



MAPUA UNIVERSITY

SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING

Experiment 3:

Object Oriented Design and Implementation

CPE106L (Software Design Laboratory)

Darren M. Briones

Date: 2021. 03. 25 14:53:50

Edmund Hans Tapado

Date: 2021. 03. 25 14:02:28

Justine Angelo Ariel V. Villespin

Date: 2021. 03. 24 20:12:30

Group No.: **Group 3**
Section: **B2**



Readings, Insights, and Reflection

Insights

This chapter is all about Design and Classes. We learn how to determine the attributes and behavior of certain classes of objects within the program which helps us to understand how the code flows considering that in this chapter we also learned about methods, parameters, return types, and creating a data structure that fits the attributes of a class of objects. Also, in the laboratory report, we learned and used UMLs, which is a diagram that shows the classes and sub classes and shows its attributes, variables, functions, etc... **(BRIONES)**

A UML diagram is the main building block for object-oriented modelling. They are used to represent the objects in a program or system, their attributes, operations and functions, and their relationships to one another. Using a UML diagram is a great way to visually represent how the program works because of its readability. **(TAPADO)**

When performing the assigned tasks that I need to do in Laboratory report 3, I realized that understanding and creating a UML class diagram is important in becoming a programmer. It is important to understand because a good programmer always has his/her blueprint which is the UML class diagram, flowcharts, and other applications that can be used to demonstrate the structure of the program that you are creating. **(VILLESPIN)**

Reflection

Well, I see the data structure of the program to be the most important part of programming. Without essential knowledge and skills, the program code will just be messy and not modular. Which means other programmers will be lost on how to edit the code. Alongside with the usage of UMLs, it gives further visualization of how a program might work and it shows all the elements within classes and sub classes. We have already learned about this on our CPE103L course, and this chapter refreshed my knowledge on that and even made it clearer. **(BRIONES)**

When creating or starting a program, it is best to formulate the structure such as the classes and functions that will be involved running the program properly. For programmers, it is a good practice to create a diagram of the program because it allows an easier way to code the flow of the program. Also, it becomes easier for future updates to be implemented in the program because the diagram can show where and what should be changed. In Lab Report 3, I was able to explore what should be the elements of a UML diagram. It helped me reverse-engineer the code into simplified version where I can see what the functions are and if there

are other functions that could be added or changed. **(TAPADO)**

In lab rep 3, I learned that UML class diagram is one of the tools that can be used to create your blueprint for the program you are creating. Also, I learned that it is vital to programmers since programmers should always start in creating their class diagrams before performing coding. Moreover, when I performed my assigned task, I was able to show my understanding in creating class diagrams for the program that was given. **(VILLESPIN)**

Answers to Questions

1. A (is owned by a particular instance of a class and no other)
2. C (self)
3. B (set the instance variables to initial variables)
4. B (always must have at least one parameter name, called self)
5. B (the entire class in which it is introduced)
6. B (when it can no longer be referenced anywhere in a program)
7. A (all instances of class have in common)
8. B (A.__init__())
9. B (pickle them using the pickle function dump)
10. A (has a single header but different bodies in different classes)

InLab

- **Objectives**

1. **Use** Visual Studio Code of Linux in debugging the python programs.
2. **Use** UMLet to create a class diagram of the program.

- **Tools Used**

- Linux Terminal
- Visual Studio Code
- UMLet

- **Procedure.**

In Figure 1.1, we created the UML class diagram of the python program file student.py. We created a class diagram for this program to know and visualize the structure of the program. UML diagram helps the programmers to create a program using the constructed UML diagram. It will serve as a blueprint or guide in creating the program.

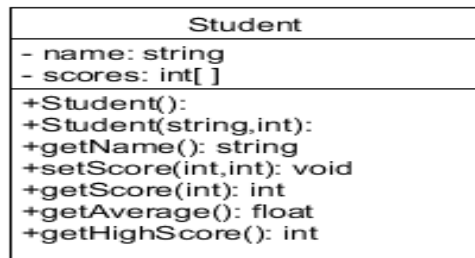
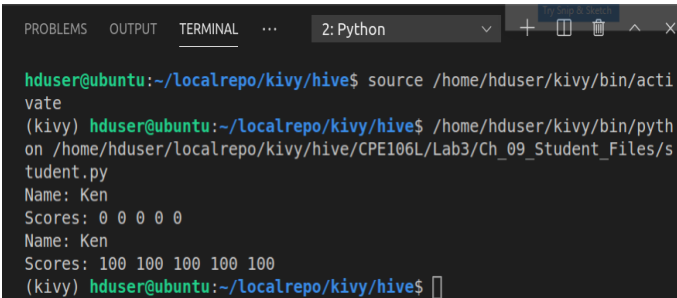


Figure 1.1. UML class diagram for student.py.

