```
/* CS-112 FINAL PROJECT
   File Name:           ItemGui.java
   Programmer:          James Watkins
   Date Last Modified:  May 19, 2012

   Problem Statement: Define a Gui interface which allows a user to track
   and view items of class Item.  Use a PropertyList ArrayList for
   managing and containing the Items.  User must be able to navigate through,
   add to, and remove items from the PropertyList ArrayList.  User must be
   protected from program failures through input filtering and exception
   handling.

GUI Components
   1. Create a main JFrame with a BorderLayout and a menuBar.
   2. Populate the menuBar with menus: File, Edit and Search.
   3. The File menu will allow the user to load files, save files and export
   the contents of a file to a .txt file.
   4. The Edit menu must allow the user to add Items to the database, remove
   single Items from the database, modify existing Items and clear the entire
   database.
   5. Create a Panel to be applied to the CENTER region of the JFrame, divide
   the JPanel into 1 row and two columns.
   6. On the left column, add a JLable for displaying pictures of Items.
   7. The Right Column of the will be subdivided into two columns and 9 rows.
   These subdivision will be used for labeling and displaying the variables
   associated with the item displayed. One row will be used for displaying
   the total value of all assets.
   8. Create a panel for displaying JButtons allowing navigation through
   the PropertyList's elements.  Apply this panel to the main JFrame's SOUTH
   region.  Divide this Panel into 1 row and 5 columns.
   9. Create the JButtons for navigation. Buttons will allow navigation to
   the first, last, next and previous elements.
****End Of GUI components
*Methods
    1. Provide a method which launches a JOPtionPane Input dialog in order to
    collect and return the String name of a text file.
    2. Provide a method which launches a JOption Pane Input dialog to collect
    and return a user's search criteria.
    3. Provide a method to alert the user to illegal operations using a
    JOptionPane ERROR_MESSAGE dialog.
    4. Provide a method to alert the user of too many failed attempts at data
    entry using JOptionPane ERROR_MESSAGE dialog.
    5. Provide a method with uses a JOptionPane YES_NO dialog to allow the
    user to confirm deletions prior to execution.
    6. Specify a method which retrieves the name of the currentItem's image
    file and then passes that name to be displayed on the main JFrame.
    7. Method createItem() will launch a new JFrame which has labels and
    textfields for the entry of information pertinent to the creation of a
    new Item. The JFrame will have a submission button and a cancel button.
    When one of the buttons is depressed, it fires an event to the
    AddItemListener class.
    8. Specify a method to allow the modification of existing Items.  The
    method behaves just like the createItem() method, except the modify()
    method calls up the variables of the Item currently displayed in the
    main JFrame.  The user modifies existing data and submits via submit or
    cancel buttons.
    9. Method updateJFrame() is used to update the information and images
    displayed on the main JFrame.  It calls up the current element's
    variables and passes them to the JFrame's components.
    10. The clearJFrame() method uses a String parameter to overwrite the
    text fields of the main JFrame.
End of Methods***
*Inner Classes
    1. The text FileListener class is used for exporting the PropertyList
    to a text file. The class collects a String name for the file and then
    passes that name to PropertyList's printTextFile() method.
    2. Class OpenFileListener uses the JFileChooser class permitting the user
```

```
68          to navigate folders and select an input file. Once the user selects a
69          file, its information is loaded into a PropertyList ArrayList and the
70          JFrame is updated to display element zero. The class also uses
71          PropertyList's readFromFile() method for data I/O.
72          3. SaveFileListner uses JFileChooser to allow the user to browse to the
73          a desired destination folder and then specify a file name.  When the
74          user inputs a file name and clicks save, the JFileChooser passes the
75          file name to PropertyList's writeToFile() method.
76          4. EditListener class is wholly associated with the Edit menu and
77          facilitates the addition, removal, modification and erasure of Items of
78          the PropertyList.  The class requests user confirmation before any
79          deletion, and passes all inputs to be verified against the try/catch
80          criteria. The class relies on implementations of PropertyList's
81          ArrayList methods.
82          5. Class SearchListener collects a search query as input from the user
83          and then passes the query to PropertyList searchDataBase() method.
84          6. The MovementListener class is responsible for user navigation throug
85          the PropertyList.  The buttons are associated to the numeric values of
86          ArrayList elements and scale in response to additions and deletions.
87          7. The AddItemListner class is responsible for ensuring data integrity
88          and handel inputs from the createItem() and modifyItem() methods. The
89          class collects data from the text fields of the create and modify JFram
90          using setText() and getText(). Once the inputs are collected they are
91          temporarily assigned to variables and passed through a battery of test
92          inside of try / catch blocks. If an input fails its test, an exception
93          is triggered and the user is given one additional chance to correct the
94          mistake. A subsequent failur terminates the modification or addition.
95          If all data passes verification, then it is passed to either a method
96          for modification of existing elements or a constructor for the creatior
97          of new Items.  New Items are added to the database and updateJFrame is
98          invoked to display the addtion. The total value of all assets is
99          re-calculated and passed to the JFrame.
100
101       Classes needed and Purpose (Input, Processing, Output)
102       String - input, output                    LineBorder - formatting (view)
103       Integer - input, output                   GridLayout - formatting (view)
104       Double - input, output                    BorderLayout - formatting (view
105       JFileChooser - input, output              BorderFactory - formatting (vie
106       JFrame - input, output                    JPanel - formatting (view)
107       JTextField - input, output                Font - formatting (view)
108       File - input, output                      NumberFormat - output
109       JButton - processing                      JLabel - output
110       JMenuBar - Processing                     ImageIcon - output
111       JMenu - Processing                        JmenuItem - Processing
112       ActionListener - processing
113       JOptionPane - input, output, processing
114       InvalidInputException - error handling
115       PropertyList - input, output, processing
116       Item - input, output, processing
117       FileListener - output
118       OpenFileListener - input
119       SaveFileListener - output
120       EditFileListener - processing
121       SearchFileListener - output
122       MovementFileListener - Processing
123       AddItemListener - input
124    */
125    import java.awt.*;
126    import java.awt.event.*;
127    import javax.swing.*;
128    import javax.swing.border.LineBorder;
129    import java.text.NumberFormat;
130    import java.io.*;
131
132    public class ItemGui extends JFrame
133    {
134        public static void main(String [] args)
```

```
    {
        ItemGui window = new ItemGui();
    }
//GUI Components
    private JFrame inputWindow;
    private JPanel textPanel, centerPanel, buttonPanel;
    private JButton first, previous, next, last, addButton, cancelButton;
    private JTextField description, make, model, serial, year, price, qty;
    private JTextField descriptionText, makeText, modelText, serialText,
                       yearText, priceText, qtyText, pictureText, totalValu
    private JLabel imageLabel;
    private LineBorder trim;
    private JMenuBar menuBar;
    private JMenu file, edit, search;
    private ImageIcon currentImage;
    private Font labelFont;
//variables used in program
    private final int WIDTH = 700, SMALL_WIDTH = 375;
    private final int HEIGHT = 500, SMALL_HEIGHT = 300;
    private PropertyList localList = null;
    private Item currentItem;
    private int location;
    private String fileName;
    private boolean modifyNotAdd = false;

    NumberFormat money = NumberFormat.getCurrencyInstance();

    public ItemGui()
    {
        super("Inventory Management");
        setSize(WIDTH, HEIGHT);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setResizable(false);
        setLocationRelativeTo(null);

        BorderLayout manager = new BorderLayout();
        setLayout(manager);

//layout for center panel
        GridLayout centerLayout = new GridLayout(1,2,10,10);
//layout for button panel
        GridLayout buttonLayout = new GridLayout(1, 5, 10, 10);
//layout for text area
        GridLayout textAreaLayout = new GridLayout(9,2,5,5);
//set a font profile for JFrame
        labelFont = new Font("COURIER", Font.BOLD, 14);
        trim = (LineBorder)BorderFactory.createLineBorder(Color.BLACK, 2);

//Create menuBars, menus and menu Items, attach ActionListeners to each
//menuItem and add the manuItems to the menu
        file = new JMenu("File");
        JMenuItem fileOpen = new JMenuItem("Open");
        fileOpen.addActionListener(new OpenFileListener());
        JMenuItem fileSave = new JMenuItem("Save");
        fileSave.addActionListener(new FileSaveListener());
        JMenuItem fileExport = new JMenuItem("Export");
        fileExport.addActionListener(new TextFileListener());
        file.add(fileOpen);
        file.add(fileSave);
        file.add(fileExport);

        edit = new JMenu("Edit");
        JMenuItem editAdd = new JMenuItem("Add");
        editAdd.addActionListener(new EditListener());
        JMenuItem editRemove = new JMenuItem("Remove");
        editRemove.addActionListener(new EditListener());
        JMenuItem modify = new JMenuItem("Modify");
```

```
202             modify.addActionListener(new EditListener());
203             JMenuItem editClear = new JMenuItem("Clear All");
204             editClear.addActionListener(new EditListener());
205             editClear.setActionCommand("Clear");
206             edit.add(editAdd);
207             edit.add(modify);
208             edit.add(editRemove);
209             edit.add(editClear);
210
211             search = new JMenu("Search");
212             JMenuItem searchAll = new JMenuItem("Search");
213             searchAll.addActionListener(new SearchListener());
214             search.add(searchAll);
215             menuBar = new JMenuBar();
216             menuBar.add(file);
217             menuBar.add(edit);
218             menuBar.add(search);
219     //create a lable for displaying ImageIcons on the JFrame
220             currentImage = new ImageIcon("splash.jpg");
221             imageLabel = new JLabel();
222             imageLabel.setIcon(currentImage);
223
224     //create a panel to dispaly information for the current item selected
225     //set panel layout, add text fields to panel
226             textPanel = new JPanel();
227             textPanel.setLayout(textAreaLayout);
228
229     //creates the JLabels and text fields used to display information about an
230     //Item.
231             description = new JTextField(20);
232             description.setEditable(false);
233             description.setBackground(Color.WHITE);
234             JLabel descriptionLabel = new JLabel("Description:");
235             descriptionLabel.setFont(labelFont);
236
237             make = new JTextField(20);
238             make.setEditable(false);
239             make.setBackground(Color.WHITE);
240             JLabel makeLabel = new JLabel("Manufacturer:");
241             makeLabel.setFont(labelFont);
242
243             model = new JTextField(20);
244             model.setEditable(false);
245             model.setBackground(Color.WHITE);
246             JLabel modelLabel = new JLabel("Model Number:");
247             modelLabel.setFont(labelFont);
248
249             serial = new JTextField(20);
250             serial.setEditable(false);
251             serial.setBackground(Color.WHITE);
252             JLabel serialLabel = new JLabel("Serial Number:");
253             serialLabel.setFont(labelFont);
254
255             year = new JTextField(20);
256             year.setEditable(false);
257             year.setBackground(Color.WHITE);
258             JLabel yearLabel = new JLabel("Year Acquired:");
259             yearLabel.setFont(labelFont);
260
261             price = new JTextField(20);
262             price.setEditable(false);
263             price.setBackground(Color.WHITE);
264             JLabel priceLabel = new JLabel("Original Price:");
265             priceLabel.setFont(labelFont);
266
267             qty = new JTextField(20);
268             qty.setEditable(false);
```

```
269             qty.setBackground(Color.WHITE);
270             JLabel qtyLabel = new JLabel("Available QTY:");
271             qtyLabel.setFont(labelFont);
272
273             totalValue = new JTextField(20);
274             totalValue.setEditable(false);
275             totalValue.setBackground(Color.WHITE);
276             JLabel totalLabel = new JLabel("Total Assets:");
277             totalLabel.setFont(labelFont);
278     //adds lables and text fields to the JPanel.
279             textPanel.add(descriptionLabel);
280             textPanel.add(description);
281             textPanel.add(makeLabel);
282             textPanel.add(make);
