Experiment 2

Saturday, March 7, 2020 3:01 PM

Mapúa University

School of Electrical, Electronics, and Computer Engineering

Experiment 2: Programming Paradigms/Object Oriented Design

CPE106L (Software Design Laboratory)

Gian Karlo Madrid

Rendell Sheen Suliva

Rane Gillian Villanueva

Group: 01

Section: B1

PreLab

- **Readings, Insights, and Reflection**

  Fundamentals of Python: First Programs and Data Structures by Kenneth A. Lumbert; Chapter 9: Design with Classes pg. 294
  The book by Lumbert is all about Python programming. Under Chapter 9 of it tackled Design with Classes. Reading about the fundamentals gives me now more idea how classes and object behave. A class simply has a header and instructions or methods definitions on it. This now enables me to utilize object-oriented-programming towards python which will increase the efficiency and security of following python projects to be done.

  Core Python Programming by Dr. R. Nageswara Rao; Chapter 13: Classes and Object pg. 351
  On this book, it talks about classes and objects in python. As I'm reading the book it has some similarities with Lumbart with I first read but overall it has the same concept. This chapter also teach us how classes and objects work in python as how to create and deal with them. With the knowledge I've read here, this allows me to utilize the object-oriented-programming to make my future program more efficient and more reliable in comes of handling operation and inputs from the user. This will also enhance the security of the program.

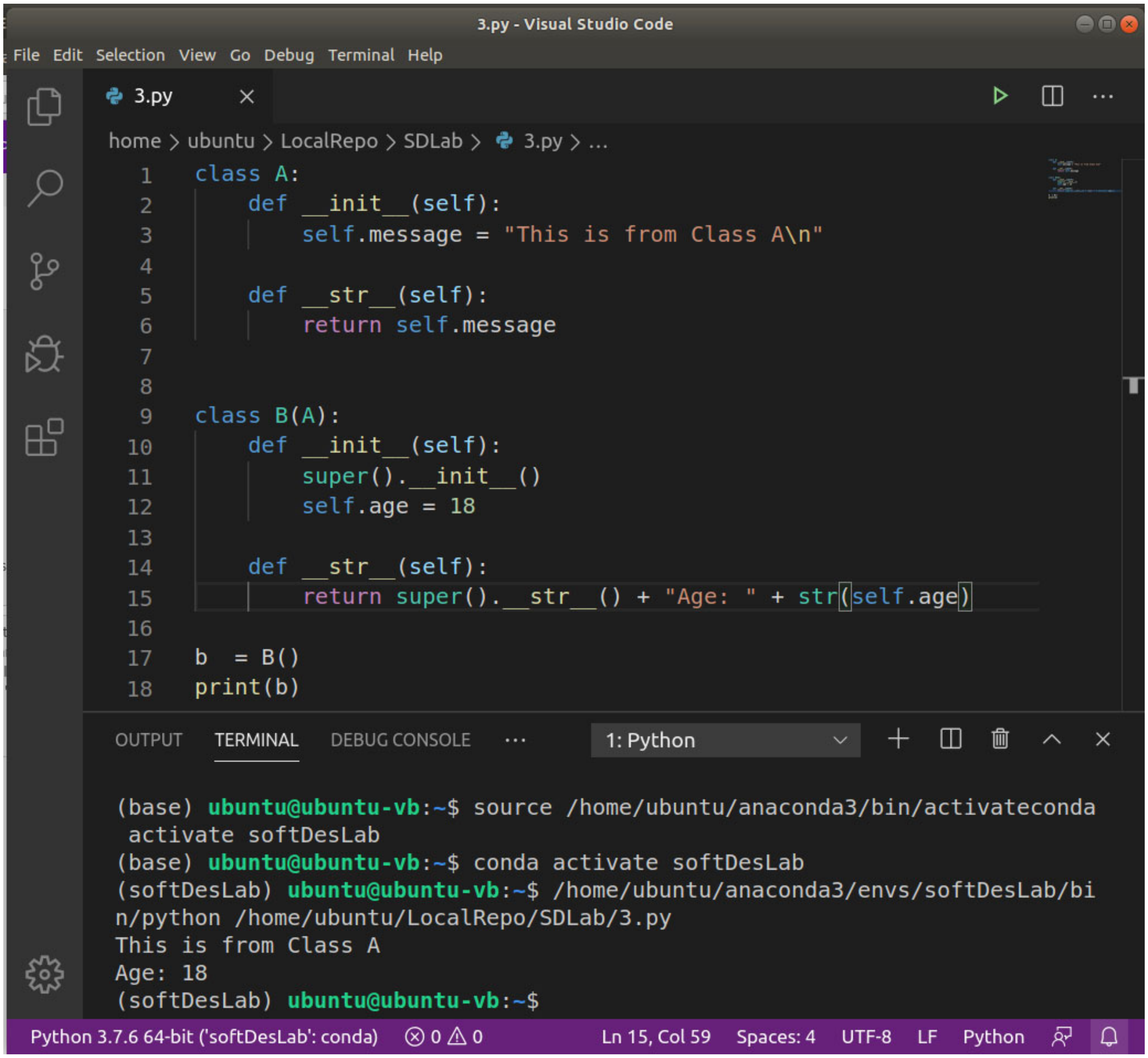
  UML 2 Toolkit by Eriksson, Hans-Erik; Chapter 4: Classes, Objects, and Their Relationships pg.87
  Chapter 4 was all about Classes and Objects, so basically a class in UML has 3 different name compartment which is the top is the name of the class then its attributes and its operations. The classes has attributes which describes the characteristics of the objects and the operations are also known as functions. UML provides flow or outline or a guideline and semantics to create a model of your desired program. Practicing creating an UML diagram would help a programmer to outline what are the classes, and functions and their attributes to be used while coding the program since UML allows the specification of constraints and rules to provide detail on how to implement a software system.

- **Answers to Questions**

1. The benefits of class B inheriting from class A is that it is transitive, if class B inherits from class A, then the subclasses of class B would automatically inherit from class A. It provides reusability of a code which means you would not need to write the same code again.

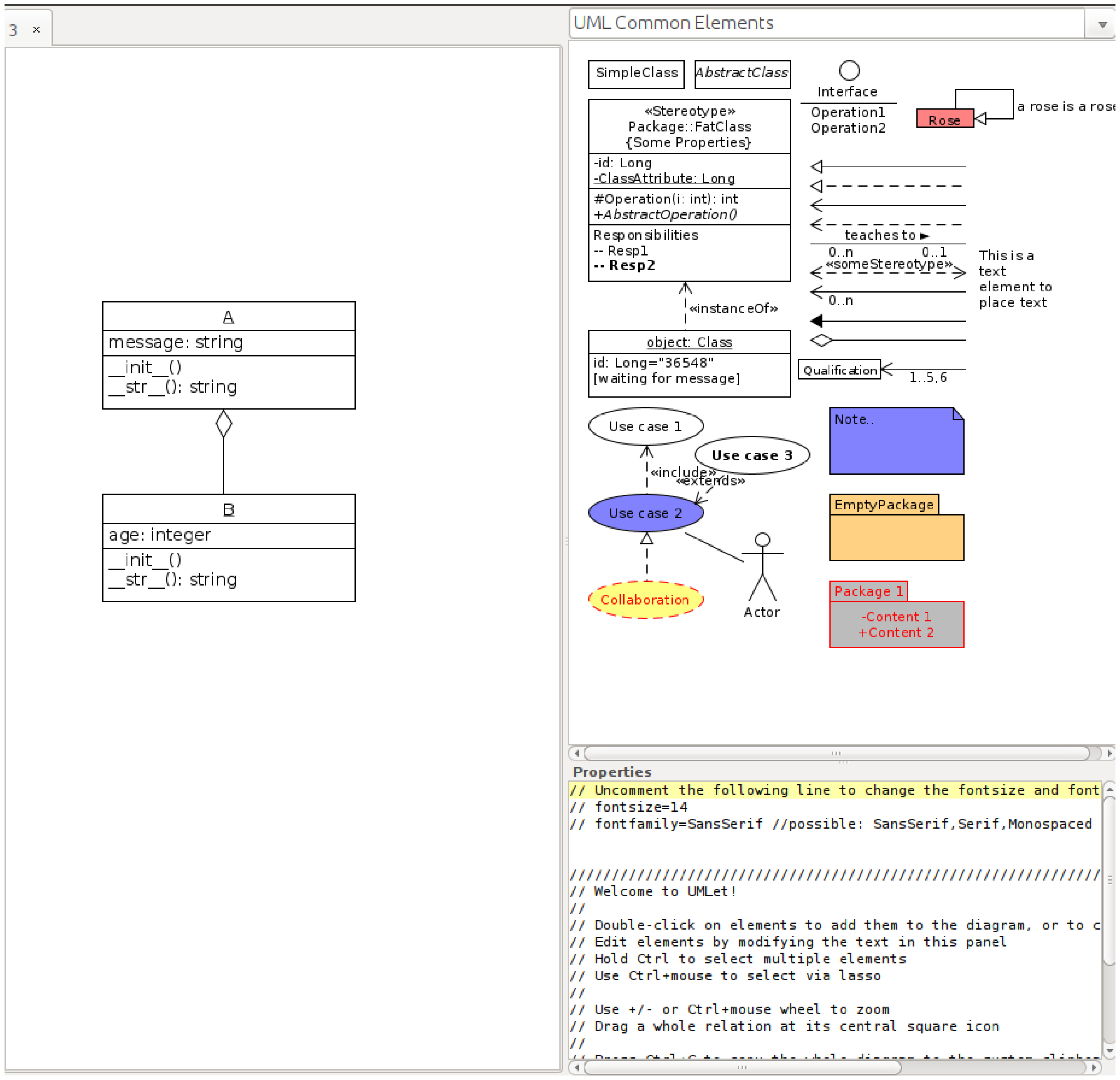
2. The _init_ method is being inherited from the extended class.

