Currency Converter

Mr. Ariel Papa and Mr. Haim Michael	Software Engineering department
Java / Scala Project	Authors: Tom Goldberg (ID : 302815279) [redacted]

1. EXECUTIVE SUMMARY

A currency converter is software code that is designed to convert one currency into another in order to check its corresponding value. The code is generally a part of a web site or it forms a mobile app and it is based on current .market or bank exchange rates

In order to convert one currency into another, a user enters an amount of money (e.g. '1000') and chooses the currency he/she wishes to check the monetary value of (e.g. 'United States Dollar'). After that, the user selects one, or sometimes several other currencies, he/she would like to see the result in. The application software then calculates and displays the corresponding amount of money.

Currency converters aim to maintain real-time information on current market or bank exchange rates, so that the calculated result changes whenever the value of either of the component currencies does. They do so by connecting to a database of current currency exchange rates. The frequency at which currency converters update the exchange rates they use varies: this currency converter updates its rates every day, while Convert My Money< every ten minutes.

TABLE OF CONTENT

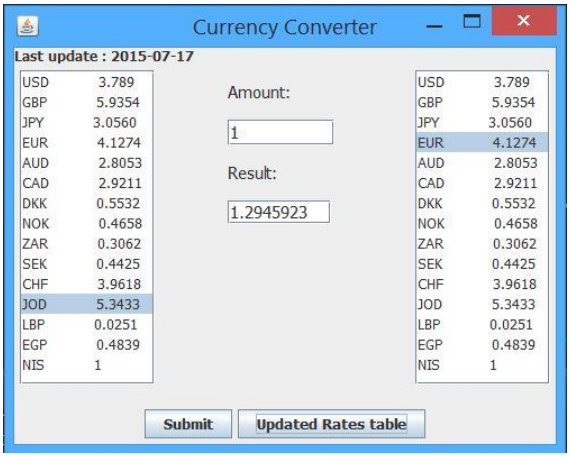
2. INTRODUCTION

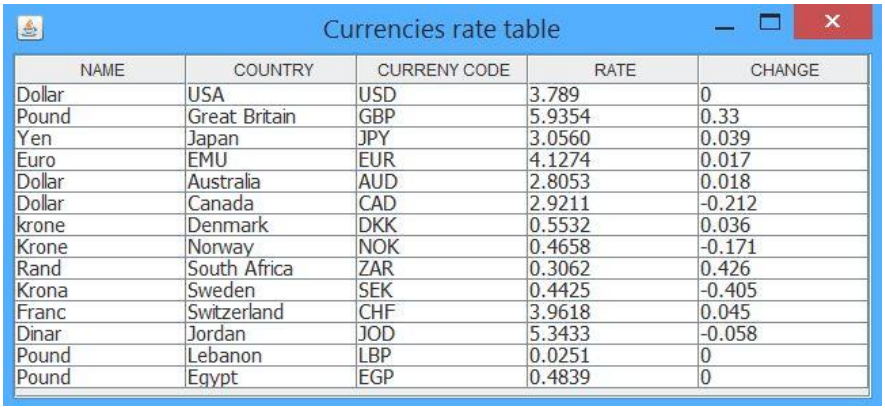
This Currency Converter was designed and developed for the Final Assignment Java course by Tom Goldberg and Lishai Asaraf, two students from the software engineering department at Shenkar College of Engineering and Design.

The application includes:

- ✓ Api documentation
- ✓ Easy-to-use interface
- ✓ Updated currencies rates table (updating every 10 minutes) from the official Bank of Israel
- ✓ QA verification by authorized person

