PACE UNIVERSITY

Thesis
Master of Science in Computer Science

Client-Server Applications in Java

by
Jasmine J. Ahuja

Advisor
Dr. Howard Blum

December 1997

CONTENTS

Client-Server Applications in Java

1. Introduction

During the first two decades of their existence, computer systems were highly centralized. A computer was usually placed within a large room and the information to be processed had to be taken to it. This had two major flaws, a) the concept of a single large computer doing all the work and b) the idea of users bringing work to the computer instead of bringing the computer to the user.

This was followed by 'stand alone PCs' where the complete application had to be loaded on to a single machine. Each user had his/her own copy of the software. The major problems were a) sharing information and b) redundancy.

These two concepts are now being balanced by a new concept called computer networks. In computer networking a large number of separate but interconnected computers work together. An application that requires two or more computers on the network is called a network application. The client–server model is a standard model for network applications. A server is a process that is waiting to be contacted by a client process so that the server can do something for it. A client is a process that sends a request to the server. The idea behind a network application is to offload some utilities like the Graphical User Interface (GUI) and in some cases, some processing, from the server to the client. As a result the GUI can be elaborate, thus making the application more user friendly. The other issues involved in client-servers are inter-process communication, security, maintenance and requirement for special software to run the client.

In a 'conventional' client–server model, a copy of the client process lies on every client machine and the client sends a request to the server through a socket to the port that the server is listening on. This model is popular as it offers improved graphical user interface. However, it has the following disadvantages: a) a copy of the client process has to be installed on every user's machine and b) any modification or update to the client process needs to be distributed and installed on every client machine.

In a new, modified view, there is no preinstalled client on the client machine. The client process is loaded onto the client machine each and every time the user initiates a request through his/her web browser using the server's URL/IP address. This 'web oriented view' does not require the user to have a special software to run the application. Moreover maintenance is much easier. The server's administrator can ensure that the users always download the latest version of the client.

In this paper, attempt has been made to study the basic issues and requirements for design and development of web oriented client-server applications and to examine the Java language in the context of these requirements. This paper especially concentrates on the inter-process communication and the user interface.

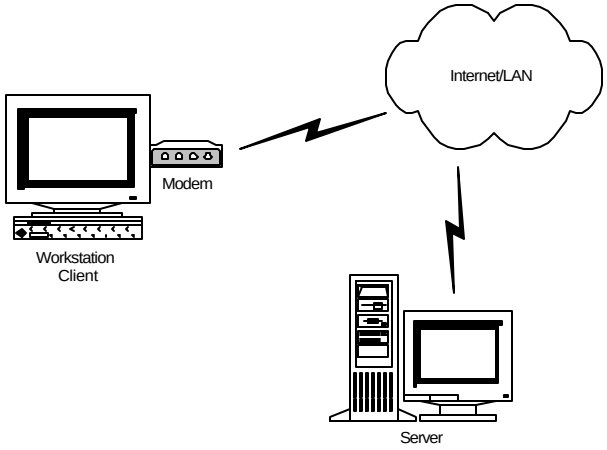
The paper starts with an overview of concepts commonly used in networking. It goes on to discuss communications using sockets in detail and three alternative Java based examples of client-server implementation. The first example is a conventional client-server in which the client as well as the server are applications. The second example is web based, where the client is the web browser and the server is an application and they use HTML and a Telnet applet to communicate. The third example is also web based, but uses a Java applet along with the HTML. The paper also discusses other methods of communication like Remote Method Invocation and CGI scripts in brief. Finally a discussion on applet security is followed by ideas for further studies in this field.

2. Overview of Concepts

2.1 Distributed Applications

The term *distributed applications,* is used for applications that require two or more autonomous computers or processes to cooperate in order to run them. Thus, the distributed system considered in this paper, involves three resources, processing, data and user interface. Both processing and data can be distributed over many computers. The user interface is usually local to the user so that the graphical interface, which consumes high bandwidth, does not have to be transmitted from one location to another.

A Typical Distributed Application Scenario



In distributed computing, the computer network is used to support the execution of program units, called processes, that cooperate with one another to work towards a common goal. This approach has become popular due to a number of developments like:

- ✓ Increase in the number of personal computers
- ✓ Low cost of establishing computer networks with the advancement of technology
- ✓ Computer manufacturers now offer networking software as a part of the basic operating system
- ✓ Computer networks are now an established way of disseminating information

2.2 The Java Language [22]

Java is a new programming language invented by Sun Microsystems. Sun's goal was to allow programmers to create one copy of a program that users could run on almost any computer and operating system. This capability was designed to make Java a vital component of programmability on the Web.

The Java Language (contd.)

Programs written in Java are translated into their own native architecture, in a format called bytecodes. In order to run a Java program, the user needs another program that can interpret the Java program and provide it with the environment and services it needs. The software layer, the Java Virtual Machine (JVM), makes just about any hardware and software platform look the same to the Java program. In effect, a JVM is a device driver for Java programs. Running a Java program with a JVM is currently slower than running an equivalent C program, but JVM technology is improving at a rapid pace.

Java does not support pointers to memory and so Java programs cannot corrupt a computer's memory. It also ensures all array and string references are within each item's bound. It performs automatic garbage collection, which frees up unused memory, so Java programs cannot leak resources.

Java's potential of platform-independent bytecodes and a language designed to go hand in hand with the Internet is enormous. However, it is still a new language and has not been tested on very large projects where applications can migrate over many platforms. It is only when its tools are tested thoroughly will businesses be willing to adopt it as 'the' programming language of the future.

2.3 World Wide Web [15]

The World Wide Web can be briefly described as a global, interactive, dynamic, cross-platform, distributed, graphical hypertext information system that runs over the Internet. The primary concept behind the World Wide Web is hypertext – i.e. instead of reading text in a rigid, linear structure (e.g. a book), the user can easily skip from one point to another using available links. The Web is not limited to any one kind of machine, it can be accessed using an application called a browser like Netscape's Navigator. It is successful in providing so much information because information is distributed globally across thousands of sites. The users do not have to install anything, they can go to a particular site to view information and when they are done their system reclaims the disk space. Information on the Web can be updated any time as it is contained on the site that published it and thus is accessible to the administrator of the site.

No single entity "owns" the World Wide Web. Given the enormous number of independent sites that supply information to the Web, it is impossible for any single organization to set rules or guidelines. There is however, a World Wide Web (W3) Consortium, based at MIT (USA) and INRIA (Europe). The Consortium is a congregation of individuals and organizations interested in supporting and defining the languages and protocols that make the Web (HTTP, HTML etc). The W3 Consortium also provides products (browsers, server and so on) that are freely available to anyone who wants to use them.

