UNIT -III (WORKING WITH DATABASE)

What is ODBC?

In computing, Open Database Connectivity (**ODBC**) is a standard application programming interface (API) for accessing database management systems (DBMS). The designers of **ODBC** aimed to make it independent of database systems and operating systems.

What is JDBC?

JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

JDBC Architecture

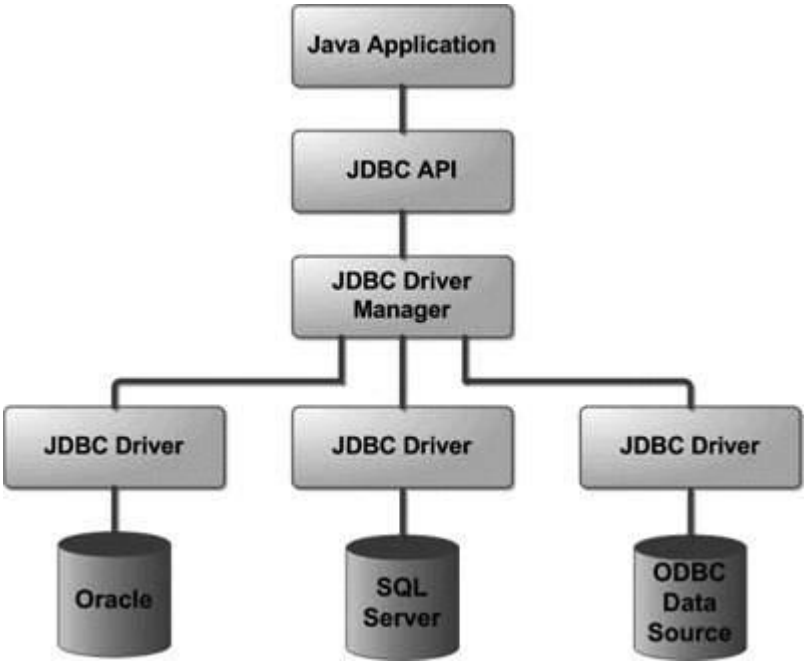
The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

- **JDBC API:** This provides the application-to-JDBC Manager connection.
- **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.

Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application −



Common JDBC Components

The JDBC API provides the following interfaces and classes −

- **DriverManager:** This 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 subprotocol under JDBC will be used to establish a database Connection.

- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.

- **Connection:** This 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.

- **Statement:** You 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.

- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

What is JDBC Driver?

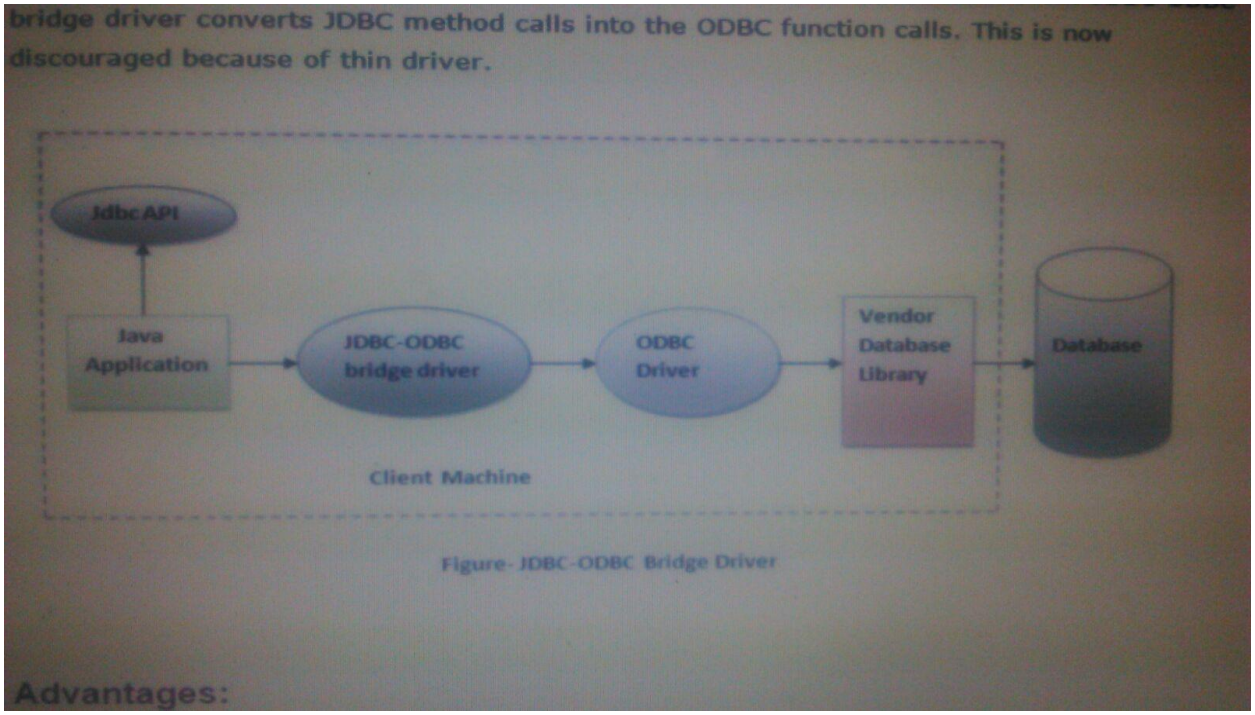
JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server.

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.

Types of JDBC Drivers:

JDBC Driver Type	Description
TYPE-1 Driver	This type of drivers are Bridge Driver(JDBC-ODBC bridge)
TYPE-2 Driver	This driver I partly java and partly Native Code Driver
TYPE-3 Driver	It is a Pure java driver that uses middle ware drive to connect to database
TYPE-4 Driver	It is Pure Java Driver and is directly connected to database

Type 1: JDBC-ODBC Bridge Driver



The TYPE-1 driver acts as a bridge between JDBC and ODBC. An example of this driver is the SUN JDBC-ODBC driver implementation. Java application os programmed using JDBC API and will use JDBC_ODBC bridge to get access to database server.