2.1 QUICK VIEW





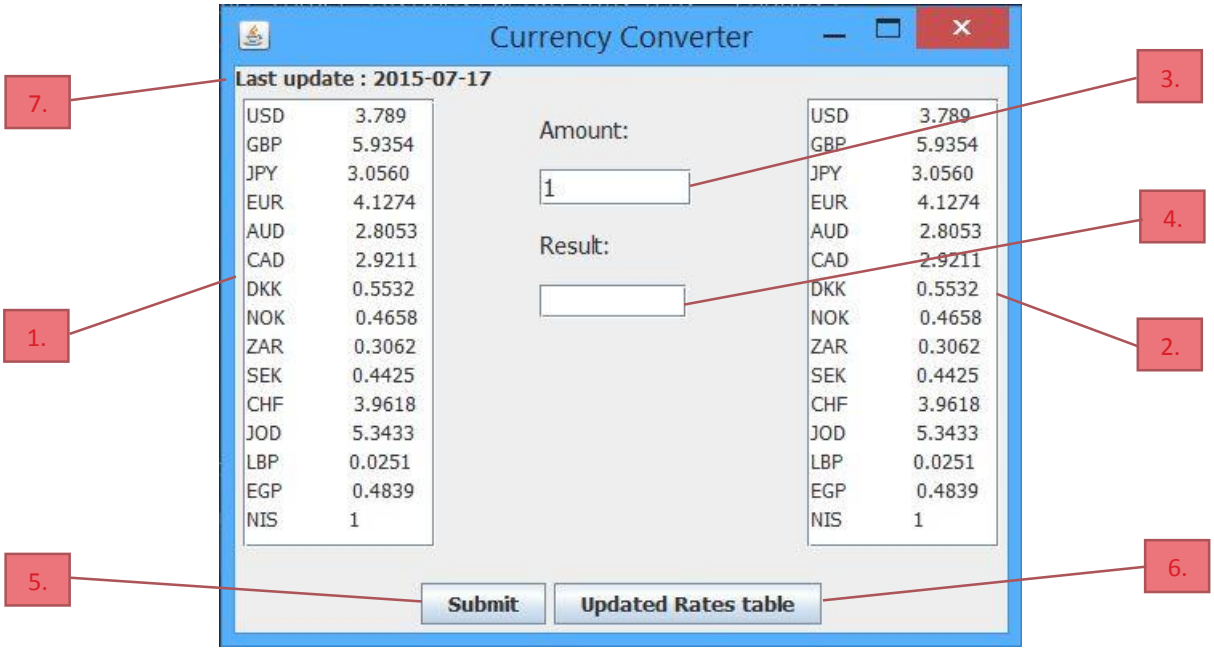
NAME	COUNTRY	CURRENY CODE	RATE	CHANGE
Dollar	USA	USD	3.789	0
Pound	Great Britain	GBP	5.9354	0.33
Yen	Japan	JPY	3.0560	0.039
Euro	EMU	EUR	4.1274	0.017
Dollar	Australia	AUD	2.8053	0.018
Dollar	Canada	CAD	2.9211	-0.212
krone	Denmark	DKK	0.5532	0.036
Krone	Norway	NOK	0.4658	-0.171
Rand	South Africa	ZAR	0.3062	0.426
Krona	Sweden	SEK	0.4425	-0.405
Franc	Switzerland	CHF	3.9618	0.045
Dinar	Jordan	JOD	5.3433	-0.058
Pound	Lebanon	LBP	0.0251	0
Pound	Egypt	EGP	0.4839	0

3. UTILITIES

Tools :

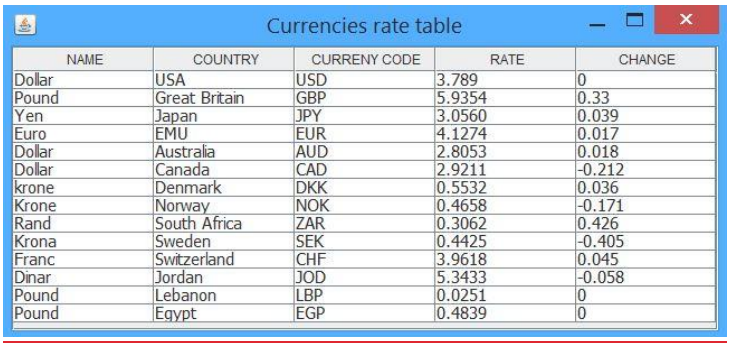
- ✓ IntelliJ IDEA Community Edition 14.1.4
- ✓ Log4J - Java based open source logging framework
- ✓ Module SDK for Scala in IntelliJ Idea (Version 2.11.7)
- ✓ Java Platform Standard Edition
- ✓ Java and Scala api documentation
- ✓ Daily-updated XML doc from the official Israel Bank (http://www.boi.org.il/currency.xml)

4. MAIN FRAME



4.1 INCLUDES

1. FROM LIST Represent the list of currencies to convert from
2. TO LIST Represent the list of currencies to convert to
3. AMOUNT TEXT FIELD Represent the amount of coin to convert
4. RESULT TEXT FIELD Represent the result of the calculation (not editable)
5. SUBMIT BUTTON Will start the calculation
6. UPDATED RATES TABLE BUTTON Represent the up-to-date currencies rate table



