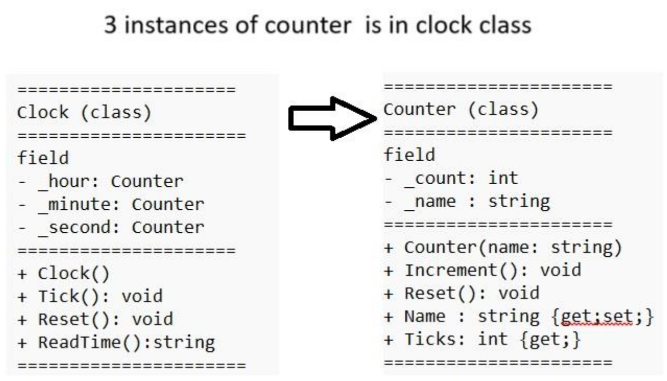
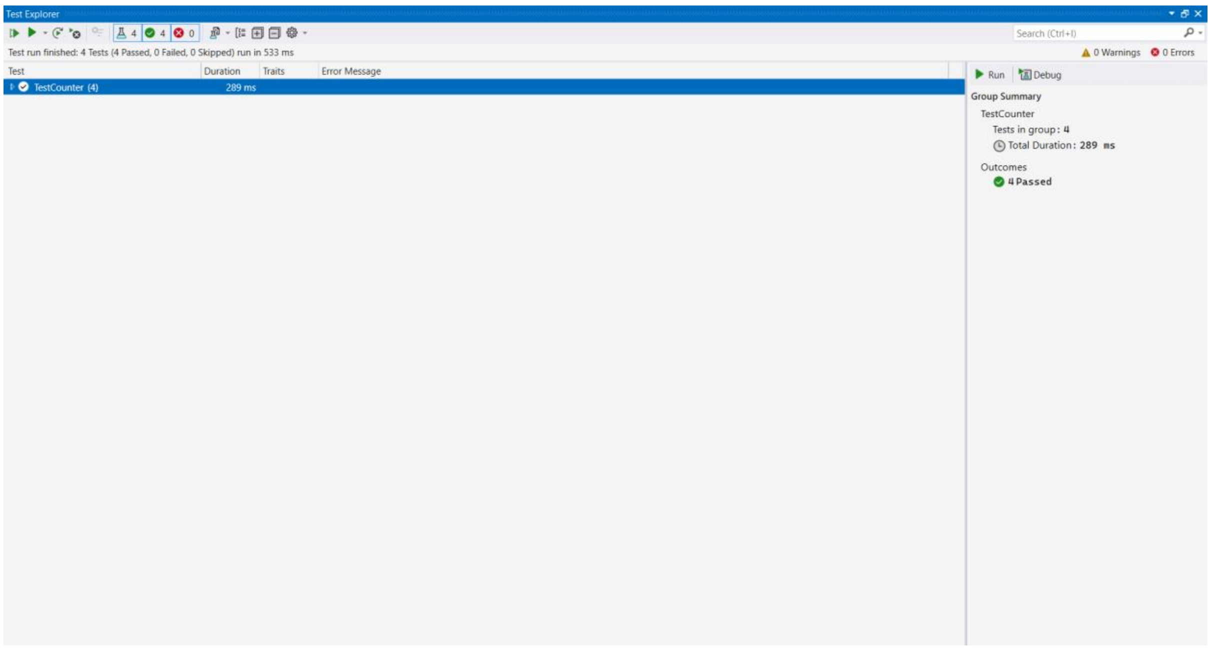
UML Diagram


3 instances of counter is in clock class

```
=====================
Clock (class)
======================
field
- _hour: Counter
- _minute: Counter
- _second: Counter
=====================
+ Clock()
+ Tick(): void
+ Reset(): void
+ ReadTime():string
======================
```

```
======================
Counter (class)
======================
field
- _count: int
- _name : string
======================
+ Counter(name: string)
+ Increment(): void
+ Reset(): void
+ Name : string {get;set;}
+ Ticks: int {get;}
======================
```

