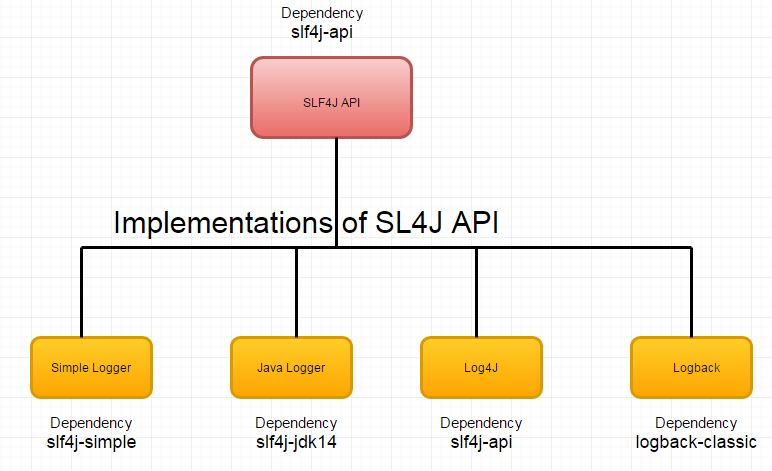
**SL4J (Simple Logging Façade for Java)**

SLF4J provides an abstract logging API, allowing the end-user to plug-in the desired logging framework,

be it logback-classic, log4j, or java.util.logging, at deployment time. SLF4J is only a facade, meaning that it does not provide a complete logging solution. It provides a common interface for various Java loggers. So it is used with actual loggers which actually log the events. Examples of such loggers are Logback, Apache Commons Logging, JDK 1.4 Logging or Log4j. Simple Logging Facade for Java (SLF4J) is an abstraction of different logging frameworks (eg. log4j, java.util.logging, commons logging etc.). This gives the developer an opportunity to plug-in desired logging framework according to the requirement at deployment time without changing the code.



Logger delegates the task of writing logging events to Appenders. Appenders are named references and will be used by logger to connect to appropriate appenders. Appenders utilize the doAppend method to append logs to the destinaton.

The levels supported in SLF4J according to the order of priority are:

* Trace – Lowest level. Used to track the application flow.
* Debug – used to add diagnostic messages of the application. Generally useful for debugging when there is an error.
* Info – Used to indicate important flows of the application.
* Warn – Used to indicate potential error scenarios of the application.
* Error – Used to log errors and exceptions of the application.

First Example:

We will directly dive into a logging example. We will look at the initial setup for our application. Our project uses Maven as the dependency manager and the application’s dependencies will be detailed as and when we encounter them.

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.jcg</groupId>

    <artifactId>slf4j</artifactId>

    <version>1.0-SNAPSHOT</version>

    <dependencies>

        <dependency>

            <groupId>org.slf4j</groupId>

            <artifactId>slf4j-api</artifactId>

            <version>1.7.25</version>

        </dependency>

    </dependencies>

    <build>

        <plugins>

            <plugin>

                <groupId>org.apache.maven.plugins</groupId>

                <artifactId>maven-compiler-plugin</artifactId>

                <version>3.6.1</version>

                <configuration>

                    <source>1.8</source>

                    <target>1.8</target>

                </configuration>

            </plugin>

        </plugins>

    </build>

</project>

* We have specified our artifact as SLF4J in line 8.
* We specify slf4j-api as dependency in lines 13-15. Maven downloads the SLF4J dependency in our classpath to enable the logging interface.
* We provide Java8 as the target compiler for our application.

Single SLF4J dependency is enough for us to get started with logging. Now we will create a simple class for logging.

**i**mport org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.lang.invoke.MethodHandles;

import java.util.stream.IntStream;

public class LoggerRoot {

    private static final Logger logger = LoggerFactory.getLogger(MethodHandles.lookup().lookupClass().getSimpleName());

    public static void main(String... args) {

        IntStream.rangeClosed(1, 10).forEach(counter -> {

            logger.info("Counter:" + counter);

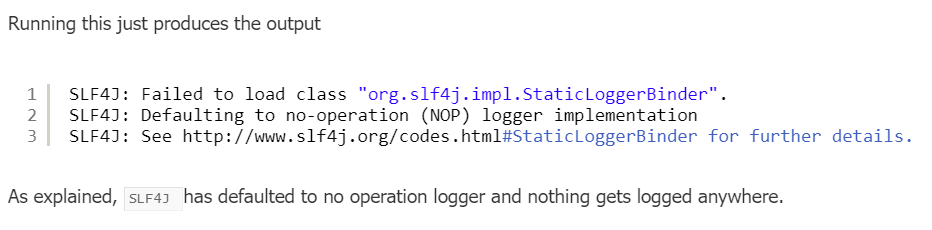
        });

    }

}

We import **LoggerFactory** and Logger from SLF4J for logging use. **LoggerFactory** provides various factory methods to initialize an appropriate logger for the use case. The returned Logger from the factory is used to append the logs in our application.

We have to provide a name for the logger as input to the factory method. This is one of the factory methods available as part of **LoggerFactory**. We use Java Core API to lookup our class name which can be useful to copy and paste in other classes. This is equivalent to LoggerRoot.class.getName() but it mandates that the correct class is specified in java file or loggers would end up with the same name.



**LogBack:**

Now we will provide a logging framework implementation. We will use first java.util.logging. To enable this, We need to include it as a dependency.

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.3.0-alpha4</version>

<exclusions>

<exclusion>

<artifactId>slf4j-api</artifactId>

<groupId>org.slf4j</groupId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-core</artifactId>

<version>1.3.0-alpha4</version>

</dependency>

We have defined logback as the runtime dependency. SLF4J will pick up logback as the logging implementation. We will cover the logback configuration to log the output to the console.

*logback.xml*

<configuration>

    <appender name="console" class="ch.qos.logback.core.ConsoleAppender">

        <target>System.out</target>

        <encoder>

            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

        </encoder>

    </appender>

    <root level="info">

        <appender-ref ref="console"/>

    </root>

</configuration>

* We create a console appender with the target as System.Out.
* W use PatternLayoutEncoder and provide a pattern containing the class name and logger name.
* We specify the level as INFO and tie the appender to root logger.

