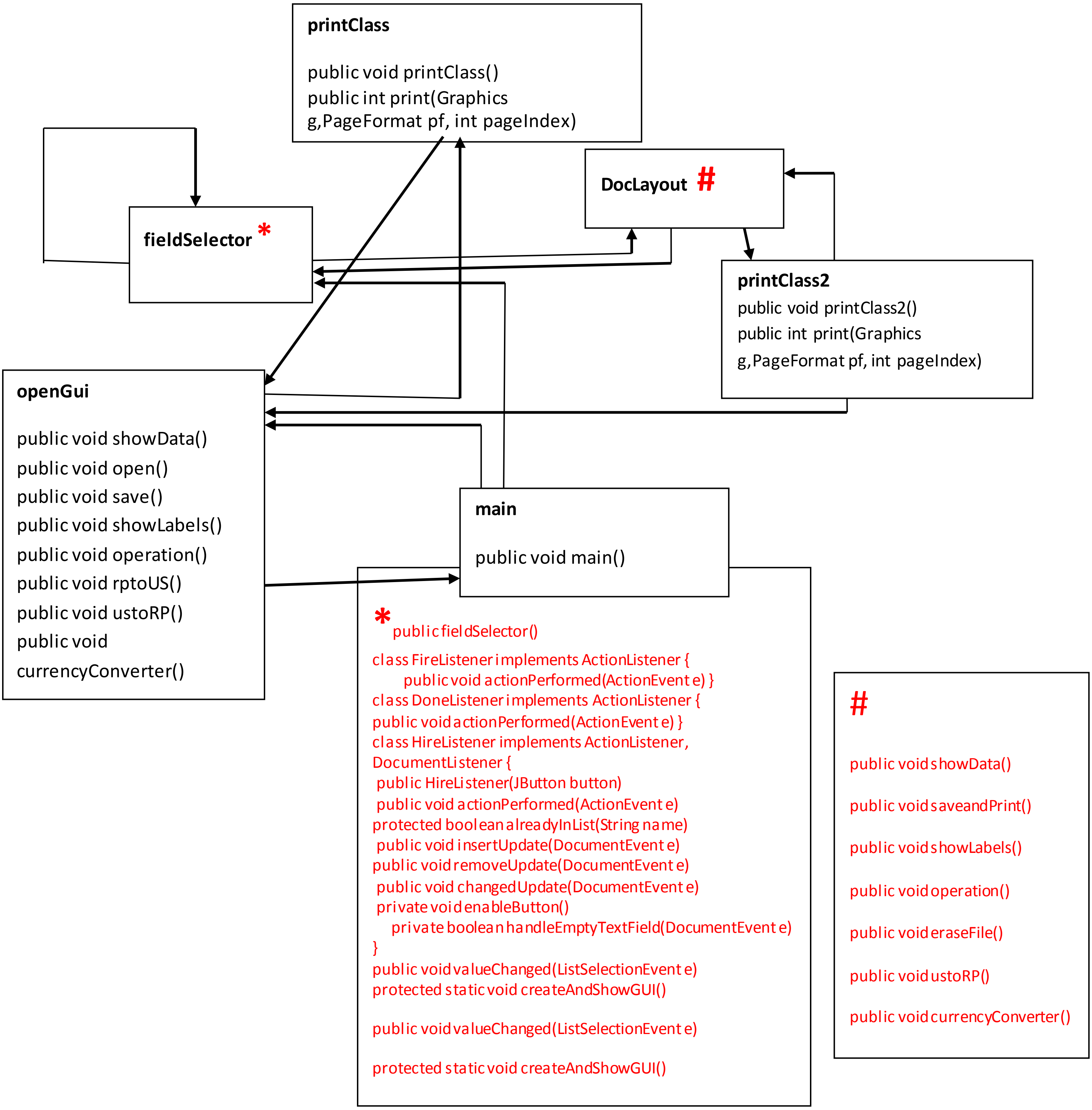
Criterion C-Product Development

Relationships between user defined classes and methods



printClass

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

fieldSelector *

DocLayout #

printClass2
public void printClass2()
public int print(Graphics g,PageFormat pf, int pageIndex)

openGui

public void showData()
public void open()
public void save()
public void showLabels()
public void operation()
public void rptoUS()
public void ustoRP()
public void currencyConverter()

main

public void main()

```
* public fieldSelector()
class FireListener implements ActionListener {
       public void actionPerformed(ActionEvent e) }
class DoneListener implements ActionListener {
public void actionPerformed(ActionEvent e) }
class HireListener implements ActionListener,
DocumentListener {
 public HireListener(JButton button)
 public void actionPerformed(ActionEvent e)
protected boolean alreadyInList(String name)
 public void insertUpdate(DocumentEvent e)
public void removeUpdate(DocumentEvent e)
 public void changedUpdate(DocumentEvent e)
 private void enableButton()
    private boolean handleEmptyTextField(DocumentEvent e)
}
public void valueChanged(ListSelectionEvent e)
protected static void createAndShowGUI()

public void valueChanged(ListSelectionEvent e)

protected static void createAndShowGUI()
```

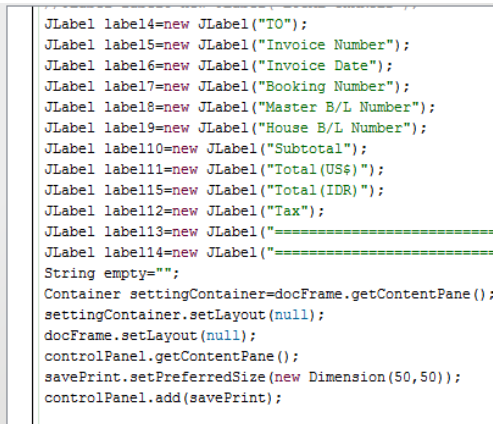
```
#

public void showData()

public void saveandPrint()

public void showLabels()

public void operation()

public void eraseFile()

public void ustoRP()

public void currencyConverter()
```

Techniques Used:

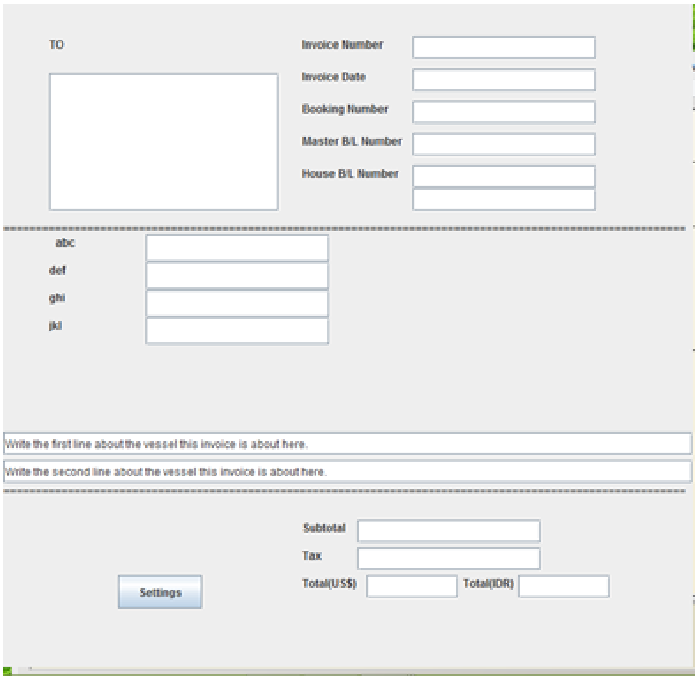
1. Extensive GUI Employment
2. File and ArrayList Handling
3. Implementation of interfaces(Printing and ListSelectionListener)
   and subclasses
4. Extensive event handling
5. Looping

Extensive GUI Employment

This is present in two modes of the software solution: creating a new document and opening an existing one for editing. Methods such as showData() and showLabels() implement various GUI elements ranging from JTextField, JButton JCheckBox, JPanel, JFrame, JLabel, Container and JTabbedPane.



