```
package homeinventory;
 import javax.swing.*;
 import javax.swing.filechooser.*;
 import java.awt.*;
 import java.awt.event.*;
 import java.beans.*;
 import com.toedter.calendar.*;
 import java.awt.geom.*;
 import java.io.*;
 import java.util.*;
 import java.text.*;
 import java.awt.print.*;
 public class HomeInventory extends JFrame
 {
 // Toolbar
 JToolBar inventoryToolBar = new JToolBar();
 JButton newButton = new JButton(new ImageIcon("new.gif"));
 JButton deleteButton = new JButton(new ImageIcon("delete.gif"));
 JButton saveButton = new JButton(new ImageIcon("save.gif"));
 JButton previousButton = new JButton(new ImageIcon("previous.gif"));
 JButton nextButton = new JButton(new ImageIcon("next.gif"));
 JButton printButton = new JButton(new ImageIcon("print.gif"));
 JButton exitButton = new JButton();

// Frame
 JLabel itemLabel = new JLabel();
 JTextField itemTextField = new JTextField();
 JLabel locationLabel = new JLabel();
 JComboBox locationComboBox = new JComboBox();
 JCheckBox markedCheckBox = new JCheckBox();
 JLabel serialLabel = new JLabel();
```

```
 JTextField serialTextField = new JTextField();
 JLabel priceLabel = new JLabel();
 JTextField priceTextField = new JTextField();
 JLabel dateLabel = new JLabel();
 JDateChooser dateDateChooser = new JDateChooser();
 JLabel storeLabel = new JLabel();
 JTextField storeTextField = new JTextField();
 JLabel noteLabel = new JLabel();
 JTextField noteTextField = new JTextField();
 JLabel photoLabel = new JLabel();
 static JTextArea photoTextArea = new JTextArea();
 JButton photoButton = new JButton();
 JPanel searchPanel = new JPanel();
 JButton[] searchButton = new JButton[26];
 PhotoPanel photoPanel = new PhotoPanel();
 static final int maximumEntries = 300;
 static int numberEntries;
 static InventoryItem[] myInventory = new InventoryItem[maximumEntries];
 int currentEntry;
 static final int entriesPerPage = 2;
 static int lastPage;
 public static void main(String args[])
{
 // create frame
 new HomeInventory().show();
 }
 public HomeInventory()
{
 // frame constructor
setTitle("Home Inventory Manager");
 setResizable(false);
```

```
 setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
 addWindowListener(new WindowAdapter()
 {
 public void windowClosing(WindowEvent evt)
 {
exitForm(evt);
 }
 });
 getContentPane().setLayout(new GridBagLayout());
 GridBagConstraints gridConstraints;
 inventoryToolBar.setFloatable(false);
 inventoryToolBar.setBackground(Color.BLUE);
inventoryToolBar.setOrientation(SwingConstants.VERTICAL);
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 0;
 gridConstraints.gridy = 0;
 gridConstraints.gridheight = 8;
 gridConstraints.fill = GridBagConstraints.VERTICAL;
 getContentPane().add(inventoryToolBar, gridConstraints);
 inventoryToolBar.addSeparator();
 Dimension bSize = new Dimension(70, 50);
 newButton.setText("New");
 sizeButton(newButton, bSize);
 newButton.setToolTipText("Add New Item");
 newButton.setHorizontalTextPosition(SwingConstants.CENTER);
newButton.setVerticalTextPosition(SwingConstants.BOTTOM);
 newButton.setFocusable(false);
 inventoryToolBar.add(newButton);
newButton.addActionListener(new ActionListener()
 {
 public void actionPerformed(ActionEvent e)
```

```
{
newButtonActionPerformed(e);
}
});
deleteButton.setText("Delete");
sizeButton(deleteButton, bSize);
deleteButton.setToolTipText("Delete Current Item");
deleteButton.setHorizontalTextPosition(SwingConstants.CENTER);
deleteButton.setVerticalTextPosition(SwingConstants.BOTTOM);
deleteButton.setFocusable(false);
inventoryToolBar.add(deleteButton);
deleteButton.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{
deleteButtonActionPerformed(e);
}
});
saveButton.setText("Save");
sizeButton(saveButton, bSize);
saveButton.setToolTipText("Save Current Item");
saveButton.setHorizontalTextPosition(SwingConstants.CENTER);
