**Lab 8**

Josue Ponce

Montgomery College

4/4/2018

Author’s Note

This lab report was prepared for CMSC 203 CRN #30672, taught by professor Ahmed Tarek

**Table of Contents**

Task #1 Print Array to Console Code………………………………………………………….2

Task #1 Print Array to Console Sample Output……………………………………………….3

Task #2: Set All Elements in A Row to A Specified Value Code…………………………….6

Task #2: Set All Elements in A Row to A Specified Value Sample Output………………….7

Task #3: Find Minimum Value in the Specified Column Code……………………………....10

Task #3: Find Minimum Value in the Specified Column Output………………………….....11

Task #4 Find Frequency of a Given Value in the Array Code………………………………..13

Task #4 Find Frequency of a Given Value in the Array Output………………………………14

TwoDimArrayPractice Code………………………………………………………………….16

**Task #1 Print Array to Console Code**

**public** **void** **printArray**( )

{

// Note: To animate the algorithm, put this method call as the

// last element in your inner for loop

// animate( row, column );

// where row is the index of the array's current row

// and column is the index of the array's current column

// Write your code here:

**int** row;

**for** (row = **0**; row < intArray.length; row++) {

**for**(**int** column = **0**; column <intArray[row].length; column++) {

System.out.print(intArray[row][column]+ " ");

animate(row,column);

}

}

**Task #1 Print Array to Console Sample Output**

The section down below is the sample output in console after the **printArray() method** was implemented.

------------------------------------------------------------------------------------------------------------------

**Sample Output from Console**

**------------------------------------------------------------------------------------------------------------------**

Row Value

0 58 70 54 74 71 74 53 57 65 69 76 63 72 73 77 53 50 62 55 57

1 51 56 76 80 55 52 56 73 75 70 57 65 56 53 74 53 77 51 50 77

2 56 68 79 71 58 78 78 59 75 52 72 79 54 77 57 63 68 56 62 67

3 62 55 52 60 64 55 50 56 67 54 65 80 78 71 67 67 66 73 66 58

58 70 54 74 71 74 53 57 65 69 76 63 72 73 77 53 50 62 55 57 51 56 76 80 55 52 56 73 75 70 57 65 56 53 74 53 77 51 50 77 56 68 79 71 58 78 78 59 75 52 72 79 54 77 57 63 68 56 62 67 62 55 52 60 64 55 50 56 67 54 65 80 78 71 67 67 66 73 66 58

**------------------------------------------------------------------------------------------------------------------**

The following screenshots depict the sample output of the program’s GUI. Figure 1 and figure 2 displays the output from the **printArray( ) method.** The implementation was a success because the **printArray() method** was able to meet the requirements for task#1 which means that the method has been implemented correctly**.**

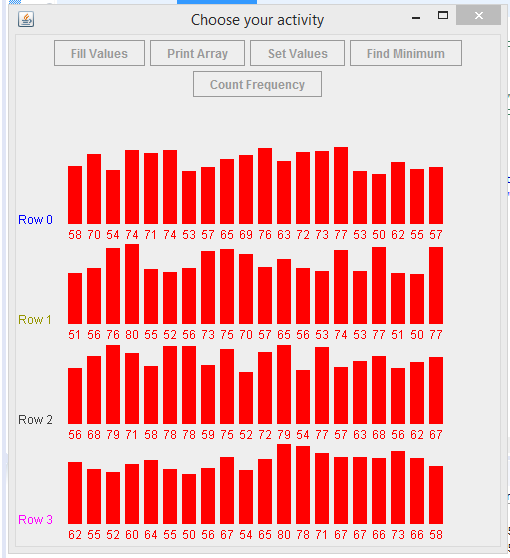
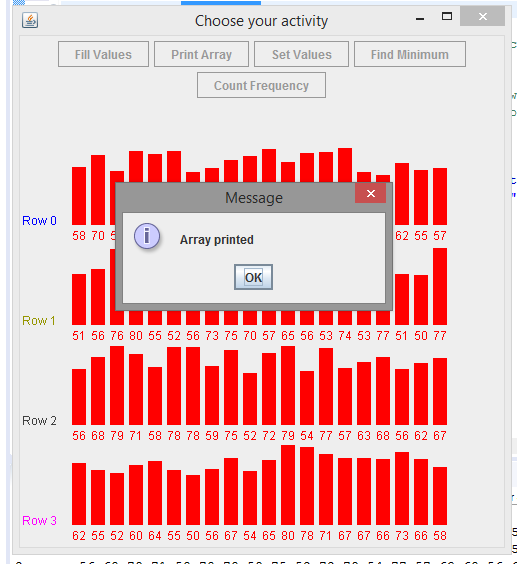
****

Figure 1. Sample output of the program’s GUI displaying correct results



*Figure 2.* Sample output of the program’s GUI displaying the correct message to user after the array has been printed.

