Bank app

-manage your transactions-

Bank App

Javafx Project

Programming 3

🍍 Anescu adelina Maria 🍍

Table of Contents

Model

- diagrams
- Collections
- Start program
- Tests
- Exceptions

DATABASE

- Tables
- Connection
- Insert data to query

User Interface

- Validate data
- Scene builder
- Fxml scenes

MODEL

User interface Requirements:

1. Create a diagram for your project which models the classes, interfaces and relationships between them.You must model at least 3 classes and 2 interfaces

   Next slides will present that. Also, for Collections, I USED Observable Lists from fxCollectons, with different objects(String,Transaction) in order to create an array of transactions and to set the field for ChoiceBoxes.

2. Use configuration files and/or program arguments to start the program in different ways.

   You can start the project by introducing 1 or 2 as input, "1" will start the Login view, and "2" will try autologin with the information provided in the program arguments.

3. Create at least one test case per class in your model.

   I do not have for all classes, but I tried to to ne for the ones where that was possible. See slides.

User interface Requirements:

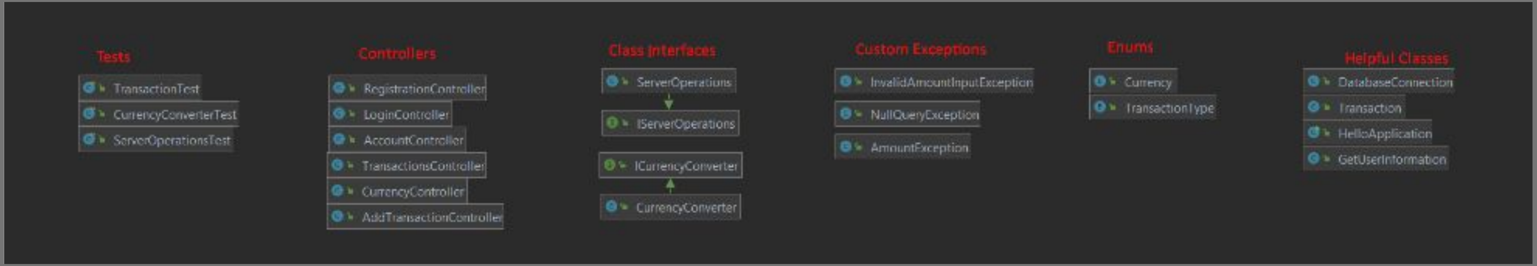
4. Identify and handle at least 3 language Exceptions (FileNotFound, IOException, etc.) that apply to your project

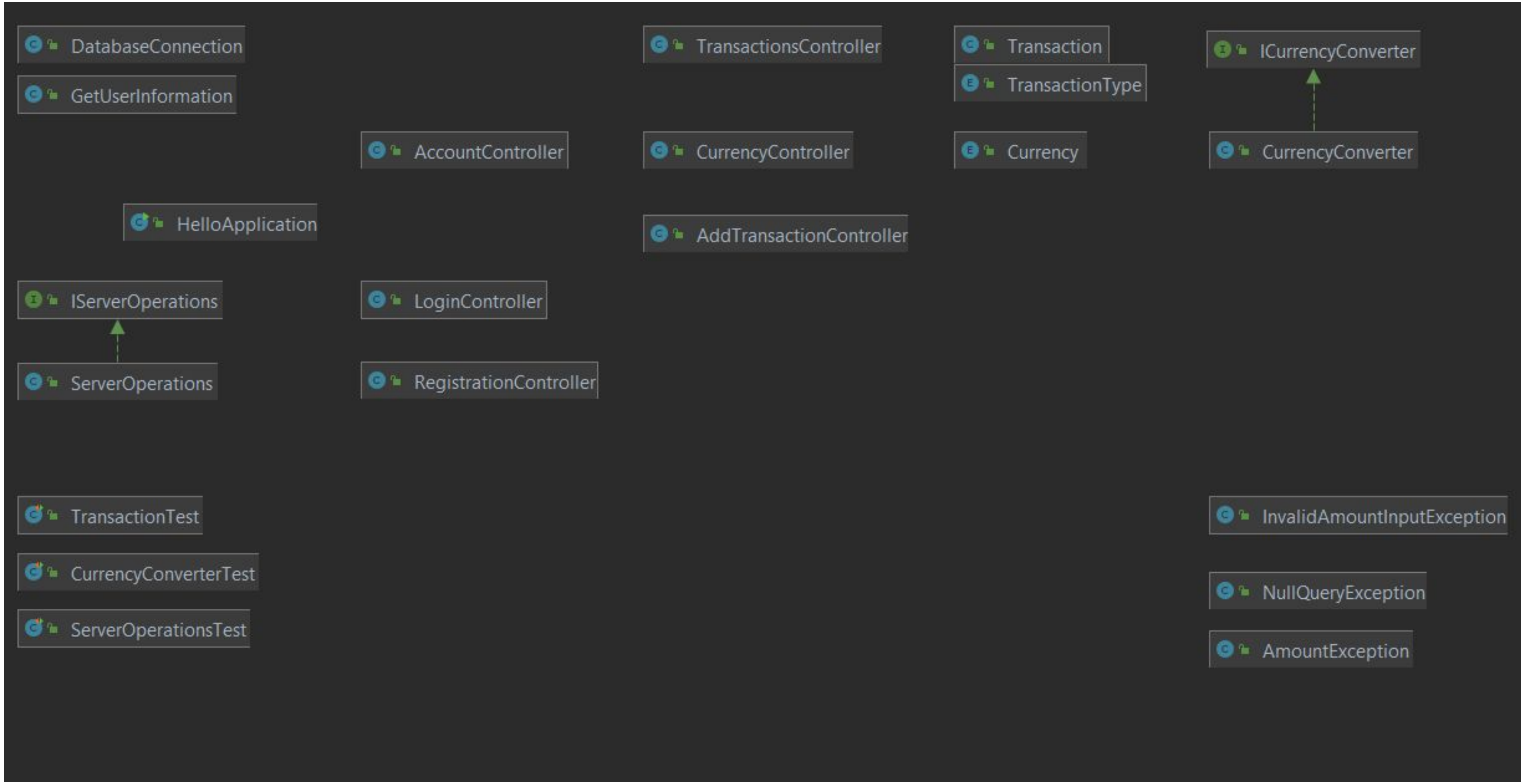
Language Exceptions used: Exception, SQLException, IOException, NumberFormatException

