JAVA-BASED DYNAMIC WEB APPLICATION

BY

AHMED MUSA MIDILA
(PG/M.SC/07/42484)

BEING A MSC PROJECT PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF A MASTER OF SCIENCE DEGREE IN COMPUTER SCIENCE.

SUPERVISOR: DR. G.A.M. IKEKEONWU

DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF POST GRADUATE STUDIES
UNIVERSITY OF NIGERIA, NSUKKA

MARCH, 2010

JAVA-BASED DYNAMIC WEB APPLICATION

BY

AHMED MUSA MIDILA
(PG/M.SC/07/42484)

DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF POST GRADUATE STUDIES
UNIVERSITY OF NIGERIA, NSUKKA

MARCH, 2010

CERTIFICATION PAGE

AHMED Musa Midila, a postgraduate student in the Department of Computer Science with registration number PG/M.Sc/07/42484 has satisfactorily completed the research work for the degree of Msc in Computer Science.

The work embodied in this project report has not been submitted in part or full for any diploma or Degree of this university or any other university.

DR G.A.M. Ikekeonwu --------------------------------
Supervisor Signature/date

DR(MRS) M.N. Agu ---------------------------------
Head Of Department Signature/date

External Supervisor ---------------------------------
Signature/date

DEDICATION

This work is dedicated to my late father Baba Ahmadu Isa Hong who passed away 19 days to the submission of this work, May his soul rest in perfect peace, Amin.

ACKNOWLEDGEMENT

A work of this nature relies influences, direct and indirect, for accomplishment. To those who cannot be mentioned in this write-up, I hereby acknowledge indebtedness.

Worthy of special mentioned is my supervisor, DR G.A.M. Ikekeonwu, I owe immeasurable gratitude to his patience, encouragement and diligent supervision of this work.

I am very grateful to Prof H.C. Inyiama, Former HOD Computer Science, Prof. A. A. Ubachuckwu, the current HOD Computer Science, Dr(mrs) M.N. Agu, O.E. Oguike, the Secretary and other lecturers of the department who either read this work at various stages and offered valuable advice or encouragement.

Equally, my colleagues in the computer science profession, Dr A.A, Musa, Dr N. Bulama, John Anthony, ChukwuEmeka Nzeih, Lady Cynthia Enweonwu deserve special mention and thanks for their useful and invaluable contributions.

The total support of my wife Aishatu Musa Ahmed, my Cousin Aishatu Musa, Mr K. M. Fasasi, The HOD Science Education, Federal university of Technology, Yola, DR John Sakiyo, Shetima Abdullahi, my most senior Brother Alhaji Mohammed Njidda Ahmed and my other brothers and sisters towards the completion of this work is inestimable. I am deeply grateful to them.

Finally, I remain eternally grateful to the Almighty Allah for sustaining me throughout the period of this study.

ABSTRACT

This project report demonstrated the Echoclient and Echoserver network programming, while the server program runs on the server machine rendered the server listens to connection request, then the echoclient program on the client machine requested for connection.Then connection was established, so the client and server machines communicates through the internet. The researcher reviewed the six popular Java-Based approaches that supports persistent connection between web clients and database server; they are java applets, java sockets, java servlets, remote method invocation (RMI), CORBA and Mobile Agent Technology (MAT). In order to understand how output from these systems actually gets from the server to the browser, the researcher investigated how the server uses standard internet conversion and protocols to send resources to client, he briefly described the following: The TCP/IP Network protocol, The Communication Ports, The Hypertext Transfer protocols (HTTP), The Web Browser, The Uniform Resources Locator, The resources MIME types. The system development requirement was identified, and described how to download and install all the necessary software to implement the application. MySQL database was set up and a table named 'result' was created in the database. MySQL database connection was first created in the NetBeans IDE before connecting to a web application. The web Application was connected to MySQl database with JDBC datasource and the resultset management was performed using CreateStatement and PreparedStatement methods of the connection object.

TABLE OF CONTENTS

List of figures

List of Tables

CHAPTER ONE

INTRODUCTION

1.0 Java Programming

Java is an object-oriented language designed to support the development of distributed, secure and portable applications. The uniqueness of java lies on the fact that it's compiled and run on any platform which supports a java run time environment. There is much excitement about the internet and the World Wide Web (www), The internet ties the world's information together.

Java provides a number of built in networking capabilities that makes it easy to develop internet-based and web-based application. Java can enable programs to search the world for information and collaborate with programs running on other computers internationally, nationally or just within an organization.

This study can make enormous potential impact on fostering the modernization by providing student examination information on the World Wide Web. The student examination result is vital information that students need to know as early as possible in order to prepare for task ahead.

The socket program which is being applied in this study for connecting to the internet is a powerful technique which has advantage of high technical efficiency.

1.1 Statement of the Problem

There has been an increase in the use of the web/internet as a means for world wide access to information /resources. The internet could be effectively used to present and/or view information and other resources of universities among others globally. This has been found to provide a convenient way to view university information and resources on the web internationally, nationally or within a community.

Although much has been done on web designing and implementation, sufficient attention has not been paid to dynamic web application design that use database as a means of storing logically related data on the internet.

More importantly, more work need to be done in improving the way examination records is managed. This study aims to develop a java-based dynamic web application to read student examination records from a remote database.

1.2 Objectives

The aims of this study are:

i) Investigate the concepts of client/server network programming using Java socket.

ii) Develop echo client software that communicates with echo server software.

iii) Develop echo server software.

iv) Design a database for student examination record.

v) Set up a client/server communication to read data from the database.

1.3 **Significant of the Study**

This project can make enormous potential impact on fostering the modernization by providing student examination information on the World Wide Web. The student examination result is vital information that students need to know as early as possible in order to prepare for task ahead.

The socket program which is being applied in this study for connecting to the internet is a powerful technique which has advantage of high technical efficiency and wide area of application. Researchers in dynamic web application development and related areas can use this project as a reference material.

1.4 Literature Review

This subsection studies the relevant key concepts related to this study, it
is organized under the following headings an object, a socket, socket connection and Interprocess communication (IPC) Interface.

1.4.1 An Object

A Java class is a specific category of objects, an object is an instance of a class and may contain many variables, the composite of those values is called the state of the object, (Hubbard,1999).

Objects are key to understanding *object-oriented* technology, look around right now and you'll find many examples of real-world objects: your dog, your desk, your television set, your bicycle etc. Real-world objects share two characteristics: They all have *state* and *behavior*. Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.

For each object that you see, ask yourself two questions: "What possible states can this object be in?" and "What possible behavior can this object perform?”. you'll notice that real-world objects vary in complexity; your desktop lamp may have only two possible states (on and off) and two possible behaviors (turn on, turn off), but your desktop radio might have additional states (on, off, current volume, current station) and behavior (turn on, turn off, increase volume, decrease volume, seek, scan, and tune). You may also notice that some objects, in turn, will also contain other objects. These real-world observations all translate into the world of object-oriented programming.

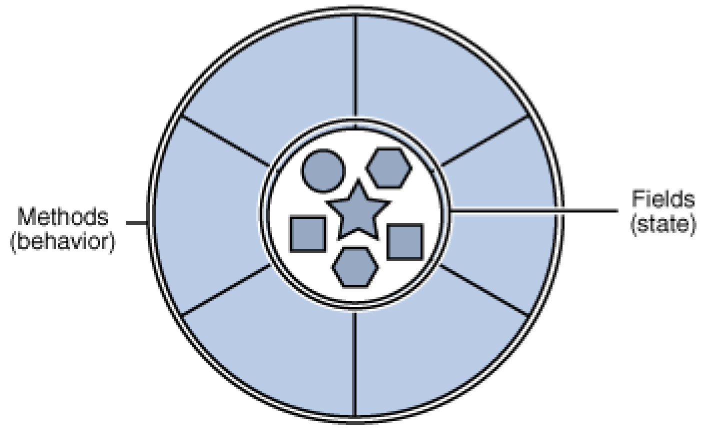


Fig. 1: A software object (javaworld.2002).

Software objects are conceptually similar to real-world objects: they too consist of state and related behavior. An object stores its state in *fields* (variables in some programming languages) and exposes its behavior through *methods* (functions in some programming languages). Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication. Hiding internal state and requiring all interaction to be performed through an object's methods is known as *data encapsulation* — a fundamental principle of object-oriented programming. Consider a bicycle, for example

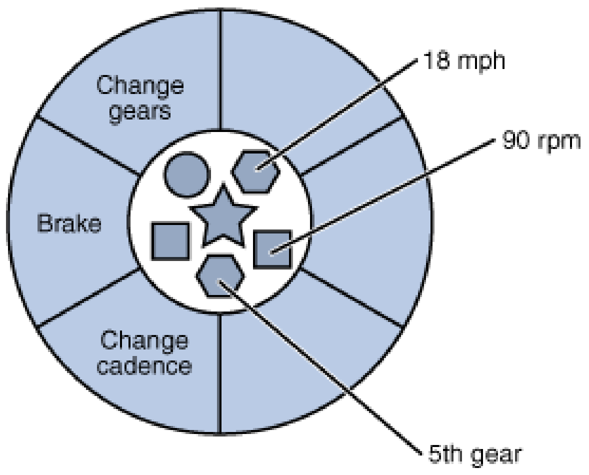


Fig.2: A bicycle modeled as a software object (javaworld.2002).

By attributing state (current speed, current pedal cadence, and current gear) and providing methods for changing that state, the object remains in control of how the outside world is allowed to use it. For example, if the bicycle only has 6 gears, a method to change gears could reject any value that is less than 1 or greater than 6.

Bundling code into individual software objects provides a number of benefits, including:

1. Modularity: The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object can be easily passed around inside the system.
2. Information-hiding: By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
3. Code re-use: If an object already exists (perhaps written by another software developer), you can use that object in your program. This allows specialists to implement/test/debug complex, task-specific objects, which you can then trust to run in your own code.
4. Pluggability and debugging ease: If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace *it*, not the entire machine.

A typical Java program creates many objects, which interact by invoking methods through these object interactions, a program can carry out various tasks, such as implementing a GUI, running an animation, or sending and receiving information over a network. Once an object has completed the work for which it was created, its resources are recycled for use by other objects.

In the real world, you'll often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your bicycle is an *instance* of the *class of objects* known as bicycles. A *class* is the blueprint from which individual objects are created.

In mathematics an object is a kind of thing we call algebra, we have a set of objects with operations to be performed on them. So a class in java is a collection of data objects and operation. Consider the function

$$F = xy + z$$

Here F is the class, x, y and z are objects of the class F and the operation multiply x and y add the result to z is the method.

The object-oriented design and programming methodology has been notably successful in helping software developers achieve the goals of reliability, cost effectiveness, adaptability, understandability and reusability. Cohoon and Davidson(2004).

Programming languages generally give programmers the ability to specify the types of objects to use in their programs. Where a type is a collection of values and the operations that can be performed on those values. Object-oriented languages provide programmers the ability to create a type known as a class for representing the properties and message handling behaviors of a type of object. Classes normally are organized so that the manipulation of a property is handled via a message to the object. Programmers use a class to create instance of it. An instance is an object with a particular value for each property and attribute of the class.

1.4.2 A Socket

A socket is one end of a two-way communication link between two programs on the network, Sun(2004). A socket is bound to a port number 7 so that the transmission control protocol (TCP) layer can identify the application that data is destined to be sent. The list of well known ports used by TCP is given in table 1 below. A socket is the interface between the application layer and the transport layer. In less technical terms, a socket can be thought of as a mailbox. A process delivers to and receives information from its socket. Since this application is presented in Java, you will see how to create and use sockets using the Java API java.net.Socket to read and write students' examination record to a remote database. Message destinations are specified as socket address in a communication identifier that consists of a port number and internet address.

Table 1: Well known ports used by TCP outline by Forouzan(2006)

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP.Data	File Transfer Protocol (data connection)
21	FTP.Control	File Transfer Protocol (data connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol

53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext transport Protocol
111	RPC	Remote procedure Cal

1.4.3 Socket Connection

In order to do communication over the TCP protocol, a connection must first be established between the pair of sockets. While one of the sockets listens for a request (server), the other asks for a connection (client). Once two socket have been connected, they can be used to transmit data in both (or either one of the direction) direction.

On the client-side: The client knows the host name of the machine on which the server is running and the port number on which server is listening. The client makes a connection request with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

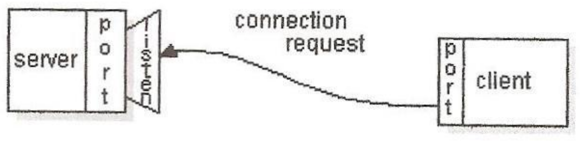


Fig. 3: Diagram for connection request

On the server-side: If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has it remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connection client.

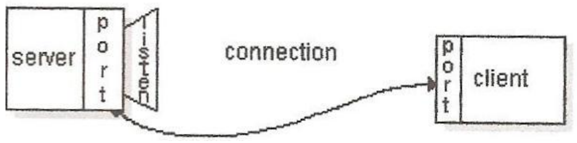


Fig. 4: Diagram for connection

If the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

The client and server can now communicate by writing to or reading from their sockets, and finally close the socket.

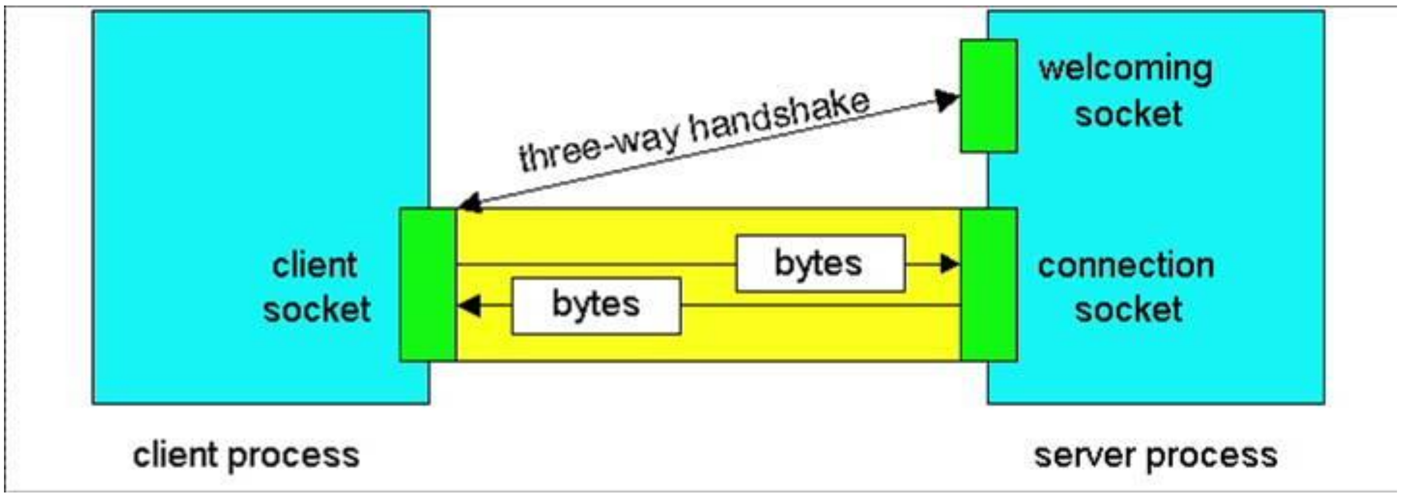


Fig.5: Client Socket, Welcoming Socket and Connection Socket(Sun,2002)

1.4.4 Interprocess Communication (IPC) Interface

The IPC operations are based on socket pairs belonging to a communication processing Sun(2004). It is done by exchanging some data through transmitting that data in a message between a socket in one process and another socket in another process. When messages are queued at the receiving socket until the receiving process makes the necessary calls to receive them.

