# Object-Oriented Programming – Practical Exam

## Problem 1 – HTML Rendering Engine

A software company has designed a very simple **HTML rendering engine**. It supports only two types of elements: HTML elements and HTML tables. **HTML elements** have name and text content and may optionally have a list of child nested HTML elements. **HTML tables** are special kind of HTML elements that have fixed size (rows and columns) and hold rows \* columns cells which are HTML elements.

### Design the Class Hierarchy

Your **first task** is to **design an object-oriented class hierarchy** to model the HTML rendering engine **using the best practices for object-oriented design (OOD) and object-oriented programming (OOP)**. Avoid duplicated code though abstraction, inheritance, and polymorphism and encapsulate correctly all fields.

You are given few C# **interfaces** that you should **obligatory** implement and use as a basis of your code:

|  |
| --- |
| namespace HTMLRenderer  {  public interface IElement  {  string Name { get; }  string TextContent { get; set; }  IEnumerable<IElement> ChildElements { get; }  void AddElement(IElement element);  void Render(StringBuilder output);  string ToString();  }  public interface ITable : IElement  {  int Rows { get; }  int Cols { get; }  IElement this[int row, int col] { get; set; }  }  public interface IElementFactory  {  IElement CreateElement(string name);  IElement CreateElement(string name, string content);  ITable CreateTable(int rows, int cols);  }  public class HTMLElementFactory : IElementFactory  {  // TODO: implement the IElementFactory interface  }  } |

All your HTML elements should implement IElement. All your HTML table elements should implement ITable. HTML tables have no text content and their name is "table". HTML tables cannot have child elements.

Your IElement.Render(StringBuilder output) implementation should render the element as follows:

|  |
| --- |
| **<(*name)*>*(text\_content)(child\_content)*</(*name)*>** |

If ***(text\_content)*** is missing, it is not rendered. The *(child\_content)* is rendered by rendering all child elements in the order of their addition. If the element has no child elements, the ***(child\_content)*** is empty.

Your ITable.Render(StringBuilder output) implementation should render the table as follows:

|  |
| --- |
| **<table><tr><td>*(cell\_0\_0)*</td><td>*(cell\_0\_1)*</td>…</tr><tr><td>*(cell\_1\_0)*</td><td>*(cell\_1\_1)*</td>…</tr>…</table>** |

For each row its content is rendered enclosed between the <tr> and </tr> tags. For each column inside a row its element content is rendered enclosed between the <td> and </td> tags.

The IElement.ToString() method renders the element into a string and returns it as result.

The Name and TextContent properties are optional and can be null (null values are not rendered). When a TextContent is rendered, the HTML special characters <, > and & are escaped as &lt;, &gt;, &amp; respectively. When elements are rendered, no spacing or new lines is put between them.

### Write a Program to Execute C# Statements Using Your Classes

Your **second task** is to write a program that **executes a standard C# code block** (sequence of C# statements) using your classes and interfaces and prints the results on the console. A sample C# code block is given below:

|  |
| --- |
| **IElementFactory htmlFactory = new HTMLElementFactory();**  **IElement html = htmlFactory.CreateElement("html");**  **IElement h1 = htmlFactory.CreateElement("h1", "Welcome!");**  **html.AddElement(h1);**  **Console.WriteLine(html);** |

The statements are guaranteed to be **valid C# sequences of statements** that will compile correctly if you implement correctly the interfaces and classes described above. The statements will be no more than **500**. Each statement will be less than **100 characters** long. The statements end with an empty line. The code block will not throw any exceptions at runtime and will not get into an endless loop.

### Additional Notes

To simplify your work you are given a C# code block execution engine that compiles and executes a sequence of C# statements read from the console using the classes and interfaces in your project (see the file HTMLRenderer-Skeleton.rar). Please put all your code directly in the namespace “HTMLRenderer”.

You are only **allowed to write classes**. You are **not allowed to modify the existing interfaces**.

### Sample Input

|  |
| --- |
| **IElementFactory htmlFactory = new HTMLElementFactory();**  **IElement html = htmlFactory.CreateElement("html");**  **IElement h1 = htmlFactory.CreateElement("h1", "Welcome!");**  **html.AddElement(h1);**  **Console.WriteLine(html);**  **ITable table = htmlFactory.CreateTable(3, 2);**  **table[0, 0] = htmlFactory.CreateElement("b", "First Name");**  **table[0, 1] = htmlFactory.CreateElement("b", "Last Name");**  **table[1, 0] = htmlFactory.CreateElement(null, "Svetlin");**  **table[1, 1] = htmlFactory.CreateElement(null, "Nakov");**  **table[2, 0] = htmlFactory.CreateElement(null, "George");**  **table[2, 1] = htmlFactory.CreateElement(null, "Georgiev");**  **html.AddElement(table);**  **IElement br = htmlFactory.CreateElement("br", null);**  **html.AddElement(br);**  **IElement div = htmlFactory.CreateElement("div",**  **"(c) Nakov & Joro @ <Telerik Software Academy>");**  **html.AddElement(div);**  **Console.WriteLine(html);**  ***(empty line)*** |

### Sample Output

|  |
| --- |
| **<html><h1>Welcome!</h1></html>**  **<html><h1>Welcome!</h1><table><tr><td><b>First Name</b></td><td><b>Last Name</b></td></tr><tr><td>Svetlin</td><td>Nakov</td></tr><tr><td>George</td><td>Georgiev</td></tr></table><br></br><div>(c) Nakov &amp; Joro @ &lt;Telerik Software Academy&gt;</div></html>** |