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: ".


3.py - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
3.py
home > ubuntu > LocalRepo > SDLab > 3.py > ...
OUTPUT TERMINAL DEBUG CONSOLE ...
1: Python
Python 3.7.6 64-bit ('softDesLab': conda) ⊗ 0 ⚠ 0 Ln 15, Col 59 Spaces: 4 UTF-8 LF Python

```python
class A:
    def __init__(self):
        self.message = "This is from Class A\n"

    def __str__(self):
        return self.message


class B(A):
    def __init__(self):
        super().__init__()
        self.age = 18

    def __str__(self):
        return super().__str__() + "Age: " + str(self.age)

b  = B()
print(b)
```

```
(base) ubuntu@ubuntu-vb:~$ source /home/ubuntu/anaconda3/bin/activateconda
 activate softDesLab
(base) ubuntu@ubuntu-vb:~$ conda activate softDesLab
(softDesLab) ubuntu@ubuntu-vb:~$ /home/ubuntu/anaconda3/envs/softDesLab/bi
n/python /home/ubuntu/LocalRepo/SDLab/3.py
This is from Class A
Age: 18
(softDesLab) ubuntu@ubuntu-vb:~$
```

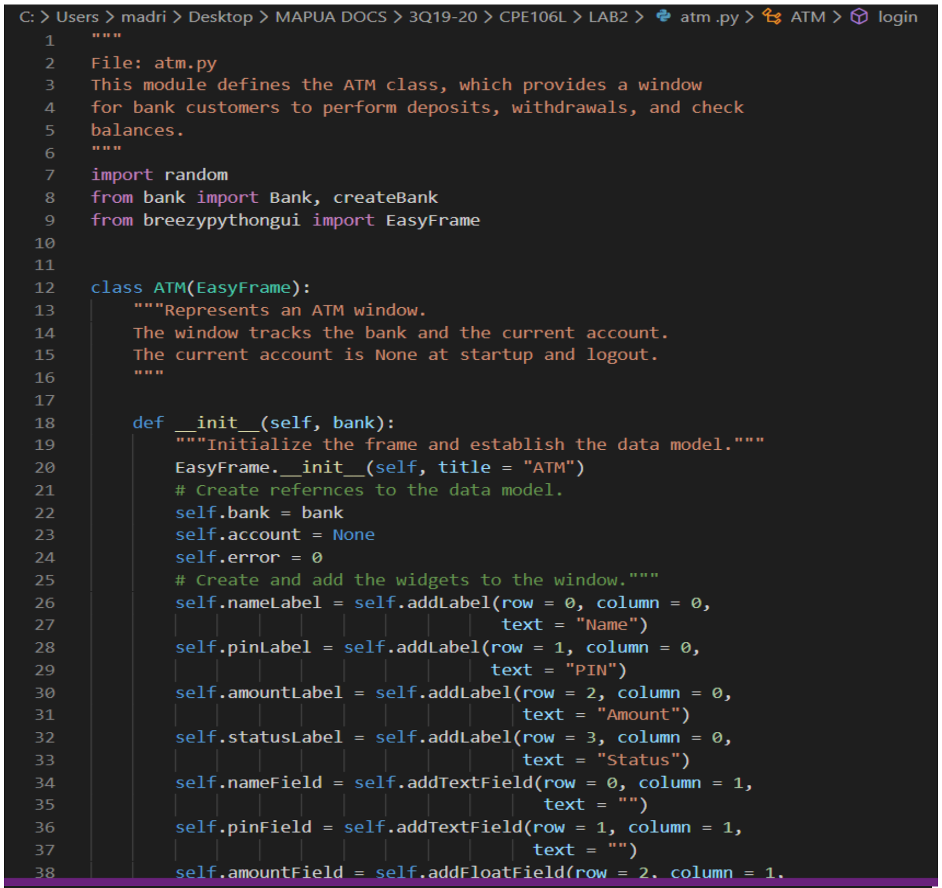


```
// Uncomment the following line to change the fontsize and font
// fontsize=14
// fontfamily=SansSerif //possible: SansSerif,Serif,Monospaced


//////////////////////////////////////////////////////////////
// Welcome to UMLet!
//
// Double-click on elements to add them to the diagram, or to c
// Edit elements by modifying the text in this panel
// Hold Ctrl to select multiple elements
// Use Ctrl+mouse to select via lasso
//
// Use +/- or Ctrl+mouse wheel to zoom
// Drag a whole relation at its central square icon
//
```

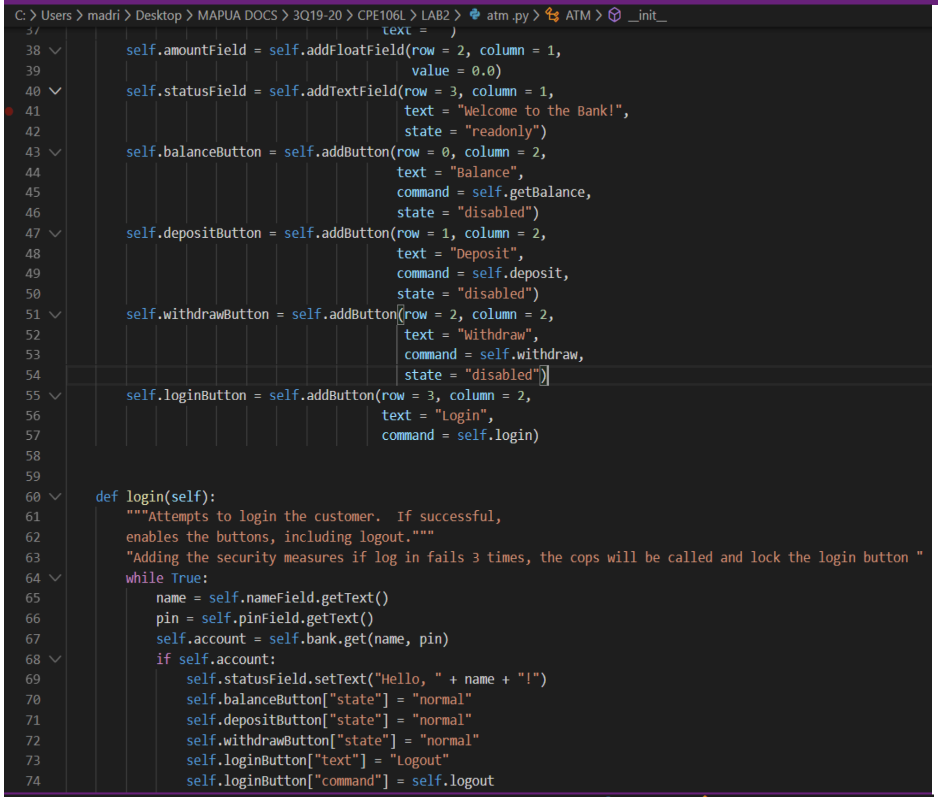
InLab

- **Objectives**
  - **Learn to interpret and make a UML Diagram**
  - **Learn to modify programs to add additional features and security**
  - **Learn the concept of classes**
  - **Learn the concept of inheritance**
  - **Learn the attributes and behavior of a class object**

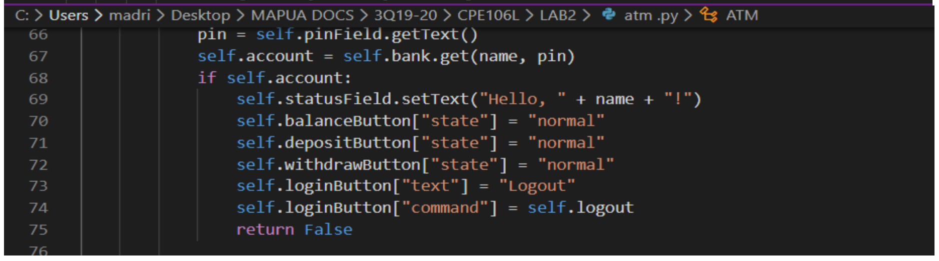
- **Source Code of Machine Problem 4 and its UML diagram:**


C: > Users > madri > Desktop > MAPUA DOCS > 3Q19-20 > CPE106L > LAB2 > atm.py > ATM > login

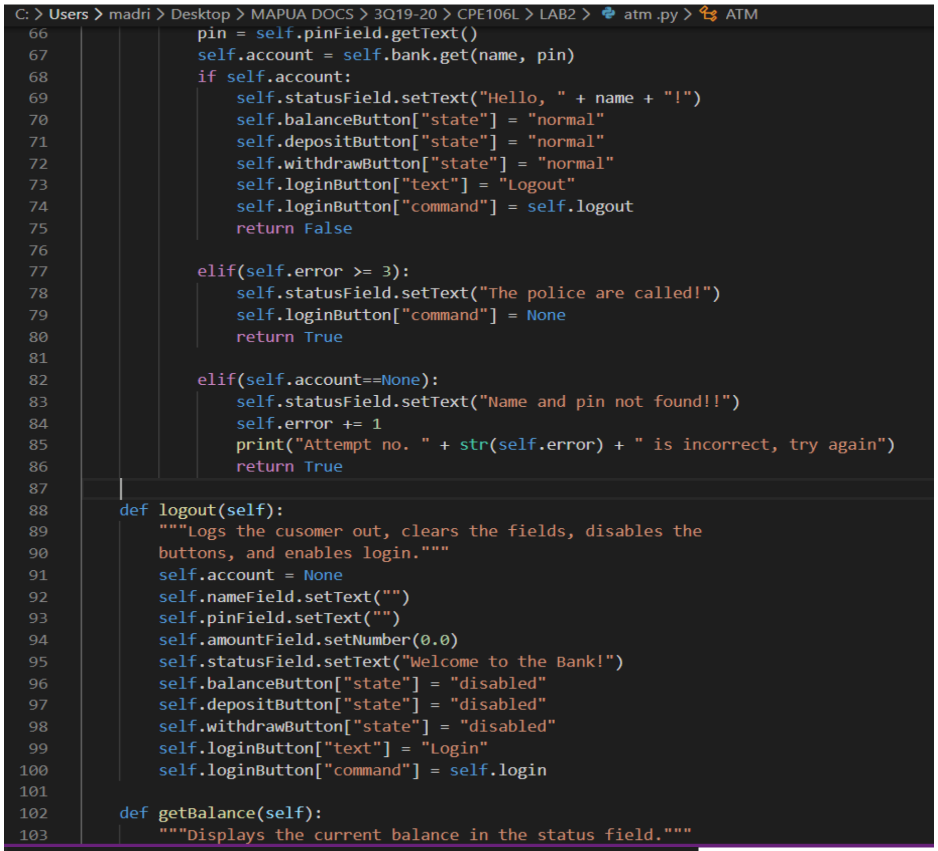
```python
"""
File: atm.py
This module defines the ATM class, which provides a window
for bank customers to perform deposits, withdrawals, and check
balances.
"""
import random
from bank import Bank, createBank
from breezypythongui import EasyFrame


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,
                                         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,
```