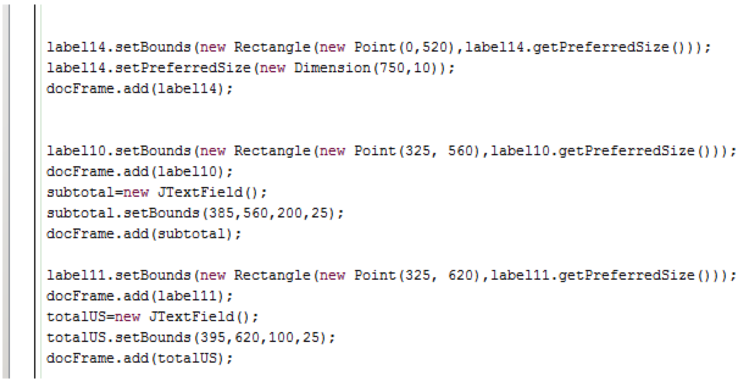
```
JLabel label4=new JLabel("TO");
JLabel label5=new JLabel("Invoice Number");
JLabel label6=new JLabel("Invoice Date");
JLabel label7=new JLabel("Booking Number");
JLabel label8=new JLabel("Master B/L Number");
JLabel label9=new JLabel("House B/L Number");
JLabel label10=new JLabel("Subtotal");
JLabel label11=new JLabel("Total(US$)");
JLabel label15=new JLabel("Total(IDR)");
JLabel label12=new JLabel("Tax");
JLabel label13=new JLabel("=========================
JLabel label14=new JLabel("=========================
String empty="";
Container settingContainer=docFrame.getContentPane();
settingContainer.setLayout(null);
docFrame.setLayout(null);
controlPanel.getContentPane();
savePrint.setPreferredSize(new Dimension(50,50));
controlPanel.add(savePrint);
```



Write the first line about the vessel this invoice is about here.

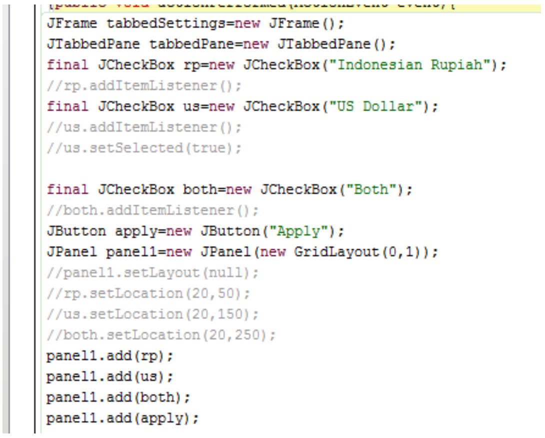
Write the second line about the vessel this invoice is about here.

Various JLabels are declared, as well as Container settingContainter and the JFrame docFrame are initialized to be free of any layout pattern, making room for the GUI elements to be position. The extensive use of GUI results into the planned interface on the right agreed for this software solution.

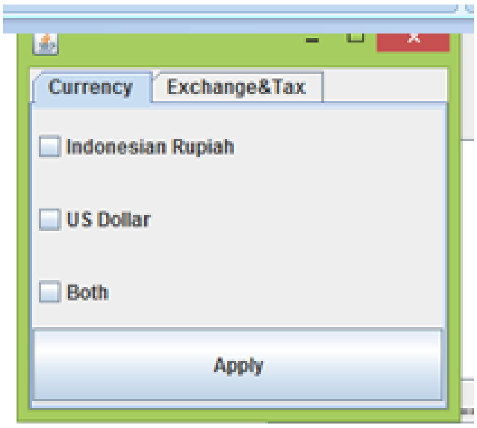


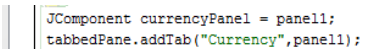
```
label14.setBounds(new Rectangle(new Point(0,520),label14.getPreferredSize()));
label14.setPreferredSize(new Dimension(750,10));
docFrame.add(label14);


label10.setBounds(new Rectangle(new Point(325, 560),label10.getPreferredSize()));
docFrame.add(label10);
subtotal=new JTextField();
subtotal.setBounds(385,560,200,25);
docFrame.add(subtotal);

label11.setBounds(new Rectangle(new Point(325, 620),label11.getPreferredSize()));
docFrame.add(label11);
totalUS=new JTextField();
totalUS.setBounds(395,620,100,25);
docFrame.add(totalUS);
```

The labels declared earlier are now customized with the JTextFields they accompany. label10 is the label containing the string 'Subtotal'. It's size and positio n is declared. At the same time, it's corresponding JTextField, where the subtotal value is needed to be input in is also declared with subtotal=new JTextField, as well as it's size and positioning properties set. docFrame.add(subtotal); places this JTextField into the frame as required.

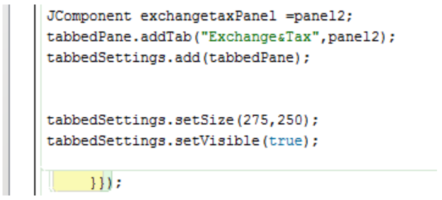


```
JFrame tabbedSettings=new JFrame();
JTabbedPane tabbedPane=new JTabbedPane();
final JCheckBox rp=new JCheckBox("Indonesian Rupiah");
//rp.addItemListener();
final JCheckBox us=new JCheckBox("US Dollar");
//us.addItemListener();
//us.setSelected(true);

final JCheckBox both=new JCheckBox("Both");
//both.addItemListener();
JButton apply=new JButton("Apply");
JPanel panel1=new JPanel(new GridLayout(0,1));
//panel1.setLayout(null);
//rp.setLocation(20,50);
//us.setLocation(20,150);
//both.setLocation(20,250);
panel1.add(rp);
panel1.add(us);
panel1.add(both);
panel1.add(apply);
```



