Currency.java

```java
/**
 * The currency enum defines all currencies CODES
 * and DESCRIPTION
 *
 * @author      Netta Richer
 * @author      Sagi Granot
 */
package sagi.neta.CurrencyExchanger;

enum Currency {
    USD("United States Dollar"),
    GBP("Great Britain Pound"),
    JPY("Yen - Japan"),
    EUR("Euro"),
    AUD("Dollar Australia"),
    CAD("Dollar Canada"),
    DKK("Krone Denmark"),
    NOK("Krone Norway"),
    ZAR("Rand South Africa"),
    SEK("Krona Sweden"),
    CHF("Franc Switzerland"),
    JOD("Dinar Jordan"),
    LBP("Pound Lebanon"),
    EGP("Pound Egypt");

    private String description;

    Currency(String description) {
        this.description = description;
    }

    @Override public String toString() {
        return this.name() + " - " + this.description;
    }
}
```

CurrencyPair.java

```java
/**
 * The CurrencyPair class defines a pair of currencies
 * which represent a FROM currency and a TO currency
 *
 * @author      Netta Richer
 * @author      Sagi Granot
 */
package sagi.neta.CurrencyExchanger;

public class CurrencyPair{
    private final Currency from;
    private final Currency to;

    public Currency getFrom() {
        return from;
    }

    public Currency getTo() {
        return to;
    }

    public CurrencyPair(Currency from, Currency to) {
        this.from = from;
        this.to = to;
    }

    @Override public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        CurrencyPair that = (CurrencyPair) o;
        if (from != that.from) return false;
        return to == that.to;
    }

    @Override public int hashCode() {
        int result = from.hashCode();
        result = 31 * result + to.hashCode();
        return result;
    }
}
```

CurrencyException.java

```java
/**
 * The CurrencyException class defines an exception
 * with a message
 *
 * @author      Netta Richer
 * @author      Sagi Granot
 */
package sagi.neta.CurrencyExchanger;

public class CurrencyException extends Exception {
    CurrencyException(String msg, Throwable e){
        super(msg,e);
    }
}
```

Model.java

```java
/**
 * The model interface is responsible for updating an hashmap
 * with all currency exchange rates.
 *
 * @author      Netta Richer
 * @author      Sagi Granot
 */
package sagi.neta.CurrencyExchanger;
import java.util.Map;
public interface Model {
    /**
     * Get a reference to the hashmap
     *
     * @return          <code>Map<CurrencyPair, Double></code> with ratio rates
     *                  between all currencies
     */
    public abstract Map<CurrencyPair, Double> getExchangeRates() throws
CurrencyException;
    /**
     * Getting from the xml file (which was created in a different
     * thread) rates and calculates
     * all rates and updates in an hash map.
     *
     * @see         xmlParser
     */
    public abstract void updateHashMap() throws CurrencyException;
}
```

xmlParser.java

```java
/**
 * The xmlParser class is the Model implementation.
 * it implements Runnable in order to provide an option
 * to run a thread for continuous checking of the up-to-date XML data
 * and updating the data stored local in a file.
