Arrays

An *array* is a container object that holds a fixed number of values of the same type. You could think of an array as one variable name that can hold an unlimited number of values. The length of an array is defined when the array is created. An array that holds five values is displayed in the figure below. Each value in the array is assigned an index number. An array always starts with zero (0). In this example the 5th value, A is assigned to index number four.

0	1	2	3	4
A	B	C	D	F

← The length of the array →

In the code example below there are two different array types defined. On line 6 an array of integers has been defined called **grades.** It does not matter which side of the name of the array the square brackets go. The statement can be written like this: int grades[]; You will see code examples with arrays written both ways. It is a matter of preference in which way you choose to write the array statement.

On line 7 the array is initialized with the size or length of 30. This would mean that the index numbers assigned to the values would be 0 – 29.

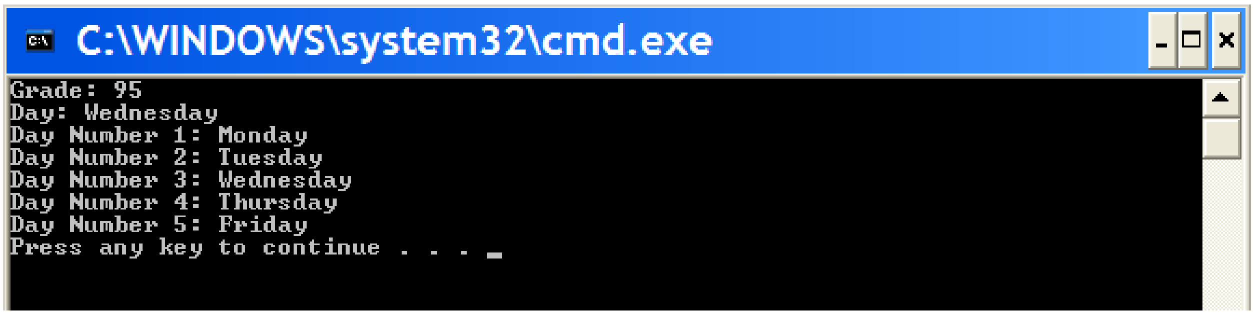
On line 9 the place holder or index zero (0) is assigned the integer 95.

```
 1  class ArrayExamples
 2  {
 3
 4  public static void main(String[] args)
 5     {
 6      int[] grades;          //Defines the array as an integer
 7      grades = new int[30]; //initializes the array with the size of 30
 8
 9      grades[0] = 95;
10
11      String[] daysOfWeek = {"Monday", "Tuesday", "Wednesday",
12                             "Thursday", "Friday"};
13
14      System.out.println("Grade: " + grades[0]);
15      System.out.println("Day: " + daysOfWeek[2]);
16
17      for(int x = 0; x < daysOfWeek.length; x++)
18      {
19          System.out.println("Day Number " + (x+1) + ": " + daysOfWeek[x]);
20      }
21    }
22
23  }
24
```

Line 11 defines an array of strings, initializes and assigns the values all in one statement. The days of the week are enclosed in the curly brackets and because they are string values they have double quotations around them.

Lines 14 and 15 are printing out the index of 0 for grades, the only one that a value is assigned to in this example and daysOfWeek's index 2. The results will print Grade: 95, Day Wednesday.

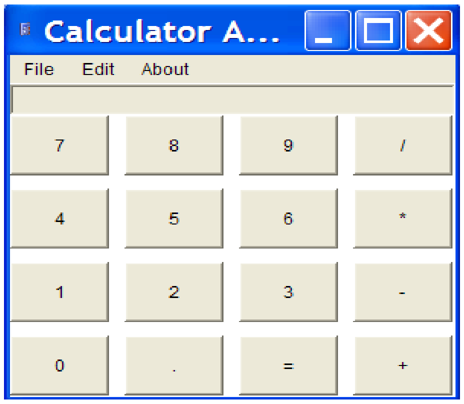
Line 17 uses the for loop with the length command so that all values in the array will be printed out. On line 19 there is a calculation of **x+1** to start off the Number of the week with 1 as shown in the code results below.



```
C:\WINDOWS\system32\cmd.exe
Grade: 95
Day: Wednesday
Day Number 1: Monday
Day Number 2: Tuesday
Day Number 3: Wednesday
Day Number 4: Thursday
Day Number 5: Friday
Press any key to continue . . .
```

Button Arrays and Menu System

In this code example we will use some new features and concepts to create a calculator as shown in the figure below.



