/\*

\*

\* Programmer: Victor W. Corsi

\*

\* While working on this program I frequently checked my progress by have the program run

\* without debugging.

\*

\*/

import java.io.File;

import java.io.FileOutputStream;

import java.io.OutputStream;

import java.util.ArrayList;

import java.util.List;

import java.awt.Dimension;

import java.awt.EventQueue;

import java.awt.GridLayout;

import javax.swing.JFrame;

import javax.swing.JPanel;

import java.awt.BorderLayout;

import javax.swing.JButton;

import java.awt.event.ActionListener;

import java.awt.event.ActionEvent;

import javax.swing.DefaultListModel;

import javax.swing.JFileChooser;

import javax.swing.JLabel;

import javax.swing.JMenuBar;

import javax.swing.JMenuItem;

import javax.swing.JList;

import javax.swing.JMenu;

import javax.swing.ListSelectionModel;

import java.awt.Color;

import javax.swing.SwingConstants;

import java.awt.SystemColor;

import javax.xml.bind.JAXBContext;

import javax.xml.bind.JAXBException;

import javax.xml.bind.Marshaller;

import javax.xml.bind.Unmarshaller;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.transform.Result;

public class Window {

/\*

\* In order for the save and load methods to work properly the JLists and DefaultListModel

\* have to be implemented within the Window Class because if implemented within the Initialize

\* Method it would not be able to be called again.

\*/

JList<?> PartList, AddedToList;

DefaultListModel<String> PartName, PartNamed;

private JFrame frame;

/\*\*

\* Launch the application. The main method is the entry point to a Java application.

\* For this assessment, you shouldn't have to add anything to this.

\*/

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

public void run() {

try {

Window window = new Window();

window.frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Create the application. This is the constructor for this Window class.

\* All of the code here will be executed as soon as a Window object is made.

\*/

public Window() {

initialize();

}

/\*\*

\* Initialize the contents of the frame. This is where Window Builder

\* will generate its code.

\*/

public void initialize() {

frame = new JFrame();

frame.setBounds(100, 100, 450, 300);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

/\*

\*

\* This is the Panel for the Original JList: "Parts List"

\* The List has to be set with the set elements of the Tool Names

\* I setup the JList so that the list would have a large text format

\* covering 1/3 of the Window.

\*

\*/

JPanel panel = new JPanel();

frame.getContentPane().add(panel, BorderLayout.WEST);

panel.setPreferredSize(new Dimension(150,300));

//Setup the PartName and PartNamed DefaultListModels to be used

PartName = new DefaultListModel<String>();

PartNamed = new DefaultListModel<String>();

//Here are all the Parts that will set to be used

//and stored in the PartName ListDefaultModel

PartName.addElement("Case");

PartName.addElement("Motherboard");

PartName.addElement("CPU");

PartName.addElement("GPU");

PartName.addElement("PSU");

PartName.addElement("RAM");

PartName.addElement("HDD");

PartList = new JList<String>(PartName);

PartList.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

PartList.setSelectedIndex(0);

PartList.setVisibleRowCount(30);

PartList.setBackground(Color.WHITE);

PartList.setPreferredSize(new Dimension(150,300));

panel.add(PartList);

/\*

\*

\* This is the Panel is for the JList: "Added To List"

\* In order to allow the the JList to appear I had to setup similar

\* to the "Parts List" without actually having any added element within

\* the JList

\*

\*/

JPanel panel\_1 = new JPanel();

frame.getContentPane().add(panel\_1, BorderLayout.EAST);

panel\_1.setPreferredSize(new Dimension(150,300));

AddedToList = new JList<String>(PartNamed);

panel\_1.add(AddedToList);

AddedToList.setBackground(Color.WHITE);

AddedToList.setPreferredSize(new Dimension(150,300));

/\*

\* This is the Panel for the Buttons:

\* >> which I have as the add button

\* << which I have as the remove button

\*/

JPanel panel\_2 = new JPanel();

frame.getContentPane().add(panel\_2, BorderLayout.CENTER);

panel\_2.setPreferredSize(new Dimension(150,300));

panel\_2.setLayout(new GridLayout(2,30,5,50));

/\*

\* With the Add Button ">>" it takes the the items from the JList

\* that holds the names and add it to the JList that is blank, I

\* set it to only place it in the blank JList.

