**An example of a program using GUI to manipulate data to support Project 2 of CMIS242.**

The following is a program written to demonstrate results that change a value in a pre-set array. The program uses GUI to manipulate data. There are basically two operations for this program: increasing and decreasing an integer number. The data is all initialized in a two-dimensional array. The values used by this array are in the following table.

|  |  |
| --- | --- |
| Account Number | Initial Balance |
| 2331 | 500 |
| 5432 | 1000 |
| 6323 | 20000 |
| 1023 | 3000 |
| 7821 | 3200 |
| 9242 | 400 |
| 8921 | 2500 |
| 3523 | 320 |
| 4342 | 4000 |

There is a test plan used to ensure that the program is working correctly. This test plan is in the attached document titled ***Test Plan Project 2 Program***.

When the program runs, the user enters one of the account numbers included in the above table. Then the user clicks the Request Balance button. The result of clicking that event, displays the initial balance that is derived from the second column of the two-dimensional array above. To change the initial balance, the user enters an amount to change the balance in Transaction Amount. If the Increase button is clicked, the event will display a line at the bottom of the window with the account number and the new balance being an increase from the current balance and the amount entered in Transaction Amount. The user can change the value in the Transaction Amount box to decrease or increase the current balance as many times as desired. Once the user is finished with that account number, the user can enter a new account number and make changes to that new account’s balance. Until the program ends the new values are maintained by the program for each account. If an account number is entered that is not in the table, an error message displays at the bottom of the window and a 0 appear the Initial Balance. Any value in the Transaction Amount is not affected by the error.

A complete copy of the code is below. Since you will be generating your own Project on you IDE, the two highlighted lines that identify the program created from this example would need replacement by your lines.

/\*

\* Using JFrame to demonstrate the use of GUI for processing.

\* Purpose is to demonstrate the change in the values with

\* increase and decrease buttons.

\* This program manipulates data internally with a pre-determined

\* two-dimensional array. Modifications of this program could

\* use files to read the data into the program and write to new

\* files that hold the changed set of values from the origianl file.

\*/

package cmis242updown;

import java.awt.\*;

import java.awt.event.\*;

import java.awt.GridBagLayout;

import java.awt.GridBagConstraints;

import javax.swing.\*;

public class CMIS242UpDown {

public static class ChangeAmount extends JFrame {

JButton submit, increase, decrease, exit; //The three identifiers used with the events of the program

JTextField number1, number2; //The values that are used to manipulate the data of the array

JLabel entry1, entry2, entry2a, entry3, result1, result2; //References to aid the user in the program

//The pre-determined set of values for the program

int [ ][ ] aryValues = { {2331, 500},

{5432, 1000},

{6323, 20000},

{1023, 3000},

{7821, 3200},

{9242, 400},

{8921, 2500},

{3523, 320},

{4342,4000}

};

//the Constructor for the class

public ChangeAmount () {

// Established type of Java layout: GridBagLayout ()

setLayout(new GridBagLayout ());

//Creates an instance of the layout

GridBagConstraints window = new GridBagConstraints();

/\* The following sets of coding defines the attributes for various objects in the layout.

\*

\* This program is using the GridBagLayout. This layout and the "null" layout are the most

\* customizeable layouts in Java. The setLayout(null) allows for absolute positioning.

\* The other layouts available for use in Java are FlowLayout, BorderLayout, boxLlayout,

\* CardLayout, GridLayout, GroupLayout, and SpringLayout.

\*

\* This layout uses the combination of x and y axies to position the various objects on the

\* window/panel. The first position is 0, 0. The width determines the length of the object.

\* The use of 0 allows for the object to span the entire width of the columns displayed.

\* The fill directs the program to determine a direction. For this program it is horizontal.

