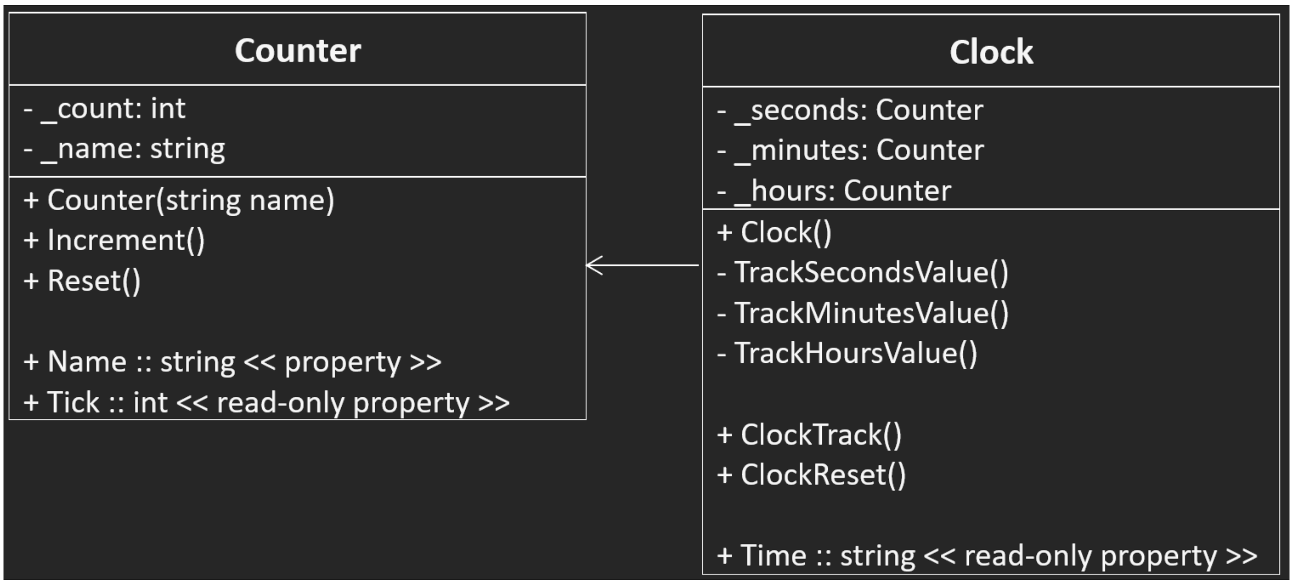
Swinburne University of Technology

COS20007 Object Oriented Programming

3.1P - Clock Class

PDF generated at 23:54 on Friday 14th April, 2023


Counter
- _count: int
- _name: string
+ Counter(string name)
+ Increment()
+ Reset()
+ Name :: string << property >>
+ Tick :: int << read-only property >>
Clock
- _seconds: Counter
- _minutes: Counter
- _hours: Counter
+ Clock()
- TrackSecondsValue()
- TrackMinutesValue()
- TrackHoursValue()
+ ClockTrack()
+ ClockReset()
+ Time :: string << read-only property >>

```csharp
using System;
using System.Threading;
using SplashKitSDK;

namespace Clock
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Clock clock = new Clock();

            int index = 0;

            while (true)
            {
                Console.Clear();
                clock.ClockTrack();

                Console.WriteLine(clock.Time);

                Thread.Sleep(10);

                // The clock will end after 3600 'second' ticks
                index++;
                if (index == 3600)
                {
                    break;
                }
            }
        }
    }
}
```