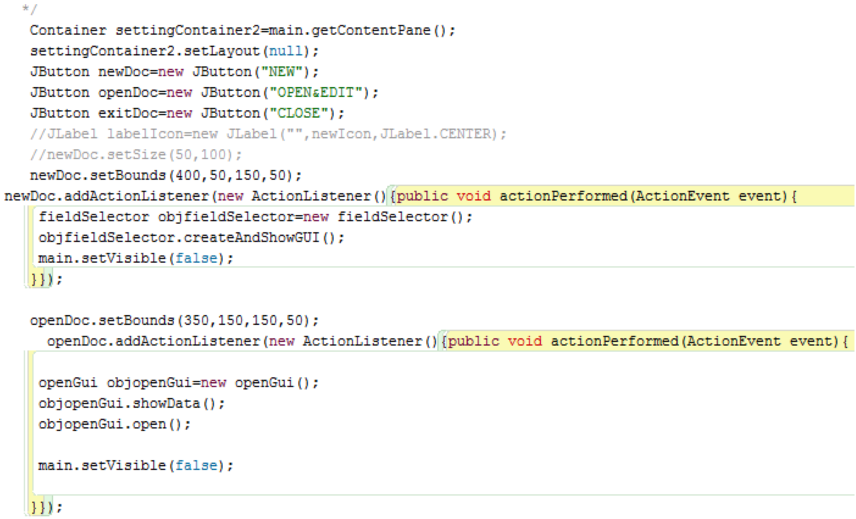


```
JComponent currencyPanel = panel1;
tabbedPane.addTab("Currency",panel1);
```

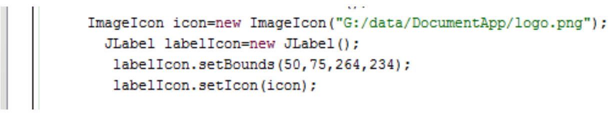


```
JComponent exchangetaxPanel =panel2;
tabbedPane.addTab("Exchange&Tax",panel2);
tabbedSettings.add(tabbedPane);


tabbedSettings.setSize(275,250);
tabbedSettings.setVisible(true);

    }});
```

This code revolves around the implementation of 2 GUI elements to be focused upon, the JTabbedPane and the JCheckBox. The different JCheckBox allows currency settings to be made during calculations of total and subtotal in the document. All elements in the pane(button and checkboxes) are added to a panel, which in turn is declared as one tab, 'Currency' belonging to the entire JTabbedPane tabbedPane. The executed code results in the tab pane on the right.

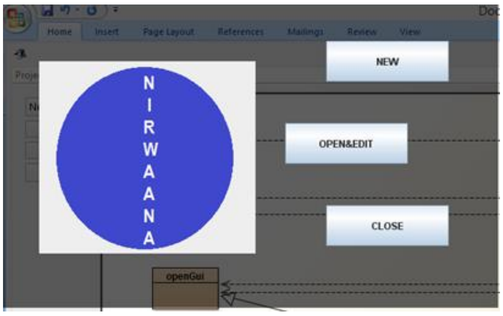


```
  */
   Container settingContainer2=main.getContentPane();
   settingContainer2.setLayout(null);
   JButton newDoc=new JButton("NEW");
   JButton openDoc=new JButton("OPEN&EDIT");
   JButton exitDoc=new JButton("CLOSE");
   //JLabel labelIcon=new JLabel("",newIcon,JLabel.CENTER);
   //newDoc.setSize(50,100);
   newDoc.setBounds(400,50,150,50);
newDoc.addActionListener(new ActionListener(){public void actionPerformed(ActionEvent event){
    fieldSelector objfieldSelector=new fieldSelector();
    objfieldSelector.createAndShowGUI();
    main.setVisible(false);
   }});

   openDoc.setBounds(350,150,150,50);
     openDoc.addActionListener(new ActionListener(){public void actionPerformed(ActionEvent event){

    openGui objopenGui=new openGui();
    objopenGui.showData();
    objopenGui.open();

    main.setVisible(false);

   }});
```

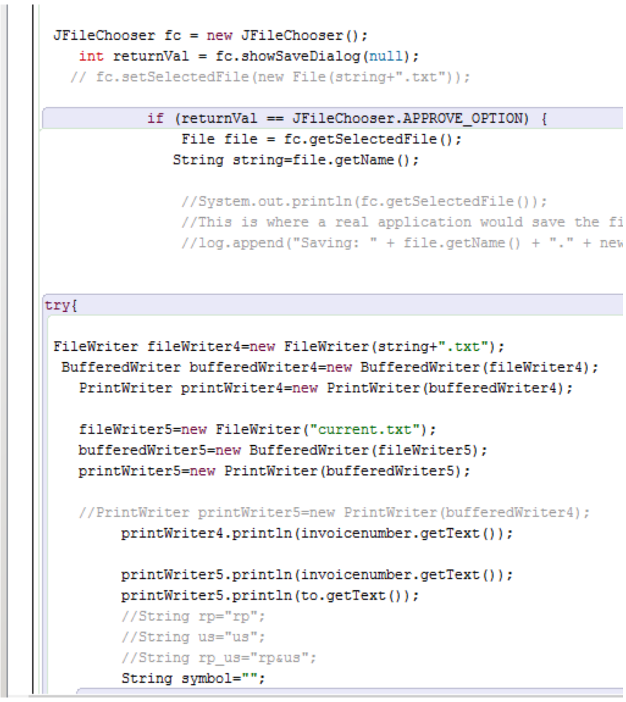


```
ImageIcon icon=new ImageIcon("G:/data/DocumentApp/logo.png");
  JLabel labelIcon=new JLabel();
   labelIcon.setBounds(50,75,264,234);
   labelIcon.setIcon(icon);
```

This GUI implementations focuses on the 3 buttons in the main menu below that direct the user into different modes of operation of the software(new document, editing one or exiting the system). Each JButton is declared with their respective labels, size and position. The methods following the button's declaration that lead it to respective classes for each mode will be explained in the extensive event handling section. The following interface also shows the use of the company's icon by specifying the directory from which the image used is to be extracted and modified to it's respective dimension and position in the interface.