There are two communication protocols that can be used for socket programming:

(a) Datagram Communication,(User Datagram protocol, UDP)

(b) Stream Communication,(Transmission control protocol, TCP)

(a) Datagram Communication (UDP)

- UDP is connectionless protocol, every time you send a datagram, you have to send the local descriptor and the socket address of the receiving socket along with it.
- There is a size limit of 64 kilobytes on datagram you can send to a specified location.
- UDP is unreliable protocol, there is no guarantee that the datagrams you have sent will be received in the same order by the receiving socket.
- UDP is less complex and often used in implementing client/server application in distributed systems built over local area networks.

(b) Stream Communication (TCP)

It is a process to process (program to program) protocol, which means delivery of a packet (a piece of information that form part of a message sent through a computer network) from the sending process to the destination process.

TCP allows the sending process to deliver data as a stream of bytes and allow receiving process to deliver data as a stream of bytes. The TCP creates an environment in which two processes seem to be connected by an imaginary “tube” that carries data across the internet. This imaginary environment is shown in the figure below.

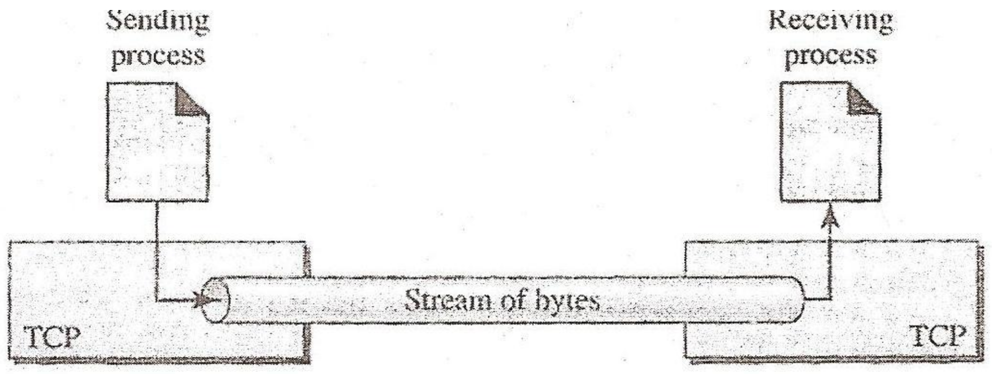


Fig.6: TCP Stream of delivery (Forouzan,2006).

The sending process produces (writes to) to the stream of bytes, and the receiving process consumes (reads from) them. TCP is a connection-oriented protocol, a connection must be established before communications between the pair of sockets start.

- All available data are read immediately in the same order in which they are received.
- It is guaranteed that the packets you send will be received in the order in which they are sent, it is a reliable protocol.
- It is useful for implementing network services such as remote login and file transfer (FTP) which require data of indefinite length to be transferred.

It is the responsibility of the programmer to implement an accurate communication protocol at the application level to cause the data to flow in an orderly manner.

CHAPTER TWO

THEORETICAL BACKGROUND

2.0 Introduction

The two most important concepts in object-oriented programming is class and object, an object is a thing both tangible and intangible that we can imagine. An object is an instance of a class; inside a program we write instructions to create objects of a class. For the computer to be able to create an object, we must provide a definition called a class. A class is a kind of mold or template that dictates the behaviors of objects. A class must be defined before you can create an instance (object) of the class, Wu(2004).

To instruct a class or an object to perform a task, we send a message to it. We can send a message only to the classes and objects that understand the message sent to them using a matching method or a sequence of instructions that a class or an object follows to perform a task. A method defined for a class is called a class method and a method defined for an object is called an instance method. A value we pass to an object is called an argument of a message.

There are two types of classes in java, user defined classes developed by programmers to suit their purpose and standard classes developed by expert developers which can be trusted and used by other programmers. This application uses two standard classes the Echoclient class and the Echoserver class developed by Sun Microsystems.

2.1 JDBC API

Providing efficient and secure access to remote databases using a web browser is crucial for the emerging cooperative information systems.

Lets briefly discuss the six popular java-based approaches that support persistent connections between web clients and database servers. The approaches according to

Monson-Haefel(2001) include java applets, java sockets, java servlets, Remote Method invocation (RMI), CORBA and Mobile agents technology (MAT).

2.1.1 The Java socket Approach

In this first approach, the middleware that connect software components or applications is a stand-alone java application server; an *application server* is a *server* program on a computer in a distributed network that provides the business logic for an *application* program running on the web server machine, examples are Apache Tomcat, glassfish etc. Web server is a piece of computer software that responds to a browser through the internet, examples are Apache http web server; Internet Information services (IIS) etc. The client collaborates with application server by establishing an explicit socket connection. The client summits the query through the socket connected to the application server, which decodes the incoming stream of data and executes the query on the database server. The result table is then passed to the client applet again by the means of data streams.

The cost of the first query in this approach is:

(i) Initialization phase:

- The time for the client to open a socket connection with application server.

(ii) Execution phase:

- The time for the client to pass to the application server the data stream containing SQL statement.
- The time for the application server to execute the query, obtain the results and return them to the client.

All subsequent queries require only the execution phase. We will latter discuss **the java sockets approach** in more detail, and then get to the coding.

2.1.2 Java servlets Approach

In the servlet approach, the middleware program is a java servlet which is a java program that runs as a child process within the context of a web server program, Monson-Haefel(2001). The web server is responsible for loading, maintaining and terminating servlets.

Client's queries are rooted by the web server to a servlet, which summit them to the database server for processing. The results are returned to the client again through the server. All queries involve both an initialization and an execution phase. Thus, the cost of any query in this approach is:

(i) Initialization phase:

- The time for the client to open a URL connection with the web server.

(ii) Execution phase:

- The time for the applet to invoke, through the web server, the corresponding servlet passing the SQL statement as a parameter (stating explicitly the servers name and type of operation).
- The time for the servlet to execute the request obtain and return the entire result table to the client.

2.1.3 The RMI Approach

Java remote method invocation (RMI) is a java application interface for implementing remote procedure calls between distributed java objects, Monson-Haefel(2001). In RMI, the middleware consists of two objects; The first object is the application server which is responsible for handling requests by allowing clients to remotely invoke method on it. The second object is the installer object, which is used to start up the application server, and register it under a unique service name with the java virtual machine running on the web server.

The client calls a bind method of the RMI to obtain a reference of the application server. The parameters of this bind method are the URL of the machine on which the application server object was registered, and the unique service name with which it was registered. Using this reference, the client calls a method on the remote application server, executes the query at the database server, and returns the results table to the client as the method called.

The cost of the initial query is

(i) Initialization phase;

- The time for the applet to obtain a reference to the remote application server (bind to it).

(ii) Execution phase;

- The time for client to invoke a method on the application server passing the SQL statement as a parameter.
- The time for the application server to execute the SQL statement, obtain and return the results.

The time required for a subsequent query is the execution phase.

2.1.4 The CORBA Approach

CORBA, the Common Object Gateway Request Broker Architecture according to Monson-Haefel(2001) is an emerging distributed object standard that defines client/server relationships between objects in a common interface language. Unlike RMI, CORBA object can be implemented in any programming language. In order for a CORBA client object to utilize a CORBA server object, an implementation of

CORBA's basic functionality called the Object Request Broker (ORB), has to be loaded at both the client and the server sites.

The middleware in the CORBA approach is similar to the one in RMI approach. There is an application server object and an installer object. The installer object in this case is also used to load the ORB and register the application server with a unique service name with the ORB. After the client loads the ORB, it bounds to the application server via the gatekeeper by calling the special bind method and passing as parameter only the unique service name of the application server to carry out the request. The application server will execute the client's request on the database and return the result table as the return value of the method called.

The cost for the first query is;

(i) Initialization phase:

- The time for the client to initialize core ORB classes.
- The time for the client to bind to the application server.

(ii) Execution phase:

- The time for the client to invoke a method on the application server passing the SQL statement as a parameter.
- The time for the application server to execute the SQL, obtain and return the results to the client.

Any subsequent query requires execution phase only.

2.1.5 The Applets JDBC Approach

Applets that use directly the JDBC API. In this approach, the client applet downloads a JDBC driver and uses directly the JDBC API to connect to the database. After the client downloads the JDBC driver, it establishes database connectivity issuing JDBC calls on the Gateway, which are subsequently mapped on the database server. The cost is

1. Initialization phase:

a. The time for JDBC driver to be downloaded from the Web server and initiated by the applet.

b. The time for the applet to establish connection to the database through the gateway program.

2. Execution phase:

a. The time for the applet to issue an SQL statement to the database and obtain the results.

All subsequent queries require only the execution phase.

2.1.6 Java Mobile Agents (JMA)

Finally, in this subsection, we describe the approach of using mobile agents to achieve Web database connectivity. Mobile agents are processes capable of pausing their execution on one machine, dispatching themselves on another machine and resuming their execution on the new machine. The idea in the JMA approach is to use one or more mobile agents to implement the middleware and carry out the requests of the client. In the best variant, the results as well as subsequent queries are sent to and from the client using a message. This message passing is implemented implicitly as an RPC invocation from the client applet on the dispatched mobile agent.

In the JMA approach, the middleware consists of three components: The *DBMSaglet*, the (*Stationary) Assistant-aglet* and the *Aglet Router*. **The DBMS-aglet** can connect to a database and submit queries. Each database server is associated with an Assistant-aglet identified by a unique aglet ID and the URL. **An Assistant-aglet** provides the information necessary for a DBMS-agent to load the appropriate JDBC driver and connect to the database server. **An *Aglet Router*** is required to route aglets and messages, dispatched from a FijiApplet to any destination, and vice versa, because of the Java security restrictions. An aglet created within a FijiApplet is neither allowed to dispatch, nor to send a message directly to any URL other than the Web server URL. It then loads the JDBC-ODBC driver, connects to the database server and executes the client's request. After sending the query result in a message to the client, the DBMS-aglet remains connected to the database server, waiting for a message with new requests from the client. The cost of the initial query is:

1. Initialization phase:

a. The time for the client to create the DBMS-aglet

b. The time for the client to initialize the DBMS-aglet (SQL statement, etc.)

c. The time for the DBMS-aglet to travel to the remote database server

d. The time for the DBMS-aglet to negotiate with the assistant aglet

e. The time for the DBMS-aglet to establish connection with the database

2. Execution phase:

a. The time for the DBMS-aglet to query the database and send the results to the client using a message.

All subsequent requests required only one message from the client to DBMS-aglet, which includes the new SQL statement, plus the execution phase.

2.2 Socket Programme

```
// This program reads input from the user on the standard input stream and then forwards that text to
//  the EchoServer  by writing the text to a Socket. The Server echoes the input back through the
//Socket to the client. The client program reads and displays the data passed back to it from the
//server.

import java.io.*;
import java.net.*;
class EchoClient {
public static void main(String[] args) throws IOException {

      Socket echoSocket = null;
      PrintWriter out = null;
      BufferedReader in = null;

// These lines establish the socket connection between the client and server, and open a PrintWriter
//and BuferredReader on the socket.
```

```
     try {
         echoSocket = new Socket("Machine name", 7);
   out = new PrintWriter(echoSocket.getOutputStream(), true);
     in = new BufferedReader(new InputStreamReader(
           echoSocket.getInputStream()));

      } catch (UnknownHostException e) {
         System.err.println("Don't know about host: Machine name.");
         System.exit(1);

      } catch (IOException e) {
      System.err.println("Couldn't get I/O for " + "the connection to:
       Machine name.");
        System.exit(1);
      }
      BufferedReader stdIn = new BufferedReader(
                 new InputStreamReader(System.in));

// This loop reads a line at a time from the standard input stream and immediately sends it to the
//server by writing it to the PrintWriter connected to the socket.

        String userInput;
        while ((userInput = stdIn.readLine()) != null) {
            out.println(userInput);
            System.out.println("echo: " + in.readLine());
        }

// These statements close the readers and writers connected to the Socket and to the standard input
//stream and close the socket connected to the Server.

        out.close();
        in.close();
        stdIn.close();
        echoSocket.close();
   }
}


// Note that EchoClient both writes to and reads from its socket, thereby sending data to and receiving
//data from the Echoserver.
```

2.2.1 Explanation of the EchoClient Program

Line 1 in the program:

import java.io.*;

The java.io package provides an interface for serializing objects so that they can easily be written over the network. This application needs to write serialized objects over the network. Thus we need to implement this interface and define its behavior for this application.

Line 2 in the program:

import java.net.*;

The java.net class hierarchy provides the basic tools for developing both client and server application for the internet. With java.net.socket class which is one of the java.net packages, you can create a network socket to a server and begin sending data back and forth.

Line 3 in the program:

class EchoClient {

This line declares a class Echoclient, every java program begins in this way. You can name your program what ever you want; any nonempty string of letters and digits can be used for class name as long as it begins with a letter and contains no blanks. This is followed by left brace “ { “ , this is required immediately after class name. This is close by the last right brace “ } “ to form the program block.

Line 4 in the program:

public static void main(String[] args) throws IOException {

The word **public** means the contents of the following block are accessible from all other class.

static means that the method being defined applies to the class itself rather than the objects of the class.

void means that the method being defined has no return value.

main is the name of the method being defined.

The parenthesized string following main is the parameter list for the main method, which are local variables used to transmit information to the method from the outside world. It always has this form **(string[] args)** meaning this method has one parameter and its name is args and is an array of string object. The "**throws IOException"** clause allows the use of the readLine() method.

Line 5 in the program:

Socket echoSocket = null;

A socket is declared, named **echosocket** and is initialized to null(empty).

Line 6 in the program:

PrintWriter out = null;

A printWriter is declared, named **out** and initialized to null(empty). PrintWriter is an autoflush object that flushes the buffer on every call of println.

Line 7 in the program:

BufferedReader in = null;

A BufferedReader is declared, named "in" and initialized to null(empty). There are four buffered stream classes used to wrap unbuffered streams:

BufferedInputStream and BufferedOutputStream create buffered byte streams, while BufferedReader and BufferedWriter create buffered character streams.

Line 8 in the program:

try{

The three statements in the try block establish the socket connection between the client and the server and open a printWriter and BufferedReader on the socket.

Line 9 in the program:

EchoSocket = new Socket(“machine Name”, 7);

A new socket is created named echosocket. The socket constructor used here required the name of the machine and the port number connected to. The machine name connected to must be a fully qualified IP name and port number “7” is the port on which the echoserver listens.

Line 10 in the program:

Out = new printWriter(echosocket.getOutputStream(), true);

This statement gets the socket’s output stream and opens a PrintWriter on it, named “out”. PrintWriter as discussed earlier is an autoflush object that flushes the buffer out from the socket. The method “getOutputStream” means get out the stream. The word “true” will cause the output stream to plush automatically. Proper flushing is an important aspect of socket programming.

Line 11 in the program:

In = new BufferedReader(new InputStreamReader(echosocket.getInputSream()));

This statement gets the socket’s input stream and open a BufferedReader on it named “In”. The Object BufferedReader contains an object InputStreamReader through which the “getInputStream” method returns the inputstream object to the echosocket. To send data through the socket to the server, echoclient simply needs to write to the PrintWriter. To get the server’s response, “echoclient” reads from the BufferedReader.

