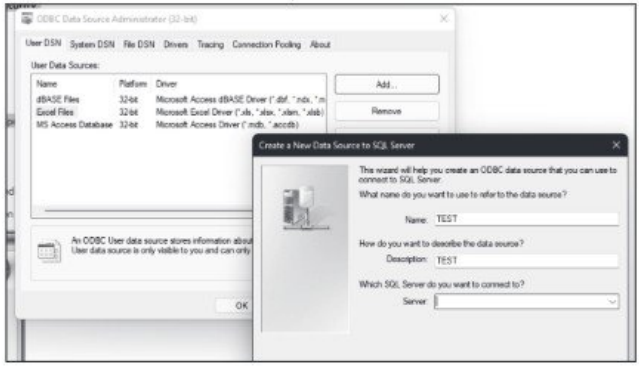
- To create the Employees table in TEST database, use the following steps:
  `On cmd window type:`
- Open the Data Sources (ODBC) program:
  (a) In the Windows Control Panel, click System and Security.
  The System and Security window opens.
  (b) Click Administrative Tools.
  The Administrative Tools window opens.

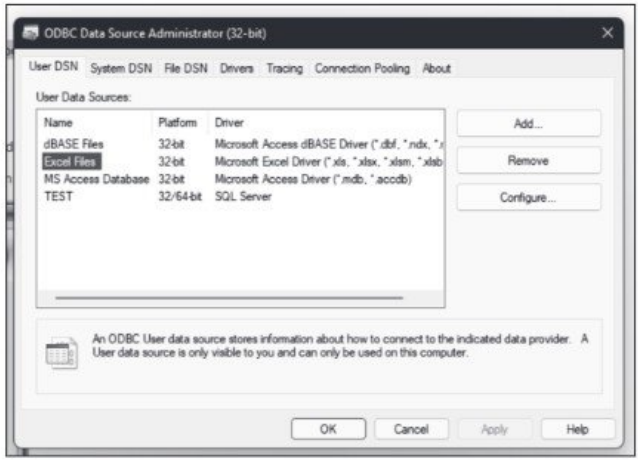
(c) Double-click Data Sources (ODBC).

The ODBC Data Source Administrator dialog box opens.

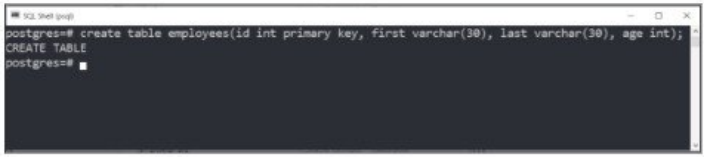
In the User DSN tab, click Add... .

The Create New Data Source dialog box opens.



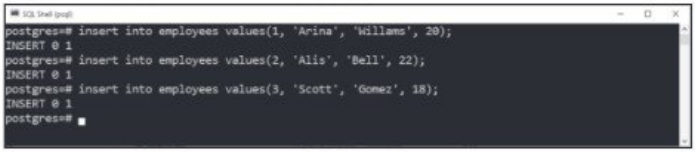


To create table:



Create Data Records:

- Finally We create few records in Employee table as follows:




```
select * from Employees;
 id  | age | first |  last
-----+-----+-------+---------
  1 |  Arina | Willams | 20
  2 |  Alis  | Bell    | 22
  3 |  Scott | Gomez   | 18
(3 rows)
```

Accessing a Database:

- Here is an example which shows how to access TEST database using Servlet.

```
// Loading required libraries
import java.io.*;
importjava.util.*;
importjavax.servlet.*;
importjavax.servlet.http.*;
import java.sql.*;
public class DatabaseAccess extends HttpServlet
{
   public void doGet(HttpServletRequest request, HttpServletResponse
   response)
   throwsServletException, IOException
   {
   Connection conn = null;
      Statement stmt = null;
```

```
// Set response content type
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<h1>DataBase Result</h1>");
try
{
   // Register JDBC driver
   Class.forName("org.postgresql.Driver");
   // Open a connection
   conn = DriverManager.getConnection
       ("jdbc:postgresql://localhost:5432/postgres","postgres","admin");
                                                    // Execute SQL query
   stmt = conn.createStatement();
   String sql;
   sql = "SELECT id, first, last, age FROM Employees";
   ResultSetrs = stmt.executeQuery(sql);
   // Extract data from result set
   while(rs.next())
   {
      //Retrieve by column name
      int id  = rs.getInt("id");
      int age = rs.getInt("age");
      String first = rs.getString("first");
      String last = rs.getString("last");
      //Display values
      out.println("ID: " + id + "<br>");
      out.println("Age: " + age + "<br>");
      out.println("First: " + first + "<br>");
      out.println("Last: " + last + "<br>");
   }
   out.println("</body></html>");
   // Clean-up environment
   rs.close();
   stmt.close();
   conn.close();
   }
```

```
        catch(SQLException se)
        {
           //Handle errors for JDBC
           se.printStackTrace();
        }
        catch(Exception e)
        {
           //Handle errors for Class.forName
           e.printStackTrace();
        }
        finally
        {
           //finally block used to close resources
           try
           {
              if(stmt!=null)
              stmt.close();
           }
           catch(SQLException se2)
           {
           }// nothing we can do
           try
           {
              if(conn!=null)
              conn.close();
           }
           catch(SQLException se)
           {
              se.printStackTrace();
           }//end finally try
           } //end try
        }
     }
```

- Now let us compile above servlet and create following entries in web.xml

```
....
<servlet>
   <servlet-name>DatabaseAccess</servlet-name>
   <servlet-class>DatabaseAccess</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>DatabaseAccess</servlet-name>
    <url-pattern>/servlet/DatabaseAccess</url-pattern>
</servlet-mapping>
....
```

- Now call this servlet using URL http://localhost:8080/DatabaseAccess which would display following response:

4.9 INTRODUCTION TO JSP

- JavaServer Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.
- JSP is a standard for developing interactive Web applications, (pages containing dynamic content).
- A JSP web page may display different content based on certain parameters (information stored in a database, the user preferences etc.), while a classic webpage (with the .htm or .html extension) will continuously display the same information.
- JSP is actually a powerful scripting language executed on the server side (like CGI, PHP, ASP) and not on the client side (Java applets which run in the browser of the user connected to a site).
- JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, JSTL etc.
- A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tag etc.

Why Use JSP?

1. Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
2. JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.

3. JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.
4. JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Comparison between JSP and Servlet:

Terms	JSP	Servlet
What are they?	JSP is a webpage scripting language, generally used to create the dynamic web content.	Servlets are Java programs that are already compiled and which also create dynamic web content.
Typically	JSP is typically more oriented towards displaying information.	Servlet is more oriented towards processing information.
Role in MVC (Model View Controller)	JSP acts as a viewer.	Servlet acts as a controller.
Applicable at the time of	They are generally preferred when there is not much processing of data required.	They are generally preferred when there is more processing and manipulation involved.
Running speed	JSP runs slower as compared to a Servlet. JSP compiles into Java Servlets.	Servlets run faster as compared to JSP.
Code complications	The code programming is easy as compared to that of Servlets.	The code programming is difficult as compared to that of JSP.
Facility	Here, we can build custom tags which can directly call Java beans.	No such facility is available in servlets.
Consists of	JSP are Java HTML representation mixed with JAVA scriptlets.	Servlet are full functional Java codes.
Consistence of objects	JSP has Implicit objects.	Servlets does not have such type of objects.
Examples	To display a report.	To process a user submitted form.

4.9.1 JSP Architecture

- The web server needs a JSP engine i.e., container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages.
- A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.
- Fig. 4.16 shows the position of JSP container and JSP files in a Web Application.

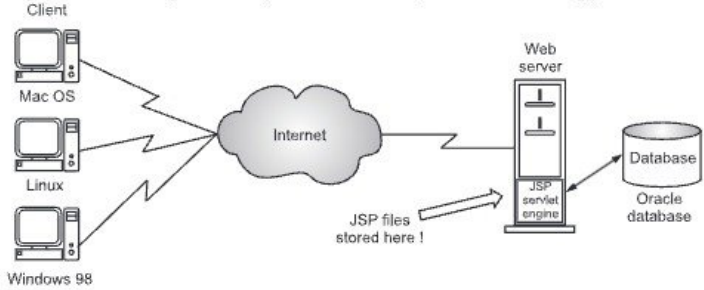


Fig. 4.16: JSP Architecture

4.9.2 JSP Processing

- The following steps explain how the web server creates the web page using JSP:
  - **Step 1:** As with a normal page, the browser sends an HTTP request to the web server.
  - **Step 2:** The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jsp instead of .html.
  - **Step 3:** The JSP engine loads the JSP page from disk and converts it into Servlet content. This conversion is very simple in which all template text is converted to println() statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.
  - **Step 4:** The JSP engine compiles the Servlet into an executable class and forwards the original request to a Servlet engine.
  - **Step 5:** A part of the web server called the Servlet engine loads the Servlet class and executes it. During execution, the Servlet produces an output in HTML format, which the Servlet engine passes to the web server inside an HTTP response.
  - **Step 6:** The web server forwards the HTTP response to the browser in terms of static HTML content.
  - **Step 7:** Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.

- All the above mentioned steps can be shown below in the Fig. 4.17.
- Typically, the JSP engine checks to see whether a Servlet for a JSP file already exists and whether the modification date on the JSP is older than the Servlet.
- If the JSP is older than its generated Servlet, the JSP container assumes that the JSP hasn't changed and that the generated Servlet still matches the JSP's contents.
- This makes the process more efficient than with other scripting languages (such as PHP) and therefore faster.

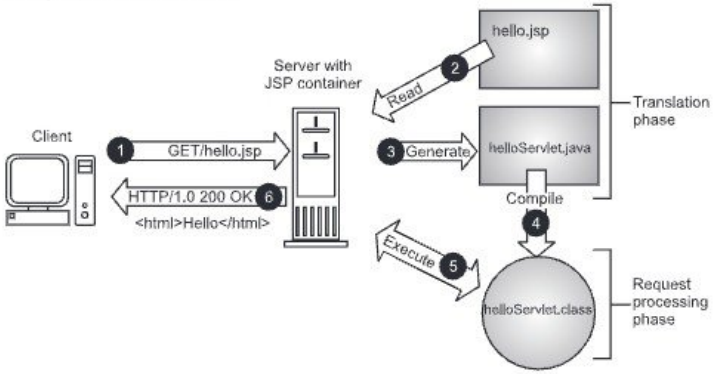


Fig. 4.17: Steps the Web Server creates Web Page using JSP

- So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular Servlet.

4.9.3 Accessing Model of JSP

- JSP is a server side technology which helps to create a webpage dynamically using java as the programming language.
- JSP is a specification from Sun Microsystems. It is an extension to Servlet API.
- The java server pages can be accessed in two different ways :
  1. Access through a client request.
  2. Access through a servlet request.
- **Model 1: Architecture or Access through a Client Request:** In this Model, JSP plays a key role and it is responsible for of processing the request made by client. Client (Web browser) makes a request; JSP then creates a bean object which then fulfills the request and passes the response to JSP. JSP then sends the response back to client. Unlike Model2 architecture in this Model, most of the processing is done by JSP itself.

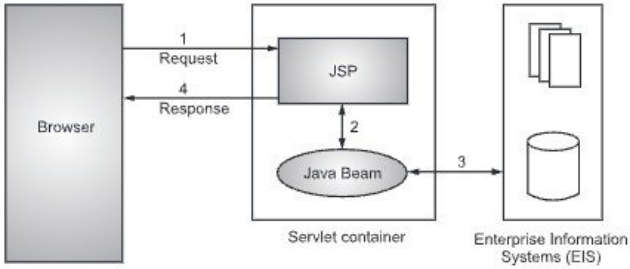


Fig. 4.18: Architecture of Client Request

- **Model 2: Architecture or Access through a Servlet Request:** In this Model Servlet plays a major role and it is responsible for processing the client's (web browser) request. Presentation part (GUI part) will be handled by JSP and it performs this with the help of bean as shown in below figure. The Servlet acts as controller and in charge of request processing. It creates the bean objects if required by the jsp page and calls the respective JSP page. The JSP handles the presentation part by using the bean object. In this Model, JSP doesn't do any processing, Servlet creates the bean Object and calls the JSP program as per the request made by client.

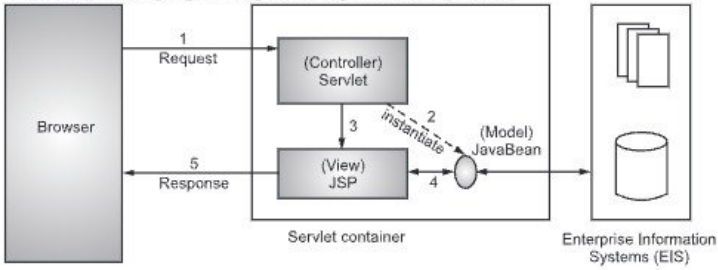


Fig. 4.19: Architecture of Servlet Request

4.9.4 Advantages of JSP

- Following is the list of other advantages of using JSP over other technologies:
  1. **vs. Active Server Pages (ASP):** The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft web servers.
  2. **vs. Pure Servlets:** It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.

3. **vs. Server-Side Includes (SSI):** SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
4. **vs. JavaScript:** JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.
5. **vs. Static HTML:** Regular HTML, of course, cannot contain dynamic information.

4.10 SIMPLE JSP PROGRAM

- Let us start learning with a simple JSP program. All JSP programs are stored as a .jsp files.
- Following is a simple JSP code, MyJSP.jsp which prints Hello, Sample JSP code.
- As discussed JSP is used for creating dynamic webpages. Dynamic webpages can have two types of contents i.e., static and dynamic content.
- The static contents can have text-based formats such as HTML, XML etc and the dynamic contents are generated by JSP elements.

```
<%-- JSP comment --%>
<html>
   <head>
     <title>Message</title>
   </head>
   <body>
      <%out.print("Hello, Sample JSP code");%>
   </body>
</html>
```

Analysis of the above code:

1. The line **<%–JSP Comment–%>** represents the JSP element called JSP Comment, While adding comments to a JSP page you can use this tag. JSP Comments must starts with a tag <%-- and ends with --%>.
2. **HEAD, TITLE and BODY tags are HTML tags**: They are HTML tags, frequently used for static web pages.
3. **<%out.print("Hello, Sample JSP code");%>** is a JSP element, which is known as Scriptlet. Scriptlets can contain Java codes. Syntax of scriptlet is: <%Executable java code%>. As the code in Scriptlets is java statement, they must end with a semicolon(;). out.print("Hello, Sample JSP code") is a java statement, which prints" Hello, Sample JSP code".

