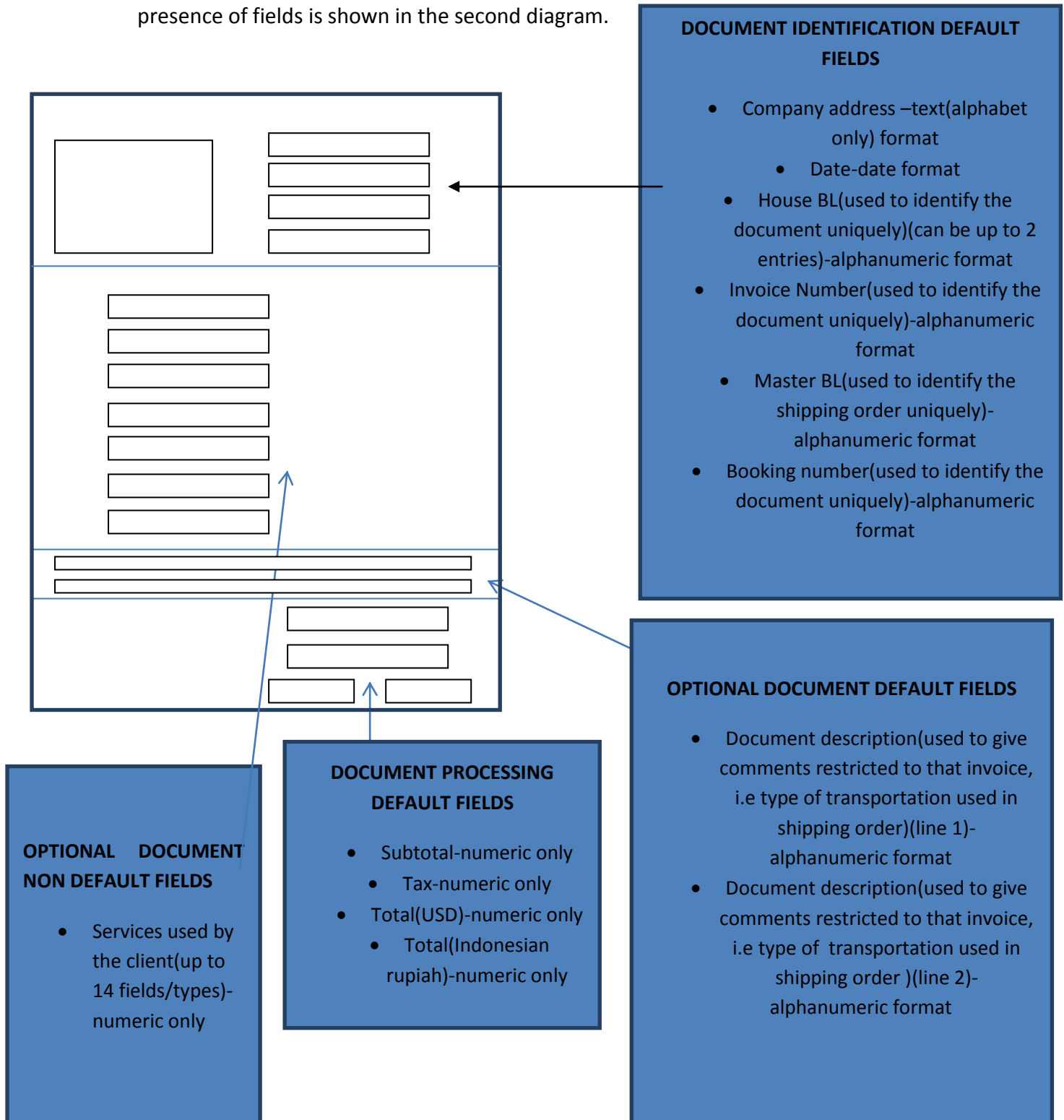


## Criterion B-Design

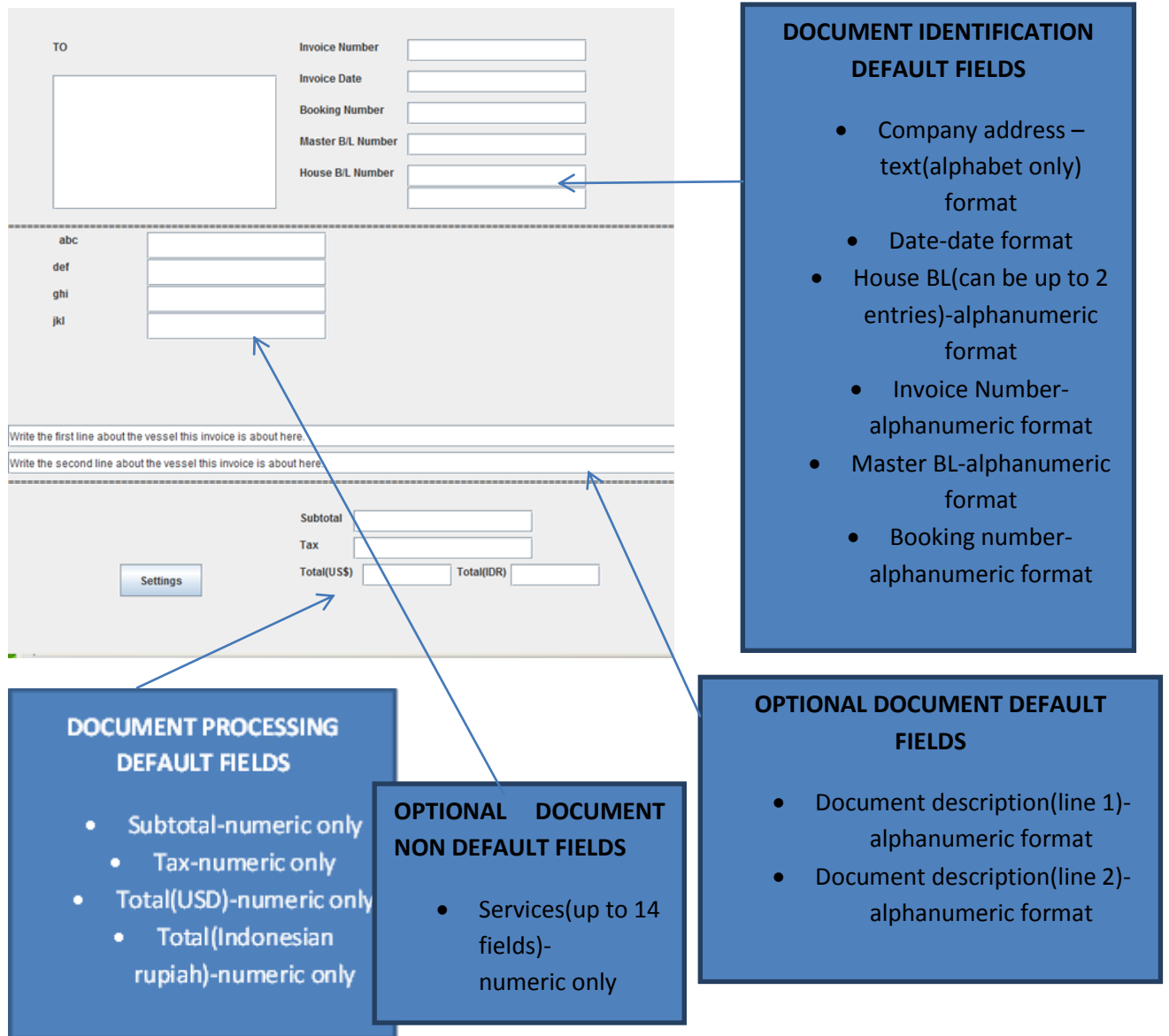
These are the proposed fields that should accept data from the user in the actual program(input interface). It processes the same data that the user processed manually to create customer invoices. The data type of these fields is mentioned next to the field name in the blue boxes below as well as the supposed physical arrangement of the fields on the user interface. A more visual representation of the input user interface and the presence of fields is shown in the second diagram.



There are several fields that are fixed for every document. They accompany every document created and must be filled. They are document identification as well as document processing default fields. The only data that makes each document unique are the services data fields. Each client uses different services of the company each time. There can be 1-14 services used and this is always variable.

