Natalia Palej A00279259

A00279259@student.ait.ie

ATM APP

Automated Teller Machine App

Assigement ESSAY

Software Development for Cloud 2.2

Contents

[Introduction 1](#_Toc130935155)

[LoginScreen.py 1](#_Toc130935156)

[NewAccount.py 1](#_Toc130935157)

[Bank.py 2](#_Toc130935158)

[CustomerDetails.py 2](#_Toc130935159)

[Help 2](#_Toc130935160)

[Python Code 2](#_Toc130935161)

[LoginScreen.py 2](#_Toc130935162)

[Bank.py 5](#_Toc130935163)

[File I/O 9](#_Toc130935164)

[Testing 11](#_Toc130935165)

[Conclusion 17](#_Toc130935166)

[References 18](#_Toc130935167)

# 

# Introduction

This ATM Application is built in Python language. It contains two main classes: LoginScreen.py and Bank.py. The LoginScreen class is responsible for the user login process, whereas the Bank class provides various functionalities like balance inquiry, funds transfer and withdrawal/deposit of funds. The application has a help page that displays transfer limits and allows users to edit their details.

# LoginScreen.py

In the LoginScreen class, users can log in using their account number and a 4-digit PIN. To facilitate this process, I have included buttons 0-9 for the account number and PIN input fields. To enhance user experience, the focus is automatically selected on the account number input field when the LoginScreen is first loaded.

Additionally, I have created a CANCEL button that can be used to delete the last digit entered in case the user makes a mistake while entering their login credentials. If the details are correct, the user is successfully logged in and redirected to the BankScreen.

## NewAccount.py

If the user is not registered, LoginScreen also features an OPTIONS menu that allows users to create a new account. When the user selects this option, a new window will open where they can input their personal details. The account number will be automatically generated and populated for them, and they will have the option to save the account or clear all inputs.

# Bank.py

The Bank class provides various functionalities to the user. It includes:

* Balance Inquiry: The user can check their account balance on the Bank. It displays the current balance in the account.
* Funds Transfer: The user can transfer funds to other users by entering their account number and the amount to be transferred. The application also displays the transfer limit on the help page.
* Withdrawal/Deposit: The user can withdraw or deposit money by entering the amount to be withdrawn or deposited.

## CustomerDetails.py

Edit Details: The user can edit their details like name, password, and contact information by selecting the edit details submenu.

## Help

The Help page displays the transfer limits for the user. It also provides information about the application and how to use it. The user can access this page by selecting the help submenu on the Bank.

# Python Code

### LoginScreen.py

Graphical user interface, application, table

Description automatically generated

Upon displaying the LoginScreen, the code initializes an empty dictionary to store customer data from file. It then reads data from the file and inserts customer data into the dictionary, using the account number as the key. This is accomplished through the read\_data() method which will be explained below.

Text

Description automatically generated

To allow user entry I implemented button\_handler method that first checks which entry field is currently in focus using the focus\_get() method. Once the focused entry field is known, the method inserts the clicked number at the end of the using the insert() method. It basically appends the numbers. If the PIN entry field is selected, the same process is applied to the PIN entry field instead. For testing purposes, if neither the account number/PIN entry field is in focus, I print a message "Nothing Selected" to the console. This method proved to be extremely helpful as I was initially unsure about how to retrieve the current focus id. By printing it to the console, I was able to identify that the focus id's for the account number and PIN entry fields are ".!entry" and ".!entry2", respectively. I’m still unsure why these specific ID’s were assigned to these fields, but, this approach allowed me to implement the desired functionality for the button\_handler method.

Text

Description automatically generated

The clear() method is a function within the LoginScreen class. Its purpose is to delete the last character in the currently focused entry field. This function is executed when the cancel button is clicked, allowing the user to correct any mistakes made while entering their account number or PIN.

Similarly, to button\_handler, the method first checks which field is focused. To delete the last element I have used slicing [:-1] which extracts a substring of the selected field from the beginning to the second last character.

Text

Description automatically generated

The read\_data method reads the customer data from a file named customers.txt and stores the data into a dictionary where the account number is the key and the other customer details (such as PIN, name, surname etc) are the values.

