MAPÚA UNIVERSITY
SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING

Lab 2: Programming Paradigms/Object Oriented Design

CPE106L (Software Design Laboratory)

Hannah Mae Antaran

Darrel Umali Virtusio

Kathleen Joy Tupas

Group: 01

Section: E01

PreLab

Readings, Insights, and Reflection

Lambert, K. A. (2019). Design with Classes. *Fundamentals of Python: First Programs and Data Structures.* pg. 294

This chapter discusses the attributes and behavior of a class of objects required by a program and the appropriate data structures to represent the attributes of a class of objects. As reflection to what we read, we learned that object-oriented programming attempts to control the complexity of the program while still modeling data that change their state. This style divides up the data into relatively small units called objects. Each object is then responsible for managing its own data. If an object needs help with its own tasks, it can call upon another object or relies on methods defined in its superclass. The main goal is to divide responsibilities among small. Relatively independent or loosely coupled components.

Rao, N. R. (2018). Classes and Objects. *Core Python Programming*. pg. 351.

This chapter tackles about python classes and objects. This chapter teaches us all about object-oriented programming mainly about classes and objects and how to create and use them in programming. With the things that we learned from reading this chapter, we can utilize object-oriented programming in future programming projects. This will make our codes more efficient, more reliable, and more secure.

Laster, B. (2016). *Professional Git*. Wrox.

This book takes a professional approach to learning this massively popular software development tool, Git and provides an up to date guide for users. As reflection to what we read, we learned that Git greatly simplifies the software development cycle, enabling users to create, use, and switch between versions. Through this book, we learned about the basic Git model and its overall workflow; how to track changes, work with branches, and take advantages of Git's full functionality.

Eriksson et. al. (2004). Classes, Objects, and their Relationships. *UML 2 Toolkit*. pg. 87.

UML (Unified Modeling language) is a modeling language used to generate models that are easy to construct, visualize, and execute documents within a software system. In Chapter 4, static modeling was discussed as it is a concept of class diagrams that consists of classes and relationships between them. A class in UML comprises of three different naming compartments, these are the name of the class, the attributes of the class, and the operations involved. The attributes of the class describe the characteristics of the objects. UML diagrams show the outline in creating a user's model program. Making use of UML makes the user specify constraints and rules in making detail to implement software systems.

Answers to Questions

1. What are the benefits of having class B extend or inherit from class A?

   Having class B extend or inherit from class A is beneficial because it provides reusability of the code wherein designers don't have to write the same code again and again. It allows to add more features to a class without modifying it. It's also transitive in nature which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

2. Describe what the __init__ method should do in a class that extends another class.

   The __init__ method can be called when an object is created from the class, and access is required to initialize the attributes of the class. This constructor is being inherited from the extended class.

3. Class B extends class A. Class B defines an __str__ method that returns the string representation of its instance variables. Class B defines a single instance variable named age, which is an integer. Write the code to define the __str__ method for class B. This method should return the combined string information from both classes. Label the data for age with the string "Age: ".

age.py > ...

```
1   class A:
2       def __init__(self):
3           self.message = "This is from Class A\n"
4
5       def __str__(self):
6           return self.message
7
8   class B(A):
9       def __init__(self, age):
10          super().__init__()
11          self.age = 15
12
13      def __str__(self):
14          return super().__str__() + "Age: " + str(self.age)
15
16  b = B(A)
17  print(b)
```

Figure 1.1 Source code of age.py.

```
This is from Class A
Age: 15
```

Figure 1.2 Output of age.py.

InLab

- ## Objectives

1. Learn to understand and make UML diagram.
2. Learn and understand the concept of Object-Oriented Programming.
3. Learn the concept of classes and objects.
4. Learn the attributes and behaviors of classes.
5. Understand the concept of inheritance in OOP.

