CS-UY 1114: Lab 13

Object Oriented Programming

You must get checked out by your lab CA **prior to leaving early.** If you leave without being checked out, you will receive 0 credits for the lab.

Restrictions

The Python structures that you use in this lab should be restricted to those you have learned in lecture so far. Please check with your teaching assistants in case you are unsure whether something is or is not allowed!

Create a new python file for each of the following problems.

Your files should be named *lab[num]q[num].py* **similar to homework naming conventions.**

Problem 1: *Off to Bobst*

Given the following Library class and test code, write the values that would be output for each print statement.

```
class Library:
    def __init__(self):
        self.books = []

    def __str__(self):
        return str(self.books)

    def add_book(self, book):
        self.books.append(book)

    def borrow_book(self, book):
        ind = self.available(book)
        if ind >= 0:
            self.books.pop(ind)
            return True
        else:
            return False

    def available(self, book):
        for i in range(len(self.books)):
            if self.books[i] == book:
                return i
        return -1

def main():
    lib = Library()
    lib.add_book("A Game of Thrones")
    lib.add_book("1984")
    lib.add_book("Moby Dick")
    print(lib.available("1984"))
    lib.borrow_book("Moby Dick")
    print(lib)
    print(lib.borrow_book("Becoming"))

main()
```

Problem 2: *Running on Steam*

Write a `VideoGameStore` class. At a `VideoGameStore`, we'll be able to purchase video game titles available at this store. Write the following methods for the `VideoGameStore` class:

```
def __init__(self, owner, games):
    """
    Initializes the owner of the store and the available games

    :param owner: str of this store's owner
    :param games: list of tuples of format (str game title, str game company)
    """
```

```
def __str__(self):
    """
    Returns a string where we display the owner of the store and all the video
games at this store
    """
```

The string representation of the `VideoGameStore` class should be the following:

```
Welcome to <owner>'s Video Game Store!
We have the following titles available for you:
    <game>
    <game>
```

```
def buy_game(self, game):
    """
    Allows the purchase of a game. If the game is purchasable, the game is removed
from the list of available titles and a success message is displayed. If not, a
failure message is displayed

    :param game: tuple of (str game title, str game company)
    :return bool: True if able to purchase game, False otherwise
    """
```

```
def view_games(self):
    """
    Displays all available video game titles
    """
```

`view_games()`'s output should be as follows:

```
Available titles:
    <game_title>
    <game_title>
```

You may use the following `main()` to test your class:

```
def main():
    bobs_store = VideoGameStore("Bob", [("Overwatch", "Blizzard"), ("Super Meat
Boy", "Team Meat"), ("Deltarune", "Toby Fox")])
    print(bobs_store)
    bobs_store.buy_game(("League of Legends", "Riot Games"))
    bobs_store.buy_game(("Overwatch", "Blizzard"))
    bobs_store.view_games()
```

Which should yield the following output:

```
Welcome to Bob's Video Game Store!
We have the following titles available for you:
    Overwatch by Blizzard
    Super Meat Boy by Team Meat
    Deltarune by Toby Fox

Sorry, we do not have League of Legends
Thank you for your purchase of Overwatch!
Available titles:
    Super Meat Boy
    Deltarune
```

Problem 3: *Flying Away*

This problem will be a simple model of an airplane with passengers and cargo.

Lets start by setting up the `Passenger` class. The passenger class should have the following attributes:

```
name
country of citizenship
number of bags
```

The Passenger class should have the following methods:

```
def __init__(self, name, citizenship, num_bags):
```

```
def __str__(self):
    """
    Returns a string representation of the passenger
    """
```

The format of the passenger string should be:

```
<name> is a citizen of <country> and has <num_bags> bags
```

```
def add_bag(self):
    """
    Adds a bag to the passenger
    """
```

```
def remove_bag(self):
    """
    Removes a bag from the passenger
    """
```

The Flight will be modeled as a class with the following attributes:

```
number
aircraft
passengers
departure
arrival
departure_time
arrival_time
```

```
def __init__(self, number, aircraft, departure_location, arrival_location,
departure_time, arrival_time):
```

```
def __str__(self):
    """
    Returns a string representation of the flight
    """
```

The format of the flight's string should be:

```
<number> <aircraft> departs <departure_location> at <departure_time> and arrives
at <arrival_location> at <arrival_time>
```

```
def print_passengers(self):
    """
    Prints the passenger list
    """
```

```
There are 3 passengers on this flight:
Bob is a citizen of France and has 2 bags
Alice is a citizen of France and has 1 bags
Henry is a citizen of UK and has 1 bags
```

```
def add_passenger(self, passenger):
    """
    Add a passenger to a flight. Duplicate passengers should not be allowed

    :param passenger: Passenger object
    :return bool: True if passenger was added, False otherwise
    """
```

```
def remove_passenger(self, passenger):
    """
    Removes a passenger from the flight

    :param passenger: Passenger object
    :return bool: True if passenger was removed, False otherwise
    """
```

```
def get_passenger_count(self):
    """
    Returns number of passengers on the flight
    """
```

