UNIVERSITY OF TORONTO
Faculty of Arts and Science

Midterm 1 Solutions
CSC148H1F – L0101 (Horton)

October 21, 2016 (**50 min.**)
Examination Aids: Provided aid sheet (back page, detachable!)

Name:

Student Number:

Please read the following guidelines carefully!

- Please write your name on the front **and back** of the exam.
- This examination has **4** questions. There are a total of **11 pages, DOUBLE-SIDED**.
- You may always write helper functions/methods unless explicitly asked not to.

Take a deep breath.

This is your chance to show us

How much you've learned.

We **WANT** to give you the credit

That you've earned.

A number does not define you.

Good luck!

1. The following questions test your understanding of the terminology and concepts from the course. You may answer in either point form or full sentences; **you do not need to write much to get full marks**!

   (a) **[2 marks]** Suppose we are designing a class called `Club` for keeping track of the members of a club. We plan to have a private attribute that is a list of the members' email addresses. Write the portion of the docstring for class `Club` that describes this attribute (you can choose its name).

Solution

```
@type emails: list[str]
    The email addresses of the members of the club.
```

   (b) **[1 mark]** Suppose the `Club` class will also have a method called `sign_up` which is called when a new member joins the club, and will add their name to the front of the list of members. Assuming that we are committed to using a list of email addresses, what simple change to the class will make the `sign_up` method run faster?

Solution

Inserting at the front of a Python list is slower than inserting at the end, so we can speed this method up by adding new members at the end of the list.

   (c) **[1 mark]** Name a class from Assignment 1 that was abstract (approximate name is fine):

Solution

Class `Container` was abstract, as was `Scheduler`.

   (d) **[1 mark]** What is the one thing that client code should never do with an abstract class?

Solution

Client code should never construct an instance of an abstract class.

   (e) **[2 marks]** Sometimes a class is so abstract that it has no attributes and no method bodies. What do we gain by defining such a seemingly useless class?

Solution

From the point of view of client code, we know what all instance of any descendant of the abstract class have in common. This allows client code to take an instance and operate on it regardless of which specific kind it is.

From the point of view of a child class, the abstract parent defines exactly what the child is responsible for implementing.

   (f) **[2 marks]** Suppose we have a `Queue` class (a regular queue, not a priority queue), and it includes this method:

```
def num_items(self):
    count = 0
    while not self.is_empty():
        temp = self.dequeue()
        count += 1
        self.enqueue(temp)
    return count
```

   Explain why the loop in this method is an infinite loop.

Solution

The method is trying to restore the queue to its state at the start of the call. But by doing that in the same loop that is removing the items, the queue never becomes empty and the loop never ends.

(g) **[1 mark]** Suppose we fix method `num_items`. But rather than the usual way of calling a method, such as `q.num_items()`, we want to be able to call it as `len(q)`. How must we change the method to make this possible?

Solution

We simply need to change the method's name to `__len__`.

2. [**4 marks**] We want to create a subclass of `Queue` called `DoubleQueue`, which has the following differences:

   - It has a new attribute `is_special`, which is a **function** that takes an object and returns a boolean value. This attribute is initialized from a parameter to the `__init__` method.
   - When enqueueing a new item into a `DoubleQueue`, its `is_special` attribute is first called on the item. If this returns `True` the item is goes into the `DoubleQueue` twice, otherwise it goes in once as usual.

   Here is a Python interactive session that demonstrates the usage of `DoubleQueue`.

```
>>> def loud_word(s):
...     return s in ['crash', 'bang', 'bellow', 'holler', 'honk']
...
>>> q = DoubleQueue(loud_word)
>>> q.enqueue('quiet')
>>> q.enqueue('honk')
>>> q.enqueue('peaceful')
>>> q.dequeue()
'quiet'
>>> q.dequeue()
'honk'
>>> q.dequeue()
'honk'
>>> q.dequeue()
'peaceful'
```

In the space below, show how to override the relevant `Queue` methods in the `DoubleQueue` class. You *must* properly call superclass methods when appropriate.

Solution

Docstrings were not required, but are provided here to help you understand the solution.

Notice that the solution has only 2 + 3 lines of code.

```
class DoubleQueue(Queue):
    """A first-in-first-out (FIFO) collection in which "special" items
    are added to the queue twice rather than just once.

    === Attributes ===
    @type is_special: Callable[[Object], bool]
      If self.is_special(x) is true, then x should be added to the queue
      twice when it is enqueued.
    """

    def __init__(self, is_special):
        """Initialize this to an empty DoubleQueue.

        @type self: DoubleQueue
        @type is_special: Callable[[object], bool]
            Determines whether an object to be added to this queue is "special"
            and therefore should be added twice.
        @rtype: None
        """
        Queue.__init__(self)
        self.is_special = is_special
```

```
    def enqueue(self, item):
        """Enqueue twice if it's special.

        >>> def loud_word(s):
        ...     return s in ['crash', 'bang', 'bellow', 'holler', 'honk']
        ...
        >>> q = DoubleQueue(loud_word)
        >>> q.enqueue('quiet')
        >>> q.enqueue('honk')
        >>> q.enqueue('peaceful')
        >>> q.dequeue()
        'quiet'
        >>> q.dequeue()
        'honk'
        >>> q.dequeue()
        'honk'
        >>> q.dequeue()
        'peaceful'
        """
        if self.is_special(item):
            Queue.enqueue(self, item)
        Queue.enqueue(self, item)
```

