1.What is Log4J and how do I use it?

Chapter 1

Log4j is a Java library that specializes in logging. great job covering many of the standard features of Log4j. At its most basic level, you can think of it as a replacement for System.out.println's in your code. Why is it better than System.out.println's? The reasons are numerous.

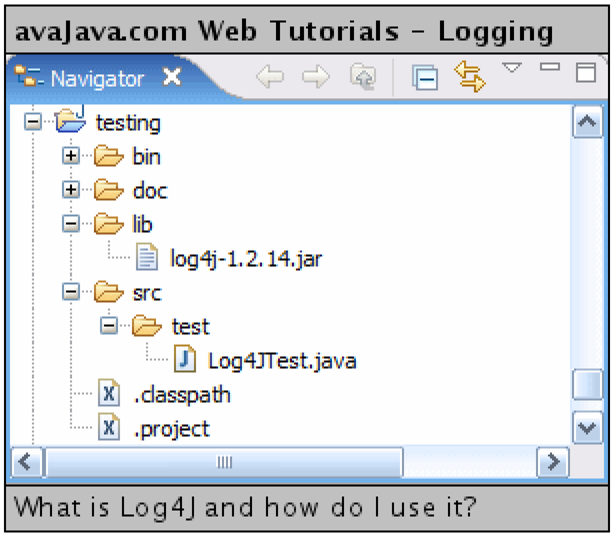
To begin with, System.out.println outputs to standard output, which typically is a console window. The output from Log4j can go to the console, but it can also go to an email server, a databasew table, a log file, or various other destinations.

Another great benefit of Log4j is that different levels of logging can be set. The levels are hierarchical and are as follows: TRACE, DEBUG, INFO, WARN, ERROR, and FATAL. If you set a particular log level, messages will get logged for that level and all levels above it, and not for any log levels below that. As an example, if your log level is set to ERROR, you will log messages that are errors and fatals. If your log level is set to INFO, you will log messages that are infos, warns, errors, and fatals. Typically, when you develop on your local machine, it's good to set the log level to DEBUG, and when you deploy a web application, you should set the log level to INFO or higher so that you don't fill up your error logs with debug messages.

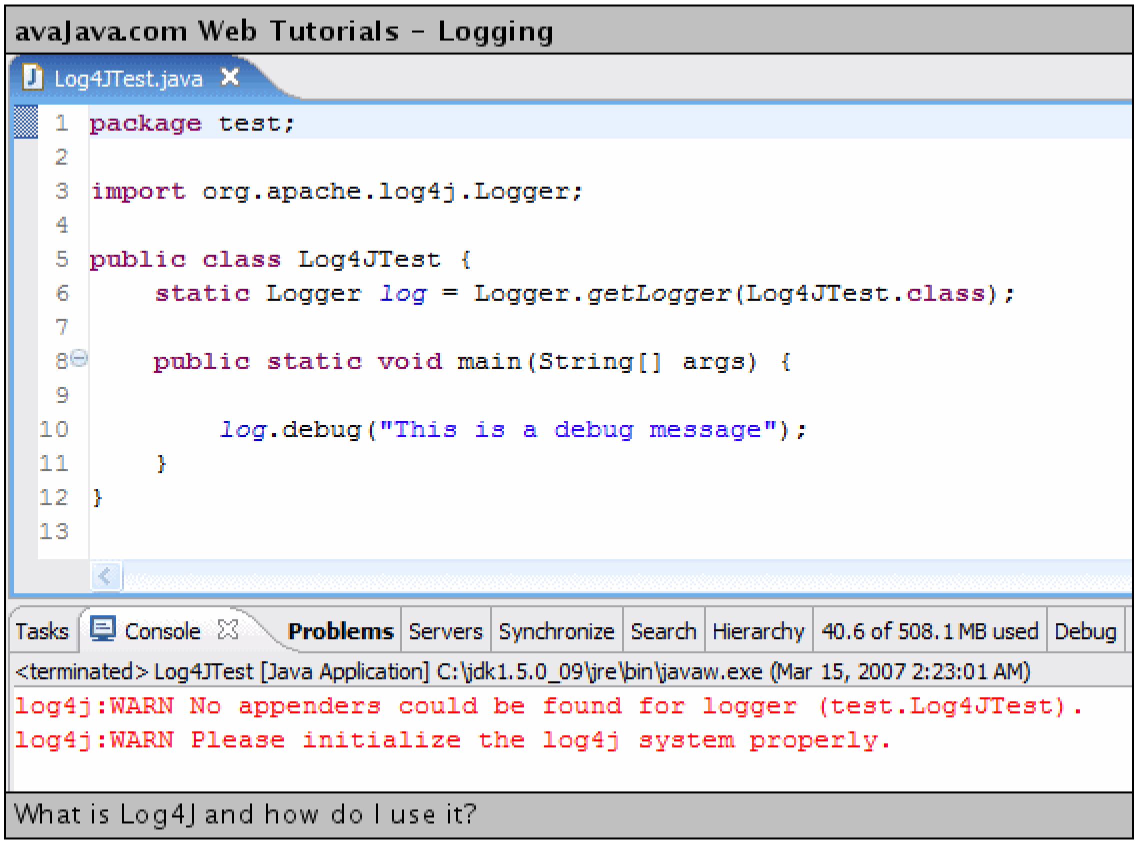
Log4j can do other great things. For instance, you can set levels for particular Java classes, so that if a particular class spits out lots of warnings, you can set the log level for that class to ERROR to suppress all the warning messages.