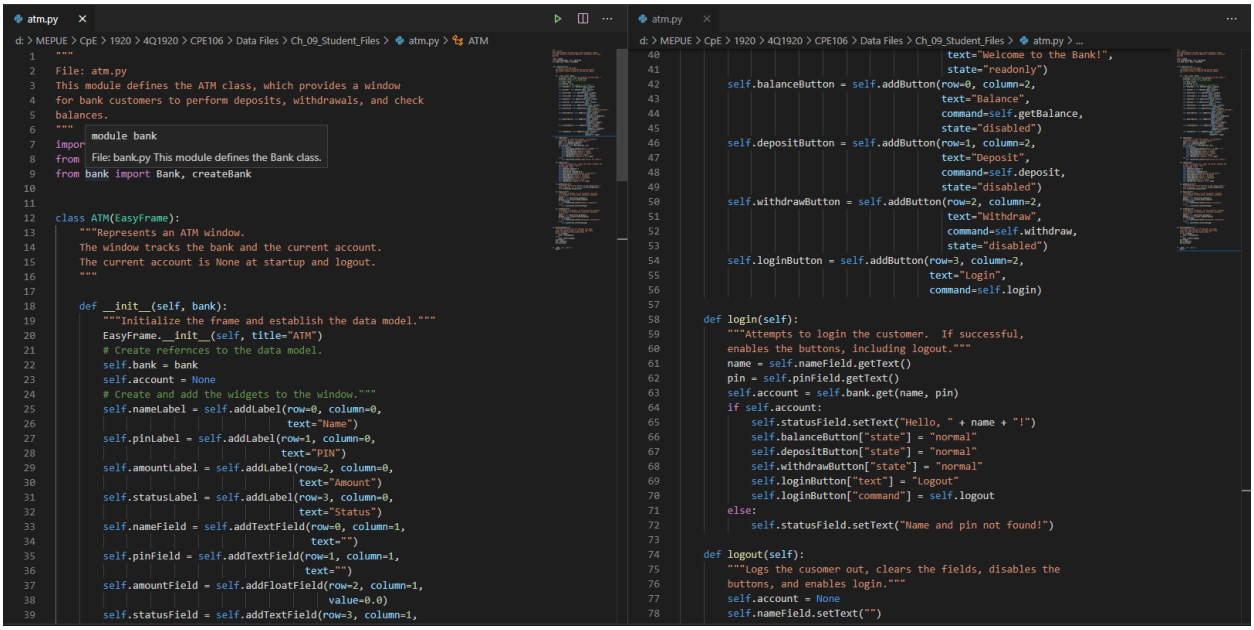
- ## Tools Used

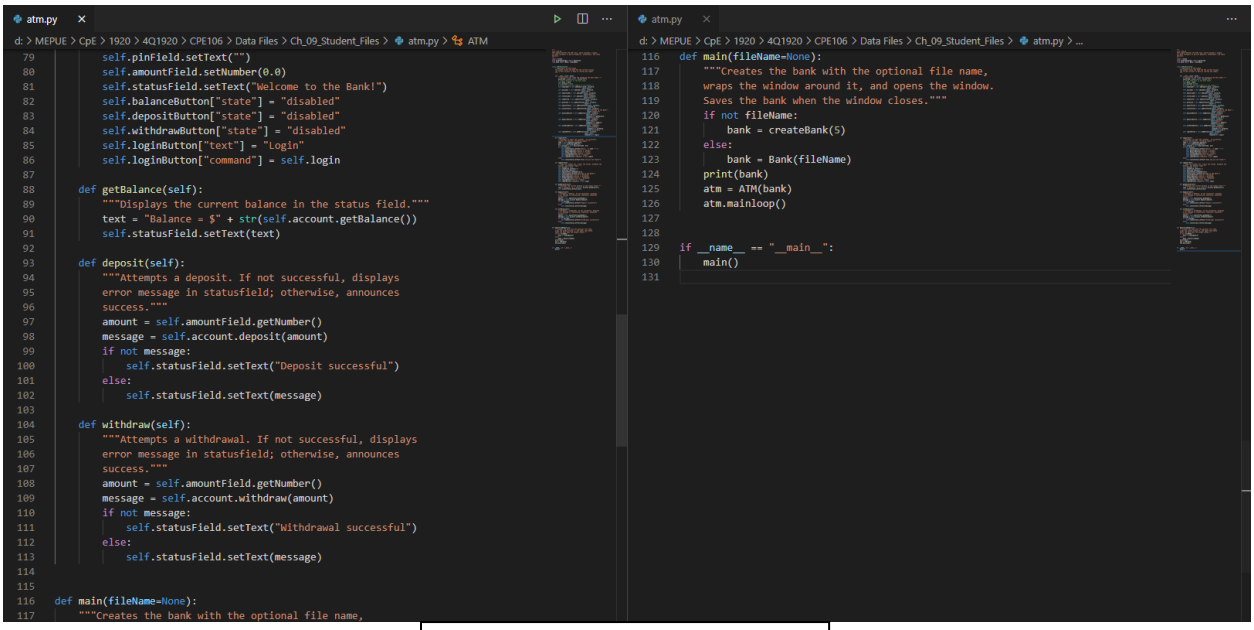
1. Visual Studio Code
2. UMLet

- ## Procedure

Editing the Source Code of atm.py

The python file that we will make a class diagram is the atm.py. The class an ATM window. The window tracks the bank and the current account. The current account is None at startup and logout. The python file also inherits different classes from another python file. The class that atm.py inherits from bank.py is the class Bank. Below is the UML diagram of atm.py.





```
        self.pinField.setText("")
        self.amountField.setNumber(0.0)
        self.statusField.setText("Welcome to the Bank!")
        self.balanceButton["state"] = "disabled"
        self.depositButton["state"] = "disabled"
        self.withdrawButton["state"] = "disabled"
        self.loginButton["text"] = "Login"
        self.loginButton["command"] = self.login

    def getBalance(self):
        """Displays the current balance in the status field."""
        text = "Balance = $" + str(self.account.getBalance())
        self.statusField.setText(text)

    def deposit(self):
        """Attempts a deposit. If not successful, displays
        error message in statusfield; otherwise, announces
        success."""
        amount = self.amountField.getNumber()
        message = self.account.deposit(amount)
        if not message:
            self.statusField.setText("Deposit successful")
        else:
            self.statusField.setText(message)

    def withdraw(self):
        """Attempts a withdrawal. If not successful, displays
        error message in statusfield; otherwise, announces
        success."""
        amount = self.amountField.getNumber()
        message = self.account.withdraw(amount)
        if not message:
            self.statusField.setText("Withdrawal successful")
        else:
            self.statusField.setText(message)


def main(fileName=None):
    """Creates the bank with the optional file name,
```

```
def main(fileName=None):
    """Creates the bank with the optional file name,
    wraps the window around it, and opens the window.
    Saves the bank when the window closes."""
    if not fileName:
        bank = createBank(5)
    else:
        bank = Bank(fileName)
    print(bank)
    atm = ATM(bank)
    atm.mainloop()


if __name__ == "__main__":
    main()
```

Figure 2.1 Source code of atm.py.