In Figure 1.2, it shows the output program of student.py. Looking at the output of the program, we can see that the student named Ken has a test scores of five 0 in the first output and in the second output we can see that student Ken again has a test score of five 100.

A terminal window titled '2: Python' showing the execution of a Python script. The user enters 'source /home/hduser/kivy/bin/activate' and then runs the script. The output shows the name 'Ken' and five test scores of 0 in the first run, and five test scores of 100 in the second run.

```
hduser@ubuntu:~/localrepo/kivy/hive$ source /home/hduser/kivy/bin/activate
(kivy) hduser@ubuntu:~/localrepo/kivy/hive$ /home/hduser/kivy/bin/python /home/hduser/localrepo/kivy/hive/CPE106L/Lab3/Ch_09_Student_Files/student.py
Name: Ken
Scores: 0 0 0 0 0
Name: Ken
Scores: 100 100 100 100 100
(kivy) hduser@ubuntu:~/localrepo/kivy/hive$
```

Figure 1.2. Start and end of student.py program.

In Figure 1.3, this code outputs the five 100 test score of Ken in the second output of Fig. 1.2. If we manually change 100, we can change the test second output test score of Ken.

```
for i in range(1, 6):
    student.setScore(i, 100)
print(student)
```

Figure 1.3. Tracing the second output.

In Figure 2.1, we created the UML class diagram of the python program file bank.py and savingsaccount.py. SavingsAccount is the containing class that is why the diamond symbol is pointed to it.

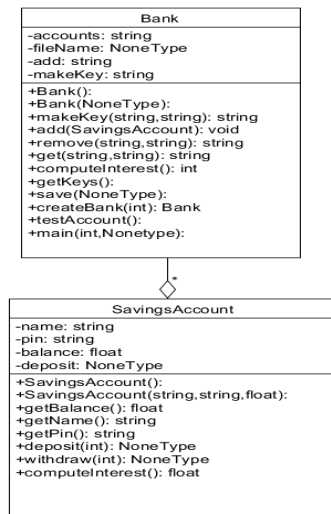


Figure 2.1. UML class diagram for bank.py and savingsaccount.py.

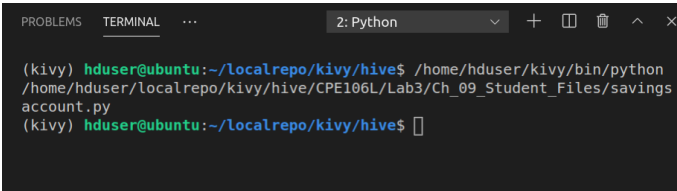
In Figure 2.2, the program shows the bank info of the person. We can see in the output the name of the person, the PIN, and the balance of the person's money.

```

PROBLEMS  TERMINAL  ...  2: Python
(kivy) hduser@ubuntu:~/localrepo/kivy/hive$ /home/hduser/kivy/bin/python
/home/hduser/localrepo/kivy/hive/CPE106L/Lab3/Ch_09_Student_Files/bank.py
Name: Ken
PIN: 1000
Balance: 500.0
None
Expect 600: 600.0
None
Expect 600: 550.0
None
Expect 500: 450.0
Amount must be >= 0
Expect 500: 450.0
Insufficient funds
Expect 500: 450.0
(kivy) hduser@ubuntu:~/localrepo/kivy/hive$ 
  
```

Figure 2.2. Start and end of bank.py program.

In Figure 2.3, it doesn't show any output since savingsaccount.py is just a class file. Which means that, there is no output for the file but it can be used in other files.

A terminal window with a dark background. The title bar shows '2: Python' and standard window controls. The terminal text shows a user running a python command to execute a file, with no visible output.

```
PROBLEMS  TERMINAL  ...  2: Python  +  [ ]  [ ]  ^  x  
  
(kivy) hduser@ubuntu:~/localrepo/kivy/hive$ /home/hduser/kivy/bin/python  
/home/hduser/localrepo/kivy/hive/CPE106L/Lab3/Ch_09_Student_Files/savings  
account.py  
(kivy) hduser@ubuntu:~/localrepo/kivy/hive$ [ ]
```

Figure 2.3. Start and end of savingsaccount.py program.