saveButton.setVerticalTextPosition(SwingConstants.BOTTOM);
saveButton.setFocusable(false);
inventoryToolBar.add(saveButton);
saveButton.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{
saveButtonActionPerformed(e);
}
});
```

```
 inventoryToolBar.addSeparator();

 previousButton.setText("Previous");

 sizeButton(previousButton, bSize);

 previousButton.setToolTipText("Display Previous Item");
previousButton.setHorizontalTextPosition(SwingConstants.CENTER);
previousButton.setVerticalTextPosition(SwingConstants.BOTTOM);
previousButton.setFocusable(false);

 inventoryToolBar.add(previousButton);

 previousButton.addActionListener(new ActionListener()

 {

 public void actionPerformed(ActionEvent e)

 {

 previousButtonActionPerformed(e);

}

 });

 nextButton.setText("Next");

sizeButton(nextButton, bSize);

 nextButton.setToolTipText("Display Next Item");
nextButton.setHorizontalTextPosition(SwingConstants.CENTER);
nextButton.setVerticalTextPosition(SwingConstants.BOTTOM);

 nextButton.setFocusable(false);

 inventoryToolBar.add(nextButton);

 nextButton.addActionListener(new ActionListener()

 {

 public void actionPerformed(ActionEvent e)

 {

 nextButtonActionPerformed(e);

 }

});

 inventoryToolBar.addSeparator();

printButton.setText("Print");

 sizeButton(printButton, bSize);
```

```
 printButton.setToolTipText("Print Inventory List");
printButton.setHorizontalTextPosition(SwingConstants.CENTER);
printButton.setVerticalTextPosition(SwingConstants.BOTTOM);
 printButton.setFocusable(false);
 inventoryToolBar.add(printButton);
 printButton.addActionListener(new ActionListener()
 {
 public void actionPerformed(ActionEvent e)
{
printButtonActionPerformed(e);
 }
 });
 exitButton.setText("Exit");
 sizeButton(exitButton, bSize);
 exitButton.setToolTipText("Exit Program");
exitButton.setFocusable(false);
 inventoryToolBar.add(exitButton);
 exitButton.addActionListener(new ActionListener()
 {
 public void actionPerformed(ActionEvent e)
 {
 exitButtonActionPerformed(e);
 }
 });
 itemLabel.setText("Inventory Item");
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 1;
 gridConstraints.gridy = 0;
gridConstraints.insets = new Insets(10, 10, 0, 10);
 gridConstraints.anchor = GridBagConstraints.EAST;
 getContentPane().add(itemLabel, gridConstraints);
 itemTextField.setPreferredSize(new Dimension(400, 25));
```

```
gridConstraints = new GridBagConstraints();
gridConstraints.gridx = 2;
gridConstraints.gridy = 0;
gridConstraints.gridwidth = 5;
gridConstraints.insets = new Insets(10, 0, 0, 10);
gridConstraints.anchor = GridBagConstraints.WEST;
getContentPane().add(itemTextField, gridConstraints);
itemTextField.addActionListener(new ActionListener ()
{
public void actionPerformed(ActionEvent e)
{
itemTextFieldActionPerformed(e);
}
});
locationLabel.setText("Location");
gridConstraints = new GridBagConstraints();
gridConstraints.gridx = 1;
gridConstraints.gridy = 1;
gridConstraints.insets = new Insets(10, 10, 0, 10);
gridConstraints.anchor = GridBagConstraints.EAST;
getContentPane().add(locationLabel, gridConstraints);
locationComboBox.setPreferredSize(new Dimension(270, 25));
locationComboBox.setFont(new Font("Arial", Font.PLAIN, 12));
locationComboBox.setEditable(true);
locationComboBox.setBackground(Color.WHITE);
gridConstraints = new GridBagConstraints();
gridConstraints.gridx = 2;
gridConstraints.gridy = 1;
gridConstraints.gridwidth = 3;
gridConstraints.insets = new Insets(10, 0, 0, 10);
```

```
gridConstraints.anchor = GridBagConstraints.WEST;
 getContentPane().add(locationComboBox, gridConstraints);
locationComboBox.addActionListener(new ActionListener ()
 {
 public void actionPerformed(ActionEvent e)
 {
 locationComboBoxActionPerformed(e);
 }
});
 markedCheckBox.setText("Marked?");
 markedCheckBox.setFocusable(false);
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 5;
