Data Structures and Algorithms

Lecture 3 — Logging

Alan P. Sexton

University of Birmingham

Spring 2019

Logging

It is common to write print statements in your code to

- Provide debug output to analyse
- Monitor status of running program

Logging

It is common to write print statements in your code to

- Provide debug output to analyse
- Monitor status of running program

Problems for Debugging:

- When bug found and fixed, code is polluted with print statements that have to be deleted before release
- Another bug leads to re-entering print statements
- Better to leave them there but switch them on and off programmatically

Logging

It is common to write print statements in your code to

- Provide debug output to analyse
- Monitor status of running program

Problems for Debugging:

- When bug found and fixed, code is polluted with print statements that have to be deleted before release
- Another bug leads to re-entering print statements
- Better to leave them there but switch them on and off programmatically

Problems for monitoring long jobs, e.g. Web Servers, System,

- Where do the messages go?
- How can the messages be kept, archived, searched, managed?

Can roll your own solution, but better to use a tried and trusted **Logging** framework

Logging in Java

- There are many logging libraries (engines) in Java...
- and many front ends (facades) that provide a common interface to different logging libraries
- One of the most capable and used engines in industry is `log4j version 1.2` (basis for many modern logging systems)
  - Officially `log4j 1.2` is no longer under development and log4j 2.x has replaced it
  - 2.x is faster, memory leak free, has some extra facilities but...
  - is a bit more cumbersome and prone to configuration errors...
  - and industry has not switched wholeheartedly to it
- There is a logging library in the Java JDK distribution (`Java.Util.Logging`):
  - Less capable than log4j 1.2
  - Okay for small simple projects, but not flexible or powerful enough for large ones
  - Not very popular in industry

Sample Application

For practicing with logging, clone the module's `video-store` GIT repository and install it as a project:

- `https://git.cs.bham.ac.uk/aps/video-store.git`
- This is a read-only repository so you will not be able to push to it. If you want to modify it, copy it and initialize an Eclipse project on it as described in the Eclipse handout.
- For the remainder of this handout, I will assume you have followed the instructions in the Eclipse handout to configure Eclipse, install the libraries and create and execute a run configuration

Getting Started — Simple Basic log4j Configuration

There is already a log4j configuration file in the `video-store` project. Changing this file is how you control your logging behaviour without having to edit or recompile your Java sources. The file must be called "`log4j.properties`" and must be in your source directory. A simple such file could contain:

```
# Levels, in order, are:
#  [ALL], TRACE, DEBUG, INFO, WARN, ERROR, FATAL, [OFF]

log4j.rootLogger=DEBUG, MyConsole
log4j.appender.MyConsole=org.apache.log4j.ConsoleAppender
log4j.appender.MyConsole.layout=\
        org.apache.log4j.PatternLayout
log4j.appender.MyConsole.layout.ConversionPattern=\
        %l %-4r [%t] %-5p %c %x - %m%n
```

- Note that the "\" character at the end of a line in this file is a continuation marker, indicating that the current line is continued on the next.
- Note that this configuration file will only output log messages that are of level `DEBUG` or higher (`INFO`, `WARN`, etc.).

Getting Started — Simple Basic log4j Logging

In each source file from which you want to output log messages, add the following import:

- `import org.apache.log4j.Logger;`

For each class, `MyClass`, from which you want to output log messages, add a new field:

- `final static Logger logger = Logger.getLogger(MyClass.class);`

For each log message you want to write, add the statement:

- `logger.<LEVEL>(message);` or `logger.<LEVEL>(message, throwable);`

where

- `<LEVEL>` can be any one of `trace`, `debug`, `info`, `warn`, `error` or `fatal`, indicating the level of the log message
- `message` is a `String` describing the event being logged
- `throwable` is a throwable object resulting from a caught exception: this results in a stack trace being output to the log

log4j Templates in Eclipse

Adding the logger in every Java class can get tedious: you can add a template in Eclipse to make it easier:
`Window|Preferences|Java|Editor|Templates` and set:

- Name to "`logger`"
- Context to "`Java type members`"
- Description to "Add a log4j 1.2 logger, updating the imports if necessary"
- Pattern to:

```
${:import(org.apache.log4j.Logger)}
private static final Logger logger =
    Logger.getLogger(${enclosing_type}.class);
```

- This and other templates for writing log messages can be imported from `Eclipse_templates_log4j_1.xml` in the root of the sample application, as described in the Eclipse handout

Using log4j Templates in Eclipse

- Now when inside a class but outside any method, typing `logg<CTRL-SPACE><CTRL-SPACE><ENTER>` will add the import if it was not already there and add the appropriate logger declaration.
- The other templates work when inside a method and are:
  - logtrace
  - logdebug
  - logwarn
  - logerror
  - logfatal
- For example: `logd<CTRL-SPACE><CTRL-SPACE><ENTER>` will insert the `logger.debug("");` statement. Use `<TAB>` to step through the parts to change
- In practice it is very rarely necessary to use more than one logger object in a class, so one should almost never change the logger variable name from "`logger`"

Debugging your log4j Configuration

All the logging frameworks are notorious for making it easy to mess up your configuration. To find out what is going on when log4j is misbehaving you can output log4j's own log messages:

- This only works to the console, as it uses minimal features of log4j to limit the dependencies on possibly broken configurations
- In your run configuration, go to the `Arguments` tab and, in the `VM Arguments` panel, add `-Dlog4j.debug=true`
- This will then output log messages to console recording log4j's efforts to find and read the configuration file and setting up the various log4j objects (loggers, appenders, layouts, etc.)

