Practical Questions

CSCA48 – Week 3

Question #1

a) Implement the stack adt with the following conditions:
   i. use a list and **must** use the list method append()
   ii. use a list and **must** use the list method insert()
   iii. do **not** use a list
   iv. don't use a list or a dictionary (try something creative, forget about making it sensible or efficient, have fun with it)

b) Implement a bounded stack – a stack with a fixed capacity. Here is an example use of the class *BoundedStack* that you will write.

```
if __name == "__main__":
    s = BoundedStack(2)
    try:
        s.push("Brian")
        s.push("Nick")
        s.push("Vc")
    except StackFullException:
        print("Stack is full!")
    try:
        print(s.pop())
        print(s.pop())
        s.pop()
    except StackEmptyException:
        print("Stack is empty!")
```

This program should produce the following output:

```
>>> Stack is full!
        Nick
        Brian
        Stack is empty!
```

You may assume that *StackFullException* and *StackEmptyException* have been implemented as sub-classes of *Exception*. Hint: do part a) first, use the power of inheritance.

c) Write a function called binary_convertion_plus that takes 2 parameters both as int. The first int is the number will be converted where the second int is the length of the binary string; we call it **n** here. This function returns true if the number can be converted into a binary less or equal to **n**, otherwise return false.

   You may want to use the BoundedStack from part c)

d) Implement a priority queue

e) Based on the implemented from d), Nick decided to create a new type of queue called *UnFairQueue*, it contains methods:
   - all methods from normal queue
   - reverse() – reverse the queue
   - to_front(element) – bump an existing element to the head of the queue so that it will be remove next (if the element in not in queue, do nothing)

   Help Nick to complete the implementation of *UnFairQueue*

f) (Challenge)

Brian just heard about Nick's *UnFairQueue* so decides to also make his own queue named *CrazyQueue* with following methods:
   - all methods from *UnFairQueue* queue
   - convert() – convert the queue into stack, or back into queue
     hint! Carefully think of the difference between them. Only work on one of add/remove method is enough.
   - backtrack() – reset the queue one step backward. If a change has been made (add/remove), the backtrack method can go back to where the queue was before that change.

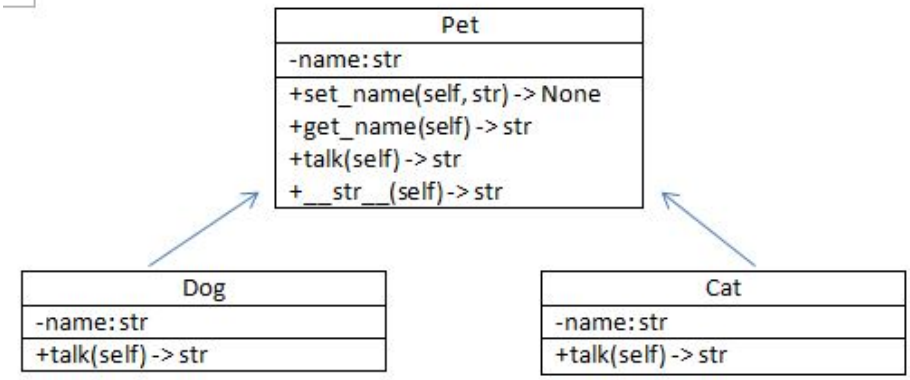
   Help Brian to implement the *CrazyQueue*

g) **(Challenge)**
   - Implement a queue using two stacks
   - Implement a stack using two queues

h) Design carefully and draw an UML diagram representing classes below and the relationship between them
   - The biological phylum (https://en.wikipedia.org/wiki/Phylum), or at least the first few levels
   - Things you might find on a farm (e.g., cow, horse, farmer tractor, tree)
   - The objects in the room around you right now

i) Design carefully and draw an UML diagram representing classes below and the relationship between them

