Logging

YURIY MISCHERYAKOV

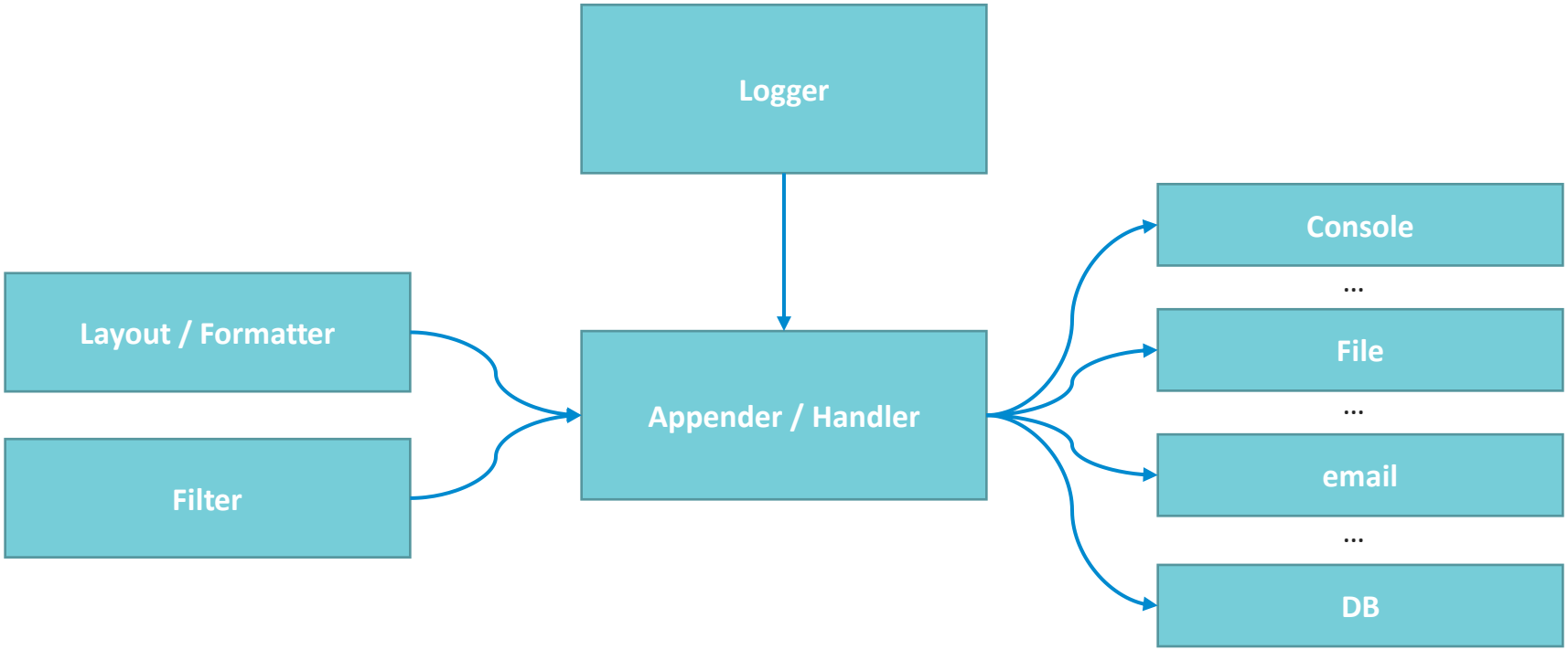
Agenda

- Concept of logging.
- Java Util Logger
- Log4j framework architecture:
    - Loggers,
    - Level,
    - Appenders,
    - Layouts.
- Logging system configuration
- Logging facades

CONCEPT OF LOGGING.

Logging Frameworks

- Some Logging Frameworks
  - Java Logging Framework (java.util.logging, part of the JDK)
  - Apache Log4J
  - LogBack
- Some Logging Facades
  - SLF4J (for LogBack, but has adapters for other frameworks)
  - Apache Commons Logging (July 2014)


Logger
Layout / Formatter
Filter
Appender / Handler
Console
...
File
...
email
...
DB

Loggers hierarchy

- Loggers are organized into a naming hierarchy based on their dot separated names.
- The Logger class named "com.foo" is a parent of the Logger class named "com.foo.Bar". Similarly, "java" is a parent of "java.util" and an ancestor of "java.util.Vector".
    - This naming scheme should be familiar to most developers.
- The root Logger class resides at the top of the Logger class hierarchy. It is exceptional in that it always exists, and it is part of every hierarchy.

```
Logger logger  = Logger.getLogger(""); // root logger
Logger logger1 = Logger.getLogger("com");
Logger logger2 = Logger.getLogger("com.epam");
Logger logger3 = Logger.getLogger("com.epam.training");
```

