# CONTENTS AT A GLANCE

1. Introduction 2
2. Objective 4
3. System Development Life Cycle 7
4. Feasibility Study 11
5. Software Paradigm used 15
6. Software And Hardware Requirement Specifications 19
7. Information System 25
8. Software Process Models 30
9. Design 35
   * 1. Entity Relationship Diagram 37
     2. Data Flow Diagram 41
10. Coding 47
11. Data Report 101
12. Scope and Limitation 103
13. Bibliography 104

# INTRODUCTION

**INTRODUCTION**

For all types of Banking need to manage their records in such a way that they can retrieve the information as per requirement. They need a set of rulebooks, such as Detail of Banking, Customers etc. Unless the transactions are recorded properly, Accountant will not be in a position to know where exactly he stands. Therefore, for any business record keeping is of foremost importance.

Handling the BankingSystem manually is quite difficult so this software provides aid to the customer by providing easy access to the records. This will improve the efficiency of the work because manually he would take a long time to search for and keep track of all the records of the transactions taking place in the company.

Following is the complete **cycle of Banking System:**

1. The Detail ofCustomers and other detail from opening and day-to-day transactions are first recorded in a file.
2. Periodically these transactions are transferred to concerned Database.
3. At the end of every information balanced can be seen in the reports.

### An additional feature of the software is that it provides security to the Management. It is Password-Protected software, which provides for two types of logins, which are User login and Administrator login.

Another salient feature is the **Reports Generation** facility, which includes the reports for daily transactions and monthly reports. Above - mentioned are some of the prime features of the software developed. It also covers all the basic tasks covered by Information.

# OBJECTIVE

## OBJECTIVE

The main objective of the project is to computerize all the aspects related to Banking System.

The main aim is to develop software to get fast transactionsof money from account at user’s choice; also to completely eliminate the errors occurred during transaction of money.

In brief, the following points describes the main objectives to be achieved through this software:

* To fully **computerize** all the aspects of BankingSystem.
* To design a system that gives the **faster** results and is **more efficient.**
* To provide a **user-friendly** system that is easier to access.
* To make the package easy to access by providing **Menu-Driven** options.
* To design a system that is **reliable.**
* System could add new records maintain them in database.
* To develop **highly secure** software system to avoid any unauthorized user to access any kind of information by providing some security features like:

**Administrator: -**Administrator has his own user name and password to access the package. The Administrator has privileges to update, delete and create user. Administrators can also rollback the database, if required.

**User: *-*** User has his own user name and password. User has all the privileges except deletion, updating and creation of the user. User cannot rollback the database without the permission of administrator.

* To maintain integrity and prevents the unauthorized access.
* To provide better information facilities.

The main objective of the system is to improve the BankingSystem, reduce paperwork, better transaction handling, up to date transaction and possible reduction in staffing requirements and faster retrieval of information and to generate Quick reports. **In short, its main aim is to have a faster and accurate information.**

# SYSTEM ANALYSIS

#### IDENTIFICATION OF NEED

Performing required and defined functions manually may result in a number of different Complications/Problems.

**The Problem of Reliability**

However, manual approach of maintaining Banking information may be reliable but not often so. The reliability of manual approach depends heavily upon the person's nature, behavior, interpersonal relationships etc. Therefore a reliable system is required that can overcome these problems. Since the reliability of computer systems has proven its worth over the time being hence it can be relied upon.

The Reliability means the probability that the software will operate as expected over a specified time interval. The notion of reliability is relative if the consequence of a software error is not serious, the incorrect software may still be reliable.

**The Problem of Accuracy**

A system is inaccurate when processing is error prone. For e.g. In Banking information posting procedure is complex and the number of transactions is large, a fair number of errors may occur. Therefore, routine, transaction based manual procedures are basically suitable for conversion to computer-based methods of processing because the computer is far more accurate than human beings, provided the software is written properly.

**PRELIMINARY INVESTIGATION**

The concept of life cycle of a system is central to **System Investigation**. Every system moves through the several phases of a life cycle during its development, after which it functions with only minor maintenance for a period of years. The system gradually deteriorates to the point where it ceases to function effectively, and a new life cycle begins with the development of a new system.

**System Investigation** is the process of finding every aspect of required system. **System Investigation** provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating reasonable solution, specifying the solution unambiguously, validating the specification and managing the requirements as they are transformed into an operational system.

It certainly seems simple enough – ask the customer, the users, and others what the objective for the system or product are, what is to be accomplished, how the system or product fits into the needs of the business, and finally, how the system or product is to be used on a day - to - day basis.

There are number of **problems** during this phase:

The boundary of the system is ill defined or the customers / users specify unnecessary technical details that may confuse, rather than clarify, overall system objectives.

**Problem of Understanding**

The customers / users are not completely sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment, don’t have a full understanding of the problem, have trouble communicating needs to the system engineer, omit information that is believed to be “obvious”, specify requirements that conflict with the needs of other customer / users, or specify requirements that are ambiguous and untestable.

**Problem of Volatility**

The requirement changes over time. The systems developer built is seldom finished. Demands for new abilities keep coming from user’s side.

To help overcome these problems, system engineers must Approach the requirements gathering activity in an organized manner. The System Investigation is a complex activity with many variables, and there is ample scope to permit wide variety serious blunders.

A **System Investigation** is a project that involves people working together toward the common goal of improving the information system. These problems are due to personality traits that to some degree prevent a system from being properly developed or properly used after development:

**Commitment to the Old Information System**

Users of the existing information system who do not wish to learn to use a new system may oppose it because they are committed to the old system.

# FEASIBILITY

# STUDY

### FEASIBILITY STUDY

**Feasibility is the determination of whether or not a project is worth doing**. The process followed in making this determination is called a Feasibility Study. Generally, feasibility studies are undertaken within tight time constraints and normally culminate in a written and oral feasibility report.

#### DIFFERENT TYPES OF FEASIBILITY

In the conduct of the feasibility study, the analyst will usually consider seven distinct, but inter-related types of feasibility. They are:

**Technical Feasibility**

This is concerned with specifying equipment and software that will successfully satisfy the user requirements. The technical needs of the system may vary considerably, but might include:

* The facility to produce outputs in a given time.
* Response time under certain conditions.
* Ability to process a certain volume of transaction at a particular speed.
* Facility to communicate data to distant location*.*

At the feasibility stage, it is desirable that two or three different configurations will be pursued that satisfy the key technical requirements but which represent different levels of ambition and cost. Out of all types of feasibility, technical feasibility generally is the most difficult to determine.

There are a number of technical issues, which are raised during the feasibility stage of investigation. In respect of proposed system we found that:

* Necessary technology exists to do what is suggested
* Proposed equipment has the technical capacity to hold the data required using the new system.
* System is upgradeable if developed.
* There are technical guarantees of accuracy, reliability, ease of use and data security.

Thus the system is technically feasible.

**Operational Feasibility**

It is mainly related to human organizational and political aspects. The points to be considered are:

* **What changes will be brought with the system?**
* **What organizational structures are disturbed?**
* **What new skills will be required?**
* **Do the existing staff members have these skills? If not, can they be trained in due course of time?**

People are inherent to change. An estimate should be made of how strong a reaction the user staff is likely to have towards the development of a computerized system. Therefore it is understandable that the introduction of a candidate system requires a special effort to educate, sell, and train the staff on new ways of conducting business.

The user can update their database and also use the processed records.

The second such community was the developer. They too were getting benefited as they were getting this software, which can be enhanced very much.

As the proposed system is very user friendly, and asked by the customer showing the willingness to adapt. Thus making the proposed system operational feasible.

**Economic Feasibility**

Economic analysis is the most frequently used technique for evaluating the effectiveness of a proposal system. More commonly known as Cost / Benefit analysis; the procedure is to determine the benefits and savings that are expected from a proposed system and compare them with costs.

Among the most important information contained in feasibility study is cost benefit analysis-an assessment of the economic justification for a computer based system project. Cost-benefit analysis delineates costs for project development and weight them against tangible and intangible benefits of a system.

Economic analysis is the most frequently used method for evaluating the effectiveness of a candidate system. Otherwise, further justification or alterations in the proposed system will have to be made if it is to have a chance for being approved

**Social Feasibility**

Social feasibility is a determination of whether a proposed project will be acceptable to the people or not. This determination typically examines the probability of the project being accepted by the group directly affected by the proposed system change. This project is proved to be socially acceptable and thus social feasible.

# SOFTWARE ENGINEERING PARADIGM USED

**SOFTWARE ENGINEERING PARADIGM USED**

To solve actual problems, developer must incorporate a development strategy that encompasses the process, methods, and tools. This strategy is often referred to as a process model or a software engineering paradigm. Whenever software is engineered, a paradigm is used to design it. The “**Fourth Generation Techniques**” is the paradigm used in this development.

The term **Fourth Generation Techniques** encompasses a broad array of software tools that have one thing in common; each enables the software engineer to specify some characteristics of software at a high level. The tool then automatically generates source code based on the developer’s specification. There is little debate that the higher the level at which software can be specified to a machine, the faster a program can be built. The **4GT** paradigm for software engineering focuses on the ability to specify software using specialized language forms or a graphic notation that describes the problem to be solved in terms that the customer can understand.

Currently software development environment that supports the 4GT paradigm includes some or all of the following tools: **Non-Procedural Languages for database query, Report Generation, Data Manipulation, Screen Interaction and Definition, Code Generation; High-Level Graphics Capability; Spreadsheet Capability and Automated Generation of HTML and similar languages used for Website creation using advanced software tools**. Initially many of the tools noted previously were available only for very specific application domains, but today 4GT environments have been extended to address most software application categories.

Like other paradigms, 4GT begins with a requirement-gathering step. Ideally, the customer would describe requirements and these would be directly translated into an operational prototype. But this is unworkable. The customer may be unsure of what is required, may be ambiguous in specifying facts that are known, and may be unable or unwilling to specify information in a manner that a 4GT tool can consume. For this reason, the customer / developer dialog described for other process models remains an essential part of the 4GT approach.

For small applications, it may be possible to move directly from requirement gathering step to implementation using a non-procedural fourth generation language (4GL) or a model composed of a network of graphical icons. However, for larger efforts, it is necessary to develop a design strategy for the system, even if a 4GL is to be used. The use of 4GT without design (for larger projects) will cause the same difficulties (poor quality, poor maintainability, poor customer acceptance) that have been encountered when developing software using conventional approaches.

Like all software engineering paradigms, the 4GT model has its own advantages and disadvantages. Proponents claim dramatic reduction in software development time and greatly improved productivity for people who build software. Opponent claim that current 4GT tools are not all that much easier to use the programming languages, that the resultant source code produced by such tools is inefficient and that the maintainability of large software systems developed using 4GT is open to the question.

There is some merit in the claims of both sides and it is possible to summarize the current the state of 4GT approaches: -

1. The use of 4GT is viable approach for many different application areas.

1. Data collected from companies that use 4GT indicate that the time required to produce software is greatly reduced for small intermediate applications and that the amount of design and analysis for small applications is also reduced.
2. However the use of 4GT for large software development efforts demands as much or more analysis, design and testing (Software Engineering Activities) to achieve substantial time savings that result from the elimination of coding.

**Here in this project**, the paradigm used is **Fourth Generation Techniques (4GT)**. As the software tools that are being used in this system are **‘Java’** and **‘File Handling’**, both are 4 Generation Tools.

As the project neither belongs to large application category nor to smaller one, hence one can’t discard the requirement-gathering step completely. The project needs the intermediate efforts. The requirement gathering for the successful completion of project was performed in Analysis Phase.

# SOFTWARE

# &

# HARDWARE REQUIREMENT SPECIFICATIONS

**SOFTWARE AND HARDWARE REQUIREMENTS**

The technical needs of a user that are required for implementation of the new system are as follows.

**HARDWARE RESOURCES:**

1. **WINDOWS 2000 Server: (Standard Configuration)**

* Pentium IV based CPU
* 512 MB RAM
* 40 GB HDD
* 15'' color monitor
* Scroll mouse
* 104 keyboard
* VRAM with SVGA card

1. **Printer [If required]**

**SOFTWARE RESOURCES:**

* The Microsoft Window 2000 Server (Operating System)
* Java Enterprise Edition (Front End)
* File Handling (Back End)

**Performance Factor**

The system's performance heavily depends upon hardware configuration. Using latest hardware configuration could rectify hardware problems.

Application on which the system is being developed is the next factor that might affect the performance of the whole system. An application build on GUI is not as fast as an application build on 3GLs but on account of its compatibility to any RDBMS package and its user friendliness makes it a popular application development tool. Since ours package is based on RDBMS hence File Handling as backend and Java as front-end tool has been used extensively.

# SOFTWARE PLATFORM USED

**ABOUT JAVA**

Java is a set of several computer software products and specifications from Sun Microsystems (which has since merged with Oracle Corporation), that together provide a system for developing application software and deploying it in a cross-platform computing environment. Java is used in a wide variety of computing platforms from embedded devicesand mobile phones on the low end, to enterprise servers andsupercomputers on the high end. While less common, Java applets are sometimes used to provide improved and secure functions while browsing the World Wide Web on desktop computers.

Writing in the Java programming language is the primary way to produce code that will be deployed as Java bytecode. There are, however, bytecode compilers available for other languages such as Ada, JavaScript, Python, and Ruby. Several new languages have been designed to run natively on the Java Virtual Machine (JVM), such as Scala, Closure and Groovy. Java syntax borrows heavily from C and C++, but object-oriented features are modeled after Smalltalk and ObjectiveJava eliminates certain low-level constructs such as pointers and has a very simple memory model where every object is allocated on the heap and all variables of object types are references. Memory management is handled through integrated automatic garbage collection performed by the JVM.

