ATM Application

*Introducing Database Programming and Client-Server Processing*

Objectives

In this tutorial, you will learn to:

Utilize an existing mySQL database.

Connect to databases.

Create SQL queries.

Retrieve and update Information In databases.

Use client-server techniques.

Outline

1. mySQL Database

2. Test-Driving the ATM Application

3. Planning the ATM Application

4. Relational Database Overview: The ATM Database

5. SQL

6. Creating Database Connections

7. Programming the ATM Application

8. Wrap-Up

Previously, you learned how to create sequential-access files and how to search through such files to locate information. Sequential-access files are inappropriate for so-called instant-access applications, in which information must be located immediately. Popular instant-access applications include airline ­reservation systems, banking systems, point-of-sale systems, automated teller machines (ATMs) and other transaction-processing systems that require rapid access to specific data. The bank where you have your account might have hun­dreds of thousands, or even millions, of other customers, but when you use an ATM. the bank's computers retrieve your account information in as little as a fraction of a second. This type of instant access is made possible by databases. Individual database records can be accessed directly (and quickly) without sequentially searching through large numbers of other records, as is required with sequential-access files. In this tutorial, you will be introduced to databases and the part of Java-the SQLDll API-used to interact with databases. You will learn about databases and the SQLDll as you create the ATM application.

1. **mySQL Database**

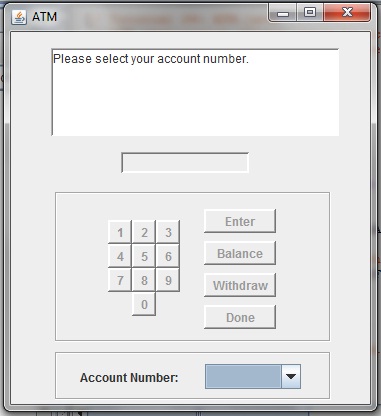
For the purposes of this tutorial, a mySQL database has been created and is hosted by the instructor’s Yahoo web site. It resides at www.boehnecamp.com/phpMyAdmin.

1. **Starting the ATM Application**

Many banks offer ATMs to provide their customers with quick and easy access to their bank accounts. When customers use these machines, their account information is updated immediately to reflect the transactions they perform. such as deposits, withdrawals and fund transfers between accounts. This application must meet the following requirements:

*A local bank has asked you to create a prototype automated teller machine (ATM) application to access a database that contains sample customer records. Each record consists of an account number, a Personal Identification Number (PIN), a first name and a balance amount. For testing purposes, valid account numbers will be provided in a ComboBox. The ATM application should allow the user to log into an account by providing a valid PIN. Once logged in, the user should be able to view the account balance deposit money into the account and withdraw money from the account (if the account contains sufficient funds). If money is withdrawn or deposited, the application should update the database.*

Your ATM application will allow the user to enter a PIN. If the user provides a correct PIN. then the ATM will retrieve information about the requested account. such as the account holder's name and balance from the database. If the PIN entered is invalid, a message is displayed asking the user to re-enter the PIN. You must first create a new C# Windows Forms Application that looks something like the following:



1. **Planning the ATM Application**

Now that you created the ATM Windows Forms Application, you will begin by analyzing the requirements. The following pseudocode describes the basic operation of the ATM application.

*When the user selects an account number from the ComboBox:*

*Disable the ComboBox*

*Clear the TextFleld for the PIN*

*Prompt the user to enter a PIN*

*Enable the keypad Buttons*

*Enable the Done Button*

*When the user enters the PIN:*

*Enable the Enter Button*

*Append the number to the PIN*

*When the user clicks the Enter Button to submit the PIN:*

*Search the database for the account number's corresponding account Information*

*If the user provided a correct PIN:*

*Clear the TextFleld*

*Disable the Enter Button*

*Disable the keypad Buttons*

*Enable the Balance and Withdraw Buttons*

*Display the status to the user*

*Else*

*Clear the TextFleld*

*Prompt the user to enter a valid PIN*

*When the user clicks the Balance Button:*

*Display the balance*

*When the user clicks the Withdraw Button:*

*Disable the Balance and Withdraw Buttons*

*Enable the keypad Buttons*

*Prompt the user to enter the withdrawal amount*

*When the user clicks the Enter Button to submit the withdrawal amount:*

*Disable the Enter Button*

*Disable the keypad Buttons*

*Process the withdrawal and display the withdrawal amount*

*Clear the withdrawal amount in the TextFleld*

*Enable the Balance and Withdraw Buttons*

*When the user clicks the Done Button:*

*Disable the keypad Buttons*

*Disable the Enter, Balance, Withdraw and Done Buttons*

*Enable the ComboBox*

*Display instructions for the next customer In the TextArea*

Now that you have test-driven the ATM application and studied its pseudocode representation, you will use an ACE table to help you convert the pseudocode to C#. The following table lists the actions, components and events required to complete your own version of this application.

