## The test

### Description

The test is based on writing code to first describe and then transform a collection of nodes which are arranged in a tree. The node definitions are as follows:

```
public abstract class Node
{
    public string Name { get; }
    protected Node(string name)
    {
        Name = name;
    }
}

public class NoChildrenNode : Node
{
    public NoChildrenNode(string name) : base(name)
    {
    }
}

public class SingleChildNode : Node
{
    public Node Child { get; }
    public SingleChildNode(string name, Node child) : base(name)
    {
        Child = child;
    }
}

public class TwoChildrenNode : Node
{
    public Node FirstChild { get; }
    public Node SecondChild { get; }
    public TwoChildrenNode(string name, Node first, Node second) : base(name)
    {
        FirstChild = first;
        SecondChild = second;
    }
}

public class ManyChildrenNode : Node
{
```

```
    public IEnumerable<Node> Children { get; }
    public ManyChildrenNode(string name, params Node[] children) : base(name)
    {
        Children = children;
    }
}
```

There are three additional interfaces which you will implement as part of the test. These are:

```
public interface INodeDescriber
{
    string Describe(Node node);
}

public interface INodeTransformer
{
    Node Transform(Node node);
}

public interface INodeWriter
{
    Task WriteToFileAsync(Node node, string filePath);
}
```

**Part one - C# describer**

Write a describer implementing the `INodeDescriber` interface which will output a C# description of how to create the tree of nodes. The output should have node per line, indented with four spaces per nesting level.

```
INodeDescriber implementation = ...
var testData = new SingleChildNode("root",
    new TwoChildrenNode("child1",
        new NoChildrenNode("leaf1"),
        new SingleChildNode("child2",
            new NoChildrenNode("leaf2"))));
var result = implementation.Describe(testData);
// result is:
// new SingleChildNode("root",
//     new TwoChildrenNode("child1",
//         new NoChildrenNode("leaf1"),
//         new SingleChildNode("child2",
//             new NoChildrenNode("leaf2"))))
```

**Part two - transformer**

Write a transformer which will transform a tree of nodes into a matching tree that uses the correct node types (e.g. a `ManyChildrenNode` with no children should be transformed into a `NoChildNode`).

```
INodeTransformer implementation = ...
var testData = new ManyChildrenNode("root",
    new ManyChildrenNode("child1",
        new ManyChildrenNode("leaf1"),
        new ManyChildrenNode("child2",
            new ManyChildrenNode("leaf2"))));
var result = implementation.Transform(testData);
// result is equivalent to:
// new SingleChildNode("root",
//     new TwoChildrenNode("child1",
//         new NoChildrenNode("leaf1"),
//         new SingleChildNode("child2",
//             new NoChildrenNode("leaf2"))))
```

**Part three - integration**

Write a class that implements the `INodeWriter` interface. The class should be designed to receive any `INodeDescriber`, which is used to get a string representation of the tree of nodes. This string representation is then written to the specified file. You should provide at least one integration test using an off-the-shelf IoC / DI container.

```
INodeWriter implementation = ...
var filePath = ...
var testData = new NoChildrenNode("root");
await implementation.WriteToFileAsync(testData, filePath);
var result = File.ReadAllText(filePath);
// result now contains the output from whichever
// INodeDescriber that was injected.
```