Try using the following to test your code. Feel free to make your own tests!

```
def main():
    flight_eu = Flight(6783, "Airbus A320", "London", "Paris", "12:00", "14:00")
    bob = Passenger("Bob", "France", 2)
    alice = Passenger("Alice", "France", 1)
    henry = Passenger("Henry", "UK", 1)
    flight_eu.add_passenger(bob)
    flight_eu.add_passenger(alice)
    flight_eu.add_passenger(henry)
    flight_eu.add_passenger(henry)
    print(flight_eu)
    flight_eu.print_passengers()

    print("There are", flight_eu.get_passenger_count(), "passengers on this
flight")
    flight_eu.remove_passenger(bob)
    flight_eu.print_passengers()
```

Problem 4: *I Want A Pet*

Lets make a program that lets you adopt and manage pets!

Write a class called `Pet`. It should have:

- A constructor that takes in a name, a type, and an age as parameters. A `Pet` should have 4 attributes: `name`, `type`, `age`, and `fav_treats`. `fav_treats` should be a list, and may be initialized to the empty list.
- A `rename` method, which takes in a new name as a parameter, and sets the pet's name to the new name.
- A `birthday` method, which just increments the pet's age by one.
- An `add_treat` method, which takes in a treat as a parameter and adds it to the pet's `fav_treats` list.
- A `__str__` that returns a string in this format:

  ```
  <name> is a <type> that is <age> years old.
  Favorite treats:
      <fav_treats[0]>
      <fav_treats[1]>
      ...
  ```

Next, write a `main()` function that creates a list of pets, then will repeatedly ask user for a command and read user input until the user enters `Q` or `q`. Your program will do different things depending on what command the user entered. The valid commands are:

- `adopt`
    - Ask the user for the name of the pet, the type of pet (i.e., Cat, Dog, etc.), and the age of the pet.
    - Creates a new pet using these information and add it to the pets list
    - You may assume the user will always input an integer for the age of the pet
- `rename`
    - Ask the user for the old name of the pet and a new name
    - Change the name of the pet with the old name to the new name (must call the pet's `rename` method).
- `birthday`
    - Ask the user for the name of a pet
    - Increment the age of the pet (must call the pet's `birthday` method)
- `treat`
    - Ask the user for the name of a pet and the name of a treat.
    - Adds the treat to the pet's `fav_treats` list (must call the pet's `add_treat` method)
- `pets`
    - Displays every pet's information (must use `print` on each pet in the pets list)

A sample output of the entire program should look something like this:

```
Enter a command: adopt
What is the name of the pet? Momo
What type of pet is it? Dog
How old is the pet? 1

Enter a command: treat
Who is this treat for? Momo
What is the name of the treat? Biscuits

Enter a command: treat
Who is this treat for? Momo
What is the name of the treat? Bone

Enter a command: adopt
What is the name of the pet? Peluchito
What type of pet is it? Dog
How old is the pet? 13

Enter a command: rename
What is the current name of the pet? Peluchito
What is the new name of the pet? Peluche

Enter a command: birthday
Whose birthday is it? Peluche

Enter a command: pets
Momo is a Dog that is 1 years old.
Favorite treats:
        Biscuits
        Bone
```

```
Peluche is a Dog that is 14 years old.
Favorite treats:

Enter a command: q
```

Below are some cute pet pictures from our CAs and Professors!

Problem 5: Pirate World

This problem will model a pirate world where there exists `Pirate`s and `Crew`s that Pirates can be a part of.

Part 1A: Creating the `Pirate` class

The `Pirate` class will have the following attibutes:

```
name (str)              # The Pirate's name
status (str)            # Can be "Captain", "Member", or "Solo"
collected_loot (int)    # The total loot a Pirate owns
```

The `Pirate` class should implement the following methods:

```
def __init__(self, name, loot = 0):
```

Note: Every `Pirate` is assumed to have a status of `"Solo"` at initialization

```
    def __str__(self):
    """
    Returns a string representation of a Pirate
    """
```

The string representation should be:

```
<status> <name>
```

```
def update_loot(self, loot_change):
    """
    Updates the value of the pirate's collected loot.
    loot_change may be positive or negative
    collected_loot cannot go below 0
    """
```

```
def update_status(self, status):
    """
    Updates the pirate's status to the given status
    """
```

```
def get_loot(self):
    """
    Returns the pirate's collected loot
    """
```

Part 1B: Creating the `Crew` class

The `Crew` class will have the following attributes:

```
name (str)                    # Name of the Crew
pirates (list of Pirates)     # All pirate crew members
captain (Pirate)              # Pirate that's the captain of the Crew (only 1
captain allowed per Crew)
total_loot (int)              # Total loot based on sum of each pirate's collected
loot
```

The `Crew` class should implement the following methods:

```
def __init__(self, name):
```