283             textPanel.add(modelLabel);
284             textPanel.add(model);
285             textPanel.add(serialLabel);
286             textPanel.add(serial);
287             textPanel.add(yearLabel);
288             textPanel.add(year);
289             textPanel.add(priceLabel);
290             textPanel.add(price);
291             textPanel.add(qtyLabel);
292             textPanel.add(qty);
293             textPanel.add(totalLabel);
294             textPanel.add(totalValue);
295     //add the image panel and text panel to the center panel
296             centerPanel = new JPanel();
297             centerPanel.setLayout(centerLayout);
298             centerPanel.setBorder(trim);
299     //      centerPanel.add(imagePanel);
300             centerPanel.add(imageLabel);
301             centerPanel.add(textPanel);
302
303     //create the button Panel
304             buttonPanel = new JPanel();
305             buttonPanel.setLayout(buttonLayout);
306     //create buttons and actionListeners for each button
307             first = new JButton("|<<First");
308             first.addActionListener(new MovementListener());
309             previous = new JButton("<Previous");
310             previous.addActionListener(new MovementListener());
311             next = new JButton("Next>");
312             next.addActionListener(new MovementListener());
313             last = new JButton("Last>>|");
314             last.addActionListener(new MovementListener());
315     //add the buttons to the button panel
316             buttonPanel.add(first);
317             buttonPanel.add(previous);
318             buttonPanel.add(next);
319             buttonPanel.add(last);
320     //add all components to the JFrame
321             add(buttonPanel, BorderLayout.SOUTH);
322             add(centerPanel, BorderLayout.CENTER);
323             setJMenuBar(menuBar);
324             setVisible(true);
325         }
326         private String setTextFile()
327         {
328     //allows user to specify the name of a text output file
329             fileName = JOptionPane.showInputDialog(null, "Enter the " +
330             "destination file (.txt)", "Text Output", JOptionPane.PLAIN_MESSAGE
331
332             return fileName;
333         }
334         private String setQuery()
335         {
```

```
336   //Collect information from a user to use in searches and deletions.
337
338           String query = JOptionPane.showInputDialog(null, "Enter the value "
339           "to search.\n -Description\n -Make or Model\n -Serial Number",
340           "Search Input", JOptionPane.PLAIN_MESSAGE);
341
342           return query;
343       }
344       private void alert()
345       {
346   //Notifies user when operations are not available due to no data being
347   //present in the PropertyList ArrayList
348           JOptionPane.showMessageDialog(null, "You must open a file before "+
349           "attempting this operation.\n","Illegal Operation",
350           JOptionPane.ERROR_MESSAGE);
351       }
352       private void failure()
353       {
354   //JOptionPane used to alert user of failure in the try/catch blocks.
355           JOptionPane.showMessageDialog(null, "Too many failed attempts, "+
356           "aborting data entry.", "Invalid Input Exception",
357           JOptionPane.ERROR_MESSAGE);
358       }
359       private int confirmDeletion()
360       {
361   //Creates a JOptionPane dialog box to get user confirmation before deletinc
362   //item(s)
363           String message = "To continue select \"OK\"";
364           String title = "Confirm Deleletion";
365
366           int answer = JOptionPane.showConfirmDialog(null, message, title,
367                       JOptionPane.OK_CANCEL_OPTION);
368
369           return answer;
370       }
371       public void setImage(String imageName)
372       {
373   //calls up the name of an Items image file then passes that to the ImageIcc
374   //constructor.  The ImageIcon is passed to the imageLabel for display.
375           currentImage = new ImageIcon(imageName);
376           imageLabel.setIcon(currentImage);
377       }
378       private void createItem()
379       {
380   /**Creates a JFrame for modification of existing Items.  JFrame displays
381    *blank textfields for user data entry.  Information is passed to the
382    *try/catch blocks for analysis and verification before being accepted.*/

383           inputWindow = new JFrame("Add an Item");
384           inputWindow.setSize(SMALL_WIDTH, SMALL_HEIGHT);
385           inputWindow.setResizable(false);
386           inputWindow.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
387
388           GridLayout inputWindowLayout = new GridLayout(9,2, 5, 5);
389           inputWindow.setLayout(inputWindowLayout);
390   //create a label and a blank text field
391           JLabel descriptionLabel = new JLabel("Enter the Item's description"
392           descriptionText = new JTextField(30);
393
394           JLabel makeLabel = new JLabel("Enter the Item's manufacturer");
395           makeText = new JTextField(30);
396
397           JLabel modelLabel = new JLabel("Enter the Item's model");
398           modelText = new JTextField(30);
399
400           JLabel serialLabel = new JLabel("Enter the Item's serial number");
401           serialText = new JTextField(30);
```

```

        JLabel yearLabel = new JLabel("Enter the year Item purchased");
        yearText = new JTextField(30);

        JLabel priceLabel = new JLabel("Enter the Item's purchase price");
        priceText = new JTextField(30);

        JLabel pictureLabel = new JLabel("Enter the picture file's name");
        pictureText = new JTextField(30);

        JLabel qtyLabel = new JLabel("Enter the Item's quantity");
        qtyText = new JTextField(30);
//create a submit and a cancel button.
        JButton addButton = new JButton("Submit");
        addButton.addActionListener(new AddItemListener());

        JButton cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(new AddItemListener());

        inputWindow.add(descriptionLabel);
        inputWindow.add(descriptionText);
        inputWindow.add(makeLabel);
        inputWindow.add(makeText);
        inputWindow.add(modelLabel);
        inputWindow.add(modelText);
        inputWindow.add(serialLabel);
        inputWindow.add(serialText);
        inputWindow.add(yearLabel);
        inputWindow.add(yearText);
        inputWindow.add(priceLabel);
        inputWindow.add(priceText);
        inputWindow.add(qtyLabel);
        inputWindow.add(qtyText);
        inputWindow.add(pictureLabel);
        inputWindow.add(pictureText);
        inputWindow.add(addButton);
        inputWindow.add(cancelButton);

        inputWindow.setLocationRelativeTo(null);
        inputWindow.setVisible(true);
    }
    private void modifyItem()
    {
/**Creates a JFrame for modification of existing Items.  JFrame loads the
 *data of the Item displayed in the main window.  User is then able to modi
 *the existing Item's variables.  Modifications are passed to the try/catch
 *blocks for analysis and verification before being accepted.*/
        inputWindow = new JFrame("Modify Existing Item");
        inputWindow.setSize(SMALL_WIDTH, SMALL_HEIGHT);
        inputWindow.setResizable(false);
        inputWindow.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);

        GridLayout inputWindowLayout = new GridLayout(9 ,2, 5, 5);
        inputWindow.setLayout(inputWindowLayout);
//create a label and a text field, call up the variables of Item displayed

        JLabel descriptionLabel = new JLabel("Update the Item's
description");
        descriptionText = new JTextField(30);
        descriptionText.setText(currentItem.getDescription());

        JLabel makeLabel = new JLabel("Update the Item's manufacturer");
        makeText = new JTextField(30);
        makeText.setText(currentItem.getMake());

        JLabel modelLabel = new JLabel("Update the Item's model");
        modelText = new JTextField(30);
```

```
467             modelText.setText(currentItem.getModel());
468
469             JLabel serialLabel = new JLabel("Update the Item's serial number");
470             serialText = new JTextField(30);
471             serialText.setText(currentItem.getSerial());
472
473             JLabel yearLabel = new JLabel("Enter the year Item purchased");
474             yearText = new JTextField(30);
475             yearText.setText(Integer.toString(currentItem.getDate()));
476
477             JLabel priceLabel = new JLabel("Enter the Item's purchase price");
478             priceText = new JTextField(30);
479             priceText.setText(Double.toString(currentItem.getPrice()));
480
481             JLabel pictureLabel = new JLabel("Enter the picture file's name");
482             pictureText = new JTextField(30);
483             pictureText.setText(currentItem.getPic());
484
485             JLabel qtyLabel = new JLabel("Enter the Item's quantity");
486             qtyText = new JTextField(30);
487             qtyText.setText(Integer.toString(currentItem.getQty()));
488     //create a submit and a cancel button.
489             JButton addButton = new JButton("Submit");
490             addButton.addActionListener(new AddItemListener());
491
492             JButton cancelButton = new JButton("Cancel");
493             cancelButton.addActionListener(new AddItemListener());
494     //add all the textfields and lables to the JFrame
495             inputWindow.add(descriptionLabel);
496             inputWindow.add(descriptionText);
497             inputWindow.add(makeLabel);
498             inputWindow.add(makeText);
499             inputWindow.add(modelLabel);
500             inputWindow.add(modelText);
501             inputWindow.add(serialLabel);
502             inputWindow.add(serialText);
503             inputWindow.add(yearLabel);
504             inputWindow.add(yearText);
505             inputWindow.add(priceLabel);
506             inputWindow.add(priceText);
507             inputWindow.add(qtyLabel);
508             inputWindow.add(qtyText);
509             inputWindow.add(pictureLabel);
510             inputWindow.add(pictureText);
511             inputWindow.add(addButton);
512             inputWindow.add(cancelButton);
513
514             inputWindow.setLocationRelativeTo(null);
515             inputWindow.setVisible(true);
516     //boolean controls whether Item is a modification or an addition to the
517     //PropertyList
518             modifyNotAdd = true;
519         }
520         private void updateJFrame(int location)
521         {
522     //used for updating the information and images displayed on the JFrame.
523             currentItem = localList.getItem(location);
524
525             description.setText(currentItem.getDescription());
526             make.setText(currentItem.getMake());
527             model.setText(currentItem.getModel());
528             serial.setText(currentItem.getSerial());
529             year.setText(Integer.toString(currentItem.getDate()));
530             price.setText(money.format(currentItem.getPrice()));
531             qty.setText(Integer.toString(currentItem.getQty()));
532             setImage(currentItem.getPic());
533         }
```

```
534         private void clearJFrame(String wipe)
535         {
536     //this method is used for clearing JFrame text fields of latent data.
537             description.setText(wipe);
538             make.setText(wipe);
539             model.setText(wipe);
540             serial.setText(wipe);
541             year.setText(wipe);
542             price.setText(wipe);
543             qty.setText(wipe);
544         }
545         private class TextFileListener extends JFrame implements ActionListener
546         {
547             public void actionPerformed(ActionEvent e)
548             {
549                 String userResponse = setTextFile();
550                 System.out.println(userResponse);
551     //if the PropertyList is not null, and the user specifies a name, output
552     //propertyList to a user specified file.
553                 if((localList != null)&&(userResponse != null)&&
554                     (!userResponse.equals("")))
555                     localList.printTextFile(userResponse);
556                 else if (localList == null)
557                     alert();
558             }
559         }//end private inner class TextFileListener
560         private class OpenFileListener extends JFrame implements ActionListener
561         {
562             private File fileName;
563
564             public void actionPerformed(ActionEvent e)
565             {
566                 analyzePath();
567             }//end of Open
568             public File getFileOrDirectory()
569             {//instantiation of JFileChooser for browsing of file structure,
570             //FILES_AND_DIRECTORIES constant allows user to navigate / view
571             //both files and directories.
572                 JFileChooser fileChooser = new JFileChooser();
573                 fileChooser.setFileSelectionMode
574                                             (JFileChooser.FILES_AND_DIRECTORIES
575                 //sets the JFileChooser to be configured for filesaving
576                 int result = fileChooser.showOpenDialog(this);
577
578                 if (result == JFileChooser.CANCEL_OPTION)
579                     dispose();
580                 else
581                 {
582                     fileName = fileChooser.getSelectedFile();
583
584                     if((fileName == null)||(fileName.getName().equals("")))
585                     {
586                         JOptionPane.showMessageDialog(this, "Invalid Name",
587                         "Error", JOptionPane.ERROR_MESSAGE);
588                     }
589                 }
590                 return fileName;
591             }
592             public void analyzePath()
593             {
594                 File name = getFileOrDirectory();
595
596                 if(name != null)
597                 {
598                     if(name.exists())
599                     {