\*/

JButton addbutton = new JButton(">>");

addbutton.addActionListener(new ActionListener() {

/\*

\* The way the Add Button(>>) works is by adding the selected string

\* that is on the DefaultListModel PartName which holds the Parts List

\* as a string element to the DefaultListModel PartNamed.

\*/

public void actionPerformed(ActionEvent arg0) {

String name = (String) PartList.getSelectedValue();

PartNamed.addElement(name);

}

});

panel\_2.add(addbutton);

/\*

\* With the Remove Button "<<" it takes the the selected item from the JList

\* PartNamed and removes it from the JList.

\*/

JButton removebutton = new JButton("<<");

removebutton.addActionListener(new ActionListener() {

/\*

\* The way the Remove Button(<<) works is by removing the selected value

\* on the right side DefaultListModel PartNamed.

\*/

public void actionPerformed(ActionEvent e) {

String name = (String) AddedToList.getSelectedValue();

PartNamed.removeElement(name);

}

});

panel\_2.add(removebutton);

/\*

\* This is the Menu Bar and it's functions:

\* Save File, Load File, and Exit Program.

\*/

JMenuBar menuBar = new JMenuBar();

frame.setJMenuBar(menuBar);

JMenu mnNewMenu = new JMenu("File");

menuBar.add(mnNewMenu);

JMenuItem mntmLoadFile = new JMenuItem("Load File");

mntmLoadFile.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

//Since the Right side will load in a file of elements

//it is best to remove all elements that are already in

//the right-side JList to make room for the new elements that

//will be added.

PartNamed.removeAllElements();

//call the Load Method:

try {

load();

} catch (Exception e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

}

});

mnNewMenu.add(mntmLoadFile);

JMenuItem mntmSaveFile = new JMenuItem("Save File");

mntmSaveFile.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

//call the Save Method when the Save File Menu item is selected

try {

save(PartNamed);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

});

mnNewMenu.add(mntmSaveFile);

JMenuItem mntmExitProgram = new JMenuItem("Exit Program");

mntmExitProgram.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

//when Exit Program menu item is selected it closes the program

System.exit(0);

}

});

mnNewMenu.add(mntmExitProgram);

}

public static void save(DefaultListModel<String> PartNames) throws Exception {

//Load the elements from the PartsNamed into an Array list to be stored in the

//order it was placed into file.xml

ArrayList<String> save = new ArrayList<>();

for (int i = 0; i<PartNames.size(); i++){

save.add((String)PartNames.getElementAt(i));

}

XMLParts parts = new XMLParts();

//setup the parts from the array list using the XMLParts Class

parts.setPart(save);

//Call The JAXBContext annotated in the XMLParts Class.

//Since the Marshaller class contains JAXB package annotations that must be processed

//so that the Parts can be formated from string into XML.

JAXBContext context =JAXBContext.newInstance(XMLParts.class);

Marshaller marshaller = context.createMarshaller();

marshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, true);

//The OutputStream calls the file and marshaller.marshal saves the parts into

//file.xml

OutputStream file = new FileOutputStream("C:\\file.xml");

marshaller.marshal(parts, file);

//This is to test to see if the program will save to a xml file format.

marshaller.marshal(parts, System.out);

}

public void load() throws Exception {

//Call The JAXBContext annotated in the XMLParts Class.

//Since the Unmarshaller class contains JAXB package annotations that must be

//processed

JAXBContext context =JAXBContext.newInstance(XMLParts.class);

//The Unmarshaller governs the process of deserializing XML data by

//validating the XML data within the file.xml as it is unmarshalled.

Unmarshaller unmarshaller = context.createUnmarshaller();

File file = new File("C:\\file.xml");

XMLParts parts = (XMLParts) unmarshaller.unmarshal(file);

//Create a String Array List to contain the list of parts from the file

//so that the array can be be loaded in order through a for loop into the

//DefaultListModel.

List<String> load = new ArrayList<>();

load = parts.getPart();

for (int i = 0; i<load.size(); i++){

PartNamed.addElement(load.get(i));

}

//This is a test to see if the elements are loading from the file

System.out.println(load);

}

}