Day 7

date 13-08-25

VIJAY M

1. Create a Dependency Injection sample for painter class

Inject tools (or paintbrush) to painter object via dependency injection

Program.cs

```
using System;

namespace PainterApp
{
    public interface IPaintTool
    {
        string Paint();
    }

    public class PaintBrush : IPaintTool
    {
        public string Paint()
        {
            return "Painting with a brush";
        }
    }

    public class SprayGun : IPaintTool
    {
        public string Paint()
        {
            return "Spraying with a spray gun";
```

```
    }
}

public class Painter
{
    private readonly IPaintTool _paintTool;

    public Painter(IPaintTool paintTool)
    {
        _paintTool = paintTool ?? throw new ArgumentNullException(nameof(paintTool));
    }

    public string PerformPainting()
    {
        return _paintTool.Paint();
    }
}

class Program
{
    static void Main(string[] args)
    {
        IPaintTool brush = new PaintBrush();
        Painter painterWithBrush = new Painter(brush);
        Console.WriteLine(painterWithBrush.PerformPainting());

        IPaintTool sprayGun = new SprayGun();
        Painter painterWithSprayGun = new Painter(sprayGun);
        Console.WriteLine(painterWithSprayGun.PerformPainting());
    }
}
```

}

output

```
Painting with a brush
Spraying with a spray gun

C:\Users\vijay.m\source\repos\day 7\day 7\b
To automatically close the console when deb
le when debugging stops.
Press any key to close this window . . .
```

2.Create at least 10 Nunit tests

PaintTests.cs

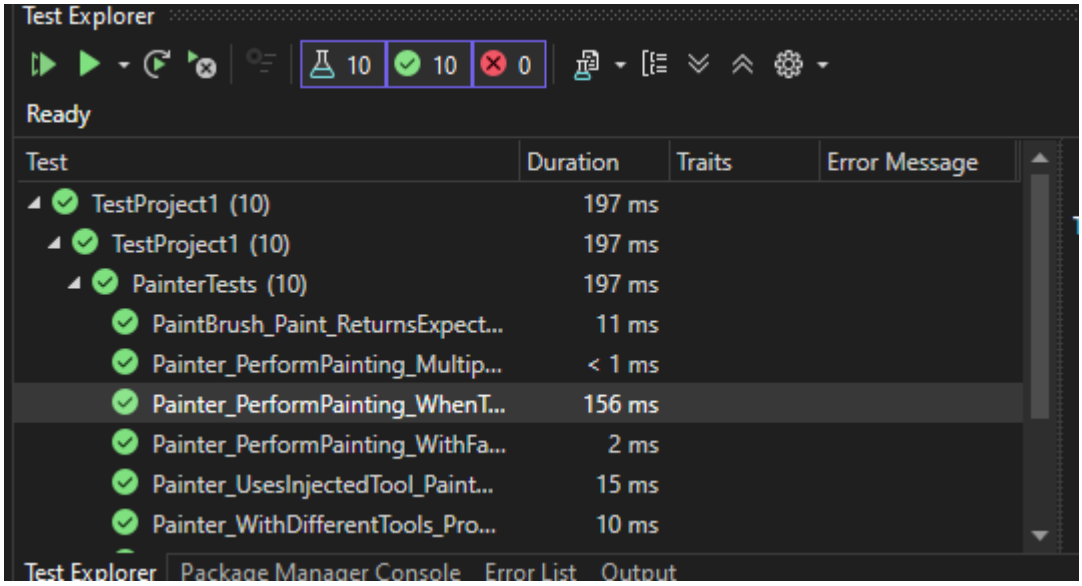
```
using Moq;

using PainterApp;


namespace TestProject1

{

    [TestFixture]

    public class PainterTests

    {

        [Test]

        public void Painter_WithPaintBrush_ReturnsBrushMessage()

        {

            var brush = new PaintBrush();

            var painter = new Painter(brush);


            var result = painter.PerformPainting();
```

