Industrial Programming

F21SC

Coursework 1

Developing a Simple Web Browser

Student Name: Jiancheng Zhang

Student ID: H00341619

Contents

Introduction 3
Requirements' checklist 3
Design Considerations 3
Class design 3
Data structures 3
GUI design 4
Advanced language constructs 4
User Guide 5
Developer Guide 8
Testing 10
Reflections on programming language and implementation 11
Conclusions 12

Introduction

Within this report, I will be documenting the design principles and thinking used for the development of the web browser and will also use it as a guide for users wanting to understand my programme more clearly. Finally, I will make reflections and conclusions on what I have learnt.

It is assumed throughout my work that no URL and/or website title contains the vertical bar (i.e. "|") character.

Requirements' checklist

All requirements are achieved.

Design Considerations

Class design

We have eight classes, with each of them having a clear responsibility:

- **FileIO** class handles the problem of reading and writing a file.
- **BrowserManager** class is where we deal with history and favourites.
- **MainForm** class for achieving buttons and other GUI functions.
- **MyTabPage** inherits from **TabPage** class to add HTTP functions in each TabPage.
- **URL class** has three properties, a title, a URL, and a timestamp.
- **Program class** has the Main method.
- **FormEditHomePage** class is a form that edit home page.
- **EditFavourite** class is a form that edit favourites

Data structures

For favourites, I use Dictionary to store favourites, since we have to keep URLs in favourites unique, and Dictionary can determine whether a URL is in the favourites already or not (in O(1) time).The pros of this are that we can add, delete, and edit a URL in O(1) time complexity. But the consequence of this is when we load all the favourites into the ListBox after the program starts, we have to sort all the favourites by the added time ascending order, and this will cost O(N*logN). On the other hand, we can use SortedDictionary, which is a red-black tree, s previous four operations take O(logN) time, and load favourites takes O(N). However, in this case, we may use those four operations more frequently, so Dictionary could be better.

For history, I use List to keep the history as a list, since history should be in the order of access-time and does not have to keep the URL unique. Whenever we read and write to the history.txt, we keep every URL in the original order, so no extra sort is needed. Once an HTTP request is sent, a URL will be added into the history which will cost O(N) since we insert it into the first position of the List, which is the same as when we delete a URL in the list by index. The advantages

are we can get a URL by index in O(1) time complexity, and when loading history into ListBox, it spends O(N) time complexity. If we use LinkedList rather than List, we could easily add a new URL to the front of the list in O(1) time, but then we cannot use the index to get a URL in O(1) time. Since we will be inserting URLs into the history often, LinkedList may the best choice. On the other hand, it is difficult to keep a LinkedList and a List (ListBox) the same.

For MyTabPage class, I use a List to store all URLs that are requested in this page, so a different TabPage has a different URL history list. This makes it convenient to achieve moving to the previous or next page. Additionally, we must maintain an index that points at the position where the current URL in the list. A solution is to make a LinkedList by ourselves in the MyTabPage class, where we will maintain a head rather than an index, and each node (with two pointers, one previous, the other next) representing a URL class instance. Both solutions can finish the same job in the same time complexity O(1).

GUI design

The GUI is like any typical browser. All the buttons, along with the URL textbox, sit at the top of the form, with previous, next, refresh buttons on the left of the URL textbox, and the rest of the buttons on the right side of the URL textbox. F5 is the shortcut for reloading a page and Enter is the shortcut for redirecting a HTTP request. The AddToFav button has two colours:

- Red, indicating the current URL is not in the favourites list, and
- Lime, indicating that the current URL is on the favourites list.

Clicking on the AddToFav button will change the colour if the URL is on the favourites list and then removed, otherwise it will just add the URL to the list.

The Add button adds a new TabPage, and the Remove button deletes the TabPage. The Home button will create a new TabPage with the homepage, and if we right-click on the home button, there will be a new form showing up where we can edit the home page in it.