I wish to make this user interface with similar to the design of the special paper on which customer invoices are printed(attached in the Appendix). Just as the paper is flexible in accommodating variable amounts of data, so must the user interface do the same. This is able to easily organize changes in data. Being familiar with the paper's structure also induces the same familiarity in the user interface.

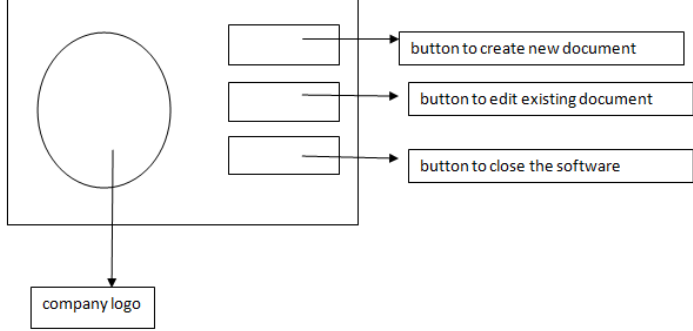
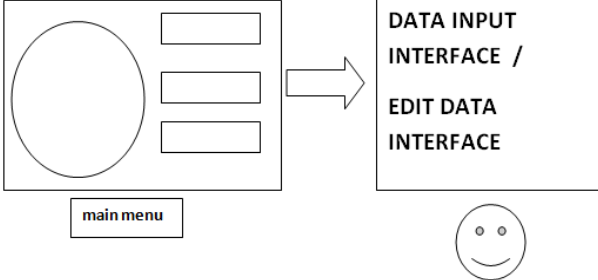
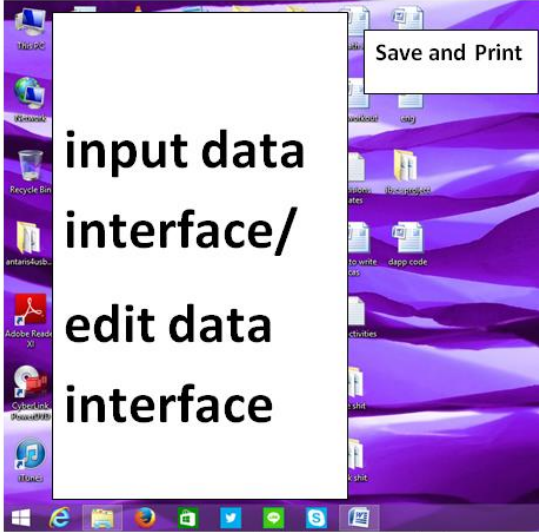
Modeling the user interface in Java based on the paper used to print invoices, produces this kind of an interface. The interface must emphasize on neatness and organization as shown below.

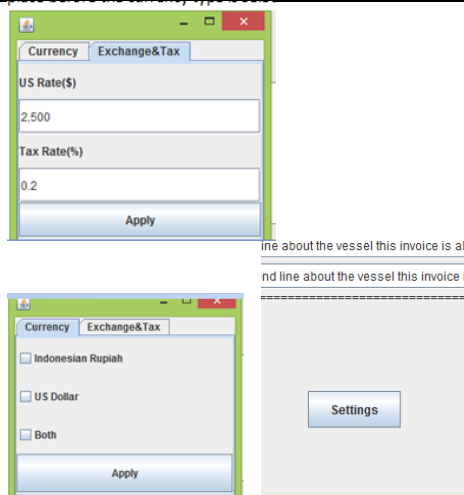


As the services are at the center of the interface area, the interface is more robust as only the middle section is to be modified with more services, new calculations etc. It does not strike the idea of an interface that is constantly changing and digressing from the way data has to be entered.

The following graphic sketches show how different aspects of the system are designed.