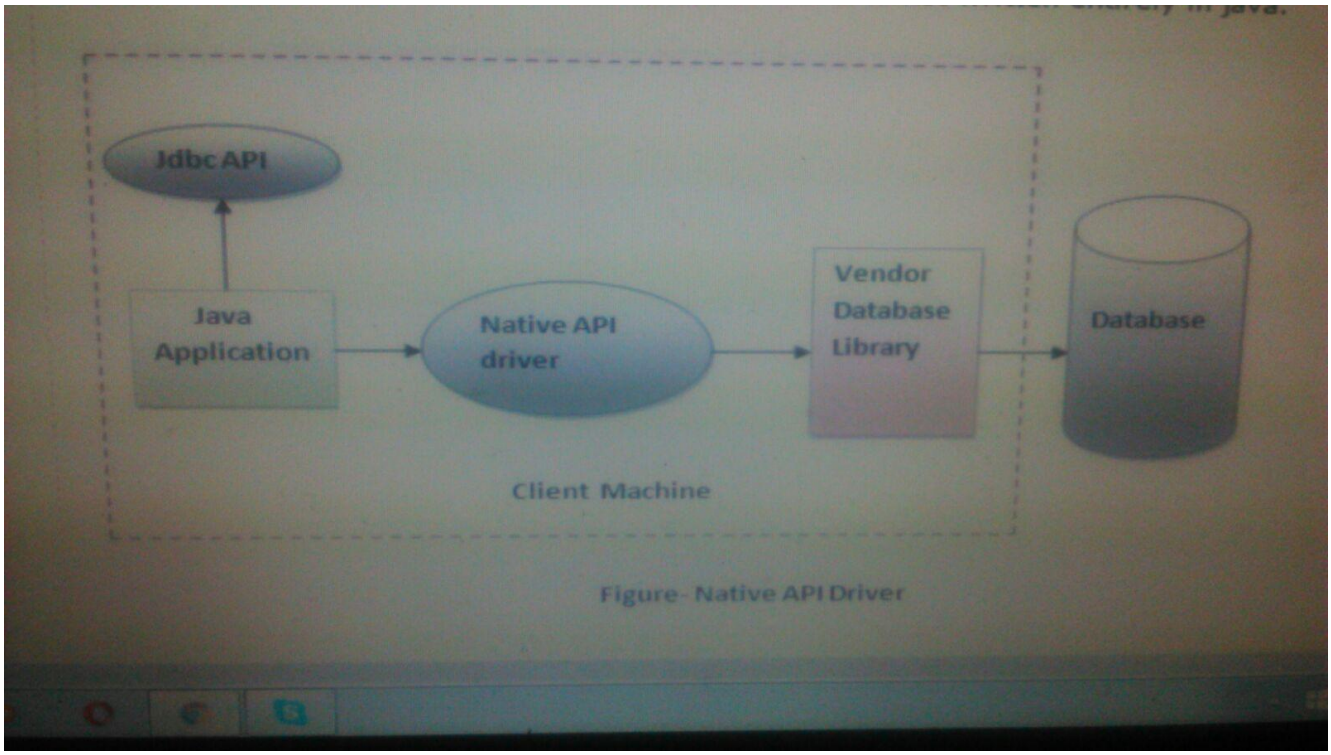
Advantages :

- Single Driver implementation can be used to interact with different data stores.
- This I vendor independent driver.

Disadvantages:

- Due to a large number of translations ,the execution speed is decresed.
- This driver depends on ODBC and therefore ,java applications also become indirectly depended on ODBC drivers.

Type 2: JDBC-Native API



Type-2 driver converts JDBC call into DB vendor specific native call. The application is built by using JDBC API ,but JDBC call are converted to database specific native library calls

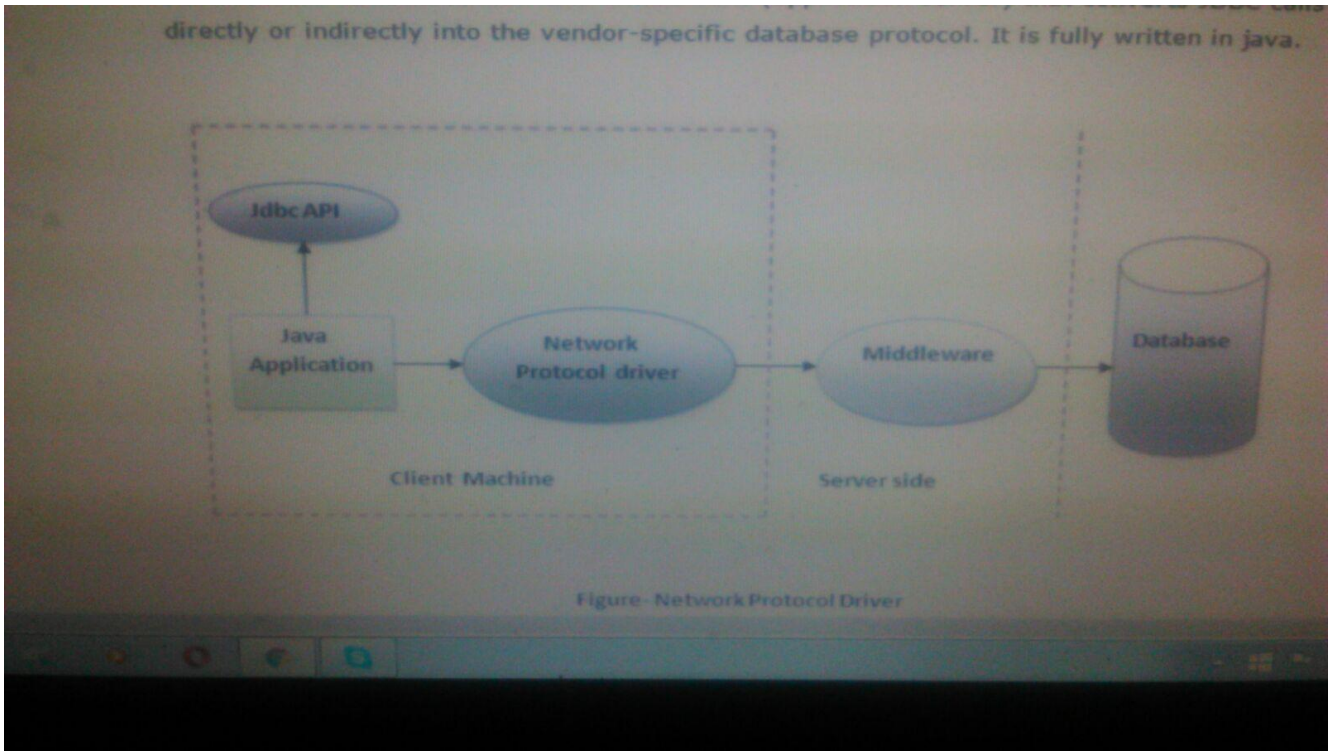
Advantages:

- Fastest driver when compared to other type of drivers

Disadnatage:

- Native libraries must be installed on client machine since the conversion of JDBC calls to native calls is done on the client machine.
- It may increase cost of the application if we want to install all vendors native libraries .