Typically, you can configure Log4j via a configuration file. This means that you can change your logging configuration without requiring code updates. If you need to do something like set the log level to DEBUG for your deployed application in order to track down a particular bug, you can do this without redeploying your application. Instead, you can change your log configuration file, reread the configuration file, and your logging configuration gets updated.

Let's try out Log4j. First off, we can download the log4j jar file from the address mentioned above. I downloaded the jar file and added it to my project's build path.



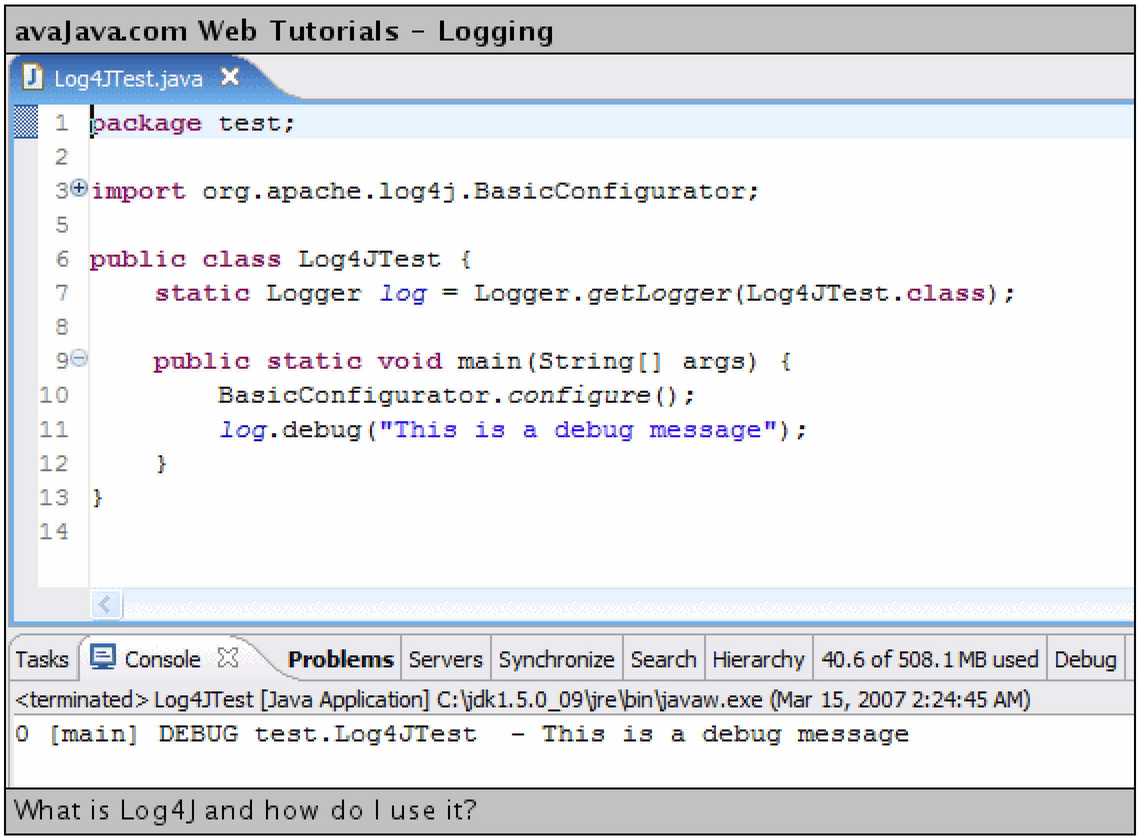
I created a Log4JTest class. In this class, notice that I have a static Logger object called log, and this object is obtained from Logger.getLogger, with the Log4JTest class passed as a parameter to getLogger. This is basically saying that we have a Logger object that we can use in our Log4JTest class, and that the Log4JTest.class is the name that the logger is recognized as. In the main method, we have one log.debug call. Let's see what happens if we try to run our class at this stage.



```java
package test;

import org.apache.log4j.Logger;

public class Log4JTest {
    static Logger log = Logger.getLogger(Log4JTest.class);

    public static void main(String[] args) {

        log.debug("This is a debug message");
    }
}