Note: `Crew`'s are assumed to have no pirate members nor a captain at initalization. This also means the total loot will be 0 at initalization.

```
def __str__(self):
    """
    Returns a string representation of a Pirate Crew
    """
```

The string representation should be:

```
<name> under <captain> with the following crew members:
<Pirate in pirates OR "No crew members yet." if no pirates>
With a total loot of: <total_loot>
```

You may look at the example output for further examples on `Crew`'s string representation.

```
def add_crew_member(self, pirate, role):
    """
    Adds the given pirate to this crew's member list according to the given role
("Captain" or "Member").
    Updates the crew's total loot with the new pirate's loot and appropriate
pirate attributes.
    Returns False if adding a second captain, True if the operation is successful.
    """
```

```
def remove_crew_member(self, pirate):
    """
    Removes the given pirate from this crew's member list.
    Updates the crew's total loot and appropriate pirate attributes.
    Returns False if removing a pirate not in the crew, True if the operation is
successful
    """
```

Note: With each method, think about what attributes need to be updated in either `Crew` OR `Pirate` to keep the information consistent.

You may use the following `main()` definition to help test your code:

```
def main():
    # Creating our pirates
    pirate_jack = Pirate("Jack Sparrow")
    pirate_will = Pirate("Will Turner", 5)
    pirate_eliza = Pirate("Elizabeth Swann", 100)
    pirate_barbossa = Pirate("Hector Barbossa")
    pirate_rag = Pirate("Ragetti")
    pirate_pin = Pirate("Pintel", 10)

    # Updating the loot of some pirates
    pirate_barbossa.update_loot(40)
    pirate_eliza.update_loot(-20)
    pirate_rag.update_loot(-10)

    # Printing the current status of all our pirates
    all_pirates = [pirate_jack, pirate_will, pirate_eliza, pirate_barbossa,
pirate_rag, pirate_pin]
    for pirate in all_pirates:
        print(pirate, pirate.get_loot())
    print()

    # Creating our 2 crews
    crew1 = all_pirates[:3]
    crew2 = all_pirates[3:]

    jack_sparrow_crew = Crew("Jack Sparrow Crew")
    black_pearl_crew = Crew("Black Pearl Crew")

    # Print the empty crews
    print("---------------------------Printing empty crews------------------------
---")
    print(jack_sparrow_crew)
    print()
    print(black_pearl_crew)
    print()

    # Add members to each crew
    for pirate in crew1:
        is_pirate_added = jack_sparrow_crew.add_crew_member(pirate, "Captain")

        if not is_pirate_added:
            # Unable to add pirate due to attempting to add a 2nd captain
            # Intead, add the pirate as a member
            jack_sparrow_crew.add_crew_member(pirate, "Member")

    for pirate in crew2:
        is_pirate_added = black_pearl_crew.add_crew_member(pirate, "Captain")

        if not is_pirate_added:
            # Unable to add pirate due to attempting to add a 2nd captain
            # Instead, add the pirate as a member
            black_pearl_crew.add_crew_member(pirate, "Member")

    # Print the newly made crews
    print("---------------------------Printing the newly made crews---------------
------------")
    print(jack_sparrow_crew)
    print()
    print(black_pearl_crew)
    print()

    # Adding and removing a Pirate to a crew
    pirate_blackbeard = Pirate("Blackbeard", 50)

    # Removing a pirate not part of the crew
    black_pearl_crew.remove_crew_member(pirate_blackbeard)

    # Adding a Pirate to the crew then removing the pirate
    black_pearl_crew.remove_crew_member(pirate_barbossa)
    black_pearl_crew.add_crew_member(pirate_blackbeard, "Captain")
    print("---------------------------Printing the new crew after captain changes-
--------------------------")
    print(black_pearl_crew)
    print()
    print(pirate_barbossa)
```

Whose expected output should be:

```
Solo Jack Sparrow 0
Solo Will Turner 5
Solo Elizabeth Swann 80
Solo Hector Barbossa 40
Solo Ragetti 0
Solo Pintel 10

---------------------------Printing empty crews---------------------------
Jack Sparrow Crew under None with the following crew members:
No crew members yet.
With a total loot of: 0

Black Pearl Crew under None with the following crew members:
No crew members yet.
With a total loot of: 0
```

```
---------------------------Printing the newly made crews--------------------------
-
Jack Sparrow Crew under Captain Jack Sparrow with the following crew members:
Captain Jack Sparrow
Member Will Turner
Member Elizabeth Swann
With a total loot of: 85

Black Pearl Crew under Captain Hector Barbossa with the following crew members:
Captain Hector Barbossa
Member Ragetti
Member Pintel
With a total loot of: 50

---------------------------Printing the new crew after captain changes------------
---------------
Black Pearl Crew under Captain Blackbeard with the following crew members:
Member Ragetti
Member Pintel
Captain Blackbeard
With a total loot of: 60

Solo Hector Barbossa
```