Favourites and History are two ListBoxes. Clicking any URL in them leads to a new TabPage, showing the page the URL corresponds to.

The Bulk button will open a File Explorer; we can select a txt file, and a new TabPage will show the result of the bulk download.

Advanced language constructs

When reading from the corresponding txt files, both history and favourite file have multiple lines, but the difference is feeding these into different data structures. So, I use delegate and generics here to reduce duplicate code in FileIO class. In BrowserManager class, we can pass the different method that shows how to transfer each line into Dictionary and List.

I also use LINQ when loading favourites to sort every favourite by time.

Events are used in the BrowserManager class. Since adding a URL to history is performed by calling MyTabPage class instance, the MyTabPage class has no idea what URL in the ListBox is in the MainForm class, but the BrowserManager class does. So, after a URL is added into the history List in the BrowserManager class, it must tell the MainForm class to add the same URL into the history ListBox. This is what happens to the favourites too. In the MainForm class, I define two event handler methods, where the signature matches the delegates in the BrowserManager class when initializing MainForm instance, two methods will subscribe to the events respectively.

User Guide

First of all, I will introduce some very basic operations of this browser program.

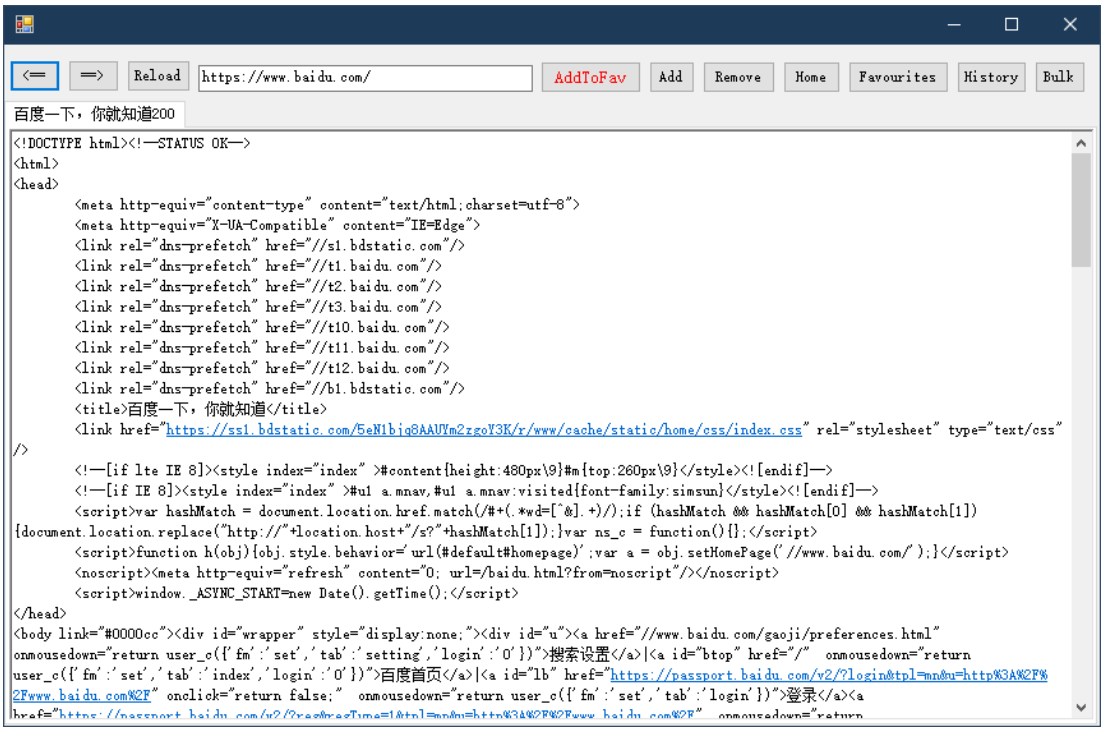

Figure 1: the overview of the MyBrowser

In the top middle, there is a URL TextBox where we can input a URL here, and once we press the “Enter” key on the keyboard, the current tab page will show the information. Normally, if the URL is correct and the website can be accessed successfully, the website source code will be listed in the panel. Otherwise, the related error information will show up, such as 4xx and 5xx HTTP status codes.