4.11 LIFE CYCLE OF JSP

- A JSP life cycle can be defined as "the entire process from its creation till the destruction which is similar to a Servlet life cycle with an additional step which is required to compile a JSP into Servlet".
- A JSP page services requests as a servlet. Thus, the life cycle and many of the capabilities of JSP pages (in particular the dynamic aspects) are determined by Java Servlet technology.
- The following are the paths followed by a JSP:
  1. Compilation.
  2. Initialization.
  3. Execution.
  4. Cleanup.
- The four major phases of JSP life cycle are very similar to Servlet Life Cycle as shown in Fig. 4.20.

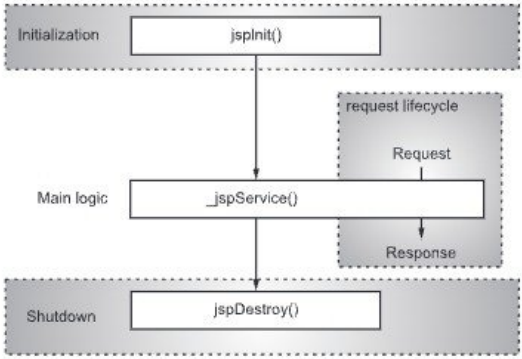


Fig. 4.20: Life cycle of JSP

- Fig. 4.20 shows following parts of a JSP page:

1. JSP Compilation:

- When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page.
- If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.
- The compilation process involves three steps:
  (i) Parsing the JSP.
  (ii) Turning the JSP into a servlet.
  (iii) Compiling the servlet.

2. JSP Initialization:

- When a container loads a JSP it invokes the jspInit() method before servicing any requests.
- If you need to perform JSP-specific initialization, override the jspInit() method:

```
public void jspInit(){
// Initialization code...
}
```

- Typically initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files and create lookup tables in the jspInit() method.

3. JSP Execution:

- This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.
- Whenever, a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the _jspService() method in the JSP.
- The _jspService() method takes an HttpServletRequest and an HttpServletResponse as its parameters as follows:

```
void _jspService(HttpServletRequest request,
                 HttpServletResponse response)
{
// Service handling code...
}
```

- The _jspService() method of a JSP is invoked once per a request and is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods i.e. GET, POST, DELETE etc.

4. JSP Cleanup:

- The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.
- The jspDestroy() method is the JSP equivalent of the destroy method for servlets.
- Override jspDestroy() when you need to perform any cleanup, such as releasing database connections or closing open files.

4.12 JSP IMPLICIT OBJECTS

- JSP Implicit Objects are the Java objects that the JSP Container makes available to developers in each page and developer can call them directly without being explicitly declared.
- JSP Implicit Objects are also called pre-defined variables.

- There are total nine implicit objects available in JSP as given in following Table:

Sr. No.	Object	Description
1.	request	This is the HttpServletRequest object associated with the request.
2.	response	This is the HttpServletResponse object associated with the response to the client.
3.	out	This is the PrintWriter object used to send output to the client.
4.	session	This is the HttpSession object associated with the request.
5.	application	This is the ServletContext object associated with application context.
6.	config	This is the ServletConfig object associated with the page.
7.	pageContext	This encapsulates use of server-specific features like higher performance JspWriters.
8.	page	This is simply a synonym for this, and is used to call the methods defined by the translated servlet class.
9.	Exception	The Exception object allows the exception data to be accessed by designated JSP.

4.13 SCRIPTING ELEMENTS OF JSP

- When we are developing java server page, the JSP allows us to use the scriptlets (JSP script tags) and also allows us to invoke java components.
- The JSP script may contain HTML code. This document can be developed in any editor with .htm or .html file. We can directly use it as a .jsp file.
- The JSP page is built up using the following components:
  - Directives.
  - Declarations.
  - Scriptlets.
  - Expressions.
  - Standard Actions.
  - Custom Tags.

4.13.1 Declarations

- A declaration declares one or more variables or methods that you can use in Java code later in the JSP file.
- We must declare the variable or method before you use it in the JSP file.
- Following is the syntax of JSP Declarations:

```
<%! declaration;[ declaration;]+... %>
```

- We can write XML equivalent of the above syntax as follows:

```
<jsp:declaration>
   code fragment
</jsp:declaration>
```

- Following is the simple example for JSP Declarations:

```
<%!int i =0; %>
<%!int a, b, c; %>
<%!Circle a =newCircle(2.0); %>
```

Program 4.4: Program of Variables Declaration.

- In this example we have declared two variables inside declaration tag and displayed them on client using expression tag.

```
<html>
<head>
 <title>Declaration tag Example1</title>
</head>
<body>
<%! String name="Chaitanya"; %>
<%! int age=27; %>
<%= "Name is: "+ name %><br>
<%= "AGE: "+ age %>
</body>
</html>
```

Output:

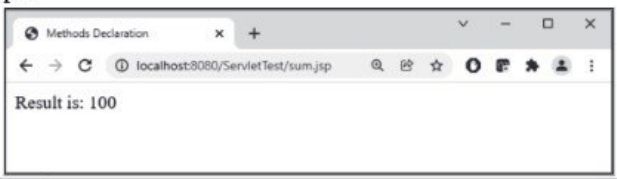
Program 4.5: Program of Methods Declaration.

- In this example we have declared a method sum using JSP declaration tag.

```
<html>
<head>
<title>Methods Declaration</title>
```

```
</head>
<body>
  <%!
  int sum(int num1, int num2, int num3){
  return num1+num2+num3;
  }
  %>
  <%= "Result is: " + sum(10,40,50) %>
</body>
</html>
```

Output:



4.13.2 Expressions

- Expression is used to insert values directly to the output. An expression tag places an expression to be evaluated inside the java servlet class.
- A JSP expression element contains a scripting language expression that is evaluated, converted to a String and inserted where the expression appears in the JSP file.
- Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.
- The expression element can contain any expression that is valid according to the Java language specification but you cannot use a semicolon to end an expression.
- Following is the syntax of JSP Expression:

```
<%= expression %>
```

- You can write XML equivalent of the above syntax as follows:

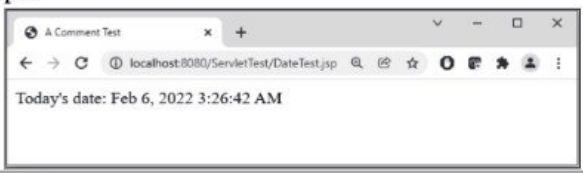
```
<jsp:expression>
   expression
</jsp:expression>
```

- Following is the simple example for JSP Expression:

```
<html>
<head><title>A Comment Test</title></head>
<body>
```

```
<p>
Today's date: <%=(new java.util.Date()).toLocaleString()%>
</p>
</body>
</html>
```

Output:



4.13.3 Scriptlets

- A scriptlet can contain any number of java language statements, variable or method declarations, or expressions that are valid in the page scripting language.
- JSP Scriptlets begins with <% and ends %>. We can embed any amount of java code in the JSP Scriptlets.
- **Syntax for Scriptlet:**

```
<% // any java source code here ... %>
```

- A scriptlet is a fragment of Java code that is run when the user requests the page.
- For example, any Java if/for/while blocks opened in one scriptlet element must be correctly closed in a later element for the page to successfully compile. Markup which falls inside a split block of code is subject to that code, so markup inside an if block will only appear in the output when the if condition evaluates to true; likewise, markup inside a loop construct may appear multiple times in the output depending upon how many times the loop body runs.
- The following would be a valid for loop in a JSP page:

```
<p>Counting to three:</p>
<% for(int i=1; i<4; i++){ %>
<p>This number is <%= i %> </p>
<% } %>
<p>OK</p>
```

- The output displayed in the user's web browser would be:

```
Counting to three:
This number is 1.
This number is 2.
This number is 3.
OK.
```

4.13.4 Comments

- JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out" part of your JSP page.
- Following is the syntax of JSP comments:

```
<%-- This is JSP comment ... --%>
```

- Following is the simple example for JSP Comments:

```
<html>
<head><title>A Comment Test</title></head>
<body>
<h2>A Test of Comments</h2>
<%-- This comment will not be visible in the page source --%>
</body>
</html>
```

Output: `A Test of Comments`

- There are a small number of special constructs you can use in various cases to insert comments or characters that would be treated specially. Here's a summary:

Syntax	Purpose
`<%-- comment --%>`	A JSP comment. Ignored by the JSP engine.
`<!-- comment -->`	An HTML comment. Ignored by the browser.
`<\%`	Represents static <% literal.
`%\>`	Represents static %> literal.
`\'`	A single quote in an attribute that uses single quotes.
`\"`	A double quote in an attribute that uses double quotes.

4.14 JSP DIRECTIVES

- JSP directives provide directions and instructions to the container, telling it how to handle certain aspects of JSP processing.
- JSP directive tag gives special information about the page to the JSP engine.
- A JSP directive affects the overall structure of the servlet class. It usually has the following form/syntax:

```
<%@ directive attribute="value" %>
```

- Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.
- The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

- There are three types of directive tag:

Directive	Description
`<%@ page ... %>`	Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
`<%@ include ... %>`	Includes a file during the translation phase.
`<%@ taglib ... %>`	Declares a tag library, containing custom actions, used in the page.

4.14.1 Page Directive

- The page directive is used to provide instructions to the container that pertain to the current JSP page.
- We may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.
- Following is the basic syntax of page directive:

```
<%@ page attribute="value" %>
```

- We can write XML equivalent of the above syntax as follows:

```
<jsp:directive.pageattribute="value"/>
```

- **Attributes:** Following is the list of attributes associated with page directive:

Attribute	Purpose
`Buffer`	Specifies a buffering model for the output stream.
`autoFlush`	Controls the behavior of the servlet output buffer.
`contentType`	Defines the character encoding scheme.
`errorPage`	Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
`isErrorPage`	Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.
`Extends`	Specifies a superclass that the generated servlet must extend.
`Import`	Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.
`Info`	Defines a string that can be accessed with the servlet's getServletInfo() method.
`isThreadSafe`	Defines the threading model for the generated servlet.
`Language`	Defines the programming language used in the JSP page.
`Session`	Specifies whether or not the JSP page participates in HTTP sessions.
`isELIgnored`	Specifies whether or not EL expression within the JSP page will be ignored.
`isScriptingEnabled`	Determines if scripting elements are allowed for use.

4.14.2 Include Directive

- The include directive is used to includes a file during the translation phase.
- This directive tells the container to merge the content of other external files with the current JSP during the translation phase.
- You may code include directives anywhere in your JSP page.
- The general usage form/syntax of this directive is as follows:

```
<%@ include file="relative url">
```

- The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.
- You can write XML equivalent of the above syntax as follows:

```
<jsp:directive.includefile="relative url"/>
```

taglib Directive:

- The JavaServer Pages API allows you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.
- The taglib directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides a means for identifying the custom tags in your JSP page.
- The taglib directive follows the following syntax:

```
<%@ taglib uri="uri" prefix="prefixOfTag">
```

Where, the uri attribute value resolves to a location the container understands and the prefix attribute informs a container what bits of markup are custom actions.

4.15 MIXING SCRIPTLETS AND HTML

- We have already seen how to use the "out" variable to generate HTML output from within a scriptlet.
- For more complicated HTML, using the out variable all the time loses some of the advantages of JSP programming. It is simpler to mix scriptlets and HTML.
- Suppose we have to generate a table in HTML. This is a common operation, and you may want to generate a table from a SQL table, or from the lines of a file.
- But to keep our example simple, we will generate a table containing the numbers from 1 to N. Not very useful, but it will show you the technique.
- Here, is the JSP fragment to do it:

```
<table border=2>
<%
    for( int i = 0; i < n; i++ ) {
        %>
```

```
            <tr>
            <td>Number</td>
            <td><%= i+1 %></td>
            </tr>
            <%
        }
    %>
    </table>
```

- We would have to supply an `int` variable "n" before it will work, and then it will output a simple table with "n" rows.
- The important things to notice are how the `%>` and `<%` characters appear in the middle of the "`for`" loop, to let you drop back into HTML and then to come back to the Scriptlet.
- The concepts are simple here - as we can see, you can drop out of the scriptlets, write normal HTML, and get back into the Scriptlet.
- Any control expressions such as a "while" or a "for" loop or an "if" expression will control the HTML also. If the HTML is inside a loop, it will be emitted once for each iteration of the loop.
- Another example of mixing Scriptlets and HTML is shown below - here it is assumed that there is a Boolean variable named "`hello`" available. If we set it to true, you will see one output, if we set it to false, we will see another output.

```
<%
    if( hello ) {
        %>
        <p>Hello, world
        <%
    } else {
        %>
        <p>Goodbye, world
        <%
    }
%>
```

- It is little difficult to keep track of all open braces and scriptlet start and ends, but with a little practice and some good formatting discipline, you will acquire competence in doing it.