Formatting Messages

Each `Appender` object has a `Layout` object assigned whose task is to format the log message for output.

- See the Javadoc for `Layout` to see all possible subclasses, but `PatternLayout` is the most common and powerful formatter for plain text messages (the Javadoc explains all the variables you can use in `PatternLayout`'s message format)
- To see the Javadoc (assuming you have attached the Javadoc for log4j as described earlier) do the following: in your `Package Explorer` window, expand the `log4j` library, expand the `log4j-1.2.17.jar` file, expand `org.apache.log4j`, select the `Layout` class, then `Navigate|Open Attached Javadocs`
- You may prefer to use your standard browser, rather than the one built in to Eclipse: `Window|Preferences|General|Web Browser`, Then select "`Use external web browser`"

Configuring log4j

log4j can be configured programmatically through its API, or through an XML or a *properties* file.

- The real power of logging frameworks is that they can be controlled, without recompilation, through their configuration files
- XML configuration is a bit more powerful than property file configuration, but not hugely and is more complex and verbose
- A property file is just a text file with a list of assignments of values to variables

Controlling the Log

- *Loggers* define what level messages should be generated and identifies the *Appenders* that should output them. Each *Logger* has one level (default is DEBUG but can be inherited from an ancestor *Logger*)
- *Appenders* identify where the message goes and which *Layout* to use to format the message. They can also have a filter or *Threshold* set to limit which messages they actually output
- *Layouts* define the detailed format of the message.
- There is a root *Logger* called `rootLogger` which must be configured. All other *Loggers* are optional and must be given names (usually the fully qualified name of a class or a package)
- *Loggers* can have multiple different *Appenders*
- *Appenders* can have multipart names but usually only have simple names that indicate the target they send messages to
- Each *Appender* has a single *Layout*

Simple log4j Configuration

```
# Levels, in order, are:
#  [ALL], TRACE, DEBUG, INFO, WARN, ERROR, FATAL, [OFF]

log4j.rootLogger=DEBUG, MyConsole
log4j.appender.MyConsole=org.apache.log4j.ConsoleAppender
log4j.appender.MyConsole.layout=\
        org.apache.log4j.PatternLayout
log4j.appender.MyConsole.layout.ConversionPattern=\
        %l %-4r [%t] %-5p %c %x - %m%n
```

File log4j Configuration

```
# Levels, in order, are:
#  [ALL], TRACE, DEBUG, INFO, WARN, ERROR, FATAL, [OFF]

log4j.rootLogger=ALL, MyConsole, MyFile

log4j.appender.MyConsole=org.apache.log4j.ConsoleAppender
log4j.appender.MyConsole.threshold=ALL
log4j.appender.MyConsole.layout=\
        org.apache.log4j.PatternLayout
log4j.appender.MyConsole.layout.ConversionPattern=\
        %l: %-4r [%t] %-5p %c %x - %m%n

log4j.appender.MyFile=org.apache.log4j.FileAppender
log4j.appender.MyFile.file=my_log_file.log
log4j.appender.MyFile.append=FALSE
log4j.appender.MyFile.threshold=ALL
log4j.appender.MyFile.layout=org.apache.log4j.PatternLayout
log4j.appender.MyFile.layout.ConversionPattern=\
                %l: %-4r [%t] %-5p %c %x - %m%n
```

Complex log4j Configuration

```
# Levels, in order, are:
#  [ALL], TRACE, DEBUG, INFO, WARN, ERROR, FATAL, [OFF]

log4j.rootLogger=FATAL, MyConsole, MyFile
log4j.logger.project.sample.Calculator=ALL
#log4j.logger.project.controller=DEBUG

log4j.appender.MyConsole=org.apache.log4j.ConsoleAppender
log4j.appender.MyConsole.threshold=ALL
log4j.appender.MyConsole.layout=\
        org.apache.log4j.PatternLayout
log4j.appender.MyConsole.layout.ConversionPattern=\
        %l: %-4r [%t] %-5p %c %x - %m%n

log4j.appender.MyFile=org.apache.log4j.FileAppender
log4j.appender.MyFile.file=my_file_log.log
log4j.appender.MyFile.append=FALSE
log4j.appender.MyFile.threshold=ALL
log4j.appender.MyFile.layout=org.apache.log4j.PatternLayout
log4j.appender.MyFile.layout.ConversionPattern=\
                %l: %-4r [%t] %-5p %c %x - %m%n
```