After we input a lot of URLs in a tab page, if we want to see the previous webpage, we can click the "🡐" button on the top left corner, and we will be redirected to the previous page. And if we want to see the next page, we can click the "🡒" button which is on the right-hand side of "🡐" button. We note that if the current page is the last or first one, then "🡒" or "🡐" button is unavailable.

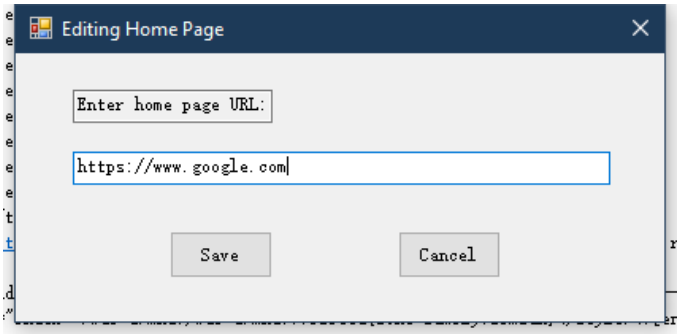

Figure 2: Editing Home Page Form

We can create a new tab page by clicking the "Add" button and delete the current tab page by clicking the "Remove" button. If we want to see the home page again, we just click the "Home" button. Furthermore, if we want to change the home page, we can right-click "Home" button, and a new form will show up. We confirm the new home page URL by clicking on the save button. If we change our mind, click the "Cancel" button or the top right "X" button, keeping the settings as before.