2.4 Client-Server

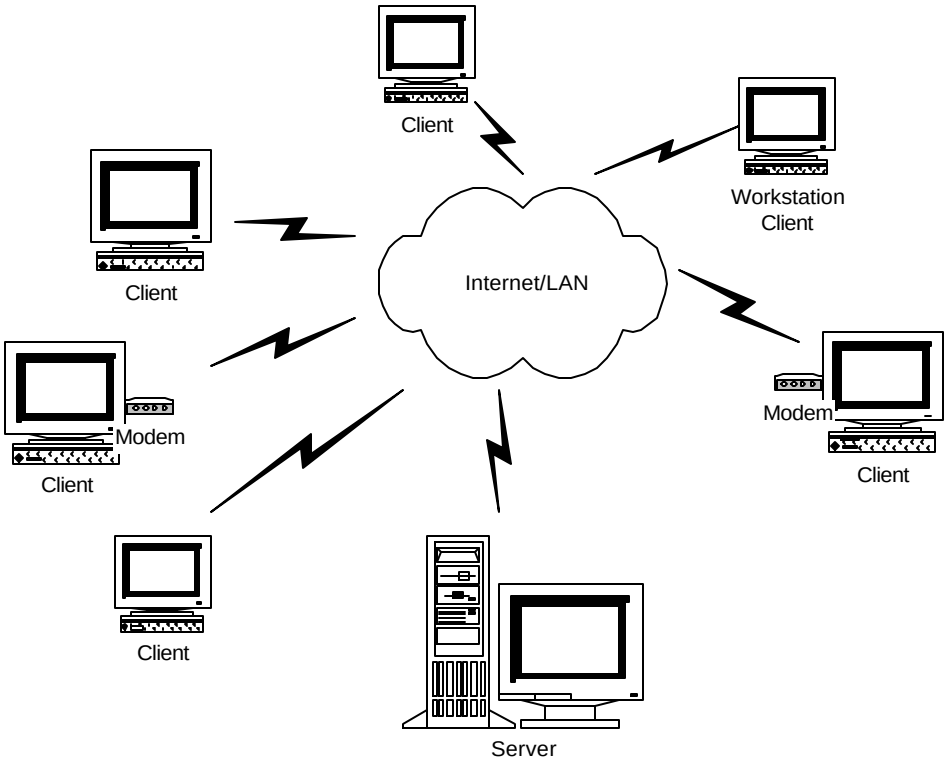
The client–server model is a standard model for network applications. A server is a process that is continuously running and waiting to be contacted by a client process. A client process initiates contact with the server by connecting to it at a specified port.

Client–Server (contd.)

The following is a typical scenario:

a) The server process is started on a computer system. It initializes itself and then waits for a client process to contact it with a service request.
b) The client process, usually started on another system, is connected to the server's system over a network. The client process sends a request across the network to the server requesting service of some form, e.g. reading or writing a file on the server's system.
c) When the server finishes processing the request, it waits for the next client request to arrive.

Client-Server Environment



The server can be divided into two types:

a) When a client's request can be handled by the server in a short amount of time, the server process handles the request itself. This sequential or non-concurrent server handles one client request at a time e.g. returning the time-of-day to the client.

b) When the amount of time to service a client depends on the client, e.g. a bank ATM where the user may take his/her own time to enter information or make a transaction, the server handles it in a concurrent fashion. Concurrent servers invoke another process (a child process) to handle each client request, so that the parent server process can go back to waiting for the next client request. This server requires an operating system that allows multiple processes to run at the same time.

In a majority of applications, the client handles most of the user interface but may also take over a part of the processing from the server in some cases.

3. Java vs. C++

Java is similar to C++ to some degree, which carries the benefit of it being familiar to many programmers. This section describes some features of Java relevant to the topic and points out where the language diverges from its ancestor C++.

Syntax
Java was designed to resemble C++, probably the most commonly used language today, in order to make the system more comprehensible.

Object-Oriented
Java is an object oriented programming language i.e. focus is on data in the application and methods that manipulate that data, rather than strictly on procedures. The object-oriented facilities of Java are essentially those of C++, with extensions from Objective C for more dynamic method resolution.

Network-Oriented
Java's ***net*** class, which is a part of the java package, makes networking easier in Java than in C++. Sample code for setting up a simple server to listen on a port in C++ and Java is shown below:

C++ ->

```
#include <socket.h>

main()    /* a non-concurrent stream server */
{  int sockfd, newsockfd, clilen;
 struct sockaddr_in cli_addr, serv_addr;
  if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0)
        error("server: cannot open socket");
   bzero( (char *)&serv_addr, sizeof(serv_addr));
   serv_addr.sin_family = AF_INET;
   serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
   serv_addr.sin_port =htons(SERV_TCP_PORT);
   if(bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
        error( "server: cannot bind address");
   listen(sockfd, 5);
   for(;;) {
     clilen = sizeof(cli_addr);
     newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr, &clilen);
     if (newsockfd < 0)
        error("server: cannot accept connection");
     str_echo(newsockfd);     /* serve a client */
     close(newsockfd);
   }
}
```

Java->

```
import java.net.*;

public static void main(String[] args)
```

Java vs. C++ (contd.)

```
{       public void runServer()
        {   ServerSocket server;
            Socket connection;
            try {
                server = new ServerSocket( port, number of users);
                for(;;)
                {  connection = server.accept();
                   display.setText( "Connection received...\n" );
                                                    // Serve the client
                   display.appendText("\nClosing socket.\n" );
                   connection.close();
                }
              }catch ( IOException e ) { e.printStackTrace(); }
         }
}
```

To be fair, this C++ code uses a larger number of parameters and statements because it is more general, allowing selection of a number of protocol suites, TCP/IP being one. Furthermore, within the TCP/IP suite either connection oriented TCP or connectionless UDP may be selected. The Java library however, has separate classes specializing in UDP (e.g. DatagramSocket, DatagramPacket) and TCP (e.g. ServerSocket, Socket).

Secure

Java is intended for use in networked/distributed environments. Towards that end, a lot of emphasis has been placed on security. Java enables the construction of virus-free, tamper-free systems. The authentication techniques are based on public-key encryption. Changes to the semantics of pointers make it impossible for applications to forge access to data structures or to access private data in objects that they do not have access to. This closes the door on most activities of viruses.

Portable

Being architecture neutral plays a major role in being portable. Unlike C and C++, there are no "implementation dependent" aspects of the specification. The sizes of the primitive data types are specified, as is the behavior of arithmetic on them. For example, "int" always means a signed two's complement 32 bit integer. Making these choices is feasible in this day and age because essentially all CPUs share these characteristics. The Java system itself is quite portable. The compiler is written in Java and the runtime is written in ANSI C with a clean portability boundary. The portability boundary is essentially a POSIX subset.

Interpreted

Unlike the C++ compiler, the Java compiler generates bytecodes which are translated at runtime to native machine instructions (interpreted). The Java interpreter can execute Java bytecodes directly on any machine to which the interpreter and run-time system have been ported.

Transmitting Structures (Object Serialization) [4, page 483][13, page 39]

Object serialization, a new feature in Sun's Java Development Kit version 1.1 (JDK1.1), is the ability to write the complete state of an object to an output stream, and then recreate that object at some later time by reading its serialized state from an input stream. Object serialization is of interest because it is common to read and write complex data in networking. An object can be serialized by passing it to the *writeObject()* method of the ObjectOutputStream class. Similarly, an object can be created from a serialized object

Java vs. C++ (contd.)

stream by calling the *readObject()* method of the ObjectInputStream class and casting it to its correct type. Both these object stream types are part of the java.io package. Code required for sending an object from the client to a concurrent server is as follows:

In the Client

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

public class AtmApplet extends Applet
{    ---
     initialization, networking etc.
      ---
     int dacct = 111;                    // hard coded data values for this example
     int dpin = 1111;
     double dbal = 0.0;
     DataObject dob = new DataObject(dacct, dpin, dbal);

     ObjectOutputStream outter;
     ObjectInputStream inner;
     try {
              outter = new ObjectOutputStream(client.getOutputStream());
              inner = new ObjectInputStream(client.getInputStream());

