



**Loggery** w Javie



**Michał Nowakowski**

Lead Software Engineer @EPAM

[michal@nowakowski.me.uk](mailto:michal@nowakowski.me.uk)

# Loggery

// Informacje ogólne

// Informacje techniczne

// java.util.logging

// Apache Log4j 2

// SLF4J

// Q&A

# Materiały do zajęć

// <https://github.com/infoshareacademy/jddd5-materialy-loggery>



# **Loggery**

## Informacje ogólne

# Czym jest Logger?

- // Dziennik zdarzeń aplikacji
- // Alternatywne medium komunikacji między aplikacją i użytkownikiem ☺
- // Zapis kolejnych operacji wykonywanych przez aplikację z uwzględnieniem priorytetu i źródła komunikatu

# Cel używania Loggerów

- // Debugowanie aplikacji podczas procesu wytwarzania
- // Wykrywanie błędów niewidocznych na etapie kompilacji / testów
- // Pomaganie w wykrywaniu i diagnozowaniu błędów w środowisku innym niż deweloperskie (!!?)
- // Śledzenie dostępu do aplikacji (bezpieczeństwo)
- // Tworzenie danych statystycznych

# Kto czyta logi?

// Programista

// Administrator

// Integrator

// Biznes / analityk biznesowy

// Każdy, kto analizuje zdarzenia (zachowanie) aplikacji

# Co warto logować?

- // Wejście oraz wyjście metod
- // Inicjalizacja oraz usuwanie modułów aplikacji
- // Błędy, wyjątki a także poprawne wykonania operacji
- // Ustawienia aplikacji, tryb działania
- // Sesje użytkowników oraz **ich sposób interakcji z aplikacją**

# Jak logować?

- // Językiem zrozumiałym dla odbiorcy logów
- // Czytelnie (brak skrótów, żargonu, nieznanych symboli)
- // W ustrukturyzowany sposób, konsekwentnie używając podobnego formatu wiadomości
- // Jak najwięcej z uwzględnieniem odpowiednich poziomów logowania



# **Loggery**

Informacje techniczne

# Java Logging Frameworks

// Java Logging API (Oracle)

// Log4j (Apache)

// SLF4J

// Logback

// tinylog

// i wiele wiele innych ...

# Wizualizacja logów

// Logować możemy najróżniejsze zdarzenia: koniecznie błędy, ale również każde logowanie użytkownika, informacje o pełnym załadowaniu się serwisu, założonych kontach i rozkładzie tych operacji w poszczególnych dniach, godzinach, itp.

// <https://play.grafana.org>

# Nazewnictwo i hierarchia

- // Każdy Logger ma swoją nazwę – najczęściej jest to nazwa klasy (z pełną nazwą pakietu), w której jest on użyty
- // Pełna nazwa klasy określa hierarchię loggerów: Logger o nazwie `com.isa.domain.Product` należy do hierarchii `com`, `com.isa`, ale nie należy do `org.springframework`
- // Dla każdej hierarchii możemy stosować osobną konfigurację, co uelastycznia podejście do tematu logów (np. w pliku tesktowym mogą być zapisywane tylko logi z danej hierarchii / pakietu)

# Poziomy logowania

// Poziom logowania to „ważność” danej wiadomości zapisywanej w logach

// Każda biblioteka ma różne zestawy poziomów:

„FATAL, ERROR, WARNING, INFO, DEBUG, TRACE”

„SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST”

# Poziomy logowania

Poziom (priorytet)	Opis
<b>FATAL (SEVERE)</b>	Poważne błędy powodujące przedwczesne zakończenie działania aplikacji (np. brak dostępnej pamięci).
<b>ERROR</b>	Błędy wykonania (np. błąd parsowania pliku JSON).
<b>WARN</b>	Sytuacje niespodziewane niewpływające na wadliwe działanie (np. błąd w komunikacji sieciowej, złapany wyjątek).
<b>INFO</b>	Śledzenie działania aplikacji.
<b>DEBUG (CONFIG/FINE/FINER)</b>	Szczegółowe informacje dotyczące przepływu w działaniu aplikacji w celach diagnostycznych.
<b>TRACE (FINEST)</b>	Bardzo szczegółowe informacje.
<b>ALL / OFF</b>	Wszystkie poziomy / wyłączenie logów.