**Task #2: Set All Elements in A Row to A Specified Value Code**

**public** **void** **setValues**( **int** value, **int** row )

{

// Note: To animate the algorithm, put this method call as the

// last element in your for loop

// animate( row, column );

// where row is the index of the array's current row

// and column is the index of the array's current column

// Write your code here:

**int** column;

**for**(column = **0**; column < intArray[row].length; column++){

intArray[row][column] = value;

animate(row, column);

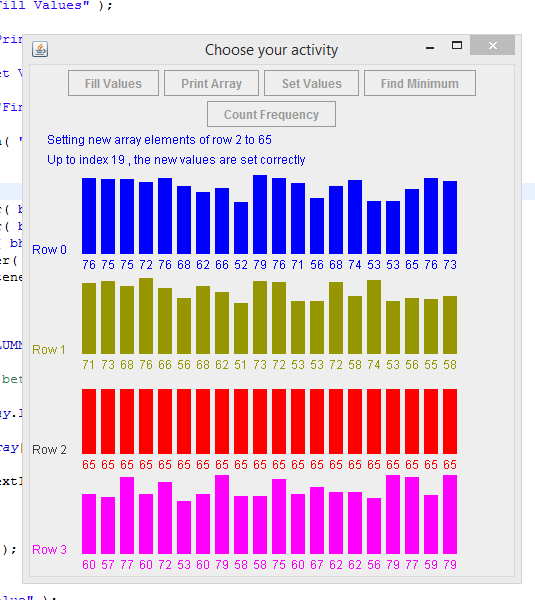
}

}

// end of setValues method

**Task #2: Set All Elements in A Row to A Specified Value Sample Output**

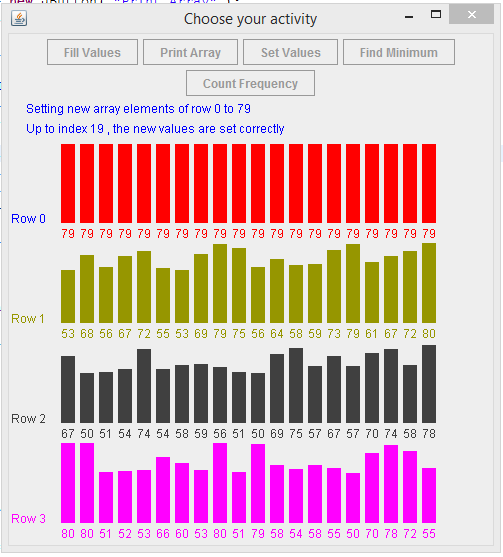
The following screenshots depict the sample output of the program’s GUI for task #2. Figure 1 and figure 2 displays the output of **setValues(65,2) method**. The implementation was a success because the **setValues(65,2) method** was able to meet the requirements for task#2 . Figure 3 and figure 4 is just another test conducted to ensure that implementation of the **setValues() method** works as intended.



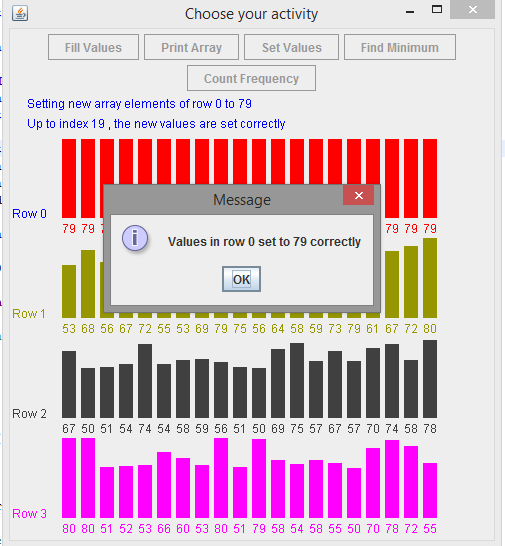
*Figure 1.* Screenshot displays the required output from **setValues(65,2) method.**



*Figure 2.* Screenshot displays correct message prompt to user.