Assign logger level

Logger Name	Assigned Config	Config Level	level
root	root	DEBUG	DEBUG
X	X	ERROR	ERROR
X.Y	X.Y	INFO	INFO
X.Z	---	---	ERROR
A.B	---	---	DEBUG

JAVA UTIL LOGGER

Java Logging Framework

CONFIG LEVELS

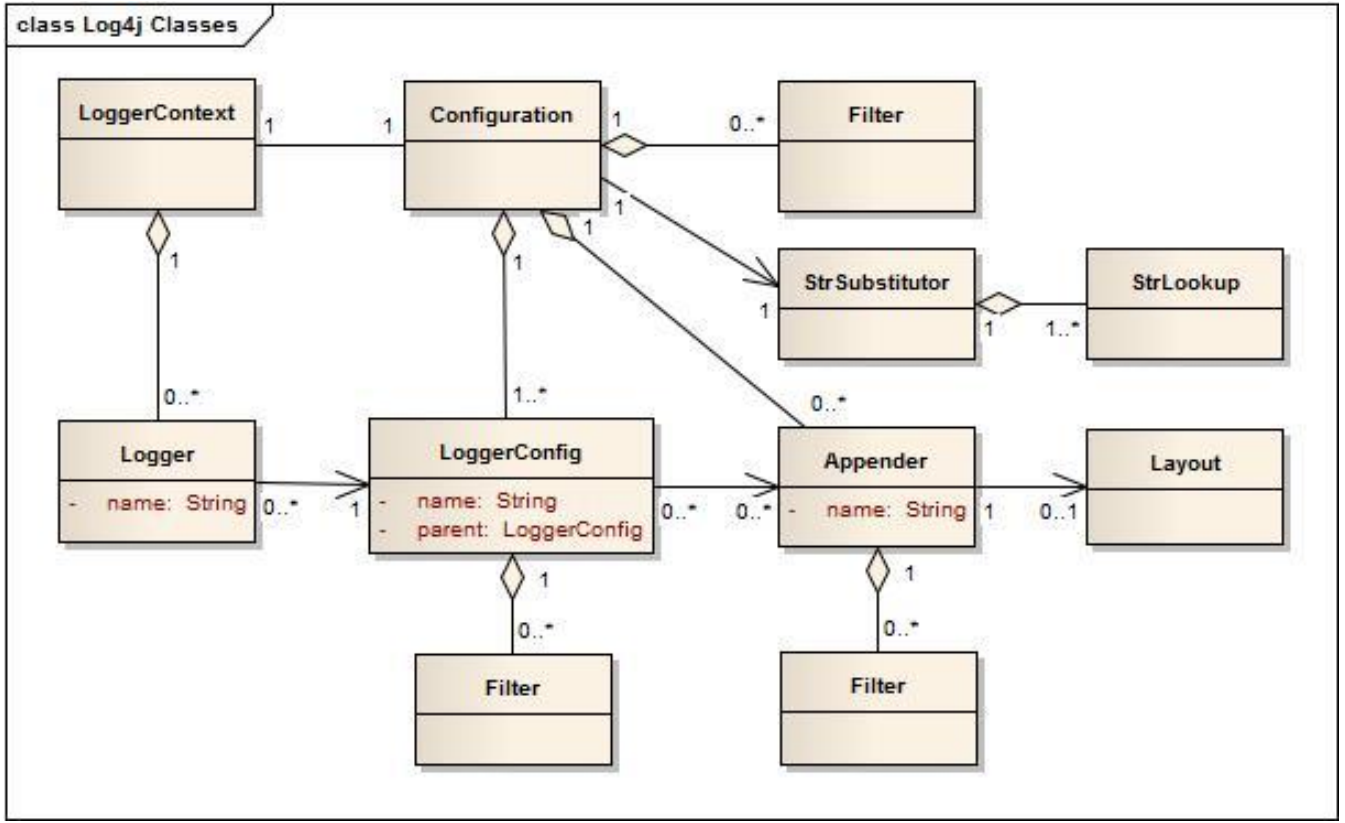
- ALL — Turn on all logging
- FINEST — Most detailed information
- FINER — More detailed information
- FINE — Detailed information on the flow through the system
- CONFIG — Static configuration
- INFO — Interesting runtime events
- WARNING — Potentially harmful situations
- SEVERE — Severe errors that cause premature termination
- OFF — Turn off all logging

Configuration

- By default, the LogManager reads its initial configuration from a properties file "lib/logging.properties" in the JRE directory. If you edit that property file you can change the default logging configuration for all uses of that JRE.
- In addition, the LogManager uses two optional system properties that allow more control over reading the initial configuration:
  - "java.util.logging.config.class"
  - "java.util.logging.config.file“
- The alternate configuration class can use `readConfiguration(InputStream)` to define properties in the `LogManager`.