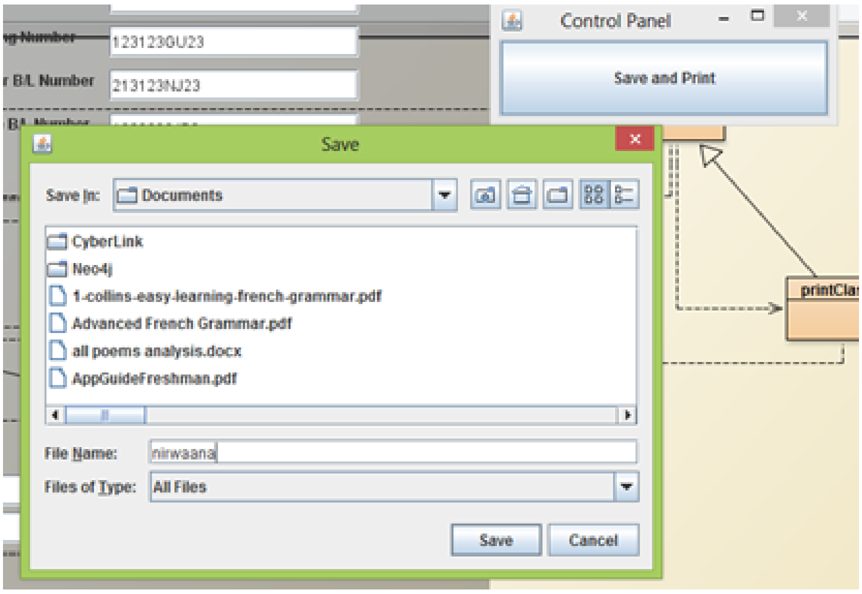


File and ArrayList Handling



```
JFileChooser fc = new JFileChooser();
   int returnVal = fc.showSaveDialog(null);
  // fc.setSelectedFile(new File(string+".txt"));

            if (returnVal == JFileChooser.APPROVE_OPTION) {
                File file = fc.getSelectedFile();
               String string=file.getName();

                //System.out.println(fc.getSelectedFile());
                //This is where a real application would save the fi
                //log.append("Saving: " + file.getName() + "." + new


try{

FileWriter fileWriter4=new FileWriter(string+".txt");
 BufferedWriter bufferedWriter4=new BufferedWriter(fileWriter4);
   PrintWriter printWriter4=new PrintWriter(bufferedWriter4);

   fileWriter5=new FileWriter("current.txt");
   bufferedWriter5=new BufferedWriter(fileWriter5);
   printWriter5=new PrintWriter(bufferedWriter5);

   //PrintWriter printWriter5=new PrintWriter(bufferedWriter4);
        printWriter4.println(invoicenumber.getText());

        printWriter5.println(invoicenumber.getText());
        printWriter5.println(to.getText());
        //String rp="rp";
        //String us="us";
        //String rp_us="rp&us";
        String symbol="";
```

The above code implements a JFileChooser shown below that allow the user to browse the directory to select a file that needs to be edited. The variable string gets the filename through file.getName() and when the user has finished editing, all changes in the JTextField are read from it through get.Text() and is passed onto a FileWriter object, containing the destination of the original filename to update it's contents. The same edited data is also written to current.txt which is then passed onto the printing class. The reason the data had to be written to a buffer file in a different order, accommodates the sorting order of the data that takes place while printing.



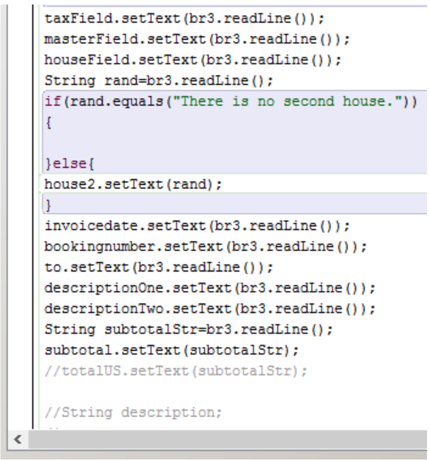
g Number: 1231230U23

r B/L Number: 213123NJ23

Save In: Documents

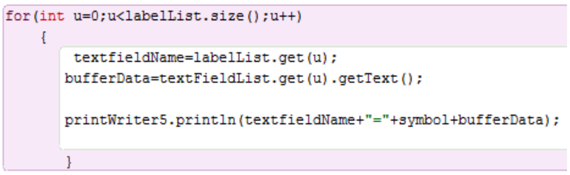
File Name: nirwaana

Files of Type: All Files



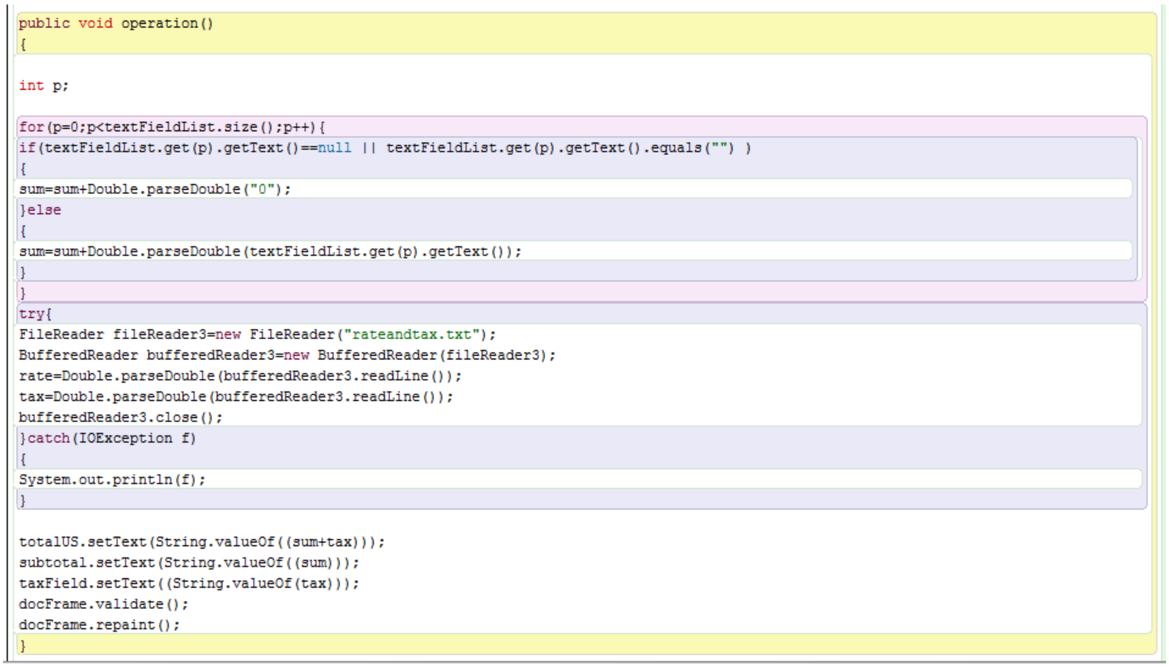
```
taxField.setText(br3.readLine());
masterField.setText(br3.readLine());
houseField.setText(br3.readLine());
String rand=br3.readLine();
if(rand.equals("There is no second house."))
{

}else{
house2.setText(rand);
}
invoicedate.setText(br3.readLine());
bookingnumber.setText(br3.readLine());
to.setText(br3.readLine());
descriptionOne.setText(br3.readLine());
descriptionTwo.setText(br3.readLine());
String subtotalStr=br3.readLine();
subtotal.setText(subtotalStr);
//totalUS.setText(subtotalStr);

//String description;
```

Initially, when JFileChooser is dismissed after the file the user has chosen to use is being loaded, br3.readLine() reads contents from the textfile and places it into the respective textfields for the user to keep track after what is being edited. br3 comprises a pair of a BufferedReader and FileReader Object.