5. Create at least 3 custom Exception useful for your project.

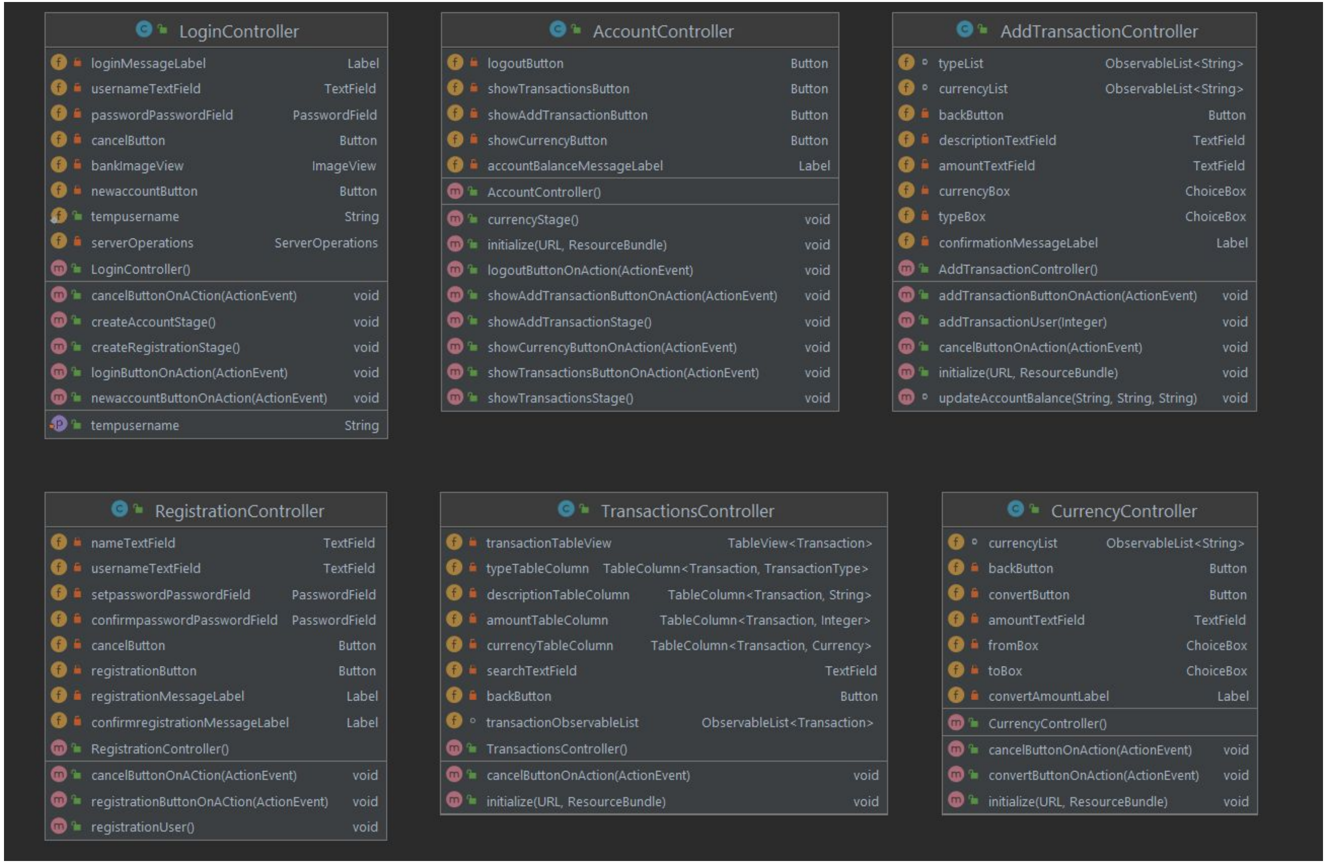
Custom Exceptions: NullQueryException, AmountException, InvalidAMountInputException.
See next slides.

