CONTENTS

LAB MANUAL

INTERNET PROGRAMMING LAB

S.NO	NAME OF THE EXPERIMENT	PAGE NO
	INTERNET & JAVA PROGRAMMING	
	HTML	
1	Application form	
2	Table Creation	
3	Framesets	
	JAVA SCRIPT	
4	Factorial of a Given Number	
5	Setting Background Color Depending on the System Time	
6	Displaying Multiples of 7	
7	Finding a number greater than 1000 & Divisible by 99	
8	User Define Functions	
9	Built-in Java Script Objects	
	EVENT HANDLING	
10	Web Page Creation using Event Handlers	
11	Image Roll Over Effects	
	SOCKET PROGRAMMING	
12	Datagrams	
13	Creation of TCP Client & Server	
	ACTIVE SERVER PAGES	
14	Displaying Calendar	
15	Registration Form	
	DATA BASE CONNECTIVITY	
16	Student Mark List	
17	Library Management	
	SERVLET	
18	Student Information System	
	JAVA SERVER PAGES	
19	Displaying the Current Date	
20	Tour Plan of a VIP	

SYLLABUS

(2 Experiments under each of the following)

1. Client side / Server side scripting programs for the Web Pages.
2. Experiments with Active / JAVA server pages.
3. Socket Programming.
4. JAVA Servlets
5. On-line Transactions – Database connectivity

HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENTS

Processor	- i3 Processor and above
RAM	- 2GB
Hard disk	- 20GB

SOFTWARE REQUIREMENTS

HTML

THE WEB IS BECOMING AN INTEGRAL PART of our working (and playing) world. You cannot spit anymore these days without hitting a URL. In a very short time span, the web has revolutionized the way we access information, education, business, entertainment. It has created industries where there were none before.

Being able to develop information on the web might be a job skill, a class requirement, a business necessity, or a personal interest. Unlike any other previous medium, the ability to "write" HTML allows you to potentially connect with millions of other people, as your own self-publisher.

PUT MOST SIMPLY, HTML, is a format that tells a computer how to display a web page. The documents themselves are plain text files (ASCII) with special "tags" or codes that a web browser knows how to interpret and display on your screen.

This tutorial teaches you how to create web pages the old-fashioned way -- by hand. There are software "tools" that allow you to spin web pages without touching any HTML. But if you are serious about doing more than a page or two, we believe a grounding in the basics will greatly accelerate what you can do. Everything you create in this tutorial is designed to run from any desktop computer; it does not depend on access to a web server or specialized computer programming.

YOU WILL ALSO NEED A TEXT EDITOR PROGRAM capable of creating plain text files.

Start with a title

Every HTML document needs a title. Here is what you need to type:

<title>My first HTML document</title>

Change the text from "My first HTML document" to suit your own needs. The title text is preceded by the start tag <title> and ends with the matching end tag </title>. The title should be placed at the beginning of your document.

To try this out, type the above into a text editor and save the file as "test.html", then view the file in a web browser. If the file extension is ".html" or ".htm" then the browser will recognize it as HTML. Most browsers show the title in the window caption bar.

Add headings and paragraphs

If you have used Microsoft Word, you will be familiar with the built in styles for headings of differing importance. In HTML there are six levels of headings. H1 is the most important, H2 is slightly less important, and so on down to H6, the least important.

Here is how to add an important heading:

<h1>An important heading</h1>

and here is a slightly less important heading:

<h2>A slightly less important heading</h2>

Each paragraph you write should start with a tag. The is optional, unlike the end tags for elements like headings. For example:

** This is the first paragraph. **

** This is the second paragraph. **

Adding a bit of emphasis

You can emphasize one or more words with the tag, for instance:

This is a really interesting topic!

Adding interest to your pages with images

Images can be used to make your Web pages distinctive and greatly help to get your message across. The simple way to add an image is using the <img> tag. Let's assume you have an image file called "peter.jpg" in the same folder/directory as your HTML file. It is 200 pixels wide by 150 pixels high.

<img src="peter.jpg" width="200" height="150">

The src attribute names the image file. The width and height aren't strictly necessary but help to speed the display of your Web page. Something is still missing! People who can't see the image need a description they can read in its absence. You can add a short description as follows:

<img src="peter.jpg" width="200" height="150" alt="My friend Peter">

The alt attribute is used to give the short description, in this case "My friend Peter". For complex images, you may need to also give a longer description. Assuming this has been written in the file "peter.html", you can add one as follows using the longdesc attribute:

<img src="peter.jpg" width="200" height="150" alt="My friend Peter" longdesc="peter.html">

You can create images in a number of ways, for instance with a digital camera, by scanning an image in, or creating one with a painting or drawing program. Most browsers understand GIF and JPEG image formats, newer browsers also understand the PNG image format. To avoid long delays while the image is downloaded over the network, you should avoid using large image files.

Generally speaking, JPEG is best for photographs and other smoothly varying images, while GIF and PNG are good for graphics art involving flat areas of color, lines and text. All three formats support options for progressive rendering where a crude version of the image is sent first and progressively refined.

Adding links to other pages

What makes the Web so effective is the ability to define links from one page to another, and to follow links at the click of a button. A single click can take you right across the world!

Links are defined with the <a> tag. Lets define a link to the page defined in the file "peter.html":

This a link to Peter's page.

The text between the <a> and the is used as the caption for the link. It is common for the caption to be in blue underlined text.

To link to a page on another Web site you need to give the full Web address (commonly called a URL), for instance to link to www.w3.org you need to write:

This is a link to W3C.

You can turn an image into a hypertext link, for example, the following allows you to click on the company logo to get to the home page:

<img src="logo.gif" alt="home page">

Three kinds of lists

HTML supports three kinds of lists. The first kind is a bulletted list, often called an ***unordered list***. It uses the <ul> and <li> tags, for instance:

```
<ul>
  <li>the first list item</li>

  <li>the second list item</li>

  <li>the third list item</li>
</ul>
```

Note that you always need to end the list with the end tag, but that the is optional and can be left off. The second kind of list is a numbered list, often called an ***ordered list***. It uses the <ol> and <li> tags. For instance:

```
<ol>
  <li>the first list item</li>

  <li>the second list item</li>

  <li>the third list item</li>
</ol>
```

Like bulletted lists, you always need to end the list with the end tag, but the end tag is optional and can be left off.

The third and final kind of list is the definition list. This allows you to list terms and their definitions. This kind of list starts with a <dl> tag and ends with </dl> Each term starts with a <dt> tag and each definition starts with a <dd>. For instance:

```
<dl>
  <dt>the first term</dt>
  <dd>its definition</dd>

  <dt>the second term</dt>
  <dd>its definition</dd>

  <dt>the third term</dt>
  <dd>its definition</dd>
</dl>
```

The end tags </dt> and </dd> are optional and can be left off. Note that lists can be nested, one within another. For instance:

```
<ol>
  <li>the first list item</li>

  <li>
    the second list item
    <ul>
      <li>first nested item</li>
      <li>second nested item</li>
    </ul>
  </li>

  <li>the third list item</li>
</ol>
```

You can also make use of paragraphs and headings etc. for longer list items.