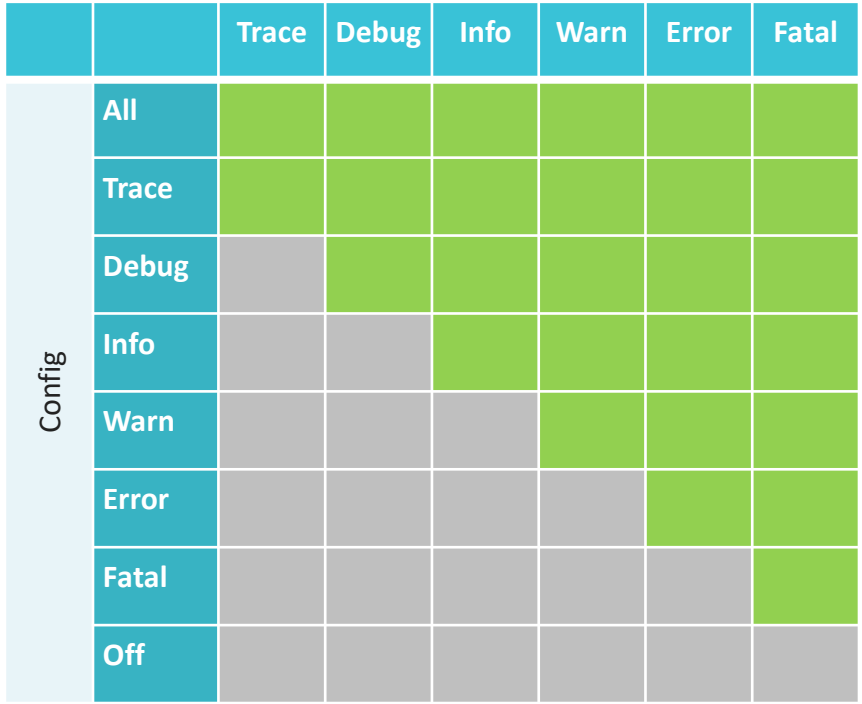
LOG4J

Log4j Classes



Log4j2 Levels

ALL	Turn on all logging
TRACE	Most detailed information
DEBUG	Detailed information on the flow through the system
INFO	Interesting runtime events
WARN	Potentially harmful situations
ERROR	Other runtime errors or unexpected conditions
FATAL	Severe errors that cause premature termination
OFF	Turn off all logging



Log4j Configuration

- Through a configuration file written in XML, JSON, YAML, or properties format.
  - log4j2.properties
  - log4j2.yaml
  - log4j2.json
  - log4j2.xml

Searching

- Programmatically, by creating a ConfigurationFactory and Configuration implementation.
- Programmatically, by calling the APIs exposed in the Configuration interface to add components to the default configuration.
- Programmatically, by calling methods on the internal Logger class.

Appenders

- Async
- Cassandra
- Console
- Failover
- File
- Flume
- JDBC
- JMS
- JPA
- HTTP
- Kafka
- Memory Mapped File
- NoSQL
- NoSQL for MongoDB
- NoSQL for MongoDB 2
- NoSQL for MongoDB 3
- NoSQL for CouchDB
- Output Stream
- Random Access File
- Rewrite
- Rolling File
- Rolling Random Access File
- Routing
- SMTP
- ScriptAppenderSelector
- Socket
- SSL
- Syslog
- ZeroMQ/JeroMQ
- …

Console Appender

Parameter Name	Type	Description
filter	Filter	A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter.
layout	Layout	The Layout to use to format the LogEvent. If no layout is supplied the default pattern layout of "%m%n" will be used.
follow	boolean	Identifies whether the appender honors reassignments of System.out or System.err via System.setOut or System.setErr made after configuration. Note that the follow attribute cannot be used with Jansi on Windows. Cannot be used with direct.

Console Appender

Parameter Name	Type	Description
direct	boolean	Write directly to java.io.FileDescriptor and bypass java.lang.System.out/.err. Can give up to 10x performance boost when the output is redirected to file or other process. Cannot be used with Jansi on Windows. Cannot be used with follow. Output will not respect java.lang.System.setOut()/.setErr() and may get intertwined with other output to java.lang.System.out/.err in a multi-threaded application. New since 2.6.2. Be aware that this is a new addition, and it has only been tested with Oracle JVM on Linux and Windows so far.
name	String	The name of the Appender.

