A

PROJECT REPORT

ON

**INTRANET BANKING**

*Submitted in partial fulfillment of the requirement for the award of*

**BACHELOR OF TECHNOLOGY**

**IN**

**INFORMATION TECHNOLOGY**

**ABSTRACT**

**AIM**

The aim of our project was to explore new avenues in computingLike the distributed systems along with traditional concepts like OOPS And networking.

**SCOPE**

This project aspires to be a simulation of sorts for a Network Bank in the near future. If coupled with appropriate hardware this system can be turned into ATM software.

**FEATURES**

* + Client-Server Model based on sockets and ports.
  + Multithreaded Server having Microsoft Database.
  + Intermediate Administrator for deposits.
  + Client Side is platform Independent

**MODUELS**

This project Consist of three modules.

1) Server Module

2) Intermediate Administrator

3) Client Module

**Modules Description:**

1. Server:

The Following are the Server side Functionalities.

* Account Creation, Deletion, Updating.
* View Account, Reports, Account Logs.
* Client Authentication.
* Client Monitoring.
* Centralized Data Processing.
* Creation of Account Logs

1. Intermediate Administrator

* Privilege to Deposit.
* Can check Account Status.

3) Client

The Following are the Client side Functionalities.

* Withdraw and Transfer Money.
* View Account, Account Logs.
* Change PIN, Password

**HARDWARE AND SOFTWARE SPECIFICATIONS**

**HARDWARE REQUIREMENTS:**

* PIV 2.8 GHz Processor and Above
* RAM 512MB and Above
* HDD 20 GB Hard Disk Space and Above

**SOFTWARE REQUIREMENTS:**

* WINDOWS XP /2000
* JDK 1.6
* Awt , swing
* Socket Programming

MsAccess 2007 jd

**LIST OF CONTENTS**

**S. No Title Page. No**

**1. Introduction 1**

1.1Purpose 1

1.2Scope 1

1.3 Overview 1

**2. Literature Survey 2**

2.1About Java 2

2.1.1Features of Java 3

2.1.2Java Architecture 5

2.1.3Compilation of code 6

2.2Servlets 7

2.3JavaScript 11

2.4JDBC 14

2.5JSP 16

2.6 Awt and swings 18

2.6.1 Awt 18

2.6.1.1 Awt classes 19

2.6.2 SWINGS 21

2.7 Networking 22

**3. System Analysis 28**

3.1 Existing System 28

3.1.1 Problem statement 29

3.2 Proposed System 29

3.2.1 Advantages 30

**4. System Requirements Specification 30**

**5. System Design 31**

5.1System Specification 31

5.2System Components 33

5.2.1 Modules 33

5.2.2 Modules Description 34

5.3Uml Diagrams 35

**6. Implementation 41**

6.1Source Code 41

**7. System Testing 79**

7.1 Testing 79

7.2 Level of Testing 79

7.2.1 Unit Testing 79

7.2.2 System Testing 80

7.2.3 Integrating Testing 80

7.2.4 Acceptance Testing 81

7.2.5 Validation 81

7.3 Discussion of Results 82

**8. Conclusion and Future Enhancements 90**

**9. References 91**

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| Fig. No | Name of the Figure | Page No’s |
| 1 | Use case diagram of server | 35 |
| 2 | Use case diagram of client | 36 |
| 3 | Intermediate Administrator | 37 |
| 4 | Class diagram | 38 |
| 5 | Sequence diagram | 39 |
| 6 | Collaboration diagram | 40 |

**LIST OF ABBREVIATIONS**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **SYMBOL** | **DESCRIPTION** |
| 1 | OOP | Object Oriented Programming |
| 2 | JVM | Java Virtual Machine |
| 3 | AWT | Abstract Windows Toolkit |
| 4 | TCP | Transmission Control Protocol |
| 5 | UDP | Use Datagram Protocol |
| 6 | HTTP | Hyper Text Transfer Protocol |
| 7 | FTP | File Transfer Protocol |
| 8 | HTML | Hyper Text Markup Language |
| 9 | JDK | Java Development Kit |
| 10 | URL | Uniform Resource Locator |
| 11 | IP | Internet Protocol |
| 12 | SDLC | Software Development Life Cycle |
| 13 | DFD | Data Flow Diagrams |
| 14 | UT | Unit Testing |
| 15 | BBT | Black Box testing |
| 16 | WBT | White Box Testing |
| 17 | IT | Integrating Testing |
| 18 | ST | System Testing |
| 19 | AT | Acceptance Testing |
|  |  |  |

1. INTRODUCTION

1.1 PURPOSE

The project was to explore new avenues in computing like the distributed systems along with traditional concepts like OOPS and networking. An intranet is a private network that is contained within an enterprise. It may consist of many interlinked local area networks and also use leased lines in the wide area network. Typically, an intranet includes connections through one or more gateway computers to the outside Internet. The main purpose of an intranet is to share company information and computing resources among employees. An intranet can also be used to facilitate working in groups and for teleconferences.

**1.2 SCOPE**

This project aspires to be a simulation of sorts for a Network Bank in the near future. If coupled with appropriate hardware this system can be turned into ATM software.

•Any bank, which wants to automate their internal operations, can use our

system.

•The system is being developed exclusively for offline banking, yet it can

easily be integrated with any existing online banking system.

•The scope of our system is limited to the accounts, transaction

management, and logins creation/editing.

•Our system is limited to the bank.

**1.3 OVERVIEW**

The Banking System automates the operations of a bank, which were done manually. It will reduce the overall administrative time of the various processes. Every user of the system should have his own set of privileges, which determines his/her scope**.** It deals with the operations of the bank like account creation and manipulation, transaction management and other value added services .

This system should creates user friendly environment to members of the bank like manager, cashier and clerk etc., by providing interfaces for all of them. It enables faster communication between different users of the system which reduces the overall administrative time. It increases the efficiency of the business by providing efficient and timely service to customer, leading to customer satisfaction thus increasing the overall business of the organization.

The project when completed has the following advantages

* Hassle free system
* No Redundancy
* No chance of fraud
* User friendly
* Saving Resources

**2. LITERATURE SURVEY**

**2.1 ABOUT JAVA AND OOP (OBJECT ORIENTED PROGRAMMING)**

Initially the language was called as “oak” but it was renamed as “Java” in 1995. The primary motivation of this language was the need for a platform-independent (i.e., architecture neutral) language that could be used to create software to be embedded in various consumer electronic devices.

* Java is a programmer’s language.
* Java is cohesive and consistent.
* Except for those constraints imposed by the Internet environment, Java gives the programmer, full control.

Finally, Java is to Internet programming where C was to system programming.

* **IMPORTANCE OF JAVA TO THE INTERNET**

Java has had a profound effect on the Internet. This is because; Java expands the Universe of objects that can move about freely in Cyberspace. In a network, two categories of objects are transmitted between the Server and the Personal computer. They are: Passive information and Dynamic active programs. The Dynamic, Self-executing programs cause serious problems in the areas of Security and probability. But, Java addresses those concerns and by doing so, has opened the door to an exciting new form of program called the Applet.

* **JAVA CAN BE USED TO CREATE TWO TYPES OF PROGRAMS**

An application is a program that runs on our Computer under the operating system of that computer. It is more or less like one creating using C or C++. Java’s ability to create Applets makes it important. An Applet is an application designed to be transmitted over the Internet and executed by a Java –compatible web browser. An applet is actually a tiny Java program, dynamically downloaded across the network, just like an image. But the difference is, it is an intelligent program, not just a media file. It can react to the user input and dynamically change.

* **Applications and Applets**

An application is a program that runs on our Computer under the operating system of that computer. It is more or less like one creating using C or C++. Java’s ability to create Applets makes it important. An Applet is an application designed, to be transmitted over the Internet and executed by a Java –compatible web browser. An applet is actually a tiny Java program, dynamically downloaded across the network, just like an image. But the difference is, it is an intelligent program, not just a media file. It can react to the user input and dynamically change.

**2.1.1 FEATURES OF JAVA**

* **SECURITY**

Every time you that you download a “normal” program, you are risking a viral infection. Prior to Java, most users did not download executable programs frequently, and those who did scanned them for viruses prior to execution. Most users still worried about the possibility of infecting their systems with a virus. In addition, another type of malicious program exists that must be guarded against. This type of program can gather private information, such as credit card numbers, bank account balances, and passwords. Java answers both these concerns by providing a “firewall” between a network application and your computer.

When you use a Java-compatible Web browser, you can safely download Java applets without fear of virus infection or malicious intent.

* **PORTABILITY**

For programs to be dynamically downloaded to all the various types of platforms connected to the Internet, some means of generating portable executable code is needed .As you will see, the same mechanism that helps ensure security also helps create portability. Indeed, Java’s solution to these two problems is both elegant and efficient.

* **THE BYTE CODE**

The key that allows the Java to solve the security and portability problems is that the output of Java compiler is Byte code. Byte code is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (JVM). That is, in its standard form, the JVM is an interpreter for byte code.

Translating a Java program into byte code helps makes it much easier to run a program in a wide variety of environments. The reason is, once the run-time package exists for a given system, any Java program can run on it.

Although Java was designed for interpretation, there is technically nothing about Java that prevents on-the-fly compilation of byte code into native code. Sun has just completed its Just In Time (JIT) compiler for byte code. When the JIT compiler is a part of JVM, it compiles byte code into executable code in real time, on a piece-by-piece, demand basis. It is not possible to compile an entire Java program into executable code all at once, because Java performs various run-time checks that can be done only at run time. The JIT compiles code, as it is needed, during execution.

* **JAVA VIRTUAL MACHINE(JVM)**

Beyond the language, there is the Java virtual machine. The Java virtual machine is an important element of the Java technology. The virtual machine can be embedded within a web browser or an operating system. Once a piece of Java code is loaded onto a machine, it is verified. As part of the loading process, a class loader is invoked and does byte code verification makes sure that the code that’s has been generated by the compiler will not corrupt the machine that it’s loaded on. Byte code verification takes place at the end of the compilation process to make sure that is all accurate and correct. So byte code verification is integral to the compiling and executing of Java code.

* **Overall Description**

# Java Source

## Java byte code

# JavaVM

Java

.Class

Fig 2.1.1 Picture showing the development process of JAVA Program

Java programming uses to produce byte codes and executes them. The first box indicates that the Java source code is located in a. Java file that is processed with a Java compiler called javac. The Java compiler produces a file called a. class file, which contains the byte code. The. Class file is then loaded across the network or loaded locally on your machine into the execution environment is the Java virtual machine, which interprets and executes the byte code

**2.1.2 JAVA ARCHITECTURE**

Java architecture provides a portable, robust, high performing environment for development. Java provides portability by compiling the byte codes for the Java Virtual Machine, which is then interpreted on each platform by the run-time environment. Java is a dynamic system, able to load code when needed from a machine in the same room or across the planet.

**2.1.3 COMPILATION OF CODE**

When you compile the code, the Java compiler creates machine code (called byte code) for a hypothetical machine called Java Virtual Machine (JVM). The JVM is supposed to execute the byte code. The JVM is created for overcoming the issue of portability. The code is written and compiled for one machine and interpreted on all machines. This machine is called Java Virtual machine.

**Source**

**Code**

**………..**

**………..**

**………..**

**…………**

# PC Compiler

**Macintosh**

**Compiler**

**SPARC**

###### Compiler

**Java**

**Byte code**

**(Platform**

**indepen**

**dent)**

**Java**

**Interpreter**

**(PC)**

**Java**

**Interpreter**

**(Macintosh)**

**Java**

**Interpreter**

**(Sparc)**

Fig 2.2 compilation of code

During run-time the Java interpreter tricks the byte code file into thinking that it is running on a Java Virtual Machine. In reality this could be a Intel Pentium Windows 95 or Suns ARC station running Solaris or Apple Macintosh running system and all could receive code from any computer through Internet and run the Applets.

### SIMPLE

Java was designed to be easy for the Professional programmer to learn and to use effectively. If you are an experienced C++ programmer, learning Java will be even easier. Because Java inherits the C/C++ syntax and many of the object oriented features of C++. Most of the confusing concepts from C++ are either left out of Java or implemented in a cleaner, more approachable manner. In Java there are a small number of clearly defined ways to accomplish a given task.