600     /**if the name is valid, and the name exists, load the date into a
```

```
    *ProperyList, set the current location to slot 0 and update the display.
    *calculate the value of the PropertyList.*/
                    localList = new PropertyList();
                    localList.readFromFile(name.getName());
                    updateJFrame(0);
                    location = 0;
                    totalValue.setText(money.format(localList.getTotalValue()))
                    }
                }//end outer if

                else //is not a file directory, or user cancels,
                // generate error message
                {
                    JOptionPane.showMessageDialog(this,"Aborting request",
                    "Open File Aborted", JOptionPane.ERROR_MESSAGE);
                }
            }
        }//end private inner class OpenFileListener
        public class FileSaveListener extends JFrame implements ActionListener
        {
            private File fileName;
            private int result;

            public void actionPerformed(ActionEvent e)
            {
                if(localList != null)
                    analyzePath();
                else
                    alert();
            }
            private File getFileOrDirectory()
            {
            //instantiates an instance of JFileChooser, allowing user to select
            //destination file and folder
                JFileChooser fileChooser = new JFileChooser();
                fileChooser.setFileSelectionMode
                                            (JFileChooser.FILES_AND_DIRECTORIES
                //sets JFile Chooser to a save file format
                result = fileChooser.showSaveDialog(this);

                if (result == JFileChooser.CANCEL_OPTION)//allows cancelation
                    dispose();
                else
                {
                    fileName = fileChooser.getSelectedFile();
                //prevents user from saving files without a name
                    if((fileName == null)||(fileName.getName().equals("")))
                    {
                        JOptionPane.showMessageDialog(this, "Invalid Name",
                        "Error", JOptionPane.ERROR_MESSAGE);
                    }
                }
                return fileName; //returns class File to calling method
            }
            public void analyzePath()
            {
                File name = getFileOrDirectory();
            //if the user chooses to save the file, get its name
                if(result == JFileChooser.APPROVE_OPTION)
                    localList.writeToFile(name.getName());

                else //is not a file directory, or user cancels,
                // generate error message
                {
                    JOptionPane.showMessageDialog(this,"Aborting Request",
                    "Save file aborted", JOptionPane.ERROR_MESSAGE);
                }
```

```
            }
        }//end private inner class FileSaveListener
        private class EditListener implements ActionListener
        {
            public void actionPerformed(ActionEvent e)
            {
                int reply; //used for confirming deletions
                String junk = e.getActionCommand();

                if(localList != null)
                {
                    if (junk.equals("Add"))
                        createItem();

                    else if (junk.equals("Remove"))
                    {
                        String query = setQuery();
                        int temp = -1;

                        if((query != null)&&(!query.equals("")))
                            temp = localList.searchDataBase(query);

                        if (temp>=0)
                        {

                            updateJFrame(temp);
                    //confirmation of deletion
                            reply = confirmDeletion();

                            if (reply == JOptionPane.YES_OPTION)
                            {
                                localList.removeFromDataBase(temp);

                                int newSize = localList.getDataBaseSize();

                                if((newSize>1)&&(temp!=0))
                                    updateJFrame(newSize - 1);
                    //updates the JFrame following the deletion
                                else if ((temp == 0)&&(newSize>1))
                                    updateJFrame(temp + 1);
                                else if(newSize == 1)
                                    updateJFrame(0); //update to remaining at 0
                                else
                                {//assumes 0 is last remaining element
                                    localList.clearDataBase();
                                    clearJFrame(" ");
                                }
                                totalValue.setText

(money.format(localList.getTotalValue()));
                            }
                        }
                        else
                            JOptionPane.showMessageDialog(null, "Item was not "
                            "found", "Search Result", JOptionPane.PLAIN_MESSAGE
                    }
                    else if (junk.equals("Clear"))
                    {
            //requests confirmation before proceeding with deletion
                        reply = confirmDeletion();
            //if user replies yes to the previous JOptionPane
                        if (reply == JOptionPane.YES_OPTION)
                        {
                            localList.clearDataBase();
                            clearJFrame(" ");
                            totalValue.setText(" ");
                        }
```

```
                    }
                    else//Modify event
                    {
                        System.out.println("Modify" + junk);
                        modifyItem();
                    }
                }//end if
                else
                {
                    if (junk.equals("Add"))
                    {
                        localList = new PropertyList();
                        createItem();
                    }
                    else
                        alert();
                }
            }
        }//end private inner class EditListener
        private class SearchListener implements ActionListener
        {
            public void actionPerformed(ActionEvent e)
            {
                if (localList != null)
                {//uses JOptionPane to collect user's search criteria
                    String userInput = setQuery();
                    int temp;

                    if((userInput != null)&&(!userInput.equals("")))
                    {
                        temp = localList.searchDataBase(userInput);

                        if (temp >= 0)
                            updateJFrame(temp);
                        else
                            JOptionPane.showMessageDialog(null,
                            "Item was not found", "Search Result",
                            JOptionPane.PLAIN_MESSAGE);
                    }
                }
                else
                    alert();
            }
        }//end private inner class SearchListener
        private class MovementListener implements ActionListener
        {
            public void actionPerformed(ActionEvent e)
            {
                String junk = e.getActionCommand();
    //If arraylist is empty / null, the user cannot use movement buttons

                if((localList != null)&&(localList.listIsEmpty()!=true))
                {
                    if (junk.equals("|<<First"))
                    {
                        location = 0;
                        updateJFrame(location);
                    }
                    else if (junk.equals("<Previous"))
                    {
                        if (location != 0)
                        {
                            location--;
                            updateJFrame(location);
                        }
                    }
                    else if (junk.equals("Next>"))
```

```
                    {
                        if (location != (localList.getDataBaseSize()-1))
                        {
                            location++;
                            updateJFrame(location);
                        }
                    }
                    else //last element
                    {
                        location = localList.getDataBaseSize()-1;
                        updateJFrame(location);
                    }
                }//end outer if
                else
                    alert();
            }
        }
        private class AddItemListener implements ActionListener
        {
            String itemDescr, itemMake, itemModel, itemSerial, itemPic;
            int itemYear, itemQty;
            double itemPrice;

            public void actionPerformed(ActionEvent e)
            {
                String junk = e.getActionCommand();

                if (junk.equals("Submit"))
                {
    //set the user's input as the textfield's text
                    descriptionText.setText(descriptionText.getText());

                    makeText.setText(makeText.getText());
                    modelText.setText(modelText.getText());
                    serialText.setText(serialText.getText());
                    yearText.setText(yearText.getText());
                    priceText.setText(priceText.getText());
                    qtyText.setText(qtyText.getText());
                    pictureText.setText(pictureText.getText().toLowerCase());

    //adjusting indent for this section because of long line entries
            String temp, secondAttempt, fileExtension;
            int anInt;
            double aDouble;
            boolean acceptableInput = true;
    /*logic for each of the following try-catch pairs is as follows:
    A user's input is collected from the text field and copied to a type
    appropriate variable.  The input is analyzed agains an acceptable range of
    responses. If the input does not satisfy the input criteria, a exception is
    thrown with a specific message identifying the error.  The exception is
    passed to a JOption dialog box which allows the user one attempt to correct
    their mistake.  If the user again fails to satisfy the input requirements,
    the input sesion is terminated and the information is discarded. If the use
    meets all requirement, the information is fed into a constructor and added
    to the PropertyList ArrayList and the current assets value is updated. The
    JFrame will update to display the recent addition.
    */
        try
        {
            temp = descriptionText.getText();
            if((temp == null)||(temp.equals("")))
                throw new InvalidInputException("Description may not be blank")
            else
                itemDescr = temp;
        }
        catch (InvalidInputException itemException)
        {
```

```
            secondAttempt = JOptionPane.showInputDialog(null,
            itemException.getMessage(),"Invalid Input",
            JOptionPane.PLAIN_MESSAGE);

            if((secondAttempt == null)||(secondAttempt.equals("")))
            {
                failure();
                inputWindow.dispose();
                acceptableInput = false;
            }
            else
                itemDescr = secondAttempt;
        }
        try
        {
            temp = makeText.getText();
            if((temp == null)||(temp.equals("")))
                throw new InvalidInputException("Make may not be "+
                "blank. If none or unknown, state \"none\"");
            else
                itemMake = temp;
        }
        catch(InvalidInputException itemException)
        {
            secondAttempt = JOptionPane.showInputDialog(null,
            itemException.getMessage(),"Invalid Input",
            JOptionPane.PLAIN_MESSAGE);

            if((secondAttempt == null)||(secondAttempt.equals("")))
            {
                failure();
                inputWindow.dispose();
                acceptableInput = false;
            }
            else
                itemMake = secondAttempt;
        }
        try
        {
            temp = modelText.getText();
            if((temp == null)||(temp.equals("")))
                throw new InvalidInputException("Model may not be "+
                " blank. If none or unknown, state \"none\"");
            else
                itemModel = temp;
        }
        catch(InvalidInputException itemException)
        {
            secondAttempt = JOptionPane.showInputDialog(null,
            itemException.getMessage(),"Invalid Input",
            JOptionPane.PLAIN_MESSAGE);

            if((secondAttempt == null)||(secondAttempt.equals("")))
            {
                failure();
                inputWindow.dispose();
                acceptableInput = false;
            }
            else
                itemModel = secondAttempt;
        }
        try
        {
            temp = serialText.getText();
            if((temp == null)||(temp.equals("")))
                throw new InvalidInputException("Serial number may not "+
                "be blank. If none or unknown, state \"none\"");
```

```
            else
                itemSerial = temp;
        }
        catch(InvalidInputException itemException)
        {
            secondAttempt = JOptionPane.showInputDialog(null,
            itemException.getMessage(),"Invalid Input",
            JOptionPane.PLAIN_MESSAGE);

            if((secondAttempt== null)||(secondAttempt.equals(" ")))
            {
                failure();
                inputWindow.dispose();
                acceptableInput = false;
            }
            else
                itemSerial = secondAttempt;
        }
        try
        {
            if ((yearText.getText() == null)||(yearText.getText().equals(""))
                throw new InvalidInputException("Purchase field may not be"+
                " blank.");
            else
                anInt = Integer.parseInt(yearText.getText());