 gridConstraints.gridy = 1;
 gridConstraints.insets = new Insets(10, 10, 0, 0);
 gridConstraints.anchor = GridBagConstraints.WEST;
 getContentPane().add(markedCheckBox, gridConstraints);
 serialLabel.setText("Serial Number");
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 1;
 gridConstraints.gridy = 2;
 gridConstraints.insets = new Insets(10, 10, 0, 10);
 gridConstraints.anchor = GridBagConstraints.EAST;
 getContentPane().add(serialLabel, gridConstraints);
 serialTextField.setPreferredSize(new Dimension(270, 25));
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 2;
 gridConstraints.gridy = 2;
 gridConstraints.gridwidth = 3;
 gridConstraints.insets = new Insets(10, 0, 0, 10);
```

```
 gridConstraints.anchor = GridBagConstraints.WEST;
 getContentPane().add(serialTextField, gridConstraints);
 serialTextField.addActionListener(new ActionListener ()
 {
public void actionPerformed(ActionEvent e)
 {
 serialTextFieldActionPerformed(e);
}
 });
 priceLabel.setText("Purchase Price");
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 1;
 gridConstraints.gridy = 3;
 gridConstraints.insets = new Insets(10, 10, 0, 10);
 gridConstraints.anchor = GridBagConstraints.EAST;
 getContentPane().add(priceLabel, gridConstraints);
 priceTextField.setPreferredSize(new Dimension(160, 25));
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 2;
 gridConstraints.gridy = 3;
 gridConstraints.gridwidth = 2;
 gridConstraints.insets = new Insets(10, 0, 0, 10);
 gridConstraints.anchor = GridBagConstraints.WEST;
 getContentPane().add(priceTextField, gridConstraints);
 priceTextField.addActionListener(new ActionListener ()
 {
 public void actionPerformed(ActionEvent e)
{
 priceTextFieldActionPerformed(e);
 }
 });
```

```
 dateLabel.setText("Date Purchased");
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 4;
 gridConstraints.gridy = 3;
 gridConstraints.insets = new Insets(10, 10, 0, 0);
 gridConstraints.anchor = GridBagConstraints.WEST;
 getContentPane().add(dateLabel, gridConstraints);

dateDateChooser.setPreferredSize(new Dimension(120, 25));
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 5;
 gridConstraints.gridy = 3;
 gridConstraints.gridwidth = 2;
 gridConstraints.insets = new Insets(10, 0, 0, 10);
 gridConstraints.anchor = GridBagConstraints.WEST;
 getContentPane().add(dateDateChooser, gridConstraints);
dateDateChooser.addPropertyChangeListener(new PropertyChangeListener()
 {
 public void propertyChange(PropertyChangeEvent e)
{
 dateDateChooserPropertyChange(e);
 }
 });
 storeLabel.setText("Store/Website");
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 1;
 gridConstraints.gridy = 4;
 gridConstraints.insets = new Insets(10, 10, 0, 10);
gridConstraints.anchor = GridBagConstraints.EAST;
 getContentPane().add(storeLabel, gridConstraints);
 storeTextField.setPreferredSize(new Dimension(400, 25));
```

```
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 2;
 gridConstraints.gridy = 4;
 gridConstraints.gridwidth = 5;
 gridConstraints.insets = new Insets(10, 0, 0, 10);
 gridConstraints.anchor = GridBagConstraints.WEST;
 getContentPane().add(storeTextField, gridConstraints);
 storeTextField.addActionListener(new ActionListener ()
 {
 public void actionPerformed(ActionEvent e)
{
storeTextFieldActionPerformed(e);
 }
 });
 noteLabel.setText("Note");
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 1;
 gridConstraints.gridy = 5;
 gridConstraints.insets = new Insets(10, 10, 0, 10);
 gridConstraints.anchor = GridBagConstraints.EAST;
 getContentPane().add(noteLabel, gridConstraints);
 noteTextField.setPreferredSize(new Dimension(400, 25));
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 2;
 gridConstraints.gridy = 5;
 gridConstraints.gridwidth = 5;
 gridConstraints.insets = new Insets(10, 0, 0, 10);
 gridConstraints.anchor = GridBagConstraints.WEST;
