```java
/*****************************************************************************
 *
 * File: CoffeeOrderGUI.java
 *
 * Author: Joshua Wiley
 *
 * Description:  Models a coffee shops ordering system that prints an invoice
 *               to the GUI component or the system console
 *
 * Date: 4-24-15
 *
 * Comments: I made the GUI non-resizable.  See tool tips for help.  Shipping
 *           invoices uses all caps.
 *
 *****************************************************************************/

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.util.Date;
import java.text.*;
import java.time.LocalDate;

public class CoffeeOrderGUI
                             implements ActionListener
{
    /*** Class Constants ***/

    private static final int MAX_SIZE_STRING = 20;
    private static final int MAX_SIZE_STATE  = 2;
    private static final int MAX_SIZE_ZIP    = 5;

    /*** Class Variables ***/

                         //Creates text fields
    private JTextField txtName;
    private JTextField txtStreet;
    private JTextField txtCity;
    private JTextField txtState;
    private JTextField txtZipcode;
    private JTextField txtQtyOrdered;
                     //Creates buttons
    private JButton btnCreate;
    private JButton btnPrint;
    private JButton btnClear;
    private JButton btnExit;
                       //Creates a text area
    private JTextArea txtADisplay = null;
                //Makes accessing these values less messy
    private int zipCode = 0;
    private int qtyOrdered = 0;

    /*** Class Methods ***/
           //displays the GUI
    public CoffeeOrderGUI()
    {
        startCoffeeOrderGUI();
    }
                //Where the magic starts
    public void startCoffeeOrderGUI()
    {
        JFrame jfrWindow = null;
        Container c      = null;

        /*** Create the object ***/

        jfrWindow = new JFrame();
        c = jfrWindow.getContentPane();

```

```
71          /*** Create the GUI components ***/
72
73          buildGUI( c );
74          setWindowAttributes( jfrWindow );
75      }
76                  //Where the magic is built
77      private void buildGUI( Container c )
78      {
79          /*** Local variables ***/
80
81          JPanel pnlInput   = null;
82          JPanel pnlButton  = null;
83          JPanel pnlDisplay = null;
84
85          /*** Sets what the panels to their respective methods will do ***/
86
87          pnlInput   = inputPanel();
88          pnlButton  = buttonPanel();
89          pnlDisplay = displayPanel();
90
91          /*** Adds the panels to the GUI ***/
92
93          c.setLayout( new BoxLayout( c, BoxLayout.PAGE_AXIS ) );
94
95          c.add( pnlInput );
96          c.add( pnlButton );
97          c.add( pnlDisplay );
98      }
99                  //Sets how the magic looks
100     public void setWindowAttributes( JFrame jfrWindow )
101     {
102         jfrWindow.setSize( 800, 700 );
103         jfrWindow.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
104         jfrWindow.setResizable( false );
105         jfrWindow.setLocation( 200, 40 );
106         jfrWindow.setTitle( "MyJava Coffee Outlet" );
107         //jfrWindow.pack();  Pack made my GUI look like someone sat on it
108         jfrWindow.setVisible( true );
109     }
110
111     private JPanel inputPanel()
112     {
113         /*** Local variables ***/
114
115         JPanel pnlInput      = null;
116         JLabel lblName       = null;
117         JLabel lblStreet     = null;
118         JLabel lblCity       = null;
119         JLabel lblState      = null;
120         JLabel lblZipcode    = null;
121         JLabel lblQtyOrdered = null;
122
123         /*** Sets a New panel appearance ***/
124                                     //Use titled border
125         pnlInput = new GUIPanel( 2, "Enter Shipping Information" );
126
127         pnlInput.setLayout( new GridBagLayout() );
128
129         GridBagConstraints c = new GridBagConstraints();
130         c.fill = GridBagConstraints.HORIZONTAL;
131
132         /*** Creates GUI components ***/
133
134         lblName = new GUILabel( "Name:" );
135         txtName = new GUITextField( MAX_SIZE_STRING, true,
136                                     "Full Name i.e. John Smith" );
137         txtName.setText( txtName.getText() );
138
139         lblStreet = new GUILabel( "Street Address:" );
140         txtStreet = new GUITextField( MAX_SIZE_STRING, true,
```

```
