latex input: mmd-article-header Title: Java Libraries and Frameworks Notes Author: Ethan C. Petuchowski Base Header Level: 1 latex mode: memoir Keywords: Java, programming language, syntax, fundamentals CSS: http://fletcherpenney.net/css/document.css xhtml header: copyright: 2014 Ethan Petuchowski latex input: mmd-natbib-plain latex input: mmd-article-begin-doc latex footer: mmd-memoir-footer

JUnit

5/20/14

JUnit Homepage, JUnit Wikipedia

- Simple framework for writing repeatable tests
- Uses xUnit unit-testing architecture

From the Github Wiki & Tutorial

- There are various assertions you can make like
    - `testAssertEquals(msg, input, expected)`
    - `testAssertNotNull(msg, input)`
    - `assertThat(input, both(containsString("a")).and(containsString("b")))`
    - `assertThat(input, anyOf(equalTo("bad"), equalTo("good"))`
- There are provided abstractions for [**test fixtures**][]

Logging

Java Standard Logger

1. Has different settable *levels*
    1. SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST
2. Can be passed to a custom formatter (e.g. for HTML)
3. An app can have multiple loggers
4. By default logging configuration is done in a configuration file

Simple Usage

For example executing

```
Logger.getGlobal().info("\nFile->Open menu item selected");
```

would print

```
May 10, 2013 10: 12: 15 PM LoggingImageViewer fileOpen INFO:
File-> Open menu item selected
```

Calling

```
Logger.getGlobal().setLevel(Level.OFF);
```

before executing any log statements from the Global Logger would suppress all logging.

Next Level

Define your own logger

```
private static final Logger myLogger = Logger.getLogger("com.mycompa
```

By default, the log level is "INFO" (prints top 3 levels), change it with

```
logger.setLevel(Level.FINE);
```

You log at a particular level with

```
logger.warning(moreImportantMessage);
logger.fine(lessImportantMessage);
logger.log(Level.FINE, lessImportantMessageAgain);
```

There's more to it, but I've never wished I knew anything more than what's already written above.

Log4J-2

Components

1. **logger** --- logs the messages
2. **appender** --- publishes messages to outputs
    1. **RollingFileAppender** --- the basic appender that writes log messages to *files* following a configured rollover (??) strategy
3. **layout** --- formats log messages

Configuration

Configure *components* [above] in `log4j2.xml` in `src/main/resources`.

Possible log-levels are (most to least logging)

```
ALL, DEBUG, INFO, WARN, ERROR, FATAL, OFF
```

Here is an example (tested) configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- if you turn this to ALL you'll get a LOT of msgs from random cr
<Configuration status="WARN">
    <Appenders>
        <Console name="CONSOLE" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %
        </Console>
    </Appenders>
    <Loggers>
        <Root level="error">
            <AppenderRef ref="Console" />
        </Root>
        <!-- NOTE! name="full.package.name" -->
        <logger name="p2p" level="ALL" />
        <Root level="ERROR">
            <AppenderRef ref="CONSOLE"/>
        </Root>
    </Loggers>
</Configuration>
```

Usage

Make a field in the class like

```
static final Logger log = LogManager.getLogger(MyClassName.class.get
```

Then use it like

```
log.info("I've come back now to release this info")
log.error("brah, you messed up.")
```