HTML has a head and a body

If you use your web browser's view source feature (see the View or File menus) you can see the structure of HTML pages. The document generally starts with a declaration of which version of HTML has been used, and is then followed by an <html> tag followed by <head> and at the very end by </html>. The <html> ... </html> acts like a container for the document. The <head> ... </head> contains the title, and information on style sheets and scripts, while the <body> ... </body> contains the markup with the visible content. Here is a template you can copy and paste into your text editor for creating your own pages:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title> replace with your document's title </title>
</head>
<body>

replace with your document's content

</body>
</html>
```

1. WEB PAGE CREATION

```
<HTML>
<HEAD>
        <TITLE> SOFTWARE ENGINEERING </TITLE>
</HEAD>
<BODY BGCOLOR = "GRAY">
<MARQUEE DIRECTION = "RIGHT">
<FONT SIZE=8 COLOR="GREEN" FACE="ARIEL">
WELCOME TO WEB DESIGNING LAB
</FONT>
</MARQUEE>
<IMG SRC="C:\IMG1.JPG" ALIGN="RIGHT">
<FONT SIZE=5 FACE="CASTELLAR" COLOR="PINK">
<UL><U>AVAILABLE COURSES</U><BR><BR>
<LI><A HREF="CO1.HTML">JAVA</A>
<LI><A HREF="CO2.HTML">COBOL</A>
<LI><A HREF="CO3.HTML">PROGRAMMING C</A>
<LI><A HREF="CO4.HTML">C++</A>
<LI><A HREF="CO5.HTML">DBMS</A>
</UL>
<BR> <BR> <BR> <BR> <BR>
<TABLE BORDER=10 WIDTH=5%>
</TABLE>
<HR>
<PRE>
        THE ABOVE COURSES ARE OFFERED HERE. INTERESTING
STUDENTS MAY APPLY.
                                                SUN SYSTEMS,
                                                17, GANDHI STREET,
                                                ERODE-02.
</PRE>
</FONT>
</BODY>
</HTML>
```

2. TABLE CREATION

```
<HTML>
<HEAD>
        <TITLE> TEMPERATURE OF MAJOR CITIES </TITLE>
</HEAD>
<BODY BGCOLOR="VIOLET">
<TABLE BORDER=5 BGCOLOR="INDIGO" CELLPADDING=5
CELLSPACING=5>
<CAPTION> TEMPERATURE </CAPTION>
<TR>
<TH>
<TH>MAX
<TH>MIN
</TH>
</TR>
<TR>
<TD>DELHI
<TD>35
<TD>15
</TR>
<TR>
<TD>BOMBAY
<TD>33
<TD>12
</TR>
<TR>
<TD>CALCUTTA
<TD>32
<TD>14
</TR>
</BODY>
</HTML>
```

3. FRAME SET

```
<HTML>
<HEAD> FRAME
</HEAD>

<BODY>
<FRAMESET COLOS="50%,*">

<FRAMESET ROWS="50%,*">
<FRAME SRC="C:HOME.JPEG">
<FRAME SRC="LINK.HTML">
</FRAMESET>

<FRAMESET ROWS="50%,*">
<FRAME SRC="C:PEACE.JPEG">
<FRAMESET NAME="MARGDISP">
</FRAMESET>

</FRAMESET>
</BODY>
</HTML>
```

JAVA SCRIPT

Introduction

- Prerequites: HTML skills, some programming experience, and good PC skills. I will not be able to spend time teaching the basics.
- Software requirements: an editor (notepad, Frontpage, etc.) and a WWW browser (IE 4.0 or NS 4.0). If you log in as 'internet' you should have access to these things.

JavaScript is a language

- Invented by Netscape in 1995 in Navigator 2.0.
- Designed as a browser, client-side scripting language for the Web, supplementing HTML.
- Similar syntax to C, C++, and Java.
- Interpreted, scripting language, not a full programming language.
- Platform-independent.
- Also has been adopted by Microsoft: they call it "JScript".
- Has been proposed as a standard called ECMAScript.
- Not really related to Java.
- Both client-side (browser-based) JavaScript and server-side JavaScript. Most focus is on the client.

JavaScript is a way to enhance WWW pages within a WWW browser

- Netscape and Microsoft both enabled browser elements to be controlled by JavaScript.
- Pop-ups windows, form verification, dynamic pages. Often those effects are created by JavaScript.
- Netscape and Microsoft browsers support JavaScript 1.1. The 1.2 release came out with Netscape 4.0.
- But it is supported very unevenly between operating systems and between Netscape and Microsoft.
- Alternatives: CGI, Java, VBScript, Dynamic HTML, ASP.

JavaScript can also be used elsewhere

- As a server-side scripting language for Microsoft Active Server Pages.
- As a server-side scripting language in Netscape Livewire.
- As a scripting language embedded in other applications.

JavaScript contains many of the same elements that other programming languages do:

- variables, constants (pieces of data)

- data structures like array and objects (ways to hold together and access bigger sets of data)
- user-defined functions
- built-in functions (math, dates, etc.)
- operators (+,-,/,*)
- statements (if, while, for)
- object-based programming (but not fully object-oriented)

But, it is most powerful when the language elements are combined with the "browser object model":

- HTML
- Windows
- Frames
- Hyperlinks
- Forms
- Page elements

Your first program

One way to put JavaScript in a page is with the <SCRIPT> tag

- Start tag with <SCRIPT LANGUAGE="javascript">
- End tag with </SCRIPT>
- Put JavaScript between the tags.
- There are several places on an HTML page where script tags can go, and a page will often have several.

A simple example:

```
<SCRIPT LANGUAGE="javascript">
document.write("Hello, world")
</SCRIPT>
```

Output:

 Hello, world

4. FACTORIAL OF A GIVEN NUMBER

```
<Html>
<Head>
<script language="JavaScript">
<!--
var fac=1,i,n;
n=prompt("Enter the value");
for(i=1;i<=n;i++)
{
        Fac=fac*i;
}
Document.write("Factorial of given number is : "+ fac);
//--
</script>
</Head>
</Html>
```

5. SETTING BACKGROUND COLOR

```
<html>
<head>
<Script language="JavaScript">
<!--
var today=new Date();
minutes = today.getMinutes();
if(minutes>=0 && minutes<=30)
        document.write("<Body text=White bgcolor=blue>");
else
        document.write("<Body text=White bgcolor=black>");
//--
</Script>
</head>
</html>
```

6. MULTIPLES OF 7

```
<hTML>
<Head>
<Title> Program for finding multiples of 7 </Title>
<Script Language="JavaScript">
<!--
var loop=0;
while(++loop<200)
{
        if(loop % 7 != 0)
        continue;
        else
        document.write(loop+" ");
}
//--
</Script>
</Head>
</hTML>
```

7. FINDING BIGGEST NUMBER

```
<html>
<head> The greatest number 1000 that is divisible by 99
<title> divisor </title>
<Script Language="JavaScript">
<!--
var loop;
for(loop=1000;loop<1099;loop++)
{
        if(loop%99 == 0)
        break;
}
Document.write(loop);
</Script>
</head>
</html>
```

8. USER DEFINED FUNCTION

```
<html>
<head>
<Script Language="JavaScript">
<!--
Function hello()
{
document.write("Hello");
document.write("********** Welcome to Hello() function **********");
return;
}
Function sum(one,two)
{
var res=one+two;
return res;
}
Function sum_all()
{
var loop=0,sum=0;
for(loop=arguments.length-1;loop>=0;loop--)
sum += arguments[loop];
return sum;
}
hello();
var total=sum(8,10);
document.write(" Sum of 8 & 10 = "+total+" Sum of 10+20 is : " + sum(10,20));
document.write(" Sum of the given series is : " + sum_all(10,20,40,50,70));
/-->
</Script>
</head>
</html>
```

9. BUILT IN OBJECTS