This is the complete working program of the atm.py python file. It has a class named ATM with EasyFrame as its parameter. It also inherits the class Bank and the function createBank from another python file named bank.py. Another class named EasyFrame from breezypythongui was inherited for designing its GUI.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS D:\MEPUE\CpE\1920\4Q1920\CPE106\MP1> & C:/Users/dtvir/AppDat
Name:    Mark
PIN:     1000
Balance: 187.0
Name:    Brandon
PIN:     1001
Balance: 933.0
Name:    Elena
PIN:     1002
Balance: 170.0
Name:    Mark
PIN:     1003
Balance: 934.0
Name:    Jack
PIN:     1004
Balance: 900.0
```

Figure 2.2 Output of the program.

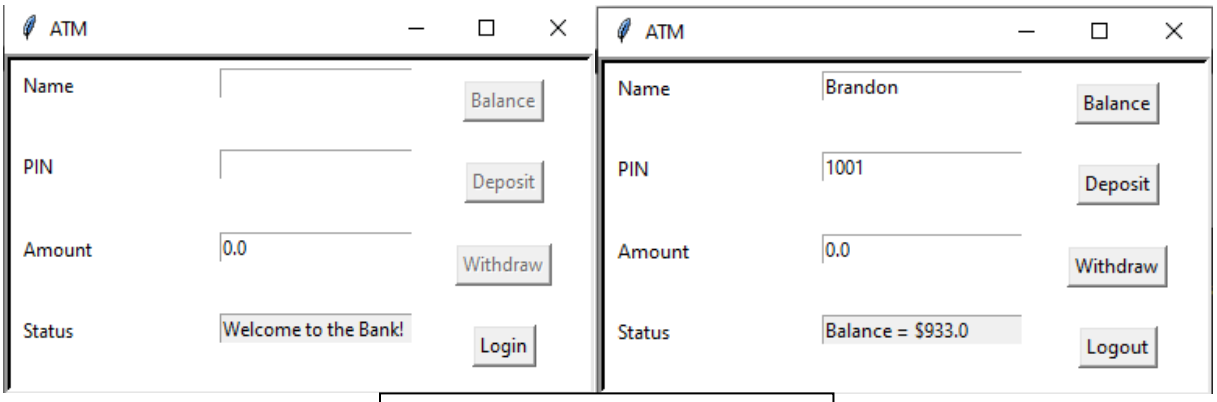


Figure 2.3 Sample run of the program.

We tested the program in this part, and it has 5 accounts saved. The accounts saved was created from a different file which is inherited in atm.py to integrate the program. Brandon was logged in checked the balance to test if the GUI is properly working.

Creating the UML Diagram

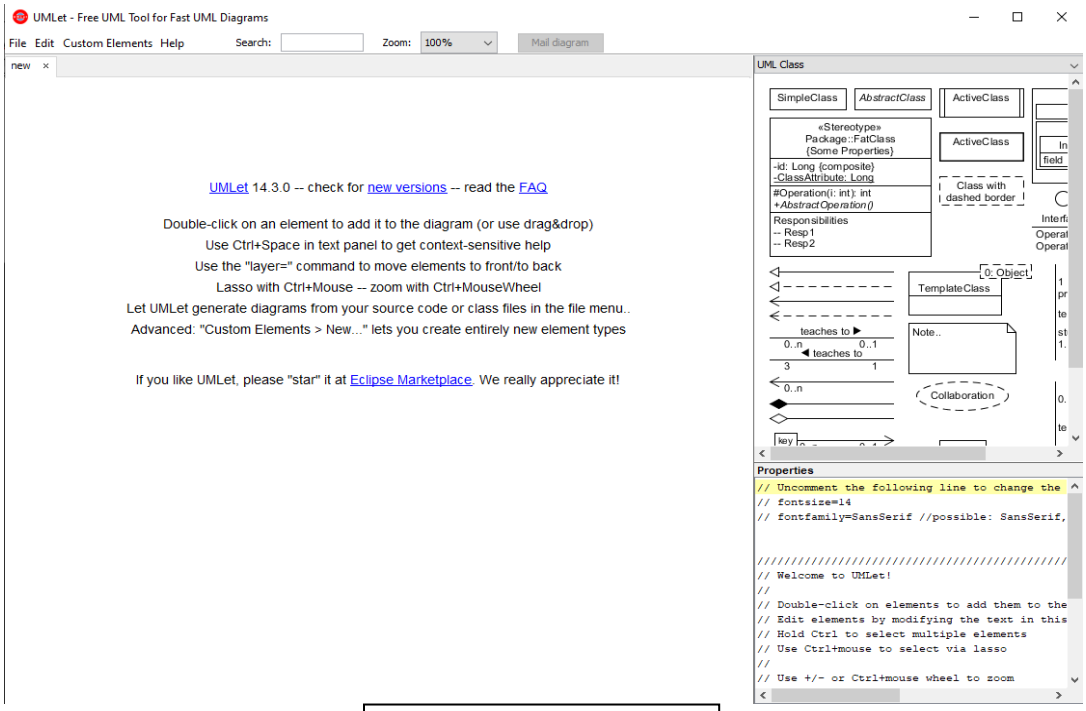


Figure 2.4 UMLet Interface.

This is the interface of the software application, UMLet. UMLet is the app where we will be going to design and create our UML diagram of atm.py

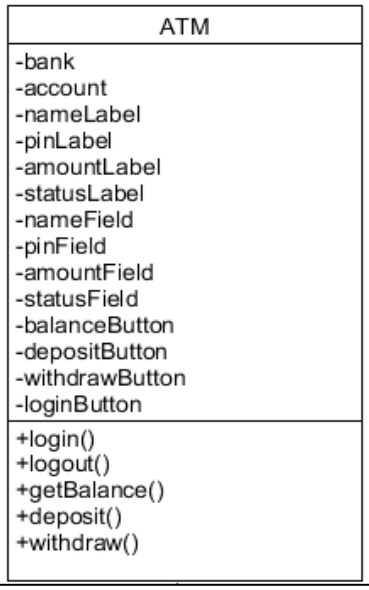


Figure 2.5 UML diagram of atm.py.

The class ATM in the python file atm.py has fourteen 14 private variables and five 5 public functions. The fourteen private variables in the ATM class are named bank, account, nameLabel, pinLabel, amountLabel, statusLabel, nameField, pinField, amountField, statusField, balanceButton, depositButton, withdrawButton, and loginButton. The five public functions in the ATM class are login(), logout(), getBalance(), deposit(), and withdraw().