4.16 JSP WITH DATABASE

- The database is used for storing various types of data which are huge and has storing capacity in gigabytes. JSP can connect with such databases to create and manage the records.
- Following example shows how we can execute the **SQL SELECT** statement using JTSL in JSP programming –

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>
<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>
<html>
   <head>
      <title>SELECT Operation</title>
   </head>

   <body>
      <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
         url = "jdbc:mysql://localhost/TEST"
         user = "root"  password = "pass123"/>

      <sql:query dataSource = "${snapshot}" var = "result">
         SELECT * from Employees;
      </sql:query>

      <table border = "1" width = "100%">
         <tr>
            <th>Emp ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Age</th>
         </tr>

         <c:forEach var = "row" items = "${result.rows}">
            <tr>
               <td><c:out value = "${row.id}"/></td>
               <td><c:out value = "${row.first}"/></td>
               <td><c:out value = "${row.last}"/></td>
               <td><c:out value = "${row.age}"/></td>
            </tr>
         </c:forEach>
      </table>

   </body>
</html>
```

Output:

Emp ID	First Name	Last Name	Age
100	sunil	shetty	40
101	sara	Khan	25
102	madhuri	Dikshit	35
103	Akshay	khanna	28

Insert command:

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>
<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>

<html>
   <head>
      <title>JINSERT Operation</title>
   </head>

   <body>
      <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
         url = "jdbc:mysql://localhost/TEST"
         user = "root"  password = "pass123"/>
         <sql:update dataSource = "${snapshot}" var = "result">
         INSERT INTO Employees VALUES (104, 32, 'saha', 'chopra');
      </sql:update>

      <sql:query dataSource = "${snapshot}" var = "result">
         SELECT * from Employees;
      </sql:query>

      <table border = "1" width = "100%">
         <tr>
            <th>Emp ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Age</th>
         </tr>

         <c:forEach var = "row" items = "${result.rows}">
            <tr>
               <td><c:out value = "${row.id}"/></td>
               <td><c:out value = "${row.first}"/></td>
               <td><c:out value = "${row.last}"/></td>
               <td><c:out value = "${row.age}"/></td>
```

```
            </tr>
          </c:forEach>
        </table>

    </body>
</html>
```

Output:

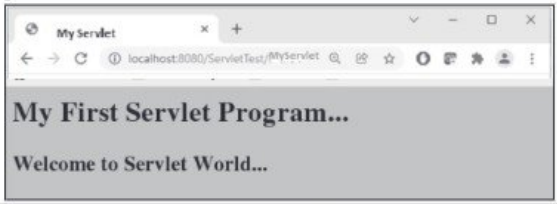
Emp ID	First Name	Last Name	Age
100	sunil	shetty	40
101	sara	Khan	25
102	madhuri	Dikshit	35
103	Akshay	khanna	28
104	saha	chopra	32

ADDITIONAL PROGRAMS

Program 4.6: Program for displaying the message using Servlet.

```
// Firstservlet.java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Firstservlet extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse res)
                                        throws IOException, ServletException
   {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
       out.println("<html>");
      out.println("<head>");
      out.println("<title>MyServlet</title>");
      out.println("</head>");
      out.println("<body bgcolor=\"cyan\">");
      out.println("<h1> My First Servlet Program...</h1>");
      out.println("<h2>Welcome to Servlet World...");
      out.println("</body>");
      out.println("</html>");
   }
}
```

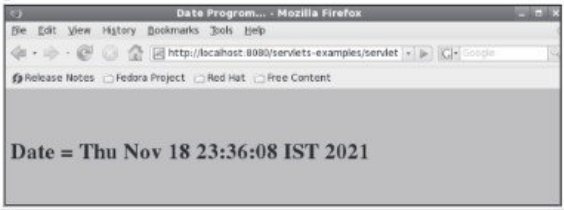
Output:



Program 4.7: Program for displaying the Date using Servlet.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyDate extends HttpServlet
 {
   public void doGet(HttpServletRequest req,HttpServletResponse res)
                                        throws IOException, ServletException
   {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();

      out.println("<html>");
      out.println("<head>");

      out.println("<title>Date Progrom...</title>");
      out.println("</head>");
      out.println("<body bgcolor=\"cyan\">");
      Date d = new Date();

      out.println("<br><br><h1>Date = "+d+"</h1>");

      out.println("</body>");
      out.println("</html>");
   }
}
```

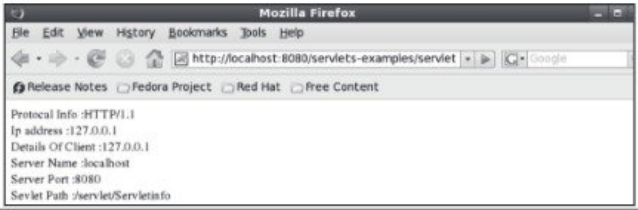
Output:



Program 4.8: Program to displaying the Servlet information.

```
// Servletinfo.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Servletinfo extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse res)
                                        throws ServletException,IOException
   {
      res.setContentType("text/html");
      PrintWriter out=res.getWriter();

      out.println("Protocal Info :"+req.getProtocol());
      out.println("<br>Ip address :"+req.getRemoteAddr());
      out.println("<br>Details Of Client :"+req.getRemoteHost());
      out.println("<br>Server Name :"+req.getServerName());
      out.println("<br>Server Port :"+req.getServerPort());
      out.println("<br>Sevlet Path :"+req.getServletPath());
   }
}
```

Output:

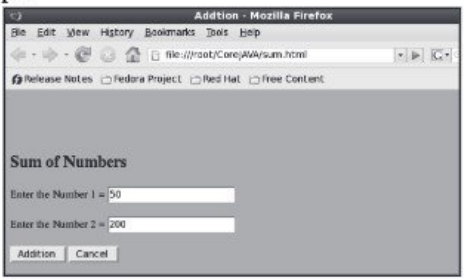


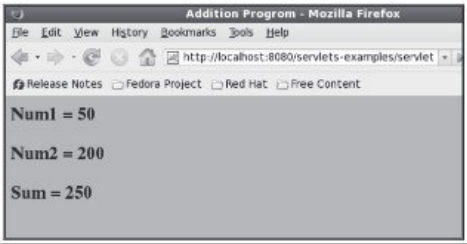
Program 4.9: Program for displaying the sum of two numbers gets the inputs from user.

```
//Sum.html
<html>
<head>
<title>Addtion</title>
</head>
<body bgcolor = "cyan">
<form action = "http:
   //localhost:8080/servlets-examples/servlet/AddNumServlet" method ="GET">
<br><br><br>
<h2> Sum of Numbers </h2>
Enter the Number 1 = <input type = "text" name = "n1"><br><br>
Enter the Number 2 = <input type = "text" name = "n2">
<br><br>
<input type = "submit" value = "Addition ">
<input type = "reset" value = "Cancel ">
</form>
</body>
</html>
// AddNumServlet.java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class AddNumServlet extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse res)
                                        throws IOException, ServletException
   {
      int num1 = Integer.parseInt(req.getParameter("n1"));
      int num2 = Integer.parseInt(req.getParameter("n2"));
      int add = num1 + num2;
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      out.println("<html>");
      out.println("<head>");
```

```
        out.println("<title>Addition Progrom</title>");
        out.println("</head>");
        out.println("<body bgcolor=\"cyan\">");
        out.println("<h2> Num1 = "+num1+"</h2>");
        out.println("<h2> Num2 = "+num2+"</h2>");
        out.println("<h2> Sum = "+add + "</h2>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Output:



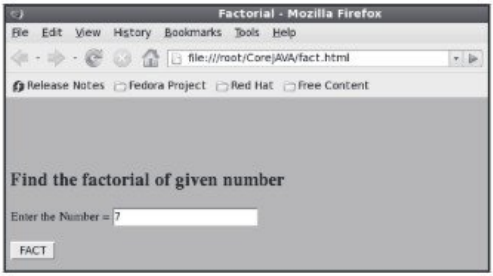


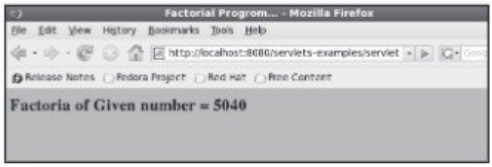
Program 4.10: Program to find the factorial of given number using Servlet program.

```
//fact.html
<html>
<head>
<title>Factorial</title>
</head>
```

```
<body bgcolor = "cyan">
<form action = "http://localhost:8080/servlets-examples/
                                        servlet/Factservlet" method ="GET">
<br><br><br>
<h2> Find the factorial of given number </h2>
Enter the Number = <input type = "text" name = "txt">
<br><br>
<input type = "submit" value = "FACT">
</form>
</body>
</html>
//Factservlet.java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Factservlet extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse res)
                                        throws IOException, ServletException
   {
      int no = Integer.parseInt(req.getParameter("txt"));
      int i,f=1;
      for(i=1;i<=no;i++)
      {
         f = f * i;
      }
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      out.println("<html>");
      out.println("<head>");
      out.println("<title> Factorial Progrom...</title>");
      out.println("</head>");
      out.println("<body bgcolor=\"cyan\">");
      out.println("<h2> Factoria of Given number = "+f+"</h2>");
      out.println("</body>");
      out.println("</html>");
   }
}
```

Output:



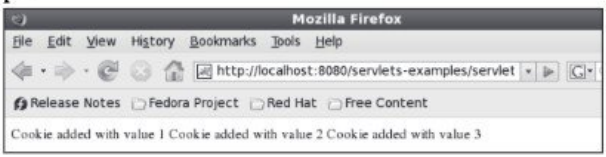


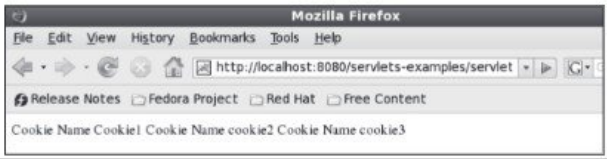
Program 4.11: Program of add Cookie and get the information of Cookies.

```
//Addcookie.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class Addcookie extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse res)
                                        throws ServletException,IOException
   {
      Cookie c1 = new Cookie("Cookie1","1");
      res.addCookie(c1);
      res.setContentType("text/html");
      PrintWriter pw = res.getWriter();
      pw.println("Cookie added with value 1");
      Cookie c2 = new Cookie("cookie2","2");
      res.addCookie(c2);
      pw.println("Cookie added with value 2");
```

```
        Cookie c3 = new Cookie("cookie3","3");
        res.addCookie(c3);
        pw.println("Cookie added with value 3");
    }
}
// Getcookie.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class Getcookie extends HttpServlet
{
    public void doGet(HttpServletRequest req,HttpServletResponse res)
                                        throws ServletException,IOException
    {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        Cookie c[] = req.getCookies();
        for(int i=0;i<c.length;i++)
        out.println("Cookie Name "+c[i].getName());
    }
}
```

Output:



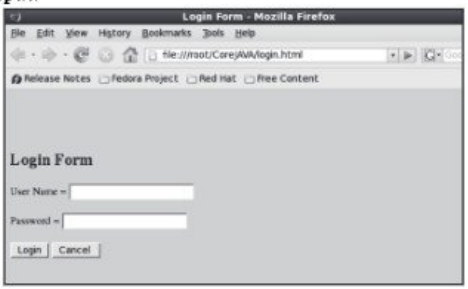


Program 4.12: Program to display login successfully or not get the input from user and check with respect to the admin table (use the PostgreSQL database).

```
//Login.html
<html>
<head>
<title>Login Form</title>
</head>
<body bgcolor = "pink">
<form action = "http://localhost:8080/
                    servlets-examples/servlet/Loginservlet" method ="GET">
<br><br><br>
<h2> Login Form</h2>
User Name = <input type = "text" name = "us"><br><br>
Password = <input type = "text" name = "pd">
<br><br>
<input type = "submit" value = "Login">
<input type = "reset" value = "Cancel ">
</form>
</body>
</html>
// Loginservlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Loginservlet extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse res)throws
   IOException, ServletException
   {
      String uname = req.getParameter("us");
      String pwd = req.getParameter("pd");
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      try
      {
         out.println("<html>");
```

```
            out.println("<body>");
            Class.forName("postgresql.Driver");
            out.println("<h1>Driver Loaded</h1>");
            Connection c=DriverManager.getConnection
                                        ("jdbc:postgresql:mj","postgres","");
            out.println("<br><h1>Connection Established</h1>");
            Statement st=c.createStatement();
            ResultSet rs=st.executeQuery("select * from admin");
            while(rs.next())
            {
                if(uname.equals(rs.getString(1))&&pwd.equals(rs.getString(2)))
                out.println("<h2>Login Successfully....</h2>");
                else
                out.println("<h3> NOT ! Try Again...</h3>");
            }
        }
        catch(Exception e)
        {
            out.println("error"+e);
        }
        out.println("Successfully JDBC Done...");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Output:

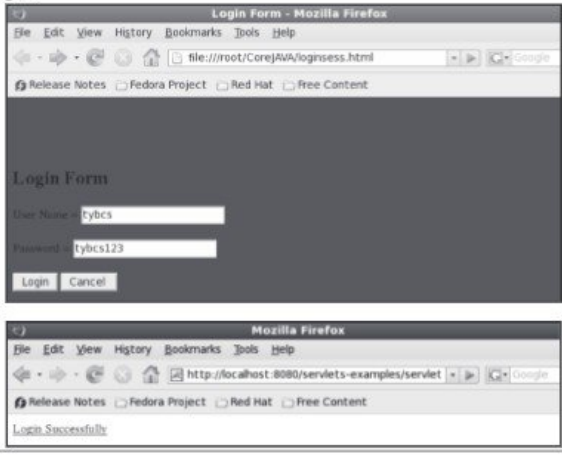


Program 4.13: Program for Servlet session.

```
//Sess.html
<html>
<head>
<title>Login Form</title>
</head>
<body bgcolor = "purple">
<form action = "http://localhost:8080/servlets-examples/servlet/Sess1"
method ="GET">
<br><br><br>
<h2> Login Form</h2>
User Name = <input type = "text" name = "us"><br><br>
Password = <input type = "text" name = "pd">
<br><br>
<input type = "submit" value = "Login">
<input type = "reset" value = "Cancel ">
</form>
</body>
</html>
// Sess1.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Sess1 extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse res)throws
   IOException, ServletException
```

```
    {
        HttpSession hs = req.getSession(true);
        String uname = req.getParameter("us");
        String pwd = req.getParameter("pd");
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        if(uname.equals("tybcs")&&pwd.equals("tybcs123"))
        {
           out.println("<a href=
             http://localhost:8080/servlets-examples/servlet/Sess.html> Login
                                                    Successfully </a>");
           hs.setAttribute("LoginID",uname);
        }
        else
        {
           hs.setAttribute("LoginID",pwd);
        }
    }
}
```

Output:

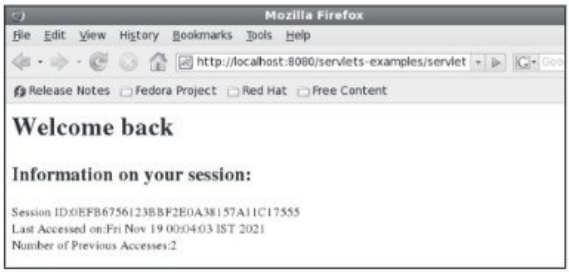


Program 4.14: Program for session.

```
// Webcount.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class Webcount extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse res)throws
   ServletException,IOException
   {
      Integer count;
      res.setContentType("text/html");
      PrintWriter out=res.getWriter();
      HttpSession session=req.getSession(true);
      String heading;
      count=(Integer)session.getAttribute("accessCount");
      if(count==null)
      {
         count=new Integer(0);
         heading="Welcome,newcomer";
      }
      else
      {
         heading="Welcome back";
         count=new Integer((count.intValue())+1);
      }
      session.setAttribute("accessCount",count);
      out.println("<BODY><HTML>" + "<H1>" + heading + "</H1>\n"
         + "<H2>Information on your session:</H2>\n"
         + "Session ID:"+session.getId()
         + "<br>Last Accessed on:"
         + new Date(session.getLastAccessedTime())
         + "<br>Number of Previous Accesses:"
         + count + "<br></BODY></HTML>");
   }
```

```
    public void doPost(HttpServletRequest req,HttpServletResponse res)throws
    ServletException,IOException
    {
        doGet(req,res);
    }
}
```

Output:

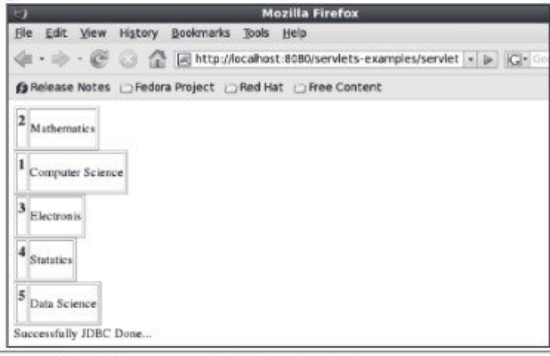


Program 4.15: Program to displaying the information of Department which store in the database (Servlet and PostgreSQL).

