==========

CSCI 5300: Software Design
Spring 2015: Phil Pfeiffer
Assignment 5: Python – singletons
Terms: As individuals or in pairs – no trading of code beyond this, please
Due date: Saturday night, 18 April, close of business
Value: 13 points –points as marked

==========

Background. I started using Python in 5300, in part, to get people to think through design patterns instead of cutting and pasting them into their codes. Implementations of patterns for mainstream compiled languages have been widely available on websites for at least ten years. Implementations of patterns for Python, on the other hand, have only started to become readily available[1]. Until now, I've been content to give culminating assignments that asked people to implement a half dozen or so patterns, in Python, relative to a few simple problem domains. While I may give variants of those problems in a final assignment, I now see a need to assign more complex pattern-related tasks.

This assignment involves one such task. During the past week, I've discovered that different implementations of the singleton pattern have quite different characteristics. This includes two strategies for implementing singletons:

- The first strategy, shown in singletons as statics.py, treats all shared mutable data as static. This approach allows individual instances of a class to retain separate identities.
- The second, shown in singletons via metaclasses.py, uses a metaclass to force all instantiations of a singleton class to reference a common object. This approach eliminates the need for mutable static data while blurring distinctions between instantiations of the singleton.

While I didn't intend to present metaclasses this semester[2], the metaclass strategy for singleton implementation merits study. Its loss of instance identity, however, can affect its usefulness, as shown below.

Requirements. This assignment assumes the use of a platform that supports

- a Python v3.4 interpreter;
- a text editor, like vi, notepad++, or PFE32, with line numbers;
- GUI environment that supports
  - the ability to do screen captures of individual windows—e.g., Windows with the Windows snipping tool; and
  - the ability to store captures in standard formats—e.g., as .gifs or .pngs or in Word for Windows files

Overview. Your primary task for this assignment is to develop a singleton collection class for text files that can memento-ize its state while preserving the identities of its instances. The class must support the following methods:

- an **__init__** method that associates all instances of this class with a (possibly empty) list of file specifications.
  - __init__ should accept one argument: a possibly empty list of file specifications, each of which consists of
    - a file name (required)
    - a position value, which specifies the position to which __init__ must set the file's internal cursor (default: position 0, the head of the file)
    - a references value, which lists those objects that, in addition to the caller, have open references to this file (default: empty set)
  - The method should operate as follows::
    - If the file specifications list repeats filenames, __init__ must raise an exception
    - Otherwise, for each specification in the list of file specifications, __init__ should
      - add the file to the shared collection if the file is not yet in the collection
      - seek to the specified position in the file

[1] See, for example, https://www.youtube.com/watch?v=1Sbzmz1Nxvo and https://www.youtube.com/watch?v=tJXhtncDBu4

[2] While crafting this assignment, I learned a few things about metaclasses from http://stackoverflow.com/questions/6760685/creating-a-singleton-in-python and http://eli.thegreenplace.net/2011/08/14/python-metaclasses-by-example. I recommend reviewing both.

        - add the contents of the references argument, together with a reference to the current object, to the list of references.
- a **seek** method that seeks to a caller-specified position in a caller-specified file.
    - This method should accept two arguments: a file name and an offset from byte 0 to which to seek
    - This method should operate as follows:
        - The method should raise an exception if either the specified file is not in the collection or the current instance lacks an open reference to the file
        - Otherwise, the method should
            - use seek to update the file's cursor
            - raise an exception if seek failed
- a **readline** method that reads the next line from a caller-specified file.
    - This method should accept one argument: a file name
    - This method should operate as follows:
        - The method should raise an exception if either the specified file is not in the collection or the current instance lacks an open reference to the file
        - Otherwise, the method should
            - use readline to read up to the end of the current line
            - raise an exception if readline failed; otherwise, return the result from readline
- a **close** method that releases the instance's handle on a specified file.
    - This method should accept one argument: a file name
    - This method should operate as follows:
        - The method should raise an exception if either the specified file is not in the collection or the current instance lacks an open reference to the file
        - Otherwise, it should
            - remove the current instance from the collection of open references associated with the file
            - if this was the last instance to reference the file, close the file and remove it from the collection
- a **__repr__** method that acts as a memento pattern implementation: i.e., that, when processed with eval, yields an object that has the same state as the repr-ized object.