DIAGRAMS


Tests
TransactionTest
CurrencyConverterTest
ServerOperationsTest
Controllers
RegistrationController
LoginController
AccountController
TransactionsController
CurrencyController
AddTransactionController
Class Interfaces
ServerOperations
IServerOperations
ICurrencyConverter
CurrencyConverter
Custom Exceptions
InvalidAmountInputException
NullQueryException
AmountException
Enums
Currency
TransactionType
Helpful Classes
DatabaseConnection
Transaction
HelloApplication
GetUserInformation


DatabaseConnection
GetUserInformation
TransactionsController
Transaction
TransactionType
ICurrencyConverter
AccountController
CurrencyController
Currency
CurrencyConverter
HelloApplication
AddTransactionController
IServerOperations
LoginController
ServerOperations
RegistrationController
TransactionTest
CurrencyConverterTest
ServerOperationsTest
InvalidAmountInputException
NullQueryException
AmountException

CONTROLLERS



LoginController	
loginMessageLabel	Label
usernameTextField	TextField
passwordPasswordField	PasswordField
cancelButton	Button
bankImageView	ImageView
newaccountButton	Button
tempusername	String
serverOperations	ServerOperations
LoginController()	
cancelButtonOnACtion(ActionEvent)	void
createAccountStage()	void
createRegistrationStage()	void
loginButtonOnAction(ActionEvent)	void
newaccountButtonOnAction(ActionEvent)	void
tempusername	String

AccountController	
logoutButton	Button
showTransactionsButton	Button
showAddTransactionButton	Button
showCurrencyButton	Button
accountBalanceMessageLabel	Label
AccountController()	
currencyStage()	void
initialize(URL, ResourceBundle)	void
logoutButtonOnAction(ActionEvent)	void
showAddTransactionButtonOnAction(ActionEvent)	void
showAddTransactionStage()	void
showCurrencyButtonOnAction(ActionEvent)	void
showTransactionsButtonOnAction(ActionEvent)	void
showTransactionsStage()	void

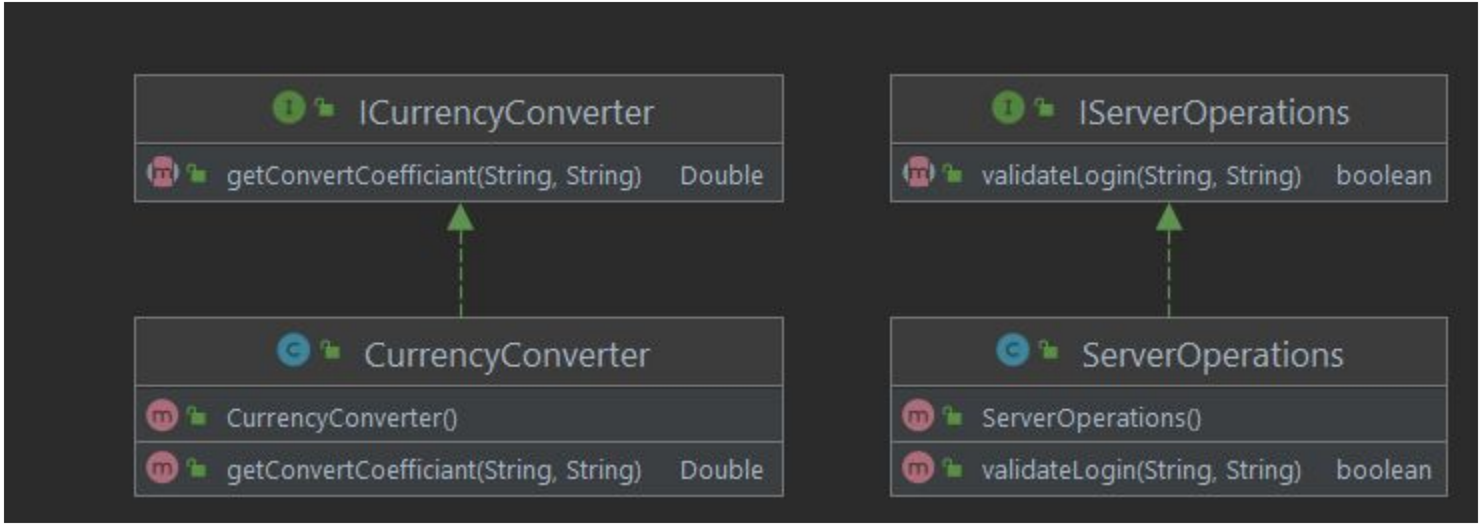
AddTransactionController	
typeList	ObservableList<String>
currencyList	ObservableList<String>
backButton	Button
descriptionTextField	TextField
amountTextField	TextField
currencyBox	ChoiceBox
typeBox	ChoiceBox
confirmationMessageLabel	Label
AddTransactionController()	
addTransactionButtonOnAction(ActionEvent)	void
addTransactionUser(Integer)	void
cancelButtonOnAction(ActionEvent)	void
initialize(URL, ResourceBundle)	void
updateAccountBalance(String, String, String)	void

RegistrationController	
nameTextField	TextField
usernameTextField	TextField
setpasswordPasswordField	PasswordField
confirmpasswordPasswordField	PasswordField
cancelButton	Button
registrationButton	Button
registrationMessageLabel	Label
confirmregistrationMessageLabel	Label
RegistrationController()	
cancelButtonOnACtion(ActionEvent)	void
registrationButtonOnACtion(ActionEvent)	void
registrationUser()	void