            // sending data to Server
               outter.writeObject(dob);
               client.close();
           } catch ( IOException e ) { System.out.println(e);}
 }

class DataObject implements Serializable
{ public DataObject(int a, int p, double b)
  {   dacct = a;
      dpin = p;
      dbal = b;
  }

   private int dacct, dpin;
   private double dbal;
}
```

The server

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

class Server
{  public static void main(String[] args)
   {   ---
       NET INFORMATION
    }
}
class ServerHandler extends Thread
{      ObjectOutputStream outter;
```

```
    ObjectInputStream inner;
```

Java vs. C++ (contd.)

```
    public void run()
    { try
      { outter = new ObjectOutputStream(incoming_socket.getOutputStream());
        inner = new ObjectInputStream(incoming_socket.getInputStream());
       DataObject sdo = (DataObject)inner.readObject();
      } catch {catch(Exception e) {System.out.println(e);}}
    }
}

class DataObject imp lements Serializable
{ public DataObject(int a, int p, double b)
  {   dacct = a;
      dpin = p;
      dbal = b;
  }
   private int dacct, dpin;
   private double dbal;
}
```

In C++, objects can be input and output using operator overloading. For object input, the stream-extraction operator >> is overloaded for the appropriate istream classes and for object output, the stream-extraction operator << is overloaded for the appropriate ostream classes. In each case, only the object's data members are input or output in a form meaningful for objects of that particular abstract data type. When object data members are output, the object's type information is lost in a way. However, if the program that is going to read this data knows what object type it corresponds to, then the data is simply read into objects of that type.

Some other features of C++ and Java are described in the table below.

Feature	Java	C++
Program exit value	Main() must be declared a void i.e. a value cannot be returned with a return statement. The only way to do so is to call System.exit()	Main() can return a value
Global variables	Are not allowed-every field or method is associated with classes and referred to by its fully qualified name. This care is taken to avoid conflicts	Are allowed
Filenames	Source file should have the same name as the public class in it	No such rule
Comments	Three kinds i.e. /* */ block comment // single line comment /** */ java doc comment The java doc comment is specially processed by the javadoc program to produce simple online documentation	Two kinds /* */ block comment // single line comment

Java vs. C++ (contd.)

Feature	Java	C++
Header files	Does not have header files or function prototypes. A single Java™ object file serves as the interface definition and implementation for a class	Declaration of member data and functions can be made in separate files
Primitive data types	- Has specific byte and boolean types apart from the standard C types. - All variables have guaranteed default values i.e. boolean = false, byte = 0, int = 0 - Boolean values are not integers - All integral types are signed	- Does not have byte and boolean types - An un-initialized local variable usually has garbage as its value - Boolean values are integers - Integral types may be signed or unsigned
Creating Objects	Declaring a variable to hold an object does not create an object, objects are created using the *new* Keyword	C++ uses the *new* keyword to create objects dynamically
Accessing Objects	Fields in an object are accessed by using the dot notation	Fields are accessed using the dot notation as well as -> or pointer notation (when pointer are used)
Garbage collection	There is no delete() or free() in Java - use is made of a technique called garbage collection to automatically detect objects or arrays no longer being used and to free them	Use of *delete* made to de-reference variables or objects created dynamically i.e. created using the *new* keyword
Strings	Strings are instances of the *java.lang.String* class. They are treated as primitive types – i.e. the String object is automatically created when a double quoted constant is seen. An operator + operates on String for concatenation	Strings are null terminated character arrays
Synchronized statement	Java provides the synchronized statement to protect multiple threads from modifying objects simultaneously. Methods and objects can also be synchronized	Does not provide for a synchronized statement.
Exception Handling	Java requires that any method that can cause an exception(error) to occur must either catch it using the try/catch/finally statement or specify the type of exception with a throws clause in the method declaration	C++ has exception handling but it is not mandatory
Local variable declaration	Forward referencing allowed	Forward referencing not allowed

4. Communication between Client and Server using sockets

Sockets were an innovation of Berkeley UNIX, version 4.3 BSD [2, page 11]. They allow the programmer to treat a network connection as another stream that bytes can be written onto or read from. Historically, sockets are an extension of one of UNIX's ideas: that all I/O should look like file I/O to the programmers, whether they are working with a keyboard, a graphic display, a regular file, or a network connection. Sockets shield the programmer from low-level details of the network, like media types, packet sizes, packet retransmission, network addresses and more. A socket can perform the following basic operations:

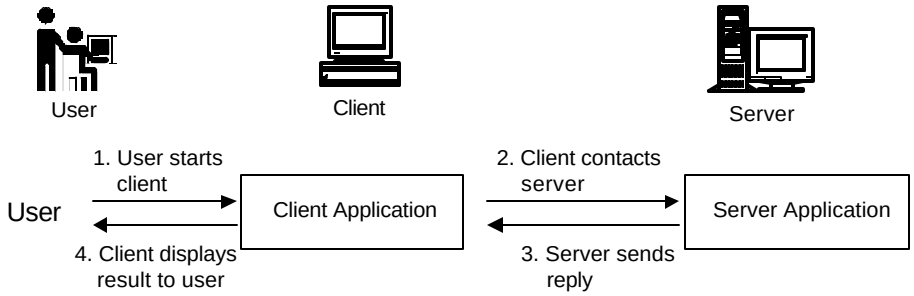
- Connect to a remote machine
- Send data
- Receive data
- Close a connection
- Bind to a port
- Listen for incoming data
- Accept connections from remote machines on the bound port

Assuming that we are using TCP for a connection oriented implementation, Java has two related classes that are required i.e. the *Socket* class and the *ServerSocket* class. The *socket* class is used by both the client and the server and has methods that correspond to the first four of these operations. The last three are only needed by the server and are implemented by the *ServerSocket* class. [6, page 149]

Using sockets, communication can be established between the client and server in the following ways: In all the cases mentioned below, the server can make use of a database to retrieve or save data.

4.1 Server application – Client application (Conventional method)

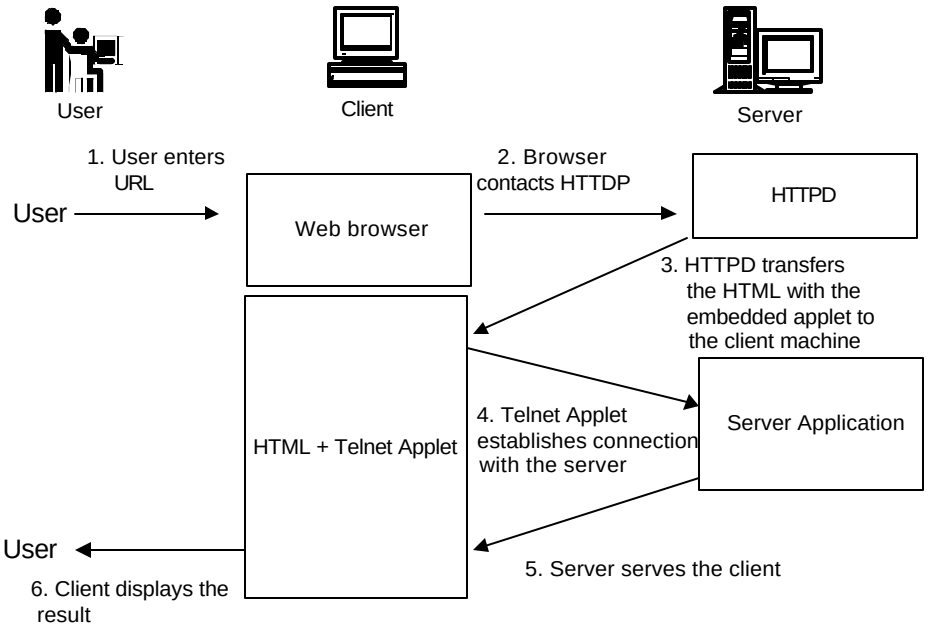
In this method, both the server and the client is a Java application. The client application has to be loaded onto the client's machine. The server can be concurrent i.e. serve more than one client at a time. The server and database lie on the server machine. The major disadvantage of this method, as already stated earlier, is that the software for the client has to be physically present on the client machine and the client machine has to have the Java interpreter to run it. If any changes are made to the client application, it has to be redistributed to the clients.



