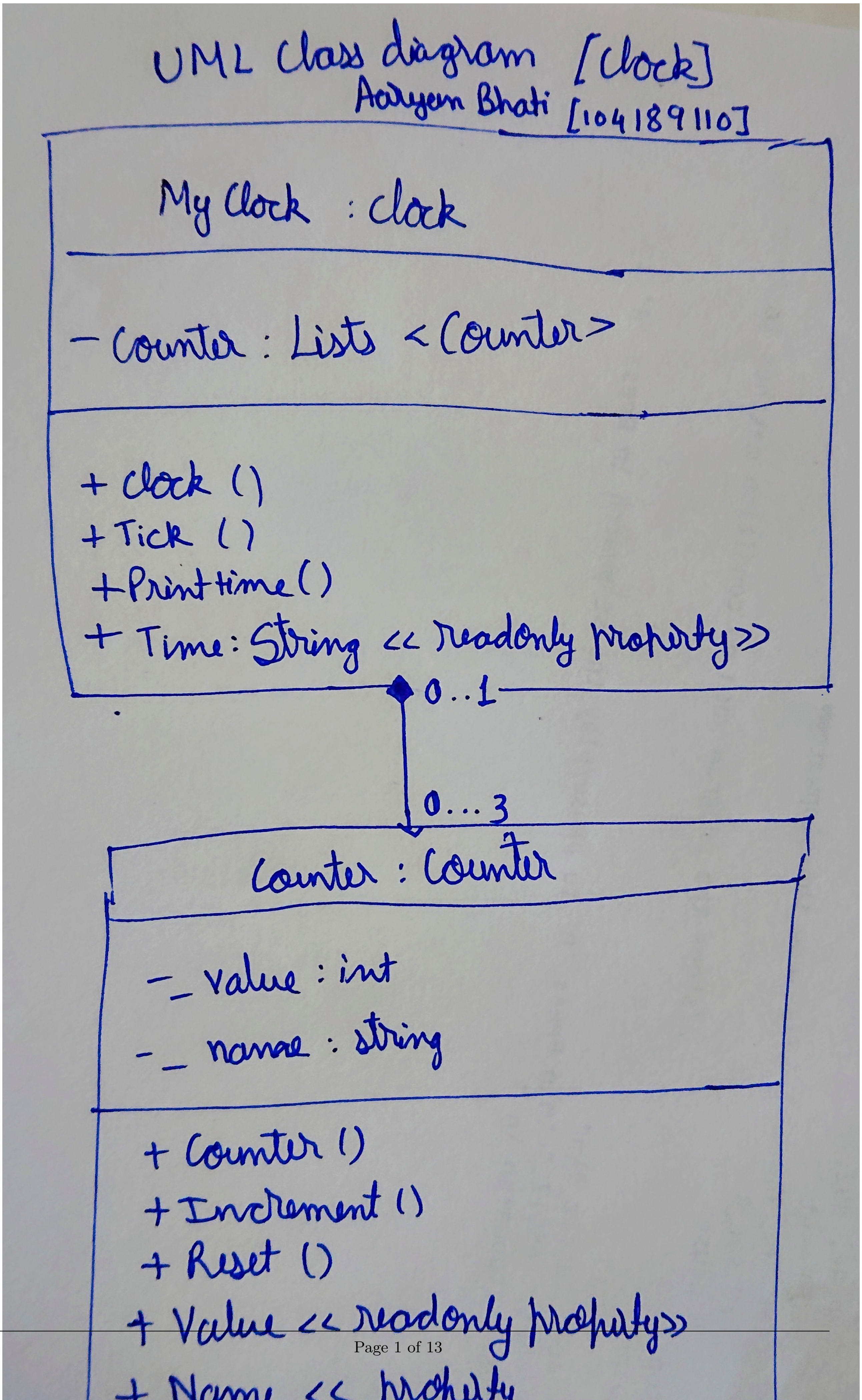
Swinburne University of Technology

COS20007 Object Oriented Programming

Clock Class

PDF generated at 10:35 on Tuesday 29th August, 2023


UML Class diagram [Clock]
Aaryan Bhati [104189110]
My Clock : Clock
- Counter : Lists <Counter>
+ Clock ()
+ Tick ()
+ Printtime ()
+ Time : String << readonly property>>
0..1
0...3
Counter : Counter
-_value : int
-_name : string
+ Counter ()
+ Increment ()
+ Reset ()
+ Value << readonly property>>
+ Name << property

```csharp


namespace ClockClass
{
    class ClockProgram
    {
        static void Main()
        {

// create a clock object

            Clock clock = new Clock();
            int i;

// increment the clock 86400 times (once per second for 24 hours)

            for (i = 0; i < 86400; i++)
            {

// increment the clock and display the time

                clock.IncrementClock();

// display the time

                Console.WriteLine(clock.ReadClock());
            }
        }
    }
}
```

```
using System;


namespace  ClockClass
{
    public class Clock
    {

// Counter instances to track seconds, minutes, and hours
        Counter _seconds = new("seconds");
        Counter _minutes = new("minutes");
        Counter _hours = new("hours");


// Method to increment the clock by 1 second
        public void IncrementClock()
        {

// increment the seconds counter and checks if it is greater than 59
            _seconds.IncrementCounter();
            if (_seconds.Ticks > 59)
            {

// if the seconds counter is greater than 59, reset it to 0 and increment the minutes
↪   counter

                _seconds.ResetCounter();
                _minutes.IncrementCounter();