Aspects	Comments
<div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid black; padding: 5px; margin: 5px;">DATA INPUT INTERFACE</div> <div style="margin: 0 10px;">↔</div> <div style="border: 1px solid black; padding: 5px; margin: 5px;">EDIT DATA INTERFACE</div> </div>	<p>View data the same way it was originally created; minimize drastic changes in printouts</p> <p>Similar methods for generating</p>

 <p>main menu interface</p>	<p>user input and editing interfaces</p> <p>Not many interface elements on screen; easier to direct software into a single mode</p>
 <p>user modes possible</p>	<p>Better not support multitasking since interfaces for input and editing documents are similar, accidental erasure of data can occur. Data will be volatile to change with this nature of interface.</p> <p>Software can run separately on another computer to work on another mode, this tasking is possible.</p> <p>This task in the company(input/editing) is meant for an individual only.</p>
 <p>save and print button and method</p>	<p>Button should not be placed inside the interface window, because premature work cannot be saved. Must prevent cluttering of important interface elements(adjusting settings and saving at the same time) and accidental pressing of the save and print button.</p> <p>Save and print method when pressed must allow remaining data fields to be calculated based on values of other data fields(i.e addition of client services cut by tax to calculate totals, subtotals)</p>



settings component

Mainly for adjusting currency in which data fields processed by calculation should be in

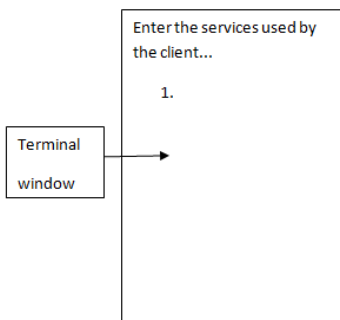
Also must enable inter conversion between US Dollar and Indonesian Rupiah based on fluctuating rates((allows representation of total, subtotal fields in one of either currencies or both)

Tax percentage must be modified when needed

Settings component button must be placed within the document interface, because document is the source of modification

Pressing 'Apply' must reflect changes in the total, subtotal data fields in both the tabs

Tabbed panes do not distract the overall view of the document being worked upon. this is a minor feature.

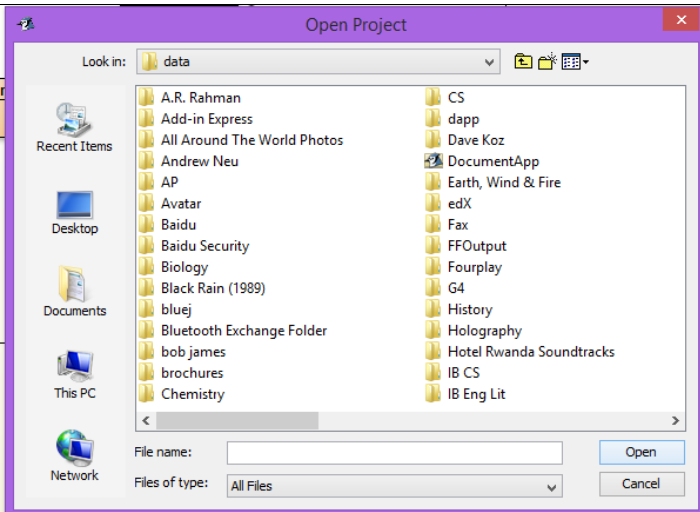


By using the terminal window, as thought initially, will importantly disable the user to not amend or remove their input once the enter key is pressed

There must be a way for users to amend, remove and edit their input before finally importing all these fields to the document

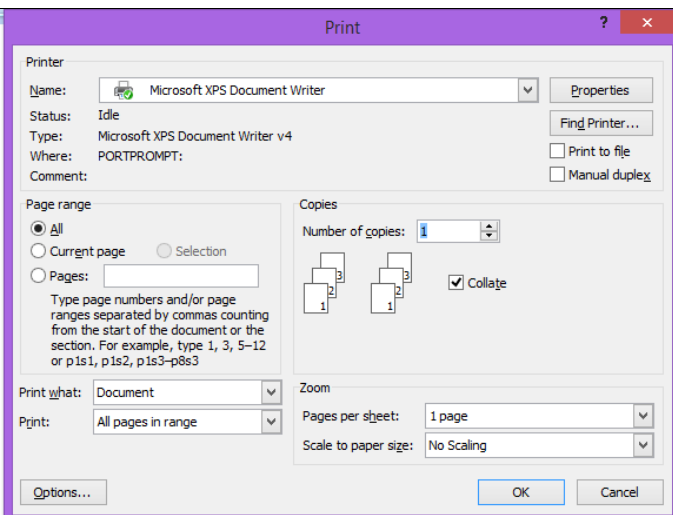
Input through thhe terminal window cannot be considered

Input must be versatile to reflect recent changes by the user on client services before finally importing it to the new document



**File chooser to lead recent document for editing**

Before editing an existing document, users must open a file they want to edit. this has to be done using a file chooser. It makes the process easier . it makes the process swift for the user to browse the directory of the computer and makes no room necessary for file handling code when the user types incorrectly the file name as well as when the file does not exist at all.