* **OBJECT ORIENTED**

Java was not designed to be source-code compatible with any other language. This allowed the Java team the freedom to design with a blank slate. One outcome of this was a clean usable, pragmatic approach to objects. The object model in Java is simple and easy to extend, while simple types, such as integers, are kept as high-performance non-objects.

* **ROBUST**

The multi-platform environment of the Web places extraordinary demands on a program, because the program must execute reliably in a variety of systems. The ability to create robust programs was given a high priority in the design of Java. Java is strictly typed language; it checks your code at compile time and run time.

Java virtually eliminates the problems of memory management and deallocation, which is completely automatic. In a well-written Java program, all run time errors can –and should –be managed by your program.

**2.2SERVLETS**

* **INTRODUCTION**

The Java web server is JavaSoft's own web Server. The Java web server is just a part of a larger framework, intended to provide you not just with a web server, but also with tools. To build customized network servers for any Internet or Intranet client/server system. Servlets are to a web server, how applets are to the browser.

* **ABOUT SERVLETS**

Servlets provide a Java-based solution used to address the problems currently associated with doing server-side programming, including inextensible scripting solutions, platform-specific APIs, and incomplete interfaces.

Servlets are objects that conform to a specific interface that can be plugged into a Java-based server. Servlets are to the server-side what applets are to the client-side - object byte codes that can be dynamically loaded off the net. They differ from applets in that they are faceless objects (without graphics or a GUI component). They serve as platform independent, dynamically loadable, plugable helper byte code objects on the server side that can be used to dynamically extend server-side functionality.

For example, an HTTP Servlets can be used to generate dynamic HTML content. When you use Servlets to do dynamic content you get the following advantages:

* They’re faster and cleaner than CGI scripts
* They use a standard API (the Servlets API)
* They provide all the advantages of Java (run on a variety of servers without needing to be rewritten).
* **ATTRACTIVENESS OF SERVLETS**

There are many features of Servlets that make them easy and attractive to use. These include:

* Easily configured using the GUI-based Admin tool
* Can be [loaded and invoked](load.html) from a local disk or remotely across the network.
* Can be linked together, or [chained](filter.html), so that one Servlets can call another Servlets, or several Servlets in sequence.
* Can be called dynamically from within HTML pages, using [server-side include](ssinclude.html) tags.
* Are secure - even when downloading across the network, the Servlets security model and Servlets sandbox protect your system from unfriendly behavior.
* **ADVANTAGES OF SERVLET API**

One of the great advantages of the [Servlet API](api.html) is protocol independence. It assumes nothing about:

* The protocol being used to transmit on the net
* How it is loaded
* The server environment it will be running in

These qualities are important, because it allows the Servlet API to be embedded in many different kinds of servers. There are other advantages to the Servlet API as well. These include:

* It’s extensible - you can inherit all your functionality from the base classes made available to you.
* it's simple, small, and easy to use.
* **FEATURES OF SERVLETS**
* Servlets are fast. Since Servlets only need to be loaded once, they offer much better performance over their CGI counterparts.
* Servlets are platform independent.
* Servlets are extensible. Java is a robust, object-oriented programming language, which easily can be extended to suit your needs
* Servlets are secure.
* Servlets can be used with a variety of clients.
* **LOADING SERVLETS**

Servlets can be loaded from three places

From a directory that is on the CLASSPATH. The CLASSPATH of the JavaWebServer includes service root/classes/ which is where the system classes reside.

From the <SERVICE\_ROOT /Servlets/ directory. This is \*not\* in the server's classpath. A class loader is used to create Servlets from this directory. New Servlets can be added - existing Servlets can be recompiled and the server will notice these changes.

From a remote location. For this a code base like http: // nine.eng / classes / foo / is required in addition to the Servlets class name. Refer to the admin GUI docs on Servlet section to see how to set this up.

* **LOADING REMOTE SERVLETS**

Remote Servlets can be loaded by:

1. Configuring the [Admin Tool](../administration/servlet_load.html) to setup automatic loading of remote Servlets
2. Setting up [server side include](ssinclude.html) tags in .shtml files
3. Defining a [filter chain](filter.html) configuration

* **INVOKING SERVLETS**

A Servlet invoker is a Servlet that invokes the "service" method on a named Servlet. If the Servlet is not loaded in the server, then the invoker first loads the Servlet (either from local disk or from the network) and the then invokes the "service" method. Also like applets, local Servlets in the server can be identified by just the class name. In other words, if a Servlet name is not absolute, it is treated as local.

A client can invoke Servlets in the following ways:

* The client can ask for a document that is served by the Servlet.
* The client (browser) can invoke the Servlet directly using a URL, once it has been mapped using the [Servlet Aliases](../administration/servlet_alias.html) section of the admin GUI.
* The Servlet can be invoked through [server side include](ssinclude.html) tags.
* The Servlet can be invoked by placing it in the Servlets/ directory.
* The Servlet can be invoked by using it in a filter chain.

**2.3 JAVASCRIPT**

JavaScript is a script-based programming language that was developed by Netscape Communication Corporation. JavaScript was originally called Live Script and renamed as JavaScript to indicate its relationship with Java. JavaScript supports the development of both client and server components of Web-based applications. On the client side, it can be used to write programs that are executed by a Web browser within the context of a Web page. On the server side, it can be used to write Web server programs that can process information submitted by a Web browser and then updates the browser’s display accordingly

Even though JavaScript supports both client and server Web programming, we prefer JavaScript at Client side programming since most of the browsers supports it. JavaScript is almost as easy to learn as HTML, and JavaScript statements can be included in HTML documents by enclosing the statements between a pair of scripting tags

<SCRIPTS>..</SCRIPT>.

<SCRIPT LANGUAGE = “JavaScript”>

* **JAVASCRIPT STATEMENTS**

</SCRIPT>

Here are a few things we can do with JavaScript :

* Validate the contents of a form and make calculations.
* Add scrolling or changing messages to the Browser’s status line.
* Animate images or rotate images that change when we move the mouse over them.
* Detect the browser in use and display different content for different browsers.
* Detect installed plug-ins and notify the user if a plug-in is required.

We can do much more with JavaScript, including creating entire application.

* **JAVASCRIPT VERSUS JAVA**

JavaScript and Java are entirely different languages. A few of the most glaring differences are:

* Java applets are generally displayed in a box within the web document; JavaScript can affect any part of the Web document itself.
* While JavaScript is best suited o simple applications and adding interactive features to Web pages; Java can be used for incredibly complex applications.

There are many other differences but the important thing to remember is that JavaScript and Java are separate languages. They are both useful for different things; in fact they can be used together to combine their advantages.

* **ADVANTAGES**
* JavaScript can be used for Sever-side and Client-side scripting.
* It is more flexible than VBScript.
* JavaScript is the default scripting languages at Client-side since all the browsers supports it.
* **HYPERTEXT MARKUP LANGUAGE**

Hypertext Markup Language (HTML), the languages of the World Wide Web (WWW), allows users to produces Web pages that include text, graphics and pointer to other Web pages (Hyperlinks).

HTML is not a programming language but it is an application of ISO Standard 8879, SGML (Standard Generalized Markup Language), but specialized to hypertext and adapted to the Web. The idea behind Hypertext is that instead of reading text in rigid linear structure, we can easily jump from one point to another point. We can navigate through the information based on our interest and preference. A markup language is simply a series of elements, each delimited with special characters that define how text or other items enclosed within the elements should be displayed. Hyperlinks are underlined or emphasized works that load to other documents or some portions of the same document.

HTML can be used to display any type of document on the host computer, which can be geographically at a different location. It is a versatile language and can be used on any platform or desktop.

HTML provides tags (special codes) to make the document look attractive. HTML tags are not case-sensitive. Using graphics, fonts, different sizes, color, etc., can enhance the presentation of the document. Anything that is not a tag is part of the document itself.

* **BASIC HTML TAGS**

**<!-- -->** Specifies comments

**<A>……….</A>** Creates hypertext links

**<B>……….</B>** Formats text as bold

**<BIG>……….</BIG>** Formats text in large font.

**<BODY>…</BODY>** Contains all tags and text in the HTML document

**<CENTER>...</CENTER>** Creates text

**<DD>…</DD>** Definition of a term

**<DL>...</DL>**  Creates definition list

**<FONT>…</FONT>** Formats text with a particular font

**<FORM>...</FORM>** Encloses a fill-out form

**<FRAME>...</FRAME>** Defines a particular frame in a set of frames

**<H#>…</H#>** Creates headings of different levels

**<HEAD>...</HEAD>** Contains tags that specify information about a document

**<HR>...</HR>** Creates a horizontal rule

**<HTML>…</HTML>** Contains all other HTML tags

**<META>...</META>** Provides meta-information about a document

**<SCRIPT>…</SCRIPT>** Contains client-side or server-side script

**<TABLE>…</TABLE>**  Creates a table

**<TD>…</TD>** Indicates table data in a table

**<TR>…</TR>** Designates a table row

**<TH>…</TH>** Creates a heading in a table

* **ADVANTAGES**
* A HTML document is small and hence easy to send over the net. It is small because it does not include formatted information.
* HTML is platform independent.
* HTML tags are not case-sensitive.

**2.4 JDBC**

JDBC is a Java API for executing SQL statements. (As a point of interest, JDBC is a trademarked name and is not an acronym; nevertheless, JDBC is often thought of as standing for Java Database Connectivity. It consists of a set of classes and interfaces written in the Java programming language. JDBC provides a standard API for tool/database developers and makes it possible to write database applications using a pure Java API.

Using JDBC, it is easy to send SQL statements to virtually any relational database. One can write a single program using the JDBC API, and the program will be able to send SQL statements to the appropriate database. The combinations of Java and JDBC lets a programmer write it once and run it anywhere.

* **FUNCTIONS OF JDBC**

Simply put, JDBC makes it possible to do three things:

* Establish a connection with a database
* Send SQL statements
* Process the results.
* **JDBC VERSUS ODBC**

At this point, Microsoft's ODBC (Open Database Connectivity) API is that probably the most widely used programming interface for accessing relational databases. It offers the ability to connect to almost all databases on almost all platforms.

So why not just use ODBC from Java? The answer is that you can use ODBC from Java, but this is best done with the help of JDBC in the form of the JDBC-ODBC Bridge, which we will cover shortly. The question now becomes "Why do you need JDBC?" There are several answers to this question:

1. ODBC is not appropriate for direct use from Java because it uses a C interface. Calls from Java to native C code have a number of drawbacks in the security, implementation, robustness, and automatic portability of applications.
2. A literal translation of the ODBC C API into a Java API would not be desirable. For example, Java has no pointers, and ODBC makes copious use of them, including the notoriously error-prone generic pointer "void \*". You can think of JDBC as ODBC translated into an object-oriented interface that is natural for Java programmers.
3. ODBC is hard to learn. It mixes simple and advanced features together, and it has complex options even for simple queries. JDBC, on the other hand, was designed to keep simple things simple while allowing more advanced capabilities where required.
4. A Java API like JDBC is needed in order to enable a "pure Java" solution. When ODBC is used, the ODBC driver manager and drivers must be manually installed on every client machine. When the JDBC driver is written completely in Java, however, JDBC code is automatically installable, portable, and secure on all Java platforms from network computers to mainframes.

* JDBC DRIVER TYPES

The JDBC drivers that we are aware of at this time fit into one of four categories:

* JDBC-ODBC bridge plus ODBC driver
* Native-API partly-Java driver
* JDBC-Net pure Java driver
* Native-protocol pure Java driver
* **JDBC-ODBC BRIDGE**

If possible, use a Pure Java JDBC driver instead of the Bridge and an ODBC driver. This completely eliminates the client configuration required by ODBC. It also eliminates the potential that the Java VM could be corrupted by an error in the native code brought in by the Bridge (that is, the Bridge native library, the ODBC driver manager library, the ODBC driver library, and the database client library).

* **JDBC- ODBC BRIDGE**

The JDBC-ODBC Bridge is a JDBC driver, which implements JDBC operations by translating them into ODBC operations. To ODBC it appears as a normal application program. The Bridge implements JDBC for any database for which an ODBC driver is available. The Bridge is implemented as the sun.jdbc.odbc Java package and contains a native library used to access ODBC. The Bridge is a joint development of Intersolv and JavaSoft.

