BINAR

Java Logging

Gold - Chapter 6 - Topic 3

BINAR

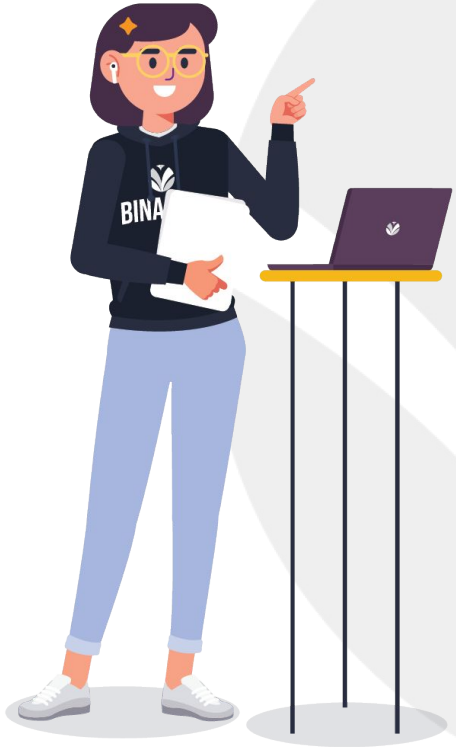
Selamat datang di Chapter 6 Topic 3 online course Back End Java dari Binar Academy!

Ada yang masuk ke topik 3, nih~

Siapa lagi dong kalau bukan Binarian yang rajin banget belajar dan semangatnya mengalahi sinaran ultraspeng □

Pada topik sebelumnya, kita udah belajar tentang Spring Security Part 2. Di topik ketiga ini, kita bakal mengelaborasi tentang **Java Logging,** mulai dari konsepnya hingga logging framework yang bisa kita gunakan.

Yuk, langsung aja kita kepoin~



Dari sesi ini, kita bakal bahas hal-hal berikut:

- Konsep dasar logging
- Logging framework yang sudah disediakan oleh Spring dan logging framework lainnya
- Logging level
- Implementasi logging
- Cara membaca log
- Tempat menyimpan log
- Logging rolling file rules

Binarian, kamu tahu nggak kalau logging biasanya menjadi tempat menyimpan info penting?

Walaupun begitu, log itu nggak boleh dipakai untuk informasi sensitif kayak nomor NIK.

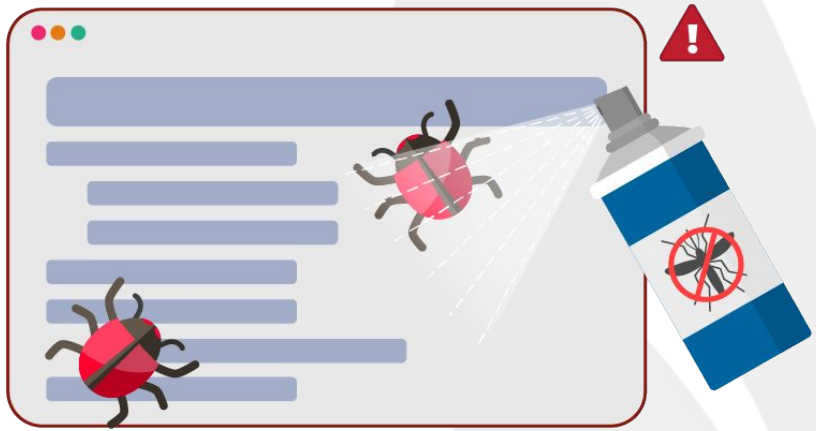
Kalau kamu penasaran sama alasanya, yuk kita coba bahas **Konsep Dasar Logging**

Apa itu Logging?

Logging merupakan tools untuk membantu dalam memahami dan melakukan debugging behavior yang terjadi saat program berjalan.

Log biasanya dipakai untuk meng-capture dan menyimpan informasi penting dari proses ataupun event tertentu, sehingga bisa dianalisis kapan pun.