*Figure 3.* Screenshot displays the required output from **setValues(79,0) method**.



*Figure 4.* Screenshot displays correct message prompt to user.

**Task #3: Find Minimum Value in the Specified Column Code**

**public** **int** **findMinimum**( **int** column )

{

// Note: To animate the algorithm, put this method call as the

// last element in your for loop

// animate( row, column, minimum );

// where row is the index of the array's current row,

// column is the index of the array's current column

// minimum is the local variable storing the current minimum

// Write your code here:

**int** minimum = intArray[**0**][**0**];

**int** row;

**for**(row = **0**; row < intArray.length; row++){

**if**(intArray[row][column] < minimum){

minimum = intArray[row][column];

}

animate (row, column, minimum);

}

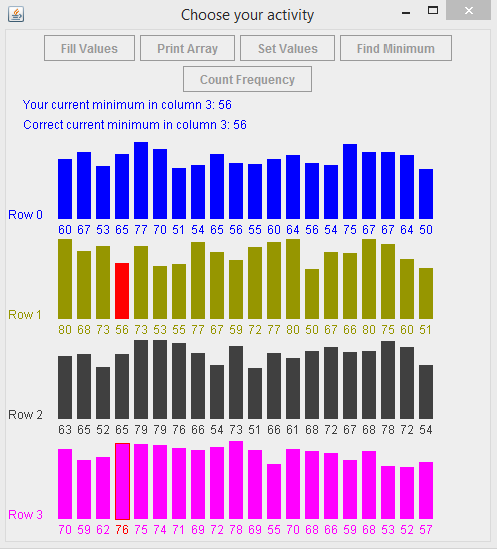
**return** minimum; // replace this line with your return statement

}

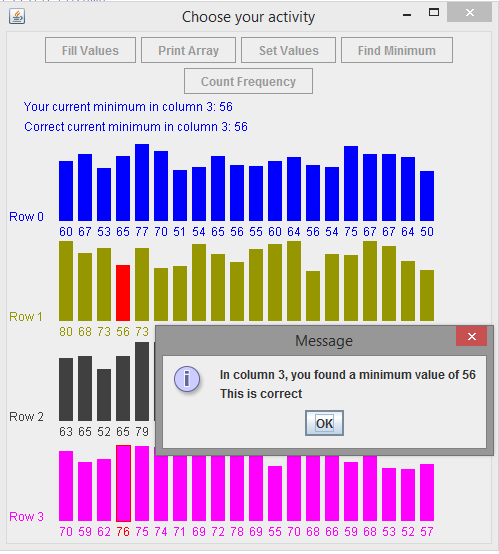
// end of findMinimumn method

**Task #3: Find Minimum Value in the Specified Column Output**

The following screenshots display the sample output of the program’s GUI for task #3. Figure 1 and figure 2 displays the output of **findMinimum(3) method**. The implementation was a success because the **findMinimum(3) method** was able to meet the requirements for task#3. The program found the minimum number of the column and displayed a message saying the minimum was correct.



*Figure 1.* Screenshot displays the output for **findMinimum(3) method.**



*Figure 2.* Screenshot displays minimum value of the column being correct to user.

**Task #4 Find Frequency of a Given Value in the Array Code**

**public** **int** **countFound**( **int** value )

{

// Note: To animate the algorithm, put this method call as the

// last element in your inner for loop

// animate( row, column, num );

// where row is the index of the array's current row,

// column is the index of the array's current column, and

// num is the local variable storing the current frequency count

// Write your code here:

**int** num = **0**;

**int** row;

**for**(row = **0**; row < intArray.length; row++){

**for**(**int** column = **0**; column < intArray[row].length; column++){

**if**(intArray[row][column] == value){

num++;

}

animate(row, column, num);