A sample application using this method is attached (Appendix A)

4.2 Server application – Client web browser (using HTML and telnet)

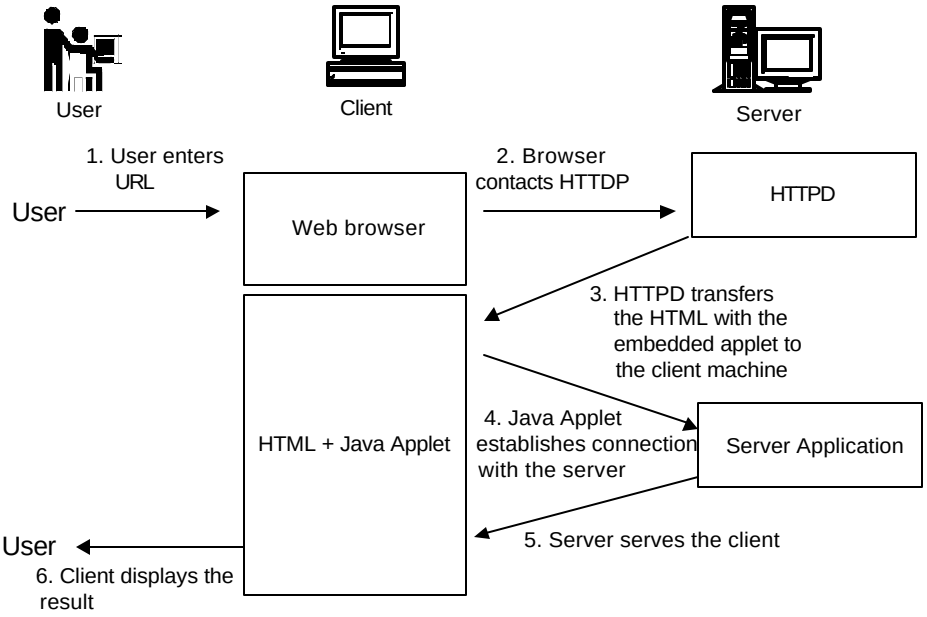
In this method, the server is a Java application and the client is a web browser like Netscape navigator. The user enters the URL/IP address of the server and is connected to HTTP daemon (HTTPD) web server on the server machine that loads the HTML file onto the client's machine. This file provides part of the GUI and connects to the concurrent Java server using Java's telnet applet. The advantage here is that the client's machine only needs a web browser to be able to connect to the server. Moreover, any changes to the client can be made easily as the actual program lies on the server machine itself and is easily accessible by the administrator.



A sample application is attached (Appendix B)

4.3 Server application – Client web browser (using HTML and Java applet)

In this method the server is a Java application and the client is a Web browser like Netscape Navigator. The user enters the URL/IP address of the concurrent server and the HTTPD web browser loads the HTML file that has the Java applet embedded in it. The HTML file here is just a vehicle to transport the applet. The applet provides the complete GUI and opens a TCP/IP socket connection to the server on a designated port. The server may also be connected to a database (use of JDBC/ODBC bridge was made when implementing this method, this technique can also be used in the other methods described earlier)



Also JDBC allows use of any ODBC database by the server i.e. anyone providing the server service can use their existing database if it is ODBC complaint.

The advantage here is that the GUI is more user friendly. The applet allows restricted and relevant use only, an important security issue.

A sample application is attached (Appendix C)

5. Sample Applications

The following applications describe the different methods in which communication can be established between a client and server. These three versions follow the same basic concept, connection is established between the client and server using sockets and information is passed to and from them based on logic of the program. In all three versions, the server is a Java Application.

Version 1 Client Java Application

In this simple program the client application uses the features of the *java.net* package to establishes a TCP/IP connection using sockets on a specified port with the server. The client and server open input and output streams over the socket using Java's *java.io* package. The client sends a message to the server and the server acknowledges the message and closes the socket. (See Appendix A for program code.)

Version 2 Client HTML file, telnet applet and text database file

This application involves an HTML program i.e. the bankatm.html. The user downloads the html file on his/her machine by using a web browser and giving the IP address of the machine where the concurrent server is running. The html establishes a TCP/IP connection with the server on a specified port using the telnet applet. The html file then sends requests like balance inquiry, request for deposit/withdrawal to the server using the output stream (*java.io* package). The server receives the request through its input stream, processes it and displays the result using its output stream. Once the user is finished he/she breaks connection by disconnecting. The server is also connected to a database file (a flat text file in this case). (See Appendix B for program code)

Version 3 Client HTML file, Java Applet and MS Access database file (using JDBC)

This application involves a Java ATM applet and an HTML file. The user downloads the HTML file which has the java applet embedded in it, on his/her machine using a web browser and giving the IP address of the machine where the concurrent server is running. The applet starts executing and establishes TCP/IP socket connection with the server on a specified port using the *java.net* package. The applet then sends requests like balance inquiry to the server using the output stream (*Java.io* package). The server receives the request through its input stream, processes it and sends the result back to the client using its output stream. Once the user is finished, the socket connection between the client and server is closed. In this version, the server is connected to a database file (an MS Access file in this case) using the JDBC/ODBC driver. (See Appendix C for program code)

