Lesson 03.B
Log Management

DEVOPS - ITI 4
HEI 2021-2022

What is the interest of the logs?

It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.

Sherlock Holmes

A problem is detected on your application.
What do you need to identify it ?

Logging overview

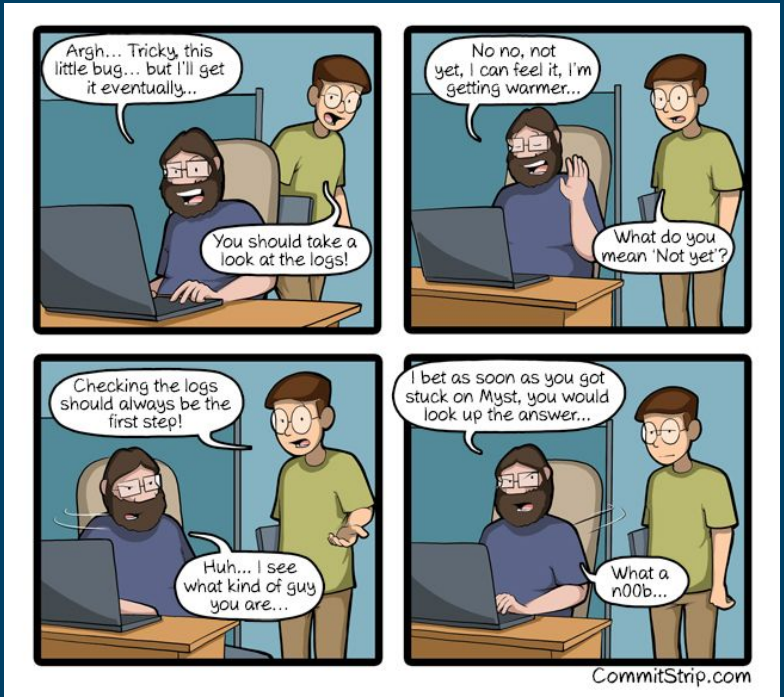
- Message with informations about program execution
- Generated by program

```
2018-08-25 09:56:46.041  INFO 10448 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto
handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2018-08-25 09:56:46.238  INFO 10448 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice:
org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationContext@5ea434c8: startup date [Sat Aug 25 09:56:43
CEST 2018]; root of context hierarchy
2018-08-25 09:56:46.298  INFO 10448 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/]}" onto public
java.lang.String hei.devops.logs.controller.HelloController.index()
2018-08-25 09:56:46.301  INFO 10448 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public
org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>>
org.springframework.boot.autoconfigure.web.servlet.error.BasicErrorController.error(javax.servlet.http.HttpServletRequest)
2018-08-25 09:56:46.302  INFO 10448 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}"
onto public org.springframework.web.servlet.ModelAndView
org.springframework.boot.autoconfigure.web.servlet.error.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest,javax.servlet.htt
p.HttpServletResponse)
2018-08-25 09:56:46.324  INFO 10448 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler
of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2018-08-25 09:56:46.324  INFO 10448 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type
[class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2018-08-25 09:56:46.455  INFO 10448 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on
startup
2018-08-25 09:56:46.560  INFO 10448 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http)
with context path ''
2018-08-25 09:56:46.563  INFO 10448 --- [           main] hei.devops.logs.Application              : Started Application in 3.177 seconds (JVM
running for 3.693)
```

Log use case

- Investigation to find the source of an anomaly
- Detect suspicious behavior (to be alert before users)
- Monitor the use of the software (with Elasticsearch for example)

How to use logs ?



Quizz !!!

https://forms.gle/vNyoPqNUvSfAL1T76

Log levels

Log levels	Use case
FATAL	Unexpected events that prevent the application from running.
ERROR	Unexpected events with impact for the user but does not prevent the application from running.
WARN	Unexpected events without impact for the user
INFO	Expected events with high added value (for examples : user actions, long treatments status, ...)
DEBUG	Information about main methods with parameters and result
TRACE	All other informations

Good practices

- Add timestamp to the logs
- Use a specific format for logs
- Show stacktrace to an unexpected exception

Bad practices

- Show personal data (only id)
- Show password
- Show stacktrace to an expected exception

How to write Log with Log4J ?

