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 JDBC API-used to interact with databases. You will learn about databases and the JDBC API 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.

2. Test Driving 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 JComboBox. 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 begin by test-driving the completed application. Next, you learn the additional Java technologies you will need to create your own version of this application.

1. Locating the completed application. Use jGrasp to find the folder that contains ATM.java as well as the mySQL API programs, Connection.class, Statement.class, ResultSet.class, SQL.class and Webpage.class. Open the java file ATM.java.
2. Compiling the application. Compile the ATM.java file.
3. Running the ATM application. Run the ATM.class file. The application will appear with the keypad JButtons and the Enter, Balance, Withdraw and Done JButtons initially disabled. The JTextArea at the top of the ATM displays a message informing you to select an account number.
4. Selecting an account number. Use the Account Number: JComboBox at the bottom of the application to select your account number (assume that it is 12548693). Notice that the JTextArea now prompts the user to provide a PIN. The JComboBox has been disabled, preventing the user from selecting another account until the current transaction is complete. The Done JButton has been enabled, allowing you to indicate that the current transaction is complete, so the next user can begin a transaction. The keypad JButtons have also been enabled allowing the user to enter the PIN.
5. Entering a PIN. Use the keypad to input 1234 for the PIN. Notice that each time you click a digit on the numeric keypad, an asterisk (*) is displayed to conceal your input. The Enter JButton should be enabled now.
6. Clicking the Enter JButton. Click Enter to submit your PIN. The JTextArea at the top of the ATM will display a welcome message telling you to select a transaction. Notice that the Balance and WIthdraw JButtons have been enabled, allowing you to perform these types of transactions. The Enter JButton and the keypad JButtons are disabled to prevent you from entering numbers without selecting the transaction type.
7. Viewing the account balance. Click the Balance JButton to view the account balance. The amount displays in the JTextArea at the top of the application.
8. Withdrawing money from the account. Click the Withdraw JButton to initiate a withdrawal. The JTextArea at the top of the application asks you to input the amount you want to withdraw. The keypad JButtons are enabled, allowing you to enter the withdrawal amount. The Balance and Withdraw JButtons are disabled, preventing you from performing another transaction without first completing the withdrawal.
9. Entering the withdrawal amount. Use the keypad to input 20 for the withdrawal amount. Notice that once you click the 2 JButton, the Enter JButton is enabled. After inputting 20, click Enter. The JTextArea displays the withdrawal amount, which is $20. 00. Both the Balance and Withdraw JButtons are enabled for the next transaction, and the Enter JButton and the keypad JButtons are disabled
10. Confirm that the account information has been updated. Click the Balance JButton to check the balance of the account. Notice that the balance amount reflects the withdrawal you performed in Step 9.
11. Ending the transaction. Click the Done JButton to complete the transaction. The transaction ends and the instructions for the next customer are displayed in the JTextArea. The Enter, Balance, Withdraw, Done and keypad JButtons are disabled. The JComboBox is enabled, allowing the new customer to select an account number.

3. Planning the ATM Application

Now that you have test-driven the ATM application, you will begin by analyzing the application. The following pseudocode describes the basic operation of the ATM application.

When the user selects an account number from the JComboBox:
Disable the JComboBox
Clear the JTextFleld for the PIN
Prompt the user to enter a PIN
Enable the keypad JButtons
Enable the Done JButton
When the user enters the PIN:
Enable the Enter JButton
Append the number to the PIN
When the user clicks the Enter JButton to submit the PIN:
Search the database for the account number's corresponding account Information
If the user provided a correct PIN:
Clear the JTextFleld
Disable the Enter JButton
Disable the keypad JButtons
Enable the Balance and WIthdraw JButtons
Display the status to the user
Else
Clear the JTextFleld
Prompt the user to enter a valid PIN
When the user clicks the Balance JButton:
Display the balance
When the user clicks the Withdraw JButton:
Disable the Balance and Withdraw JButtons
Enable the keypad JButtons
Prompt the user to enter the withdrawal amount
When the user clicks the Enter JButton to submit the withdrawal amount:
Disable the Enter JButton
Disable the keypad JButtons
Process the withdrawal and display the withdrawal amount
Clear the withdrawal amount In the JTextFleld
Enable the Balance and WIthdraw JButtons
When the user clicks the Done JButton:
Disable the keypad JButtons
Disable the Enter, Balance, Withdraw and Done JButtons
Enable the JComboBox
Display instructions for the next customer In the JTextArea

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 java. The following table lists the actions, components and events required to complete your own version of this application.