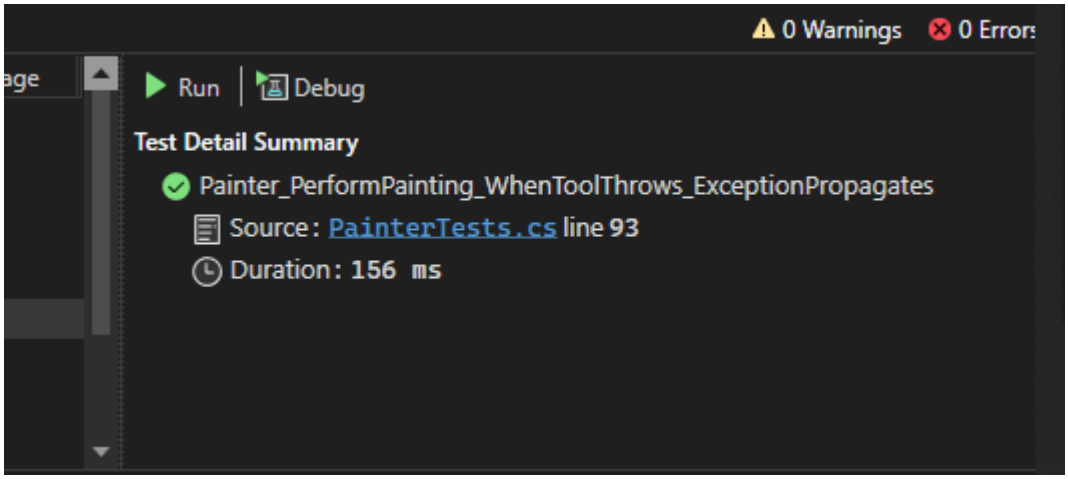
```
        Assert.AreEqual("Painting with a brush", result);
    }

    [Test]
    public void Painter_WithSprayGun_ReturnsSprayMessage()
    {
        var sprayGun = new SprayGun();
        var painter = new Painter(sprayGun);

        var result = painter.PerformPainting();

        Assert.AreEqual("Spraying with a spray gun", result);
    }

    [Test]
    public void Painter_WithNullTool_ThrowsArgumentNullException()
    {
        Assert.Throws<ArgumentNullException>(() => new Painter(null));
    }

    [Test]
    public void Painter_UsesInjectedTool_PaintMethodCalled()
    {
        var mockTool = new Mock<IPaintTool>();
        mockTool.Setup(t => t.Paint()).Returns("Mock paint");

        var painter = new Painter(mockTool.Object);
        var result = painter.PerformPainting();

        mockTool.Verify(t => t.Paint(), Times.Once);
        Assert.AreEqual("Mock paint", result);
```

```
}

[Test]
public void PaintBrush_Paint_ReturnsExpectedString()
{
    var brush = new PaintBrush();
    var result = brush.Paint();

    Assert.AreEqual("Painting with a brush", result);
}

[Test]
public void SprayGun_Paint_ReturnsExpectedString()
{
    var sprayGun = new SprayGun();
    var result = sprayGun.Paint();

    Assert.AreEqual("Spraying with a spray gun", result);
}

[Test]
public void Painter_PerformPainting_MultipleCalls_ReturnsConsistentResult()
{
    var brush = new PaintBrush();
    var painter = new Painter(brush);

    var result1 = painter.PerformPainting();
    var result2 = painter.PerformPainting();

    Assert.AreEqual(result1, result2);
}
```

```
[Test]
public void Painter_PerformPainting_WithFakeTool_ReturnsCustomMessage()
{
    var fakeTool = new Mock<IPaintTool>();
    fakeTool.Setup(t => t.Paint()).Returns("Faking painting");

    var painter = new Painter(fakeTool.Object);
    var result = painter.PerformPainting();

    Assert.AreEqual("Faking painting", result);
}

[Test]
public void Painter_PerformPainting_WhenToolThrows_ExceptionPropagates()
{
    var tool = new Mock<IPaintTool>();
    tool.Setup(t => t.Paint()).Throws(new InvalidOperationException("Tool broken"));

    var painter = new Painter(tool.Object);

    var ex = Assert.Throws<InvalidOperationException>(() => painter.PerformPainting());
    Assert.AreEqual("Tool broken", ex.Message);
}

[Test]
public void Painter_WithDifferentTools_ProducesDifferentResults()
{
    var brushPainter = new Painter(new PaintBrush());
    var sprayPainter = new Painter(new SprayGun());
```

```
            var resultBrush = brushPainter.PerformPainting();

            var resultSpray = sprayPainter.PerformPainting();


            Assert.AreNotEqual(resultBrush, resultSpray);

        }

    }

}
```





2.1 run the test using visual studio and comand line

command line command for run appliction

dotnet build

dotnet run

```
C:\Users\vijay.m\source\repos\day 7\day 7>dotnet run
Painting with a brush
Spraying with a spray gun

C:\Users\vijay.m\source\repos\day 7\day 7>
```

command line command for run test

dotnet test

```
C:\Users\vijay.m\source\repos\day 7\TestProject1>d
Restore complete (0.9s)
  day 7 succeeded (0.3s) → C:\Users\vijay.m\source
  TestProject1 succeeded (0.4s) → bin\Debug\net8.0
NUnit Adapter 4.5.0.0: Test execution started
Running all tests in C:\Users\vijay.m\source\repos
   NUnit3TestExecutor discovered 10 of 10 NUnit te
NUnit Adapter 4.5.0.0: Test execution complete
  TestProject1 test succeeded (3.0s)

Test summary: total: 10, failed: 0, succeeded: 10,
Build succeeded in 5.8s
```