



MAPÚA UNIVERSITY

SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING

Experiment 3:

Object Oriented Design and Implementation

CPE106L (Software Design Laboratory)

Member 1 Aljehro R. Abante

Member 2 Cerdan Karl T. Alarilla

Member 3 Daryl Jake A. Fernandez

Group No.: **3**
Section: **E01**



Readings, Insights, and Reflection

Systems Analysis and Design: An Object-Oriented Approach with UML

Alan Dennis, Barbara Haley Wixom, David Tegarden

Edition 5

ISBN: 9781119561217

Insights and Reflections

Abante (Chapter 4: Use Case Diagram)

- Chapter 4 of the reading highlights the critical role of use case diagrams in defining user requirements during the systems development lifecycle. This chapter emphasizes how use cases illustrate the interactions between users and the system, helping analysts understand the functionalities that are essential for the new system. Mapping out user actions and system responses proves that use case diagrams facilitate clear communication with stakeholders, ensuring that both primary requirements and important exceptions are captured. This user-focused approach enhances the design process, making it easier to create systems that meet users' needs, as demonstrated through practical examples like the vending kiosk.

Alarilla (Chapter 5: Class Diagram)

- This chapter emphasizes the importance of class diagrams in system modeling. These diagrams visually represent classes, attributes, methods, and their relationships, making it easier for analysts and designers to grasp the system's structure. Class diagrams help developers communicate requirements clearly with stakeholders. The chapter also emphasizes the collaborative nature of refining these diagrams, ensuring everyone's input is valued, which leads to a stronger and more flexible design.

Fundamentals of Python: First Programs

Kenneth A. Lambert

Edition 2

ISBN: 9781337671019

Cengage Learning US

2019

Insights and Reflections

Abante and Alarilla (Chapter 9: Design with Classes)

- This chapter of the reading demonstrates how to use classes to organize and simplify code. It highlights the power of object-oriented programming (OOP) in managing complex projects by focusing on an object's purpose rather than the details of how it works. The chapter explains how to create and customize classes with attributes and methods, including special methods like `__init__` to set up new objects and `__str__` to display them. By using concepts like inheritance and polymorphism, developers can extend existing classes and add new features more easily.

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 values
4. a. can have zero or more parameter names
5. b. the entire class in which it is introduced, as well as any module in which an object that owns the variable is referenced
6. b. when it can no longer be referenced anywhere in a program
7. a. all instances of a class have in common
8. b. `A.init(self)`
9. b. pickle them using the pickle function `dump`
10. a. has a single header but different bodies in different classes

InLab

Objectives

1. **Analyze** the programs that were provided
2. **See** if you can determine the UML diagram of your selected program
3. Pick at least 2 programs to analyze.

• Tools Used

- VS Code
- Linux Ubuntu (Optional)

• Procedure.

We first picked the sample programs that we will use to analyze and discuss their respective UML diagrams.

The programs that we picked are: **cards.py**, and **blackjack.py**.

These are their respective sample outputs:

#blackjack.py

```
PS C:\Users\aljeh> & C:/Users/aljeh/AppData/Local/Microsoft/WindowsApps/python3.13.exe "f:/Download Location/blackjack.py"
Player:
 8 of Clubs, Queen of Spades
 18 points
Dealer:
 3 of Spades
Do you want a hit? [y/n]: n
Dealer:
 3 of Spades, 2 of Hearts, 2 of Spades, King of Spades
 17 points
You win
PS C:\Users\aljeh> 
```

#cards.py (The list is too long so we've decided to crop out some parts)

```
PS C:\Users\aljeh> & C:/Users/aljeh/AppData/Local/Microsoft/WindowsApps/python3.13.exe "f:/Download Location/cards.py"
A new deck:
Ace of Spades
2 of Spades
3 of Spades
4 of Spades
5 of Spades
6 of Spades
7 of Spades
8 of Spades
9 of Spades
10 of Spades
Jack of Spades
Queen of Spades
```

Now that we've seen their respective output, we now determine each program's classes and their respective attributes and methods.