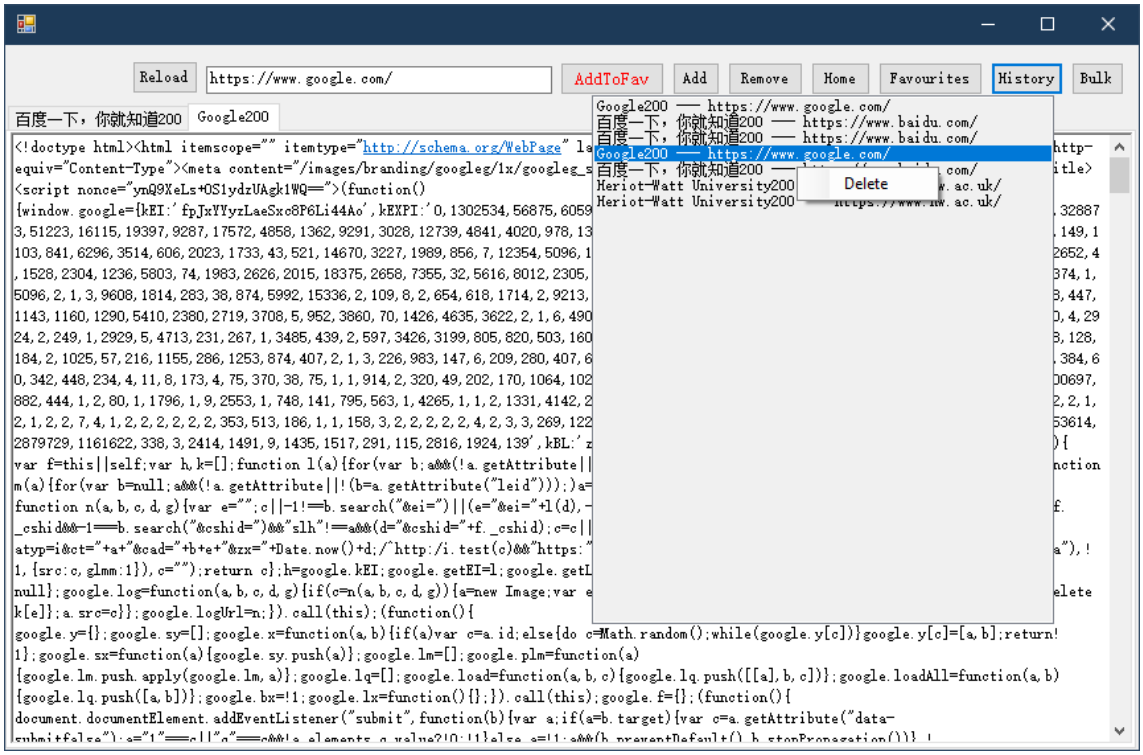

Figure 3: History

Once we click the “History” button, a history list shows up, and if clicked again, the list will disappear. In the list, we can see all URLs that have been accessed by the browser. If we left click one of them, a new tab page is generated with the URL’s content. On the other hand, if we right-click one of them, we can easily remove it by clicking the “Delete” option.

