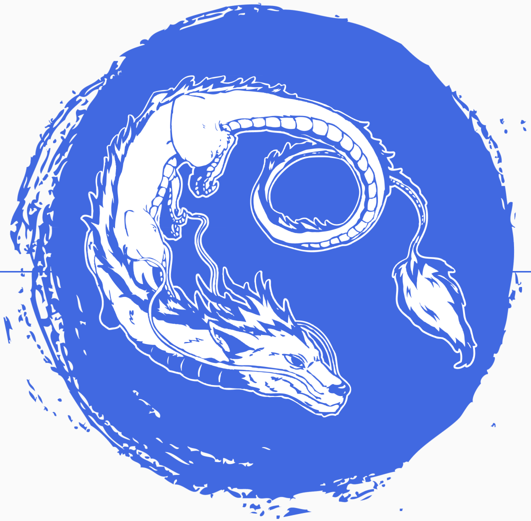
Logging

YAKA 2022 Team



Copyright

What is it?

- Structured
- Text
- Tells the story of the application

Why is it needed ?

- Cross-application standardization
- Structure -> knowledge
- Flexibility

How to implement ?

- By hand
    - Much more complicated than it seems to get right.
- Using frameworks
    - Obviously the right choice, who needs a new wheel concept ?
    - We will use `logback` as an implementation during the workshop.

How to produce logs ?

```
final Logger logger = LoggerFactory.getLogger(this.getClass());
logger.info("Hello world");
```

Will produce:

```
08:53:01.896 [main] INFO com.epita.scratch.Logging - Hello, world
```

Notice the added information:

- Some form of timestamp
- The thread from which the log has been created
- The logging level
- The source logger (usually the name of the class).
- The actual message.

About log levels

- Levels are used to specify the importance and granularity of a message.
- The logger can be configured to ignore messages lower than a given priority.
- The usual threshold for a production system is `INFO`.

Usual levels

- ALL: lowest possible rank, everything is printed.
- TRACE: Very fine grained tracking.
- DEBUG: Fine grained activity tracking for specific debugging purposes.
- INFO: Coarse grained activity tracking.
- WARN: A suspicious situation has been encountered, check soonish maybe ?
- ERROR: Indicates a recoverable error.
- FATAL: Used to indicate an error from which the program won’t recover.
- OFF: Turns off logging entirely.

Parameter expansion

- Logback allows parameter expansion instead of string concatenation
- Saves costly string serialization and concatenation if the log is below threshold.
- Simple use {} as a placeholder in your log messages.

```java
logger.warn("An error occurred while parsing string {} from user {}", string, user);
```

Exceptions

- Logback knows how to print exceptions properly.
- Most logging methods have an overload taking an exception as its last parameter.

```java
catch (final Exception exception) {
  logger.error("Unknown error", exception);
}
```

Where should I use logging ?

- In place of any System.out.println you might already have.
- In case of an anomaly: fatal/error/warn, depending of the severity.
- Anywhere a meaningful branch/decision has been taken: info.
- For fine grained debugging: debug/trace.

Beyond the scope of the workshop

- Appenders allow to route message to a variety of targets:
    - Files, with various formats
    - Network, with various protocols
    - Database
    - Custom
- Level can be configured on a per-logger and/or per-appender basis.