UNIVERSITY OF TORONTO

Faculty of Arts and Science

term test #1, Version 3
CSC1481S

Date: Wednesday February 7, 2:10-3:00pm or 3:10–4:00 p.m.
Duration: 50 minutes
Instructor(s):
AbdulAziz Alhelali
Arnamoy Bhattacharyya
Danny Heap

No Aids Allowed

Name:

utorid:

U of T email:

Please read the following guidelines carefully!

- **Please write your name, utorid, and student number on the front of this exam.**
- This examination has **3** questions. There are a total of **6 pages, DOUBLE-SIDED**.
- Answer questions clearly and completely.
- You will receive 20% of the marks for any question you leave blank or indicate "I cannot answer this question."

Take a deep breath.
This is your chance to show us
How much you've learned.

We **WANT** to give you the credit

Good luck!

1. **[10 marks]** (≈ 25 minutes) Below we have an implementation of class Vehicle. On the following pages, implement two subclasses:

Truck has a load capacity of 5, 10, or 20 tons, which does not need to be in their string representation. Their monthly expenses are estimated as 30 * capacity dollars.

Car has a seating capacity of 4 or 7 persons, which does not need to be in its string representation. Their monthly expenses are estimated at $1,000.

Your implementation should provide a string representation of Vehicle objects that shows the plate number and monthly expenses. You do **not** need to provide __eq__ methods.

You must write docstrings for each class and method with type signatures/annotations for parameters and public attributes given in the format of the example code below.

No examples (such as doctests) are required. Indicate which methods are overriding others with a brief comment in the docstring of the method.

```
class Vehicle:
    """ Represent an Vehicle information

    plate_number - plate_number
    """
    plate_number: str

    def __init__(self, plate_number: str) -> None:
        """ Initialize a new Vehicle
        """
        self.plate_number = plate_number

    def __str__(self) -> str:
        """
        Return a string representation of the Vehicle information.
        """
        return ("Plate Number: {}\nMonthly Expenses:{}"
                .format(self.plate_number, self.get_monthly_expenses()))

    def get_monthly_expenses(self) -> float:
        """
        Return the monthly expenses of the Vehicle.
        """
        raise NotImplementedError
```

Solution

```
class Truck(Vehicle):
    """ Represent a Vehicle of type Truck information

    capacity - load capacity which is either 5 Ton, 10 Ton or 20 Ton
    """
    load_capacity: int

    def __init__(self, plate_number: str, load_capacity: int) -> None:
        """ Initialize a new Vehicle of type Truck with load capacity
        Extends Vehicle.__init__
        """
        Vehicle.__init__(self, plate_number)
        self.load_capacity = load_capacity

    def get_monthly_expenses(self) -> float:
        """
        Return the predicted monthly expenses of the Truck.
        Overrides Vehicle.get_monthly_expenses
        """
        return self.load_capacity * 300

class Car(Vehicle):
    """ Represent a Vehicle of type Car information

    seating_capacity - seating capacity which is either 4 or 7 seats
    """
    seating_capacity: int

    def __init__(self, plate_number: str, seating_capacity: int) -> None:
        """ Initialize a new Vehicle of type Car with seating capacity
        Extends Vehicle.__init__
        """
        Vehicle.__init__(self, plate_number)
        self.seating_capacity = seating_capacity

    def get_monthly_expenses(self) -> float:
        """
        Return the predicted monthly expenses of the Car.
        Overrides Vehicle.get_monthly_expenses
        """
        return 1000
```

2. **[6 marks]** (≈ 10 minutes) **Linked lists**: Below is an implementation of classes **LinkedListNode** and **LinkedList**, which you've seen in lecture since last week. At the bottom of the next page, write the body of method **sum**. Use only LinkedList methods implemented here, and do not use Python **lists**! **NB**: built-in **sum** function **will not** work here.

```
from typing import Optional, Any

class LinkedListException(Exception):
    pass

class LinkedListNode()                                                          :
    """ Node to be used in linked list

    next_ - successor to this LinkedListNode
    value - data represented by this LinkedListNode
    """
    next_: Optional["LinkedListNode"]
    value: object

    def __init__(self, value: object,
                 next_: Optional["LinkedListNode"] = None) -> None:
        """ Create LinkedListNode self with data value and successor next

        >>> LinkedListNode(5).value
        5
        >>> LinkedListNode(5).next_ is None
        True
        """
        self.value, self.next_ = value, next_

    def __str__(self) -> str:
        """ Return a user-friendly representation of this LinkedListNode.

        >>> n = LinkedListNode(5, LinkedListNode(7))
        >>> print(n)
        5 ->7 ->|
        """
        cur_node = self
        result = ''
        while cur_node is not None:
            result += '{} ->'.format(cur_node.value)
            cur_node = cur_node.next_
        return result + '|'

class LinkedList:
    """ Collection of LinkedListNodes

    front - first node of this LinkedList
    back - last node of this LinkedList
    size - number of nodes in this LinkedList, >= 0
    """
    front: Optional[LinkedListNode]
    back: Optional[LinkedListNode]
    size: int
    def __init__(self) -> None:
        """ Create an empty linked list.
        """
        self.front, self.back, self.size = None, None, 0

    def prepend(self, value: object) -> None:
        """ Insert value before LinkedList self.front.

        >>> lnk = LinkedList()
        >>> lnk.prepend(0)
        >>> lnk.prepend(1)
        >>> lnk.prepend(2)
        >>> str(lnk.front)
        '2 ->1 ->0 ->|'
        >>> lnk.size
        3
        """
        self.front = LinkedListNode(value, self.front)
        if self.back is None:
            self.back = self.front
        self.size += 1

    def sum(self) -> int:
        """ Return sum of int values in LinkedList self.
        Raise LinkedListException if any value is not an int.

        >>> lnk1 = LinkedList()
        >>> lnk1.prepend(1)
        >>> isinstance(lnk1.front.value, int)
        True
        >>> lnk1.prepend(2)
        >>> lnk1.sum()
        3
        """
```

Solution

```
        sum = 0
        self_node = self.front
        while self_node is not None:
            if not isinstance(self_node.value, int):
                raise LinkedListException("bad value!")
            sum += self_node.value
            self_node = self_node.next_
        return sum
```

3. **[5 marks]** (≈ 10 minutes) **queues:** Three empty Queues are created and then loaded with some strings:

```
q1 = Queue()
q1.add("N")
q1.add("P")
q2 = Queue()
q2.add("O")
q2.add("I")
q3 = Queue()
q3.add("T")
```

Choose a sequence of commands from the table below to load **q3** so that it contains "**P**", "**O**", "**I**", "**N**", "**T**", in order, with "**T**" added last. When you're done the code at the bottom of the page should run as stated.

You may not use **any** other Python expressions except those in the table. You may use some of the commands in the table more than once, some of them not all.

Hint: Try to draw what the queues contain to start with, and come up with the sequence of actions needed (in picture form, crossing out elements you remove) before writing any python code.

q1.remove()	q1.add(q2.remove())	q1.add(q3.remove())
q2.remove()	q2.add(q1.remove())	q2.add(q3.remove())
q3.remove()	q3.add(q1.remove())	q3.add(q2.remove())

```
result = ""
while not q3.isempty():
    result = result + q3.remove()
result == "POINT"  # this should be True
```

Solution

```
q2.add(q1.remove())
q2.add(q3.remove())
q3.add(q1.remove())
q3.add(q2.remove())
q3.add(q2.remove())
q3.add(q2.remove())
q3.add(q2.remove())
```