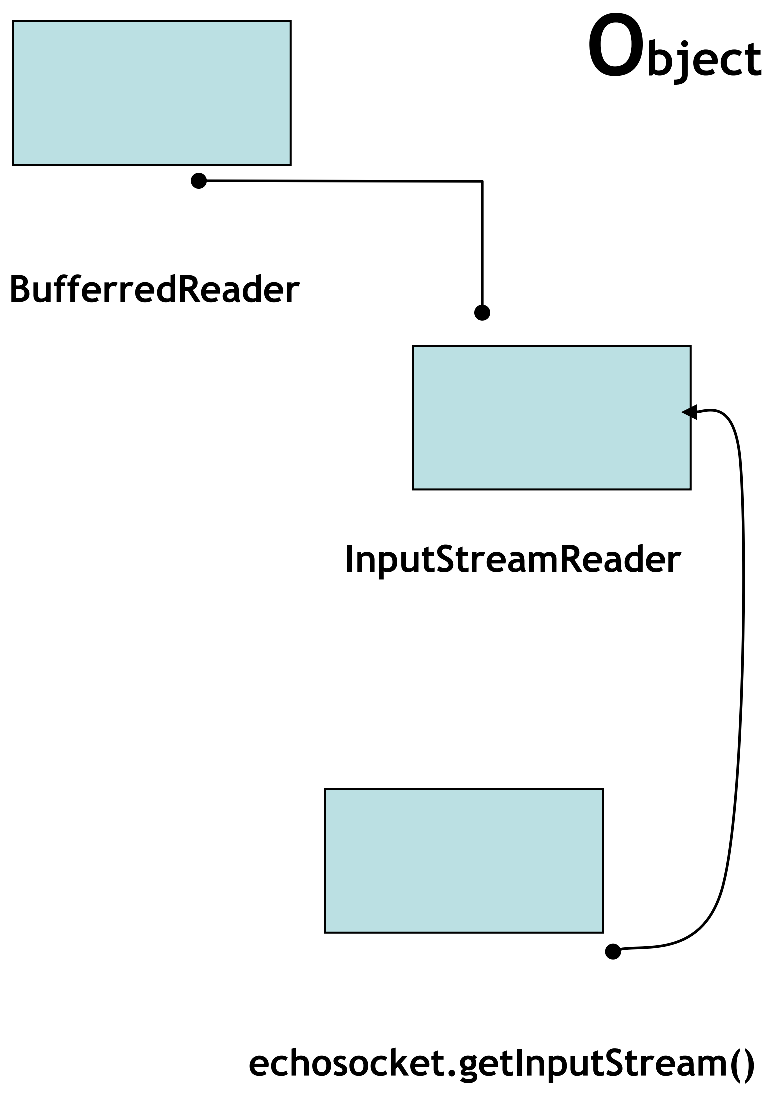


Fig. 7: The object InputStreamReader is an instance of the object BufferedReader through which the method echoSocket.getInputStream() is called.

Echosocket gets in “bytes” the inputstream through the InputStreamReader which converts the “bytes” to characters; the characters are wrapped into lines of text by the BufferedReader and named it “In”.

Line 12 in the program:

} catch (UnknownHostException e) {

If a method throws an exception, it must assume that exception must be “caught” and dealt with. One of the advantages of exception handling is that it allows you to concentrate on the problem you are trying to solve in one place and then deal with the errors from that code in another place. This particular object will catch Host exception as an error handling class and open a block for that purpose.

Line 13 in the program:

System.err.println(“Don’t know about host: machine name.”);

This displays a line of error massage, the string in quote, if the machine name to which you want to connect is not a fully qualified IP name. The result is printed to the console standard error stream by writing to the system.err, a better place to send error information than system.out.

Line 14 in the program:

System.exit(1);

This line is to quit from this block and close the block by “}” in the beginning of the next line.

Line 15 in the program:

} catch (IOException e) {

This will catch the Input/Output exception as an error handling class and open a block for that purpose.

Line 16 in the program:

System.err.println("Couldn't get I/O for " + "the connection to: Machine name.");

This will displays a line of error message, the string in quote with the "+" concatenating the two parts of the sentence if an Input/output error is encountered.

Line 17 in the program:

System.exit(1);

}

This line is to quit from this block and close the block by "}".

Line 18 in the program:

```
BufferedReader stdIn = new BufferedReader(
                new InputStreamReader(System.in));
```

BufferedReader: Read text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines. The buffer size may be specified, or the default size may be used. The default is large enough for most purposes.

In general, each read request made of a Reader causes a corresponding read request to be made of the underlying character or byte stream. It is therefore advisable to wrap a BufferedReader around any Reader whose read() operations may be costly, such as FileReaders and InputStreamReaders.

InputStreamReader: An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters .using a specified charset. The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

Each invocation of one of an InputStreamReader's read() methods may cause one or more bytes to be read from the underlying byte-input stream. To enable the efficient conversion of bytes to characters, more bytes may be read ahead from the underlying stream than are necessary to satisfy the current read operation.

For top efficiency, consider wrapping an InputStreamReader within a BufferedReader. For example:

```
 BufferedReader stdin
   = new BufferedReader(new InputStreamReader(System.in));
```

Here, the system read-in data in byte, The InputStreamReader converts the bytes to characters then wrapped by BufferedReader to text so as to provide efficient reading of characters, arrays, and lines.

Line 19 in the program

String userInput;

This line declares a variable named **userInput** as a string variable.

Line 20 and Line 21 in the program

while ((userInput = stdIn.readLine())!= null) { out.println(userInput);

This loop reads a line at a time from the standard input stream and immediately sends it to the server by writing it to the PrintWriter connected to the socket. Each time you type in a line of text and press enter, it will echo the same line of text back to you until you enter nothing!

Line 22 in the program

System.out.println("echo: " + in.readLine());

}

The last statement in the while loop reads a line of information from the BufferedReader connected to the socket. The readLine method waits until the server echoes the information back to echoclient. When readLine returns, echoclient prints the information to the standard output.

The while loop continuous until the user types in end-of-input character. That is, Echoclient reads input from the user, sends it to the Echo server, gets a response from the server and displays it, until it reaches the end-of-input, The while loop then terminates by “}” and the program continues.

Line 23 in the program

out.close();

This statement close the readers connected to the socket.

Line 24 in the program

in.close();

This statement close the writers connected to the socket.

Line 25 in the program

stdIn.close();

This statement close the writers connected to the standard input streams.

Line 26 in the program

echoSocket.close();

This statement closed the socket connection to server. A well-behaved program always cleans up after itself and this program is well-behaved. The order here is important because you should close any streams connected to a socket before you close the socket itself.

```
   }
}
```

The basic steps followed in this program are:

1. Open a socket.
2. Open an input stream and output stream to the socket.
3. Read from and write to the stream according to the server’s protocol.
4. Close the streams.
5. Close the socket.

This program was run in netbeans 5.5.

2.3 Echoserver program

Echo Server is a simple TCP server application which listens to port 7, and when it receives a data packet, it returns the data packet back to the client.

The Echoserver program is :

```
import java.io.*;
import java.net.*;

public class EchoServer{

public static void main(String[ ] args) throws IOException{

ServerSocket ss = null;
Socket cs = null;
PrintWriter out = null;
8. BufferedReader in = null;

9. try{
10. ss = new ServerSocket(3333);
       }
11. catch (IOException e) {
12. System.out.println ("Could not listen on port: 3333");
13. System.exit(1);
      }

14. try{
15. cs = ss.accept( );
      }
16. catch (IOException e) {
17. System.out.println("Accept failed: 3333");
18. System.exit(1);
       }

19. out = new PrintWriter (cs.getOutputStream ( ),true);
20. in = new BufferedReader (new InputStreamReader
      (cs.getInputStream( )));

21. String clientInput;

22. while (true) {
23. clientInput = in.readLine ( );
24. if (clientInput == null)
25. break;
26. out.println (clientInput);
       }
27. out.close( );
28. in.close ( );
29. ss.close ( );
30. cs.close ( );
       }
       }
```

2.3.1 Explanation of the Echoserver program

Line 1 in the program:

import java.io.*;

The java.io package provides an interface for serializing objects so that they can easily be written over the network. This application needs to write serialized objects over the network. Thus we need to implement this interface and define its behavior for this application.

Line 2 in the program:

import java.net.*;

The java.net class hierarchy provides the basic tools for developing both client and server application for the internet. With java.net.socket class which is one of the java.net packages, you can create a network socket to a client and begin sending data back and forth.

Line 3 in the program:

class Echoserver {

This line declares a class Echoserver, every java program begins in this way. You can name your program what ever you want; any nonempty string of letters and digits can be used for class name as long as it begins with a letter and contains no blanks. This is followed by left brace " { " , this is required immediately after class name. This is close by the last right brace " } " to form the program block.

Line 4 in the program:

public static void main(String[] args) throws IOException {

The word **public** means the contents of the following block are accessible from all other class.

static means that the method being defined applies to the class itself rather than the objects of the class.
void means that the method being defined has no return value.
main is the name of the method being defined.

The parenthesized string following main is the parameter list for the main method, which are local variables used to transmit information to the method from the outside world. It always has this form **(string[] args)** meaning this method has one parameter and its name is args and is an array of string object. The "**throws IOException"** clause allows the use of the readLine() method.

Line 5 in the program:

5. ServerSocket ss = null;

A ServerSocket is declared, named ss and is initialized to null(empty).

Line 6 in the program:

6. Socket cs = null;
A socket is declared, named cs and is initialized to null(empty).

Line 7 in the program:

7. PrintWriter out = null;
A printWriter is declared, named **out** and initialized to null(empty). PrintWriter is an autoflush object that flushes the buffer on every call of println.

Line 8 in the program:

8. BufferedReader in = null;
A BufferedReader is declared, named "in" and initialized to null(empty).

Line 9 in the program:

9. try{

This open a block to create a ServerSocket object.

Line 10 in the program:

10. ss = new ServerSocket(3333);
}
A new ServerSocket object is created named ss, this creates a stream socket on the specified port number on which the server is listening and close the try block.

Line 11 in the program:

11. catch (IOException e) {
This will catch the Input/Output exception as an error handling class and open a block for that purpose.

Line 12 in the program:

12. System.out.println ("Could not listen on port: 3333");

This will displays a line of error message, the string in quote with the specified port if an Input/output error is encountered.

Line 13 in the program:

13. System.exit(1);
}
This line is to quit from this block and close the block by "}".

Line 14 in the program:

14. try{
This open a block to call the ServerSocket's accept method to accept a socket connection requested by the choclient.

Line 15 in the program:

15. cs = ss.accept( );
}

The statement in the try block call the ServerSocket's accept method to accept a connection request from the ClientSocket to establish the socket connection between the client and the server.

Line 16 in the program:

16. catch (IOException e) {

This will catch the Input/Output exception as an error handling class and open a block for that purpose.

Line 17 in the program:

17. System.out.println("Accept failed: 3333");

This will displays a line of error message, the string in quote with the specified port if an Input/output error is encountered.

Line 18 in the program:

18. System.exit(1);
}

This line is to quit from this block and close the block by "}".

Line 19 in the program:

19. out = new PrintWriter (cs.getOutputStream ( ),true);

This statement gets the socket's output stream and opens a PrintWriter on it, named "out". PrintWriter called the clientsocket's "getOutputStream" method.The word "true" will cause the output stream to plush automatically. Proper flushing is an important aspect of socket programming.

Line 20 in the program:

20. in = new BufferedReader (new InputStreamReader (cs.getInputStream( )));

This statement gets the socket's input stream and open a BufferedReader on it named "In". The Object BufferedReader contains an object InputStreamReader through which the "getInputStream" method returns the inputstream object to the clientsocket 'cs'. To send data through the socket to the client, echoserver simply needs to write to the PrintWriter. To get the client's request, "echoserver" reads from the BufferedReader.

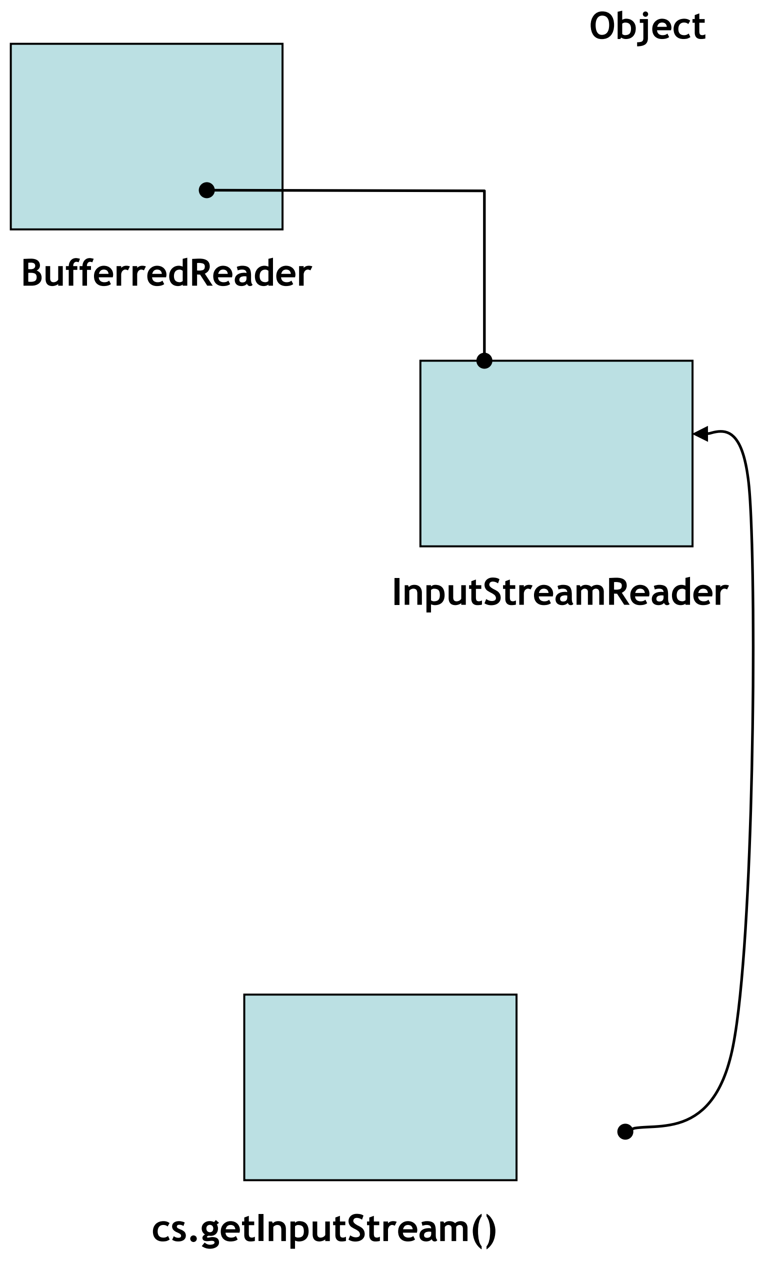


Fig. 8: The object InputStreamReader is an instance of the object BufferedReader through which the method cs.getInputStream() is called.

Client socket ‘cs’ gets in “bytes” the inputstream through the InputStreamReader which converts the “bytes” to characters; the characters are wrapped into lines of text by the BufferedReader and named it “In”.

Line 21 in the program:

21. String clientInput;

This line declares a variable named **clientInput** as a string variable.

Line 22 in the program:

22. while (true) {

This loop reads a line at a time from the standard input stream and immediately sends it to the client by writing it to the PrintWriter connected to the socket. Each time you type in a line of text and press enter, it will echo the same line of text back to you until you enter nothing!

Line 23 in the program:

23. clientInput = in.readLine ( );
This statement reads a line at a time sent to it by the server.

Line 24 in the program:

24. if (clientInput == null)
This conditional statement determines if the line sent by the server is null that is nothing sent.

Line 25 in the program:

25. break;
This is to terminate the communication if nothing is sent by the server.

Line 26 in the program:

26. out.println (clientInput);
 }
This statement will echo the same line of text back to the client until nothing entered!

Line 27 in the program:

27. out.close( );
This statement close the PrntWriter connected to the socket.

Line 28 in the program:

28. in.close ( );
This statement close the BufferedReaders connected to the socket.

Line 29 in the program:

29. ss.close ( );
This statement close the ServerSocket connected to the socket.

Line 30 in the program:

30. cs.close ( );
 }
 }
This statement close the ClientSocket connected to the socket then close the main method and finally close the class.

The Echoserver program was run first, while listening for incoming connection request then run Echoclient to request for a connection. When connection is established the two machines communicates sending and receiving data across the internet.

