```
package RH.PE;

import java.io.IOException;

/**
* Includes the main function and the execution thread
*/
public class Exec{

	public static void main(String[] args) throws IOException{

		Thread execute = new Thread(new Manag(), "executeThread");
		execute.start();
	}
}
```

```
package RH.PE;

import java.io.IOException;

/**
* Includes the runnable implementation for the execution thread
*/
public class Manag implements Runnable {

	private GUI software;
	@Override
	public void run() {
		try {
			software = new GUI();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

}
```

```
package RH.PE;

/**
* This Class is used to store parameters of food received from the XML.
*/

public class Food
{
	private String name;
	private String price;
	private String Descr;
	private String Calories;

	public Food() {
		price = "";
		name = "";
		Descr = "";
		Calories = "";
	}

	public Food(String name,String Descr, String price, String Calories) {
		this.price = "$"+price;
		this.name = name;
		this.Descr = Descr;
		this.Calories = Calories;
	}

	/**
	* These are the class' Setters and Getters:
```

```
        */
        // setters
        public void setPrice(String Price) {this.price = Price;}
        public void setDescr(String Descr) {this.Descr = Descr;}
        public void setName(String name) {this.name = name;}
        public void setCalories(String Calories) {this.Calories = Calories;}

        // getters
        public String getPrice() {return this.price;}
        public String getDescr() {return this.Descr;}
        public String getName() {return this.name;}
        public String getCalories() {return this.Calories;}
}
```

```
package RH.PE;

import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.table.DefaultTableModel;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import java.awt.*;
import java.awt.event.*;
import java.io.IOException;

public class GUI extends JFrame {
  /**
        * This Class is used to create the engine's UI
        */
        private JTextField leftPrice, rightPrice, caloriesLeft, caloriesRight,jtfFilter;
        private JButton reset, compare, update, foodMenu, addVal;
        private JPanel panel, leftPanel, comparePanel, searchPanel;
        private JFrame menuFrame;
        private GridBagConstraints location;
        private JComboBox<String> comboxLeft;
        private JComboBox<String> comboxRight;
        private Model exec;
        private static final Logger logger = LogManager.getLogger(GUI.class.getName());
        private JLabel leftLabel, rightLabel;
        private int addedVals = 0;

        public GUI() throws IOException {
```

```
super("Food Calories Comparsion");
setLayout(new FlowLayout());

//creation of the engine
logger.info("Starting up the engine...");
exec = new Model();

this.setTitle("Food Calories Comparsion");
panel = new JPanel(new GridBagLayout());
location = new GridBagConstraints();
location.insets = new Insets(10,10,10,10);

// Text Field Creation (price and calories):
// ********************** Left fields:
//Price field
logger.info("Creating text fields....");
leftPrice = new JTextField("0", 5);
leftPrice.setEditable(false);
location.gridx = 0;
location.gridy = 1;
panel.add(leftPrice, location);

//Calories field
caloriesLeft = new JTextField("0", 5);
caloriesLeft.setEditable(false);
location.gridx = 0;
location.gridy = 2;
panel.add(caloriesLeft, location);
```

```
//************************ Right fields:
//Price field
caloriesRight = new JTextField("0", 5);
caloriesRight.setEditable(false);
location.gridx = 2;
location.gridy = 2;
panel.add(caloriesRight, location);

//Calories field
rightPrice = new JTextField("0", 5);
rightPrice.setEditable(false);
location.gridx = 2;
location.gridy = 1;
panel.add(rightPrice, location);

// Button Creation:
logger.info("Creating buttons...");
foodMenu = new JButton("Menu");
location.gridx = 0;
location.gridy = 3;
panel.add(foodMenu, location);

update = new JButton("Update");
location.gridx = 1;
location.gridy = 3;
panel.add(update, location);

reset = new JButton("Reset");
location.gridx = 2;
```

```
            location.gridy = 3;
            panel.add(reset, location);

            compare = new JButton("Compare");
            compare.setBackground(Color.orange);
            location.gridx = 1;
            location.gridy = 2;
            panel.add(compare, location);

            //selection box creation
            logger.info("Creating panels...");
            leftPanel = new JPanel(new BorderLayout());
            comparePanel = new JPanel(new BorderLayout());

            //search in the menu
            jtfFilter = new JTextField(10);
            jtfFilter.getDocument().addDocumentListener(new DocumentListener(){

public void insertUpdate(DocumentEvent e) {
   String text = jtfFilter.getText();

   if (text.trim().length() == 0) {
      exec.getRowSorter().setRowFilter(null);
   } else {
            exec.getRowSorter().setRowFilter(RowFilter.regexFilter("(?i)" + text));
   }
}
public void removeUpdate(DocumentEvent e) {
   String text = jtfFilter.getText();
```

```
        if (text.trim().length() == 0) {
                exec.getRowSorter().setRowFilter(null);
        } else {
                exec.getRowSorter().setRowFilter(RowFilter.regexFilter("(?i)" + text));
        }
    }
    public void changedUpdate(DocumentEvent e) {
      logger.info("Unsupported operation");
        throw new UnsupportedOperationException("Not supported yet.");
    }
});


                leftLabel = new JLabel("Food:");
                rightLabel = new JLabel("Food:");
                leftLabel.setVisible(true);
                rightLabel.setVisible(true);

                // Combo Box Creation