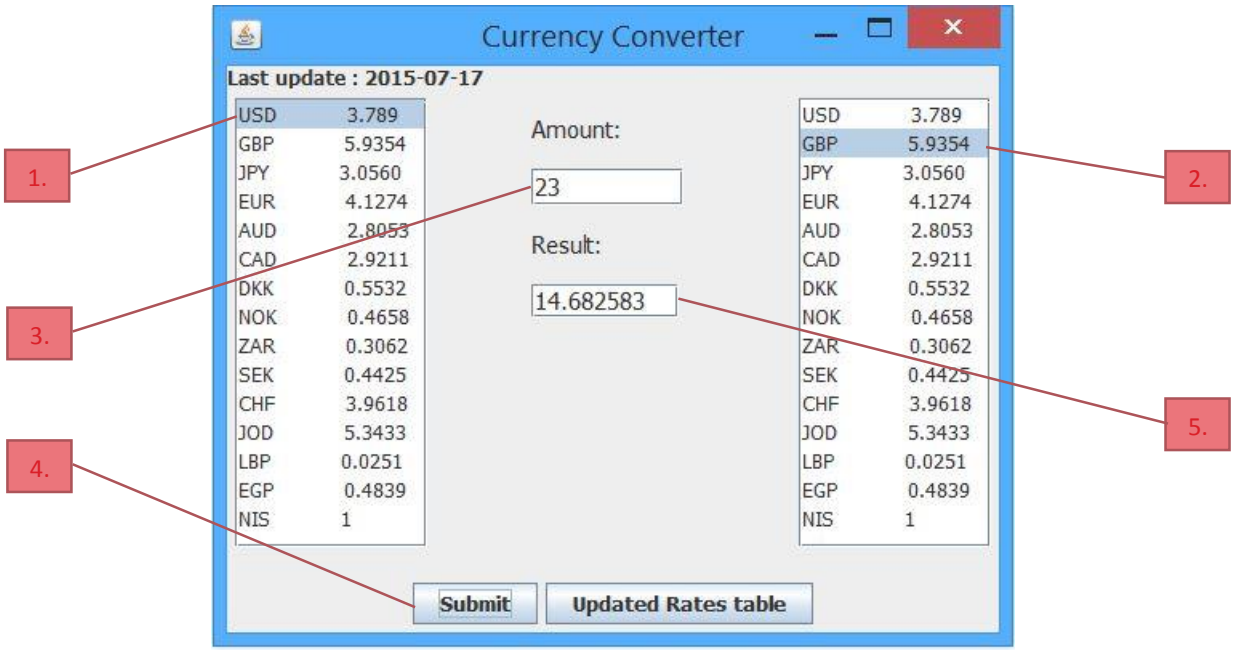
NAME	COUNTRY	CURRENY CODE	RATE	CHANGE
Dollar	USA	USD	3.789	0
Pound	Great Britain	GBP	5.9354	0.33
Yen	Japan	JPY	3.0560	0.039
Euro	EMU	EUR	4.1274	0.017
Dollar	Australia	AUD	2.8053	0.018
Dollar	Canada	CAD	2.9211	-0.212
krone	Denmark	DKK	0.5532	0.036
Krone	Norway	NOK	0.4658	-0.171
Rand	South Africa	ZAR	0.3062	0.426
Krona	Sweden	SEK	0.4425	-0.405
Franc	Switzerland	CHF	3.9618	0.045
Dinar	Jordan	JOD	5.3433	-0.058
Pound	Lebanon	LBP	0.0251	0
Pound	Egypt	EGP	0.4839	0

7. LAST UPDATE LABEL Represent the last update

5. CONVERT

5.1 TO CONVERT PLEASE FOLLOW

1. CHOSE RATE TO COVERT FROM from the "From List" at the left side and click it
2. CHOSE RATE TO COVERT TO from the "To List" at the right side and click it
3. PROVIDE AMOUNT of coins to convert in the "Amount text field"
4. SUBMIT IT with a click on the "Submit" button
5. NEW EXCHANGE RATE will be shown at the "Result text field"



6. FILES

6.1 SCALA CLASS COIN

```
/**
* Class "Coin" represent the main object of this project - Currency coin
* Every constructed object from this has fields and functions
*/
class Coin {
/**
* String casted fields declarations
*/
  private var NAME: String = null
  private var UNIT: String = null
  private var CURRENCYCODE: String = null
  private var COUNTRY: String = null
  private var RATE: String = null
  private var CHANGE: String = null
  /**
   * Constructor of "Coin" using fields
   */
  def this(NAME: String, UNIT: String, CURRENCYCODE: String, COUNTRY: String, RATE: String, CHANGE: String)
{
    this()
    setNAME(NAME)                                  //  set the Name field
    setUNIT(UNIT)                                  //  set the Unit field
    setCURRENCYCODE(CURRENCYCODE)                  //  set the Currency Code field
    setCOUNTRY(COUNTRY)                            //  set the Country field
    setRATE(RATE)                                  //  set the Rate field
    setCHANGE(CHANGE)                              //  set the Change field
    System.out.println("Default C'tor -> " + this.toString)
  }
  /**
  * String casted getters
  */
  def getNAME: String = { return NAME }
  def getCURRENCYCODE: String = { return CURRENCYCODE }
  def getCOUNTRY: String = { return COUNTRY }
  def getRATE: String = { return RATE }
  def getCHANGE: String = { return CHANGE }
  /**
  * String casted setters
  */
  def setNAME(NAME: String) { this.NAME = NAME }
  def setUNIT(UNIT: String) { this.UNIT = UNIT }
  def setCURRENCYCODE(CURRENCYCODE: String) { this.CURRENCYCODE = CURRENCYCODE }
  def setCOUNTRY(COUNTRY: String) { this.COUNTRY = COUNTRY }
  def setRATE(RATE: String) { this.RATE = RATE }
  def setCHANGE(CHANGE: String) { this.CHANGE = CHANGE }

  /**
  * String casted toString function ; This prints the details of the object from class "Coin"
  */
  override def toString: String = {
    return "Coin: {" + "NAME=" + NAME + ",\t\t UNIT=" + UNIT + ",\t\t CURRENCYCODE=" + CURRENCYCODE +
      ",\t\t COUNTRY=" + COUNTRY + ",\t\t RATE=" + RATE + ",\t\t CHANGE=" + CHANGE + "  }"
  }
}
```