Actlon	Component	Event
	accountNumber JComboBox	User selects an account number from JComboBox
Disable the JComboBox	accountNumber JComboBox	
Clear the JTextField for the PIN	number JTextField	
Prompt the user to enter a PIN	messageJTextArea	
Enable the keypad JButtons	zeroJButton.oneJButton, twoJButton,threeJButton. fourJButton,fiveJButton. sixJButton,sevenJButton. eightJButton.nineJButton	
Enable the Done JButton	done JButton	
	zeroJButton.oneJButton, twoJButton,threeJButton. fourJButton,fiveJButton. sixJButton,sevenJButton. eightJButton.nineJButton	User clicks keypad JButton
Enable the Enter JButton	enter JButton	
Append the number to the PIN	number JButton	
	enter JButton	User clicks Enter JButton
Search the database for the account number's corresponding account information	mySTatement myResultSet	
If the user provided a correct PIN Clear the JTextField	number JTextField	
Disable the Enter JButton	enter JButton	
Disable the keypad JButtons	zeroJButton.oneJButton, twoJButton,threeJButton. fourJButton,fiveJButton. sixJButton,sevenJButton. eightJButton.nineJButton	
Enable the Balance and Withdraw JButtons	balance JButton withdraw JButton	
Display status to the user	message JTextArea	
Else Clear the JTextField	number JTextField	
Prompt the user to enter a valid PIN	message JTextArea	
	balance JButton	User clicks Balance JButton
Display the balance	message JTextArea	

Actlon	Component	Event
	withdrawJButton	User clicks WIthdraw JButton
Disable the Balance and Withdraw JButtons	balanceJButton withdrawJButton	
Enable the keypad JButtons	zeroJButton.oneJButton, twoJButton,threeJButton. fourJButton,fiveJ8utton, sixJButton,sevenJButton. eightJButton.nineJButton	
Prompt the user to enter the withdrawal amount	messageJTextArea	
	enterJButton	User clicks Enter JButton
Disable the Enter JButton	enterJButton	
Disable the keypad JButtons	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	
Enable the Balance and Withdraw JButtons	balanceJButton withdrawJButton	
	doneJButton	User clicks Done JButton
Disable the keypad JButtons	zeroJButton,oneJButton. twoJButton.threeJButton, fourJButton,fiveJButton, sixJButton.sevenJButton. eightJButton,nineJButton	
Disable the Enter, Balance, Withdraw and Done JButtons	enterJButton, balanceJButton, withdrawJButton, doneJButton	
Enable the JComboBox	accountNumberJComboBox	
Display Instructions for the next customer In the JTextArea	messageJTextArea	

4. 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 accountInformation 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 criteria
```

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 accountInformation
        SET balanceAmount = 1000
        WHERE accountNumber = '12548693'
```

updates a row in the accountInformation 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

Java applications communicate with databases and manipulate their data using the JDBC API. In this tutorial, you will be introduced to JDBC 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 JDBC driver is a class provided by a DBMS vendor that enables Java applications to access a particular database. To access a database with JDBC, you first must connect to the database.

1. Importing the java.sql package. Make sure the folder you are using includes the sql class files: Connection.class, Statement.class, ResultSet.class and SQL.class. You must also import the java.sql package by adding the line:

```
Import java.sql.*;
```

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 com.mysql.jdbc.Driver and the JDBC URL is the web location of the database 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 ( SQLException exception ) {
             exception.printStackTrace();
           }
        catch (Exception e) {
             System.err.println ("Cannot connect to database server");
           }
```

4. Changing the constructor call. To pass the database driver and URL information to the constructor, change the following lines:

```
  // method main
      public static void main( String[] args )
     {
        String databaseDriver = "com.mysql.jdbc.Driver";
        String databaseURL =
"http://www.boehnecamp.com/phpMyAdmin/razorsql_mysql_bridge.php?database=ATM&tablename=acc
ountInformation";
       // create new ATM
        ATM atm = new ATM( databaseDriver, databaseURL );

    } // end method main
```

5. Closing the statement and database connection. To properly close the Statement and Connection objects, change the following lines:

```
  // close statement and database connection
      private void frameWindowClosing( WindowEvent event )
     {
     // close myStatement and database connection
        myStatement.close();
        myConnection.close();
        System.exit( 0 );
    } // end method frameWindowClosing
```

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.java), 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 JComboBox 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 JComboBox

1. Filling the JComboBox 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 accountInformation 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() )
        {
          accountNumberJComboBox.addItem(myResultSet.getString( "accountNumber" ) );
        }

        myResultSet.close(); // close myResultSet

      } // end try

         catch ( SQLException exception )
        {
          exception.printStackTrace();
        }

    } // end method loadAccountNumbers
```

The loop processes the ResultSet and fills the accountNumber JComboBox 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 JComboBox using the JComboBox method addItem, which adds an item to the end of the list of items in the JComboBox. 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 retrieveAccountInformation 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 ( SQLException exception )
        {
          exception.printStackTrace();
        }

    } // 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 exe cuteUpdate method to submit an SQL statement that updates the balanceAmount column in the accountInformation 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 ( SQLException exception )
        {
          exception.printStackTrace();
        }

    } // 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

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