 getContentPane().add(noteTextField, gridConstraints);
 noteTextField.addActionListener(new ActionListener ()
 {
```

```
 public void actionPerformed(ActionEvent e)
{
 noteTextFieldActionPerformed(e);
 }
 });
 photoLabel.setText("Photo");
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 1;
 gridConstraints.gridy = 6;
 gridConstraints.insets = new Insets(10, 10, 0, 10);
gridConstraints.anchor = GridBagConstraints.EAST;
 getContentPane().add(photoLabel, gridConstraints);
 photoTextArea.setPreferredSize(new Dimension(350, 35));



photoTextArea.setFont(new Font("Arial", Font.PLAIN, 12));
 photoTextArea.setEditable(false);
photoTextArea.setLineWrap(true);
 photoTextArea.setWrapStyleWord(true);
 photoTextArea.setBackground(new Color(255, 255, 192));
photoTextArea.setBorder(BorderFactory.createLineBorder(Color.BLACK));
photoTextArea.setFocusable(false);
 gridConstraints = new GridBagConstraints();
gridConstraints.gridx = 2;
 gridConstraints.gridy = 6;
 gridConstraints.gridwidth = 4;
 gridConstraints.insets = new Insets(10, 0, 0, 10);
 gridConstraints.anchor = GridBagConstraints.WEST;
 getContentPane().add(photoTextArea, gridConstraints);
 photoButton.setText("...");
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 6;
```

```
 gridConstraints.gridy = 6;
 gridConstraints.insets = new Insets(10, 0, 0, 10);
 gridConstraints.anchor = GridBagConstraints.WEST;
 getContentPane().add(photoButton, gridConstraints);
 photoButton.addActionListener(new ActionListener ()
 {
 public void actionPerformed(ActionEvent e)
{
 photoButtonActionPerformed(e);
}
 });
 searchPanel.setPreferredSize(new Dimension(240, 160));
searchPanel.setBorder(BorderFactory.createTitledBorder("Item Search"));
searchPanel.setLayout(new GridBagLayout());
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 1;
gridConstraints.gridy = 7;
 gridConstraints.gridwidth = 3;
 gridConstraints.insets = new Insets(10, 0, 10, 0);
 gridConstraints.anchor = GridBagConstraints.CENTER;
 getContentPane().add(searchPanel, gridConstraints);
 int x = 0, y = 0;
 // create and position 26 buttons
 for (int i = 0; i < 26; i++)
 {
 // create new button
 searchButton[i] = new JButton();
 // set text property
 searchButton[i].setText(String.valueOf((char) (65 + i)));
searchButton[i].setFont(new Font("Arial", Font.BOLD, 12));
 searchButton[i].setMargin(new Insets(-10, -10, -10, -10));
```

```
 sizeButton(searchButton[i], new Dimension(37, 27));
searchButton[i].setBackground(Color.YELLOW);
 searchButton[i].setFocusable(false);
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = x;
 gridConstraints.gridy = y;
 searchPanel.add(searchButton[i], gridConstraints);
 // add method
 searchButton[i].addActionListener(new ActionListener ()
{
 public void actionPerformed(ActionEvent e)
 {
 searchButtonActionPerformed(e);
 }
 });
 x++;
 // six buttons per row
 if (x % 6 == 0)
 {
 x = 0;
y++;
 }
 }
 photoPanel.setPreferredSize(new Dimension(240, 160));
 gridConstraints = new GridBagConstraints();
 gridConstraints.gridx = 4;
 gridConstraints.gridy = 7;
 gridConstraints.gridwidth = 3;
 gridConstraints.insets = new Insets(10, 0, 10, 10);
 gridConstraints.anchor = GridBagConstraints.CENTER;
 getContentPane().add(photoPanel, gridConstraints);
```

```
pack();
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
 setBounds((int) (0.5 * (screenSize.width - getWidth())), (int) (0.5 * (screenSize.height - getHeight())),
getWidth(), getHeight());
 int n;
// open file for entries
 try { BufferedReader inputFile = new BufferedReader(new FileReader("inventory.txt"));
numberEntries = Integer.valueOf(inputFile.readLine()).intValue();
 if (numberEntries != 0)
 {
 for (int i = 0; i < numberEntries; i++)
 {
 myInventory[i] = new InventoryItem();
myInventory[i].description = inputFile.readLine();
 myInventory[i].location = inputFile.readLine();
 myInventory[i].serialNumber = inputFile.readLine();