6.2 SCALA CLASS MAIN

```
/**
* Object Main is the main thread
*/
object Main {
  def main(args: Array[String]) {
    val xml: ParsingXML = new ParsingXML                    // creating a new ParsingXML object
    val xmlThread: Thread = new Thread(xml)                 // creating new Thread object
    xmlThread.start                                         // start(run) on the Thread object
  }
}
```

6.3 JAVA CLASS CLIENTGUI

```
/*
* imports dictionaries
*/
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;


/**
* Class "ClientGUI" represent the model interface (view) and makes calculations as well
*/
public class ClientGUI {
/**
* fields declarations for class "ClientGUI"
*/
    private static Logger = Logger.getLogger(ClientGUI.class.getName());            // log4j declaration
    private JFrame mainFrame, tableFrame;
    private JButton btSubmit, btTable;
    private JPanel panelFrom, panelTo, panelResult, panelBt, panelSum;
    private JList fromList , toList;
    private JTextArea fromTextArea, toTextArea ;
    private JTextField textFieldResult, amountTextField;
    private JLabel result,lastUpdate, sum;
    private ActionListener listener;                                        // ActionListener declaration
    private JTable table;                                                           // JTable declaration
    /**
    * "Start" Causes this thread to begin execution; the Java Virtual Machine calls the run method of this
thread. The result is that two threads are running concurrently: the current thread (which returns from the
call to the start method)
    *  and the other thread (which executes its run method).
    * It is never legal to start a thread more than once. In particular, a thread may not be restarted once it
has completed execution
    * This actually adds components into the interface with specifics properties which were selected by the
authors
     */
    public void start() throws IllegalThreadStateException{
        PropertyConfigurator.configure("temp/log4j.properties");                // Configure log4j from a file
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);// Determine program when close button pressed
        fromList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);// Enable to choose only one rate from
                                                                                fromList
        JScrollPane JSfromlist = new JScrollPane(fromList);                     // Makes the fromList visible
        JSfromlist.setPreferredSize(new Dimension(120, 280));// Sets the dimensions of white bix which contains
                                                                                fromList
        toList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);// Enable to choose only one rate from
                                                                                toList
        JScrollPane JStolist = new JScrollPane(toList);                         // Makes the toList visible
        JStolist.setPreferredSize(new Dimension(120, 280));   // Sets the dimensions of white bix which
contains toList
```

```
        mainFrame.setLayout(new BorderLayout());                                // Set manager Layout for mainFrame
        panelBt.add(btSubmit);                                                    // Add the "Submit" button
        panelBt.add(btTable);                                                  // Add the "Updated rates table"
        panelSum.add(sum, BorderLayout.NORTH);                                    // Add the JLabel "Amount:"
        panelSum.add(amountTextField, BorderLayout.NORTH);                         // Add the amountTextField
        panelSum.add(result, BorderLayout.SOUTH);                                   // Add the result JLabel
        panelSum.add(textFieldResult, BorderLayout.SOUTH);                  // Add the textFieldResult TextField
        panelFrom.setLayout(new FlowLayout());                           // Set default manager Layout for panelFrom
        panelTo.setLayout(new FlowLayout());                              // Set default manager Layout for panelTo
        panelTo.add(JStolist);                                             // Add white box to attach the fromList
        panelFrom.add(JSfromlist);                                          // Add white box to attach the toList
        amountTextField.setText("1");                                  // Sets the default value to exchange to "1"
        amountTextField.setFont(new java.awt.Font("Tahoma", 0, 15));                    //      with new design
        mainFrame.add(panelTo, BorderLayout.EAST);                          // Sets toList panel at EAST (right)
        mainFrame.add(panelFrom,BorderLayout.WEST);                        // Sets fromList panel at WEST (left)
        mainFrame.add(panelBt,BorderLayout.SOUTH);                        // Sets buttons panel at SOUTH (bottom)
        mainFrame.add(panelSum, BorderLayout.CENTER);              // Sets main panel at CENTER (middle interface)
        mainFrame.add(lastUpdate,BorderLayout.NORTH);                      // Sets lastUpdate label at NORTH (up)
        mainFrame.setSize(500, 400);                                        // Sets dimensions to the main window
        mainFrame.setVisible(true);                                                  // Sets interface visible
        btSubmit.addActionListener(listener);                        // Activating ActionListener on "Submit" button
        btTable.addActionListener(listener);          // Activating ActionListener on "Updated rates table" button
    }
    /**
    * Constructor using fields
    */
    public ClientGUI  (final Coin[] coinsArrayVal, String LastUpdate){
        /*
        * New fields objects
        */
        DefaultListModel model = new DefaultListModel();
        mainFrame = new JFrame("Currency Converter");                                                  // Main window
        btSubmit = new JButton("Submit");                                                             // Submit button
        btSubmit.setFont(new java.awt.Font("Tahoma", 1, 13));                                 //    with font design
        btTable = new JButton("Updated Rates table");                                         // Updated Rates table
        btTable.setFont(new java.awt.Font("Tahoma", 1, 13));                                  //    with font design
        lastUpdate = new JLabel("Last update : " + LastUpdate);                               // "Last update" text
        lastUpdate.setFont(new java.awt.Font("Tahoma", 1, 12));                                 //     - new design
        sum = new JLabel("Amount:");                                                              // "Amount:" text
        sum.setFont(new java.awt.Font("Tahoma", 0, 15));                                        //     - new design
        result = new JLabel("Result:  ");                                                         // "Result:" text
        result.setFont(new java.awt.Font("Tahoma", 0, 15));                                     //     - new design
        panelFrom = new JPanel();                                                              // JPanel new object
        panelTo = new JPanel();
        panelResult = new JPanel();
        panelBt = new JPanel();
        panelSum = new JPanel(new FlowLayout(0, 60, 15));                  // JPanel with FlowLayout custom manager
