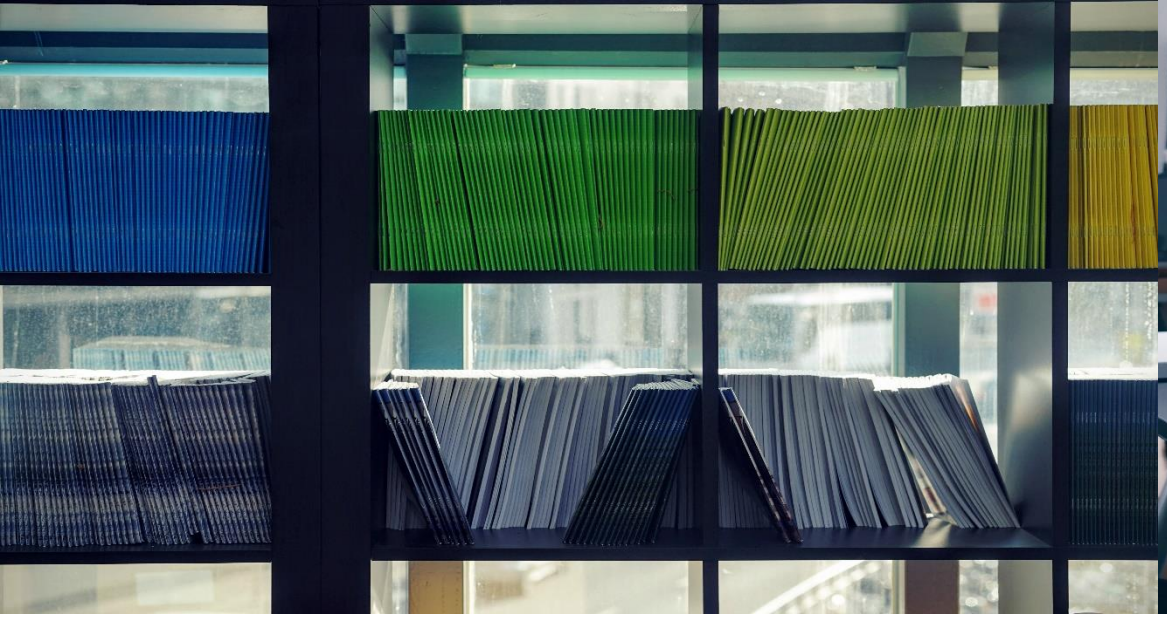
Logging & Lombok

김기정 (bangry313@gmail.com)



1. Logging

1.1 Logging 개요 (1/3)

✓ 로깅(Logging) 이란?

- 운영 중인 애플리케이션에 문제가 발생했을 경우, 문제의 원인을 파악하기 위해 당시의 정보가 필요하다.
- 이런 정보를 얻기 위해서 예외가 발생했거나, 중요 기능이 실행되는 부분에서 적절한 로그를 남겨야 한다.
- 스트림 API를 활용한 System.out.println(" 로그메시지 "); 사용은 애플리케이션 개발 완료 후 반드시 삭제해야 하며, 애플리케이션의 성능을 떨어트리고, 일시적인로그만을 기록할 수 있는 단점이 있다 (운영 서버에서는 절대 사용하면 안된다)
- 일반적으로 자바 기반의 Logging 라이브러리를 사용해서 로그를 관리한다.

✓ 로깅 라이브러리 종류

- log4j
  - Apache 재단에서 개발하여 2015년을 끝으로 개발이 중단된 라이브러리로 기존에 표준으로 가장 많이 사용되던 오픈 소스 로깅 라이브러리이다.
- logback
  - log4j를 구현한 개발자가 개발한 라이브러리로 log4j 보다 약 10배 빠른 성능, 메모리 효율성, 필터링 옵션을 제공하며 자동 리로드도 가능하다.
  - Spring Boot에서는 spring-boot-starter-logging 안에 기본적으로 포함되어 있어서 따로 dependency를 추가하지 않고 바로 사용 가능하다.
- log4j2
  - 가장 최근에 등장한 라이브러리로 logback과 동일하게 자동 리로드 기능과 필터링 기능을 제공하며, 멀티 스레드 환경에서 비동기 로깅의 경우 대용량 처리를 자랑한다.
- Slf4j (Simple Logging Facade for Java)
  - 다양한 로그 라이브러리들을 통합해서, 간단하고 일관된 방식으로 사용할 수 있도록 표준 인터페이스를 제공한다.
  - 사용하는 로깅 라이브러리(Slf4j 구현체) 종류에 상관없이 로깅을 사용할 수 있다.