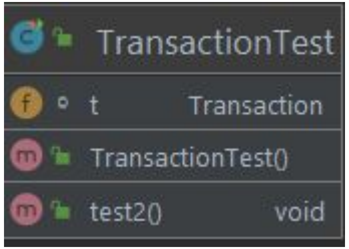
TransactionsController	
transactionTableView	TableView<Transaction>
typeTableColumn	TableColumn<Transaction, TransactionType>
descriptionTableColumn	TableColumn<Transaction, String>
amountTableColumn	TableColumn<Transaction, Integer>
currencyTableColumn	TableColumn<Transaction, Currency>
searchTextField	TextField
backButton	Button
transactionObservableList	ObservableList<Transaction>
TransactionsController()	
cancelButtonOnAction(ActionEvent)	void
initialize(URL, ResourceBundle)	void

CurrencyController	
currencyList	ObservableList<String>
backButton	Button
convertButton	Button
amountTextField	TextField
fromBox	ChoiceBox
toBox	ChoiceBox
convertAmountLabel	Label
CurrencyController()	
cancelButtonOnAction(ActionEvent)	void
convertButtonOnAction(ActionEvent)	void
initialize(URL, ResourceBundle)	void

INterfaces


ICurrencyConverter
getConvertCoefficiant(String, String) Double
CurrencyConverter
CurrencyConverter()
getConvertCoefficiant(String, String) Double
IServerOperations
validateLogin(String, String) boolean
ServerOperations
ServerOperations()
validateLogin(String, String) boolean

Tests



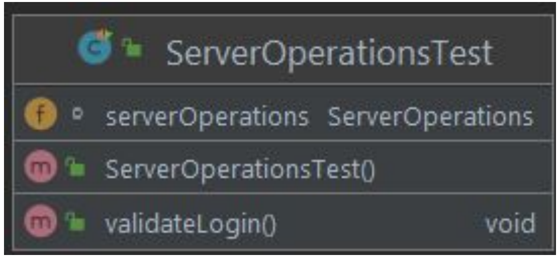
TransactionTest

t Transaction

TransactionTest()

test2() void

Checks if a new transaction object is created properly with the constructor.



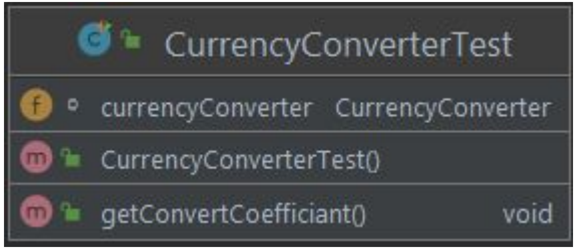
ServerOperationsTest

serverOperations ServerOperations

ServerOperationsTest()

validateLogin() void

Checks if the validate input function words and does not return a false value



CurrencyConverterTest

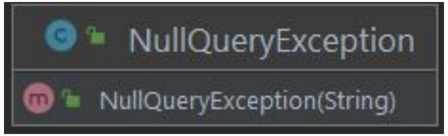
currencyConverter CurrencyConverter

CurrencyConverterTest()

getConvertCoefficiant() void

Checks if the currency coefficient is correct.

Custom Exceptions

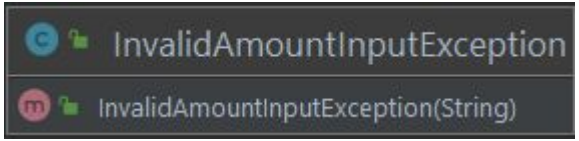


NullQueryException

NullQueryException(String)

Null Query

- Throws exception if the resulted query after the SQL statement is null

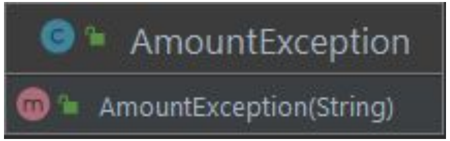


InvalidAmountInputException

InvalidAmountInputException(String)

Invalid Amount input

- Throws exception if the amount input was with comma (,), not point (.)