\*/

// Establishes the pixels suronding each object within the layout

window.insets = new Insets (5, 5, 5, 5);

entry1 = new JLabel ("Enter Account # (four digits):");

entry1.setHorizontalAlignment(JLabel.RIGHT);

window.fill = GridBagConstraints.HORIZONTAL;

window.gridx = 0;

window.gridy = 0;

window.gridwidth = 1;

add (entry1, window);

number1 = new JTextField(10);

window.fill = GridBagConstraints.HORIZONTAL;

window.gridx = 1;

window.gridy = 0;

window.gridwidth = 3;

add (number1, window);

submit = new JButton("Request Balance");

window.fill = GridBagConstraints.HORIZONTAL;

window.gridx = 0;

window.gridy = 1;

window.gridwidth = 0;

add (submit, window);

entry2 = new JLabel ("Initial Balance:");

entry2.setHorizontalAlignment(JLabel.RIGHT);

window.fill = GridBagConstraints.HORIZONTAL;

window.gridx = 0;

window.gridy = 2;

window.gridwidth = 1;

add (entry2, window);

entry2a = new JLabel (" ");

window.fill = GridBagConstraints.HORIZONTAL;

window.gridx = 1;

window.gridy = 2;

window.gridwidth = 3;

add (entry2a, window);

entry3 = new JLabel ("Transaction Amount:");

entry3.setHorizontalAlignment(JLabel.RIGHT);

window.fill = GridBagConstraints.HORIZONTAL;

window.gridx = 0;

window.gridy = 3;

window.gridwidth = 1;

add (entry3, window);

number2 = new JTextField(10);

window.fill = GridBagConstraints.HORIZONTAL;

window.gridx = 1;

window.gridy = 3;

window.gridwidth = 3;

add (number2, window);

increase = new JButton("Increase");

window.fill = GridBagConstraints.HORIZONTAL;

window.gridx = 0;

window.gridy = 4;

window.gridwidth = 0;

add (increase, window);

decrease = new JButton("Decrease");

window.fill = GridBagConstraints.HORIZONTAL;

window.gridx = 0;

window.gridy = 5;

window.gridwidth = 0;

add (decrease, window);

result1 = new JLabel(" ");

window.fill = GridBagConstraints.HORIZONTAL;

window.gridx = 0;

window.gridy = 6;

window.gridwidth = 2;

add (result1, window);

result2 = new JLabel(" ");

window.fill = GridBagConstraints.HORIZONTAL;

window.gridx = 0;

window.gridy = 7;

window.gridwidth = 2;

add (result2, window);

exit = new JButton("Exit");

window.fill = GridBagConstraints.HORIZONTAL;

window.gridx = 0;

window.gridy = 8;

window.gridwidth = 0;

add (exit, window);

//Generates instances for the events in this program.

//Each event is attached to the listener.

event mathAction = new event();

submit.addActionListener(mathAction);

increase.addActionListener(mathAction);

decrease.addActionListener(mathAction);

exit.addActionListener(mathAction);

}// end Constructor ChangeSize ()

public class event implements ActionListener {

/\*

\* Declares the variables used in this class. The first two relate to

\* the events. The remaining three deal with the processing of

\* the balance in each account.

\*/

private int operand1, operand2 , balance , newBalance, accountIndex;

public void actionPerformed (ActionEvent mathAction) {

//Declares the identifer that accepts from the click of a button

String operators = mathAction.getActionCommand();

/\*

\* When the Request Balance button is used, the following code is processed.

\*

\* The value in number1 related to the label entry1: " Enter Account # (four digits "

\* is converted into an integer.

\*

\* The for loop attempts to match the value entered in number1 object with each of the

\* account numbers in the array. The account number is in the first column of the row. Using an

\* if statement, the value in operand1 is checked against the first column in each entry in the array.

\* If there is a match, then the balance found in column 1 of the appropriate row of the array

\* is assigned to the balance identifier. To bookmark the appropriate account row to preserve

\* for future processing of the maatched account in this program, the row index is assigned to

\* accountIndex. The balance is converted to a String and displayed in entry2a that has the

\* prompt of "Current Balance ". A flag is used to check for a match. with a match. A 1 is

\* assigned to the match identifier -- a Boolean couild also be used -- when the if statement

\* evaluates to true. The match identifier is reset to zero the next time the event is processed.

\*

\* After the for loop ends, if there was no match (match is still zero), then a message is displayed

\* in the result1 label: "Account number does not exist. Please try again." A zero is assigned to the

\* entry2a label.