**2.5 JAVA SERVER PAGES (JSP)**

Java server Pages is a simple, yet powerful technology for creating and maintaining dynamic-content web pages. Based on the Java programming language, Java Server Pages offers proven portability, open standards, and a mature re-usable component model .The Java Server Pages architecture enables the separation of content generation from content presentation. This separation not eases maintenance headaches, it also allows web team members to focus on their areas of expertise. Now, web page designer can concentrate on layout, and web application designers on programming, with minimal concern about impacting each other’s work.

* **FEATURES OF JSP**
* **PORTABILITY**

Java Server Pages files can be run on any web server or web-enabled application server that provides support for them. Dubbed the JSP engine, this support involves recognition, translation, and management of the Java Server Page lifecycle and its interaction components.

* **COMPONENTS**

It was mentioned earlier that the Java Server Pages architecture can include reusable Java components. The architecture also allows for the embedding of a scripting language directly into the Java Server Pages file. The components current supported include Java Beans, and Servlets.

* PROCESSING

A Java Server Pages file is essentially an HTML document with JSP scripting or tags. The Java Server Pages file has a JSP extension to the server as a Java Server Pages file. Before the page is served, the Java Server Pages syntax is parsed and processed into a Servlet on the server side. The Servlet that is generated outputs real content in straight HTML for responding to the client.

* ACCESS MODELS

A Java Server Pages file may be accessed in at least two different ways. A client’s request comes directly into a Java Server Page. In this scenario, suppose the page accesses reusable Java Bean components that perform particular well-defined computations like accessing a database. The result of the Beans computations, called result sets is stored within the Bean as properties. The page uses such Beans to generate dynamic content and present it back to the client.

In both of the above cases, the page could also contain any valid Java code. Java Server Pages architecture encourages separation of content from presentation.

* **STEPS IN THE EXECUTION OF A JSP APPLICATION**

1. The client sends a request to the web server for a JSP file by giving the name of the JSP file within the form tag of a HTML page.
2. This request is transferred to the Java Web Server. At the server side Java Web Server receives the request and if it is a request for a jsp file server gives this request to the JSP engine.
3. JSP engine is program which can understands the tags of the jsp and then it converts those tags into a Servlet program and it is stored at the server side. This Servlet is loaded in the memory and then it is executed and the result is given back to the Java Web Server and then it is transferred back to the result is given back to the Java Web Server and then it is transferred back to the client.

* JDBC CONNECTIVITY

The JDBC provides database-independent connectivity between the J2EE platform and a wide of tabular data sources. JDBC technology allows an Application Component Provider range to:

* Perform connection and authentication to a database server
* Manager transactions
* Move SQL statements to a database engine for preprocessing and execution
* Execute stored procedures

**2.6 INTRODUCTION TO AWT AND SWINGS**

**2.6.1 AWT**

AWT stands for Abstract Window Toolkit. All the capabilities of Graphics, GUI components and Triggers (events) are embedded in AWT. AWT is also a package name, which we must import into our source code with the statement import java.awt. \*;. AWT gives all the classes to make a front-end through which user can interact with the system.

AWT is a gigantic chapter of all packages of JAVA involving graphics, components, layout managers and events. It is easy to manipulate AWT.

Manipulation of Graphics is easier with JAVA than with C/C++. With graphics we can manipulate:

* Color: We can have our choice of background and foreground colors.
* Font: The text’s font size, type and name can be dictated.
* FontMetrics: Font particulars can be obtained.
* Polygon: This class helps us to draw filled and unfilled polygons.
* Toolkit: With this class we can get the operating systems information like fonts it supports.
* **Graphics class**

Graphics class is an abstract class and is very important in the whole AWT. It includes many methods to draw graphics like lines, strings, rectangles and polygons, etc. The paint() method of Component class which writes on the screen takes always a Graphics class object as parameters.

* **paint(), repaint() and update() methods**

The paint() is called implicitly by the JVM for the first time. If needed subsequently, the programmer can call the paint() method indirectly by calling repaint() method. repaint() method calls update() method implicitly. update() method clears the screen of earlier drawings and paint() method (called by repaint()) paints over the cleared screen with the latest settings.

**2.6.1.1AWT CLASSES**

At the top of AWT (Abstract Windows Toolkit) classes is Component class.

* **Component**

It is the top class of AWT hierarchy.All the elements that are displayed on the screen and that interact with the user are subclasses of Component.It contains many methods that are responsible for managing events such as mouse and keyboard listener,positioning and sizing the window and repainting.

This class contains methods for foreground color and background color and selecting text font.

* **Container:**

This class is subclass of component class.so we can access all the methods present in super class(Component).Container is superclass for Panel class and window class.Container class is mainly responsible for laying out components in a frame.

* **Four types of layouts:**

1. FlowLayout

2. BorderLayout

3. GridLayout

4. CardayOut

* **Panel:**

It is a subclass of Container and super class of Applet. When screen output is directed to an applet it is drawn on surface of a Panel. A panel is a window that does not contain titlebar, menubar, border. This is why u don't see these items when an applet is run inside a browser(internet explorer).When u run ur applet using an applet viewer, the applet viewer provides the title and border.

If u want to add some components inside a Panel use add() method inherited from Container class.

* **Window:**

It is subclass of Container and super class of Frame.Window is a top-level window that sits directly on desktop.

* **Frame:**

It is subclass of Window.It is a window that has titlebar,menubar,borders and resizing corners.

* **Canvas:**

It is not part of the hierarchy for applet or frame.It is actually a blank window in which u can draw.

**2.6.2 Swing**

Swing is a set of classes that provides more powerful and flexible components than are possible with the AWT. In addition to the familiar components, such as buttons, checkboxes, and labels, Swing supplies several exciting additions, including tabbed panes, scroll panes, trees, and tables. Even familiar components such as buttons have more capabilities in Swing. For example, a button may have both an image and a text string associated with it. Also, the image can be changed as the state of the button changes.

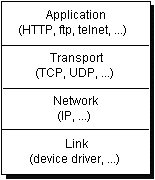
Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written entirely in JAVA and, therefore, are platform-independent. The term lightweight is used to describe such elements.

The number of classes and interfaces in the Swing packages is substantial. Swing is an area that you will want to explore further on your own.

The Swing-related classes are contained in javax.swing and its subpackages, such as javax.swing.tree.

**2.7** **Networking**

Computers running on the Internet communicate to each other using either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), as this diagram illustrates:



When you write Java programs that communicate over the network, you are programming at the application layer. Typically, you don't need to concern yourself with the TCP and UDP layers. Instead, you can use the classes in the java.net package. These classes provide system-independent network communication. However, to decide which Java classes your programs should use, you do need to understand how TCP and UDP differ.

### TCP

When two applications want to communicate to each other reliably, they establish a connection and send data back and forth over that connection. This is analogous to making a telephone call. If you want to speak to Aunt Beatrice in Kentucky, a connection is established when you dial her phone number and she answers. You send data back and forth over the connection by speaking to one another over the phone lines. Like the phone company, TCP guarantees that data sent from one end of the connection actually gets to the other end and in the same order it was sent. Otherwise, an error is reported.

TCP provides a point-to-point channel for applications that require reliable communications. The Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Telnet are all examples of applications that require a reliable communication channel. The order in which the data is sent and received over the network is critical to the success of these applications. When HTTP is used to read from a URL, the data must be received in the order in which it was sent. Otherwise, you end up with a jumbled HTML file, a corrupt zip file, or some other invalid information.

* **Definition:**  TCP (Transmission Control Protocol) is a connection-based protocol that provides a reliable flow of data between two computers.

### UDP

The UDP protocol provides for communication that is not guaranteed between two applications on the network. UDP is not connection-based like TCP. Rather, it sends independent packets of data, called datagram’s, from one application to another. Sending datagram’s is much like sending a letter through the postal service: The order of delivery is not important and is not guaranteed, and each message is independent of any other.

* **Definition:**  UDP (User Datagram Protocol) is a protocol that sends independent packets of data, called datagram’s, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP.

For many applications, the guarantee of reliability is critical to the success of the transfer of information from one end of the connection to the other. However, other forms of communication don't require such strict standards. In fact, they may be slowed down by the extra overhead or the reliable connection may invalidate the service altogether.

The UDP protocol provides for communication that is not guaranteed between two applications on the network. UDP is not connection-based like TCP. Rather, it sends independent packets of data from one application to another. Sending datagram’s is much like sending a letter through the mail service: The order of delivery is not important and is not guaranteed, and each message is independent of any others.

**Note:**  Many firewalls and routers have been configured not to allow UDP packets. If you're having trouble connecting to a service outside your firewall, or if clients are having trouble connecting to your service, ask your system administrator if UDP is permitted.

### Understanding Ports

Generally speaking, a computer has a single physical connection to the network. All data destined for a particular computer arrives through that connection. However, the data may be intended for different applications running on the computer. So how does the computer know to which application to forward the data? Through the use of ports*.*

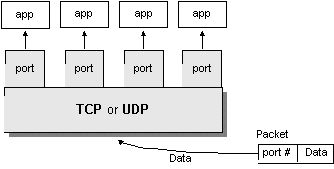
Data transmitted over the Internet is accompanied by addressing information that identifies the computer and the port for which it is destined. The computer is identified by its 32-bit IP address, which IP uses to deliver data to the right computer on the network. Ports are identified by a 16-bit number, which TCP and UDP use to deliver the data to the right application.

In connection-based communication such as TCP, a server application binds a socket to a specific port number. This has the effect of registering the server with the system to receive all data destined for that port. A client can then rendezvous with the server at the server's port, as illustrated here:

2tcp

* **Definition:**  The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer.

In datagram-based communication such as UDP, the datagram packet contains the port number of its destination and UDP routes the packet to the appropriate application, as illustrated in this figure:



Port numbers range from 0 to 65,535 because ports are represented by 16-bit numbers. The port numbers ranging from 0 - 1023 are restricted; they are reserved for use by well-known services such as HTTP and FTP and other system services. These ports are called well-known ports. Your applications should not attempt to bind to them.

### Networking Classes in the JDK:

Through the classes in java.net, Java programs can use TCP or UDP to communicate over the Internet. The URL, URL Connection, Socket, and Server Socket classes all use TCP to communicate over the network. The Datagram Packet, Datagram Socket, and Multicast Socket classes are for use with UDP.

## What Is a URL?

If you've been surfing the Web, you have und

It's often easiest, although not entirely accurate, to think of a URL as the name of a file on the World Wide Web because most URLs refer to a file on some machine on the network. However, remember that URLs also can point to other resources on the network, such as database queries and command output.

* **Definition:**  URL is an acronym for Uniform Resource Locator and is a reference (an address) to a resource on the Internet.

The following is an example of a URL which addresses the Java Web site hosted by Sun Microsystems:

4url

As in the previous diagram, a URL has two main components:

* Protocol identifier
* Resource name

Note that the protocol identifier and the resource name are separated by a colon and two forward slashes. The protocol identifier indicates the name of the protocol to be used to fetch the resource. The example uses the Hypertext Transfer Protocol (HTTP), which is typically used to serve up hypertext documents. HTTP is just one of many different protocols used to access different types of resources on the net. Other protocols include File Transfer Protocol (FTP), Gopher, File, and News.

The resource name is the complete address to the resource. The format of the resource name depends entirely on the protocol used, but for many protocols, including HTTP, the resource name contains one or more of the components listed in the following table:

|  |  |
| --- | --- |
| **Host Name** | The name of the machine on which the resource lives. |
| **Filename** | The pathname to the file on the machine. |
| **Port Number** | The port number to which to connect (typically optional). |
| **Reference** | A reference to a named anchor within a resource that usually identifies a specific location within a file (typically optional). |

For many protocols, the host name and the filename are required, while the port number and reference are optional. For example, the resource name for an HTTP URL must specify a server on the network (Host Name) and the path to the document on that machine (Filename); it also can specify a port number and a reference. In the URL for the Java Web site java.sun.com is the host name and the trailing slash is shorthand for the file named /index.html.

* **Sequence of socket calls for connection-oriented protocol**

# System Calls

Socket - create a descriptor for use in network communication. On success, socket system call returns a small integer value similar to a file descriptor Name.