// check if the minutes counter is greater than 59

                if (_minutes.Ticks > 59)
                {

// if the minutes counter is greater than 59, reset it to 0 and increment the hours
↪   counter
                    _minutes.ResetCounter();
                    _hours.IncrementCounter();

// check if the hours counter is greater than 23

                    if (_hours.Ticks > 23)
                    {
// if the hours counter is greater than 23, reset it to 0

                        _hours.ResetCounter();
                    }
                }
            }
        }


// Method to reset the clock to 00:00:00

        public void ResetClock()
        {
            _seconds.ResetCounter();
            _minutes.ResetCounter();
            _hours.ResetCounter();
        }

// Method to read the clock

        public string ReadClock()
        {
            return _hours.Ticks.ToString("00") + ":" + _minutes.Ticks.ToString("00")
↪   + ":" + _seconds.Ticks.ToString("00");
        }
    }
}
```

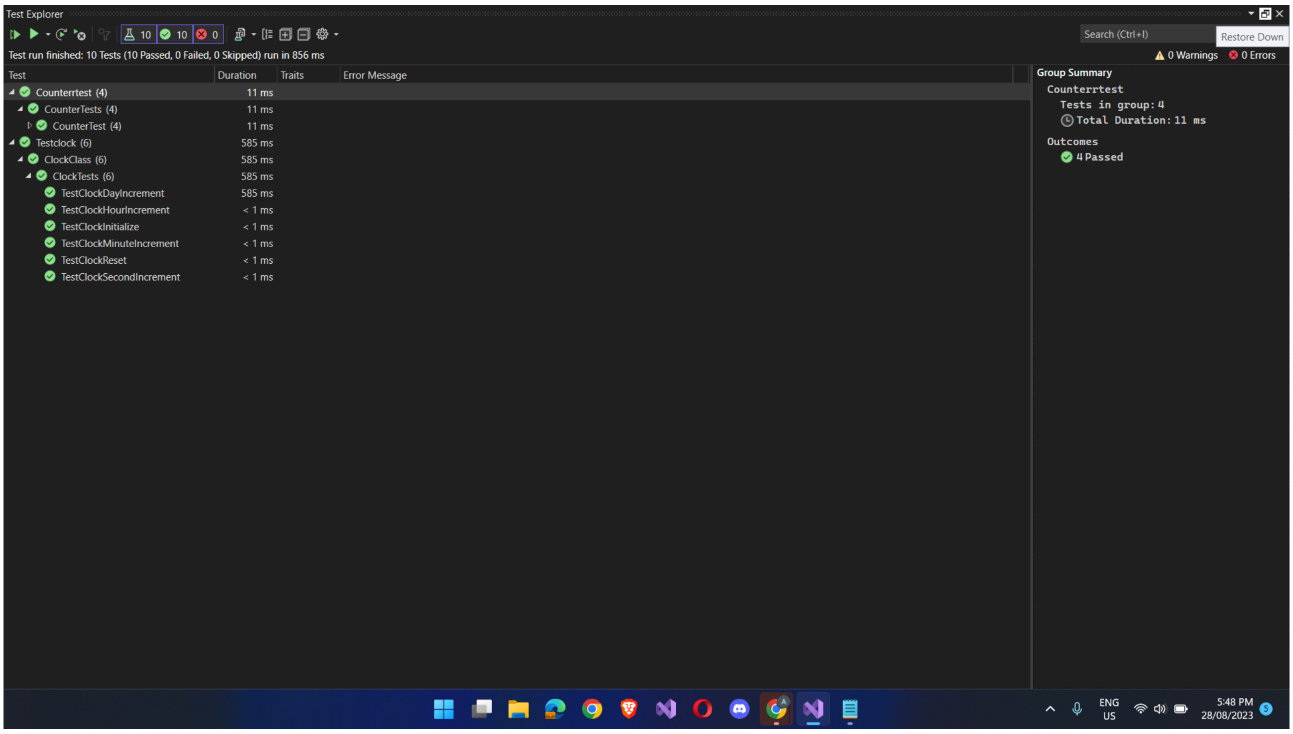
```
using NUnit;
using NUnit.Framework;
using ClockClass;

namespace ClockClass
{

// declare a test fixture

    [TestFixture()]
    internal class ClockTests
    {

// declare a variable to hold the clock instance

        Clock testClock;

// This method is called before each test to set up the test environment.

        [SetUp()]
        public void Setup()
        {

// initaliaze a new clock instance

            testClock = new Clock();
        }


// test case to check if the clock is initialized to 00:00:00

        [Test()]
        public void TestClockInitialize()
        {

// assert that the initial time is 00:00:00