                                    "Delivery Address i.e. 1000 College Ave." );
        txtStreet.setText( txtStreet.getText() );

        lblCity = new GUILabel( "City:" );
        txtCity = new GUITextField( MAX_SIZE_STRING, true,
                                   "City i.e. Green Bay" );
        txtCity.setText( txtCity.getText() );

        lblState = new GUILabel( "State:" );
        txtState = new GUITextField( MAX_SIZE_STATE, true,
                                    "U.S. Abbreviation i.e. WI " );
        txtState.setText( txtState.getText() );

        lblZipcode = new GUILabel( "Zipcode:" );
        txtZipcode = new GUITextField( MAX_SIZE_ZIP, false,
                                      "U.S. 5 Digit Zipcode in the range [01001-99950]" );
        txtZipcode.setText( txtZipcode.getText() );

        lblQtyOrdered = new GUILabel( "Number of Bags to order:" );
        txtQtyOrdered = new GUITextField( MAX_SIZE_ZIP, false,
                        "Sold in 2 lb. bags for $5.50 per bag.  Must be between ( 1-1500 )" );
        txtQtyOrdered.setText( txtQtyOrdered.getText() );

        /*** Adds the GUI components to the panel ***/

                  //Constraints for lblName
        c.weighty   = .1;
        c.gridx     = 0;
        c.gridy     = 0;
        pnlInput.add( lblName, c );
                  //Constraints for txtName
        c.weightx   = .1;
        c.gridx     = 1;
        c.gridy     = 0;
        c.gridwidth = 2;  //Expands the cell width 2 cells
        pnlInput.add( txtName, c );
                //Constraints for lblStreet
        c.gridx     = 0;
        c.gridy     = 1;
        c.gridwidth = 1;  //Back to default cell width 1 cells
        pnlInput.add( lblStreet, c );
                //Constraints for txtStreet
        c.gridx     = 1;
        c.gridy     = 1;
        c.gridwidth = 2;
        pnlInput.add( txtStreet, c );
                  //Constraints for lblCity
        c.weightx   = 0.1;
        c.gridx     = 0;
        c.gridy     = 2;
        c.gridwidth = 1;
        pnlInput.add( lblCity, c );
                  //Constraints for txtCity
        c.weightx   = 0.1;
        c.gridx     = 1;
        c.gridy     = 2;
        pnlInput.add( txtCity, c );
                  //Constraints for lblState
        c.weightx   = 0.1;
        c.gridx     = 2;
        c.gridy     = 2;
        pnlInput.add( lblState, c );
                  //Constraints for txtState
        c.weightx   = 0.1;
        c.gridx     = 3;
        c.gridy     = 2;
        pnlInput.add( txtState,c );
                  //Constraints for lblZipcode
        c.weightx   = 0.1;
        c.gridx     = 4;
```

```
211             c.gridy     = 2;
212             pnlInput.add( lblZipcode, c );
213                       //Constraints for txtZipcode
214             c.weightx   = 0.1;
215             c.gridx     = 5;
216             c.gridy     = 2;
217             pnlInput.add( txtZipcode, c );
218                       //Constraints for lblQtyOrdered
219             c.weightx   = 0.1;
220             c.gridx     = 1;
221             c.gridy     = 3;
222             pnlInput.add( lblQtyOrdered, c );
223                     //Constraints for txtQtyOrdered
224             c.gridx    = 2;
225             c.gridy    = 3;
226             pnlInput.add( txtQtyOrdered, c );
227 
228             return pnlInput;
229         }
230 
231         private JPanel buttonPanel()
232         {
233             /*** Local variables ***/
234 
235             JPanel pnlButton = null;
236 
237             pnlButton = new GUIPanel( 1, "" );  //Use untitled border
238 
239             pnlButton.setLayout( new FlowLayout() );
240 
241             /*** sets the name of the button and enables the action listener ***/
242 
243             btnCreate = new GUIButton( "Create Invoice" );
244             btnCreate.addActionListener( this );
245 
246             btnPrint  = new GUIButton( "Print Invoice" );
247             btnPrint.addActionListener( this );
248 
249             btnClear  = new GUIButton( "Clear" );
250             btnClear.addActionListener( this );
251 
252             btnExit   = new GUIButton( "Exit" );
253             btnExit.addActionListener( this );
254 
255             /*** Adds the GUI components to the panel ***/
256 
257             pnlButton.setBorder( BorderFactory.createEmptyBorder( 15, 15, 15, 15 ) );
258             pnlButton.add( btnCreate );
259             pnlButton.add( btnPrint );
260             pnlButton.add( btnClear );