```
<html>
<head>
<Script Language = "JavaScript">
<!--
var bstr="big";
var sstr="small";
var blstr="bold";
var blkstr="blink";
var ucase="Uppercase";
var lcase="Lowercase";
var fon="fontcolor";
var itl="italics";
document.write("<br> This is "+bstr.big()+"text"):
document.write("<br> This is "+sstr.small()+"text"):
document.write("<br> This is "+blstr.bold()+"text"):
document.write("<br> This is "+blkstr.blink()+"text"):
document.write("<br> This is "+ucase.toUpperCase()+"text"):
document.write("<br> This is "+lcase.toLowerCase()+"text"):
document.write("<br> This is "+fon.fontColor()+"text"):
document.write("<br> This is "+itl.italics()+"text"):
//--
</Script>
</head>
</html>
```

EVENT HANDLING

Introduction

Events are the beating heart of any JavaScript application. On this page I give an overview of what event handling is, what its problems are and how to write proper cross-browser scripts. I will also provide pages with the gory details of event handling.

Events List

Window Events

Only valid in body and frameset elements.

Attribute	Value	Description
onload	*script*	Script to be run when a document loads
onunload	*script*	Script to be run when a document unloads

Form Element Events

Only valid in form elements.

Attribute	Value	Description
onchange	*script*	Script to be run when the element changes
onsubmit	*script*	Script to be run when the form is submitted
onreset	*script*	Script to be run when the form is reset
onselect	*script*	Script to be run when the element is selected
onblur	*script*	Script to be run when the element loses focus
onfocus	*script*	Script to be run when the element gets focus

---

Keyboard Events

Not valid in base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, and title elements.

Attribute	Value	Description
onkeydown	*script*	What to do when key is pressed
onkeypress	*script*	What to do when key is pressed and released
onkeyup	*script*	What to do when key is released

Mouse Events

Not valid in base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, title elements.

Attribute	Value	Description
onclick	*script*	What to do on a mouse click
ondblclick	*script*	What to do on a mouse double-click
onmousedown	*script*	What to do when mouse button is pressed
onmousemove	*script*	What to do when mouse pointer moves
onmouseout	*script*	What to do when mouse pointer moves out of an element
onmouseover	*script*	What to do when mouse pointer moves over an element
Onmouseup	*script*	What to do when mouse button is released

Application

- **Content Management** – Author/edit content, manage navigation, menus, audit trails, workflow, approvals
- **Web Site Analytics** - track visitors, report on most requested pages, see where they came from and where they are going.
- **HTML Form Engin**e – Create a two-way dialog with visitors using our wizard-like HTML form-building engine with validation and database support
- **Web Alerts** – Manage memberships and subscriptions and automatically email visitors when you add or update content that's relevant to them
- **Online Communities** - post **blogs**, add **forums** and message boards, rate content, post polls and surveys
- **Multilingual Support** – Integrated translation tools help you reach multiple audiences by speaking their language
- **Document Management** – Collaborate via the Web on Microsoft Word, PowerPoint, other documents, "save as" PDF & HTML, publish to Web
- **Online Calendars** – Build and manage Web-based calendars and enable users to customize a view based on what's important to them

10. WEB PAGE CREATION USING EVENT HANDLER

```
<html>
<head>
<Body>
<Font Size="10"> Welcome to net software solution </Font>
<Marquee Bgcolor="chyan"> Type your resume here </Marquee>
<Img Src="C:\image.jpg" Align="Right">
</Body>
<Head>
<pre>
<Form>
NAME                : <Input type="text" Name="TODAYSDATE" Value=""
Size="30" onFocus="this.selecr"><BR>
QUALIFICATION    : <Input type="text" Name="TODAYSDATE" Value=""
Size="30" onFocus="this.select"><BR>
AGE                 : <Input type="text" Name="TODAYSDATE" Value=""
Size="30" onFocus="this.select"><BR>
PHONE NUMBER    : <Input type="text" Name="TODAYSDATE" Value=""
Size="30" onFocus="this.select"><BR>
E-MAIL ID           : <Input type="text" Name="TODAYSDATE" Value=""
Size="30" onFocus="this.select"><BR>
GENDER              : <Input type="RADIO" Name="REDBUT">MALE
                      <Input type="RADIO" Name="REDBUT">FEMALE
ADDRESS             :
<TEXTAREA Name="TODAY" ROWS=3 COLS=20>
</TEXTAREA>
HOBBIES             :
<SELECT>
<OPTION> Watching TV
<OPTION> Playing Games
<OPTION>Reading Books
<OPTION>Others
</SELECT>
MARITAL STATUS :
<SELECT>
<OPTION>Single
<OPTION>Married
<OPTION>Wido
<OPTION>Others
</SELECT>
PREVIOUS EXPERIENCE  :
<Input type="RADIO" Name="REDBUT">YES<Input type="RADIO"
NAME="REDBUT">NO
PROJECT DETAILS :
UG:<Input type="text" Name="TODAYSDATE" Value="" Size="30"
onFocus="this.select">
PG:<Input type="text" Name="TODAYSDATE" Value="" Size="30"
onFocus="this.select">
PROFUNDITY       :
<B> LANGUAGES   : <B>
```

```
<Input type = "CHECKBOX">C
<Input type = "CHECKBOX">C++
<Input type = "CHECKBOX">Other
<B> DATABASE                : </B>
<Input type = "CHECKBOX">FOXPRO
<Input type = "CHECKBOX">VISUAL FOXPRO
<Input type = "CHECKBOX">Other
<B> OPERATING SYSTEM        : </B>
<Input type = "CHECKBOX">Windows
<Input type = "CHECKBOX">Linux
<Input type = "CHECKBOX">Other
</SELECT>
FATHERS NAME     : <Input Type="Text" Name="TODAYSDATE" Value=""
Size="20">
DOB                         : <Input type="text" Name="TODAYSDATE" Value=""
Size="30" onFocus="this.select">
</Pre>
</Form>
</head>
<Body bgcolor=blue >
REFERENCE:1:<TEXTAREA Name="TODAY" Rows=3 Cols=10>
</TEXTAREA>2:<TEXTAREA Name="TODAY" Rows=3 Cols=10>
</TEXTAREA><BR>
<CENTER>
<Input type="SUBMIT" value="SUBMIT">
<Input type="RESET" Value="RESET">
</CENTER>
</Form>
<HR>
<PRE>
        NET SOFTWARE SOLUTION
        17, SKS STREET,
        ERODE.
        WWW.NETSOFT.COM
        NETSFOT@YAHOO.COM
</PRE>
</Body>
</html>
```

11. IMAGE ROLLOVER EFFECTS

```
<html>
<head>
<title> Image rollover </title>
<Script language="JavaScript">
var high=new  image(200,200);
var low= new  image(200,200);
high.src="c:\water.jpg";
low.src="c:\sample.jpg"
function chhigh(n)
{
Document.images[n].src="c:\water.jpg";
}
Function chlow(n)
{
Document.images[n].src="c:\sample.jpg";
}
</Script>
<body>
<A HREF="" onclick="returnfalse" onmouseover="chhigh(0)"
onmouseout="chlow(0)"> </A>
<Img Src="c:\sample.jpg">
</head>
</body>
</html>
```

SOCKET PROGRAMMING

Introduction

This is an introduction to socket programming. The language will be Java, which is fast becoming the dominant language of the Networking world. Java allows a person to write and compile a program on a platform such as Windows 95, and take that code and run it without any changes on a Macintosh, Unix, Linux or Windows NT machine. That is why Microsoft are so afraid of the power of Java and have tried to stop Java from becoming the language for Network programming.

What is a Socket

The ***socket*** is the method for accomplishing interprocess communication (IPC). What this means is a socket is used to allow one process to speak to another, very much like the telephone is used to allow one person to speak to another. The telephone analogy is usually a good one, and will be used repeatedly to describe socket behavior.

How to listen for socket connections

In order for a person to receive telephone calls, he must first have a telephone installed. Likewise you must create a socket to listen for connections. This process involves several steps. First you must make a new socket, which is similar to having a telephone line installed. The **socket()** command is used to do this.

Since sockets can have several types, you must specify what type of socket you want when you create one. One option that you have is the addressing format of a socket. Just as the mail service uses a different scheme to deliver mail than the telephone company uses to complete calls, so can sockets differ. Sockets are useed for IPC between processes on the same machine. Socket addressing uses Internet addresses which are four-byte numbers usually written as four decimal numbers separated by periods (such as 192.9.200.10). In addition to the machine address, there is also a port number which allows more than one socket on each machine.

Another option which you must supply when creating a socket is the type of socket. The type that we will use here is the **Datagram** type. **Datagram** indicates that data will come in bunches (called ***datagrams***). Streaming is another type.

To create a socket:

DatagramSocket s = new DatagramSocket();

This is so much more easier than in C code. 1 line! After creating a socket, we Client part of the code (only) must give the socket an address to listen to, just as you get a telephone number so that you can receive calls.

int port = Integer.parseInt(argv[1]);
InetAddress in[] = InetAddress.getAllByName(argv[0]);

The above sample piece of code is used to do this (it binds a socket to an address).

Sockets have the ability to queue incoming connection requests, which is a lot like having "call waiting" for your telephone. If you are busy handling a connection, the connection request will wait until you can deal with it.

byte buf[] = new byte[10];
DatagramPacket dp = new DatagramPacket(buf, 10);

The above piece of code creates a buffer (10 in length) and waits for incoming messages to fill the buffer. After you create a socket to get calls, you must wait for calls to that socket. The **receive()** function is used to do this. Calling **receive()** is analogous to picking up the telephone if it's ringing. **receive()** returns a new socket which is connected to the caller. The following function can be used to accept a connection on a socket that has been created using the **DatagramSocket** code above:

s.receive(dp);

Unlike with the telephone, you may still accept calls while processing previous connections. For this reason you usually fork off jobs to handle each connection. You would use threads for this, however we do not need to concern ourselves with this at this stage.

How to call a socket

You now know how to create a socket that will accept incoming calls. So how do you call it? As with the telephone, you must first have the phone before using it to call. You use the **inetaddress()** function to do this, exactly as you establish a socket to listen to.

After getting a socket to make the call with, and giving it an address, you use the **DatagramSocket()()** function (yet again) to try to connect to a listening socket. The following code, creates a socket, binds to the Internet address where the Server is listening and then calls a particular port number on a particular host:

int port = Integer.parseInt(argv[1]);
InetAddress in[] = InetAddress.getAllByName(argv[0]);
DatagramSocket s = new DatagramSocket();

The above code basically returns a connected socket through which data can flow between client and server. It is basically creating a pipe between two processes.

How to talk between sockets

Now that you have a connection between sockets you want to send data between them. The **receive()** and **send()** functions are used to do this, just as they are for normal files.

The code to send in Java is as follows: **s.send(dp);**

There is only one major difference between socket reading and writing and file reading and writing: you don't usually get back the same number of characters that you asked for, so you must loop until you have read the number of characters that you want. A simple function to read a given number of characters into a buffer is:

Server Sample Code:

for (int i=0; i<buf.length; ++i) System.out.println(buf[i]);

Client Sample Code:

for (int i=0; i<buf.length; ++i) buf[i] = (byte)i;

What to do when you're done with a socket

Just as you hang up when you're through speaking to someone over the telephone, so must you close a connection between sockets. The normal **close()** function is used to close each end of a socket connection. If one end of a socket is closed and the other tries to write to its end, the write will return an error. The code to close a socket in Java is :

s.close();

12. DATAGRAM CLIENT AND SERVER