                logger.info("Creating ComboBoxes...");
                comboxLeft = new JComboBox<String>();
                fillCombox(comboxLeft);
                location.gridx = 0;
                location.gridy = 0;
                leftPanel.add(comboxLeft, BorderLayout.EAST);
                leftPanel.add(leftLabel, BorderLayout.WEST);
                panel.add(leftPanel, location);
```

```
comboxRight = new JComboBox<String>();
fillCombox(comboxRight);
location.gridx = 2;
location.gridy = 0;
comparePanel.add(comboxRight, BorderLayout.EAST);
comboxRight.setSelectedIndex(1);
comparePanel.add(rightLabel, BorderLayout.WEST);
panel.add(comparePanel, location);

// Adding action listeners
logger.info("Adding listeners...");
theHandler handler = new theHandler();
leftPrice.addActionListener(handler);
reset.addActionListener(handler);
compare.addActionListener(handler);
foodMenu.addActionListener(handler);
update.addActionListener(handler);



//adding the food menu table
logger.info("Adding the food table (menu)....");
menuFrame = new JFrame();
menuFrame.setLayout(new FlowLayout());

addVal = new JButton("Add");
addVal.addActionListener(handler);
menuFrame.add(addVal, BorderLayout.EAST);
```

```
                menuFrame.add(new JScrollPane(exec.getTable()));
                menuFrame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
                menuFrame.setSize(700, 600);



                //search box for menu table
                searchPanel = new JPanel(new BorderLayout());
                searchPanel.add(new JLabel("Specify a word to match:"), BorderLayout.WEST);
                searchPanel.add(jtfFilter, BorderLayout.CENTER);
                menuFrame.add(searchPanel);



    Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
                menuFrame.setLocation(dim.width/2-this.getSize().width/2 - 480, dim.height/2-
this.getSize().height/2);
                this.setLocation(dim.width/2-this.getSize().width/2, dim.height/2-
this.getSize().height/2);
                add(panel);


                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setSize(650,230);
                setVisible(true);
        }



        private class theHandler implements ActionListener {
                public void actionPerformed(ActionEvent event) {
                        String tempCalc = "";
```

```
                double secondComp;
                double tempINS;
                if (event.getSource() == foodMenu) {
                        menuFrame.setVisible(true);

                } else if (event.getSource() == update){
                        updateDataBase();

                }
                else if (event.getSource() == addVal){

                  /**
                        * Adding a value to the food menu:
                        * If one of the dialogs is cancelled, the operation will abort,
                        * and the value will not be added
                        */
                        DefaultTableModel temp =
(DefaultTableModel)exec.getTable().getModel();
                        String insertName = JOptionPane.showInputDialog("Enter Food Name");
                        if (insertName!=null)
                        {
                                String insertDescr = JOptionPane.showInputDialog("Enter
Description");
                                if (insertDescr!=null)
                                {
                                        try {
                                                String insertPrice =
JOptionPane.showInputDialog("Enter Price (in $)");
                                                Double.parseDouble(insertPrice);
                                                        if (insertPrice!=null)
```

```
                                                    {
                                                        String insertCalories =
JOptionPane.showInputDialog("Enter Calories");
        Double.parseDouble(insertCalories);
                                                        if (insertCalories!=null)
                                                        {

        comboxLeft.addItem(insertName);
        comboxRight.addItem(insertName);
        exec.getFoodList().add(new Food(insertName, insertDescr, insertPrice, insertCalories));
        exec.addValue(insertName, insertDescr, insertPrice, insertCalories);
        temp.fireTableDataChanged();
                                                                addedVals++;
                                                        }
                                                    }
                                                }
                                            catch(java.lang.NumberFormatException e)
                                            {
                                                logger.error("Non-number input detected while
adding item to the menu");
                                                JOptionPane.showMessageDialog(null, "Invalid
input","Error",JOptionPane.ERROR_MESSAGE);
                                            }
                                        }
                                    }
                                }
```

```
                else if (event.getSource() == reset) {
                        reset();
                }
                else if (event.getSource() == compare) {
                  /**
                        * This function compares the values of Price and Calories of both
selected food items
                        * It gets both selected indexes from the Combo Boxes and gets their
respective details from the food list.
                        *
                        * The function seeks the lower values of Price and Calories between
both food items.