 myInventory[i].marked = Boolean.valueOf(inputFile.readLine()).booleanValue();
myInventory[i].purchasePrice =inputFile.readLine();
 myInventory[i].purchaseDate = inputFile.readLine();
myInventory[i].purchaseLocation = inputFile.readLine();
 myInventory[i].note = inputFile.readLine();
 myInventory[i].photoFile = inputFile.readLine();
 }
 }
 // read in combo box elements
 n = Integer.valueOf(inputFile.readLine()).intValue();
 if (n != 0)
{
 for (int i = 0; i < n; i++)
{
 locationComboBox.addItem(inputFile.readLine());
 }
```

```
 }
 inputFile.close();
 currentEntry = 1;
 showEntry(currentEntry);
 } catch (Exception ex) {
numberEntries = 0;
 currentEntry = 0;
 }
 if (numberEntries == 0)
 {
 newButton.setEnabled(false);
 deleteButton.setEnabled(false);
nextButton.setEnabled(false);
 previousButton.setEnabled(false);
 printButton.setEnabled(false);
}
 }



private void exitForm(WindowEvent evt)
 {
 if (JOptionPane.showConfirmDialog(null, "Any unsaved changes will be lost.\nAre you sure you want to exit?", "Exit Program", JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE) == JOptionPane.NO_OPTION) return;
// write entries back to file
 try { PrintWriter outputFile = new PrintWriter(new BufferedWriter(new FileWriter("inventory.txt")));
outputFile.println(numberEntries);
 if (numberEntries != 0)
 {
 for (int i = 0; i < numberEntries; i++)


{
```

```
outputFile.println(myInventory[i].description);
 outputFile.println(myInventory[i].location);
outputFile.println(myInventory[i].serialNumber);
 outputFile.println(myInventory[i].marked);
 outputFile.println(myInventory[i].purchasePrice);
outputFile.println(myInventory[i].purchaseDate);
outputFile.println(myInventory[i].purchaseLocation);
 outputFile.println(myInventory[i].note);
outputFile.println(myInventory[i].photoFile);
 }
 }
 // write combo box entries
 outputFile.println(locationComboBox.getItemCount());
 if (locationComboBox.getItemCount() != 0)
 {
 for (int i = 0; i < locationComboBox.getItemCount(); i++)
outputFile.println(locationComboBox.getItemAt(i));
}
 outputFile.close();
 }



catch (Exception ex) { } System.exit(0);
}
 private void newButtonActionPerformed(ActionEvent e)
 {
 checkSave();
 blankValues();
 }
private void deleteButtonActionPerformed(ActionEvent e)
{
```

```
 if (JOptionPane.showConfirmDialog(null, "Are you sure you want to delete this item?", "Delete
Inventory Item", JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE) ==
JOptionPane.NO_OPTION) return;
 deleteEntry(currentEntry);
 if (numberEntries == 0) { currentEntry = 0;
 blankValues();
}
Else
 {
 currentEntry--;
 if (currentEntry == 0) currentEntry = 1;
showEntry(currentEntry);
 }
 }
 private void saveButtonActionPerformed(ActionEvent e)
{
 // check for description


itemTextField.setText(itemTextField.getText().trim());
 if (itemTextField.getText().equals(""))
{
 JOptionPane.showConfirmDialog(null, "Must have item description.", "Error",
JOptionPane.DEFAULT_OPTION, JOptionPane.ERROR_MESSAGE);
 itemTextField.requestFocus();
 return;
 }
 if (newButton.isEnabled())
{
 // delete edit entry then resave
 deleteEntry(currentEntry);
 }
```

```
 // capitalize first letter
 String s = itemTextField.getText();
 itemTextField.setText(s.substring(0, 1).toUpperCase() + s.substring(1));
 numberEntries++;
 // determine new current entry location based on description
 currentEntry = 1;
 if (numberEntries != 1)
 {
 Do
 {
 if (itemTextField.getText().compareTo(myInventory[currentEntry - 1].description) < 0) break;
currentEntry++;
 }
 while (currentEntry < numberEntries);
 }
 // move all entries below new value down one position unless at end
 if (currentEntry != numberEntries)
{
 for (int i = numberEntries; i >= currentEntry + 1; i--)
 {
 myInventory[i - 1] = myInventory[i - 2];
myInventory[i - 2] = new InventoryItem();
 }
 }
