UDACITY

PROJECT SPECIFICATION

Parallel Web Crawler

Section 1: Basic Functionality

CRITERIA	MEETS SPECIFICATIONS
Write code that passes all unit tests.	**All** unit tests must pass: `mvn test`

Write a crawler that successfully runs on real web pages (not just tests).	The following command should output valid results:
	`mvn package`
	`java -cp target/udacity-webcrawler-1.0.jar com.udacity.webcrawler.main.WebCrawlerMain src/main/config/sample_config.json`

Respect the configured timeout for the parallel crawler.	The crawler should stop downloading new URLs after the configured "timeoutSeconds" is reached. The easiest way to test this is to configure a large "maxDepth" (for example, 10) and a small "timeoutSeconds" (for example, 1). The crawler should stop running after about 1 or 2 seconds.

CRITERIA	MEETS SPECIFICATIONS
Use dynamic proxy to always record method invocation times for annotated methods.	Only record profile data for methods annotated with the `@Profiled` annotation. Methods with this annotation should be recorded even if they throw an exception.
Use dynamic proxy to return the correct values and handle exceptions correctly.	`ProfilingMethodInterceptor#invoke` should invoke the intercepted method and return the same value. Or, if the method invocation throws an exception, that exception must be correctly propagated back to the caller. Be sure not to accidentally throw an `UndeclaredThrowableException`! `Object#equals(Object)` should behave correctly.

Section 2: Parallelism & Synchronization

CRITERIA	MEETS SPECIFICATIONS

CRITERIA	MEETS SPECIFICATIONS
Fetch and process pages from multiple threads running in parallel.	The crawler must fetch and process pages from multiple threads concurrently. The crawler should be implemented using one or more of the following standard Java frameworks: • Executors • ForkJoinPool Different threads must actually run in parallel, meaning the solution is not allowed to use an executor with only one thread, and threads should not be synchronized such that they run serially.
Correctly synchronize shared data structures to detect and avoid revisiting already seen URLs.	The crawler should avoid visiting the same web page multiple times. The crawler should track which pages it has already visited so that it will not re-crawl such pages. An in-memory data structure should be used for this purpose. A URL is considered "visited" even if the HTTP response to that URL is an error, but revisits (if any) to that same URL should not count toward the final `urlsVisited` number.

CRITERIA	MEETS SPECIFICATIONS
Analyze and reason about concurrent programming scenarios.	Questions Q1 and Q2 in the `written-questions.txt` file should be satisfactorily answered.

Section 3: File I/O

CRITERIA	MEETS SPECIFICATIONS
Correctly parse and load JSON configuration using their Crawler.	The `ConfigurationLoader` class correctly parses the configuration using a JSON parsing library. The input should be read using the `try-with-resources` idiom, to prevent resource leaks.
Correctly write the result to file in the specified JSON format, which contains the number of pages visited and the top	The `CrawlResultWriter` class correctly uses a JSON library to write the crawl output to a file, or to standard output if no file path is provided. The output should be written using the `try-with-resources` idiom, to prevent resource leaks.

CRITERIA	MEETS SPECIFICATIONS
popular words.	
Program the profiler to correctly write its data to file or standard output.	When opening input and output streams and writers, make sure you close them. Also, be sure not to close the same stream twice.

Section 4: Code Design

CRITERIA	MEETS SPECIFICATIONS
Write a crawler that sorts and returns the correct word counts using *only* functional programming techniques such as the `Stream` API, lambdas, and method references.	The `sort` method in `WordCounts.java` should be implemented using only the `Stream` API, lambdas, and method references. Using the `WordCountComparator` is also allowed.

CRITERIA	MEETS SPECIFICATIONS
Make effective use of dependency injection and other design patterns.	Any parameters you add to the `ParallelWebCrawler` constructor should be injected using dependency injection. If it makes sense for your design, you should apply the builder pattern and/or factory pattern to construct subtasks.
Recognize design patterns and can evaluate their effectiveness.	Questions Q3 and Q4 in the `written-questions.txt` file should be satisfactorily answered.

Suggestions to Make Your Project Stand Out!

1. Respecting robots.txt files on crawled pages. You can read more about robots.txt here.
2. Managing memory use by limiting the growth of the popular word count and profiling data structures, while not compromising accuracy.
3. Throttling HTTP requests (*e.g.*, per domain) in order to not overwhelm crawled servers.