            if((anInt <1800)||(anInt>2012))
                throw new InvalidInputException("Date of purchase is"+
                " outside the acceptable range.");
            else
                itemYear = anInt;
        }
        catch(InvalidInputException itemException)
        {
            secondAttempt = JOptionPane.showInputDialog(null,
            itemException.getMessage(),"Invalid Input",
            JOptionPane.PLAIN_MESSAGE);

            if((secondAttempt == null)||(secondAttempt.equals(""))||
            (Integer.parseInt(secondAttempt)<1800)||
            (Integer.parseInt(secondAttempt)<2012))
            {
                failure();
                inputWindow.dispose();
                acceptableInput = false;
            }
            else
                itemYear = Integer.parseInt(secondAttempt);
        }
        try
        {
            if((priceText.getText() == null)||(priceText.getText().equals("")
                throw new InvalidInputException("Price may not be blank.");
            else
                aDouble = Double.parseDouble(priceText.getText());

            if(aDouble<0)
                throw new InvalidInputException("Purchase price "+
                "may not be less than zero.");
            else
                itemPrice = aDouble;
        }
        catch(InvalidInputException itemException)
        {
            secondAttempt = JOptionPane.showInputDialog(null,
            itemException.getMessage(),"Invalid Input",
            JOptionPane.PLAIN_MESSAGE);
```

```
1000
1001            if(Double.parseDouble(secondAttempt)<0)
1002            {
1003                failure();
1004                inputWindow.dispose();
1005                acceptableInput = false;
1006            }
1007            else
1008                itemPrice = Double.parseDouble(secondAttempt);
1009        }
1010        try
1011        {
1012            if((qtyText.getText() == null)||(qtyText.getText().equals("")))
1013                throw new InvalidInputException("Quantity may not be blank.")
1014            else
1015                anInt = Integer.parseInt(qtyText.getText());
1016
1017            if(anInt<0)
1018                throw new InvalidInputException("Quantity may not be negative
1019            else
1020                itemQty = anInt;
1021        }//last try
1022        catch(InvalidInputException itemException)
1023        {
1024            secondAttempt = JOptionPane.showInputDialog(null,
1025            itemException.getMessage(),"Invalid Input",
1026            JOptionPane.PLAIN_MESSAGE);
1027
1028            if((secondAttempt == null)||(secondAttempt.equals(""))||
1029            (Integer.parseInt(secondAttempt)<0))
1030            {
1031                failure();
1032                inputWindow.dispose();
1033                acceptableInput = false;
1034            }
1035            else
1036                itemQty = Integer.parseInt(secondAttempt);
1037        }
1038        try
1039        {
1040            temp = pictureText.getText();
1041
1042            if((temp!=null)&&(!temp.equals("")))
1043            {
1044                if(temp.equalsIgnoreCase("none"))
1045                    temp = temp.toLowerCase()+".jpg";
1046
1047                fileExtension = temp.substring(temp.length()-3, temp.length()
1048
1049                if((fileExtension.equals("jpg"))||(fileExtension.equals("gif"
1050                ||(fileExtension.equals("png")))
1051                    itemPic = temp;
1052
1053                else
1054                    throw new InvalidInputException("File must be in format"+
1055                    " .jpg, .gif, or .png");
1056            }
1057            else
1058                throw new InvalidInputException("Picture file cannot be "+
1059                "blank. If no picture, state \"none\".");
1060        }
1061        catch(InvalidInputException itemException)
1062        {
1063            secondAttempt = JOptionPane.showInputDialog(null,
1064            itemException.getMessage(),"Invalid Input",
1065            JOptionPane.PLAIN_MESSAGE);
1066
```

```
1067            if((secondAttempt!=null)&&(!secondAttempt.equals("")))
1068            {
1069                if (secondAttempt.equalsIgnoreCase("none"))
1070                    secondAttempt = secondAttempt.toLowerCase()+".jpg";
1071
1072                fileExtension = secondAttempt.substring(secondAttempt.length(
1073                                                        secondAttempt.length
1074
1075                if((fileExtension.equals("jpg"))||(fileExtension.equals("gif"
1076                ||(fileExtension.equals("png")))
1077                    itemPic = secondAttempt;
1078
1079                else
1080                {
1081                    failure();
1082                    inputWindow.dispose();
1083                    acceptableInput = false;
1084                }
1085            }
1086            else
1087            {
1088                failure();
1089                inputWindow.dispose();
1090                acceptableInput = false;
1091            }
1092        }//last catch
1093
1094        if((acceptableInput)&&(modifyNotAdd == false))
1095        {
1096            Item newItem = new Item(itemDescr, itemModel, itemMake, itemSeria
1097            itemQty,itemYear, itemPrice, itemPic);
1098
1099            localList.addToDataBase(newItem);
1100            totalValue.setText(money.format(localList.getTotalValue()));
1101            updateJFrame(localList.getDataBaseSize()-1);
1102        }
1103        if((acceptableInput)&&(modifyNotAdd))
1104        {
1105            currentItem.updateAll(itemDescr, itemModel, itemMake, itemSerial,
1106            itemQty,itemYear, itemPrice, itemPic);
1107            modifyNotAdd = false;
1108            totalValue.setText(money.format(localList.getTotalValue()));
1109            updateJFrame(location);
1110        }
1111
1112        inputWindow.dispose();
1113
1114        }//end if statement
1115            else
1116                inputWindow.dispose();
1117        }//end method
1118
1119        }//end private inner additem class
1120    }//end ItemGui
```

```java
/* CS-112 FINAL PROJECT
   File Name:           PropertyList.java
   Programmer:          James Watkins
   Date Last Modified:  May 19, 2012

   Problem Statement: Define a class which uses ArrayList objects to
   manage multiple objects of class Item.  This class must be serializable
   and should also support text I/O operations.  This class will interface
   with a gui.

   Overall Plan:
   1. Provide constructors for creating ArrayLists of type Item.
   2. Include a means of copying one propertyList to another (included for
   good measure, not required for objectives).
   3. Define class specific methods which invoke ArrayList's methods add(),
   remove(), isEmpty(), clear(), get(), and size().
   4. Create a method to calculate the total value of all assets using an
   enhanced for loop and invocations of Item's getPrice() and getQty().
   5. Provide a search method which takes in a String query to search and
   then returns the location of items matching the query.
   6. Define a method which steps through all the elments of the ArrayList
   and invokes each Item's toString().
   7. Design a method to write the contents of the ArrayList to a text file.
   **invocation of Item's toString() did not yield aesthetically pleasing
   results.  Had to manually control the printing format.  Use a
   FileOutputStream with PrintWriter to write the text to a file.  Allow the
   user to specify the destination file name as a String parameter.
   8. Specify a method for writing PropertyList ArrayLists to a file using
   the Serializable interface.  Create instances of ObjectOuputStream and
   FileOutputStream to facilitate this requirement.  Allow the user to
   specify the file's name by collecting a String parameter.  Echo progress
   and any exceptions to the command line for analysis and user awareness.
   9. Allow the user to load files written using the Serializable interface
   by creating a readFromFile() method.  User will provide the source file's
   name as a String.  Method will invoke instances of ObjectInputStream and
   FileInputStream to recover data and copy it to a PropertyList ArrayList.
   Handle any Exceptions internally, echo progress and any exceptions to the
   command line for situational awareness.

   Classes needed and Purpose (Input, Processing, Output)
   Item - input, output, processing
   ArrayList - input, output, processing
   NumberFormat - output            PrintWriter - output
   FileInputStream - input          FileNotFoundException - error handling
   FileOuputStream - output         IOException - error handling
   ObjectInputStream - input        ClassNotFoundException - error handling
   ObjectOuputStream - ouput
*/
import java.util.ArrayList;
import java.io.*;
import java.util.*;
import java.text.NumberFormat;
import java.io.Serializable;

public class PropertyList implements Serializable
{
    Scanner keyboard = new Scanner(System.in);
    ArrayList <Item> dataBase;
    PrintWriter textOutput;
    private ObjectOutputStream output;
    private ObjectInputStream inputStream = null;

    NumberFormat money = NumberFormat.getCurrencyInstance();

    public PropertyList()
    {
        dataBase = new ArrayList <Item>();
```

```
        }
        public PropertyList(int size)
        {
            dataBase = new ArrayList <Item>(size);
        }
        public void copyDataBase()
        {
            ArrayList<Item> newArrayList = new ArrayList<Item>(getDataBaseSize(

            for (Item element : dataBase)
            {
                newArrayList.add(element);
            }
        }
        //provide access to required ArrayList methods.
        public void addToDataBase(Item anItem)
        {
            dataBase.add(anItem);
        }
        public void removeFromDataBase(int theLocation)
        {
                dataBase.remove(theLocation);
        }
        public boolean listIsEmpty()
        {
            return dataBase.isEmpty();
        }
        public void clearDataBase()
        {
            dataBase.clear();
        }
        public int getDataBaseSize()
        {
            return dataBase.size();
        }
        public Item getItem(int location)
        {
            return dataBase.get(location);
        }
        public double getTotalValue()
        {
            double totalValue = 0;
        //method to calculate the current value of all assets
            for (Item element : dataBase)
            {
                int aQuantity = element.getQty();
                if(aQuantity == 1)
                    totalValue += element.getPrice();
                else
                    totalValue += (element.getPrice() * aQuantity);

            }
            return totalValue;
        }
        public int searchDataBase(String query)
        {
        //search by description, serialnumber, make and model
            int itemLocation = -1, i = 0;

            for(Item element : dataBase)
            {
                if (query.equalsIgnoreCase(element.getDescription())||
                    query.equalsIgnoreCase(element.getSerial())||
                    query.equalsIgnoreCase(element.getModel())||
                    query.equalsIgnoreCase(element.getMake()))
                {
                    itemLocation = i;
```

```
                    break;
                }
                i++;
            }
            //if query not found, a negative returned value can be filtered out
            return itemLocation;
        }
        public void listAllRecords()
        {
            dataBase.trimToSize();
            for(int i = 0; i < dataBase.size(); i++)
            {
                System.out.println(dataBase.get(i).toString());
            }
        }
        public void printTextFile(String fileName)
        {
            int item = 1;

            try
            {
                textOutput = new PrintWriter(new FileOutputStream(fileName));

                for(Item element : dataBase)
                {
  //Item's toString didn't preserve formatting when passed through PrintWrite
                    textOutput.println("Item Number "+item);
                    textOutput.println("Description\t"+element.getDescription()
                    textOutput.println("Manufacturer\t"+element.getMake());
                    textOutput.println("Model Number\t"+element.getModel());
                    textOutput.println("Serial Number\t"+element.getSerial());
                    textOutput.println("Purchase Date\t"+element.getDate());
                    textOutput.println("Purchase Price\t"+
                                            money.format(element.getPrice()));
                    textOutput.println("Quantity Avail.\t"+element.getQty());
                    textOutput.println("Picture File\t" + element.getPic());
                    textOutput.println("");
                    item++;
                }
                textOutput.println("Total Inventory:\t"+
                                        money.format(getTotalValue()))
                textOutput.close();
                System.out.println("File " + fileName + " created.");
            }
            catch(IOException e)
            {
                System.out.println("Couldn't write to file " + fileName);
            }

        }
        public void writeToFile(String fileName)
        {
            //removes any unused elements prior to writing to file.
            dataBase.trimToSize();
            try
            {
                ObjectOutputStream outputStream = new ObjectOutputStream
                                        (new FileOutputStream(fileName));
            //write file only if dataBase isn't empty
                if (!dataBase.isEmpty())
                {
                    outputStream.writeObject((ArrayList<Item>)dataBase);
                    outputStream.close();
                    System.out.println("Attempting to write.");
                }
                System.out.println("WriteToFile complete.");
            }
```