```
import java.io.*;
import java.lang.*;
import java.net.*;
class writeServer
{
public static int serverport=998;
public static int clientport=99;
public static int buffer_size=1024;
public static DatagramSocket ds;
public static byte buffer[]=new byte[buffer_size];
public static void TheServer() throws Exception
{ int pos=0;
 while(true) {
int c=System.in.read();
switch(c)
{
case     -1:
          System.out.println("Server quits");
case     '\r':
          break;
case     '\n':
ds.send(new DatagramPacket(buffer,pos,InetAddress.getLocalHost(),clientport));
pos = 0;
break;
default :
buffer[pos++]=(byte)c;
} } }
public static void TheClient() throws Exception
{
while(true) {
DatagramPacket p=new DatagramPacket(buffer,buffer.length);
ds.receive(p);
System.out.println(new String(p.getDate(),0,p.getLength()));
} }
public static void main(String args[]) throws Exception
{
if(args.length==1)
{
ds=new DatagramSocket(serverport);
TheServer();
}
Else {
ds=new DatagramSocket(clientport);
TheClient();
} } }
```

13. CREATION TCP CLIENT SERVER

```
import java.io.*;
import java.net.*;
import java.lang.*;
public class simpleserver
{
public static void main(String args[])
{
ServerSocket s=null;
try
{
s=new ServerSocket(5433);
}
catch(IOException e)
{
e.printStackTrace();
}
while(true)
{
try{
DataInputStream in = new DataInputStream(System.in);
Socket s1 = s.accept();
OutputStream s1out = s1.getOutputStream();
BufferedWriter bw=new BufferedWriter(new OutputStreamWriter(s1out));
bw.write("hello");
bw.close();
s1.close();
}
catch(IOException e)
{
e.printStackTrace();
}
}
}
}
```

```
import java.io.*;
import java.net.*;
import java.lang.*;
public class simpleclient
{
public static void main(String args[])
{
try
{
socket s1=new socket("172.16.0.96",5432);
BufferedReader br = new BufferedReader(new
inputStreamReader(s1.getInputStream()));
System.out.println(br.readLine());
br.close();
s1.close();
}
catch(ConnectException ce)
{
System.out.println("connection failed");
}
catch(IOException e)
{
}
}
}
```

ACTIVE SERVER PAGES

Introduction:

An ASP file can contain text, HTML tags and scripts. Scripts in an ASP file are executed on the server

- ASP stands for **A**ctive **S**erver **P**ages
- ASP is a program that runs inside **IIS**
- IIS stands for **I**nternet **I**nformation **S**ervices
- IIS comes as a free component with **Windows 2000**
- IIS is also a part of the **Windows NT 4.0 Option Pack**
- The Option Pack can be **downloaded** from Microsoft
- **PWS** is a smaller - but fully functional - version of IIS
- PWS can be found on your **Windows 95/98 CD**

Applications:

The Application object is used to store and access variables from any page, just like the Session object. The difference is that ALL users share one Application object, while with Sessions there is one Sessio. The Application object should hold information that will be used by many pages in the application n object for EACH user.

Syntax:

The Basic Syntax Rule

An ASP file normally contains HTML tags, just like an HTML file. However, an ASP file can also contain **server scripts**, surrounded by the delimiters **<%** and **%>**. Server scripts are **executed on the server,** and can contain any expressions, statements, procedures, or operators valid for the scripting language you prefer to use.

Write Output to a Browser

The response.write command is used to write output to a browser. The following example sends the text "Hello World" to the browser:

```
<html>
<body>
<%
response.write("Hello World!")
%>
</body>
<html>
```

There is also a shorthand method for the response.write command. The following example also sends the text "Hello World" to the browser:

```
<html>
<body>
<%="Hello World!"%>
</body>
<html>
```

JavaScript

To set JavaScript as the default scripting language for a particular page you must insert a language specification at the top of the page:

```
<%@ language="javascript"%>
<html>
<body>
<%
Response.Write("Hello World!")
%>
</body>
</html>
```

Advantages

ASP is a server side script and it can perform many backend processing jobs for a web site, same way it can't do many things particularly the client end operations as it has no control over the client end events or user controls

Disadvantages

It is better to say the limitation of any server side scripting language, as these limitations are faced by other scripting languages like PHP, JSP, Perl etc. We will try to understand **where ASP can be used and where it should not be used**. Using server side scripting language like ASP we can manage the content of any page and such dynamic code (or content) for the web browsers can be generated based on various conditions we set in our ASP program. ASP engine finishes its job of processing the code and then send the codes to users browser. From this point on words till again the page request comes back to server, there is no control of ASP on the page

14. DISPLAYING CALENDAR

```
<html>
<body>
<%@Language="VBScript"%>
<%Function MonthName(iMonth)
Select Case iMonth
Case 1:
        MonthName = "January"
Case 2:
        MonthName = "February"
Case 3:
        MonthName = "March"
Case 4:
        MonthName = "April"
Case 5:
        MonthName = "May"
Case 6:
        MonthName = "June"
Case 7:
        MonthName = "July"
Case 8:
        MonthName = "August"
Case 9:
        MonthName = "September"
Case 10:
        MonthName = "October"
Case 11:
        MonthName = "November"
Case 12:
        MonthName = "December"
Case Else
        MonthName = "Invalid"
End Select
End Function

Dim dbCurrentDate
dbCurrentDate = Date()
Dim CalDays(42)
Dim WeekDay
WeekDay =
DatePart("w",DateSerial(Year(dbCurrentDate),Month(dbCurrentDate),1))
Dim DaysMonth
DaysMonth =
DatePart("d",DateSerial(Year(dbCurrentDate),Month(dbCurrentDate)+1,1-1))
Dim iLoop
For iLoop = 1 to DaysMonth
        calDays(iLoop + WeekDay - 1) = iLoop
Next
Dim iCols, iRows
iCols = 7
```