            Assert.That(testClock.ReadClock(), Is.EqualTo("00:00:00"));
        }



// test case to check if the clock is incrementing seconds correctly

        [Test()]
        public void TestClockSecondIncrement()
        {

// increment the clock by 1 second

            testClock.IncrementClock();

// assert that the time is 00:00:01
```

```

            Assert.That(testClock.ReadClock(), Is.EqualTo("00:00:01"));
        }


// test case to check if the clock is incrementing minutes correctly

        [Test()]
        public void TestClockMinuteIncrement()
        {
            for (int i = 0; i < 60; i++)
            {
                testClock.IncrementClock();
            }
            Assert.That(testClock.ReadClock(), Is.EqualTo("00:01:00"));
        }

// test case to check if the clock is incrementing hours correctly

        [Test()]
        public void TestClockHourIncrement()
        {

// increment the clock by 3600 seconds or (1 hour)

            for (int i = 0; i < 3600; i++)
            {
                testClock.IncrementClock();
            }

// assert that the time is 01:00:00

            Assert.That(testClock.ReadClock(), Is.EqualTo("01:00:00"));
        }


// test case to check if the clock is incrementing days correctly
        [Test()]
        public void TestClockDayIncrement()
        {

// increment the clock by 86400 seconds or (1 day)

            for (int i = 0; i < 86400; i++)
            {
                testClock.IncrementClock();
            }

// assert that the time is 00:00:00

            Assert.That(testClock.ReadClock(), Is.EqualTo("00:00:00"));
        }

```

```

// test case to check if the clock is resetting correctly
        [Test()]
        public void TestClockReset()