\*/

if (operators.equals("Request Balance")) {

operand1 = Integer.parseInt(number1.getText());

int match = 0;

for (int prefix = 0; prefix < aryValues.length; prefix++) {

int accountNo = aryValues[prefix][0];

result1.setText(" ");

if (accountNo == operand1) {

balance = aryValues[prefix][1];

accountIndex = prefix;

entry2a.setText(Integer.toString(balance));

entry2a.setForeground(Color.BLACK);

match = 1;

}// end if (accountNo == operand1)

if (match != 1) {

entry2a.setText(Integer.toString(0));

result1.setText("Account number does not exist. Please try again.");

result1.setForeground(Color.RED);

}// end if (match != 1)

}// end for (int prefix = 0; prefix < aryValues.length; prefix++)

}//end if (operators.equals("Request Balance"))

/\* When the Increase button is used, the following code is processed.

\*

\* The text value of number2 from the entry window after the "Transaction Amount " label

\* is asssigned to operand2. For use in a mathematical process, the value in number2 is

\* changed to an integer.

\*

\* The balance that was assigned a value in the Request Balance event or generated from an

\* increase or decrease event after the initial process of the iniital balance is added to the

\* value in the operand2 and the result is assigned to the second column of the array for the

\* account that was a match using accountIndex. These two two values are added with the

\* result being assigned to column 1 of the row containing the index for aryValues held in

\* the accountIndex.

\*

\* To ensure that the array is updated, the value is placed in the newBalance. Then the value in

\* in newBalance is used for the current balance, thus perserving the new value in the array

\* when the value in balance is changed in each new process.

\*

\* The newBalance identifier is used to make appropriate data type changes to allow for the

\* value to be displayed in result1. The text is blue.

\*/

if (operators.equals("Increase")) {

operand2 = Integer.parseInt(number2.getText());

aryValues[accountIndex][1] = balance + operand2;

newBalance = aryValues[accountIndex][1];

balance = newBalance;

String newSBalance = Integer.toString(newBalance);

result1.setText(" The new Balance for Account " + aryValues[accountIndex][0] + " is " + newSBalance);

result1.setForeground(Color.BLUE);

}// end if (operators.equals("Increase"))

/\* When the Increase button is used, the following code is processed.

\*

\* The text value of number2 from the entry window after the "Transaction Amount " label

\* is asssigned to operand2. For use in a mathematical process, the value in number2 is

\* changed to an integer.

\*

\* The balance that was assigned a value in the Request Balance event or generated from an

\* increase or decrease event after the initial process of the iniital balance is subtracted from

\* the value in the operand2 and the result is assigned to the second column of the array for

\* the account that was a match using accountIndex. These two two values are added with the

\* result being assigned to column 1 of the row containing the index for aryValues held in

\* the accountIndex.

\*

\* To ensure that the array is updated, the value is placed in the newBalance. Then the value in

\* in newBalance is used for the current balance, thus perserving the new value in the array

\* when the value in balance is changed in each new process.

\*

\* The newBalance identifier is used to make appropriate data type changes to allow for the

\* value to be displayed in result1. The text is blue.

\*/

if (operators.equals("Decrease")) {

operand2 = Integer.parseInt(number2.getText());

aryValues[accountIndex][1] = balance - operand2;

newBalance = aryValues[accountIndex][1];

balance = newBalance;

String newSBalance = Integer.toString(newBalance);

result1.setText(" The new Balance for Account " + aryValues[accountIndex][0] + " is " + newSBalance);

result1.setForeground(Color.BLUE);

}// end if (operators.equals("Decrease"))

/\* When the exit button is used, the following code is processed.

\*

\* Basically, the use of the gnerice code is used to close the window.

\*

\* If there is additional processing when the Exit button is clicked, then that

\* code is processed before the System.exit(0); command is invoked.

\*/

if (operators.equals("Exit")) {

System.exit(0);

}//end if (operators.equals("Exit"))

}// end public void actionaPerformed (ActionEvent mathAction)

}// end public class event implements ActionListener

}// end ChangeAmount

public static void main(String[] args) {

//An instance of the class ChangeAmount is generated.

ChangeAmount CalcBox = new ChangeAmount ();

// Establish basic parameters for the GUI

CalcBox.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

CalcBox.setSize(450, 350);

CalcBox.setTitle("Change the Balance");

CalcBox.setLocationRelativeTo(null);

CalcBox.setVisible(true);

}// end main ()

}// end CMIS242UpDown

**Initial Screen:**

**A screenshot of a cell phone

Description automatically generated**

**Initial Balance for Account 1023 Screen:**

**A screenshot of a cell phone

Description automatically generated**

**Increased Balance for Account 1023 by $505 Screen:**

**A screenshot of a cell phone

Description automatically generated**