|  |  |  |
| --- | --- | --- |
| Actlon | Component | Event |
|  | accountNumber ComboBox | User selects an account number from ComboBox |
| Disable the ComboBox | accountNumber ComboBox |
| Clear the TextField for the PIN | numberTextField |
| Prompt the user to enter a PIN | messageTextArea |
| Enable the keypad JButtons | zeroButton.oneButton, twoButton,threeButton. fourButton,fiveButton. sixButton,sevenButton. eightButton.nineButton |
| Enable the Done JButton | done Button |
|  | zeroButton.oneButton, twoButton,threeButton. fourButton,fiveButton. sixButton,sevenButton. eightButton.nineButton | User clicks keypad Button |
| Enable the Enter JButton | enter Button |
| Append the number to the PIN | number Button |
|  | enter Button | User clicks Enter Button |
| Search the database for the account number’s corresponding account information | myStatement  myResultSet |
| If the user provided a correct PIN  Clear the TextField | numberTextField |
| Disable the Enter Button | enterButton |
| Disable the keypad Buttons | zeroButton.oneButton, twoButton,threeButton. fourButton,fiveButton. sixButton,sevenButton. eightButton.nineButton |
| Enable the Balance and Withdraw Buttons | balanceButton  withdrawButton |
| Display status to the user | messageTextArea |
| Else  Clear the TextField | numberTextField |
| Prompt the user to enter a valid PIN | messageTextArea |
|  | balanceButton | User clicks Balance Button |
| Display the balance | message TextArea |

|  |  |  |
| --- | --- | --- |
| Actlon | Component | Event |
|  | withdrawButton | User clicks WIthdraw Button |
| Disable the Balance and Withdraw JButtons | balanceButton withdrawButton |
| Enable the keypad Buttons | zeroButton.oneButton, twoButton,threeButton. fourButton,fiveButton, sixButton,sevenButton. eightButton.nineButton |
| Prompt the user to enter the withdrawal amount | messageTextArea |
|  | enterButton | User clicks Enter Button |
| Disable the Enter Button | enterButton |
| Disable the keypad Buttons | zeroButton.oneButton, twoButton,threeButton. fourButton,fiveButton. sixButton,sevenButton. eightButton.nineButton |
| Process the withdrawal and display the withdrawal amount | myStatement messageTextArea |
| Clear withdrawal amount In the TextField | numberTextField |
| Enable the Balance and Withdraw Buttons | balanceJButton withdrawJButton |
|  | doneButton | User clicks Done Button |
| Disable the keypad Buttons | zeroButton,oneButton. twoButton.threeButton, fourButton,fiveButton, sixButton.sevenButton. eightButton,nineButton |
| Disable the Enter, Balance, Withdraw and Done Buttons | enterButton, balanceButton, withdrawButton, doneButton |
| Enable the ComboBox | accountNumberComboBox |
| Display Instructions for the next customer In the TextArea | messageTextArea |

1. **Relational Database Overview: The ATM Database**

In this section, you will become familiar with the ATM database used in this applica­tion. A database is an organized collection of data. Many different strategies exist for organizing databases to allow easy access to and manipulation of the data within them. A database management system (DBMS) enables applications to access and store data without worrying about how the data is organized.