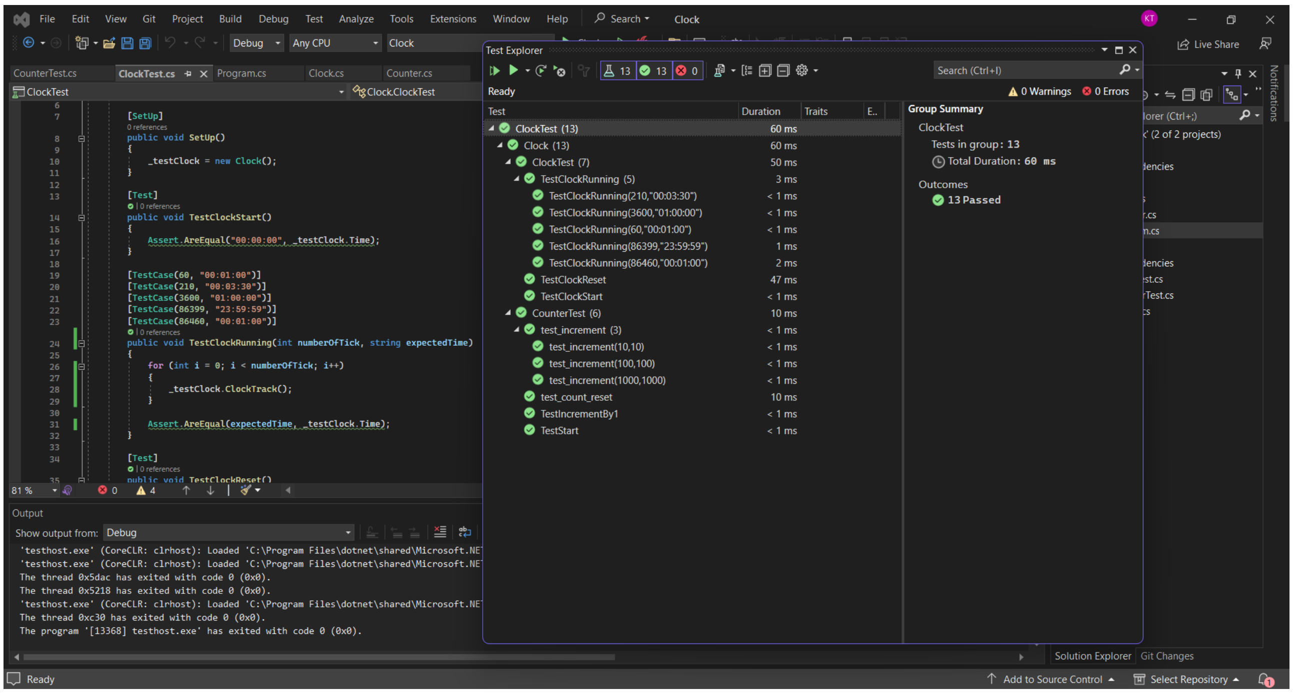
```
using System;

namespace Clock
{
    public class Clock
    {
        private Counter _seconds;
        private Counter _minutes;
        private Counter _hours;

        public Clock()
        {
            _seconds = new Counter("Seconds");
            _minutes = new Counter("Minutes");
            _hours = new Counter("Hours");
        }

        private void TrackSecondsValue()
        {
            if (_seconds.Tick < 59)
            {
                _seconds.Increment();
            }
            else
            {
                _seconds.Reset();
                _minutes.Increment();
            }
        }

        private void TrackMinutesValue()
        {
            if (_minutes.Tick > 59)
            {
                _minutes.Reset();
                _hours.Increment();
            }
        }

        private void TrackHoursValue()
        {
            if (_hours.Tick > 23)
            {
                _hours.Reset();
            }
        }

        public void ClockTrack()
        {
            TrackSecondsValue();
            TrackMinutesValue();
            TrackHoursValue();
        }
```

```csharp

        public string Time
        {
            get
            {
                return $"{_hours.Tick:D2}:{_minutes.Tick:D2}:{_seconds.Tick:D2}";
            }
        }

        public void ClockReset()
        {
            _seconds.Reset();
            _minutes.Reset();
            _hours.Reset();
        }
    }
}
```

```
namespace Clock
{
    public class ClockTest
    {
        Clock _testClock;

        [SetUp]
        public void SetUp()
        {
            _testClock = new Clock();
        }

        [Test]
        public void TestClockStart()
        {
            Assert.AreEqual("00:00:00", _testClock.Time);
        }

        [TestCase(60, "00:01:00")]
        [TestCase(210, "00:03:30")]
        [TestCase(3600, "01:00:00")]
        [TestCase(86399, "23:59:59")]
        [TestCase(86460, "00:01:00")]
        public void TestClockRunning(int numberOfTick, string expectedTime)
        {
            for (int i = 0; i < numberOfTick; i++)
            {
                _testClock.ClockTrack();
            }

            Assert.AreEqual(expectedTime, _testClock.Time);
        }

        [Test]
        public void TestClockReset()
        {
            for (int i = 0; i < 300; i++)
            {
                _testClock.ClockTrack();
            }

            Assert.AreEqual("00:05:00", _testClock.Time);

            _testClock.ClockReset();

            Assert.AreEqual("00:00:00", _testClock.Time);
        }
    }
}
```

```
using System;

namespace Clock
{
    public class Counter
    {
        private int _count;
        private string _name;

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

        public void Increment()
        {
            _count++;
        }

        public void Reset()
        {
            _count = 0;
        }

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

        public int Tick
        {
            get
            {
                return _count;
            }
        }
    }
}
```

```
namespace Clock
{
    public class CounterTest
    {
        Counter _testCounter;

        [SetUp]
        public void Setup()
        {
            _testCounter = new Counter("Test");
        }

        [Test]
        public void TestStart()
        {
            Assert.AreEqual(0, _testCounter.Tick);
        }

        [Test]
        public void TestIncrementBy1()
        {
            int index = _testCounter.Tick;
            _testCounter.Increment();
            Assert.AreEqual(index + 1, _testCounter.Tick);
        }

        [TestCase(10, 10)]
        [TestCase(100, 100)]
        [TestCase(1000, 1000)]
        public void test_increment(int tick, int result)
        {
            for (int i = 0; i < tick; i++)
            {
                _testCounter.Increment();
            }
            Assert.AreEqual(result, _testCounter.Tick);
        }

        [Test]
        public void test_count_reset()
        {
            _testCounter.Reset();
            Assert.AreEqual(0, _testCounter.Tick);
        }
    }
}
```