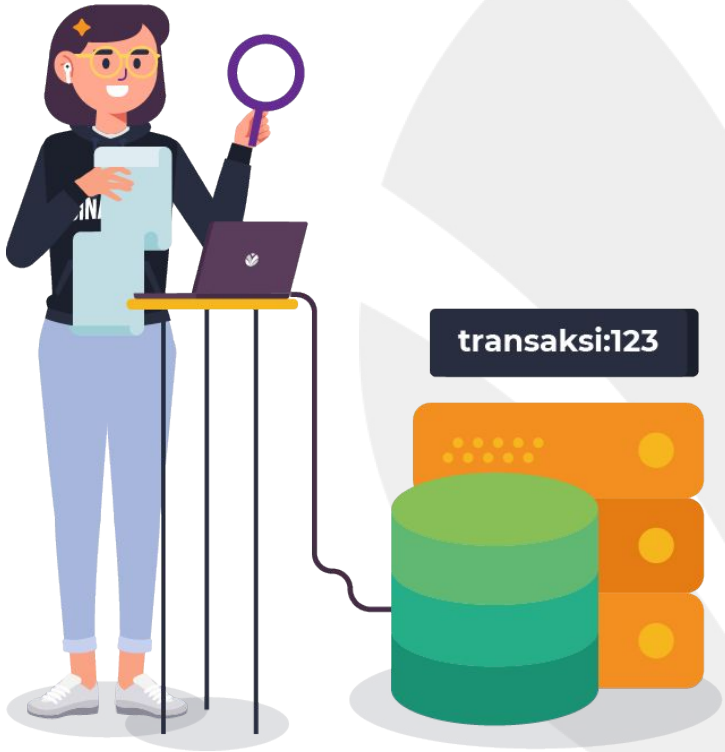
Event tersebut misalnya kayak pas terjadi exception dan error. Informasi penting yang dicantumkan pun biasanya berupa **data yang bisa bantu buat tracing error**.



“Coba kasih contoh, dong!”

Sipp deh! Misalkan ada transaksi A yang punya id transaksi:123.

Untuk mempermudah melakukan tracing terhadap transaksi A, kita bisa menyematkan informasi transaksi id 123 pada log untuk mempermudah pencarian log transaksinya.



Kenapa nggak semua informasi disematkan di dalam log supaya bisa gampang melakukan tracing error?

Sayangnya, nggak bisa sesederhana itu

Kalau meletakkan seluruh informasi di dalam log, **hal ini bakal mengakibatkan penurunan performance aplikasi.**

Contohnya kayak penggunaan memory yang lebih banyak dan waktu pemrosesan yang lebih lama.

Selain itu, nggak semua informasi bisa disematkan di dalam sebuah log karena **informasi sensitive dilarang banget untuk diletakkan di dalam log**.

Informasi sensitif itu bisa berupa nomor NIK atau password, sob.

Seluruh logging framework yang ada di Java mengimplementasi SL4J, lho gengs~

SL4J atau Simple Logging Facade for Java merupakan logging API yang jadi abstraksi dari segala logging framework pada Java.

Jadi nggak heran kalau seluruh logging framework yang ada di Java mengimplementasi SL4J.

Untuk memahami suatu logging, kita harus tahu beberapa level dari logging dan hierarki logging dulu.

Di topic ini, kita cuma bakal bahas tentang penggunaan logging framework pakai logback yang disediakan pada Spring Boot aja.

Pasti kamu penasaran, kan?

Kenapa kalau makan mie instan yang beli di burjo itu lebih enak daripada pas kita bikin sendiri?

Yap, jawabannya adalah karena udah disediakan.

Begitu pula sama Logging, ternyata udah ada Logging framework yang disediakan oleh Spring. Yuk kita intip!

Logging yang udah disediakan oleh Spring adalah Logback~

Logback merupakan framework yang dipakai di Spring Boot.

