UNIVERSITY OF TORONTO

Faculty of Arts and Science

term test #1, Version 2

CSC1481S

Date: Wednesday February 7, 10:10–11:00pm or 11:10–noon

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 8 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]** ($\approx$ 25 minutes) Below we have an implementation of class SparePart. On the following pages, implement two subclasses:

   LocalSparePart has a batch number and manufacturing date, which do not need to be in its string representation. Their selling price is 1.2 * cost.

   ImportedSparePart has a supplier and shipping expenses, which do not need to be in its string representation. Their selling price is (1.4 * cost) + (shipping expenses * 1.6).

   Your implementation should provide a string representation of SparePart objects that shows the part number, the description, the cost, and the selling price. 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 SparePart:
    """ Represent a SparePart information

    part_number - part number
    description -  description
    cost - cost
    """
    part_number: str
    description: str
    cost: float

    def __init__(self, part_number: str, description: str, cost: float) -> None:
        """ Initialize a new SparePart
        """
        self.part_number, self.description = part_number, description
        self.cost = cost

    def __str__(self) -> str:
        """ Return a string representation of the SparePart information.
        """
        return ("Part Number: {}\\nDescription:{} "
                "\\nCost:{}\\nSelling Price: {}"
                .format(self.part_number, self.description, self.cost,
                        self.get_selling_price()))

    def get_selling_price(self) -> float:
        """
        Return the selling price of the SparePart.
        """
        raise NotImplementedError
```

Solution

never, **ever**, write below this line...

```
class LocalSparePart(SparePart):
    """ Represent a SparePart of type LocalSparePart information

    manufact_date  - manufacturing date
    batch_number - batch number
    """
    manufact_date: str
    batch_number: str

    def __init__(self, part_number: str, description: str, cost: float,
                 manufact_date: str, batch_number: str) -> None:
        """ Initialize a new LocalSparePart with manufacturing date and
        batch number
        Extends SparePart.__init__
        """
        SparePart.__init__(self, part_number, description, cost)
        self.manufact_date, self.batch_number = manufact_date, batch_number

    def get_selling_price(self) -> float:
        """ Return the selling price of the LocalSparePart.

        Overrides SparePart.get_selling_price
        """
        return self.cost * 1.2

class ImportedSparePart(SparePart):
    """ Represent a SparePart of type ImportedSparePart information

    supplier   - supplier name
    shipping_expenses - shipping expenses
    """
    supplier: str
    shipping_expenses: float

    def __init__(self, part_number: str, description: str, cost: float,
                 supplier: str, shipping_expenses: float) -> None:
        """ Initialize a new ImportedSparePart with supplier name and
        shipping expenses

        Extends SparePart.__init__
```

```
        """
        SparePart.__init__(self, part_number, description, cost)
        self.supplier, self.shipping_expenses = supplier, shipping_expenses

    def get_selling_price(self) -> float:
        """ Return the selling price of the ImportedSparePart.

        Overrides SparePart.get_selling_price
        """
        return self.cost * 1.4 + self.shipping_expenses * 1.6
```

2. **[6 marks]** ($\approx$ 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 **concat**. Use only LinkedList methods implemented here, and do not use Python **lists**!

```
from typing import Union, 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_: Union["LinkedListNode", None]
    value: object

    def __init__(self, value: object,
                 next_: Union["LinkedListNode", None] = 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: Union[LinkedListNode, None]
    back: Union[LinkedListNode, None]
    size: int
```


never, **ever**, write below this line...

```
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 concat(self, other: "LinkedList") -> None:
    """ Concatenates other into self and sets other to contain no values.
    (that is, other should have its .front attribute None)
    Raise exception if other starts empty.

    >>> lnk1 = LinkedList()
    >>> lnk1.prepend(2)
    >>> lnk1.prepend(1)
    >>> lnk1.prepend(0)
    >>> lnk2 = LinkedList()
    >>> lnk2.prepend(5)
    >>> lnk2.prepend(4)
    >>> lnk2.prepend(3)
    >>> lnk1.concat(lnk2)
    >>> print(lnk1.front)
    0 ->1 ->2 ->3 ->4 ->5 ->|
    >>> print(lnk2.front)
    None
    """
```

Solution

```
        if other.size == 0:
            raise LinkedListException("list must not be empty")
        if self.size == 0:
```

```
            self.front = other.front
        else:
            self.back.next_ = other.front
        self.back = other.back
        self.size += other.size
        other.front, other.back, other.size = None, None, 0
```

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

```
q1 = Queue()
q1.add("A")
q1.add("F")
q2 = Queue()
q2.add("L")
q2.add("O")
q3 = Queue()
q3.add("T")
```

Choose a sequence of commands from the table below to load **q3** so that it contains "**F**", "**L**", "**O**", "**A**", "**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 == "FLOAT"  # 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())
```


never, **ever**, write below this line...