On November 13, 2006, Sun Microsystems made the bulk of its implementation of Java available under the GNU General Public License (GPL).

An edition of the Java platform is the name for a bundle of related programs from Sun that allow for developing and running programs written in the Java programming language. The platform is not specific to any one processor or operating system, but rather an execution engine (called a virtual machine) and a compiler with a set of libraries that are implemented for various hardware and operating systems so that Java programs can run identically on all of them.

•**Java Card:** A technology that allows small Java-based applications (applets) to be run securely on smart cards and similar small-memory devices.

•**Java ME (Micro Edition):** Specifies several different sets of libraries (known as profiles) for devices with limited storage, display, and power capacities. Often used to develop applications for mobile devices, PDAs, TV set-top boxes, and printers.

• **Java SE (Standard Edition):** For general-purpose use on desktop PCs, servers and similar devices.

•**Java EE (Enterprise Edition):** Java SE plus various APIs useful for multi-tier client–server enterprise applications.

The Java platform consists of several programs, each of which provides a portion of its overall capabilities. For example, the Java compiler, which converts Java source code into Java bytecode (an intermediate language for the JVM), is provided as part of the Java Development Kit (JDK). The Java Runtime Environment (JRE), complementing the JVM with a just-in-time (JIT) compiler, converts intermediate bytecode into native machine code on the fly. An extensive set of libraries are also part of the Java platform.

The essential components in the platform are the Java language compiler, the libraries, and the runtime environment in which Java intermediate bytecode "executes" according to the rules laid out in the virtual machine specification.

**Java Virtual Machine**

The heart of the Java platform is the concept of a "virtual machine" that executes Java bytecode programs. This bytecode is the same no matter what hardware or operating system the program is running under. There is a JIT (Just in Time) compiler within the Java Virtual Machine, or JVM. The JIT compiler translates the Java byte code into native processor instructions at run-time and caches the native code in memory during execution.

The use of byte code as an intermediate language permits Java programs to run on any platform that has a virtual machine available. The use of a JIT compiler means that Java applications, after a short delay during loading and once they have "warmed up" by being all or mostly JIT-compiled, tend to run about as fast as native programs. Since JRE version 1.2, Sun's JVM implementation has included a just-in-time compiler instead of an interpreter.

Although Java programs are cross-platform or platform independent, the code of the Java Virtual Machines (JVM) that execute these programs is not. Every supported operating platform has its own JVM.

**Class libraries**

In most modern operating systems (OSs), a large body of reusable code is provided to simplify the programmer's job. This code is typically provided as a set of dynamically loadable libraries that applications can call at runtime. Because the Java platform is not dependent on any specific operating system, applications cannot rely on any of the pre-existing OS libraries. Instead, the Java platform provides a comprehensive set of its own standard class libraries containing much of the same reusable functions commonly found in modern operating systems. Most of the system library is also written in Java. For instance, Swing library paints the user interface and handles the events itself, eliminating many subtle differences between how different platforms handle even similar components.

The Java class libraries serve three purposes within the Java platform. First, like other standard code libraries, the Java libraries provide the programmer a well-known set of functions to perform common tasks, such as maintaining lists of items or performing complex string parsing. Second, the class libraries provide an abstract interface to tasks that would normally depend heavily on the hardware and operating system. Tasks such as network access and file access are often heavily intertwined with the distinctive implementations of each platform. The java.net and java.io libraries implement an abstraction layer in native OS code, then provide a standard interface for the Java applications to perform those tasks. Finally, when some underlying platform does not support all of the features a Java application expects, the class libraries work to gracefully handle the absent components, either by emulation to provide a substitute, or at least by providing a consistent way to check for the presence of a specific feature.

# Information System

# INFORMATION SYSTEM

### MEANING AND OBJECTIVES

Despite the fact that the computer is nothing more than a tool for processing data, many managers view it as the central element in an information system. This attitude tends to overrate and distort the role of computer. Its real role is to provide information for decisions and for planning a controlling operations.

Judging from the business press, the brave new world of management information system (MIS) is upon us. These is hardly a business magazine today that does not contain articles on information systems, data banks and related subjects. Despite this proliferation of books, articles, seminars and courses surrounding this area, few efforts have managed to synthesize the separate subjects of management, information, and systems and to show how these are related to computers.

**A management Information system Defined**

MIS is not new; only its computerization is new. Before computers, MIS techniques existed to supply managers with the information that would permit them to plan and control operations. The computer has added one or more dimensions, such as speed, accuracy, and increased volumes of data, that permit the consideration of more alternatives in a decision.

The scope and purpose of MIS is better understood if each part of them is defined.

**Management**

Management has been defined in a variety of ways, but for our purposes it comprises the processes or activities that describe what managers do in the operation of their organization: Plan, organize, initiate, and control operations. They plan by setting strategies and goals and selecting the best course of action to achieve the plan. They organize thetask necessary for the operational plan, set these tasks up into homogeneous groups, and assign authority delegation, they control the performance of the work by setting performance standards and avoiding deviations from standard.

Because decision making is such a fundamental prerequisite to each of the foregoing processes, the job of an MIS becomes of facilitating decisions necessary for planning, organizing, and controlling the work and functions of the business.

**Information**

Data must be distinguished from information, and this distinction is clear and important for our purposes. Data are facts and figures that are not currently being used in a decision process and usually take the form of historical records that are recorded and filed without immediate intent to retrieve for decision making. An example would be and one of the supporting documents, ledgers, and so on that comprise the source material for profit and loss statements. Such material would only be of historical interest to an external auditor.

Information consists of data that have been retrieved, processed, or other- wise used for informative or inference purposes, argument or as a basis for supporting documents already mentioned, but in this case the data could be used by an internal auditor, the management service department of an external auditor, or internal management for profit planning and control or for other decision-making purposes.

**Systems**

A system can be described simply as a set of elements joined together for a common objective. A subsystem is part of a larger system with which we are conceded. All systems are parts of larger systems. For our purposes the organization is the system, and the parts (divisions, departments, functions, units, etc.) are the subsystems.

Whereas we have achieved a very high degree of degree of automation and joining together of subsystems in scientific, mechanical, and factory manufacturing operations, we have barely scratched the surface of applying systems principles to organizational or business systems, The concept of synergism has not generally been applied to the business organization, particularly as it applies to the integration of the subsystems through information interchange. Marketing, operations, and finance are frequently on diverse parts and working at cross purposes. The systems concept of MIS is therefore on of optimizing the output of the organization by connecting the operating subsystems through the medium of information exchange.

A business comes into existence primarily to make profit. In this attempt it provides goods and services to the community. It also provides means of livelihood to some members of the community by creating employment opportunities. These are some subsidiary or secondary motives. The all-important motive of a business, however, is to make profit. **“Profit to a business is like food to a human body: the body must grow and develop - with the assistance of food. Take away the supply of food and the body wastes away and dies.”** Every businessman, therefore, is interested to find out the amount of profit earned (or loss incurred) during a certain period (called‘accountingperiod’), usually a year. In order to find out the amount of profit or loss, he must have a complete and systematic record of the business transactions (i.e., dealings in money or money’s worth) entered into during the year.

Accounting helps a business in having a complete and systematic record of its business transactions in terms of money, reporting results of business activities and interpreting such results for purposes of effective control of future operations or activities. In view of this helpful role of accounting, it has often been called the **language of business**. Like any language, it can never express our thoughts with absolute precision and clarity. Our task of learning this language is complicated by the fact that many of the terms used in accounting mean almost, **but not quite**, the same as they mean in everyday life. You must learn to think of these terms in their accounting rather than their popular meaning.

# SOFTWARE PROCESS MODELS

* **The waterfall model**
  + Plan-driven model. Separate and distinct phases of specification and development.
* **Incremental development**
  + Specification, development and validation are interleaved. May be plan-driven or agile.
* **Reuse-oriented software engineering**
  + The system is assembled from existing components. May be plan-driven or agile.

In practice, most large systems are developed using a process that incorporates elements from all of these models.

The waterfall model

# 2.1.Waterfall-model.eps

**Waterfall model phases**

* There are separate identified phases in the waterfall model:
  + Requirements analysis and definition
  + System and software design
  + Implementation and unit testing
  + Integration and system testing
  + Operation and maintenance
* The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

**Waterfall model problems**

* Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
* Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
* Few business systems have stable requirements.
* The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

# In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

# DESIGN

**Architectural Design**

Architectural Design represents the structure of data and program components that are required to build this computer based system.

The software requirement can be mapped into various representation of the design model. Structure design is often characterized often as a dataflow oriented design method because it provides a convenient transition from a DFD to a software.

Data Design of Accounts Managing Package creates a model of data and information that is represented at a high level of abstraction.

Architectural design is represented through **ERDs, DFDs, Data Structure used.**

* **Interface Design:**

Once task analysis has been completed, all tasks required by the end user have been identified in details and interface design activity commences.

Firstly in the design phase the goals are identified which, include a consideration of the usefulness of the task, its effectiveness in accomplishing the overriding official objectives and the degree to which the task can be learn quickly with the ultimate implementation of the task and also the intention.

Each goal and intention must be mapped in to a sequence of specific actions.

**Interface design includes form designs and layouts.**

* **Component-level Design**

This design is also being called procedural design, the intent is to translate the design model into operational software. The main construct of this design is sequence, condition and repetition.

**Pseudo codes and Flow Charts are used to represent the component level design through these entire three constructs*.***

ENTITY RELATIONSHIP DIAGRAM

Date of Account

Created

Account

Account no.

BANKING SYSTEM

Name

Creates

Banking

System

Creates

Account

Name

Deposit

Amount

Add info.

To records

Deposit

Date

Amount

Withdrawl

Date of withdrawl

Type of Banking

Name

Account no.

Account

Creates

Banking

System

DFD

(Data Flow Diagram)

