WebCrawler
Project 1

CPSC 2150

Colin Fitt - cfitt

Due: 10/06/14

```
    }

    public void download(String savePath) throws IOException
    {
            for (WebFile webFile : files)
               webFile.saver(savePath);
            for (WebImage webImage : images)
               webImage.saver(savePath);
    }

        public void addWebElements(WebPage webPage) {
                for (WebFile webFile : webPage.getFiles())
                        files.add(webFile);
                for (WebImage webImage : webPage.getImages())
        images.add(webImage);
        }
}
```

********** **DownloadRepository.java** **********

```
package cu.cs.cpsc215.project1;

import java.io.IOException;
import java.util.ArrayList;

/**
 * Downloads and passes WebElements
 * key piece in saving, which is the last step
 * Previous implementations of this class were intended to directly save each
element
 *                  this was a bad idea, plus it was slow
 * Singleton class, hopefully I set it up correctly
 *
 * @author cfitt
 *
 */

public class DownloadRepository implements Repository{
   private static DownloadRepository instance;
   private ArrayList<WebFile>  files;
   private ArrayList<WebImage> images;
   //Combine above to have a list of all places visited
   private ArrayList<String> names;

   private DownloadRepository()
   {
      files  = new ArrayList<WebFile>();
      images = new ArrayList<WebImage>();
   }

   public static DownloadRepository getInstance()
   {
      if (instance == null)
         instance = new DownloadRepository();
      return instance;
   }

   public void addName(String urlName){
         names.add(urlName);
   }

   public boolean visted(String urlCheck){
         if(!names.contains(urlCheck))
               return true;
         return false;
```

********** **Repository.java** **********

```
package cu.cs.cpsc215.project1;

import java.io.IOException;

public interface Repository {
        public void addWebElements(WebPage webPage);
   public void download(String savePath) throws IOException;
}
```

********** **WebCrawler.Java** **********

```
package cu.cs.cpsc215.project1;

import java.io.File;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;

/**
 * WebCrawler
 * Classes Required:
        WebElement interface
        WebCrawler class, The Working Class
        WebPage class, impl WebElement
        WebFile class, impl WebElement
        WebImage class, impl WebElement
        DownloadRepository interface
        Repository, impl DownloadRepository (Singleton class)

 * WebCrawler:
        Main Driver In Program - Take in only 3 arguments from command line
                <URL>, <Depth of crawl>, <Save Directory Loc>

        Uses of each given Param:
        <ULR> -> save to string -> Param for, WebPage(StringOfULR)
        <Depth> -> save to int -> Objective, crawl into pages of set depth
                                Find uses when developing:
                                Max crawl depth first?
                                recursion?
                                Loop with <Depth> as Param?
        <Dir Loc> -> save to string -> Main Directory for saving
                                Will create a Files and Images folder.
 *
 * Personal Notes:
 *              The save directory needs to already exist or you will get an error.
 *              This project was especially hard, because of my unfamiliarity with
Java, Jsoup, HTML...
 *              but I feel like I produced something that actually "fitts" the
requirements
 *              I want to improve this, so future implementations may include my
first version,
 *                      which I attempted to make my own parser.
 *
 *              in all, I hope this works...
```

```
 *
 * @author cfitt
 *
 */

public class WebCrawler {

	private static String url;
	private static int depthCrawl;
	private static String savePath;

	public static void main(String[] args) throws IOException{//Check If number
of arguments is correct
    if(args.length != 3) {
	 System.err.println("Please Submit args in format: <URL> <int
DepthofCrawl> <Save Directory>");
	 return;
     } else {

		try {
			url = args[0];
			URL testUrl = new URL(url);
			URLConnection urlConnection = testUrl.openConnection();
			urlConnection.connect();
		} catch (MalformedURLException e) {
			System.err.println("Not a valid URL");
	   return;
	} catch (IOException e) {
		System.err.println("Could not open URL");
		return;
	}
//---------------------------------------------------------------------------
		try {
			depthCrawl = Integer.parseInt(args[1]);
	} catch (NumberFormatException e) {
		System.err.println("Argument is not of int type");
		System.err.println("Please Submit args in format: <URL> <int
DepthofCrawl> <Save Directory>");
	   return;
	}
//---------------------------------------------------------------------------
		savePath = args[2];
	File path = new File(savePath);
	if(!path.exists()) {
		System.err.println("File Path Invalid");
```

```
                System.err.println("Please specify of ammend the File Path within
arguments");
                System.err.println("Please Submit args in format: <URL> <int
DepthofCrawl> <Save Directory>");
            return;
          }
//---------------------------------------------------------------------------
    System.out.println("Attempting to Crawl on " + url);
                WebPage initPage = new WebPage(url);
                initPage.crawl(depthCrawl);
                try {
                        DownloadRepository.getInstance().download(savePath);
                } catch (IOException e) {
                        System.err.println("");
                        }
    }
    System.out.println("Finished Crawling");
        }
}
```