```
// Jdbcservlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Jdbcservlet extends HttpServlet
{
    public void doGet(HttpServletRequest req,HttpServletResponse res)throws
    IOException, ServletException
```

```
    {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        try
        {
           out.println("<html>");
           out.println("<body>");
           Class.forName("postgresql.Driver");
           out.println("<h1>driver loaded</h1>");
           Connection c=DriverManager.getConnection
                                    ("jdbc:postgresql:serv","postgres","");
           out.println("<br><h1>connection created</h1>");
           Statement st=c.createStatement();
           ResultSet rs=st.executeQuery("select * from department");
           while(rs.next())
           {
               out.println("<table border= 1>");
               out.println("<tr>");
               out.print("<td><h3>"+rs.getInt(1)+
                                  "</td><td>"+rs.getString(2)+"</td></h3>");
               out.println("</tr>");
               out.println("</table>");
           }
        }
        catch(Exception e)
        {
           out.println("error"+e);
        }
        out.println("Successfully JDBC Done...");
        out.println("</body>");
        out.println("</html>");
    }
}
```

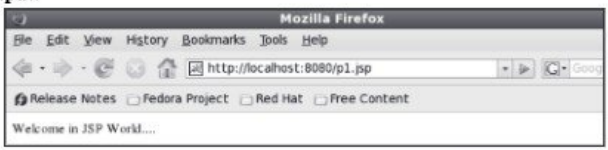
Output:



Program 4.16: JSP program to display the string.

```
<html>
<body>
<%
   out.println("Welcome in JSP World....");
%>
</body>
</html>
```

Output:

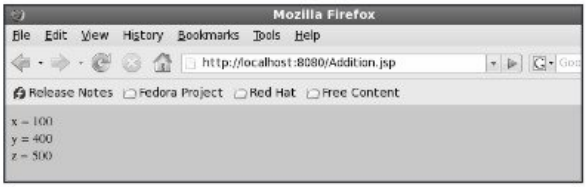


Program 4.17: JSP program for addition of two numbers.

```
<html>
<body bgcolor="cyan">
<%! int x,y,z;%>
<%
   x = 100;
   y = 400;
   z = x + y;
%>
```

```
<%= "x = "+ x%><br>
<%= "y = "+ y%><br>
<%= "z = "+ z %>
</body>
</html>
```

Output:

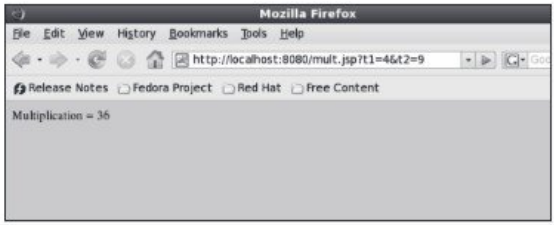


Program 4.18: Program to get the input from user and display the multiplication using JSP.

```
//Mult.html
<html>
<body bgcolor="cyan">
<br>
<form action="http://localhost:8080/mult.jsp" method="GET">
<center>
<h2>Enter the First Number :<input type = text name ="t1">
<br>Enter the Second Number : <input type = text name= "t2">
<br>
<br>
<input type="Submit" value="Mult">
<input type="reset" value="Cancel">
</form>
</body>
</html>
//Mult.jsp
<%@ page language="java"%>
<html>
<body bgcolor="pink">
<%
     int x = Integer.parseInt(request.getParameter("t1"));
     int y = Integer.parseInt(request.getParameter("t2"));
```

```
        int z = x * y;
        out.println("Multiplication = "+z);
    %>
    </body>
    </html>
```

Output:

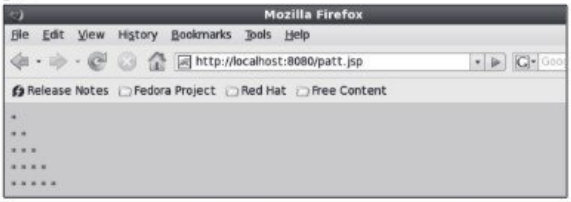


Program 4.19: Program to display the pattern.

```
<html>
<body bgcolor="pink">
<%! int n,i,j;%>
    <%
    n = 5;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=i;j++)
        {
            %>
            <%= "*"%>
```

```
            <% }%>
          <br>
      <%
      }
      %>
  </body>
  </html>
```

Output:



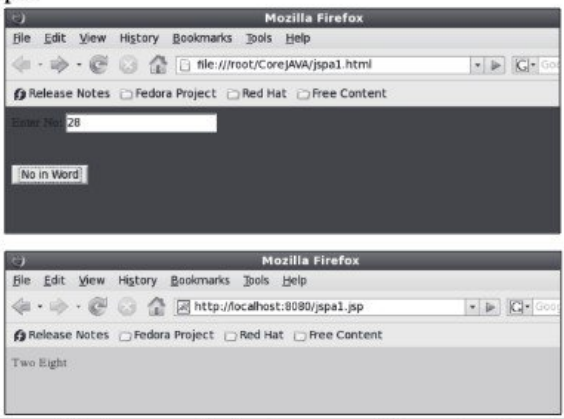
Program 4.20: Program to display the number into the word.

```
//jspa1.jsp
<html>
<body bgcolor="green">
<form action="http://localhost:8080/jspa1.jsp" method="post" >
Enter No: <input type="text" name="txt"><br><br><br>
<input type="submit" value="No in Word">
</form>
</body>
</html>
//jspa1.jsp
<%@ page import="java.lang.*" %>
<html>
<body bgcolor="pink" text="red">
<%
try
{
   String wd=request.getParameter("txt");
   int no=Integer.parseInt(wd);
   int rem=0,rev=0,n;
```

```
        while(no!=0)
        {
            rem=no%10;
            no=no/10;
            rev=rev*10+rem;
        }
        while(rev!=0)
        {
            n=rev%10;
            rev=rev/10;
            switch(n)
            {
                case 0:out.println("Zero");
                       break;
                case 1:out.println("One");
                       break;
                case 2:out.println("Two");
                       break;
                case 3:out.println("Three");
                       break;
                case 4:out.println("Four");
                       break;
                case 5:out.println("Five");
                       break;
                case 6:out.println("Six");
                       break;
                case 7:out.println("Seven");
                       break;
                case 8:out.println("Eight");
                       break;
                case 9:out.println("Nine");
                       break;
            }
        }
    }
```

```
catch(Exception e)
{
   out.println("Please Enter No.");
}
%>
```

Output:

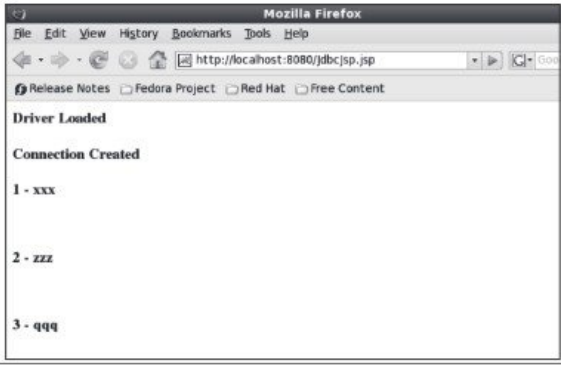


Program 4.21: Program to display the customer information like cust_no and cust_name with store in the table customer using JSP (JSP and PostgreSQL).

```
<%@page language="java"
import="java.io.*"
import="java.sql.*"%>
<html>
<body>
<%!
   Connection con;
   Statement st;
   ResultSet rs;
   %>
      <%
         try
         {
            Class.forName("postgresql.Driver");
```