#blackjack.py

1. Player Class

Attributes:

cards: A list of Card objects representing the player's hand.

Methods:

- **__init__(self, cards):** Initializes the player with a hand of cards.
- **__str__(self):** Returns a string representation of the player's cards and their total points.
- **hit(self, card):** Appends a new card to the player's hand.
- **getPoints(self):** Calculates and returns the total points of the player's hand.
- **hasBlackjack(self):** Checks if the player has a blackjack (21 points with 2 cards).

2. Dealer Class (inherits from Player)

Method Overrides:

- **__init__(self, cards):** Initializes the dealer with a hand of cards, showing only one card at the start.
- **__str__(self):** Returns a string representation of the dealer's cards, showing only one card if the dealer hasn't hit yet.
- **hit(self, deck):** Deals cards from the deck until reaching 17 points or higher, then shows all cards.

3. Blackjack Class

Attributes:

- **deck:** An instance of the Deck class representing the deck of cards.
- **player:** An instance of the Player class representing the player.
- **dealer:** An instance of the Dealer class representing the dealer.

Methods:

- **__init__(self):** Initializes a new game of Blackjack, shuffling the deck and dealing cards to the player and dealer.
- **play(self):** Manages the game play, allowing the player to hit or stand and determining the outcome of the game.

Below is the figure of a sample UML diagram of blackjack.py:

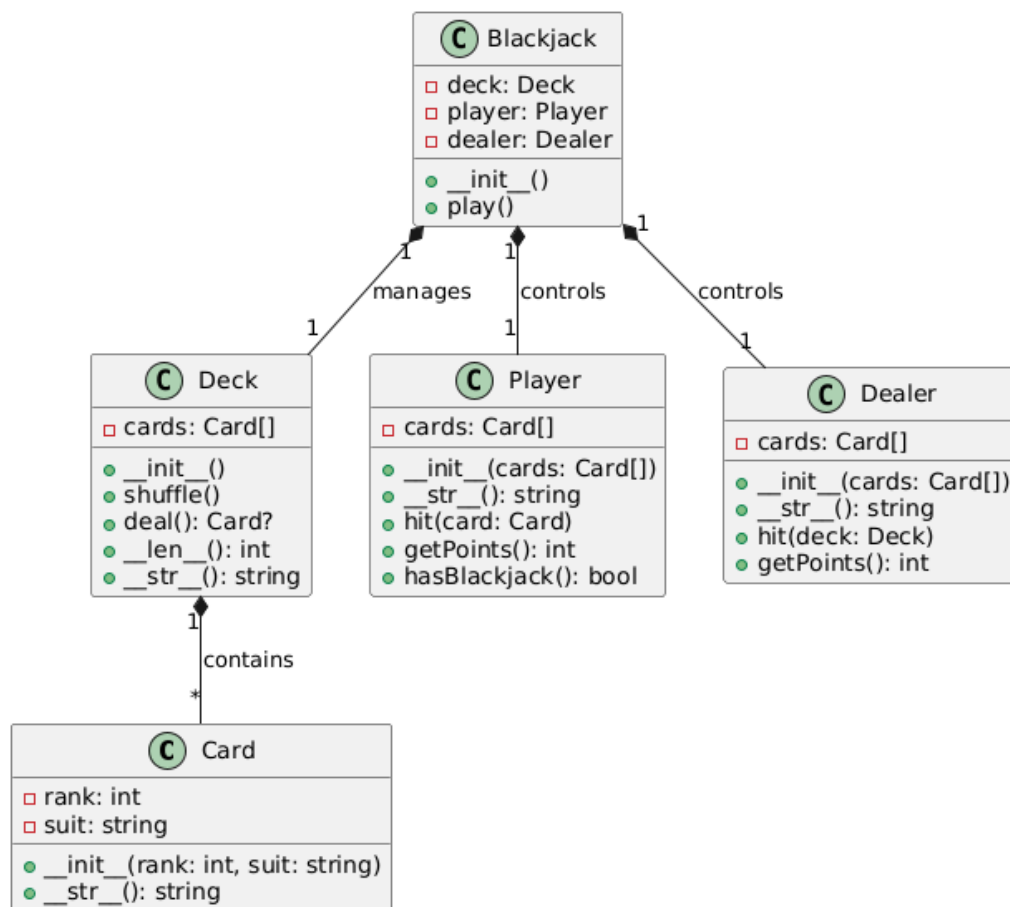


Figure 1 (Sample UML Diagram for blackjack.py)

- Figure 1 is a sample UML diagram for the blackjack.py. The UML diagram explains and delineates the roles of Player and Dealer as independent classes without the confusion for inheritance. The Card class represents a playing card with attributes for rank and suit, while the Deck class handles a collection of cards, allowing for shuffling and dealing. The Player class maintains the player's hand, enabling actions like hitting (drawing a card), calculating total points, and checking for blackjack. The Dealer class similarly manages its own hand and follows different rules, justifying its separation from the Player. Lastly, the Blackjack class coordinates the game's flow, managing one Deck, one Player, and one Dealer, making it clear how the game operates.

#cards.py

1. Card Class

Attributes:

- **rank:** The rank of the card (1-13, with 1 being Ace and 13 being King).
- **suit:** The suit of the card (Spades, Diamonds, Hearts, Clubs).

Methods:

- **__init__(self, rank, suit):** Initializes the card with its rank and suit.
- **__str__(self):** Returns a string representation of the card (e.g., "Ace of Spades").

2. Deck Class

Attributes:

- **cards:** A list of Card objects representing the complete deck of 52 cards.

Methods:

- **__init__(self):** Initializes a full deck of cards by creating Card instances for each rank and suit.
- **shuffle(self):** Shuffles the list of cards in the deck.
- **deal(self):** Removes and returns the top card from the deck, or None if the deck is empty.
- **__len__(self):** Returns the number of cards remaining in the deck.
- **__str__(self):** Returns a string representation of the entire deck of cards.

Below is the sample UML diagram of cards.py

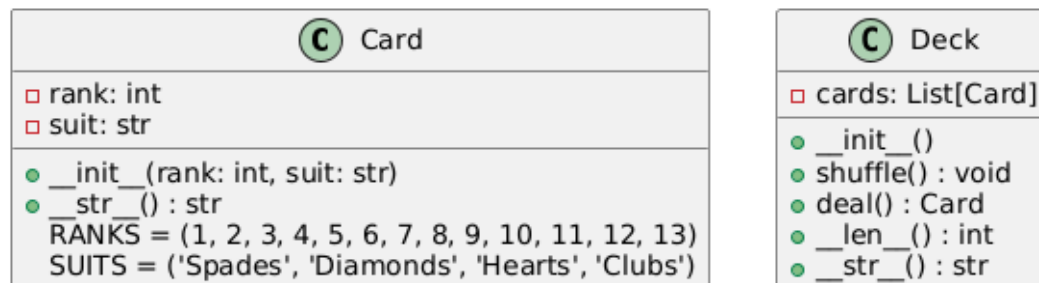


Figure 2 (Sample UML Diagram for cards.py)

- The provided sample UML diagram for cards.py represents the structure and relationships of the classes in the cards.py file. The Card class has private attributes for its rank (an integer) and suit (a string), with methods for initializing and representing the card as a string. The Deck class holds a list of Card objects and includes methods for creating the deck, shuffling, dealing cards, and checking how many cards are left.

Programming Problems

1. Add three methods to the Student class (in the file student.py) that compare two Student objects. One method should test for equality. A second method should test for less than. The third method should test for greater than or equal to. In each case, the method returns the result of the comparison of the two students' names. Include a main function that tests all of the comparison operators.