Using Log4J

- Java logging framework

- To add the libraries to the project with maven :

```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.14.1</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.14.1</version>
</dependency>
```

Add a Logger in a class

- Add the following to the imports section of your java code :

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
```

- Add this class variable :

```
static final Logger LOGGER = LogManager.getLogger();
```

```
package hei.devops.y2019.lesson03b;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class LogExampleService {

      static final Logger LOGGER = LogManager.getLogger();
}
```

Write a log (1/2)

```
LOGGER.trace("Hello, I am a trace log");

LOGGER.debug("Hello, I am a debug log");

LOGGER.info("Hello, I am an info log");

LOGGER.warn("Hello, I am a warn log");

LOGGER.error("Hello, I am an error log");

LOGGER.fatal("Hello, I am a fatal log");
```

Write a log (2/2)

- Add variable in a log message with {} :

```
LOGGER.info("Hello, I am an info log with 2 variables : first {} ; second {}",
                    "I am the first variable", "I am the second variable");
```

- Add stacktrace in a log message :

```
LOGGER.error("Hello, I am an error log with an exception", new Exception());
```

How to configure Log4j ?

Overview

- Log4j2 can be configured in different ways:
  - Key/Value (.properties)
  - YAML (.yaml / .yml)
  - JSON (.json / .jsn)
  - XML (.xml)

⇒ It's different formats but they do the same things !

XML Overview

- Log4j2 configuration files are :
  - log4j2.xml in *src/main/resources* (If file does not exist then use the default configuration)
  - log4j2-test log4j2-test.xml in *src/test/resources* (If file does not exist then use log4j2.xml)

- XML file with root element tag *configuration* contains :
  - *appenders* with
    - one or more *appender*
  - *loggers* with
    - one *root*
    - zero or more *logger*

Appender

- appender defines how write logs

- Each appender contains :
  - Mandatory attribute *name*: Name of the appender
  - Mandatory attribute *type* : Type of the appender (Console/File/…)
  - Elements related to the type of the appender
  - Not Mandatory element layout : To define the pattern of the logs

Example of appenders

To write logs in console

```
<appender type="Console" name="CONSOLE">
        <layout type="PatternLayout"
        pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
</appender>
```

To write logs in a file