PostLab

Programming Problems

POSTLAB #1

For project#1, three methods will be required for the Student class that will compare two Student object's names. The first method will test for equality. The second method will test if the first instance is less than the second instance. The third method will test for greater than or equal. The codes for these methods are shown in figure 1.

```
def __eq__(self, other):
    """Returns if equal or not"""
    if (self.name == other.name):
        return "Equal"
    else:
        return "Not Equal"
def __lt__(self, other):
    """Returns if less than or not"""
    if (self.name < other.name):
        return "Less Than"
    else:
        return "Not less than"
def __gt__(self, other):
    if self.name > other.name:
        return "Greater than"
    elif self.name == other.name:
        return "Both are equal"
    else:
        return "Not greater than or equal"
```

Figure 1. The three (3) methods source code

As seen in Figure 2-a, the comparison will happen between the string name **StudentBoy** and **StudentGirl** after being called in the Student class. The result is shown in Figure 2-b.


```
def main():
    """A simple test."""
    student = Student("StudentBoy", 5)

    for i in range(1, 6):
        student.setScore(i, 100)
    print(student)

    student2 = Student("StudentGirl", 5)

    for i in range(1, 6):
        student2.setScore(i, 100)
    print(student2)
    print("\n")
    print("-----COMPARISON-----")
    print(student==student2)
    print(student<student2)
    print(student2<student)
    print(student>student2)
    print("\n")
```

Figure2-a. main function test

Anaconda Prompt (anaconda3)

```
(base) C:\Users\Monmon Tapado\OneDrive - Mapúa University\2y3t\cpe1061\week03\lr3\postlab_proj1>python student.py
Name: StudentBoy
Scores: 100 100 100 100 100
Name: StudentGirl
Scores: 100 100 100 100 100

-----COMPARISON-----
Not Equal
Less Than
Not less than
Not greater than or equal
```

Figure2-b. Program execution

POSTLAB #2

For project#2, the program shuffles the list of students initialized using the *random.shuffle()* function. In order to call the *random.shuffle()* function, the *import random* must be called at the start of the program. After which, the program sorts the unordered list using the *sort()* function. The *sort()* uses a lambda parameter in order to arrange the elements of the **studentList** by their key. Figure 3 shows the whole process of initialization of the elements down to sorting the elements.

```
def main():
    """A simple test."""
    studentsList = []

    student = Student("Justine", 5)
    for i in range(1, 6):
        student.setScore(i, 100)
    studentsList.append(student)

    student1 = Student("Mon", 8)
    for i in range(1, 9):
        student1.setScore(i, 100)
    studentsList.append(student1)

    student2 = Student("Darren", 5)
    for i in range(1, 6):
        student2.setScore(i, 100)
    studentsList.append(student2)

    random.shuffle(studentsList)

    print("Shuffled list of Students: ")

    for i in studentsList:
        print(i.__str__())

    print("\nSorted list of Students: ")

    studentsList.sort(key = lambda x:x.name)

    for i in studentsList:
        print(i.__str__())

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

Figure 3. main function

The program execution result is shown in Figure 4.

```
(base) C:\Users\Monmon Tapado\OneDrive - Mapúa University\2y3t\cpe1061\week03\lr3\postlab_proj2>python student.py
Shuffled list of Students:
Name: Mon
Scores: 100 100 100 100 100 100 100 100
Name: Justine
Scores: 100 100 100 100 100
Name: Darren
Scores: 100 100 100 100 100