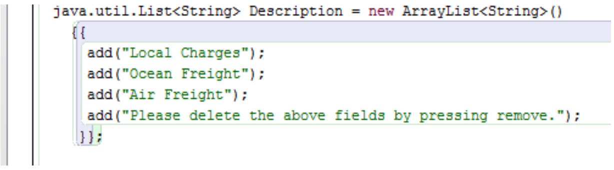
```
for(int u=0;u<labelList.size();u++)
    {
       textfieldName=labelList.get(u);
      bufferData=textFieldList.get(u).getText();

      printWriter5.println(textfieldName+"="+symbol+bufferData);

      }
```

The above code shows, how the ArrayList labelList(contains name of the services the client used) is being read by order of increasing index in a for loop to be written into a file for storage by the printWriter5.println statement. The contents of labelList matches textFieldName, which makes it easy to retrieve the contents of the textfield to store into the textfile while labelList is being read.



```
public void operation()
{

int p;

for(p=0;p<textFieldList.size();p++){
if(textFieldList.get(p).getText()==null || textFieldList.get(p).getText().equals("") )
{
sum=sum+Double.parseDouble("0");
}else
{
sum=sum+Double.parseDouble(textFieldList.get(p).getText());
}
}
try{
FileReader fileReader3=new FileReader("rateandtax.txt");
BufferedReader bufferedReader3=new BufferedReader(fileReader3);
rate=Double.parseDouble(bufferedReader3.readLine());
tax=Double.parseDouble(bufferedReader3.readLine());
bufferedReader3.close();
}catch(IOException f)
{
System.out.println(f);
}

totalUS.setText(String.valueOf((sum+tax)));
subtotal.setText(String.valueOf((sum)));
taxField.setText((String.valueOf(tax)));
docFrame.validate();
docFrame.repaint();
}
```

The following code shows how the ArrayList TextFieldList(contains the TextField objects revolving around the services the client is using) is being read in content by a loop to add all it's individual values to calculate it's total stored in the sum variable that will be put into the totals textfield. Individual values of rate(exchange rate between rupiah and dollar currency and tax are read by bufferedReader3.readLine() from rateandtax.txt and processed with the value of sum calculated earlier. The totalUS.setText() and similar methods updates the entire calculation process by placing the final total values in the JTextField, with respect to rate and tax.

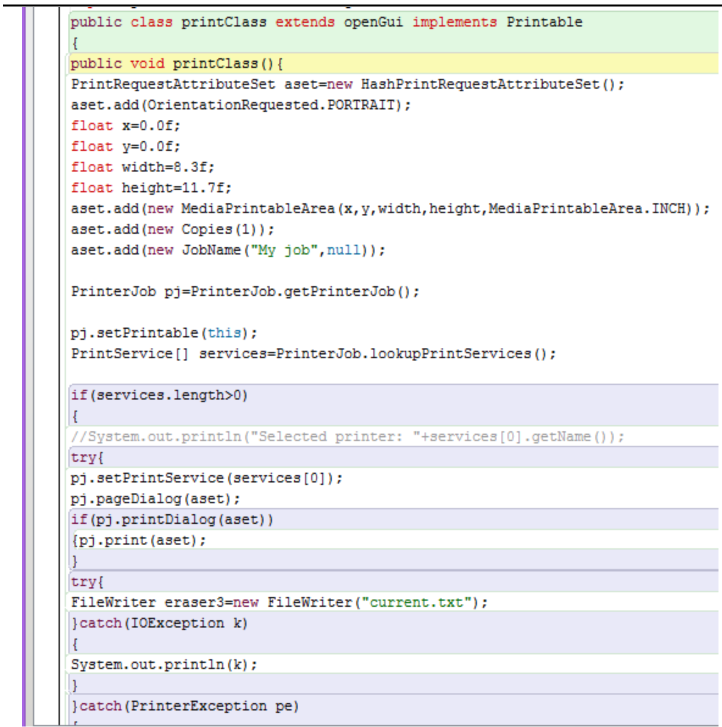


```
java.util.List<String> Description = new ArrayList<String>()
  {{
    add("Local Charges");
    add("Ocean Freight");
    add("Air Freight");
    add("Please delete the above fields by pressing remove.");
  }};
```

The above code shows a declaration of ArrayList Description that holds all the names for JLabels describing the services the client has used in the FieldSelector class. add() methods are their contents add sample values to the ArrayList as an example, that users can certainly remove or modify.

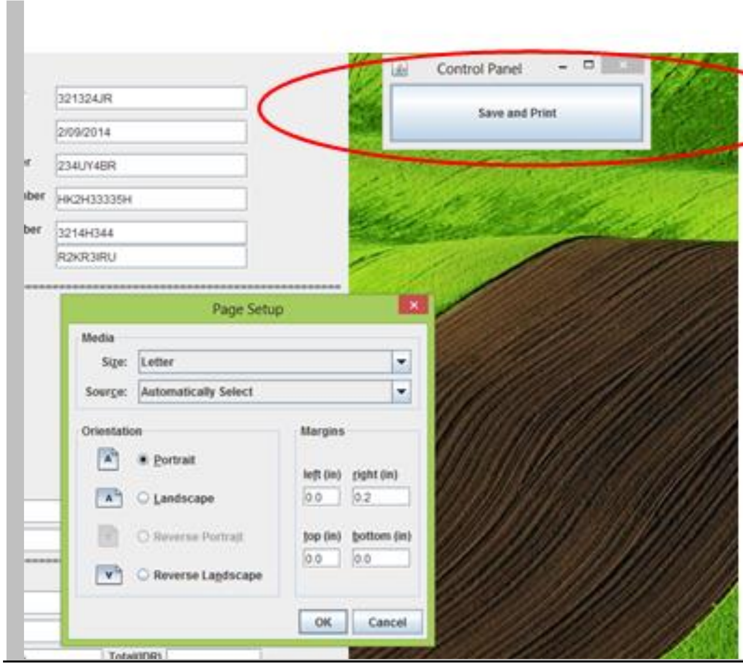
Implementation of interface(Printing and ListSelectionListener)and subclasses

Code	Implementation



```
public class printClass extends openGui implements Printable
{
public void printClass(){
PrintRequestAttributeSet aset=new HashPrintRequestAttributeSet();
aset.add(OrientationRequested.PORTRAIT);
float x=0.0f;
float y=0.0f;
float width=8.3f;
float height=11.7f;
aset.add(new MediaPrintableArea(x,y,width,height,MediaPrintableArea.INCH));
aset.add(new Copies(1));
aset.add(new JobName("My job",null));

PrinterJob pj=PrinterJob.getPrinterJob();

pj.setPrintable(this);
PrintService[] services=PrinterJob.lookupPrintServices();

if(services.length>0)
{
//System.out.println("Selected printer: "+services[0].getName());
try{
pj.setPrintService(services[0]);
pj.pageDialog(aset);
if(pj.printDialog(aset))
{pj.print(aset);
}
try{
FileWriter eraser3=new FileWriter("current.txt");
}catch(IOException k)
{
System.out.println(k);
}
}catch(PrinterException pe)
```

printClass implements the Printable interface. The interface allows the usage of different methods, the setting of the attributes of the paper, the document is to be printed on. These attributes are held in the object aset, that has aset.add(OrientationRequested.PORTRAIT), aset.add(new Copies(1)) and aset.add(new JobName()). The method setPrintService(), stores the name of the selected printer the user chooses to use. pj.PrintDialog shows the print dialog reflecting the aset attributes and proceeds with pj.print(aset) as long as services exist.