 myInventory[currentEntry - 1] = new InventoryItem();
 myInventory[currentEntry - 1].description = itemTextField.getText();
 myInventory[currentEntry - 1].location = locationComboBox.getSelectedItem().toString();
myInventory[currentEntry - 1].marked = markedCheckBox.isSelected();
 myInventory[currentEntry - 1].serialNumber = serialTextField.getText();
myInventory[currentEntry - 1].purchasePrice = priceTextField.getText();
```

```
 myInventory[currentEntry - 1].purchaseDate = dateToString(dateDateChooser.getDate());
myInventory[currentEntry - 1].purchaseLocation = storeTextField.getText();
myInventory[currentEntry - 1].photoFile = photoTextArea.getText();
 myInventory[currentEntry - 1].note = noteTextField.getText();
showEntry(currentEntry);
 if (numberEntries < maximumEntries) newButton.setEnabled(true);
 else
 newButton.setEnabled(false);
 deleteButton.setEnabled(true);
 printButton.setEnabled(true);
 }
 private void previousButtonActionPerformed(ActionEvent e)
 {
checkSave();
currentEntry--;
 showEntry(currentEntry);
 }
 private void nextButtonActionPerformed(ActionEvent e)
{
 checkSave();
 currentEntry++;
 showEntry(currentEntry);



}
private void printButtonActionPerformed(ActionEvent e)
 {
 lastPage = (int) (1 + (numberEntries - 1) / entriesPerPage);
 PrinterJob inventoryPrinterJob = PrinterJob.getPrinterJob();
 inventoryPrinterJob.setPrintable(new InventoryDocument());
 if (inventoryPrinterJob.printDialog())
 {
```

```
 try { inventoryPrinterJob.print();
 }
 catch (PrinterException ex)
 {
 JOptionPane.showConfirmDialog(null, ex.getMessage(), "Print Error",
JOptionPane.DEFAULT_OPTION, JOptionPane.ERROR_MESSAGE);
 }
 }
 }
 private void exitButtonActionPerformed(ActionEvent e)
 {
 exitForm(null);
 }
 private void photoButtonActionPerformed(ActionEvent e)
 {
 JFileChooser openChooser = new JFileChooser();
openChooser.setDialogType(JFileChooser.OPEN_DIALOG);
 openChooser.setDialogTitle("Open Photo File");
 openChooser.addChoosableFileFilter(new FileNameExtensionFilter("Photo Files", "jpg"));
 if (openChooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
showPhoto(openChooser.getSelectedFile().toString()); }


private void searchButtonActionPerformed(ActionEvent e)
 {
 int i;
 if (numberEntries == 0) return;
 // search for item letter String
 letterClicked = e.getActionCommand();
 i = 0;
 do { if (myInventory[i].description.substring(0, 1).equals(letterClicked))
{
```

```
currentEntry = i + 1;
 showEntry(currentEntry);
 return;
 }
 i++;
 }
 while (i < numberEntries);
 JOptionPane.showConfirmDialog(null, "No " + letterClicked + " inventory items.", "None Found",
JOptionPane.DEFAULT_OPTION, JOptionPane.INFORMATION_MESSAGE);
}
 private void itemTextFieldActionPerformed(ActionEvent e)
 {
 locationComboBox.requestFocus();
}
private void locationComboBoxActionPerformed(ActionEvent e)
 {
 // If in list - exit method
 if (locationComboBox.getItemCount() != 0)
 {
 for (int i = 0; i < locationComboBox.getItemCount(); i++)
{
 if
(locationComboBox.getSelectedItem().toString().equals(locationComboBox.getItemAt(i).toString()))
 {
 serialTextField.requestFocus();
return;
 }
 }
 }
 // If not found, add to list box
 locationComboBox.addItem(locationComboBox.getSelectedItem());
 serialTextField.requestFocus();
```

```
}
 private void serialTextFieldActionPerformed(ActionEvent e)
{
 priceTextField.requestFocus();
 }
 private void priceTextFieldActionPerformed(ActionEvent e)
 {
 dateDateChooser.requestFocus();
}
 private void dateDateChooserPropertyChange(PropertyChangeEvent e)
 {
 storeTextField.requestFocus();
}
 private void storeTextFieldActionPerformed(ActionEvent e)
 {
 noteTextField.requestFocus();
}
 private void noteTextFieldActionPerformed(ActionEvent e)
 {
 photoButton.requestFocus();
 }
private void sizeButton(JButton b, Dimension d)
{
 b.setPreferredSize(d);
 b.setMinimumSize(d);
 b.setMaximumSize(d);