The provided code defines a Student class that manages a student's name and their test scores. Upon initialization, each instance of this class takes a student's name and the number of scores they will have, initializing their scores to zero. The class includes methods for setting scores, retrieving specific scores, calculating the average score and the highest score, and string representation for easy display. It also implements comparison methods to allow direct comparisons between students based on their names, thus facilitating sorting or equality checks.

The main() function tests the functionality of the Student class by creating multiple student instances and printing their details. It checks for equality and comparison (less than and greater than) between students based on their names. The output includes the name and scores of the students, along with the results of the comparisons, demonstrating how the class can be used to manage and compare student data effectively. For example, it will show whether "Alice" (student1) is equal to "Bob" (student2) and the results of various relational comparisons between them.


```

postlab_proj1.py - Lab3 - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
LAB3
  postlab_proj1
    __pycache__
    postlab_proj1.py
    test_postlab_proj...
  postlab_proj2
    postlab_proj2.py
  postlab_proj3
    __pycache__
    bank.py
    savingsaccount.py

  OUTLINE
  TIMELINE

postlab_proj1 > postlab_proj1.py
1  """
2  File: student.py
3  Resources to manage a student's name and test scores.
4  """
5
6  class Student(object):
7      """Represents a student."""
8
9      def __init__(self, name, number):
10         """All scores are initially 0."""
11         self.name = name
12         self.scores = []
13         for count in range(number):
14             self.scores.append(0)
15
16     def getName(self):
17         """Returns the student's name."""
18         return self.name
19
20     def setScore(self, i, score):
21         """Resets the ith score, counting from 1."""
22         self.scores[i - 1] = score
23
24     def getScore(self, i):
25         """Returns the ith score, counting from 1."""
26         return self.scores[i - 1]
27
28     def getAverage(self):

```

Figure 1. postlab_proj1_codes

```

(base) cerdan@DESKTOP-ICVPFLB:~/LocalRepo/cpe1001-4/lab/lab3/postlab_proj1$ python3 postlab_proj1.py
Student 1: Name: Alice
Scores: 0 0 0
Student 2: Name: Bob
Scores: 0 0 0 0 0
Is Student 1 equal to Student 2? False
Is Student 1 equal to Student 3? True
Is Student 1 less than Student 2? True
Is Student 2 less than Student 3? False
Is Student 1 greater than or equal to Student 2? False
Is Student 1 greater than or equal to Student 3? True