```

```
<terminated> Log4JTest [Java Application] C:\jdk1.5.0_09\jre\bin\javaw.exe (Mar 15, 2007 2:23:01 AM)
log4j:WARN No appenders could be found for logger (test.Log4JTest).
log4j:WARN Please initialize the log4j system properly.
```

So, does this mean that Log4j doesn't work? Actually, if we read the console output, we can see that the problem is that we need to initialize Log4j. This is typically done via a Configurator. If we were using a properties file to set the logging configuration, we would use the PropertyConfigurator class, but for the sake of simplicity, let's use BasicConfigurator, which gives us a plain vanilla Log4j configuration without requiring any properties to be set. A call to the configure() method initializes Log4j, after which we can use the static Logger object in our class (and any other loggers throughout our application). If we now execute our class, we see the following:



```java
package test;

import org.apache.log4j.BasicConfigurator;

public class Log4JTest {
    static Logger log = Logger.getLogger(Log4JTest.class);

    public static void main(String[] args) {
        BasicConfigurator.configure();
        log.debug("This is a debug message");
    }
}

```

```
<terminated> Log4JTest [Java Application] C:\jdk1.5.0_09\jre\bin\javaw.exe (Mar 15, 2007 2:24:45 AM)
0 [main] DEBUG test.Log4JTest  - This is a debug message
```

Log4j outputs our "This is a debug message" to the console. The number at the front of the line is the number of milliseconds since the application started. The [main] indicates the name of the thread. In addition, the line displays the log level (DEBUG) and the class name that we specified in our Logger.getLogger method call.

Example : 1

```java
package com.sashi;

import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.Logger;

public class Log4j1 {

	static Logger log = Logger.getLogger(Log4j1.class);

	public static void main(String[] args) {

		BasicConfigurator.configure();

		log.debug("This is dubug");
		log.error("This is error");
		log.info("This is info");

		myMethod();

		log.info("End of the Program");
	}

	public static void myMethod() {

		try {
			throw new Exception("My Exception");
		}
		catch(Exception ex) {

			log.error("Exception : " + ex);
		}
	}
}
```

Output:

```
0 [main] DEBUG com.sashi.Log4j1  - This is dubug
1 [main] ERROR com.sashi.Log4j1  - This is error
1 [main] INFO com.sashi.Log4j1  - This is info
1 [main] ERROR com.sashi.Log4j1 - Exception : java.lang.Exception: My Exception
1 [main] INFO com.sashi.Log4j1  - End of the Program
```

Explanation:

We mentioned “Log4j1.class” in the,

```java
	static Logger log = Logger.getLogger(Log4j1.class);
```

so output is printing as , class name with package name,

```
0 [main] DEBUG com.sashi.Log4j1  - This is dubug
```

Example : 2

```
package com.sashi;

public class Log4j1 {

	static Logger log = Logger.getLogger("My Class");

	public static void main(String[] args) {

		BasicConfigurator.configure();

		log.info("Starting of the Program");

		myMethod();

		log.info("End of the Program");
	}

	public static void myMethod() {

		try {
			int i = 10/0;
		}
		catch(ArithmeticException ex) {

			log.error("ArithmeticException is occured" + ex);
		}
	}
}
```