The code, moreover, must represent text files as instances of a class that

- derives from the variant of the singleton metaclass that segregates class instances by filename, rather than class
- supports an **__init__** method that
    - takes two arguments, a filename (required) and an initial position (default: 0)
    - opens the file if the file isn't yet opened
    - seeks to the specified initial position
    - raises exceptions on error
- supports **seek** and **readline** methods like those supported by the text file collection class, but without the filename argument
- supports a **close** method that closes the file
- supports a **__repr__** method that acts as a memento-pattern implementation for class instances.

The code must generate fully descriptive error messages for all exceptions, including messages that specify the class and routine in which the exceptions occurred, the parameters for the erroneous operation, and any value attributes from exceptions returned by the Python run-time system.

The following sample output from my implementation of the assignment should help to clarify these requirements:

```
> import os
> os.chdir("H:\\phil\\classes-ETSU\\csci5300\\assign\\2015\\assign 05"

> desid = "desiderata.txt"
> invictus = "invictus.txt"
> no_enemies = "no enemies.txt"

> c_fail = TextFileCollection(desid, desid, invictus, invictus, no_enemies)      # fails: repeated file names
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 19, in __init__
AssertionError: ?? TextFileCollection.init: ?? duplicate filenames in filespec: {'invictus.txt', 'desiderata.txt'}

> c0 = TextFileCollection(desid, invictus)        # add desiderata and invictus to collection, opening at position 0

> c0.readline(desid)                    # returns desiderata, line 1
'Go placidly amid the noise and the haste, and remember what peace there may be in silence. \n'

> c1 = TextFileCollection((desid, {'position': 40}), no_enemies)  # adds no_enemies, opening at position 0; bumps desid to character 40

> c0.readline(desid)                    # gets balance of line starting at line 40
', and remember what peace there may be in silence. \n'

> c1.readline(desid)                    # gets next line
'As far as possible, without surrender, be on good terms with all persons.\n'

> c1.readline(no_enemies)                 # gets first line from no_enemies
'You have no enemies, you say?\n'

> c0.readline(no_enemies)                 # fails: c0 has no handle on no_enemies
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 27, in readline
  File "<stdin>", line 57, in assert_open
AssertionError: ?? TextFileCollection.readline: ?? handle on no enemies.txt not held by 8505456

> c0.close(no_enemies)                # fails: c0 has no handle on no_enemies
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 39, in close
  File "<stdin>", line 57, in assert_open
AssertionError: ?? TextFileCollection.close: ?? handle on no enemies.txt not held by 8505456

> c1.readline(no_enemies)                 # gets second line from no_enemies
'Alas, my friend, the boast is poor,\n'

> c1.close(no_enemies)                # succeeds, deleting no_enemies from the collection

> c1.readline(no_enemies)                 # fails: no_enemies removed from collection
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 27, in readline
  File "<stdin>", line 54, in assert_open
AssertionError: ?? TextFileCollection.readline: ?? handle on no enemies.txt not in collection

> c0.close(desid)                 # succeeds, releasing c0's handle on desiderata

> c0.close(desid)                 # fails: handle already closed
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 39, in close
  File "<stdin>", line 57, in assert_open
AssertionError: ?? TextFileCollection.close: ?? handle on desiderata.txt not held by 8505456

> c0.readline(desid)                    # fails: handle closed
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 27, in readline
  File "<stdin>", line 57, in assert_open
AssertionError: ?? TextFileCollection.readline: ?? handle on desiderata.txt not held by 8505456

> c1.readline(desid)                    # succeeds
'Speak your truth quietly and clearly; and listen to others, even to the dull and the ignorant; they too have their story.\n'

>
```

Figure 1: sample interaction with classes

The assignment:

1. 12 points. Implement the classes as described above. Point values:
   - The text file class: 5 points
   - The text file collection class: 5 points
   - Exception message quality: 2 points

2. 1 point. Due to time constraints, I'm not going to ask you to reimplement this code in either Java, C#, or C++. I will, however, ask you to discuss what would be required to do the port.

Deliverables:

Problem 1: your code plus screen shots of sample test runs.

Problem 2: your description.