Bind - Bind a local IP address and protocol port to a socket

When a socket is created it does not have nay notion of endpoint address. An application calls bind to specify the local; endpoint address in a socket. For TCP/IP protocols, the endpoint address uses the socket address in structure. Servers use bind to specify the well-known port at which they will await connections.

Connect - connect to remote client

After creating a socket, a client calls connect to establish an actual connection to a remote server. An argument to connect allows the client to specify the remote endpoint, which include the remote machines IP address and protocols port number. Once a connection has been made, a client can transfer data across it.

Accept () - accept the next incoming connection

Accept creates a new socket for each new connection request and returns the descriptor of the new socket to its caller. The server uses the new socket only for the new connections it uses the original socket to accept additional connection requests once it has accepted connection, the server can transfer data on the new socket.

* **Return Value:**

This system-call returns up to three values

bd14579_ An integer return code that is either an error indication or a new socket description

bd14579_ The address of the client process

bd14579_ The size of this address

Listen - place the socket in passive mode and set the number of incoming TCP connections the system will en-queue. Backlog - specifies how many connections requests can be queued by the system while it wants for the server to execute the accept system call it us usually executed after both the socket and bind system calls, and immediately before the accept system call.

send, send to, recv and recvfrom system calls

These system calls are similar to the standard read and write system calls, but additional arguments are requested.

close - terminate communication and de-allocate a descriptor. The normal UNIX close system call is also used to close a socket.

**3. SYSTEM ANALYSIS**

Efficient and effective management of Core banking activities has become an increasingly imperative and complex activity. Thus automating the entire activity for easier tracking of huge database has become indispensable, for providing cost effective and reliable banking services thus leading customer satisfaction.

This module covers all the Core Banking aspects right from opening a bank account to applying for loans. The system records basic demographic, address data, transaction details, possessions data and also the customer business data. Time consuming tasks such as opening an account, granting OD have become extremely friendly, secure and error free. And also the assignment of privileges to the employees has made the system more robust and secure.

Thus enabling the Bank officials to concentrate more on business development rather than dealing with the current activities of the banks. This ultimately leads to the innovation in the arena of core banking.

**3.1 EXISTING SYSTEM**

The existing system manually records the transactions of the bank like a/c creation, crediting and debiting of accounts etc. It also grants different types of loans to its customers like home loans, mortgage loans vehicle loans etc. Each transaction in this system has to be manually signed. The following are the problems of the existing system.

* Time consuming.
* Requires lot of paper work.
* Laborious
* Tedious
* More complexity
* Vulnerable to natural disasters.

**3.1.1 Problem Statement**

The problem pertains to developing a offline baking system for XYZ Credit Union Ltd., India. The three types of accounts to be covered are:

* Savings account – General savings account without a cheque book; minimum balance required is Rs. 500.00.
* Checking account – General savings account with cheque book; minimum balance is Rs. 1000.00.
* OD account – For corporate customers, this allows negative credit.

The other Details are:

* Approval of the manager is not required for withdrawing cash.
* For savings and checking accounts, a person holding an operational account must introduce.
* Credit limits for OD accounts will be determined based on customers record and history of operations with the bank.
* Customers will not have access to the system.

**3.2** **PROPOSED SYSTEM**

The proposed system automates the operations of a bank, which were done manually. It reduces the overall administrative time of the various processes. Every user of the system has his own set of privileges, which determines his/her scope**.** This system deals with the operations like account creation and manipulation, transaction management and other value added services depending upon user requirements.

This system creates user-friendly environment to members of the bank like manager, cashier and clerk etc., by providing interfaces for each of them. It enables faster communication between different users of the system; whichs reduces the overall administrative time. It increases the efficiency of the business by providing efficient and timely service to customer, leading to customer satisfaction thus increasing the overall business of the organization.

**3.2.1 ADVANTAGES**

* Hassle free system
* No Redundancy
* No chance of fraud
* User friendly
* Saving Resources

1. **SYSTEM REQUIREMENTS AND SPECIFICATION**

**SOFTWARE REQUIREMENTS**

* WINDOWS XP /2000
* JDK 1.6
* Awt , swing
* Socket Programming
* MsAccess 2007

**HARDWARE REQUIREMENTS**

* PIV 2.8 GHz Processor and Above
* RAM 512MB and Above
* HDD 20 GB Hard Disk Space and Above

**5. SYSTEM DESIGN**

**5.1 SYSTEM SPECIFICATION**

Design is the first step in moving from problem domain to the solution domain. Design is essentially the bridge between requirements specification and the final solution.

The goal of design process is to produce a model or representation of a system, which can be used later to build that system. The produced model is called the “Design of the System”. It is a plan for a solution for the system.

The present system is designed in two basic levels.

* System Design
* Detailed Design
* **SYSTEM DESIGN**

# In this level the focus is on dividing which modules are needed for the system and the specification of these modules.

* **DETAILED DESIGN**
* **DATA FLOW DIAGRAM**

A graphical tool used to describe and analyze the moment of data through a system manual or automated including the process, stores of data, and delays in the system. Data Flow Diagrams are the central tool and the basis from which other components are developed. The transformation of data from input to output, through processes, may be described logically and independently of the physical components associated with the system. The DFD is also know as a data flow graph or a bubble chart.

**Context Diagram:**

The top-level diagram is often called a “*context diagram”*. It contains a single process, but it plays a very important role in studying the current system. The context diagram defines the system that will be studied in the sense that it determines the boundaries. Anything that is not inside the process identified in the context diagram will not be part of the system study. It represents the entire software element as a single bubble with input and output data indicated by incoming and outgoing arrows respectively.

* **Types of data flow diagrams**

**DFD’s are of two types**

* Physical DFD
* Logical DFD
* **Physical DFD:**

Structured analysis states that the current system should be first understand correctly. The physical DFD is the model of the current system and is used to ensurethat the current system has been clearly understood. Physical DFDs shows actual devices, departments, and people etc., involved in the current system

* **Logical DFD:**

Logical DFDs are the model of the proposed system. They clearly should show the requirements on which the new system should be built. Later during design activity this is taken as the basis for drawing the system’s structure charts.

The Basic Notation used to create a DFD’s are as follows:

**Dataflow:** Data move in a specific direction from an origin to a

Destination.

**Process:** People, procedures, or devices that use or produce

(Transform) Data. The physical component is not identified.

**Source:** External sources or destination of data, which may be

People, programs, organizations or other entities.

**Data Store:** Here data are stored or referenced by a process in theSystem

**5.2 SYSTEM COMPONENTS**

**5.2.1 MODULES**

This project Consist of three modules.

* Server Module
* Intermediate Administrator
* Client Module

**5.2.2 Modules Description**

* **Server:**

The Following are the Server side Functionalities.

* Account Creation, Deletion, Updating.
* View Account, Reports, Account Logs.
* Client Authentication.
* Client Monitoring.
* Centralized Data Processing.
* Creation of Account Logs
* **Intermediate Administrator**
* Privilege to Deposit.
* Can check Account Status.
* **Client**

The Following are the Client side Functionalities.

* Withdraw and Transfer Money.
* View Account, Account Logs.
* Change PIN, Passwords.

**5.3 UML DIAGRAMS**

**USECASE DIAGRAM OF SERVER**



Fig 5.3.1 Usecase diagram of server

**USECASE DIAGRAM OF CLIENT**



Fig 5.3.2 Usecase diagram of client

**INTERMEDIATE ADMINISTRATOR**



Fig 5.3.3 Intermediate administrator diagram

**CLASS DIAGRAM**



Fig 5.3.4 Class diagram for client and server

**SEQUENCE DIAGRAM**



Fig 5.3.5 Sequence diagram for client and server

**COLLABORATION DIAGRAM**



Fig 5.3.6 Collabration diagram for client and server

**6. IMPLEMENTATION**

**6.1 SOURCE CODE**

import javax.swing.\*;

import javax.swing.event.\*; // for ChangeListener of JTabbedPane.

import java.awt.event.\*;

import java.awt.\*;

import java.io.\*;

import java.net.\*;

import java.util.\*;

import java.text.\*;

import java.sql.\*;

public class Server extends JFrame implements ActionListener, ChangeListener, Runnable {

// a list of the clients that are logged currently:

private static Vector clients = new Vector();

// number of ports to receive messages from clients:

private static int client\_port;

//object to AccessServer class

AccessServer lastClient;

AccessDbase aDbase;

// keeps server's address:

private static InetAddress localHost = null;

// GUI's

AdminEntryLevel adminEntry;

AdminCreateAcc adminCreate;

AdminDeleteAcc adminDelete;

AdminEditAcc adminEdit;

AdminMainMenu adminMain;

AdminUpdateAcc adminUpdate;

AdminViewAccount adminViewAcct;

AdminViewReport adminView;

// AdminMainMenu adminMain;

// AdminCreateAcc adminCreate;

//server sockets for receiving connection from clients:

ServerSocket socketForClient = null;

// visual interface:

String Dattee;

// char dt = new char[50] ;

// button for termination of a client:

JButton btnTerminate = new JButton("Terminate ");

//button for administrator previlages

JButton btnAdministrator = new JButton("Administrator ");

JLabel lblRunning; // label to show how many clients are logged.

JLabel timeRunning; // label to show the updated time.

/\*

JLabel lblWaiting = new JLabel("lblWaiting"); // expected connection status.

JLabel lblWaitAddress = new JLabel("lblWaitAddress"); // waiting player host.

JLabel lblWaitPort = new JLabel("lblWaitPort"); // waiting player port.

JButton btnCancelWait = new JButton("Cancel Waiting"); // button to cancel waiting.

\*/

long acctno,balance;

ImageIcon icon = new ImageIcon("pic.gif"); // picture for tabbedPane.

JTextArea txtInfo = new JTextArea(); // text area for information.

JTabbedPane tabbedPane = new JTabbedPane(); // tabs to select clients.

JPanel pInnerTab = new JPanel (new BorderLayout()); // panel inside the tabbedPane.

JLabel lblDateRunning ;

// thread that cares about accepting new clients:

Thread thClientAccept = null;

// thread that cares about updating the client info:

Thread thUpdateClientInfo = null;

// thread that cares about updating the date:

Thread clockThread = null;

Thread dateThread = null;

String dtString = new String("");;

String currentTime = new String("");

/\*\*

\* Default Constructor.

\*/

public Server() {

super ("The Server"); // set window caption.

// set server sockets:

try {

socketForClient = new ServerSocket(client\_port);

} catch (IOException ioe) {

System.err.println("Cannot open server socket: " + ioe);

System.exit(0);

}

adminEntry= new AdminEntryLevel(this);

adminCreate = new AdminCreateAcc(this);

adminDelete = new AdminDeleteAcc(this);

adminEdit = new AdminEditAcc(this);

adminMain = new AdminMainMenu(this);

adminUpdate = new AdminUpdateAcc(this);

adminViewAcct = new AdminViewAccount(this);

setDisplay();

// adminView = new AdminViewReport(this);

// look & feel setup:

try {

UIManager.setLookAndFeel(

UIManager.getSystemLookAndFeelClassName() );

} catch (Exception e) {

System.err.println("Couldn't use the system "

+ "look and feel: " + e);

}

// update look & feel for those components created in

// declaration (if required):

btnTerminate.updateUI();

btnAdministrator.updateUI();

txtInfo.updateUI();

tabbedPane.updateUI();

pInnerTab.updateUI();

/\* The default value is: HIDE\_ON\_CLOSE,

we need to ask user "Are you sure?" when there are clients logged.

\*/

setDefaultCloseOperation(WindowConstants.DO\_NOTHING\_ON\_CLOSE);

// processing window events:

WindowListener L= new WindowAdapter () {

public void windowClosing(WindowEvent e) {

closeApplication();

}

};

addWindowListener(L);

// prepare the layout:

JPanel pMain = new JPanel (new BorderLayout()); // main layout.

JPanel pUpper = new JPanel (new GridLayout(1,2)); // upper layout.

JPanel pLLBL = new JPanel (new GridLayout(5,1)); // for labels at left.