```
<appender type="File" name="FILE" fileName="target/test.log">
        <layout type="PatternLayout"
        pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
</appender>
```

Logger

- logger defines specific log management for a package and its sub-packages

- Each logger contains :
  - Mandatory attribute *level*: Log level (ALL/TRACE/DEBUG/INFO/WARN/ERROR/FATAL/OFF)
  - Mandatory attribute *name* : Targeted package
  - zero or more element *appender-ref* :
    - To declare appender with mandatory attribute *ref*
    - Each appender-ref add its logs to its output

Root

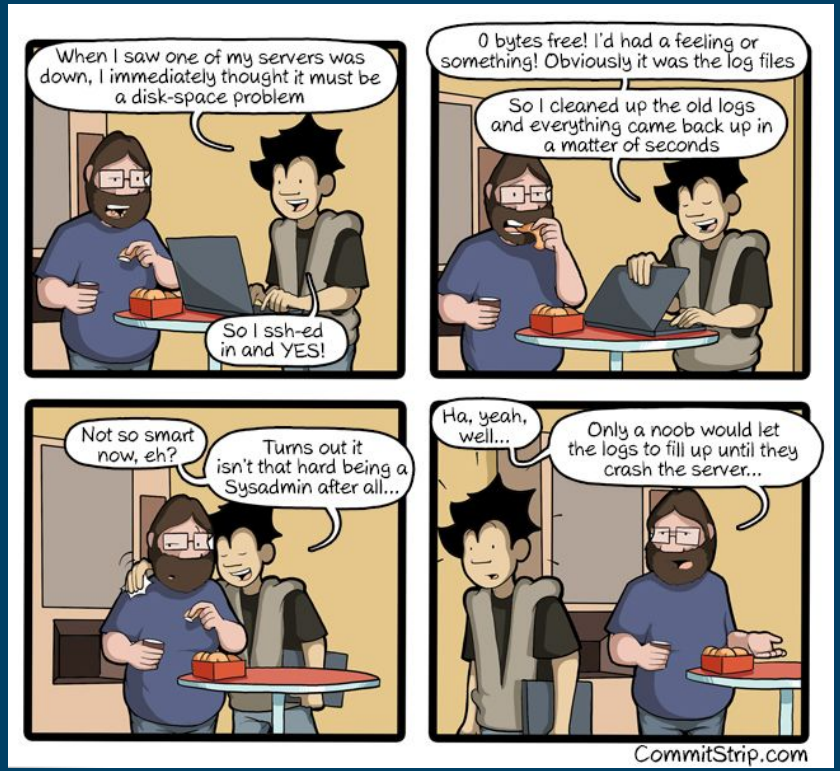
- *root* defines default log management for the project
  - root is unique
  - root do not need define targeted package by *name*

- root works as the other logger

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration strict="true">
        <appenders>
                <appender type="Console" name="CONSOLE">
                        <layout type="PatternLayout"
                                pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
                </appender>
                <appender type="File" name="FILE" fileName="target/test.log">
                        <layout type="PatternLayout"
                                pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
                </appender>
        </appenders>
        <loggers>
                <logger name="hei.devops" level="warn" >
                        <appenderRef ref="CONSOLE" />
                </logger>
                <root level="error">
                        <appenderRef ref="FILE" />
                        <appenderRef ref="CONSOLE" />
                </root>
        </loggers>
</configuration>
```

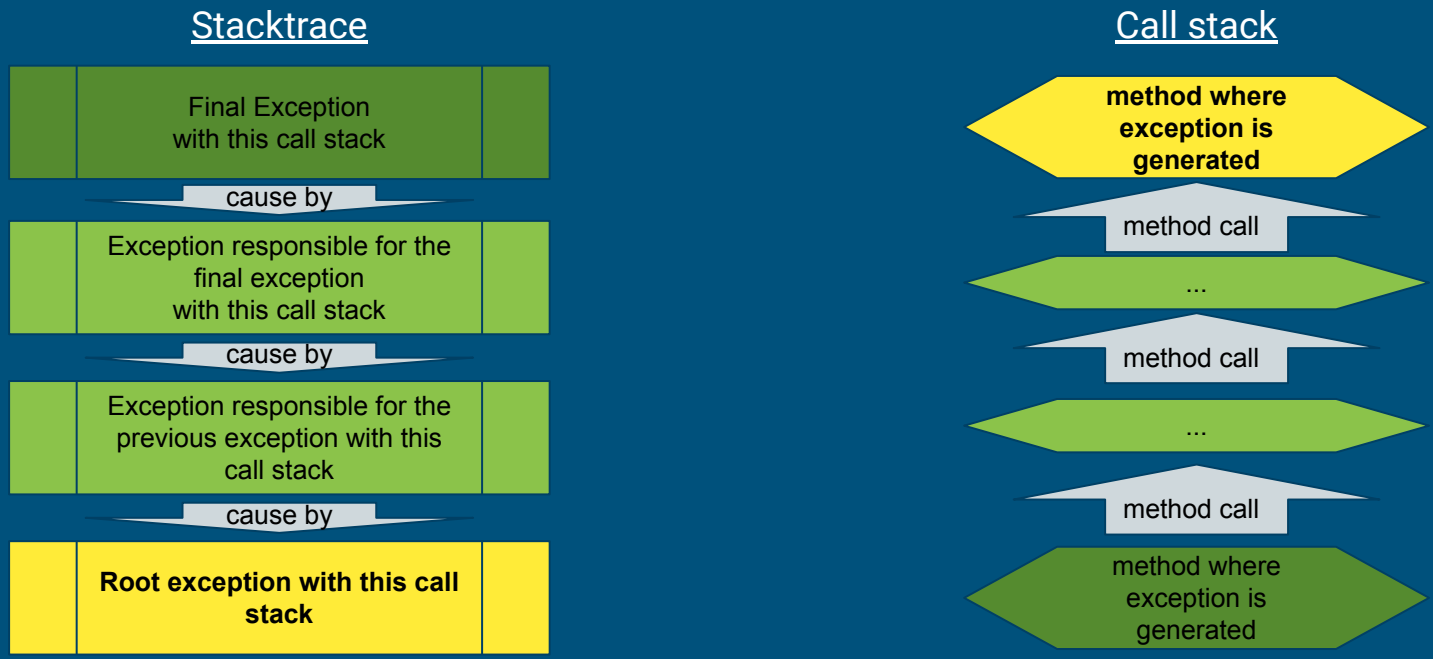
How to read a stacktrace ?



