Framework Concepts - Log4j API

After executing different complex automation scripts, we need to know in plain English on how the scripts got executed and error details if any.

- The below are the two reasons for **logging**:
  - We can know what got executed in the code and how it got executed.
  - We can know where exactly in the code, Exceptions / errors occurred.
- Implement logging in Selenium Automation code using **System.out.println()** statements - Demonstrate here
  - Create a Project, Package and Class
  - Configure the Project with Selenium WebDriver
  - Create Selenium Automation code to visit Omayo blog, navigate to Compendium site, navigate back to Omayo blog, forward again to Compendium site and close the browser.
  - Write **System.out.println()** statements for logging
- Disadvantages of SOP logging
  - SOP's are heavy in nature and decrease the performance of programs if used more
  - We cannot turn off the logging when and then required
  - SOP's are for simple logging, and cannot be used for advanced logging.
- To resolve the above disadvantages, we have to use **Log4j logging**
- Similar to Java, Selenium and POI API's , Log4j is released into market as API by Apache guys
- Implementing Log4j in Selenium Automation - Demonstrate here
  - Step1: Download log4j jar file from https://logging.apache.org/log4j/1.2/download.html by selecting to download zip file.
  - Step2: Configure the downloaded log4j jar file in Java Project
  - Step3: Create log4j.properties file under 'src' and paste this code from here - view configuration properties here
    - log4j is the predefined name, hence dont change the name of the file
    - Copy this file only under 'src' folder
    - Understand the configuration properties one by one:
      - **log4j.logger.devpinoyLogger=DEBUG, dest1**
        - Find all the log levels here
        - dest1 will be given in another property
      - **log4j.appender.dest1=org.apache.log4j.RollingFileAppender**
        - RollingFileAppender delivers log events to the log file destination
      - **log4j.appender.dest1.maxFileSize=5000KB**
        - logs will be added to a single file until this 5000KB limit is reached
        - After 5000KB, another new log file will be created
      - **log4j.appender.dest1.maxBackupIndex= 3**
        - Maximum number of log files than can be backup-ed is 3
      - **log4j.appender.dest1.layout=org.apache.log4j.PatternLayout**
        - Used to specify the format in which the logs needs to be displayed (i.e. PatternLayout format)
      - **log4j.appender.dest1.layout.ConversionPattern= %d{dd MMM yyyy HH:mm:ss} %c %m%n**
        - ConversionPattern and its associated value is the format of the timestamp (%d{dd MMM yyyy HH:mm:ss}) + logger name(%c) + acutal log text(%m%n) in which the logs needs to be created/tracked.
      - **log4j.appender.dest1.File=E:\\Application.log**
        - Specifies the path at which the log files needs to be created.
      - **log4j.appender.dest1.Append=false**
        - Creates a new file on new execution, instead of appending the logs to the older log files.
  - Step4: Replace SOP's in Java Class, and use the above created 'log4j.properties' file to log with the help of **Logger.getLogger()**

- **Logger** is the predefined class of Log4j API and **getLogger()** is the static predefined method of Logger Class
- Use **debug()** predefined method of log4j api for logging the details in the program
- Based on the log level configured in log4j.properties file, the predefined commands like debug() will execute
    - If the configuration is set to 'info' level, then debug() commands in the execute code wont execute
    - If the configuration is set to 'off' level, then no logging can be performed even after specifying any predefined methods to log.