Private Variables

This application uses private variables to ensure that there will be no interference from any other class. These variables can only be used by this application and are define on lines 15 -23.

```
 1  /*
 2      Assignment 8: A Java Calculator
 3      Filename:   Calculator.java
 4      Purpose:    This program creates a calculator with a menu.
 5  */
 6
 7  import java.awt.*;
 8  import java.awt.event.*;
 9  import java.awt.datatransfer.*;
10  import java.text.DecimalFormat;
11  import javax.swing.JOptionPane;
12
13  public class Calculator extends Frame implements ActionListener
14  {
15     private Button keys[];
16     private Panel keypad;
17     private TextField lcd;
18     private double op1;
19     private boolean first;
20     private boolean foundKey;
21     private boolean clearText;
22     private int lastOp;
23     private DecimalFormat calcPattern;
```

This application will also use **java.awt.datatransfer.*** to be used the menu system Edit feature to copy and paste data. The **DecimalFormat** will be used as a variable **calcPattern** to hold the pattern for the output display. When this variable is initialized on line 77 it will look like this: calcPattern = new DecimalFormat("########.########");

A **Panel** is an AWT component that serves as an invisible container to further refine the placement of components within the container **Frame**.

A **Button Array** is defined on line 15 to hold the 16 values that will be used in the calculator.

The **Boolean** variable type will be used to set a true or false value when a button is clicked.

Creating the Menu

To create the menu three different components are constructed as shown in the table below.

Java Menu Components		
Menu Component	Description	Constructor Method
MenuBar	Creates a menu bar	MenuBar mnuBar = new MenuBar();
Menu	Creates a menu bar command	Menu mnuEdit = new Menu(“Edit”, true);
MenuItem	Creates a command on a menu	MenuItem mnuEditCopy = new MenuItem(“Copy”);

After the components are constructed, then methods are used to assign and populate the menus. The table below displays some of the more common method used with menus.

Menu Methods		
Method	Description	Example
SetMenuBar()	Automatically displays a previously constructed menu bar at the top of the frame	setMenuBar (myMenu);

add()	Adds a command to the menu bar or menu	myMenu.add(mnuHelp);
		mnuHelp.add(mnuHelpAbout);

addActionListener()	Makes menu clickable	mnuEditCut.addActionListener(this);
setActionCommand()	Sets a String to represent the menu item	mnuEdit.setActionCommand(“Edit”);
remove()	Removes a menu item from the menu bar or menu	Remove(mnuTools);
insertSeparator()	Inserts a horizontal line in the index of the menu	mnuTable.insertSeparator(1); The 1 would be after the first item in the menu

This application will use three commands on the menu bar: File, Edit and About. The File menu item will only contain the Exit command. Edit will have three commands: Clear, Copy and Paste. There is a horizontal separator line between Clear and Copy. The About menu will only contain one command About Calculator.

It is common practice to use the prefix **mnu** when defining and constructing menu systems. The code below constructs and populates the menu used for the Calculator application. Line 28 create the menu bar and assigns the name mnuBar. Line 29 set the menu bar to the frame.

```
24
25      public Calculator()
26      {
27          // create an instance of the menu
28          MenuBar mnuBar = new MenuBar();
29          setMenuBar(mnuBar);
30
31          // construct and populate the File menu
32          Menu mnuFile = new Menu("File", true);
33          mnuBar.add(mnuFile);
34              MenuItem mnuFileExit = new MenuItem("Exit");
35              mnuFile.add(mnuFileExit);
36
37          // construct and populate the Edit menu
38          Menu mnuEdit = new Menu("Edit", true);
39          mnuBar.add(mnuEdit);
40              MenuItem mnuEditClear = new MenuItem("Clear");
41              mnuEdit.add(mnuEditClear);
42              mnuEdit.insertSeparator(1);
43              MenuItem mnuEditCopy = new MenuItem("Copy");
44              mnuEdit.add(mnuEditCopy);
45              MenuItem mnuEditPaste = new MenuItem("Paste");
46              mnuEdit.add(mnuEditPaste);
47
48          // construct and populate the About menu
49          Menu mnuAbout = new Menu("About", true);
50              mnuBar.add(mnuAbout);
51              MenuItem mnuAboutCalculator = new MenuItem("About Calculator");
52              mnuAbout.add(mnuAboutCalculator);
```

In the next lines of code the addActionListener() and setActionCommand() methods are added and set. There is not an ActionListener for the menu bar because one is not needed. They are only needed for items that will be using and event or action.

```
53
54          // add the ActionListener to each menu item
55          mnuFileExit.addActionListener(this);
56          mnuEditClear.addActionListener(this);
57          mnuEditCopy.addActionListener(this);
58          mnuEditPaste.addActionListener(this);
59          mnuAboutCalculator.addActionListener(this);
60
61          // assign an ActionCommand to each menu item
62          mnuFileExit.setActionCommand("Exit");
63          mnuEditClear.setActionCommand("Clear");
64          mnuEditCopy.setActionCommand("Copy");
65          mnuEditPaste.setActionCommand("Paste");
66          mnuAboutCalculator.setActionCommand("About");
```