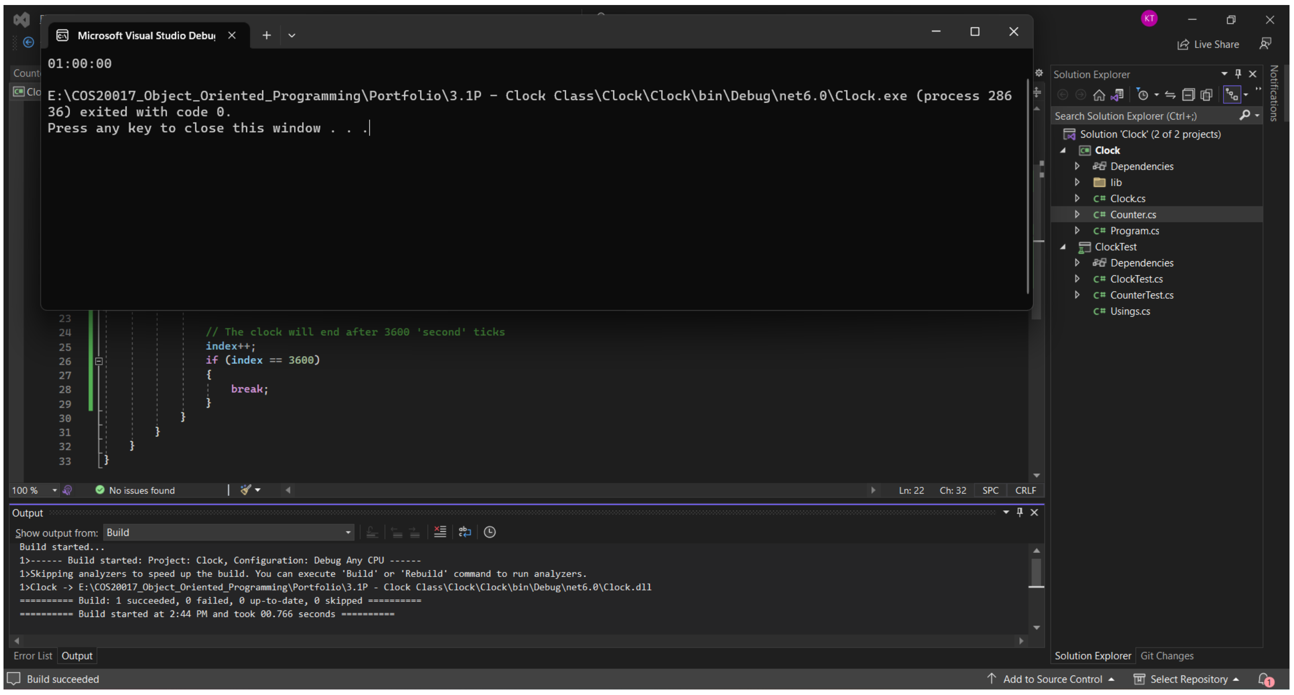
```
        [SetUp]
        0 references
        public void SetUp()
        {
            _testClock = new Clock();
        }

        [Test]
        | 0 references
        public void TestClockStart()
        {
            Assert.AreEqual("00:00:00", _testClock.Time);
        }

        [TestCase(60, "00:01:00")]
        [TestCase(210, "00:03:30")]
        [TestCase(3600, "01:00:00")]
        [TestCase(86399, "23:59:59")]
        [TestCase(86460, "00:01:00")]
        | 0 references
        public void TestClockRunning(int numberOfTick, string expectedTime)
        {
            for (int i = 0; i < numberOfTick; i++)
            {
                _testClock.ClockTrack();
            }

            Assert.AreEqual(expectedTime, _testClock.Time);
        }

        [Test]
        | 0 references
        public void TestClockReset()
```

