Currency Converter App Project Design

Revision 10

CMSC 495 6380

June 15, 2021

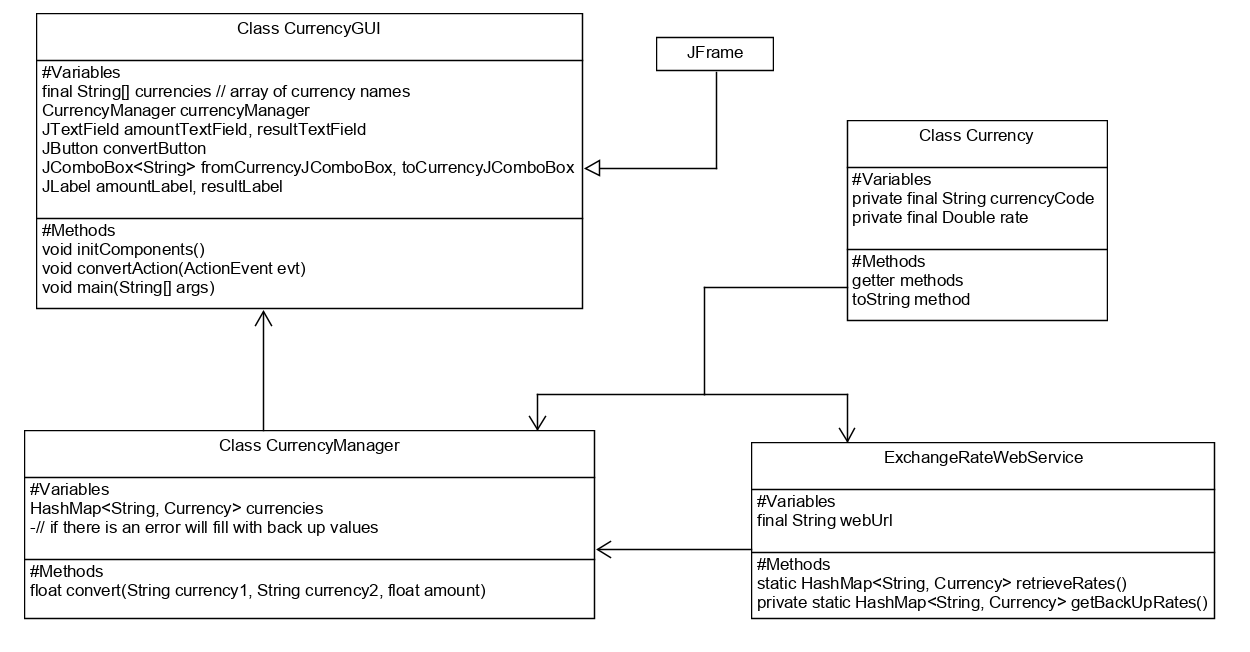
Group 4

Roy Auh, Ronald DeSears, Stephen Snelling

## Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision Number** | **Date** | **Description** | **Name** |
| 1 | 6/11 | Creation of Google Doc, outline | Roy |
| 2 | 6/12 | Added working copy of pseudocode | Ron |
| 3 | 6/13 | Added the list of scenarios to diagram | Roy |
| 4 | 6/13 | Added error-handling scenarios to diagram | Ron |
| 5 | 6/14 | Added Class Diagram | Stephen |
| 6 | 6/14 | Added to the Pseudo Code to reflect the current class diagram. Minor formatting | Stephen, Ron, Roy |
| 7 | 6/14 | Modified Pseudo Code and added Start-up Scenario | Roy |
| 8 | 6/14 | Worked on Scenario Diagrams | Stephen |
| 9 | 6/14 | Proofread Scenarios, added Unresolved Risks | Roy |
| 10 | 7/5 | Updated to reflect design changes and feedback | Stephen |