JPanel pRLBL = new JPanel (new GridLayout(1,1)); // for labels at right.

lblRunning = new JLabel(" Currently logged : 0 client(s).");

lblDateRunning = new JLabel("");

pLLBL.add(new JLabel(" "));

pLLBL.add(new JLabel(" "));

pLLBL.add(lblDateRunning);

pLLBL.add(new JLabel(" Server is running on host: " + localHost));

pLLBL.add(lblRunning);

timeRunning = new JLabel(" ");

// add labels for displaying the time:

pRLBL.add(timeRunning);

// upper part conatins server state labels on one side,

// and current connection state on the other:

pUpper.add(pLLBL);

pUpper.add(pRLBL);

JPanel pBtns = new JPanel (new FlowLayout()); // for "Terminate" button.

btnTerminate.addActionListener(this);

pBtns.add(btnTerminate);

// shown inside the tabbedPane:

pInnerTab.add(txtInfo, BorderLayout.CENTER);

pInnerTab.add(pBtns, BorderLayout.SOUTH);

pMain.add(pUpper, BorderLayout.NORTH);

tabbedPane.addChangeListener(this);

pMain.add(tabbedPane, BorderLayout.CENTER);

JPanel adminBtns = new JPanel (new BorderLayout());

JPanel adminDisplay = new JPanel (new FlowLayout());

adminDisplay.add(new JLabel("Login as : "));

btnAdministrator.addActionListener(this);

adminDisplay.add(btnAdministrator);

adminBtns.add(adminDisplay,BorderLayout.SOUTH);

pMain.add(adminBtns,BorderLayout.SOUTH);

// set content pane:

setContentPane (pMain);

// it doesn't work with our JTabbedPane !!! ---> pack();

setSize(600, 500);

setBounds(100,20,600,500);

// show the window:

setVisible(true);

setResizable(false);

aDbase= new AccessDbase();

aDbase.connectionDb();

pause(2000);

adminView = new AdminViewReport(this);

// start threads:

thClientAccept = new Thread(this);

thUpdateClientInfo = new Thread(this);

clockThread = new Thread(this);

dateThread = new Thread(this);

thClientAccept.start(); // start accepting new clients.

thUpdateClientInfo.start(); // start to care about updating the info.

clockThread.start();

dateThread.start(); // start to care about the time.

}

/\*\*

\* The run method is used by all threads.

\*/

public void run( ) {

Thread thisThread = Thread.currentThread();

// thread that cares about accepting new clients:

while (thClientAccept == thisThread) {

try {

// wait for client to connect, and then validate connection:

// use temporary pointer for new connection:

Socket clientConnection = socketForClient.accept();

System.out.println("Client accepted!");

// register a new client:

addNewTab();

lastClient = new AccessServer(this);

clients.addElement(lastClient);

lblRunning.setText("Currently logged :" + clients.size() + " client(s)" );

// connect client to AccessServer:

lastClient.connectClient(clientConnection);

}catch (Exception e) {

if (thClientAccept != null) // not shutting down yet?

System.err.println("Accept client -> " + e);

}

}

// thread that cares about updating the client info:

// automatic update of info for selected client:

while (thUpdateClientInfo == thisThread) {

if (clients.size() > 0) // no clients?

showClientData();

thUpdateClientInfo.yield(); // this thread isn't so important, think of others.

pause(1200); // update every 1.2 of a second.

}

while (clockThread == thisThread) {

iterateTime();

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

System.err.println("clock thread -> " + e);

// the VM doesn't want us to sleep anymore,

// so get back to work

}

}

while (dateThread == thisThread) {

iterateDate();

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

System.err.println("clock thread -> " + e);

// the VM doesn't want us to sleep anymore,

// so get back to work

}

}

} // end of run().

private void iterateTime() {

// get the time and convert it to a date

Calendar cal = Calendar.getInstance();

java.util.Date date = cal.getTime();

//System.out.println("Display time : " + date.toString());

//char dts[] = date.toString().toCharArray() ;

// Dattee ;

//System.out.println("Display time : " + dts);

//Dattee.valueOf(dts,12,9);

//System.out.println("Display time : " + Dattee);

// format it and display it

DateFormat dateFormatter = DateFormat.getTimeInstance();

SimpleDateFormat dateFormatterH = new SimpleDateFormat("HH:mm:ss");

currentTime = dateFormatterH.format(date);

timeRunning.setText(" Time : " + dateFormatter.format(date));

}

private void iterateDate() {

// get the time and convert it to a date

// format it and display it

Calendar cal = Calendar.getInstance();

java.util.Date date = cal.getTime();

SimpleDateFormat dt = new SimpleDateFormat ("dd MMM yyyy '(' EE ')' ");

//java.util.Date currTime = new java.util.Date();

dtString = dt.format(date);

lblDateRunning.setText(" Date : " + dtString);

}

/\*\*

\* Makes current thread to pause.

\*

\* @param time miliseconds to sleep.

\*/

public void pause(int time) {

try {

Thread.sleep(time);

}

catch (InterruptedException e) {

System.err.println(e.getMessage());

}

}

/\*\*

\* Processes clicks on buttons.

\*

\* @param e the ActionEvent object.

\*/

public void actionPerformed(ActionEvent e){

// JButton tempBtn = (JButton)e.getSource();

JButton src = (JButton)e.getSource();

if (src==btnTerminate) { // terminate the client

try {

AccessServer t; // temporary pointer.

t = (AccessServer) clients.get(tabbedPane.getSelectedIndex());

t.sendToClient("TERMINATED");

removeClient(t);

}

catch (ArrayIndexOutOfBoundsException ae) {

txtInfo.setText( "No client with index: " + tabbedPane.getSelectedIndex());

}

}

else if (src == btnAdministrator){

btnAdministrator.setEnabled(false);

adminEntry.setClear();

adminEntry.setVisible(true);

}

else if(src == adminEntry.logIn)

{

String s=new String(adminEntry.pField.getPassword());

if((adminEntry.txtID.getText().equalsIgnoreCase("Administrator"))&&

(s.equals("Admin") ))

{

System.out.println("AdminEntryLevel: Logged In");

adminEntry.setVisible(false);

adminMain.setClear();

adminMain.setVisible(true);

}

else {

System.out.println("AdminEntryLevel:Log In Failed");

JOptionPane.showMessageDialog(adminEntry,

"Log In Failed",

"Admin Entry Level",

JOptionPane.ERROR\_MESSAGE

);

}

}

else if(src==adminEntry.cancelLogIn)

{

adminEntry.setVisible(false);

btnAdministrator.setEnabled(true);

}

else if(src==adminMain.btnCreate)

{

adminMain.setVisible(false);

adminCreate.setClear();

adminCreate.setVisible(true);

System.out.println("Admin Create Acc");

}

else if(src==adminMain.btnDelete)

{

adminMain.setVisible(false);

adminDelete.setClear();

adminDelete.setVisible(true);

System.out.println("Admin Delete Acc");

}

else if(src==adminMain.btnEdit )

{

adminMain.setVisible(false);

adminEdit.setClear();

adminEdit.setVisible(true);

System.out.println("Admin Edit Acc");

}

else if(src==adminMain.btnViewAcct)

{

adminMain.setVisible(false);

adminViewAcct.setClear();

adminViewAcct.setVisible(true);

}

else if(src == adminMain.btnViewReport)

{

adminMain.setVisible(false);

adminView.setActionCmd();

//adminView.setMaximizedBounds(new Rectangle(0,0,Integer.MAX\_VALUE,Integer.MAX\_VALUE));

adminView.setVisible(true);

}

else if(src==adminMain.btnLogout)

{

adminEntry.setClear();

adminMain.setVisible(false);

adminEntry.setVisible(true);

}

else if(src==adminCreate.btnSubmit)

{

try { if((adminCreate.fields[1].getText().trim().equals(""))||(adminCreate.fields[2].getText().trim().equals(""))||(adminCreate.fields[3].getText().trim().equals(""))||(adminCreate.fields[4].getText().trim().equals(""))||(adminCreate.fields[5].getText().trim().equals(""))||(adminCreate.fields[6].getText().trim().equals(""))||(adminCreate.fields[7].getText().trim().equals("")))

{

JOptionPane.showMessageDialog(adminCreate,

"Fields are incomplete.\n Status :Error" ,

"Admin Create Account",

JOptionPane.ERROR\_MESSAGE

);

}

else

{

aDbase.stmt.executeUpdate("INSERT INTO ClientInfo" + " VALUES( " + adminCreate.fields[0].getText() + ",'" + adminCreate.fields[1].getText() + "','" + adminCreate.fields[2].getText() + "','" + adminCreate.fields[3].getText() + "','" + adminCreate.fields[4].getText() + "','" + adminCreate.fields[5].getText() + "','" + adminCreate.fields[6].getText() + "'," + "'" + adminCreate.fields[7].getText() + "',True," + adminCreate.fields[8].getText() + " )" );

aDbase.stmt.executeUpdate("INSERT INTO ClientAccStatus" + " VALUES( " + adminCreate.fields[0].getText() + ",'" + adminCreate.fields[1].getText() + "',500, No)");

System.out.println("Dbase Created");

JOptionPane.showMessageDialog(adminCreate,

"Account No : " + adminCreate.fields[0].getText() +"\nName : " + adminCreate.fields[1].getText() + "\nPassword : " + adminCreate.fields[2].getText() + "\nPin : " + adminCreate.fields[8].getText() + "\nStatus : Account Created" ,

"Admin Create Account",

JOptionPane.INFORMATION\_MESSAGE

);

adminCreate.setVisible(false);

adminMain.setVisible(true);

}

}

catch(SQLException sqle)

{

System.out.println("Error:"+sqle);

}

}

else if(src==adminCreate.btnCancel)

{

adminCreate.setVisible(false);

adminMain.setVisible(true);

}

else if(src==adminDelete.btnDelete)

{

try{

String s = adminDelete.txtAcctNo.getText();

String updateQuery = "UPDATE ClientInfo SET Validity = False WHERE AccountNo = " + s ;

aDbase.stmt.executeUpdate(updateQuery);

aDbase.uprs = aDbase.stmt.executeQuery("SELECT Name FROM ClientInfo WHERE AccountNo = " + s );

aDbase.uprs.next();

String name = aDbase.uprs.getString(1);

JOptionPane.showMessageDialog(adminDelete,

"Account No : " + s +"\nName : " + name + " \nStatus : Deleted" ,

"Admin Delete Account",

JOptionPane.INFORMATION\_MESSAGE

);

System.out.println("Dbase Deleted");

adminDelete.setVisible(false);

adminMain.setVisible(true);

}

catch(SQLException sqle)

{

JOptionPane.showMessageDialog(adminDelete,

"Invalid Account Number",

"Admin Delete Account",

JOptionPane.ERROR\_MESSAGE

);

System.out.println("Error:"+sqle);

}

}

else if(src==adminDelete.btnCancel)

{

adminDelete.setVisible(false);

adminMain.setVisible(true);

}

else if(src==adminEdit.btnEdit)

{

try {

String s = adminEdit.txtAcctNo.getText();

aDbase.uprs = aDbase.stmt.executeQuery("SELECT \* FROM ClientInfo WHERE AccountNo = " + s );

aDbase.uprs.next();

adminUpdate.fields[0].setText(s);

adminUpdate.fields[0].setEditable(false);

adminUpdate.fields[1].setText(aDbase.uprs.getString(2));

adminUpdate.fields[2].setText(aDbase.uprs.getString(3));

adminUpdate.fields[3].setText(aDbase.uprs.getString(4));

adminUpdate.fields[4].setText(aDbase.uprs.getString(5));

adminUpdate.fields[5].setText(aDbase.uprs.getString(6));

adminUpdate.fields[6].setText(aDbase.uprs.getString(7));

adminUpdate.fields[7].setText(aDbase.uprs.getString(8));

adminEdit.setVisible(false);

adminUpdate.setVisible(true);

}

catch(SQLException sqle)

{

JOptionPane.showMessageDialog(adminEdit,

"Invalid Account Number",

"Admin Edit Account",

JOptionPane.ERROR\_MESSAGE

);

}

}

else if(src==adminEdit.btnCancel)

{

adminEdit.setVisible(false);

adminMain.setVisible(true);

}

else if(src==adminUpdate.btnUpdate)