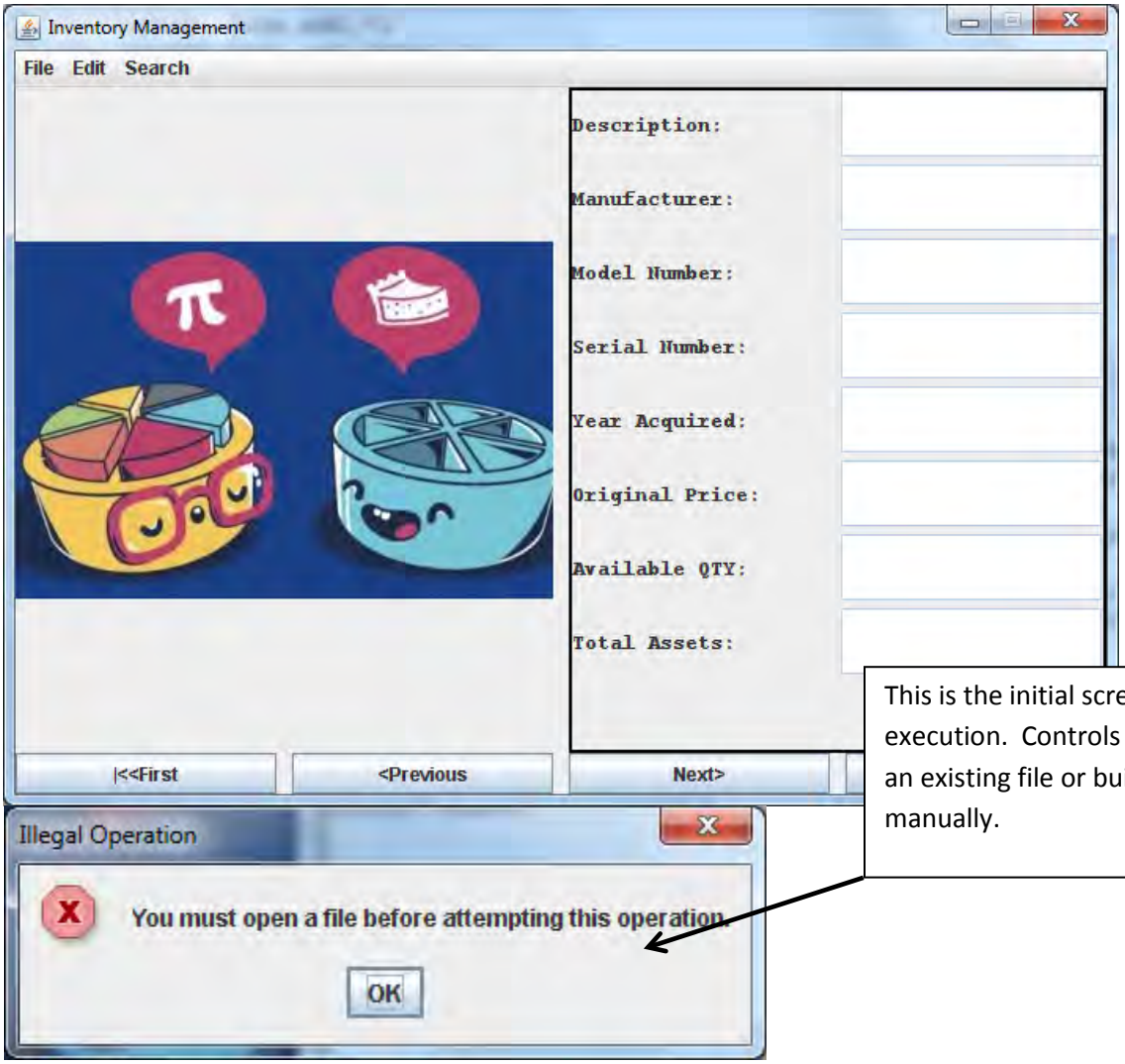
```
            catch(IOException e)
            {
                System.err.println("(IO)Error creating binary file " + fileName
            }
        }
        public void readFromFile(String fileName)
        {
            try
            {
                ObjectInputStream inputStream =
                        new ObjectInputStream(new FileInputStream(fileName));

                dataBase    = (ArrayList<Item>)inputStream.readObject();

                inputStream.close();
                System.out.println("Read from file complete");
            }
            catch(FileNotFoundException e)
            {
                System.out.println("Can not find binary file " + fileName);
            }
            catch(ClassNotFoundException e)
            {
                System.out.println("Can not find class specified.");
            }
            catch(IOException e)
            {
                System.out.println("(IO)Can not find binary file " + fileName);
            }
        }
    }
```

```java
/* CS-112 FINAL PROJECT
   File Name:           Item.java
   Programmer:          James Watkins
   Date Last Modified:  May 19, 2012

   Problem Statement: Define a class to contain the specific details of items
   which will be used within an inventory management application.  Class must
   account for image files as well as text and numeric variables.  Enable
   binary I/O.

   Overall Plan:
   1. Define the constructors which will be used to create Item objects.
   2. Provide mutator and accessor methods for each of an Item's variables.
   3. Override equals.
   4. Override toString.
   5. Make the class serializable to support binary I/O operations.

   Classes needed and Purpose (Input, Processing, Output)
   NumberFormat - output
   String - input, output
*/
import java.text.NumberFormat;
import java.io.Serializable;
import java.io.*;

public class Item implements Serializable
{
    private String description, model, maker, serialNumber, pictureFileName;
    private int quantity, yearPurchased;
    private double price;

    NumberFormat money = NumberFormat.getCurrencyInstance();

    public Item()
    {
        description = null;
        model = null;
        maker = null;
        serialNumber = null;
        quantity = 0;
        yearPurchased = 1000;
        price = 0.0;
        pictureFileName = null;
    }
    public Item(String descr, String aModel, String make, String SN, int QTY,
                int datePurchase, double aPrice)
    {
        description = descr;
        model = aModel;
        maker = make;
        serialNumber = SN;
        quantity = QTY;
        yearPurchased = datePurchase;
        price = aPrice;
        pictureFileName = null;
    }
    public Item(String descr, String aModel, String make, String SN, int QTY,
                int datePurchase, double aPrice, String picFile)
    {
        description = descr;
        model = aModel;
        maker = make;
        serialNumber = SN;
        quantity = QTY;
        yearPurchased = datePurchase;
        price = aPrice;
        pictureFileName = picFile;
```