Today's most popular database systems are relational databases. A relational database consists of data items stored in simple tables from which the data can be accessed. A table is used to store information in rows and columns. A row uniquely represents a set of values in a relational database. A column represents an individ­ual data attribute. Some popular relational database management systems (RDBMSs) are Microsoft SQL Server, Oracle, Sybase, IBM DB2, Informix and mySQL. In this tutorial, we present examples using mySQL, a PHP-based RDBMS from mySQL. Below displays the simple database used in the ATM application. This database consists of a single table, accountInformation.

|  |  |  |  |
| --- | --- | --- | --- |
| **accountNumber** | **pin** | **firstName** | **balanceAmount** |
| 12548693 | 1234 | John | 980.00 |
| 24578648 | 8568 | Susan | 125.00 |
| 35682458 | 5689 | Joseph | 3400.99 |
| 45632598 | 8790 | Michael | 1254.76 |
| 52489635 | 2940 | Donna | 9200.02 |
| 55698632 | 3457 | Elizabeth | 788.90 |
| 69857425 | 6765 | Jennifer | 677 .87 |
| 71869534 | 5678 | Al | 7799.24 |
| 88965723 | 1245 | Ben | 736.78 |
| 98657425 | 2456 | Bob | 946.09 |

The name of the table is accountInformation, and the table's primary pur­pose is to store the attributes of multiple accounts. This table contains ten rows and four columns. For example. in this table, the row containing 12548693, 1234, John and 980.00 represents a single account. The accountNumber, pin. firstName and balanceAmount columns represent the data in each row. The values stored in the accountNumber, pin and firstName columns are Strings. The values stored in the balanceAmount column are doubles.

In addition to rows and columns. a table should contain a primary key, which is a column (or combination of columns) that contains unique values. Primary keys are used to distinguish rows from one another. In this table, the accountNumber column is the primary key for referencing the data. Because no two account num­ber values are the same, the accountNumber column can act as the primary key.

Different users of a database often are interested in different data and different relationships among those data. Users often require only subsets of the rows and columns. To obtain these subsets. we use Structured Query Language (SQL) to specify which data to select from a table. SQL-pronounced "sequel"-is the inter­national standard language, used almost universally with relational databases to perform queries-requests for information that satisfy a given criteria-and to manipulate data. In the next section. you will learn how to write basic SQL state­ments.

**Self-Review**

1. Relation databases are composed of one or more \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

a.) charts b.) relatives

c.) tables d.) None of the above.

2. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ contains unique values that are used to distinguish rows from one another.

a.) primary key b.) SQL

c.) query d.) None of the above.

**Answers:** 1)c 2)a

**5. SQL**

In this section, you will learn to use basic SQL queries in the context of our ATM sample database. You will write your own SQL queries as you answer the exercises at the end of the tutorial.

**5.1 Basic SELECT Query**

You begin by considering several SQL.queries that extract information from the ATM database. A SQL query "selects" rows and columns from one or more tables in a database. Such selections are performed by SELECT queries. The basic form of a query is:

SELECT \* FROM *tableName*

In the preceding query, the FROM keyword indicates the table from which the query is taken and the asterisk (\*) indicates that all columns from the *tableName* table should be retrieved. For example, to retrieve all the data in the accountlnforma­tion table, you would use the query

SELECT \* FROM accountInformation

To retrieve only specific columns from a table, replace the asterisk (\*) with a comma-separated list of the column names. For example, to retrieve only the col­umns accountNumber and firstName for all rows in the accountInformation table, use the query

SELECT accountNumber, firstName FROM accountInformation

This query returns the data following:

12548693 John

24578648 Susan

35682458 Joseph

45632598 Michael

52489635 Donna

55698632 Elizabeth

69857425 Jennifer

71869534 Al

88965723 Ben

98657425 Bob

**5.2 WHERE Clause**

In most cases. it is necessary to locate rows in a database that satisfy certain selec­tion criteria. Only rows that satisfy the selection criteria are selected. SQL uses the optional WHERE clause to-specify the selection criteria for a query. The basic form of a SELECT query with selection criteria is:,

SELECT *columnName1, columnName2, …* FROM tableName WHERE *criteria*

For example, to select from table accountInformation the pin, firstName and balanceAmount columns for which the accountNumber equals "12548693", use the query:

SELECT pin, firstName, balanceAmount

FROM accountInformation

WHERE accountNumber = '12548693'

will return:

1234 John 980.0

Above shows the single row that is the result of the preceding query. Notice that SQL uses the single quote (‘) character as a delimiter for strings. The accountNumber 12548693 appears in quotes. because the values stored in the accountNumber column are represented as Strings in this database. Quotes are not used with numeric values, such as ints and doubles.