C: > Users > madri > Desktop > MAPUA DOCS > 3Q19-20 > CPE106L > LAB2 > atm.py > ATM > _init_

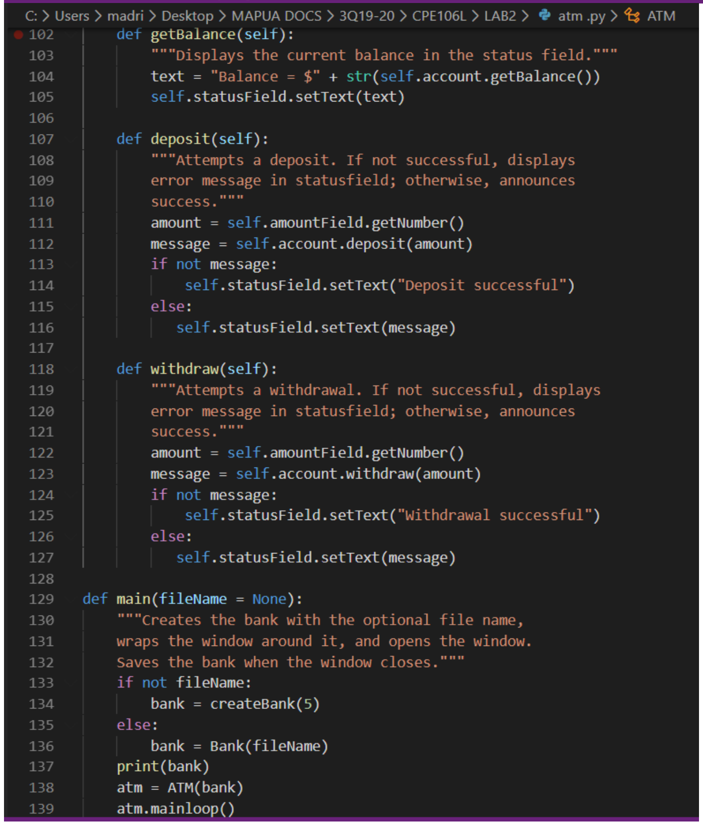
```python
        self.amountField = self.addFloatField(row = 2, column = 1,
                                              value = 0.0)
        self.statusField = self.addTextField(row = 3, column = 1,
                                             text = "Welcome to the Bank!",
                                             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."""
        "Adding the security measures if log in fails 3 times, the cops will be called and lock the login button "
        while True:
            name = self.nameField.getText()
            pin = self.pinField.getText()
            self.account = self.bank.get(name, pin)
            if self.account:
                self.statusField.setText("Hello, " + name + "!")
                self.balanceButton["state"] = "normal"
                self.depositButton["state"] = "normal"
                self.withdrawButton["state"] = "normal"
                self.loginButton["text"] = "Logout"
                self.loginButton["command"] = self.logout
```


C: > Users > madri > Desktop > MAPUA DOCS > 3Q19-20 > CPE106L > LAB2 > atm.py > ATM

```python
            pin = self.pinField.getText()
            self.account = self.bank.get(name, pin)
            if self.account:
                self.statusField.setText("Hello, " + name + "!")
                self.balanceButton["state"] = "normal"
                self.depositButton["state"] = "normal"
                self.withdrawButton["state"] = "normal"
                self.loginButton["text"] = "Logout"
                self.loginButton["command"] = self.logout
                return False
```

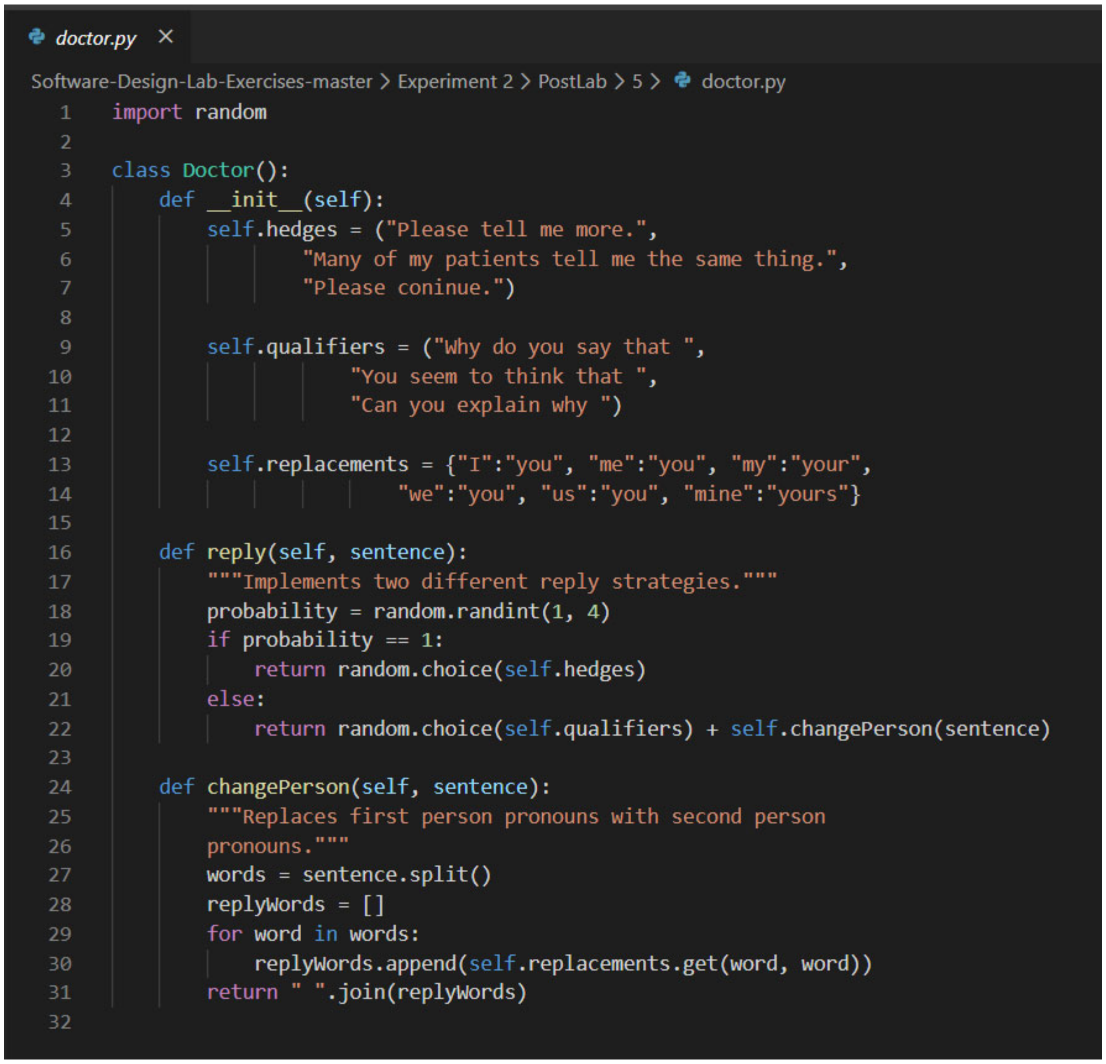

C: > Users > madri > Desktop > MAPUA DOCS > 3Q19-20 > CPE106L > LAB2 > atm .py > ATM

```python
            pin = self.pinField.getText()
            self.account = self.bank.get(name, pin)
            if self.account:
                self.statusField.setText("Hello, " + name + "!")
                self.balanceButton["state"] = "normal"
                self.depositButton["state"] = "normal"
                self.withdrawButton["state"] = "normal"
                self.loginButton["text"] = "Logout"
                self.loginButton["command"] = self.logout
                return False

            elif(self.error >= 3):
                self.statusField.setText("The police are called!")
                self.loginButton["command"] = None
                return True

            elif(self.account==None):
                self.statusField.setText("Name and pin not found!!")
                self.error += 1
                print("Attempt no. " + str(self.error) + " is incorrect, try again")
                return True

    def logout(self):
        """Logs the cusomer out, clears the fields, disables the
        buttons, and enables login."""
        self.account = None
        self.nameField.setText("")
        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."""
```


C: > Users > madri > Desktop > MAPUA DOCS > 3Q19-20 > CPE106L > LAB2 > atm .py > ATM

```python
    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,
    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()
```