1.1 Logging 개요 (2/3)

✓ Logback 라이브러리 특징

- 로그 메시지에 대해 다양한 로그레벨을 설정할 수 있다.
  - TRACE < DEBUG < INFO < WARN < ERROR
- 개발 서버와 운영 서버에 대해 각각 다른 로그레벨을 설정하여 로그를 확인할 수 있다.
- 다양한 로그 메시지 출력 대상(콘솔, 파일, 메일, DB)을 설정할 수 있다.
- 로그 메시지를 파일에 저장 시 최대 사이즈를 설정해서 파일을 분할하여 저장할 수 있고, 압축해서 백업할 수 있다.
- 로그파일 보관 기간을 설정할 수 있다.
- Spring Boot에는 기본적으로 포함되어 있어서 따로 dependency를 추가하지 않고 바로 사용 가능하다.

✓ Logback 사용을 위한 설정

- 일반적으로 Classpath에 존재하는 Logback 설정 파일을 참조
  - Java Application, Spring 프레임워크의 경우 logback.xml 파일 참조
  - Spring Boot의 경우 logback-spring.xml 파일 참조

1.1 Logging 개요 (3/3)

✓ **로그 레벨**

- 코드 수정 없이 로그레벨 설정 만으로 로그 출력을 조절할 수 있다.
- 낮은 레벨에서 높은 레벨로의 5단계 로그 레벨에 따라 로그 메시지의 종류가 달라진다.
  - 레벨은 출력 범위를 나타내고, 설정한 레벨 이상의 로그만을 화면에 출력한다.
- TRACE < DEBUG < INFO < WARN < ERROR
  - 예) INFO 레벨로 설정하면 INFO 레벨 이상의 로그(INFO, WARN, ERROR)만 화면에 출력된다.

✓ **TRACE**

- 추적 레벨은 DEBUG보다 좀 더 상세한 정보를 표시

✓ **DEBUG**

- 애플리케이션 디버깅을 위한 정보를 표시 (주로 개발 서버에서 사용)

✓ **INFO**

- 상태 변경과 같은 정보성 로그를 표시 (주로 운영 서버에서 사용)

✓ **WARN**

- 시스템 에러의 원인이 될 수 있는 경고성 메시지를 표시 (처리 가능한 사항)

✓ **ERROR**

- 요청을 처리하는 중 시스템적으로 심각한 문제가 발생하여 운영이 불가한 경우 메시지 표시

1.2 Logging 구성 요소 (1/4)

✓ Logger

- 로깅을 수행하는 핵심 구성요소로 레벨 속성을 통해서 출력할 로그의 레벨을 조절할 수 있다.
- 출력하고자 하는 로그 메시지를 Appender에게 전달한다.