## I. Class Diagram



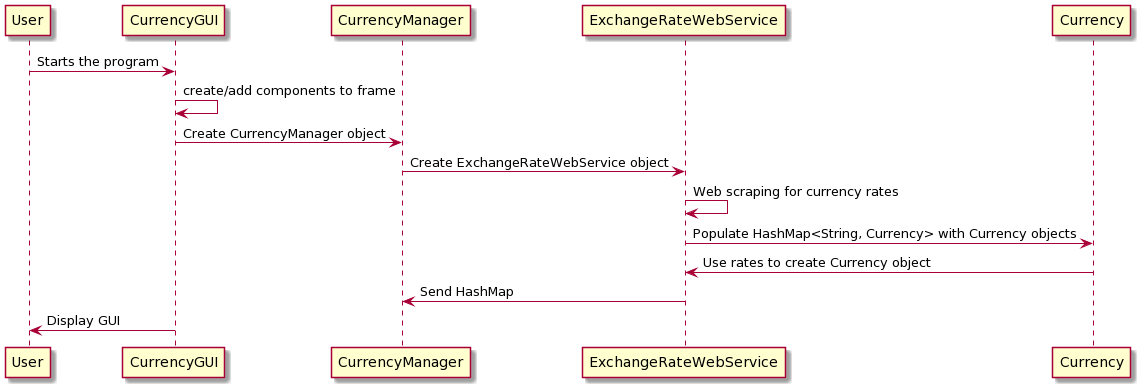
## II. Sequence Diagrams

**Start-up Scenario:**

**Description:** User starts up the application.

**Pre-Condition:** No hardware or system failure that would hinder the application occurs. Internet access is available.

**Post-Condition**: The GUI components, an ExchangeRateWebService object scraps currency rates, and the CurrencyManager object containing an accurate HashMap<String, Currency> currencyList are all successfully generated. GUI is then successfully rendered for display.

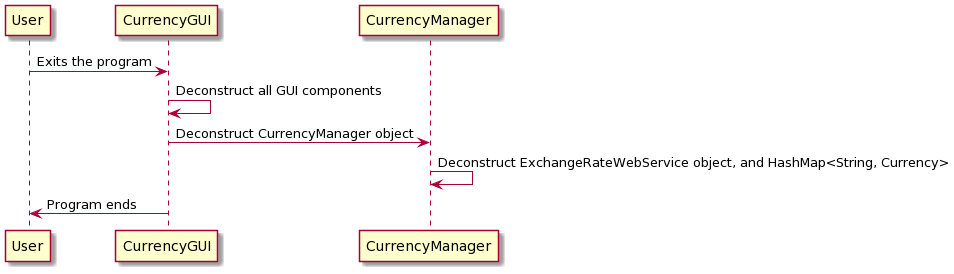


**Shut-down Scenario:**

**Description:** User exits the program.

**Pre-Condition:** Program has successfully been running without crashing.

**Post-Condition**: Program is terminated and all objects are deconstructed.

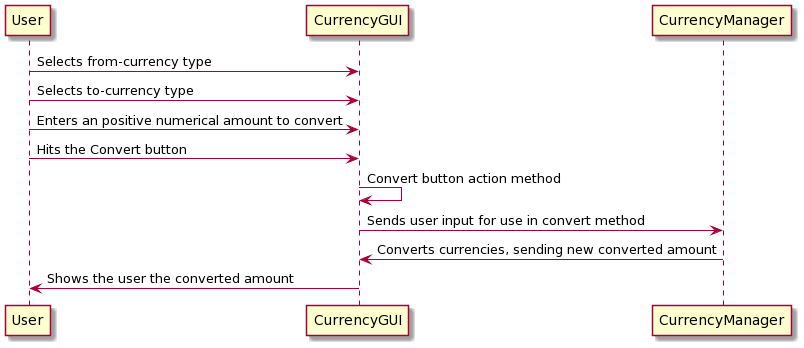


**Normal Operation Scenario:**

**Description:** The user selects a “from-currency” type, a “to-currency” type, enters a proper amount to convert, and clicks the “Convert” JButton. Doing so will return to the user graphically the accurate conversion amount that has been handled by CurrencyManager.