********** **WebElement.Java** **********

```
package cu.cs.cpsc215.project1;

public interface WebElement {
	void saver(String savePath);
}
```

********** **WebFile.Java** **********

```
package cu.cs.cpsc215.project1;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.MalformedURLException;
import java.net.URL;

public class WebFile implements WebElement{

	String fileurl;

	public WebFile(String fileurl){
		this.fileurl = fileurl;
	}

	public void saver(String filePath){
		URL url;

			try {
				url = new URL(fileurl);
				String nameofFile =
fileurl.substring(fileurl.lastIndexOf("/") + 1);
				filePath += "files/";
			new File(filePath).mkdirs();
			//Makes a new folder within the directory specified

			File localFile = new File(filePath + nameofFile);
			try {
			    InputStream fileReader= url.openStream();
			    OutputStream fileWriter = new FileOutputStream(localFile);

				byte[] b = new byte[2048];//This should be big enough
				int len;

					while ((len = fileReader.read(b)) != -1) {
						fileWriter.write(b, 0, len);
					}

				fileReader.close();
				fileWriter.close();
			} catch (IOException e) {
```

```
                    System.err.println("Could not save file " + nameofFile + " to " +
filePath);
                }
                } catch (MalformedURLException e1) {
                        System.err.println("Could not save file to " + filePath);
                }
        }
}
```

********** **WebImage.Java** **********

```
package cu.cs.cpsc215.project1;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.URL;


public class WebImage implements WebElement{

        String imgurl;

   public WebImage(String imgurl) {
        this.imgurl = imgurl;
   }

        public void saver(String savePath) {
                URL url;

                try {
                        url = new URL(imgurl);
                        String imgName = imgurl.substring(imgurl.lastIndexOf("/") +
1);
                        savePath += "images/";
        new File(savePath).mkdirs();

        File localFile = new File(savePath + imgName);

        InputStream reader = url.openStream();
        OutputStream writer = new FileOutputStream(localFile);

        byte[] b = new byte[2048];
        int length;
        while ((length = reader.read(b)) != -1) {
         writer.write(b, 0, length);
        }
        reader.close();
        writer.close();
                } catch (IOException e){
          System.err.println("Could Not Save Image " + imgurl);
     }
         }
```