**5.3 UPDATE Statement**

An UPDATE statement modifies data in a table. The basic form of the UPDATE state­ment is:

UPDATE *tableName*

SET *columnName1=value1, columnName2=value2, .., columnNameN= valueN*

WHERE c*riteria*

where *tableName* is the table to update. The *tableName* is followed by the SET key­word and a comma-separated list of column name/value pairs in the format *columnName=value.* The WHERE clause provides criteria to determine which rows to update. The UPDATE statement:

UPDATE accountlnformation

SET balanceAmount = 1000

WHERE accountNumber = '12548693'

updates a row in the accountlnformation table. The statement indicates that bal­anceAmount will be assigned the value 1000 for the row in which accountNumber is equal to 12548693. The balanceAmount 1000 does not appear in quotes, because the values stored in the accountNumber column are represented as doubles in this database. Below shows the accountInformation table after the UPDATE opera­tion completes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 12548693 | 1234 | |  | John | 1000.0 |
| 24578648 | 8568 | |  | Susan | 125.0 |
| 35682458 | 5689 | |  | Joseph | 3400.99 |
| 45632598 | 8790 | |  | Michael | 1254.76 |
| 52489635 | 2940 | |  | Donna | 9200.02 |
| 55698632 | 3457 | |  | Elizabeth | 788.9 |
| 69857425 | 6765 | |  | Jennifer | 677 .87 |
| 71869534 | 5678 | |  | Al | 7799.24 |
| 88965723 | 1245 | |  | Ben | 736.78 |
| 98657425 | 2456 | |  | Bob | 946.09 |

**Self-Review**

1. SQL keyword \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is followed by the selection criteria that specify the rows to select in a query.

a.) SELECT b.) WHERE

c.) UPDATE d.) None of the above.

2. USE a SQL \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ statement to modify data in a table.

a.) SELECT b.) WHERE

c.) UPDATE d.) None of the above.

**Answers:** 1)b 2)c

**6. Creating Database Connections**

C# applications communicate with databases and manipulate their data using the SQLDll API. ln this tutorial, you will be introduced to SQLDll and use it to manipu­late a mySQL database. The techniques demonstrated here also can be used to manipulate other databases that have JDBC drivers. A SQLDll driver is a class pro­vided by a DBMS vendor that enables Java applications to access a particular data­base. To access a database with SQLDll, you first must connect to the database.

**1. Referencing the SQLDll package.** Download the attached dll file supplied as part of this project. It includes the sql class files: Connection.class, Statement.class, ResultSet.class and SQL.class. You must also reference the dll by adding a reference (Project -> Add Reference) and by adding the following line to your project:

using SQLDll;

**2. Declaring instance variables for database processing.** Add the following lines to declare the instance variables used to manipulate the database. myConnection is the Connection object. This Connection object manages the connection between the Java application and the data­base. Connection objects enable applications to create SQL statements that manipulate databases. As long as the connection remains open, SQL state­ments may be executed. myStatement is the Statement object. This statement object enables applications to execute SQL. If the SQL executed is a query, a ResultSet containing the rows and columns selected from the database is returned. The rows of the table are returned in sequence. myResultSet is the ResultSet object. Only one ResultSet can be open per Statement at any time.

// instance variables used to manipulate database  
 private Connection myConnection;  
 private Statement myStatement;  
 private ResultSet myResultSet;

**3. Connecting to the database.** To access and manipulate data in the database, you must first establish a connection to the database. Add the following lines to the ATM constructor to connect to the database. The try block will establish the database connection and create a Statement object.

The getConnection method attempts to connect to the database specified by its argument, databaseURL (received from main), utilizing the appropriate database driver specified by its argument, databaseDriver (received from main). The database driver is included in the SQLDll and the URL is the web location string of the database [www.boehnecamp.com/phpMyAdmin](http://www.boehnecamp.com/phpMyAdmin). If the DriverManager cannot connect to the database, the getConnec­tion method throws a SQLException. A SQLException indicates database connection or processing errors.

// establish connection to database  
 try  
 {  
 // connect to database  
 SQL sql = new SQL();  
 myConnection = sql.getConnection(databaseURL);  
 // create Statement for executing SQL  
 myStatement = myConnection.createStatement(databaseURL);  
 }  
 catch (Exception) {

Console.WriteLine ("Cannot connect to database server");

}

**4. Changing the constructor call.** To pass the database driver and URL information to the application, you will need following line:

String databaseURL = “http://www.boehnecamp.com/phpMyAdmin/razorsql\_mysql\_bridge.php";  
   
**5. Closing the statement and database connection.** To properly close the Statement and Connection objects, you will need the following lines in the FormClosing event of the Form:

// close statement and database connection  
 myStatement.close();  
 myConnection.close();

**6. Compiling the application.** Compile your application. If your application does not compile properly, fix the errors in your code before continuing.