In the next lines of code the beginning values for the variables are set and the construction of the button array on line 72. The variable calcPattern is assigned to eight decimal places before and after the decimal point.

```
67
68          // construct components and initialize beginning values
69          lcd = new TextField(20);
70              lcd.setEditable(false);
71          keypad = new Panel();
72          keys = new Button[16];
73          first = true;
74          op1 = 0.0;
75          clearText = true;
76          lastOp = 0;
77          calcPattern = new DecimalFormat("########.########");
```

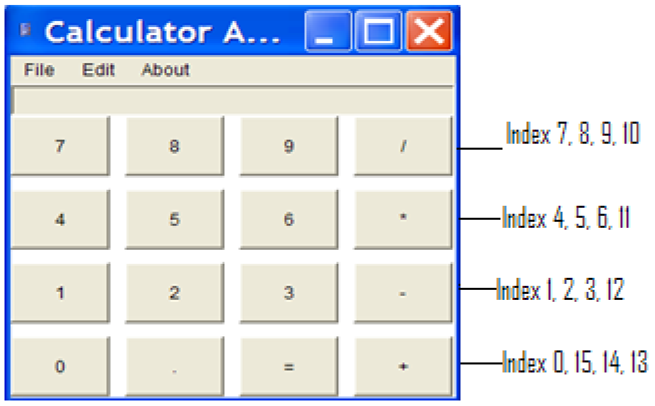
Button Array

The calculator application will use a button array to construct the button 0 – 9, the decimal point, equal sign, addition sign, subtraction sign, multiplication sign and the division sign. There will be 16 buttons needed in all. Using a button array for the button will allow for the same code to be used when the button is clicked. The construction of the numeric keys can be done in a for loop with two lines of code. Other wise it would take 10 lines of code. No curly brackets are needed for the for loop because there is only 2 lines of code. On line 81 String.valueOf(i) method converts the numeric value of I to a string. Now the button values will be labels. The other buttons need to be assigned to the index numbers because each one has a unique string value. Line 91 set the layout of the **Frame** to BorderLayout and line 92 sets the layout of the keypad **Panel** to GridLayout with four rows and four columns. The 10, 10 is the amount of space between the buttons.

```
78
79          // construct and assign captions to the Buttons
80          for (int i=0; i<=9; i++)
81              keys[i] = new Button(String.valueOf(i));
82
83          keys[10] = new Button("/");
84          keys[11] = new Button("*");
85          keys[12] = new Button("-");
86          keys[13] = new Button("+");
87          keys[14] = new Button("=");
88          keys[15] = new Button(".");
89
90          // set Frame and keypad layout to grid layout
91          setLayout(new BorderLayout());
92          keypad.setLayout(new GridLayout(4,4,10,10));
```

Add Components to the Interface

When adding the components to the layout it is helpful to draw the layout on paper and assign the index numbers to each button. The mockup below gives a clear picture of what button goes where. Note that the order of most of the buttons of the last row is in descending order.



The reason for using the for loop in to cut down on the number of lines of code to build the interface. Where each for loop is adding three or four buttons at a time that would save about ten extra lines of code. The for loop on lines 94 and 95 adds the first row of buttons to the grid. The first row will be buttons 7, 8, 9, and /. The second row will be the same adding 4, 5, and 6. On line 100 the index number 11 is adding the multiply sign to the second row. It can not be added in the loop because of the order. In row 3 the same situation occurs where the first three buttons can be added in the loop but the number of the fourth button, 12, can not be added in the sequence and needs to be added on it's own in line 105. The forth row first adds the index 0 on line 107 and then on line 109 in descending order adds the decimal point, equal sign and the plus sign using i—instead of i++.

```
 93
 94          for (int i=7; i<=10; i++) // 7, 8, 9, divide
 95              keypad.add(keys[i]);
 96
 97          for (int i=4; i<=6; i++) // 4, 5, 6
 98              keypad.add(keys[i]);
 99
100          keypad.add(keys[11]); // multiply
101
102          for (int i=1; i<=3; i++) // 1, 2, 3
103              keypad.add(keys[i]);
104
105          keypad.add(keys[12]); // subtract
106
107          keypad.add(keys[0]); // 0 key
108
109          for (int i=15; i>=13; i--)
110              keypad.add(keys[i]); // decimal point, =, add (+) keys
```