**printing dialog before printing**

The user must have complete control over the way the documents are printed. A printer dialog box is necessary to guide users in their printing preferences. This dialog box has to appear when the user pressed the save and print button.

## Proposed Classes

### Name

main

### Description

Runs the main menu of the program, calls openGui and DocLayout depending on the user to create a new document(DocLayout) or modify an existing one(openGui).

### printClass

reads all data input in the user interface of openGui class and issues commands to position these data on certain fixed locations on paper, and print it.

### openGui

Opens a dialog box to select a file to load in a similar user interface present in DocLayout, takes

<b>fieldSelector</b>	care of calculations and save the changes made. Takes user input to customize the data fields the form should have(depends on client's services).
<b>DocLayout</b>	Imports data fields from the fieldSelector class , generates fixed fields for all documents and places them in the appropriate places in a user interface generated, in the type of a form. Takes care of calculations and save the changes made.
<b>printClass2</b>	reads all data input in the user interface of DocLayout class and issues commands to position these data on certain fixed locations on paper, and print it.

<u>Proposed Subclasses</u>		
<u>Class</u>	<u>Subclass</u>	<u>Description</u>
main	----	----
DocLayout	----	----
openGui	----	----
printClass <u>extends</u> openGui <u>implements</u> Printable	----	----
printClass2 <u>extends</u> DocLayout <u>implements</u> Printable	----	----
fieldSelector <u>extends</u> JPanel <u>implements</u> ListSelectionListener	FireListener implements ActionListener	Remove whatever data fields are selected in JPanel. Store data fields in a text file to customize the new document on DocLayout.
	HireListener implements ActionListener, DocumentListener	Add data fields input by the user one at a time. Each field will be visible in JPanel upon acceptance. Store data fields in a text file to customize the new document on DocLayout, with these fields chosen.
	DoneListener implements ActionListener	Calls the DocLayout class. Invokes a method to load fields from the text file into appropriate sections of the DocLayout GUI.

<u>Proposed Methods</u>			
<u>Class</u>	<u>Subclass</u>	<u>Method</u>	<u>Description</u>
main	-----	public void main()	Set company icon on frame,

		set background color and 3 buttons , add 3 buttons('NEW', 'OPEN&EDIT' and 'CLOSE'). public void actionPerformed(ActionEvent event) for the first two buttons trigger the calling of classes DocLayout, openGui and initialization of user interfaces. For the last button, System.exit(0) to quit the program.
	public static void main(String fcuk[])	Declare mainObj, object for the class main. public void main() called.

Class	Subclass	Method	Description
openGui	-----	public void showData()	Declares controlPanel JFrame to accommodate savePrint JButton, that calls public void save() when pressed, to save the user's document. Creates and positions numerous JLabels and JTextField within settingContainer container's getContentPane() to set the document's input interface. Creates JTabbedPane in JPanel as well as 'Settings' JButton to open JTabbed Pane. Add checkboxes for selection of different currencies in the first 'Currency' tab. Variables used to keep track of which currency type is selected, when 'Apply button is pressed'.



		<p>Adds second tab to JTabbed Pane, 'Exchange&amp;Tax' and JTextFields to allow input of tax rate and exchange rate between dollar and rupiah. When 'Apply' is pressed. Tax and change values are updated in textfile for future operations.</p>
	<pre>public void open()</pre>	<p>Creates a JFileChooser object, and calls openFileDialog ()for user to select a document to open from the computer directory. Loads each line of data from the file into appropriate JTextFields. Calls public void showLabels(), to label the data loaded.</p>
	<pre>public void save()</pre>	<p>Writes all data modified in the opened document to a textfile called current.txt to proceed for printing in a different order. Current file opened contents are erased with the use of File Write eraser2 object, and all data is rewritten along with the modifications made by the user to ensure all data of the document is up to date.</p>
	<pre>public void showLabels()</pre>	<p>This is used to especially load client services that varies</p>