Console Appender

Parameter Name	Type	Description
ignoreExceptions	boolean	The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender.
target	String	Either "SYSTEM_OUT" or "SYSTEM_ERR". The default is "SYSTEM_OUT".

File Appender

Parameter Name	Type	Description
append	boolean	When true - the default, records will be appended to the end of the file. When set to false, the file will be cleared before new records are written.
bufferedIO	boolean	When true - the default, records will be written to a buffer and the data will be written to disk when the buffer is full or, if immediateFlush is set, when the record is written. File locking cannot be used with bufferedIO. Performance tests have shown that using buffered I/O significantly improves performance, even if immediateFlush is enabled.

File Appender

Parameter Name	Type	Description
bufferSize	int	When bufferedIO is true, this is the buffer size, the default is 8192 bytes.
createOnDemand	boolean	The appender creates the file on-demand. The appender only creates the file when a log event passes all filters and is routed to this appender. Defaults to false.
filter	Filter	A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter.
fileName	String	The name of the file to write to. If the file, or any of its parent directories, do not exist, they will be created.

File Appender

Parameter Name	Type	Description
immediateFlush	boolean	When set to true - the default, each write will be followed by a flush. This will guarantee the data is written to disk but could impact performance. Flushing after every write is only useful when using this appender with synchronous loggers. Asynchronous loggers and appenders will automatically flush at the end of a batch of events, even if immediateFlush is set to false. This also guarantees the data is written to disk but is more efficient.
layout	Layout	The Layout to use to format the LogEvent. If no layout is supplied the default pattern layout of "%m%n" will be used.

File Appender

Parameter Name	Type	Description
locking	boolean	When set to true, I/O operations will occur only while the file lock is held allowing FileAppenders in multiple JVMs and potentially multiple hosts to write to the same file simultaneously. This will significantly impact performance so should be used carefully. Furthermore, on many systems the file lock is "advisory" meaning that other applications can perform operations on the file without acquiring a lock. The default value is false.
name	String	The name of the Appender.

File Appender

Parameter Name	Type	Description
ignoreExceptions	boolean	The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender.
filePermissions	String	File attribute permissions in POSIX format to apply whenever the file is created. Underlying files system shall support POSIX file attribute view. Examples: rw------- or rw-rw-rw- etc...

File Appender

Parameter Name	Type	Description
fileOwner	String	File owner to define whenever the file is created. Changing file's owner may be restricted for security reason and Operation not permitted IOException thrown. Only processes with an effective user ID equal to the user ID of the file or with appropriate privileges may change the ownership of a file if POSIX_CHOWN_RESTRICTED is in effect for path. Underlying files system shall support file owner attribute view.
fileGroup	String	File group to define whenever the file is created. Underlying files system shall support POSIX file attribute view.

Pattern Layout

Conversion Pattern	Description
c{precision} **logger**{precision}	Outputs the name of the logger that published the logging event. The logger conversion specifier can be optionally followed by precision specifier, which consists of a decimal integer, or a pattern starting with a decimal integer. When the precision specifier is an integer value, it reduces the size of the logger name.
d{pattern} **date**{pattern}	Outputs the date of the logging event. The date conversion specifier may be followed by a set of braces containing a date and time pattern string per SimpleDateFormat.

https://logging.apache.org/log4j/2.x/manual/layouts.html#PatternLayout

Pattern Layout

Conversion Pattern	Description
C{precision} **class**{precision}	Outputs the fully qualified class name of the caller issuing the logging request.
L **line**	Outputs the line number from where the logging request was issued.
M **method**	Outputs the method name where the logging request was issued.
marker	The full name of the marker, including parents, if one is present.

Pattern Layout

Conversion Pattern	Description
t **tn** **thread** **threadName**	Outputs the name of the thread that generated the logging event.
ex|exception|throwable { … }	Outputs the Throwable trace bound to the logging event, by default this will output the full trace as one would normally find with a call to Throwable.printStackTrace().
n	Outputs the platform dependent line separator character or characters.

Pattern Layout

Conversion Pattern	Description
m{nolookups}{ansi} **msg**{nolookups}{ansi} **message**{nolookups}{ansi}	Outputs the application supplied message associated with the logging event.