Type 3: JDBC-Net pure Java(using middleware)



The type-3 driver translates jdbc calls into middleware server specific calls ,and then converted to database server specific calls by middleware server.

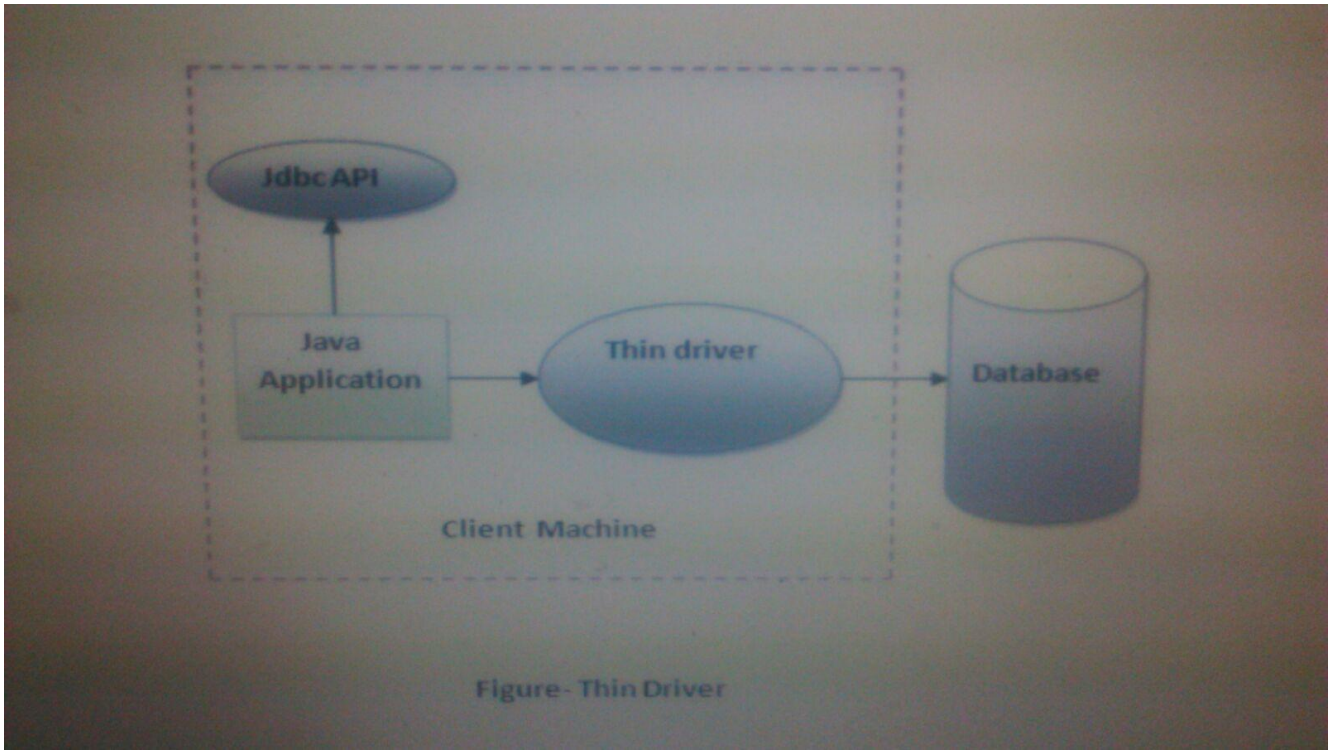
Advantages:

- Type-3 drivers are pure java drivers and auto downloadable.
- No native libraries are required on the client machine.
- A single driver provides accessibility to multiple databases, imply that this driver is database independent.

Disadvantages:

- Compared to Type-2 driver,Type-3 drivers are slow due to the increased number of network calls.
- Type-3 drivers are costlier compared to the other drivers.

Type 4: 100% Pure Java(Thin Driver)



The Type-4 driver is a pure java driver, which implements the database protocol to interact directly with a database. This type of drivers does not require any native database libraries to retrieve the records from the database. This type of drivers translates JDBC calls into database specific network calls. This type of drivers is light weight known as thin driver.

Advantages of Type-4 Driver:

- This type of drivers are pure java drivers and hence auto downloadable.
- No native libraries are required to be installed.
- Secure to use since it uses database specific protocols.

Disadvantages:

- It uses database specific proprietary protocol and is DBMS vendor independent.

Communicating with Database using JDBC API'S :

how to get a java application to connect with database and execute sql statements :