Output:

```
0 [main] INFO My Class  - Starting of the Program
1 [main] ERROR My Class  - ArithmeticException is occuredjava.lang.ArithmeticException: /
by zero
1 [main] INFO My Class  - End of the Program
```

Explanation:

We mentioned “My Class” in the,

```
	static Logger log = Logger.getLogger("My Class");
```

so output is printing as ,

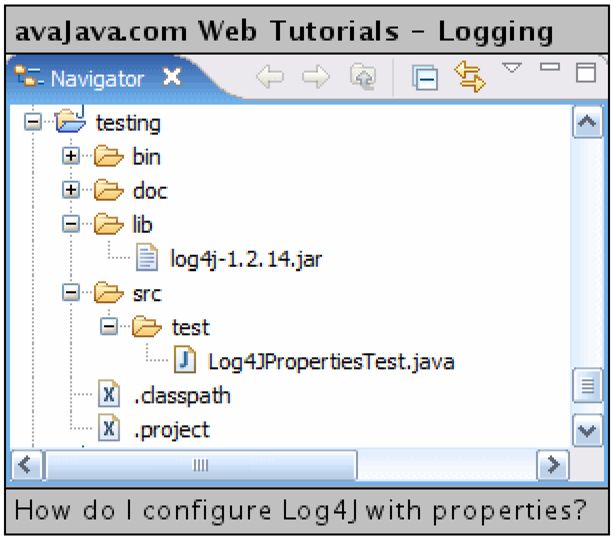
```
   0  [main] INFO My Class  - Starting of the Program
```

2.How do I configure Log4J with properties?

Chapter 1

In an earlier lesson, I described Log4j and initialized logging in a small test application using the BasicConfigurator class. This is a nice starting point, but a much more practical approach is to initialize log4j in an application using properties.

In your project, you need the log4j jarW file in the build path. I created a Log4JPropertiesTest class in this project.