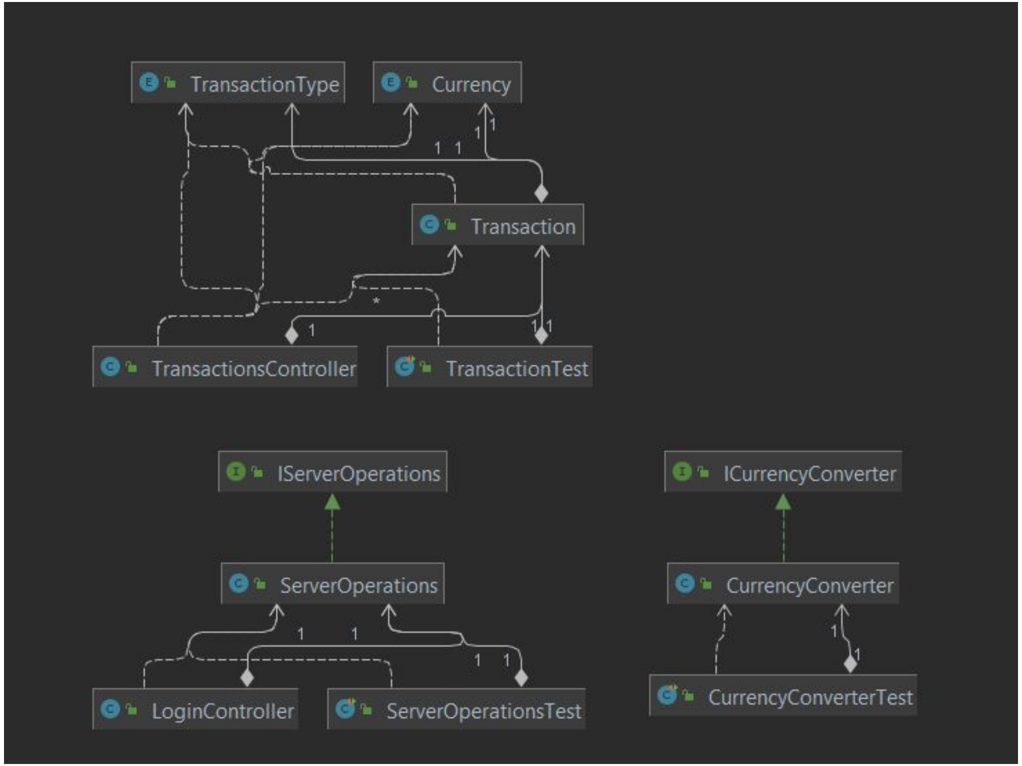
AmountException

AmountException(String)

Amount underflow

- Throws exception if the user want to do a transaction and does not have enough money

DEPENDENCIES



*ONLY CLASSES THAT ARE INTERCONNECTED ARE PRESENTED IN THIS DIAGRAM

Database connection

User interface Requirements:

1. Install a database management software and create the entities related to the model of your project (classes, relations).

   I used XAMPP Control Pane and MySQL WorkBench and later changed to phpMyAdmin.

.

2. Create a connection with the database, query data and show it on screen (or UI).

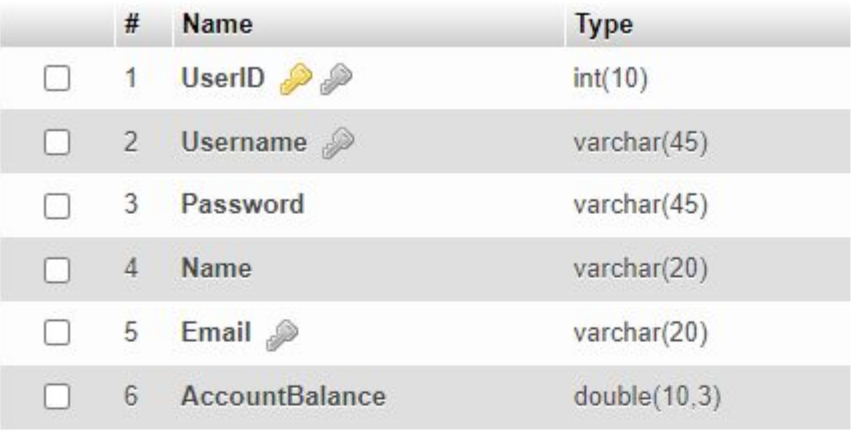
   I use 3 tables, each with an important role, you can see the transactions in "Search transactions".

3. Insert or update data into the DB tables.

   I insert new users and new transactions, also the account balance is updated after every transaction.

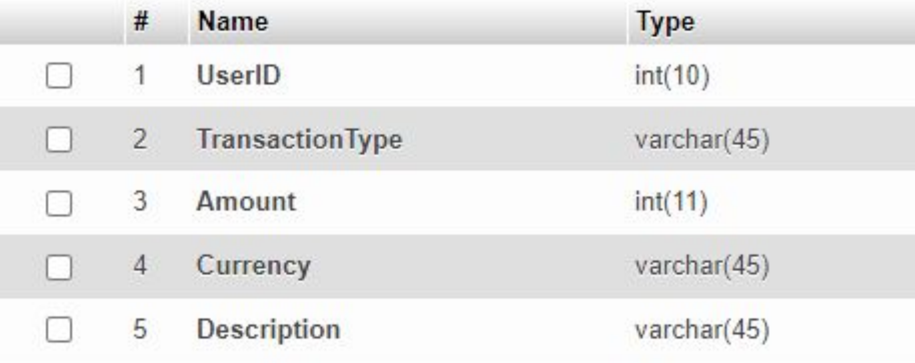
Tables Fields

//useraccounts



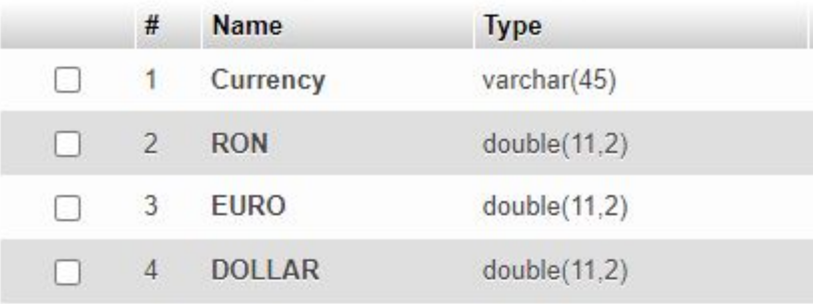
	#	Name	Type
□	1	UserID 🔑 🔑	int(10)
□	2	Username 🔑	varchar(45)
□	3	Password	varchar(45)
□	4	Name	varchar(20)
□	5	Email 🔑	varchar(20)
□	6	AccountBalance	double(10,3)