# Poziomy logowania

- // Wysyłanie maili / notyfikacji o logach na poziomie FATAL/SEVERE
  
- // Zapisywanie w dzienniku logów (pliku?) informacji wyłącznie o logach na poziomie np. WARNING lub wyższym

# Zadanie 1

// Pakiet **workshop01**

// Wypisz na ekran jak najwięcej informacji używając System.out i System.err



# **Loggery**

java.util.logging

# java.util.logging

- // Stanowi standard języka Java – nie są wymagane żadne dodatkowe zależności w celu użycia biblioteki
- // Charakterystyczne poziomy logowania: CONFIG, FINE, FINER
- // Każda wiadomość poprzedzona jest wartością enum (poziom wpisu)
- // Najniższy poziom logowania opisany w pliku logging.properties w katalogu JRE\_HOME/lib

# java.util.logging – Konfiguracja

// Ustawienie atrybutu JVM wskazującego na konfigurację:  
-Djava.util.logging.config.file=logging.properties

// Przykładowa zawartość (logujemy wszystko (ALL) do konsoli):

```
handlers=java.util.logging.ConsoleHandler
.level=ALL
java.util.logging.ConsoleHandler.level=ALL
```

# java.util.logging – Inicjalizacja

// Pole statyczne i finalne dla całej klasy:

```
private static final Logger LOG =  
    Logger.getLogger(JavaUtilLogger.class.getName());
```

# java.util.logging – Użycie

```
LOG.fine("Hello");
```

```
LOG.severe("Another message");
```

```
LOG.log(Level.WARNING, "Some warning, please check the files");
```

# Zadanie 2

// Pakiet **workshop02**

// Zaimplementuj logowanie na odpowiednim poziomie z wykorzystaniem biblioteki `java.util.logging`



# **Loggery**

## Apache Log4j 2

# Apache Log4j 2

- // Najpopularniejsza biblioteka do logowania w Javie
- // Łatwiejsza w konfiguracji niż java.util.logging
- // Istnieją implementacje dla innych środowisk (Ruby – log4r,  
Angular – log4ng, C# - log4net)

# Apache Log4j 2 – Komponenty

// **Loggers**: przechwytują zdarzenia z aplikacji i przekazują je do odpowiedniego appendera.

// **Layouts** (formatters): konwertują oraz formatują dana zawarte w zdarzeniach logowania. Determinują jak dane będą wyglądały jako wpisy w miejscu docelowym.

// **Appenders** (handlers): zapisują zdarzenia logowania do zdefiniowanej destynacji. Appender używa Layoutu do sformatowania zdarzenia przed wysłaniem go do miejsca docelowego.

# Apache Log4j 2 – Konfiguracja

- // Konfiguracja opisana w pliku zewnętrzny – np. XML, JSON lub YAML – np. plik log4j2.xml w classpath
- // Konfiguracja na poziomie języka programowania, np. fabryka konfiguracji i implementacja konkretnej konfiguracji
- // W przypadku braku konfiguracji, log4j używa domyślnej

# Apache Log4j 2 – Konfiguracja

// log4j2.xml w classpath (np. w resources/)

// Logowanie do konsoli wszystkich wpisów (ALL)

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="ALL">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

# Apache Log4j 2 – Zależności

```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.11.1</version>
</dependency>

<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.11.1</version>
</dependency>
```

# Apache Log4j 2 – Inicjalizacja

// Pole statyczne i finalne dla całej klasy:

```
private static final Logger LOG =  
    LogManager.getLogger(Log4jLogger.class);
```