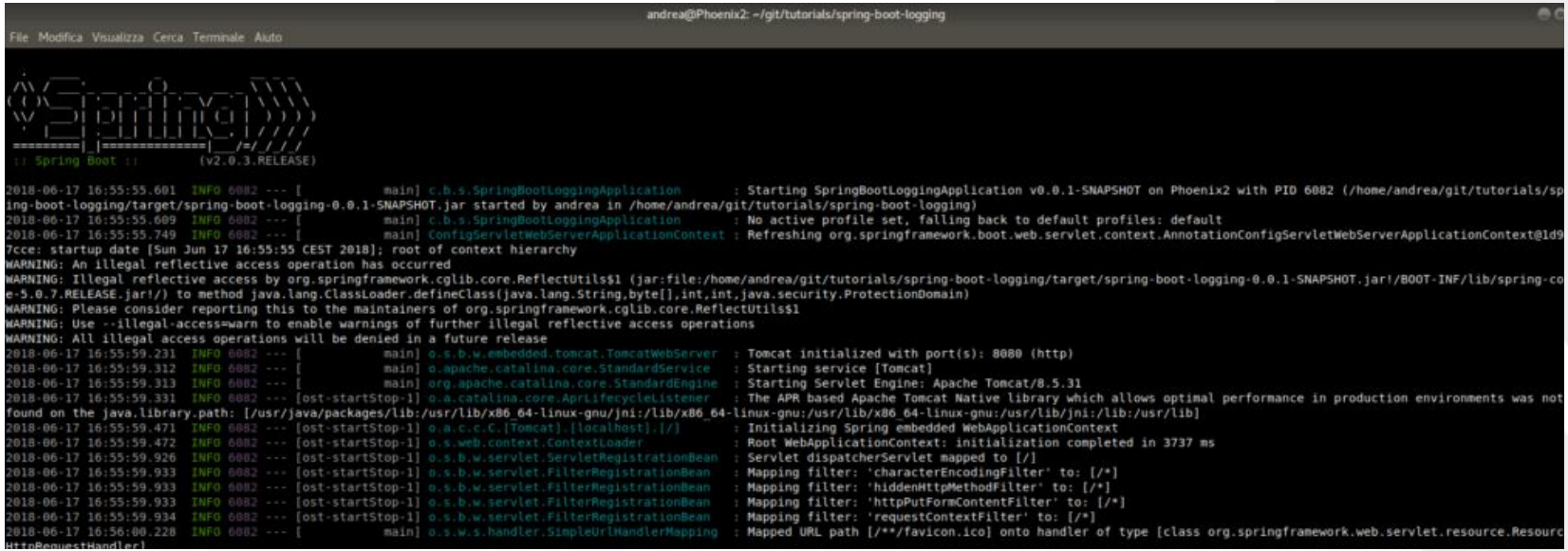
Tanpa melakukan setup, logback udah bisa dipakai tanpa harus melakukan konfigurasi terlebih dahulu.

Ajib nggak sih? Selanjutnya **untuk mengatur konfigurasi logging, kita bisa pakai file xml.**

Selain itu, Logback juga merupakan project penerus dari log4J, lho.

Jadi, masih termasuk salah satu framework logging Java~

Nah, berikut adalah tampilan console dari Logback pada Spring



Seperti yang udah dijelasin sebelumnya, Logback merupakan Logging yang udah tersedia.

Usut punya usut nih, ternyata ada Logging Framework lainnya yang bisa kita pakai.