Description

- Is textual representation of a Java Exception
- Tracks the sequence of nested functions to the point where the stack trace is generated
- Is your best friend to understand root cause of a bug

Composition



Example

```
Exception in thread "main" hei.devops.exception.HelloWorldException: hei.devops.exception.UserInvalidException: java.lang.NullPointerException
        at hei.devops.service.impl.HelloServiceImpl.printHelloWorld(HelloServiceImpl.java:34)
        at hei.devops.service.impl.HelloServiceImpl.printHelloWithoutSafeMode(HelloServiceImpl.java:18)
        at hei.devops.App.main(App.java:12)
Caused by: hei.devops.exception.UserInvalidException: java.lang.NullPointerException
        at hei.devops.service.impl.HelloServiceImpl.returnMessage(HelloServiceImpl.java:42)
        at hei.devops.service.impl.HelloServiceImpl.printHelloWorld(HelloServiceImpl.java:31)
        ... 2 more
Caused by: java.lang.NullPointerException
        at hei.devops.service.impl.HelloServiceImpl.returnMessage(HelloServiceImpl.java:40)
        ... 3 more
```

NullPointerException

- Most famous exception
- Cause : A call to a method from a null object
- Fix : Check object is not null before to call a method

Example : NullPointerException

Dangerous Code

```
public void printHelloWithoutSafeMode(User user){
    System.out.println( "Hello "+user.getFirstName+" !");
}
```

Safe code

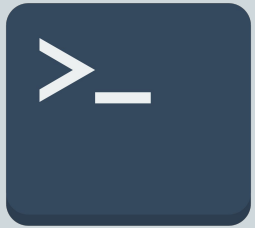
```
public void printHelloWithSafeMode(User user) throws HelloWorldException {
    if(user !=null) {
        System.out.println( "Hello "+user.getFirstName+" !");
    }else {
        throw new HelloWorldException("User is null");
    }
}
```

To conclude

Question

What's going to happen ?

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration strict="true">
        <appenders>
                <appender type="Console" name="CONSOLE">
                        <layout type="PatternLayout"
                                pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
                </appender>
                <appender type="File" name="FILE" fileName="target/test.log">
                        <layout type="PatternLayout"
                                pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
                </appender>
        </appenders>
        <loggers>
                <logger name="hei.devops" level="warn" >
                        <appenderRef ref="CONSOLE" />
                </logger>
                <root level="error">
                        <appenderRef ref="FILE" />
                        <appenderRef ref="CONSOLE" />
                </root>
        </loggers>
</configuration>
```



Thank you for you attention !

Links :

- Sources
  - http://www.commitstrip.com/
  - https://giphy.com/
  - https://logging.apache.org/log4j/2.x/javadoc.html

- Examples :
  - https://gitlab.com/hei-devops/lesson/lesson-2021/lesson-03b