321324JR

2/09/2014

234UY4BR

HK2H33335H

3214H344

R2KR3IRU

Media

Size: Letter

Source: Automatically Select

Orientation

● Portrait

○ Landscape

○ Reverse Portrait

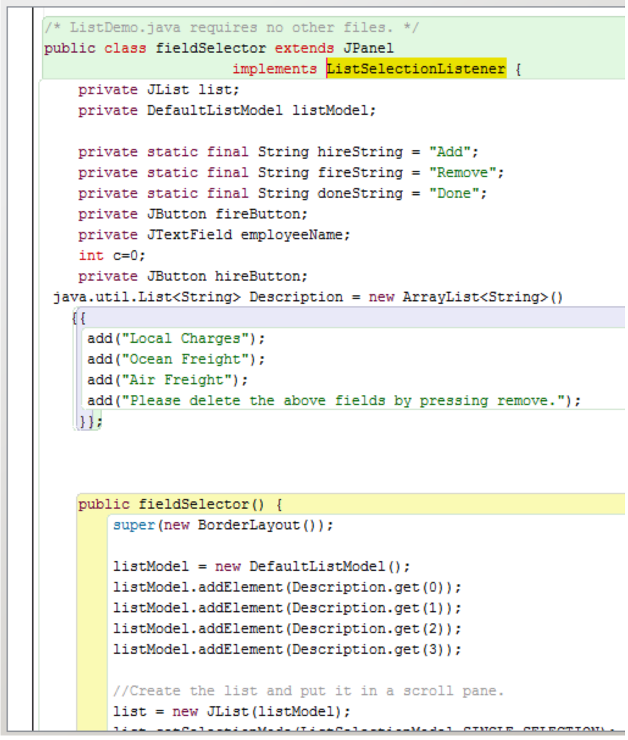
○ Reverse Landscape

Margins

left (in) 0.0 right (in) 0.2

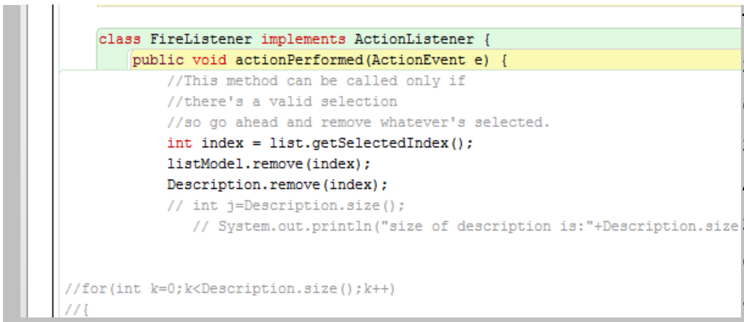
top (in) 0.0 bottom (in) 0.0

The following image shows the visual interpretation of the printing dialog that allows the software to interface with the user and the printer selected itself. The print dialog is the standard dialog seen in windows operating systems to adjust printer settings. The user can select the printer to be used, size of paper, orientation by default and set paper margins. This user interface adds to the easiness in the customization of the printing process.



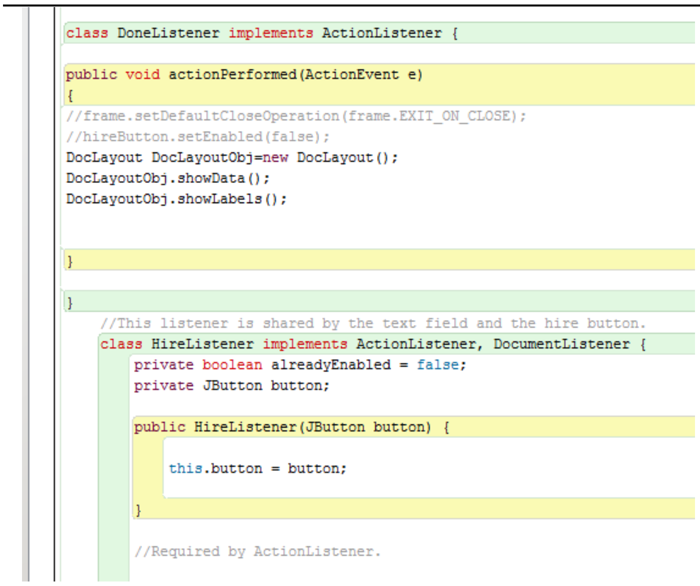
```
/* ListDemo.java requires no other files. */
public class fieldSelector extends JPanel
                      implements ListSelectionListener {
    private JList list;
    private DefaultListModel listModel;

    private static final String hireString = "Add";
    private static final String fireString = "Remove";
    private static final String doneString = "Done";
    private JButton fireButton;
    private JTextField employeeName;
    int c=0;
    private JButton hireButton;
 java.util.List<String> Description = new ArrayList<String>()
   {{
    add("Local Charges");
    add("Ocean Freight");
    add("Air Freight");
    add("Please delete the above fields by pressing remove.");
   }};


    public fieldSelector() {
        super(new BorderLayout());

        listModel = new DefaultListModel();
        listModel.addElement(Description.get(0));
        listModel.addElement(Description.get(1));
        listModel.addElement(Description.get(2));
        listModel.addElement(Description.get(3));

        //Create the list and put it in a scroll pane.
        list = new JList(listModel);
        list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

The code shows the implementation of the interface ListSelectionListener. JList and DefaultListModel are data structures that implement a GUI element that adds data in the form of a list. This is part of the FieldSelector class, where this list plays an important role in providing an interactive user interface where users are able to add, delete and amend the client services used, required for the document. ArrayList Description stores the data values that are being displayed as a list on the GUI. Through the ListenSelectionListener, the listModel is enabled to be updated by the user. listodel.addElement() for instance updates the list with new data elements.



