Project 2 – Document Builder

Course	INFO-3137, Software Design Patterns 1
Professor	Daniel Maclam
Due	See FoL Dropbox
Weight	20%

Your Task

You are to create a class library that uses a builder to generate composites that represent JSON and XML documents. You are then to create a Windows Console interface capable of interacting with this library to create and display documents of both types. You have been given a demo application to show what your program should eventually do.

NOTE: See *How to Create Your Application* shown at the end of this handout

Details – Class Library

Start by implementing these three interfaces **exactly as shown**. You will lose a lot of marks if you modify these interfaces.

IComposite
- AddChild(IComposite child): void
- Print(int depth): string

IBuilder
- BuildBranch(string name): void
- BuildLeaf(string name, string content): void
- CloseBranch(): void
- GetDocument(): IComposite

IDirector
- BuildBranch(): void
- BuildLeaf(): void
- CloseBranch(): void

The implementations for these three interfaces should be as follows:

IComposite
- Needs a concrete class for both JSON and XML.
- Needs to have separate implementations for Branch and Leaf types
    - Branches can have children, but no text content
    - Leaves can have text content, but no children
- Printing a composite should return a string representing its full content
- The print method accepts an int for depth. This will add tabs to the beginning of each line that is printed.
- Branch nodes should print all of their child nodes
- Leaf nodes should only print their content
- Printing should be formatted for JSON or XML as appropriate (formatting details follow)

IBuilder

- Needs concrete classes for both JSON and XML
- On creation, should create a root Branch for the document
- When creating a Composite, the builder should maintain a reference to the last opened
- Branch
    - At Builder creation, this should be the root Branch
- When CloseBranch is called, the maintained Branch reference should be changed to the current Branch's parent
- In-class we used a Queue or List to do this
- GetDocument returns the root node