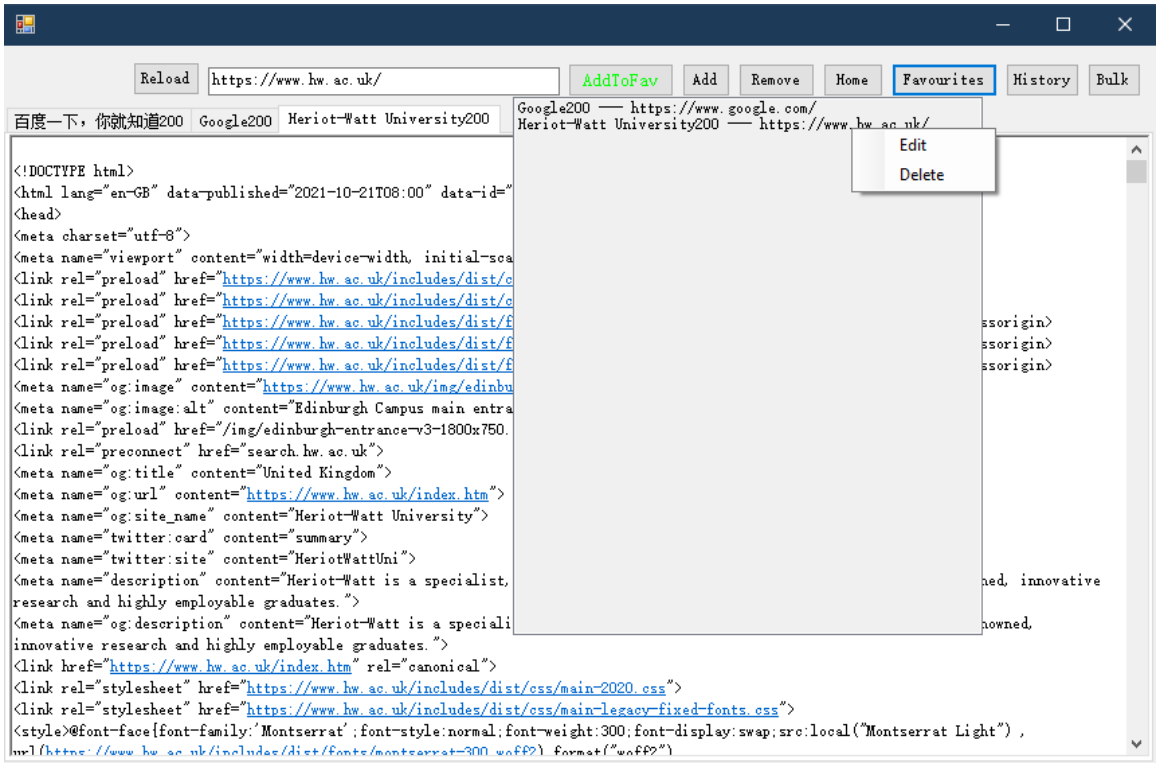

Figure 4: Favourites

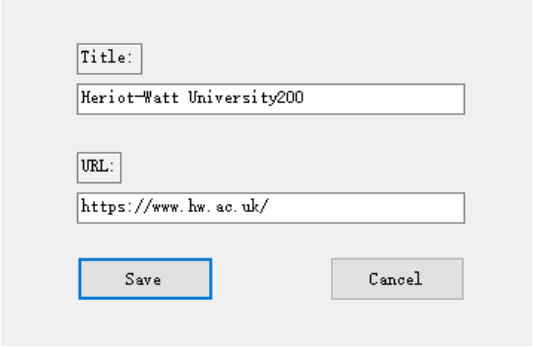

Figure 5: Edit Form

We can access the favourites list by clicking the “Favourite” button. Most operations are the same as in the history list, except for the “Edit” option. If we click that, a new single form shows up where we can edit the title and URL. We note that the title cannot contain “|” character. Moreover, if a URL is in the

favourites list, then “AddToFav” button is green, otherwise the button is red. The “AddToFav” button can add or remove the current URL in the favourites.

Finally, the “Bulk” button will open a file explorer, where the user will be able to open a txt file only. A new tab page will be generated with the download status of each URL in the selected file.

Developer Guide

URL class: this class has three properties:

1. title is the website title;
2. url is the URL of the instance;
3. time is the timestamp that creates this instance.

```
private String title;
private String url;
private DateTime time;

public URL(String title, String url, DateTime time)
{
    this.time = time;
    this.title = title;
    this.url = url;
}
```

URL class overrides the ToString() method, which is a form of serialization.

```
public override String ToString()
{
    //this is actually a danger way to add "|" as split sign,
    //Although it won't shows in url and time,
    //the user may input this in the title
    return title + "|" + url + "|" + time.ToFileTime();
}
```

FileIO class: the readFile method needs a function that represents how to transfer each line into data structures.

```
public void readFile<T>(ref T t, String fileName, SpecificStructure<T>
structure)
{
    ……
                //because it is reference, so we can change the
original data structure
                structure(ref t, temp, line);
    ……
}
```

MyTabPage class: this class is inherited from the TabPage class, with two overload constructor methods.

```
    //newPanel shows the code of a website
    public RichTextBox newPanel = new RichTextBox();
```

```
    //because we have to move forward and backward on a tab,
    //so we need a list to keep all url that shown on this tab
    public List<String> urlList = new List<string>();
    //this is a pointer to indicate the position in the list
    private int index = -1;

    private BrowserManager browserManager;

    /// <summary>
    /// this constructor starts a rich text box in a new tab
    /// </summary>
    /// <param name="browserManager"></param>
    public MyTabPage(ref BrowserManager browserManager)
    {
        ……//start a RichTextBox
    }

    /// <summary>
    /// this is a overload constructor, starts with a url
    /// </summary>
    /// <param name="browserManager"></param>
    /// <param name="url"></param>
    public MyTabPage(ref BrowserManager browserManager, String url)
    {
        ……
    }
```

sendHttpRequest is the most important method in this class; it accepts a URL and tries to send an HTTP request to this URL. If the server correctly responds, the source code will be shown in the TextBox, and the title will be present in the title of the tab page. If an error has occured, the error code or message shows in the TextBox. The bulkDownload method is similar to this method since the source code size shows rather than the code itself.