   A teacher has a list of students that he/she teaches. Teachers are also able to add/remove any student to/from the list. Teacher has employee number whereas student has student number. One thing common between them is the fact that they are considered a person. A person should have name, age, and gender.




```
>>> a = Pet("Poro")
>>> b = Dog("Nasus")
>>> c = Cat("Rengar")
>>> print(a)
    My name is Poro. I say Hello!
>>> print(b)
    My name is Nasus. I say Woof!
>>> print(c)
    My name is Rengar. I say Moew!
```

j) Carefully read the UML diagram and outputs given above, implement class Pet, Dog, and Cat
(Hint: the – sign means private whereas + sign means public)

k) Remember Ex8 from CSCA08? Here is the link
http://www.utsc.utoronto.ca/~bharrington/csca08/exercises/exercise8.pdf
Draw an UML diagram for lightSwitch and switchboard
(No codes are needed)

l) Remember Ex9 from CSCA08? Here is a sample solution of it

```
import math
class Parallelogram():
    def __init__(self, base, side, theta):
        self._base = base
        self._side = side
        self._theta = theta
        self._name = "Parallelogram"

    def area(self):
        radians = math.radians(self._theta)
        area = self._base * self._side * math.sin(radians)
        return area

    def bst(self):
        bst = [self._base, self._side, self._theta]
        return bst

    def __str__(self):
        return "I am a " + self._name + " with area " + str(self.area())

class Rectangle(Parallelogram):
    def __init__(self, base, side):
        Parallelogram.__init__(self, base, side, 90)
        self._name = "Rectangle"

class Rhombus(Parallelogram):
    def __init__(self, base, theta):
        Parallelogram.__init__(self, base, base, theta)
        self._name = "Rhombus"

class Square(Rhombus):
    def __init__(self, base):
        Rhombus.__init__(self, base, 90)
        self._name = "Square"
```

Draw an UML diagram representing these classes

m) <u>(Challenge)</u>

Design and draw UML diagram. Explain your diagram to your TA why you choose those classes to exist and what are the advantages of having them as separate classes. There is not a correct answer for this question; but to get a full challenge point, you must provide at least a **good** design and explanation is needed.

A good design of this question should have no less than 15 classes.
A better design should have more than 20 classes.

Vincent is a student from University of Toronto Scarborough Campus. He watched the movie Bat-Man and decided to raise some bats; so he currently has a nest of bats at home. Some of the bats he has are a new type of bat called DragonBat that only eats dragonfly; normal bats eat all kinds of insect. Brian Cheng is a professor from University of Toronto St George Campus. He watched the movie Spider-Man. You guessed it; he now has a cage of spiders. Some of the spiders he has only eat dragonfly and they are called DragonSpider. You guessed it again!

Hint: In Python, a class can have multiple parents
A bat has 2 wings whereas Dragonfly has 4 wings.
Insects only have 6 legs, but spiders have 8 legs.

Discussion Question

1. What is the difference between abstract data type and encapsulation of data?
2. Python doesn't have any mechanism to enforce that instance variables are actually private unlike other languages. Putting an underscore before a name encourages developers not to mess with it, but it doesn't actually stop anyone. What are the advantages/disadvantages of this approach?

3. If you were designing your own programming language, how would you have it handle privacy? Would that decision affect the popularity of your language? How so?

Logic Question **(Challenge)**

When no one shows up to Nick's office hours, he gets bored. So he's developed his own solitaire game. Starting with a randomly shuffled pack of 52 standard playing cards, he deals out each card face up, one at a time. As he deals each card, he counts "Ace, 2, 3, 4, 5, 6, 7, 8, 9, Jack, Queen, King", and then returns to Ace. Saying one card value each time he flips over a card. If he says the value of the card he just flipped over, he loses. If he can get through all 52 cards in the deck without flipping a card with the same value he's saying, he wins[1].

What is the probability that Nick will win any given game?

Bonus 1: If he flips over one card every second. How long will Nick expect to play before winning?

Bonus 2: Write a simulation to check and see if your answers hold up.

[1] Nick has a very low threshold for excitement.