        fromList = new JList(model);                                          // JList of convert from currencies
        fromList.setFont(new java.awt.Font("Tahoma", 0, 13));                                 //    with new AMAZING font
        toList = new JList(model);                                          // JList of convert to currencies
        toList.setFont(new java.awt.Font("Tahoma", 0, 13));                                   //    with new AMAZING font
        fromTextArea = new JTextArea();                                                    // JTextArea new objects
        toTextArea = new JTextArea();
        amountTextField = new JTextField(7);                                          // JTextField new sized objects
        textFieldResult = new JTextField(8);
        for (int i = 0; i < 14 ; i++) {                                       // Adding elements which represent coins
            try { model.addElement(coinsArrayVal[i].getCURRENCYCODE() + "      "// into model type using a loop
                    + coinsArrayVal[i].getRATE());
            } catch (NullPointerException e){
                    e.printStackTrace();
                    System.out.println("Null Pointer Exception about connectivity -> (" +
                                e.getMessage() + ")");
            }
        }
        model.addElement("NIS" + "           " + 1);                    // Adding element which represent Shekel
```

```
        /*
        * Defining a new listener object which represent "ActionListener" function
        * The listener interface for receiving action events. The class that is interested in processing an
action event implements this interface,
        * and the object created with that class is registered with a component, using the component's
"addActionListener" method.
        * When the action event occurs, that object's "actionPerformed" method is invoked.
        */
        listener = new ActionListener() {
        /*
        * Overrides "actionPerformed" Invoke when an action occurs.
        * This implementation suggest for action to occur and dealing with it as specified
        *   the program will wait until the user clicks on "Submit" or "Updated Rates table"
        */
            @Override
            public void actionPerformed(ActionEvent evt) {
                Object src = evt.getSource();                    // Declaration of source event handling object
                float rateFrom, rateTo;                          // Declaration of floats which takes the rates
                if (src == btSubmit) try {                                  // If "Submit" button was pressed
                    try { rateFrom = Float.parseFloat(coinsArrayVal[fromList.getSelectedIndex()].getRATE()); }
                    catch (ArrayIndexOutOfBoundsException ex) {
                        ex.printStackTrace();
                        rateFrom = (float) 1.0;
                        JFrame frame = new JFrame();
                        JOptionPane.showMessageDialog(frame, "No currency from convert was chosen\n" +
                                "Or NIS was chosen\n" + "Turning default currency -> Shekel");
                        logger.info("Turning default currency to convert from Shekel");
                    }
                    try { rateTo = Float.parseFloat(coinsArrayVal[toList.getSelectedIndex()].getRATE()); }
                    catch (ArrayIndexOutOfBoundsException e) {
                        e.printStackTrace();
                        rateTo = (float) 1.0;
                        JFrame frame = new JFrame();
                        JOptionPane.showMessageDialog(frame, "No currency to convert was chosen\n" +
                                "Or NIS was chosen\n" + "Turning default currency -> Shekel");
                        logger.info("Turning default currency to convert to Shekel");
                    }
                    float sum =Float.parseFloat(amountTextField.getText());// Declaration of float object "sum"
                    float result = sum * rateFrom / rateTo;// The Calculation result into float object "result"
                    textFieldResult.setText(String.valueOf(result));   // Putting result into Result JTextField
                    textFieldResult.setFont(new java.awt.Font("Tahoma", 0, 15));   //      with new font design
                    logger.info("Exchange button pressed.The exchange is:"+result); // log4j : new INFO message
                } catch (NumberFormatException ex) {                                        // Exception :
                    JFrame frame = new JFrame();                                  // New JFrame (new window)
                    JOptionPane.showMessageDialog(frame, "Please insert valid decimal amount\n" +
ex.getMessage() + " is not valid");                                    // Error message
                    logger.error("Wrong input -> (" + ex.getMessage() + ")");      // log4j : new ERROR message
                    amountTextField.setText("");                                 // Clearing Amount JTextField
                } catch (ArrayIndexOutOfBoundsException ex) {                               // Exception :
                    JFrame frame = new JFrame();                                  // new JFrame(new window)
                    JOptionPane.showMessageDialog("Please choose currencies to convert");      // Error message
                    logger.error("Wrong input(The currencies not been chosen)");   // log4j : new ERROR message
                    amountTextField.setText("");                                 // Clearing Amount JTextField
                }
                if (src == btTable) {                            // If "Updated Rates table" button was pressed
                    tableFrame=new JFrame("Currencies rate table");// New JFrame object referenced to tableFram
                    tableFrame.setLayout(new BorderLayout());                     // Default layout manager
                    tableFrame.setSize(650, 300);                                  // Dimensions of JFrame
                    Object[] columnNames = {"NAME","COUNTRY","CURRENY CODE","RATE","CHANGE"};  // Columns Names
                    Object[][] data = new Object[14][5];  // Inserting data from coinsArray into matrix of data
                    for (int row = 0; row <= 13; row++){      //    which will be shown as a JTable with a loop
                        for (int col = 0; col <= 4; col++) {
                            switch (col) {
                                case 0:data[row][col] = coinsArrayVal[row].getNAME(); break;
                                case 1:data[row][col] = coinsArrayVal[row].getCOUNTRY();break;
                                case 2:data[row][col] = coinsArrayVal[row].getCURRENCYCODE();break;
                                case 3:data[row][col] = coinsArrayVal[row].getRATE();break;
                                case 4:data[row][col] = coinsArrayVal[row].getCHANGE();break;
                                }
                        }
                    }
                    table = new JTable(data, columnNames); // New JTable object with data and the columns names
                    table.setFont(new java.awt.Font("Tahoma", 0, 15));              //      with new design
                    tableFrame.add(new JScrollPane(table));                         //
                    tableFrame.setVisible(true);                          // Set the JTable Frame visible (true)
                    }
                }
        }; } }
```