 * it has an hashmap with all currency rates. -> (currency)FROM, (currency)TO,
RATE
 *
 *
 * @author      Netta Richer
 * @author      Sagi Granot
 * @see         sagi.neta.CurrencyExchanger.Currency
 * @see         sagi.neta.CurrencyExchanger.CurrencyPair
 */
package sagi.neta.CurrencyExchanger;
import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.xml.sax.*;
import org.w3c.dom.*;
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.Logger;

public class xmlParser implements Runnable, Model{
    private static final String BACKUP = "currency.xml";          //path for local
xml file in case internet fails
    private static final String XML_PATH = "gui.xml";             //path for new
updated xml created locally
    private Map<CurrencyPair, Double> exchangeRates;              //hashmap to
hold all rates
    static Logger logger = Logger.getLogger("xmlParser");         //Logger
    private String Date;                                          //Date of update
    /**
     * Class constructor sets a new hashmap
     */
    public xmlParser() {
        this.exchangeRates = new HashMap<>();
        BasicConfigurator.configure();
        logger.info("xmlParser Constructor init");
    }
    /**
     * Hashmap getter
     * @return the hash map
     */
    public Map<CurrencyPair, Double> getExchangeRates() throws
CurrencyException{
        if (exchangeRates.size() <= 0) {
            logger.info("HAsmap empty ... throwing exception");
            throw new CurrencyException("HashMap is empty!", new Error());
        }
        logger.info("returning hashmap ref");
        return exchangeRates;
    }
    /**
     * Date getter
     * @return the date of last update
     */
    public String getDate(){
```

```
        logger.info("Responding with date");
        return Date;
    }

    /**
     * This method opens a locally saved xml file,
     * and parsing the data into an hashmap
     * the method calculates convertion ratio between
     * every currency to every other currency
     *
     */
    public void updateHashMap() throws CurrencyException{
        NodeList LAST_UPDATE    = null;
        NodeList RATE           = null;
        DocumentBuilderFactory factory;
        DocumentBuilder builder;
        Document doc;

        try {
            logger.info("Reading locally saved updated xml file...");
            factory = DocumentBuilderFactory.newDefaultInstance();
            builder = factory.newDocumentBuilder();
            doc = builder.parse(new InputSource(XML_PATH));
            RATE            = doc.getElementsByTagName("RATE");
            LAST_UPDATE     = doc.getElementsByTagName("LAST_UPDATE");
            Date = LAST_UPDATE.item(0).getFirstChild().getNodeValue();
        } catch (java.net.MalformedURLException e) {
            e.printStackTrace();
            throw new CurrencyException("MalformedURLException",e);
        } catch (java.io.IOException e) {
            e.printStackTrace();
            throw new CurrencyException("IOException",e);
        } catch (javax.xml.parsers.ParserConfigurationException e) {
            e.printStackTrace();
            throw new CurrencyException("ParserConfigurationException",e);
        } catch (org.xml.sax.SAXException e) {
            e.printStackTrace();
            throw new CurrencyException("SAXException",e);
        }
        //
        int i = 0, j;
        for (sagi.neta.CurrencyExchanger.Currency from :
sagi.neta.CurrencyExchanger.Currency.values()) {
            Double toShekels =
Double.parseDouble(RATE.item(i).getFirstChild().getNodeValue());
            j = 0 ;
            logger.info("Calculating rates from " + from + " to all other
currencies...");
            for (sagi.neta.CurrencyExchanger.Currency to :
sagi.neta.CurrencyExchanger.Currency.values()) {
                Double toCurr =
Double.parseDouble(RATE.item(j).getFirstChild().getNodeValue());
                Double newRate = toShekels / toCurr;
                if (from == sagi.neta.CurrencyExchanger.Currency.JPY) {
                    newRate /= 100;
                } else if (from == sagi.neta.CurrencyExchanger.Currency.LBP) {
                    newRate /= 10;
                } else if (to == sagi.neta.CurrencyExchanger.Currency.JPY) {
                    newRate *= 100;
                } else if (to == sagi.neta.CurrencyExchanger.Currency.LBP) {
                    newRate *= 10;
                }
                exchangeRates.put(new CurrencyPair(from, to), newRate);
                ++j;
            }
            ++i;
```

```
        }
        logger.info("Done calculating hashmap with all rates.");

    }
    /**
     * This run method is responsible of fetching new data from
     * bank api, and creating a new identical xml file,
     * and storing that locally.