 }
private void showEntry(int j)
 {
 // display entry j (1 to numberEntries) itemTextField.setText(myInventory[j - 1].description);
locationComboBox.setSelectedItem(myInventory[j - 1].location);
markedCheckBox.setSelected(myInventory[j - 1].marked);
```

```
 serialTextField.setText(myInventory[j - 1].serialNumber);
 priceTextField.setText(myInventory[j - 1].purchasePrice);
dateDateChooser.setDate(stringToDate(myInventory[j - 1].purchaseDate));
storeTextField.setText(myInventory[j - 1].purchaseLocation);
noteTextField.setText(myInventory[j - 1].note);
 showPhoto(myInventory[j - 1].photoFile);
nextButton.setEnabled(true);
 previousButton.setEnabled(true);
 if (j == 1) previousButton.setEnabled(false);
 if (j == numberEntries) nextButton.setEnabled(false);
 itemTextField.requestFocus();
 }
private Date stringToDate(String s)
 {
 int m = Integer.valueOf(s.substring(0, 2)).intValue() - 1;
int d = Integer.valueOf(s.substring(3, 5)).intValue();
int y = Integer.valueOf(s.substring(6)).intValue() - 1900;
 return(new Date(y, m, d));
 }
 private String dateToString(Date dd)
{
 String yString = String.valueOf(dd.getYear() + 1900);
 int m = dd.getMonth() + 1;
 String mString = new DecimalFormat("00").format(m);
 int d = dd.getDate();
 String dString = new DecimalFormat("00").format(d);
 return(mString + "/" + dString + "/" + yString);
 }
 private void showPhoto(String photoFile)
 {
 if (!photoFile.equals(""))
 {
```

```
 try { photoTextArea.setText(photoFile);
 }
 catch (Exception ex)
 {
 photoTextArea.setText("");
 }
 }
 Else
 {
 photoTextArea.setText("");
}
 photoPanel.repaint();
}
 private void blankValues()
 {
 // blank input screen
 newButton.setEnabled(false);
 deleteButton.setEnabled(false);
 saveButton.setEnabled(true);
 previousButton.setEnabled(false);

nextButton.setEnabled(false);
 printButton.setEnabled(false);
itemTextField.setText("");
 locationComboBox.setSelectedItem("");
markedCheckBox.setSelected(false);
 serialTextField.setText("");
 priceTextField.setText("");
 dateDateChooser.setDate(new Date());
storeTextField.setText("");
 noteTextField.setText("");
```

```
 photoTextArea.setText("");
photoPanel.repaint();
 itemTextField.requestFocus();
 }
 private void deleteEntry(int j)
{
// delete entry j
if (j != numberEntries)
 {
 // move all entries under j up one level
 for (int i = j; i < numberEntries; i++)
 {
 myInventory[i - 1] = new InventoryItem();
myInventory[i - 1] = myInventory[i];
 }
 }
 numberEntries--;
}
 private void checkSave()
{
 boolean edited = false;
if (!myInventory[currentEntry - 1].description.equals(itemTextField.getText())) edited = true;
 else if (!myInventory[currentEntry –1].location.equals(locationComboBox.getSelectedItem().toStrin
g())) edited = true;
else if (myInventory[currentEntry - 1].marked != markedCheckBox.isSelected()) edited = true;
else if (!myInventory[currentEntry - 1].serialNumber.equals(serialTextField.getText())) edited = true;
else if (!myInventory[currentEntry - 1].purchasePrice.equals(priceTextField.getText())) edited = true;
else if (!myInventory[currentEntry -
1].purchaseDate.equals(dateToString(dateDateChooser.getDate()))) edited = true;
else if (!myInventory[currentEntry - 1].purchaseLocation.equals(storeTextField.getText())) edited =
true;
 else if (!myInventory[currentEntry - 1].note.equals(noteTextField.getText())) edited = true;
 else if (!myInventory[currentEntry - 1].photoFile.equals(photoTextArea.getText())) edited = true;
```

```
if (edited)
{
 if (JOptionPane.showConfirmDialog(null, "You have edited this item. Do you want to save the changes?", "Save Item", JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE) == JOptionPane.YES_OPTION) saveButton.doClick();
 }
 }
 }
 class PhotoPanel extends JPanel { public void paintComponent(Graphics g)
 {
 Graphics2D g2D = (Graphics2D) g;
 super.paintComponent(g2D);