2.4 Client and Server Communication

The Java jva.net package hierarchy provides the basic tools for developing both client and server applications for the internet. With the java.net.socket class, we can create a network socket to a server and begin sending data back and forth. With the java.net.ServerSocket class, we can implement a network server application that will listen on he network for incoming client request , set up the necessary network communication, and begin communicating, Berg and Fritzinger(1997).

The Internet (or an intranet) is a network that links different computers together. We must understand how the output from these systems actually gets from the server to the browser, which means that we have to learn how the Internet and the Web work. The server use standard Internet conventions and protocols to send resources to a client. The most important parts of this interchange are:

- A TCP/IP network to connect the server to the client
- A software communication port to serve as a collection point for incoming requests
- A transfer protocol called HTTP to govern how server and client communicate
- A client program called a web browser to allow users to request and receive resources from the server
- A uniform resource locator (URL) to allow the browser to find a particular resource
- A MIME type to tell the browser what to do with resources once received from the server.

The following sections briefly describe each of these parts.

2.4.1 The TCP/IP Network Protocol

Browsers connect to a server using the TCP/IP networking protocol. Although there are a number of different types of networking protocols web systems only work with TCP/IP. Every machine on a TCP/IP network is identified by a four-part IP address. Each number in the address can range from 0 to 255, and the four numbers are separated by periods. For example, 253.4.99.17 might be the address for the machine running the human resources department's web server. Every machine on a TCP/IP network has a unique IP address.

Most TCP/IP networks have a special class of servers called Domain Name Servers (DNSs). Their job is to translate IP addresses into meaningful hostnames that are easy to remember. For example, assigning the address 253.4.99.17 to the name "HumanResources" in the DNS allows users to refer to the human resources server as "HumanResources," rather than its actual IP address.

2.4.2 The Communication Port

A *software port* (as opposed to a physical hardware port) is a common reference point on the server that is used to exchange messages. Each TCP/IP-based networking application is assigned a specific port that it monitors for incoming requests. Client programs that need to communicate with the server connect to the server's assigned port. Once connected, the two systems exchange information according to a standard protocol (HTTP, FTP, etc.). Each port is identified by a port number, its ordinal position in the range of all ports.

2.4.3 The HyperText Transfer Protocol (HTTP)

A transfer protocol is a convention that governs how systems exchange information. Take, for example, a phone conversation. When you call someone, you (hopefully!) don't start blurting out whatever comes to mind as soon as they pick up the receiver.

Instead, your conversation follows a set pattern that civilized society has agreed upon to make communication more efficient:

1. I initiate a conversation by calling you.
2. You say "Hello."
3. I identify myself.
4. We exchange a message.
5. We say "Goodbye."
6. We hang up.

This sort of formalized exchange is the idea behind a protocol: it lets the sender and receiver knows the order in which communications will occur. While computers use much more formalized systems than humans, the idea is basically the same. The server follow a standard Internet protocol called HyperText Transfer Protocol (HTTP) to communicate with client web browsers.

By convention, several special TCP ports are associated with specific protocols. For example, port 21 is usually used for FTP, port 25 is used for SMTP (a common email protocol), and port 80 is used for HTTP.

2.4.4 The Web Browser

Users request information from a server using a web browser such as Microsoft Internet Explorer, firefox or Netscape Communicator. The browser is responsible for presenting web content on these servers to the user. In the early days of the Web, a browser could handle only basic HTML and text documents, but the explosion of web content has turned the browser into an information kiosk, multimedia center, and minicomputer all rolled into one. For example, most modern browsers can display an HTML document filled with pictures, sounds, and even movies. With the advent of Java, the browser has become a *virtual machine*, a computer within a computer capable of running Java programs.

2.4.5 The Uniform Resource Locator

Uniform Resource Locators (URLs) are used to request a resource from a server independently of the operating system used on the machine. A URL abstracts the machine name, resource path, and resource name into a string with the following syntax:

protocol://server:port/path/resource?query_string

protocol

> Specifies the network protocol that the browser and the server use to communicate. The most common values are HTTP and FTP.

server

> Identifies the name of the machine that hosts the resource. Although you can use the machine's IP address, it's better to use the name defined in the DNS since it helps isolate the URL from the network reconfiguration.

port

> Specifies the TCP port used by the server. If the port is omitted, then port 80 is used by default.

path

> Specifies the virtual directory or schema containing the resource. The path usually maps to either a virtual directory mapping on the web server or, to a

Database Access Descriptor (DAD), a logical name used to map a procedure call in a URL to the database schema in which it resides.

resource name

Typically specifies the actual name of the file to return. If the name is omitted, the listener returns a default file, if one is available. The name of the default file varies: *index.html* is used on many Unix systems, and *default.htm* is usually used on Windows NT systems.

query string (optional)

Optionally passes parameters to dynamic resources. The string begins with a question mark (?) and is followed by ampersand (&)-delimited sets of name/value pairs. Each name/value pair consists of a parameter name followed by an equals sign (=) and a value for the parameter.

2.4.6 The Resource MIME Type

Every resource is associated with a *MIME type* that tells the browser what to do with the resource once the transfer is complete (e.g., display it in the main window, launch a file viewer, and so on). MIME, which stands for Multipurpose Internet Mail Extensions, is a standard for exchanging various types of files (such as images, text, and video) over the Internet so that each computer platform, whether NT, Unix, or VMS, will interpret and correctly handle the resource's contents.

MIME types describe the data format using two parts. The first part, the *type*, identifies the resource's general format, such as text, image, or audio. The second part, the *subtype*, identifies the resource's specific data format. For example, the subtypes for the image type include gif and jpeg. The type and subtype are delimited with a slash (/); for example, a picture's full MIME type could be image/gif or image/jpeg.

Browsers must be configured to handle each MIME type. Almost all browsers can display text/plain, text/html, and image/jpeg documents without any extra configuration. When a browser receives a document with a MIME type it doesn't recognize, it asks the user to select a helper program to display the document. This is similar to selecting a file association based on a file extension in Windows (i.e., mapping the *.doc* extension to the Microsoft Word application). Once the user makes an association, subsequent requests for that MIME type are opened using the associated application.

2.5 Dynamic Web Application

Building a website requires familiarity with specific concepts, as well as software products and tools. This section provides a brief overview of these requirements, so get a better understanding of the difference between building a static versus a dynamic website.

A static site displays the same information to all users, whenever they access it. Consider a site that makes a book available via the Internet. The site's content is the same for all users at all times: each page of the book, with links to the next and the previous page.

What would happen if the site owners decided to allow users to add comments on each page? The site would become dynamic: over time its contents would change as users added comments. If the site became very popular, the owners might decide to add another feature that enabled moderators to reject inappropriate comments. In this case, the site would work one way when accessed by a regular user and another way when accessed by a moderator.

A key difference between a static website and a dynamic website is the source of the information that is displayed by the site.

Static websites

All the content data for a static website is contained in the files that correspond to the pages of the site. These files store data in a special format, HTML.

Returning to the example of static website displaying the pages of a book, each page of the book would be stored as HTML in a separate file with the .html extension. In order to change the contents of a book page, you would have to edit the corresponding HTML file.

To run a static website you need a *web server*, which is a software application that:

- Receives requests for displaying a web page
- Reads the corresponding file
- Sends the contents of the file back to the computer that made the request (see the Figure below).

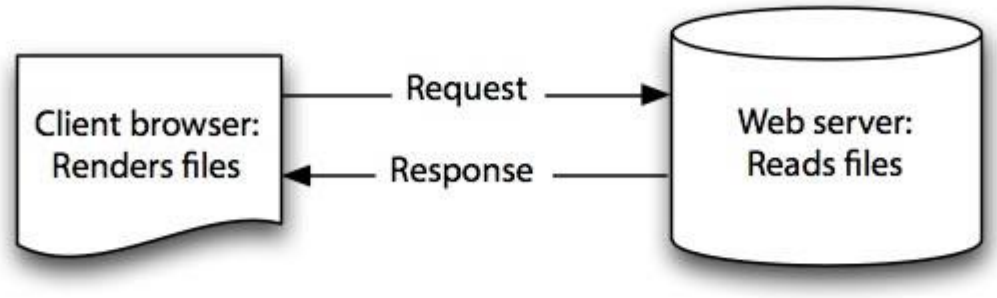


Fig. 9. A typical setup for static websites(Dumitrascu, 2008)

Examples of common web servers include Apache and Microsoft Internet Information Services (IIS).

Dynamic websites

In contrast, dynamic websites must be able to display information that changes over time or varies depending on the user who accesses it. For this reason dynamic sites use a software program that acts as a data repository, usually a database system. Information can be added to or retrieved from this repository dynamically.

Furthermore, dynamic sites must be able to integrate the static parts of a site—for example, the book contents in the book website—with the dynamic ones, for example, comments. This mix of static and dynamic information is realized, in addition to using a web server, by:

- Writing the pages in a way, that mixes the static part (HTML) with instructions for processing dynamic data.
- Running software on the server that interprets the instructions in dynamic pages and executes them to retrieve information (data) from the database, add information to the database, or modify it. This software is commonly known as an *application server*.

Therefore, in addition to a web server, the following software is required in order to run a dynamic site(see the figure below) :

- **Database server:** Software that manages the database. Some commonly used database servers for the web are **Mysql**, **Oracle** and **SQL server.**
- **Application server:** Examples of application servers are **Apache tomcat application server**, Internet Information Services,**IIS** (which acts as a web server and an application server), and the **glassfish application server**.

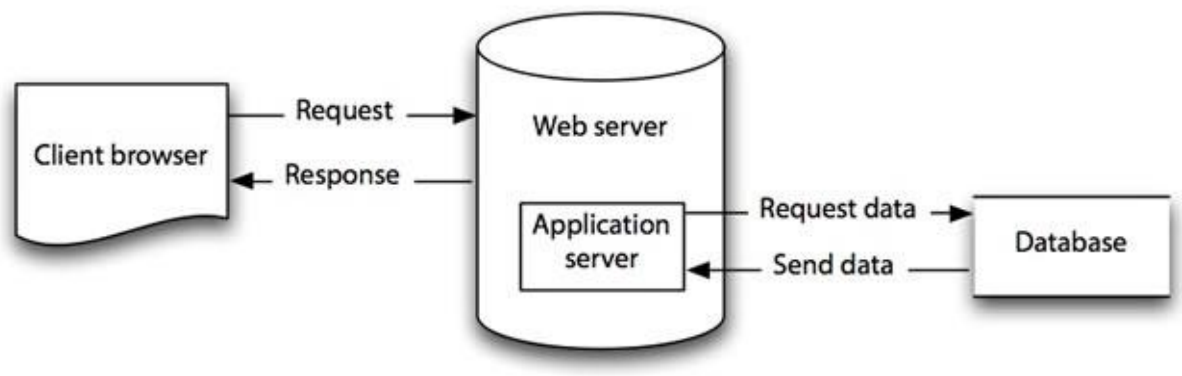


Figure 10. A typical setup for dynamic websites(Dumitrascu, 2008)

2.6 Introduction to Jsp

JavaServer Pages (JSP) is a technology based on the Java language and enables the development of dynamic web sites. JSP was developed by Sun Microsystems to allow server side development. JSP files are HTML files with special Tags containing Java source code that provide the dynamic content. JSP source code runs on the web server in the JSP Servlet Engine. The JSP Servlet engine dynamically generates the HTML and sends the HTML output to the client's web browser.

JSPs are built on top of SUN Microsystems' servlet technology. JSPs are essentially an HTML page with special JSP tags embedded. These JSP tags can contain Java code. The JSP file extension is .jsp rather than .htm or .html. The JSP engine parses the .jsp and creates a Java servlet source file. It then compiles the source file into a class file, this is done the first time and this is why the JSP is probably slower the first time it is accessed. Any time after this the special compiled servlet is executed and is therefore returns faster.

Following steps are followed when any JSP page is called through browser.

(i) The user goes to a web site made using JSP. The user goes to a JSP page (ending with .jsp). The web browser makes the request via the Internet.

(ii) The JSP request gets sent to the Web server.

(iii) The Web server recognises that the file required is special (.jsp), therefore passes the JSP file to the JSP Servlet Engine.

(iv) If the JSP file has been called the first time, the JSP file is parsed, otherwise go to step 7.

(v) The next step is to generate a special Servlet from the JSP file. All the HTML required is converted to println statements.

(vi) The Servlet source code is compiled into a class.

(vii) The Servlet is instantiated, calling the init and service methods.

(viii) HTML from the Servlet output is sent via the Internet.

(ix) HTML results are displayed on the user's web browser.

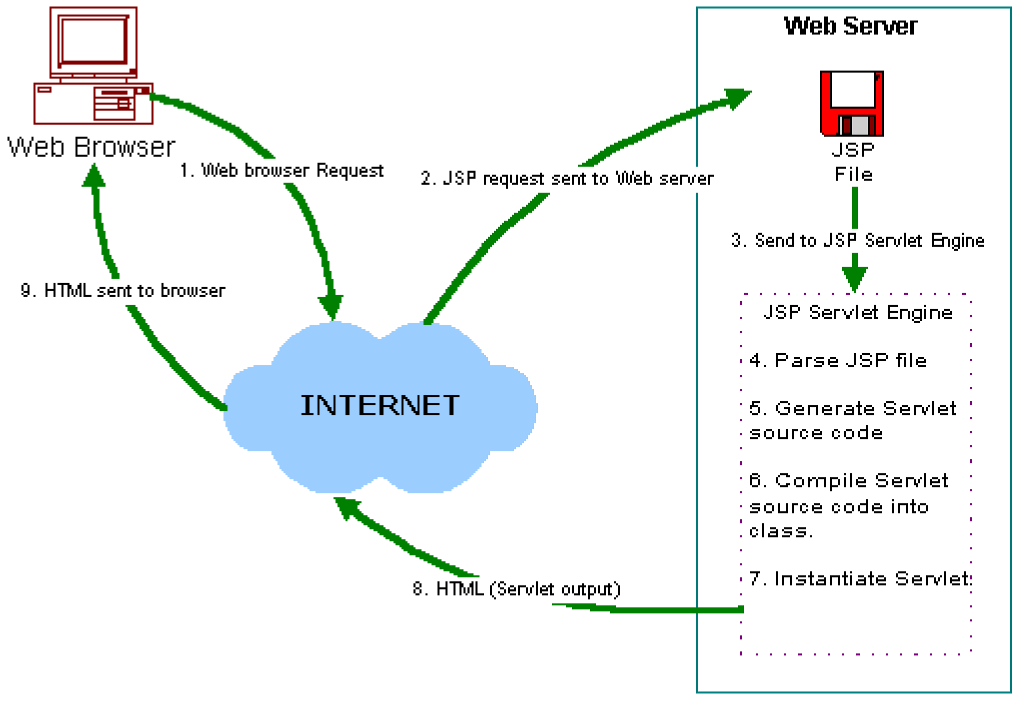


Fig.11: Steps Required For A Jsp Request(Sun,2004)

2.6.1 Connection Pooling

Establishing JDBC connections is resource-expensive, especially when the JDBC API is used in a middle-tier server environment, such as connectors *for* JDBC running on a Java-enabled web server. In this type of environment, performance can be improved significantly when connection pooling is used. *Connection pooling* means that connections are reused rather than created each time a connection is requested. To facilitate connection reuse, a memory cache of database connections, called a *connection pool,* is maintained by a connection pooling module as a layer on top of any standard JDBC driver product.

Connection pooling is performed in the background and does not affect how an application is coded; however, the application must use a DataSource object (an object implementing the DataSource interface) to obtain a connection instead of using the DriverManager class. A class implementing the DataSource interface may or may not provide connection pooling. A DataSource object registers with a JNDI naming service. Once a DataSource object is registered, the application retrieves it from the JNDI naming service in the standard way.

For example:

```
Context c = new InitialContext();
    return (DataSource)c.lookup("java:comp/env/jdbc/connectionPool ");
```

If the DataSource object provides connection pooling, the lookup returns a connection from the pool if one is available. If the DataSource object does not provide connection pooling or if there are no available connections in the pool, the lookup creates a new connection. The application benefits from connection reuse without requiring any code changes. Reused connections from the pool behave the same way as newly created physical connections. The application makes a connection to the database and data access works in the usual way. When the application has finished its work with the connection, the application explicitly closes the connection.