}

********** **WebImage.Java** **********

```
package cu.cs.cpsc215.project1;

import java.io.IOException;
import java.util.ArrayList;
import org.jsoup.Connection.Response;
import org.jsoup.Jsoup;
import org.jsoup.helper.HttpConnection;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

/**
 * WebPage (imp WebElement):
        Constructor()
                Pass an address and save said address

        Crawl(<Depth>)
                Multiple ways I could do this
                after time thinking about it, I feel the simplest method would be
                        to increment/decrement depth
                1) Create ULR object and data structures needed
                        Data Structure choice is vast for this project
                        play around with a few to find the best
                                so far: ArrayList, Map, or Tree of some sort
                2) Check if depth is < 0
                        if so, you're done, return page
                3) Save string of ULR passed in, to ULR object
                4) BufferedReader that poop
                        find we elements
                        Sort them
                5) Check if reading is empty, then end?
                6) decrement depth

                What does this still need to be functional?
                        A way to "parse" what the BufferedReader gives?
                                Use JSoup?
                                        JSoup is OK, but keep experimenting…
                                Make a Parser?
                                        More Control of what's being passed.
                                        Make a parserBeReadin(), It can't be that hard
                Getters()
                        Still undecided on Data Structure I will use.
                        Once decided, just return the Data together. Easy.
                Sorter()
```

```
                        part of step 4 above
                        A way to sort through to see if an type
                                        of element as been read
                        pages will require you to enter and download within
                                same method of reading in,
                                just with <Depth> involved
                        basic alg:
                        if(WebPage)
                                add Web page to it's own Data for that type
                        if(WebImage)
                                add Image to it's own Data
                        if(WebFile)
                                add File to it's own Data?
                                or file contents?
 * @author cfitt
 *
 */

public class WebPage implements WebElement {

        String url;
        ArrayList<WebPage> pages = new ArrayList<WebPage>();
        ArrayList<WebImage> images = new ArrayList<WebImage>();
        ArrayList<WebFile> files = new ArrayList<WebFile>();
        ArrayList<String> visited = new ArrayList<String>();
        private static Elements media;
        private static Elements links;


        public WebPage(String url) {
                this.url = url;
        }

        public void crawl(int depth){
                if (depth < 0)//End Recursion Case
       return;

                System.out.println("Crawling " + url + " with a depth of " + depth);
                try { //Start Parse for page
                   Document doc = Jsoup.connect(url).get();
                   links = doc.select("[href]");//All Links
                   media = doc.select("[src]");//All Image Types

                   if(doc != null) { //If not a blank page, or down?
                           for (Element src : media) {
                                   String imgtopass = src.absUrl("src");
```

```
                        if (src.tagName().equals("img") &&
!visited.contains(imgtopass)) {
                                WebImage newImage = new
WebImage(imgtopass);
                            images.add(newImage);
                            visited.add(imgtopass);
                             System.out.println(imgtopass + " was downloaded");
                          }
                        }
                        /**
                         * this next part was tricky
                         *
                         * I was running into problems differentiating between
files/pages
                         * finally, after parousing the Jsoup Cookbook, I found a neat
recipe
                         *              after including I had a bunch of new problems,
                         *              but this current version works.
                         *
                         * Method:
                         *              1) check if the link is useable
                         *              2) set the content type
                         *              3) Use the content type passed to determine
                         *                              if link is a page or file
                         *              4) Pass pages into pages, which could be used
later if depth permits
                         *              5) pass files into folders, from
DownloadRepository
                         */
                        HttpConnection testLink = null;
                    Response testResponse = null;

                        for (Element link : links) {
                            //test if the link is another use-able link
                            String baselinkUrl = link.absUrl("href");
                            baselinkUrl = baselinkUrl.trim();
                      if(!baselinkUrl.contains("#")) {
                        try {
                          testLink = (HttpConnection) Jsoup.connect(baselinkUrl);
                          testLink.ignoreContentType(true);
                          testLink.ignoreHttpErrors(true);
                          testResponse = (Response) testLink.execute();
                        } catch (Exception e) {
                          System.out.println(e.getLocalizedMessage());
                        }
                        String contentType = null;
```

```
                        if(testResponse != null) {
                          contentType = testResponse.contentType();
                        }
                        if(contentType == null) {continue;}
                        if(contentType.toString().equals("text/html")) {
                         if(!visited.contains(baselinkUrl)) {
                                WebPage newPage = new WebPage(baselinkUrl);
                                pages.add(newPage);
                                visited.add(baselinkUrl);
                                System.out.println(baselinkUrl + " Added Page");
                         }
                        } else if(!visited.contains(baselinkUrl)) {
                                String fileUrl = baselinkUrl;
                                WebFile newFile = new WebFile(fileUrl);
                                files.add(newFile);
                                visited.add(baselinkUrl);
                                System.out.println(fileUrl + " Added File");
                        }
                       }
                          }
                   }
                } catch (IOException e){
                        System.out.println("Page Not Found");
                        return;
                }
                DownloadRepository downloadRepository =
DownloadRepository.getInstance();
     downloadRepository.addWebElements(this);
     for (WebPage page : pages)
                        page.crawl(depth - 1);
        }
        @Override
        public void saver(String savePath) {
    //unused in pages, since they hold elements, and are not elements
        }

        public ArrayList<WebImage> getImages(){
                return images;
        }

        public ArrayList<WebFile> getFiles(){
                return files;
        }

        public ArrayList<WebPage> getWebPages(){
                return pages;
```

```
        }
}
```