3. **[9 marks]**

   You are responsible for designing a class to keep track, for each value in a set of values, the number of times it occurs. (This is called a "frequency distribution".) Here is an example of how we want to use it:

```
>>> d = Distribution('Grades for Lab 3', ['A', 'B', 'C', 'D', 'F'])
>>> d.add_occurrence('A')
>>> d.add_occurrence('C')
>>> d.add_occurrence('B')
>>> d.add_occurrence('A')
>>> d.add_occurrence('B')
>>> d.add_occurrence('B')
>>> d.num_occurrences('D')
0
>>> d.num_occurrences('B')
3
>>> d.num_occurrences('A')
2
```

   Below and on the next page, we have a very incomplete class design for this class. You have four tasks:

   - In the class docstring fill in all the **attributes** of the `Distribution` class. You may choose any reasonable way to store the necessary data. Make all attributes private.
   - Implement the `__init__` method. A method docstring is not necessary.
   - Implement method `add_occurrence`. It's up to you to decide what happens when this method is called on an item that is was not in the list provided when the `Distribution` was constructed.
   - Complete the docstring for `add_occurrence`.

   Ensure that the code above would run, assuming we defined the additional method `num_occurences`.

Solution

Docstrings here are more complete that was required on the test.

Notice that this solution has only 4 + 1 lines of code. There are many other ways to implement this class, including keeping a list of frequencies and a "parallel" list of the corresponding values. For example, in position 0 of one list we might have 'A' and in position 0 of the other we would have the frequency of 'A'.

There are also other options for how to handle an a call to `add_occurrence` with a value that wasn't in the list provided at the time of construction. Here, we took the easy way out and just stated a precondition that disallows such a call. Other options include failing silently, and raising an error.

```
class Distribution:
    """A frequency distribution.

    === Attributes ===
    @type _description: str
       A description of what this is a distribution of.
    @type _frequencies: dict[str, int]
       Each value is the number of occurrences of its associated key
       that have been recorded.
    """
    def __init__(self, description, values):
        """Initialize this Distribution with frequency 0 for all values.

        @type self: Distribution
        @type description: str
            A description of what this is a distribution of.
        @type values: list[str]
            The values whose frequency will be recorded.
        @rtype: None
        """
        self._description = description
        self._frequencies = {}
        for item in values:
            self._frequencies[item] = 0

    def add_occurrence(self, value):
        """Record that <value> occurred once / once more.

        Precondition: <value> is among those given when this Distribution
        was constucted.

        @type self: Distribution
        @type value: str
        @rtype: None
        """
        self._frequencies[value] += 1
```

4. For this question, you should refer to the documentation of the `LinkedList` class found on the aid sheet. You may use all attributes of the `LinkedList` and `_Node` classes, as well as their constructors. You may also access the `__str__` method for `LinkedList`, but no other methods.

   (a) **[3 marks]** Suppose we have done this:

   ```
   >>> linky = LinkedList([6, 12, 18, 24, 30])
   ```

   Fill in the gaps below to show the the value of each Python expression. If an expression raises an error, explain why.

   ```
   >>> type(linky.next.next.item)
   ```

   ```
   >>> linky.next.next.item
   ```

   ```
   >>> type(linky.item.next)
   ```

   ```
   >>> linky.item.next
   ```

   ```
   >>> type(linky._first.next.item)
   ```

   ```
   >>> linky._first.next.item
   ```

Solution

Note: You did not need to know the exact syntax of errors raised or of how types are reported.

(1) Since the type of `linky` is `LinkedList`, its only attribute is `_first`. `linky` is not a `_Node`, so it does not have an attribute `next`. So both of these fail:

```
>>> type(linky.next.next.item)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'LinkedList' object has no attribute 'next'

>>> linky.next.next.item
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'LinkedList' object has no attribute 'next'
```

(2) Similarly, these fail because `linky` does not have an attribute `item`.

```
>>> type(linky.item.next)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'LinkedList' object has no attribute 'item'
```

```
>>> linky.item.next
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'LinkedList' object has no attribute 'item'
```

(3) Finally something that makes sense! `linky` has an attribut `_first`, which is a `_Node`, and so it has a `next`. That in turn references a `_Node`, and so has an `item`, so these succeed:

```
>>> type(linky._first.next.item)
<class 'int'>

>>> linky._first.next.item
12
```

(b) **[1 mark]** In the gap below, write a one-line snippet of code to remove the first node in the linked list and ensure the output shown.

```
>>> linky = LinkedList([5, 8, 2, 9, 3])

>>> print(linky)

[5 -> 8 -> 2 -> 9 -> 3]

>>>                                                    # YOUR WORK GOES ON THIS LINE

>>> print(linky)

[8 -> 2 -> 9 -> 3]
```

Solution

```
>>> linky._first = linky._first.next
```

(c) **[1 mark]** Why does the time required to insert at the front of a linked list **not** grow in proportion to the length of the list?

Solution

Inserting at the front of a linked list with a million nodes takes no more time than inserting at the front of a linked list with 1 node, because we don't need to look at all the nodes. We just need to do a fixed amount of work to link the new node in, by updating the `_first` attribute of the linked list and creating and properly setting the attributes of a new node.

Use this page for rough work. If you want work on this page to be marked, please indicate this clearly *at the location of the original question.*

Name:

	Q1	Q2	Q3	Q4	Total
Grade					
Out Of	10	4	9	5	28