		<p>for each document invoice created. While the description of the data fields unique and fixed from the file name prescribed to open in openGui, is being read. The field descriptions while read, are being modified using the functions of substring() and charAt(). This is because the names of the fields stored are similar to the JTextField and JLabels object names stored. While reading, these new GUI elements are created to open the document accurately with correct services etc. Positioning of elements also occur, where some fields and labels have fixed positions based on a fixed format(some fields are required for all documents) and the variable service fields are fixed by incrementing loops, to display the GUI as neat as possible.</p>
	public void operation()	<p>This method is used to calculated the data fields of total and subtotal, based on variable services used by clients in each document. The ArrayList textFieldList</p>

		stores the JTextField objects that contain the values to be added up. A for loop reads this list and adds the contents of the objects storing it in the variable 'sum'. Exchange rate and tax values are read from rateandtax.txt to further process the sum, into total and subtotal.
	Public void eraseFile()	Erases the contents of the current file opened, to rewrite all data, modified or unmodified into the same file opened for editing.
	public void rptoUS()	This method works similarly as the previous, using a loop and reading the textFieldList to convert all values in the JTextField objects to US Dollars to Indonesian Rupiah. Totals and subtotals after tax are also converted into this currency.
	public void ustoRP()	This method works similarly as the previous, using a loop and reading the textFieldList to convert all values in the JTextField objects to Indonesian Rupiah to US Dollars. Totals and subtotals after tax are also converted into this currency.

		<pre>public void currencyConverter()</pre>	<p>This method calls the previous methods <code>ustoRP()</code> and <code>rptoUS()</code> in direct response to the user's action in the <code>JTabbedPane</code> triggered by the <code>actionPerformed()</code> method in the <code>showData()</code> method. The user's action consist of selecting checkboxes involving Indonesian, US or both currencies to present the document in. This method checks the current currency the data in the document is in and then calls the above methods appropriately to produce changes in data fields' values pertaining to client services used and totals calculations.</p>
--	--	--	---

Class	Subclass	Method	Description
<b>DocLayout</b>		<code>public void showData()</code>	Same mechanism as <code>openGui's showData()</code> method.
	-----	<pre>public void saveandPrint()</pre>	Writes all data to a textfile called <code>current.txt</code> to proceed for printing in a different order. Also writes data to a textfile whose file name is specified by the user.
		<code>public void showLabels()</code>	Same mechanism as <code>openGui's showLabels()</code> method.

		public void operation()	Same mechanism as openGui's operation() method.
		public void eraseFile()	Erases the contents of a textfile called 'Description.txt' by creating a new FileWriter object. This textfile holds all the client services fields used imported by the fieldSelector class into the new document, so make it available to store the next new document's fields.
		public void ustoRP()	Same mechanism as openGui's ustoRP() method.
		public void rptoUS();	Same mechanism as openGui's rptoUS() method.
		public void currencyConverter()	Same mechanism as openGui's currencyConverter() method.

Class	Subclass	Method	Description
<b>printClass and printClass2</b>	-----	public void printClass() and public void printClass2()	Defines the attributes of the media(paper) used to print the document on. aset.add (new MediaPrintableArea (x,y,width,height,MediaPrintableArea.INCH)), aset.add(new Copies(1)); define this attributes. With the help of the print job chosen by the user in the print dialog box called, pj.printDialog(aset), the method checks if the length of services/jobs is >0 and executes printing by, pj.print(aset). After printing, the contents of current.txt that holds all data within the document in a different order, is erased, to make room for the next document's contents.
		public int print(Graphics g,PageFormat	This methods returns the value of whether a 'PAGE' exists to be printed. This method deals with the mechanics of placing the data

		pf, int pageIndex)	on the paper at fixed positions using methods such as paper.setImageableArea(),g2d.drawString() where a Graphics object is employed Graphics2D g2d=(Graphics2D)g; to enable 2D printing in Java. Other attributes such as Font font=new Font("Arial",Font.PLAIN,10); is also set.
--	--	-----------------------	---