261             pnlButton.add( btnExit );
262 
263             return pnlButton;
264         }
265 
266         private JPanel displayPanel()
267         {
268             /*** Local variables ***/
269 
270             JPanel pnlOutput = null;
271             pnlOutput   = new GUIPanel( 2, "Invoice" ); //Use titled border
272                               //Sets the display size
273             txtADisplay = new GUITextArea( 19, 52 );
274 
275             /*** Adds the GUI components to the panel ***/
276 
277             pnlOutput.add( txtADisplay );
278 
279             return pnlOutput;
280         }
```

```
281
282       /*-------------------------------------------------------------------
283       * Handle button events -- i.e. user clicks button
284       *-------------------------------------------------------------------*/
285
286       public void actionPerformed( ActionEvent e)
287       {
288           /*** Performs an action when a button is pressed ***/
289
290           if ( e.getSource() == btnCreate)
291           {
292               processInvoice(1);
293           }
294           else if (e.getSource() == btnPrint )
295           {
296               processInvoice(2);
297           }
298           else if (e.getSource() == btnClear )
299           {
300               clearDisplay();
301           }
302           else if (e.getSource() == btnExit )
303           {
304               System.exit(0);
305           }
306       }
307                   //Clears all the fields
308       public void clearDisplay()
309       {
310           txtName.setText( " " );
311           setTextField( txtName, true );
312           txtStreet.setText( " " );
313           setTextField( txtStreet, true );
314           txtCity.setText( " " );
315           setTextField( txtCity, true );
316           txtState.setText( " " );
317           setTextField( txtState, true );
318           txtZipcode.setText( " " );
319           setTextField( txtZipcode, true );
320           txtQtyOrdered.setText( " " );
321           setTextField( txtQtyOrdered, true );
322           txtADisplay.setText( " " );
323       }
324
325                    //The id is passed to so the program knows where to print
326       private void processInvoice( int id )
327       {
328           txtADisplay.setText( " " );
329
330           if ( invoiceValidation() )
331           {
332               displayInvoice( id );
333           }
334       }
335                    //The id is passed to so the program knows where to print
336       private void displayInvoice( int id )
337       {
338           /*** Local Constants ***/
339
340           final int ARRIVAL_DATE = 14;
341
342           final String FORMATSTRING_0 = "%n";
343           final String FORMATSTRING_1 = "%26s  %-1s %n";
344           final String FORMATSTRING_2 = "%26s %13s $%7.2f %n";
345           final String FORMATSTRING_3 = "%26s%n";
346           final String FORMATSTRING_4 = "%30s %3d %5s $%7.2f %n";
347           final String FORMATSTRING_5 = "%18s    %-1s %n";
348           final String FORMATSTRING_6 = "%18s    %-1s, %-1s %5s %n";
349
350           /*** Local Variables ***/
```

```
351
352         String displayString = "";
353         int large = CoffeeOrders.largeBoxesNeeded( qtyOrdered );
354         int medium = CoffeeOrders.mediumBoxesNeeded( qtyOrdered );
355         int small = CoffeeOrders.smallBoxesNeeded( qtyOrdered );
356
357         /*** The date stuff ***/
358
359         SimpleDateFormat dateFormat = new SimpleDateFormat( "MMMMM dd, YYYY" );
360         Date currentDate = new Date();
361
362         String currentDateString = dateFormat.format( currentDate );
363         String arrivalDateString = futureDate( currentDate, ARRIVAL_DATE );
364
365         /*** Display code ***/
366
367         displayString += String.format( FORMATSTRING_0 );
368         displayString += String.format( FORMATSTRING_1, "Customer Name:",
369                                         txtName.getText().toUpperCase() );
370         displayString += String.format( FORMATSTRING_1, "Number of Bags Ordered:",
371                                         txtQtyOrdered.getText() );
372         displayString += String.format( FORMATSTRING_2, "Purchase Price:", " ",
373                                         CoffeeOrders.calculatePurchasePrice( qtyOrdered ) );
374         displayString += String.format( FORMATSTRING_0 );
375         displayString += String.format( FORMATSTRING_3, "Boxes Used:" );