For example:

```
Connection connection = datasource.getConnection();

// Do some database activities using the connection...
connection.close();
```

The closing event on a pooled connection signals the pooling module to place the connection back in the connection pool for future reuse.

CHAPTER 3:

SYSTEM ANALYSIS AND DESIGN

3.0 Introduction

This chapter focuses on the system analysis and design methodology used. This work used object-oriented systems analysis and design methodology and elements from other approaches, such as relational database system design which is part of the systems development requirements. Here system requirements are identified and installed and relational database system designed for the current business environment.

3.1 Getting softwares for the application

This section describes how to download and install softwares necessary to implement the application. The softwares are sun java system application server, Apache Web Server, MySQL Server and connector J(JDBC Driver for MySQL).

3.1.1 To Install Sun Java System Application Server 9.1 on Windows

If you have downloaded the Sun Java System Application Server standalone installer.

i) Navigate to the directory where you downloaded the .exe file
ii) Double-click the .exe file to start the installation program.
iii) Follow the instructions on the wizard screens of the installation program.
iv) Enter the directory where you want to install Application Server.
v) Select the components to install.

The following components are available for installation:

- Node Agent – All machines that contain application server instances should have a node agent installed.

- High Availability Database Server -This component stores session information so requests can be failed over if you are using the load balancing plugin.
- Load Balancing Plugin - Install the load balancing plugin on the machine where a web server is installed. If you do not already have a web server installed on the machine where you are installing the load balancing plugin, you cannot continue to install the load-balancer plugin.
- Domain Administration Server and Administration Tool – The Domain Administration Server (DAS) acts as a central repository for applications and configuration information for server instances, even if the instances are located on a remote machine. It includes graphical and command-line administration tools.
- Sample Applications - Samples come with the source, schema, Ant build scripts, and EAR files. Any existing data associated with the database-related samples is available in the included JavaDB database.
- Specify the directory where you want to install JDK 5. If you already have JDK 5 installed, specify the path to the JDK 5 installation.
- In the Admin Configuration page, enter the following:

■ **Admin UserName** – Name of the user who administers the server.

■ **Password** – Admin user’s password to access the Admin Server (8-character minimum).

■ **Master Password** – The master password (8-character minimum).

■ **Admin Port** – Administration port number for initial server instance. The default port is 4848.

■ **HTTP Port -** Port number to access the default server instance

■ **HTTPS Port -** Secure port number to access the default server instance

Verify the installation by following the instructions of the *Sun Java System Application Server 9.1 Quick Start Guide*, located in *install-dir*\docs\QuickStart.html or on docs.sun.com.

3.1.2 Downloading and Installing Apache Web Server

A Web server handles the HTTP protocol. When the Web server receives an HTTP request, it responds with an HTTP response, such as sending back an HTML page. To process a request, a Web server may respond with a static HTML page or image, send a redirect, or delegate the dynamic response generation to some other program such as CGI scripts, JSPs (JavaServer Pages), servlets, ASPs (Active Server Pages), server-side JavaScripts, or some other server-side technology. Whatever their purpose, such server-side programs generate a response, most often in HTML, for viewing in a Web browser.

Understand that a Web server's delegation model is fairly simple. When a request comes into the Web server, the Web server simply passes the request to the program best able to handle it. The Web server doesn't provide any functionality beyond simply providing an environment in which the server-side program can execute and pass back the generated responses. The server-side program usually provides for itself such functions as transaction processing, database connectivity, and messaging.

While a Web server may not itself support transactions or database connection pooling, it may employ various strategies for fault tolerance and scalability such as load balancing, caching, and clustering—features oftentimes erroneously assigned as features reserved only for application servers.

i. If you haven't done so already, download the Apache HTTPD Web Server from the Apache web site . Be sure to download the **apache_2.2.11-win32-x86-no_ssl** MSI installer here:

(http:// httpd.apache.org/download.cgi)

ii. Save the file to your Windows Desktop.

Double click the msi file saved on your Windows Desktop. You will see a window,

Apache Web Server Install Window

iii. Click "Next>".

iv. Click the radio button "I accept the terms in the license agreement"

v Click "Next>".

vi. On the next window, again click "Next>".

vii. Next, fill in all the text boxes with the following information:

"Network Domain": localhost

"Server Name": localhost

"Administrator's Email Address": your email address

viii. Make sure the radio button "for all users, on port 80, as a service - recommended" is selected.

ix. Click "Next>".

x. On the next window, click the radio button "Custom", and then click "Next>".

xi. On the next window highlight "Apache HTTP Server" and click the "change" button.

xii. We are going to install all the packages and scripts in the folder C:\Server\Apache2\ (assuming C: is your main hard drive). So in the text box "Folder name:" type in "C:\Server\Apache2\". The ending backslash is important.

xiii. After you have typed in the path, click "OK" and then click "Next>". At this point, you should see a window, **Apache Ready to Install**

xiv. Click "Install" to begin the installation.

xv. Once the Apache installation software has finished installing all the files on your computer, you will see a final window letting you know the installation was a success. Click the "Finish" button.

EDIT TIPS

To check that the software was in fact installed successfully, open your favorite browser and type "http://localhost/" into the address bar. You should see an **Apache Success** page.

3.1.3 Downloading and Installing MySQL Database Server on Your Windows PC

MySQL is a popular Open Source database management system commonly used in web applications due to its speed, flexibility and reliability. MySQL employs SQL, or *Structured Query Language*, for accessing and processing data contained in databases.

Steps

i) Download the free MySQL Server Community Edition -- you can download the software from the MySQL website(www.mysql.com). Be sure to download the Windows (x86) version (mysql-essential-5.0.8-win32.msi), which includes a Windows Installer. Save the file on your Windows Desktop.

It's time to install MySQL. The installation file comes as a .zip file. Double click the file. Your unzipping software should open the file and show you a list of files inside the archive. There should be only 1, "Setup.exe". There is no need to extract the file from the archive, just go ahead and double click the "Setup.exe" file. After you've done that, you should see a window, **MySQL Install Window**

Click "Next>".

ii) On the next window, click the radio button "Custom", and then click "Next>". Since we have installed Apache in C:\Server, we are going to install MySQL in the same directory.

iii) On the next window, highlight "MySQL Server", and then click the "change" button.

iv) On the next window, in the text box "Folder name:", change the directory to "C:\Server\MySQL\". Make sure you include the ending backslash. Then click "OK".

v) On the next window, click "Next>".

Now MySQL is ready to install. You should see a window, **MySQL Ready to Install.**

vi) Click "Install".

vii) Once the installation is complete, you will be presented with a "MySQL Sign-Up" window. Click the radio button "Skip Sign-Up", and then click "Next>". You can sign-up for MySQL.com later if you like.

If the installation was successful, you should see a **MySQL success** window

MySQL was successfully installed. Now we want to configure it. Leave the check box "Configure the MySQL Server Now" checked, and click "Next>". After you do that, you should see a **MySQL Configure** window.

Click "Next>".

viii) On the next window, click the radio button "Standard Configuration", and then click "Next>".

xiv) On the next window make sure "Install As Windows Service" and "Launch the MySQL Server Automatically" check boxes are checked. Click "Next>".

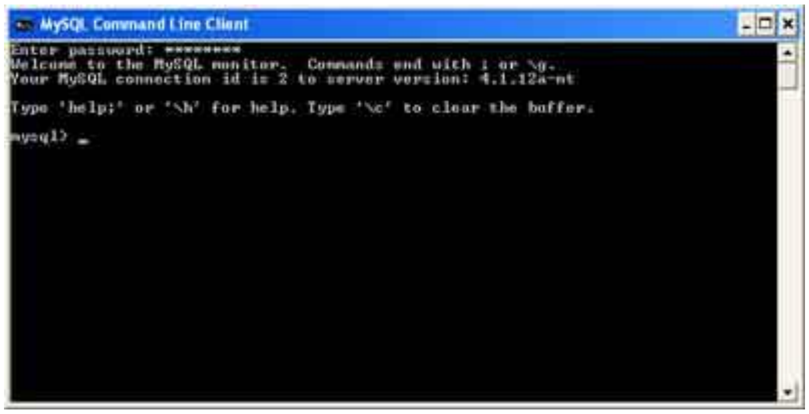
xv) On the next window you need to create a root password. Type in what you want your root password be and make sure "Enable root access from remote machines" is checked. Make sure you choose a difficult to guess password and write it down so you don't forget it. Click "Next>".

xvi) On the next window, click "Execute". This will start the MySQL

server After mysql has done it things, cick “finish”.

xvii) Now we need to make sure MySQL was in fact installed successfully. On the Windows task bar, click "Start">"All Programs ">"MySQL">"MySQL Server 5.0">"MySQL Command line client".

xviii) This will open a command window that is asking you for a password. Enter the password that you chose as your root password. Hit enter and you should see a window that looks like this:



```
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.1.12a-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> _
```

MySQL Command Line Client

ix) And that's it, you're done installing MySQL.

3.1.4 Downloading and Installing Connector J(JDBC Driver for MySQL),

To use MySQL with JDBC, MySQL Connector/J a JDBC driver that allows programs to use JDBC to interact with MySQL need to be installed. MySQL connector/J can be downloaded from

Dev.mysql.com/downloads/connectorj/5.0.5.html

To install MySQL Connector/J:

i) Download mysql-connector-java-5.0.5.zip

ii) Open mysql-connector-java-5.0.5.zip with file extractor, such as winzip (www.winzip.com). Extract its contents to the C:\ drive. This will create a directory named mysql-connector-java-5.0.5.

3.1.5 Set-Up Mysql User Account

For the Mysql to execute correctly, you need to set up a user account that allows users to create, delete, and modify a database. After Mysql is Installed, follow the steps below to set up a user account.

If you've just installed MySQL or are just using it for the first time, there are a few things that you need to do before using it in your applications. All of these operations are done via the mysql command line client, so open a terminal and type:

```
mysql -u root
```

This should put you at a mysql prompt like this:

```
mysql>
```

The first thing to do is set a password on the root account, for security purposes; having no password is highly insecure. Type the following command, replacing *secret* with your new password, and leaving the quotes in place:

```
mysql> UPDATE mysql.user SET Password = PASSWORD('secret')
    -> WHERE User = 'root';

mysql> FLUSH PRIVILEGES;
```

Now type "exit" to quit the mysql client, then start it again like this:

```
mysql -u root -p
```

It will now prompt you for your password.

Second, you'll want to delete the built-in anonymous access accounts:

```
mysql> DELETE FROM mysql.user WHERE User = '';

mysql> FLUSH PRIVILEGES;
```

Third, you need to create a database for your application(s) to use:

```
mysql> CREATE DATABASE Examination;
```

Finally, you need to create a user account in MySQL for your application(s) to use (replace *username* and *password* with an actual username and a password:

```
mysql> GRANT ALL PRIVILEGES ON Examination.* TO 'username'@'localhost'
    -> IDENTIFIED BY 'password';

mysql> GRANT ALL PRIVILEGES ON Examination.* TO 'username'@'%'
    -> IDENTIFIED BY 'password';
```

Now type "exit" to quit the mysql client.

3.2 What Is A Database?

A database is a means of storing information in such a way that information can be retrieve from it. A relational database is one that presents information in tables with rows and columns. A table is referred to as a relation in sense that it is a collection of objects of the same type(rows).According to Gillenson(1984) Data in a table can be related according to common keys or concepts and the ability to retrieve related data from a table is the basis for the term relational database. A database management system(DBMS) handles the way data is stored, maintained and retrieved. In the case of a relational database, relational database management system(RDBMS) performs these tasks. DBMS is a general term that includes RDBMS.

3.2.1 Creating table In Mysql Database

To create table in mysql database, identify all the entities that will form the tables of the database and create table for each entity. A MySQL table is completely different from the normal table that you eat dinner on. In MySQL and other database systems, the goal is to store information in an orderly fashion. The table gets this done by making the table up of columns and rows. Before you can enter data (rows) into a table, you must first define what kinds of data will be stored (columns). We are now going to design a MySQL table StudentResult for our database Examination. Use a CREATE TABLE statement to specify the layout of your table:

```
mysql> CREATE TABLE result(
        -> Sno INT NOT NULL AUTO_INCREMENT,
        -> IDno VARCHAR(12) NOT NULL,
        -> CourseCode VARCHAR(6) NOT NULL,
        -> Marks INT NOT NULL,
       -> PRIMARY KEY(Sno));
```

The first part of the *mysql_query* told MySQL that we wanted to create a new table. The two capitalized words are reserved MySQL keywords.

The word " result " is the name of our table, as it came directly after "CREATE TABLE". It is a good idea to use descriptive names when creating a table, Clear names will ensure that you will know what the table is about when revisiting it a year after you make it.

'Sno INT NOT NULL AUTO_INCREMENT'

Here we create a column "Sno" that will automatically increment each time a new entry is added to the table. This will result in the first row in the table having an Sno = 1, the second row Sno = 2, the third row Sno = 3, and so on.

The column "Sno" is not something that we need to worry about after we create this table, as it is all automatically calculated within MySQL.

Reserved MySQL Keywords:
Here are a few quick definitions of the reserved words used in this line of code:

> **INT -** This stands for integer or whole number. 'Sno' has been defined to be an integer.
> **NOT NULL -** These are actually two keywords, but they combine together to say that this column cannot be null. An entry is NOT NULL only if it has some value, while something with no value is NULL.
> **AUTO_INCREMENT** - Each time a new entry is added the value will be incremented by 1.

IDno VARCHAR(12) NOT NULL,

Here we make a new column with the name "IDno"! VARCHAR stands for "variable character". "Character" means that you can put in any kind of typed information in this column (letters, numbers, symbols, etc). It's "variable" because it can adjust its size to store as little as 0 characters and up to a specified maximum number of characters.

CourseCode VARCHAR(6) NOT NULL,

Here we make a new column with the name " CourseCode "! VARCHAR stands for "variable character". "Character" means that you can put in any kind of typed information in this column (letters, numbers, symbols, etc). It's "variable" because it can adjust its size to store as little as 0 characters and up to a specified maximum number of characters.

Marks INT NOT NULL,

The “Marks” column stores an integer. Notice that there are no parentheses following "INT". MySQL already knows what to do with an integer. The possible integer values that can be stored in an "INT" are -2,147,483,648 to 2,147,483,647, which is more than enough to store someone's Marks.

PRIMARY KEY(Sno));

PRIMARY KEY is used as a unique identifier for the rows. Here we have made "Sno" the PRIMARY KEY for this table. This means that no two Snos can be the same, or else we will run into trouble. This is why we made "Sno" an auto-incrementing counter in the previous line of code.

3.3 Connecting to a MySQL Database in NetBeans IDE

This section describes how to configure MySQL on your system and set up a connection to the database from NetBeans IDE. Once connected, you can begin working with MySQL in the IDE's Database explorer by creating new tables, populating tables with data, and running SQL queries on database structures and content. MYSQL is a popular Open Source database management system commonly used in web applications due to its speed, flexibility and reliability. MySQL employs SQL, or *Structured Query Language*, for accessing and processing data contained in databases.

3.3.1 Registering the Database in NetBeans IDE

Now that you have the database server installed and configured, and have created a new database, you can register the MySQL server in NetBeans IDE. Begin by examining the functionality offered by the Database explorer located in the IDE's Runtime window (Ctrl+5). The Database explorer is represented by the Databases node (🛢). From this interface you can connect to databases, view current connections, add database drivers, as well as create, browse or edit database structures.