```
            out.println("<h3>Driver Loaded</h3>");
            con=DriverManager.getConnection
                                ("jdbc:postgresql:TEST","postgres","");
            out.println("<h3>Connection Created</h3>");
            st=con.createStatement();
            rs=st.executeQuery("select * from customer");
            while(rs.next())
            {
               out.print("<h3>"+rs.getInt(1)
                                        +" - "+rs.getString(2)+"</h3>");
               out.println("<br>");
            }
         }
         catch(Exception e)
            {
               out.println("error"+e);
            }
            %>
      </body>
</html>
```

Output:



SUMMARY

- A Java Servlet is a server side program that services HTTP requests and returns the result as HTTP responses.
- Servlet API consists of two important packages that encapsulate all the important classes and interface, namely javax.servlet and javax.servlet.http.
- A Servlet is a java program that runs on web server. This program will start running when there is request from client side. A servlet is loaded by the servlet container the first time the Servlet is requested.
- There are two types of servlets: GenericServlet and HttpServlet. GenericServlet defines the generic or protocol independent servlet. HttpServlet is subclass of GenericServlet and provides some http specific functionality like doGet and doPost methods.
- All Servlets must implement a service() method, which is responsible for handling Servlet requests.
- For generic Servlets, simply override the service method to provide routines for handling requests.
- HTTP Servlets provide a service method that automatically routes the request to another method in the Servlet based on which HTTP transfer method is used. HTTP servlets, override doPost() to process POST requests, doGet() to process GET requests,
- Servlet life cycle has three methods: init(), service() and destroy().
- HTTP request can be having two methods GET and POST.
- The GET method sends the encoded user information appended to the page request.
- A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL it sends it as a separate message.
- Servlets handles form data parsing automatically using the methods: `getParameter()`, `getParameterValues()`, `getParameterNames()`.
- There are following techniques used in session tracking:
  1. User authorization.
  2. Cookies.
  3. Hidden Form Field.
  4. URL Rewriting.
  5. HttpSession.
- A cookie is a small piece of information that is passed back and forth in the HTTP request and response.
- The HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from HttpSession object.

- JavaServer Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.
- JSP life cycle has four phases: jspCompilation, jspInit, jspService, jspDestroy.
- The JSP page is built up using the following components: Directives, Declarations, Scriptlets, Expressions, Standard Actions, Custom Tags.

Check Your Understanding

1. Which object of HttpSession can be used to view and manipulate information about a session?
   (a) creation time (b) last accessed time
   (c) session identifier (d) All mentioned above
2. How constructor can be used for a servlet?
   (a) Initialization
   (b) Constructor function
   (c) Initialization and Constructor function
   (d) Setup() method
3. Which page directive should be used in JSP to generate a PDF page?
   (a) contentType (b) generatePdf
   (c) typePDF (d) contentPDF
4. In JSP Action tags which tags are used for bean development?
   (a) jsp:useBean (b) jsp:setPoperty
   (c) jsp:getProperty (d) All mentioned above
5. Which of the following is stored at client side?
   (a) URL rewriting (b) Hidden form fields
   (c) Cookies (d) Session
6. Which of the below is not a session tracking method?
   (a) URL rewriting (b) Cookies
   (c) HTTP (d) SSL sessions
7. Which of the following code is used to get an attribute in a HttpSession object in servlets?
   (a) session.alterAttribute(String name)
   (b) session.updateAttribute(String name)
   (c) session.setAttribute(String name)
   (d) session.getAttribute(String name)
8. Which technology do we mix our business logic with the presentation logic?
   (a) Servlet (b) JSP
   (c) Both (a) and (b) (d) ASP

9. Which one is the correct order of phases in JSP life cycle?
   (a) Initialization, Cleanup, Compilation, Execution
   (b) Initialization, Compilation, Cleanup, Execution
   (c) Compilation, Initialization, Execution, Cleanup
   (d) Cleanup, Compilation, Initialization, Execution
10. Which tag should be used to pass information from JSP to included JSP?
   (a) Using <%jsp:page> tag (b) Using <%jsp:useBean> tag
   (c) Using <%jsp:import> tag (d) Using <%jsp:param> tag
11. The Java _____ specification defines an application programming interface for communication between the Web server and the application program.
   (a) Servlet (b) Server
   (c) Program (d) Randomize
12. Which of the following techniques can be used for wireless session tracking?
   (a) Cookies (b) URL Rewriting
   (c) Hidden Fields (d) None of the above
13. A JSP is transformed into a(n):
   (a) Java applet (b) Java servlet
   (c) Either 1 or 2 above (d) Neither 1 nor 2 above
14. In Java servlet method init( ) is called _____ times.
   (a) 1 (b) 2
   (c) 0 (d) multiple
15. Can servlet class declare constructor with ServletConfig object as an argument?
   (a) True (b) False
16. What is the difference between servlets and applets?
   (i) Servlets execute on Server; Applets execute on browser
   (ii) Servlets have no GUI; Applet has GUI
   (iii) Servlets creates static web pages; Applets creates dynamic web pages
   (iv) Servlets can handle only a single request; Applet can handle multiple requests
   (a) (i), (ii), (iii) are correct (b) (i), (ii) are correct
   (c) (i), (iii) are correct (d) (i), (ii), (iii), (iv) are correct
17. Which of the following code retrieves the body of the request as binary data?
   (a) DataInputStream data = new InputStream()
   (b) DataInputStream data = response.getInputStream()
   (c) DataInputStream data = request.getInputStream()
   (d) DataInputStream data = request.fetchInputStream()

18. Which of the following is true about servlets?
    (a) Servlets execute within the address space of web server.
    (b) Servlets are platform-independent because they are written in java.
    (c) Servlets can use the full functionality of the Java class libraries.
    (d) Servlets execute within the address space of web server, platform independent and uses the functionality of java class libraries.
19. How is dynamic interception of requests and responses to transform the information done?
    (a) servlet container (b) servlet config
    (c) servlet context (d) servlet filter

Answers

1. (d)	2. (c)	3. (a)	4. (d)	5. (c)	6. (c)	7. (d)	8. (a)	9. (c)	10. (a)
11. (a)	12. (b)	13. (b)	14. (a)	15. (b)	16. (b)	17. (c)	18. (d)	19. (d)	

Practice Questions

Q.I Answer the following questions in short:

1. What is Servlet? Explain the servlet types.
2. Differentiate between Get and Post.
3. Explain Generic Servlet and HttpServlet.
4. Explain session tracking.
5. Explain types of cookies.
6. Differentiate between Session and cookie.
7. Explain URL rewriting.
8. Explain hidden form field.
9. What is scriptlet in JSP?

Q.II Answer the following questions:

1. What is a servlet? Explain the lifecycle of servlet in detail.
2. Explain HTTPRequest and HTTPResponse.
3. Explain the lifecycle of JSP in detail.
4. Why is JSP fly compilation? Explain implicit object in JSP?
5. State the purpose of implicit objects. Explain any four implicit objects.

Q.III Define the Terms:

1. Servlet
2. Session
3. Cookies
4. scriptlet

■■■

5... Spring and Hibernate

Learning Objectives ...

Students will be able:

- To understand Basic Concept of Spring.
- To learn Spring Architecture and MVC.
- To study different Applications of Spring.

5.1 INTRODUCTION

- Spring is an open source framework created to address the complexity of enterprise application development.
- Spring framework is an open source Java platform that provides comprehensive infrastructure support for developing robust Java applications very easily and very rapidly.
- Spring was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003.
- Spring Framework is an open source framework that we use to develop Java applications with very ease and with a rapid pace.
- Spring is a very lightweight framework which provides well-defined infrastructure support for developing Java application.
- Hibernate is an open-source, non-invasive, light-weight java ORM(Object-Relational Mapping) framework to develop objects which are independent of the database software and make independent persistence logic in all JAVA, JEE.
- Hibernate is a framework which provides some abstraction layer, means the programmer does not have to worry about the implementations, Hibernate does the implementations for you internally like Establishing a connection with the database, writing query to perform operations on the database.

5.2 APPLICATIONS AND BENEFITS OF SPRING

- The Spring Framework is one of the most popular Java-based application Frameworks. It is an application framework and Inversion of Control (IoC) container for the Java platform.
- The Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable, and reusable code.
- Fig. 5.1 shows the Spring Framework. The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building Web applications on top of the Java EE platform.
- The Spring Framework is a mature, powerful and highly flexible framework focused on building Web applications in Java.
- The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.
- The Spring Framework (Spring) is an open-source application framework that provides infrastructure support for developing Java applications.
- One of the most popular Java Enterprise Edition (Java EE) frameworks, Spring helps developers create high performing applications using Plain Old Java Objects (POJOs).

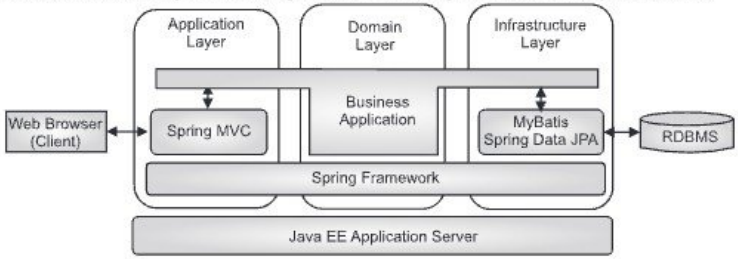


Fig. 5.1: Spring Framework

Advantages of Spring Framework:

- Advantages of Spring Framework are as follow:
  1. Spring Framework is lightweight in nature due to its POJO (Plain Old Java Object) implementation which does not need an enterprise container like an application server.
  2. Spring supports the use of both XML configuration as well as Java-based annotations for configuring the Spring Beans. Therefore, it provides the flexibility of using any of them for developing the application.

3. The Spring application is loosely couple due to Dependency Injection. Dependency Injection (or sometime called wiring) helps in gluing the classes in Java together and at the same time keeping them independent.
4. Spring provides a consistent transaction management interface that can scale down to a local transaction (for example, using a single database) and scale up to global transactions (for example, using JTA).
5. Spring Framework supports other frameworks and its integration makes Spring easier to develop.
6. Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBean style POJOs, it becomes easier and simpler to use dependency injection for injecting test data.
7. Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC) into consistent, unchecked exceptions.
8. The Spring Framework provides modularity to Java developers/programmers which helps them to choose which packages or classes can be used or ignored. With tons of classes and packages, it comes as a benefit to developers to identify and choose the packages or classes without any problems.
9. The Spring Framework AOP (Aspect Oriented Programming) module allows a Java developer/programmer to have different compilation unit or separate class loader. Along with that, it uses IoC (Inversion of Control) for dependency injection which allows aspects to be configured normally.
10. Spring is lightweight, loosely coupled and open source.

Disadvantages of Spring Framework:

- Disadvantages of Spring Framework are as follow:
  1. Developing a Spring application requires lots of XML coding.
  2. The learning curve for Spring Framework is very high as most developers find it hard to understand and apply.
  3. For many its time-consuming process as Spring Framework has lots of integration with another framework due to which it is hard to know all the options which are available.
  4. The nature of Spring Framework keeps changing over the time which makes it harder to grasp, (for example, the annotation-based Spring).
  5. No clear guidance in Spring Framework on cross-site scripting attacks and cross-site request attacks in Spring MVC documentation. Also, it suffers from a several security holes.

5.2.1 Spring Applications

- Following is the list of few of the great benefits of using Spring Framework:
  1. **POJO Based:** Spring enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product such as an application server but you have the option of using only a robust Servlet container such as Tomcat or some commercial product.
  2. **Modular:** Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about the ones you need and ignore the rest.
  3. **Integration with existing Frameworks:** Spring does not reinvent the wheel, instead it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, and other view technologies.
  4. **Testablity:** Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBeanstyle POJOs, it becomes easier to use dependency injection for injecting test data.
  5. **Web MVC:** Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over-engineered or less popular web frameworks.
  6. **Central Exception Handling:** Spring provides a convenient API to translate technology-specific exceptions (for example, thrown by JDBC, Hibernate, or JDO) into consistent, unchecked exceptions.
  7. **Lightweight:** Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
  8. **Transaction Management:** Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

5.3 SPRING ARCHITECTURE AND ENVIRONMENTAL SETUP

5.3.1 Spring Architecture

- The Fig. 5.2 shows architecture of Spring Framework with its important modules. The modules of Spring Framework are organized into multiple containers like Core, Web and Data.
- The basic idea behind the development of Spring Framework was to make it a one-stop shop where we can integrate and use the modules according to the need/requirement of the application.

- Spring is modular, allowing us to pick and choose which modules are applicable to an application, without having to bring in the rest.
- The core container consists of **core, beans, context and expression** language modules that are fundamental to any Spring application.
- The **core** module is the core of the component model; it provides fundamental features like dependency injection.
- The **Bean** module provides Bean Factory, which is a sophisticated implementation of the factory pattern. The bean module is useful for the instantiation of the beans and is used extensively by the Spring component framework.
- The **context** module is used on top of the core and bean modules for initializing the context for the beans. The context module plays an important role in loading the components defined and configured for the application.
- The Spring 3.0 introduces a new expression language – Spring Expression Language (SpEL). It is a powerful expression language based on Java Server Pages (JSP) Expression Language (EL). The **EL module** is used for dynamic querying of components.
- The web container is built on top of the core container. The web and web servlet modules that are part of the web container are useful for building the web components for the enterprise applications using the spring framework. The web module provides the basic features of a web application and the web servlet module provides an implementation of the spring MVC architecture for web applications.
- The Web-Portlet module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.
- Modules in the data layer help to provide data access from the web and business components. The data layer container various modules is listed below:
  - **JDBC module:** This module provides the JDBC abstraction layer and helps to avoid tedious JDBC coding.
  - **ORM module:** It provides integration for object relational mapping APIs such as JPA, Hibernate, JDO, etc.
  - **JMS (Java Messaging Service) Module:** It contains features for producing and consuming messages.
  - **OXM module:** This module provides Object/XML binding.
  - **Transaction Module:** This model supports programmatic and declarative transaction management for classes that implement special interfaces and for all the POJOs.

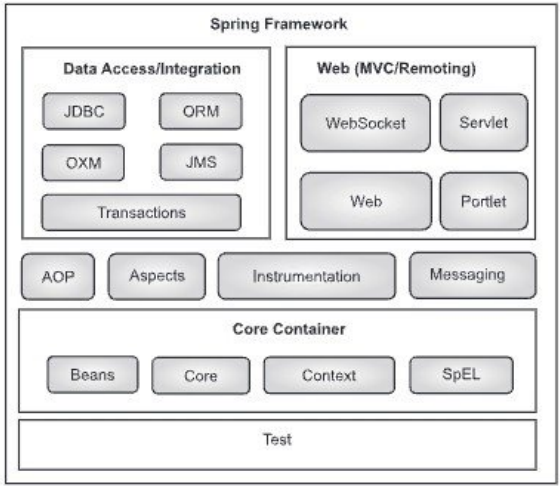


Fig. 5.2: Architecture of Spring Framework with Modules

- There are few other important modules which plays vital role in the Spring Framework namely, AOP, Aspects, Instrumentation, Web and Test modules. These modules are explained as follows:
  1. The **AOP module** provides an aspect-oriented programming implementation allowing us to define method-interceptors and point-cuts to cleanly decouple code that implements functionality that should be separated.
  2. The **Aspects module** provides integration with AspectJ, which is again a powerful and mature AOP framework.
  3. The **Instrumentation module** provides class instrumentation support and class loader implementations to be used in certain application servers.
  4. The **Messaging module** provides support for STOMP as the WebSocket sub-protocol to use in applications. It also supports an annotation programming model for routing and processing STOMP messages from WebSocket clients.
  5. The **Test module** supports the testing of Spring components with JUnit or TestNG frameworks.

5.3.2 Environmental Setup

Step 1 : Setup Java Development Kit (JDK)

Install jdk and set the path.

To setup environment variables, right-click on My Computer, select Properties → Advanced → Environment Variables. Then, you will have to update the PATH value and click the OK button.

Step 2 : Install Apache Common Logging API

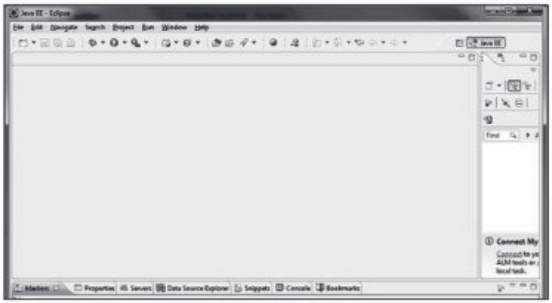
To install Apache API, download the latest version of Apache Commons Logging API from https://commons.apache.org/logging. After installation, unpack the binary distribution into a convenient location. This directory will have the following jar files and other supporting documents, etc.

- commons-logging-1.1.1
- spring-aop-4.1.6.RELEASE
- spring-aspects-4.1.6.RELEASE
- spring-beans-4.1.6.RELEASE
- spring-context-4.1.6.RELEASE
- spring-context-support-4.1.6.RELEASE
- spring-core-4.1.6.RELEASE
- spring-expression-4.1.6.RELEASE
- spring-instrument-4.1.6.RELEASE
- spring-instrument-tomcat-4.1.6.RELEASE
- spring-jdbc-4.1.6.RELEASE
- spring-jms-4.1.6.RELEASE
- spring-messaging-4.1.6.RELEASE
- spring-orm-4.1.6.RELEASE
- spring-oxm-4.1.6.RELEASE
- spring-test-4.1.6.RELEASE
- spring-tx-4.1.6.RELEASE
- spring-web-4.1.6.RELEASE
- spring-webmvc-4.1.6.RELEASE
- spring-webmvc-portlet-4.1.6.RELEASE
- spring-websocket-4.1.6.RELEASE

Set your CLASSPATH variable on this directory properly.

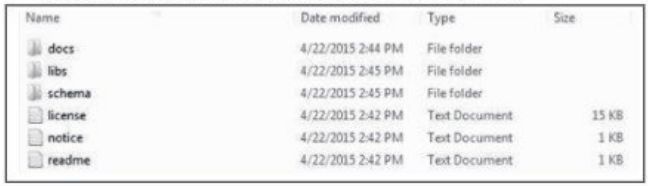
Step 3 : Setup Eclipse IDE

To install Eclipse IDE, download the latest Eclipse binaries from https://www.eclipse.org/downloads. After successful installation, the following window will display.



Step 4 : Setup Spring Framework Libraries

- First download .zip file for Windows
- Download the latest version of Spring framework binaries from https://repo.spring.io/release/org/springframework/spring and spring-framework-4.1.6.RELEASE-dist.zip was downloaded on Windows machine.
- It gives the following directory structure inside E:\spring. Now your setup is ready and you can start with your first example.



Name	Date modified	Type	Size
docs	4/22/2015 2:44 PM	File folder	
libs	4/22/2015 2:45 PM	File folder	
schema	4/22/2015 2:45 PM	File folder	
license	4/22/2015 2:42 PM	Text Document	15 KB
notice	4/22/2015 2:42 PM	Text Document	1 KB
readme	4/22/2015 2:42 PM	Text Document	1 KB

5.4 HELLO WORLD EXAMPLE

- Let us start with simple example in Spring Framework with following steps:

Step 1 : **Create Java Project:** The first step is to create a simple Java Project using Eclipse IDE. Go to the option File → New → Project and then select Java Project wizard from the wizard list. Now name the project as HelloSpringProject using the window as shown in Fig. 5.3.

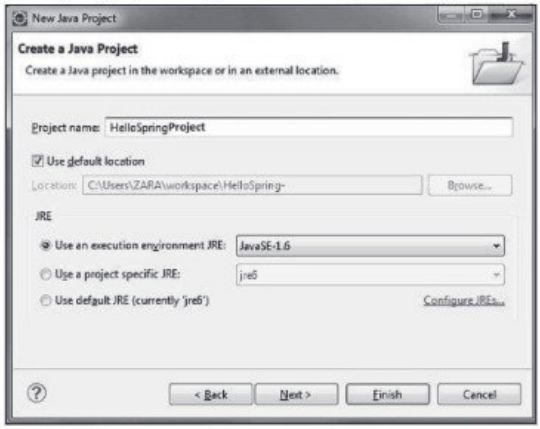


Fig. 5.3

Then the project is created successfully, the following content in the Project Explorer will be displayed as shown in Fig. 5.4.

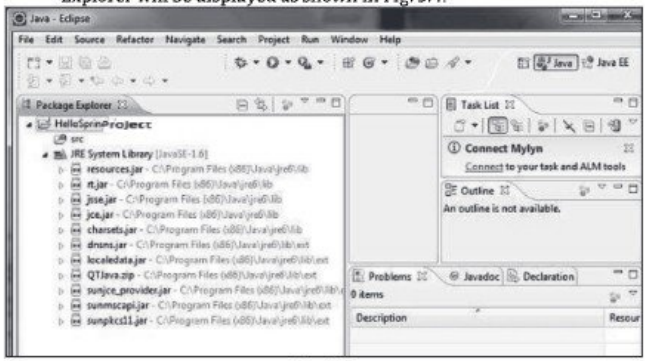


Fig. 5.4

Step 2 : Let us add Spring Framework and common logging API libraries in our project. To do this, right-click on the project name HelloSpringProject and then follow the following option available in the context menu – Build Path → Configure Build Path to display the Java Build Path window as follows – as shown in Fig. 5.5 to add Required Libraries.

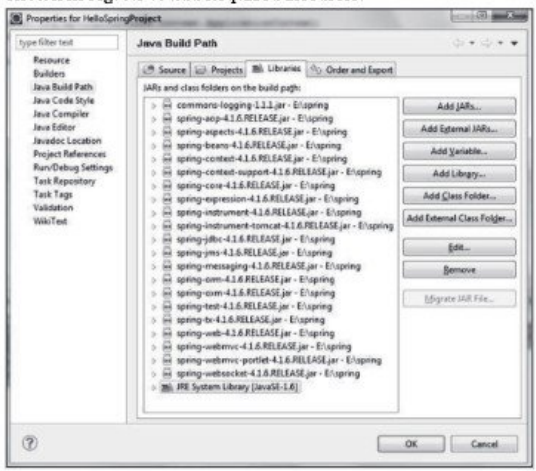


Fig. 5.5

Now use Add External JARs button available under the Libraries tab to add the following core JARs from Spring Framework and Common Logging installation directories:

- commons-logging-1.1.1
- spring-aop-4.1.6.RELEASE
- spring-aspects-4.1.6.RELEASE
- spring-beans-4.1.6.RELEASE
- spring-context-4.1.6.RELEASE
- spring-context-support-4.1.6.RELEASE
- spring-core-4.1.6.RELEASE
- spring-expression-4.1.6.RELEASE
- spring-instrument-4.1.6.RELEASE
- spring-instrument-tomcat-4.1.6.RELEASE
- spring-jdbc-4.1.6.RELEASE
- spring-jms-4.1.6.RELEASE
- spring-messaging-4.1.6.RELEASE
- spring-orm-4.1.6.RELEASE
- spring-oxm-4.1.6.RELEASE
- spring-test-4.1.6.RELEASE
- spring-tx-4.1.6.RELEASE
- spring-web-4.1.6.RELEASE
- spring-webmvc-4.1.6.RELEASE
- spring-webmvc-portlet-4.1.6.RELEASE
- spring-websocket-4.1.6.RELEASE

Step 3 : Now create actual source files under the HelloSpringProject. First we need to create a package called com.TYBBA. To do this, right click on src in package explorer section and follow the option – New → Package.

Next we will create HelloWorld.java and MainApp.java files under the com.

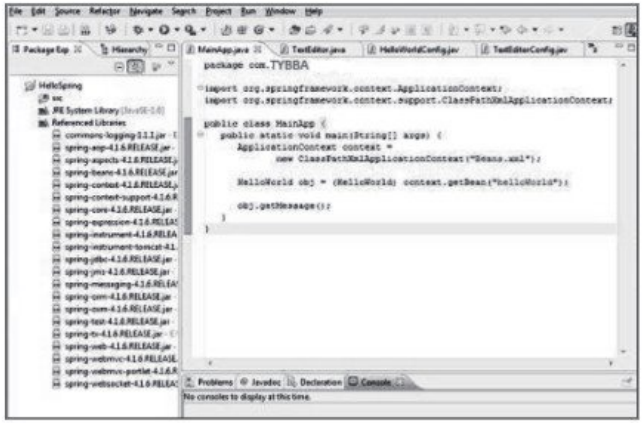


Fig. 5.6

The HelloWorld.java file is as follows:

```
package com.TYBBA;
public class HelloWorld
{
     private String msg;
     public void setMessage(String msg)
     {
        this.msg = msg;
     }
     public void getMessage()
     {
        System.out.println("Your Message : " + msg);
     }
}
```

Following is the content of the second file MainApp.java –

```
package com.TYBBA;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
                                        ClassPathXmlApplicationContext;
```

```
public class MainApp
{
    public static void main(String[] args)
    {
        ApplicationContext context = new
                        ClassPathXmlApplicationContext("Beans.xml");
        HelloWorld obj = (HelloWorld)
                                        context.getBean("helloWorld");
        obj.getMessage();
    }
}
```

In the main program firstly, the framework API ClassPathXmlApplicationContext() was used to create an application context. It loads beans configuration file and it takes care of creating and initializing all the objects, i.e. beans. Secondly, getBean() method is used to get the bean. This method uses bean ID to return a generic object, which finally can be casted to the actual object. Once you have an object, you can use this object to call any class method.

Step 4 : Next is to create a Bean Configuration file which is an XML file. This file needs to be created under the src directory as shown in the Fig. 5.7.

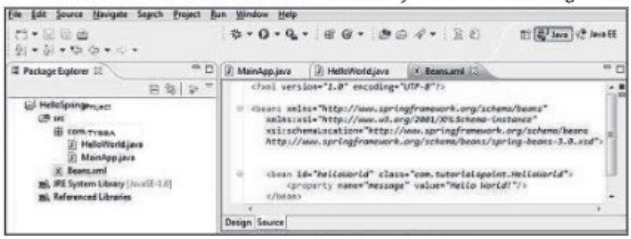


Fig. 5.7

The name of the file is Beans.xml but we can change the name. We have to make sure that this file is available in CLASSPATH and use the same name in the main application while creating an application context as shown in MainApp.java file.

The Beans.xml is used to assign unique IDs to different beans and to control the creation of objects with different values without impacting any of the Spring source files. For example: we can pass any value for "msg" variable and we can print different values of message without impacting

HelloWorld.java and MainApp.java files using the following file. We can use <property> tag to pass the values of different variables used at the time of object creation.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation =
                        "http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans
                                                        3.0.xsd">
   <bean id = "helloWorld" class = "com.TYBBA.HelloWorld">
      <property name = "msg" value = "Hello World 2022!"/>
   </bean>
</beans>
```

Step 5 : Once the source and beans configuration files are created, we can compile and run the program. To do this, keep MainApp.Java file tab active and use either Run option available in the Eclipse IDE or use Ctrl + F11 to compile and run the MainApp application. The output will print the following message in Eclipse IDE's console:

```
Your Message : Hello World 2022!
```

5.5 CORE SPRING-IOC CONTAINERS, SPRING BEAN DEFINITION, SCOPE

5.5.1 IoC Containers

- The Spring IoC container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The Spring container uses Dependency Injection (DI) to manage the components that make up an application.
- An IoC container is a common characteristic of frameworks that implement IoC. In the Spring framework, the interface ApplicationContext represents the IoC container. The Spring container is responsible for instantiating, configuring and assembling objects known as beans, as well as managing their life cycles.
- IoC is also known as Dependency Injection (DI). It is a process whereby objects define their dependencies, that is, the other objects they work with, only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method.
- The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.

- Spring provides two types of containers:
  1. BeanFactory container
  2. ApplicationContext container
- The org.springframework.beans and org.springframework.context packages are the basis for Spring Framework's IoC container.
- The BeanFactory provides the configuration framework and basic functionality, and the ApplicationContext adds more enterprise-specific functionality. The ApplicationContext is a complete superset of the BeanFactory, and is used exclusively in this chapter in descriptions of Spring's IoC container.
- The interface org.springframework.context.ApplicationContext represents the Spring IoC container and is responsible for instantiating, configuring, and assembling the aforementioned beans. The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata.
- The Fig. 5.8 shows the spring IOC container. The Spring IoC container consumes a form of configuration metadata; this configuration metadata represents how can you as an application developer tell the Spring container to instantiate, configure, and assemble the objects in your application.

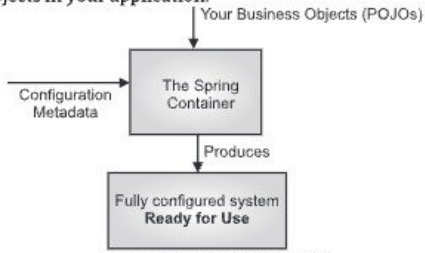


Fig. 5.8: Spring IOC container

- The ApplicationContext is the interface for an advanced factory capable of maintaining a registry of different beans and their dependencies. Using the method T getBean(String name, Class<T> requiredType) you can retrieve instances of your beans.
- The ApplicationContext enables you to read bean definitions and access them as follows:

```
// create and configure beans
ApplicationContext context = new ClassPathXmlApplicationContext(new String[]
                                                {"services.xml", "daos.xml"});
// retrieve configured instance
PetStoreServiceImpl service = context.getBean
                                ("petStore", PetStoreServiceImpl.class);
// use configured instance
List userList = service.getUsernameList();
```

- Spring configuration consists of at least one and typically more than one bean definition that the container must manage. XML-based configuration metadata shows these beans configured as <bean/> elements inside a top-level <beans/> element.
- These bean definitions correspond to the actual objects that make up your application. Typically you define service layer objects, data access objects (DAOs), presentation objects such as Struts Action instances, infrastructure objects such as Hibernate SessionFactories, JMS Queues, and so forth.

5.5.2 Spring Bean Definition

- A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.
- A Spring IoC container manages one or more beans. These beans are created with the configuration metadata that you supply to the container, for example, in the form of XML <bean/> definitions.
- Bean definition contains the information called configuration metadata, which is needed for the container to know the following:
  - How to create a bean
  - Bean's life cycle details
  - Bean's dependencies
- Within the container itself, these bean definitions are represented as BeanDefinition objects, which contain (among other information) the following metadata:
  - A package-qualified class name: typically the actual implementation class of the bean being defined.
  - Bean behavioral configuration elements, which state how the bean should behave in the container (scope, life cycle callbacks, and so forth).
  - References to other beans that are needed for the bean to do its work; these references are also called collaborators or dependencies.
- Other configuration settings to set in the newly created object, for example, the number of connections to use in a bean that manages a connection pool, or the size limit of the pool.
- This metadata translates to a set of properties that make up each bean definition.
- The following table shows a set of the following properties that make up each bean definition from the configuration metadata.

Sr. No.	Property	Description
1.	class	This attribute is mandatory and specifies the bean class to be used to create the bean.
2.	name	This attribute specifies the bean identifier uniquely. In XML based configuration metadata, the id and/or name attributes used to specify the bean identifier(s).
3.	scope	This attribute specifies the scope of the objects created from a particular bean definition.
4.	constructor-arg	This is used to inject the dependencies.
5.	properties	This is used to inject the dependencies.
6.	autowiring mode	This is used to inject the dependencies.
7.	initialization method	A callback to be called just after all necessary properties on the bean has been set by the container.
8.	lazy-initialization mode	A lazy-initialized bean tells the IoC container to create a bean instance when it is first requested, rather than at the startup.
9.	destruction method	A callback to be used when the container containing the bean is destroyed.

5.5.3 Bean Scope

- When you create a bean definition what you are actually creating instances of the class defined by that bean definition.
- You can control not only the various dependencies and configuration values that are to be plugged into an object that is created from a particular bean definition, but also the scope of the objects created from a particular bean definition.
- The Spring Framework supports the following five scopes, three of which are available only if you use a web-aware ApplicationContext:

Scope	Description
singleton	Scopes a single bean definition to a single object instance per Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	Scopes a single bean definition to the life cycle of a single HTTP request; that is each and every HTTP request will have its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext.
session	Scopes a single bean definition to the life cycle of a HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext.
global session	Scopes a single bean definition to the life cycle of a global HTTP Session. Typically only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext.

- Here we will discuss two scopes:

1. **The singleton scope:**
   - When a bean is a singleton, only one shared instance of the bean will be managed, and all requests for beans with an id or id's matching that bean definition will result in that one specific bean instance being returned by the Spring container.
   - The default scope is always singleton.
   - When you define a bean definition and it is scoped as a singleton, then the Spring IoC container will create exactly one instance of the object defined by that bean definition. This single instance will be stored in a cache of such singleton beans, and all subsequent requests and references for that named bean will result in the cached object being returned.
   - The snippet of the Bean configuration file is as follows:

```
<!-- A bean definition with singleton scope -->
<bean id = "..." class = "..." scope = "singleton">
   <!-- collaborators and configuration for this bean go here -->
</bean>
```

2. **The prototype scope:**
   - The non-singleton, prototype scope of bean deployment results in the creation of a new bean instance every time a request for that specific bean is made (that is, it is injected into another bean or it is requested via a programmatic `getBean()` method call on the container). As a rule of thumb, you should use the prototype scope for all beans that are stateful, while the singleton scope should be used for stateless beans.
   - The snippet of the Bean configuration file is as follows:

```
<!-- A bean definition with prototype scope -->
<bean id = "..." class = "..." scope = "prototype">
   <!-- collaborators and configuration for this bean go here -->
</bean>
```

5.5.4 Life Cycle of Bean

- A Spring bean needs to be instantiated when the container starts, based on Java or XML bean definition.
- When a bean is instantiated, it may be required to perform some initialization to get it into a usable state. Similarly, when the bean is no longer required and is removed from the container, some cleanup may be required.
- The Spring bean factory is responsible for managing the life cycle callbacks of the beans which are created in the spring containers.
- Spring bean factory controls the creation and destruction of beans. To execute some custom code, the bean factory provides two callback methods:
  1. **initmethod:** The init-method attribute specifies a method that is to be called on the bean immediately upon instantiation.

2. **destroy-method:** It specifies a method that is called just before a bean is removed from the container.

- Spring framework provides the following four ways for controlling life cycle events of a bean:
  1. InitializingBean and DisposableBean callback interfaces.
  2. *Aware interfaces for specific behavior.
  3. Custom init() and destroy() methods in bean configuration file.
  4. @PostConstruct and @PreDestroy annotations.

Initialization callbacks:

- We can simply implement the InitializingBean interface and initialization work can be done inside afterPropertiesSet() method as follows:

```
public class demoBean implements InitializingBean
{
   public void afterPropertiesSet()
   {
      // do some initialization work
   }
}
```

- In the case of XML-based configuration metadata, you can use init-method attribute to specify the name of the method that has a void no-argument signature. For example:

```
<bean id = "demoBean" class = "examples.demoBean" init-method = "init"/>
```

- Following is the class definition:

```
public class demoBean
{
   public void init()
   {
      // do some initialization work
   }
}
```

Destruction callbacks:

- The interface DisposableBean is used to specifies a single method:

```
void destroy() throws Exception;
```

- The finalization work can be done inside destroy() method as follows:

```
public class demoBean implements DisposableBean
{
   public void destroy()
   {
      // do some destruction work
   }
}
```

- In the case of XML-based configuration metadata, you can use the destroy-method attribute to specify the name of the method that has a void no-argument signature. For example:

```
<bean id = "demoBean" class = "examples.demoBean" destroy-method = "destroy"/>
```

- Following is the class definition:

```
public class demoBean
{
   public void destroy()
   {
     // do some destruction work
   }
}
```

- A sample bean implementing the above interfaces would look like this:

demoBean.java

```
package com.TYBBA;
import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;
public class demoBean implements InitializingBean, DisposableBean
{
   //Other bean attributes and methods

   @Override
   public void afterPropertiesSet() throws Exception
   {
      //Bean initialization code
   }

   @Override
   public void destroy() throws Exception
   {
      //Bean destruction code
   }
}
```

5.6 HIBERNATE

- Hibernate is a framework which is used to develop persistence logic which is independent of Database software. In JDBC to develop persistence logic we deal with primitive types. Whereas Hibernate framework we use Objects to develop persistence logic which are independent of database software.