376         displayString += String.format( FORMATSTRING_4, "Large:", large, "Cost:",
377                                         CoffeeOrders.boxSizeCost( large,
378                                                                   CoffeeOrders.LARGE_BOX_PRICE ) );
379         displayString += String.format( FORMATSTRING_4, "Medium:", medium, "Cost:",
380                                         CoffeeOrders.boxSizeCost( medium,
381                                                                   CoffeeOrders.MEDIUM_BOX_PRICE ) );
382         displayString += String.format( FORMATSTRING_4, "Small:", small, "Cost:",
383                                         CoffeeOrders.boxSizeCost( small,
384                                                                   CoffeeOrders.SMALL_BOX_PRICE ) );
385         displayString += String.format( FORMATSTRING_0 );
386         displayString += String.format( FORMATSTRING_0 );
387         displayString += String.format( FORMATSTRING_2, "Total Cost:", " ",
388                                         CoffeeOrders.totalPrice( qtyOrdered,
389                                                                  large, medium, small ) );
390         displayString += String.format( FORMATSTRING_0 );
391         displayString += String.format( FORMATSTRING_1, "Date of Order:",
392                                         currentDateString ); // CoffeeOrders.getDate( 10 ) );
393         displayString += String.format( FORMATSTRING_1, "Expected Date of Arrival:",
394                                         arrivalDateString );
395         displayString += String.format( FORMATSTRING_0 );
396
397         displayString += String.format( FORMATSTRING_5, "Shipping Address:",
398                                         txtStreet.getText().toUpperCase() );
399         displayString += String.format( FORMATSTRING_6, " ", txtCity.getText().toUpperCase(),
400                                         txtState.getText(), txtZipcode.getText().toUpperCase() );
401
402         /*** This code is either display it or print it ***/
403
404         if ( id == 1 )
405             txtADisplay.setText( displayString );
406         if ( id == 2 )
407            System.out.print( displayString );
408     }
409
410     private String futureDate( Date currentDate, int plusDays )
411     {
412         SimpleDateFormat dateFormat = new SimpleDateFormat( "MMMMM dd, YYYY" );
413         Calendar cal = Calendar.getInstance();
414         cal.add( Calendar.DATE, plusDays );
415         Date arrivalDate = cal.getTime();
416
417         String arrivalDateString = dateFormat.format( arrivalDate );
418
419         return arrivalDateString;
420     }
```

```
421
422     private boolean invoiceValidation()
423     {
424         /*** Local variables ***/
425
426         String errorMessage = "";
427         boolean status = false; //Return variable
428         boolean error = false;  //Error checking
429
430         /*** Validates the information before it is displayed ***/
431
432         if ( !CoffeeOrders.validateString( txtName.getText() ) )
433         {
434             errorMessage += "Blank Name \n";
435             setTextField( txtName, false );
436             error = true;
437         }
438         else
439             setTextField( txtName, true );
440
441         if ( !CoffeeOrders.validateString( txtStreet.getText() ) )
442         {
443             errorMessage += "Blank Street Address \n";
444             setTextField( txtStreet, false );
445             error = true;
446         }
447         else
448             setTextField( txtStreet, true );
449
450         if ( !CoffeeOrders.validateString( txtCity.getText() ) )
451         {
452             errorMessage += "Blank City \n";
453             setTextField( txtCity, false );
454             error = true;
455         }
456         else
457             setTextField( txtCity, true );
458
459         if ( txtState.getText().length() < 2 ||
460              !CoffeeOrders.validateState( txtState.getText().substring( 0, 2 ).toUpperCase() ) )
461         {
462             errorMessage += "US Only - State Must be Abbr. i.e. WI \n";
463             setTextField( txtState, false );
464             error = true;
465         }
466         else
467         {                                          //Capitalizes the first two letters entered
468             txtState.setText( txtState.getText().substring( 0, 2 ).toUpperCase() );