Unit testing for counter class

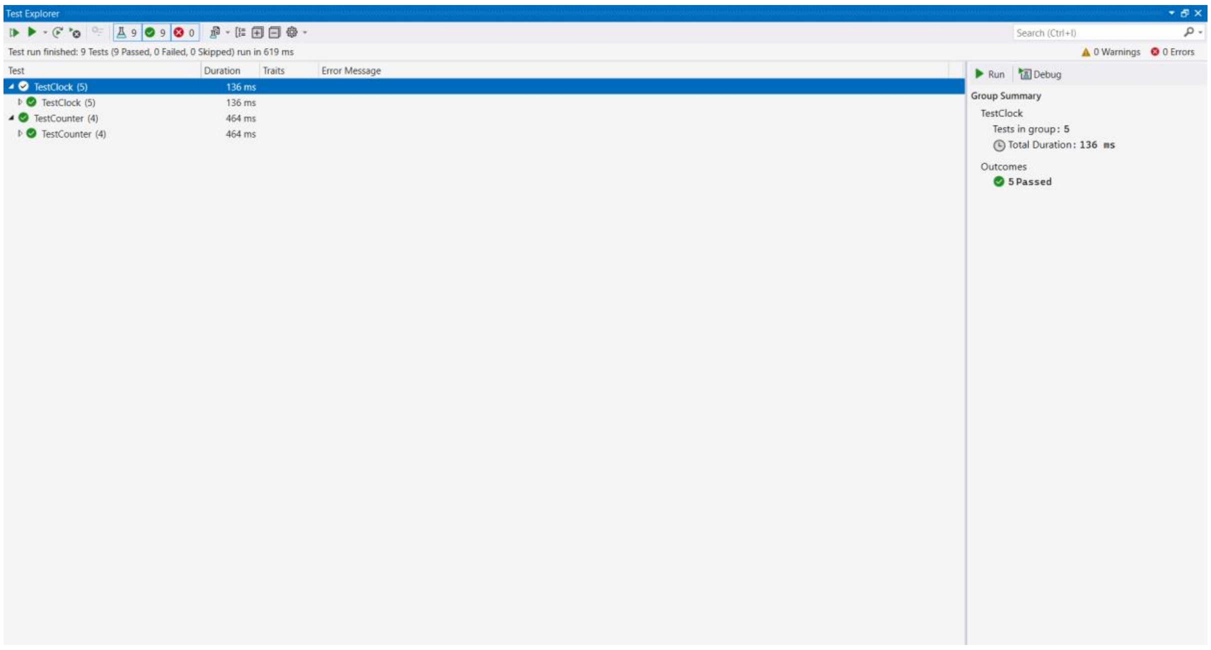


```
Test run finished: 4 Tests (4 Passed, 0 Failed, 0 Skipped) run in 533 ms        0 Warnings   0 Errors

Test                    Duration   Traits   Error Message
TestCounter (4)         289 ms

Group Summary
  TestCounter
    Tests in group: 4
    Total Duration: 289 ms
  Outcomes
    4 Passed
```

Unit testing for clock class



Test Explorer

Test run finished: 9 Tests (9 Passed, 0 Failed, 0 Skipped) run in 619 ms

⚠ 0 Warnings ✖ 0 Errors

Test	Duration	Traits	Error Message
TestClock (5)	136 ms		
TestClock (5)	136 ms		
TestCounter (4)	464 ms		
TestCounter (4)	464 ms		