     *
     */
    @Override
    public void run() {
        InputStream is          = null;
        HttpURLConnection con   = null;
        NodeList LAST_UPDATE    = null;
        NodeList NAME           = null;
        NodeList UNIT           = null;
        NodeList CURRENCYCODE   = null;
        NodeList COUNTRY        = null;
        NodeList RATE           = null;
        NodeList CHANGE         = null;
        URL url;
        DocumentBuilderFactory factory;
        DocumentBuilder builder;
        Document doc = null;
        //Fetch XML from server
        try {
            logger.info("trying to fetch xml from server...");
            url = new URL("https://www.boi.org.il/currency.xml");
            con = (HttpURLConnection) url.openConnection();
            con.setRequestMethod("GET");
            con.connect();
            is = con.getInputStream();
            factory = DocumentBuilderFactory.newDefaultInstance();
            builder = factory.newDocumentBuilder();
            doc = builder.parse(is);

        } catch (java.net.MalformedURLException e) {
            //e.printStackTrace();
        } catch (java.io.IOException e) {
//            If could not GET xml file, open it locally.
            try {
                logger.info("fetch from server failed. opening xml locally");
                File fXmlFile = new File(BACKUP);
                DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
                DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
                doc = dBuilder.parse(fXmlFile);
            }catch(javax.xml.parsers.ParserConfigurationException ex){
                ex.printStackTrace();
            }
            catch (org.xml.sax.SAXException ex){ ex.printStackTrace();}
            catch (IOException ex){ ex.printStackTrace();}
        }
        catch (javax.xml.parsers.ParserConfigurationException e) {
          //  e.printStackTrace();
        }
        catch (org.xml.sax.SAXException e) {
          //  e.printStackTrace();
        }
        NAME            = doc.getElementsByTagName("NAME");
        UNIT            = doc.getElementsByTagName("UNIT");
        CURRENCYCODE    = doc.getElementsByTagName("CURRENCYCODE");
        COUNTRY         = doc.getElementsByTagName("COUNTRY");
        RATE            = doc.getElementsByTagName("RATE");
        CHANGE          = doc.getElementsByTagName("CHANGE");
```

```
        LAST_UPDATE     = doc.getElementsByTagName("LAST_UPDATE");
        //Create new XML for local storage
        Document dom        = null;
        Element e           = null;
        Element currencyEle = null;
        String _LAST_UPDATE = null;
        String _NAME        = null;
        String _UNIT        = null;
        String _CURRENCYCODE = null;
        String _COUNTRY     = null;
        String _RATE        = null;
        String _CHANGE      = null;
        Element rootEle     = null;
        //
        // instance of a DocumentBuilderFactory
        logger.info("Building a new xml copy");
        try {
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            // use factory to get an instance of document builder
            DocumentBuilder db = dbf.newDocumentBuilder();
            // create instance of DOM
            dom = db.newDocument();
            // create the root element
            rootEle = dom.createElement("CURRENCIES");
            // create data elements and place them under root
            e = dom.createElement("LAST_UPDATE");
            _LAST_UPDATE = LAST_UPDATE.item(0).getFirstChild().getNodeValue();
            e.appendChild(dom.createTextNode(_LAST_UPDATE));
            rootEle.appendChild(e);
        }catch(ParserConfigurationException ex){ex.printStackTrace();}
        //
        //Create all currencies
        int i = 0;
        for (sagi.neta.CurrencyExchanger.Currency from : Currency.values()) {
            // create node currency
            currencyEle = dom.createElement("CURRENCY");
            rootEle.appendChild(currencyEle);
            //
            // add children to node currency
            e = dom.createElement("NAME");
            _NAME = NAME.item(i).getFirstChild().getNodeValue();
            e.appendChild(dom.createTextNode(_NAME));
            currencyEle.appendChild(e);

            e = dom.createElement("UNIT");
            _UNIT = UNIT.item(i).getFirstChild().getNodeValue();
            e.appendChild(dom.createTextNode(_UNIT));
            currencyEle.appendChild(e);