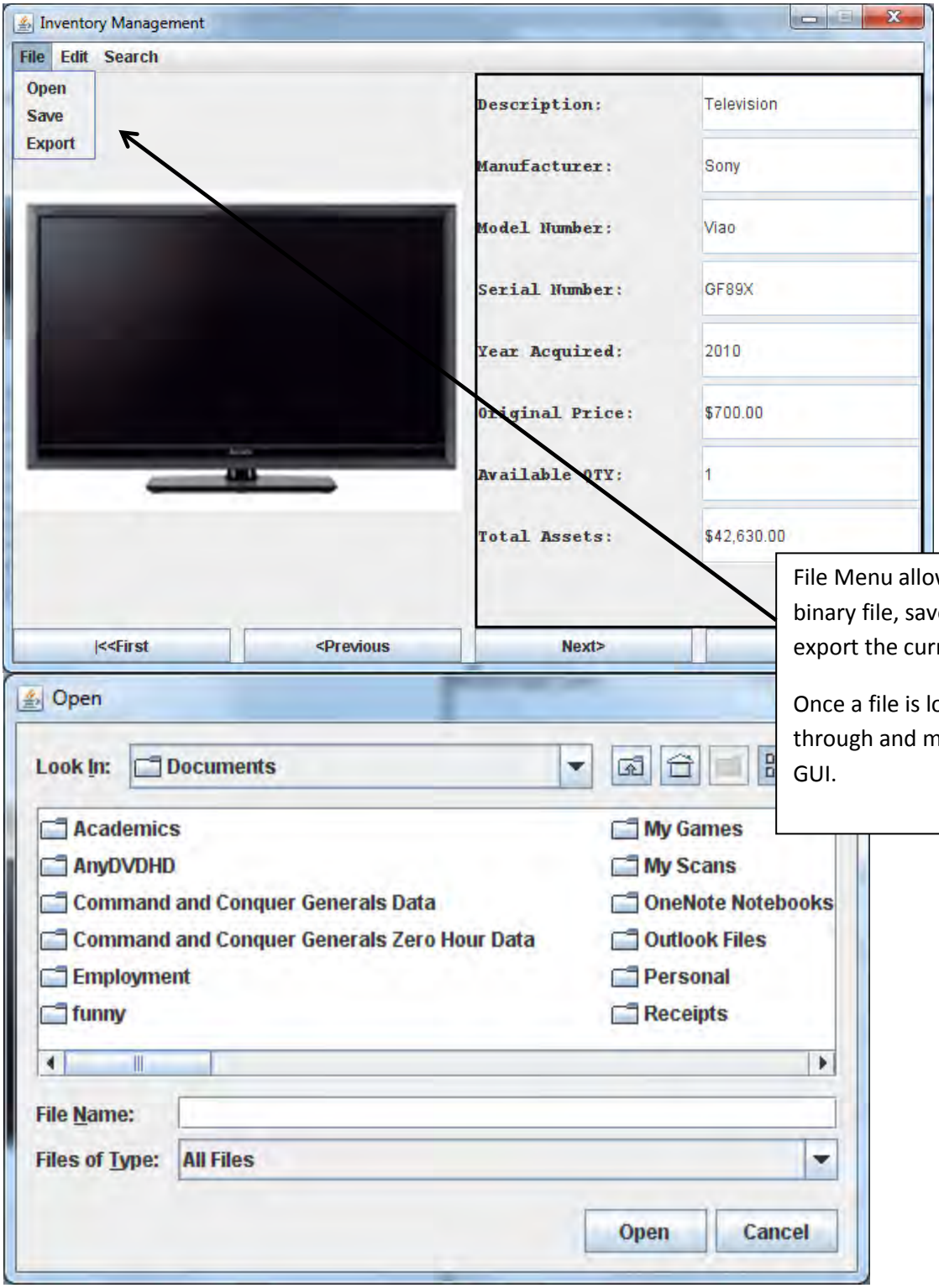
```
        }
        public void updateAll(String descr, String aModel, String make,
        String SN, int QTY, int datePurchase, double aPrice, String picFile)
        {
            description = descr;
            model = aModel;
            maker = make;
            serialNumber = SN;
            quantity = QTY;
            yearPurchased = datePurchase;
            price = aPrice;
            pictureFileName = picFile;
        }
        public void setPic(String fileName)
        {
            pictureFileName = fileName;
        }
        public String getPic()
        {
            return pictureFileName;
        }
        public void setDescription(String descr)
        {
            description = descr;
        }
        public String getDescription()
        {
            return description;
        }
        public void setModel(String input)
        {
            model = input;
        }
        public String getModel()
        {
            return model;
        }
        public void setMake(String input)
        {
            maker = input;
        }
        public String getMake()
        {
            return maker;
        }
        public void setSerial(String SN)
        {
            serialNumber = SN;
        }
        public String getSerial()
        {
            return serialNumber;
        }
        public void setQty(int QTY)
        {
            quantity = QTY;
        }
        public int getQty()
        {
            return quantity;
        }
        public void setDate(int datePurchase)
        {
            yearPurchased = datePurchase;
        }
        public int getDate()
        {
```

```java
            return yearPurchased;
        }
        public void setPrice(double aPrice)
        {
            price = aPrice;
        }
        public double getPrice()
        {
            return price;
        }
        public boolean equals(Item other)
        {
            boolean flag;

            if (this == other)
                flag = true;
    //assumes that purchase price, and purchase date do not make an item unique
            else if ((description.equalsIgnoreCase(other.description))&&
            (serialNumber.equalsIgnoreCase(other.serialNumber))&&
            (model.equalsIgnoreCase(other.model))&&
            (maker.equalsIgnoreCase(other.maker)))
                flag = true;
            else
            //items are not identical
                flag = false;

            return flag;
        }
        public String toString()
        {
            return ("Description:\t"+description+"\nManufacturer:\t"+maker+
            "\nModel Number:\t"+model+"\nSerial Number:\t"+serialNumber+
            "\nYear Purchased:\t"+yearPurchased+"\nQuantity:\t"+quantity+
            "\t\tPurchase Price:\t"+ money.format(price) +
            "\nPicture file:\t" + pictureFileName + "\n");
        }
    }
```

```java
/* CS-112 FINAL PROJECT
   File Name:           InvalidInputException
   Programmer:          James Watkins
   Date Last Modified:  May 19, 2012

   Problem Statement: Define custom Exception class to filter unacceptable
   input to class Item.

   Overall Plan:
   1. Define a constructors for class.

   Classes needed and Purpose (Input, Processing, Output)
   Exception - parent class
*/
public class InvalidInputException extends Exception
    {
        public InvalidInputException()
        {
            super("Invalid input detected.");
        }
        public InvalidInputException(String message)
        {
            super(message);
        }
    }
```

