DEVELOPMENT ENVIRONMENTS
RAÚL BAÑÓ TORREGROSA

TASK 6

– EXTRACT PDF'S FROM A WEBSITE.

In this task, we will create a Java program using Maven and Java libraries to access a website and all the links associated with it, and download any PDFs it contains.

Key Libraries Used:

- **Selenium WebDriver:** A tool for automating web browsers. It is used in this project for navigating the website, finding links, and interacting with web elements.
- **Apache Commons IO:** This library simplifies file input/output operations. Specifically, it is used to download files from URLs.

Flow of the Code:

1. **WebDriver Setup:**
    - The first step is to configure the Chrome WebDriver. The `System.setProperty` method is used to set the path of the ChromeDriver executable.
    - The ChromeOptions are configured to run the browser in headless mode, meaning it operates without a graphical interface. This is ideal for automated tasks like web scraping.
    - A `WebDriver` instance is created using the `ChromeDriver` class, which allows interaction with the browser.

2. **Navigating the Base URL:**
    - The base URL `https://nachoiborraies.github.io/java/` is loaded into the browser.
    - The program waits for elements to load with an implicit wait of 10 seconds, allowing the page to fully load before attempting to interact with it.
3. **Finding Links:**
    - The program finds all anchor (`<a>`) tags on the page, which contain links to other pages or resources.
    - For each link, it checks if the `href` attribute contains a valid URL (specifically from the same domain `nachoiborraies.github.io`).
4. **Navigating to Subpages:**
    - For each valid link found, the program navigates to the subpage by calling `driver.get(subpageUrl)`.
    - The program then searches for links on the subpage that contain the substring `.pdf` in the `href` attribute, which are assumed to be links to PDF files.
5. **Validating and Downloading PDF Files:**
    - Each PDF link is validated using the `isValidURL` method. This method attempts to open a connection to the URL and checks if it is accessible by examining the response code. A valid URL should return a status code between 200 and 400.
    - If the URL is valid, the program proceeds to download the PDF file using the `downloadFile` method. The file is saved in the specified directory (`C:\\Users\\raulb\\Downloads\\`), and the file's name is extracted from the URL.
    - The program uses Apache Commons IO's `FileUtils.copyURLToFile` method to download and save the file.

6. **Navigating Back:**
    - After processing each subpage and downloading any PDFs, the program navigates back to the base page using `driver.navigate().back()`, and the process is repeated for the next link.

7. **Error Handling:**
    - The program handles exceptions using `try-catch` blocks. If any error occurs during the scraping or downloading process (such as a failed connection or an invalid URL), an error message is printed to the console.
8. **Program Termination:**
    - Finally, the WebDriver is properly shut down with `driver.quit()` to close the browser and free up resources.

Methods:

- **downloadFile(String fileURL, String filePath):**
  This method is responsible for downloading a file from a given URL and saving it to the specified local file path. It uses the `copyURLToFile` method from Apache Commons IO to handle the download.
- **isValidURL(String urlString):**
  This method checks if a given URL is valid and accessible by sending a HEAD request and analyzing the HTTP response code. A response code between 200 and 400 indicates that the URL is valid.

Conclusion:

This program successfully automates the extraction and downloading of PDF files from a website using Selenium WebDriver and Apache Commons IO. It efficiently navigates through the site, validates links, and downloads files, providing a powerful tool for web scraping tasks. However, further optimizations could include adding more robust error handling, logging, and implementing rate-limiting strategies to prevent server overload.