            e = dom.createElement("CURRENCYCODE");
            _CURRENCYCODE = CURRENCYCODE.item(i).getFirstChild().getNodeValue();
            e.appendChild(dom.createTextNode(_CURRENCYCODE));
            currencyEle.appendChild(e);

            e = dom.createElement("COUNTRY");
            _COUNTRY = COUNTRY.item(i).getFirstChild().getNodeValue();
            e.appendChild(dom.createTextNode(_COUNTRY));
            currencyEle.appendChild(e);

            e = dom.createElement("RATE");
            _RATE = RATE.item(i).getFirstChild().getNodeValue();
            e.appendChild(dom.createTextNode(_RATE));
            currencyEle.appendChild(e);

            e = dom.createElement("CHANGE");
            _CHANGE = CHANGE.item(i).getFirstChild().getNodeValue();
```

```
            e.appendChild(dom.createTextNode(_CHANGE));
            currencyEle.appendChild(e);
            ++i;
        }
        dom.appendChild(rootEle);
        // write the content into xml file
        try {
            TransformerFactory transformerFactory =
TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            DOMSource source = new DOMSource(dom);
            StreamResult result = new StreamResult(new File("gui.xml"));
            transformer.transform(source, result);
            logger.info("updated xml file saved!");
        } catch (TransformerException te) {
            logger.info("Failed to save file");
            logger.info(te.getMessage());
        }

    }
}
```

Client.java

```java
/**
 * Client is the UI class that creates GUI elements using Swing
 * providing the user some input options
 * <ul>
 * <li>The amount of money to convert
 * <li>From currency
 * <li>To currency
 * <li>Convert option
 * <li>Show all rates option
 * </ul>
 * <p>
 * @see         sagi.neta.CurrencyExchanger.Model
 * @see         sagi.neta.CurrencyExchanger.xmlParser
 * @see         sagi.neta.CurrencyExchanger.Currency
 * @author      Netta Richer
 * @author      Sagi Granot
 */
package sagi.neta.CurrencyExchanger;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.text.NumberFormat;
import java.util.HashMap;
import java.util.Locale;
import java.util.Map;
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.Logger;

import static javax.swing.JFrame.EXIT_ON_CLOSE;
public class Client extends JPanel{
    /**
     * This class is responsible for all UI objects
     *
     */
    private static final String XML_PATH = "gui.xml";
    private Map<CurrencyPair, Double> exchangeRates;
    static Logger logger = Logger.getLogger("Client GUI");
    private String Date;
    //

    public Client() {
        /**
         * Class constructor
         *
         */
        super(new FlowLayout(FlowLayout.LEADING));
        exchangeRates = new HashMap<>();
        BasicConfigurator.configure();
        logger.info("GUI Constructor init");
        // Amount
        JTextField amountInput = new JTextField(20);
        JPanel amount = new JPanel();
        amount.add(amountInput);
        amount.setBorder(BorderFactory.createTitledBorder("Enter Ammount"));
        add(amount, BorderLayout.CENTER);
        //Date updated
        JLabel dateText = new JLabel();
```

```
        add(dateText, BorderLayout.CENTER);
        dateText.setText("(Click 'Rates Table' to update)");

        // From
        JPanel from = new JPanel();
        JComboBox fromOptions = new JComboBox(Currency.values());
        from.add(fromOptions);
        from.setBorder(BorderFactory.createTitledBorder("Select currency"));
        add(from, BorderLayout.CENTER);

        // To
        JComboBox toOptions = new JComboBox(Currency.values());
        JPanel to = new JPanel();
        to.add(toOptions);
        to.setBorder(BorderFactory.createTitledBorder("Convert to"));
        add(to, BorderLayout.CENTER);

        // Convert Action
        JLabel convertText = new JLabel();
        JButton convertCmd = new JButton("Convert");
        JPanel convert = new JPanel();
        convert.add(convertCmd);
        convert.add(convertText);
        JButton getRates = new JButton("Rates Table");
        add(getRates);
        add(convert);

        //Table
        Object rows[][] = new
Object[Currency.values().length*Currency.values().length][];
        Object columns[] = { "From", "To","Rate" };
        DefaultTableModel model = new DefaultTableModel(rows, columns);
        JTable table = new JTable(model);
        JScrollPane scrollPane = new JScrollPane(table);
        add(scrollPane, BorderLayout.CENTER);