**7. Programming the ATM Application**

Now that you have established a connection to the ATM mySQLdatabase and have obtained a Statement object to manipulate the database, you will write the necessary code to complete the application. When you view the template code file (ATM.cs), you will notice that the basic functionality of the ATM application is already provided. However, you will be declaring the loadAccountNumbers, retrieveAccountInformation and updateBalance methods that will access the database. The empty method headers for these have been provided for you. Your task will be to code their functionality

Now you will declare the loadAccountNumbers method to fill the account­Number JComboBox with a list of account numbers from the database. This will allow you to select an existing account number when the ATM application is executed. [*Note:* You provide the ComboBox containing the account numbers for demonstra­tion purposes only. A real ATM would not provide a list of account numbers like this.]

**7.1 Displaying Existing Account Numbers in the ComboBox**

**1. Filling the ComboBox with account numbers.** Insert the following lines of code into the loadAccountNumbers method. These lines use the Statement object’s executeQuery method to submit a SQL query to the database. The specified query selects all the account numbers from the accountlnformation table. The executeQuery method takes a String argument that specifies the query and returns an object that implements the ResultSet interface. The ResultSet object enables the application to manipulate the query results. The executeQuery method throws a SQLException if an error occurs while accessing the database.

// load account numbers to accountNumberJComboBox  
 private void loadAccountNumbers()  
 {  
 // get all account numbers from database  
 try   
 {  
 myResultSet = myStatement.executeQuery( "SELECT accountNumber FROM

accountInformation" );  
   
 // add account numbers to accountNumberJComboBox  
 while ( myResultSet.next() )  
 {  
 accountNumberComboBox.Items.Add(myResultSet.getString( "accountNumber" ) );  
 }  
   
 myResultSet.close(); // close myResultSet  
   
 } // end try  
   
 catch ( Exception)  
 {  
 Console.WriteLine(“Error in loadAccountNumbers”);  
 }  
   
 } // end method loadAccountNumbers

The loop processes the ResultSet and fills the account­Number ComboBox with account numbers. Before processing the data in a ResultSet, you must position the ResultSet cursor (a pointer which points to a row in a ResultSet) to the first row in the ResultSet. Initially, the ResultSet cursor is positioned before the first row. As a result. the first time you call the next method, it will move the cursor to the first row. The next method will move the cursor down one row from its current position. The cursor points to the current row in the ResultSet. The next method returns the boolean value true if it can be positioned in the next row; otherwise, the method returns false to indicate that there are no more rows to process in the ResultSet.

If there are rows in the ResultSet, the getString method will extract the accountNum­ber of the current row and add it to the accountNumber ComboBox using the ComboBox method addltem, which adds an item to the end of the list of items in the ComboBox. When processing a ResultSet, you extract each column of the ResultSet as a specific java type. For example, ResultSet method getString returns the column value as a String, method getInt returns the column value as an int and method getDouble returns the column value as a double. In this example, ResultSet method getString is used to get the column value as a String. The getString method accepts as an argument a column name (as a String, such as "accountNumber"), indicating which column's value to obtain.

The ResultSet object's close method is used to close the ResultSet. Closing the ResultSet releases its resources and prevents it from being used to continue processing results of a prior query. The close method throws a SQLException if an error occurs while accessing the database.

**2. Compiling the application.** Compile your application. If your application does not compile correctly, fix the errors in your code before continuing.

Now you are ready to define the retrieveAccountlnformation method, which will determine whether the PIN number provided by the user is valid.

**7.2 Retrieving Account Information from the Database**

**1. Retrieving account information.** Insert the following lines into the retrieveAccountInformation method. These lines use the Statement object's executeQuery method to submit a query that selects from the accountInforrnation table the pin, firstName and balanceAmount values for the specified account number (userAccountNumber). The oject userAccountNumber (an instance variable provided in the tem­plate) contains the account number selected by the user from accountNum­ber JComboBox. Note the use of single quotes and dynamically created selection criteria of the WHERE clause.

private void retrieveAccountInformation()  
 {  
 // get account information  
 try   
 {  
 myResultSet = myStatement.executeQuery( "SELECT pin, " +   
 "firstName, balanceAmount FROM accountInformation " +  
 "WHERE accountNumber = '" + userAccountNumber + "'" );  
   