Test	Duration	Traits	E..
ClockTest (13)	60 ms		
Clock (13)	60 ms		
ClockTest (7)	50 ms		
TestClockRunning (5)	3 ms		
TestClockRunning(210,"00:03:30")	< 1 ms		
TestClockRunning(3600,"01:00:00")	< 1 ms		
TestClockRunning(60,"00:01:00")	< 1 ms		
TestClockRunning(86399,"23:59:59")	1 ms		
TestClockRunning(86460,"00:01:00")	2 ms		
TestClockReset	47 ms		
TestClockStart	< 1 ms		
CounterTest (6)	10 ms		
test_increment (3)	< 1 ms		
test_increment(10,10)	< 1 ms		
test_increment(100,100)	< 1 ms		
test_increment(1000,1000)	< 1 ms		
test_count_reset	10 ms		
TestIncrementBy1	< 1 ms		
TestStart	< 1 ms		

Group Summary
ClockTest
Tests in group: 13
Total Duration: 60 ms
Outcomes
13 Passed

0 Warnings 0 Errors

```
'testhost.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NE
'testhost.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NE
The thread 0x5dac has exited with code 0 (0x0).
The thread 0x5218 has exited with code 0 (0x0).
'testhost.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NE
The thread 0xc30 has exited with code 0 (0x0).
The program '[13368] testhost.exe' has exited with code 0 (0x0).
```



```
01:00:00

E:\COS20017_Object_Oriented_Programming\Portfolio\3.1P - Clock Class\Clock\Clock\bin\Debug\net6.0\Clock.exe (process 286
36) exited with code 0.
Press any key to close this window . . .
```

```
                    // The clock will end after 3600 'second' ticks
                    index++;
                    if (index == 3600)
                    {
                        break;
                    }
                }
            }
        }
    }
```

```
Build started...
1>------ Build started: Project: Clock, Configuration: Debug Any CPU ------
1>Skipping analyzers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analyzers.
1>Clock -> E:\COS20017_Object_Oriented_Programming\Portfolio\3.1P - Clock Class\Clock\Clock\bin\Debug\net6.0\Clock.dll
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build started at 2:44 PM and took 00.766 seconds ==========
```