Currency Converter
Spring 2018

Michael Ibanez

1. Outline :

   This project is a currency converter that use CurrenyLayer's API to get real time conversions from one currency to another.

2. Vision Statement :

   To create an application with a Graphical User Interface that will demonstrate proper use of Java tools and documentation

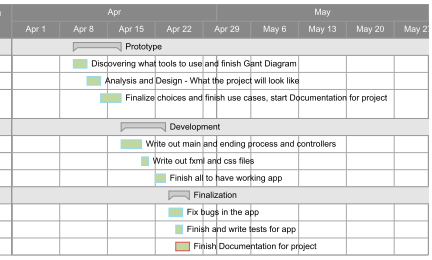
3. All roles

   (a) Client
   (b) MainWindow
   (c) MenuController
   (d) LoadingController
   (e) SettingsController
   (f) ConvertController
   (g) ExitMain

4. Gant diagram with timing:

Currency Converter Project

Task Name	Start Date	End Date	Assigned To	Duration
Prototype	04/10/18	04/16/18	Michael Ibar	5d
Discovering what tools to use and finish Gant Diagram	04/10/18	04/11/18	Michael Ibanez	2d
Analysis and Design - What the project will look like	04/12/18	04/13/18	Michael Ibanez	2d
Finalize choices and finish use cases, start Documentation for project	04/14/18	04/16/18	Michael Ibanez	2d
Development	04/17/18	04/23/18	Michael Ibar	4.5d
Write out main and ending process and controllers	04/17/18	04/19/18	Michael Ibanez	3d
Write out fxml and css files	04/20/18	04/20/18	Michael Ibanez	1d
Finish all to have working app	04/22/18	04/23/18	Michael Ibanez	1.5d
Finalization	04/24/18	04/26/18	Michael Ibar	3d
Fix bugs in the app	04/24/18	04/25/18	Michael Ibanez	2d
Finish and write tests for app	04/25/18	04/25/18	Michael Ibanez	1d
Finish Documentation for project	04/25/18	04/26/18	Michael Ibanez	2d



Exported on April 24, 2018 7:25:17 PM PDT

5. Requirements :

   Injection option given to user to inject css into files

   Easy to navigate

   User interface is simple to use

   Should comply with business rules

   Software is developed with five or more design patterns

   User interface should be consistent with color, font, and size.

   Software should be able to convert one currency to another

   Software should convert in a quick amount of time(2 seconds ¿)

6. Business rules (3 or more) :

   Software is developed with 1000 LOC at least

   Software should be tested with JUnit

   Software should be mainted with Git and at least 30 commits

   Software needs to have a graphical user interface

   Software should be able to convert from any currency to any other currency

   Software should only convert currencies from the CurrencyLayer API

7. Use cases (3 or more) :

   Use Case Name: Currency Convert

   Actors: MenuScreenController , Client , ConvertScreenController

   Triggers : The Client has past loading screen and wants to convert currencies

   Preconditions : Client has passed loading screen.

   Post-conditions : Client will know the currency conversion of a single unit, and the amount the Client entered. System will be ready for another currency as soon as it is over.

   Normal Flow:

   (a) Client will indicate that he/she will want to convert currencies
   (b) MainWindow will load the menu screen
   (c) The client will then choose their option to go the convert screen
   (d) MenuScreenController will then load the convert screen
   (e) Client will input amount of currency to change
   (f) Client will choose the currencies to convert from
   (g) ConvertScreenController will send the call to CurrencyLayer API
   (h) ConvertScreenController send the information back to Client by updating window fields
   (i) Client will see results

Alternate flows

bA1 : MainWindow will not load
bA2 System will display error message and wait for Client to close application

eA1 : Client input invalid numers
eA2 System will display error message and wait for Client to choose valid numers

Summary : The Client, after going past loadingScreenController, will head to the menu(MenuScreenController) and then the Client will choose the convert button to go to the convert page (ConvertScreenController) from there the user inputs in the value to convert as well as the two currencies to convert from.

Use Case Name: InjectCSS

Actors: MenuScreenController , Client , SettingScreenController, System

Triggers : The Client intents to go to the settings and inject into a css file

Preconditions : The Client has opened application.