 // get next result  
 if ( myResultSet.next() )  
 {  
 pin = myResultSet.getString( "pin" );  
 firstName = myResultSet.getString( "firstName" );  
 balance = myResultSet.getDouble( "balanceAmount" );  
 }  
   
 myResultSet.close(); // close myResultSet  
   
 } // end try  
   
 catch (Exception)  
 {  
 Console.WriteLine(“Error in retrieveAccountInformation”);  
 }  
   
 } // end method retrieveAccountInformation

The if statement accesses the information that is returned from the database. Recall that the next method must be called to position the cursor to the ResultSet's first row. The instance variables pin, firstName and balance are set to the PIN, first name and balance amount stored in the database for the requested account number. Each account number is unique, so there can be only one row in the ResultSet if the specified accountNumber is found in the database. Recall that the getString method returns the column value as a String, so the String representations of the PIN number and first name are retrieved. The column balanceAmount contains floating-point numbers, so the getDouble method is used to the double value of the account balance. The getDouble method accepts as an argument a column name (as a String, such as "balanceAmount") indicating which column's value to obtain.

**2. Compiling the application.** Compile your application. If your application does not compile correctly, fix the errors in your code before continuing.

Next, you will implement the updateBalance method, which will be invoked if the user-requested withdrawal amount can be deducted from the account balance. The updateBalance method updates the account balance in the database.

**7.3 Updating the Balance Amount in the Database**

**1. Updating the balance amount.** Insert the following lines into updateBalance method. These lines use the Statement object's executeUpdate method to submit an SQL statement that updates the balanceAmount column in the accountlnformation table for the row with the specified accountNumber. The balanceAmount for that row will be set to the value of balance, which is an instance variable that contains the new balance amount after a withdrawal. The executeUpdate method takes a String argument that specifies the SQL to execute and returns an int that indicates how many rows were updated. You should use the executeUpdate method for SQL statements that modify database data, such as UPDATE statements. The executeUpdate method throws a SQLException if an error occurs while accessing the database.

// update database after withdrawing

private void updateBalance()

{

// update balance in database

try

{

myStatement.executeUpdate( "UPDATE accountInformation" +

" SET balanceAmount = " + balance + " WHERE " +

"accountNumber = '" + userAccountNumber + "'" );

}

catch (Exception)

{

Console.WriteLine(“Error in updateBalance”);

}

} // end method updateBalance

**2. Compiling the application.** Compile your application. If your application does not compile correctly, fix the errors in your code before continuing.

**8. Wrap-Up**

|  |  |  |
| --- | --- | --- |
| Actlon | Component | Event |
|  | withdrawJButton | User clicks WIthdraw lButton |
| Disable the 6alanc:e and Withdraw J6uttons | balanceJButton withdrawJButton |
| Enable the keypad J6uttone | zeroJButton.oneJButton, twoJButton,threeJButton. fourJButton,fiveJ8utton, sixJButton,sevenJButton. eightJButton.nineJButton |
| Prompt the user to enter the withdrawal amount | messageJTextArea |
|  | enterJButton | User clicks Enter JButton |
| Dlsa17'e the Enter J6utton | enterJButton |
| Disable the keypad J6uttons | zeroJButton.oneJButton, twoJButton,threeJButton. fourJButton,fiveJButton. sixJButton,sevenJButton. eightJButton.nineJ8utton |
| Process the withdrawal and display the withdrawal amount | myStatement messageJTextArea |
| Clear withdrawal amount In the JTextField | numberJTextField |
| Enal7le the 6alance and Withdraw J6uttons | balanceJButton withdrawJ8utton |
|  | doneJButton | User clicks Done JButton |
| Disable the keypad J6uttons | zeroJButton,oneJButton. twoJButton.threeJButton, fourJButton,fiveJButton, sixJButton.sevenJButton. eightJButton,nineJButton |
| Disable the Enter, 6alance, Withdraw and Done J6uttons | enterJButton, balanceJButton, wi thdrawJButton, doneJButton |
| Enable the JCombo6ox | accountNumberJComboBox |
| Display Instructions for the next customer In the JTextArea | messageJTextArea |

You must now review the ACE table to finish the remaining items required by the local bank client (such as a Deposit Button, proper enabling and disabling of Buttons, etc.).