```
============
Logging
============

=> The process of storing application execution details to a file.

=> To understand runtime behaviour of the application we will use logging in the application.

=> With the help of logging we can identify root cause of the exception.

=> To implement logging, we have several logging frameworks

                1) log4j
                2) log4j2
                4) logback
                5) logstash

=======================
Logging Architecture
=======================

1) Logger

2) Layout

3) Appender

==================
What is Logger ?
==================

=> Logger is a predefined class which provides several methods to perform logging.

        Ex : trace (), debug(), info()....

=> We will use Logger methods like below

Ex:

void m1(){

        logger.info("m1() execution started....");

                // logic

        logger.info("m1() execution completed....");
}

Note: For every java class we will create one logger object.


=================
What is Layout ?
=================

=> Layout represents log msg pattern

Ex : <date time> <log-level>  <thread>  <class> <msg>

=> We have several types of layout

        Ex: SimpleLayout, PatternLayout..
```

```
====================
What is Appender ?
====================

=> Appender is used to write the log msg to destination

        Ex : ConsoleAppender, FileAppender, JDBCAppender


=======================
Spring Boot logging
=======================

=> Create springboot application with 'starter-web' dependency

=> Run the boot application from the start class and check console.

Note: We can see several log msgs on the console. That means springboot is using logging to provide
runtime behaviour of springboot.

Note: In springboot by default Pattern Layout and Console Appender will be used.


Note: To print log msgs in the log file then add below property in application.properties file

                logging.file.name=app.log

--------------------------------------------------------------------------------
@RestController
public class MsgRestController {

        private Logger logger = LoggerFactory.getLogger(MsgRestController.class);

        @GetMapping("/welcome")
        public String getWelcomeMsg() {

                logger.info("****** welcome () - execution started *******");

                String msg = "Welcome to Ashok IT..!!";

                logger.info("****** welcome () - execution completed *******");

                return msg;
        }
}
-----------------------------------------------------------------------------
@RestController
public class ContactRestController {

        private Logger logger = LoggerFactory.getLogger(ContactRestController.class);

        @GetMapping("/contact")
        public String getContactInfo() {

                logger.info("***** getContact() - started *****");

                String response = "Contact Us : + 91 - 9985396677";

                logger.info("***** getContact() - ended *****");

                return response;
        }

}
```

```
===============
Logging Levels
===============

=> Log msgs will be generated based on log level

=> Logging having several levels like below

        TRACE > DEBUG > INFO > WARN > ERROR

=> TRACE  is used to store every line execution details

        EX: logger.trace("msg");

=> DEBUG is used to store execution flow at low level

        Ex: logger.debug("msg") ;

=> INFO is used to store execution flow at high level

        Ex: logger.info("msg");

=> WARN is used to store warnings in code execution flow

        Ex: logger.warn("msg");

=> ERROR is used to store execptions occured in code execution flow

        Ex: logger.error("msg");


############ Note: In springboot, default log level is INFO ###########

=> We can change log level in the application using below property

        logging.level.root=ERROR

Note: When we set log level, log msgs will be printed from that level to all higher levels also.


======================
Logging with Rolling
======================

=> If we use single log file to store log msgs then after few days log file size will become very
very big.

=> If file size is keep on increasing then it will become difficult to open/read log file data.

Note: To avoid this problem we will use rolling mechanism.

=> Rolling we can implement in 2 ways

                1) Size Based Rolling

                2) Time Based Rolling

=> Size Based Rolling is used to create new log file once old log is reached to given limit.

                Ex: 1 GB

=> Time based rolling is used to create every day new log file.
```

=> To configure rolling, we will use RollingFileAppender.

=> We can configure rolling in 2 ways

1) application.properties

2) logback.xml

========================
What is logback.xml ?
========================

=> logback.xml is used to customize logging in our application.

=> In logback.xml we will configure below components

1) Rolling File Appender with policy

2) Log Msg Pattern

3) Log Level

=> We will keep logback.xml under "src/main/resources" folder

------------------------------ logback.xml ---

```
<configuration>

	<appender name="RollingFile"
		class="ch.qos.logback.core.rolling.RollingFileAppender">
		<file>ashokit.log</file>
		<encoder>
			<pattern>%d [%thread] %-5level %-50logger{40} - %msg%n</pattern>
		</encoder>
		<rollingPolicy
			class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
			<fileNamePattern>ashokit-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
			<maxFileSize>1GB</maxFileSize>
			<maxHistory>30</maxHistory>
		</rollingPolicy>
	</appender>
	<root level="INFO">
		<appender-ref ref="RollingFile" />
	</root>

</configuration>
```

================
Log Monitoring
================

=> It is the process of reading or observing log msgs available in log file.

Note: In-Realtime log files will be stored in linux machines only.

=> To get log msgs from log file we have several tools.

1) Putty

2) WinScp

3) ELK (Elastic Search + Log stash + Kibana)

4) Splunk (Commercial S/w)

Note: Putty and WinScp softwares are used to connect with Linux VM.

=> Putty Is cli based

=> WinSCP is GUI based

=> In Realtime, operations team will share log server details like below

Log Server IP : 192.156.87.7

username : loguser

Password: loguser@ashokit

Log Files Path : /home/ubuntu/logs

=> ELK and Splunk softwares provides UI to monitor logs.

=> ELK is open source

=> Splunk is licensed

===
What is the difference between SLF4J and Log4J ?
===

=> SLF4J stands for Simple Logging Facade for Java

=> SLF4J serves as a simple facade or abstraction for various logging frameworks (e.g. java.util.logging, logback, log4j) allowing the end user to plug in the desired logging framework at deployment time.

```
        Java application -----------> Logging Frameworks ( Tightly coupling )
```

Note: If we use logging framework directley then our classes will become tightly coupled with logging framework. If we want to change from one framework to another framework then we need to modify all our classes.

```
        Java application -----------> SLF4J (loosely coupling)
```

Note: If we use SLF4J then our application will be loosely coupled with logging frameworks. We can configure any logging framework dependency to perform logging. SLF4J will pickup available logging framework from build path.

=========
Summary
=========

1) What is Logging

2) Why Logging

```
  3) Logging Architecture

  4) Log Levels

  5) Logging in SpringBoot

  6) Rolling Policies

  7) SL4J Vs Log4J

  8) Log Monitoring

                - Putty
                - WinScp
                - ELK
                - SPlunk
```