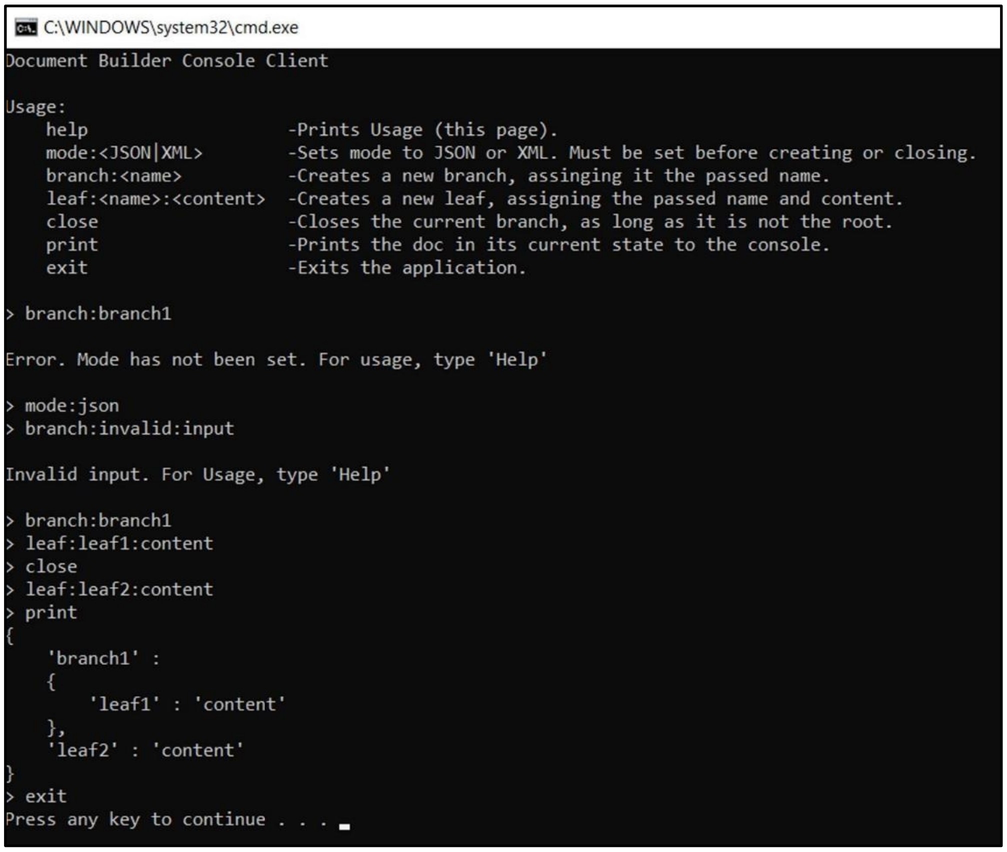
IDirector

- Director should be implemented for the Console client (details below)
- Should be only one concrete Director class
- On creation, concrete Director's constructor should be parameterized with a type: "JSON" or "XML"
- Director implementation will need to be adapted to support returning the content of the built Composite

Details – Console Client

Your application will be executed from the command line.

Your content should be similar to the output shown opposite, and document printing should display the appropriate tabbing for child nodes.



```
C:\WINDOWS\system32\cmd.exe
Document Builder Console Client

Usage:
    help                   -Prints Usage (this page).
    mode:<JSON|XML>        -Sets mode to JSON or XML. Must be set before creating or closing.
    branch:<name>          -Creates a new branch, assinging it the passed name.
    leaf:<name>:<content>  -Creates a new leaf, assigning the passed name and content.
    close                  -Closes the current branch, as long as it is not the root.
    print                  -Prints the doc in its current state to the console.
    exit                   -Exits the application.

> branch:branch1

Error. Mode has not been set. For usage, type 'Help'

> mode:json
> branch:invalid:input

Invalid input. For Usage, type 'Help'

> branch:branch1
> leaf:leaf1:content
> close
> leaf:leaf2:content
> print
{
    'branch1' :
    {
        'leaf1' : 'content'
    },
    'leaf2' : 'content'
}
> exit
Press any key to continue . . .
```

- Syntax for the console client is detailed in the Usage page (see sample output)
- Input should be **case-insensitive**
- Console client should **own** a Director
- The console client should:
    - Display Usage page when the user types "help"
    - Manipulate the document according to Usage page
    - Print the document when the user types "print"
    - Exit when the user types "exit"
- Errors should display if the user:
    - Has not set the document type
    - Enters invalid input

JSON and XML Format Details

To get a feel for the formats, play with the demo application.

Tabbing

- Tabs in the document are determined by the depth passed to the Print method
- Tabs can be either actual tabs or spaces, as long as the number of tabs/spaces is consistent.
    - Specifically, that it is number_of_spaces_or_tabs * depth

XML

- The root node for the XML document should always be <root></root>
- Branch nodes print their open tag, their close tag, and their contents each on a separate line
- Leaf nodes are printed on their own line, with their open and close tags wrapping the content at the beginning and end of each line

JSON

- The root node for the JSON document should always be nothing more than curly braces
- Branch nodes print
- Their name, followed by a colon and an opening brace, all on one line
    - Their child nodes, each on their own line (or lines, if they are themselves branches)
    - A closing brace, on its own line
- Every child node should be followed by a semi colon on the same line that the node appears
- For Leaf nodes, this means after the contents
    - For Branch nodes, this means after their closing brace (but on the same line)
    - If the node is the last child in its parent, it should not print a comma

Patterns Involved

Builder

- details of how the object is built differ somewhat between the JSON and XML implementations The Builder pattern is implemented by the IBuilder implementations
- Note that they allow for the creation of complex objects by adding parts
- Also note that the

Composite

- The documents themselves are implemented using the composite pattern
- The composites are capable of having an arbitrary number of sub-composites (leaves and branches), but all client code (other than the builder) can treat the root as though it is the entire object

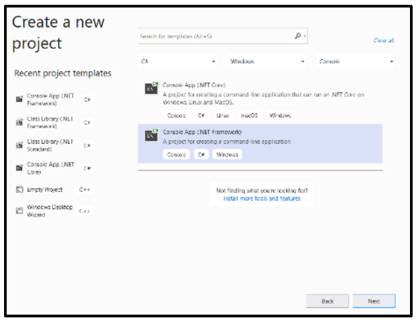
Adapter

- The IDirector's implementers need to adapt the interface for the specifics of handling input from and output to Console clients