```

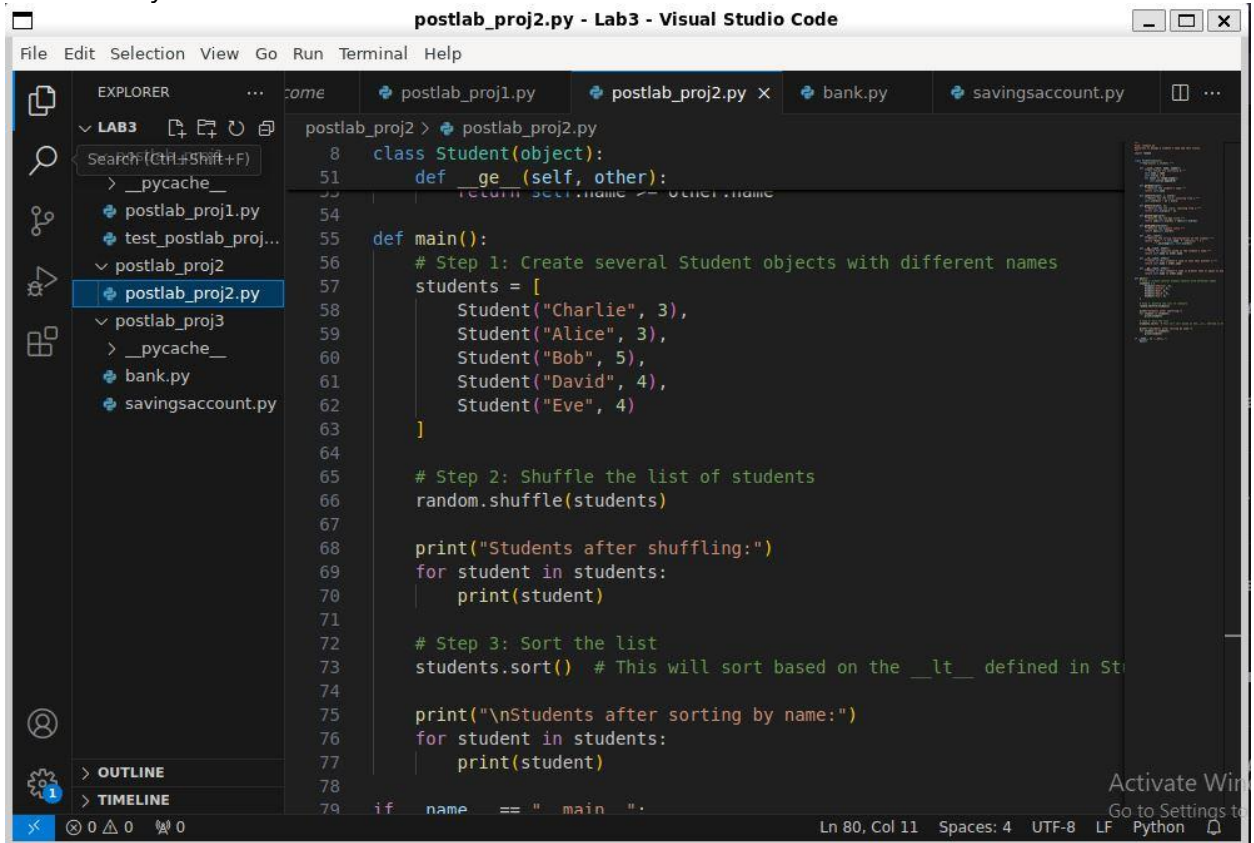
Figure 2. output of postlab_proj1.py

2. This exercise assumes that you have completed Programming Exercise 1. Place several Student objects containing different names into a list and shuffle it. Then run the sort method with this list and display all of the students' information.

The provided code defines a Student class that facilitates the management of students' names and their test scores. Each Student object initializes with a specified name and a designated number of scores, all set to zero. The class includes methods to modify and retrieve scores, compute average and high scores, and provide a string representation of the student. Additionally, it implements comparison methods that allow students to be sorted based on their names.

In the main() function, several Student instances are created with different names, and their order is randomized using random.shuffle(). The shuffled list of students is then

printed to the console. Subsequently, the list is sorted alphabetically by student names, leveraging the comparison methods defined in the class, and the sorted list is printed as well. The output will show the initial randomized order of students followed by their names in ascending order, demonstrating both the randomization and sorting functionality.



```
postlab_proj2.py - Lab3 - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
LAB3
  > Search (Ctrl+Shift+F)
  > __pycache__
  > postlab_proj1.py
  > test_postlab_proj...
  > postlab_proj2
    > postlab_proj2.py
  > postlab_proj3
  > __pycache__
  > bank.py
  > savingsaccount.py

OUTLINE
TIMELINE

postlab_proj2 > postlab_proj2.py
8 class Student(object):
51 def __lt__(self, other):
53     return self.name < other.name
54
55 def main():
56     # Step 1: Create several Student objects with different names
57     students = [
58         Student("Charlie", 3),
59         Student("Alice", 3),
60         Student("Bob", 5),
61         Student("David", 4),
62         Student("Eve", 4)
63     ]
64
65     # Step 2: Shuffle the list of students
66     random.shuffle(students)
67
68     print("Students after shuffling:")
69     for student in students:
70         print(student)
71
72     # Step 3: Sort the list
73     students.sort() # This will sort based on the __lt__ defined in St
74
75     print("\nStudents after sorting by name:")
76     for student in students:
77         print(student)
78
79 if __name__ == "__main__":
```

