Theory: Standard logger

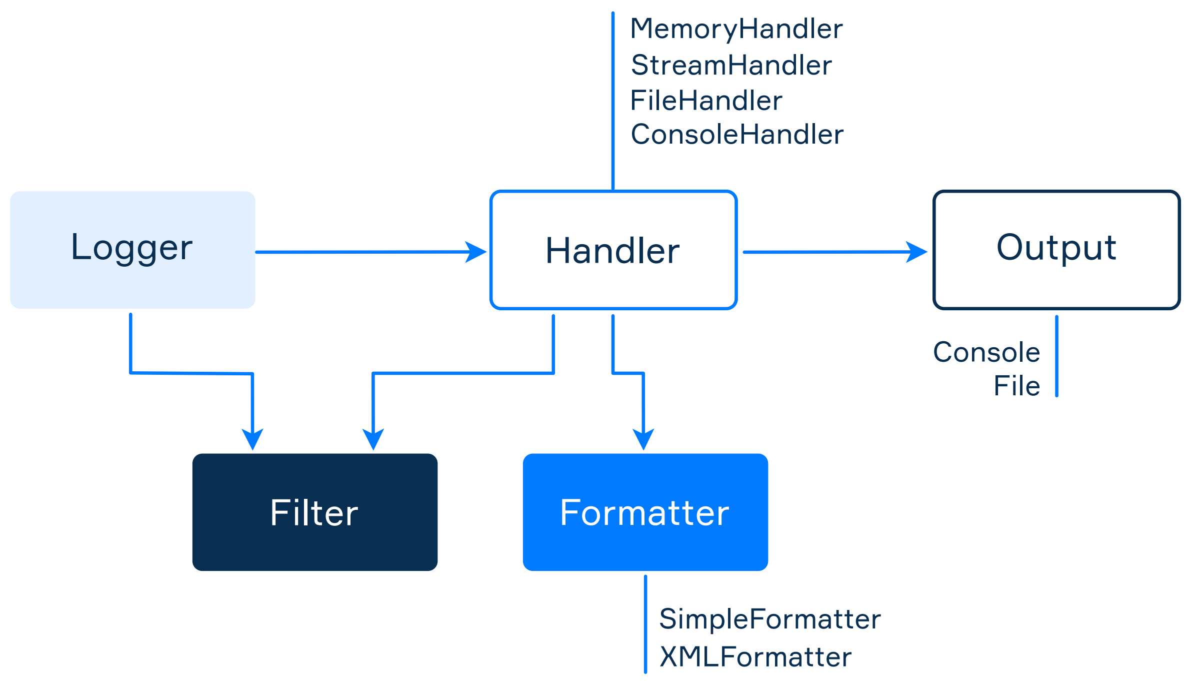
17 minutes 0 / 5 problems solved

Skip this topic Start practicing

509 users solved this topic. Latest completion was 2 days ago.

§1. Introduction to Java Logging

Logs are records of a software application which we choose to save to a file or display in a console. These records could be anything such as an event in the application, a value of a variable, error or an exception in the application. Logs are mostly used for debugging purposes. Today, we learn about `java.util.logging` package which is responsible for giving the developers logging capabilities within Java standard SDK. There are several components you need to learn when working with Java Logging. Those are `Logger` class, `FileHandler` class, `ConsoleHandler` class, `SimpleFormatter`, `XMLFormatter`, `Level`, `LogRecord` and `LogManager`. We will be discussing the first 6 components in this lesson. Following images show how Loggers, Handlers, Filters, and Formatters work together.



§2. Logger class

`Logger` class is the most important and fundamental component in the logging package. The standard practice is to create a logger instance for each class. `Logger` class introduces several methods to print log messages. `log()` method is one of them. Check the following example.

```
import java.util.logging.*;

public class Main {
    public static void main(String[] args) {
        Logger logger = Logger.getLogger(Main.class.getName());
        logger.log(Level.WARNING, "Hello " + logger.getName());
    }
}
```

It will output

```
WARNING: Hello Main
```

First, we create a logger instance using the class name. Then we call the `log()` method to print the log message. `log()` method takes two arguments. Its first argument is a **Level** object and the second argument is a message. `Level.WARNING` is a constant from **Level** class in logging package.

Every log message is related to a certain log level. In this example, it is **Warning**. Java uses **Info** as its **default log level**. There are **seven** log levels in Java logging package. The list below shows them from the highest severity to lowest severity.