//usertransactions



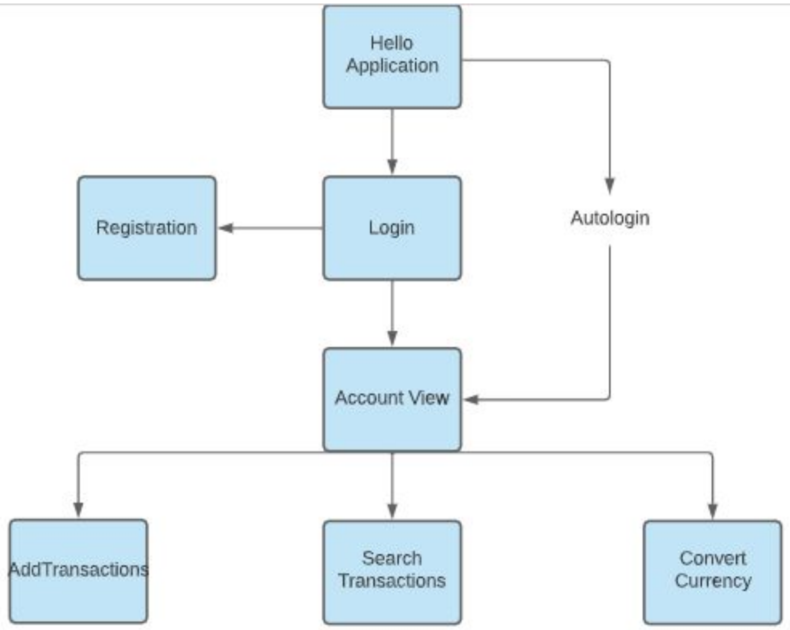
	#	Name	Type
□	1	UserID	int(10)
□	2	TransactionType	varchar(45)
□	3	Amount	int(11)
□	4	Currency	varchar(45)
□	5	Description	varchar(45)

//changecurrency



	#	Name	Type
□	1	Currency	varchar(45)
□	2	RON	double(11,2)
□	3	EURO	double(11,2)
□	4	DOLLAR	double(11,2)

User interface



1. Run application with login(press 1), with autologin(press2).
2. If you do not have an account, from login form go to "new account" button
3. In you account, you can see your account balance.
4. Form there, you have three options
   a. Add transaction
   b. Search transaction
   c. Convert currency

User interface Requirements:

1. Validate input data before using it in your program (especially when interacting with the DB). All inputs must be validated.

   I validated the login registration and validated the “Amount”field to check if is a number.

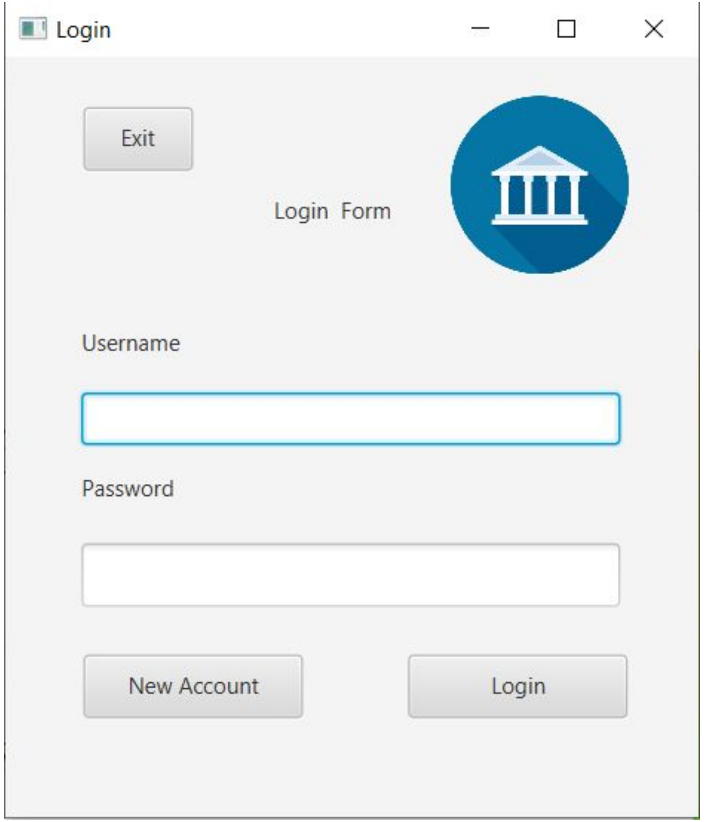
2. Create Scenes for your project. Each Scene must provide a different behaviour for your project.

   I have 6 scenes with different behavior.