}

}

**return** num; // replace this line with your return statement

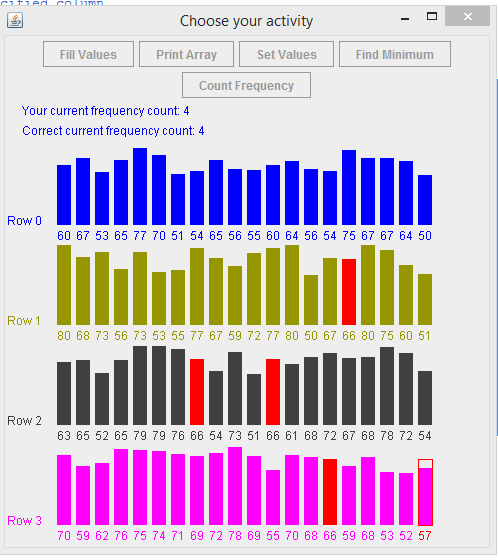
}

// end of countFound method

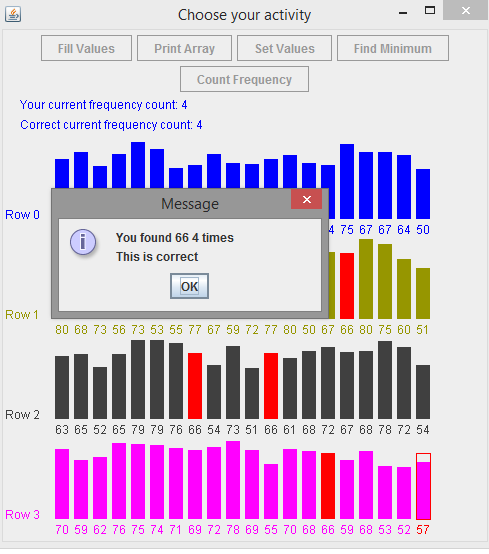
**Task #4 Find Frequency of a Given Value in the Array Output**

The following screenshots display the sample output of the program’s GUI for task #4.

Figure 1 and figure 2 displays the output of **countFound(66) method.** The implementation was a success because the **countFound(66) method** was able to meet the requirementsfor task #4. The method counted the number of times 66 was found in the graph correct.



*Figure 1.* Screenshot displays the number of times 66 was found in the graph.



*Figure 2.* Screenshot displays the message prompt to user saying how many times the number 66 was found in the bar graph.

**TwoDimArrayPractice Code**

/\* TwoDimArrayPractice

\* Students Work with this Java file

\* On Multi-dimensional Arrays

\*/

/\*\*

\* TwoDimArrayPractice

\* Students Work with this Java file On Multi-dimensional Arrays

\* @Author Josue Ponce

\* @Date 4/4/18

\* @Version 1.0

\*/

**import** **java.awt.\***;

**import** **javax.swing.\***;

**import** **javax.swing.JOptionPane**;

**import** **java.awt.event.\***;

**import** **java.util.\***;

**public** **class** **TwoDimArrayPractice** **extends** JFrame

{

// GUI components

**private** JButton fillValues;

**private** JButton printArray;

**private** JButton setValues;

**private** JButton findMinimum;

**private** JButton countFrequency;

**private** ButtonHandler bh;

**private** **static** **int** [][] intArray;

**private** **final** **int** ROWS = **4**;

**private** **final** **int** COLUMNS = **20**;

**private** **static** **int** current1 = -**1**;

**private** **static** **int** current2 = -**1**;

**private** **int** key;

**private** **int** rowSelected = -**1**;

**private** **int** columnSelected = -**1**;

**private** BarChart bc;

**private** **static** **int** counter = **0**;

**private** **static** TwoDimArrayPractice app;

**private** **boolean** firstTime = **true**;

**private** Image offscreen;

**public** **TwoDimArrayPractice**( )

{

**super**( "Choose your activity" );

Container c = getContentPane( );

c.setLayout( **new** FlowLayout( ) );

fillValues = **new** JButton( "Fill Values" );

c.add( fillValues );

printArray = **new** JButton( "Print Array" );

c.add( printArray );

setValues = **new** JButton( "Set Values" );

c.add( setValues );

findMinimum = **new** JButton( "Find Minimum" );

c.add( findMinimum );

countFrequency = **new** JButton( "Count Frequency" );

c.add( countFrequency );

bh = **new** ButtonHandler( );

fillValues.addActionListener( bh );

printArray.addActionListener( bh );

setValues.addActionListener( bh );

findMinimum.addActionListener( bh );

countFrequency.addActionListener( bh );

setSize( **500**,**550** );

intArray = **new** **int**[ROWS][COLUMNS];