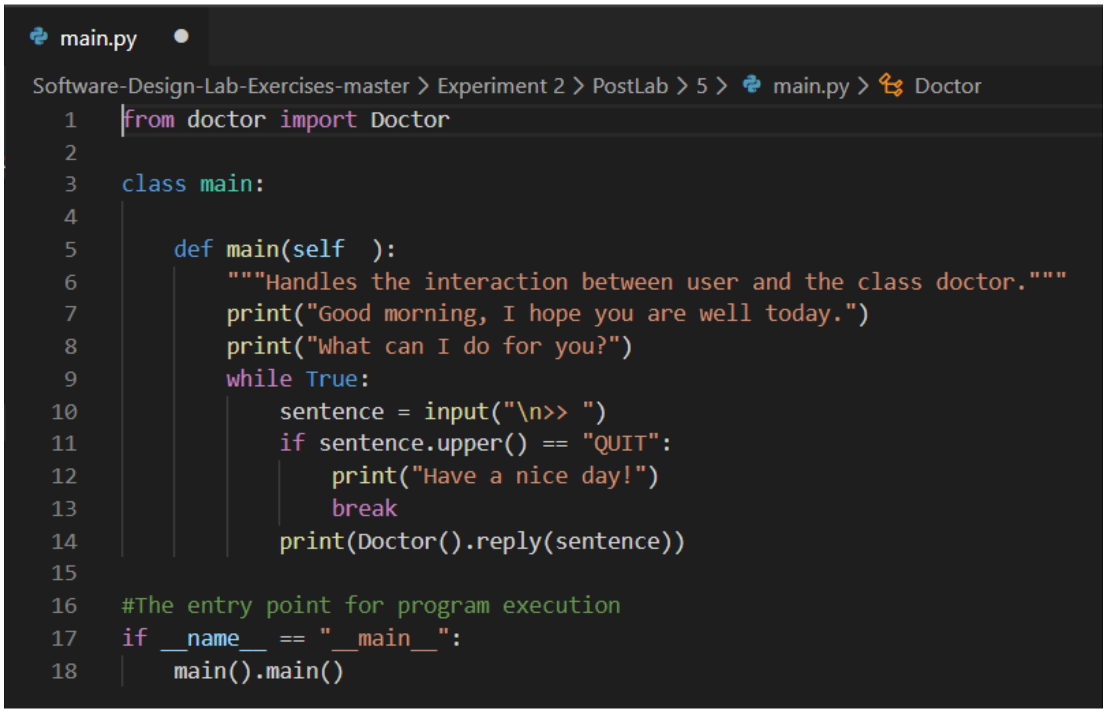
- **Source code of Programming Exercise 5:**



doctor.py ×

Software-Design-Lab-Exercises-master > Experiment 2 > PostLab > 5 > doctor.py

```python
import random

class Doctor():
    def __init__(self):
        self.hedges = ("Please tell me more.",
                "Many of my patients tell me the same thing.",
                "Please coninue.")

        self.qualifiers = ("Why do you say that ",
                    "You seem to think that ",
                    "Can you explain why ")

        self.replacements = {"I":"you", "me":"you", "my":"your",
                        "we":"you", "us":"you", "mine":"yours"}

    def reply(self, sentence):
        """Implements two different reply strategies."""
        probability = random.randint(1, 4)
        if probability == 1:
            return random.choice(self.hedges)
        else:
            return random.choice(self.qualifiers) + self.changePerson(sentence)

    def changePerson(self, sentence):
        """Replaces first person pronouns with second person
        pronouns."""
        words = sentence.split()
        replyWords = []
        for word in words:
            replyWords.append(self.replacements.get(word, word))
        return " ".join(replyWords)

```

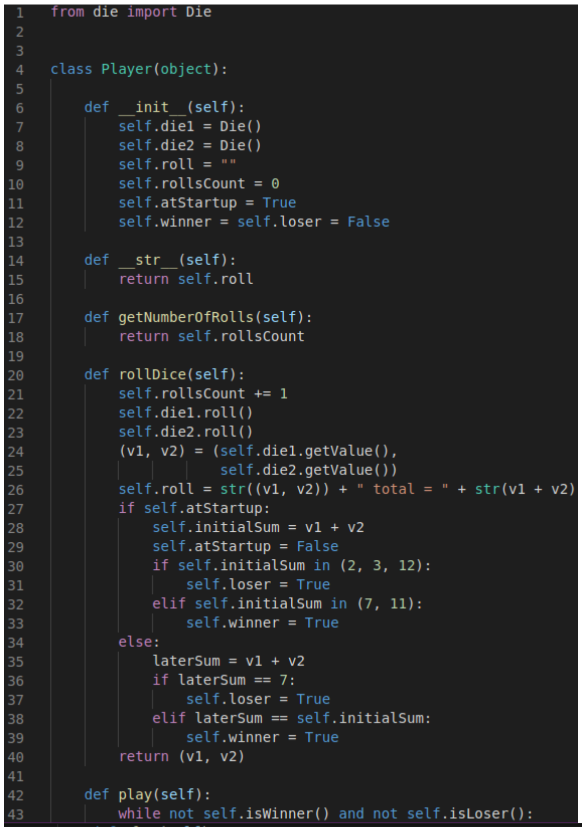


main.py ●

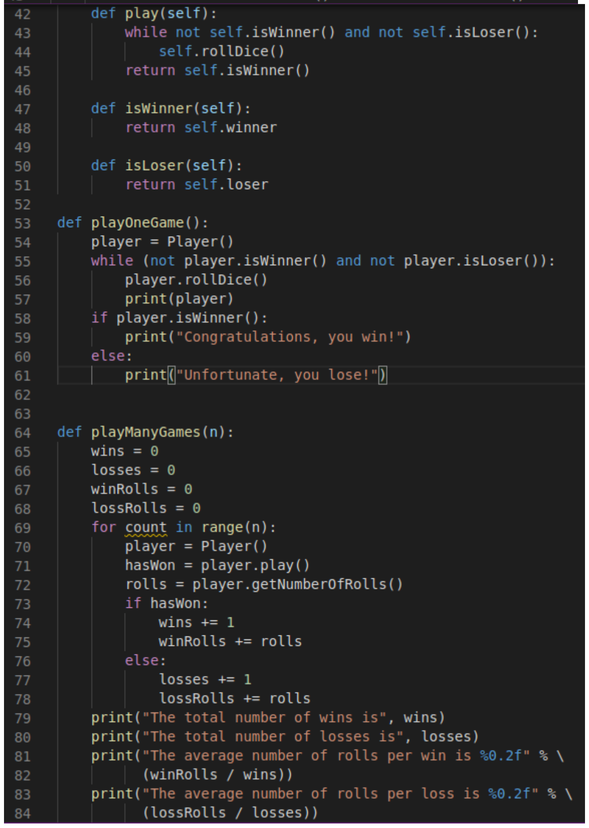
Software-Design-Lab-Exercises-master > Experiment 2 > PostLab > 5 > main.py > Doctor

```python
from doctor import Doctor

class main:

    def main(self  ):
        """Handles the interaction between user and the class doctor."""
        print("Good morning, I hope you are well today.")
        print("What can I do for you?")
        while True:
            sentence = input("\n>> ")
            if sentence.upper() == "QUIT":
                print("Have a nice day!")
                break
            print(Doctor().reply(sentence))

#The entry point for program execution
if __name__ == "__main__":
    main().main()
```

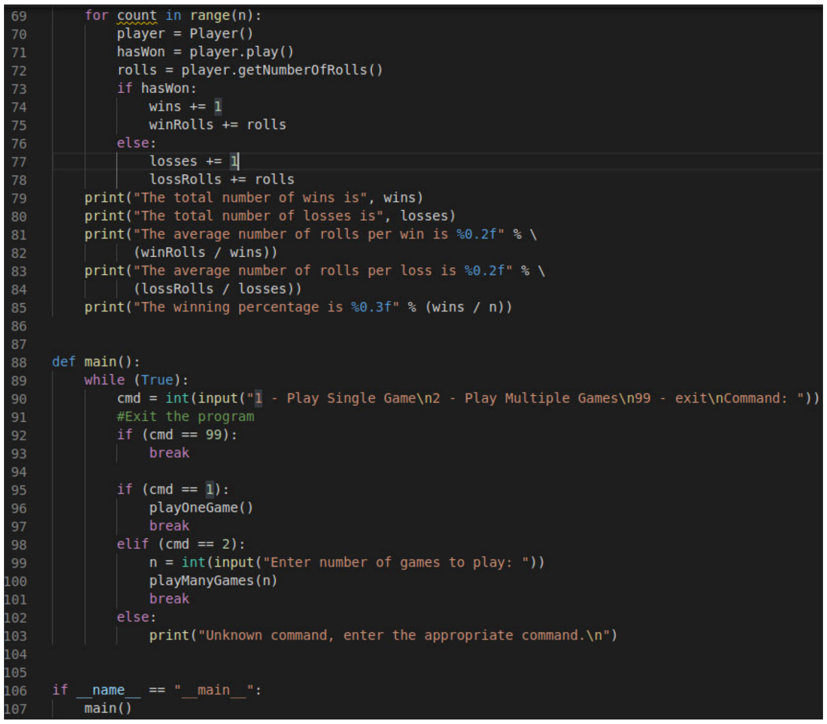
-
- **Source Code of Machine Problem 6 and its UML diagram:**