Role=Banking

User

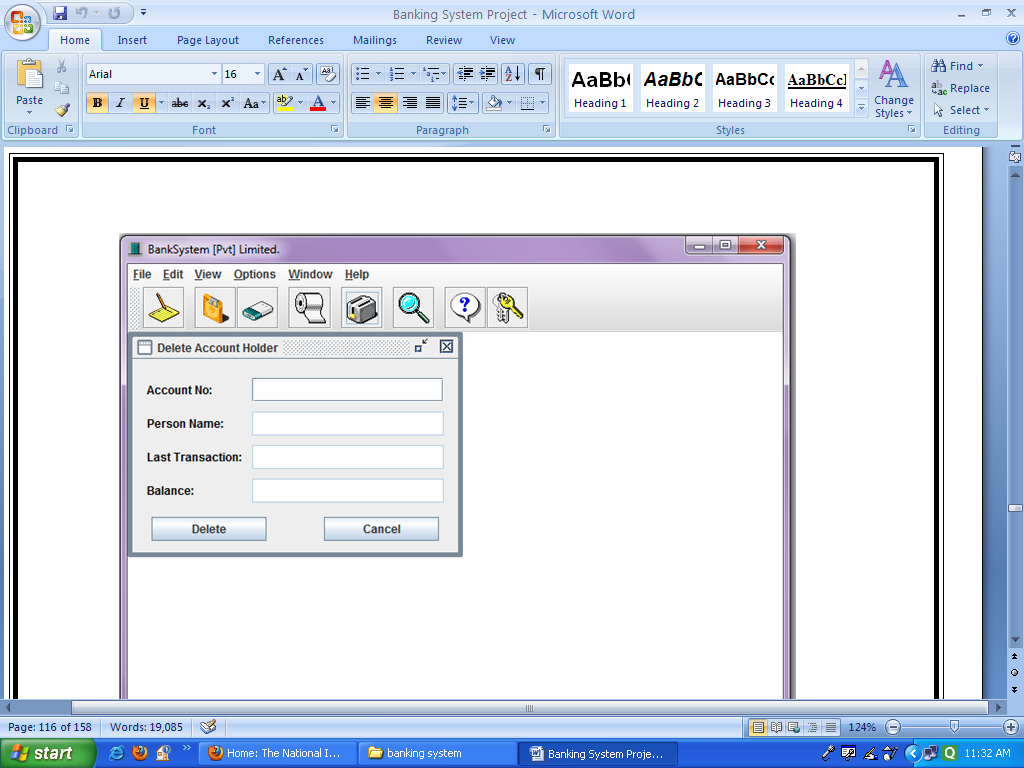
Role=Banking

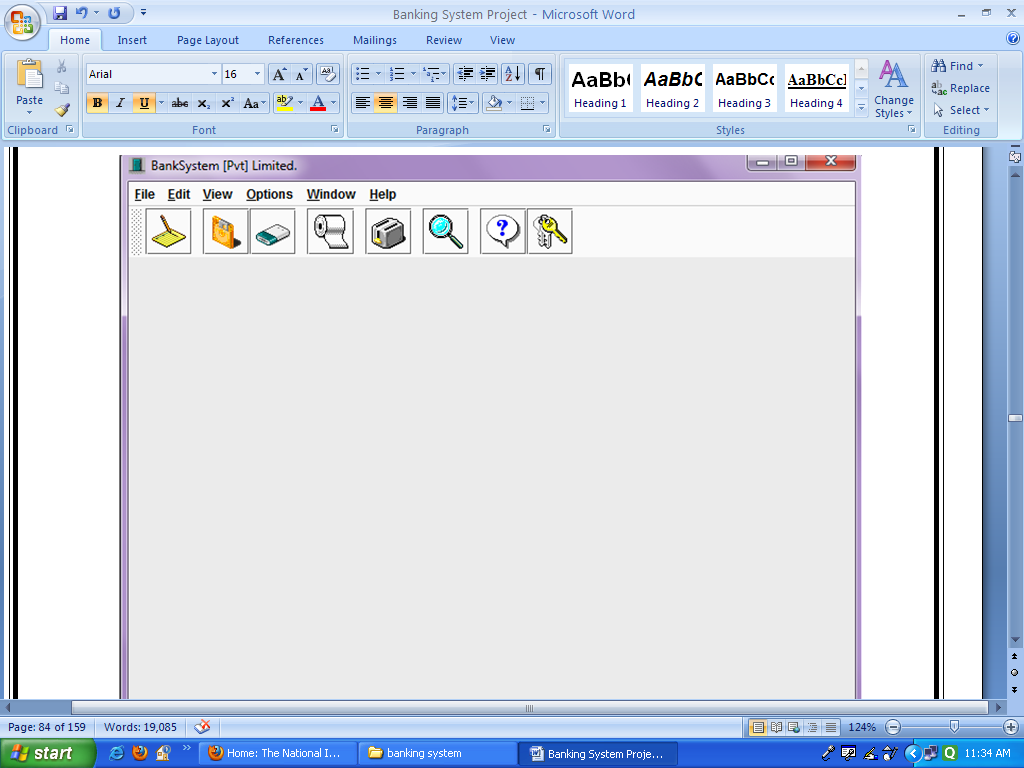
Create Account

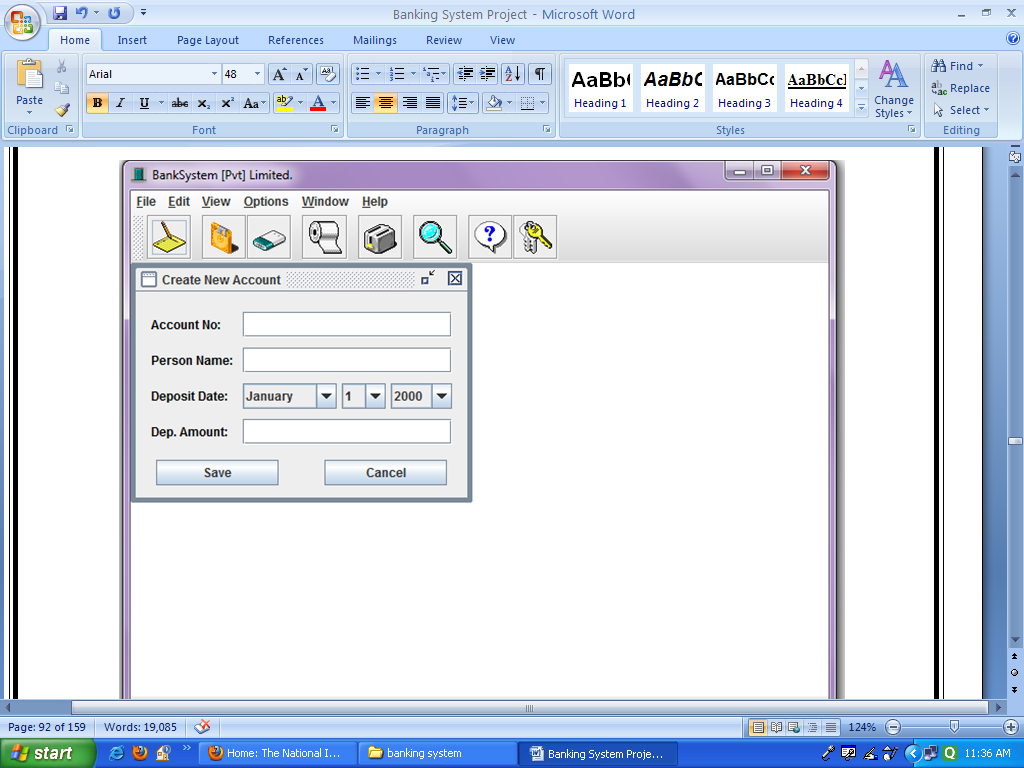
Account no.

New user

Users

****

****

****

**CODING**

**CODING**

**Main Class**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import javax.swing.event.\*;

import java.util.\*;

import java.text.\*;

import java.io.\*;

import java.awt.PrintJob.\*;

import javax.swing.plaf.metal.\*;

public class BankSystem extends JFrame implements ActionListener, ItemListener {

//Main Place on Form where All Child Forms will Shown.

private JDesktopPane desktop = new JDesktopPane ();

//For Program's MenuBar.

private JMenuBar bar;

//All the Main Menu of the Program.

private JMenu mnuFile, mnuEdit, mnuView, mnuOpt, mnuWin, mnuHelp;

private JMenuItem addNew, printRec, end;

//File Menu Options.

private JMenuItem deposit, withdraw, delRec, search, searchName;

//Edit Menu Options.

private JMenuItem oneByOne, allCustomer;

//View Menu Options.

private JMenuItem change, style, theme;

//Option Menu Options.

private JMenuItem close, closeAll;

//Window Menu Options.

private JMenuItem content, keyHelp, about;

//Help Menu Options.

//PopupMenu of Program.

private JPopupMenu popMenu = new JPopupMenu ();

//MenuItems for PopupMenu of the Program.

private JMenuItem open, report, dep, with, del, find, all;

//For Program's ToolBar.

private JToolBar toolBar;

//For ToolBar's Button.

private JButton btnNew, btnDep, btnWith, btnRec, btnDel, btnSrch, btnHelp, btnKey;

//Main Form StatusBar where Program's Name & Welcome Message Display.

private JPanel statusBar = new JPanel ();

//Labels for Displaying Program's Name & saying Welcome to Current User on StatusBar.

private JLabel welcome;

private JLabel author;

//Making the LookAndFeel Menu.

private String strings[] = {"1. Metal", "2. Motif", "3. Windows"};

private UIManager.LookAndFeelInfo looks[] = UIManager.getInstalledLookAndFeels ();

private ButtonGroup group = new ButtonGroup ();

private JRadioButtonMenuItem radio[] = new JRadioButtonMenuItem[strings.length];

//Getting the Current System Date.

private java.util.Date currDate = new java.util.Date ();

private SimpleDateFormat sdf = new SimpleDateFormat ("dd MMMM yyyy", Locale.getDefault());

private String d = sdf.format (currDate);

//Following all Variables are use in BankSystem's IO's.

//Variable use in Reading the BankSystem Records File & Store it in an Array.

private int count = 0;

private int rows = 0;

private int total = 0;

//String Type Array use to Load Records From File.

private String records[][] = new String [500][6];

//Variable for Reading the BankSystem Records File.

private FileInputStream fis;

private DataInputStream dis;

//Constructor of The Bank Program to Iniatilize all Variables of Program.

public BankSystem () {

//Setting Program's Title.

super ("BankSystem [Pvt] Limited.");

UIManager.addPropertyChangeListener (new UISwitchListener ((JComponent)getRootPane()));

//Creating the MenuBar.

bar = new JMenuBar ();

//Setting the Main Window of Program.

setIconImage (getToolkit().getImage ("Images/Bank.gif"));

setSize (700, 550);

setJMenuBar (bar);

//Closing Code of Main Window.

addWindowListener (new WindowAdapter ()

{

public void windowClosing (WindowEvent we)

{

quitApp ();

}

}

);

//Setting the Location of Application on Screen.

setLocation((Toolkit.getDefaultToolkit().getScreenSize().width - getWidth()) / 2,

(Toolkit.getDefaultToolkit().getScreenSize().height - getHeight()) / 2);

//Creating the MenuBar Items.

mnuFile = new JMenu ("File");

mnuFile.setMnemonic ((int)'F');

mnuEdit = new JMenu ("Edit");

mnuEdit.setMnemonic ((int)'E');

mnuView = new JMenu ("View");

mnuView.setMnemonic ((int)'V');

mnuOpt = new JMenu ("Options");

mnuOpt.setMnemonic ((int)'O');

mnuWin = new JMenu ("Window");

mnuWin.setMnemonic ((int)'W');

mnuHelp = new JMenu ("Help");

mnuHelp.setMnemonic ((int)'H');

//Creating the MenuItems of Program.

//MenuItems for FileMenu.

addNew = new JMenuItem ("Open New Account", new ImageIcon ("Images/Open.gif"));

addNew.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_N, Event.CTRL\_MASK));

addNew.setMnemonic ((int)'N');

addNew.addActionListener (this);

printRec = new JMenuItem ("Print Customer Balance", new ImageIcon ("Images/New.gif"));

printRec.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_R, Event.CTRL\_MASK));

printRec.setMnemonic ((int)'R');

printRec.addActionListener (this);

end = new JMenuItem ("Quit BankSystem ?", new ImageIcon ("Images/export.gif"));

end.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_Q, Event.CTRL\_MASK));

end.setMnemonic ((int)'Q');

end.addActionListener (this);

//MenuItems for EditMenu.

deposit = new JMenuItem ("Deposit Money");

deposit.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_T, Event.CTRL\_MASK));

deposit.setMnemonic ((int)'T');

deposit.addActionListener (this);

withdraw = new JMenuItem ("Withdraw Money");

withdraw.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_W, Event.CTRL\_MASK));

withdraw.setMnemonic ((int)'W');

withdraw.addActionListener (this);

delRec = new JMenuItem ("Delete Customer", new ImageIcon ("Images/Delete.gif"));

delRec.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_D, Event.CTRL\_MASK));

delRec.setMnemonic ((int)'D');

delRec.addActionListener (this);

search = new JMenuItem ("Search By No.", new ImageIcon ("Images/find.gif"));

search.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_S, Event.CTRL\_MASK));

search.setMnemonic ((int)'S');

search.addActionListener (this);

searchName = new JMenuItem ("Search By Name");

searchName.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_M, Event.CTRL\_MASK));

searchName.setMnemonic ((int)'M');

searchName.addActionListener (this);

//MenuItems for ViewMenu.

oneByOne = new JMenuItem ("View One By One");

oneByOne.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_O, Event.CTRL\_MASK));

oneByOne.setMnemonic ((int)'O');

oneByOne.addActionListener (this);

allCustomer = new JMenuItem ("View All Customer", new ImageIcon ("Images/refresh.gif"));

allCustomer.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_A, Event.CTRL\_MASK));

allCustomer.setMnemonic ((int)'A');

allCustomer.addActionListener (this);

//MenuItems for OptionMenu.

change = new JMenuItem ("Change Background Color");

change.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_B, Event.CTRL\_MASK));

change.setMnemonic ((int)'B');

change.addActionListener (this);

//Menu For Changing the Program's Layout.

style = new JMenu ("Change Layout Style");

style.setMnemonic ((int)'L');

for( int i = 0; i < radio.length ; i++ ) { //Creating the subMenus of Style Menu.

radio[i] = new JRadioButtonMenuItem (strings[i]);

//Build an Array of Layouts to Apply.

radio[i].addItemListener (this);

//Setting their Actions.

group.add (radio[i]);

//Making them Grouped.

style.add (radio[i]);

//Adding to Style MenuOption.

}

//SubMenu of Theme For Applying different Themes to Program By Building an Array of Themes to Apply.

MetalTheme[] themes = { new DefaultMetalTheme(), new GreenTheme(), new AquaTheme(),

new SandTheme(), new SolidTheme(), new MilkyTheme(), new GrayTheme() };

theme = new MetalThemeMenu ("Apply Theme", themes);

//Putting the Themes in ThemeMenu.

theme.setMnemonic ((int)'M');

//MenuItems for WindowMenu.

close = new JMenuItem ("Close Active Window");

close.setMnemonic ((int)'C');

close.addActionListener (this);

closeAll = new JMenuItem ("Close All Windows...");

closeAll.setMnemonic ((int)'A');

closeAll.addActionListener (this);

//MenuItems for HelpMenu.

content = new JMenuItem ("Help Contents", new ImageIcon ("Images/paste.gif"));

content.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_H, Event.CTRL\_MASK));

content.setMnemonic ((int)'H');

content.addActionListener (this);

keyHelp = new JMenuItem ("Help on Shortcuts...");

keyHelp.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_K, Event.CTRL\_MASK));

keyHelp.setMnemonic ((int)'K');

keyHelp.addActionListener (this);

about = new JMenuItem ("About BankSystem", new ImageIcon ("Images/Save.gif"));

about.setAccelerator (KeyStroke.getKeyStroke(KeyEvent.VK\_C, Event.CTRL\_MASK));

about.setMnemonic ((int)'C');

about.addActionListener (this);

//Adding MenuItems to Menu.