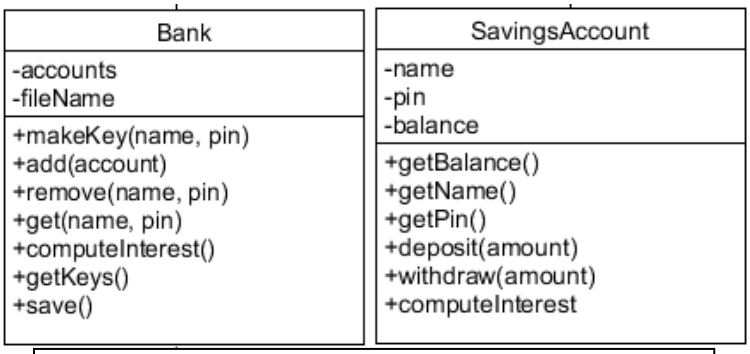


Figure 2.6 UML Diagram of bank.py and savingsaccount.py.

We also created and added the UML diagram of the python files bank.py and savingsaccount.py. This is due to the fact that the main python file which is the atm.py also inherits different classes from other files. In this case, bank.py and savingsaccount.py. For the bank.py file, it has two (2) private variables and seven (7) public functions. The private variables are named accounts and filename. The public functions are makeKey(name, pin), add(account), remove(name, pin), get(name, pin), computeInterest(), getKeys(), and save(). For the savingsaccount.py file, it consists of three (3) private variables and six (6) public functions. The private variables are name, pin, and balance. The public functions are getBalance(), getName(), getPin(), deposit(amount), withdraw(amount), and computeInterest().

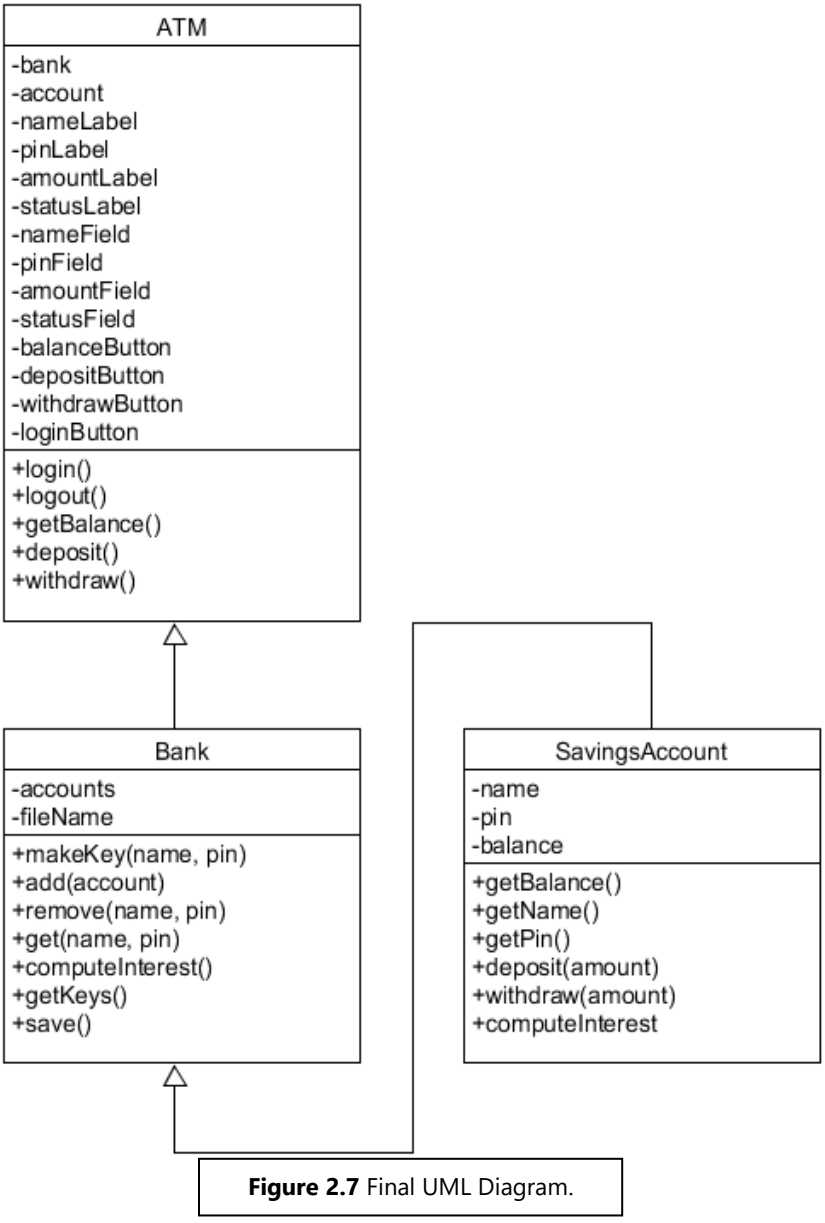


Figure 2.7 Final UML Diagram.

This is the final UML diagram of the python files: atm.py, bank.py, and savingsaccount.py. They are connected by arrows to the class ATM because atm.py inherits the class from bank.py and savingsaccount.py.

PostLab

Machine Problem 4

The ATM program allows a user an indefinite number of attempts to log in. Fix the program so that it displays a popup message that the police will be called after a user has had three successive failures. The program should also disable the login button when this happens.

atm.py