{

try {

String s= adminEdit.txtAcctNo.getText();

String updateQuery = "UPDATE ClientInfo SET Name = '" + adminUpdate.fields[1].getText() + "', Password = '" + adminUpdate.fields[2].getText() + "', AddressLine1 = '" + adminUpdate.fields[3].getText() + "', AddressLine2 = '" + adminUpdate.fields[4].getText() + "',City = '" + adminUpdate.fields[5].getText() + "', State = '" + adminUpdate.fields[6].getText() + "', Phone = '" + adminUpdate.fields[7].getText() + "', Validity = True WHERE AccountNo = " + s ;

aDbase.stmt.executeUpdate(updateQuery);

aDbase.uprs = aDbase.stmt.executeQuery("SELECT Name,Password FROM ClientInfo WHERE AccountNo = " + s );

aDbase.uprs.next();

String name = aDbase.uprs.getString(1);

String pword = aDbase.uprs.getString(2);

JOptionPane.showMessageDialog(adminUpdate,

"Account No : " + s +"\nName : " + name + "\nPassword : " + pword + "\nStatus : Updated" ,

"Admin Update Account",

JOptionPane.INFORMATION\_MESSAGE

);

adminUpdate.setVisible(false);

adminMain.setVisible(true);

}

catch(SQLException sqle)

{

System.out.println("Error" + sqle);

}

}

else if(src==adminUpdate.btnCancel)

{

adminUpdate.setVisible(false);

adminMain.setVisible(true);

}

else if(src==adminViewAcct.btnDbBegin)

{

try{

String query1 = " SELECT \* FROM ClientInfo ";

String query2 = " SELECT Balance FROM ClientAccStatus ";

aDbase.tmpuprs = aDbase.tmpStmt.executeQuery(query1);

aDbase.tmpuprs.first();

acctno = aDbase.uprs.getLong(1);

String AcNo = Long.toString(acctno);

boolean val = aDbase.uprs.getBoolean(9);

String valid;

if (val)

valid = "Yes";

else

valid = "No";

adminViewAcct.fields[0].setText(" " + AcNo);

adminViewAcct.fields[1].setText(" " + aDbase.uprs.getString(2));

adminViewAcct.fields[3].setText(" " + valid);

adminViewAcct.fields[4].setText(" " + aDbase.uprs.getString(4));

adminViewAcct.fields[5].setText(" " + aDbase.uprs.getString(5));

adminViewAcct.fields[6].setText(" " + aDbase.uprs.getString(6));

adminViewAcct.fields[7].setText(" " + aDbase.uprs.getString(7));

adminViewAcct.fields[8].setText(" " + aDbase.uprs.getString(8));

balance = aDbase.tmpuprs.getLong(1);

String Bal = Long.toString(balance);

adminViewAcct.fields[2].setText(" Rs " + Bal + "\\-");

System.out.println("Admin View Acct |<");

}

catch(SQLException sqle)

{

System.out.println("Error :" + sqle);

}

}

else if(src==adminViewAcct.btnDbBwd)

{

//long acct = Long.parseLong(adminViewAcct.fields[0].getText()) ;

try{

if(!aDbase.uprs.isFirst())

{

aDbase.uprs.previous();

aDbase.tmpuprs.previous();

acctno = aDbase.uprs.getLong(1);

String AcNo = Long.toString(acctno);

boolean val = aDbase.uprs.getBoolean(9);

String valid;

if (val)

valid = "Yes";

else

valid = "No";

adminViewAcct.fields[0].setText(" " + AcNo);

adminViewAcct.fields[1].setText(" " + aDbase.uprs.getString(2));

adminViewAcct.fields[3].setText(" " + valid);

adminViewAcct.fields[4].setText(" " + aDbase.uprs.getString(4));

adminViewAcct.fields[5].setText(" " + aDbase.uprs.getString(5));

adminViewAcct.fields[6].setText(" " + aDbase.uprs.getString(6));

adminViewAcct.fields[7].setText(" " + aDbase.uprs.getString(7));

adminViewAcct.fields[8].setText(" " + aDbase.uprs.getString(8));

balance = aDbase.tmpuprs.getLong(1);

String Bal = Long.toString(balance);

adminViewAcct.fields[2].setText(" Rs " + Bal + "\\-");

System.out.println("Admin View Acct <<");

}

}

catch(SQLException sqle)

{

System.out.println("Error :" + sqle);

}

}

else if(src==adminViewAcct.btnDbFwd)

{

try{

if(!aDbase.uprs.isLast())

{

//acct = acct -1 ;

//String Actno = Long.toString(acct) ;

//String query1 = " SELECT \* FROM ClientInfo WHERE AccountNo = " + Actno ;

//String query2 = " SELECT Balance FROM ClientAccStatus WHERE AccountNo = " + Actno;

//aDbase.uprs = aDbase.tmpStmt.executeQuery(query1);

aDbase.uprs.next();

aDbase.tmpuprs.next();

acctno = aDbase.uprs.getLong(1);

String AcNo = Long.toString(acctno);

boolean val = aDbase.uprs.getBoolean(9);

String valid;

if (val)

valid = "Yes";

else

valid = "No";

adminViewAcct.fields[0].setText(" " + AcNo);

adminViewAcct.fields[1].setText(" " + aDbase.uprs.getString(2));

adminViewAcct.fields[3].setText(" " + valid);

adminViewAcct.fields[4].setText(" " + aDbase.uprs.getString(4));

adminViewAcct.fields[5].setText(" " + aDbase.uprs.getString(5));

adminViewAcct.fields[6].setText(" " + aDbase.uprs.getString(6));

adminViewAcct.fields[7].setText(" " + aDbase.uprs.getString(7));

adminViewAcct.fields[8].setText(" " + aDbase.uprs.getString(8));

balance = aDbase.tmpuprs.getLong(1);

String Bal = Long.toString(balance);

adminViewAcct.fields[2].setText(" Rs " + Bal + "\\-");

System.out.println("Admin View Acct >>");

}

}

catch(SQLException sqle)

{

System.out.println("Error :" + sqle);

}

}

}

/\*\*

\* processes events for the JTabbedPane.

\*

\* @param e the ChangeEvent object.

\*/

public void stateChanged(ChangeEvent e) {

Object src=e.getSource();

if (src == tabbedPane) { // click on a tab

showClientData();

}

}

/\*\*

\* Adds new tab to tabbedPane,

\* the same component is displayed for all tabs,

\* so if there are not tabs the pInnerTab is added.

\*/

private void addNewTab() {

try {

int curTabs = tabbedPane.getTabCount();

if (curTabs == 0) { // no tabs in tabbedPane?

tabbedPane.addTab("Client 1", icon, pInnerTab);

tabbedPane.setSelectedIndex(0);

}

else {

// add empty tab, component from Tab#0 will be used:

tabbedPane.addTab("Client " + (curTabs+1), icon, null);

// activate last tab (newly added):

tabbedPane.setSelectedIndex(curTabs);

}

}

catch (Exception e) {

System.err.println("addNewTab() -> " + e);

}

}

/\*\*

\* Removes the last tab from tabbedPane,

\* (used when game is terminated).

\*/

private void removeLastTab() {

try {

int curTabs = tabbedPane.getTabCount();

tabbedPane.removeTabAt(curTabs-1);//check for the correction..PENDING......

// activate first tab (if any):

if (curTabs > 1)

tabbedPane.setSelectedIndex(0);

}

catch (Exception e) {

System.err.println("removeLastTab() -> " + e);

}

}

/\*\*

\* Updates the text box that shows the information for

\* currectly selected client (by the tabbedPane).

\*/

private void showClientData() {

try {

AccessServer temp; // temporary pointer.

temp = (AccessServer) clients.get(tabbedPane.getSelectedIndex());

String sInfo = temp.getInfo();

if (!sInfo.equals( txtInfo.getText() )) // update text only when required.

txtInfo.setText(sInfo);

}

catch (ArrayIndexOutOfBoundsException ae) {

txtInfo.setText( "No client with index: " + tabbedPane.getSelectedIndex());

}

}

/\*\*

\* Main function, from where Server starts.

\*/

public static void main(String args[]) {

// validate parameter count:

if (args.length!=1) {

System.err.println("Wrong parameters! Usage:");

System.err.println("java Server <client\_port> ");

System.exit(1);

}

// process parameters:

try {

client\_port = Integer.parseInt(args[0]);

}

catch (NumberFormatException e) {

System.err.println("Wrong number for a port -> " + e);

System.exit(1);

}

// get address of the server:

try {

localHost = InetAddress.getLocalHost();

} catch (UnknownHostException e) {

System.out.println("Unknown host - probably localhost with no IP!");

// no exit, since can work on "localhost" without internet.

}

// print out the info (the same info is also shown on the server's

// GUI window).

System.out.println("Server is running on host: " + localHost);

System.out.println("Waiting clients on port: " + client\_port);

// create & start server GUI engine:

new Server();

} // end of main().

/\*\*

\* Closes the server, cares about closing all sockets,

\* and informing the clients are running

\* currenlty about the shutdown, and terminating clients.

\*/

private void closeApplication() {

// ask user if he/she is sure to shut down the server when

// there are clients running:

if (clients.size() > 0) {

int result;

result = JOptionPane.showConfirmDialog(this,

"Are you sure you want to shut down the SERVER?\n" +

"All clients will be terminated!",

"Close anyway?",

JOptionPane.YES\_NO\_OPTION,

JOptionPane.WARNING\_MESSAGE);

if (result != 0) // no, cancel.

return;

// otherwise - yes, close.

// send termination message to all clients:

for (int i=clients.size()-1; i>=0; i--) {

AccessServer temp;

temp = (AccessServer) clients.get(i);

temp.sendToClient("TERMINATED");

removeClient(temp);

}

}

// stop the server's threads:

thClientAccept = null;

// close sockets:

try {

socketForClient.close();

}catch (IOException e) {

System.err.println("On close -> " + e);

}

// close everything:

System.exit(0);

}

public void setDisplay(){

adminEntry.setVisible(false);

adminCreate.setVisible(false);

adminDelete.setVisible(false);

adminEdit.setVisible(false);

adminMain.setVisible(false);

adminUpdate.setVisible(false);

adminViewAcct.setVisible(false);

//adminView.setVisible(false);

}

/\*\*

\* Terminates the client.

\*

\* @param clientToDelete the pointer to client to be deleted.

\*/

public void removeClient(AccessServer clientToDelete) {

if (clients.contains(clientToDelete)) { // check if not removed already.

// close sockets, streams, stop threads:

pause(1500);

clientToDelete.closeEverything();

clients.remove(clientToDelete); // remove from vector.

lblRunning.setText("Currently logged: " + clients.size() + " client(s).");

removeLastTab();

}

}

} //end of server class.

**7. SYSTEM TESTING**

#### 7.1 TESTING

Testing is a process, which reveals errors in the program. It is the major quality measure employed during software development. During testing, the program is executed with a set of conditions known as test cases and the output is evaluated to determine whether the program is performing as expected.

In order to make sure that the system does not have errors, the different levels of testing strategies that are applied at differing phases of software development.

**7.2 LEVELS OF TESTING**

The two levels of Testing are

* Unit Testing
* System Testing
  + 1. **Unit Testing:**

Unit Testing is done on individual modules as they are completed and become executable. It is confined only to the designer's requirements.

Each module can be tested using the following two strategies:

* **Black Box Testing (BBT)**

In this strategy some test cases are generated as input conditions that fully execute all functional requirements for the program. This testing has been uses to find errors in the following categories:

1. Incorrect or missing functions
2. Interface errors
3. Errors in data structure or external database access
4. Performance errors
5. Initialization and termination errors.

In this testing only the output is checked for correctness. The logical flow of the data is not checked.

* **White Box testing (WBT)**

In this the test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases.

It has been u ses to generate the test cases in the following cases:

1. Guarantee that all independent paths have been executed.
2. Execute all logical decisions on their true and false sides.
3. Execute all loops at their boundaries and within their operational bounds.
4. Execute internal data structures to ensure their validity.

**7.2.2 System Testing (ST)**

Involves in-house testing of the entire system before delivery to the user. Its aim is to satisfy the user the system meets all requirements of the client's specifications.