✓ Appender

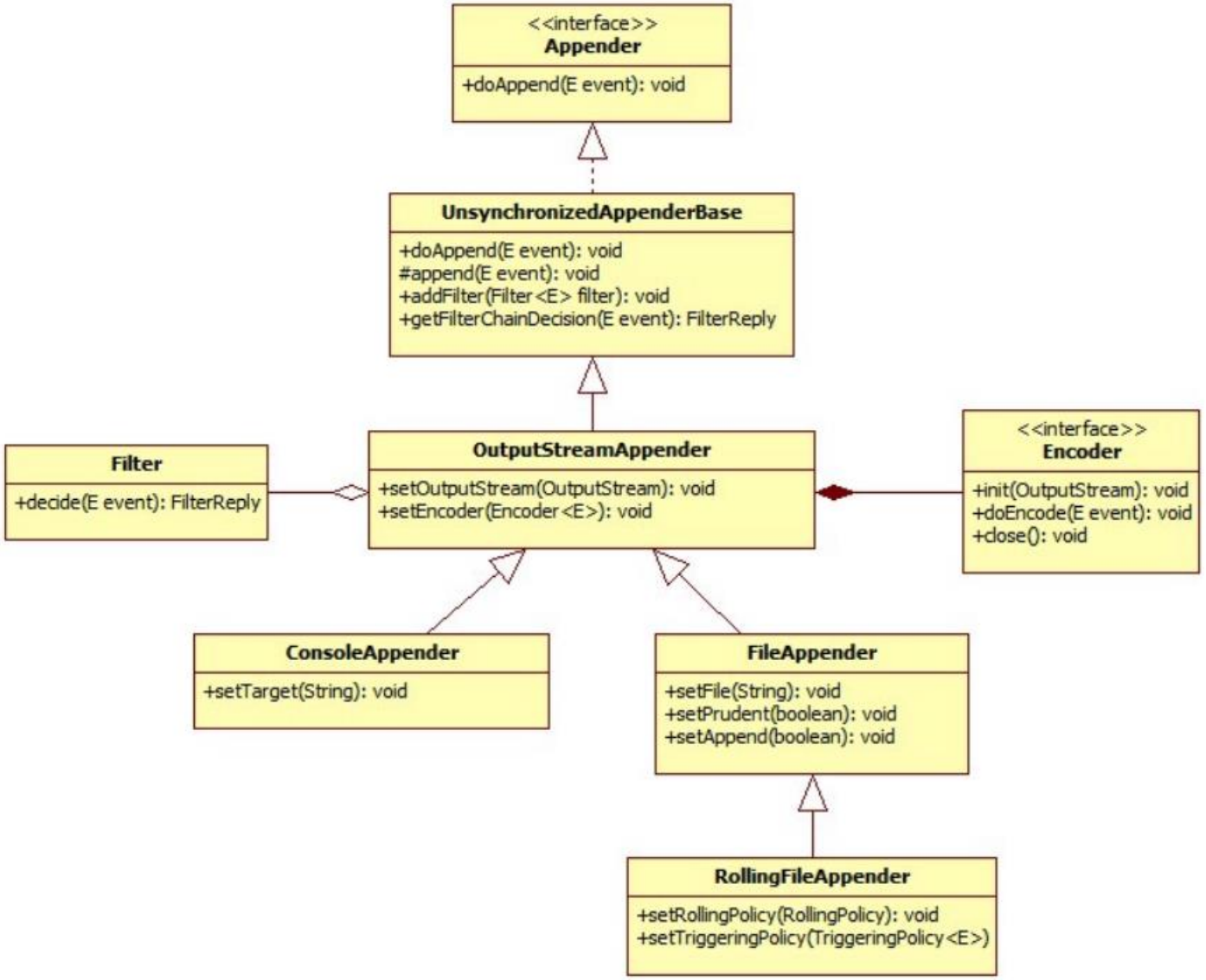
- Logger로부터 전달받은 로그 메시지를 출력할 대상을 결정하는 구성요소이다.
- Logger 1개에 여러 개의 Appender를 설정할 수 있다.
- Logger 구현체로 ConsoleAppender, FileAppender, DailyRollingFileAppender, SMTPAppender, DBAppender 등을 제공한다.

✓ Encoder

- Appender에 포함되어 사용자가 지정한 형식으로 표현 될 로그메시지를 변환하는 역할을 담당하는 구성요소이다.

1.2 Logging 구성 요소 (2/4)

✓ Logback 구조



1.2 Logging 구성 요소 (3/4)

✓ Encoder 로그 메시지 출력 형식 설정

형식	설명
%p	trace, debug, info, warn, error 등의 로그레벨(priority) 설정
%m	로그메시지 설정
%M	로깅이 발생한 메소드 이름 설정
%d	로그 이벤트가 발생한 시간을 설정 포맷은 %d{HH:mm:ss, SSS}, %d{yyyy MMM dd HH:mm:ss, SSS} 같은 형식으로 설정
%thread	로그 이벤트가 발생된 쓰레드 이름 설정
%n	줄바꿈 설정
%C	클래스명 설정 예) 클래스가 org.apache.xyz.SomeClass 처럼 되어있다면 %C{2}는 xyz.SomeClass 출력
%F	로깅이 발생한 프로그램 파일명 설정
%L	로깅이 발생한 라인 설정
%r	애플리케이션 실행 후 로깅이 발생한 시점 까지의 시간(milliseconds) 설정

1.2 Logging 구성 요소 (4/4)

✓ Encoder 로그 파일명 형식 설정

형식	설명
'.'yyyy-MM	매달 첫 번째 날에 로그파일을 변경
'.'yyyy-ww	매주의 시작 시 로그파일을 변경
'.'yyyy-MM-dd	매일 자정에 로그파일을 변경
'.'yyyy-MM-dd-a	자정과 정오에 로그파일을 변경
'.'yyyy-MM-dd-HH	매 시간의 시작마다 로그파일을 변경
'.'yyyy-MM-dd-HH-mm	매분 마다 로그파일을 변경