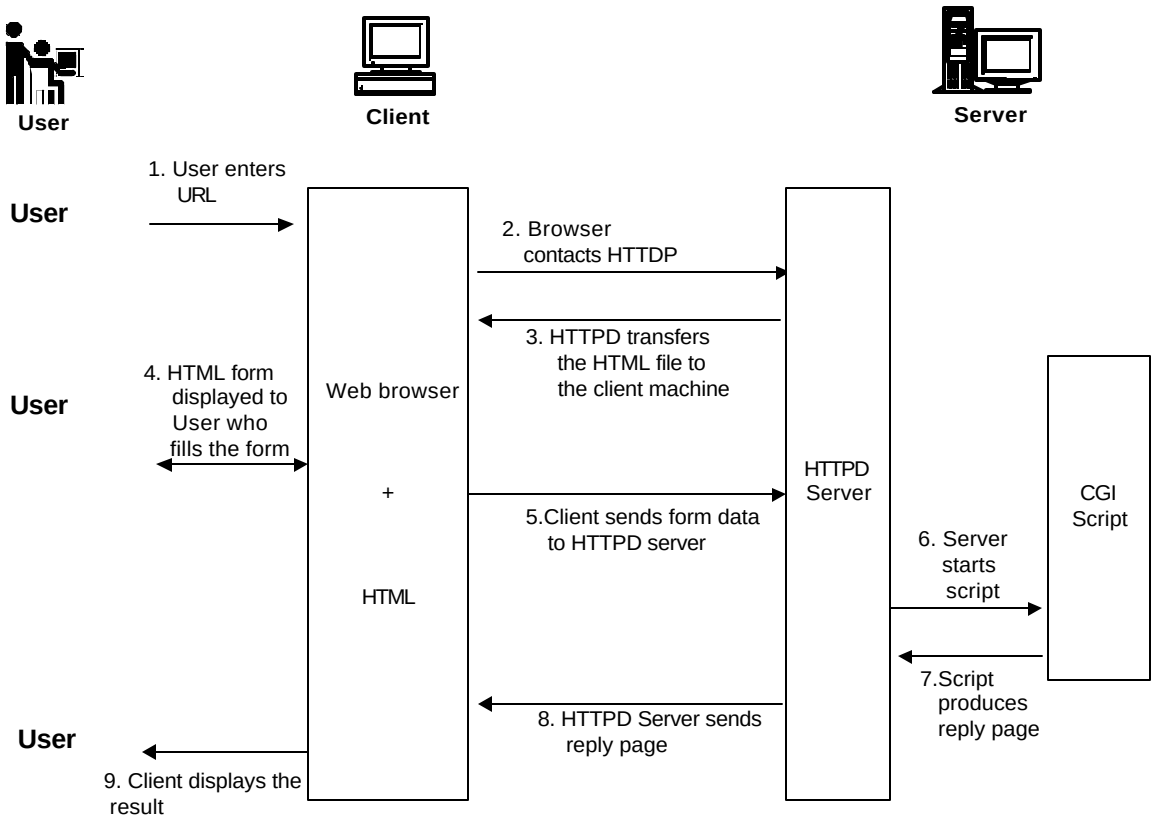
6. Some other methods of Communication between Client and Server

6.1 CGI Scripts [4, page 573]

A CGI script is a simple protocol that can be used to communicate between HTML forms and a program. A CGI script can be written in any language that can read STDIN, write to STDOUT, and read environment variables, i.e. virtually any programming language, including C, Perl, or even shell scripting.

In CGI scripting, a user fills out an HTML form and sends the text in the text fields and the setting of the check boxes and radio buttons to a CGI script on the server by clicking on the Submit button. CGI scripts are programs that reside on the server computer. Usually, there are many CGI scripts on a server. The HTML tag of the Submit button specifies the script to use to process that particular form. The HTTPD on the server launches the CGI script and feeds it the form data. The CGI script processes the form data and sends another HTML page back to the web browser who then displays the response page. CGI can be written in Java. This script is an application though and not an applet and will have to be launched by the HTTPD web server whenever a client submits a query naming that script as the processing agent.

Data Flow during execution of a CGI script [4]

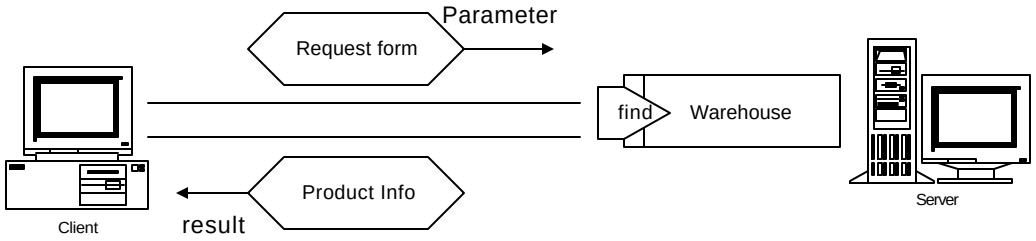


6.2 Remote Method Invocation [4, page 645][6, page 347]

The central concept in the Java remote object implementation is the remote method invocation or RMI. Code on the client computer invokes a method on an object on the server. To do so, it actually calls a regular Java method that is encapsulated in a surrogate object called a stub that resides on the client only. The stub takes the parameters used in the remote method and packages them up as a block of bytes. This packaging uses a device-independent encoding for each parameter e.g. numbers are always sent in big-endian format. The process of encoding the parameters into a format that is suitable for transporting them across the net is called *parameter marshalling*. The stub method builds an information block and then sends this information to the server. On the server side, there is a skeleton object that makes sense out of the information contained in the packet and passes that information to the actual object executing the remote method. It then captures the return value or exception of the call on the method, marshals that value and sends it back to the stub. The stub *unmarshals* the return value or exception from the server. This becomes the return value of the remote method call. Or, if the remote method threw an exception, the stub re-throws it in the process space of the caller. This process is largely transparent for the programmer. Also, remote objects look and feel the same as local objects. The syntax for a remote call is the same as for a local call.

Thus, for example, a client seeking product information can query a warehouse object on the server. It calls a remote method *find* which has one parameter, the request form object. The *find* method returns an object to the client, the product information object. This can be depicted as follows:

Invoking a remote method on a server object [4]



The purpose behind RMI is to make all the implementation details transparent to the programmer so that remote objects and methods work just like the local objects and methods.

7. Miscellaneous Concerns

Applet Security

Web browsers like Netscape allow applets to read and write data only on the host that serves the applet [4, pg. 589] i.e. it will let the applet connect only to sockets on the computer from which the applet came. The write restriction prevents applets from planting viruses or altering important files on the local computer. The read restriction does not allow applets to browse files on the local computer for sensitive information, such as credit card numbers, open a socket connection to the applet host, and write the information back.

8. Recommendation for future Studies.

1. Implementing a client server using RMI. Communication between the client and server using RMI is relatively new but is fast gaining popularity as an alternative to sockets. A study could be done to explore this concept and its advantages over the sockets.

2. Security issues with applets: When using the web, security is a big issue. A user may open a web page and interact with an applet that does something useful and be completely unaware of what that applet does in other threads. It would be interesting to study how web browsers restrict applets in terms of reading and writing only to/from the originating host. References for further study in applet security can include http://java.sun.com/sfaq/, http://www.javasoft.com and Core Java™ 2nd and 3rd Editions.

9. Bibliography

1) Laura Lemay
Web Publishing with HTML, Sams net, 1995

2) W. Richard Stevens
UNIX Network Programming, Prentice Hall, 1990

3) David Flanagan
Java in a Nutshell, Second Edition O'Reilly and Associates, Inc, 1997

4) Gary Cornell, Cay S. Horstmann
Core Java, Second Edition, Prentice Hall, 1997

5) Gary Cornell, Cay S. Horstmann
Core Java, Third Edition, Volume 1, Prentice Hall, 1997

6) Elliotte Rusty Harold
Java Network Programming, O'Reilly and Associates, Inc, 1997

7) Laura Lemay. Charles L Perkins
Teach yourself Java in 21 days, Sams net, 1996

8) Robert Orfali, Dan Harkey, Jeri Edwards
The Essential Client/Server Survival Guide, Second Edition, Wiley Computer Publishing, 1996

9) Windows Magazine, Nov 1997

10) H.M Deitel, P.J. Deitel
Java, How to Program, Prentice Hall, 1997

11) Ken Arnold & James Gosling
The Java programming Language, Addison Wesley, 1996

12) Stephen R. Davis
Learn Java Now, Microsoft Press, 1996

13) Prashant Sridharan
Advanced Java Networking, Prentice Hall, 1997

14) H.M. Deitel , P.J. Deitel
C++ How to Program, Prentice Hall, 1994

15) http://www.w3.org/. w3 consortium's home address

16) http://java.sun.com Java documentation

17) http://www.gamelan.com Java program examples

18) http://www.javauniverse.com JDK tutorials

19) http://lulasrv.lulala.org/internet/cgi_tests	CGI script
20) http://www.javasoft.com	Java Tutorial
21) http://sol.pace.edu/java	Java examples
22) http://java.sun.com/docs/white/langenv	Java Language