You can use the Database explorer to register MySQL in the IDE. There are two simple steps that need to be performed:

i Adding the Driver to the IDE

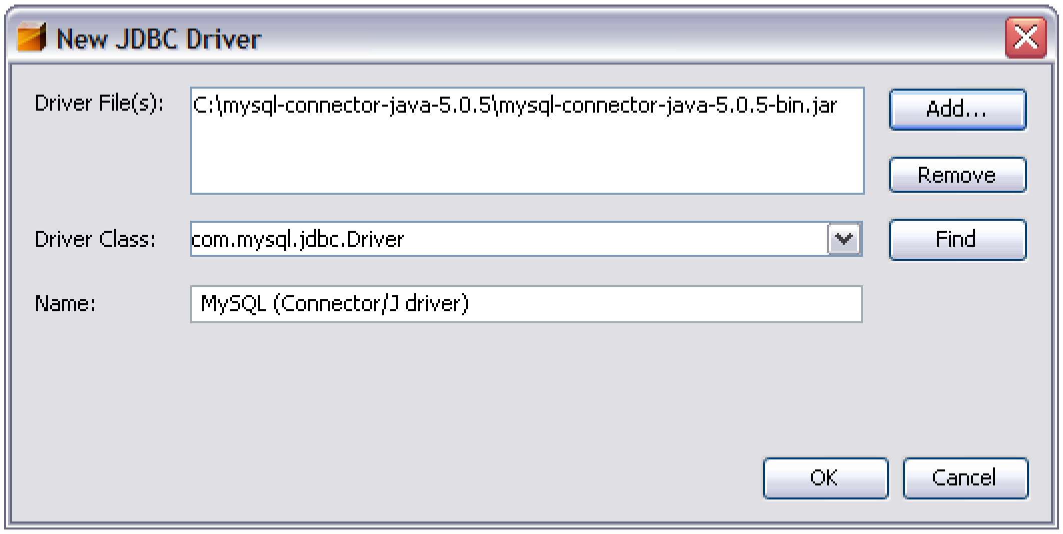
ii, Creating a Database Connection

a) Adding the Driver to the IDE

In order to allow NetBeans IDE to communicate with a database, you need to employ a Java-based *driver*. Generally speaking, drivers in NetBeans IDE use the JDBC (*Java Database Connectivity*) API to communicate with databases supporting SQL. The JDBC API is contained in the java.sql Java package. A driver therefore serves as an interface that converts JDBC calls directly or indirectly into a specific database protocol.

This application used the MySQL Connector/J driver, which is a pure Java implementation of the JDBC API, and communicates directly with the MySQL server using the MySQL protocol. To add the driver to the IDE:

- If you have just downloaded the driver, first extract it to a location on your computer. Set the root directory to: C:\mysql-connector-java-5.0.5.
- In the IDE, in the Database explorer from the Runtime window (Ctrl+5) expand the Databases node and right-click the Drivers node. Choose New Driver. The New JDBC Driver dialog opens.
- Click the Add button in the top right corner. Navigate to the driver's root directory and select the driver's jar file (e.g. mysql-connector-java-5.0.5-bin.jar). Click Open. The New JDBC Driver dialog should look like this:

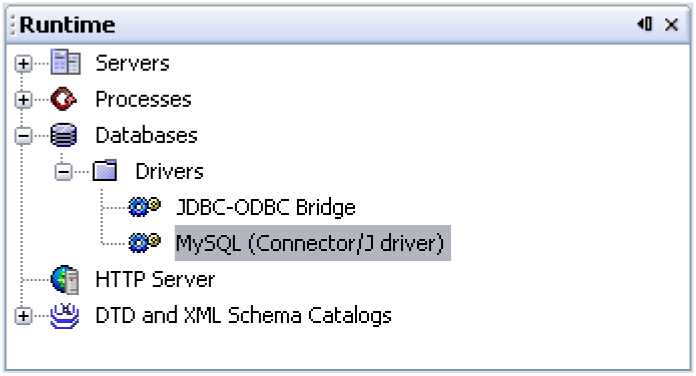

New JDBC Driver

Driver File(s): C:\mysql-connector-java-5.0.5\mysql-connector-java-5.0.5-bin.jar

Driver Class: com.mysql.jdbc.Driver

Name: MySQL (Connector/J driver)

- Click OK. In the Runtime window expand the Databases > Drivers nodes and note that a new MySQL driver node is displayed:



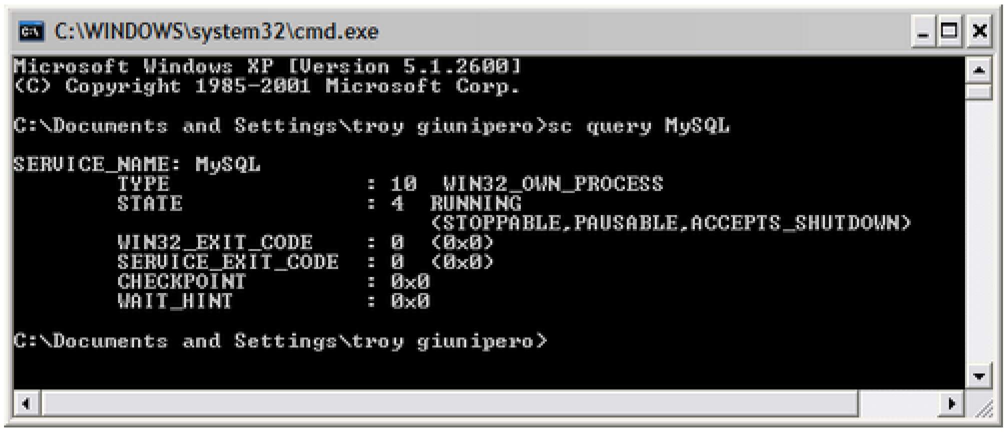
Note: While you just made the database driver available to the IDE, you have not yet made it available to any specific application. At this stage, you can use the IDE to access and modify the database, but *cannot* do so from an application yet.

b) Creating a Database Connection

You can now set up a connection to the database through the appropriate driver. Make sure your database service is running prior to continuing. You can do so by performing the following simple test:

i) Open a Windows command shell. A command line window displays.

ii) At the prompt, type sc query MySQL and press Enter. The Windows sc command allows you to query the state of services. The output should indicate that the current state of the MySQL service is RUNNING:

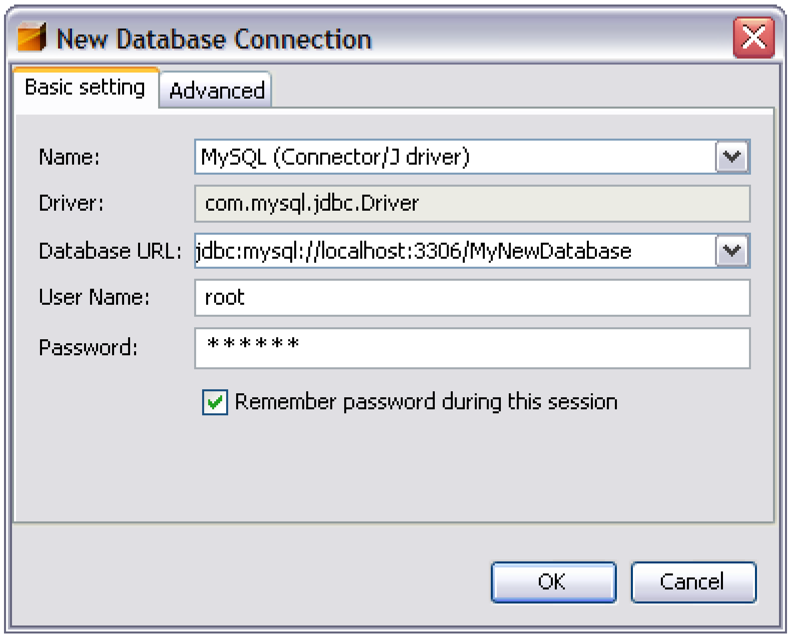


```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\troy giunipero>sc query MySQL

SERVICE_NAME: MySQL
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE,PAUSABLE,ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0

C:\Documents and Settings\troy giunipero>
```

If the service is STOPPED, you can start it by either typing sc start MySQL at the command line prompt, or using the Windows Services GUI (Start > Control Panel > Administrative Tools > Services).

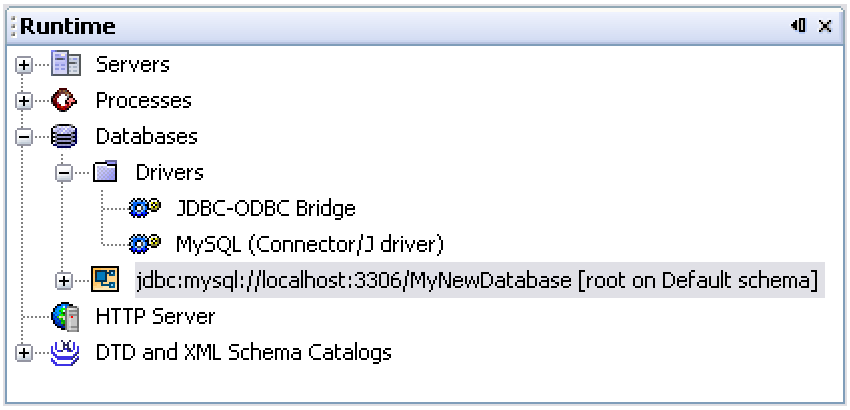
Now, in NetBeans IDE, create a connection to MyNewDatabase:

a. In the Runtime window (Ctrl+5) choose Connect Using from the right-click menu of the driver you just added. The New Database Connection dialog opens.
b. In the Basic Setting tab, enter the Database's URL in the corresponding text field. The URL is used to identify the type and location of a database server. In this example, you need to specify the host name (i.e. the location of the server), the port number on which the database communicates, and the name of the database instance being used. In this case you can enter: jdbc:mysql://localhost:3306/MyNewDatabase.
c. For User Name and Password, enter root, and password respectively:



Name: MySQL (Connector/J driver)
Driver: com.mysql.jdbc.Driver
Database URL: jdbc:mysql://localhost:3306/MyNewDatabase
User Name: root
Password: ******
[✓] Remember password during this session

d. Click OK, then click OK again to exit the dialog. A new Connection node displays in the Runtime window's Database explorer under the Databases node:


Runtime
Servers
Processes
Databases
Drivers
JDBC-ODBC Bridge
MySQL (Connector/J driver)
jdbc:mysql://localhost:3306/MyNewDatabase [root on Default schema]
HTTP Server
DTD and XML Schema Catalogs

You are now connected to the NewDatabase in the IDE. Note that the new connection node icon appears whole () when you are connected to a database. Likewise, it appears broken () when there is no connection.

At later stages, when working with databases through the Database explorer, you may need to manually connect to a database. You can do so by right-clicking the broken database connection node and choosing Connect.

3.3.2 Creating Database Tables

Now that you have connected to the database, you can begin exploring how to create tables, populate them with data, and modify data maintained in tables on the IDE. This allows you to take a closer look at the functionality offered by the Database explorer, as well as NetBeans IDE's support for SQL files. You can prepare the database for use in the web application to be developed.

We have created a table StudentResults in Mysql Database. Alternatively, if the NewDatabase database that you are using is currently empty. In NetBeans IDE it is possible to add a database table by either using the Create Table dialog, or by inputting an SQL query and running it directly from the SQL editor. Here are the two methods:

i. Using the Create Table Dialog
ii. Using the SQL Editor

i. Using the Create Table Dialog

a. In the Database explorer, expand the NewDatabase (i.e Examination) connection node (▣) and note that there are three subfolders: Tables, Views and Procedures. Right-click the Tables node and choose Create Table. The Create Table dialog opens.
b. In the Table Name text field, type the NewTableName(i.e results).
c. In the first row displayed, select the Key check box. You are specifying the primary key for your table. All tables found in relational databases must contain a primary key. Note that when you select the Key check box, the Index and Unique check boxes are also automatically selected and the Null check box is deselected. This is because primary keys are used to identify a unique row in the database, and by default form the table index. Because all rows need to be identified, primary keys cannot contain a Null value.
d. For Column Name, enter Sno. For Data Type, select INT from the drop-down list. Then click the Add Column button.
e. Repeat this procedure by specifying all remaining fields, as shown in the table below:

Key	**Index**	**Null**	**Unique**	**Column name**	**Data type**	**Size**
[checked]	[checked]		[checked]	Sno	INT	0
		[checked]		IDno	VARCHAR	12
		[checked]		CourseCode	VARCHAR	6
		[checked]		Marks	INT	0

f. You are creating a table named results that will hold the following data for each record:

- Sno
- IDno
- CourseCode
- Marks

When you are sure that your Create Table dialog contains the same specifications as those shown above, click OK. The IDE generates the ressult table in the database, and you see a new result table node (▦) display under Tables in the Database explorer. Beneath the table node there are the columns (fields) you created, starting with the primary key (▦):

ii. Using the SQL Editor

a. In the Database explorer, right-click the Tables node beneath MyNewDatabase (i.e Examination) connection node and choose Execute Command. A blank canvas opens in the SQL Editor in the main window.

b. In the SQL Editor, type in the following query. This is a table definition for the StudentResults table you are about to create:

c. CREATE TABLE results (

d. Sno INT UNSIGNED NOT NULL AUTO_INCREMENT,

e. IDno VARCHAR (12) NOT NULL,

f. CourseCode VARCHAR (6) NOT NULL,

g. Marks INT NOT NULL,

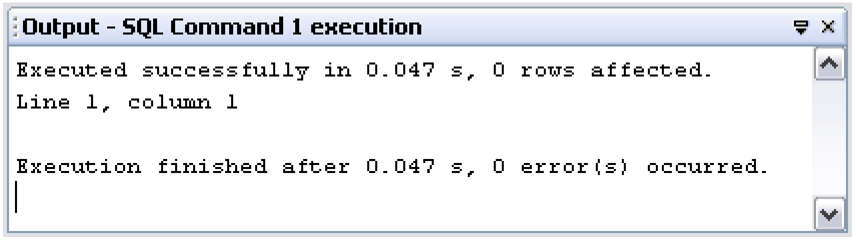
h. PRIMARY KEY (Sno)

i.);

Note: Queries formed in the SQL Editor are parsed in Structured

Query Language (SQL). SQL adheres to strict syntax rules which you should be familiar with when working in the IDE's Editor. Upon running a query, feedback from the SQL engine is generated in the Output window indicating whether execution was successful or not.

j. Click the Run SQL (🔷) button in the task bar at the top (or, press Ctrl+Shift+E to execute the query). You should receive the following message in the Output window:



```
Executed successfully in 0.047 s, 0 rows affected.
Line 1, column 1

Execution finished after 0.047 s, 0 error(s) occurred.
```

k. To verify changes, right-click the Tables node in the Database explorer and choose Refresh. The Refresh option updates the Database explorer's UI component to the current status of the specified database. This step is necessary when running queries from the SQL Editor in NetBeans IDE. Note that the new results table node (▦) now displays under Tables in the Database explorer.

3.3.3 *Working with Table Data*

In order to work with table data, you can make use of the SQL Editor in NetBeans IDE. By running SQL queries on a database, you can add, modify and delete data maintained in database structures. To add a new record (row) to the result table, do the following:

a. Choose Execute Command from the StudentResult table node in the Database explorer. A blank canvas opens in the SQL Editor in the main window.
b. In the SQL Editor, type in the following query:
c.
```
INSERT INTO StudentResult
   VALUES (1, '09/0001', 'CS101', 65, 'B');
```
d. Click the Run SQL (▶) button in the task bar at the top, or press Ctrl+Shift+E to execute the query. You should receive a message in the Output window indicating that the query was executed successfully.
e. To verify that the new record has been added to the StudentResult table, in the Database explorer, right-click the result table node and choose View Data. A new SQL Editor pane opens in the main window. When you choose View Data, a query to select all the data from the table is automatically generated in the upper region of the SQL Editor. The results of the statement are displayed in a table view in the lower region. The sample data in result table displays. Note that a new row has been added with the data you just supplied from the SQL query:
f. Repeat these steps to insert each row in the table.