6.4 SCALA CLASS PARSINGXML

```
/*
* imports dictionaries
*/
import javax.xml.transform.TransformerException
import org.apache.log4j.{PropertyConfigurator, Logger}
import org.w3c.dom.Document
import org.w3c.dom.NodeList
import org.xml.sax.SAXException
import javax.xml.parsers.DocumentBuilder
import javax.xml.parsers.DocumentBuilderFactory
import javax.xml.parsers.ParserConfigurationException
import java.io.{File, PrintWriter, IOException, InputStream}
import java.net.{MalformedURLException, HttpURLConnection, URL, UnknownHostException}


/**
* class ParsingXML parses the content from the site of the bank of Israel
 * into NodeLists which contains any single row from the xml and categorized
 * This runs as a Runnable object
*/
class ParsingXML extends Runnable{                                          // declaration of the class
  private val logger: Logger = Logger.getLogger(classOf[ParsingXML].getName)      // log4j declaration
  PropertyConfigurator.configure("temp/log4j.properties")                   // log4j new configuration
  private var nameList: NodeList = null                                          //
  private var unitList: NodeList = null                                          //
  private var currencyCodeList: NodeList = null                                  //
  private var countryList: NodeList = null                                    // declarations of fields
  private var rateList: NodeList = null                                       // of class ParsingXML
  private var changeList: NodeList = null                                        //
  private var lastUpdate: NodeList = null                                        //
  private var lastUpdateTime: String = null                                      //
  private var file : Model = null                                                //
  private var is: InputStream = null                                             //
  private var con: HttpURLConnection = null                                      //
  private var i = 0
  /**
   * If this thread was constructed using a separate Runnable run object, then that Runnable object's run
method is called;
   * otherwise, this method does nothing and returns.
   * Subclasses of Thread should override this method.
   */
  def run {                                                                      // run object
    var coinArray: Array[Coin] = null                                            // Array of Coins
    while(true){                                                                 // While(true) loop
    try {
      file = new Model("Currency.txt")                                           // Makes a new stream
      val url: URL = new URL("http://www.boi.org.il/currency.xml")
      con = url.openConnection.asInstanceOf[HttpURLConnection]
      con.setRequestMethod("GET")
      con.connect
      is = con.getInputStream
      val factory: DocumentBuilderFactory = DocumentBuilderFactory.newInstance
      val builder: DocumentBuilder = factory.newDocumentBuilder
      val doc: Document = builder.parse(is)
      nameList = doc.getElementsByTagName("NAME")                          // Parsing data into nodelists
      unitList = doc.getElementsByTagName("UNIT")
      currencyCodeList = doc.getElementsByTagName("CURRENCYCODE")
      countryList = doc.getElementsByTagName("COUNTRY")
      rateList = doc.getElementsByTagName("RATE")
      changeList = doc.getElementsByTagName("CHANGE")
      lastUpdate = doc.getElementsByTagName("LAST_UPDATE")                       // Inserting the lastUpdate
      file.writeToFile(nameList, unitList, currencyCodeList,                 // Writing the nodelists to File
                       countryList, rateList, changeList, lastUpdate)
      coinArray = file.readFromFile                                       // Read from the File into array of coins
      lastUpdateTime = file.readLastUpdate                                   // Read from the File the last update
      System.out.println("The last update is: " + lastUpdateTime)
    }
    catch {                                                                      // Exceptions cases :
      case e: (UnknownHostException) => {                                     // ? Unable to connect
        System.out.println("Unable to connect the server. taking data from the last update.")