//File Menu Items.

mnuFile.add (addNew);

mnuFile.addSeparator ();

mnuFile.add (printRec);

mnuFile.addSeparator ();

mnuFile.add (end);

//Edit Menu Items.

mnuEdit.add (deposit);

mnuEdit.add (withdraw);

mnuEdit.addSeparator ();

mnuEdit.add (delRec);

mnuEdit.addSeparator ();

mnuEdit.add (search);

mnuEdit.add (searchName);

//View Menu Items.

mnuView.add (oneByOne);

mnuView.addSeparator ();

mnuView.add (allCustomer);

//Options Menu Items.

mnuOpt.add (change);

mnuOpt.addSeparator ();

mnuOpt.add (style);

mnuOpt.addSeparator ();

mnuOpt.add (theme);

//Window Menu Items.

mnuWin.add (close);

mnuWin.add (closeAll);

//Help Menu Items.

mnuHelp.add (content);

mnuHelp.addSeparator ();

mnuHelp.add (keyHelp);

mnuHelp.addSeparator ();

mnuHelp.add (about);

//Adding Menues to Bar.

bar.add (mnuFile);

bar.add (mnuEdit);

bar.add (mnuView);

bar.add (mnuOpt);

bar.add (mnuWin);

bar.add (mnuHelp);

//MenuItems for PopupMenu.

open = new JMenuItem ("Open New Account", new ImageIcon ("Images/Open.gif"));

open.addActionListener (this);

report = new JMenuItem ("Print BankSystem Report", new ImageIcon ("Images/New.gif"));

report.addActionListener (this);

dep = new JMenuItem ("Deposit Money");

dep.addActionListener (this);

with = new JMenuItem ("Withdraw Money");

with.addActionListener (this);

del = new JMenuItem ("Delete Customer", new ImageIcon ("Images/Delete.gif"));

del.addActionListener (this);

find = new JMenuItem ("Search Customer", new ImageIcon ("Images/find.gif"));

find.addActionListener (this);

all = new JMenuItem ("View All Customer", new ImageIcon ("Images/refresh.gif"));

all.addActionListener (this);

//Adding Menues to PopupMenu.

popMenu.add (open);

popMenu.add (report);

popMenu.add (dep);

popMenu.add (with);

popMenu.add (del);

popMenu.add (find);

popMenu.add (all);

//Following Procedure display the PopupMenu of Program.

addMouseListener (new MouseAdapter () {

public void mousePressed (MouseEvent me) { checkMouseTrigger (me); }

public void mouseReleased (MouseEvent me) { checkMouseTrigger (me); }

private void checkMouseTrigger (MouseEvent me) {

if (me.isPopupTrigger ())

popMenu.show (me.getComponent (), me.getX (), me.getY ());

}

}

);

//Creating the ToolBar's Buttons of Program.

btnNew = new JButton (new ImageIcon ("Images/NotePad.gif"));

btnNew.setToolTipText ("Create New Account");

btnNew.addActionListener (this);

btnDep = new JButton (new ImageIcon ("Images/ImationDisk.gif"));

btnDep.setToolTipText ("Deposit Money");

btnDep.addActionListener (this);

btnWith = new JButton (new ImageIcon ("Images/SuperDisk.gif"));

btnWith.setToolTipText ("Withdraw Money");

btnWith.addActionListener (this);

btnRec = new JButton (new ImageIcon ("Images/Paproll.gif"));

btnRec.setToolTipText ("Print Customer Balance");

btnRec.addActionListener (this);

btnDel = new JButton (new ImageIcon ("Images/Toaster.gif"));

btnDel.setToolTipText ("Delete Customer");

btnDel.addActionListener (this);

btnSrch = new JButton (new ImageIcon ("Images/Search.gif"));

btnSrch.setToolTipText ("Search Customer");

btnSrch.addActionListener (this);

btnHelp = new JButton (new ImageIcon ("Images/Help.gif"));

btnHelp.setToolTipText ("Help on Bank System");

btnHelp.addActionListener (this);

btnKey = new JButton (new ImageIcon ("Images/Keys.gif"));

btnKey.setToolTipText ("Shortcut Keys of BankSystem");

btnKey.addActionListener (this);

//Creating the ToolBar of Program.

toolBar = new JToolBar ();

toolBar.add (btnNew);

toolBar.addSeparator ();

toolBar.add (btnDep);

toolBar.add (btnWith);

toolBar.addSeparator ();

toolBar.add (btnRec);

toolBar.addSeparator ();

toolBar.add (btnDel);

toolBar.addSeparator ();

toolBar.add (btnSrch);

toolBar.addSeparator ();

toolBar.add (btnHelp);

toolBar.add (btnKey);

//Creating the StatusBar of Program.

author = new JLabel (" " + "BankSystem [Pvt] Limited.", Label.LEFT);

author.setForeground (Color.black);

author.setToolTipText ("Program's Title");

welcome = new JLabel ("Welcome Today is " + d + " ", JLabel.RIGHT);

welcome.setForeground (Color.black);

welcome.setToolTipText ("Welcoming the User & System Current Date");

statusBar.setLayout (new BorderLayout());

statusBar.add (author, BorderLayout.WEST);

statusBar.add (welcome, BorderLayout.EAST);

//For Making the Dragging Speed of Internal Frames Faster.

desktop.putClientProperty ("JDesktopPane.dragMode", "outline");

//Setting the Contents of Programs.

getContentPane().add (toolBar, BorderLayout.NORTH);

getContentPane().add (desktop, BorderLayout.CENTER);

getContentPane().add (statusBar, BorderLayout.SOUTH);

//Showing The Main Form of Application.

setVisible (true);

}

//Function For Performing different Actions By Menus of Program.

public void actionPerformed (ActionEvent ae) {

Object obj = ae.getSource();

if (obj == addNew || obj == open || obj == btnNew) {

boolean b = openChildWindow ("Create New Account");

if (b == false) {

NewAccount newAcc = new NewAccount ();

desktop.add (newAcc);

newAcc.show ();

}

}

else if (obj == printRec || obj == btnRec || obj == report) {

getAccountNo ();

}

else if (obj == end) {

quitApp ();

}

else if (obj == deposit || obj == dep || obj == btnDep) {

boolean b = openChildWindow ("Deposit Money");

if (b == false) {

DepositMoney depMon = new DepositMoney ();

desktop.add (depMon);

depMon.show ();

}

}

else if (obj == withdraw || obj == with || obj == btnWith) {

boolean b = openChildWindow ("Withdraw Money");

if (b == false) {

WithdrawMoney withMon = new WithdrawMoney ();

desktop.add (withMon);

withMon.show ();

}

}

else if (obj == delRec || obj == del || obj == btnDel) {

boolean b = openChildWindow ("Delete Account Holder");

if (b == false) {

DeleteCustomer delCus = new DeleteCustomer ();

desktop.add (delCus);

delCus.show ();

}

}

else if (obj == search || obj == find || obj == btnSrch) {

boolean b = openChildWindow ("Search Customer [By No.]");

if (b == false) {

FindAccount fndAcc = new FindAccount ();

desktop.add (fndAcc);

fndAcc.show ();

}

}

else if (obj == searchName) {

boolean b = openChildWindow ("Search Customer [By Name]");

if (b == false) {

FindName fndNm = new FindName ();

desktop.add (fndNm);

fndNm.show ();

}

}

else if (obj == oneByOne) {

boolean b = openChildWindow ("View Account Holders");

if (b == false) {

ViewOne vwOne = new ViewOne ();

desktop.add (vwOne);

vwOne.show ();

}

}

else if (obj == allCustomer || obj == all) {

boolean b = openChildWindow ("View All Account Holders");

if (b == false) {

ViewCustomer viewCus = new ViewCustomer ();

desktop.add (viewCus);

viewCus.show ();

}

}

else if (obj == change) {

Color cl = new Color (153, 153, 204);

cl = JColorChooser.showDialog (this, "Choose Background Color", cl);

if (cl == null) { }

else {

desktop.setBackground (cl);

desktop.repaint ();

}

}

else if (obj == close) {

try {

desktop.getSelectedFrame().setClosed(true);

}

catch (Exception CloseExc) { }

}

else if (obj == closeAll) {

JInternalFrame Frames[] = desktop.getAllFrames (); //Getting all Open Frames.

for(int getFrameLoop = 0; getFrameLoop < Frames.length; getFrameLoop++) {

try {

Frames[getFrameLoop].setClosed (true); //Close the frame.

}

catch (Exception CloseExc) { } //if we can't close it then we have a problem.

}

}

else if (obj == content || obj == btnHelp) {

boolean b = openChildWindow ("BankSystem Help");

if (b == false) {

BankHelp hlpBank = new BankHelp ("BankSystem Help", "Help/Bank.htm");

desktop.add (hlpBank);

hlpBank.show ();

}

}

else if (obj == keyHelp || obj == btnKey) {

boolean b = openChildWindow ("BankSystem Keys");

if (b == false) {

BankHelp hlpKey = new BankHelp ("BankSystem Keys", "Help/Keys.htm");

desktop.add (hlpKey);

hlpKey.show ();

}

}

else if (obj == about) {

String msg = "BankSystem [Pvt] Limited.\n\n" + "Created & Designed By:\n" +

"Siddhartha Rastogi\n\n" + "E-mail me:\n sid91rastogi@gmail.com";

JOptionPane.showMessageDialog (this, msg, "About BankSystem", JOptionPane.PLAIN\_MESSAGE);

}

}

//Function Perform By LookAndFeel Menu.

public void itemStateChanged (ItemEvent e) {

for( int i = 0; i < radio.length; i++ )

if(radio[i].isSelected()) {

changeLookAndFeel (i);

}

}

//Function For Closing the Program.

private void quitApp () {

try {

//Show a Confirmation Dialog.

int reply = JOptionPane.showConfirmDialog (this,

"Are you really want to exit\nFrom BankSystem?",

"BankSystem - Exit", JOptionPane.YES\_NO\_OPTION, JOptionPane.PLAIN\_MESSAGE);

//Check the User Selection.

if (reply == JOptionPane.YES\_OPTION) {

setVisible (false); //Hide the Frame.

dispose(); //Free the System Resources.

System.out.println ("Thanks for Using BankSystem\nAuthor - Siddhartha Rastogi");

System.exit (0); //Close the Application.

}

else if (reply == JOptionPane.NO\_OPTION) {

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

}

}

catch (Exception e) {}

}

//Function for Changing the Program's Look.

public void changeLookAndFeel (int val) {

try {

UIManager.setLookAndFeel (looks[val].getClassName());

SwingUtilities.updateComponentTreeUI (this);

}

catch (Exception e) { }

}

//Loop Through All the Opened JInternalFrame to Perform the Task.

private boolean openChildWindow (String title) {

JInternalFrame[] childs = desktop.getAllFrames ();

for (int i = 0; i < childs.length; i++) {

if (childs[i].getTitle().equalsIgnoreCase (title)) {

childs[i].show ();

return true;

}

}

return false;

}

//Following Functions use for Printing Records & Report of BankSystem.

void getAccountNo () {

String printing;

rows = 0;

boolean b = populateArray ();

if (b == false) { }

else {

try {

printing = JOptionPane.showInputDialog (this, "Enter Account No. to Print Customer Balance.\n" +

"(Tip: Account No. Contains only Digits)", "BankSystem - PrintRecord", JOptionPane.PLAIN\_MESSAGE);

if (printing == null) { }

if (printing.equals ("")) {

JOptionPane.showMessageDialog (this, "Provide Account No. to Print.",

"BankSystem - EmptyField", JOptionPane.PLAIN\_MESSAGE);

getAccountNo ();

}

else {

findRec (printing);

}

}

catch (Exception e) { }

}

}

//Function use to load all Records from File when Application Execute.

boolean populateArray () {

boolean b = false;

try {

fis = new FileInputStream ("Bank.dat");

dis = new DataInputStream (fis);

//Loop to Populate the Array.

while (true) {

for (int i = 0; i < 6; i++) {

records[rows][i] = dis.readUTF ();

}

rows++;

}

}

catch (Exception ex) {

total = rows;

if (total == 0) {

JOptionPane.showMessageDialog (null, "Records File is Empty.\nEnter Records First to Display.",

"BankSystem - EmptyFile", JOptionPane.PLAIN\_MESSAGE);

b = false;

}

else {

b = true;

try {

dis.close();

fis.close();

}

catch (Exception exp) { }

}

}

return b;

}

//Function use to Find Record by Matching the Contents of Records Array with InputBox.

void findRec (String rec) {

boolean found = false;

for (int x = 0; x < total; x++) {

if (records[x][0].equals (rec)) {

found = true;

printRecord (makeRecordPrint (x));

break;

}

}

if (found == false) {

JOptionPane.showMessageDialog (this, "Account No. " + rec + " doesn't Exist.",

"BankSystem - WrongNo", JOptionPane.PLAIN\_MESSAGE);

getAccountNo ();

}

}

//Function use to make Current Record ready for Print.

String makeRecordPrint (int rec) {

String data;

String data0 = " BankSystem [Pvt] Limited. \n"; //Page Title.

String data1 = " Customer Balance Report. \n\n"; //Page Header.

String data2 = " Account No.: " + records[rec][0] + "\n";

String data3 = " Customer Name: " + records[rec][1] + "\n";

String data4 = " Last Transaction: " + records[rec][2] + ", " + records[rec][3] + ", " + records[rec][4] + "\n";

String data5 = " Current Balance: " + records[rec][5] + "\n\n";

String data6 = " Copyright © 2003 Siddhartha Rastogi.\n"; //Page Footer.