```
iRows = 6-Int((42-(Weekday+DaysMonth))/7) %>
<Table align=center border=1 cellspacing=1 width=75% height=75%>
<th colspan=7>
<font size = +2>
<% Response.Write MonthName(Month(dbCurrentDate))
Response.Write "," & Year(dbCurrentdate)%>
</font></th>
<% Dim iRowsLoop,iColsLoop
For iRowsLoop = 1 to iRows
        Response.Write "<TR>"
        For iColsLoop = 1 to iCols
                ifCalDays((iRowsLoop-1)*7 + iColsLoop)>0 then
                        Response.Write "<TD VALIGN=TOP ALIGN=RIGHT>"
                        Response.Write CalDays(iRowsLoop-1)*7+iColsLoop)
                        Response.Write "</TD>"
                Else
                        Response.Write "<TD BGCOLOR=forestgreen> </td>"
                End if
        Next
        Response.Write "</tr>"
Next %>
</table>
</body></html>
```

15. REGISTRATION FORM

```
<html>
<head>
<title> Registration Form </title>
</head>
<body>
<h2>
<center>
Registration Form </center>
</h2>
<br>
<br>
<FORM action="confirm.asp" method="post">
<p>
Name:
<Input type=text size=20 Name="firstname"><br><br>
Address:
<textarea cols=20 rows=3 NAME="address"></textarea><br>
<br>
Qualification:
<Input type="radio" Name="qualif" value="Post Graduate">
Post Graduate:
<br>
<br>
Select Start Date:
<Select name="Startdate" Size="1">
<OPTION VALUE="1" SELECTED>10TH  JANUARY, 2006
<OPTION VALUE="2" >17TH JANUARY, 2006
<OPTION VALUE="3" >24TH JANUARY, 2006
</SELECT>
<br>
<br>
<INPUT TYPE = "SUBMIT" VALUE = "REGISTER">
<INPUT TYPE = "RESET" VALUE = "RESET">
</Form>
</body>
</html>
```

```
<html>
<body>
<%@ language="VBScript" %>
<%
If Request.Form("firstname")="" Then Response.Redirect
http://localhost/tha/tha/regis.html
%>
<%
If Request.Form("address")="" Then Response.Redirect
http://localhost/tha/tha/regis.html
%>
<%
If Request.Form("qualif")="" Then Response.Redirect
http://localhost/tha/tha/regis.html
%>
<%
firstname=Request.Form("firstname")
startdate=Request.form("startdate")
%>
<%
Select Case startdate
case 1
disp="10th January,2006"
case 2
disp="17th January,2006"
case 3
disp="24th January,2006"
End Select
%>
</body>
</html>
```

JAVA TO DATABASE CONNECTIVITY

Introduction

The Java Database Connectivity (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases – SQL databases and other tabular data sources, such as spreadsheets or flat files. The JDBC API provides a call-level API for SQL-based database access.

Steps For JDBC

- **Creating a database**. You can either create the database outside of Java, via tools supplied by the database vendor, or via SQL statements fed to the database from a Java program.
- **Connecting to an ODBC data source**. An ODBC data source is a database that is registered with the ODBC driver. In Java you can use either the JDBC to ODBC bridge, or JDBC and a vendor-specific bridge to connect to the datasource.
- **Inserting information into a database**. Again, you can either enter data outside of Java, using database-specific tools, or with SQL statements sent by a Java program.
- **Selectively retrieving information**. You use SQL commands from Java to get results and then use Java to display or manipulate that data.
-

Components

Load the JDBC-ODBC bridge. You must load a driver that tells the JDBC classes how to talk to a data source. In this case, you will need the class JdbcOdbcDriver:

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

This can also be specified from the command line via the jdbc.drivers system property:

Connect to a data source. A URL is used to connect to a particular JDBC data source. See Section 3.1.2 **URLs in General USE** and 3.1.3 **JDBC URLs** in the JDBC Guide for more information. Given that you have loaded the JDBC-ODBC bridge driver, URLs should be in the following form jdbc:odbc:*data-source-name*. Using the DriverManager class, you request a connection to a URL and the DriverManager selects the appropriate driver; here, only the driver JdbcOdbcDriver is loaded.

Connection con = DriverManager.getConnection(*URL, username, password*);

Where the *username* and *password* are empty strings, "", in this case because text files acting as ODBC data sources cannot have such attributes.

Send SQL statements to create the table. Ask the connection object for a Statement object:

Statement stmt = con.createStatement();

Queries:

All pl\sql queries are works under the jdbc.

Features

Rowsets

A rowset encapsulates a set of rows from a result set and may maintain an open database connection or be disconnected from the data source. A rowset is a JavaBeans *tm* component; it can be created at design time and used in conjunction with other JavaBeans components in a visual JavaBeans builder tool to construct an application.

JNDI *tm* for Naming Databases

The Java *tm* Naming and Directory Interface *tm* (JNDI) makes it possible to connect to a database using a logical name instead of having to hard code a particular database and driver.

Connection Pooling

A connection pool is a cache of open connections that can be used and reused, thus cutting down on the overhead of creating and destroying database connections.

Distributed Transaction Support

Support for distributed transactions allows a JDBC driver to support the standard two-phase commit protocol used by the Java Transaction API (JTA). This feature facilitates using JDBC functionality in Enterprise JavaBeans components

16. STUDENT MARKLIST

```
import java.sql.*;
import java.io.*;
import java.lang.*;
class mark
{
public String name, rno;
public int ma1,ma2,ma3,ma4,ma5,tot,avg;
private void getdata() throws IOException, SQLException
{
try
{
DataInputStream in = new DataInputStream(System.in);
System.out.println("Creation Details");
System.out.print("Enter the Name");
name = in.readLine();
System.out.print("Enter the rollno:");
rno = in.readLine();
System.out.print("Enter the mark 1");
ma1 = Integer.parseInt(in.readLine());
System.out.print("Enter the mark 2");
ma2 = Integer.parseInt(in.readLine());
System.out.print("Enter the mark 3");
ma3 = Integer.parseInt(in.readLine());
System.out.print("Enter the mark 4");
ma4 = Integer.parseInt(in.readLine());
System.out.print("Enter the mark 5");
ma5 = Integer.parseInt(in.readLine());
}
catch(Exception e)
{
System.out.println(e);
}
}
public void create() throws IOException, SQLException
{
Connection con;
Statement st;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con= DriverManager.getConnection("jdbc:odbc:mark","","");
st = con.createStatement();
DataInputStream in = new DataInputStream(System.in);
getdata();
tot = ma1 + ma2 + ma3 + ma4+ ma5;
avg = tot/5;
```

```
String query = “insert into stud
values(‘”+name+”’,”’+rno+”’,”+ma1+”,”+ma2+”,”+ma3+”,”+ma4+”,”+ma5+”,”+tot
+”,”+avg+”);”;
st.execute(query);
st.close();
con.close();
}
catch(Exception e)
{
System.out.println(“This roll no : “+rno+” is already created”);
}
}
void cancel()
{
Connection con;
Statement st;
ResultSet rs;
try
{
Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
con= DriverManager.getConnection(“jdbc:odbc:mark”,””,””);
st = con.createStatement();
DataInputStream in = new DataInputStream(System.in);
System.out.print(“enter roll no:”);
rno = in.readLine();
String query1 = “delete * from stud where rno = ‘”+rno+”’;”;
st.executeUpdate(query1);
System.out.println(“The acc no: “+rno+” is canceled”);
st.close();
}
catch(Exception ex)
{
System.out.println(“Exception :”+ex);
}
}
void read()
{
Connection con;
Statement st;
ResultSet rs;
try
{
Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
con= DriverManager.getConnection(“jdbc:odbc:mark”,””,””);
st = con.createStatement();
String query = “select * from stud order by rno;”;
rs = st.executeQuery(query);
System.out.println(“\n\tName\tRollno\tmark1\tmark2\mark3\tmark4\tmark5\tTotal\tA
verage”);
while(rs.next())
{
```