        {
            testClock.IncrementClock();
            testClock.ResetClock();

// assert that the time is 00:00:00 after resetting the clock

            Assert.That(testClock.ReadClock(), Is.EqualTo("00:00:00"));
        }
    }
}
```

```


namespace ClockClass
{
    public class Counter
    {

        // private fields to store the name and count value of the counter

        private string _name;
        private int _count;

        // constructor to create a counter object with a name and a count value of 0
        public Counter(string name)
        {
            _name = name;
            _count = 0;
        }

        // property to get the current count vaalue of the counter

        public int Ticks
        {
            get
            {
                return _count;
            }
        }

        // property to get and set the name of the counter
        public string NameCounter
        {
            get
            {
                return _name;
            }
            set
            {
                _name = value;
            }
        }

        // methos to inncrement the counter value by 1

        public void IncrementCounter()
        {
            _count += 1;
        }

        // method to reset the counter value to 0

        public void ResetCounter()
        {
                _count = 0;
            }
        }
}
```

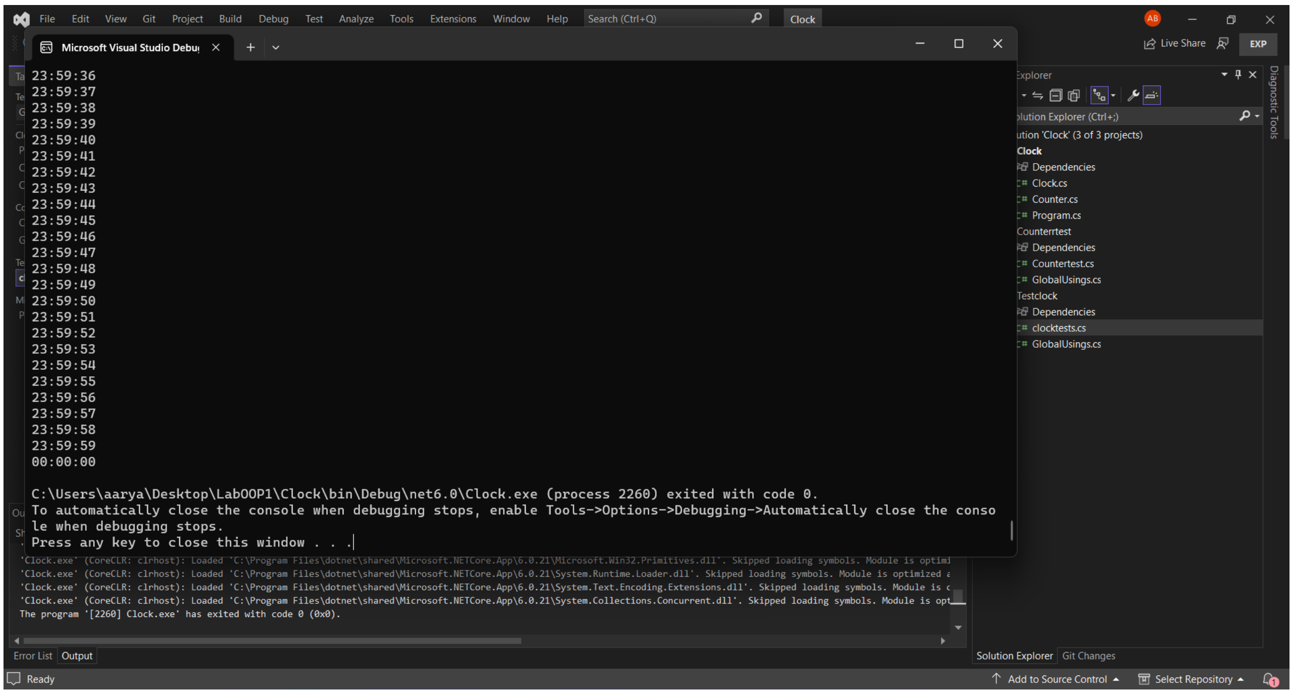
```
using ClockClass;
using NUnit.Framework;

namespace CounterTests
{

// declare a test fixture

    [TestFixture]
    public class CounterTest
    {

// declare a counter instance

        private Counter myCounter;


// This method is called before each test to set up the test environment.