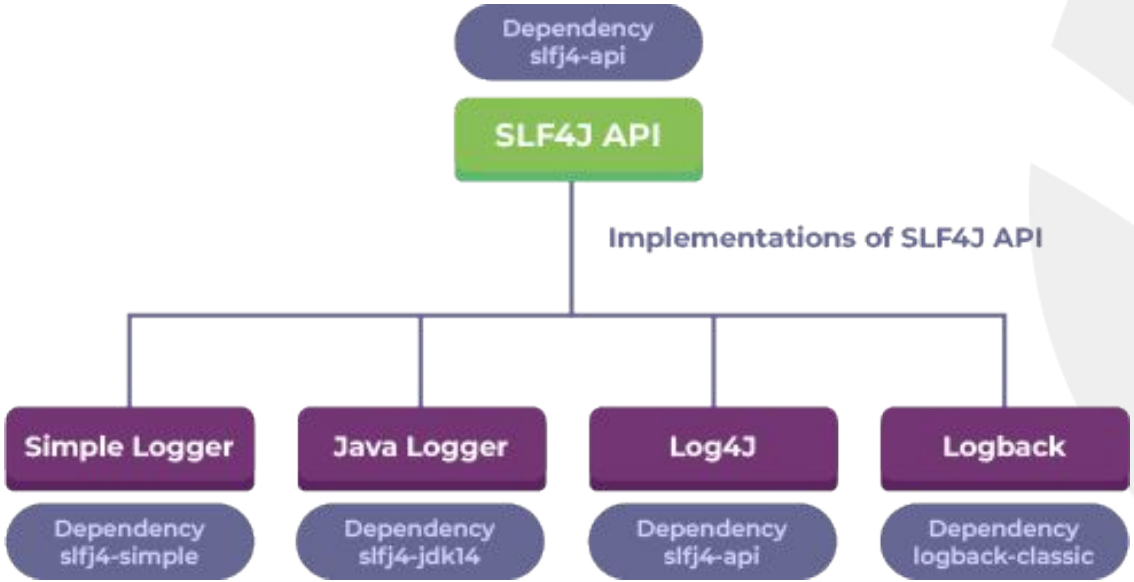
Satu titik dua koma. Nggak usah panik, yuk kita bahas materinya!

Kenapa harus pakai logging framework lain padahal udah disediakan SL4J?

Yepp, karena SL4J ini merupakan abstraksi dari seluruh logging framework pada Java.

Sayangnya, **SL4J ternyata nggak cukup baik untuk menangani kasus-kasus tertentu**. Misalnya tuh kayak proses yang sifatnya asynchronous. Jadi, kita perlu pakai logging framework lain deh~

Logging framework lainnya yaitu log4J, Java.util.logging, dan simple logger~



Nggak cuma game aja yang ada levelnya, Logging juga sama, lho!

Penasaran sama materinya? Yuk yuk kita masuk ke materi Logging level.

Berangkatt~

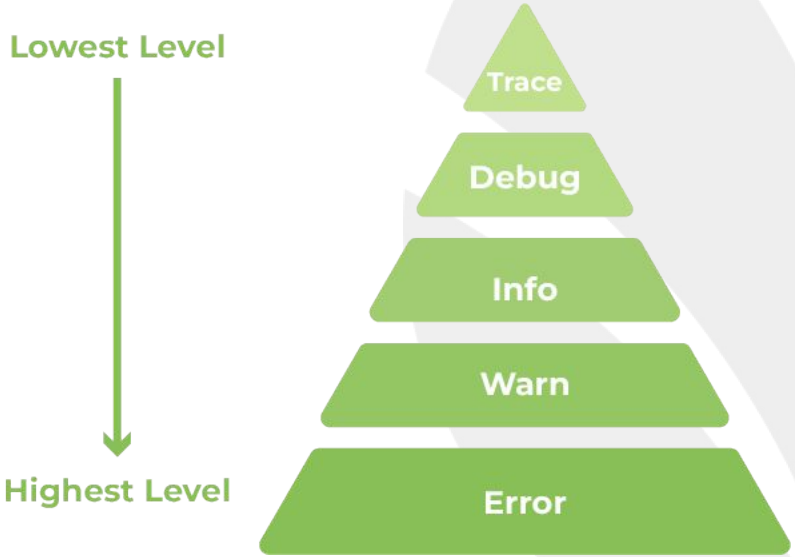
Buat memahami logging, kita harus paham hierarki logging dan level logging-nya dulu~

Yap, bener banget. Untuk menjadi ahli dari suatu game, kita harus menyelesaikan tantangannya dulu supaya bisa naik level. Begitupun dengan logging.

Berikut adalah urutan level logging dari yang terkecil sampai ke yang terbesar:

ERROR - WARN - INFO - DEBUG - TRACE

Logging ini bakal menentukan log mana yang bakal dimunculkan.



Berikut adalah basic rule dari logging. Perhatikan baik-baik, yaaa~

P adalah request untuk memunculkan log, sedangkan q merupakan effective level dari log.

