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()

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

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

A screenshot of a computer

AI-generated content may be incorrect.

# Memory usage and execution time Comparison

## C#

A screen shot of a computer

AI-generated content may be incorrect.

## Python

### A number on a dark background AI-generated content may be incorrect. 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.