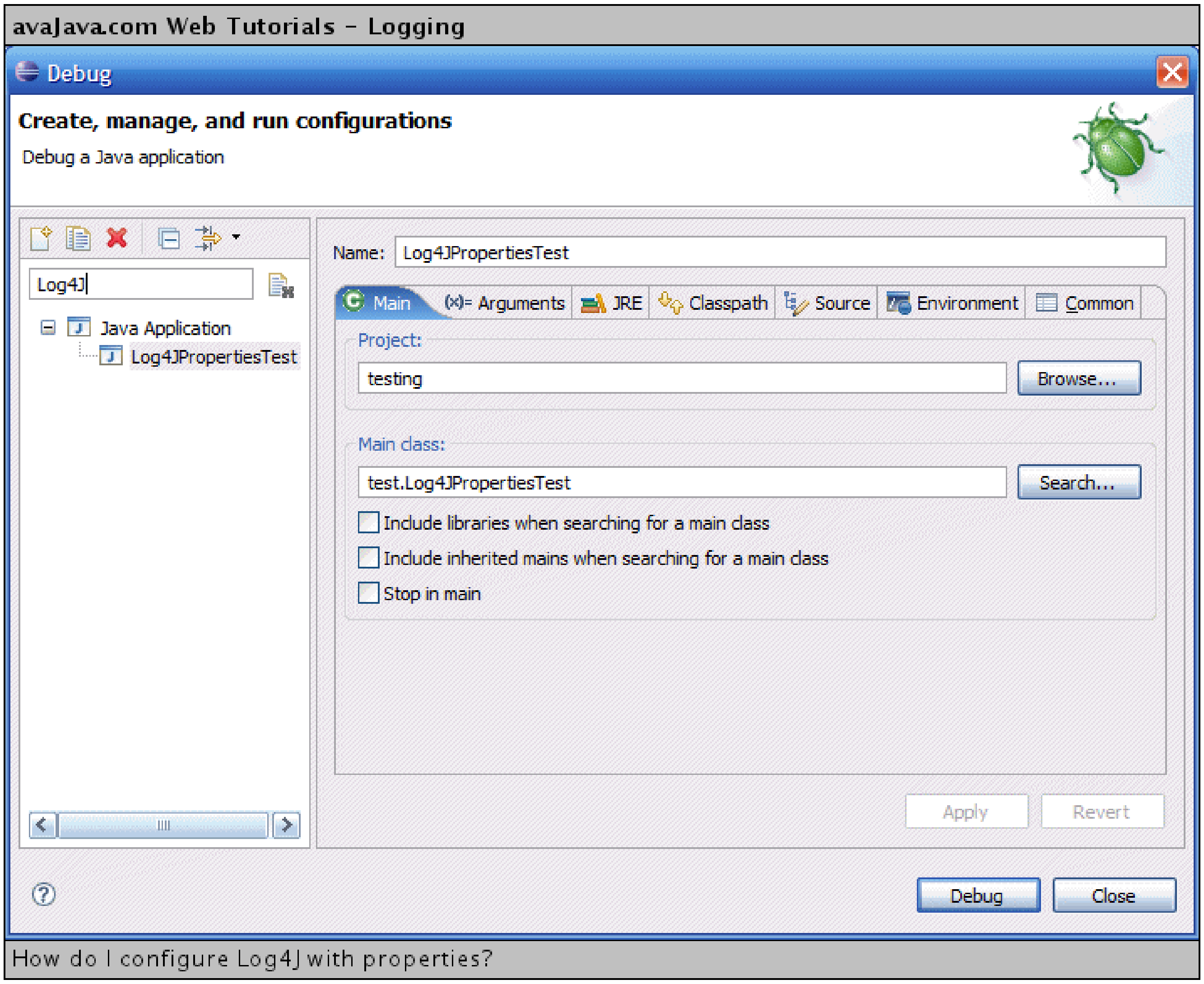


Log4JPropertiesTest is a very simple class that we'll use as a starting point to test out configuring Log4j via properties.

Log4JPropertiesTest.java

```
package test;

public class Log4JPropertiesTest {
	static Logger log = Logger.getLogger(Log4JPropertiesTest.class);

	public static void main(String[] args) {
		log.debug("This is a debug message");
		myMethod();
		log.info("This is an info message");
	}

	private static void myMethod() {
		try {
			throw new Exception("My Exception");
		} catch (Exception e) {
			log.error("This is an exception", e);
		}
	}
}
```

I created an Eclipse Debug Configuration for Log4JPropertiesTest.