Effective log adalah log yang diperbolehkan untuk muncul di aplikasi, jadi log bakal muncul kalau memenuhi p >= q.

level of request *p*	effective level *q*					
	TRACE	DEBUG	INFO	WARN	ERROR	OFF
TRACE	YES	NO	NO	NO	NO	NO
DEBUG	YES	YES	NO	NO	NO	NO
INFO	YES	YES	YES	NO	NO	NO
WARN	YES	YES	YES	YES	NO	NO
ERROR	YES	YES	YES	YES	YES	NO

“Terus penggunaan dari log levelnya gimana?”

Penggunaan log level pada masing-masing level bisa kamu simak di bawah ini, yaa~

- **TRACE**, dipakai untuk melakukan tracing pada fase development.
- **DEBUG**, dipakai untuk menandakan method mana yang sedang berjalan, dengan argumen apa, dan apa hasil dari method-nya.

- **INFO**, dipakai untuk memberikan informasi start-stop aplikasi atau proses, memulai proses penyimpanan data, dan menampilkan query. Biasanya berisi informasi yang berkaitan dengan business process.
- **WARN**, dipakai untuk memberikan informasi yang bersifat peringatan dan konsekuensinya.

- **ERROR**, dipakai untuk memberikan informasi berupa error ataupun exception.

  Misalnya exception di block try catch, karena file not found atau salah memasukan data. Biasanya level ini dipakai untuk logging exception ketika menjalankan suatu proses bisnis atau logic.

  Idealnya adalah pas kita melakukan try-catch terhadap suatu exception.

Karena dari tadi kita udah bahas konsepnya, sekarang saatnya kita melakukan **Implementasi Logging**.

Siap-siap nih kita mau praktek~

“Emang implementasinya gimana, sih?”

Untuk menggunakan logback pada Spring Boot, kita pasti bisa menggunakannya secara langsung karena logback udah disediakan pada masing-masing Spring boot Starter.

Buat contohnya bisa kamu cek di samping, ya~

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Kalu kita mau pakai logging framework selain logback pada Spring Boot, kita bisa menambahkan dependency berikut.

Di contoh ini, kita pakai framework log4J, ya.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>
```

Terus…

Untuk meng-customize konfigurasi log, kita bisa pakai logback.xml.

Logback.xml berisi pengaturan format yang bakal ditampilkan di console atau file di mana file log bakal disimpan.

File log bakal dibikin jadi file baru. Berdasarkan apa? Yaitu disesuaikan dengan apa file-nya dan dengan rule apa effective log yang ditampilkan.

BINAR

Berikut adalah contoh file logback.xml yang mudah~

```
<configuration>
    <include resource="/org/springframework/boot/logging/logback/base.xml"/>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n
            </pattern>
        </encoder>
    </appender>
    <root level="error">
        <appender-ref ref="STDOUT"/>
    </root>
    <springProfile name="logback-test1">
        <logger name="com.binar.testloglevel" level="info"/>
    </springProfile>
    <springProfile name="logback-test2">
        <logger name="com.binar.testloglevel" level="trace"/>
    </springProfile>
</configuration>
```

Untuk menambahkan log pada suatu class, kita perlu mendeklarasi logger yang mau dipakai di class tersebut.

Berikut adalah contohnyaaa~

```
private static final Logger LOG = LoggerFactory.getLogger(TestLogLevelController.class);
```

Terus, untuk meletakkan message pada log tersebut, kita bisa pakai method di samping, gengs.

Sederhana banget, kannn?