Appendix

Client Version 1

```
// Set up a Client that will read information sent
// from a Server and display the information.
import java.io.*;
import java.net.*;
import java.awt.*;

public class Client extends Frame {
  TextArea display;

  public Client()
{
    super( "Client" );
    display = new TextArea( 20, 10 );
    add( "Center", display );
    resize( 300, 150 );
    show();
  }

  public void runClient()
  {
    Socket client;
    OutputStream output;
    InputStream input;

    try {
      client = new Socket( “172.17.52.149”, 5000 );          // IP address and Port Number
      display.appendText( "Created Socket\n" );

      output = client.getOutputStream();
      input = client.getInputStream();
      display.appendText( "Sending Data .. Hello World\n" );
      String s = new String("Hello World \n");

      for (int i = 0; i < s.length(); i++)
            output.write( (int) s.charAt(i));


     display.appendText("\n Received from Server:\n \t");
     char c;

      while((c = (char) input.read() ) != '\n')
            display.appendText(String.valueOf(c));

      display.appendText( "\nTransmission Complete\n" );
      client.close();
    }
    catch ( IOException e ) {
      e.printStackTrace();
    }
  }
```

```
  public boolean handleEvent( Event e )
  {
     if ( e.id == Event.WINDOW_DESTROY ) {
       hide();
       dispose();
       System.exit( 0 );
     }

     return super.handleEvent( e );
  }

  public static void main( String args[] )
  {
     Client c = new Client();

     c.runClient();
  }
}
```

Server Version 1

```
// Set up a Server that will receive a connection
// from a client, send a string to the client,
// and close the connection.
import java.io.*;
import java.net.*;
import java.awt.*;

public class Server extends Frame {
 TextArea display;

 public Server()
 {  super( "Server" );
    display = new TextArea( 20, 5 );
    add( "Center", display );
    resize( 300, 150 );
    show();
 }

 public void runServer()
 {  ServerSocket server;
    Socket connection;
    PrintStream output;
    DataInputStream input;

    try {
      server = new ServerSocket( 5000, 100 );              // Open ServerSocket
      connection = server.accept();                        // Wait for connection
      display.setText( "Connection received...\n" );
      output = new PrintStream(connection.getOutputStream());    // Set up IO streams
      input = new DataInputStream(connection.getInputStream());
      String serv = new String( "Connection successful\n\n");
      String c;
      System.out.println("steps 1");
      c = input.readLine();
      System.out.println(c);
      display.appendText(c);
      System.out.println("steps 1");
      display.appendText("\n Receiving data Hello!\n\n");
      // The next two lines receive data from client
      System.out.println("steps 1");
      for(int i = 0; i < serv.length(); i++) {
        output.write( (int) serv.charAt( i ) ); display.appendText("\n Sent data.\n");
      }
      System.out.println("steps 3");
      display.appendText("\nClosing socket.\n" );
      connection.close();
    }
    catch ( IOException e ) {

      e.printStackTrace();
    }
 }
```

```
  public boolean handleEvent( Event e )
  {    if ( e.id == Event.WINDOW_DESTROY ) {
       hide();
       dispose();
       System.exit( 0 );
  }
    return super.handleEvent( e );
  }

  public static void main( String args[] )
  {
    Server s = new Server();

    s.runServer();
  }

}
```

Client Version 2

```
<HTML>
<HEAD>
<TITLE>BANK ATM</TITLE>
</HEAD>

<BODY Background="" BGColor=#0000ff Text=#ffffff Link=#0000ff VLink=#400080 ALink=#80ffff>
<center>
<h2>BOHIEM BANK ATM 97'</h2>
<br>

<APPLET CODE="appWrapper.class" align = right WIDTH=100% HEIGHT=280>
  <PARAM NAME=applet  VALUE= "telnet">

  <!-- applet initialization: address and port -->
// Sending the IP address and port number of the server to the telnet applet as a parameter
  <PARAM NAME=address VALUE=172.17.52.149>
 <PARAM NAME=port VALUE="8888">
 <PARAM NAME=font VALUE="Courier">
  <PARAM NAME=fontsize VALUE="28">
  // Creating the GUI
  <!-- modules: #1 is a buttonbar--->
  <PARAM NAME=module#1   VALUE="ButtonBar">
  <PARAM NAME=1#Button    VALUE="connect|\$connect()">
  <PARAM NAME=2#Button    VALUE="disconnect|\$disconnect()">
  <PARAM NAME=3#Button    VALUE="Detach/Delete Window|\$detach()">
  <PARAM NAME=4#Button    VALUE="SUBMIT:|\@send@\r\n">
  <PARAM NAME=5#Input     VALUE="send#20|testing">

  <!---- NEW BUTTONS FOR CODE  OF BANK ATM 97 ------>
  <PARAM NAME=6#Button    VALUE="DEPOSIT:|\D\r\n">
  <PARAM NAME=7#Button    VALUE="WITHDRAW:|\W\r\n">
  <PARAM NAME=8#Button    VALUE="BALANCE:|\B\r\n">
<!--<PARAM NAME=9#Button    VALUE="VERIFY:|\V\r\n"> -->


  <!-- modules: #3 is a scripting module -->
  <PARAM NAME=module#3    VALUE="Script">
  <PARAM NAME=script     VALUE="login:|ahuja">

        <!-- make sure, non-java-capable browser get a message: -->
  <B>
        Your Browser seems to have no <A HREF="http://java.sun.com/">Java</A>
        support. Please get a new browser or enable Java to see this applet!
        </B>
</APPLET>
<!-- End of applet code -->
<table width=100% bgcolor="aaaaab"><tr><td><BR></td></tr></table>
<!-- <form action="http://172.17.52.149:8888" method="post"> -->
<!-- <input type="text" size=18><input type="submit" value="Bank account number"> -->
 </form>

</BODY>

</HTML>
```

Server Version 2

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

class ServerHandler extends Thread        //  Enabling threading
{
  Socket incoming;
  int counter;

  ServerHandler (Socket i, int c)
   { incoming = i; counter = c; }

  public void run()
  { try
    { DataInputStream in = new DataInputStream(incoming.getInputStream());    // Creating IO Streams
      PrintStream out = new PrintStream(incoming.getOutputStream());
      int prod;
      int check = 0;
      String mgrpass = new String("admin");
      int done = 0;
      while (done == 0)
      { out.println( "Connected to BOHIEM Bank..\r" );
        out.println("    ----------------------------------------\r");
        out.println("    |             WELCOME              |\r");
        out.println("    ----------------------------------------\r");
        out.println("    \r\r\r");
//        if ((str.compareTo("Br")) || (str.compareTo("Dr")) || (str.compareTo("Wr")))
//        {
          out.println(" \rEnter Account No.: \r");
          String AcctNo = in.readLine();
//        }
        done = 2;
        while (done == 2)
        {  out.println(" \rEnter Password: \r");
           String Password = in.readLine();
             done = 1;
             while(done == 1)
             { out.println
               (" \r\nUsing your mouse click on the transaction type on the top right of the screen\r");
             String str = in.readLine();
             check = 0;
             //********************* BALANCE **********************
             if (( str.compareTo("Br")) == 0 )
             {  out.println("Looking For " + AcctNo + "\r");
               String SEARCHFOUND = new String(searchInfo(AcctNo, Password));
               if (SEARCHFOUND.compareTo("WRONG PASSWORD")== 0)
               { out.println(SEARCHFOUND + "...ENTER AGAIN\r\n");
                 done = 2;
               }
               else if (SEARCHFOUND.compareTo("ACCOUNT NOT FOUND...") == 0)
               { out.println(SEARCHFOUND + "\r\n");
                 done = 0;
               }
               else
                 out.println("Your balance is $" + SEARCHFOUND + "\r\n");
```