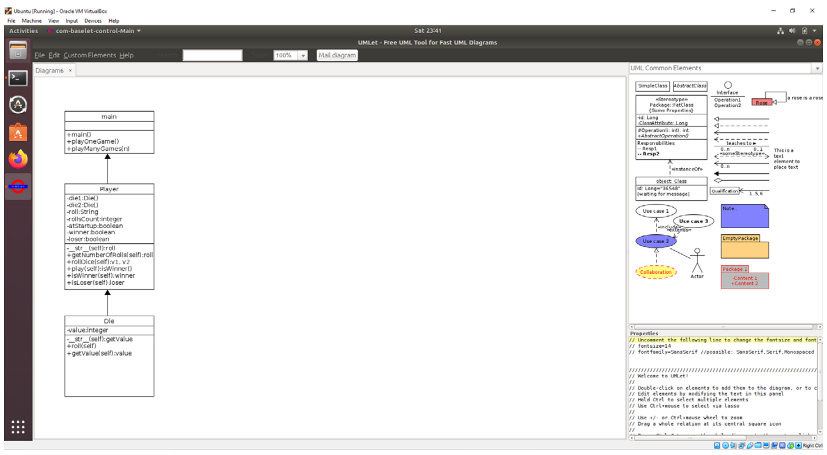
```python
from die import Die


class Player(object):

    def __init__(self):
        self.die1 = Die()
        self.die2 = Die()
        self.roll = ""
        self.rollsCount = 0
        self.atStartup = True
        self.winner = self.loser = False

    def __str__(self):
        return self.roll

    def getNumberOfRolls(self):
        return self.rollsCount

    def rollDice(self):
        self.rollsCount += 1
        self.die1.roll()
        self.die2.roll()
        (v1, v2) = (self.die1.getValue(),
                    self.die2.getValue())
        self.roll = str((v1, v2)) + " total = " + str(v1 + v2)
        if self.atStartup:
            self.initialSum = v1 + v2
            self.atStartup = False
            if self.initialSum in (2, 3, 12):
                self.loser = True
            elif self.initialSum in (7, 11):
                self.winner = True
        else:
            laterSum = v1 + v2
            if laterSum == 7:
                self.loser = True
            elif laterSum == self.initialSum:
                self.winner = True
        return (v1, v2)

    def play(self):
        while not self.isWinner() and not self.isLoser():
```



```python
    def play(self):
        while not self.isWinner() and not self.isLoser():
            self.rollDice()
        return self.isWinner()

    def isWinner(self):
        return self.winner

    def isLoser(self):
        return self.loser

def playOneGame():
    player = Player()
    while (not player.isWinner() and not player.isLoser()):
        player.rollDice()
        print(player)
    if player.isWinner():
        print("Congratulations, you win!")
    else:
        print("Unfortunate, you lose!")


def playManyGames(n):
    wins = 0
    losses = 0
    winRolls = 0
    lossRolls = 0
    for count in range(n):
        player = Player()
        hasWon = player.play()
        rolls = player.getNumberOfRolls()
        if hasWon:
            wins += 1
            winRolls += rolls
        else:
            losses += 1
            lossRolls += rolls
    print("The total number of wins is", wins)
    print("The total number of losses is", losses)
    print("The average number of rolls per win is %0.2f" % \
          (winRolls / wins))
    print("The average number of rolls per loss is %0.2f" % \
          (lossRolls / losses))
```

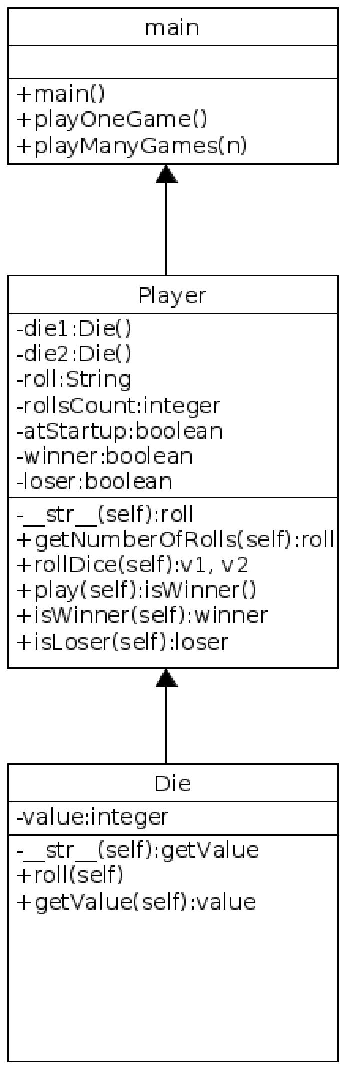


```python
    for count in range(n):
        player = Player()
        hasWon = player.play()
        rolls = player.getNumberOfRolls()
        if hasWon:
            wins += 1
            winRolls += rolls
        else:
            losses += 1
            lossRolls += rolls
    print("The total number of wins is", wins)
    print("The total number of losses is", losses)
    print("The average number of rolls per win is %0.2f" % \
          (winRolls / wins))
    print("The average number of rolls per loss is %0.2f" % \
          (lossRolls / losses))
    print("The winning percentage is %0.3f" % (wins / n))


def main():
    while (True):
        cmd = int(input("1 - Play Single Game\n2 - Play Multiple Games\n99 - exit\nCommand: "))
        #Exit the program
        if (cmd == 99):
            break

        if (cmd == 1):
            playOneGame()
            break
        elif (cmd == 2):
            n = int(input("Enter number of games to play: "))
            playManyGames(n)
            break
        else:
            print("Unknown command, enter the appropriate command.\n")


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



```
main
+main()
+playOneGame()
+playManyGames(n)

Player
-die1:Die()
-die2:Die()
-roll:String
-rollsCount:integer
-atStartup:boolean
-winner:boolean
-loser:boolean
-__str__(self):roll
+getNumberOfRolls(self):roll
+rollDice(self):v1, v2
+play(self):isWinner()
+isWinner(self):winner
+isLoser(self):loser

Die
-value:integer
-__str__(self):getValue
+roll(self)
+getValue(self):value
```

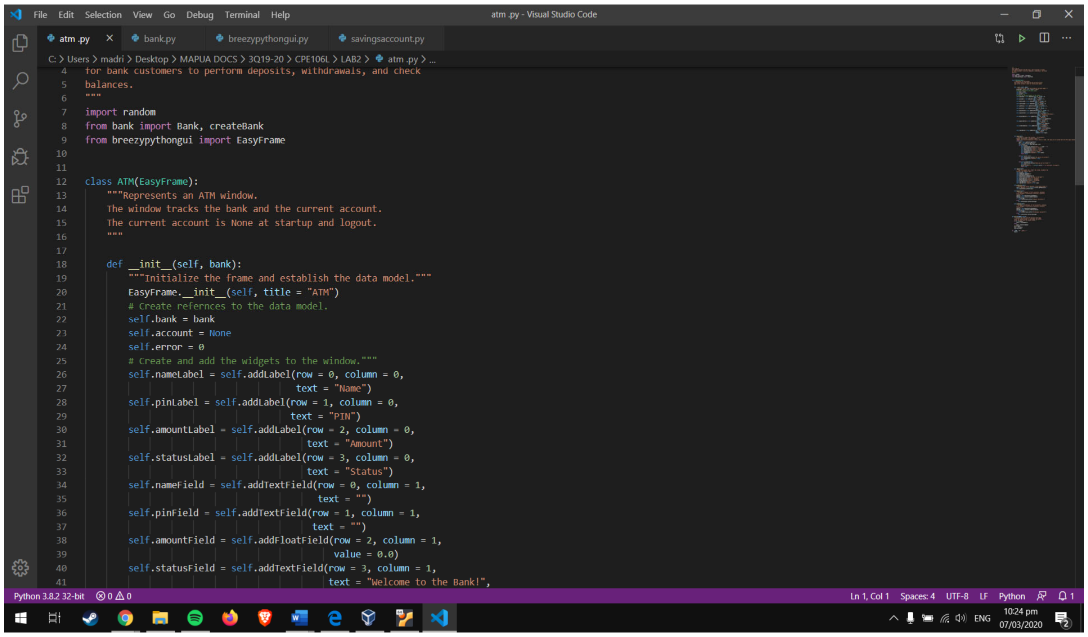
```
// Uncomment the following line to change the fontsize and font
// fontsize=14
// fontfamily=SansSerif //possible: SansSerif,Serif,Monospaced


//////////////////////////////////////////////////////////////
// Welcome to UMLet!
//
// Double-click on elements to add them to the diagram, or to c
// Edit elements by modifying the text in this panel
// Hold Ctrl to select multiple elements
// Use Ctrl+mouse to select via lasso
//
// Use +/- or Ctrl+mouse wheel to zoom
// Drag a whole relation at its central square icon
//
```



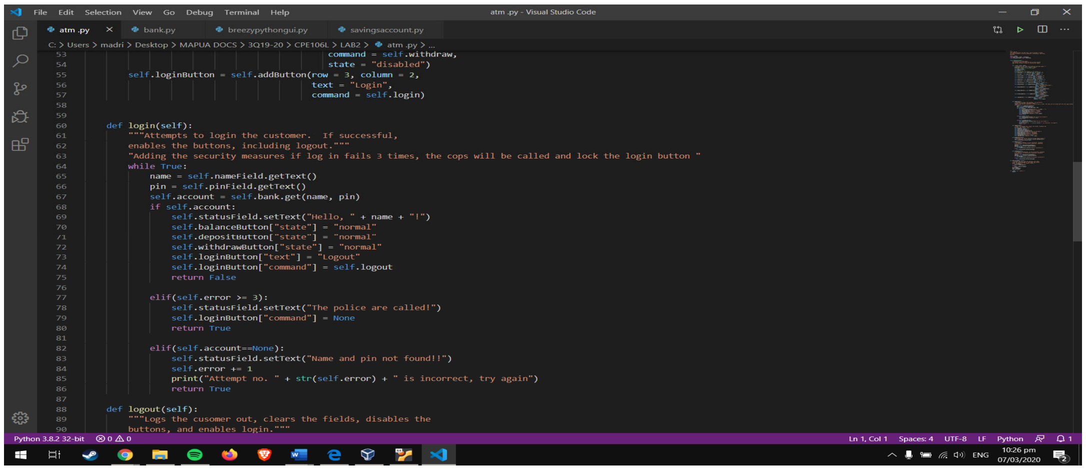
PostLab

- **Machine Problems**
  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.

     Source Code:



```python
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,
                                         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!",
```

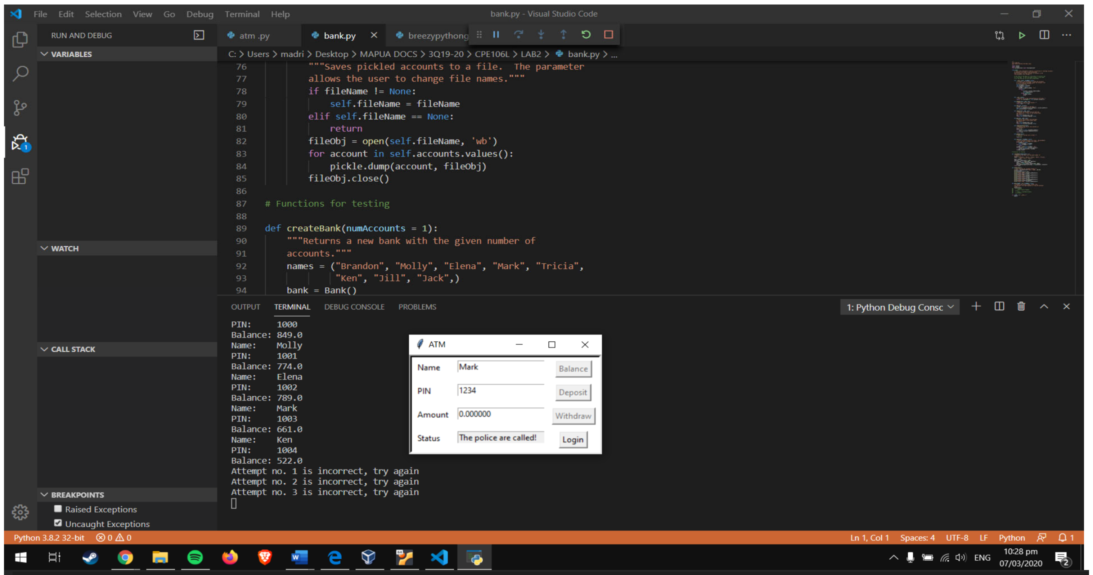


```python
                                             command = self.withdraw,
                                             state = "disabled")
        self.loginButton = self.addButton(row = 3, column = 2,
                                          text = "Login",
                                          command = self.login)


    def login(self):
        """Attempts to login the customer.  If successful,
        enables the buttons, including logout."""
        "Adding the security measures if log in fails 3 times, the cops will be called and lock the login button "
        while True:
            name = self.nameField.getText()
            pin = self.pinField.getText()
            self.account = self.bank.get(name, pin)
            if self.account:
                self.statusField.setText("Hello, " + name + "!")
                self.balanceButton["state"] = "normal"
                self.depositButton["state"] = "normal"
                self.withdrawButton["state"] = "normal"
                self.loginButton["text"] = "Logout"
                self.loginButton["command"] = self.logout
                return False

            elif(self.error >= 3):
                self.statusField.setText("The police are called!")
                self.loginButton["command"] = None
                return True

            elif(self.account==None):
                self.statusField.setText("Name and pin not found!!")
                self.error += 1
                print("Attempt no. " + str(self.error) + " is incorrect, try again")
                return True

    def logout(self):
        """Logs the cusomer out, clears the fields, disables the
        buttons, and enables login."""
```

Figures 4.1

Machine Problem 4 is about recreating the ATM program which allows the user to attempt multiple times be it correct input or not. The program should be able to control the limit of attempts into 3 and call the police if the user failed to log in correctly. So importing from the class bank and breezypythongui allowed the program to have a GUI and data so that the program ATM will run properly. A new function was added to the previous source code which is the self.error that limits the incorrect attempts of the users into 3 and if so, the program will then call the police and it will lock the login button since it was included in the elif statement that if the error of attempts is equal or greater than 3 the self.loginButton[] will return None and be unresponsive, which means the user cannot login any accounts unless the user will re-run the program s

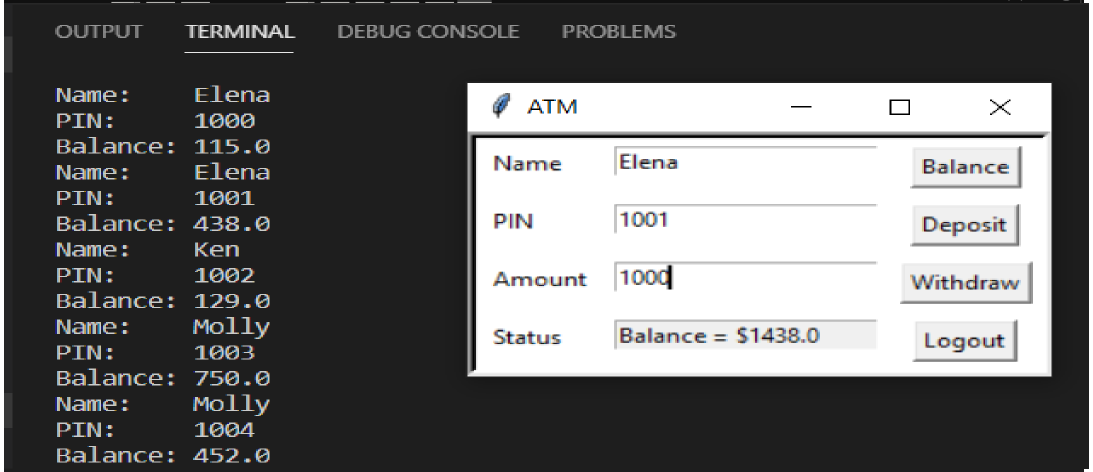
Sample Output:



```python
        """Saves pickled accounts to a file.  The parameter
        allows the user to change file names."""
        if fileName != None:
            self.fileName = fileName
        elif self.fileName == None:
            return
        fileObj = open(self.fileName, 'wb')
        for account in self.accounts.values():
            pickle.dump(account, fileObj)
        fileObj.close()

# Functions for testing

def createBank(numAccounts = 1):
    """Returns a new bank with the given number of
    accounts."""
    names = ("Brandon", "Molly", "Elena", "Mark", "Tricia",
             "Ken", "Jill", "Jack",)
    bank = Bank()
```

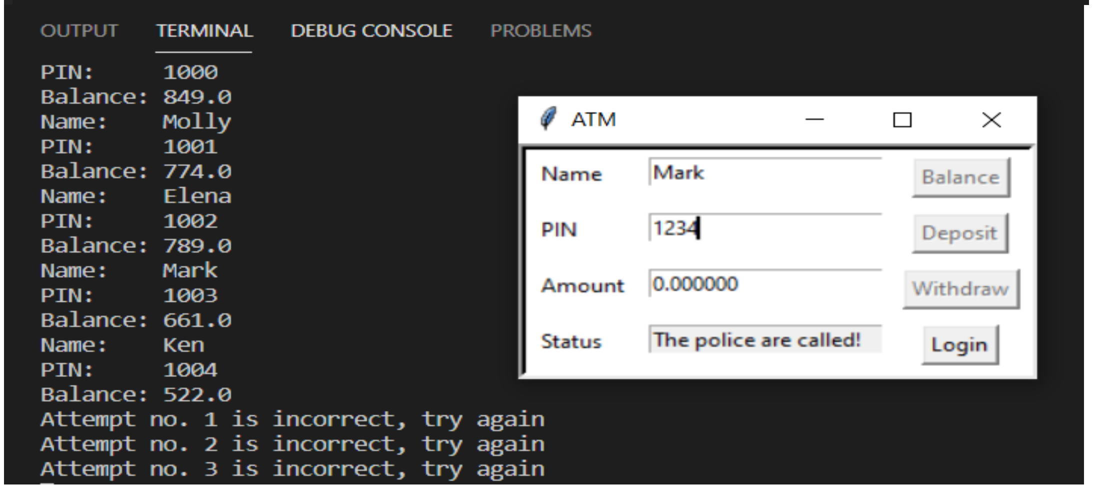
```
PIN:     1000
Balance: 849.0
Name:    Molly
PIN:     1001
Balance: 774.0
Name:    Elena
PIN:     1002
Balance: 789.0
Name:    Mark
PIN:     1003
Balance: 661.0
Name:    Ken
PIN:     1004
Balance: 522.0
Attempt no. 1 is incorrect, try again
Attempt no. 2 is incorrect, try again
Attempt no. 3 is incorrect, try again
```

ATM — Name: Mark; PIN: 1234; Amount: 0.000000; Status: The police are called!



```
Name:    Elena
PIN:     1000
Balance: 115.0
Name:    Elena
PIN:     1001
Balance: 438.0
Name:    Ken
PIN:     1002
Balance: 129.0
Name:    Molly
PIN:     1003
Balance: 750.0
Name:    Molly
PIN:     1004
Balance: 452.0
```

ATM — Name: Elena; PIN: 1001; Amount: 1000; Status: Balance = $1438.0

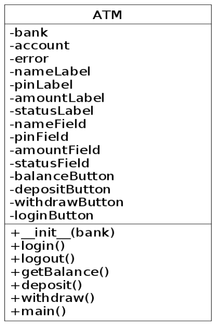


