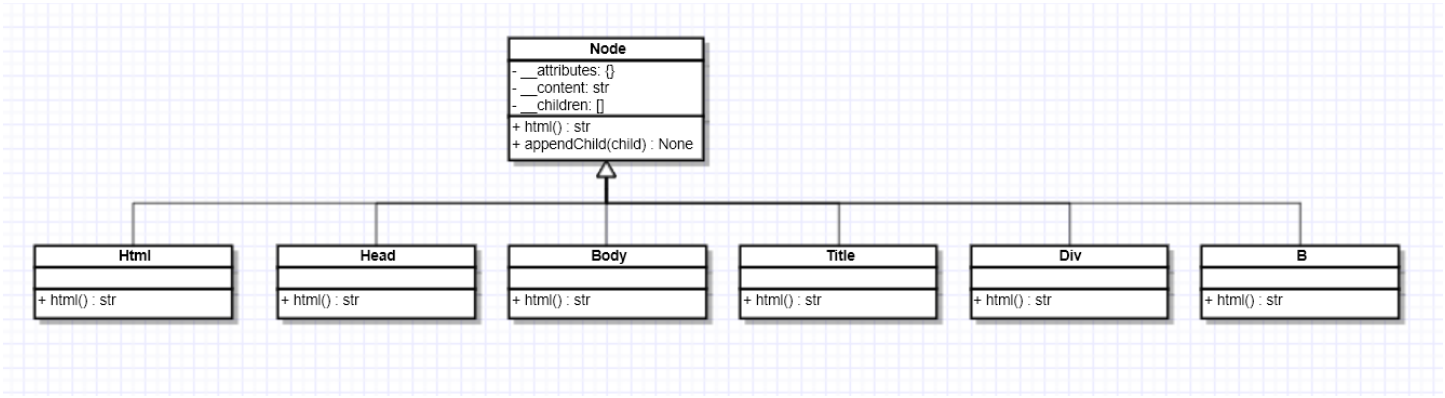
Question 1:

The class diagram below illustrates a hierarchy of classes that model simple HTML documents. The classes of this hierarchy suffice to model the following HTML page:




```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>Example</title>
</head>

<body>
    <div id="first" class="foo">This is a test A</div>
    <div id="second" class="bar">This is a test B</div>
    <div id="third" class="dump"><b>This is a simple HTML file</b>This is a test C</div>
</body>

</html>
```

Each class in the hierarchy overrides the method Node.html() so that calling text() on a node computes a string that reflects the entire subtree rooted at that node.

a) Implement the class hierarchy and use the following main() to test it.

```
def main():
    divAtts = {}
    divAtts['id'] = 'first'
    divAtts['class'] = 'foo'
    divA = Div('This is a test A', divAtts)
    divAtts = {}
    divAtts['id'] = 'second'
    divAtts['class'] = 'bar'
    divB = Div('This is a test B', divAtts)

    divAtts = {}
    divAtts['id'] = 'third'
    divAtts['class'] = 'dump'
    divC = Div('This is a test C', divAtts)
```

```
b = B('This is a simple HTML file')
divC.appendChild(b)

body = Body()
body.appendChild(divA)
body.appendChild(divB)
body.appendChild(divC)

title = Title('Example')
head = Head()
head.appendChild(title)

htmlAtts = {}
htmlAtts['lang'] = 'en'
html = Html('', htmlAtts)
html.appendChild(head)
html.appendChild(body)
print(html.html())
```

Expected output:

<!DOCTYPE html><html lang="en"><head><title>Example</title></head><body> This is a test A This is a test B This is a simple HTML fileThis is a test C </body></html>

b) Now, create a new version of the previous app in which all nodes are created using an instance of the factory design pattern. Let's call AbstractNodeFactory and StandardNodeFactory for the classes involved in the pattern. As such, you do not create concrete objects directly. Instead, you use the factory to create these objects.

Example,

div = Div('This is a test')
replaced by
div = factory.makeNode('div', 'This is a test')

More examples:

```
factory = StandardNodeFactory()
divAtts = {}
divAtts['id'] = 'second'
divAtts['class'] = 'bar'
divC = factory.makeNode('div', 'This is a test B', divAtts)
b = factory.makeNode('b')
b.appendChild(divC)
```

c) Now define a DebugNodeFactory that class defines another concrete factory which prints the textual representation of each node during its creation.

Expected output:

Div node is created.
Div node is created.
Div node is created.
B node is created.

Body node is created.
Title node is created.
Head ~~Div~~ node is created.
Html node is created.
<!DOCTYPE html><html lang="en"><head><title>Example</title></head><body> This is a test A This is a test B This is a simple HTML fileThis is a test C </body></html>