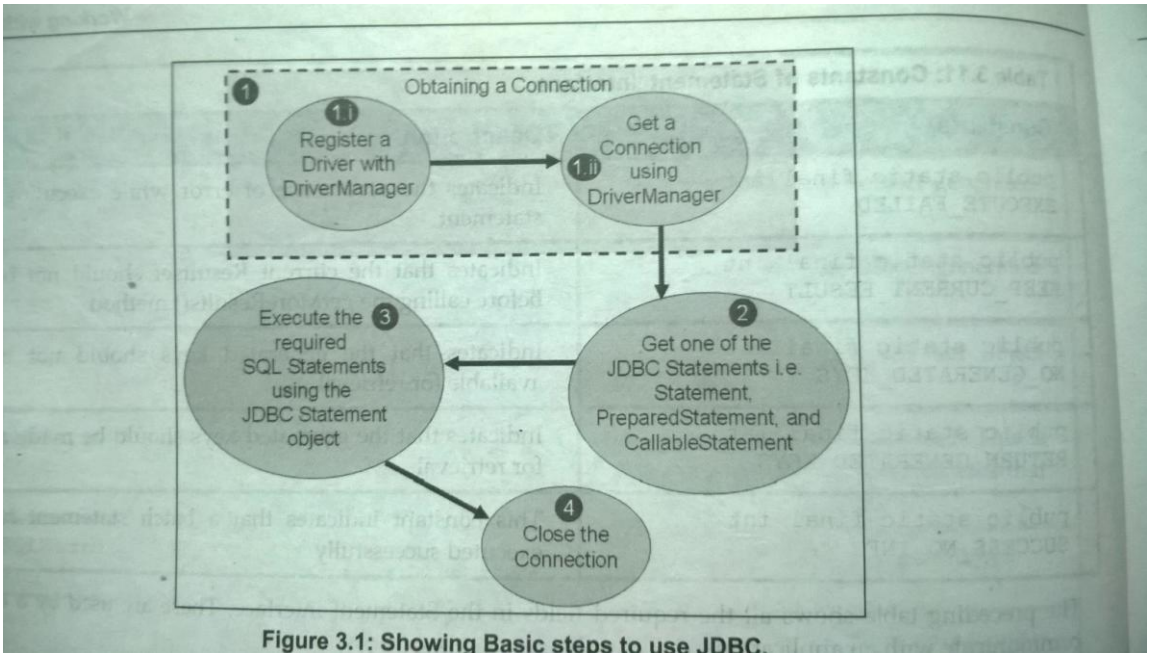


Figure 3.1: Showing Basic steps to use JDBC.

Step 1: Obtaining and establishing Connection of java application with DataBase:

(i) Register a Driver with DriverManager.

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

(ii)Get a Connection using Drivermanager

```
con=DriverManager.getConnection("jdbc:odbc:data4","system","admin");
```

Step 2: Creating a JDBC Statement Object :

After the connection is made, we need to create the JDBC statement object to execute the SQL statements.

```
stmt=con . createStatement();
```

Step 3 : Executing SQL statements

statement.executeUpdate() , statement.executeQuery() methods are used to execute SQL statements.

Step 4: Closing the connection by using connection.close();

Example 1:(creating table in database using java application)

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

class Db1
{
     public static void main(String args[])
     {
            Connection con=null;
            Statement stmt=null;
            try
             {
                      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");


con=DriverManager.getConnection("jdbc:odbc:data4","system","pru");

                     stmt=con.createStatement();

                    stmt.executeUpdate("create table manager(id
varchar2(20),name varchar2(20),salary int)");

                    System.out.println("table created successfully");

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

DESCRIBING JDBC STATEMENTS:

A JDBC Statement is used to execute a SQL statement . JDBC API supports three types of JDBC Statement objects to work with SQL Statements which are as follows :

Statement : executes normal SQL Statement to update or query database.
PreparedStatement: executes parameterized SQL statements that supports IN parameters .
Callable Statement : executes parameterized SQL statements that supports IN and OUT parameters .

Working with Statement Interface:

1. executes normal SQL Statement to update or query database.
2. Methods supported by statement object:
   i. executeUpdate()
   ii.executequery()
   iii.execute()
3. Process Flow of Statement Object:

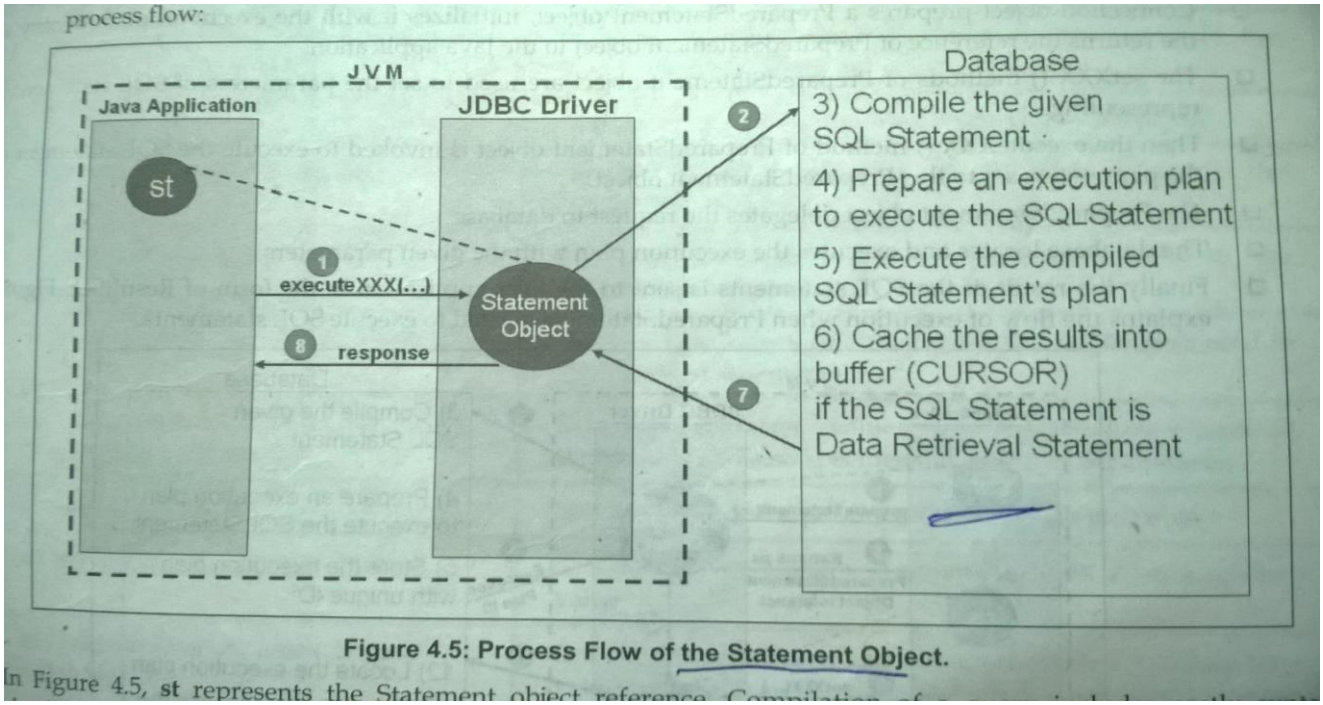


Figure 4.5: Process Flow of the Statement Object.