Post-conditions : Client will have injected his choice of elements into a css file and will then be able to reload project to see updated settings

Normal Flow:

(a) Client will indicate that he/she will want to inject into css files
(b) MainWindow will load the load screen
(c) The client will then choose their option to go the menu scree
(d) LoadingScreenController will then load the menu screen
(e) Client will then choose to go the settings screen
(f) MenuScreenController will then load the settings screen
(g) Client will input class, property, and value of property to inject
(h) Client will inject
(i) ConvertScreenController will then inject into the css files the chosen values
(j) System will have the files changed
(k) Client will reload application to see results

Alternate flows

bA1 : MainWindow will not load
bA2 System will display error message and wait for Client to close application

cA1 : Client chose wrong option
cA2 System will display othermenu that has a back button
cA3 Client will go back and choose correct option

gA1 : Client left fields blank or incorrect values
gA2 ConvertSystemController will pick default values to load the incorrect or blank values

Summary : The Client, after loading application, will head to the loading screen(loadingScreenController) and then the Client will choose the menu button to go to the menu page (MenuScreenController) then

the Client will choose the settings button to go to the settings page (SettingsScreenController). The client will then choose appropriate calues for the injector and after hitting the inject button, will reload the application to see updated files.

Use Case Name: Exit Converter

Actors: System, Client, MainWindow, ExitMain

Triggers : The Client wants to close application

Preconditions : Client has loaded appplication

Post-conditions : Client have shutdown the application

Normal Flow:

(a) Client will indicate that he/she will want to exit application
(b) MainWindow will load the current screen it is at
(c) The client will then choose their option to exit the application by exit , or X button
(d) MainWindow will ask user if they really want to quit application
(e) Client will choose option to close
(f) MainWindow invokes its option to close window
(g) System will close the application

Alternate flows

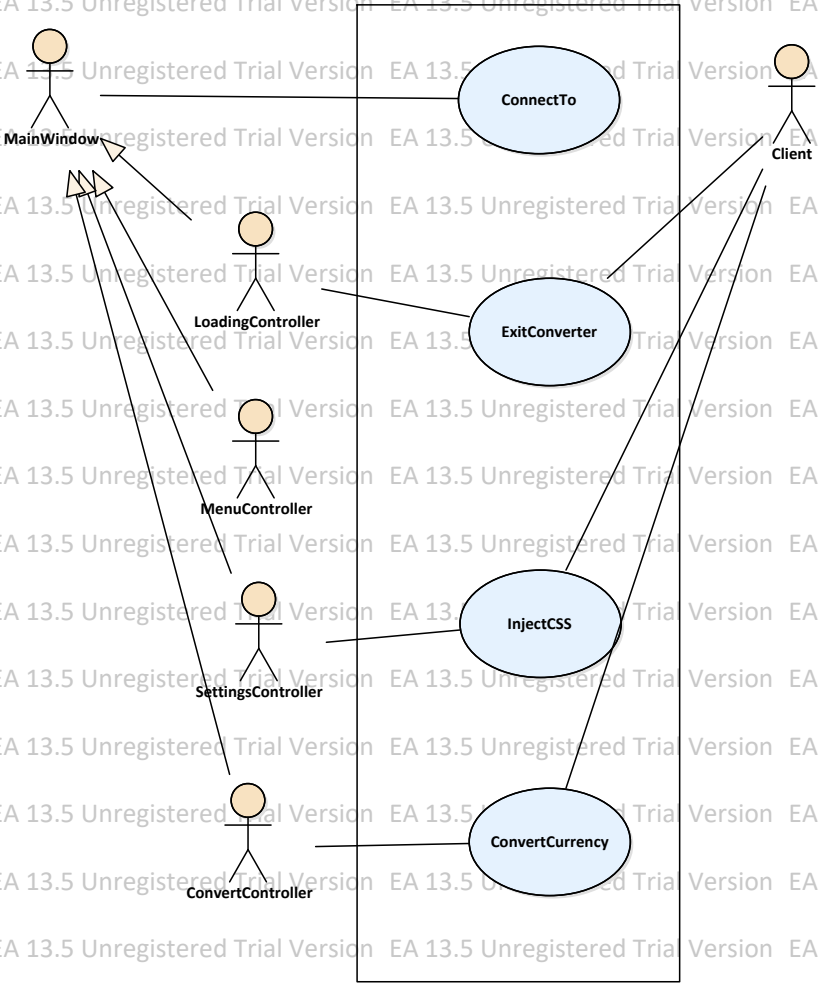
bA1 : MainWindow will not load
bA2 System will display error message and wait for Client to close application

eA1 : Client ichoose option to not close application
eA2 System will display its prior screen and continue on