// fill with random numbers between 50 and 80

Random rand = **new** Random( );

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

{

**for** ( **int** j = **0**; j < intArray[**0**].length; j++ )

{

intArray[i][j] = rand.nextInt( **31** ) + **50**;

}

}

bc = **new** BarChart( intArray );

// print the array values

System.out.println( "Row\tValue" );

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

{

System.out.print( i + "\t" );

**for** ( **int** j = **0**; j < intArray[i].length; j++ )

{

System.out.print( intArray[i][j] + " " );

}

System.out.println( );

}

System.out.println( );

setVisible( **true** );

offscreen = **this**.createImage( getSize( ).width, getSize( ).height );

}

// 1. This method has been coded as an example

/\*\* Fills the array with random numbers between 50 and 80

\* The instance variable named intArray is the integer array to be

\* filled with values

\*/

**public** **void** **fillValues**( )

{

Random rand = **new** Random( );

**for** ( **int** row = **0**; row < intArray.length; row++ )

{

System.out.print( row + "\t" );

**for** ( **int** column = **0**; column < intArray[row].length; column++ )

{

intArray[row][column] = rand.nextInt( **31** ) + **50**;

animate( row, column ); // needed to create visual feedback

}

System.out.println( );

}

}

// end of fillValues method

// 2. Student writes this method

/\*\* Prints array to the console, elements are separated by a space

\* The instance variable named intArray is the integer array to be printed

\*/

**public** **void** **printArray**( )

{

// Note: To animate the algorithm, put this method call as the

// last element in your inner for loop

// animate( row, column );

// where row is the index of the array's current row

// and column is the index of the array's current column

// Write your code here:

**int** row;

**for** (row = **0**; row < intArray.length; row++) {

**for**(**int** column = **0**; column <intArray[row].length; column++) {

System.out.print(intArray[row][column]+ " ");

animate(row,column);

}

}

}

// end of printArray method

// \*\*\*\*\* 3. Student writes this method

/\*\* Sets all the elements in the specified row to the specified value

\* The instance variable named intArray is the integer array

\* @param value the value to assign to the element of the row

\* @param row the row in which to set the elements to value

\*/

**public** **void** **setValues**( **int** value, **int** row )

{

// Note: To animate the algorithm, put this method call as the

// last element in your for loop

// animate( row, column );

// where row is the index of the array's current row

// and column is the index of the array's current column

// Write your code here:

**int** column;

**for**(column = **0**; column < intArray[row].length; column++){

intArray[row][column] = value;

animate(row, column);

}

}

// end of setValues method

// 4. Student writes this method

/\*\* Finds minimum value in the specified column

\* The instance variable named intArray is the integer array

\* @param column the column to search

\* @return the minimum value found in the column

\*/

**public** **int** **findMinimum**( **int** column )

{

// Note: To animate the algorithm, put this method call as the

// last element in your for loop

// animate( row, column, minimum );

// where row is the index of the array's current row,

// column is the index of the array's current column

// minimum is the local variable storing the current minimum

// Write your code here:

**int** minimum = intArray[**0**][**0**];

**int** row;

**for**(row = **0**; row < intArray.length; row++){

**if**(intArray[row][column] < minimum){

minimum = intArray[row][column];

}

animate (row, column, minimum);

}

**return** minimum; // replace this line with your return statement

}

// end of findMinimumn method

// 5. Student writes this method

/\*\* Finds the number of times value is found in the array

\* The instance variable named intArray is the integer array

\* @param value the value to count

\* @return the number of times value was found

\*/

**public** **int** **countFound**( **int** value )

{

// Note: To animate the algorithm, put this method call as the

// last element in your inner for loop

// animate( row, column, num );

// where row is the index of the array's current row,

// column is the index of the array's current column, and

// num is the local variable storing the current frequency count

// Write your code here:

**int** num = **0**;

**int** row;

**for**(row = **0**; row < intArray.length; row++){

**for**(**int** column = **0**; column < intArray[row].length; column++){

**if**(intArray[row][column] == value){

num++;