Login System

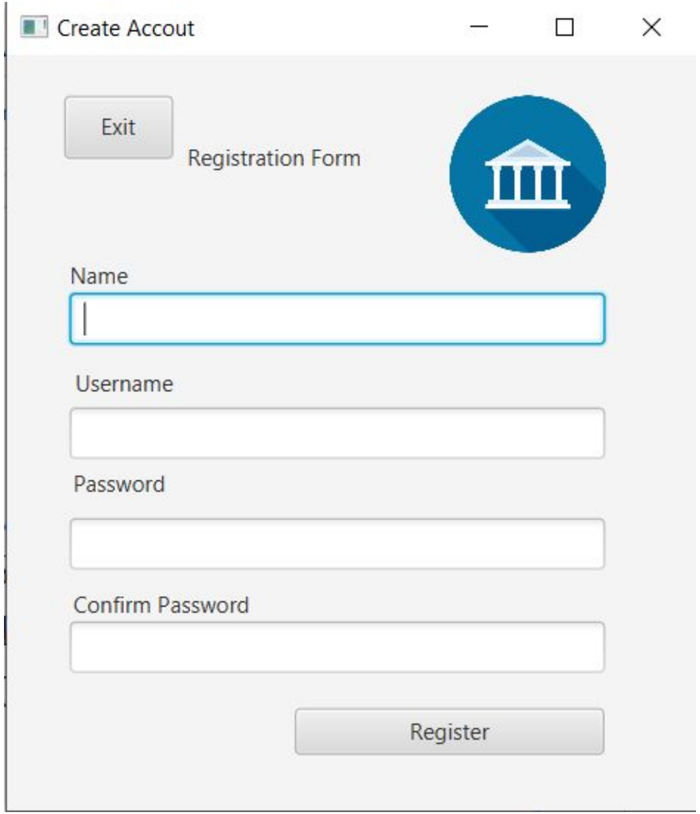
When you open the app in IntelIJ, you will see a Login Form with 2 fields: username, and Password.

In order to access the account scence, you need to enter your informations correctly.



Registration System

After pressing new account in login page, you will be redirect in registration page. Here you can create your account. You will have 100 RON in your account.



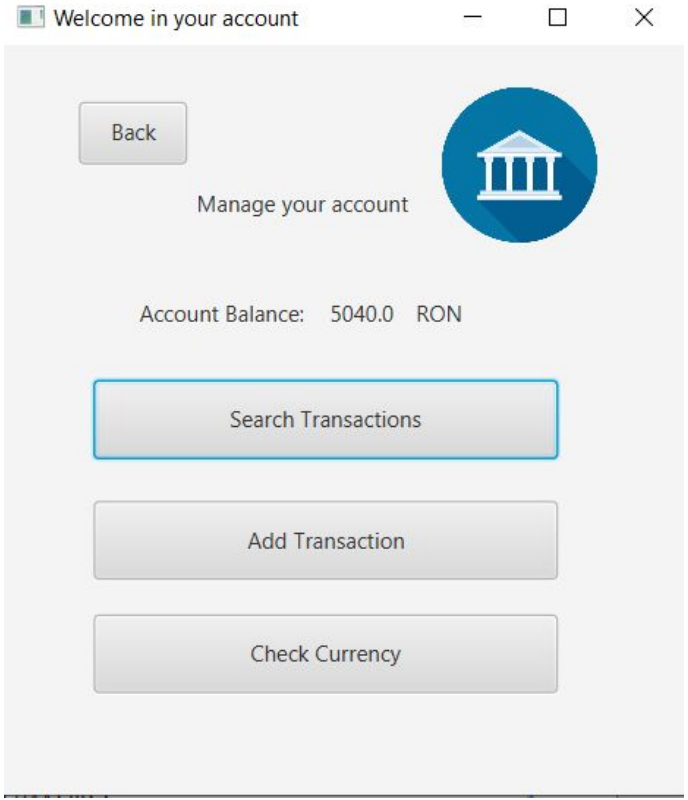
Account view

Here you can see your account balance and you can execute any of the tree options:

-Add transaction

-Search transaction

-Convert currency



Welcome in your account

Manage your account

Account Balance: 5040.0 RON

Add transaction view

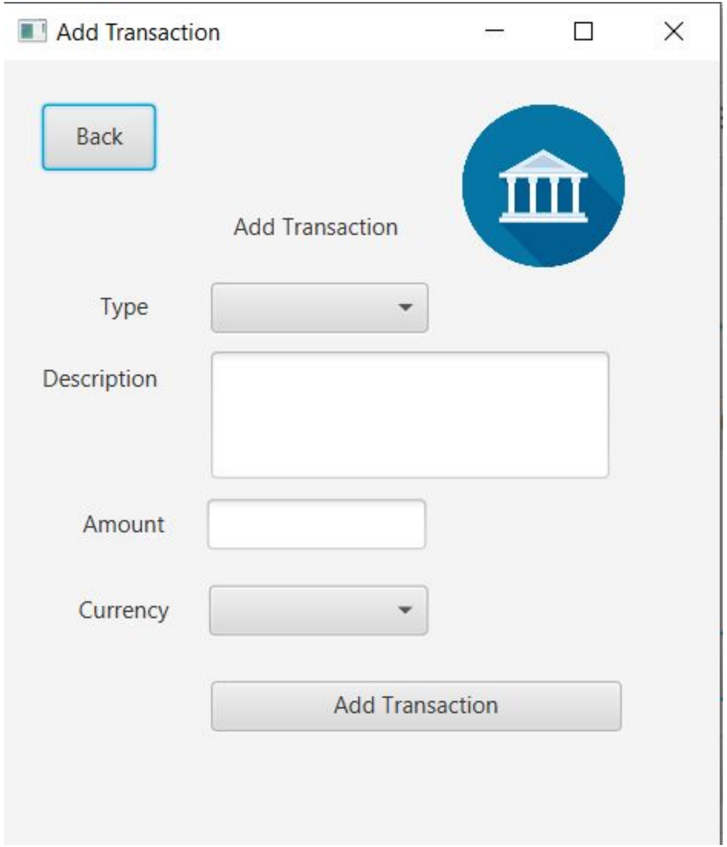
Here you can add a transaction, be sure that you have enough money if it is a expense.

