COS20007: Object Oriented Programming

Pass Task 11.1: Clock in Another Language

Show Wai Yan/105293041

counter.py

```
class Counter:
    def __init__(self, name):
        self._name = name
        self._count = 0

    def increment(self):
        self._count += 1

    def reset(self):
        self._count = 0

    def reset_by_default(self):
        # Original large value
        large_value = 2147483647041

        # Simulate int32 overflow behavior
        int32_max = 2147483647
        int32_min = -2147483648
        int32_range = int32_max - int32_min + 1

        # This simulates the unchecked overflow behavior from C#
        self._count = ((large_value - int32_min) % int32_range) + int32_min

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    @property
    def ticks(self):
        return self._count
```

clock.py

```
from counter import Counter


class Clock:
    def __init__(self):
        # Fields
        self._hour = Counter("Hour")
        self._minute = Counter("Minute")
        self._second = Counter("Second")

    # Methods
    def tick(self):
        self._increment_second()
```

```python
    def reset(self):
        self._second.reset()
        self._minute.reset()
        self._hour.reset()

    def _increment_second(self):
        self._second.increment()
        if self._second.ticks == 60:
            self._second.reset()
            self._increment_minute()

    def _increment_minute(self):
        self._minute.increment()
        if self._minute.ticks == 60:
            self._minute.reset()
            self._increment_hour()

    def _increment_hour(self):
        self._hour.increment()
        if self._hour.ticks == 13:
            self._hour.reset()
            self._hour.increment()

    def get_time(self):
        return f"{self._hour_str}:{self._minute_str}:{self._second_str}"

    # Properties
    @property
    def _hour_str(self):
        if self._hour.ticks == 0:
            self._hour.increment()
        return f"{self._hour.ticks:02d}"

    @property
    def _minute_str(self):
        return f"{self._minute.ticks:02d}"

    @property
    def _second_str(self):
        return f"{self._second.ticks:02d}"
```

main.py

```python
from clock import Clock
import tracemalloc
import time


def main():
    """Main program function - equivalent to C# Main method"""
    seconds_in_a_day = 86400
    my_clock = Clock()

    for i in range(seconds_in_a_day):
        my_clock.tick()
        print(my_clock.get_time())


tracemalloc.start()
start = time.time()
```

```
main()

usage = tracemalloc.get_traced_memory()
print("Current Usage: ", usage[0])
print("Peak Usage: ", usage[1])

end = time.time()
print("Execution Time: ", end - start, "second")
tracemalloc.stop()
```

Screenshot of the program running in Python

```
12:59:29
12:59:30
12:59:31
12:59:32
12:59:33
12:59:34
12:59:35
12:59:36
12:59:37
12:59:38
12:59:39
12:59:40
12:59:41
12:59:42
12:59:43
12:59:44
12:59:45
12:59:46
12:59:47
12:59:48
12:59:49
12:59:50
12:59:51
12:59:52
12:59:53
12:59:54
12:59:55
12:59:56
12:59:57
12:59:58
12:59:59
01:00:00
Current Usage:  0
Peak Usage:  1793
Execution Time:  0.5119118690490723 second
~/Dev/COS20007/PassTask/11.1P/Clock  main ?2 ❯
```

Memory usage and execution time Comparison

C#

```
Execution Time: 69 ms
Current process: System.Diagnostics.Process (Clock)
Physical memory usage: 44122112 bytes
Peak physical memory usage 0 bytes
```

Python

```
Current Usage:  0
Peak Usage:  1793
Execution Time:  0.5152420997619629 second
```

Why Python’s memory usage is so low compare to C#?

The memory usage difference between Python and C# implementations stems from comparing incompatible metrics rather than actual performance. Python's tracemalloc measurement of 1,793 bytes only tracks Python object allocations, excluding interpreter overhead and system libraries, while C#'s 45MB measurement encompasses the entire process including the .NET runtime infrastructure. The .NET Common Language Runtime requires 20-30MB baseline memory for the Just-In-Time compiler, garbage collector, and runtime services before any application code executes.

Why C# is so fast compare to python?

The execution time difference, 69ms for C# versus 0.51 seconds for Python, occurs because C# uses Just-In-Time compilation to convert code into optimized native machine instructions that run directly on the processor, while Python interprets bytecode through a virtual machine layer that adds significant overhead. C#'s static typing enables compile-time optimizations and eliminates runtime type checks, whereas Python's dynamic typing requires constant runtime type resolution and method lookups.