**Data Structures Fundamentals – Retake Exam – 03.10.2020**

This document defines the examination for ["Data Structures – Fundamentals (C#)" course @ Software University](https://softuni.bg/trainings/3112/data-structures-fundamentals-with-csharp-september-2020).

Please submit your solutions (source code) of all below described problems in [Judge](https://judge.softuni.bg/Contests/2580/Data-Structures-Fundamentals-with-CSharp-Retake-Exam-10-Oct-2020).

Any code files that are part of the task are provided as **Skeleton**. In the beginning import the project skeleton, do not change any of the interfaces or classes provided. You are free to add additional logic in form of methods in both interfaces and implementations you are not allowed to delete or remove any of the code provided. Do not change the names of the files as they are part of the tests logic. **Do not change the namespaces** or move any of the files provided inside the skeleton if you have to add new file add it in the same class library.

Some **tests may be provided** within the skeleton – use those for local **testing and debugging**, however there **is no guarantee that there are no hidden tests added inside Judge**.

Please follow the exact instructions on uploading the solutions for each task. Submit as **.zip archive** the files contained inside **the library** folder (everything **except** the content inside **bin** and **obj** folders) this should work for all tasks regardless of current DS implementation.

You have to **study** the provided **skeleton**. The code is **separated** inside **different** **libraries**.

There **are** **few** **entities** **inside** the **project** you are **allowed** to **add** code to those, for example **Equals()**, **GetHashCode(), CompareTo()** etc…

# Browser History

You are given a skeleton with a class **BrowserHistory** that implements the **IHistory interface.** The browser history stores **links**. A single link has a **URL** (string) and **loading time** in seconds (int). You task is to store the provided links inside the history. The **last added link** is on **top** of the **history**.

Implement the following methods**:**

* **int Size** – returns the **number of links** that are stored inside the browser history.
* **void Open(ILink link)** **– adds** a link inside the history**.**
* **ILink GetByUrl(string url) –** returns a link inside the history that **matches** the **provided URL**. Return **null** if there **isn't** one.
* **bool Contains(ILink link) –** returns **true** if the link is **stored** and **false otherwise.**
* **ILink LastVisited() –** returns the last visited link (the **last added** inside the **history**). Throws an **InvalidOperationException** if the history is **empty**.
* **ILink DeleteLast() – removes** the **last visited** link from the history and **returns it**. Throws an **InvalidOperationException** if the history is **empty**.
* **ILink DeleteFirst() – removes** the **first visited** link from the history and **returns it**. Throws an **InvalidOperationException** if the history is **empty**.
* **int RemoveLinks(string url) – removes** all llinks that **contain** the **provided URL** (case insensitive) and **returns** their **count**. Throws an **InvalidOperationException** if there aren't **any links containing the URL**.
* **void Clear ()** **–** **clears the browser history**
* **ILink[] ToArray() –** returns **all the links** inside the history as an **array**. The returned **order** is from **last** visited to **first** visited.
* **List<ILink> ToList() –** returns **all the links** inside the history as a **list**. The returned **order** is from **last** visited to **first** visited.
* **string ViewHistory –** returns a **string** result of the entire history from **last visited** to **first visited** in the following format (check the provided unit tests for an example as well):

**-- {LastVisitedURL} {LoadingTime}s**

**-- ….**

**-- ….**

**-- {FirstVisitedURL} {LoadingTime}s**

If there are **no links** return

1. **Performance Tests – Browser History**

For this task you will only be required to submit the **code from the Browser History problem**. If you are having problem with this task you should **perform detailed algorithmic complexity analysis** and try to **figure** **out** **weak** spots inside your implementation.

### Hint

Both **DeleteFirst()** and **DeleteLast()** should work in constant time **O(1)**

1. **DOM**

You are given a skeleton with a class **Document Object Model** that implements the **IDocument interface.** The DOM **stores** different types of HTML elements in **hierarchical** order. Each **HTML element** has the following properties:

* **enum ElementType – Document, Html, Body, Div, Span…**
* **IHtmlElement Parent –** the **parent** of the given element
* **List<IHtmlElement> Children -** a **collection** of children. The tree structure **should not** have a limit of how many children it holds.
* **Dictionary<string, string> Attributes –** a **collection** of attributes. Each attribute has a **unique** key value pair.

Your task is to implement the following methods:

* **public HtmlElement(ElementType type, params IHtmlElement[] children) –** HTML element **constructor** which initializes it's **properties** and sets the **parent-child** relationships.
* **public DocumentObjectModel() –** the **empty DOM constructor** which creates a **root** of type **Document**. The document should have a **single child** of type **Html**, and the **Html** should have **two children** of type **Head** and **Body**.
* **public Contains(IHtmlElement element) –** returns **true/false** if the element is **contained** inside the DOM tree. Elements are **compared** by **reference**.
* **IHtmlElement GetElementByType(ElementType type) –** returns the **first** occurrence, in **level traversal** order (BFS), of the given element type. If there are **none** return **null**.
* **List<IHtmlElement> GetElementsByType(ElementType type) –** returns a **collection**, in **depth traversal** order (DFS), of the given element type. If there aren't **any** return **an empty collection**.
* **void InsertFirst(IHtmlElement parent, IHtmlElement child) –** inserts the **new child** element as a **first child** for the given parent. If the **parent** does **not exist** throw an **InvalidOperationException**.
* **void InsertLast(IHtmlElement parent, IHtmlElement child) –** inserts the **new child** element as a **last child** for the given parent. If the **parent** does **not exist** throw an **InvalidOperationException**.
* **void Remove(IHtmlElement element) – removes** the provided HTML element from the DOM tree. When removing an HTML element don't forget to remove **all** of **its children** as well. If the HTML element is **not present** throw an **InvalidOperationException**.
* **void RemoveAll(ElementType type)** – **removes** all HTML elements that have the **provided** **element** **type** from the DOM tree. When removing an HTML element don't forget to remove **all** of **its children** as well. If there are **no such** elements, do nothing.
* **bool AddAttribute(string attrKey, string attrValue, IHtmlElement element)** – **adds** an attribute with the given **key** and **value** to the HTML element. Attributes should hold **unique** key-value pairs, so if the key **already exists** return **false**, otherwise return **true**. If the HTML element is **not present** throw an **InvalidOperationException**.
* **bool RemoveAttribute(string attrKey, IHtmlElement element)** – **removes** the attribute with the **given key** from the HTML element. Returns **true** if the key **is contained** inside the element, otherwise return **false**. If the HTML element is **not present** throw an**InvalidOperationException**.
* **IHtmlElement GetElementById(string idValue)** – returns the **first** element, in **level traversal** order (BFS), that has the **"id"** attribute as **key** that answers to the **provided value**. Return **null** if such element **does not** exist.
* **override string ToString()** – override the implementation of **ToString()** and return a **visualization** of the DOM tree in following format:
* **Document**
* **Html**
* **Head**
* **Body**
* **Paragraph**
* **Div**
* **Span**

Each level has **2 spaces more** than the **previous**, starting from **zero spaces** for the Document element. Check the provided unit test for more details. The dash symbol "-" is **not required** for the final result.

1. **Performance Tests – DOM**

For this task you will only be required to submit the **code from the DOM problem**. If you are having problem with this task you should **perform detailed algorithmic complexity analysis** and try to **figure** **out** **weak** spots inside your implementation.