Type: expense/Income

Currency: euro/ron/dollar

The amount will be converted in ron in your balance

Try to keep the description short



Add Transaction

Back

Add Transaction

Type

Description

Amount

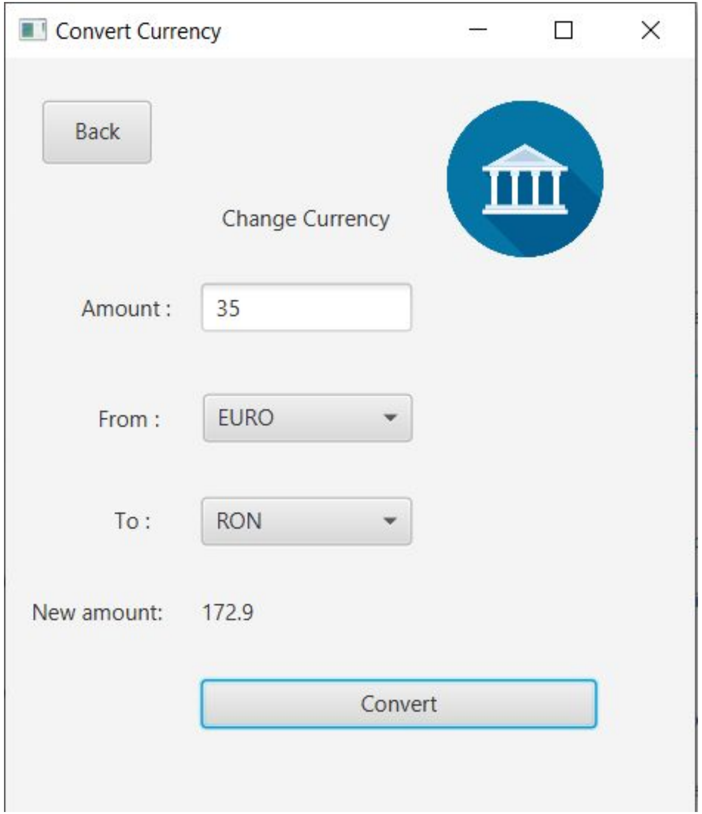
Currency

Add Transaction

Convert currency view

Here you can check the conversion from euro/ron/dollar to other currencies.

In the photo, is just an example:



Amount : 35

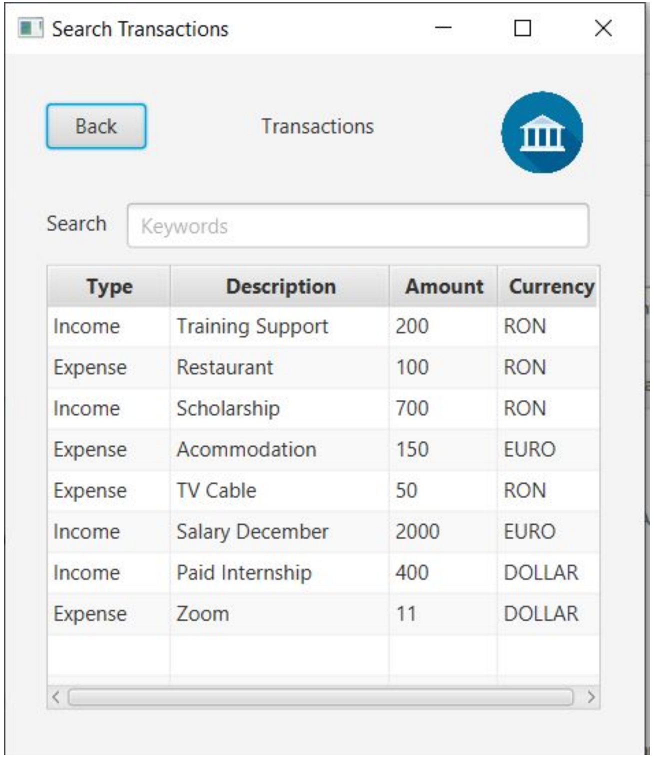
From : EURO

To : RON

New amount: 172.9

Search transactions view

Here you have all the transactions, by using any keyword they will be filtered and try to find a match in any column.



Search Keywords

Type	Description	Amount	Currency
Income	Training Support	200	RON
Expense	Restaurant	100	RON
Income	Scholarship	700	RON
Expense	Acommodation	150	EURO
Expense	TV Cable	50	RON
Income	Salary December	2000	EURO
Income	Paid Internship	400	DOLLAR
Expense	Zoom	11	DOLLAR

Thank you!