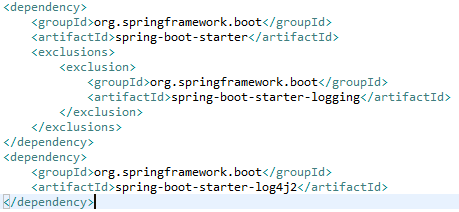
Lesson 16

* Developers are human, and we aren’t perfect, in addition to our fallible nature, technology is also not perfect, and when we look at these two factors together it is understandable why sometimes, even commonly, software runs in to errors. When errors or exceptions occur, we need to be able to troubleshoot and figure out what caused the error, so we can fix it and stop it from occurring in the future. To be able to accomplish this, we need to be able to see information about the error, such as what area of the code it was caused in, what data was being processed, and the stack trace of the exception. Up to now, we’ve been able to simply look at our console to see this type of information, but when we deploy an application to a server, and it’s accessible to the public, we can’t just have someone looking at the console 24/7, we need a way to be able to see the output of the console at any time. This is where logging comes in.
  + Logging is exactly what it sounds like, we log out data, info, exceptions, errors, and really just any useful information to files. That way, when an issue arises, we can look through the logs to start troubleshooting what happened and look for a fix.
  + There are many different logging utilities, but the one we will be using is Apache’s Log4j2. I will just call it log4j throughout this video without referencing to the version 2 for brevity.
  + Log4j allows us to configure our logging requirements via an XML file, and then we can use loggers throughout our code where we deem necessary in the same manner we’ve used system out printline in the past. As a rule of thumb, you should never push code that has any system out printline calls, always use loggers.
  + Let’s look at how we can configure and use log4j.
  + First, we need to exclude springboots logging utility in our pom.xml, then we can add log4j as a dependency
  + Next, we need to configure our loggers in our resource directory. Create a file named log4j2.xml
* <?xml version=*"1.0"* encoding=*"UTF-8"*?>
* <Configuration status=*"WARN"* monitorInterval=*"30"*>
* <Properties>
* <Property name=*"LOG\_PATTERN"*>
* %d{yyyy-MM-dd HH:mm:ss.SSS} %5p ${hostName}
* --- [%15.15t] %-40.40c{1.} : %m%n%ex
* </Property>
* </Properties>
* <Appenders>
* <Console name=*"ConsoleAppender"* target=*"SYSTEM\_OUT"*
* follow=*"true"*>
* <PatternLayout pattern=*"${LOG\_PATTERN}"* />
* </Console>
* <!-- Rolling File Appender -->
* <RollingFile name=*"FileAppender"*
* fileName=*"logs/log4j2-demo.log"*
* filePattern=*"logs/log4j2-demo-%d{yyyy-MM-dd}-%i.log"*>
* <PatternLayout>
* <Pattern>${LOG\_PATTERN}</Pattern>
* </PatternLayout>
* <Policies>
* <SizeBasedTriggeringPolicy size=*"10MB"* />
* </Policies>
* <DefaultRolloverStrategy max=*"10"* />
* </RollingFile>
* </Appenders>
* <Loggers>
* <Logger name=*"com.example.log4j2demo"* level=*"debug"*
* additivity=*"false"*>
* <AppenderRef ref=*"ConsoleAppender"* />
* </Logger>
* <Root level=*"info"*>
* <AppenderRef ref=*"ConsoleAppender"* />
* <AppenderRef ref=*"FileAppender"* />
* </Root>
* </Loggers>
* </Configuration>
  + Then, we need to add a couple imports to any class where we want to utilize our loggers:

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

* + - **import** org.apache.logging.log4j.Logger;
  + We can then create a class level instance of the logger in each class we want to use the logger:
    - **private** **static** **final** Logger ***logger*** = LogManager.*getLogger*(FriendsController.**class**);
  + And finally, we can log anywhere we want now with this logger:

***logger***.debug("Debugging log");

***logger***.info("Info log");

***logger***.warn("Hey, This is a warning!");

***logger***.error("Oops! We have an Error. OK");

* + - ***logger***.fatal("Fatal error. Please fix me.");
  + Note that there are different levels of logging, debug, info, warn, error, and fatal. Each level increasing in severity can be used to denote different levels of occurrences. Also, we can configure which levels get printed to our logs in our log4j.xml configuration file. If we set it to debug, then everything will be printed, but if we set it to warn, only warn, error, and fatal will be logged. Whatever we set the configuration to, the logger will log that level and anything higher than that level. This level can be set in the loggers elemnets for each logger in the level attribute.