In Figure 4.5, **st** represents the Statement object reference. Compilation of

Example: The above example 1 can be taken

Example 2: (Inserting values for the table created in database)

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

class Db2
{
     public static void main(String args[])
     {
            Connection con=null;
            Statement stmt=null;
            String mid="123",mname="raghu";
            int  sal=20000;
            try
             {
```

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

con=DriverManager.getConnection("jdbc:odbc:data4","system","pru");

                     stmt=con.createStatement();

                    stmt.executeUpdate("insert into manager
values('"+mid+"','"+mname+"',"+sal+")");

                    System.out.println("inserted successfully");

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

Working with Prepared Statement Interface:

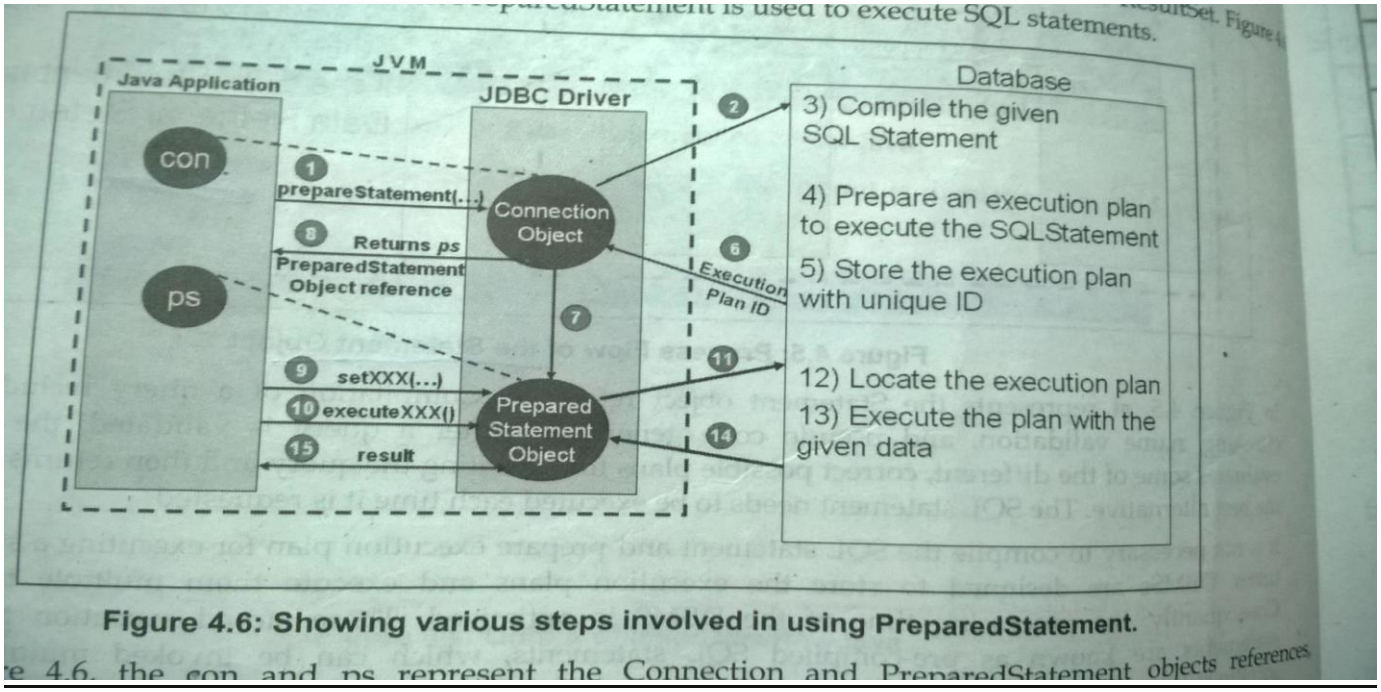


Figure 4.6: Showing various steps involved in using PreparedStatement.

Example 3:(Inserting values in to table in database using Prepared Statement):

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

class PS1
{
     public static void main(String args[])
     {
            Connection con=null;
            String mid="7676",mname="girish";
            PreparedStatement ps;
            int  sal=40000;
            try
             {
                      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");


con=DriverManager.getConnection("jdbc:odbc:data4","system","admin");
                      String q="insert into manager values(?,?,?)";
                      ps=con.prepareStatement(q);
                      ps.setString(1,mid);
                      ps.setString(2,mname);
                      ps.setInt(3,sal);
                     ps.executeUpdate();
                     System.out.println("inserted successfully");

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

Example 4: (getting values from database using resultset and prepared statement):

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

class PS2
{
     public static void main(String args[])
     {
            Connection con=null;
            ResultSet rs=null;
            PreparedStatement ps;
            try
             {
                      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
con=DriverManager.getConnection("jdbc:odbc:data4","system","pru");
                      String q="select * from manager ";
                      ps=con.prepareStatement(q);
                     rs=ps.executeQuery();
                     while(rs.next())
                     {
                         System.out.println(rs.getString("id")+"