String sep0 = " -----------------------------------------------------------\n";

String sep1 = " -----------------------------------------------------------\n";

String sep2 = " -----------------------------------------------------------\n";

String sep3 = " -----------------------------------------------------------\n";

String sep4 = " -----------------------------------------------------------\n\n";

data = data0 + sep0 + data1 + data2 + sep1 + data3 + sep2 + data4 + sep3 + data5 + sep4 + data6;

return data;

}

//Function use to Print the Current Record.

void printRecord (String rec) {

StringReader sr = new StringReader (rec);

LineNumberReader lnr = new LineNumberReader (sr);

Font typeface = new Font ("Times New Roman", Font.PLAIN, 12);

Properties p = new Properties ();

PrintJob pJob = getToolkit().getPrintJob (this, "Print Customer Balance Report", p);

if (pJob != null) {

Graphics gr = pJob.getGraphics ();

if (gr != null) {

FontMetrics fm = gr.getFontMetrics (typeface);

int margin = 20;

int pageHeight = pJob.getPageDimension().height - margin;

int fontHeight = fm.getHeight();

int fontDescent = fm.getDescent();

int curHeight = margin;

String nextLine;

gr.setFont (typeface);

try {

do {

nextLine = lnr.readLine ();

if (nextLine != null) {

if ((curHeight + fontHeight) > pageHeight) { //New Page.

gr.dispose();

gr = pJob.getGraphics ();

curHeight = margin;

}

curHeight += fontHeight;

if (gr != null) {

gr.setFont (typeface);

gr.drawString (nextLine, margin, curHeight - fontDescent);

}

}

}

while (nextLine != null);

}

catch (EOFException eof) { }

catch (Throwable t) { }

}

gr.dispose();

}

if (pJob != null)

pJob.end ();

}

}

**To run the main function**

import java.awt.\*;

import javax.swing.\*;

import javax.swing.border.LineBorder;

public class Splash extends JWindow {

private Dimension d = Toolkit.getDefaultToolkit().getScreenSize();

public Splash () {

JLabel lbImage = new JLabel (new ImageIcon ("Splash.jpg"));

Color cl = new Color (0, 0, 0);

lbImage.setBorder (new LineBorder (cl, 1));

getContentPane().add (lbImage, BorderLayout.CENTER);

pack();

setSize (getSize().width, getSize().height);

setLocation (d.width / 2 - getWidth() / 2, d.height / 2 - getHeight() / 2);

show();

for (int i = 0; i <= 1000; i++) { }

new BankSystem ();

toFront();

dispose ();

setVisible (false);

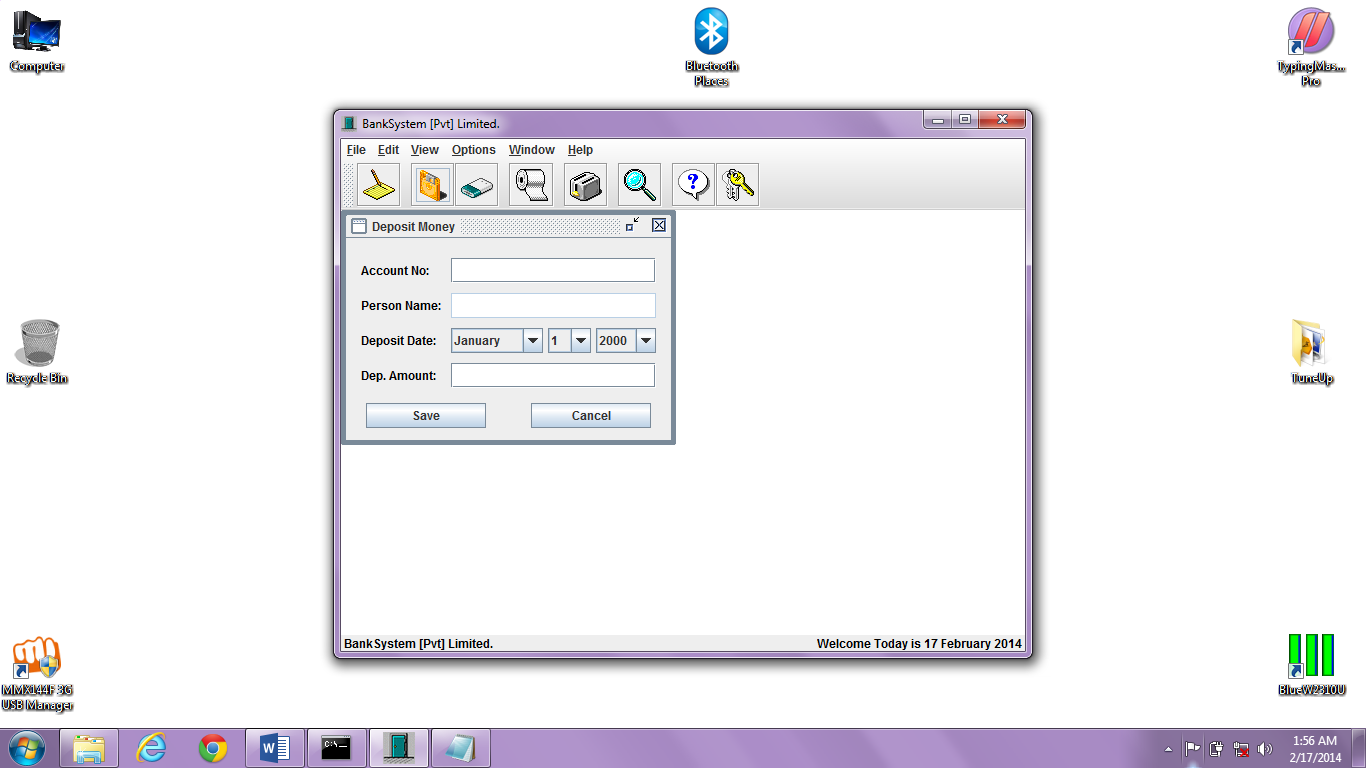
}

public static void main (String args[]) {

new Splash ();

}

}



**Withdrawal money**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.io.\*;

public class WithdrawMoney extends JInternalFrame implements ActionListener {

private JPanel jpWith = new JPanel();

private JLabel lbNo, lbName, lbDate, lbWithdraw;

private JTextField txtNo, txtName, txtWithdraw;

private JComboBox cboMonth, cboDay, cboYear;

private JButton btnSave, btnCancel;

private int recCount = 0;

private int rows = 0;

private int total = 0;

private int curr;

private int withdraw;

//String Type Array use to Load Records From File.

private String records[][] = new String [500][6];

private FileInputStream fis;

private DataInputStream dis;

WithdrawMoney () {

// super(Title, Resizable, Closable, Maximizable, Iconifiable)

super ("Withdraw Money", false, true, false, true);

setSize (335, 235);

jpWith.setLayout (null);

lbNo = new JLabel ("Account No:");

lbNo.setForeground (Color.black);

lbNo.setBounds (15, 20, 80, 25);

lbName = new JLabel ("Person Name:");

lbName.setForeground (Color.black);

lbName.setBounds (15, 55, 80, 25);

lbDate = new JLabel ("With. Date:");

lbDate.setForeground (Color.black);

lbDate.setBounds (15, 90, 80, 25);

lbWithdraw = new JLabel ("With. Amount:");

lbWithdraw.setForeground (Color.black);

lbWithdraw.setBounds (15, 125, 80, 25);

txtNo = new JTextField ();

txtNo.setHorizontalAlignment (JTextField.RIGHT);

//Checking the Accunt No. Provided By User on Lost Focus of the TextBox.

txtNo.addFocusListener (new FocusListener () {

public void focusGained (FocusEvent e) { }

public void focusLost (FocusEvent fe) {

if (txtNo.getText().equals ("")) { }

else {

rows = 0;

populateArray (); //Load All Existing Records in Memory.

findRec (); //Finding if Account No. Already Exist or Not.

}

}

}

);

txtNo.setBounds (105, 20, 205, 25);

txtName = new JTextField ();

txtName.setEnabled (false);

txtName.setBounds (105, 55, 205, 25);

txtWithdraw = new JTextField ();

txtWithdraw.setHorizontalAlignment (JTextField.RIGHT);

txtWithdraw.setBounds (105, 125, 205, 25);

//Restricting The User Input to only Numerics in Numeric TextBoxes.

txtNo.addKeyListener (new KeyAdapter() {

public void keyTyped (KeyEvent ke) {

char c = ke.getKeyChar ();

if (!((Character.isDigit (c) || (c == KeyEvent.VK\_BACK\_SPACE)))) {

getToolkit().beep ();

ke.consume ();

}

}

}

);

txtWithdraw.addKeyListener (new KeyAdapter() {

public void keyTyped (KeyEvent ke) {

char c = ke.getKeyChar ();

if (!((Character.isDigit (c) || (c == KeyEvent.VK\_BACK\_SPACE)))) {

getToolkit().beep ();

ke.consume ();

}

}

}

);

//Creating Date Option.

String Months[] = {"January", "February", "March", "April", "May", "June",

"July", "August", "September", "October", "November", "December"};

cboMonth = new JComboBox (Months);

cboDay = new JComboBox ();

cboYear = new JComboBox ();

for (int i = 1; i <= 31; i++) {

String days = "" + i;

cboDay.addItem (days);

}

for (int i = 2000; i <= 2015; i++) {

String years = "" + i;

cboYear.addItem (years);

}

//Aligning The Date Option Controls.

cboMonth.setBounds (105, 90, 92, 25);

cboDay.setBounds (202, 90, 43, 25);

cboYear.setBounds (250, 90, 60, 25);

//Aligning The Buttons.

btnSave = new JButton ("Save");

btnSave.setBounds (20, 165, 120, 25);

btnSave.addActionListener (this);

btnCancel = new JButton ("Cancel");

btnCancel.setBounds (185, 165, 120, 25);

btnCancel.addActionListener (this);

//Adding the All the Controls to Panel.

jpWith.add (lbNo);

jpWith.add (txtNo);

jpWith.add (lbName);

jpWith.add (txtName);

jpWith.add (lbDate);

jpWith.add (cboMonth);

jpWith.add (cboDay);

jpWith.add (cboYear);

jpWith.add (lbWithdraw);

jpWith.add (txtWithdraw);

jpWith.add (btnSave);

jpWith.add (btnCancel);

//Adding Panel to Window.

getContentPane().add (jpWith);

populateArray (); //Load All Existing Records in Memory.

//In the End Showing the New Account Window.

setVisible (true);

}

//Function use By Buttons of Window to Perform Action as User Click Them.

public void actionPerformed (ActionEvent ae) {

Object obj = ae.getSource();

if (obj == btnSave) {

if (txtNo.getText().equals("")) {

JOptionPane.showMessageDialog (this, "Please! Provide Id of Customer.",

"BankSystem - EmptyField", JOptionPane.PLAIN\_MESSAGE);

txtNo.requestFocus();

}

else if (txtWithdraw.getText().equals("")) {

JOptionPane.showMessageDialog (this, "Please! Provide Withdraw Amount.",

"BankSystem - EmptyField", JOptionPane.PLAIN\_MESSAGE);

txtWithdraw.requestFocus ();

}

else {

withdraw = Integer.parseInt (txtWithdraw.getText ());

if (curr == 0) {

JOptionPane.showMessageDialog (this, txtName.getText () + " doesn't have any Amount in Balance.",

"BankSystem - EmptyAccount", JOptionPane.PLAIN\_MESSAGE);

txtClear ();

}

else if (withdraw > curr) {

JOptionPane.showMessageDialog (this, "Withdraw Amount can't greater than Actual Balance.",

"BankSystem - Large Amount", JOptionPane.PLAIN\_MESSAGE);

txtWithdraw.setText ("");

txtWithdraw.requestFocus ();

}

else {

editRec (); //Update the Contents of Array.

}

}

}

if (obj == btnCancel) {

txtClear ();

setVisible (false);

dispose();

}

}

//Function use to load all Records from File when Application Execute.

void populateArray () {

try {

fis = new FileInputStream ("Bank.dat");

dis = new DataInputStream (fis);

//Loop to Populate the Array.

while (true) {

for (int i = 0; i < 6; i++) {

records[rows][i] = dis.readUTF ();

}

rows++;

}

}

catch (Exception ex) {

total = rows;

if (total == 0) {

JOptionPane.showMessageDialog (null, "Records File is Empty.\nEnter Records First to Display.",

"BankSystem - EmptyFile", JOptionPane.PLAIN\_MESSAGE);

btnEnable ();

}

else {

try {

dis.close();

fis.close();

}

catch (Exception exp) { }

}

}

}

//Function use to Find Record by Matching the Contents of Records Array with ID TextBox.

void findRec () {

boolean found = false;

for (int x = 0; x < total; x++) {

if (records[x][0].equals (txtNo.getText())) {

found = true;

showRec (x);

break;

}

}

if (found == false) {

String str = txtNo.getText ();

txtClear ();

JOptionPane.showMessageDialog (this, "Account No. " + str + " doesn't Exist.",

"BankSystem - WrongNo", JOptionPane.PLAIN\_MESSAGE);

}

}

//Function which display Record from Array onto the Form.

public void showRec (int intRec) {

txtNo.setText (records[intRec][0]);

txtName.setText (records[intRec][1]);

curr = Integer.parseInt (records[intRec][5]);

recCount = intRec;

}

//Function use to Clear all TextFields of Window.

void txtClear () {

txtNo.setText ("");

txtName.setText ("");

txtWithdraw.setText ("");

txtNo.requestFocus ();

}

//Function use to Edit an Element's Value of the Array.

public void editRec () {

records[recCount][0] = txtNo.getText ();

records[recCount][1] = txtName.getText ();

records[recCount][2] = "" + cboMonth.getSelectedItem ();

records[recCount][3] = "" + cboDay.getSelectedItem ();

records[recCount][4] = "" + cboYear.getSelectedItem ();

records[recCount][5] = "" + (curr - withdraw);

editFile (); //Save This Array to File.