```
    class FireListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            //This method can be called only if
            //there's a valid selection
            //so go ahead and remove whatever's selected.
            int index = list.getSelectedIndex();
            listModel.remove(index);
            Description.remove(index);
            // int j=Description.size();
              // System.out.println("size of description is:"+Description.size


//for(int k=0;k<Description.size();k++)
//{
```

This code is the implementation of subclass FireListener of the main class FieldSelector that is still subjected to the interface ActionListener implemented. This subclass is characterized by the use of void actionPerformed(ActionEvent e). This interface enables the list GUI to listen for what the user does to the list, be it an addition, removal and confirmation of client services. This class is responsible for the removal of elements(client services) in the list, when the user chooses to do so(which is found out by event listening and actionPerformed method) by listModel.remove(index).

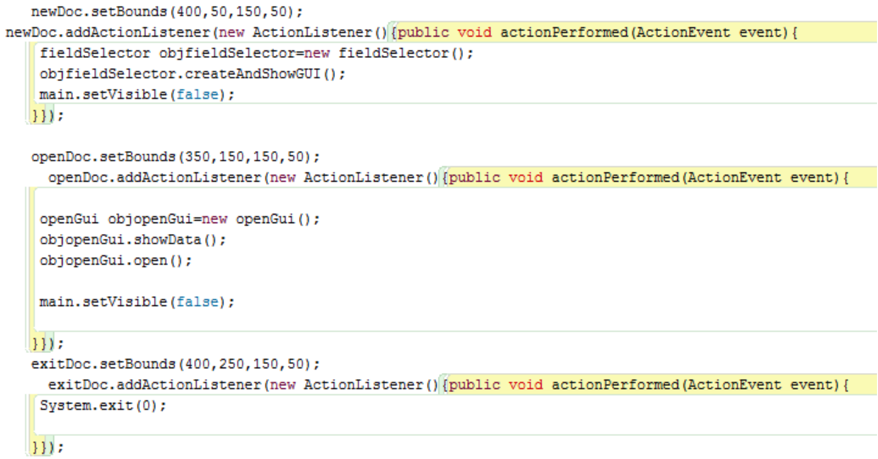


```
class DoneListener implements ActionListener {

public void actionPerformed(ActionEvent e)
{
//frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
//hireButton.setEnabled(false);
DocLayout DocLayoutObj=new DocLayout();
DocLayoutObj.showData();
DocLayoutObj.showLabels();


}

}
    //This listener is shared by the text field and the hire button.
    class HireListener implements ActionListener, DocumentListener {
        private boolean alreadyEnabled = false;
        private JButton button;

        public HireListener(JButton button) {

            this.button = button;

        }

        //Required by ActionListener.
```

Following the above pattern of implementation, subclasses DoneListener and HireListener come under the main class FieldSelector. Like FireListener, they also implement the interface ActionListener(for HireListener, DocumentListener interface also) that listens for any action the user has performed on the data(client services) stored in the list. HireListener listens for the 'Add' button pressed in order to add newer data to the list. DoneListener, directs the user to the user interface of DocLayout, to start creating their document. DoneListener listens for the user to press the 'Done' button as a confirmation that all the client services entered are correct and can be imported to a new document, where the user can begin to work on other details of the new document.

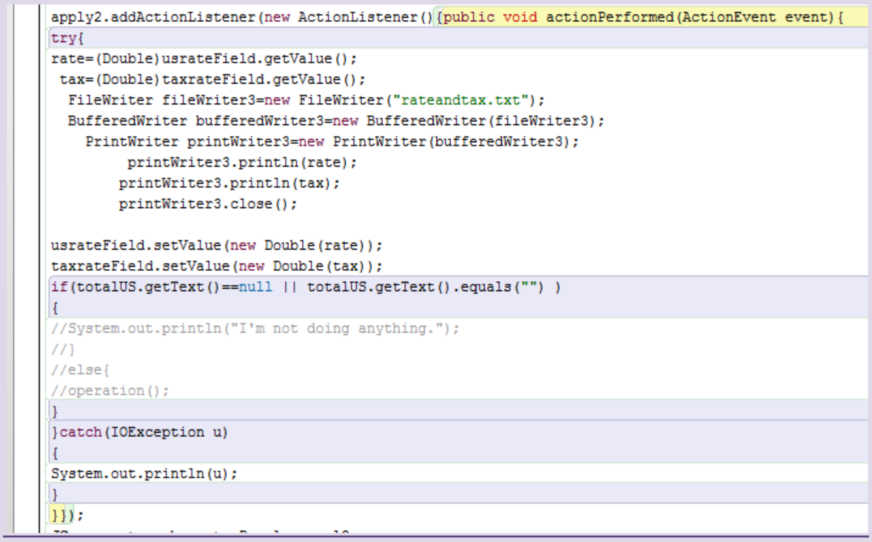
Extensive event listening

Code	Implementation
	The following code is part of the class that implements the user interface of the main menu. Each button is



```
    newDoc.setBounds(400,50,150,50);
 newDoc.addActionListener(new ActionListener(){public void actionPerformed(ActionEvent event){
     fieldSelector objfieldSelector=new fieldSelector();
     objfieldSelector.createAndShowGUI();
     main.setVisible(false);
    }});

    openDoc.setBounds(350,150,150,50);
      openDoc.addActionListener(new ActionListener(){public void actionPerformed(ActionEvent event){

     openGui objopenGui=new openGui();
     objopenGui.showData();
     objopenGui.open();

     main.setVisible(false);

    }});
    exitDoc.setBounds(400,250,150,50);
      exitDoc.addActionListener(new ActionListener(){public void actionPerformed(ActionEvent event){
     System.exit(0);

    }});
```

added an ActionListener object to listen for user action on these buttons. For example, when the event for the newDoc button is triggered by pressing, the user is directed to create a new document. The fieldSelector class is called for the user to firstly decide on what client services characterize the document, before these data is imported to the DocLayout user interface to continue the process of creating the document(link between DocLayout and fieldSelector class can be seen in the DoneListener subclass shown above earlier). The event for the openDoc button is triggered by pressing through ActionListener object and by the actionPerformed method, the user is directed to the user interface generated by the openGui class as well as the open() method is triggered, allowing the user to select the file they want to open for editing in the software. When the exitDoc button is listened for a click from the user, the actionPerformed method, implements the line System.exit(0) allowing the entire software system to halt and close.

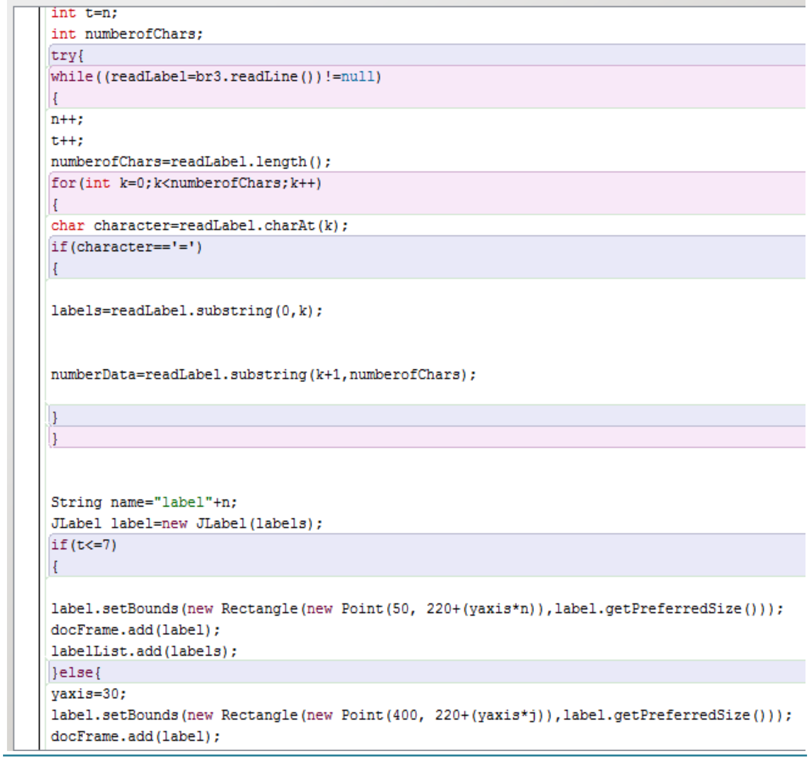