Reload, previous, and next methods are actually operating the pointer of the URL list in this class. Since after sendHttpRequest method the index will become the last one. In a normal case, this is not a problem, but if we want to move the previous one, we have to change the index to the original previous one.

```
public void reload(String url)
{
    int temp = index;
    sendHttpRequest(url);
    index = temp;
    urlList.RemoveAt(urlList.Count - 1);
}

public int previous()
{
    if (index > 0)
    {
        int temp = index - 1;
        sendHttpRequest(urlList[--index]);
        index = temp;
        urlList.RemoveAt(urlList.Count - 1);
```

```
            return 0;
        }
        …
}
```

BrowserManager class: A Dictionary<String, URL> handles all the favourites, A List<URL> keeps all the history, and a property. This class mainly provides CURD methods of those two data structures and property.

These three methods are the three different ways to read from txt file

```
public static void loadToList(ref List<URL> list, String[] temp, String
line)
{
    //Deserialization
    temp = line.Split('|');
    list.Add(new URL(temp[0], temp[1],
DateTime.FromFileTime(long.Parse(temp[2]))));
}

public static void loadToDic(ref Dictionary<String, URL> dictionary,
String[] temp, String line)
{
    temp = line.Split('|');
    dictionary.Add(temp[1], new URL(temp[0], temp[1],
DateTime.FromFileTime(long.Parse(temp[2]))));
}

public static void loadDownload(ref List<String> list, String[] temp,
String line)
{
    list.Add(line);
}
```

MainForm class: this class links the kinds of buttons and forms together, providing functions to users.

Testing

There are no known bugs.

URL TextBox:

1. Any string that does not match the regular expression will lead to the wrong URL text shown in the tab page title.
2. If the browser cannot find a URL on the internet, this will lead to a web error text in the tab page title; the panel will show the error message.

Home page:

1. If the input URL does not match the regular expression, an error dialog will show up.

Favourites:

1. In edit form, if the input URL does not match regular expression, an error dialog will show up.

Bulk download:

1. If the user tries to open a non-txt file, an error dialog will show up.
2. If the txt file contains an invalid URL or empty line, an error message will show as a result.

Reflections on programming language and implementation

In this application, I found LINQ and its properties are very easy to use and powerful. They can be easily set up, and since we can use Lambda expressions in LINQ, we can handle data more flexibly than SQL. This is just like Stream in Java, where in Java 8 the two biggest new features are Stream and Lambda. They usually show up together, helping us process complex datasets. The delegates and events have helped me reduce a lot of code. In my opinion, the reflection is the most powerful language feature; I used reflection to get a class loader in Java before, but I found it is not suitable for this application.

There are two major limitations to this application.

The first is too much file writing. One possible solution is using a buffer. For example, say we have a 10-size buffer, each time an HTTP request is sending, a URL will be added into the buffer. If the buffer is full, then it starts writing into the file. But the side effect is if the program crashed, then history in this buffer will be lost, since they are in the memory, not the disk. So, we can find a trade-off between a 0-size buffer and a huge size buffer.

The second is too many relationships between components in MainForm class. Like one component status changed, lots of other components have to change. One possible solution is decoupling, which finds out all the single direction relationships; if one component only relies on other components, then we can group them together. “Low coupling, high cohesion” is the most important target in developing software, but it is very difficult to achieve.

I believe system languages are more suitable in the browser application domain than scripting languages. The reason is we have to deal with low-level functions, like sending the HTTP request and using data structures frequently. Moreover, the Windows operating system supports C# very well, so the program can run smoothly in most Windows systems. If we write in Java, then we can use our browser on a wide range of devices, such as a washing machine. A system language can handle the tricky part of the whole project. I would use SpringBoot 2 to develop a website; I can manage MySQL database and Redis memory

database at the same time and can easily retrieve data then send out to the client, letting SpringBoot do the dirty work of the website development project.

Conclusions

It has been very beneficial to me to have used advanced features of C# I have never heard of before. Although C# and Java share a lot of the same language features, they all have their own unique features. These features help the language grow stronger and become suitable for modern technology. If I were to have more time on this coursework, I would turn it into a multithreading program, just like the advanced browsers we use in daily life.