```
name = rs.getString("name");
rno =  rs.getString("rno");
ma1 = rs.getString("ma1");
ma2 = rs.getString("ma2");
ma3 = rs.getString("ma3");
ma4 = rs.getString("ma4");
ma5 = rs.getString("ma5");
tot = rs.getString("tot");
avg = rs.getString('avg");
System.out.println(""\t"+name+"\t"+rno+"\t"+ma1+"\t"+ma2+"\t"+ma3+"\t"+ma4+"
\t"+ma5+"\t"+tot+"\t"+avg);
rs.close();
st.close();
con.close();
}
catch(Exception ex)
{
System.out.println("Exception:"+ex);
}
}
void update()
{
Connection con;
Statement st;
ResultSet rs;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con= DriverManager.getConnection("jdbc:odbc:mark","","");
st = con.createStatement();
DataInputStream in = new DataInputStream(System.in);
System.out.print("enter the name");
name = in.readLine();
System.out.print("Enter the rollno:");
rno = in,readLine();
String query = "update stud set name = '"=name+"',rno = '"+rno+"' where no =
'"+rno+"';";
st.executeUpdate(query);
System.out.println("The roll no"+rno+" is Updated");
st.close();
con.close();
}
catch(Execetion e)
{
System.out.println("Exception"+e);
}
}
public static void main(String[] args) throws IOException
{
DataInputStream in = new DataInputStream(System.in);
try
```

```
{
int choice = 0;
String flag = “y”;
mark obj = new mark();
while(flag = “y”)
{
System.out.println(“1.creation”);
System.out.println(“2.cancelation”);
System.out.println(“3.Updation”);
System.out.println(“4.Display”);
System.out.println(“Type Other Keys to Exit”);
System.out.println(“Enter the Choice”);
choice = Integer.parseInt(in.readLine());
switch(choice)
{
case 1:
obj.create();
break;
case 2:
obj.cancel();
break;
case 3:
obj.update();
break
case 4:
obj.read();
break;
default:
System.exit(0);
}
}
}
catch(Execetion e)
{System.out.println(“Exception”+e);}
}
```

17. LIBRARY MANAGEMENT

```
import java.lang.*;
class lib
{
public String name, address, bookname, acctype, author, edition;
public int pno, accno, bookno, date, redate, noofbooks, month;
public static int tot = 10;
private void getdata() throws IOException, SQLException
{
try
{
DataInputStream in = new DataInputStream(System.in);
System.out.println("Creation Details");
System.out.println("Enter the name");
name = in.readLine();
System.out.println("Enter the address");
address = in.readLine();
System.out.println("Enter the phone Number");
pno = Integer.parseInt(in.readLine());
System.out.println("Enter the ACC number");
accno = Integer.parseInt(in.readLine());
}
catch(Execetion e)
{
System.out.println("Exception"+e);
}
}
public void reserve() throws IOException, SQLException
{
Connection con;
Statement st;
ResultSet rs;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con= DriverManager.getConnection("jdbc:odbc:mark","","");
st = con.createStatement();
DataInputStream in = new DataInputStream(System.in);
getdata();
System.out.println("enter the bookno");
bookno = Integer.parseInt(in.readLine());
System.out.println("enter the date");
date = Integer.parseInt(in.readLine());
System.out.println("enter the month");
month = Integer.parseInt(in.readLine());
System.out.println("enter the bookname");
bookname = in.readLine();
```

```
redate = redate + 10;
System.out.println("your return date is "+ redate);
String query = "insert into customer1
values('"+name+"','"+address+"','"+accno+"','"+bookname+"','"+bookno+","+date
+","+month+");";
st.execute(query);
System.out.println("the Acc no :"+accno+" is created");
st.close();
con.close();
}
catch(Execetion e)
{
System.out.println("This acc no :"+ accno+" is already created");
}
}
void cancel()
{
Connection con;
Statement st;
ResultSet rs;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con= DriverManager.getConnection("jdbc:odbc:mark","","");
st = con.createStatement();
DataInputStream in = new DataInputStream(System.in);
System.out.println("enter the accno");
accno = Integer.parseInt(in.readLine());
String query1 = "delete * from customer1 where accno = "+accno+";";
st.executeUpdate(query1);
System.out.println("the acc no:"+ accno+"is canceled");
st.close();
con.close();
}
catch(Execetion e)
{
System.out.println("This acc no :"+ accno+" is already created");
}
}
void read()
{
Connection con;
Statement st;
ResultSet rs;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con= DriverManager.getConnection("jdbc:odbc:mark","","");
st = con.createStatement();
String query = "select * from customer1 order by accno;";
rs = st.executeQuery(query);
```

```
System.out.println("\n\tName\tAddress\tPhonno\taccno\tBookname\tBookno\t");
while(rs.next())
{
name = rs.getString("name");
address = rs.getString("address");
pno = rs.getString("pno");
accno = rs.getString("accno");
bookname = rs.getString("bookname");
bookno = rs.getString("bookno");
System.out.println("\t"+name+"\t"+address+"\t"+pno+"\t"+accno+"\t"+bookname+"\
t"+bookno+"\t");
rs.close();
st.close();
con.close();
}
catch(Exception e)
{
System.out.println("Exception"+e);
}
}
void bookread()
{
Connection con;
Statement st;
ResultSet rs;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con= DriverManager.getConnection("jdbc:odbc:mark","","");
st = con.createStatement();
String query = "select * from lib order by bookno;";
rs = st.executeQuery(query);
System.out.println("\n\tBookname\tBookno\tAuthor\tEdition\tNo.of books\t");
while(rs.next())
{
bookname = rs.getString("bookname");
bookno = rs.getInt("Bookno");
author = rs.getString("author");
edition = rs.getString("Edition");
noofbooks = rs.getInt("noofbooks");
        System.out.println("\t"+bookname+"\t"+bookno+"\t"+author+"\t"+edition+"\t
"+noofbooks+"\t");
}
rs.close();
st.close();
con.close();
}
catch(Exception e)
{
System.out.println("Exception:"+ex);
```

```
}
}
void update()
{
Connection con;
Statement st;
ResultSet rs;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con= DriverManager.getConnection("jdbc:odbc:mark","","");
st = con.createStatement();
read();
getdata();
String query = "update customer1 set name = '"+name+"', address = '"+address+"',
pno = "+pno+", accno = "+accno+" where accno = "+accno+";";
st.executeUpdate(query);
System.out.println("the Acc no "+ accno +" is updated");
st.close();
con.close();
}
catch(Exception e)
{
System.out.println("Exception "+e);
}
}
void withdraw()
{
Connection con;
Statement st;
ResultSet rs;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con= DriverManager.getConnection("jdbc:odbc:mark","","");
st = con.createStatement();
DataInputStream in = new DataInputStream(System.in);
String query3 = "select date, month from customer1 where accno = "+acno+";";
rs = st.executeQuery(query3);
while(rs.next())
{
date = rs.getInt("date");
month = rs.getInt("month");
}
System.out.println("enter the accno");
accno = Integer.parseInt(in.readLine());
System.out.print("enter the bookno")l
bookno = Integer.parseInt(in.readLine());
System.out.println("Enter the date, month");
d1 = Integer.parseInt(in.readLine());
low = Integer.parseInt(in.readLine());
```

```
System.out.println("Enter the redate, month");
d2 = Integer.parseInt(in.readLine());
hi = Integer.parseInt(in.readLine());
System.out.println("Enter the year");
y = Integer.parseInt(in.readLine());
int mid = (low + hi)/2;
int r = y/4;
if(low == hi)
{
a = d2 – d1;
if(a>10)
{
int ff = a-10;
int fine = ff * 2;
System.out.println("Your fine amount is"+fine);
String query1 = "delete * from customer1 where accno = "+accno+";";
st.execturteUpdate(query1);
}
else
{
System.out.println("successfully returned");
String query2 = "update customer1 set bookno = "+0+",bookname = '"+null+"'
where accno = '"+accno+";";
st.executeUpdate(query2);
}
}
else
{
for(int i = low + 1; i<hi;i++)
{
if(mid<=8)
{
r1 = mid%2;
if(r==0)
{
if(mid == 2)
dn += 29;
else if(mid == 2)
dn+=28;
else if(r1 == 0)
dn += 30;
else
dn += 31;
}
}
if(mid>8)
{
r1 = mid%2;
if(r1 == 0)
dn += 31;
else
```