                        * The lower value's text field will be colored green, and the higher
value's field will be colored red.
                        * If both values are equal, the text fields' color will be set to white.
                        */

                        //set the corresponding price values for the food in the text fields
        leftPrice.setText(exec.getFoodList().get((comboxLeft.getSelectedIndex())).getPrice());
        rightPrice.setText(exec.getFoodList().get((comboxRight.getSelectedIndex())).getPrice());


                        // get the text from the text field and change it to double
                        tempCalc = leftPrice.getText();
                        secondComp = Double.parseDouble(rightPrice.getText().substring(1));
                        tempINS = Double.parseDouble(tempCalc.substring(1));

                        if(tempINS - secondComp > 0)
                        {
```

```
                rightPrice.setBackground(Color.green);
                leftPrice.setBackground(Color.red);
            }
            else if(tempINS - secondComp < 0)
            {
                rightPrice.setBackground(Color.red);
                leftPrice.setBackground(Color.green);
            }
            else //if both values are equal
            {
                rightPrice.setBackground(Color.white);
                leftPrice.setBackground(Color.white);
            }

caloriesLeft.setText(exec.getFoodList().get((comboxLeft.getSelectedIndex())).getCalories());

caloriesRight.setText(exec.getFoodList().get((comboxRight.getSelectedIndex())).getCalories());

            // change the text to double
            tempCalc = caloriesLeft.getText();
            secondComp = Double.parseDouble(caloriesRight.getText());
            tempINS = Double.parseDouble(tempCalc);

            if(tempINS - secondComp > 0)
            {
                caloriesRight.setBackground(Color.green);
                caloriesLeft.setBackground(Color.red);
            }
```

```
                        else if(tempINS - secondComp < 0)
                        {
                                caloriesRight.setBackground(Color.red);
                                caloriesLeft.setBackground(Color.green);
                        }
                        else //if both values are equal
                        {
                                caloriesRight.setBackground(Color.white);
                                caloriesLeft.setBackground(Color.white);
                        }
                }
        }
}

public void reset() //reset all fields to their starting state
{
   /**
        * Resets all text fields to 0 and backround color to white
        */
        caloriesRight.setBackground(Color.white);
        caloriesLeft.setBackground(Color.white);
        caloriesRight.setText("0");
        caloriesLeft.setText("0");

        rightPrice.setBackground(Color.white);
        leftPrice.setBackground(Color.white);
        leftPrice.setText("0");
        rightPrice.setText("0");
```

```
            comboxLeft.setSelectedIndex(0);
            comboxRight.setSelectedIndex(1);
        }
        public void fillCombox(JComboBox<String> ComBox) //fills the combo boxes with data
        {
            for(int i=0; i<exec.getFoodList().size();i++)
            {
                ComBox.addItem(exec.getFoodList().get(i).getName());
            }
        }

        public void updateDataBase(){
            //xml parse XML
            logger.info("Resetting to XML values...");
            exec.removeFood(addedVals);
            addedVals=0;
            comboxLeft.removeAllItems();
            comboxRight.removeAllItems();
            fillCombox(comboxLeft);
            fillCombox(comboxRight);
            reset();
            DefaultTableModel dm = (DefaultTableModel)exec.getTable().getModel();
            dm.setRowCount(0);
            exec.createRows(dm);
            dm.fireTableDataChanged();
        }
}
```

```
package RH.PE;

import java.util.ArrayList;
import javax.swing.BorderFactory;
import javax.swing.JLabel;
import javax.swing.JTable;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import javax.swing.table.TableRowSorter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import java.awt.Color;
import java.io.*;
import java.net.*;

/**
* This Class includes a dynamic list of Food class objects.
* It also includes the functionality of connecting to the web service,
* Printing information to the output file and filling the food menu table.
*/
```

```
public class Model
{
		private ArrayList<Food> food;
		private InputStream url;
		private JTable foodMenu;
		private TableRowSorter<TableModel> rowSorter;
		private FileOutputStream fos = null;
		private DataOutputStream whereTo = null;
	    private static final Logger logger = LogManager.getLogger(Model.class.getName());

		public DefaultTableModel createTmodel() {
			DefaultTableModel TModel = new DefaultTableModel();
			TModel.addColumn("Name");
			TModel.addColumn("Description");
			TModel.addColumn("Price");
			TModel.addColumn("Calories");
			createRows(TModel);
			return TModel;
		}

	Model()
	{
		try
		{
			fos = new FileOutputStream("foodOutput.txt");
			whereTo = new DataOutputStream(fos);
		}
		catch (FileNotFoundException e2)
		{
```

```
                logger.error(e2.getMessage());
                e2.printStackTrace();
        }

        food = new ArrayList<Food>();

        //parsing the XML file
        try{
                // read it from the url
                url = new URL("https://www.w3schools.com/xml/simple.xml").openStream();
        }catch(Exception e){
                try
                {
                        // should the url be unavailable, the program will read from the local
xml instead
                        logger.error(e.getMessage());
                        url = new FileInputStream("foodMenu.xml");
                } catch (FileNotFoundException e1)
                {
                        logger.error(e1.getMessage());
                        e1.printStackTrace();
                }
        }
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        int i, j;

        //constructing the Food Menu Table
        logger.info("Building the Menu.....");
        try
```

```
{
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse(url);

        //get food details by tags
        logger.info("Getting food details.....");
        NodeList foodList = doc.getElementsByTagName("food");
        for(i=0; i < foodList.getLength(); ++i)
        {
            /**
            * reading the data from the XML by tags and adding it to the food list.