```
LOG.trace(“Any message");
LOG.debug(" Any message ");
LOG.info("Any message”);
LOG.error("Any message ");
```

Bentar-bentar~

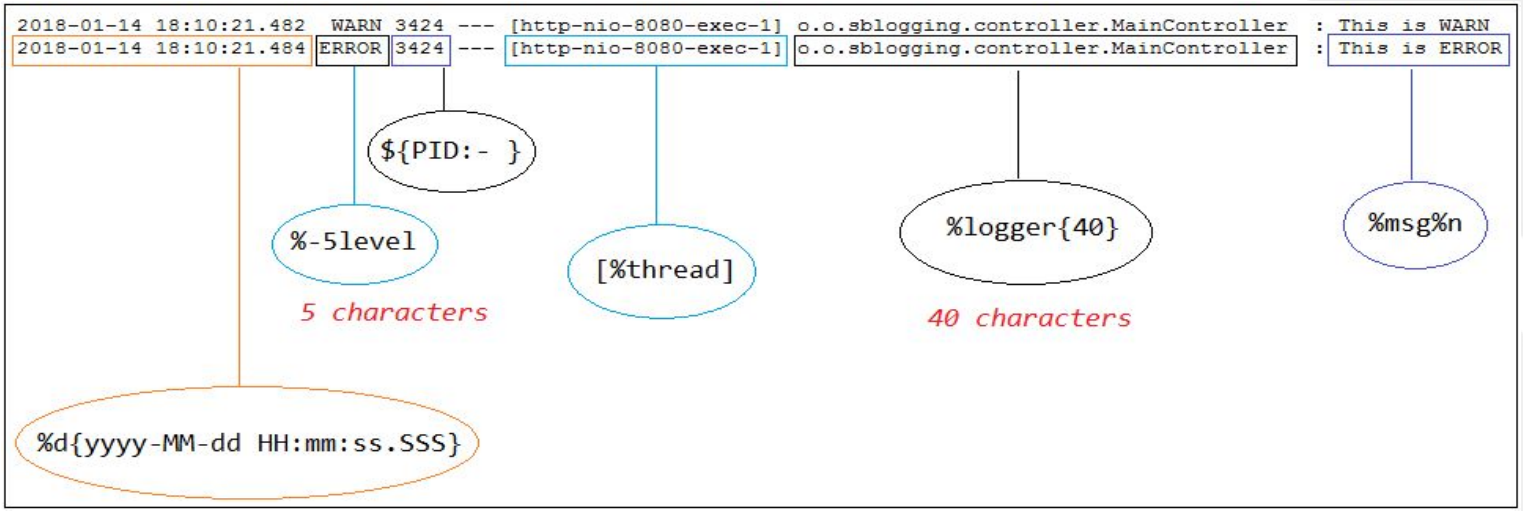
Berhubung kamu udah tahu gimana konsep dan cara implementasinya. Biar makin komplit, gimana kalau kita belajar Cara baca Log-nya?

Langsung aja cekidottt!

Begini cara baca Log-nya, bestie~

Secara default, format log yang ditampilkan adalah sebagai berikut:

Timestamp > logging level > PPID > thread > nama class logger > message



Setelah itu…

Untuk mengatur pattern dari log yang digunakan, kita bisa pakai konfigurasi secara custom dengan **logback.xml.**

Gimanaaaa?

Ternyata gampang banget, kan?!

Wuihh.. ada yang makin jago, nih.

Mumpung masih semangat, next kita bakal bahas materi tentang Tempat Menyimpan Log.

Who’s excited?

“Jadi, Log itu disimpan di mana, ya?”

Okey. Supaya log bisa dianalisis sewaktu-waktu, log biasanya disimpan dalam bentuk file tertentu. **File log tersebut bisa kita buka pakai text editor.**

Kalau kamu tanya gimana file akan disimpan, jawabannya yaitu bisa dikonfigurasi pakai **logback.xml.**

Sederhana sekaleee~

Berikut adalah contoh dari konfigurasi yang lebih kompleks!

Pada konfigurasi ini, kita mendefinisikan lokasi file log yang dibikin, yaitu di **c:/temp/logs pakai tag property dengan attribute name dan value.**

Di dalam file ini, kita juga bisa mendefinisikan nama file tersebut, gengs.