**Pre-Condition:** GUI was created and web data scraped correctly.

**Post-Condition**: Shows the user the correctly converted currency amount.

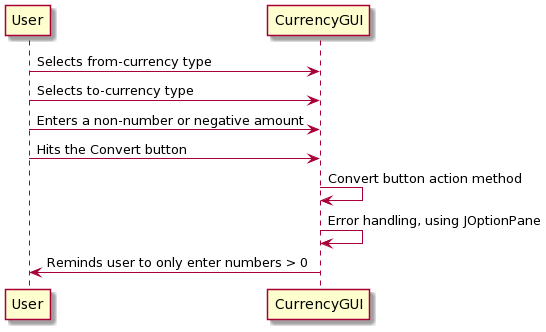


**Error-Handling Scenario 1: User input not formatted properly (Non-number or below 0)**

**Description:** The user enters an invalid currency value, chooses currency for conversion, and then selects “Convert” in GUI.

**Pre-Condition:** GUI is open and ready for user input

**Post-Condition**: Error message is displayed prompting user to enter properly formatted value.

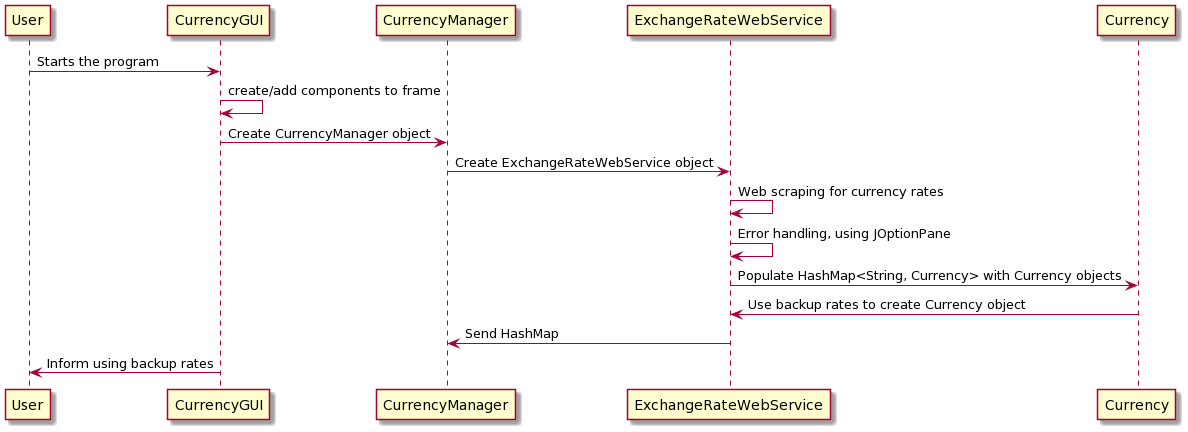


**Error-Handling Scenario 2: Website unavailable to provide current exchange rates**

**Description:** The user starts the program, but the program is unable to scrap the website for exchange rates successfully.

**Pre-Condition:** User ready to start the program.

**Post-Condition**: Error message is displayed indicating exchange rate source is temporarily unavailable and proceeds to use old or backup rates.