Text

Description automatically generated

The validate\_login method uses the read\_data method to make sure that new accounts are accessible in case if one has been created after opening the app. It gets the account number and PIN from the LoginScreen input fields, checks if the account number exists in the dictionary, and if so, checks whether the PIN matches the one in the dictionary. If the credentials are valid, the Bank screen is launched with the customer data passed as an argument. I am also printing "Valid" message to the console for testing purposes. If the credentials are not valid, the user is prompted with appropriate tkinter messagebox error – depending on if the issue is PIN or the account number.

Text

Description automatically generated

The above code creates a sub menu for the LoginScreen with a label "OPTIONS". This sub menu can be accessed by clicking on the menu bar of the window. The sub menu has one option called "CREATE NEW ACCOUNT" which can be clicked to create a new bank account. The option is created using the add\_command method of the tkinter Menu class. The code also creates an instance of the Menu class and configures the window to use it as the main menu. This allows the sub menu to be displayed on the window.

### Bank.py

The Bank class takes in two variables as parameters: customers and customer. The self.customers variable is a dictionary of all customers with their details, while the self.customer parameter is the current customer who has logged in.

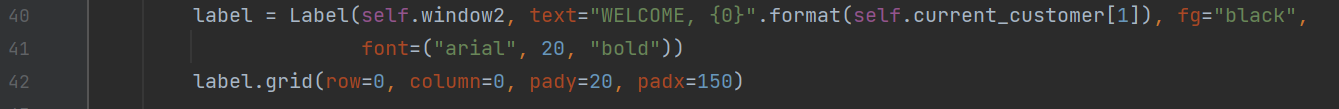
Text

Description automatically generated

In the constructor of the Bank class, a new window is created. There is also submenu created with two options: "EDIT DETAILS" that redirects customer to CustomerDetails.py class and "HELP" that displays transfer limits.

Text

Description automatically generated



The details of the current customer are fetched from the customers dictionary using the customer parameter, and a label is created with the welcome message followed by current customer name.

Text

Description automatically generated

The balance of the customer is also displayed in a label. The balance is updated dynamically using a StringVar object and the set() method is used to set the initial value of the variable to the current balance of the customer. The Label widget is then created and its textvariable parameter is set to the string variable created earlier, so that the label will display the current balance of the customer.

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

To enhance the user experience and increase interactivity, I incorporated a dropdown menu in the app using the Combobox widget. The dropdown menu displays all available customers that the user can transfer money to. This part of the app was probably the most challenging for me. I created a dictionary called customers\_list to hold all the available customers, and implemented a condition to ensure that the current user's name does not appear on the dropdown list. Next, I extracted the names and surnames of all the customers from the customers.items() list using index 1 and 2. Finally, I passed this list to the combobox dropdown for display.

The user can make transactions such as withdraw money or transfer money to another user. Here are some details about each type of transaction:

Deposit:

* If user will want to deposit more than €10,000 he will be prompted with a messagebox that will inform the user of the maximum deposit amount at one time

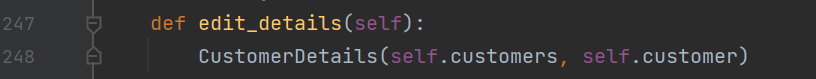
Withdraw:

* To withdraw money, the customer needs to enter the amount they want to withdraw
* If the amount is greater than their balance, they will be prompted with a message asking whether they want to proceed and have a negative balance or cancel the transaction
* If they choose to proceed, their balance will go negative, and they will be in debt to the bank
* If the negative balance exceeds - €1k, they will be informed that the maximum debt allowed is €1k

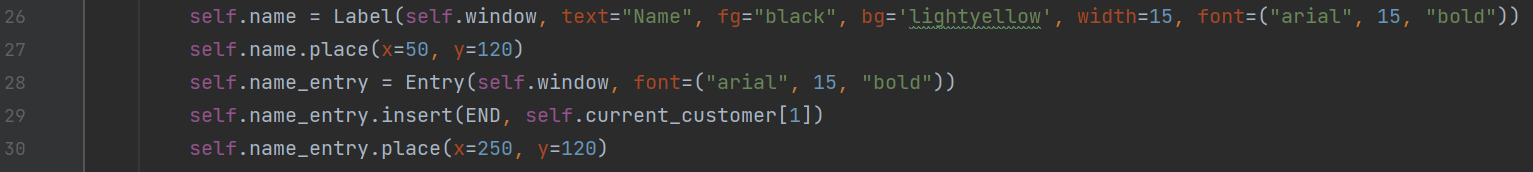
Transfer to another user:

* To transfer money, the customer needs to enter the amount they want to transfer and choose the recipient's name from the dropdown menu
* Once the transaction is completed, a message box will show up, confirming the amount transferred and the recipient's name

The Bank class also has two methods: edit\_details() and help\_details(). The edit\_details() function launches a new screen that enables editing of the current customer's details. During the creation of the CustomerDetails.py interface, I discovered that it needed to be implemented as a Toplevel() rather than a new window, as originally planned. This was necessary because I was unable to read the radiobuttons values.



The CustomerDetails class also receives two arguments: a list of customers and the current customer's index in that list. The GUI has a top-level window that displays all the customer's information, with the option to edit them.



The customer's information is displayed using various Label widgets for each field (name, surname, etc) and an Entry widget to allow the user to edit them. Each label has a corresponding Entry field, which displays the current value for that field.

Text

Description automatically generated

For the gender field, the GUI uses a set of Radiobutton widgets to provide the user with three options: Male, Female, and Undefined. The selected value is stored as a string in the radio variable.

Text

Description automatically generated

The GUI provides two buttons for the user to interact with: SAVE and CLEAR. The SAVE button calls the save\_data method, which retrieves the updated data from the Entry fields, stores it in a list, and updates the current customer's details. Whereas the CLEAR button clears all the Entry fields, allowing the user to start again.

Graphical user interface, application

Description automatically generated

The help\_details() method opens a new window with a transfer limits information.

I also had plans to add a function that displays all the transactions made by the customer, which would further enhance the main purpose of the project (reading and writing to the file). Although I had to move on to other assignments. I may add this function in the future for my own progress and development.

# File I/O

Text

Description automatically generated

The with open("customers.txt") as f: statement opens the "customers.txt" file and assigns it to the file object f. This is done within a with block, which ensures that the file is closed properly after the block is executed. Once done, the I am using for loop to iterate through each line in the file.

The line.strip().split(";") method is used to remove any trailing whitespace from the line and split it into a list of values using the semicolon ";" as a delimiter. This results in a list of customer data.

The self.customers[customer\_data[0]] = customer\_data[1:] statement assigns the customer data to a dictionary data structure called self.customers. The key of the dictionary is the customer's account number, which is stored in the first element of the customer\_data list. The remaining elements of the customer\_data list are stored as the corresponding value in the dictionary. Once all the data is read and stored, I am looping through self.customers dictionary and printing account details in the console for testing purposes.

Text

Description automatically generated

The with open('customers.txt', 'a') opens the "customers.txt" file in append mode. The 'a' mode ensures that any new data added to the file is appended to the end of the file, rather than overwriting existing data.

I am then iterating through each item in the newAcc list, which contains the details of a new customer account and separate each element by semicolon “;”. I am also converting each item to a string before it is appended to the file.

After all the data for the new account has been written to the file, the save\_data.write("\n") statement writes a newline character to the file object save\_data. This ensures that each new account is written on a separate line in the file.

Text

Description automatically generated

The updatedAccount list contains the updated details of the customer account. The self.customers[accNo] = updatedAccount statement updates the value associated with the given accNo key in the self.customers dictionary to the new updatedAccount list.

The with open('customers.txt', 'w') statement opens the "customers.txt" file in write mode. The 'w' mode ensures that the existing file contents are overwritten with the new data.