```
"""
File: atm.py
This module defines the ATM class, which provides a window
for bank customers to perform deposits, withdrawals, and check
balances.
"""
import random
from bank import Bank, createBank
from breezypythongui import EasyFrame

class ATM(EasyFrame):
    """Represents an ATM window.
    The window tracks the bank and the current account.
    The current account is None at startup and logout.
    """
    def __init__(self, bank):
        """Initialize the frame and establish the data model."""
        EasyFrame.__init__(self, title = "ATM")
        # Create refernces to the data model.
        self.bank = bank
        self.account = None
        self.error = 0
        # Create and add the widgets to the window."""
        self.nameLabel = self.addLabel(row = 0, column = 0,
                                       text = "Name")
        self.pinLabel = self.addLabel(row = 1, column = 0,
                                      text = "PIN")
        self.amountLabel = self.addLabel(row = 2, column = 0,
                                         text = "Amount")
        self.statusLabel = self.addLabel(row = 3, column = 0,
```

atm.py

```
        self.statusLabel = self.addLabel(row = 3, column = 0,
                                         text = "Status")
        self.nameField = self.addTextField(row = 0, column = 1,
                                           text = "")
        self.pinField = self.addTextField(row = 1, column = 1,
                                          text = "")
        self.amountField = self.addFloatField(row = 2, column = 1,
                                              value = 0.0)
        self.statusField = self.addTextField(row = 3, column = 1,
                                             text = "Welcome to the Bank!",
                                             state = "readonly")
        self.balanceButton = self.addButton(row = 0, column = 2,
                                            text = "Balance",
                                            command = self.getBalance,
                                            state = "disabled")
        self.depositButton = self.addButton(row = 1, column = 2,
                                            text = "Deposit",
                                            command = self.deposit,
                                            state = "disabled")
        self.withdrawButton = self.addButton(row = 2, column = 2,
                                             text = "Withdraw",
                                             command = self.withdraw,
                                             state = "disabled")
        self.loginButton = self.addButton(row = 3, column = 2,
                                          text = "Login",
                                          command = self.login)

    def login(self):
        """Attempts to login the customer.  If successful,
        enables the buttons, including logout."""
```

atm.py

atm.py > ATM > login

```
56
57      def login(self):
58          """Attempts to login the customer.  If successful,
59          enables the buttons, including logout."""
60          """Adding the security measures if log in fails 3 times,
61          the cops will be called and lock the login button"""
62          while True:
63              name = self.nameField.getText()
64              pin = self.pinField.getText()
65              self.account = self.bank.get(name, pin)
66              if self.account:
67                  self.statusField.setText("Hello, " + name + "!")
68                  self.balanceButton["state"] = "normal"
69                  self.depositButton["state"] = "normal"
70                  self.withdrawButton["state"] = "normal"
71                  self.loginButton["text"] = "Logout"
72                  self.loginButton["command"] = self.logout
73                  return False
74
75              elif(self.error >= 3):
76                  self.statusField.setText("The police are called!")
77                  self.loginButton["command"] = None
78                  return True
79
80              elif(self.account==None):
81                  self.statusField.setText("Name and pin not found!!")
82                  self.error += 1
83                  print("Attempt no. " + str(self.error) + " is incorrect, try again")
84                  return True
85
```

atm.py

atm.py > ATM