}

//Function use to Save Records to File After editing the Record of User Choice.

public void editFile () {

try {

FileOutputStream fos = new FileOutputStream ("Bank.dat");

DataOutputStream dos = new DataOutputStream (fos);

if (records != null) {

for (int i = 0; i < total; i++) {

for (int c = 0; c < 6; c++) {

dos.writeUTF (records[i][c]);

if (records[i][c] == null) break;

}

}

JOptionPane.showMessageDialog (this, "The File is Updated Successfully",

"BankSystem - Record Saved", JOptionPane.PLAIN\_MESSAGE);

txtClear ();

dos.close();

fos.close();

}

}

catch (IOException ioe) {

JOptionPane.showMessageDialog (this, "There are Some Problem with File",

"BankSystem - Problem", JOptionPane.PLAIN\_MESSAGE);

}

}

//Function use to Lock all Buttons of Window.

void btnEnable () {

txtNo.setEnabled (false);

cboMonth.setEnabled (false);

cboDay.setEnabled (false);

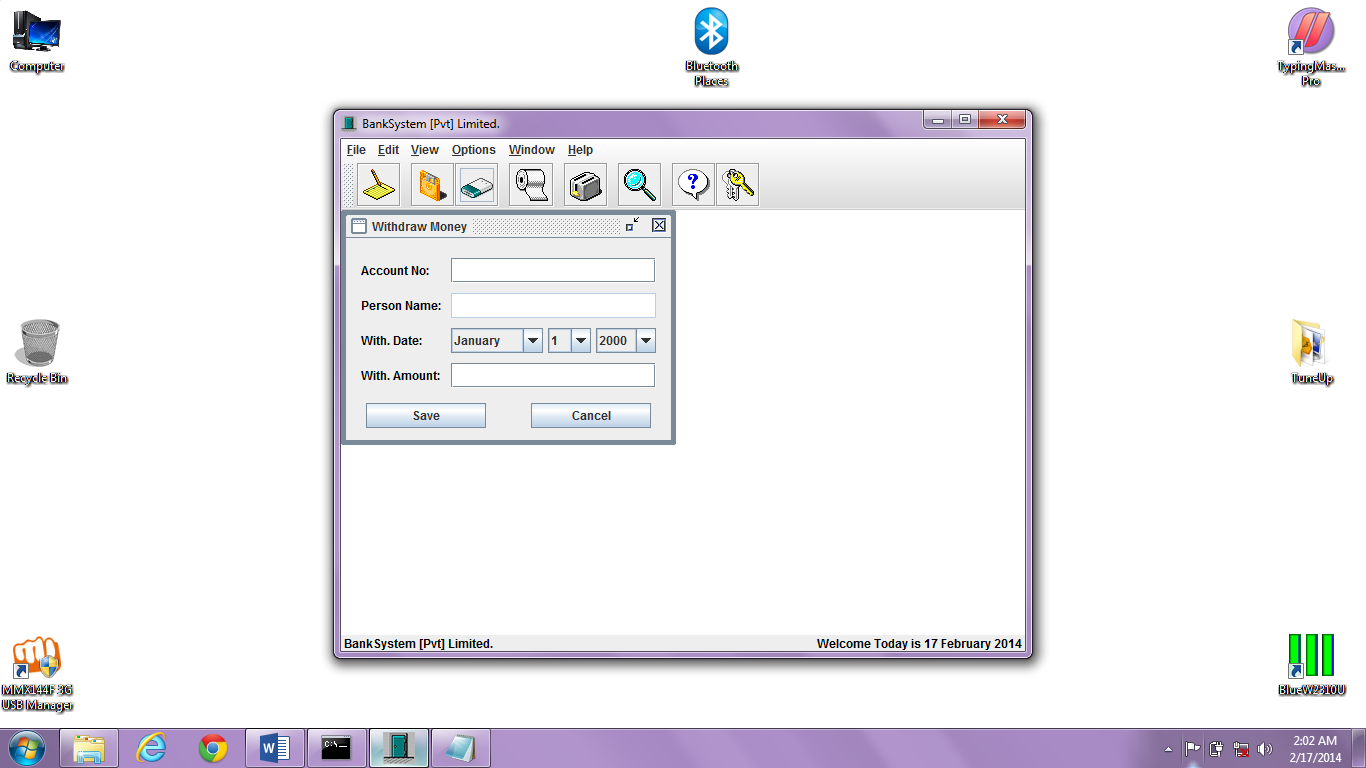
cboYear.setEnabled (false);

txtWithdraw.setEnabled (false);

btnSave.setEnabled (false);

}

}



**Delete customer**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.io.\*;

public class DeleteCustomer extends JInternalFrame implements ActionListener {

private JPanel jpDel = new JPanel();

private JLabel lbNo, lbName, lbDate, lbBal;

private JTextField txtNo, txtName, txtDate, txtBal;

private JButton btnDel, btnCancel;

private int recCount = 0;

private int rows = 0;

private int total = 0;

//String Type Array use to Load Records From File.

private String records[][] = new String [500][6];

private FileInputStream fis;

private DataInputStream dis;

DeleteCustomer () {

// super(Title, Resizable, Closable, Maximizable, Iconifiable)

super ("Delete Account Holder", false, true, false, true);

setSize (350, 235);

jpDel.setLayout (null);

lbNo = new JLabel ("Account No:");

lbNo.setForeground (Color.black);

lbNo.setBounds (15, 20, 80, 25);

lbName = new JLabel ("Person Name:");

lbName.setForeground (Color.black);

lbName.setBounds (15, 55, 90, 25);

lbDate = new JLabel ("Last Transaction:");

lbDate.setForeground (Color.black);

lbDate.setBounds (15, 90, 100, 25);

lbBal = new JLabel ("Balance:");

lbBal.setForeground (Color.black);

lbBal.setBounds (15, 125, 80, 25);

txtNo = new JTextField ();

txtNo.setHorizontalAlignment (JTextField.RIGHT);

txtNo.setBounds (125, 20, 200, 25);

txtName = new JTextField ();

txtName.setEnabled (false);

txtName.setBounds (125, 55, 200, 25);

txtDate = new JTextField ();

txtDate.setEnabled (false);

txtDate.setBounds (125, 90, 200, 25);

txtBal = new JTextField ();

txtBal.setEnabled (false);

txtBal.setHorizontalAlignment (JTextField.RIGHT);

txtBal.setBounds (125, 125, 200, 25);

//Aligning The Buttons.

btnDel = new JButton ("Delete");

btnDel.setBounds (20, 165, 120, 25);

btnDel.addActionListener (this);

btnCancel = new JButton ("Cancel");

btnCancel.setBounds (200, 165, 120, 25);

btnCancel.addActionListener (this);

//Adding the All the Controls to Panel.

jpDel.add (lbNo);

jpDel.add (txtNo);

jpDel.add (lbName);

jpDel.add (txtName);

jpDel.add (lbDate);

jpDel.add (txtDate);

jpDel.add (lbBal);

jpDel.add (txtBal);

jpDel.add (btnDel);

jpDel.add (btnCancel);

//Restricting The User Input to only Numerics in Numeric TextBoxes.

txtNo.addKeyListener (new KeyAdapter() {

public void keyTyped (KeyEvent ke) {

char c = ke.getKeyChar ();

if (!((Character.isDigit (c) || (c == KeyEvent.VK\_BACK\_SPACE)))) {

getToolkit().beep ();

ke.consume ();

}

}

}

);

//Checking the Accunt No. Provided By User on Lost Focus of the TextBox.

txtNo.addFocusListener (new FocusListener () {

public void focusGained (FocusEvent e) { }

public void focusLost (FocusEvent fe) {

if (txtNo.getText().equals ("")) { }

else {

rows = 0;

populateArray (); //Load All Existing Records in Memory.

findRec (); //Finding if Account No. Already Exist or Not.

}

}

}

);

//Adding Panel to Window.

getContentPane().add (jpDel);

populateArray (); //Load All Existing Records in Memory.

//In the End Showing the New Account Window.

setVisible (true);

}

//Function use By Buttons of Window to Perform Action as User Click Them.

public void actionPerformed (ActionEvent ae) {

Object obj = ae.getSource();

if (obj == btnDel) {

if (txtNo.getText().equals("")) {

JOptionPane.showMessageDialog (this, "Please! Provide Id of Customer.",

"BankSystem - EmptyField", JOptionPane.PLAIN\_MESSAGE);

txtNo.requestFocus();

}

else {

deletion (); //Confirm Deletion of Current Record.

}

}

if (obj == btnCancel) {

txtClear ();

setVisible (false);

dispose();

}

}

//Function use to load all Records from File when Application Execute.

void populateArray () {

try {

fis = new FileInputStream ("Bank.dat");

dis = new DataInputStream (fis);

//Loop to Populate the Array.

while (true) {

for (int i = 0; i < 6; i++) {

records[rows][i] = dis.readUTF ();

}

rows++;

}

}

catch (Exception ex) {

total = rows;

if (total == 0) {

JOptionPane.showMessageDialog (null, "Records File is Empty.\nEnter Records First to Display.",

"BankSystem - EmptyFile", JOptionPane.PLAIN\_MESSAGE);

btnEnable ();

}

else {

try {

dis.close();

fis.close();

}

catch (Exception exp) { }

}

}

}

//Function use to Find Record by Matching the Contents of Records Array with ID TextBox.

void findRec () {

boolean found = false;

for (int x = 0; x < total; x++) {

if (records[x][0].equals (txtNo.getText())) {

found = true;

showRec (x);

break;

}

}

if (found == false) {

String str = txtNo.getText ();

txtClear ();

JOptionPane.showMessageDialog (this, "Account No. " + str + " doesn't Exist.",

"BankSystem - WrongNo", JOptionPane.PLAIN\_MESSAGE);

}

}

//Function which display Record from Array onto the Form.

void showRec (int intRec) {

txtNo.setText (records[intRec][0]);

txtName.setText (records[intRec][1]);

txtDate.setText (records[intRec][2] + ", " + records[intRec][3] + ", " + records[intRec][4]);

txtBal.setText (records[intRec][5]);

recCount = intRec;

}

//Confirming the Deletion Decision made By User of Program.

void deletion () {

try {

//Show a Confirmation Dialog.

int reply = JOptionPane.showConfirmDialog (this,

"Are you Sure you want to Delete\n" + txtName.getText () + " Record From BankSystem?",

"Bank System - Delete", JOptionPane.YES\_NO\_OPTION, JOptionPane.PLAIN\_MESSAGE);

//Check the User Selection.

if (reply == JOptionPane.YES\_OPTION) {

delRec (); //Delete the Selected Contents of Array.

}

else if (reply == JOptionPane.NO\_OPTION) { }

}

catch (Exception e) {}

}

//Function use to Delete an Element from the Array.

void delRec () {

try {

if (records != null) {

for(int i = recCount; i < total; i++) {

for (int r = 0; r < 6; r++) {

records[i][r] = records[i+1][r];

if (records[i][r] == null) break;

}

}

total = total - 1;

deleteFile ();

}

}

catch (ArrayIndexOutOfBoundsException ex) { }

}

//Function use to Save Records to File After Deleting the Record of User Choice.

void deleteFile () {

try {

FileOutputStream fos = new FileOutputStream ("Bank.dat");

DataOutputStream dos = new DataOutputStream (fos);

if (records != null) {

for (int i = 0; i < total; i++) {

for (int r = 0; r < 6; r++) {

dos.writeUTF (records[i][r]);

if (records[i][r] == null) break;

}

}

JOptionPane.showMessageDialog (this, "Record has been Deleted Successfuly.",

"BankSystem - Record Deleted", JOptionPane.PLAIN\_MESSAGE);

txtClear ();

}

else { }

dos.close();

fos.close();

}

catch (IOException ioe) {

JOptionPane.showMessageDialog (this, "There are Some Problem with File",

"BankSystem - Problem", JOptionPane.PLAIN\_MESSAGE);

}

}

//Function use to Clear all TextFields of Window.

void txtClear () {

txtNo.setText ("");

txtName.setText ("");

txtDate.setText ("");

txtBal.setText ("");

txtNo.requestFocus ();

}

//Function use to Lock Controls of Window.

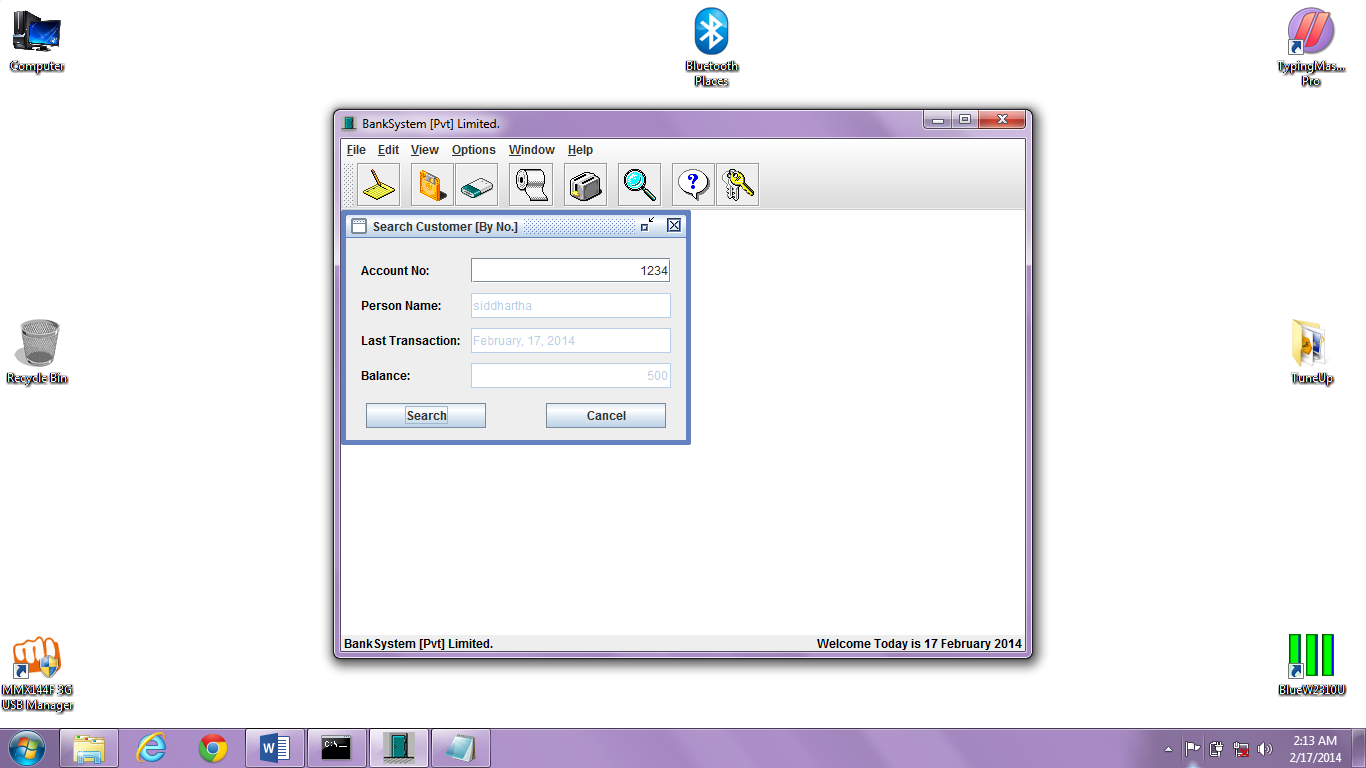
void btnEnable () {

txtNo.setEnabled (false);

btnDel.setEnabled (false);