```
dn += 30;
}
mid++;
}
if(low == 2 && r == 0)
{
d = 29-d1;
}
else if(low == 2)
d = 28-d1;
else if(low <= 8)
r2 = low%2;
if(r2==0)
d = 30-d1;
else
d = 31 – d1;
}
}
if(low>8)
r2 = low%2;
if(r2 == 0)
d = 31-d1;
else
d = 30-d1;
di = dn + d + d2;
if(di > 10)
{
int ff = di – 10;
int fine = di * 2;
System.out.println(“your Fine amount is”+fine);
String query1 = “delete * from customer1 where accno = “+accno+”;”;
st.executeUpdate(query1);
}
else
{
System.out.println(“Successfully Returned”);
String query2 = “Update customer1 set bookno = “+0+”,bookname = ‘”+null+”’,
where accno = “+accno+”,
st.executeUpdate(query2);
}
}
}
st.close();
con.close();
}
catch(Exception e)
{
System.out.println(“Exception “+e);
}
}
void borrow()
```

```
{
Connection con;
Statement st;
ResultSet rs;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con= DriverManager.getConnection("jdbc:odbc:mark","","");
st = con.createStatement();
DataInputStream in = new DataInputStream(System.in);
System.out.print("Enter the Acc Number:");
accno = Integer.parseInt(in.readLine());
System.out.println("Enter the  BOOKno:");
bookno = Integer.parseInt(in.readLine());
System.out.println("Enter the  month:");
month = Integer.parseInt(in.readLine());
System.out.println("Enter the  BOOK name:");
bookname = in.readLine();
String query = "update customer1 set bookname = '"+bookname+"', bookno =
"+bookno+",date = "+date+", month = "+month+", where accno = "+accno+";";
st.executeUpdate(query);
st.close();
con.close();
}
catch(Exception e)
{
System.out.println("Exception "+e);
}
}
public static void main(String[] args) throws IOException
{
DataInputStream in = new DataInputStream(System.in);
try
{
int choice = 0;
String flag = "y";
lib obj = new lib();
while(flag == "y")
{
System.out.println("1.Creation");
System.out.println("2.Cancellation");
System.out.println("3.Updation");
System.out.println("4.Print");
System.out.println("5.Return");
System.out.println("6.Book details");
System.out.println("7.book Borrow");
System.out.println("Type any key to exit");
System.out.println("Enter the CHoice");
choice = Integer.parseInt(in.readeLine());
switch(choice)
{
```

```
case 1:
obj.reserve();
break;
case 2:
obj.cancel();
case 3:
obj.update();
break;
case 4:
obj.read();
break;
case 5:
obj.withdraw();
break;

case 6:
obj.bookread();
break;
case 7:
obj.borrow();
break;
default:
System.exit(0);
}
}
}
catch(Exception e)
{
System.out.println("Exception "+e);
}

}
```

SERVLET

A servlet is a Java™ component that can be plugged into a Java-enabled web server to provide custom services. These services can include:

- New features
- Runtime changes to content
- Runtime changes to presentation
- New standard protocols (such as FTP)
- New custom protocols

Servlets are designed to work within a request/response processing model. In a request/response model, a client sends a request message to a server and the server responds by sending back a reply message. Requests can come in the form of an

HTTP
URL,
FTP,
URL,
or a custom protocol.

The request and the corresponding response reflect the state of the client and the server at the time of the request. Normally, the state of the client/server connection cannot be maintained across different request/response pairs. However, session information is maintainable with servlets through means to be described **later**.

The Java Servlet API includes several Java interfaces and fully defines the link between a hosting server and servlets. The Servlet API is defined as an extension to the standard JDK. JDK extensions are packaged under javax--the root of the Java extension library tree. The Java Servlet API contains the following packages:

- Package **javax.servlet**
- Package **javax.servlet.http**

Servlets are a powerful addition to the Java environment. They are fast, safe, reliable, and 100% pure Java. Because servlets plug into an existing server, they leverage a lot of existing code and technology. The server handles the network connections, protocol negotiation, class loading, and more; all of this work does not need to be replicated! And, because servlets are located at the middle tier, they are positioned to add a lot of value and flexibility to a system.

Architectural Roles for Servlets

Because of their power and flexibility, servlets can play a significant role in a system architecture. They can perform the application processing assigned to the middle tier, act as a proxy for a client, and even augment the features of the middle tier by adding support for new protocols or other features. A middle tier acts as the application server in so called three-tier client/server systems, positioning itself between a lightweight client like a web browser and a data source.

Middle-Tier Process

In many systems a middle tier serves as a link between clients and back-end services. By using a middle tier a lot of processing can be off-loaded from both clients (making them lighter and faster) and servers (allowing them to focus on their mission).

One advantage of middle tier processing is simply connection management. A set of servlets could handle connections with hundreds of clients, if not thousands, while recycling a pool of expensive connections to database servers.

Other middle tier roles include:

- Business rule enforcement
- Transaction management
- Mapping clients to a redundant set of servers
- Supporting different types of clients such as pure HTML and Java capable clients

Proxy Servers

When used to support applets, servlets can act as their proxies. This can be important because Java security allows applets only to make connections back to the server from which they were loaded. If an applet needs to connect to a database server located on a different machine, a servlet can make this connection on behalf of the applet.

Protocol Support

The Servlet API provides a tight link between a server and servlets. This allows servlets to add new protocol support to a server. (You will see how HTTP support is provided for you in the API packages.) Essentially, any protocol that follows a request/response computing model can be implemented by a servlet. This could include:

- SMTP
- POP
- FTP

Servlet support is currently available in several web servers, and will probably start appearing in other types of application servers in the near future. You will use a web server to host the servlets in this class and only deal with the HTTP protocol.

Because HTTP is one of the most common protocols, and because HTML can provide such a rich presentation of information, servlets probably contribute the most to building HTTP based systems.

HTML Support

HTML can provide a rich presentation of information because of its flexibility and the range of content that it can support. Servlets can play a role in creating HTML content. In fact, servlet support for HTML is so common, the **javax.servlet.http** package is dedicated to supporting HTTP protocol and HTML generation.
Complex web sites often need to provide HTML pages that are tailored for each visitor, or even for each hit. Servlets can be written to process HTML pages and

customize them as they are sent to a client. This can be as simple as on the fly substitutions or it can be as complex as compiling a grammar-based description of a page and generating custom HTML.

Inline HTML Generation

Some web servers, such as the Java Web Server™ (JWS), allow servlet tags to be embedded directly into HTML files. When the server encounters such a tag, it calls the servlet while it is sending the HTML file to the client. This allows a servlet to insert its contribution directly into the outgoing HTML stream.

Server-Side Includes

Another example is on the fly tag processing known as ***server-side includes*** (SSI). With SSI, an HTML page can contain special commands that are processed each time a page is requested. Usually a web server requires HTML files that incorporate SSI to use a unique extension, such as .shtml. As an example, if an HTML page (with an .shtml extension) includes the following: <!--#include virtual="/includes/page.html"--> it would be detected by the web server as a request to perform an inline file include. While server side includes are supported by most web servers, the SSI tags are not standardized. Servlets are a great way to add server side include processing to a web server. With more and more web servers supporting servlets, it would be possible to write a standard SSI processing servlet and use it on different web servers.

Replacing CGI Scripts

An HTTP servlet is a direct replacement for Common Gateway Interface (CGI) scripts. HTTP servlets are accessed by the user entering a URL in a browser or as the target of an HTML form action. For example, if a user enters the following URL into a browser address field, the browser requests a servlet to send an HTML page with the current time: http://localhost/servlet/DateTimeServlet The DateTimeServlet responds to this request by sending an HTML page to the browser.
Note that these servlets are not restricted to generating web pages; they can perform any other function, such as storing and fetching database information, or opening a socket to another machine.

Installing Servlets

Servlets are not ***run*** in the same sense as applets and applications. Servlets provide functionality that extends a server. In order to test a servlet, two steps are required:

1. Install the servlet in a hosting server
2. Request a servlet's service via a client request

There are many web servers that support servlets. It is beyond the scope of this course to cover the different ways to install servlets in each server. This course examines the JSDK's servletrunner utility and the JWS.

Temporary versus Permanent Servlets

Servlets can be started and stopped for each client request, or they can be started as the web server is started and kept alive until the server is shut down. Temporary servlets are loaded on demand and offer a good way to conserve resources in the server for less-used functions.

Permanent servlets are loaded when the server is started, and live until the server is shutdown. Servlets are installed as permanent extensions to a server when their start-up costs are very high (such as establishing a connection with a DBMS), when they offer permanent server-side functionality (such as an RMI service), or when they must respond as fast as possible to client requests. There is no special code necessary to make a servlet temporary or permanent; this is a function of the server configuration.

Because servlets can be loaded when a web server starts, they can use this auto-loading mechanism to provide easier loading of server-side Java programs. These programs can then provide functionality that is totally unique and independent of the web server. For example, a servlet could provide R-based services (rlogin, rsh, ...) through TCP/IP ports while using the servlet request/response protocol to present and process HTML pages used to manage the servlet.

Using servletrunner

For both JDK 1.1 and the Java 2 platform, you need to install the Java Servlet Development Kit (JSDK). To use servletrunner, make sure your PATH environment variable points to its directory. For the JSDK 2.0 installed with all default options, that location is: c:\jsdk2.0\bin on a Windows platform. To make sure that servletrunner has access to the Java servlet packages, check that your CLASSPATH environment variable is pointing to the correct JAR file, c:\jsdk2.0\lib\jsdk.jar on a Windows platform. With the Java 2 platform, instead of modifying the CLASSPATH, it is easier to just copy the JAR file to the ext directory under the Java runtine environment. This treats the servlet packages as standard extensions.
With the environment set up, run the servletrunner program from the command line. The parameters are:

Usage: servletrunner [options]
Options:
-p port the port number to listen on
-b backlog the listen backlog
-m max maximum number of connection handlers
-t timeout connection timeout in milliseconds
-d dir servlet directory
-s filename servlet property file name

The most common way to run this utility is to move to the directory that contains your servlets and run servletrunner from that location. However, that doesn't automatically configure the tool to load the servlets from the current directory.

18. STUDENT INFORMATION SYSTEM