**7.2.3** **Integrating Testing (IT)**

Integration testing ensures that software and subsystems work together as a whole. It tests the interface of all the modules to make sure that the modules behave properly when integrated together.

**7.2.4** **Acceptance Testing (AT)**

It is a pre-delivery testing in which entire system is tested at client's site on real world data to find errors.

**7.2.5 Validation**

The system has been tested and implemented successfully and thus ensured that all the requirements as listed in the software requirement specification are completely fulfilled. In case of erroneous input corresponding error messages are displayed.

##### COMPILING TEST

It was a good idea to do our stress testing early on, because it gave us time to fix some of the unexpected deadlocks and stability problems that only occurred when components were exposed to very high transaction volumes.

##### EXECUTION TEST

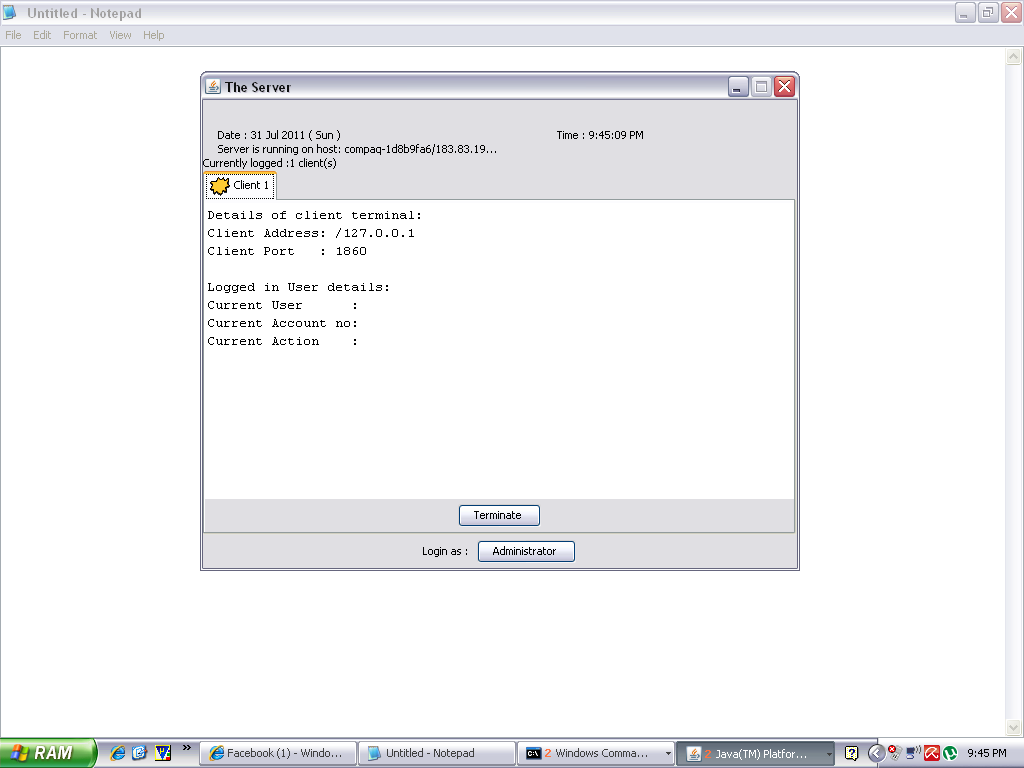
This program was successfully loaded and executed. Because of good programming there were no execution errors.

##### OUTPUT TEST

**7.3 DISCUSSION OF RESULTS**

**SCREENSHOTS**

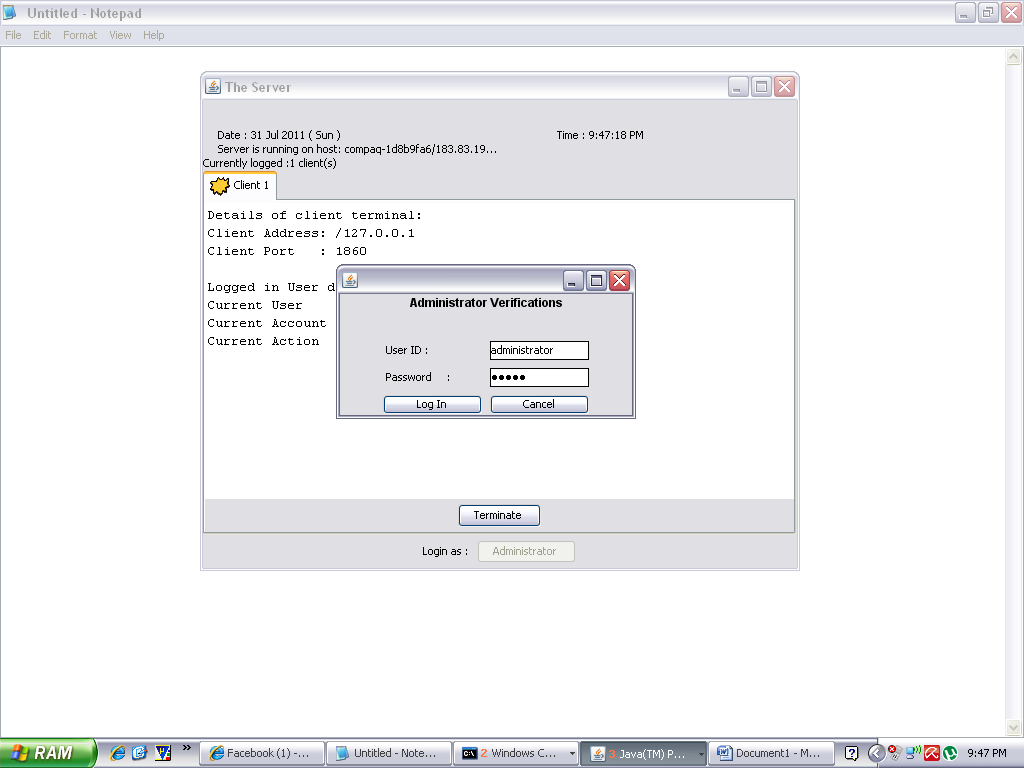
* **SCREENSHOT FOR ADMIN PAGE:**

****

**Fig: 7.3.1 screenshot for admin page.**

The screen is to enter in to the login page as after clicking the button administrator we will enter in to the login page.

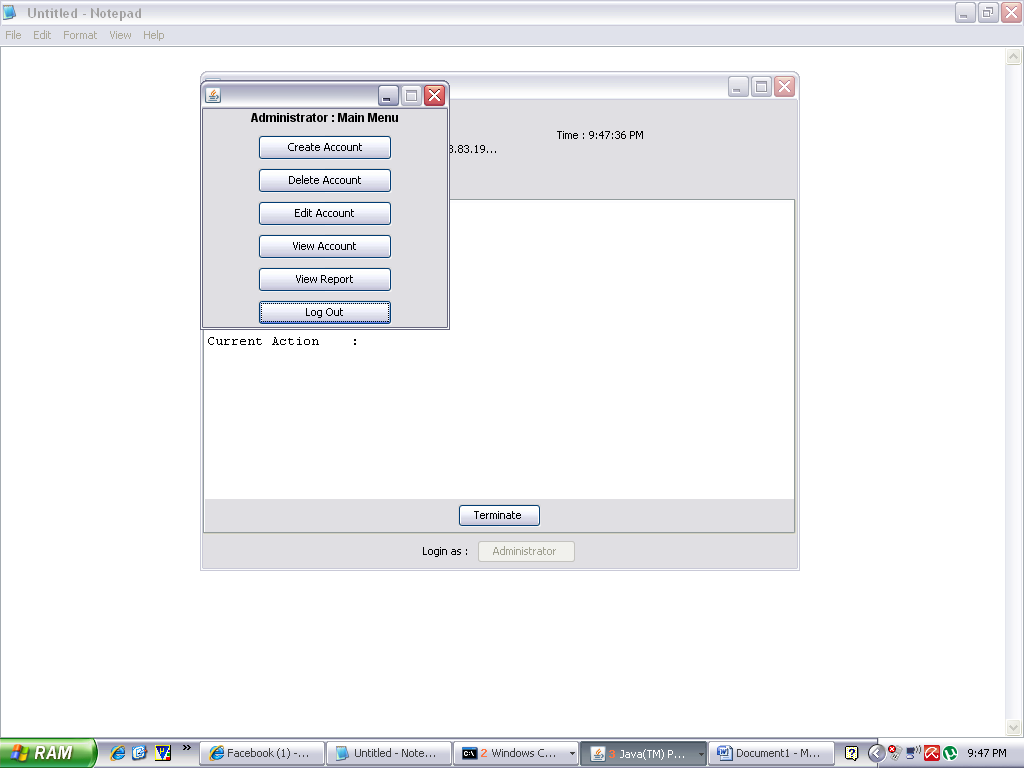
* **SCREENSHOT OF LONGIN PAGE.**

****

**Fig: 7.3.2 Screenshot of login page.**

This page is the login page as we clicked on administrator button on the admin page we entered into the login page as we can see the screen. After giving the user ID and PASSWORD admin can be login to the server side.

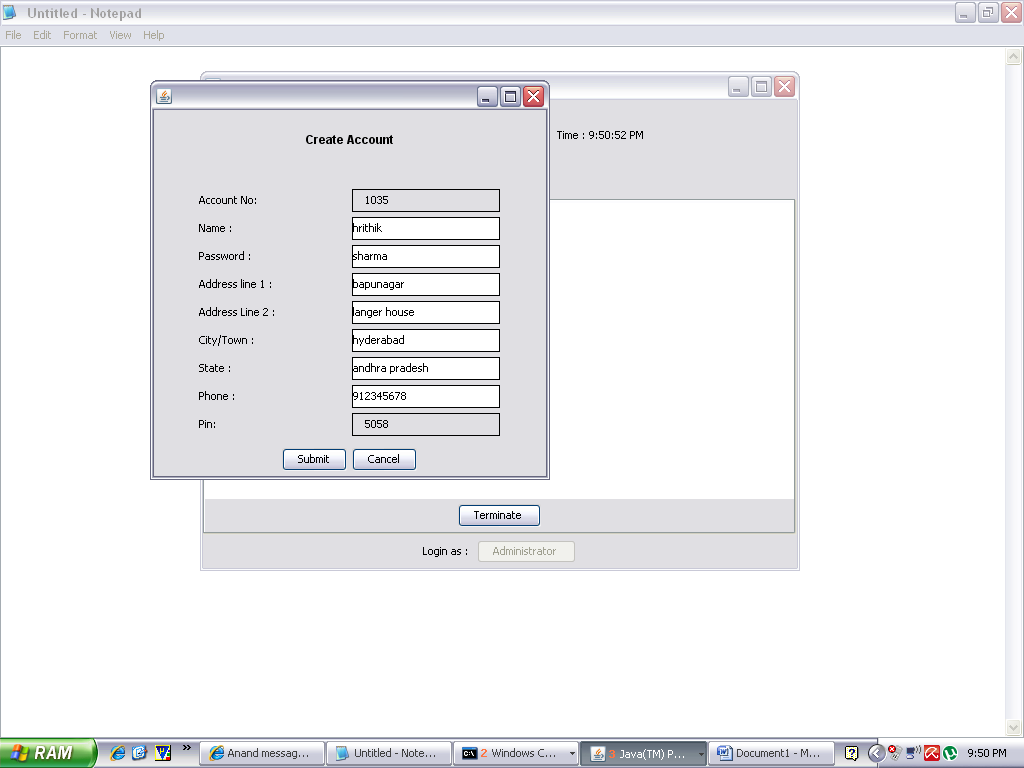
* **SCREENSHOT FOR MAIN MENU .**

****

**Fig 7.3.3 Screenshot for main menu**

This screenshot is for the main page main menu of admin where after login we get all the details of various steps performed by the admin like creating account, editing the account, deleting account etc.