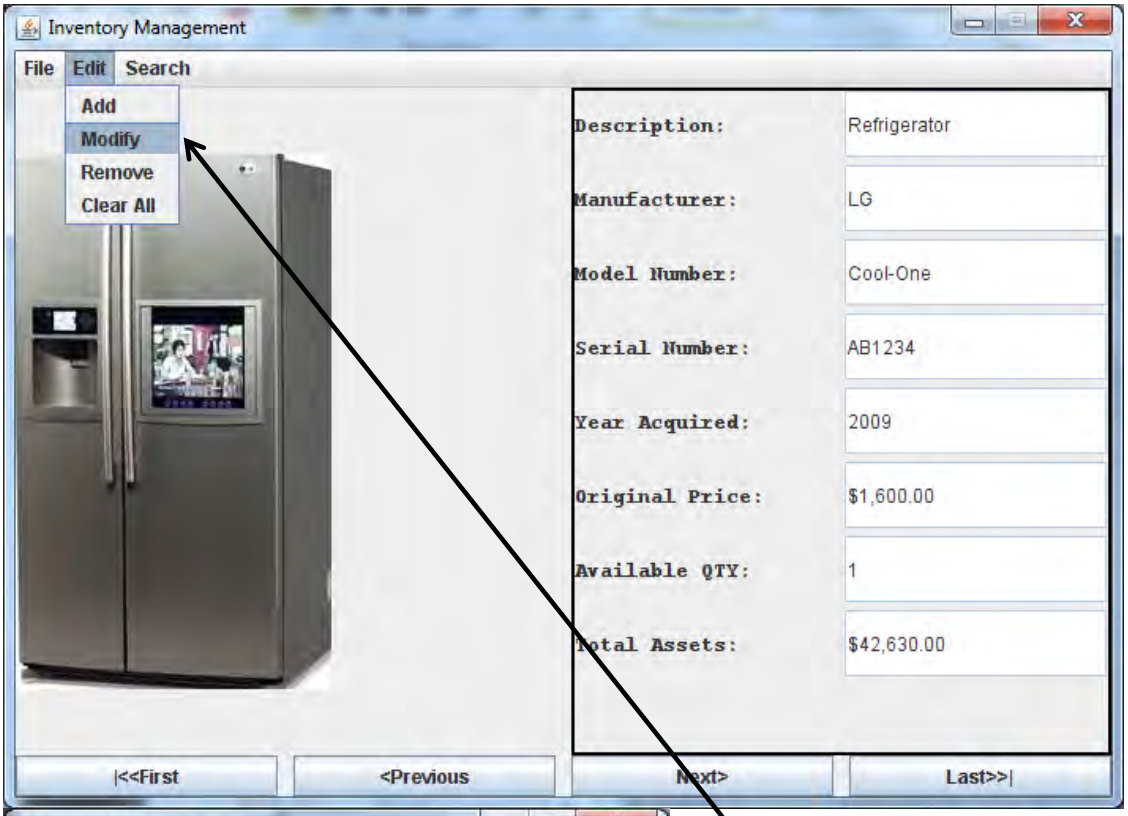


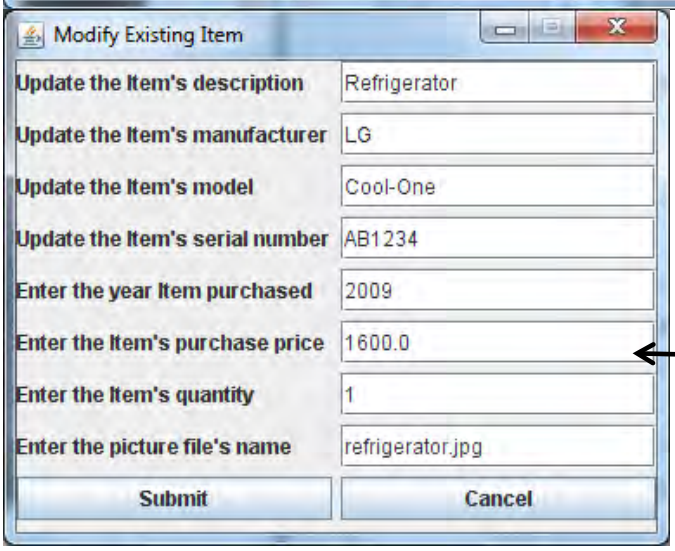
This is the initial screen that loads on execution. Controls are limited to opening an existing file or building a new file manually.



File Menu allows a user to load an existing binary file, save date to a binary file, or export the current data to a text file.