Bridge

- The IDirector and IBuilder interfaces engage in a Bridge relationship
- This is because their concrete implementations can vary independently of each other
    - The Director owns a reference to the abstract builder, but doesn't care about what kind of builder it is
    - The Director can be adapted to work in various environments (in our case, just Console) without the Builder having to know anything about this

How to Create Your Application



Create a new project

Search for templates (Alt+S)

C# · Windows · Console

Recent project templates

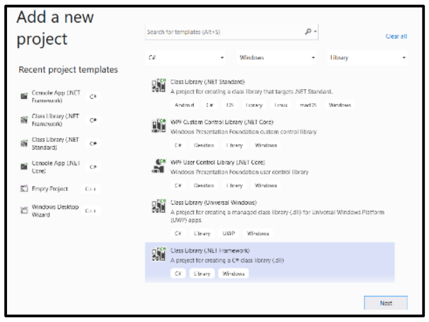
- Console App (.NET Framework) C#
- Class Library (.NET Framework) C#
- Class Library (.NET Standard) C#
- Console App (.NET Core) C#
- Empty Project C++
- Windows Desktop Wizard C++

Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
Console · C# · Linux · macOS · Windows

Console App (.NET Framework)
A project for creating a command-line application
Console · C# · Windows

Not finding what you're looking for?
Install more tools and features

1. In Visual Studio:
    - create a new project, select Console App (.NET Core) – *note that although the screenshot selects .NET Framework, which would also work fine, .NET Core lets me mark this on my mac so I slightly prefer it*
    - Name your project: Project2



Add a new project

Search for templates (Alt+S)

C# · Windows · Library

Recent project templates

- Console App (.NET Framework) C#
- Class Library (.NET Framework) C#
- Class Library (.NET Standard) C#
- Console App (.NET Core) C#
- Empty Project C++
- Windows Desktop Wizard C++

Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
Android · C# · iOS · Library · Linux · macOS · Windows

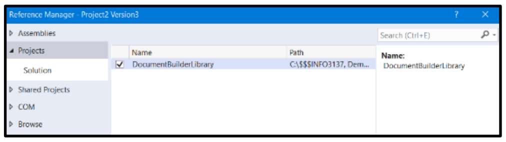
WPF Custom Control Library (.NET Core)
Windows Presentation Foundation custom control library
C# · Desktop · Library · Windows

WPF User Control Library (.NET Core)
Windows Presentation Foundation user control library
C# · Desktop · Library · Windows

Class Library (Universal Windows)
A project for creating a managed class library (.dll) for Universal Windows Platform (UWP) apps.
C# · Library · UWP · Windows

Class Library (.NET Framework)
A project for creating a C# class library (.dll)
C# · Library · Windows

2. In the Solution Explorer of your project:
    - Right-click the Solution, and add a new project
    - Select Library from the templates dropdown box
    - Select Class Library (.NET Core) – same note as above
    - Name your library: DocumentBuilderLibrary



Reference Manager - Project2 Version3

	Name	Path
✓	DocumentBuilderLibrary	C:\$$$INFO3137, Dem...

Name:
DocumentBuilderLibrary

3. To make your library available to Project2:
    - Right-click Project2
    - Add, Reference …
    - Select DocumentBuilderLibrary
    - At the top of Program.cs, type: using DocumentBuilderLibrary;

Grading Rubric

Detail	Points
Project Planning (See Project Planning Document)	
• UML Diagram showing class relationships	
• Identify pattern participants for the Builder, Composite, Bridge patterns	
• Summarize the intent for each pattern	4
JSON Builder	
• Interface fully implemented	5
• BuildBranch maintains reference to node stack	10
• CloseBranch updates references	10
XML Builder	
• Interface fully implemented	5
• BuildBranch maintains reference to node stack	10
• CloseBranch updates references	10
JSON Composite	
• Leaf implemented correctly	10
• Branch implemented correctly	10
• Prints accurately formatted JSON	5
XML Composite	
• Leaf implemented correctly	10
• Branch implemented correctly	10
• Prints accurately formatted XML	5
If you modified the interfaces	(-100)
Console Client	
• Concrete builder types only referenced when updating director	10
• Concrete composite types are never referenced	20
• Print command shows current document	2
• Help command shows usage page	2
Total	138