```
}
//********************** DEPOSIT ********************
if (( str.compareTo("Dr")) == 0 )
{ out.println(" Enter Deposit Amount: $\r");
  String Depamt = in.readLine();
  double dep = new Double(Depamt).doubleValue();
  out.println("Looking For " + AcctNo + "\r");
  String Acctbal = new String(searchInfo(AcctNo, Password));
   if (Acctbal.compareTo("WRONG PASSWORD")== 0)
  {  out.println(Acctbal + "...ENTER AGAIN\r\n");
     done = 2;
  }
  else if (Acctbal.compareTo("ACCOUNT NOT FOUND...") == 0)
  {  out.println(Acctbal + "\r\n");
     done = 0;
  }
  else
  {  double bal = new Double(Acctbal).doubleValue();
     bal += dep;
     String newbal = String.valueOf(bal);
     String Result = new String(replBal(AcctNo, Password, newbal));
     out.println(Result + "\r\n" + "Deposited $" + Depamt + "\r\n");
  }
}
//************************ WITHDRAWAL *********************
if (( str.compareTo("Wr")) == 0 )
{ out.println(" Enter Amount to Withdraw : $\r");
   String Withdamt = in.readLine();
  double withd = new Double(Withdamt).doubleValue();
  out.println("Looking For " + AcctNo + "\r");
  String Acctbal = new String(searchInfo(AcctNo, Password));
   if (Acctbal.compareTo("WRONG PASSWORD")== 0)
  {  out.println(Acctbal + "...ENTER AGAIN\r\n");
     done = 2;
  }
  else if(Acctbal.compareTo("ACCOUNT NOT FOUND...") == 0)
  {  out.println(Acctbal + "\r\n");
     done = 0;
  }
  else
  {  double bal = new Double(Acctbal).doubleValue();
     bal -= withd;
     if (bal > 0.00)
      {  String newbal = String.valueOf(bal);
        String Result = new String(replBal(AcctNo, Password, newbal));
        out.println(Result + "\r\n" + "Withdrew $" + Withdamt + "\r\n");
      }
     else
      { out.println("Insufficient Funds \r\n");
      }
  }
}
//************************ ADD NEW ACCOUNTS ***************
if (( str.compareTo("Ar")) == 0)
{  if(Password.compareTo(mgrpass) == 0)
   {  out.println("Please Enter the Following Data: \r");
```

```
            out.println("Enter Account No. \r");
             String ADDacctnum = in.readLine();
            out.println("Enter password (3 to 7 letters only)\r");
             String ADDpass = in.readLine();
             out.println("Enter Opening Balance  \r");
             String ADDbalance = in.readLine();
             String ADDeveryTHING = new String(ADDacctnum + "\r\n" + ADDpass +
             "\r\n" + ADDbalance + "\r\n");
             out.println(ADDeveryTHING + "\rAccount is added to database\r\n");
             setInfo(ADDeveryTHING);
           }
           else
             out.println("WRONG PASSWORD..\r\n");
         }
         //********************* SEARCH EXISTING ACCOUNTS **************
         if (( str.compareTo("Sr")) == 0 )
         { if(Password.compareTo(mgrpass) == 0)
           { out.println(" \rEnter Account No. to Search For: \r");
             String SearchString = in.readLine();
             out.println("Looking For " + SearchString + "\r");
             String SFOUND = new String(searchMgrInfo(SearchString));
             out.println(SFOUND + "\r\n");
           }
           else
             out.println("WRONG PASSWORD..\r\n");
         }
         //********************* PRINT EXISTING ACCOUNTS **************
         if (( str.compareTo("Pr")) == 0 )
         { if(Password.compareTo(mgrpass) == 0)
           { out.println("\r");
             printAccts(out);
           }
           else
             out.println("WRONG PASSWORD..\r\n");
         }

       }// end of while done == 1
     }// end of while done == 2
   }// end of while done == 0
  } // end of try
  catch (Exception e)
  {  System.out.println( "" );
  }
}// end of main

//******************************** METHODS ********************************
//******************** SEARCH (ATM METHOD)
   public static String searchInfo(String a, String p)
{       int match=0;
        int x = 0;
       try
        {    RandomAccessFile in =                                  // Opening a text file in Read-Write Mode
                new RandomAccessFile("account.dat", "rw") ;

             while (in.getFilePointer() < in.length())
             {     String accttemp = in.readLine();
```

```
                int templen = accttemp.length();
                String acct = new String
                                (accttemp.substring(0,templen-1));
                String accounts = new String("\rAccount : "+acct);
                String passwords = new String("");
                String balances = new String("");
                if ( acct.compareTo(a) == 0 )
                {     String passtemp = in.readLine();
                     int passlen = passtemp.length();
                     String pass = new String(passtemp.substring(0,passlen-1));
                     passwords = ("\rPassword : "+ pass);
                      if ( pass.compareTo(p) == 0 )
                      {   match = 1;
                         String balancetemp = in.readLine();
                          int ballen = balancetemp.length();
                          String balance = new String(balancetemp.substring(0,ballen-1));
                         balances = balance;
                      }
                      else
                      {   return("WRONG PASSWORD");
                      }
                }

                if (match >= 1)
                {     // RETURNS ALL INFO IN FORM OF A STRING
                    String info = new String(balances + "\n");
                    return info;
                } // END OF IF MATCH <<<----------
                else
                {   // DON't DO ANYTHING
                }   // end of else return search string NOT FOUND
            } //  <<<------- end of while inFile Pointer------->>>

             in.close();
             return ("ACCOUNT NOT FOUND...");
          } catch(Exception e)
            {return("error reading from file\n\r");}
}

//****************** DEPOSIT/WITHDRAWAL (ATM METHOD)
public static String replBal(String a, String p, String b)
{       int match=0;
        int x = 0;
       try
        {    RandomAccessFile in =
                 new RandomAccessFile("account.dat", "rw") ;

             while (in.getFilePointer() < in.length())
             {     String accttemp = in.readLine();
                   int templen = accttemp.length();
                   String acct = new String
                                 (accttemp.substring(0,templen-1));
                   String accounts = new String("\rAccount : "+acct);
                   String passwords = new String("");
                   String balances = new String("");
                   if ( acct.compareTo(a) == 0 )
```

```
                {     String passtemp = in.readLine();
                     int passlen = passtemp.length();
                     String pass = new String(passtemp.substring(0,passlen-1));
                     passwords = ("\rPassword : "+ pass);
                      if ( pass.compareTo(p) == 0 )
                      {   match = 1;
                         in.writeBytes(b);
                     }
                     else
                      {   return("Wrong password");
                      }
                }

                if (match >= 1)
                {     // RETURNS ALL INFO IN FORM OF A STRING
                    String info = new String("Transaction Complete");
                    return info;
                 } // END OF IF MATCH <<<----------
                else
                {   // DON't DO ANYTHING
                }   // end of else return search string NOT FOUND
             } //  <<<------- end of while inFile Pointer------->>>

              in.close();
              return ("Account NOT FOUND...\r\n");
           } catch(Exception e)
             {return("error reading from file\n\r");}
}