1.3 Logback 설정하기

✓ slf4j 및 logback 라이브러리 의존성 추가

build.gradle

```
// https://mvnrepository.com/artifact/org.slf4j/slf4j-api
implementation 'org.slf4j:slf4j-api:2.0.6'
// https://mvnrepository.com/artifact/ch.qos.logback/logback-classic
implementation 'ch.qos.logback:logback-classic:1.4.5'
testImplementation 'ch.qos.logback:logback-classic:1.4.5'
```

1.3 Logback 설정하기

✓ src/main/resources/logback.xml

logback.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="true" scanPeriod="30 seconds">
<property name="LOGS_ABSOLUTE_PATH" value="./log"/> <!-- 로그 파일 위치 -->
<property name="LOG_PATTERN" value="[%d{yyyy-MM-dd HH:mm:ss}][%thread][%-5level]\(%F:%L\) : %msg%n"/> <!-- 로그 출력 패턴 -->

<!-- 콘솔 출력 -->
<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
<encoder>
<pattern>${LOG_PATTERN}</pattern> <!-- 로그 출력 패턴 -->
</encoder>
</appender>

<!-- 파일 출력 -->
<appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender"> <!-- RollingFileAppender 사용 -->
<file>${LOGS_ABSOLUTE_PATH}/application.log</file> <!-- 파일 경로 및 파일 이름 -->
<encoder>
<pattern>${LOG_PATTERN}</pattern> <!-- 로그 출력 패턴 -->
</encoder>
<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy"> <!-- 시간으로 파일 분할 정책 -->
<fileNamePattern>${LOGS_ABSOLUTE_PATH}/application.%d{yyyy-MM-dd}.%i.log</fileNamePattern> <!-- 분할 파일 이름 패턴 -->
<timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
<maxFileSize>100MB</maxFileSize> <!-- 분할 파일 크기 기준 -->
</timeBasedFileNamingAndTriggeringPolicy>
<maxHistory>100</maxHistory> <!-- 분할된 파일의 최대 개수 -->
</rollingPolicy>
</appender>

<!-- 로그 출력 기준 -->
<root level="DEBUG">
<appender-ref ref="CONSOLE"/>
<appender-ref ref="FILE"/>
</root>
</configuration>
```

1.4 Logback 사용하기

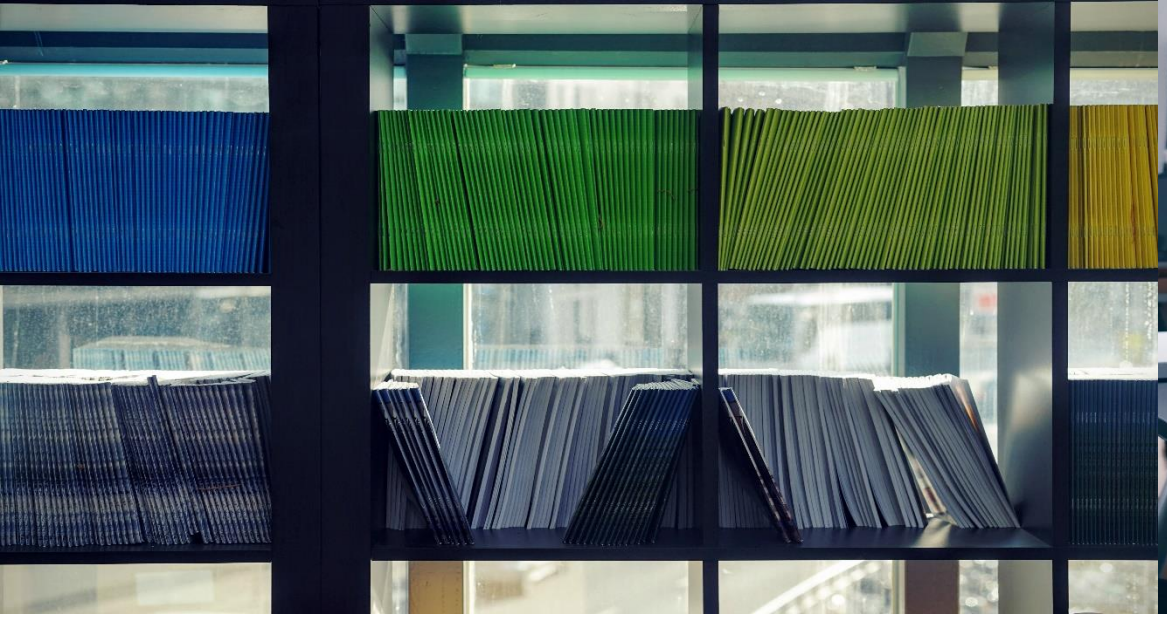
✓ src/main/java/logging_basic/LogTestExample.java

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class LoggTestExample {

    private static final Logger logger =  LoggerFactory.getLogger(LoggTestExample.class);

    public static void main(String[] args) {
        logger.trace("trace : {}", "로그 메시지 테스트입니다.");
        logger.debug("debug : {}", "로그 메시지 테스트입니다.");
        logger.info("info : {}", "로그 메시지 테스트입니다.");
        logger.warn("warn : {}", "로그 메시지 테스트입니다.");
        logger.error("error : {}", "로그 메시지 테스트입니다.");
    }
}
```