```
<configuration>
    <property name="LOG_ROOT" value="c:/temp/logs" />
    <property name="LOG_FILE_NAME" value="application" />

    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
    </appender>

    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${LOG_ROOT}/${LOG_FILE_NAME}.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
            <fileNamePattern>${LOG_ROOT}/${LOG_FILE_NAME}-%d{yyyy-MM-dd}.%i.log.gz</fileNamePattern>
            <!-- each archived file's size will be max 10MB -->
            <maxFileSize>10MB</maxFileSize>
            <!-- 30 days to keep -->
            <maxHistory>30</maxHistory>
            <!-- total size of all archive files, if total size > 100GB, it will delete old archived
file -->
            <totalSizeCap>100GB</totalSizeCap>
        </rollingPolicy>
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <logger name="com.howtodoinjava.app" level="INFO" additivity="false">
        <appender-ref ref="FILE"/>
    </logger>

    <root level="ERROR">
        <appender-ref ref="STDOUT" />
        <appender-ref ref="FILE" />
    </root>
</configuration>
```

Gimana, gimana? setuju nggak kalau tempat menyimpan Log itu simple?

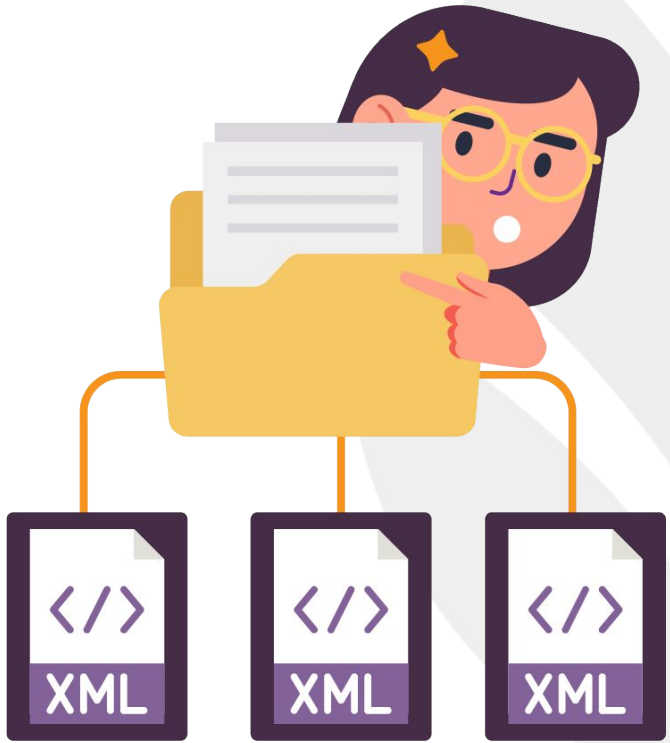
Udah nggak sabar bahas materi penutup di topic ini?

Santai ajaaa, abis ini kita bakal bahas tentang **Logging Rolling File Rules** sebagai penutup topik.

Nggak mungkin kan, sebuah file log dijadikan satu? Iya karena file log tersebut nantinya bakal jadi besar

Oleh karena itu, kita bisa melakukan konfigurasi untuk mem-backup file log tersebut yang udah dibagi-bagi dengan rules tertentu.

Kita bisa mengaturnya pakai file **logback.xml** dan mengkonfigurasikannya pakai **RollingFileAppender**.



Implementasi secara lengkap tentang rules ini bakal dijelaskan dalam dokumentasi dari logback appenders berikut.

Appenders

Simak baik-baik ya, bestie~

Latihan lagi, latihan terus~

Seperti biasa, sebelum kita move on ke topic selanjutnya ada latihan dulu:

Implementasikan logging pada code yang sudah kamu buat di project sebelumnya

Latihan ini dilakukan di kelas dan silahkan diskusikan hasil jawaban kamu dengan teman sekelas dan fasilitator. Selamat mencoba, yaa~

Refleksi Belajar

BINAR

Yesh~ itu dia pembahasan tentang Logging pada Java.

Tadi juga sempat di-mention kalau terdapat logging framework selain Logback yaitu log4J, Java.util.logging, dan Simple Logger.

Boleh banget lho kalo kamu menggunakan logging framework tersebut biar pengetahuan dan keterampilan kamu mengenai Logging di Java makin oke lagi.

Nah, selesai sudah pembahasan kita di Chapter 6 Topic 3 ini.

Selanjutnya, kita bakal bahas tentang Docker, yaitu salah satu platform untuk membuat container.

Penasaran container itu kayak gimana? Yuk langsung ke topik selanjutnya~