        coinArray = file.readFromFile                                        // ! Takes data from the last update
        lastUpdateTime = file.readLastUpdate+"  (Offline mode)"
        logger.error("Problem with parsing from net -> Offline mode")              // log4j: new ERROR message
      }
      case e : (TransformerException) => {                                      // ? Unable to transform
```

```
        e.printStackTrace
        println("Something went wrong-> ("+e.getMessage+")")
        coinArray = file.readFromFile                                    // ! Takes data from the last update
        lastUpdateTime = file.readLastUpdate+"  (Offline mode)"
        logger.error("Problem with parsing from net -> Offline mode")          // log4j: new ERROR message
      }
      case e : (SAXException) => {                                                // ? Module problems
        e.printStackTrace
        println("Something went wrong-> ("+e.getMessage+")")
        coinArray = file.readFromFile                                    // ! Takes data from the last update
        lastUpdateTime = file.readLastUpdate+"  (Offline mode)"
        logger.error("Problem with parsing from net -> Offline mode")          // log4j: new ERROR message
      }
      case e : (IOException) => {                                          // ? Files I/O streams problem
        e.printStackTrace
        println("Something went wrong-> ("+e.getMessage+")")
        coinArray = file.readFromFile                                    // ! Takes data from the last update
        lastUpdateTime = file.readLastUpdate+"  (Offline mode)"
        logger.error("Problem with parsing from net -> Offline mode")          // log4j: new ERROR message
      }
      case e : (ParserConfigurationException) => {                                // ? Parser Problems
        e.printStackTrace
        println("Something went wrong-> ("+e.getMessage+")")
        coinArray = file.readFromFile                                    // ! Takes data from the last update
        lastUpdateTime = file.readLastUpdate+"  (Offline mode)"
        logger.error("Problem with parsing from net -> Offline mode")          // log4j: new ERROR message
      }
      case e : (MalformedURLException) => {                                 // ? URL connectivity problem
        e.printStackTrace
        println("Something went wrong-> ("+e.getMessage+")")
        coinArray = file.readFromFile                                    // ! Takes data from the last update
        lastUpdateTime = file.readLastUpdate+"  (Offline mode)"
        logger.error("Problem with parsing from net -> Offline mode")          // log4j: new ERROR message
      }
      case e : (NullPointerException) => {                                  // ? NullPointerException
        e.printStackTrace
        println("Something went wrong-> ("+e.getMessage+")")
        coinArray = file.readFromFile                                    // ! Takes data from the last update
        lastUpdateTime = file.readLastUpdate+"  (Offline mode)"
        logger.error("Problem with parsing from net -> Offline mode")          // log4j: new ERROR message
      }
    }
    finally {                                                                 // Finally always occurs
      /*
      * This while loop iterates once in order to prevent opening several interfaces at the same time
       */
      while (i!=1){
      val gui : ClientGUI = new ClientGUI(coinArray, lastUpdateTime)            // New ClientGUI object
      gui.start                                                              // Start the GUI operations
      i+=1
      if (is != null) { try { is.close }                                      // Close parser condition
        catch { case e: IOException =>  e.printStackTrace  }
      }
      if (con != null) { try {con.disconnect}                                // Close connection condition
        catch { case e : UnknownHostException=>e.printStackTrace}
      }}
      println("After the update")                                           // Indicates data was updated
      Thread.sleep(600000)                                                 // Wait 10 minutes till the next loop
      println("Updating...")                                                // Indicates data is being updating
    }
  }
  }
}
```