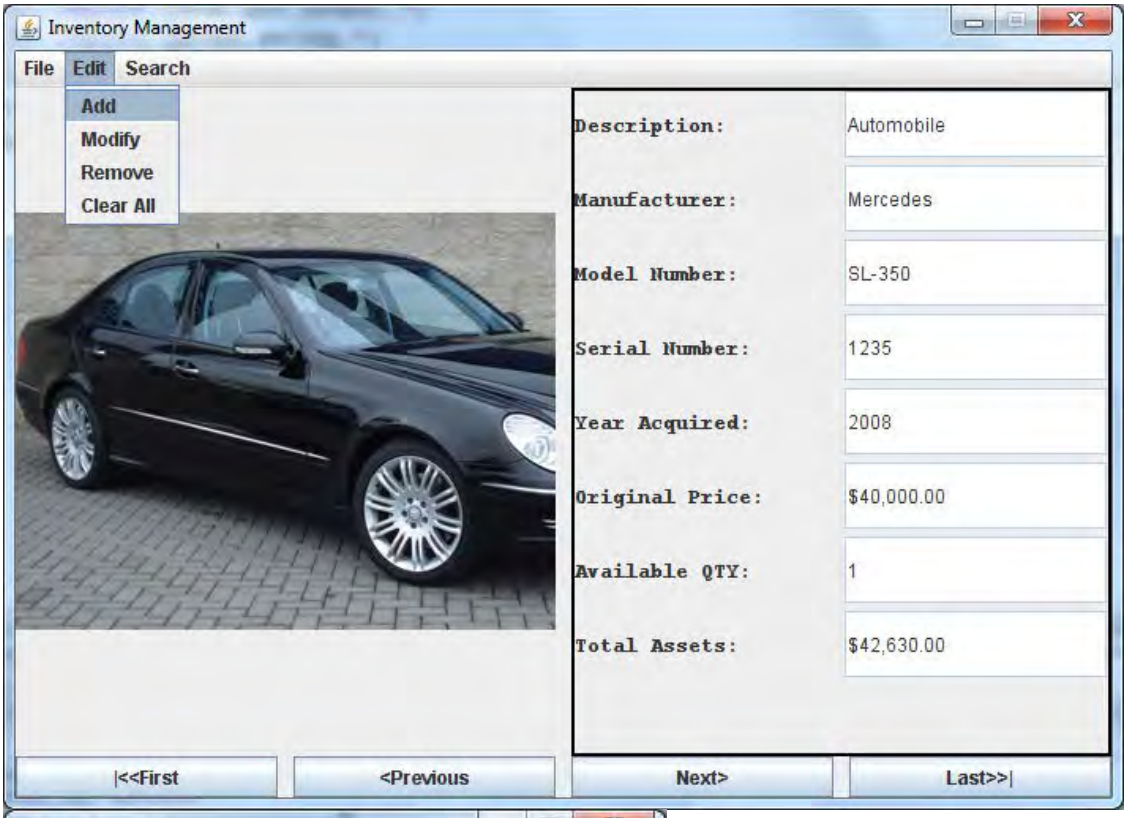
Once a file is loaded a user can navigate through and manipulate entries using the GUI.

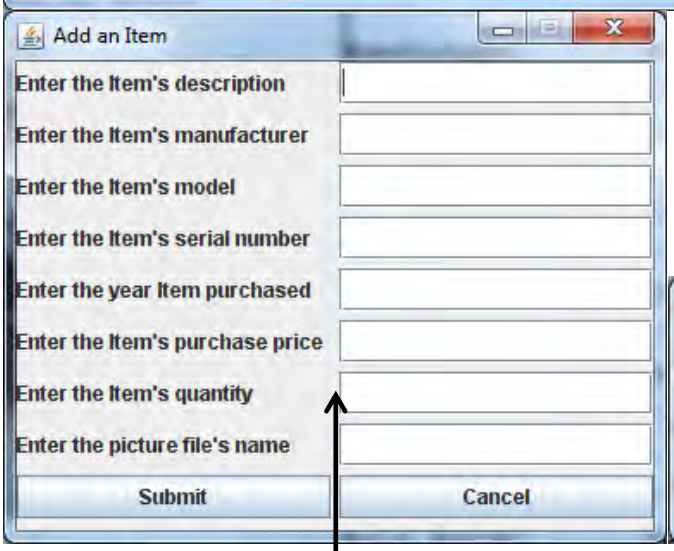


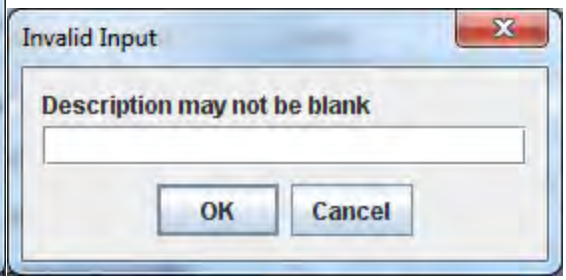


Edit Menu allows addition of new objects, modification of existing objects, single item deletions and erasure of entire data set.

Modify loads the data for the item currently displayed.



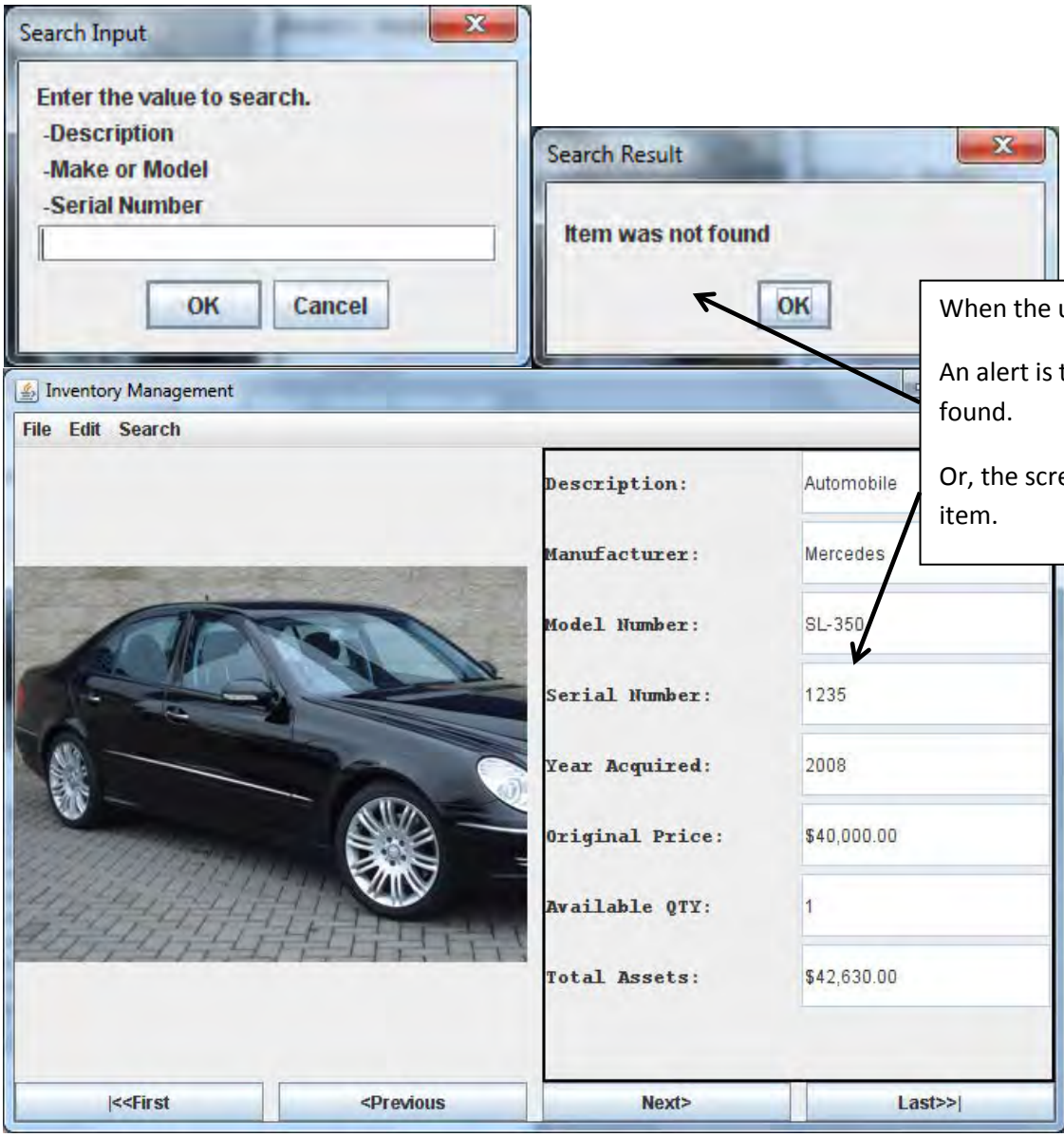




To add a new item the user populates these fields with data.

The information is then passed to a constructor, and a new Item is placed at the end of the ArrayList.

If the user violates any of the conditions for an Items' data; an exception is thrown.



When the user searches the database:

An alert is triggered if the item was not found.

Or, the screen is updated to display the item.