Sample data in the result table in Examination database are:

Sno	IDno	CourseCode	Marks
1	09_0001	CS101	65
2	08_1021	MA101	52
3	09_1200	MA201	45
4	09_1011	CS202	60
5	09_1001	CS201	75
6	09_1110	CE212	67
7	08_1015	CS211	25
8	09_2020	MA112	40
9	08_1111	CS121	80
10	09_1212	MA222	07

CHAPTER 4:
SYSTEM IMPLEMENTATION

4.0 Introduction

This section describes how to create a distributed web application that connects to a MySQL database. It also covers some basic idea and technologies in web application development, such as Java server pages and JDBC Connection. MySQL is a popular Open Source database management system commonly used in web application due to its speed, flexibility and reliability. MySQL employs SQL, or Structured Query Language, for accessing and processing data contained in databases.

Now that you already have a connection to a MySQL database created and configured in NetBeans IDE. Now proceed to establish a connection to a specific web application in the NetBeans IDE.

Before beginning this section, make sure the following software are installed on your computer:

i) NetBeans IDE 5.5

ii) Java SE Development Kit(JDK) version 5.0 or higher

iii) MySQL database

iv) MySQL ConnectorJ JDBC Driver for MySQL

v) Sun Java System Application Server

4.1 Planning the Structure

Web applications can be designed in such away that the application communicates directly with a data source using the Java Database Connectivity API. A user's requests are sent to a database, and the results are sent directly back to the user. This architecture can be easily mapped to a client-server configuration, where a user's browser serves as the client, and a remote database reachable over the Internet corresponds to the server.

The welcome page (index.jsp) presents the user with a simple HTML form. When a client requests the index.jsp page, the JSP code contained therein is parsed, and data from the IDno from the database table is gathered, added to the page, and sent to the client. The user makes a selection in the provided HTML form and submits, which causes the client to make a request for response.jsp. When response.jsp is parsed, data from result tables is gathered and inserted into the page. Finally, the page is returned to the client and the user views data based upon his or her selection.

4.2 Creating a new project

In order to implement the scenario described above, to develop a web application for student Examination record. The application enables a user to choose a IDno from a drop-down list (index.jsp), then retrieves data from the MySQL database and returns the information to the user (response.jsp).

Create a new project in the IDE by performing the following steps:

1. Launch NetBeans IDE and choose New project(Ctrl+Shift+N) from the file menu. Under Categories select Web; under Project select Web Application. Click Next.

2. In project Name, enter JDBCprogram. From the Server dropdown list, select the server you plan to work with. Leave all other settings at their default and click finish.

   If you downloaded the SJSAS but have not yet registered it in NetBeans IDE, you can do so by clicking the manager button to the right of the server drop-down list.

The server manager opens; enabling you to register the new server. The IDE creates a project template for the entire application, and opens an empty JSP page (index.jsp) in the Source Editor.

4.3 Preparing The Interface

Begin by preparing a simple interface for the two pages. Both index.jsp and response.jsp implement an HTML table to display data in a structured fashion. index.jsp also requires an HTML form that includes a drop-down list.

4.3.1 Implementing the Welcome Page

Convert index.jsp into JDBCprogram's welcome page:

1. Make sure index.jsp is open in the Source Editor. If it is not already open, double click index.jsp from JDBCprogram>Web Pages in the Projects window.Then, in the Source Editor, change the title to JDBCprogram HomePage.

2. For the body, replace what is listed with the following code:

```
<html>
   <head>
      <title align=center>Sample web application Demonstrating JDBC Connection</title>
   </head>
   <br><br>
   <body>
        <h1 align=center>web application Demonstrating JDBC Connection</h1>
      <table  width="55%" align="center">
         <thead>
            <tr>
               <th><strong>This section show student result by IDno </strong></th>
            </tr>
          </thead>
        <tbody align="center">
          <tr>
            <td><strong>To see the result of a student, select his IDno below:</strong></td>
          </tr>
          <tr>
            <td><form action="response1.jsp">
                    <strong>Select an IDno:</strong>
                    <select name="IDno">
                        <option></option>
                    </select>
                    <input type="submit" value="submit" name="submit" />
            </form></td>
          </tr>
        </tbody>
      </table>
      </body>
    </html>
```

This create a form inside a table. Later, when you implement the JSP code, you will replace the sample IDno with a loop that grabs all IDnos directly from the database. Also, note that the forms submits to the response.jsp that about to be created.

4.3.2 Implementing the Response Page

In order to prepare the interface for response.jsp you must first create the file in your application. Note that most of the content that displays in this page is generated dynamically using JSP technology. Therefore, in the following steps you add *placeholders* which you will later substitute for the JSP code.

To create a placeholder for response.jsp, do the following:

1. Right-click the JDBcprogram project node in the Projects window and choose New>JSP.The New JSp File dialog opens.

2. In the JSP File Name field, enter response. Note that Web pages is currently selected for the Location field, meaning that the file will be created in the same directory as the welcome page.

3. Accept all other defaults and click Finish. A template for the new response.jsp page is generated and opens in the Source Editor. A new JSP node also displays under Web Pages in the Projects window.

4. In the Source Editor, change the title temporarily to something like "(Chosen IDno)".

5. Next, replace the template's body with the following code:

```
   <html>
     <head>
       <title align=center>Web application Demonstrating JDBC Connection</title>
   </head>
   <br><br>
<body>
       <h1 align=center>web application Demonstrating JDBC Connection</h1>
    <table  width="55%" align="center">
          <thead>
          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <h2 align=center><strong>Result of a Student Whose IDno was Choosen in Index.jsp
</strong></h2>
      </thead>
       <body>
```

```
        <tr>
            <td valign="top" width="25%"><strong>Student IDno:
            </strong></td>
           <td><em> ( IDno description ) </em><br><br></td>
        </tr>
        <tr>
            <td valign="top"><strong>Course code: </strong></td>
            <td><em> (Course code ) </em><br><br></td>
        </tr>
        <tr>
            <td valign="top"><strong>Marks Scored: </strong></td>
            <td><em> (   Student's Marks ) </em><br><br></td>
        </tr>
      </table>
    </body>
</html>
```

This creates an HTML template for the output that will be generated once you code in the JSP. To view this page in a browser, right-click in the Source Editor and choose Run File (Shift-F6). The page is compiled, deployed to your server, and opened in your default browser. Your response.jsp page should be displayed. Note that all the fields that are enclosed in parentheses will be generated dynamically.

4.4 Deploying to the Server

To deploy the application to the server:

i) From the Projects window, right-click the project node and choose Deploy Project. NetBeans IDE automatically starts the server (if it has not already been

started), compiles it and then deploys the project to it. You can see any output generated in the Output window. The output should complete with a BUILD SUCCESSFUL message.
To check that the application has indeed been deployed to the server, open the Runtime window (Ctrl+5) and expand the Servers node. The servers that are registered in the IDE are listed here. For SJSAS, expand Applications > Web Applications to view the application.

ii) To run the project, back in the Projects window choose Run Project from the right-click menu of the project node. The index.jsp page opens in the IDE's default browser.

Tip: If you had simply chosen Run Project to begin with, the application would have automatically been compiled and deployed to the server prior to opening in a browser.

4.5 Implementing The Data Layer

Before coding in the Logic Layer, prepare the Data Layer. This step is divided into a few subtasks:

i) Preparing the Database in Netbeans IDE

ii) Setting up a JDBC Connection Pool

iii) Referencing the JDBC Resources from the Application

iv) Adding the Database Driver's JAR File to the Server

4.5.1 Preparing the Database in NetBeans IDE

After completing the Connecting to a MySQL Database in NetBeans IDE in section 3.3, you already have a connection to a MySQL database registered in the IDE. You should also have your tables containing sample data in the IDE.

4.5.2 Setting up a JDBC Connection Pool

In order to specify how the web server allows an application to communicate with the database, we need to set up a *database connection pool*. A database connection pool is basically a group of reusable connections that a server maintains for a specific database. Web applications requesting a connection to a database obtain that connection from the pool. When an application closes a connection, the connection is returned to the pool.

In order to set up a connection pool on the server, a *JDBC resource* (also called a data source) must first be created. A JDBC resource provides applications with a connection to a database. To set up a JDBC Connection Pool with SJSAS, do the following:

Setting up a JDBC connection pool with Sun Java System Application
Server can be done entirely within NetBeans IDE:

i) In the Projects window, right-click the project node and choose New > File/Folder.... Under Categories select Sun Resources; under File Types select JDBC Resource. Click Next.

ii) In the General Attributes pane, select the Create New JDBC Connection Pool option, and then enter jdbc/connectionPool for the JNDI Name below. Leave all other settings at their defaults and click Next. Then click Next again through the Additional Properties Pane.

iii) In the Choose Database Connection pane, note that a connection pool name is automatically provided based on the JNDI name specified above. Make sure the Extract from Existing Connection option is selected, and select the database connection you are using (jdbc:mysql://localhost:3306/MyNewDatabase) from the drop-down list. Click Next.

iv) In the Add Connection Pool Properties pane, leave all settings at their defaults and click Finish. The new data source and connection pool are created in the project. You can verify this by expanding the Server Resources node to view both the data source and connection pool just created:

Although you just created a data source and connection pool in the project, it is still necessary to register them with the application server. To do so:

i) Choose Register from the right-click menus of both data source and connection pool nodes. In the JDBC Resource Registration dialog that displays, click Register, then click Close.

ii) To verify that the JDBC resource and connection pool have indeed been created on the server, you can switch to the Runtime window and choose View Admin Console from the right-click menu of the Sun Java System Application Server node. The Administration login page opens in the IDE's default browser.

iii) Login to the console (by default, user name and password are: admin, adminadmin). Select Resources from the left column, then in the main window click JDBC.

iv) Now, when you explore both the JDBC Resources and Connection Pools pages, you should see the newly created data source (jdbc/connectionPool) and connection pool (connectionPool).

4.5.3 Referencing the JDBC Resource from the Application

You now need to reference the JDBC resource you just created from the web application. This means you have to access both the web application's general deployment descriptor (web.xml), as well as the server-specific deployment descriptor for the server you are using (sun-web.xml for SJSAS).

Deployment descriptors are XML documents that contain information describing how an application should be deployed. For example, they are normally used to specify location and optional parameters of servlets and JSP files, as well as implement basic security features for your application.

To reference the JDBC resource in the general deployment descriptor:

i) In the Projects window, expand the Web Pages > WEB-INF subfolder and double-click web.xml. A graphical editor for the file displays in the Source Editor.

ii) Click the References tab located along the top of the Source Editor. Expand the Resource References heading, and then click Add.... The Add Resource Reference dialog opens.

iii) For Resource Name, enter the JNDI name you gave when adding the data source to the server above (jdbc/connectionPool). For Description, enter the data source URL (jdbc:mysql://localhost:3306/MyNewDatabase). Leave all other fields that are filled by default and click OK. The new resource is added under the Resource References heading.

    To verify that the resource is now added to the web.xml file, click the XML tab located along the top of the Source Editor you'll see that the following <resource-ref> tags are now included:

```
<resource-ref>
    <description>jdbc:mysql://localhost:3306/MyNewDatabase [root on Default schema]</description>
    <res-ref-name>jdbc/connectionPool</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

Using SJSAS, perform the following steps to reference the JDBC resource in the server-specific deployment descriptor:

i) In the Projects window, expand the Web Pages > WEB-INF subfolder and double-click sun-web.xml. A graphical editor for the file displays in the Source Editor.

ii) Click Edit As XML in the upper right corner of the editor. The file displays in XML format. Enter the following `<resource-ref>` tags to the document, e.g. following the closing `</jsp-config>` tag:

```
<resource-ref>
    <res-ref-name>jdbc/ConnectionPool</res-ref-name>
    <jndi-name>jdbc/ConnectionPool</jndi-name>
</resource-ref>
```

4.5.4 **Adding the Database Driver's JAR File to the Server**

Adding the database driver's JAR file is another step that is vital to enabling the server to communicate with your database. You need to locate your database driver's installation directory. We are using MySQL Connedtor/J which was installed to C:\ on the computer. Copy the mysql-connector-java-5.0.5-bin.jar file in the driver's root directory, using SJSAS, do the following:

Locate the installation directory of SJSAS and paste the JAR file into the server's domains > domain1 > lib > ext subfolder. For example, if you installed the server to C:\, the path is: C:\Sun\AppServer\domains\domain1\lib\ext. When you connect to the SJSAS in NetBeans IDE, you are actually connecting to an instance of the application server. Each instance runs applications in a unique domain, and so here we need to place the JAR file in domain1, which is the default domain created upon installing SJSAS. If you've already started the server, make sure that you restart it after pasting in the JAR file, so that the server can then load it.

4.6 **Adding Dynamic Logic**

If you return to the index.jsp and response.jsp placeholders created earlier in the tutorial, you can add JSP code to enable pages to generate content *dynamically*, i.e., based on user input. To do so, add the code to each page. Both pages require that you implement an SQL query and the datasource created earlier.

index.jsp

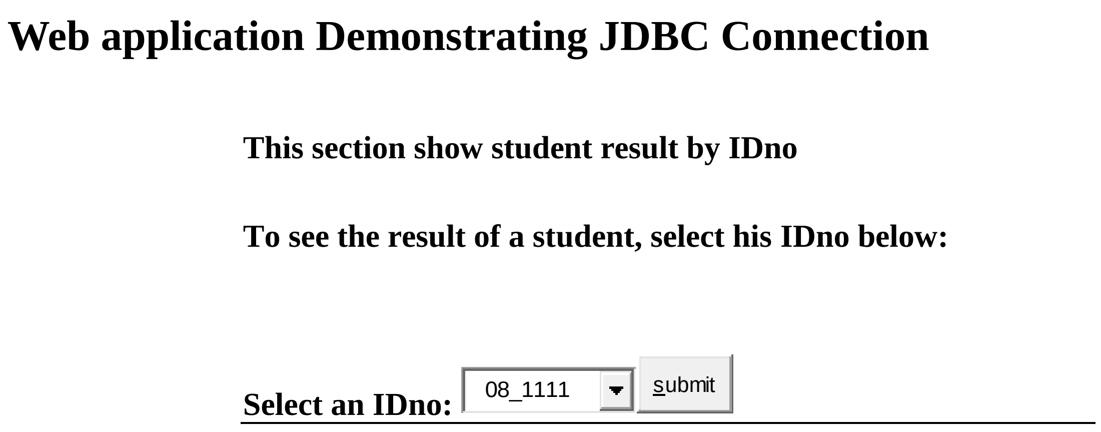
In order to dynamically display the contents of the form in index.jsp, you need to access all IDnos from the result database table. The following code performs the task.