"+rs.getString("name")+"    "+rs.getInt("salary"));
                         System.out.println("\n");
                     }


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

Working with ResultSet Interface:

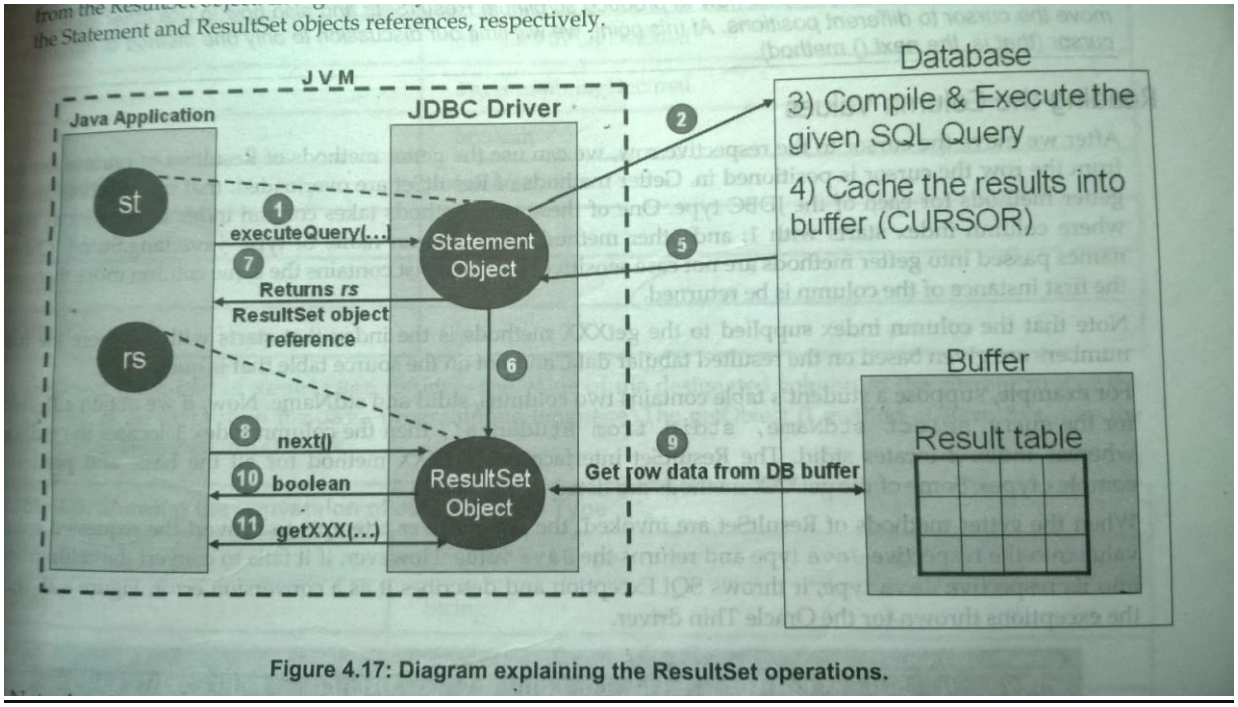


Figure 4.17: Diagram explaining the ResultSet operations.

Example 5: (getting values from database using resultset)

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

class Db3
{
     public static void main(String args[])
     {
            Connection con=null;
            Statement stmt=null;
            ResultSet rs=null;
            try
             {
                      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");


con=DriverManager.getConnection("jdbc:odbc:data4","system","pru");

                       stmt=con.createStatement();

                      rs=stmt.executeQuery("select * from manager");
                      while(rs.next())
                      {
```

```
                        System.out.println(rs.getString("id")+"
"+rs.getString("name")+"    "+rs.getInt("salary"));
                        System.out.println("\n");
                    }


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

UNIT-III(WORKING WITH SERVLETS)

What are Servlets?

Servlets are the components of java that are used to create dynamic web applications.Servlets can run on any java enabled platform.Java Servlets are usually designed to process HTTP requets.

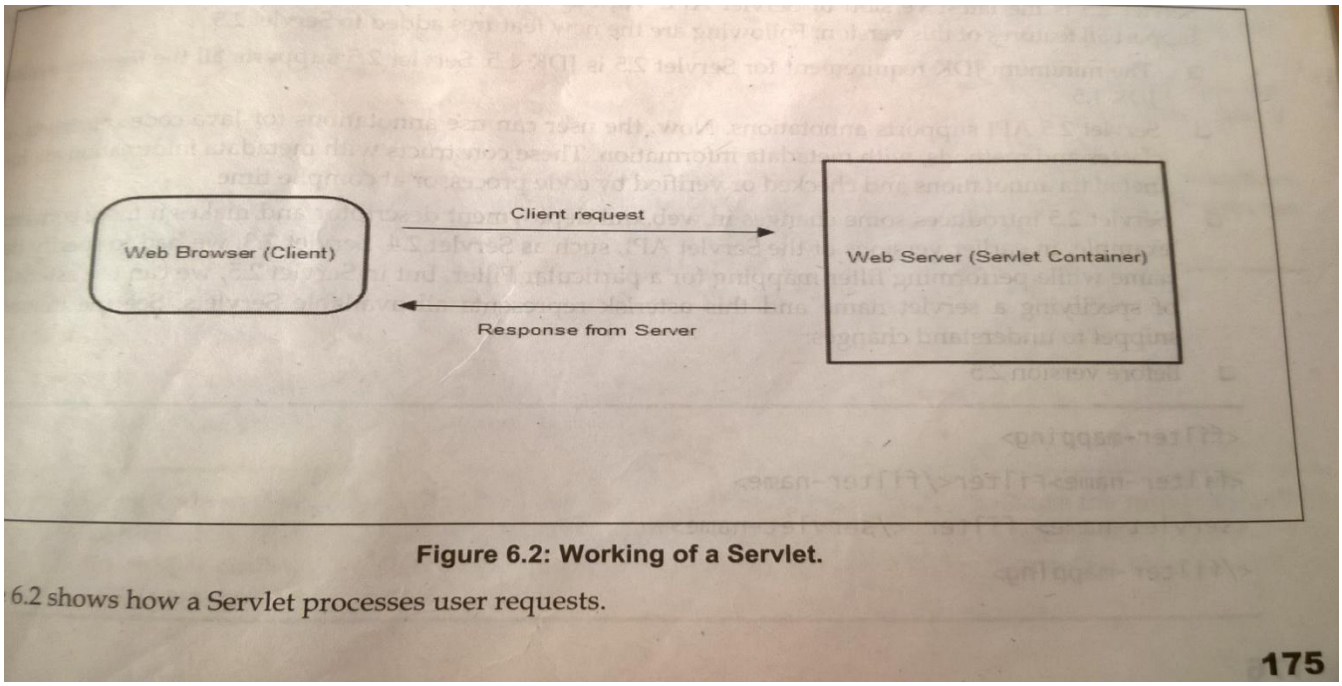


Figure 6.2: Working of a Servlet.

6.2 shows how a Servlet processes user requests.

175

The process can be summarized as follows:

- A client sends a request to a servlet container,which acts as aweb server,through a web browser.
- The Web server searches for the required servlet and initiate it.
- The Servlet then process the client request and sends the response back to the server,which is then forwarded to client.

Advantages of servlets:

- Faster than CGI
- Platform independent as servlets are written in java.Servlets can run on any servlet enabled web server.
- Use a standard vendor independent API supported by many web servers.

- Removes overhead of creating a new process for each request(unlike CGI).
- Servlets can inherit the security provided by the Web server.

Servlet Disadvantage :

1. Designing in servlet is difficult and slows down the application.

2. Writing complex business logic makes the application difficult to understand.

3. You need a Java Runtime Environment on the server to run servlets. CGI is a completely language independent protocol, so you can write CGIs in whatever languages you have available (including Java if you want to).