```
PIN:     1000
Balance: 849.0
Name:    Molly
PIN:     1001
Balance: 774.0
Name:    Elena
PIN:     1002
Balance: 789.0
Name:    Mark
PIN:     1003
Balance: 661.0
Name:    Ken
PIN:     1004
Balance: 522.0
Attempt no. 1 is incorrect, try again
Attempt no. 2 is incorrect, try again
Attempt no. 3 is incorrect, try again
```

ATM — Name: Mark; PIN: 1234; Amount: 0.000000; Status: The police are called!

Figures 4.2

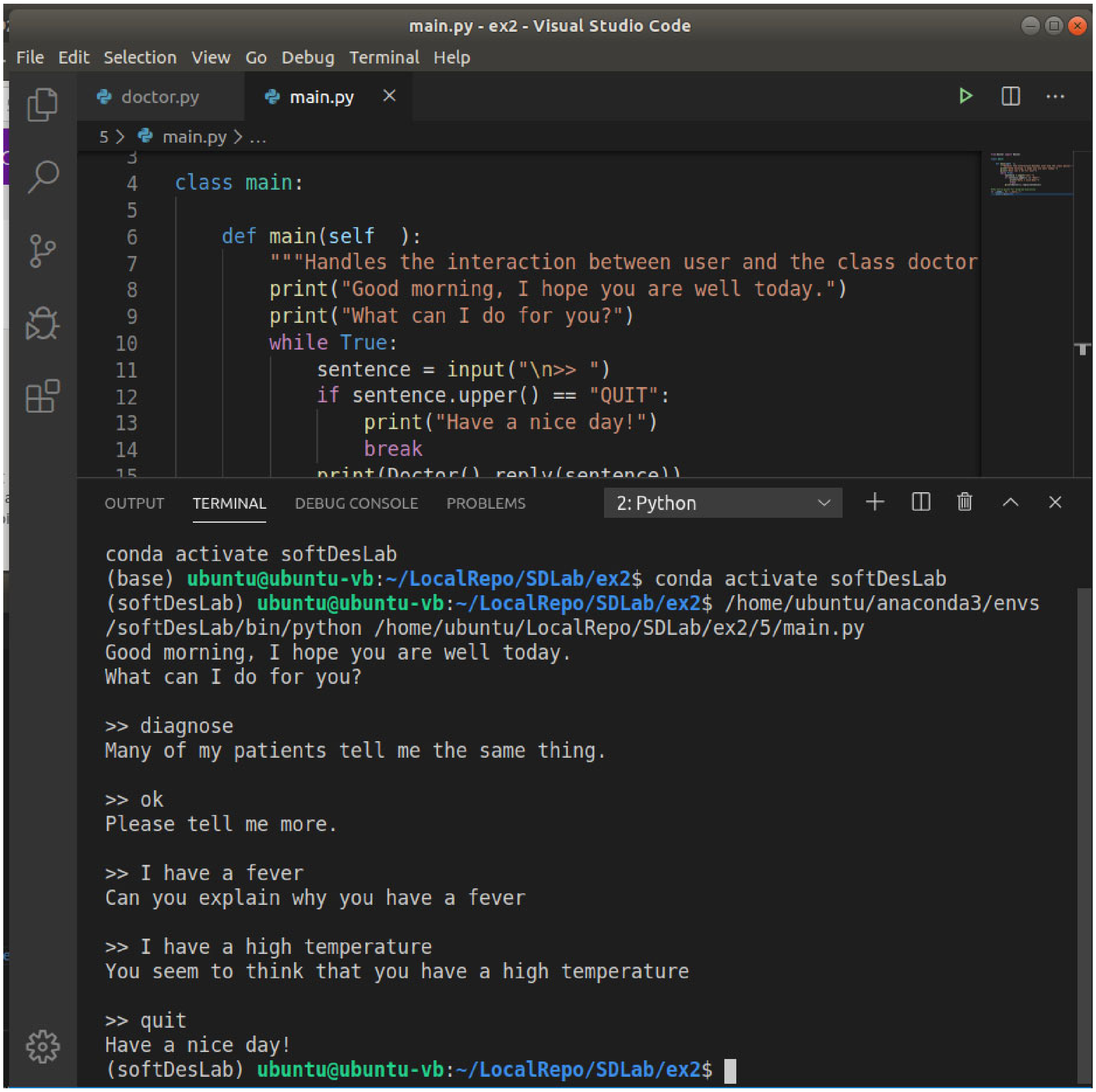
Figure 4.2 is the output which shows if the user had 3 incorrect logins, the program will call the police and then lock the Login button. The other figure shows the working atm machine if you logged in correctly wherein you can check you balance, deposit, withdraw, log out properly and re log in if the user wishes to.



```
ATM
-bank
-account
-error
-nameLabel
-pinLabel
-amountLabel
-statusLabel
-nameField
-pinField
-amountField
-statusField
-balanceButton
-depositButton
-withdrawButton
-loginButton
+__init__(bank)
+login()
+logout()
+getBalance()
+deposit()
+withdraw()
+main()
```

Figure 4.3

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.



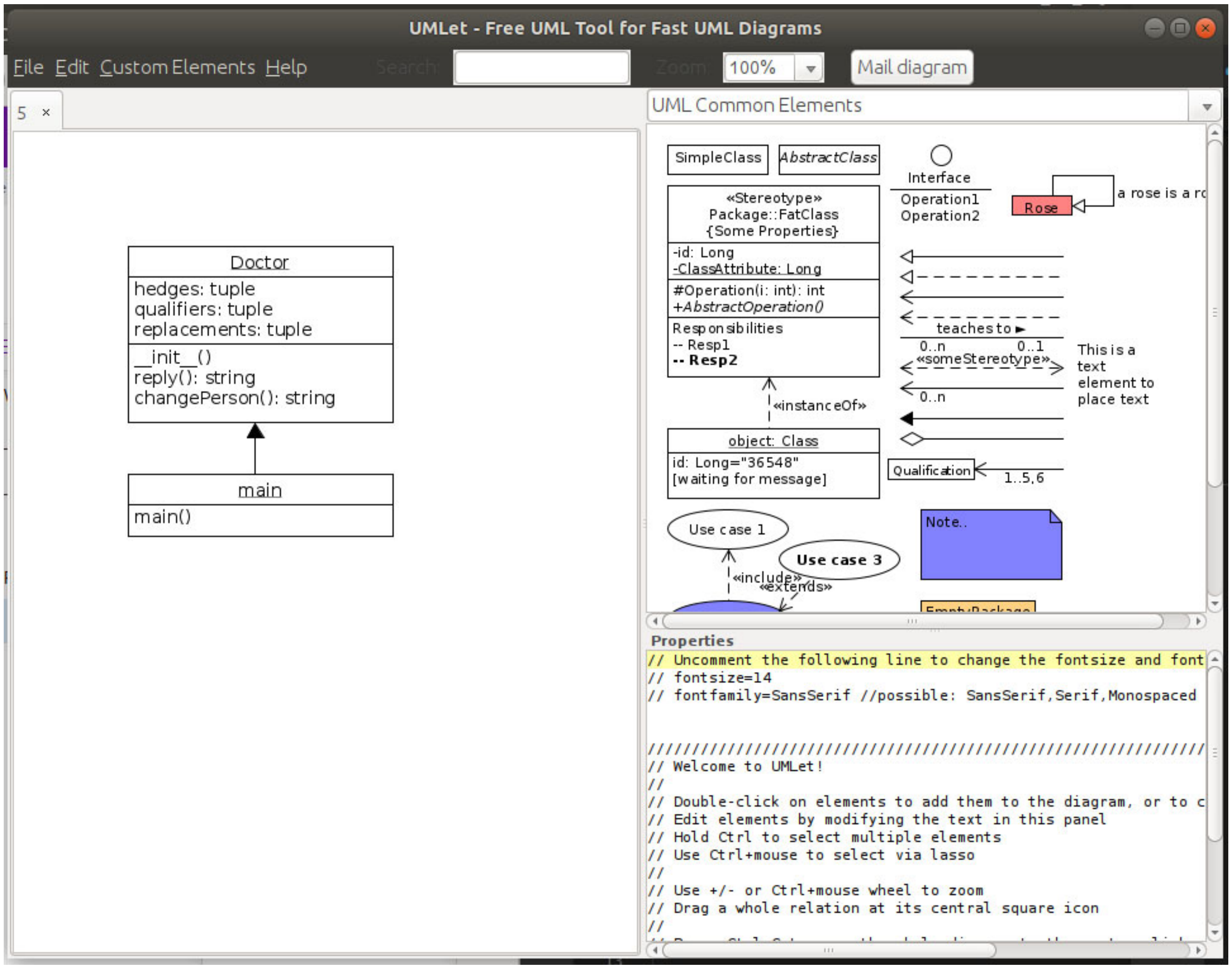
main.py - ex2 - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
doctor.py
main.py
5 > main.py > ...
OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS
2: Python

```python
class main:

    def main(self  ):
        """Handles the interaction between user and the class doctor
        print("Good morning, I hope you are well today.")
        print("What can I do for you?")
        while True:
            sentence = input("\n>> ")
            if sentence.upper() == "QUIT":
                print("Have a nice day!")
                break
            print(Doctor().reply(sentence))
```

```
conda activate softDesLab
(base) ubuntu@ubuntu-vb:~/LocalRepo/SDLab/ex2$ conda activate softDesLab
(softDesLab) ubuntu@ubuntu-vb:~/LocalRepo/SDLab/ex2$ /home/ubuntu/anaconda3/envs
/softDesLab/bin/python /home/ubuntu/LocalRepo/SDLab/ex2/5/main.py
Good morning, I hope you are well today.
What can I do for you?

>> diagnose
Many of my patients tell me the same thing.

>> ok
Please tell me more.