Location Information

If one of the layouts is configured with a location-related attribute like HTML locationInfo, or one of the patterns %C or %class, %F or %file, %l or %location, %L or %line, %M or %method, Log4j will take a snapshot of the stack, and walk the stack trace to find the location information.

Filters

- Filters allow Log Events to be evaluated to determine if or how they should be published.
- A Filter will be called on one of its filter methods and will return a Result, which is an Enum that has one of 3 values -
    - ACCEPT
    - DENY
    - NEUTRAL

Filters may be configured in one of four locations

- Context-wide Filters are configured directly in the configuration.
  - Events that are rejected by these filters will not be passed to loggers for further processing.
  - Once an event has been accepted by a Context-wide filter it will not be evaluated by any other Context-wide Filters nor will the Logger's Level be used to filter the event.
  - The event will be evaluated by Logger and Appender Filters however.
- Logger Filters are configured on a specified Logger.
  - These are evaluated after the Context-wide Filters and the Log Level for the Logger.
  - Events that are rejected by these filters will be discarded and the event will not be passed to a parent Logger regardless of the additivity setting.
- Appender Filters are used to determine if a specific Appender should handle the formatting and publication of the event.
- Appender Reference Filters are used to determine if a Logger should route the event to an appender.

MarkerFilter

- The MarkerFilter compares the configured Marker value against the Marker that is included in the LogEvent.
- A match occurs when the Marker name matches either the Log Event's Marker or one of its parents.
- Marker Filter Parameters
  - **marker** The name of the Marker to compare.
  - **onMatch** The default value is NEUTRAL.
  - **onMismatch** The default value is DENY.

ThresholdFilter

- This filter returns the onMatch result if the level in the LogEvent is the same or more specific than the configured level and the onMismatch value otherwise.
    - For example, if the ThresholdFilter is configured with Level ERROR and the LogEvent contains Level DEBUG then the onMismatch value will be returned since ERROR events are more specific than DEBUG.
- Threshold Filter Parameters
    - **level** A valid Level name to match on
    - **onMatch** The default value is NEUTRAL.
    - **onMismatch** The default value is DENY.