# Testing

Graphical user interface, application

Description automatically generated

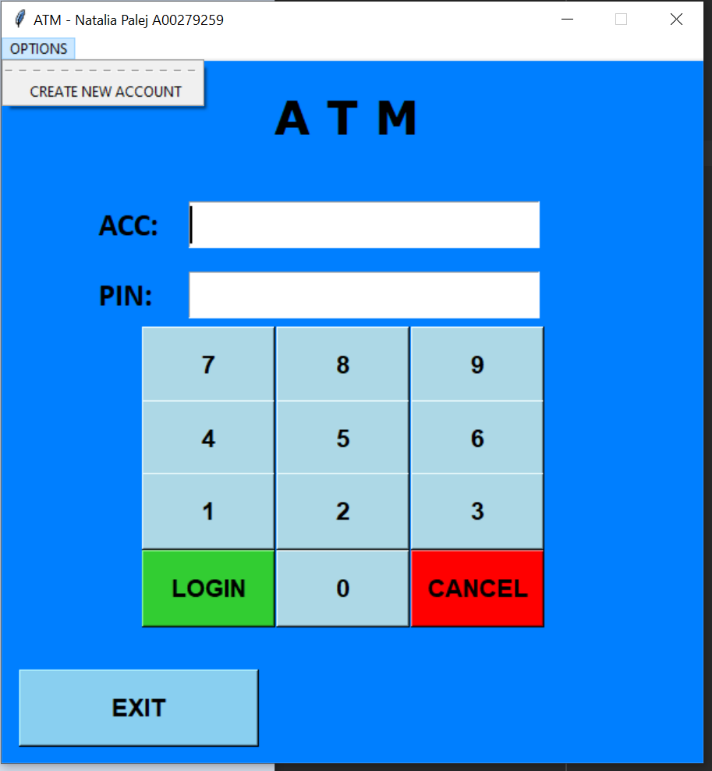
*Login Screen – Main Screen once app is launched.*

*Corresponding error message if the PIN number entered is incorrect.*

Graphical user interface, application, table

Description automatically generated

*Corresponding error message if the account number entered is incorrect.*



Graphical user interface

Description automatically generated

*NewAccount.py once “CREATE NEW ACCOUNT” option from the submenu is selected. Randomly auto-populated account number that is 6 digits long.*

Graphical user interface, application

Description automatically generatedTable

Description automatically generated

*Mike Smith selected as recipient for transfer funds method. Once “transfer” button is pressed and amount selected, message is displayed with transfer details. If user does not have enough funds, he will be notified with error message.*

*Bank.py displayed once user is successfully logged in.*

Graphical user interface, table

Description automatically generated with medium confidenceGraphical user interface

Description automatically generated

*If user’s balance will exceed dept of -1k, they will not be allowed to withdraw funds.*

*If user tries to withdraw more than the balance, warning message is displayed with an option to go ahead or cancel the transaction.*

A picture containing diagram

Description automatically generated

Graphical user interface

Description automatically generated with low confidenceText

Description automatically generatedGraphical user interface, application, Word

Description automatically generatedText

Description automatically generatedText

Description automatically generated

*CustomerDetails.py screen displayed once user selects an option to edit their details.*

*Once user tries to deposit more than 10,000, warning message will be displayed.*

*Once new account is created, user gets prompt with Account Number and reminder to remember his PIN number.*

*New user is then appended to customers.txt*

*Customers.txt updated after depositing the funds when logged in as user Lilly.*

# Conclusion

The ATM Application is a simple yet effective application that provides basic banking functionalities. It is easy to use and provides a secure login process. Developing this ATM app was a challenging experience for me since it required the use of object-oriented programming, which can be complex and difficult to implement correctly. However, I learned a lot while working on this project, including how to design classes, create objects, and use inheritance and polymorphism effectively.

In addition to the object-oriented programming skills, I also improved my understanding of file I/O operations. In particular, I learned how to read, update, and append to text files, which was a critical aspect of this project. The app uses text files to store and retrieve customer account information allowing users to check their account balances and perform various banking transactions.

Overall, this project was a valuable learning experience, and I am proud of the final product that I have created. I believe that this app will provide a reliable and user-friendly banking experience and demonstrate my abilities as a programmer.

# References

Python Software Foundation. "Reading and Writing Files." Python 3.10.2 documentation. [Online]. Available: https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files [Accessed: 11-Mar-2023]

Real Python. (2021). "Python File Handling: Append to a File." [Online]. Available: https://realpython.com/read-write-files-python/#append-to-a-file [Accessed: 12-Mar-2023].

"Tkinter StringVar Class." rin2. [Online]. Available: https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/stringvar.html [Accessed: 12-Mar-2023].

"Python - Dictionary items() Method" Tutorialspoint. [Online]. Available: https://www.tutorialspoint.com/python/dictionary\_items.html [Accessed: 26-Mar-2023]