```
85      def logout(self):
86          """Logs the cusomer out, clears the fields, disables the
87          buttons, and enables login."""
88          self.account = None
89          self.nameField.setText("")
90          self.pinField.setText("")
91          self.amountField.setNumber(0.0)
92          self.statusField.setText("Welcome to the Bank!")
93          self.balanceButton["state"] = "disabled"
94          self.depositButton["state"] = "disabled"
95          self.withdrawButton["state"] = "disabled"
96          self.loginButton["text"] = "Login"
97          self.loginButton["command"] = self.login
98
99      def getBalance(self):
100         """Displays the current balance in the status field."""
101         text = "Balance = $" + str(self.account.getBalance())
102         self.statusField.setText(text)
103
104     def deposit(self):
105         """Attempts a deposit. If not successful, displays
106         error message in statusfield; otherwise, announces
107         success."""
108         amount = self.amountField.getNumber()
109         message = self.account.deposit(amount)
110         if not message:
111             self.statusField.setText("Deposit successful")
112         else:
113            self.statusField.setText(message)
114
```

atm.py

atm.py > ATM

```
113             self.statusField.setText(message)
114
115     def withdraw(self):
116         """Attempts a withdrawal. If not successful, displays
117         error message in statusfield; otherwise, announces
118         success."""
119         amount = self.amountField.getNumber()
120         message = self.account.withdraw(amount)
121         if not message:
122             self.statusField.setText("Withdrawal successful")
123         else:
124             self.statusField.setText(message)
125
126 def main(fileName = None):
127     """Creates the bank with the optional file name,
128     wraps the window around it, and opens the window.