        /**handeling convertion*/
        ActionListener convertAction = new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                // TODO: Needs proper validation
                logger.info("Performing convertion...");
                String amountInputText = amountInput.getText();
                if ("".equals(amountInputText)) { return; }
                // Convert
                Double conversion = convertCurrency(amountInputText);
                convertText.setText(NumberFormat
                        .getCurrencyInstance(Locale.US)
                        .format(conversion));
            }
            private Double convertCurrency(String amountInputText) {
                // TODO: Needs proper rounding and precision setting
                logger.info("Convert Calculating...");
                CurrencyPair currencyPair = new CurrencyPair(
                        (Currency) fromOptions.getSelectedItem(),
                        (Currency) toOptions.getSelectedItem());
                Double rate = exchangeRates.get(currencyPair);
                Double amount = Double.parseDouble(amountInputText);
                return amount*rate;
            }
        };
        convertCmd.addActionListener(convertAction);
        //
        /** handeling rates table display*/
        ActionListener showAllRates = new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                logger.info("Displaying all rates map");
```

```
                int i = 0;
                for (Currency from : Currency.values()) {
                    for (Currency to : Currency.values()) {
                        BigDecimal val = new BigDecimal(exchangeRates.get(new
CurrencyPair(from, to)));
                        val = val.setScale(2, RoundingMode.CEILING);
                        model.setValueAt(from, i, 0);   //Set from currency
                        model.setValueAt(to, i, 1);     //Set to currency
                        model.setValueAt(val, i, 2);    //Set rate value
                        ++i;
                    }
                }
                dateText.setText("(Updated: "+Date+")");
            }
        };
        getRates.addActionListener(showAllRates);
    }
    //
    public void setExchangeRates(Map<CurrencyPair, Double> exchangeRates) {
        logger.info("Updated rates map.");
        this.exchangeRates = exchangeRates;
    }
    public void setDate(String newdate) {
        logger.info("Updateding date");
        this.Date = newdate;
    }
```

```
    public static void main(String[] args) {
        /**
         * Updates the hashmap with all rates
         * creates the graphic UI
         * Runs intervals for updating the rates hashmap
         *
         */
        try {
            //First, get the rates xml file and update hash table
            xmlParser Rates = new xmlParser();
            Rates.run();
```

```
            //Create and run GUI
            Client GUI = new Client();    //Send the c'tor and updated hashmap
containing all data parsed
            Rates.updateHashMap();            //update map according to fetched
data
            //Get updated map with all rates
            GUI.setExchangeRates(Rates.getExchangeRates());
            GUI.setDate(Rates.getDate());
            JFrame frame = new JFrame();
            frame.getContentPane().add(GUI);
            frame.setTitle("Currency Exchanger");
            frame.setSize(500, 620);
            frame.setDefaultCloseOperation(EXIT_ON_CLOSE);
            frame.setLocationRelativeTo(null);
            frame.setVisible(true);
            //Fetch new data intervals
            int delay = 15*60000; //milliseconds (15 min)
            ActionListener taskPerformer = new ActionListener() {
                public void actionPerformed(ActionEvent evt) {
                    logger.info("Update xml and hashmap interval started....");
                    try {
                        Rates.run();
                        Rates.updateHashMap();
                        //Get updated map with all rates
                        GUI.setExchangeRates(Rates.getExchangeRates());
                        GUI.setDate(Rates.getDate());
```

```
                    }catch(CurrencyException e){e.printStackTrace();}
                }
            };
            new javax.swing.Timer(delay, taskPerformer).start();
            //
        }catch (CurrencyException e){
            e.getMessage();
        }catch (Exception e){e.printStackTrace();}
    }

}
```