Sorted list of Students:
Name: Darren
Scores: 100 100 100 100 100
Name: Justine
Scores: 100 100 100 100 100
Name: Mon
Scores: 100 100 100 100 100 100 100 100
```

Figure 4. Program Execution

For both project#1 and project#2, they used the same class that contains the same functions / methods. The only difference is how *def main()* was programmed to execute specific instructions. The UML Class Diagram is shown in Figure 5. The class Student has two private attributes which are the **name** and the **score**. The public attributes are the constructor, getName(), setScore(), getScore(), getAverage(), getHighScore(), __eq__(), __lt__(), __gt__(), __str__(). The simplified diagram is shown below:

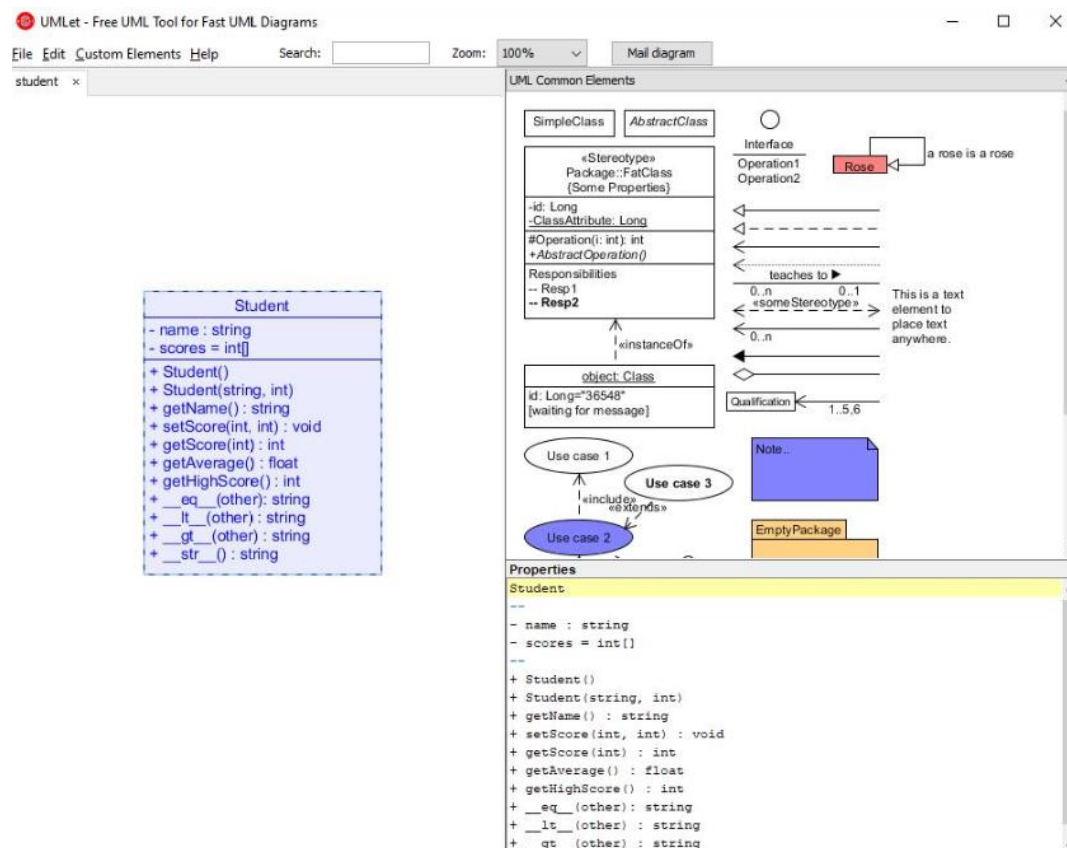


Figure 5. UML Class Diagram for Student Class

POSTLAB #3

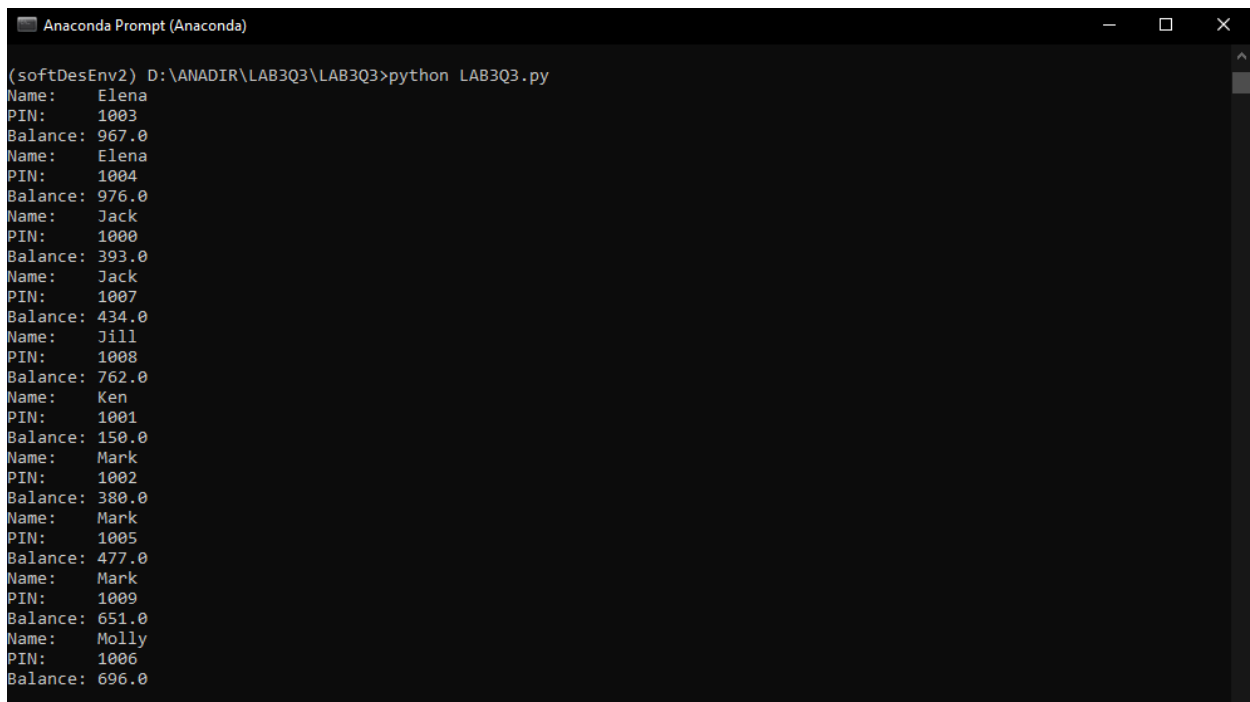
For project 3, since the problem asks for the program to return a string that should contain the accounts (THE DATA ARE PROVIDED) in a random manner and sort the strings in order. The problem wanted us to modify the str method of bank and add more methods under the SavingsAccount class. However, we found a way that only required us to modify the str data (without adding changes to the SavingsAccount class) and it still got the job done and returned the strings already sorted. The modifications we made is the use of the join and sorted methods.

Look on to Figure 6. This shows the modifications we made for the str method.