469             setTextField( txtState, true );
470         }
471             //Checks the length of the of the text field
472         if ( !CoffeeOrders.validateString( txtZipcode.getText() ) )
473         {
474             errorMessage += "Blank Zipcode \n";
475             setTextField( txtZipcode, false );
476             error = true;
477         }
478         else if ( txtZipcode.getText().trim().length() != 5 )
479         {
480             errorMessage += "Zipcode not 5 digits \n";
481             setTextField( txtZipcode, false );
482             error = true;
483         }
484         else
485          {     //Checks to that the input can be parsed
486             if ( errorInParse( txtZipcode.getText().trim(), 1 ) ||
487                  !CoffeeOrders.validateZipcode( zipCode ))
488             {
489                 errorMessage += "Zipcode Error - US Postal ( 01001-99950) \n";
490                 setTextField( txtZipcode, false );
```

```
491                 error = true;
492             }
493             else
494                 setTextField( txtZipcode, true );
495         }
496             //Checks the length of the of the text field
497         if ( !CoffeeOrders.validateString( txtQtyOrdered.getText() ) )
498         {
499             errorMessage += "Blank Quantity \n";
500             setTextField( txtQtyOrdered, false );
501             error = true;
502         }
503         else
504         {       //Checks to that the input can be parsed
505             if ( errorInParse( txtQtyOrdered.getText().trim(), 2 ) ||
506                  !CoffeeOrders.validateQuantity( qtyOrdered ) )
507             {
508                 errorMessage += "Quantity Must be in range ( 1-1500 ) \n";
509                 setTextField( txtQtyOrdered, false );
510                 error = true;
511             }
512             else
513             {
514                 setTextField( txtQtyOrdered, true );
515             }
516         }
517 
518         if ( error )
519         {                                   //Pop up message
520             JOptionPane.showMessageDialog( null,"Invalid Information Entered !\n" + errorMessage );
521             txtName.requestFocus();
522         }
523         else
524             status = true;
525 
526         return status;
527     }
528 
529     private boolean errorInParse( String parseString, int id )
530     {
531         /*** Local variable ***/
532 
533         boolean error = false;
534 
535         /*** Try/Catch for parsing the input ***/
536 
537         try
538         {
539             if ( id == 1 )
540                 zipCode = Integer.parseInt( parseString );  //converts an integer to string
541             if ( id == 2)
542                 qtyOrdered = Integer.parseInt( parseString );  //converts an integer to string
543         }
544         catch ( NumberFormatException exc )
545         {
546             if ( id == 1 )
547             {
548                 setTextField( txtZipcode, false );
549                 txtZipcode.requestFocus();
550                 txtZipcode.selectAll();
551             }
552             if ( id == 2)
553             {
554                 setTextField( txtQtyOrdered, false );
555                 txtQtyOrdered.requestFocus();
556                 txtQtyOrdered.selectAll();
557             }
558 
559             error = true;
560         }
```

```
561
562         return error;
563     }
564
565     /*** Text field validation colors ***/
566
567     private void setTextField( JTextField errorInField, boolean valid )
568     {
569         if ( valid )
570         {
571             errorInField.setBackground( Color.WHITE );
572             errorInField.setForeground( Color.BLUE );
573
574             /*** Limits the length of the textfield ***/
575
576             if ( errorInField.getText().length() > MAX_SIZE_STRING )
577                 errorInField.setText( errorInField.getText().substring( 0, MAX_SIZE_STRING - 1 ) );
578         }
579         else
580         {
581             errorInField.setBackground( Color.RED );
582             errorInField.setForeground( Color.WHITE );
583             errorInField.setText( errorInField.getText() );
584         }
585     }
586
587     /*** Application ***/
588
589     public static void main( String args[] )
590     {
591         CoffeeOrderGUI test = new CoffeeOrderGUI();
592
593         //test.startCoffeeOrderGUI(); TEST OBJECT
594     }
595 }
```