Group Summary

TestClock

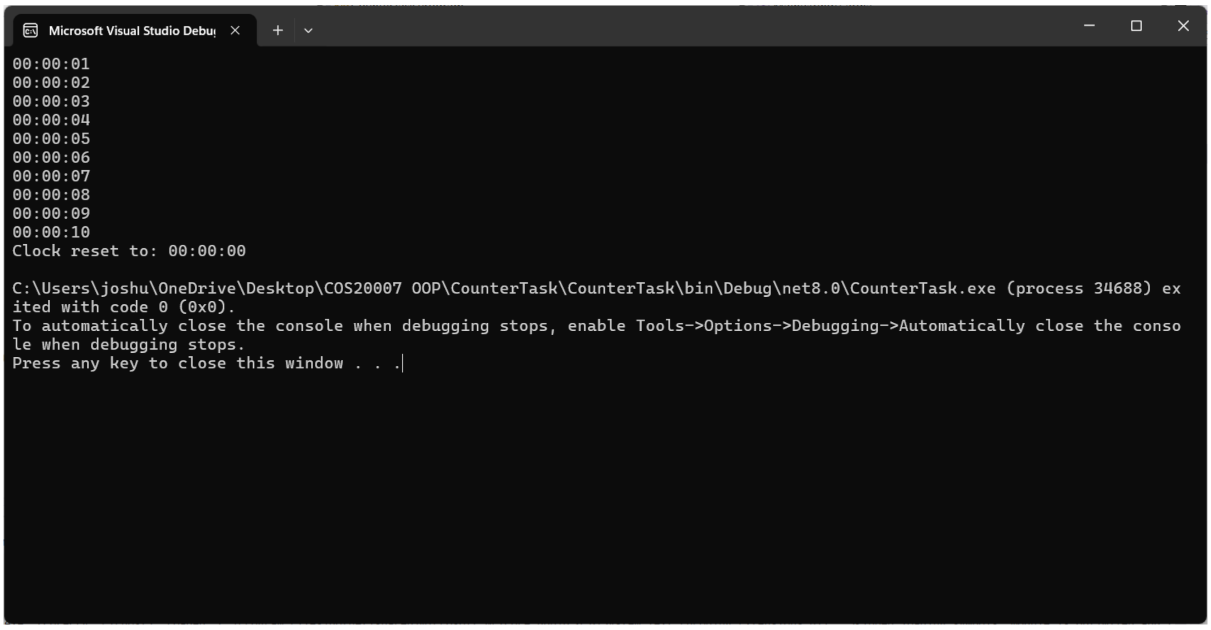
Tests in group: 5

Total Duration: 136 ms

Outcomes

5 Passed

Output program



```
00:00:01
00:00:02
00:00:03
00:00:04
00:00:05
00:00:06
00:00:07
00:00:08
00:00:09
00:00:10
Clock reset to: 00:00:00

C:\Users\joshu\OneDrive\Desktop\COS20007 OOP\CounterTask\CounterTask\bin\Debug\net8.0\CounterTask.exe (process 34688) ex
ited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Counter classusing System;

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CounterTask
{
    public class Counter
    {

        private int _count;
        private string _name;



        public Counter(string name)
        {
            _name = name;
            _count = 0;
        }
        public void Increment()
        { _count++; }

        public void Reset()
        { _count = 0; }

        public string Name
        {
            get
            {
                return _name;
            }
            set
            {
                _name = value;

            }


        }
        public int Ticks
        {
            get
            {
                return _count;

            }
        }


    }
}
```

Clock class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using CounterTask;

namespace CounterTask
```

```
{
    public class Clock
    {
        private Counter _hour;
        private Counter _minute;
        private Counter _second;

        public Clock()
        {
            _hour = new Counter("Hour");
            _minute = new Counter("Minute");
            _second = new Counter("Second");

        }
        public void Tick()
        {
            _second.Increment();  // Increment the second counter

            if (_second.Ticks == 60)  // If seconds reach 60, reset and increment minute
            {
                _second.Reset();
                _minute.Increment();
            }

            if (_minute.Ticks == 60)  // If minutes reach 60, reset and increment hour
            {
                _minute.Reset();
                _hour.Increment();
            }

            if (_hour.Ticks == 24)  // If hours reach 24, reset to 0
            {
                _hour.Reset();
            }
        }