```
def __str__(self):  
    """Returns the string representation of the bank."""  
    # using the sorted method sorting the dictionary  
    # by values's name  
    return "\n".join([str(v) for (k, v) in  
        sorted(self.accounts.items(),  
            key=lambda cv: cv[1].getName())])
```

Figure 6. Changes made in the str method.

In Figure 7, it shows the program execution after the modifications made in the str method. As you can see, the strings printed are already sorted. You may notice that some names are repeated. The reason for this is that the createBank method randomly selects names within a given list. The strings are different every time you execute the program.



Anaconda Prompt (Anaconda)

```
(softDesEnv2) D:\ANADIR\LAB3Q3\LAB3Q3>python LAB3Q3.py  
Name: Elena  
PIN: 1003  
Balance: 967.0  
Name: Elena  
PIN: 1004  
Balance: 976.0  
Name: Jack  
PIN: 1000  
Balance: 393.0  
Name: Jack  
PIN: 1007  
Balance: 434.0  
Name: Jill  
PIN: 1008  
Balance: 762.0  
Name: Ken  
PIN: 1001  
Balance: 150.0  
Name: Mark  
PIN: 1002  
Balance: 380.0  
Name: Mark  
PIN: 1005  
Balance: 477.0  
Name: Mark  
PIN: 1009  
Balance: 651.0  
Name: Molly  
PIN: 1006  
Balance: 696.0
```

Figure 7. Program execution

Lastly is the UML class diagram for the program. Refer to Figure 8. It is essentially the same UML diagram used beforehand in the INLAB section of this lab report.

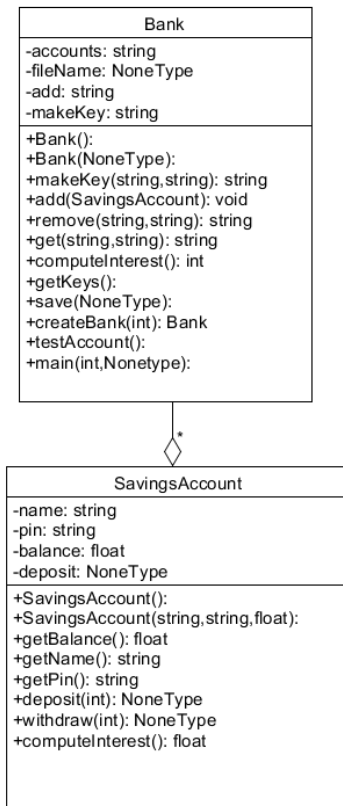


Figure 8. UML Class Diagram for Program 3.