Broken Links Finder

- Author: Michele Cattaneo
- Language: Java
- Libraries used: Lang, Util, Net, Io, Time

The goal of this software is to find all links present in some html files and check whether they are working in case they are internal links and active in the case that they are external links.

The program is divided in 6 parts.

1. Find all files that are html files.
2. For each of them, extract the content of each `<a>` tag present.
3. From the content of the `<a>` Tag, extract only the link and ignore styles, ids and classes of the tag.
4. Save the found links in a collection.
5. Check the validity
6. Print and optionally save the results.

Part 1:

Thanks to the Java.io.FilenameFilter it is possible to get an array of all Files with the wanted extension, in this case .html, given the path of a Folder.

```
File dir = new File(path);

        File[] matches = dir.listFiles(new FilenameFilter(){
            public boolean accept(File dir, String name){
                return name.endsWith(".html");
            }
        });

        return matches;
```

Part 2:

Thanks to the Java.utile.regex library, it is possible to extract specific part of a given String, according to the corresponding Regular Expression given.

```
private static final String A_TAG_PATTERN = "(?i)<a([^>]+)>(.+?)</a>";
private static final String HREF_TAG_PATTERN = "\\s*(?i)href\\s*=\\s*(\"
([^\"]*\")|'[^']*'|([^'\">\\s]+))";
```

Using this RegEx we can iterate through the .html file (given as a String), and return a List containing all matches to the RegEx.

Part 3:

From the content of the `<a>` Tag, we only want href links.

```
private static String extractLink(String in){

        int firstIndex = in.indexOf("href");
        int delta = ("href".length())+2;

        int endIndex = in.length();
        for(int i=firstIndex+delta; i<in.length();i++){
            if(in.charAt(i) == '"'){
                endIndex=i;
                break;
            }
        }
        String out;
        out = in.substring(firstIndex+delta,endIndex);
        return out;
    }
```

We can do that by finding the first substring containing "href", and we will know that out searched substring will start at the index of "href" + the length of "href=' ".

We can then search for the next double quote with a for loop starting at the first index of our wanted substring. The index that will be found is going to be the end index of our wanted substring.

Part 4:

We can then save for each found link an Object containing its informations

```
public class htmlLink {

    public enum type{
        EXTERNAL, INTERNAL
    }
```

```
        public enum status {
            OK, NOT_WORKING, TO_CHECK
        }

        static type type;
        String href;
        status status;

        public htmlLink(type type, String href, status status){
            this.type=type;
            this.href=href;
            this.status=status;
        }

    }
```

We can differentiate between internal and external links by looking at the start of the string. If it starts with the https protocol, then it is external.

Part 5:

We can then check the validity of a link in two ways:

- If it is external:

  We can get the response of the https protocol to check whether it is online or not.

```
    public static String  verifyLinkActive(String linkUrl) {
            try {
                URL url = new URL(linkUrl);
                HttpURLConnection httpURLConnect = (HttpURLConnection)
    url.openConnection();
                httpURLConnect.setConnectTimeout(3000);
                httpURLConnect.connect();
                if (httpURLConnect.getResponseCode() == 200) {
                    return linkUrl + " - " +
    httpURLConnect.getResponseMessage();
                }
                if (httpURLConnect.getResponseCode() ==
    HttpURLConnection.HTTP_NOT_FOUND) {
                   return linkUrl + " - " +
    httpURLConnect.getResponseMessage() + " - " +
    HttpURLConnection.HTTP_NOT_FOUND;
                }
                if(httpURLConnect.getResponseCode()== 404) {
                    return linkUrl+" -
    "+httpURLConnect.getResponseMessage();
                }
                if(httpURLConnect.getResponseCode()== -1) {
                    return linkUrl+" -
    "+httpURLConnect.getResponseMessage();
                }
            } catch (Exception e) {
            }
            return "other";
        }
```

- If it is internal :

  We need to cover the case that the path is absolute or relative. We can create two File Objects, one supposing that the given path is relative, and the other supposing that the path is absolute. For the first one we need to add the absolute path of the folder in which the file is present and the relative path to get an absolute path. If the path is already absolute we do nothing. We need they one of the two must exist, so we return the response of one OR (logical) the other.

```
    private static boolean isValid(String Path, String documentPath){

            //case that the path is absolute
            File f1 = new File(Path);

            //case that the path is relative, add it to the absolute path
    where the
            //file is found

            File f2 = new File(documentPath+"/"+Path);
            //either the relative or absolute path must work
            return (f1.exists() || f2.exists());
        }
```

Part 6:

We then go through the List and print the link found and status for each page that has been checked.

If it is requested to, the result can be saved in a file. To to that, while going through the list and printing the results in the console, we can create a string containing the whole result, by adding the new line printed to the string. Finally the string will be written into a file saved into the given location.

```
    if(save) {
      BufferedWriter writer = new BufferedWriter(new
     FileWriter(resultPath+"/result.txt"));
      writer.write(outputString);
      writer.close();
      System.out.println("\nProgram terminated correcly. Safe to close it.\n
    Results are located in: " + resultPath);
    }
```

Challenges

One problem that I encountered was that I supposed a Tag would only containg the link and no other strings, like id, class or style. In Part 3 there's the solution that I found to this problem.

Another problem that I had, but to whom I quickly got a solution was that of the local links, which create a problem when checked since the Java Software is not in the same location as the files checked. The solution is quickly found, since the first thing that the software ask you to give it the absolute path of the filder containing the files to check.

Final thoughts

I've been programming in Java for a while now but this was a first time for me to use Java to check files and handle them. Java is my favourite language (in part because is the one I know best, in part because I find it beautiful) but I think it is not the optimal solution for this kind of problems. I'm sure there are some languages that would have made the job a lot easier for me. I'm not happy with the code I wrote; it is not structured as I would have wanted it to be. I'm sure there are more elegant solutions to a lot of what I've done, since there is a lot of hard-coding in my code. I also don't like how the Main function is used; I should have used more classes and kept the Main cleaner.