Project #1: Web Crawler[*]

1 Objectives

This project is designed to reinforce a number of learning objectives. First, and most important, you will gain experience designing and implementing a small (but interesting) Java application. This will reinforce your object-oriented design skills, as well as your ability to effectively apply the Java development tools. More specifically, the project will reinforce your understanding of Java interfaces, the Singleton pattern, Java exceptions, and the Java IO library. It will also improve your ability to debug Java-based systems.

Before describing the project requirements, two words of caution are in order. First, you should be prepared to experience some frustration with the level of detail provided in the assignment description. In order to learn object-oriented design skills, you need to be given a chance to do some design — which means that all of the implementation steps can't be laid out for you in advance. This level of flexibility is more in line with what you will encounter when you are employed as a developer.

The second warning is that this is a time-consuming assignment. You are expected to take the time to investigate the Java class libraries. Some of the most relevant classes are identified in this document. Others could significantly simplify your implementation — but you're expected to identify those through independent exploration. You will also be expected to learn basic HTML syntax, if you haven't done so already. There are a number of websites devoted to this topic. Have fun! Explore!

2 Overview

For this project, you will implement a simple command-line *web crawler*. The application will accept three arguments. The first will specify the web address from which the crawl should begin. The second will specify the maximum crawl depth. Finally, the third argument will specify the local directory used to store results from the crawl.

When the crawl begins, it will perform a basic parse of the web page specified at the command-line. The goal of the parse is to identify the (*i*) images used within the page, the (*ii*) other files linked from within the page[1], and the (*iii*) web pages linked from within the page. All three element sets will be saved to disk in the directory specified at the command-line. The third set of elements will be used as input to the next step in the page crawl; this process will be applied recursively up to the maximum specified crawl depth.

It is worth emphasizing that the term *"parse"* is being used loosely here. You are not required to do any sophisticated HTML processing; use the simplest solution that works. The Java String library

[*]This project is based on material prepared by Dr. Jason O. Hallstrom

[1]Excluding web pages and image files

provides more than adequate functionality for completing this task. You may also use Regular Expressions or the JSoup Library.

3 Requirements

Again, much of the design work is left to you. So while you must satisfy the requirements listed here, many of the details are left to you. This is the fun part of programming!

- You must follow good design practices throughout your implementation. In particular: (*i*) Program to interfaces wherever possible. (*ii*) Never use public fields. (*iii*) Follow Java naming conventions. (*iv*) Simplify your code by eliminating redundancy. (*v*) Catch and handle all checked exceptions.
- Your main application class must be named WebCrawler.
- Your application must include a WebPage class used to represent a web page. The class must include the following methods: crawl(), getImages(), getFiles(), and getWebPages(). The crawl() method is used to parse the web page represented by the target object. The remaining methods are used to retrieve the results of the parse. Hence, the get methods must only be invoked after crawl() has terminated.
- Your application must include classes for representing the images, files, and web pages linked from within a page. These classes must be named WebImage, WebFile, and WebPage, respectively. (The last class is discussed above.) Instances of these classes will be returned by the WebPage *getter* methods discussed above.
- WebImage, WebFile, and WebPage must each implement the WebElement interface. This interface must provide methods to enable a client component (i.e., a user of a WebElement instance) to save the corresponding element to disk.
- Your application must implement a *singleton* class, DownloadRepository. An instance of this class must be suitably initialized at system startup using an appropriate static initializer. This singleton must expose an interface that can be used by the crawler class (WebPage) to save a WebElement object to the repository.

4 Submission Instructions

This project is due before class on Feburary 28thth. Absolutely no late assignments will be accepted.

When your project is complete, archive your solution and use the handin command to submit your work. Please submit for your Eclipse project as a .zip archive usin the http://handin.cs.clemson.edu website. **You must also turn in a hard copy of all of your source materials in class.**

5 Grading

Your project will be graded based on the quality of your design and implementation, as well as the functionality of the application itself. You are expected to correctly handle all basic web pages, under the assumption that the web page content is correct. If you encounter an error during processing —due to improper HTML formatting, embedded scripts that are not handled by your parser, etc.— your application must handle the error gracefully — this means terminating only the affected branch in your crawl.

This is an intermediate course in software development. Your source materials should be properly documented. Your source must compile. Your application must not crash. A violation of any of these requirements will result in an automatic zero. **Test your application thoroughly.**

6 Collaboration

You may work independently or with one partner. You must *not* discuss the problem or the solution with classmates outside of your group. Keep this in mind before you choose to work independently.

Collaborating in any manner with peers outside of your group will be treated as academic misconduct.