Logging

Logging records events that occur during software execution.

- As users execute programs on the client side, the system accumulates log entries for support teams.
- In general, it allows for develoeprs to access information about applications to which they do not have direct access.
- Without logs, we would have no idea of knowing what went wrong when application crashes, or have a way to track and monitor application performance.

Persistence

Log4J

- a logging framework, like Log4J, is critical because it allows us to use various logging levels
    - and also allows us to configure a threshold to determine which messages will be recorded in application log files.
- Reliable, fast, and flexible
- Immensely helpful for debugging

Log Levels

They only have representational meaning. Think of them like HTTP Status Codes - they are only telling us or giving us information.

ALL => All levels
DEBUG => designates fine-grained informational events that are most useful to debug an application
INFO => informational messages that highlight progress of the application (at a coarse-grained level)
WARN => designates potentially harmful situations
ERROR => designates error events that might still allow the application to continue running
FATAL => severe error evenets that presumably lead the aplication to abort
OFF => highest possible level, inteded to turn off logging

Process of adding Log4J to our project

1. Add log4j dependency to our pom.xml
2. Configure/Create our log4j2.xml file
3. Create a Logger Class - this class is responsible for the actual act of logging - it will interact with your code.
   3 1/2: Add a public static final Logger Object - use `LogManager.getLogger()`
4. Use that Logger wherever we want.