        [SetUp]
        public void Setup()
        {
            myCounter = new Counter("Counter");
        }


// test case to check if the counter is initialized to 0

        [Test]

        public void CounterStart()
        {

// assert that the counter is initialized to 0

            Assert.That(myCounter.Ticks, Is.EqualTo(0));
        }



// test case to check if the counter is incremented by 1

        [Test]

        public void IncrementTest()
        {
// call the increment method on the counter instance to increse the counter/ticks by
 ↪  1

            myCounter.IncrementCounter();

// assert that the counter is incremented by 1

            Assert.That(myCounter.Ticks, Is.EqualTo(1));
        }


// test case to check if the counter is reset to 0

        [Test]

        public void ResetTest()
        {

// call the increment method on the counter instance to increse the counter/ticks by
↪   1

            myCounter.IncrementCounter();

// call the reset method on the counter instance to reset the counter/ticks to 0

            myCounter.ResetCounter();

// assert that the counter is now equal to 0

            Assert.That(myCounter.Ticks, Is.EqualTo(0));
        }


// Test case to check if the counter is incremented multiple times

        [Test]

        public void MultipleIncrement()
        {

// set the number of incremets to be performed

            int inc = 6;

// increment the ticks multiple times

            for (int i = 0; i < inc; i++)
            {
                myCounter.IncrementCounter();
            }

// assert that the counter is incremented by the number of times specified

            Assert.That(myCounter.Ticks, Is.EqualTo(6));
        }
    }
}
```



```
Test run finished: 10 Tests (10 Passed, 0 Failed, 0 Skipped) run in 856 ms
                                                    0 Warnings   0 Errors
Test                              Duration  Traits  Error Message
◢ ✓ Counterrtest (4)                 11 ms
  ◢ ✓ CounterTests (4)               11 ms
    ▷ ✓ CounterTest (4)              11 ms
◢ ✓ Testclock (6)                   585 ms
  ◢ ✓ ClockClass (6)                585 ms
    ◢ ✓ ClockTests (6)              585 ms
        ✓ TestClockDayIncrement     585 ms
        ✓ TestClockHourIncrement    < 1 ms
        ✓ TestClockInitialize       < 1 ms
        ✓ TestClockMinuteIncrement  < 1 ms
        ✓ TestClockReset            < 1 ms
        ✓ TestClockSecondIncrement  < 1 ms
```

```
Group Summary
  Counterrtest
    Tests in group: 4
    Total Duration: 11 ms

  Outcomes
    ✓ 4 Passed
```

```
ENG
US
5:48 PM
28/08/2023
```



```
23:59:36
23:59:37
23:59:38
23:59:39
23:59:40
23:59:41
23:59:42
23:59:43
23:59:44
23:59:45
23:59:46
23:59:47
23:59:48
23:59:49
23:59:50
23:59:51
23:59:52
23:59:53
23:59:54
23:59:55
23:59:56
23:59:57
23:59:58
23:59:59
00:00:00

C:\Users\aarya\Desktop\LabOOP1\Clock\bin\Debug\net6.0\Clock.exe (process 2260) exited with code 0.
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 . . .
```

```
'Clock.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\6.0.21\Microsoft.Win32.Primitives.dll'. Skipped loading symbols. Module is optimi
'Clock.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\6.0.21\System.Runtime.Loader.dll'. Skipped loading symbols. Module is optimized a
'Clock.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\6.0.21\System.Text.Encoding.Extensions.dll'. Skipped loading symbols. Module is c
'Clock.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\6.0.21\System.Collections.Concurrent.dll'. Skipped loading symbols. Module is opt
The program '[2260] Clock.exe' has exited with code 0 (0x0).
```