}

}



**Search account by name**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.io.\*;

public class FindName extends JInternalFrame implements ActionListener {

private JPanel jpFind = new JPanel();

private JLabel lbNo, lbName, lbDate, lbBal;

private JTextField txtNo, txtName, txtDate, txtBal;

private JButton btnFind, btnCancel;

private int count = 0;

private int rows = 0;

private int total = 0;

//String Type Array use to Load Records From File.

private String records[][] = new String [500][6];

private FileInputStream fis;

private DataInputStream dis;

FindName () {

//super(Title, Resizable, Closable, Maximizable, Iconifiable)

super ("Search Customer [By Name]", false, true, false, true);

setSize (350, 235);

jpFind.setLayout (null);

lbNo = new JLabel ("Account No:");

lbNo.setForeground (Color.black);

lbNo.setBounds (15, 20, 80, 25);

lbName = new JLabel ("Person Name:");

lbName.setForeground (Color.black);

lbName.setBounds (15, 55, 80, 25);

lbDate = new JLabel ("Last Transaction:");

lbDate.setForeground (Color.black);

lbDate.setBounds (15, 90, 100, 25);

lbBal = new JLabel ("Balance:");

lbBal.setForeground (Color.black);

lbBal.setBounds (15, 125, 80, 25);

txtNo = new JTextField ();

txtNo.setHorizontalAlignment (JTextField.RIGHT);

txtNo.setEnabled (false);

txtNo.setBounds (125, 20, 200, 25);

txtName = new JTextField ();

txtName.setBounds (125, 55, 200, 25);

txtDate = new JTextField ();

txtDate.setEnabled (false);

txtDate.setBounds (125, 90, 200, 25);

txtBal = new JTextField ();

txtBal.setHorizontalAlignment (JTextField.RIGHT);

txtBal.setEnabled (false);

txtBal.setBounds (125, 125, 200, 25);

//Aligning The Buttons.

btnFind = new JButton ("Search");

btnFind.setBounds (20, 165, 120, 25);

btnFind.addActionListener (this);

btnCancel = new JButton ("Cancel");

btnCancel.setBounds (200, 165, 120, 25);

btnCancel.addActionListener (this);

//Adding the All the Controls to Panel.

jpFind.add (lbNo);

jpFind.add (txtNo);

jpFind.add (lbName);

jpFind.add (txtName);

jpFind.add (lbDate);

jpFind.add (txtDate);

jpFind.add (lbBal);

jpFind.add (txtBal);

jpFind.add (btnFind);

jpFind.add (btnCancel);

//Adding Panel to Window.

getContentPane().add (jpFind);

populateArray (); //Load All Existing Records in Memory.

//In the End Showing the New Account Window.

setVisible (true);

}

//Function use By Buttons of Window to Perform Action as User Click Them.

public void actionPerformed (ActionEvent ae) {

Object obj = ae.getSource();

if (obj == btnFind) {

if (txtName.getText().equals ("")) {

JOptionPane.showMessageDialog (this, "Please! Provide Name of Customer to Search.",

"BankSystem - EmptyField", JOptionPane.PLAIN\_MESSAGE);

txtName.requestFocus();

}

else {

rows = 0;

populateArray (); //Load All Existing Records in Memory.

findRec (); //Finding if Account No. Exist or Not.

}

}

if (obj == btnCancel) {

txtClear ();

setVisible (false);

dispose();

}

}

//Function use to load all Records from File when Application Execute.

void populateArray () {

try {

fis = new FileInputStream ("Bank.dat");

dis = new DataInputStream (fis);

//Loop to Populate the Array.

while (true) {

for (int i = 0; i < 6; i++) {

records[rows][i] = dis.readUTF ();

}

rows++;

}

}

catch (Exception ex) {

total = rows;

if (total == 0) {

JOptionPane.showMessageDialog (null, "Records File is Empty.\nEnter Records First to Display.",

"BankSystem - EmptyFile", JOptionPane.PLAIN\_MESSAGE);

btnEnable ();

}

else {

try {

dis.close();

fis.close();

}

catch (Exception exp) { }

}

}

}

//Function use to Find Record by Matching the Contents of Records Array with ID TextBox.

void findRec () {

boolean found = false;

for (int x = 0; x < total; x++) {

if (records[x][1].equalsIgnoreCase (txtName.getText())) {

found = true;

showRec (x);

break;

}

}

if (found == false) {

JOptionPane.showMessageDialog (this, "Customer " + txtName.getText () + " doesn't Exist.",

"BankSystem - WrongNo", JOptionPane.PLAIN\_MESSAGE);

txtClear ();

}

}

//Function which display Record from Array onto the Form.

public void showRec (int intRec) {

txtNo.setText (records[intRec][0]);

txtName.setText (records[intRec][1]);

txtDate.setText (records[intRec][2] + ", " + records[intRec][3] + ", " + records[intRec][4]);

txtBal.setText (records[intRec][5]);

}

//Function use to Clear all TextFields of Window.

void txtClear () {

txtNo.setText ("");

txtName.setText ("");

txtDate.setText ("");

txtBal.setText ("");

txtName.requestFocus ();

}

//Function use to Lock Controls of Window.

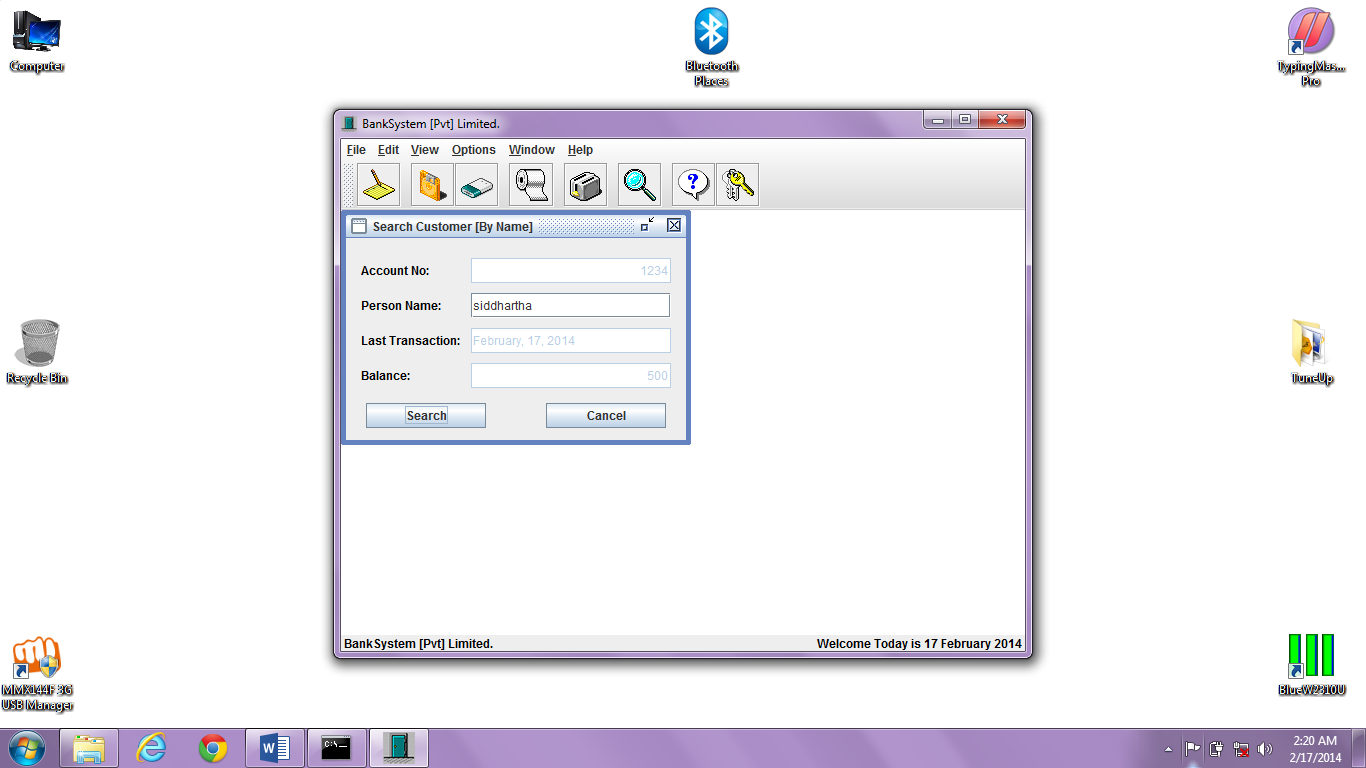
void btnEnable () {

txtName.setEnabled (false);

btnFind.setEnabled (false);

}

}



**View Account holder**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.io.\*;

public class ViewOne extends JInternalFrame implements ActionListener {

private JPanel jpRec = new JPanel();

private JLabel lbNo, lbName, lbDate, lbBal;

private JTextField txtNo, txtName, txtDate, txtBal, txtRec;

private JButton btnFirst, btnBack, btnNext, btnLast;

private int recCount = 0;

private int rows = 0;

private int total = 0;

//String Type Array use to Load Records From File.

private String records[][] = new String [500][6];

private FileInputStream fis;

private DataInputStream dis;

ViewOne () {

// super(Title, Resizable, Closable, Maximizable, Iconifiable)

super ("View Account Holders", false, true, false, true);

setSize (350, 235);

jpRec.setLayout (null);

lbNo = new JLabel ("Account No:");

lbNo.setForeground (Color.black);

lbNo.setBounds (15, 20, 80, 25);

lbName = new JLabel ("Person Name:");

lbName.setForeground (Color.black);

lbName.setBounds (15, 55, 80, 25);

lbDate = new JLabel ("Last Transaction:");

lbDate.setForeground (Color.black);

lbDate.setBounds (15, 90, 100, 25);

lbBal = new JLabel ("Balance:");

lbBal.setForeground (Color.black);

lbBal.setBounds (15, 125, 80, 25);

txtNo = new JTextField ();

txtNo.setHorizontalAlignment (JTextField.RIGHT);

txtNo.setEnabled (false);

txtNo.setBounds (125, 20, 200, 25);

txtName = new JTextField ();

txtName.setEnabled (false);

txtName.setBounds (125, 55, 200, 25);

txtDate = new JTextField ();

txtDate.setEnabled (false);

txtDate.setBounds (125, 90, 200, 25);

txtBal = new JTextField ();

txtBal.setHorizontalAlignment (JTextField.RIGHT);

txtBal.setEnabled (false);

txtBal.setBounds (125, 125, 200, 25);

//Aligninig The Navigation Buttons.

btnFirst = new JButton ("<<");

btnFirst.setBounds (15, 165, 50, 25);

btnFirst.addActionListener (this);

btnBack = new JButton ("<");

btnBack.setBounds (65, 165, 50, 25);

btnBack.addActionListener (this);

btnNext = new JButton (">");

btnNext.setBounds (225, 165, 50, 25);

btnNext.addActionListener (this);

btnLast = new JButton (">>");

btnLast.setBounds (275, 165, 50, 25);

btnLast.addActionListener (this);

txtRec = new JTextField ();

txtRec.setEnabled (false);

txtRec.setHorizontalAlignment (JTextField.CENTER);

txtRec.setBounds (115, 165, 109, 25);

//Adding the All the Controls to Panel.

jpRec.add (lbNo);

jpRec.add (txtNo);

jpRec.add (lbName);

jpRec.add (txtName);

jpRec.add (lbDate);

jpRec.add (txtDate);

jpRec.add (lbBal);

jpRec.add (txtBal);

jpRec.add (btnFirst);

jpRec.add (btnBack);

jpRec.add (btnNext);

jpRec.add (btnLast);

jpRec.add (txtRec);

//Adding Panel to Window.

getContentPane().add (jpRec);

//Load All Existing Records in Memory and Display them on Form.

populateArray ();

showRec (0);

//In the End Showing the New Account Window.

setVisible (true);

}

//Function use By Buttons of Window to Perform Action as User Click Them.

public void actionPerformed (ActionEvent ae) {

Object obj = ae.getSource();

if (obj == btnFirst) {

recCount = 0;

showRec (recCount);

}

else if (obj == btnBack) {

recCount = recCount - 1;

if (recCount < 0) {

recCount = 0;

showRec (recCount);

JOptionPane.showMessageDialog (this, "You are on First Record.",

"BankSystem - 1st Record", JOptionPane.PLAIN\_MESSAGE);

}

else {

showRec (recCount);

}

}

else if (obj == btnNext) {

recCount = recCount + 1;

if (recCount == total) {

recCount = total - 1;

showRec (recCount);

JOptionPane.showMessageDialog (this, "You are on Last Record.",

"BankSystem - End of Records", JOptionPane.PLAIN\_MESSAGE);

}

else {

showRec (recCount);

}

}

else if (obj == btnLast) {

recCount = total - 1;

showRec (recCount);

}

}

//Function use to load all Records from File when Application Execute.

void populateArray () {

try {

fis = new FileInputStream ("Bank.dat");

dis = new DataInputStream (fis);

//Loop to Populate the Array.

while (true) {

for (int i = 0; i < 6; i++) {

records[rows][i] = dis.readUTF ();

}

rows++;

}

}

catch (Exception ex) {

total = rows;

if (total == 0) {

JOptionPane.showMessageDialog (null, "Records File is Empty.\nEnter Records First to Display.",

"BankSystem - EmptyFile", JOptionPane.PLAIN\_MESSAGE);

btnEnable ();

}

else {

try {

dis.close();

fis.close();

}

catch (Exception exp) { }

}

}

}

//Function which display Record from Array onto the Form.

public void showRec (int intRec) {

txtNo.setText (records[intRec][0]);

txtName.setText (records[intRec][1]);

txtDate.setText (records[intRec][2] + ", " + records[intRec][3] + ", " + records[intRec][4]);

txtBal.setText (records[intRec][5]);

if (total == 0) {

txtRec.setText (intRec + "/" + total); //Showing Record No. and Total Records.

txtDate.setText ("");

}

else {

txtRec.setText ((intRec + 1) + "/" + total); //Showing Record No. and Total Records.