Class	Subclass	Method	Description
<b>FieldSelector</b>	FireListener implements ActionListener	public void actionPerformed(ActionEvent e)	This method is triggered after it has identified the user's action of deleting a client service through the fieldSelector() GUI. The same data is also removed from the ArrayList Description that is holding all client services to be imported to a new document GUI set by DocLayout.
	DoneListener implements ActionListener	public void actionPerformed(ActionEvent e)	This method connects the DocLayout and fieldSelector class, where after the user has confirmed all client services used in the fieldSelector GUI, the DocLayout GUI creates an interface

		containing the customized client services the user input in the fieldSelector GUI.
	HireListener implements ActionListener, DocumentListener	
	public void actionPerformed(ActionEvent e)	This method is triggered after it has identified the user's action of adding a client service through the fieldSelector() GUI. The same data is also added from the ArrayList Description that is holding all client services to be imported to a new document GUI set by DocLayout.
	protected boolean alreadyInList(String name)	This method checks for string equality. Matching results in the client services are returned in the fieldSelector GUI.
	public void insertUpdate(DocumentEvent e)	Enables the 'Add' button using enableButton() when the user has entered the

		first character in the fieldSelector input box.
	public void removeUpdate(DocumentEvent e)	This method makes the 'Add' button unable to be pressed using enableButton() when the fieldSelector input box is empty. Cannot input an empty client service field.
	public void changedUpdate(DocumentEvent e)	Enables the 'Add' button using enableButton() when a client service is modified in the fieldSelector input box, when selected.
	private void enableButton()	Enables the 'Add' button.
	private boolean handleEmptyTextField(DocumentEvent e)	Method called by removeUpdate (DocumentEvent e) to disable the 'Add' button.
-----	public void valueChanged(ListSelectionEvent e)	Keeps a check on the range in which a client service added already, is selected. If a selection is made, out of bounds, the 'Remove' button is made unavailable to press, else the button is enabled.
-----	protected static void	Create the GUI



---

createAndShowGUI()

and show it.

---

---

### **Proposed TextFiles**

<b><u>Name</u></b>	<b><u>Description</u></b>
Current.txt	After the user chooses to save and print a new document or modified document in DocLayout or openGui class, the data is read from the JTextFields into the GUI and written to this file in a particular order. This file is accessed in printClass class and printClass2 class to allow printing of the user's data onto paper.
Rateandtax.txt	Updates exchange rate and tax values in DocLayout and openGui class set by the user, for further processing
Description.txt	Used in the fieldSelector class, where the data field names of client services are stored here, for importing to a new document GUI after the user has confirmed all client services in the fieldSelector input GUI.

---

---

### **Proposed ArrayLists**

<b><u>Name</u></b>	<b><u>Description</u></b>
textFieldList	Stores the names of JTextField objects present within a document in DocLayout and openGui class, in order to read object names and values to display in the GUI or write to a file.
labelList	Stores the name of client services data fields present within a document in DocLayout and openGui class, in order to read the data field names to display in the GUI or write to a file.
labelamountList	Stores the values of client services data fields present within a document in DocLayout and openGui class, in order to read data field values to display in the GUI or write to a file.
Description	Same as labelList, only used in fieldSelector() class.

---

---

**Tests Required**

<b><u>Test type</u></b>	<b><u>Nature</u></b>	<b><u>Example</u></b>
Document data must be loaded accurately when opened	Pressing 'open' after selecting a textfile to be opened from the directory in JFileChooser	Data from each line of the text file is read completely and placed in the correct locations within the document interface, maintaining neatness and organization.
Changes made to document data must be saved correctly	Pressing the 'Save and Print' button.	All data modified or not is written completely to an existing file that will be erased first to renew its contents.
New documents initially created must be saved correctly	Pressing the 'Save and Print' button.	All data is e written completely to an existing file that will be erased first to renew its contents.
Document data must be read correctly for printing	Pressing the 'Save and Print' button.	Data from each line of the text file is read completely and attributed to the correct locations within the document paper, maintaining neatness and organization.
Client services entered must be imported correctly into the new document the user will be working on	Pressing the 'Done' button in the fieldSelector user input interface.	Services must be stored in a textfile(Description.txt) that must be read correctly and placed in the correct locations within the document interface, maintaining neatness and organization.

---