In the next lines of code the ActionListener is added to each button using a for loop. The loop on line 112 goes from 0 to the end of the array using the **length** property of the array. The key word **this** refers to control back to it's self or in this case the buttons are the keys. On line 115 the TextField lcd is added to the north region of the BorderLayout and line 116 adds the keypad to the Center region of the BorderLayout.

```
111
112          for (int i=0; i<keys.length; i++)
113              keys[i].addActionListener(this);
114
115          add(lcd, BorderLayout.NORTH);
116          add(keypad, BorderLayout.CENTER);
117
```

The panel is now complete with 16 buttons added to create the keypad. The panel is a composite component or a container with components added to it. When the panel is added to the frame all the buttons are added at once.

On line 118 the WindowListener is added and registered to the Frame. This connects the two objects together so that events from the Frame object are sent to the listener. The listener tells the Frame object to listen for an event and respond accordingly. The argument inside the parentheses for the addWindowListener, lines 119 – 126, creates a new instance of the WindowAdapter class. The WindowAdapter class provides the window event handling. In this case allows the user to click the X to terminate the program.

```
118             addWindowListener(
119                 new WindowAdapter()
120                     {
121                     public void windowClosing(WindowEvent e)
122                         {
123                             System.exit(0);
124                         }
125                     }
126             );
127
128         } // end of constructor method
129
```

This program uses the Clipboard class to copy and paste the calculation results. The clipboard is a temporary area of memory reserved for user storage. The Toolkit class is also used to be able to transfer the data back and forth using cut, copy and paste. The table below displays some of the methods used from the Clipboard.

Methods Used with the Clipboard		
Method	Use	Example
getDefaultToolkit()	Gets the system toolkit if it exists.	Toolkit myTools = Toolkit.getDefaultToolkit();
GetSystemClipboard()	Gets the most resent value of the system clipboard	Clipboard cb = myTools.getSystemClipboard();
getContents()	Returns transferable object from the clipboard	Transferable fromCb = Cb.getContents(this);
setContents()	Sets transferable object from the clipboard	Transferable ToCb = Cb.setContents(this);
getTransferData()	Returns data to be transferred.	String s = (String) Content.getTransferData (DataFlavor.stringFlavor);

Exit and Clear Commands

The code below will be performed if the menu items Exit or Clear are clicked. The getActionCommand is assigned the string value of **arg**. Then on line 134 the **arg** is compared to the key work Exit. If it is a match then the System.exit(0) method is called and the program ends. If **arg** equals Clear then all the variables listed are set to zero or null and the focus is put back to the lcd TextArea so a new calculation can begin.

```
130     public void actionPerformed(ActionEvent e)
131     {
132          //test for menu item clicks
133          String arg = e.getActionCommand();
134          if (arg == "Exit")
135              System.exit(0);
136
137          if (arg == "Clear")
138          {
139              clearText = true;
140              first = true;
141              op1 = 0.0;
142              lcd.setText("");
143              lcd.requestFocus();
144          }
145
```

If Copy is selected for the menu, then the Clipboard is activated and the contents is sent to the clipboard. When Paste is selected for the menu the contents of the Clipboard is transferred. On line 157 the try block is activated. It is parsed, formatted and then sent to the lcd TextField. If it can't do the action, then the catch is thrown, an error and the lcd is set to null. If the About menu item is selected Then a message box will be shown with the text that has been assigned to the message.

```
146          if (arg == "Copy")
147          {
148              Clipboard cb = Toolkit.getDefaultToolkit().getSystemClipboard();
149              StringSelection contents = new StringSelection(lcd.getText());
150              cb.setContents(contents, null);
151          }
152
153          if (arg == "Paste")
154          {
155              Clipboard cb = Toolkit.getDefaultToolkit().getSystemClipboard();
156              Transferable content = cb.getContents(this);
157              try
158              {
159                  String s = (String)content.getTransferData(DataFlavor.stringFlavor);
160                  lcd.setText(calcPattern.format(Double.parseDouble(s)));
161              }
162              catch (Throwable exc)
163              {
164                  lcd.setText("");
165              }
166          }
167
168          if (arg == "About")
169          {
170              String message = "Calculator ver. 2.0\nOpenSource Software
171              \nCopyright 2009\nAll rights reserved";
172              JOptionPane.showMessageDialog(null,message,
173              "About Calculator", JOptionPane.INFORMATION_MESSAGE);
174          }
```