```
<html>
   <head>
      <title align=center>Sample web application Demonstrating JDBC Connection</title>
   </head>
   <br><br>

   <body>
      <%@ page import="javax.naming.*" %>
      <%@ page import="java.sql.*" %>
<%@ page import="javax.sql.*" %>

<h1 align=center>web application Demonstrating JDBC Connection</h1>

<%

Connection conn = null;
Statement stmt = null;
ResultSet rs = null;

try {

   // Obtain our environment naming context
   Context initCtx = new InitialContext();
   Context envCtx = (Context) initCtx.lookup("java:comp/env");

   // Look up our data source
   DataSource ds = (DataSource) envCtx.lookup("jdbc/Examination");

   // Allocate and use a connection from the pool
   conn = ds.getConnection();

   // Fetch and display data
   stmt = conn.createStatement();

   // retrive IDnos from the result database table
   rs = stmt.executeQuery("SELECT IDno FROM result");

%>

<table  width="55%" align="center">
   <thead>
      <tr>
         <th><strong>This section show student result by IDno </strong></th>
      </tr>

   </thead>

   <tbody align="center">
      <tr>
         <td><strong>To see the result of a student, select his IDno below:</strong></td>
      </tr>

      <tr>
         <td><form action="response1.jsp">
               <strong>Select an IDno:</strong>
               <select name="IDno">
```

```
                    <%while (rs.next()) { String s = rs.getString("IDno");%>
                    <option><% out.print(s+"<br>" );}%></option>
                  </select>
                  <input type="submit" value="submit" name="submit" />
              </form></td>
            </tr>
          </tbody>
        </table>

        <%

        rs.close();
        rs = null;
        stmt.close();
        stmt = null;
        conn.close(); // Return to connection pool
        conn = null;  // Make sure we do not close it twice
        } catch (SQLException e) {
        out.print("Throw e" + e);
        } finally {
        // Always make sure result sets and statements are closed,
        // and the connection is returned to the pool
        if (rs != null) {
        try { rs.close(); } catch (SQLException e) { ; }
        rs = null;
        }
        if (stmt != null) {
        try { stmt.close(); } catch (SQLException e) { ; }
        stmt = null;
        }
        if (conn != null) {
        try { conn.close(); } catch (SQLException e) { ; }
        conn = null;
        }
        }

        %>

    </body>
</html>
```

NB: the changes in bold

Save changes (Ctrl+S), then right-click in the Source Editor and choose Run File (Shift-F6). The file is compiled and deployed to the server, and index.jsp renders in the browser. The drop-down list now contains subject names that were retrieved from the database:



Web application Demonstrating JDBC Connection

This section show student result by IDno

To see the result of a student, select his IDno below:

Select an IDno:

response.jsp

For `response.jsp`, you need to access data from result database tables that correspond to the IDno submitted by the user. This is accomplished using an SQL query and the datasource created. A jdbc database connection established and an SQL query applied to retrive data corresponding to the IDno chosen. The following code performs the task:

In the HTML, replace all *placeholders* with JSP code that allows you to retrieve and display the data held in the result database table (Changes below shown in **bold**):

```
<html>
 <head>
       <title align=center>Web application Demonstrating JDBC Connection</title>
```

```
    </head>
    <br><br>
  <body>
    <%@ page language="java" %>
    <%@ page import="javax.naming.*" %>
    <%@ page import="java.sql.*" %>
    <%@ page import="javax.sql.*" %>
    <%@ page import="javax.servlet.*"%>
      <h1 align=center>web application Demonstrating JDBC Connection</h1>
      <%
    Connection conn = null;
    PreparedStatement psmt=null;
    ResultSet rst= null;
      try {
      Context initCtx = new InitialContext();
      Context envCtx = (Context) initCtx.lookup("java:comp/env");
     // Look up our data source
      DataSource ds = (DataSource) envCtx.lookup("jdbc/Examination");
      // Allocate and use a connection from the pool
      conn = ds.getConnection();
      try {
        String IDno = request.getParameter("IDno");
        if (IDno != null) {
          if (!IDno.equals("")) {
         // set SQL to Fetch  data
        psmt = conn.prepareStatement("SELECT IDno, CourseCode, Marks FROM result
WHERE IDno = '?' ");

        psmt.setString(1, IDno);

         } }

        // display the data fetched

        rst = psmt.executeQuery();

        while (rst.next()){

        %>

    <table  width="55%" align="center">

        <thead>

          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

           <h2 align=center><strong>Result of a Student Whose IDno was Choosen in Index.jsp
</strong></h2>

       </thead>

      <body>

        <tr>

           <td valign="top" width="25%"><strong>Student IDno:

           </strong></td>

           <%while (rs.next()) { %>

           <td><em><%out.println(rs.getString(1)); %></em><br><br></td>

        </tr>

        <tr>

           <td valign="top"><strong>Course code: </strong></td>

           <td><em><% out.println(rs.getString(2)); %></em><br><br></td>

        </tr>

        <tr>

           <td valign="top"><strong>Marks Scored: </strong></td>
          <td><em><%out.println(rs.getInt(3)); } %></em><br><br></td>
        </tr>
    </table>
   <%
    }catch (SQLException s){
        System.out.println("SQL statement is not executed!");
   }catch (Exception e) {
    System.out.print("Throw e" + e);
  }
    rst.close();
    rst = null;
    psmt.close();
    psmt = null;
    conn.close(); // Return to connection pool
    conn = null;  // Make sure we do not close it twice
}  finally {
    // Always make sure result sets and statements are closed,
    // and the connection is returned to the pool
    if (rst != null) {
        try { rst.close(); } catch (SQLException e) { ; }
        rst = null;
    }
    if (psmt != null) {
        try { psmt.close(); } catch (SQLException e) { ; }
        psmt = null;
    }
```

```
        if (conn != null) {
            try { conn.close(); } catch (SQLException e) { ; }
            conn = null;
        }
    }
     %>
    </body>
</html>
```

Prior to sending this page back to the client browser, the server inserts all values for JSP variables directly into the HTML. You are therefore now able to access data directly from the Data Layer in the response page.

Finally, save changes (Ctrl + S), then redeploy the project to the server and try running it again (F6). When index.jsp displays in the browser, select an IDno from the drop-down list and click submit. You should now be forwarded to the response.jsp page, showing details corresponding to your selection:

When a user selected the first IDno, 09_0001 then the response page displays this record:

Web application Demonstrating JDBC Connection

Result of a Student Whose IDno was Chosen in Index.jsp

Student IDno:	*09_0001*
Course code:	*CS101*
Marks Scored:	*65*

When a user selected the sixth IDno, 09_1110 then the response page displays this record:

Web application Demonstrating JDBC Connection

Result of a Student Whose IDno was Chosen in Index.jsp

Student IDno:	*09_1110*
Course code:	*CE212*
Marks Scored:	*67*

When a user selected the nineth IDno, 08_1111 then the response page displays this record:

Web application Demonstrating JDBC Connection

Result of a Student Whose IDno was Chosen in Index.jsp

Student IDno:	*08_1111*
Course code:	*CS121*
Marks Scored:	*80*

CHAPTER 5:
SUMMARY AND CONCLUSION

5.0 Introduction

This chapter briefly discusses the key points of the project and concludes by justifying the use of MySQL and JDBC.

5.1 Summary

This project illustrates how a program can establish a connection to a server program using the Socket class and then, how the client can send data to and receive data from the server through the socket. The program implements a client, EchoClient that connects to the Echo server. The Echo server simply receives data from its client and echoes it back. The Echo server is a well-known service that clients can rendezvous with on port 7.

EchoClient creates a socket thereby getting a connection to the Echo server. It reads input from the user on the standard input stream, and then forwards that text to the Echo server by writing the text to the socket. The server echoes the input back through the socket to the client. The client program reads and displays the data passed back to it from the server:

On implementation, The JDBC interface allows developers to write applications that can be used with different databases with a minimum of porting effort. Once a driver for a given server engine is installed, JDBC applications can communicate with any server of that type. By using MySQL Connector/J, your Java programs can access MySQL databases. MySQL Connector/J works within the framework of the Java JDBC interface, an API that allows Java programs to use database servers in a portable way.

Connection pooling provides a significant improvement on performance by reusing connections rather than creating a new connection for each connection request, without requiring changes in your JDBC application code.

5.2 Conclusion

In conclusion, This document demonstrated how to create a web application that connects to a MySQL database. It also demonstrated how to construct an application using basic client-server architecture, and utilized JDBC and JSP technologies as a means of accessing and displaying data dynamically.

JDBC technology is an API *(included in both J2SE and J2EE)* that provides cross-DBMS connectivity to a wide range of SQL databases and access to other tabular data sources, such as spreadsheets or flat files. The MySQL database server is probably the world's most popular open source database software, with more than five million active installations as of September 2004.

The database server software from MySQL is available under a *"dual licensing"* model. Under this model, users may choose to use MySQL products under the free software/open source GNU General public License *(commonly known as the "GPL")* or under a commercial license..

JDBC makes it possible to write platform independent Java programs that can be used to manipulate the data in a wide range of SQL databases without the requirement to modify and/or recompile the Java programs when moving from platform to platform or from DBMS to DBMS.

MySQL is available for a wide variety of platforms. Since both JDBC and MySQL are freely available for many purposes, the combination of JDBC and MySQL is a powerful combination that should be of interest for a wide variety of applications.

BIBLIOGRAPHY

1. Behrouz A. Forouzan(2006), Data Communication and Networking. Fourth Edition. Deanza College with Sophia Chung Fegan. Tata Mcgraw hill publishers.

2. Berg D.J. and Fritzinger, J.S (1997), Advanced Techniques for Java Developers. John Wiley & sons, Inc

3. Cohoon J. and Davidson J.(2004): Java 1.5 Program Design. McGraw Hill Higher Education Publishing Company.

4. Hubbard, J.R(1999), Shaum's outline of theory an problems of programming with Java. The Mcgraw-Hill companies, Inc.

5. Gillenson, M. L.(1984), DATABASE Step-by-Step. John Willey and sons. Second Edition.

6. MySQL 5.0 Referenc Manual, *downloads.**mysql**.com/docs/**ref**man-**5.0**-fr.pdf*

7. Deitel P.J. and Deitel H.M.(2007), Java How To Program, Seventh Edition. Pearson Prentice Hall, Pearson International Edition(1232-1235).

8. Monson-Haefel, R.(2001), Enterprise JavaBeans, O'Reilly. Third Edition.

9. Sun Java System Application Server 9.1 Reference Manual. *docs.sun.com/doc/819-3675*

10. Sun Mcrosystems(2004), Network programming tutorials. http://java.sun.com/docs/books/tutorial/networking/sockets/definition.html

11. Wu, T.(2004): An Introduction to object oriented programming with Java. Update 1.5. MCGraw Hill Publishers.

12. Javaworld(2002), principles of object-oriented programming http:www.javaworld.com/javaqa/2002-08:01-qa-0823.html

13. Dumitrascu I.(2008), Building your first dynamic websites. Adobe system incorporated.

APPENDIX

Program Codes

Index.jsp

```
<html>
   <head>
      <title align=center>Sample web application Demonstrating JDBC Connection</title>
   </head>
   <br><br>
   <body>
        <h1 align=center>web application Demonstrating JDBC Connection</h1>
      <table  width="55%" align="center">
         <thead>
            <tr>
               <th><strong>This section show student result by IDno </strong></th>
            </tr>
          </thead>

         <tbody align="center">
            <tr>
               <td><strong>To see the result of a student, select his IDno below:</strong></td>
            </tr>
            <tr>
               <td><form action="response1.jsp">
                     <strong>Select an IDno:</strong>
                     <select name="IDno">
                         <option></option>
                     </select>
                     <input type="submit" value="submit" name="submit" />
               </form></td>
            </tr>
         </tbody>
      </table>
      </body>
    </html>
```

Rsponse.jsp

```
<html>
 <head>
        <title align=center>Web application Demonstrating JDBC Connection</title>
    </head>
    <br><br>
<body>
    <%@ page language="java" %>
    <%@ page import="javax.naming.*" %>
    <%@ page import="java.sql.*" %>
    <%@ page import="javax.sql.*" %>
    <%@ page import="javax.servlet.*"%>
        <h1 align=center>web application Demonstrating JDBC Connection</h1>
        <%
    Connection conn = null;
    PreparedStatement psmt=null;
    ResultSet rst= null;
        try {
        Context initCtx = new InitialContext();
        Context envCtx = (Context) initCtx.lookup("java:comp/env");
       // Look up our data source
        DataSource ds = (DataSource) envCtx.lookup("jdbc/Examination");
        // Allocate and use a connection from the pool
        conn = ds.getConnection();
        try {

          String IDno = request.getParameter("IDno");

          if (IDno != null) {

             if (!IDno.equals("")) {

            // set SQL to Fetch  data

          psmt = conn.prepareStatement("SELECT IDno, CourseCode, Marks FROM result
WHERE IDno = '?' ");

          psmt.setString(1, IDno);

           } }

          // display the data fetched

          rst = psmt.executeQuery();

          while (rst.next()){

         %>

    <table  width="55%" align="center">

          <thead>

            <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

             <h2 align=center><strong>Result of a Student Whose IDno was Choosen in Index.jsp
</strong></h2>

        </thead>

       <body>

          <tr>

             <td valign="top" width="25%"><strong>Student IDno:

             </strong></td>

             <%while (rs.next()) { %>

             <td><em><%out.println(rs.getString(1)); %></em><br><br></td>

          </tr>

          <tr>
          <td valign="top"><strong>Course code: </strong></td>
          <td><em><% out.println(rs.getString(2)); %></em><br><br></td>
        </tr>
        <tr>
          <td valign="top"><strong>Marks Scored: </strong></td>
          <td><em><%out.println(rs.getInt(3)); } %></em><br><br></td>
        </tr>
    </table>
   <%
    }catch (SQLException s){
        System.out.println("SQL statement is not executed!");
   }catch (Exception e) {
    System.out.print("Throw e" + e);
  }
    rst.close();
    rst = null;
    psmt.close();
    psmt = null;
    conn.close(); // Return to connection pool
    conn = null;  // Make sure we do not close it twice
}  finally {
    // Always make sure result sets and statements are closed,
    // and the connection is returned to the pool
    if (rst != null) {
        try { rst.close(); } catch (SQLException e) { ; }
        rst = null;
        }
        if (psmt != null) {
            try { psmt.close(); } catch (SQLException e) { ; }
            psmt = null;
        }
        if (conn != null) {
            try { conn.close(); } catch (SQLException e) { ; }
            conn = null;
        }
    }
     %>
    </body>
</html>
```

Echoclient.java

// This program reads input from the user on the standard input stream and then forwards that text to // the EchoServer by writing the text to a Socket. The Server echoes the input back through the //Socket to the client. The client program reads and displays the data passed back to it from the //server.

```
import java.io.*;
import java.net.*;
class EchoClient {
public static void main(String[] args) throws IOException {

      Socket echoSocket = null;
      PrintWriter out = null;
      BufferedReader in = null;
```

// These lines establish the socket connection between the client and server, and open a PrintWriter //and BuferredReader on the socket.

```
    try {
         echoSocket = new Socket("Machine name", 7);
  out = new PrintWriter(echoSocket.getOutputStream(), true);
    in = new BufferedReader(new InputStreamReader(
            echoSocket.getInputStream()));

     } catch (UnknownHostException e) {
         System.err.println("Don't know about host: Machine name.");
         System.exit(1);

     } catch (IOException e) {
     System.err.println("Couldn't get I/O for " + "the connection to:
       Machine name.");
       System.exit(1);
      }
     BufferedReader stdIn = new BufferedReader(
                new InputStreamReader(System.in));

// This loop reads a line at a time from the standard input stream and immediately sends it to the
//server by writing it to the PrintWriter connected to the socket.

         String userInput;
         while ((userInput = stdIn.readLine()) != null) {
             out.println(userInput);
             System.out.println("echo: " + in.readLine());
         }

// These statements close the readers and writers connected to the Socket and to the standard input
//stream and close the socket connected to the Server.

         out.close();
         in.close();
         stdIn.close();
         echoSocket.close();
   }
}


// Note that EchoClient both writes to and reads from its socket, thereby sending data to and receiving
//data from the Echoserver.
```

Echoserver.java

```
import java.io.*;
import java.net.*;
```

```
public class EchoServer{

public static void main(String[ ] args) throws IOException{

ServerSocket ss = null;
Socket cs = null;
PrintWriter out = null;
BufferedReader in = null;

try{
ss = new ServerSocket(3333);
 }
catch (IOException e) {
System.out.println (“Could not listen on port: 3333”);
System.exit(1);
}

try{
cs = ss.accept( );
}
catch (IOException e) {
System.out.println(“Accept failed: 3333”);
System.exit(1);
 }

out = new PrintWriter (cs.getOutputStream ( ),true);
in = new BufferedReader (new InputStreamReader
(cs.getInputStream( )));

String clientInput;

while (true) {
clientInput = in.readLine ( );
if (clientInput == null)
break;
out.println (clientInput);
 }
out.close( );
in.close ( );
ss.close ( );
cs.close ( );
 }
 }
```