>> I have a fever
Can you explain why you have a fever

>> I have a high temperature
You seem to think that you have a high temperature

>> quit
Have a nice day!
(softDesLab) ubuntu@ubuntu-vb:~/LocalRepo/SDLab/ex2$
```

Figure 5.1

Programming Exercise 5 is the modification of the Doctor program described in Chapter 5. The modified program is restructured to two different classes, Doctor class and the main class. The main class handles the user's interaction with the doctor class whereas the Doctor class has the ability to produce a greeting, a signoff message, and a reply to a patient's string.

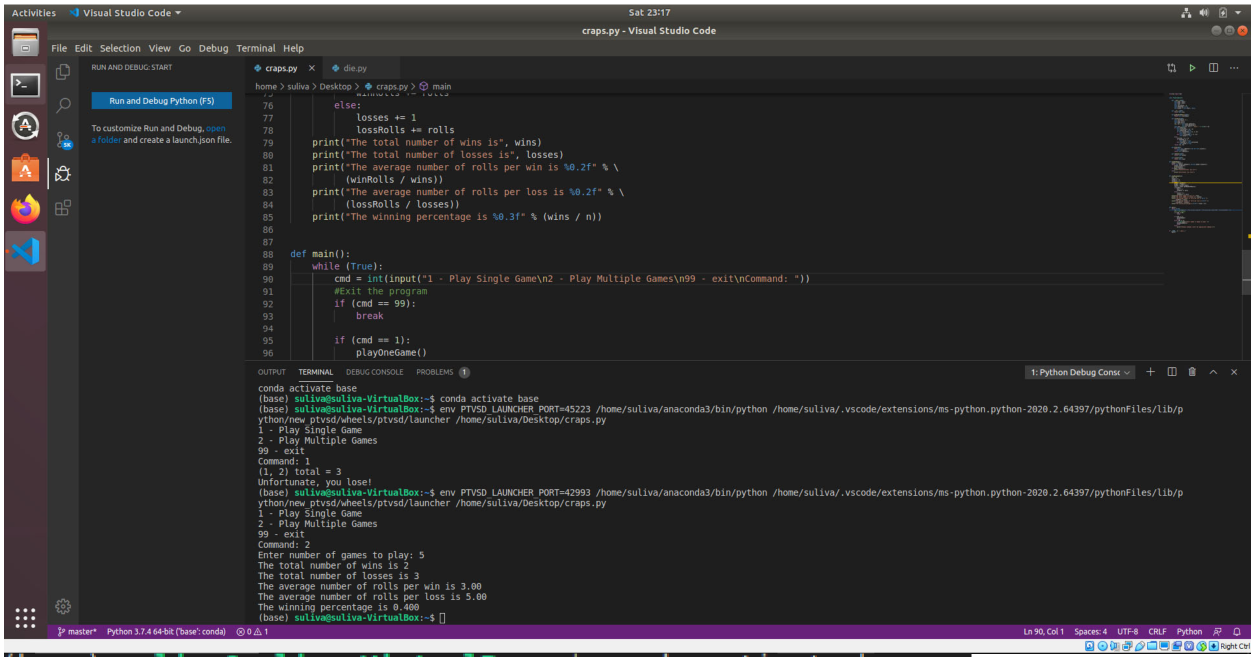


```
// Uncomment the following line to change the fontsize and font
// fontsize=14
// fontfamily=SansSerif //possible: SansSerif,Serif,Monospaced


//////////////////////////////////////////////////////////////////
// Welcome to UMLet!
//
// Double-click on elements to add them to the diagram, or to c
// Edit elements by modifying the text in this panel
// Hold Ctrl to select multiple elements
// Use Ctrl+mouse to select via lasso
//
// Use +/- or Ctrl+mouse wheel to zoom
// Drag a whole relation at its central square icon
//
```

Figure 5.2

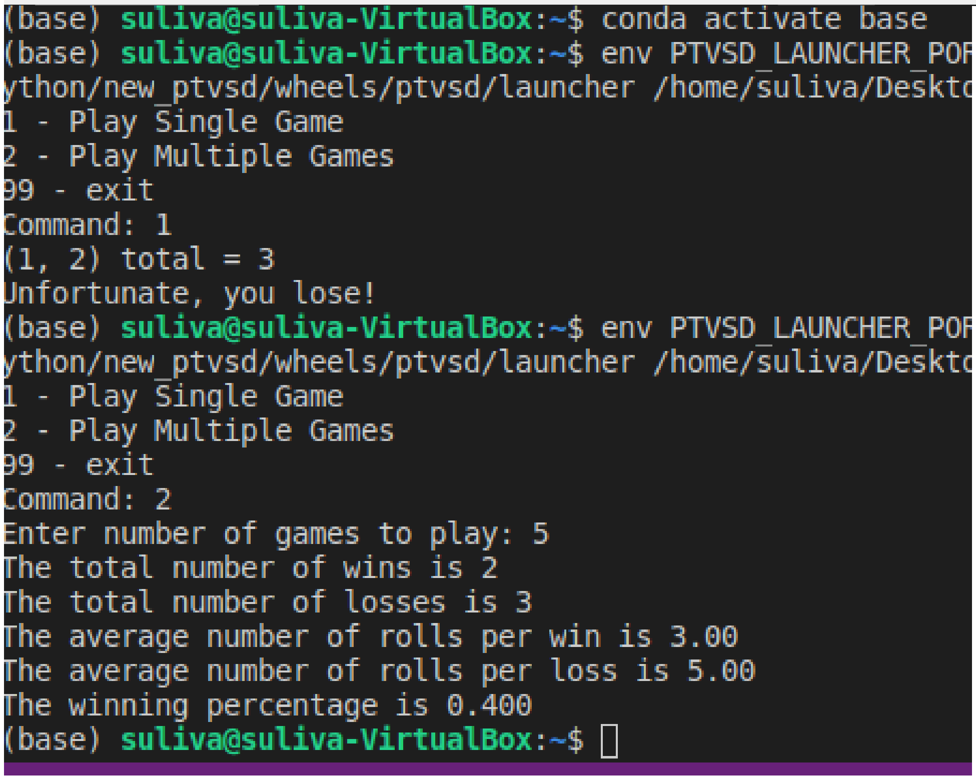
Figure 5.2 illustrates the UML diagram of the modified Doctor program which is created in UMLet, a free tool for creating UML Diagrams. The figure shows the two classes, Doctor and main in which the main class is used to obtain methods from the Doctor class.

6. On the machine problem 6, this is dice game that player 1 and player 2 plays. The program is running perfectly. The goal in the machine problem is to create a user interaction that the user can play single games and multiple games. I created a payOneGame function for the one play function to work as well as getting the winner and loser by coding getWinner and getLoser function respectively as well as the play function for the single play only in the Player Class. As shown on the output, it tells the user to input 1 if single play only and 2 for multiple plays then it ask the user for how many plays that the program will do. In addition, this is a continuous loop meaning the loop in asking the user for input will not end until the appropriate numbers has been inputted. To exit the program prematurely, simply the user will give 99 value so the program can be exited without doing anything. Also the umlet diagram has been created to show its relationships.

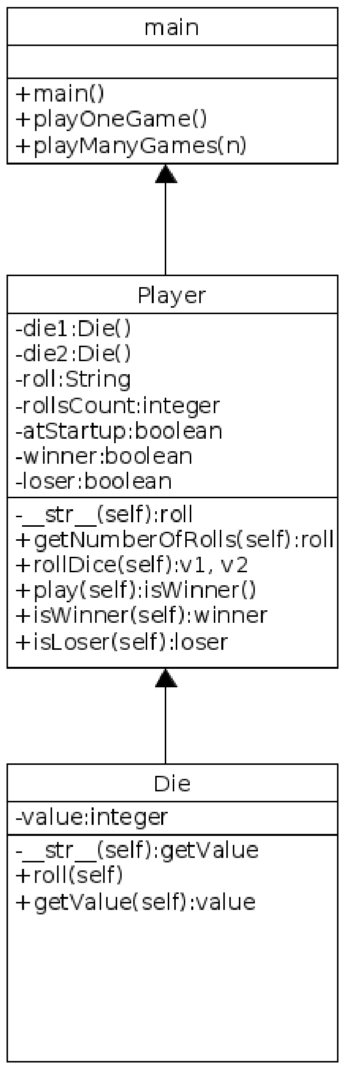


```python
        else:
            losses += 1
            lossRolls += rolls
    print("The total number of wins is", wins)
    print("The total number of losses is", losses)
    print("The average number of rolls per win is %0.2f" % \
          (winRolls / wins))
    print("The average number of rolls per loss is %0.2f" % \
          (lossRolls / losses))
    print("The winning percentage is %0.3f" % (wins / n))


def main():
    while (True):
        cmd = int(input("1 - Play Single Game\n2 - Play Multiple Games\n99 - exit\nCommand: "))
        #Exit the program
        if (cmd == 99):
            break

        if (cmd == 1):
            playOneGame()
```

```
conda activate base
(base) suliva@suliva-VirtualBox:~$ conda activate base
(base) suliva@suliva-VirtualBox:~$ env PTVSD_LAUNCHER_PORT=45223 /home/suliva/anaconda3/bin/python /home/suliva/.vscode/extensions/ms-python.python-2020.2.64397/pythonFiles/lib/p
ython/new_ptvsd/wheels/ptvsd/launcher /home/suliva/Desktop/craps.py
1 - Play Single Game
2 - Play Multiple Games
99 - exit
Command: 1
(1, 2) total = 3
Unfortunate, you lose!
(base) suliva@suliva-VirtualBox:~$ env PTVSD_LAUNCHER_PORT=42993 /home/suliva/anaconda3/bin/python /home/suliva/.vscode/extensions/ms-python.python-2020.2.64397/pythonFiles/lib/p
ython/new_ptvsd/wheels/ptvsd/launcher /home/suliva/Desktop/craps.py
1 - Play Single Game
2 - Play Multiple Games
99 - exit
Command: 2
Enter number of games to play: 5
The total number of wins is 2
The total number of losses is 3
The average number of rolls per win is 3.00
The average number of rolls per loss is 5.00
The winning percentage is 0.400
(base) suliva@suliva-VirtualBox:~$ 
```



```
(base) suliva@suliva-VirtualBox:~$ conda activate base
(base) suliva@suliva-VirtualBox:~$ env PTVSD_LAUNCHER_POR
ython/new_ptvsd/wheels/ptvsd/launcher /home/suliva/Deskto
1 - Play Single Game
2 - Play Multiple Games
99 - exit
Command: 1
(1, 2) total = 3
Unfortunate, you lose!
(base) suliva@suliva-VirtualBox:~$ env PTVSD_LAUNCHER_POR
ython/new_ptvsd/wheels/ptvsd/launcher /home/suliva/Deskto
1 - Play Single Game
2 - Play Multiple Games
99 - exit
Command: 2
Enter number of games to play: 5
The total number of wins is 2
The total number of losses is 3
The average number of rolls per win is 3.00
The average number of rolls per loss is 5.00
The winning percentage is 0.400
(base) suliva@suliva-VirtualBox:~$ 
```



- **Github:** http://bit.ly/2Ivstu4
- **OneDrive:** http://bit.ly/2xkFvbN