2. Lombok 라이브러리

2.1 Lombok 소개 (1/2)

✓ Lombok 라이브러리

- Lombok은 Java 라이브러리로 클래스 작성 시 기계적으로 반복되는 constructor, setter, getter, toString 등의 메서드 작성 코드를 줄여주는 코드 다이어트 라이브러리(컴파일러)이다.
- 보통 DTO 클래스나 Entity 같은 도메인 클래스 등에는 수많은 인스턴스변수가 있고 이에 대응되는 constructor, getter와 setter 그리고 toString() 메서드를 작성하게 되는데, 대부분 이클립스 같은 IDE의 소스코드 자동 완성 기능을 사용하지만, 이 역시도 번거로운 작업이 될 수 있다.
- Lombok은 여러가지 에노테이션을 제공하고 이를 기반으로 코드를 컴파일과정에서 생성해 주는 방식으로 동작하는 라이브러리이다.
- 코딩 과정에서는 Lombok과 관련된 에노테이션만 설정하고 constructor, getter와 setter 메서드 등은 소스코드에 보이지 않지만 실제로 컴파일 된 바이트코드(.class)에는 코드가 생성된다.

✓ Lombok 장점

- 에노테이션 기반의 코드 자동 생성을 통한 생산성 향상
- 반복 코드 다이어트를 통해 가독성 및 유지보수성 향상
- getter/setter 외 빌더 패턴이나 로그 생성 등 다양한 방면으로 활용 가능

2.1 Lombok 소개 (2/2)

✓ Lombok 주의사항

- Lombok 라이브러리는 개발자 마다 호불호가 나뉘는 라이브러리로 일부 개발자들은 코드가 직접 눈에 보이는 직관성을 유지하는 것이 좋다고 보는 의견도 있는 만큼 프로젝트 구성원의 협의를 통해 사용 여부를 결정하여야 한다.
- 또한 API 설명과 내부 동작을 어느정도 숙지하고 사용해야 한다.
- setter에 받는 Paramter는 String으로 받고 내부적으로 이 값의 변경해서 저장 한다면 Lombok이 오히려 더 불편할 수 있다.

2.2 일반 자바 코드 vs Lombok (1/3)

✓ 일반 자바 코드

Member.java

```
public class Member {
	private String id;
	private String name;
	private String passwd;
	private String email;
	private String regdate;

	public Member() {}

	public Member(String id, String name, String passwd, String email) {
		this(id, name, passwd, email, null);
	}

	public Member(String id, String name, String passwd, String email, String regdate) {
		this.id = id;
		this.name = name;
		this.passwd = passwd;
		this.email = email;
		this.regdate = regdate;
	}
	// 중략
}
```

2.2 일반 자바 코드 vs Lombok (2/3)

✓ Lombok 라이브러리 의존성 추가

```
// https://mvnrepository.com/artifact/org.projectlombok/lombok
compileOnly 'org.projectlombok:lombok:1.18.28'
```

✓ Lombok 자바 코드

User.java

```
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@ToString
public class User {
	private String id;
	private String name;
	private String passwd;
	private String email;
	private String regdate;
}
```

2.2 일반 자바 코드 vs Lombok (3/3)

✓ 일반 자바에서 Slf4j 로깅 사용

SomeServiceImpl.java

```
public class SomeServiceImpl {
      private Log log = LogFactory.getLog(SomeServiceImpl.class);
}
```