- Hibernate is an Object-Relational Mapping (ORM) solution for JAVA. It is an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.

Hibernate Advantages:

- Hibernate framework is open source under the LGPL license and lightweight.
- Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
- It provides simple querying of data.
- It manipulates Complex associations of objects of your database.
- It minimizes database access with smart fetching strategies.
- It provides simple APIs for storing and retrieving Java objects directly to and from the database.
- If there is change in the database or in any table, then you need to change the XML file properties only.
- Hibernate does not require an application server to operate.

5.6.1 Hibernate Architecture

- The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.
- The Hibernate architecture is categorized in four layers.
  - Java application layer
  - Hibernate framework layer
  - Backhand API layer
  - Database layer

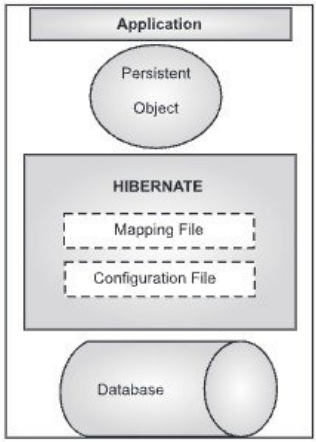


Fig. 5.9: Hibernate Architecture

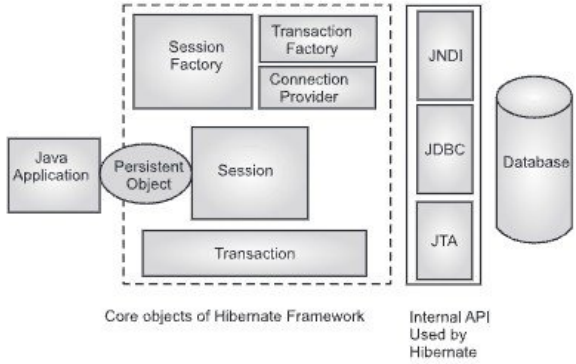


Fig. 5.10: Hibernate Architecture detail view with core classes

- Hibernate uses many objects such as session factory, session, transaction etc. along with various Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI). JDBC provides a level of abstraction of functionality common to relational databases, allowing almost any database with a JDBC driver to be supported by Hibernate. JNDI and JTA allow Hibernate to be integrated with J2EE application servers.

Hibernate Architecture Elements:

- For creating the first hibernate application, we must know the elements of Hibernate architecture. They are as follows:

SessionFactory:

- The SessionFactory is a heavyweight object; it is usually created during application start up and kept for later use. You would need one SessionFactory object per database using a separate configuration file. So, if you are using multiple databases, then you would have to create multiple SessionFactory objects.
- The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

Session:

- The session object provides an interface between the application and data stored in the database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

Transaction:

- The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

ConnectionProvider:

- It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

TransactionFactory:

- It is a factory of Transaction. It is optional.

5.6.2 Hibernate Environment

1. First Download the latest version of Hibernate from http://www.hibernate.org/downloads
2. Set classpath properly by copying all the library files from /lib into your CLASSPATH, and change your classpath variable to include all the JARs and then copy hibernate3.jar file into your CLASSPATH. This file lies in the root directory of the installation.
3. Install the required packages as listed below:
   - dom4j
   - Xalan
   - Xerces
   - Cglib
   - log4j
   - Commons
   - SLF4J

5.7 HIBERNATE CONFIGURATION

- Hibernate configurations contain the mapping information that provides different functionalities to Java classes. We generally provide database related mappings in the configuration file. Hibernate facilitates to provide the configurations with standard Java properties file called hibernate.properties, or as an XML file named hibernate.cfg.xml. The file hibernate.cfg.xml is kept in the root directory of your application's classpath.
- An instance of Configuration class allows specifying properties and mappings to applications. This class also builds an immutable SessionFactory.

- The properties are listed below:

Sr. No.	Property	Description
1.	hibernate.dialect	It makes Hibernate generate the appropriate SQL for the chosen database.
2.	hibernate.connection.driver_class	The JDBC driver class.
3.	hibernate.connection.url	The JDBC URL to the database instance.
4.	hibernate.connection.username	The database username.
5.	hibernate.connection.password	The database password.
6.	hibernate.connection.autocommit	Allows autocommit mode to be used for the JDBC connection.
7.	hibernate.connection.pool_size	Limits the number of connections waiting in the Hibernate database connection pool.

- The following are the properties to be configure if database along with an application server and JNDI is used:

Sr. No.	Property	Description
1.	hibernate.connection.datasource	The JNDI name defined in the application server context.
2.	hibernate.jndi.class	The InitialContext class for JNDI.
3.	hibernate.jndi.url	Provides the URL for JNDI.
4.	hibernate.jndi.<JNDIpropertyname>	Passes any JNDI property to the JNDI InitialContext.
5.	hibernate.connection.username	The database username.
6.	hibernate.connection.password	The database password.

- We can acquire the Configuration class instance by instantiating it directly.
- Configuration cfg = new Configuration(.addResource("employee.hbm.xml")
- XML Based Configuration using MySql database

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-5.3.dtd">
<hibernate-configuration>
    <session-factory>
```

```
        <property name="hibernate.dialect">update</property>
            <property name="dialect">org.hibernate.dialect.MySQLDialect
    </property>
            <property name="connection.url">com.mysql.jdbc.Driver</property>
            <property name="connection.username">system</property>
            <property name="connection.password">jtp</property>
            <property name="connection.driver_class">com.mysql.jdbc.Driver
    </property>
       <!-- List of XML mapping files -->
          <mapping resource = "employee.hbm.xml"/>
        </session-factory>
    </hibernate-configuration>
```

Note: For plsql databases dialect property type is:org.hibernate.dialect.PostgreSQLDialect

5.7.1 Hibernate Sessions

- A Session is used to get a physical connection with a database.
- The life cycle of a Session is bounded by the beginning and end of a logical transaction.
- The main function of the Session is to offer create, read and delete operations for instances of mapped entity classes. Instances may exist in one of three states:
  - **transient:** never persistent, not associated with any Session.
  - **persistent:** associated with a unique Session.
  - **detached:** previously persistent, not associated with any Session.
- The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.
- Transient instances may be made persistent by calling `save()`, `persist()` or `saveOrUpdate()`.
- Persistent instances may be made transient by calling delete(). Any instance returned by a `get()` or `load()` method is persistent.
- Detached instances may be made persistent by calling `update()`, `saveOrUpdate()`, `lock()` or `replicate()`. The state of a transient or detached instance may also be made persistent as a new persistent instance by calling `merge()`.
- `save()` and `persist()` result in an `SQL INSERT`, `delete()` in an `SQL DELETE` and `update()` or `merge()` in an `SQL UPDATE`. Changes to persistent instances are detected at flush time and also result in an `SQL UPDATE`. `saveOrUpdate()` and `replicate()` result in either an `INSERT` or an `UPDATE`.
- If the Session throws an exception, the transaction must be rolled back and the session discarded.

- A typical transaction should use the following code:

```
Session s = factory.openSession();
Transaction tx;
try
{
   tx = s.beginTransaction();
   //do some work
   ...
   tx.commit();
}
catch (Exception e)
{
   if (tx!=null) tx.rollback();
   throw e;
}
finally
{
   s.close();
}
```

Session interface methods:

Method	Description
`Transaction beginTransaction()`	Begin a unit of work and return the associated Transaction object.
`void cancelQuery()`	Cancel the execution of the current query.
`void clear()`	Completely clear the session.
`Criteria createCriteria(Class persistentClass)`	Create a new Criteria instance, for the given entity class, or a superclass of an entity class.
`Criteria createCriteria(String entityName)`	Create a new Criteria instance, for the given entity name.
`Connection close()`	End the session by releasing the JDBC connection and cleaning up.
`Serializable getIdentifier(Object object)`	Return the identifier value of the given entity as associated with this session.
`Query createFilter(Object collection, String queryString)`	Create a new instance of Query for the given collection and filter string.
`SQLQuery createSQLQuery(String queryString)`	Create a new instance of SQLQuery for the given SQL query string.

contd. ...

`Query createQuery(String queryString)`	Create a new instance of Query for the given HQL query string.
`void delete(Object object)`	Remove a persistent instance from the datastore.
`Session get(String entityName, Serializable id)`	Return the persistent instance of the given named entity with the given identifier, or null if there is no such persistent instance.
`SessionFactory getSessionFactory()`	Get the session factory which created this session.
`void refresh(Object object)`	Re-read the state of the given instance from the underlying database.
`Transaction getTransaction()`	Get the Transaction instance associated with this session.
`boolean isConnected()`	Check if the session is currently connected.
`boolean isOpen()`	Check if the session is still open.
`void update(Object object)`	Update the persistent instance with the identifier of the given detached instance.
`void saveOrUpdate(Object object)`	Either save(Object) or update(Object) the given instance.
`Serializable save(Object object)`	Persist the given transient instance, first assigning a generated identifier.

5.7.2 Hibernate - Persistent Class

- Java classes whose objects or instances will be stored in database tables are called persistent classes in Hibernate.
- Hibernate works best if these classes follow some simple rules, also known as the Plain Old Java Object (POJO) programming model. However, none of these rules are hard requirements. Indeed, Hibernate3 assumes very little about the nature of your persistent objects.
- The following rules are the main rules of persistent classes:
  - All Java classes that will be persisted need a default constructor.
  - All classes should contain an ID in order to allow easy identification of your objects within Hibernate and the database. This property maps to the primary key column of a database table.
  - All attributes that will be persisted should be declared private and have getXXX and setXXX methods defined in the JavaBean style.

- A central feature of Hibernate, proxies, depends upon the persistent class being either non-final, or the implementation of an interface that declares all public methods.
- All classes that do not extend or implement some specialized classes and interfaces required by the EJB framework.

Simple POJO Example:

- Based on the few rules mentioned above, we can define a POJO class as follows:

```
public class Employee
{
   private int id;
   private String fstName;
   private String lstName;
   private int salary;
   private int age;
   public Employee() {}
   public Employee(String fname, String lname, int sal, int age)
   {
      this.fstName = fname;
      this.lstName = lname;
      this.salary = sal;
      this.age=age;
   }
   public int getId()
   {
      return id;
   }
   public void setId( int id )
   {
      this.id = id;
   }
   public String getFstName()
   {
      return fstName;
   }
   public void setFstName( String first_name )
   {
      this.fstName = first_name;
   }
   public String getLstName()
   {
      return lstName;
   }
```

```
    public void setLstName( String last_name )
    {
       this.lstName = last_name;
    }
    public int getSal()
    {
       return salary;
    }
    public void setSal( int sal )
    {
       this.salary = sal;
    }
    public int getAge()
    {
       return age;
    }
    public void setAge(int age)
    {
       this.age = age;
    }
}
```

Program 5.X

5.8 HIBERNATE - MAPPING FILES

- An Object/relational mappings are usually defined in an XML document but hibernate uses number of tools to generate the mapping document. These include XDoclet, Middlegen and AndroMDA for the advanced Hibernate users.
- Consider the above program. Let us say the above objects need to be stored and retrieved into the following RDBMS table:

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   sal    INT  default NULL,
   age    INT default NULL,
   PRIMARY KEY (id)
);
```

- The following mapping file is defined which instructs Hibernate how to map the defined class or classes to the database tables.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name = "Employee" table = "EMPLOYEE">
      <meta attribute = "class-description">
         This class contains the employee detail.
      </meta>
      <id name = "id" type = "int" column = "id">
         <generator class="native"/>
      </id>
      <property name = "fstName" column = "first_name" type = "string"/>
      <property name = "lstName" column = "last_name" type = "string"/>
      <property name = "salary" column = "sal" type = "int"/>
      <property name = "age" column = "age" type = "int"/>
   </class>
</hibernate-mapping>
```

- Now save the mapping document in a file with the format <classname>.hbm.xml. We saved our mapping document in the file Employee.hbm.xml.
- The mapping elements used in the mapping file as follows:
  - The mapping document is an XML document having <hibernate-mapping> as the root element, which contains all the <class> elements.
  - The <class> elements are used to define specific mappings from a Java classes to the database tables. The Java class name is specified using the name attribute of the class element and the database table name is specified using the table attribute.
  - The <meta> element is optional element and can be used to create the class description.
  - The <id> element maps the unique ID attribute in class to the primary key of the database table. The name attribute of the id element refers to the property in the class and the column attribute refers to the column in the database table. The type attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.

- The <generator> element within the id element is used to generate the primary key values automatically. The class attribute of the generator element is set to native to let hibernate pick up either identity, sequence, or hilo algorithm to create primary key depending upon the capabilities of the underlying database.
- The <property> element is used to map a Java class property to a column in the database table. The name attribute of the element refers to the property in the class and the column attribute refers to the column in the database table. The type attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.

5.8.1 Hibernate - Mapping Types

- The types declared and used in the mapping files are not Java data types; they are not SQL database types either. These types are called Hibernate mapping types, which can translate from Java to SQL data types and vice versa.

Mapping type	Java type	ANSI SQL Type
Integer	int or java.lang.Integer	INTEGER
Long	long or java.lang.Long	BIGINT
Short	short or java.lang.Short	SMALLINT
Float	float or java.lang.Float	FLOAT
Double	double or java.lang.Double	DOUBLE
big_decimal	java.math.BigDecimal	NUMERIC
character	java.lang.String	CHAR(1)
String	java.lang.String	VARCHAR
Byte	byte or java.lang.Byte	TINYINT
boolean	boolean or java.lang.Boolean	BIT
yes/no	boolean or java.lang.Boolean	CHAR(1) ('Y' or 'N')
true/false	boolean or java.lang.Boolean	CHAR(1) ('T' or 'F')

Date and time type:

Mapping type	Java type	ANSI SQL Type
Date	java.util.Date or java.sql.Date	DATE
Time	java.util.Date or java.sql.Time	TIME
timestamp	java.util.Date or java.sql.Timestamp	TIMESTAMP
calendar	java.util.Calendar	TIMESTAMP
calendar_date	java.util.Calendar	DATE

5.9 HIBERNATE EXAMPLE

- A simple example of hibernate application using eclipse IDE. For creating the first hibernate application in Eclipse IDE, we need to follow the following steps:
  1. Create the java project.
  2. Add jar files for hibernate.
  3. Create the Persistent class.
  4. Create the mapping file for Persistent class.
  5. Create the Configuration file.
  6. Create the class that retrieves or stores the persistent object.
  7. Run the application.

Create Application Class:

- Finally, we will create our application class with the main() method to run the application. We will use this application to save few Employee's records and then we will apply CRUD operations on those records.