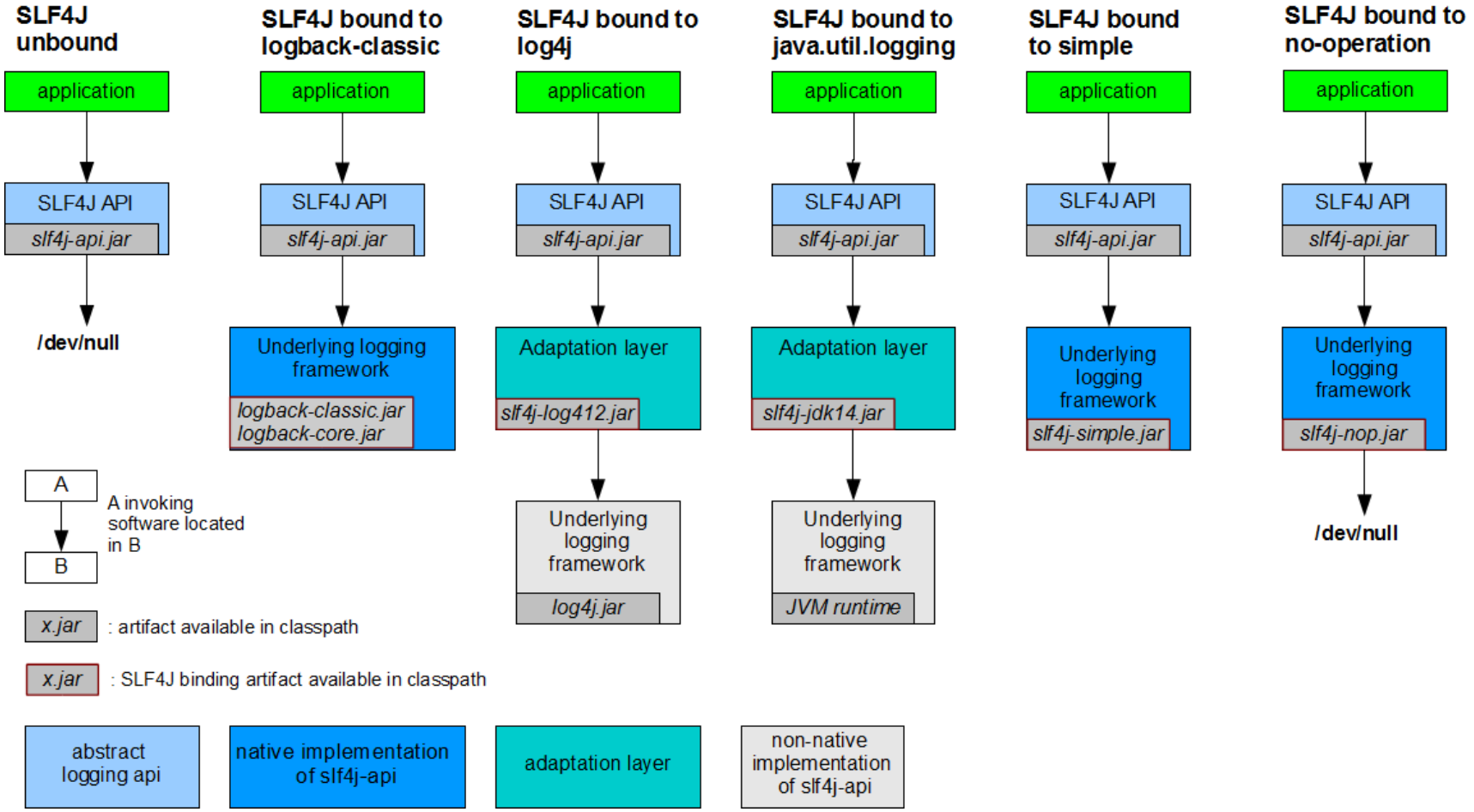
CompositeFilter

- The CompositeFilter provides a way to specify more than one filter.
- It is added to the configuration as a filters element and contains other filters to be evaluated.
- The filters element accepts no parameters.

FACADES

Slf4j

- The Simple Logging Facade for Java (SLF4J) serves as a simple facade or abstraction for various logging frameworks (e.g. java.util.logging, logback, log4j) allowing the end user to plug in the desired logging framework at deployment time.
- Note that SLF4J-enabling your library implies the addition of only a single mandatory dependency, namely slf4j-api.jar. If no binding is found on the class path, then SLF4J will default to a no-operation implementation.
- Often times, a given project will depend on various components which rely on logging APIs other than SLF4J. It is common to find projects depending on a combination of JCL, java.util.logging, log4j and SLF4J. It then becomes desirable to consolidate logging through a single channel. SLF4J caters for this common use-case by providing bridging modules for JCL, java.util.logging and log4j.