Introductio to Servlet API:

Packages:

Javax.servlet

Javax.servlet.http

The following are some of interfaces and classes in Javax.servlet

Type	Name	Description
Interface	**RequestDispatcher**	
Interface	**Servlet**	
Interface	**ServletConfig**	
Interface	**ServletContext**	
Interface	**ServletRequest**	

Interface	**ServletResponse**	
Class	**GenericServlet**	
Class	**ServletException**	

The following are some of interfaces and classes in Javax.servlet.http

Type	**Name**	**Description**
Interface	**HttpSession**	
Interface	**HttpServletRequest**	
Interface	**HttpServletResponse**	
Class	**Cookie**	
Class	**HttpServlet**	

Life Cycle of a Servlet:

The 4 stages are

- **Loading a servlet**
- **Initializing a servlet**
- **Request Handling**
- **Destroying the servlet**

Loading a servlet:

The first stage of the servlet life cycle involves loading and initializing the servlet by container.

The servlet container perorms the following operations as a part of this process:

Loading:The servlet container loads the servlet class by using class loading option from file system.

Instantiation:After a servlet class is loaded successfully,the servlet container creates an instance(i.e object)of a servlet.

Initializing a servlet:

After the servlet is instantiated successfully,the servlet container initializes the instantiated servlet object.The servlet container initializes this object by invoking ” init()” method.In init() method you can configure the servlet.

Request handling:

- After servlet is intialized ,the servlet is now ready to process client request.
- To process client request the servlet container invokes “service()” method
- This service() is responsible for handling client request and response.

Destroy(end of service):

When a servlet container decides to destroy the servlet,it do follwing operations

- It let all the threads currently running in the service method of the servlet instance to complete thier jobs and get released.
- Then sevlet container make a call to ” destroy()” method.
- Then the destroy method releases the resources held by the servlet.

Servlet Skeleton:

import javax.servlet.*;

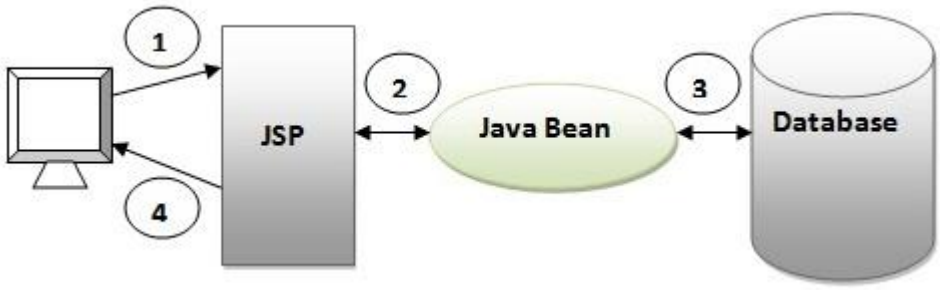
```
import java.io.*;
public class greeting extends GenericServlet
{
        public  void init(ServletConfig sc)
         {
              //initilaze the configurations for servlet
         }
        public void service(ServletRequest req,ServletResponse res)throws
ServletException,IOException
        {
              //handle request and response
         }
        public  void destroy()
        {
             //resources are released by the servlet.
         }