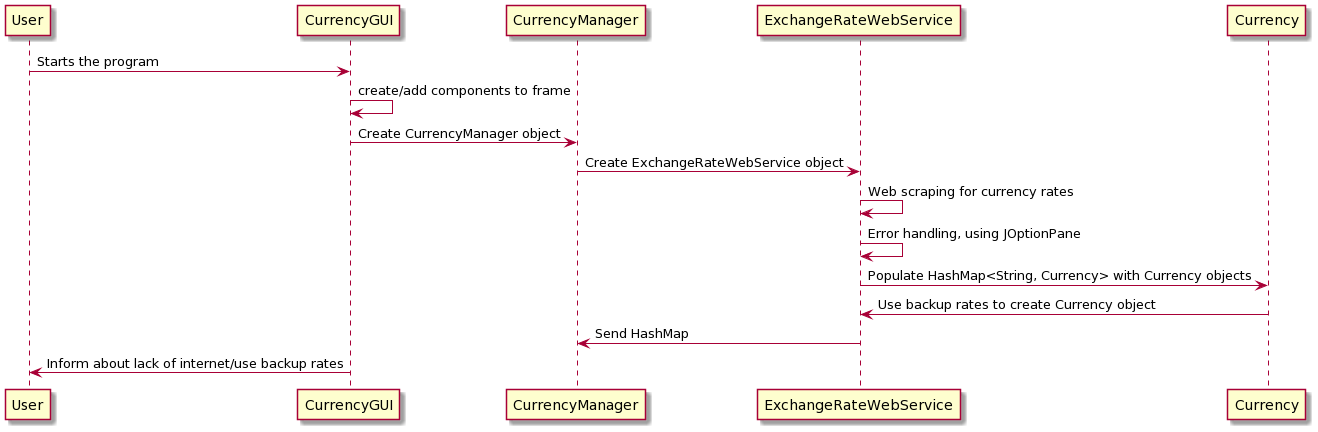


**Error-Handling Scenario 3: Internet unavailable at front-end (User location)**

**Description:** The user starts the program, but the program cannot access the website because of a lack of internet connection.

**Pre-Condition:** User read to start the program.

**Post-Condition**: Error message is displayed indicating internet connectivity issue and proceeds to use old or backup rates.



## III. Pseudo Code

### a. Currency GUI Subsystem:

Class CurrencyGui extends JFrame {

// Starts the GUI and makes it visible

public static void main() {

new CurrencyGUI().setVisible(true);

}

// Adds components to the GUI frame, used in the Constructor

private void initComponents() {

-Initialize all GUI components and place them on the JFrame

-Creates a CurrencyManager object(gets website data at this point)

}

// This method listens for the ActionEvent sent by the Convert Button when pressed

private void convertAction(ActionEvent evt) {

String currency1 = fromCurrency.getSelectedItem();

String currency2 = toCurrency.getSelectedItem();

// Checks to make sure the amount entered is formatted correctly

try {

float amount = Float.parseFloat(originalTextField.getText());

// CurrencyManager will convert for us and return the result

float result = currencyManager.convert(currency1, currency2, amount);

convertedTextField.setText(Float.toString(result)); // could format

} catch (Exception e) {

-use JOptionPane to tell user to make sure the amount they want to

-convert is a number greater than or equal to 0.

}

}

}

### b. CurrencyManager Subsystem:

Class CurrencyManager {

// Method takes in two types of currencies and an amount to convert with currency1

// representing the from-currency. It returns a converted amount in the type of currency2

public float convert(String currency1, String currency2, float amount) {

-Uses the internal HashMap of Currency objects to convert accurately

}

}

### c. Currency Subsystem:

Class Currency {

-getter methods for rate, and currencyCode, along with a toString method

}

### d. Exchange Rate Web Service:

Class ExchangeRateWebService {

string webURL

void retrieveRates() {

try {

Get JSON object containing currency rates from x-rates.com

} catch (ConnectionError e, webURL not found error e) {

Display error message via JOptionPane window

}

Parse JSON for the rates populating a HashMap<String, Currency> with

Backup rates if the JSON object couldn’t be created

}

}

## IV. Unresolved Risks and Risk Mitigations

1. *Web service access failure*: Risk involved with trusting the availability of our website providing us up to date rates while we the developers have no control over the site’s content/availability. Can be mitigated by implementing an alternate source or programming in a failsafe.
2. *Hardware or system failure*: The app cannot defend against hardware or total system failure, although the mitigation measure stated in Project Analysis, that the program itself will not be the cause of that global failure, has been implemented.