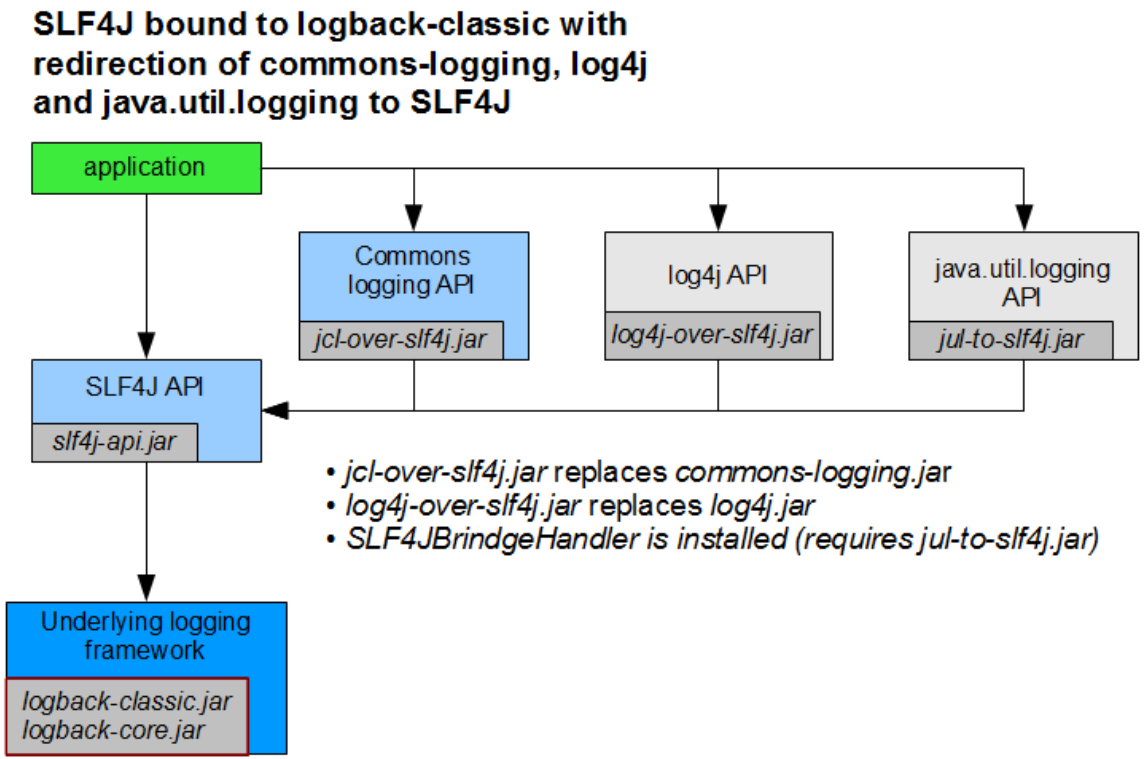

SLF4J unbound
application
SLF4J API
slf4j-api.jar
/dev/null
SLF4J bound to logback-classic
application
SLF4J API
slf4j-api.jar
Underlying logging framework
logback-classic.jar
logback-core.jar
SLF4J bound to log4j
application
SLF4J API
slf4j-api.jar
Adaptation layer
slf4j-log412.jar
Underlying logging framework
log4j.jar
SLF4J bound to java.util.logging
application
SLF4J API
slf4j-api.jar
Adaptation layer
slf4j-jdk14.jar
Underlying logging framework
JVM runtime
SLF4J bound to simple
application
SLF4J API
slf4j-api.jar
Underlying logging framework
slf4j-simple.jar
SLF4J bound to no-operation
application
SLF4J API
slf4j-api.jar
Underlying logging framework
slf4j-nop.jar
/dev/null
A
B
A invoking software located in B
x.jar
: artifact available in classpath
x.jar
: SLF4J binding artifact available in classpath
abstract logging api
native implementation of slf4j-api
adaptation layer
non-native implementation of slf4j-api

Dependencies

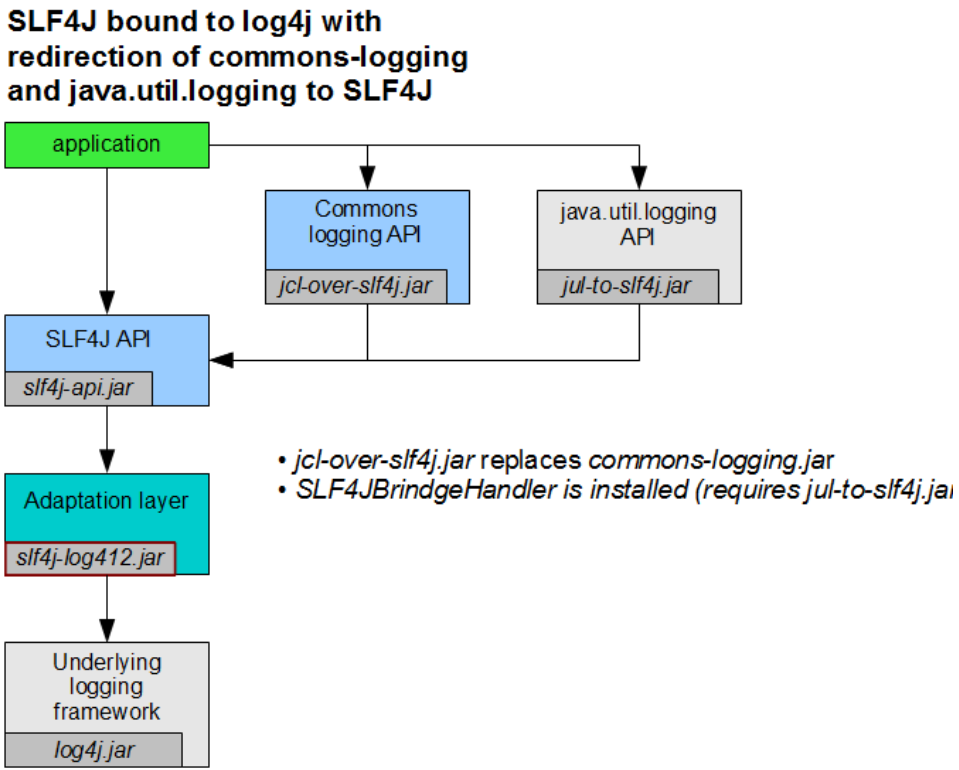
- slf4j
  - <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${slf4j.version}</version>
    </dependency>
- Slf4j over log4j
  - <dependency>

    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-slf4j-impl</artifactId>
    <version>${log4j.version}</version>
    </dependency>
- Slf4j over jul
  - <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-jdk14</artifactId>
    <version>${slf4j.version}</version>
    </dependency>
- Slf4j over logback
  - <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>${logback.version}</version>
    </dependency>