}

animate(row, column, num);

}

}

**return** num; // replace this line with your return statement

}

// end of countFound method

**public** **void** **startActivity**( **int** act )

{

bc.setActivity( act );

**boolean** goodInput = **false**;

String answer = "";

**switch**( act )

{

**case**( **0** ): fillValues( );

JOptionPane.showMessageDialog( **null**, "Array filled with new values" );

**break**;

**case**( **1** ): printArray( );

JOptionPane.showMessageDialog( **null**, "Array printed" );

**break**;

**case**( **2** ):

**while** ( !goodInput || key < **50** || key > **80** )

{

**try**

{

answer = JOptionPane.showInputDialog( **null**, "Enter a value between 50 and 80" );

**if** ( answer != **null** )

{

key = Integer.parseInt( answer );

goodInput = **true**;

}

**else**

{

goodInput = **false**;

**break**;

}

}

**catch**( Exception e )

{}

}

**if** ( goodInput )

{

goodInput = **false**;

**while** ( !goodInput || rowSelected < **0** || rowSelected > **3** )

{

**try**

{

answer = JOptionPane.showInputDialog( **null**, "Enter a row number between 0 and 3" );

**if** ( answer != **null** )

{

rowSelected = Integer.parseInt( answer );

goodInput = **true**;

}

**else**

{

goodInput = **false**;

**break**;

}

}

**catch**( Exception e )

{}

}

}

**if** ( goodInput )

{

bc.setKey ( key );

setValues( key, rowSelected );

String message = "";

**if** ( bc.getCheckNewValues( ) )

message = " correctly";

**else**

message = " incorrectly";

JOptionPane.showMessageDialog( **null**, "Values in row " + rowSelected + " set to " + key + message );

}

**break**;

**case**( **3** ):

**while** ( !goodInput || columnSelected < **0** || columnSelected > **19** )

{

**try**

{

answer = JOptionPane.showInputDialog( **null**, "Enter a column number between 0 and 19" );

**if** ( answer != **null** )

{

columnSelected = Integer.parseInt( answer );

goodInput = **true**;

}

**else**

{

goodInput = **false**;

**break**;

}

}

**catch**( Exception e )

{}

}

**if** ( goodInput )

{

**int** a = findMinimum( columnSelected );

String feedbackMin = "";

**if** ( a == bc.getExactMinimum( ) )

feedbackMin = "\nThis is correct";

**else**

feedbackMin = "\nThis is incorrect";

String displayMessageMin = "In column " + columnSelected + ", you found a minimum value of ";

displayMessageMin += a + feedbackMin;

JOptionPane.showMessageDialog( **null**, displayMessageMin );

}

**break**;

**case**( **4** ):

**while** ( !goodInput || key < **50** || key > **80** )

{

**try**

{

answer = JOptionPane.showInputDialog( **null**, "Enter a value between 50 and 80" );

**if** ( answer != **null** )

{

key = Integer.parseInt( answer );

goodInput = **true**;

}

**else**

{

goodInput = **false**;

**break**;

}

}

**catch**( Exception e )

{}

}

**if** ( goodInput )

{

**int** frequency = countFound( key );

String feedbackFrequency = "";

**if** ( frequency == bc.getExactFrequencyCount( ) )

feedbackFrequency = "\nThis is correct";

**else**

feedbackFrequency = "\nThis is incorrect";

String plural = "";

**if** ( frequency != **1** )

plural = "s";

String displayMessageFrequency = "You found " + key + " " + frequency + " time" + plural;

displayMessageFrequency+= feedbackFrequency;

**if** ( frequency != -**1** )

JOptionPane.showMessageDialog( **null**, displayMessageFrequency );

**else**

JOptionPane.showMessageDialog( **null**, "You did not find the value " + key );

}

**break**;

}

enableButtons( );

}

**public** **static** **int** **getCurrent1**( )

{

**return** current1;

}

**public** **static** **int** **getCurrent2**( )

{

**return** current2;

}

**public** **static** **int** **getCounter**( )

{

**return** counter;

}

**public** **static** **int** [][] getArray( )

{

**return** intArray;

}

**private** **void** **animate**( **int** row, **int** column )

{

**if** ( bc.getActivity( ) >= **0** && bc.getActivity( ) <= **2** )

{

**try**

{

current1 = row;

current2 = column;

bc.setArray( intArray );