129     Saves the bank when the window closes."""
130     if not fileName:
131         bank = createBank(5)
132     else:
133         bank = Bank(fileName)
134     print(bank)
135     atm = ATM(bank)
136     atm.mainloop()
137
138 if __name__ == "__main__":
139     main()
```

Figure 3.1 Source code of atm.py.

This figure shows the source code for atm.py. It is a program that allows user to check the balance of their account and make some deposits and withdrawal. However, since the program allows a user to log in with an indefinite number of attempts, the program was altered to display a message that the police will be called and disable the login button after three tries of logging in with incorrect pin. The function that was added to change the program in such way is self.error which limits the incorrect login attempts of user to three.

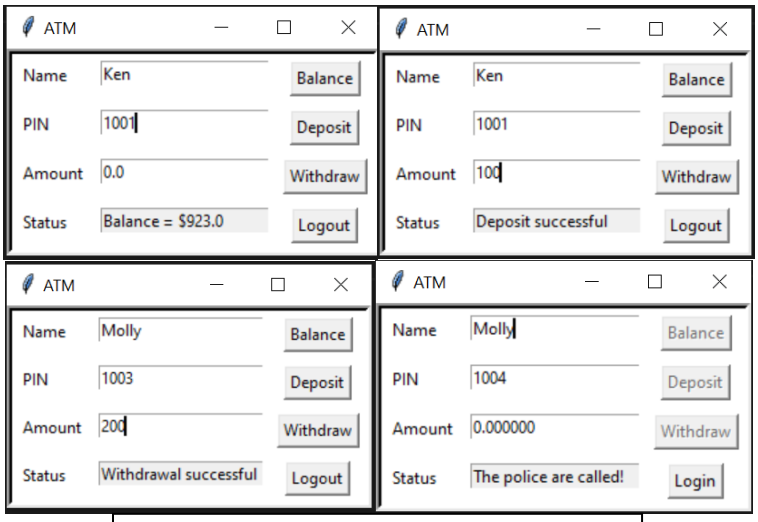


Figure 3.2 Sample run of the ATM program.

This figure shows the 4 different sample runs done to test atm.py. In the first sample run, we tested the balance function of the program using the account of Ken. Then, for the second sample run we tested the deposit function of the program using Ken's account with an amount of 100. For the third sample run, we tested the withdrawal function of the program using the account of Molly with an amount of 200. Lastly, we tested the program to check what will happen after a user had had three successive failures; and as observed from the program, a popup message appeared that the police will be called, and the login button was also disabled.

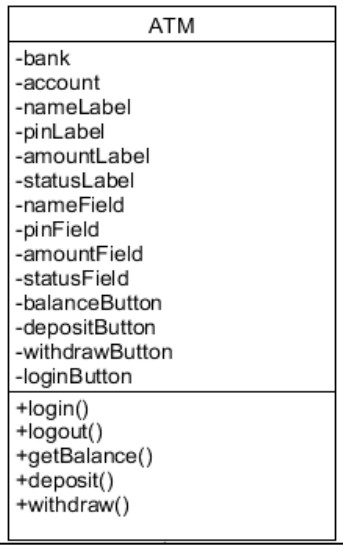
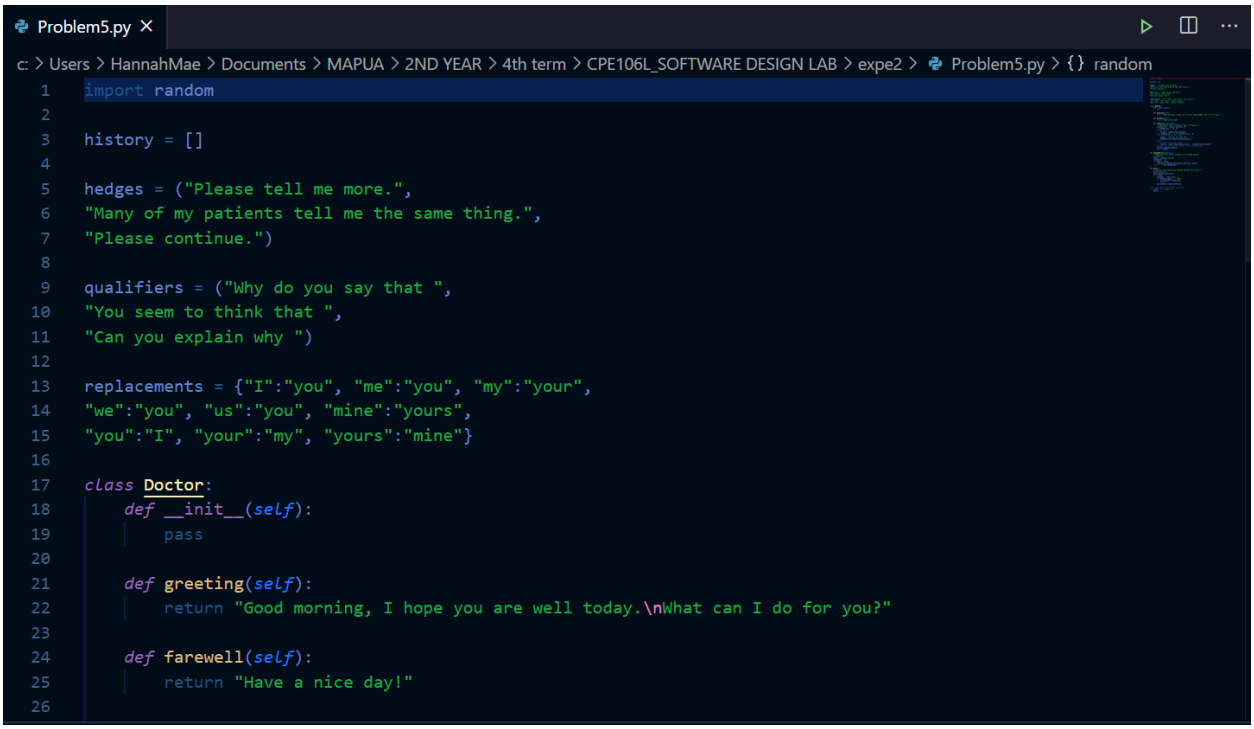


Figure 3.3 Class diagram of the ATM program.

This figure shows the class diagram of the program. The public classes were login() which let users to input their personal details; getBalance() which is used to record balance information; deposit() which let users to deposit; withdraw() which allow users to withdraw; lastly, logout() which ends the program.