Summary : The Client, after loading application, will choose the option to close the application by either an exit or X button which will then ask the user if she/he really wants to quit application. The system will then close the application after the Client chooses to do so.

8. Use Case diagram for all roles :

The Use case diagram for all roles.



9. System Operations :

    main(String[] args)

    start(Stage)

    connectToLoading()

    connectToMenu()

    connectToSettings()

    connectToConvert()

    closeProgram(Stage)

    Initialize(URL,Resourcebundle)

    getCurr()

    getWindow()

    setWindow()

    check()

    yesAction(ActionEvent)

    noAction(ActionEvent)

    linkedInAction(ActionEvent)

    githubAction(ActionEvent)

    back(ActionEvent)

    write(File)

    convertAction(ActionEvent)

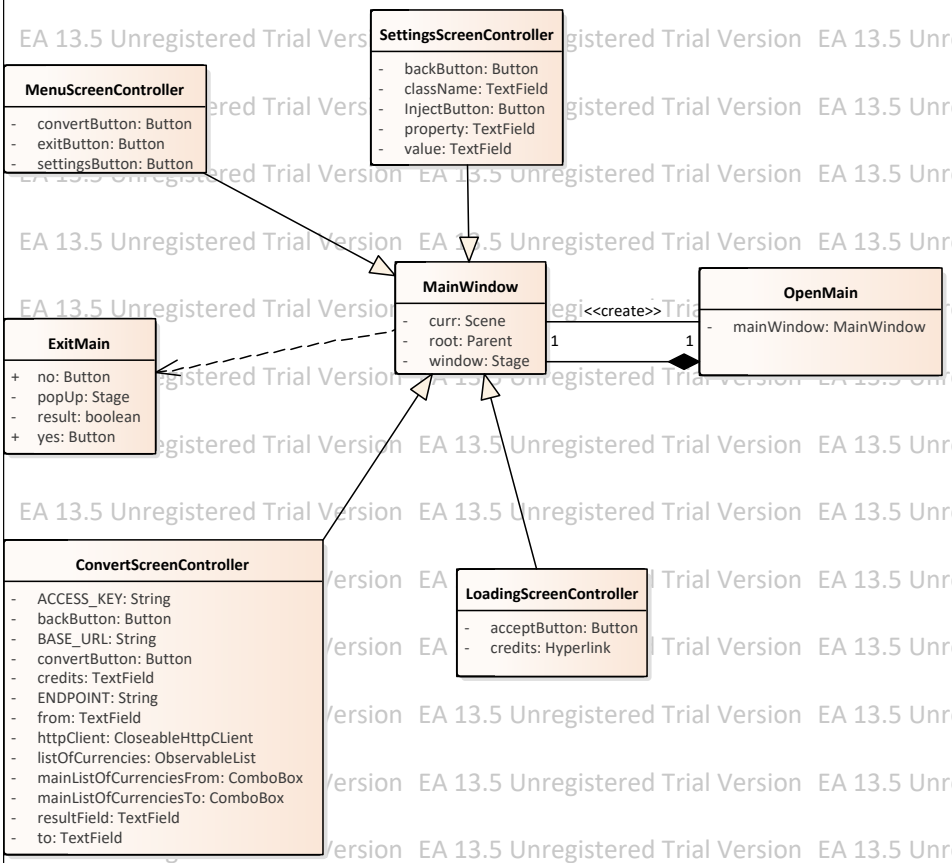
    settingAction(ActionEvent)

    creditsAction(ActionEvent)

    creditsActionFile()

    sendLiveRequest()