 // draw border
 g2D.setPaint(Color.BLACK);
g2D.draw(new Rectangle2D.Double(0, 0, getWidth() - 1, getHeight() - 1));
 // show photo Image
 photoImage = new ImageIcon(HomeInventory.photoTextArea.getText()).getImage();
 int w = getWidth();
 int h = getHeight();
 double rWidth = (double) getWidth() / (double) photoImage.getWidth(null);
 double rHeight = (double) getHeight() / (double) photoImage.getHeight(null);
if (rWidth > rHeight)
 {
 // leave height at display height, change width by amount height is changed
w = (int) (photoImage.getWidth(null) * rHeight);
 }
 else
{
 // leave width at display width, change height by amount width is changed
 h = (int) (photoImage.getHeight(null) * rWidth);
}
 // center in panel
```

```
 g2D.drawImage(photoImage, (int) (0.5 * (getWidth() - w)), (int) (0.5 * (getHeight() - h)), w, h, null);
g2D.dispose();

 }

 }

 class InventoryDocument implements Printable

 {

 public int print(Graphics g, PageFormat pf, int pageIndex)

 {

 Graphics2D g2D = (Graphics2D) g;

 if ((pageIndex + 1) > HomeInventory.lastPage)

 {

 return NO_SUCH_PAGE; } int i, iEnd;

 // here you decide what goes on each page and draw it



// header

g2D.setFont(new Font("Arial", Font.BOLD, 14));

 g2D.drawString("Home Inventory Items - Page " + String.valueOf(pageIndex + 1), (int)
pf.getImageableX(), (int) (pf.getImageableY() + 25));

 // get starting y

int dy = (int) g2D.getFont().getStringBounds("S", g2D.getFontRenderContext()).getHeight();

 int y = (int) (pf.getImageableY() + 4 * dy);

 iEnd = HomeInventory.entriesPerPage * (pageIndex + 1);

if (iEnd > HomeInventory.numberEntries) iEnd = HomeInventory.numberEntries;

 for (i = 0 + HomeInventory.entriesPerPage * pageIndex; i < iEnd; i++)

 {

 // dividing line

 Line2D.Double dividingLine = new Line2D.Double(pf.getImageableX(), y, pf.getImageableX() +
pf.getImageableWidth(), y);

 g2D.draw(dividingLine); y += dy;

 g2D.setFont(new Font("Arial", Font.BOLD, 12));
g2D.drawString(HomeInventory.myInventory[i].description, (int) pf.getImageableX(), y);
```

```
 y += dy;

g2D.setFont(new Font("Arial", Font.PLAIN, 12));

 g2D.drawString("Location: " + HomeInventory.myInventory[i].location, (int) (pf.getImageableX() +
25), y); y += dy;

 if (HomeInventory.myInventory[i].marked) g2D.drawString("Item is marked with identifying
information.", (int) (pf.getImageableX() + 25), y);

 else

 g2D.drawString("Item is NOT marked with identifying information.", (int) (pf.getImageableX() + 25),
y);

 y += dy;

 g2D.drawString("Serial Number: " + HomeInventory.myInventory[i].serialNumber, (int)
(pf.getImageableX() + 25), y);

y += dy; g2D.drawString("Price: $" + HomeInventory.myInventory[i].purchasePrice + ",

Purchased on: " + HomeInventory.myInventory[i].purchaseDate, (int) (pf.getImageableX() + 25), y);

y += dy;

 g2D.drawString("Purchased at: " + HomeInventory.myInventory[i].purchaseLocation, (int)
(pf.getImageableX() + 25), y);

 y += dy;

 g2D.drawString("Note: " + HomeInventory.myInventory[i].note, (int) (pf.getImageableX() + 25), y);

y += dy;

 try

 {

// maintain original width/height ratio Image

 inventoryImage = new ImageIcon(HomeInventory.myInventory[i].photoFile).getImage();

 double ratio = (double) (inventoryImage.getWidth(null)) / (double) inventoryImage.getHeight(null);
g2D.drawImage(inventoryImage, (int) (pf.getImageableX() + 25), y, (int) (100 * ratio), 100, null);

 }

 catch (Exception ex)

 {

 // have place to go in case image file doesn't open

 }

 y += 2 * dy + 100;

 }
```

```
return PAGE_EXISTS;
}
}
InventoryItem.java: package homeinventory;
public class InventoryItem
{
public String description;
public String location;
public boolean marked;


public String serialNumber;
public String purchasePrice;
public String purchaseDate;
public String purchaseLocation;
public String note;
public String photoFile;
}
```