# Apache Log4j 2 – Użycie

```
LOG.info("Hello!");
```

```
LOG.log(Level.ERROR, "Something went wrong");
```

```
LOG.warn("This is message with param 1 {} and param 2 {}", 12, "34");
```

# Zadanie 3

// Pakiet **workshop03**

// Zamiast java.util.logging wykorzystaj bibliotekę Apache Log4j  
(dodaj log4j2.xml ze slajdów)

# Apache Log4j 2 – Pattern

// Format wpisów (logów) definiowany jest jako wzór (pattern)

// Może mieć dowolną postać

```
<PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger - %msg%n"/>
```

# Apache Log4j 2 – Pattern

// %d{HH:mm:ss.SSS} – czas

// %t – nazwa wątku

// %-5level – poziom logowania (wyrównany do 5 znaków od lewej)

// %p – poziom logowania (ERROR, WARN, itp.)

// %logger – nazwa loggera

# Apache Log4j 2 – Pattern

// %d – pełna data

// %c{1.} – pełna nazwa klasy, przy czym poszczególne człony pakietu ograniczone są do 1 znaku ({1.}) – np. com.info.Main -> c.i.Main

// %msg lub %m – wiadomość z logu

// %n – znak nowej linii

# Zadanie 4

// Pakiet **workshop04**

// Zmodyfikuj Layout tak, aby logi w konsoli miały format:

```
[16:03:41.858 - INFO  ] Hello!
[16:03:41.872 - ERROR ] Something went wrong
[16:03:41.875 - WARN  ] This is message with param 1 12 and param 2 34
```

# Apache Log4j 2 – Logi w pliku

// Aby logować do pliku, musimy zdefiniować nowy **Appender**:

```
<File name="LogFile" fileName="app.log">
    <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
    </PatternLayout>
</File>

<Async name="Async">
    <AppenderRef ref="LogFile"/>
</Async>
```

# Apache Log4j 2 – Logi w pliku

// I dodać referencję do niego w **Loggerze**:

```
<Loggers>
    <Root level="ALL">
        <AppenderRef ref="Console" level="ALL"/>
        <AppenderRef ref="Async" level="ERROR"/>
    </Root>
</Loggers>
```

# Zadanie 5

// Pakiet **workshop05**

// Zdefiniuj dodatkowy Appender tak, aby aplikacja logowała do konsoli (ALL) i do pliku tekstowego (INFO+)



# **Loggery**

Simple Logging Facade for Java (SLF4J)

# Jaki problem rozwiązuje slf4j?

- // Wprowadza poziom abstrakcji w logowaniu zdarzeń
- // Rozdzielenie API od implementacji Loggera
- // Zmieniając framework logujący nie musimy modyfikować istniejącego kodu
- // Programiści używają tego samego API, natomiast różne środowiska używają własnych implementacji Loggerów (np. log4j, logback)

# slf4j – Zależności

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.25</version>
</dependency>

<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-slf4j-impl</artifactId>
    <version>2.11.1</version>
</dependency>
```

# slf4j – Inicjalizacja

// Pole statyczne i finalne dla całej klasy:

```
private static final Logger LOG =  
    LoggerFactory.getLogger(Slf4jLogger.class);
```

# slf4j – Użycie

```
LOG.info("Hello!");
```

```
LOG.error("Something went wrong");
```

```
LOG.warn("This is message with param 1 {} and param 2 {}", 12, "34");
```

# Zadanie 6

// Pakiet **workshop06**

// Dodaj obsługę SLF4J i zmodyfikuj kod tak, aby używać interfejsów z pakietu slf4j zamiast klas z pakietu log4j

# Zadanie 7

// Pakiet **workshop07**

// Zmodyfikuj kod tak, aby logować wg komentarzy

# Q&A

// THANKS!