Figure 3. postlab_proj2_codes

```
(base) cerdan@DESKTOP-ICVPFLB:~/localRepo/cpe1061-4/Lab/Lab3/postlab_proj1$ python3 postlab_proj2.py
Students after shuffling:
Name: David
Scores: 0 0 0 0
Name: Charlie
Scores: 0 0 0
Name: Bob
Scores: 0 0 0 0 0
Name: Eve
Scores: 0 0 0 0
Name: Alice
Scores: 0 0 0

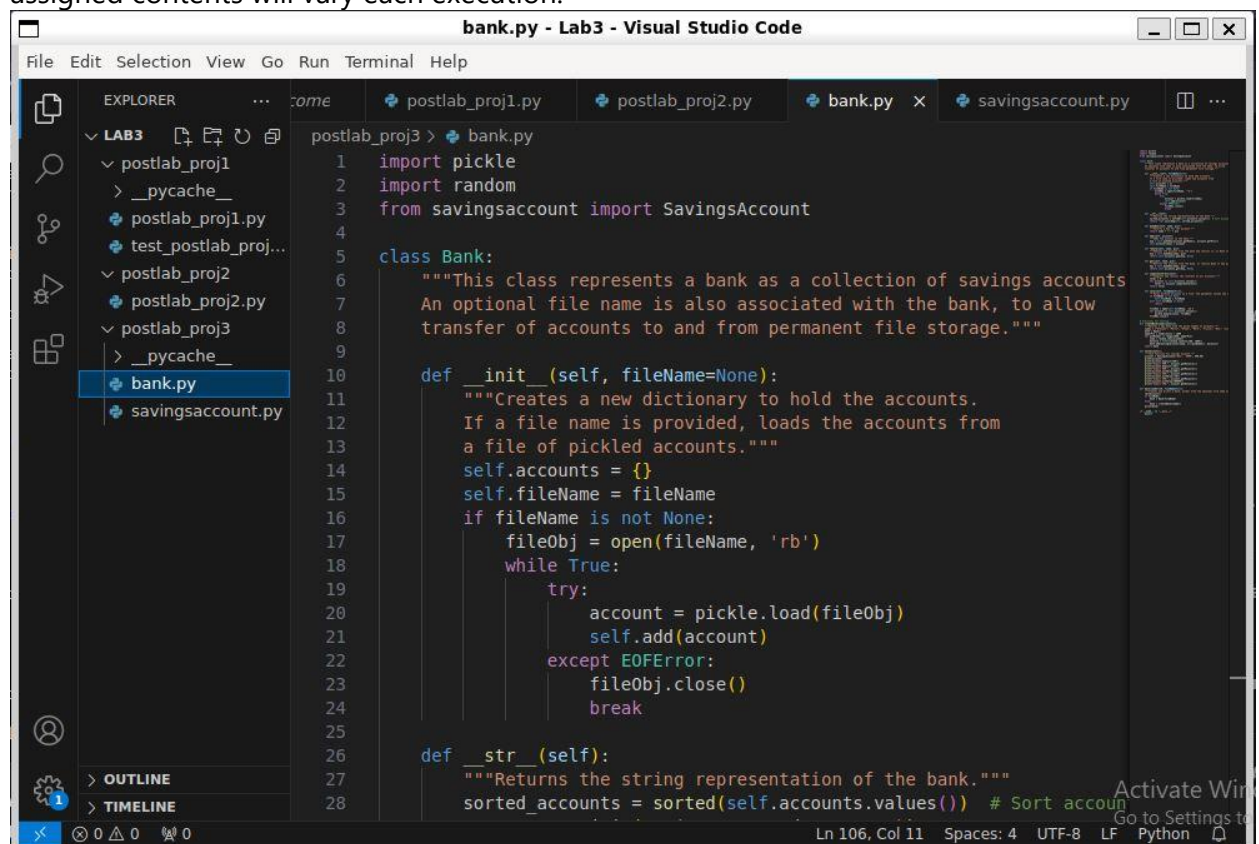
Students after sorting by name:
Name: Alice
Scores: 0 0 0
Name: Bob
Scores: 0 0 0 0 0
Name: Charlie
Scores: 0 0 0
Name: David
Scores: 0 0 0 0
Name: Eve
Scores: 0 0 0 0
```