Machine Problem 5

The Doctor program described in Chapter 5 combines the data model of a doctor and the operations for handling user interaction. Restructure this program according to the model/view pattern so that these areas of responsibility are assigned to separate sets of classes. The program should include a Doctor class with an interface that allows one to obtain a greeting, a signoff message, and a reply to a patient's string. The rest of the program, in a separate main program module, handles the user's interactions with the Doctor object. You may develop either a terminal based user interface or a GUI.



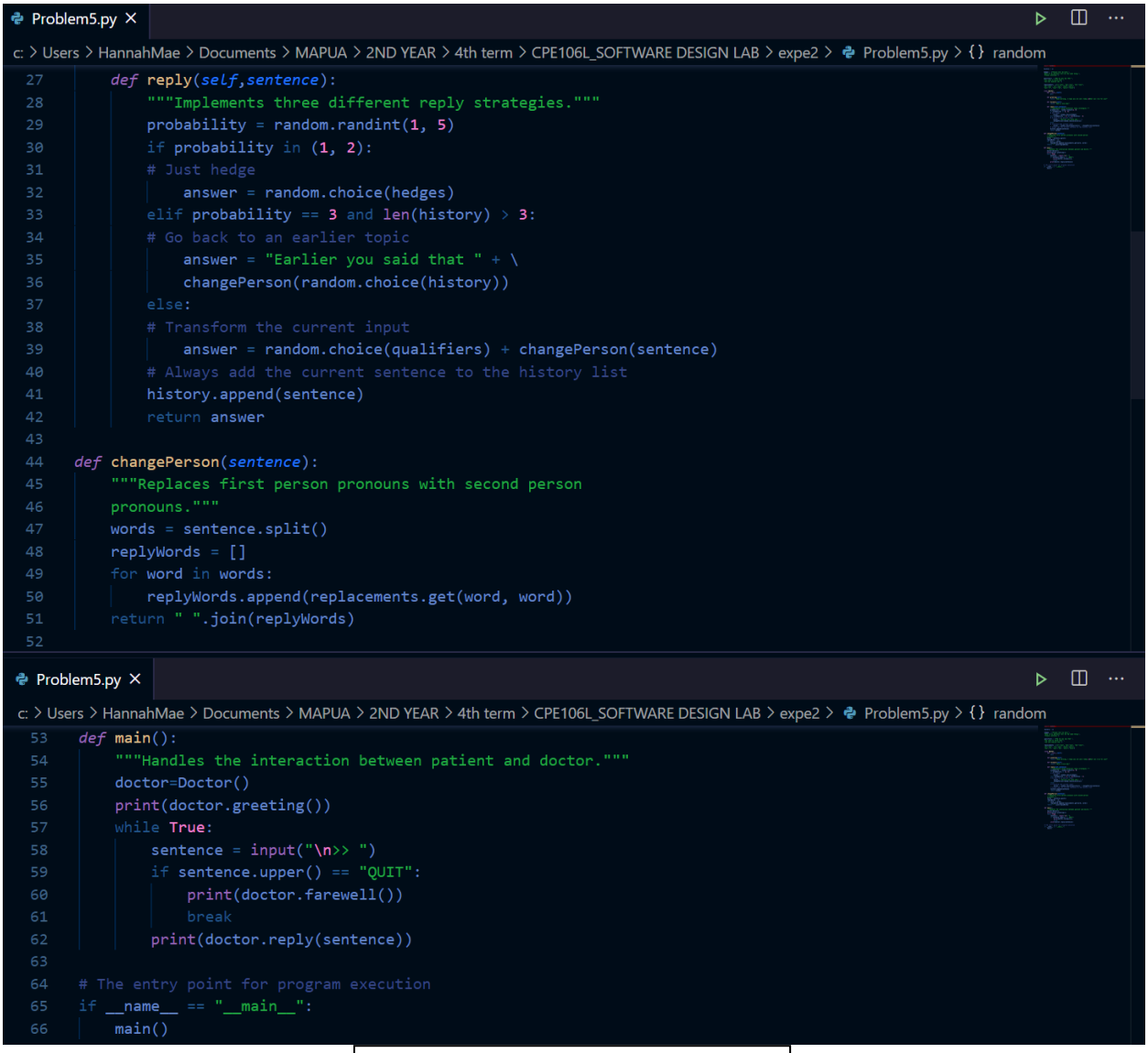


Figure 3.4 Source code of Problem5.py.

Problem 5 is a programming exercise modification of the Doctor program provided in Chapter 5. This program is restructured into two different classes, the Doctor class and the main class. The main class handles the user's interaction with the doctor class whereas the Doctor class could produce a greeting, a signoff message, and a reply to a patient's string input.

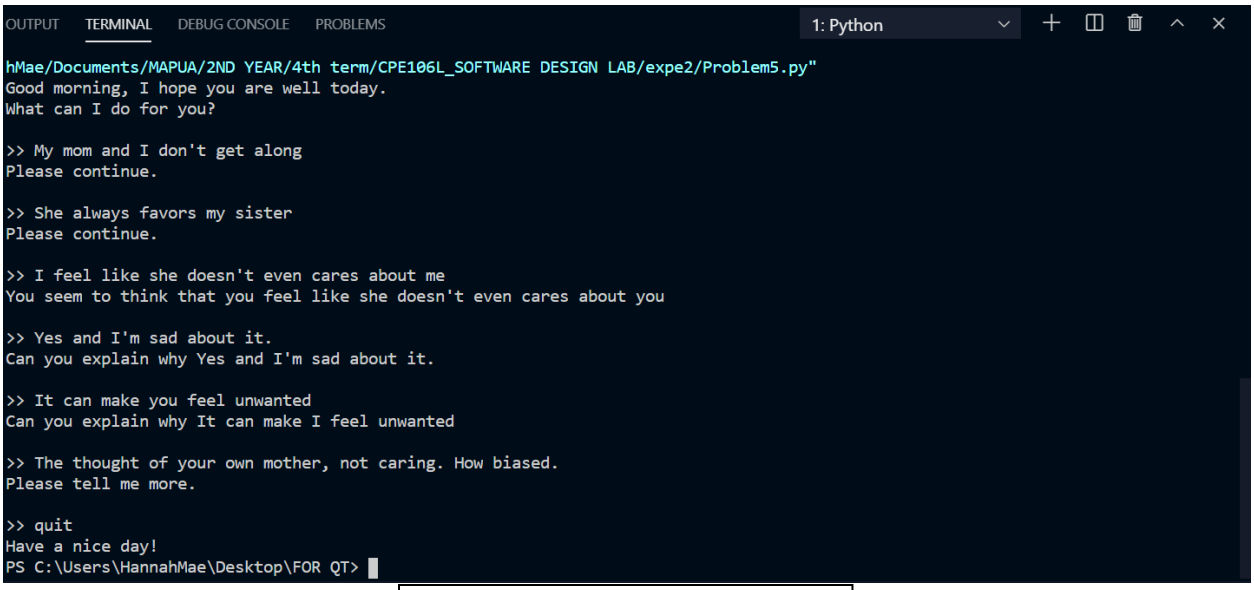


Figure 3.5 Sample run of Problem5.py.

The sample run made for Problem 5, modified Doctor program, is a terminal based interaction between the user and the program. Here, the group has entered multiple inputs to display various replies from the coded program. The program was run successfully as the greetings and replies were all possibly display within the output.

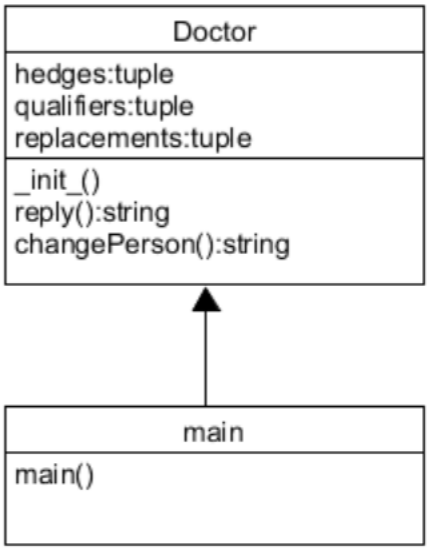


Figure 3.6 Class diagram of the Doctor program.

This figure shows the UML diagram of the modified Doctor program which describes the Doctor class and main class. It can be interpreted from this figure that the main class obtains methods from the Doctor class.