10. Domain model :



11. Operation contracts:

(a) Contract CO2: convertAction
Operation:convertAction(ActionEvent e)
Cross References - Use Cases: Currency Convert
Pre-conditions: Client has loaded application
Post-conditions: Client now knows a new currency conversion
ConvertScreenController has made a call to the CurrencyLayerAPI
ConvertScreenController is ready to make another conversion

(b) Contract CO2: creditsAction
Operation: creditsAction(ActionEvent e)
Cross References - Use Cases:
Pre-conditions: Client has loaded application into menu/ setting/ convert screen
Post-conditions: Client will see a credit to the author in console

(c) Contract CO2: back/exit
Operation: back(ActionEvent e)
Cross References - Use Cases: exitConverter
Pre-conditions: Client has loaded application
Post-conditions: Client has exited and properly closed application

(d) Contract CO2: connectToMenu
Operation: connectToMenu()
Cross References - Use Cases:
Pre-conditions: Client has loaded application
Post-conditions: Client is now in the Menu Screen

(e) Contract CO2: connectToSetting
Operation: connectToSetting()
Cross References - Use Cases:
Pre-conditions: Client has loaded application and is in the menu screen
Post-conditions: Client is now in the Setting screen

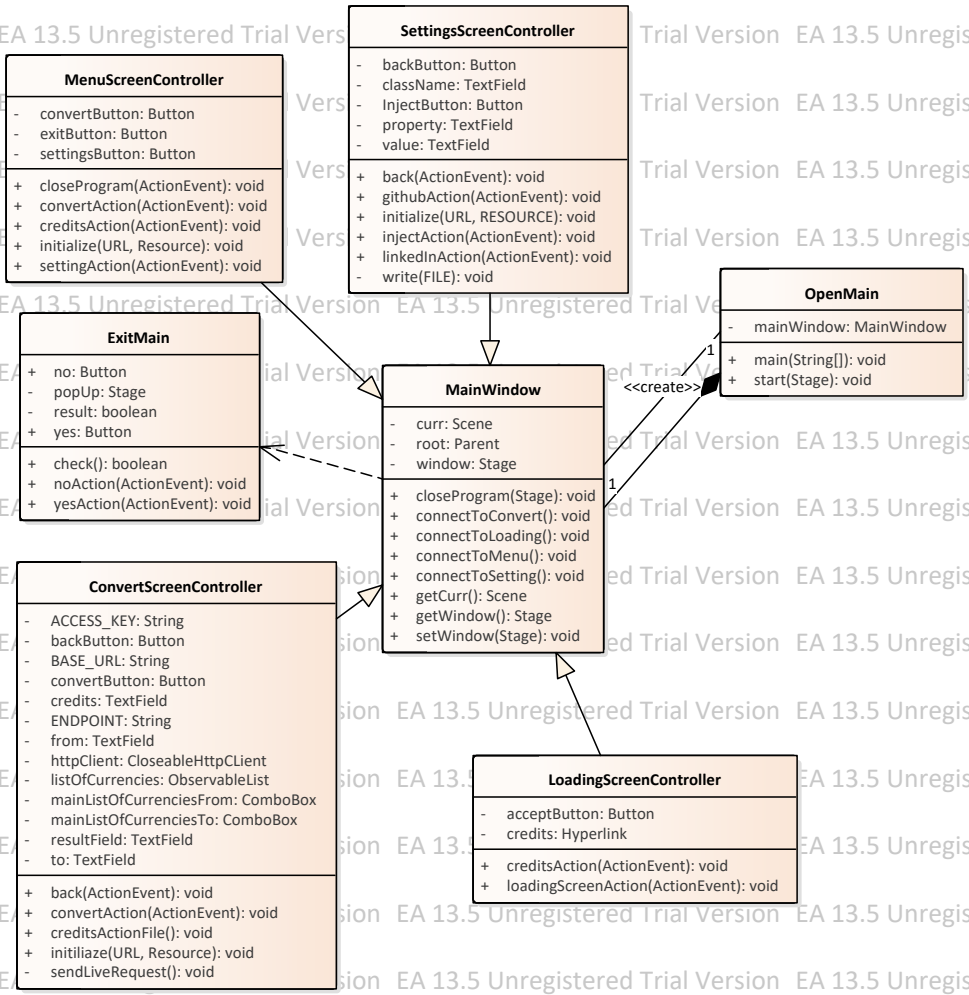
(f) Contract CO2: connectToConvert
Operation: connectToConvert()
Cross References - Use Cases:
Pre-conditions: Client has loaded application and is in the menu screen
Post-conditions: Client is now in the Convert screen