```
apply2.addActionListener(new ActionListener(){public void actionPerformed(ActionEvent event){
try{
rate=(Double)usrateField.getValue();
 tax=(Double)taxrateField.getValue();
  FileWriter fileWriter3=new FileWriter("rateandtax.txt");
  BufferedWriter bufferedWriter3=new BufferedWriter(fileWriter3);
    PrintWriter printWriter3=new PrintWriter(bufferedWriter3);
         printWriter3.println(rate);
        printWriter3.println(tax);
        printWriter3.close();

usrateField.setValue(new Double(rate));
taxrateField.setValue(new Double(tax));
if(totalUS.getText()==null || totalUS.getText().equals("") )
{
//System.out.println("I'm not doing anything.");
//}
//else{
//operation();
}
}catch(IOException u)
{
System.out.println(u);
}
}});
```

The above code is part of the Settings component present in document editing and creation modes of the software. The settings component accommodates change of currency exchange rate and tax percentage while calculating totals for client services within the document created or edited. Individual JTextFields are declared for rate and tax and a JButton called 'Apply' is declared. An ActionListener object is declared for the button to listen for the user's press of the button to apply exchange rate and tax percentage changes in currency. The actionPerformed method writes the new values into rateandtax.txt and refreshes them in the respective JTextFields in the Settings component.

Looping

Code

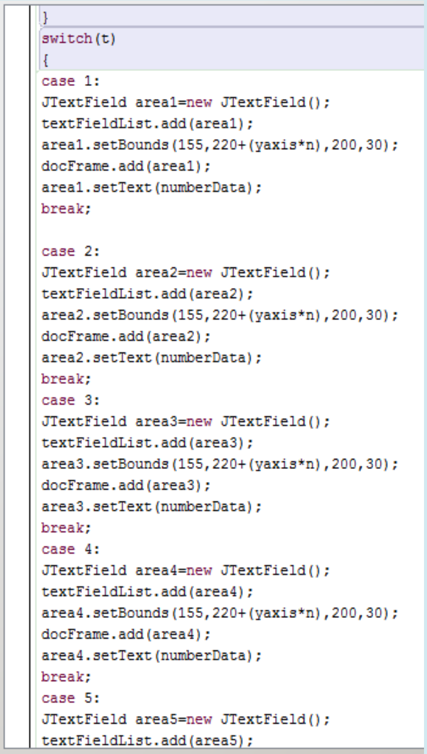


```java
int t=n;
int numberofChars;
try{
while((readLabel=br3.readLine())!=null)
{
n++;
t++;
numberofChars=readLabel.length();
for(int k=0;k<numberofChars;k++)
{
char character=readLabel.charAt(k);
if(character=='=')
{

labels=readLabel.substring(0,k);


numberData=readLabel.substring(k+1,numberofChars);

}
}


String name="label"+n;
JLabel label=new JLabel(labels);
if(t<=7)
{

label.setBounds(new Rectangle(new Point(50, 220+(yaxis*n)),label.getPreferredSize()));
docFrame.add(label);
labelList.add(labels);
}else{
yaxis=30;
label.setBounds(new Rectangle(new Point(400, 220+(yaxis*j)),label.getPreferredSize()));
docFrame.add(label);
```

Implementation

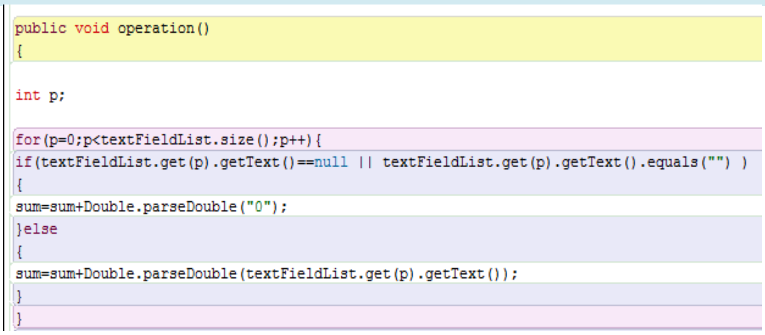
The following code implements the use of a while and a for loop. This is part of the method showLabels() in the openGui class where, labels for data fields are read from the document to be edited to show in the user interface for editing. When the labels are being read in the while loop, they are modified in the characters that make up the contents of the readLabel string through the for loop. They are modified into a new reference string , 'name' (by the functions of substring and the length of characters stored)in order to directly reference JLabel objects and decide their size and positioning on the user interface. This is a parallel task, involving reading of labels and referencing and creating JLabel objects directly to display in the user interface, when each label is being read sequentially in the while loop, as JLabel object names and the actual data labels stored in the document file have some similarity in spelling.

The following code continues from the above code sequence. A switch case is employed, as each JLabel object is being created and position, it is accompanied by JTextField that will hold data appropriate to the description of the JLabel. Each JLabel and JTextField have fixed positions. Each JTextField is identified uniquely by the variable t that holds the line number being read. Since, data is stored in a fixed order and format, it is easy to identify that which JTextField has to be generated for which JLabel and where it should be placed. numberData reads the value to be placed in the JTextField and textFieldList stores the JTextField object unique for each document, especially pertaining to the services used by the client.



```
}
switch(t)
{
case 1:
JTextField area1=new JTextField();
textFieldList.add(area1);
area1.setBounds(155,220+(yaxis*n),200,30);
docFrame.add(area1);
area1.setText(numberData);
break;

case 2:
JTextField area2=new JTextField();
textFieldList.add(area2);
area2.setBounds(155,220+(yaxis*n),200,30);
docFrame.add(area2);
area2.setText(numberData);
break;
case 3:
JTextField area3=new JTextField();
textFieldList.add(area3);
area3.setBounds(155,220+(yaxis*n),200,30);
docFrame.add(area3);
area3.setText(numberData);
break;
case 4:
JTextField area4=new JTextField();
textFieldList.add(area4);
area4.setBounds(155,220+(yaxis*n),200,30);
docFrame.add(area4);
area4.setText(numberData);
break;
case 5:
JTextField area5=new JTextField();
textFieldList.add(area5);
```

The for loop reads JTextFields objects and their contents from the ArrayList textFieldList and adds them into the double variable sum to get the total and subtotal values of the document after further processing with optional change of currency and tax value.



```
public void operation()
{

int p;

for(p=0;p<textFieldList.size();p++){
if(textFieldList.get(p).getText()==null || textFieldList.get(p).getText().equals("") )
{
sum=sum+Double.parseDouble("0");
}else
{
sum=sum+Double.parseDouble(textFieldList.get(p).getText());
}
}
```