```
import java.util.List;
import java.util.Date;
import java.util.Iterator;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class ManageEmployee
{
   private static SessionFactory factory;
   public static void main(String[] args)
   {
      try
      {
         factory = new Configuration().configure().buildSessionFactory();
      }
      catch (Throwable ex)
      {
         System.err.println("Failed to create sessionFactory object." + ex);
         throw new ExceptionInInitializerError(ex);
      }
      ManageEmployee ME = new ManageEmployee();

      /* Add few employee records in database */
      Integer empID1 = ME.addEmployee("Ashu", "Joshi", 2000,37);
```

```
        Integer empID2 = ME.addEmployee("Lily", "Das", 5000,40);
        Integer empID3 = ME.addEmployee("Geeta", "Patil", 10000,49);

        /* List down all the employees */
        ME.listEmployees();

        /* Update employee's records */
        ME.updateEmployee(empID1, 5000);

        /* Delete an employee from the database */
        ME.deleteEmployee(empID2);

        /* List down new list of the employees */
        ME.listEmployees();
    }
    /* Method to CREATE an employee in the database */
    public Integer addEmployee(String fname, String lname, int salary, int
    age)
    {
        Session s = factory.openSession();
        Transaction tx = null;
        Integer employeeID = null;
        try
        {
            tx = s.beginTransaction();
            Employee employee = new Employee(fname, lname, salary, age);
            employeeID = (Integer) s.save(employee);
            tx.commit();
        }
        catch (HibernateException e)
        {
            if (tx!=null) tx.rollback();
            e.printStackTrace();
        }
        finally
        {
            s.close();
        }
        return employeeID;
    }
    /* Method to READ all the employees */
    public void listEmployees( )
    {
        Session s = factory.openSession();
        Transaction tx = null;
```

```
      try
      {
         tx = s.beginTransaction();
         List employees = s.createQuery("FROM Employee").list();
         for (Iterator iterator = employees.iterator();
                                                    iterator.hasNext();)
         {
            Employee employee = (Employee) iterator.next();
            System.out.print("First Name: " + employee.getFstName());
            System.out.print("  Last Name: " + employee.getLstName());
            System.out.println("  Salary: " + employee.getSalary());
            System.out.println("  Age: " + employee.getAge());
         }
         tx.commit();
      }
      catch (HibernateException e)
      {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }
      finally
      {
         s.close();
      }
   }
   /* Method to UPDATE salary for an employee */
   public void updateEmployee(Integer EmployeeID, int salary)
   {
      Session s = factory.openSession();
      Transaction tx = null;

      try
      {
         tx = s.beginTransaction();
         Employee employee = (Employee)s.get(Employee.class, EmployeeID);
         employee.setSalary( salary );
         s.update(employee);
         tx.commit();
      }
      catch (HibernateException e)
      {
         if (tx!=null) tx.rollback();
```

```
            e.printStackTrace();
         } finally {
            s.close();
         }
      }
      /* Method to DELETE an employee from the records */
      public void deleteEmployee(Integer EmployeeID)
      {
         Session s = factory.openSession();
         Transaction tx = null;

         try
         {
            tx = s.beginTransaction();
            Employee employee = (Employee)s.get(Employee.class, EmployeeID);
            s.delete(employee);
            tx.commit();
         }
         catch (HibernateException e)
         {
            if (tx!=null) tx.rollback();
            e.printStackTrace();
         }
         finally
         {
            s.close();
         }
      }
   }
```

Compilation and Execution:

- Here are the steps to compile and run the above mentioned application. Make sure, you have set PATH and CLASSPATH appropriately before proceeding for the compilation and execution.
  - Create hibernate.cfg.xml configuration file as explained in configuration chapter.
  - Create Employee.hbm.xml mapping file as shown above.
  - Create Employee.java source file as shown above and compile it.
  - Create ManageEmployee.java source file as shown above and compile it.
  - Execute ManageEmployee binary to run the program.

- You would get the following result, and records would be created in the EMPLOYEE table.

```
$java ManageEmployee
.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........

First Name: Ashu  Last Name: Joshi  Salary: 2000   age: 37
First Name: Lily  Last Name: Das    Salary: 5000   age: 40
First Name: Geeta Last Name: Patil  Salary: 10000  age: 49
First Name: Ashu  Last Name: Joshi  Salary: 5000   age: 37
First Name: Geeta Last Name: Patil  Salary: 10000  age: 49
If you check your EMPLOYEE table, it should have the following records –
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+----
| id | first_name | last_name | salary |age
+----+------------+-----------+--------+----
| 29 | Ashu       | Joshi     |   5000 |37
| 31 | Geeta      | Patil     |  10000 |49
+----+------------+-----------+--------+-----
2 rows in set (0.00 sec
mysql>
```

ADDITIONAL PROGRAMS

Program 5.1: Spring core example to print Welcome to Spring Program.

HelloBean.java

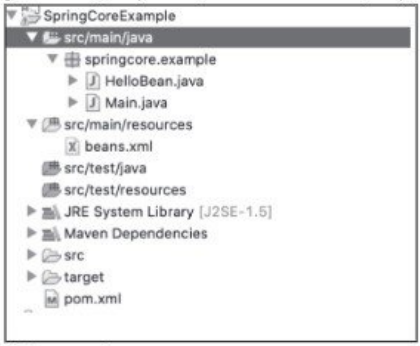
```
package springcore.example;
public classHelloBean
{
   private String name;
   public String getName()
   {
      return name;
   }
   public void setName(String name)
   {
      this.name = name;
   }
   public void sayHello()
   {
      System.out.println("Hello" + this.name);
   }
}
```

Main.java

```
package springcore.example;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main
{
   private static ApplicationContext context;
   public static void main(String[] args)
   {
      context = new ClassPathXmlApplicationContext("beans.xml");
      HelloBeanhelloBean = (HelloBean) context.getBean("HelloBean");
      helloBean.sayHello();
   }
}
Src/main/resources

<beansxmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"xmlns:p="http://www.springframework.org/schema/p"
   xmlns:aop="http://www.springframework.org/schema/aop"xmlns:context="http
   ://www.springframework.org/schema/context"
   xmlns:jee="http://www.springframework.org/schema/jee"xmlns:tx="http://ww
   w.springframework.org/schema/tx"
   xmlns:task="http://www.springframework.org/schema/task"
   xsi:schemaLocation="http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-3.2.xsd
      http://www.springframework.org/schema/jee
      http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
      http://www.springframework.org/schema/tx
      http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
      http://www.springframework.org/schema/task
      http://www.springframework.org/schema/task/spring-task-3.2.xsd">
   <context:component-scanbase-package="com.programcreek.examples"/>

   <beanid="HelloBean"class="springcore.example.HelloBean">
      <propertyname="name"value="Spring Programe"/>
   </bean>
</beans>
```

Output:

Program 5.2: Spring example for Current Date program.



HelloBean.java

```
package springcore.example;
public class HelloBean
{
   private String name;
   public String getName()
   {
      return name;
   }
   public void setName(String name)
   {
      this.name = name;
   }
```

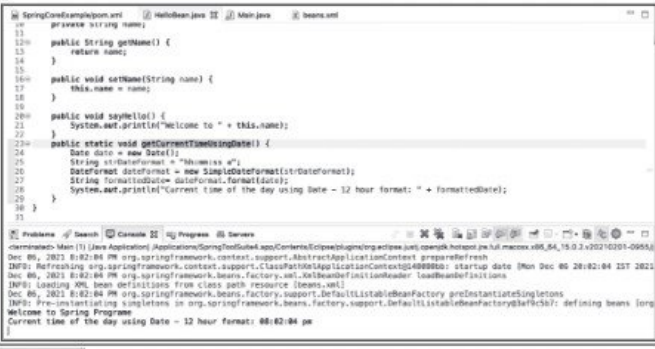
```
    public void sayHello()
    {
        System.out.println("Hello" + this.name);
    }
    public static void getCurrentTimeUsingDate()
    {
        Date date = new Date();
        String strDateFormat = "hh:mm:ss a";
        DateFormatdateFormat = newSimpleDateFormat(strDateFormat);
        String formattedDate= dateFormat.format(date);
        System.out.println("Current time of the day using Date - 12 hour
        format: " + formattedDate);
    }
}
```

Main.java

```
package springcore.example;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Main
{
    private static ApplicationContext context;
    public static void main(String[] args)
    {
        context = new ClassPathXmlApplicationContext("beans.xml");
        HelloBeanhelloBean = (HelloBean) context.getBean("HelloBean");
                                        helloBean.getCurrentTimeUsingDate();
        helloBean.sayHello();
    }
}
Src/main/resources
<beansxmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema
instance"xmlns:p="http://www.springframework.org/schema/p"
    xmlns:aop="http://www.springframework.org/schema/aop"xmlns:context="http
://www.springframework.org/schema/context"
    xmlns:jee="http://www.springframework.org/schema/jee"xmlns:tx="http://ww
w.springframework.org/schema/tx"
    xmlns:task="http://www.springframework.org/schema/task"
```

```
        xsi:schemaLocation="http://www.springframework.org/schema/aop
            http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-3.2.xsd
            http://www.springframework.org/schema/jee
            http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
            http://www.springframework.org/schema/tx
            http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
            http://www.springframework.org/schema/task
            http://www.springframework.org/schema/task/spring-task-3.2.xsd">
        <context:component-scanbase-package="com.programcreek.examples"/>
        <beanid="HelloBean"class="springcore.example.HelloBean">
            <propertyname="name"value="Spring Programe"/>
        </bean>
    </beans>
```

Output:



SUMMARY

- Spring framework is an open source Java platform that provides comprehensive infrastructure support for developing robust Java applications very easily and very rapidly.
- Hibernate is an open-source, non-invasive, light-weight java ORM(Object-Relational Mapping) framework to develop objects which are independent of the database software and make independent persistence logic in all JAVA, JEE.
- Spring is modular, allowing us to pick and choose which modules are applicable to an application, without having to bring in the rest.

- The core container consists of **core**, **beans**, **context** and **expression language modules** that are fundamental to any Spring application.
- Modules in the data layer help to provide data access from the web and business components. The data layer contain various modules are: **JDBC module, ORM module, JMS (Java Messaging Service) Module, OXM module, Transaction Module.**
- The Spring IoC container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The Spring container uses Dependency Injection (DI) to manage the components that make up an application.
- Spring provides two types of containers:
  1. BeanFactory container
  2. ApplicationContext container
- A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.
- The Spring Framework supports the following five scopes, three of which are available only if you use a web-aware ApplicationContext.
- The Spring bean factory is responsible for managing the life cycle callbacks of the beans which are created in the spring containers.
- The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.
- The Hibernate architecture is categorized in four layers.
  - Java application layer
  - Hibernate framework layer
  - Backhand API layer
  - Database layer
- Hibernate Architecture Elements are SessionFactory, session, transaction, connectionprovider
- Hibernate works best if these classes follow some simple rules, also known as the Plain Old Java Object (POJO) programming model.

Check Your Understanding

1. What is spring?
   (a) Proprietary framework
   (b) Spring is a development framework for .Net applications
   (c) Open source development framework for enterprise Java
   (d) Spring is a development framework for PHP based applications
2. Attribute used to handle web flow requests.
   (a) servlet-requests (b) servlet-attr
   (c) servlet-flow (d) servlet-mapping
3. Component which additionally provides a pop-up date picker control for its enclosed input field.
   (a) dateValidator (b) clientValidator
   (c) clientDateValidator (d) validator

4. Parameter used to specify a configuration file.
   (a) contextConfigure (b) contextConfigLocation
   (c) contextLocation (d) contentType
5. What is the basic concept of Spring?
   (a) Dependency injection or Inversion of control
   (b) Factory pattern
   (c) Abstract factory pattern
   (d) Singleton pattern
6. In Spring based applications, the objects lives in _____.
   (a) spring container (b) spring heap
   (c) spring group (d) spring pool
7. Spring Dependency Injection is useful because it makes your code _____.
   (a) easier to understand (b) simple
   (c) easier to test (d) All of the above
8. Which of the following are spring modules?
   1. Core module 2. Web module
   3. Spring jar module 4. Test module
   5. Spring Lightweight module 6. Data access module
   (a) 1, 2, 3, 4 (b) 1, 2, 4, 6
   (c) 1, 3, 5, 6 (d) 3, 4
9. The Spring container gets its instructions on what objects to instantiate, configure, and assemble by reading the _____.
   (a) configuration text file (b) spring integration data
   (c) source code (d) configuration metadata
10. Which of the following is the default scope in Spring?
    (a) singleton (b) session
    (c) http (d) request
11. Which of the following is not a core interface of Hibernate?
    (a) Configuration (b) Criteria
    (c) SessionManagement (d) Session
12. Which of the following is not a core component of Hibernate?
    (a) JDBC (b) SessionFactory
    (c) Session (d) Configuration
13. SessionFactory is a thread-safe object.
    (a) True (b) False
    (c) none (d) all
14. The SessionFactory is heavyweight object.
    (a) False (b) True

15. Which of the following method is used inside hibernate session only?
    (a) merge() (b) update()
    (c) end() (d) kill()
16. Which of the following is not a state of object in Hibernate?
    (a) Attached() (b) Detached()
    (c) Persistent() (d) Transient()
17. Which of the following is not an advantage of using Hibernate Query Language?
    (a) Database independent (b) Easy to write query
    (c) No need to learn SQL (d) Difficult to implement

Answers

1. (c)	2. (d)	3. (c)	4. (b)	5. (a)	6. (a)	7. (d)	8. (b)	9. (d)	10. (a)
11. (c)	12. (a)	13. (a)	14. (b)	15. (b)	16. (a)	17. (d)			

Practice Questions

Q.I Answer the following questions in short:

1. What is hibernate?
2. What is ORM?
3. What are the core interfaces of Hibernate?
4. Mention some of the advantages of using ORM over JDBC.
5. List the key components of Hibernate.
6. Mention two components of Hibernate configuration object.
7. How is SQL query created in Hibernate?
8. What does HQL stand for?
9. How can we add criteria to a SQL query?
10. What is SessionFactory?
11. Is Session a thread-safe object?
12. What is the difference between session.save() and session.persist() method?

Q.II Answer the following Questions:

1. Explain hibernate architecture?
2. What is the difference between get and load method?
3. What is the difference between update and merge method?
4. What are the states of the object in hibernate?
5. What are the inheritance mapping strategies?

Q.III Define the terms:

1. Hibernate
2. Criteria
3. Persistent classes
4. Session

■■■