Number Button

Line 177 test if a button has been clicked. The foundKey is set to false first to prevent errors in the later code. If a user clicks in the frame or title bar it may be set to true otherwise. Next the array is searched to see of a button has been selected on line 180. If the source stored in the variable **e** matches one of the buttons the code inside the block begins. Java uses a switch case statement to test a variable against multiple possibilities. Typically each case ends with a break command that is displayed on line 196. In this case where the user can click on a button 0 – 9 and the period on line 188 all of the cases are together. This will allow the same function for any on the keys. First the lcd is set to null and then the clearText is set to false. On line 195 the label of the button is displayed in the lcd. If multiply buttons are selected the getLabel() method is concatenated with the previous text in the lcd.

```
175
176         // test for button clicks
177         foundKey = false;
178
179         // search for the clicked key
180         for (int i=0; i<keys.length && !foundKey; i++)
181         {
182             if(e.getSource() == keys[i])
183             {
184                 foundKey=true;
185                 switch(i)
186                 {
187                     // number and decimal point buttons
188                     case 0: case 1: case 2: case 3: case 4: case 5: case 6: case 7:
189                     case 8: case 9: case 15:
190                         if(clearText)
191                         {
192                             lcd.setText("");
193                             clearText = false;
194                         }
195                         lcd.setText(lcd.getText() + keys[i].getLabel());
196                         break;
197
```

First Operator Button Click

The array is searched for the operator buttons and adds functionality to the buttons within the actionPerformed() method. On line 199 if a case is selected that would mean a user has selected one of the buttons with the index values of 10 – 14. On line 200 the clearText is set to true. The value from the lcd is parsed and then stored in the variable op1. On line 207 the first variable is set to false and the next line the clearText is set to true and the index of the operator is saved in lastOp for later use.

```
198                     // operator buttons
199                     case 10: case 11: case 12: case 13: case 14:
200                         clearText = true;
201
202                         if (first) // first operand
203                         {
204                             if(lcd.getText().length()==0) op1 = 0.0;
205                             else op1 = Double.parseDouble(lcd.getText());
206
207                             first = false;
208                             clearText = true;
209                             lastOp = i; // save last operator
210                         }
```

Second Operator and Equal Sign

Line 211 will execute if the variable called **first** is false. Line 213 continues with the switch structure using the variable **lastOp**. The program test what operator has been selected with each case. Depending on which operator has been selected, the input is parsed and the arithmetic is preformed and stored in the variable **op1**. Line 228 formats the results and set clearText to true. There is no case for the equal button. Because there is no case 14 the code will pass directly to line 231 skipping all of the other code if no other button has been selected.

```
211                         else  // second operand
212                         {
213                             switch(lastOp)
214                             {
215                                 case 10: // divide button
216                                     op1 /= Double.parseDouble(lcd.getText());
217                                     break;
218                                 case 11: // multiply button
219                                     op1 *= Double.parseDouble(lcd.getText());
220                                     break;
221                                 case 12: // minus button
222                                     op1 -= Double.parseDouble(lcd.getText());
223                                     break;
224                                 case 13: // plus button
225                                     op1 += Double.parseDouble(lcd.getText());
226                                     break;
227                             }  // end of switch(lastOp)
228                             lcd.setText(calcPattern.format(op1));
229                             clearText = true;
230
231                             if(i==14) first = true;// equal button
232                             else lastOp = i; // save last operator
233                         } // end else
234                         break;
235                 } // end of switch(i)
236             } // end of if
237         } // end of for
238     } // end of actionPerformed
239
```

The main() Method

The main() method of the Calculator application constructs an instance of the Calculator class and then sets three attributes. Line 243 an instance of the Frame named **f** is constructed. The next line sets the title to the frame. In line 245 setBounds() method sets the frame to display 300 pixels from the top and 300 pixels from the left of the screen and 200 pixels wide and 200 pixels high. The setVisible() method on line 246 will cause the frame to be displayed at the beginning of the application. The setVisible() method can be used with frames to determine when you want a frame to be displayed. Line 249 set the image to an icon and assigns the image to the frames title bar.

```
240     public static void main(String args[])
241     {
242         // set frame properties
243         Calculator f = new Calculator();
244         f.setTitle("Calculator Application");
245         f.setBounds(200,200,300,300);
246         f.setVisible(true);
247
248         // set image properties and add to frame
249         Image icon = Toolkit.getDefaultToolkit().getImage("calcImage.gif");
250         f.setIconImage(icon);
251
252     } // end of main
253 } // end of class
```

Programming Assignment

1. Recreate ArrayExamples.
2. Recreate the Calculator application.
3. Extra credit will be given for the following:
    a. Changing the colors of the buttons
    b. Changing the general layout.
    c. Any code implemented that works that we haven’t used yet.