}

}

//Function use to Lock all Buttons of Window.

void btnEnable () {

btnFirst.setEnabled (false);

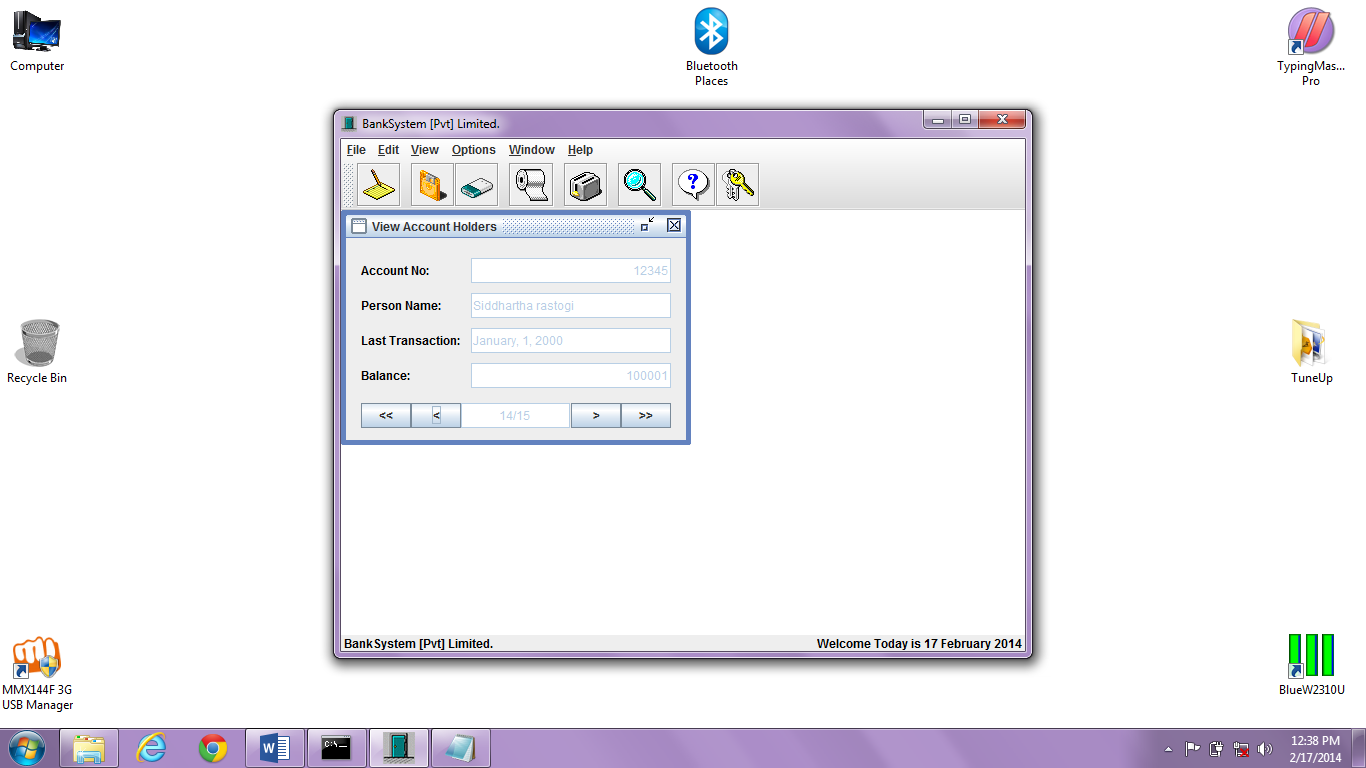
btnBack.setEnabled (false);

btnNext.setEnabled (false);

btnLast.setEnabled (false);

}

}



**View customer list**

import java.awt.\*;

import javax.swing.\*;

import java.awt.event.\*;

import java.io.\*;

import javax.swing.table.DefaultTableModel;

public class ViewCustomer extends JInternalFrame {

private JPanel jpShow = new JPanel ();

private DefaultTableModel dtmCustomer;

private JTable tbCustomer;

private JScrollPane jspTable;

private int row = 0;

private int total = 0;

//String Type Array use to Load Records into File.

private String rowData[][];

private FileInputStream fis;

private DataInputStream dis;

ViewCustomer () {

//super(Title, Resizable, Closable, Maximizable, Iconifiable)

super ("View All Account Holders", false, true, false, true);

setSize (475, 280);

jpShow.setLayout (null);

populateArray ();

tbCustomer = makeTable ();

jspTable = new JScrollPane (tbCustomer);

jspTable.setBounds (20, 20, 425, 200);

//Adding the Table to Panel.

jpShow.add (jspTable);

//Adding Panel to Window.

getContentPane().add (jpShow);

//In the End Showing the New Account Window.

setVisible (true);

}

//Function use to load all Records from File when Window Open.

void populateArray () {

//String Type Array use to Load Records into File.

String rows[][] = new String [500][6];

try {

fis = new FileInputStream ("Bank.dat");

dis = new DataInputStream (fis);

//Loop to Populate the Array.

while (true) {

for (int i = 0; i < 6; i++) {

rows[row][i] = dis.readUTF ();

}

row++;

}

}

catch (Exception ex) {

total = row;

rowData = new String [total][4];

if (total == 0) {

JOptionPane.showMessageDialog (null, "Records File is Empty.\nEnter Records to Display.",

"BankSystem - EmptyFile", JOptionPane.PLAIN\_MESSAGE);

}

else {

for (int i = 0; i < total; i++) {

rowData[i][0] = rows[i][0];

rowData[i][1] = rows[i][1];

rowData[i][2] = rows[i][2] + ", " + rows[i][3] + ", " + rows[i][4];

rowData[i][3] = rows[i][5];

}

try {

dis.close();

fis.close();

}

catch (Exception exp) { }

}

}

}

//Function to Create the Table and Add Data to Show.

private JTable makeTable () {

//String Type Array use to Give Table Column Names.

String cols[] = {"Account No.", "Customer Name", "Opening Date", "Bank Balance"};

dtmCustomer = new DefaultTableModel (rowData, cols);

tbCustomer = new JTable (dtmCustomer) {

public boolean isCellEditable (int iRow, int iCol) {

return false; //Disable All Columns of Table.

}

};

//Sizing the Columns of Table.

(tbCustomer.getColumnModel().getColumn(0)).setPreferredWidth (180);

(tbCustomer.getColumnModel().getColumn(1)).setPreferredWidth (275);

(tbCustomer.getColumnModel().getColumn(2)).setPreferredWidth (275);

(tbCustomer.getColumnModel().getColumn(3)).setPreferredWidth (200);

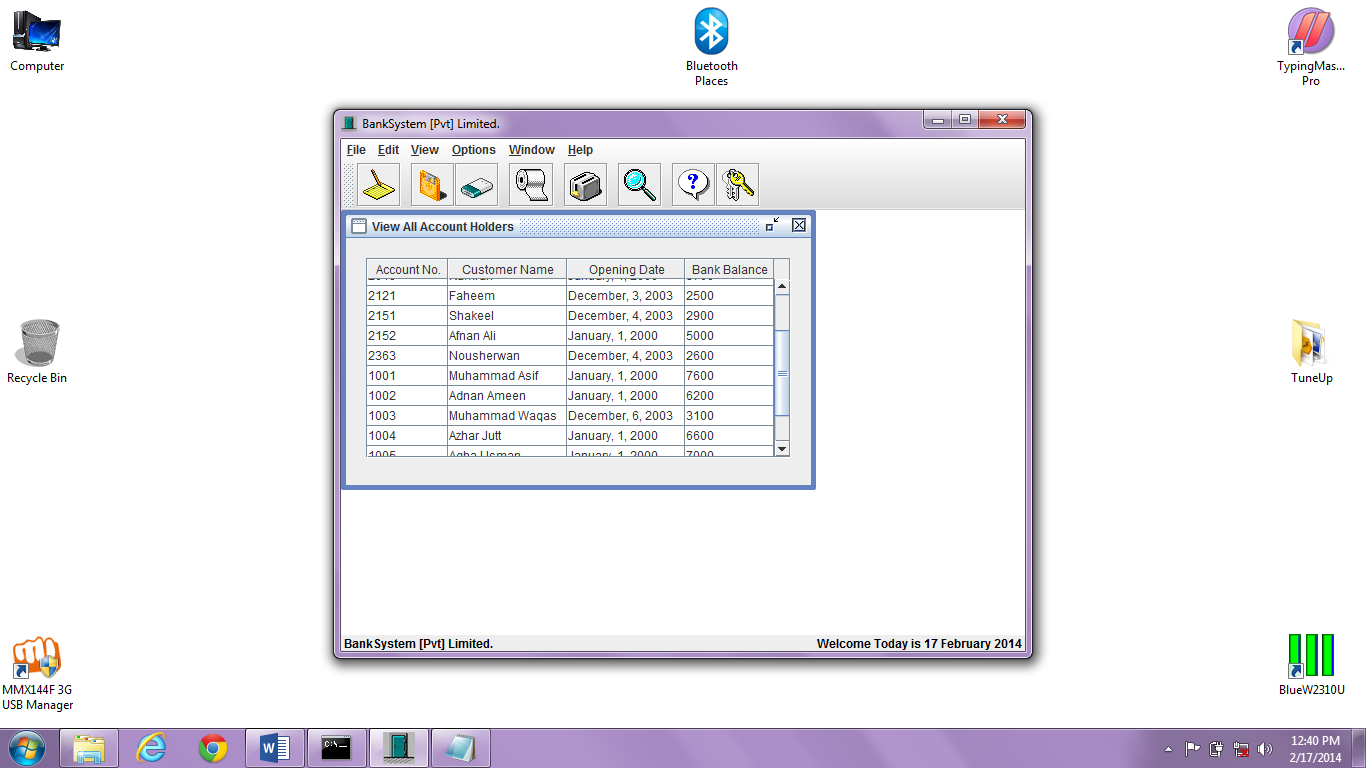
tbCustomer.setRowHeight (20);

tbCustomer.setSelectionMode (ListSelectionModel.SINGLE\_SELECTION);

return tbCustomer;

}

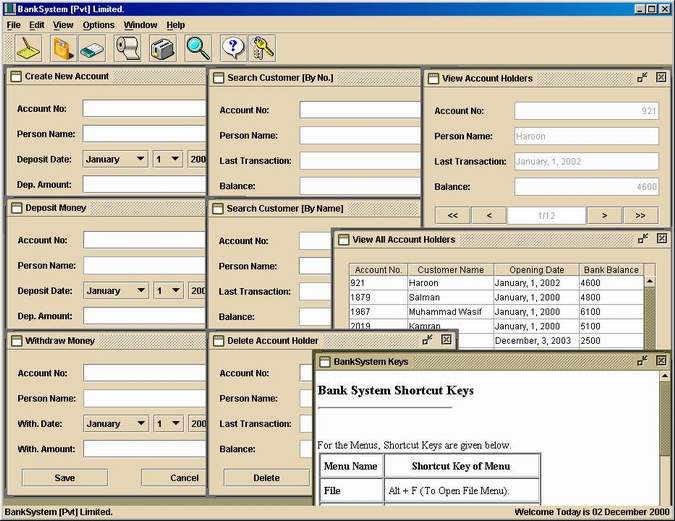
}



# 

**Data Reports**

**OUTPUT**



# Scope and Limitaions

As systems grew more complex, it become evident that the goal of the entire system cannot be easily comprehended. Hence need for the requirements analysis phase arose. Now, for large software systems, requirements analysis is perhaps the most difficult activity and also the most error prone.

Some of the difficulty is due to the scope of this phase. The software project is imitated by the client needs. In the beginning these needs are in the minds of various people in the client organization. The requirement analyst has to identify the requirements by tacking to these people and understanding there needs. In situations where the software is to automated a currently manuals process, most of the needs can be understood by observing the current practice.

BIBLIOGRAPHY

**BIBLIOGRAPHY**

1. OBJECT-ORIENTED PROGRAMMING WITH JAVA

*BY C.THOMAS WU (TATA McGRAW-HILL)*

1. CORE JAVA

*BY BALAGURU SWAMI*

1. JAVA 6 PROGRAMMING BLACK BOOK

*BY KOGENT LEARNING SOLUTION (DREAM-TECH)*

1. COMPLETE REFERENCE

*BY HERBERT SCHILDT*

1. LET US JAVA

*BY YASHAVANT KANETKAR*