(g) Contract CO2: injectAction
Operation: injectAction(ActionEvent e)
Cross References - Use Cases: InjectCSS
Pre-conditions: Client has loaded application and is in the settings menu
Post-conditions: Client has now loaded a customized version of css files

Client can reload application to see new effects

(h) Contract CO2: initialize
Operation: initialize(URL l, ResourceBundle r)
Cross References - Use Cases:
Pre-conditions: Client has started application
Post-conditions: Client has now loaded the proper css and fxml files
Client can see the effects of the files

12. Design model :



13. Justification of GRASPS :

    Creator : Was MainWindow and Initializable that the ScreenControllers were inherited from, this seemed appropriate in order to have an order of classes

    Information expert : Only classes that needed to know its specific details had access to it, like convertAction in ConvertScreenController was only available to that class. Each class was made specifically for dealing with specific parts, like exitMain was always used for exiting the application, SettingScreenController was always used for injection to the css files

    Low Coupling : Coupling was great here since the classes that depended on something else were the Controller and MainWindow classes. If something changed in MainWindow, it wasn't difficult to change it in the other classes since it was an extension of it. MainWindow was extended by the Controller classes, but that was pretty much all it had so the coupling was still good

    Controller : This looked justified by how each Controller, ExitMain, and MainWindow each took user response when needed to, so the user wasn't always going through the same object

    High Cohesion : Cohesion was more towards the middle since the Controllers classes only delegated its own functions.

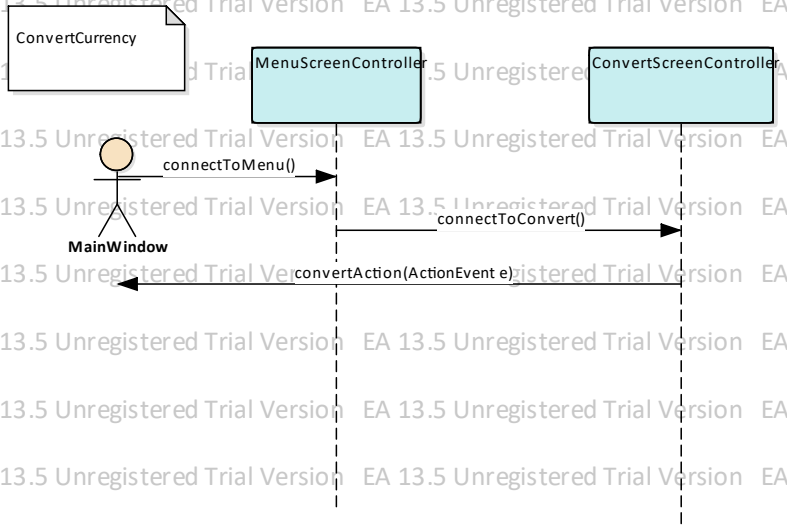
    Law of demeter : Other classes weren't allow to access or know about the other classes functions except for the Controllers to MainWindow

    Polymorphism : Justified by having each Controller implementing Initializable and then overriding the initializable function each time to a certain implementation