- SEVERE
- WARNING

Current topic:

Standard logger

Topic depends on:

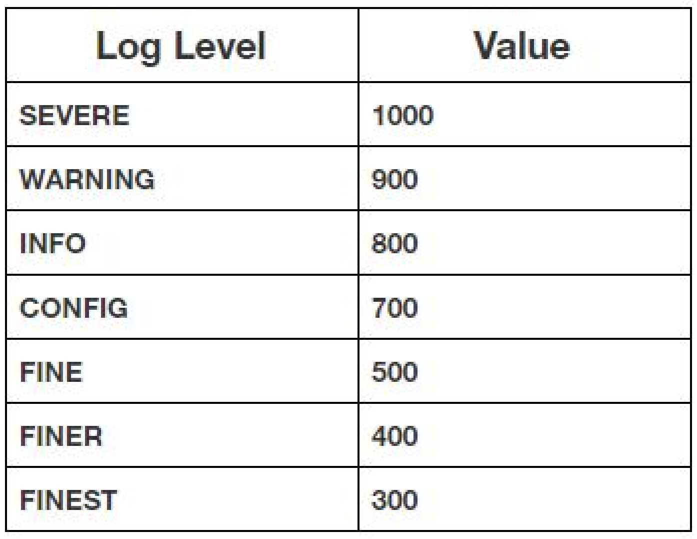
✗ Introduction to logging

✗ Interface

Table of contents:

- INFO
- CONFIG
- FINE
- FINER
- FINEST

Following image show integer values of the Log Levels.



Log Level	Value
SEVERE	1000
WARNING	900
INFO	800
CONFIG	700
FINE	500
FINER	400
FINEST	300

Logger class contains methods such as `info()`, `config()` where you don't have to provide a **log level** as an attribute.
Check the following example.

```
import java.util.logging.*;

public class Main {
    public static void main(String[] args) {
        Logger logger = Logger.getLogger( Main.class.getName());
        logger.severe("Severe Log");
        logger.warning("Warning Log");
        logger.info("Info Log");
    }

}
```

Output will be

```
Apr 04, 2019 10:01:34 PM Main main
SEVERE: Severe Log
Apr 04, 2019 10:01:34 PM Main main
WARNING: Warning Log
Apr 04, 2019 10:01:34 PM Main main
INFO: Info Log
```

§3. Handlers and Formatters

The next important components in the logging package are **Handlers** and **Formatters**. **Handlers** and **Formatters** often work together. **Handlers** are responsible for taking actual logs to the outside world. There is an abstract class called **Handler** in `java.util.logging` package. It is extended by five concrete classes. Two most important classes among them are `ConsoleHandler` and `FileHandler`. `ConsoleHandler` writes log messages to System.err while `FileHandler` writes log messages to a file.

Usually, a **Handler** uses a **Formatter** to format the log message. There are two types of Formatters in the logging package. Those are `SimpleFormatter` and `XMLFormatter`. Of course, both of them extend the **Formatter** abstract class in the logging package.
Check the following example to understand **Handlers** and **Formatters**.

```
import java.util.logging.*;

public class Main {
    public static void main(String[] args) throws Exception {
        Logger logger = Logger.getLogger( Main.class.getName());
        Handler fileHandler = new FileHandler("default.log");
        logger.addHandler(fileHandler);
        fileHandler.setFormatter(new XMLFormatter());
        logger.info("Info log message");

    }

}
```

It will create a log file called **default.log**. Default.log file will contain the following XML text.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
 <date>2019-04-04T22:26:35</date>
 <millis>1554416795693</millis>
 <sequence>0</sequence>
 <logger>Main</logger>
 <level>INFO</level>

 <class>Main</class>

 <method>main</method>

 <thread>1</thread>

 <message>Info log message</message>

</record>

</log>
```

§4. Filters

Let's discuss **Filters** now. When we are developing a software application, we write as many log messages as possible. But we don't want all the log messages to be executed every time application runs. It will waste resources and also it can create unnecessarily long log files. That's when we use filters. Let's say you want to print only info messages. For that, first, you have to create a **custom filter** class by implementing `Filter` **interface** in logging package.

```
class FilterExample implements Filter {
    public boolean isLoggable(LogRecord record) {
        if (record.getLevel() != Level.INFO) {
            return false;
        }
        return true;
    }
}
```

Now you create an object of `FilterExample` class. Then use `setFilter()` method of the Logger instance to set the filter.

```
public class Main {
    public static void main(String[] args) throws Exception {
        Logger logger = Logger.getLogger( Main.class.getName());
        Filter filter = new FilterExample();
        logger.setFilter(filter);
        logger.severe("Severe Log");
        logger.info("Info Log");
        logger.warning("Warning Log");
    }

}
```

When this code is executed, only `Info log` message will be printed.

§5. Conclusion

Let's summarize what we learned in this lesson. First, `java.util.logging` is a part of Java SDK and it is responsible for giving logging capabilities to developers. We discussed several components in the logging package. Logger instances are responsible for creating log messages. We usually create a Logger instance for every class that we are going to add logs. Handlers are responsible for sending log message out of the application. If you want to print log messages to console, use `ConsoleHandler`. If you want to write log messages to a file, use `FileHandler`. Formatters format log messages. If you want to log messages in XML format, use `XMLFormatter`. Finally, we discussed Filters which help you to manage which logs are to be executed when the application runs.