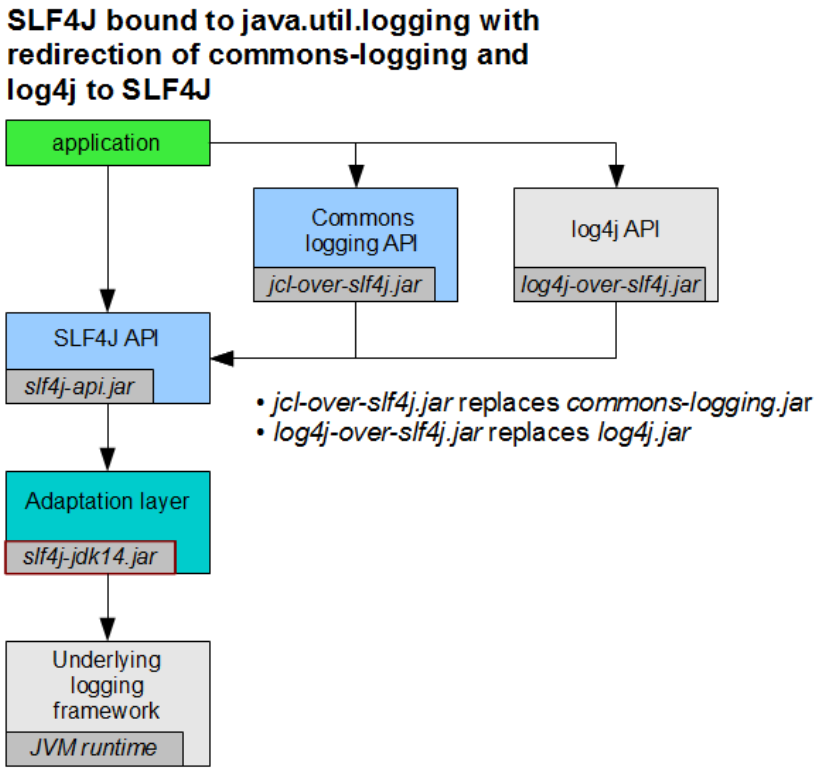
Bridging


SLF4J bound to logback-classic with redirection of commons-logging, log4j and java.util.logging to SLF4J
application
Commons logging API
jcl-over-slf4j.jar
log4j API
log4j-over-slf4j.jar
java.util.logging API
jul-to-slf4j.jar
SLF4J API
slf4j-api.jar
• jcl-over-slf4j.jar replaces commons-logging.jar
• log4j-over-slf4j.jar replaces log4j.jar
• SLF4JBrindgeHandler is installed (requires jul-to-slf4j.jar)
Underlying logging framework
logback-classic.jar
logback-core.jar

Bridging


SLF4J bound to log4j with
redirection of commons-logging
and java.util.logging to SLF4J
application
Commons
logging API
jcl-over-slf4j.jar
java.util.logging
API
jul-to-slf4j.jar
SLF4J API
slf4j-api.jar
Adaptation layer
slf4j-log412.jar
Underlying
logging
framework
log4j.jar
• jcl-over-slf4j.jar replaces commons-logging.jar
• SLF4JBrindgeHandler is installed (requires jul-to-slf4j.ja

Bridging


SLF4J bound to java.util.logging with redirection of commons-logging and log4j to SLF4J
application
Commons logging API
jcl-over-slf4j.jar
log4j API
log4j-over-slf4j.jar
SLF4J API
slf4j-api.jar
• jcl-over-slf4j.jar replaces commons-logging.jar
• log4j-over-slf4j.jar replaces log4j.jar
Adaptation layer
slf4j-jdk14.jar
Underlying logging framework
JVM runtime

Logback configuration

```
<configuration>
   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
      <!-- encoders are assigned the type
           ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
      <encoder>
         <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
      </encoder>
   </appender>
   <root level="debug">
      <appender-ref ref="STDOUT" />
   </root>
   <logger name="com.epam.training" level="INFO" />
</configuration>
```

Tasks

- Add and configure logger. The log information should be displayed in the console and saved to a file.
- Make different appenders for debug and info.
- Configure loggers for recording in the file and make the following options:
    - file will not be overwritten;
    - file will be overwritten every day;
    - file will be overwritten after reaching the size of 1 MB;
    - every day log will be written to a new file.
- Configure logger that all levels higher than “WARN” will be saved in the file.
- Configure logger that in the file will be recorded only “WARN”, and in the console – only “INFO”.
- Configure logger that “ERROR” will be sent on e-mail.