Graphics g = offscreen.getGraphics( );

paint( g );

g = **this**.getGraphics( );

g.drawImage( offscreen, **0**, **0**, **this** );

**if** ( bc.getActivity( ) == **0** )

Thread.sleep( **200** );

**else**

Thread.sleep( **500** );

}

**catch** ( InterruptedException e )

{

System.out.println( "IE Exception " + e.getMessage( ) );

System.out.println( e.toString( ) );

}

}

**else**

{

// call to animate with wrong number of arguments

JOptionPane.showMessageDialog( **null**, "Wrong number of arguments to animate method" );

System.exit( **1** );

}

}

**private** **void** **animate**( **int** row, **int** column, **int** intermedResult )

{

**if** ( bc.getActivity( ) == **3** || bc.getActivity( ) == **4** )

{

**try**

{

current1 = row;

current2 = column;

bc.setStudentResult( intermedResult );

bc.setArray( intArray );

Graphics g = offscreen.getGraphics( );

paint( g );

g = **this**.getGraphics( );

g.drawImage( offscreen, **0**, **0**, **this** );

Thread.sleep( **500** );

}

**catch** ( InterruptedException e )

{

System.out.println( "IE Exception " + e.getMessage( ) );

System.out.println( e.toString( ) );

}

}

**else**

{

// call to animate has wrong number of arguments

JOptionPane.showMessageDialog( **null**, "Wrong number of arguments to animate method" );

System.exit( **1** );

}

}

**public** **void** **paint**( Graphics g )

{

**if** ( ( current1 != -**1** && current2 != -**1** ) || firstTime )

{

**super**.paint( g );

bc.draw( g );

bc.updateBarChart( key, current1, current2, g );

firstTime = **false**;

}

}

**public** **static** **void** **main**( String [] args )

{

app = **new** TwoDimArrayPractice( );

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

}

**public** **void** **disableButtons**( )

{

fillValues.setEnabled( **false** );

printArray.setEnabled( **false** );

setValues.setEnabled( **false** );

countFrequency.setEnabled( **false** );

findMinimum.setEnabled( **false** );

}

**public** **void** **enableButtons**( )

{

fillValues.setEnabled( **true** );

printArray.setEnabled( **true** );

setValues.setEnabled( **true** );

countFrequency.setEnabled( **true** );

findMinimum.setEnabled( **true** );

}

**private** **class** **ButtonHandler** **implements** ActionListener

{

**private** **boolean** on = **true**;

**public** **void** **actionPerformed**( ActionEvent e )

{

PrintArrayT t = **new** PrintArrayT( app );

**if** ( e.getSource( ) == fillValues )

{

disableButtons( );

fillValues.requestFocus( );

bc.setActivity( **0** );

disableButtons( );

t.start( );

}

**else** **if** ( e.getSource( ) == printArray )

{

disableButtons( );

printArray.requestFocus( );

bc.setActivity( **1** );

t.start( );

}

**else** **if** ( e.getSource( ) == setValues )

{

disableButtons( );

setValues.requestFocus( );

bc.setActivity( **2** );

t.start( );

}

**else** **if** ( e.getSource( ) == findMinimum )

{

disableButtons( );

findMinimum.requestFocus( );

bc.setActivity( **3** );

t.start( );

}

**else** **if** ( e.getSource( ) == countFrequency )

{

disableButtons( );

countFrequency.requestFocus( );

bc.setActivity( **4** );

t.start( );

}

}

}

**public** **void** **resetButtonSelection**( )

{

fillValues.setSelected( **false** );

printArray.setSelected( **false** );

setValues.setSelected( **false** );

findMinimum.setSelected( **false** );

countFrequency.setSelected( **false** );

}

**private** **class** **PrintArrayT** **extends** Thread

{

**int** [][] arr;

TwoDimArrayPractice s1;

**public** **PrintArrayT** ( TwoDimArrayPractice s )

{

arr = TwoDimArrayPractice.intArray;

s1 = s;

}

**public** **void** **run**( )

{

startActivity( bc.getActivity( ) );

enableButtons( );

// deselectButtons( );

}

}

}