}
```

Web Architecture Models:

Before developing the web applications, we need to have idea about design models. There are two types of programming models (design models)

1. Model 1 Architecture
2. Model 2 (MVC) Architecture

Model 1 Architecture



As you can see in the above figure, there is picture which show the flow of the model1 architecture.

1. Browser sends request for the JSP page
2. JSP accesses Java Bean and invokes business logic
3. Java Bean connects to the database and get/save data
4. Response is sent to the browser which is generated by JSP

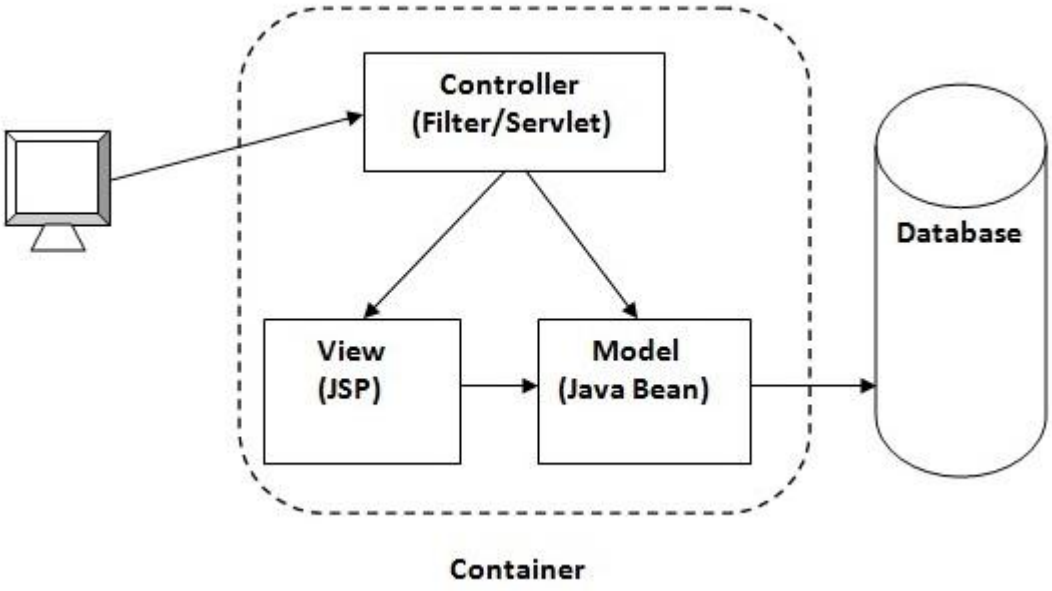
Advantage of Model 1 Architecture

- Easy and Quick to develop web application

Disadvantage of Model 1 Architecture

- **Navigation control is decentralized** since every page contains the logic to determine the next page. If JSP page name is changed that is referred by other pages, we need to change it in all the pages that leads to the maintenance problem.
- **Time consuming** You need to spend more time to develop custom tags in JSP. So that we don't need to use scriptlet tag.
- **Hard to extend** It is better for small applications but not for large applications.

Model 2 (MVC) Architecture:



Model **V**iew **C**ontroller or MVC as it is popularly called, is a software design pattern for developing web applications. It is a **design pattern** that separates the business logic, presentation logic and data.

A Model View Controller pattern is made up of the following three parts:

- **Model** - The lowest level of the pattern which is responsible for maintaining data. It can also have business logic.

  View - This is responsible for displaying all or a portion of the data to the user. represents the presentaion i.e. UI(User Interface).

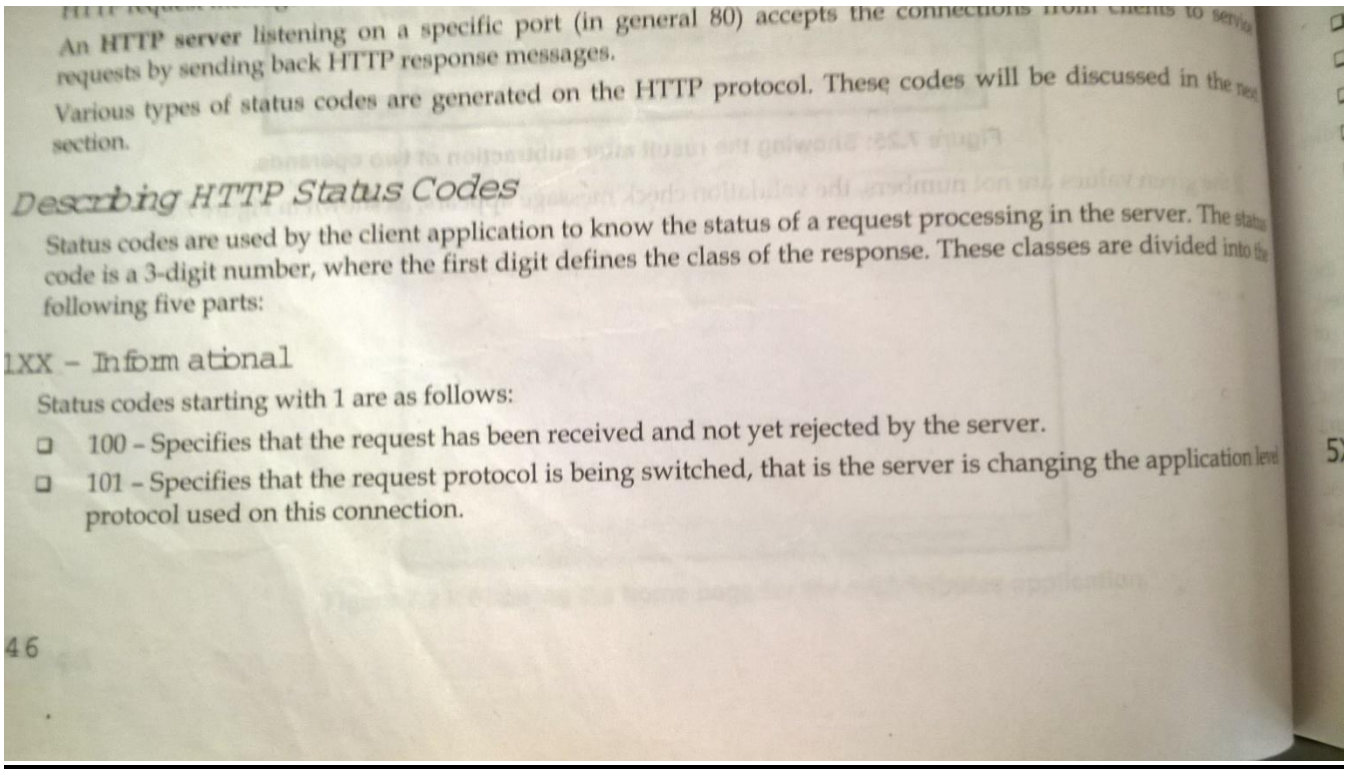
  Controller - Software Code that controls the interactions between the Model and View.

MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows.

Advantage of MVC (Model 2) Architecture

1. Navigation Control is centralized
2. Easy to maintain the large application

Describing HTTP Status Codes



An **HTTP server** listening on a specific port (in general 80) accepts the connections from clients to service requests by sending back HTTP response messages.

Various types of status codes are generated on the HTTP protocol. These codes will be discussed in the next section.

Describing HTTP Status Codes

Status codes are used by the client application to know the status of a request processing in the server. The status code is a 3-digit number, where the first digit defines the class of the response. These classes are divided into the following five parts:

1XX – Informational

Status codes starting with 1 are as follows:

- 100 – Specifies that the request has been received and not yet rejected by the server.
- 101 – Specifies that the request protocol is being switched, that is the server is changing the application level protocol used on this connection.

46

2XX – Success

The various status codes under this class defined under the HTTP specification are as follows:

- 200 – OK
- 201 – Created
- 202 – Accepted
- 203 – Non-Authoritative Information
- 204 – No Content
- 205 – Reset Content
- 206 – Partial Content

3XX – Redirection

The various status codes under this class defined under the HTTP specification are as follows:

- 300 – Multiple Choices
- 301 – Moved Permanently
- 302 – Found
- 303 – See Other
- 304 – Not Modified
- 305 – Use Proxy
- 307 – Temporary Redirect

4XX – Client Error

The various status codes under this class defined under the HTTP specification are as follows:

- 400 – Bad Request
- 401 – Unauthorized
- 402 – Payment Required
- 403 – Forbidden
- 404 – Not Found
- 405 – Method Not Allowed
- 406 – Not Acceptable
- 407 – Proxy Authentication Required
- 408 – Request Time-out
- 409 – Conflict
- 410 – Gone
- 411 – Length Required
- 412 – Precondition Failed
- 413 – Request Entity Too Large
- 414 – Request-URI Too Large
- 415 – Unsupported Media Type
- 416 – Requested range not satisfy able
- 417 – Expectation Failed

5XX – Server Error

The various status codes under this class defined under the HTTP specification are as follows:

- 500 – Internal Server Error
- 501 – Not Implemented

- 502 -- Bad Gateway
- 503 -- Service Unavailable
- 504 -- Gateway Time-out
- 505 -- HTTP Version not supported

All these status codes are given for HTTP 1.1 protocol spe
and HTTP status codes. Now it is time to discuss HttpServl

Describing HttpServlet

We know that the container provides support for the H
HttpServletResponse objects if the HTTP protocol is used fo
encapsulates all HTTP protocol based request information
HttpServletResponse objects