        // Method to reset the clock to 00:00:00
        public void Reset()
        {
            _hour.Reset();
            _minute.Reset();
            _second.Reset();
        }

        // Method to read the current time as a string in the format "hh:mm:ss"
        public string ReadTime()
        {
            // Format each component as a 2-digit number with leading zeros if necessary
            return $"{_hour.Ticks:D2}:{_minute.Ticks:D2}:{_second.Ticks:D2}";
        }
    }
}
```

Unit testing counter

```
using CounterTask;
using NUnit.Framework;

namespace TestCounter
{
    [TestFixture]
    public class CounterTest
    {
```

```
// Test that initializing the Counter starts at 0

[Test]

public void TestCounterInitialization()

{

   Counter c = new Counter("TestCounter");

   Assert.That(c.Ticks, Is.EqualTo(0));

}


// Test that incrementing the Counter adds one to the count

[Test]

public void TestCounterIncrement()

{

   Counter c = new Counter("TestCounter");

   c.Increment();

   Assert.That(c.Ticks, Is.EqualTo(1));

}


// Test that incrementing multiple times increases the count by the correct amount

[Test]

public void TestCounterMultipleIncrements()

{

   Counter c = new Counter("TestCounter");

   for (int i = 0; i < 5; i++)

   {

      c.Increment();

   }

   Assert.That(c.Ticks, Is.EqualTo(5));

}



[Test]
```

```
        public void TestCounterReset()
        {
            Counter c = new Counter("TestCounter");
            c.Increment();
            c.Increment();
            c.Reset();
            Assert.That(c.Ticks, Is.EqualTo(0));
        }
    }
}
```

Unit testing clock

```
using System;
using CounterTask;

namespace TestClock
{
    [TestFixture]
    public class ClockTest
    {
        // Test that the Clock initializes to 00:00:00
        [Test]
        public void TestClockInitialization()
        {
            Clock clock = new Clock();
            Assert.That(clock.ReadTime(), Is.EqualTo("00:00:00"), "Clock should initialize to 00:00:00.");
        }

        // Test that ticking the Clock updates the seconds
        [Test]
        public void TestClockTickSeconds()
        {
            Clock clock = new Clock();
            clock.Tick();
            Assert.That(clock.ReadTime(), Is.EqualTo("00:00:01"), "Clock should tick to 00:00:01.");
        }

        // Test that ticking the Clock from 00:00:59 updates minutes
        [Test]
        public void TestClockTickMinutes()
        {
            Clock clock = new Clock();
            for (int i = 0; i < 60; i++)
            {
                clock.Tick();
            }
            Assert.That(clock.ReadTime(), Is.EqualTo("00:01:00"), "Clock should tick from 00:00:59 to 00:01:00.");
        }

        // Test that ticking the Clock from 23:59:59 wraps around to 00:00:00
        [Test]
        public void TestClockTickWrapAround()
        {
            Clock clock = new Clock();

            // Manually setting the time to 23:59:59 by ticking 86399 times (total seconds in a day - 1)
            for (int i = 0; i < 86399; i++)
            {
                clock.Tick();
            }
            Assert.That(clock.ReadTime(), Is.EqualTo("23:59:59"), "Clock should be at 23:59:59.");


            // One more tick should reset the clock to 00:00:00
            clock.Tick();
            Assert.That(clock.ReadTime(), Is.EqualTo("00:00:00"), "Clock should wrap around to 00:00:00
after 23:59:59.");
        }


        // Test that resetting the Clock sets the time to 00:00:00
        [Test]
        public void TestClockReset()
        {
            Clock clock = new Clock();
            clock.Tick();
            clock.Tick();
            clock.Reset();
            Assert.That(clock.ReadTime(), Is.EqualTo("00:00:00"), "Clock should reset to 00:00:00.");
        }
    }
}
```