Figure 4. output of postlab_proj1.py

3. The `str` method of the `Bank` class (in the file `bank.py`) returns a string containing the accounts in random order. Design and implement a change that causes the accounts to be placed in the string by order of name. (Hint: You will also have to define some methods in the `SavingsAccount` class, in the file `savingsaccount.py`.)

The provided code defines a banking system that manages savings accounts through a `Bank` class and a `SavingsAccount` class. The `Bank` class stores multiple savings accounts using a dictionary, allowing for functionalities like adding, removing, retrieving accounts, computing total interest, and saving/loading accounts from a file using Python's `pickle` module. The `SavingsAccount` class captures details for individual accounts, including name, PIN, and balance, while offering methods to deposit and withdraw funds, as well as compute accrued interest.

In the main function, a test savings account is created and manipulated to demonstrate functionality, followed by either loading existing accounts from a file or creating a specified number of new random accounts. The bank's contents, sorted by account holder names, are printed to the console, showcasing account details such as names, PINs, and balances. The output will reflect the results of the test account interactions—like deposit and withdrawal actions—along with the display of all accounts either loaded or newly created. If accounts are generated, their unique randomly assigned contents will vary each execution.



```
bank.py - Lab3 - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
LAB3
  postlab_proj1
    __pycache__
    postlab_proj1.py
    test_postlab_proj...
  postlab_proj2
    postlab_proj2.py
  postlab_proj3
    __pycache__
    bank.py
    savingsaccount.py

postlab_proj3 > bank.py
1 import pickle
2 import random
3 from savingsaccount import SavingsAccount
4
5 class Bank:
6     """This class represents a bank as a collection of savings accounts
7     An optional file name is also associated with the bank, to allow
8     transfer of accounts to and from permanent file storage."""
9
10    def __init__(self, fileName=None):
11        """Creates a new dictionary to hold the accounts.
12        If a file name is provided, loads the accounts from
13        a file of pickled accounts."""
14        self.accounts = {}
15        self.fileName = fileName
16        if fileName is not None:
17            fileObj = open(fileName, 'rb')
18            while True:
19                try:
20                    account = pickle.load(fileObj)
21                    self.add(account)
22                except EOFError:
23                    fileObj.close()
24                    break
25
26    def __str__(self):
27        """Returns the string representation of the bank."""
28        sorted_accounts = sorted(self.accounts.values()) # Sort account
```

Figure 5. postlab_proj3_codes

```
cerdan@DESKTOP-ICVPFLB: ~/LocalRepo/cpe106l-4/Lab/Lab3/postlab_proj3
(base) cerdan@DESKTOP-ICVPFLB:~/LocalRepo/cpe106l-4/Lab/Lab3$ cd postlab_proj3
(base) cerdan@DESKTOP-ICVPFLB:~/LocalRepo/cpe106l-4/Lab/Lab3/postlab_proj3$ python3 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
Name: Elena
PIN: 1005
Balance: 816.0
Name: Jill
PIN: 1000
Balance: 814.0
Name: Jill
PIN: 1004
Balance: 373.0
Name: Jill
PIN: 1006
Balance: 730.0
Name: Ken
PIN: 1003
Balance: 341.0
```

Figure 6. output of postlab_proj1.py