            * The received data is also printed to the Output file.
            */
            food.add(i, new Food());
            Node tempFood = foodList.item(i);
            if(tempFood.getNodeType() == Node.ELEMENT_NODE)
            {
                Element tempFood_ = (Element)tempFood;
                NodeList infoList = tempFood_.getChildNodes();
                for(j=0; j < infoList.getLength(); ++j)
                {
                    Node temp = infoList.item(j);
                    if(temp.getNodeType() == Node.ELEMENT_NODE)
                    {
                        Element info = (Element) temp;
                        if(info.getTagName() == "name"){

food.get(i).setName(info.getTextContent());
```

```
                                                whereTo.writeBytes("Name: " +
food.get(i).getName()+"\n");
                                        }

                                        else if(info.getTagName() == "price"){

        food.get(i).setPrice(info.getTextContent());
                                                whereTo.writeBytes("Price: " +
food.get(i).getPrice()+"\n");
                                        }
                                        else if(info.getTagName() == "description"){

        food.get(i).setDescr(info.getTextContent());
                                                whereTo.writeBytes("Description: " +
food.get(i).getDescr()+"\n");
                                        }
                                        else if(info.getTagName() == "calories"){

        food.get(i).setCalories(info.getTextContent());
                                                whereTo.writeBytes("Calories: " +
food.get(i).getCalories()+"\n");
                                        }
                                    }
                                }
                                whereTo.writeChar('\n');
                            }
                        }
                    } catch (ParserConfigurationException e)
                    {
                        logger.error(e.getMessage());
                        e.printStackTrace();
```

```
} catch (SAXException e)
{
	logger.error(e.getMessage());
	e.printStackTrace();
} catch (IOException e)
{
	logger.error(e.getMessage());
	e.printStackTrace();
}

finally
{
	try
	{
		if(fos != null)
			fos.close();
		if(whereTo != null)
			whereTo.close();
	} catch (IOException e)
	{
		logger.error(e.getMessage());
		e.printStackTrace();
	}
}

//set the food menu (table) values
logger.info("Setting food menu values...");
DefaultTableModel TModel = createTmodel();
```

```
                foodMenu = new JTable(TModel);
                rowSorter = new TableRowSorter<>(foodMenu.getModel());                  //for text search in the food menu table
                foodMenu.setRowSorter(rowSorter);

                foodMenu.setBorder(BorderFactory.createLineBorder(Color.black));
            foodMenu.setEnabled(false);

                //center menu text
                DefaultTableCellRenderer centerRenderer = new DefaultTableCellRenderer();
                centerRenderer.setHorizontalAlignment( JLabel.CENTER );
                 for(int x = 1; x < 3 ; x++)
               foodMenu.getColumnModel().getColumn(x).setCellRenderer( centerRenderer );

        }

        public void createRows (DefaultTableModel x){
                for(int i = 0; i < food.size(); ++i)
                        x.addRow(new Object[]{food.get(i).getName(), food.get(i).getDescr(), food.get(i).getPrice(), food.get(i).getCalories() });
        }

        public void removeFood(int foodCount) // removes and amount of food items from the list
        {
                int index = food.size()-1;
                while(foodCount > 0)
                {
                        food.remove(index);
```

```
                index--;
                foodCount--;
            }
        }

        //setters and getters
        public ArrayList<Food> getFoodList() { return food; }
        public JTable getTable() { return foodMenu; }
        public TableRowSorter getRowSorter() { return rowSorter; }
        public void addValue(String Name, String Descr, String Price, String Calories){
            ((DefaultTableModel) foodMenu.getModel()).addRow(new Object[]{Name, Descr,
"$"+Price, Calories});
        }

        @Override
        public String toString() {

            String cosEmek = null;
            for (int i = 0; i < food.size(); ++i){
                cosEmek += "Name: " + food.get(i).getName() + ", Calories: " +
food.get(i).getCalories() + ", Description: "  + food.get(i).getDescr() + ", Price: " + food.get(i).getPrice() +
"\n";                                          ;
            }
            return cosEmek;
        }

}
```