```
import java.io.*;
import javax.servlet.http.*;
import java.text.*;
import javax.servlet.*;
import java.util.*;
public class Display extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException, ServletException
{
int total = 0;
float avg=0.0F;
String remark="";
int m1=Integer.parseInt(request.getParameter("m1"));
int m2=Integer.parseInt(request.getParameter("m2"));
int m3=Integer.parseInt(request.getParameter("m3"));
int m4=Integer.parseInt(request.getParameter("m4"));
int m5=Integer.parseInt(request.getParameter("m5"));
PrintWriter pw = response.getWriter();
pw.println("<html>");
pw.println("<body>");
if(m1>=50&&m2>50&& m3>50&& m4>50&&m5>=50)
{
total=m1+m2+m3+m4+m5;
avg=total/5;
}
if(avg>=80)
{
remark="A GRADE";
}
elseif(avg>70 && avg<80)
{
remark="B GRADE";
}
elseif(avg>60 && avg<70)
{
remark="C GRADE";
}
else
{
remark="FAIL";
}
pw.println("The total is"+total);
pw.println("The average is"+avg);
pw.println("The remark is"+remark);
pw.println("</body>");
pw.println("</html>");
pw.close();}}
```

```
<html>
<body>
<form method=Get action="http://localhost:8080/examples/servlet/Display">
enter m1<Input type=text name=m1 value=""><br>
enter m2<Input type=text name=m2 value=""><br>
enter m3<Input type=text name=m3 value=""><br>
enter m4<Input type=text name=m4 value=""><br>
enter m5<Input type=text name=m5 value=""><br>
<Input type=submit value="SUBMIT">
</form>
</body>
</html>
```

JAVA SERVER PAGES

JavaServer Pages (JSP) lets you separate the dynamic part of your pages from the static HTML. You simply write the regular HTML in the normal manner, using whatever Web-page-building tools you normally use. You then enclose the code for the dynamic parts in special tags, most of which start with "<%" and end with "%>".

You normally give your file a .jsp extension, and typically install it in any place you could place a normal Web page. Although what you write often looks more like a regular HTML file than a servlet, behind the scenes, the JSP page just gets converted to a normal servlet, with the static HTML simply being printed to the output stream associated with the servlet's service method. This is normally done the first time the page is requested, and developers can simply request the page themselves when first installing it if they want to be sure that the first real user doesn't get a momentary delay when the JSP page is translated to a servlet and the servlet is compiled and loaded. Note also that many Web servers let you define aliases that so that a URL that appears to reference an HTML file really points to a servlet or JSP page.

Aside from the regular HTML, there are three main types of JSP constructs that you embed in a page: scripting elements, directives, and actions. *Scripting elements* let you specify Java code that will become part of the resultant servlet, *directives* let you control the overall structure of the servlet, and *actions* let you specify existing components that should be used, and otherwise control the behavior of the JSP engine. To simplify the scripting elements, you have access to a number of predefined variables such as request in the snippet above.

JSP Scripting Elements

JSP scripting elements let you insert Java code into the servlet that will be generated from the current JSP page. There are three forms:

1. Expressions of the form <%= expression %> that are evaluated and inserted into the output,
2. Scriptlets of the form <% code %> that are inserted into the servlet's service method, and
3. Declarations of the form <%! code %> that are inserted into the body of the servlet class, outside of any existing methods.

Each of these is described in more detail below.

JSP Expressions

A JSP *expression* is used to insert Java values directly into the output. It has the following form:

<%= **Java Expression** %>

The Java expression is evaluated, converted to a string, and inserted in the page. This evaluation is performed at run-time (when the page is requested), and thus has full access to information about the request. For example, the following shows the date/time that the page was requested:

Current time: <%= new java.util.Date() %>

To simplify these expressions, there are a number of predefined variables that you can use. These implicit objects are discussed in more detail later, but for the purpose of expressions, the most important ones are:

- request, the HttpServletRequest;
- response, the HttpServletResponse;
- session, the HttpSession associated with the request (if any); and
- out, the PrintWriter (a buffered version of type JspWriter) used to send output to the client.

19. TOUR PLAN OF A VIP

```
<html>
<body>
<%! String cities[].month;%>
<p>
The Indian President is going a tour to the following Metropolitan cities.....</p>
<%
cities=request.getParameterValues("city");
month=request.getParameter("month");
if(cities!=null){
%>
<ul>
<%
for(int i=0;i<cities.length;i++){
%>
<li><%out.write(cities[i]); %>
<%
}
%>
</ul>
<%
}
else{
%>
<p>none selected</p>
<%
}
%>
<br>
<p>
In the month of<b>
<%out.write(month);%></b></p>
</body>
</html>
```

```
<html>
<body>
<form action = "http://localhost:8080/examples/jsp/city.jsp">
<Input type=checkbox name=city values="NewDelhi">NewDelhi<br>
<Input type=checkbox name=city values="Mumbai">Mumbai<br>
<Input type=checkbox name=city values="Kolkatta">Kolkatta<br>
<Input type=checkbox name=city values="Chennai">Chennai<br>
<Input type=checkbox name=month values="January">January<br>
<Input type=checkbox name=month values="March">March<br>
<Input type=checkbox name=month values="April">April<br>
<Input type=checkbox name=month values="May">May<br>
<Input type=submit name=SUBMIT>
</Form>
</body>
</html>
```

20. CURRENT DATE DISPLAYING

```
<html>
<body>
HELLO! THE TIME IS NOW; <%=new java.util.Date()%>
</body>
</html>
```

```
<html>
<body>
<Form action="http://localhost:8080/examples/jsp/currdate.jsp">
<Input type="Submit" Values="Click me">
</Form>
</body>
</html>
```