6.5 SCALA TRAIT IMODEL

```
/*
* imports dictionaries
*/
import org.w3c.dom.NodeList
/**
* Similar to interfaces in Java, traits are used to define object types by specifying the signature of the
supported methods.
* Unlike Java, Scala allows traits to be partially implemented; i.e. it is possible to define default
implementations for some methods.
* In contrast to classes, traits may not have constructor parameters.
*
* This IModel is the main trait which declaring the methods that implemented in the other files
*/
trait IModel {                                                                    // IModel trait
  /**
   * writeToFile method receives 7 node lists and write the parsed data into on single file
   * This is implemented in the Model class
   */
  def writeToFile(list1: NodeList,                                          // writeToFile declaration
                  list2: NodeList,
                  list3: NodeList,
                  list4: NodeList,
                  list5: NodeList,
                  list6: NodeList,
                  list7: NodeList)
  /**
   * readFromFile method reads any single line within a known file document
   */
  def readFromFile: AnyRef                                                  // readFromFile declaration
}
```

6.6 SCALA CLASS MODEL

```
/*
* imports dictionaries
*/
import org.w3c.dom.NodeList
import java.io._
/**
* This Model is the heart of the project.
* It implements three main functions which has been declared int the trait : readFromFile , writeToFile ,
readLastUpdate
 */
class Model extends IModel {
  private val coinsArray: Array[Coin] = new Array[Coin](14)                    // Array of 14 Coins
  private var lineCounter: Int = -1                                           // Declaration of lineCounter
  private var name: String = null                                              // Declaration of file name
  def this(nameVal: String) { this()
    name = nameVal }
/**
/ readFromFile function reads from a known file and parses the data within String which declared below
 */
  def readFromFile: Array[Coin] = {
    var fstream1 : FileReader = null                                        // Declarations of File readers
    var fstream2 : FileReader = null
    var in1: BufferedReader = null                                        // Declarations of File Buffer readers
    var in2: BufferedReader = null
    var coinName: String = null                                        // Declarations of String which contains
    var coinCode: String = null                                                //   the parsed data
    var coinCountry: String = null
    var coinUnit: String = null
    var coinRate: String = null
    var coinChange: String = null
    val line: String = null
    try {
      fstream1 = new FileReader(name)
      in1 = new BufferedReader(fstream1)
      while (in1.readLine != null) ({ lineCounter += 1; lineCounter - 1})           // Counting file lines
      fstream2 = new FileReader(name)
      in2 = new BufferedReader(fstream2)
      for (j<-0 to ((lineCounter - 1) / 6)) {                                    // Parsing with a loop
          { coinName = in2.readLine
            coinCode = in2.readLine
            coinCountry = in2.readLine
            coinUnit = in2.readLine
            coinRate = in2.readLine
            coinChange = in2.readLine
            coinsArray(j) = new Coin(coinName, coinCode, coinCountry,               // Making new Coin objects
                                     coinUnit, coinRate, coinChange)
          }
        }
    }
    catch { case e: FileNotFoundException => e.printStackTrace                  // Catch exceptions if needed
            case e: IOException => e.printStackTrace
    } finally { if (in1 != null) { try { in1.close }
                                   catch { case e: IOException => e.printStackTrace
        }
      }
    }
    return coinsArray                                                          // Return the array of Coins
  }
  /**
  * writeToFile function writes to a known file the data that has been parsed from the web
  * into different lines each time.
  */
  def writeToFile(nameList: NodeList, unitList: NodeList, currencyCodeList: NodeList, countryList: NodeList,
rateList: NodeList, changeList: NodeList, lastUpdate: NodeList) {
    var out: BufferedWriter = null                                            // Declaration of BufferedWriter
    try {
      val fstream: FileWriter = new FileWriter(name, false)       // New FileWriter (true means to append data)
      out = new BufferedWriter(fstream)                                       // New BufferedWriter object
      var i: Int = 0
        i = 0
        while (i < nameList.getLength) {              // Loop that writes to file any data that has been parsed
          { out.write(nameList.item(i).getFirstChild.getNodeValue)                 // in separate lines
            out.newLine
            out.write(unitList.item(i).getFirstChild.getNodeValue)
            out.newLine
            out.write(currencyCodeList.item(i).getFirstChild.getNodeValue)
```

```
            out.newLine
            out.write(countryList.item(i).getFirstChild.getNodeValue)
            out.newLine
            out.write(rateList.item(i).getFirstChild.getNodeValue)
            out.newLine
            out.write(changeList.item(i).getFirstChild.getNodeValue)
            out.newLine
          }
          ({ i += 1; i - 1})
        }
      out.write(lastUpdate.item(0).getFirstChild.getNodeValue)                // Writes the last update to file
      out.newLine
    }
    catch { case e: IOException => {e.printStackTrace                                  // Catch exceptions of needed
                                    System.err.println("Error: " + e.getMessage)}
    } finally { if (out != null) { try { out.close }
                                   catch { case e: IOException => e.printStackTrace
        }
      }
    }
  }
  /**
  * readLastUpdate function reads the last line of a known file and insert it into var
  */
  @throws(classOf[IOException])
  def readLastUpdate: String = {
    var fstream: FileReader = null                                                 // Declaration of file reader
    var in: BufferedReader = null                                                // Declaration of file buffer reader
    var line: String = null                                                        // Declaration of a String
    try {
      fstream = new FileReader(name)
      in = new BufferedReader(fstream)                                           // New BufferedWriter object
      val temp: Int = 0
      var counter1: Int = -1
      while (counter1 <= lineCounter) {
        if (counter1 == lineCounter) { return line }
        ({counter1 += 1; counter1 - 1})
        line = in.readLine
        }
    } catch { case e: FileNotFoundException => e.printStackTrace
    } finally { if (in != null) try { in.close }
                                catch { case e: IOException => e.printStackTrace
      }
    }
    return null                                                                    // if not succeed return null
  }
}
```