✓ Lombok Slf4j 로깅 사용

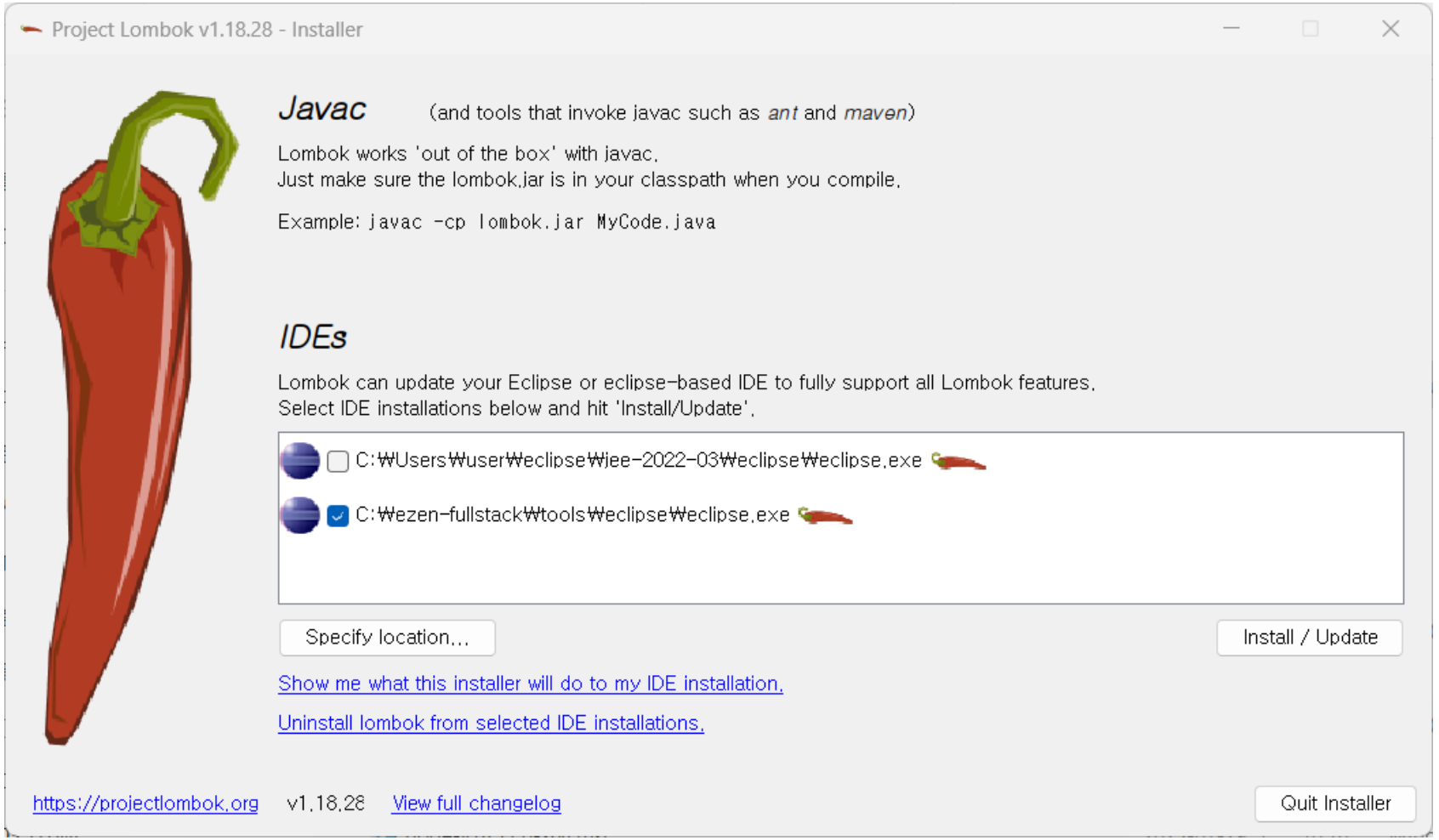
```
@Slf4j
public class LoggTest {

    @Test
    void logTest() {
        log.trace("trace : {}", "로그 메시지 테스트입니다.");
        log.debug("debug : {}", "로그 메시지 테스트입니다.");
        log.info("info : {}", "로그 메시지 테스트입니다.");
        log.warn("warn : {}", "로그 메시지 테스트입니다.");
        log.error("error : {}", "로그 메시지 테스트입니다.");
    }
}
```

2.3 Lombok 설치 및 설정

✓ 1) IDE(이클립스)에 Lombok 플러그인 다운로드 및 설치

- 다운로드 : https://projectlombok.org/download (lombok.jar)



End of Document

✓ Q&A

감사합니다...