//*********************** ADD NEW RECORDS (MANAGER METHOD)
public static void setInfo(String s)
{  int x = 0;
   try
   { RandomAccessFile in = new RandomAccessFile("account.dat", "rw");
     while (in.getFilePointer() < in.length() )
     { String cardtemp = in.readLine();
     }
     in.writeBytes(s);                                          // Writing to the file
     in.close();
   }catch(Exception e){;}
}

//****************************** SEARCH (MANAGER METHOD)
public static String searchMgrInfo(String a)
{       int match=0;
        int x = 0;
       try
        { RandomAccessFile in = new RandomAccessFile("account.dat", "rw");
          while (in.getFilePointer() < in.length())
          {     String accttemp = in.readLine();
               int templen = accttemp.length();
               String acct = new String (accttemp.substring(0,templen-1));
               String accounts = new String("\rACCOUNT : "+acct);
               String passwords = new String("");
               String balances = new String("");
```

```
                if ( acct.compareTo(a) == 0 )
                {     match = 1;
                      String passtemp = in.readLine();
                      int passlen = passtemp.length();
                      String pass = new String(passtemp.substring(0,passlen-1));
                      passwords = ("\rPASSWORD : "+ pass);
                      String balancetemp = in.readLine();
                      int ballen = balancetemp.length();
                      String balance = new String(balancetemp.substring(0,ballen-1));
                      balances = new String("\rBALANCE = $" + balance);
                }
                if (match >= 1)
                {  // RETURNS ALL INFO IN FORM OF A STRING
                  String info = new String(accounts + "\n" + passwords + "\n" +
                          balances + "\n");
                  return info;
                 } // END OF IF MATCH <<<----------
                else
                {   // DON't DO ANYTHING
                }   // end of else return search string NOT FOUND
            } //  <<<------- end of while inFile Pointer------->>>
            in.close();
            return ("ACCOUNT NOT FOUND...");
           }
          catch(Exception e) {return("error reading from file\n\r");}
  }
  //****************************** PRINT RECORDS (MANAGER METHOD)
  public void printAccts(PrintStream o)
  {  int x = 0;
     try
     { RandomAccessFile in = new RandomAccessFile("account.dat", "r");
       int i = 0;
       o.println("Acct     Password       Balance \r\n");
       while (in.getFilePointer() < in.length()-2)
       { String accttemp = in.readLine();
         String passtemp = in.readLine();
         String baltemp = in.readLine();
         String printinfo = new String(accttemp.trim() + "\t" + passtemp.trim() + "\t\t" +
              baltemp.trim() + "\r\n" );
        o.println(printinfo);
        i++;
       }
       in.close();
     }catch(Exception e){;}
  }
} // end of class ##########################

class Server
{
  public static void main(String[] args )
  { int i=1;
    try
    { ServerSocket s = new ServerSocket(8888);
      for(;;)
      { Socket incoming = s.accept( );
        new ServerHandler(incoming, i).start();
```

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

   }

}
```

Client Version 3

```
// Set up a Client that will read information sent
// from a Server and display the information.
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;


public class AtmApplet extends Applet implements ActionListener       // Client is a Java applet
{
 public final static int port = 8888;
 public final static String hostname = "172.17.52.149";           // Setting up connection with the Server
 TextArea display;
 TextField tf = new TextField("", 4);
 TextField tf1 = new TextField("", 4);


 public void init()
 {
    setLayout(new BorderLayout());            // Building up the GUI
    display = new TextArea( 4, 30 );
    display.setEditable( false );
    add( "South", display );

   Panel  p = new Panel();
   p.setLayout(new FlowLayout());
   Button B = new Button("Done");
   B.addActionListener(this);
   p.add(B);

   add("North", p);

   Panel p1 = new Panel();
   p1.setLayout(new FlowLayout());
   p1.add(new Label("Enter Account No."));            // Accept request from the User
   p1.add(tf);
   p1.add(new Label("Enter PIN"));
   p1.add(tf1);
   add("Center", p1);
 }

public void actionPerformed(ActionEvent evt)
{     String arg = evt.getActionCommand();

      if(arg.equals("Done"))
      {    Socket client;
          String acc = tf.getText();
          String pi = tf1.getText();
          String s = new String(acc + ' ' + pi + '\n');
          BufferedReader in;
          PrintWriter out;
```

```
        try
        {
            client = new Socket(hostname,port);
            in = new BufferedReader(new InputStreamReader(client.getInputStream()));
            out = new PrintWriter(client.getOutputStream(), true);

          // sending data to Server
              out.println(s);

          // receiving data from server
          String c;
          c = in.readLine();
          display.appendText (c);
          display.appendText ("\n");
          client.close();
        } catch ( IOException e ) { System.out.println("NO");}
      }
  }
}
```

Server Version 3

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

class Server
{  public static void main(String[] args)
   {   int i = 1;
       Statement stmt;

       try
       {   ServerSocket serversock = new ServerSocket(8888);     // Set up server socket

                                                                  // Connecting to the database  (JDBC)
           String url = "jdbc:odbc:atmtest";
                String user = "";
                String password = "";
           Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); // force loading of driver
           Connection con = DriverManager.getConnection(url, user, password);
           stmt = con.createStatement();

           for(;;)
           {  Socket incoming_socket = serversock.accept();
              new ServerHandler(incoming_socket, i, stmt, con).start();
              i++;
           }
       } catch(Exception e) {System.out.println(e);}
   }
}


class ServerHandler extends Thread                    // Threading
{       Socket incoming_socket;
        int counter, ctr;
        Statement stmt;
        Connection con;
        int acct_temp;
        int pin_temp;
        BufferedReader in;
        PrintWriter out;
        String temp;
        float bal_temp;

        ServerHandler (Socket i, int c, Statement s, Connection co)
        { incoming_socket = i;
          counter = c;
          stmt = s;
          con = co;
        }

        public void run()
        { try
          { in = new BufferedReader(new InputStreamReader(incoming_socket.getInputStream()));
            out = new PrintWriter(incoming_socket.getOutputStream(), true);
```

```
        int acct_num_length = 4;

        String input = in.readLine();

        System.out.println(input);
        acct_temp = java.lang.Integer.parseInt(input.substring(0,4));
        System.out.println("account number is " + acct_temp);
        pin_temp =  java.lang.Integer.parseInt(input.substring(5,input.length()));
        System.out.println("PIN " + pin_temp);
      } catch(Exception e) {System.out.println(e);}

       try
      {
        String searchquery = "SELECT BALANCE " +     //SQL Statements to connect to the database
                    "FROM ATM.ATMTABLE " +
                    "WHERE ACCOUNT_NO = ? AND PIN = ? ";
        searchquerystmt = con.prepareStatement(searchquery);
        searchquerystmt.setInt(1, acct_temp);
        searchquerystmt.setInt(2, pin_temp);
        ResultSet rs = searchquerystmt.executeQuery();
        while (rs.next())
        { //System.out.println(rs.getString(1)+ ", " + rs.getFloat(2));

          // WRITE BACK TO CLIENT
          temp = new String("Your balance is $" + rs.getFloat("BALANCE")+ '\n');
          System.out.println(temp);

          out.println(temp);
        }

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