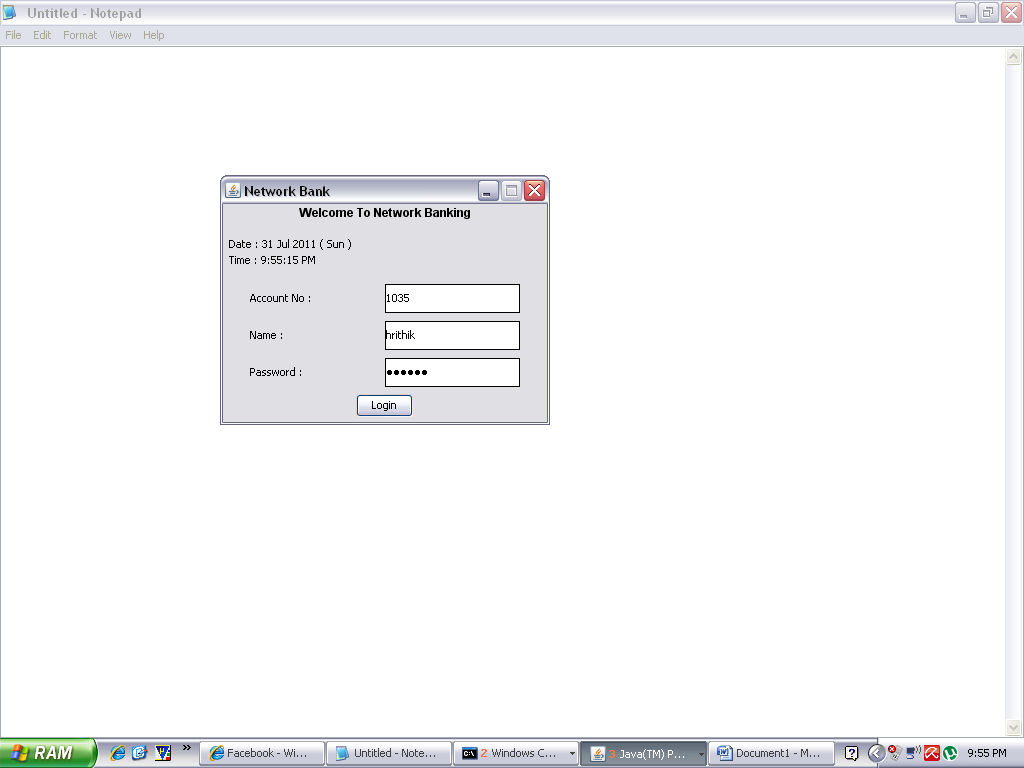
* **SCREENSHOT FOR CREATE ACCOUNT .**

****

**Fig 7.3.4 Screenshot for create account.**

Admin will create an account for the client by filling the above form and admin will provide user id and password for the client so he/she can only access their account. Admin can edit account, delete account, perform this operations by using client account number.

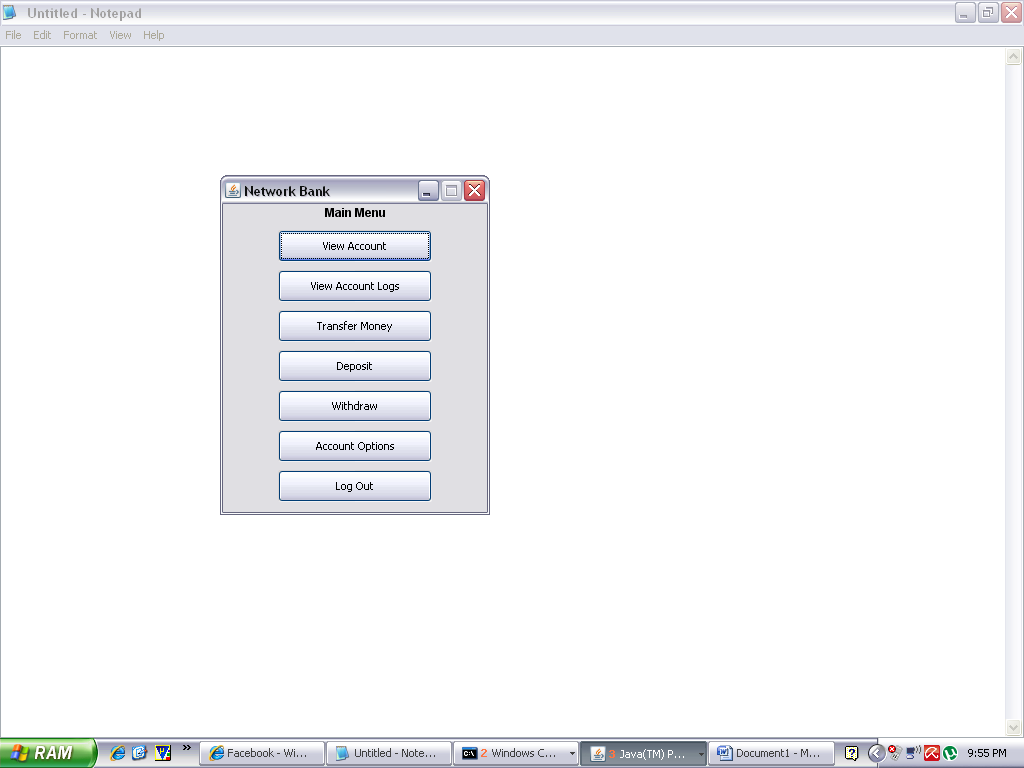
* **SCREENSHOT FOR LOGIN PAGE OF CLIENT .**

****

**Fig 7.3.5 Screenshot for login page of client.**

This screenshot is for the login page of the client and client can log in with user id and password and account number giving by the admin.

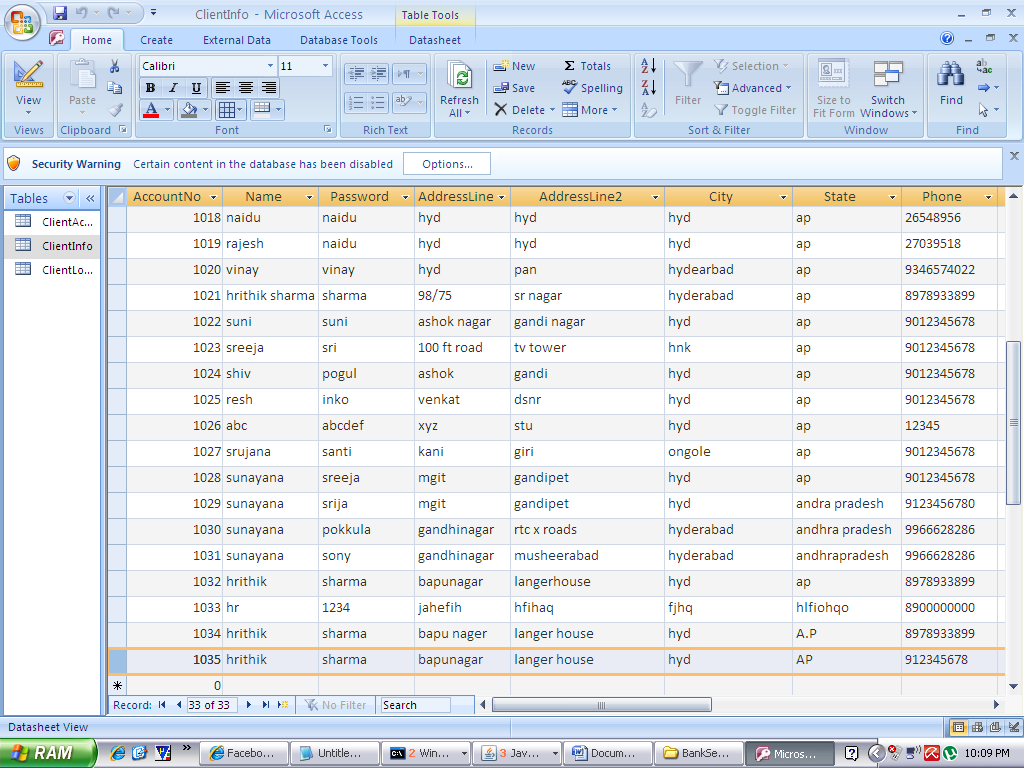
* **SCREENSHOT FOR MAIN MENU FOR THE CLIENT.**

****

**Fig 7.3.6 Screenshot for the main menu of the client.**

This screenshot for client after he/she login their account. Client can view their account by view account, transfer money, withdraw, account option, log out. Client can perform the action by using pin number giving by admin.

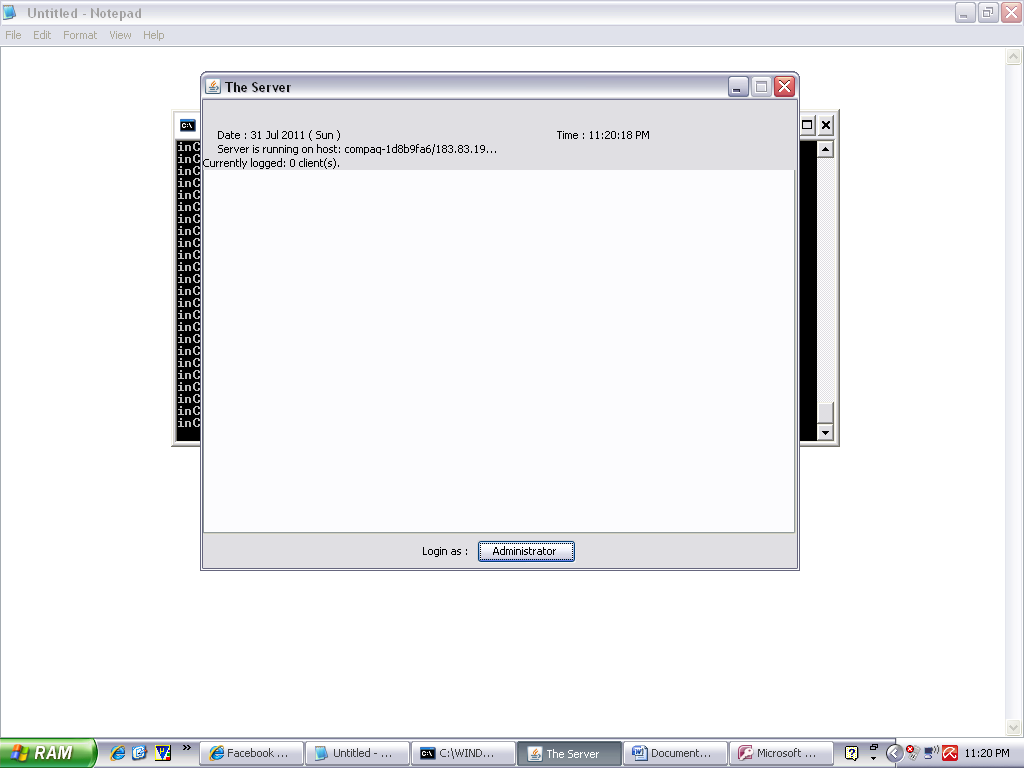
* **SCREENSHOT FOR VIEW OF CLIENT ACCOUNT TO THE ADMIN.**

****

**Fig 7.3.7 Screenshot for view of client account to the admin.**

This screenshot is for the administrator to view client account status, client account info, client log. Administrator can view all the client information. Only admin can view the above information.

* **SCREENSHOT FOR ADMINISTRATOR LOG OUT.**

****

**Fig 7.3.8 Screenshot for administrator logout page.**

This screenshot is for administrator logout page here admin is logout from the server. As the admin is logout automatically client will also logout from their account. After the logout of the client admin should logout from the server.

**8. CONCLUSION AND FUTURE ENHANCEMENTS**

**CONCLUSION**

This application can be used by any bank, which wants to automate their offline banking services. It can be used by the banks to provide their customers with robust and hassle free service.

The application has been so well designed that in case of newer modules to be added it can plug the new modules without affecting the already present modules. The application was tested and found to execute successfully.

**FUTURE ENHANCEMENTS**

The system deals only with core banking offline needs of any bank. Yet is can seamlessly integrate with any of the online banking applications if there is a need for the system to be expanded in future. As the system is built using the platform independent language-java, this system is independent of OS limitations, thus allowing the system cross platform independence.

**9. REFERENCES**

* The Complete reference. - Naughton.
* The complete reference. - Hanna.
* Head First - Bert Bates, Kathy Sierra.
* SQL/PLSQL - Ivan Bayross
* HTML - Black Book

www.javaranch.com

www.java.com

www.javaarchive.com

www.javapassion.com

**www.sun.com**

**www.windows.com**

**www.w3schools.com**