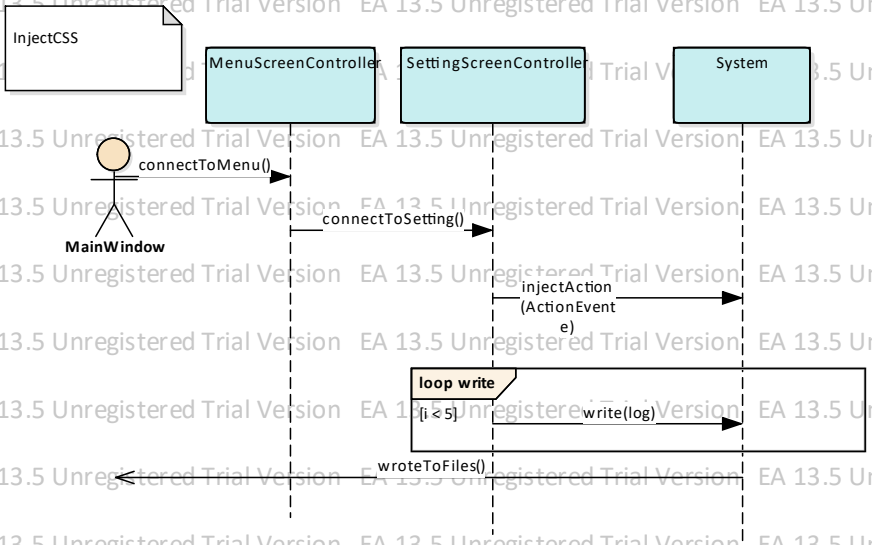
    Pure Fabrication : I made a class specifically to handle the exit methods(ExitMain) which was made up in order to support high cohesion, low coupling, and reuse of the code.

14. Sequence diagrams (for 3 use cases) :

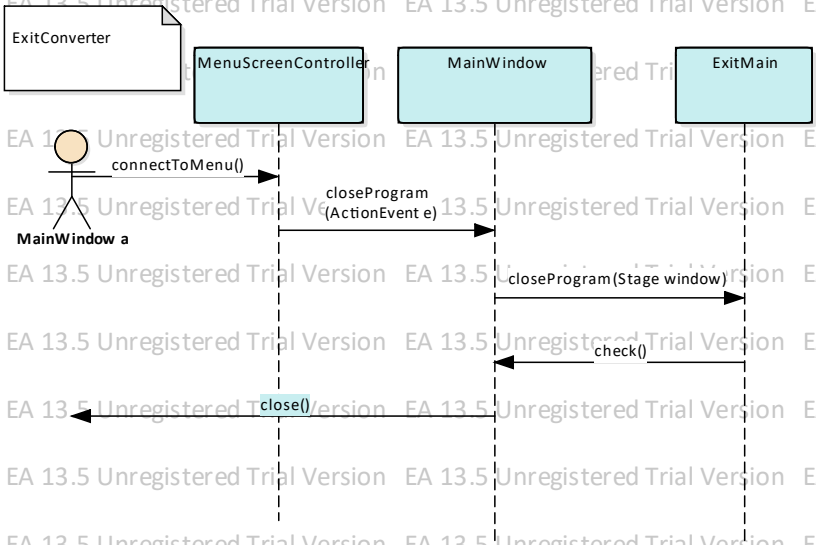
    1 : ConvertCurrency



2 : InjectCSS



3 : exitConverter



15. Where did you use a design pattern and why?

    (a) Command

        This was used in a lot of the buttons like creditAction, creditActionToFile, convertAction, etc. Here when a button was pressed, the appropriate function was called which then called other functions.

    (a) Decorator

        This was used in the initlization of each ScreenController since each one would like to have different settings. I used it here because it was helpful to change what each ScreenController initiliazed to instead ofinitiliazing everything.

    (a) Facade

        This was used in hiding the implemetation of each MainWindow by having the user run the program from a OpenMain instance. This seemed best here so the user couldnt see anything past the OpenMain instance.

    (a) Mediator

        This was used in the ScreenControllers especially for convertScreenController because it separates MainWindow from the action of converting currency (which has an API call to CurrencyLayer).

    (a) Proxy

        This was used in closeProgram because it was useful to be able to have a single function that all the ScreenControllers can use to exit the application at any time. The proxy was the window that popped up and ask the user if he/she really wanted to quit the application, and if so would return to tell the user.

16. LOC : 2173

17. Sources used :

    (a) https://www.programcreek.com/java-api-examples/?class=javafx.stage.Stage&method=setOnCloseReq

    (b) https://examples.javacodegeeks.com/desktop-java/javafx/fxml/javafx-fxml-controller-example/