## Logger in Java

* **Java Logging** API was introduced in 1.4
* You can use java logging API to log application messages.

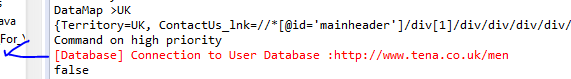
java.util.logging.Logger is the class used to log application messages in java logging API.

* [**LOGGERS**](http://www.loggly.com/ultimate-guide/logging/java-logging-basics/#loggers)**:** LOGGERS ARE RESPONSIBLE FOR CAPTURING EVENTS AND PASSING THEM TO THE APPROPRIATE APPENDER.
* [**APPENDERS**](http://www.loggly.com/ultimate-guide/logging/java-logging-basics/#appenders)**:** ALSO KNOWN AS HANDLERS, APPENDERS ARE RESPONSIBLE FOR RECORDING LOG EVENTS TO A DESTINATION. APPENDERS USE LAYOUTS TO FORMAT EVENTS BEFORE SENDING THEM TO AN OUTPUT.
* [**LAYOUTS**](http://www.loggly.com/ultimate-guide/logging/java-logging-basics/#layouts)**:** ALSO KNOWN AS FORMATTERS, LAYOUTS ARE RESPONSIBLE FOR CONVERTING AND FORMATTING THE DATA IN A LOG EVENT. LAYOUTS DETERMINE HOW THE DATA LOOKS WHEN IT APPEARS IN A LOG ENTRY.

**Example**

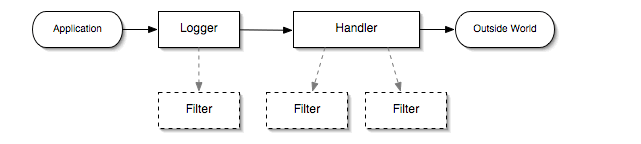
**import java.util.logging.Logger;**

**log.info("[Database] Connection to User database :http://www.tena.co.uk/men");**



When a Logger records an event, it forwards it to the appropriate Appender. The Appender then formats the log entry using a Layout before sending it to the console, to a file, or to another destination.

In addition, Filters let you further specify whether an Appender should be used for a specific log entry. Filters aren’t required in a logging configuration, but they give you greater control over the flow of your log messages.



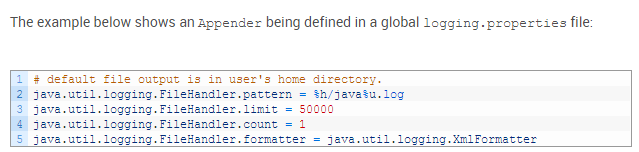
## Logging Frameworks

These frameworks provide the objects, methods, and configuration necessary to transmit log messages. Java provides a default framework in the java.util.logging package. There are also many third-party frameworks including [Log4j](http://logging.apache.org/log4j), [Logback](http://logback.qos.ch/), and[tinylog](http://www.tinylog.org/). Other packages, such as [SLF4J](http://www.slf4j.org/) and [Apache Commons Logging](http://commons.apache.org/proper/commons-logging/), provide abstraction layers which decouple your code from the underlying logging framework, allowing you to switch between logging frameworks.

## Configuration

### java.util.logging

The default Java logging framework stores its configuration in a file called logging.properties. Settings are stored per line using a dot notation format. Java installs a global configuration file in the lib folder of the Java installation directory, although you can use a separate configuration file by specifying the java.util.logging.config.file property when starting a Java program.



### Log4j

[Version 1](https://logging.apache.org/log4j/1.2/manual.html#Configuration) of Log4j uses a syntax similar to java.util.logging. Programs that use Log4j will search for a log4j.properties file in the project directory. By default, Log4j contains a default configuration that outputs all log messages to the console. Log4j also supports an XML configuration syntax, which is defined in a log4j.xml file.

[Version 2](http://logging.apache.org/log4j/2.x/manual/configuration.html) of Log4j supports XML, JSON, and YAML configuration through the log4j2.xml,log4j2.json, and log4j2.yaml files. Similar to version 1, version 2 will search for these files in the project directory. You can find configuration examples in each version’s documentation.

### Logback

Most Logback configuration is done through the [logback.xml](http://logback.qos.ch/manual/configuration.html) file, which uses an XML syntax similar to Log4j. Logback also supports configuration through the Groovy programming language using a [logback.groovy](http://logback.qos.ch/manual/groovy.html) file.

## Loggers

Loggers are the objects that trigger log events. Loggers are created and called in the code of your Java application, where they generate events before passing them to an Appender. A class can have multiple independent Loggers responding to different events, and you can nest Loggersunder other Loggers to create a [hierarchy](http://tutorials.jenkov.com/java-logging/logger-hierarchy.html).

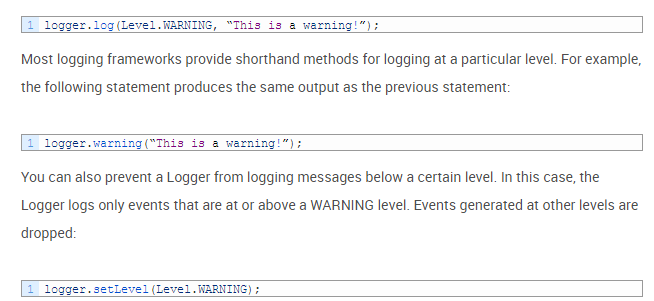
### Creating a New Logger

The process of creating a new Logger is similar across logging frameworks, although the exact method names may be different. With java.util.logging, you create a new Logger usingLogger.getLogger(). getLogger() takes a string parameter that identifies the name of aLogger. If a Logger with that name already exists, then that Logger is returned; otherwise, a newLogger is created. It’s generally good practice to name a new Logger after the current class usingclass.getName():



### Logging Events

Loggers provide several methods for triggering log events. However, before you can log an event, you need to assign a level. Log levels determine the severity of the log and can be used to filter the event or send it to a different Appender



**Example**

**import java.util.logging.Logger;**

**log.info("[Database] Connection to User database :http://www.tena.co.uk/men");**