Name: Log4JPropertiesTest

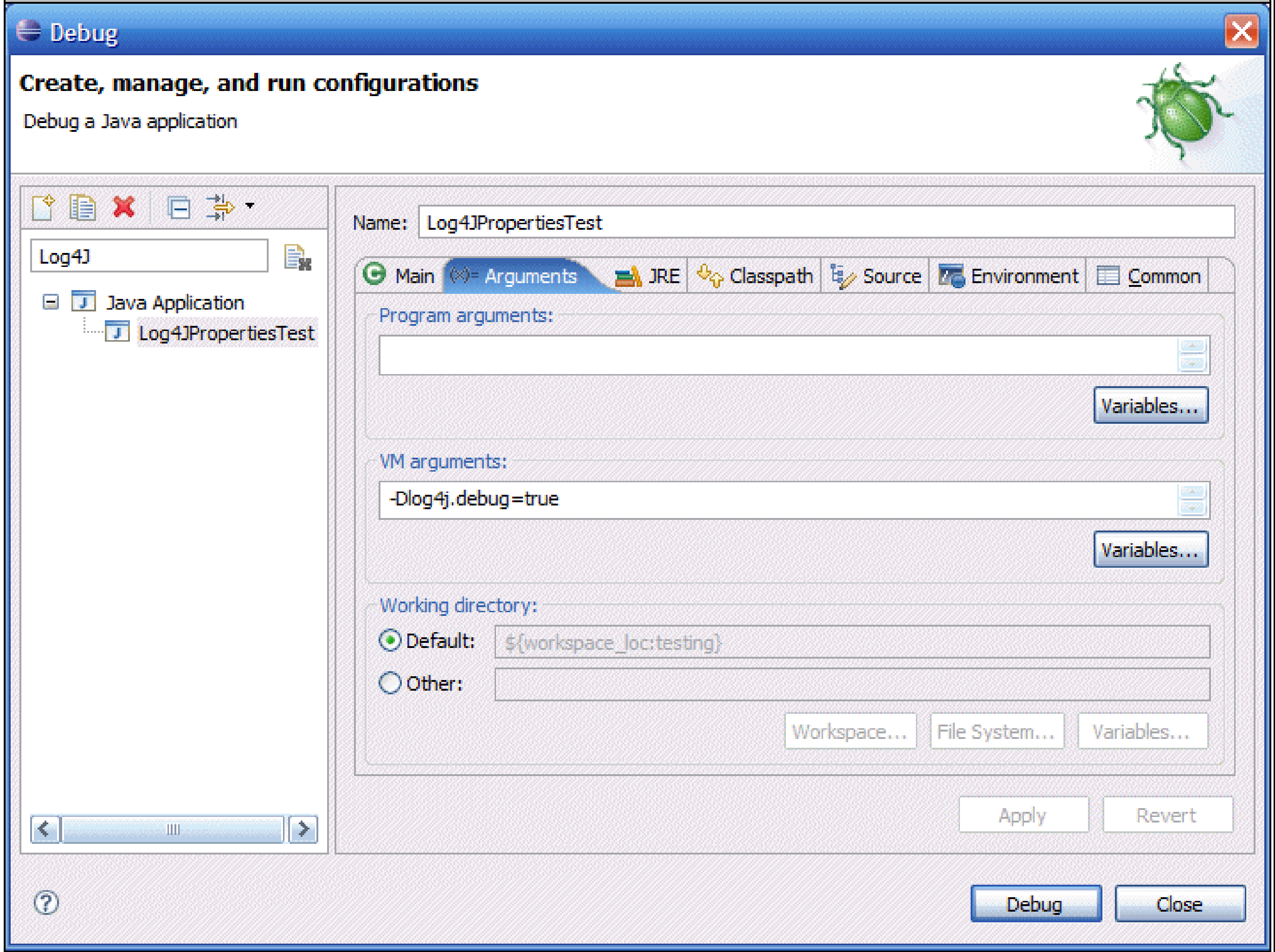
Project: testing

Main class: test.Log4JPropertiesTest

- [ ] Include libraries when searching for a main class
- [ ] Include inherited mains when searching for a main class
- [ ] Stop in main

Since we want to really see what's going on with Log4j, let's add a VM argument instructing log4j to output additional information so we can tell what's going on behind the scenes. This argument is:

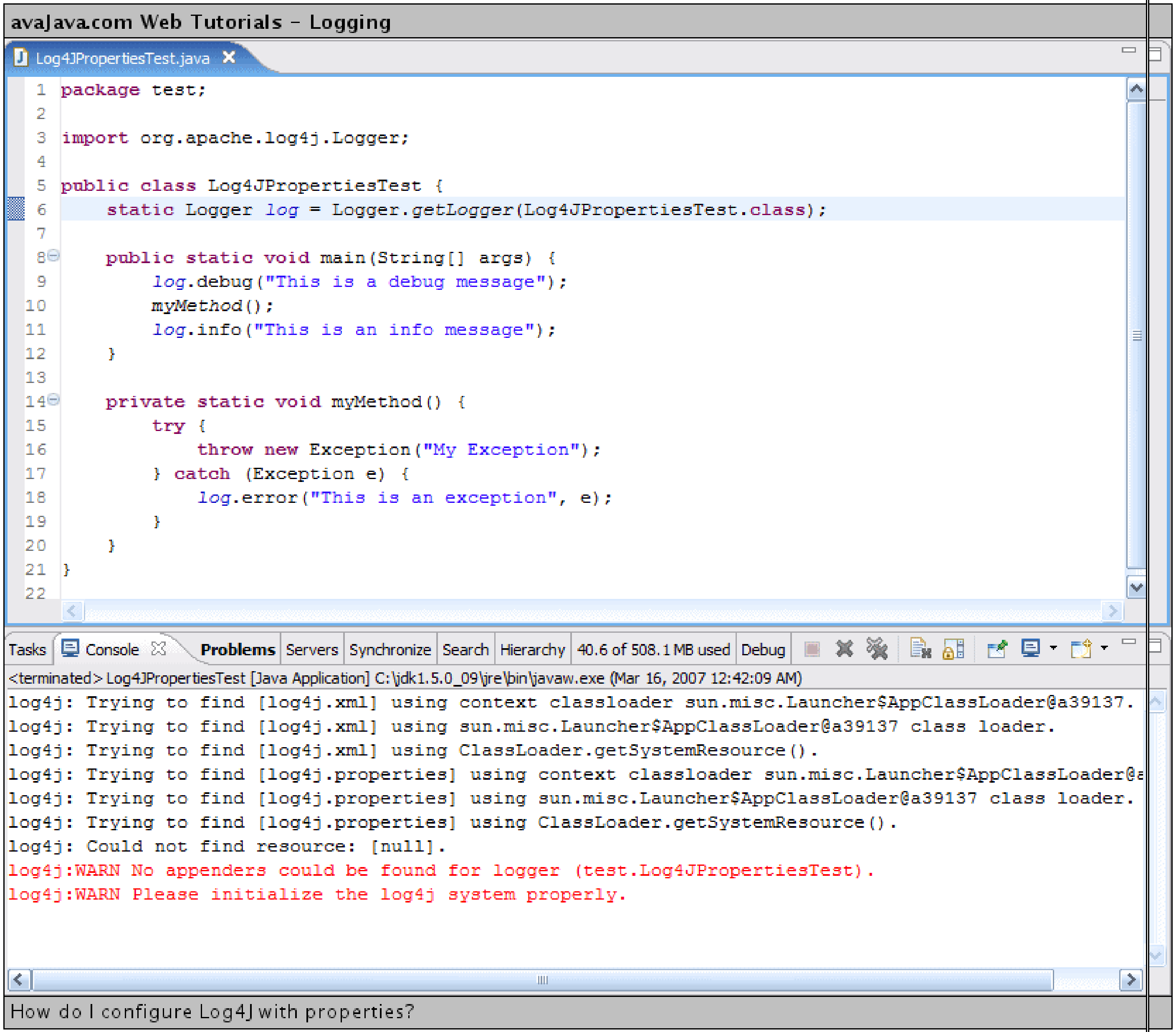
```
-Dlog4j.debug=true
```



How do I configure Log4J with properties?

If we click Debug, we can run our application. In the console window, we can see that log4j searches for a log4j.xml file and then searches for a log4j.properties file. It doesn't find either (since we haven't created either one of them), so it reports:

```
log4j: Could no  t find resource: [null].
```


avaJava.com Web Tutorials - Logging
Log4JPropertiesTest.java
Tasks Console Problems Servers Synchronize Search Hierarchy 40.6 of 508.1 MB used Debug

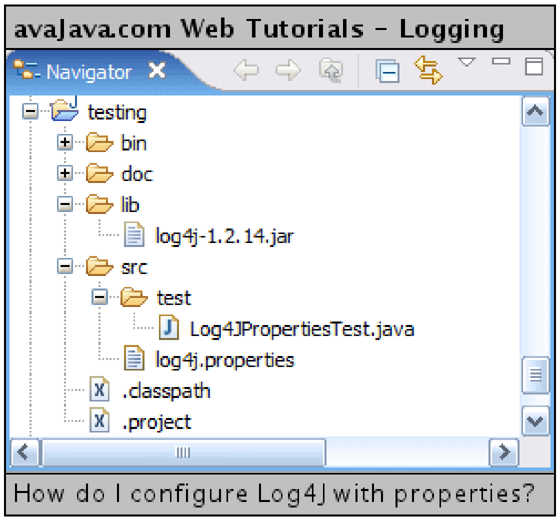
```java
package test;

import org.apache.log4j.Logger;

public class Log4JPropertiesTest {
    static Logger log = Logger.getLogger(Log4JPropertiesTest.class);

    public static void main(String[] args) {
        log.debug("This is a debug message");
        myMethod();
        log.info("This is an info message");
    }

    private static void myMethod() {
        try {
            throw new Exception("My Exception");
        } catch (Exception e) {
            log.error("This is an exception", e);
        }
    }
}
```

<terminated> Log4JPropertiesTest [Java Application] C:\jdk1.5.0_09\jre\bin\javaw.exe (Mar 16, 2007 12:42:09 AM)

```
log4j: Trying to find [log4j.xml] using context classloader sun.misc.Launcher$AppClassLoader@a39137.
log4j: Trying to find [log4j.xml] using sun.misc.Launcher$AppClassLoader@a39137 class loader.
log4j: Trying to find [log4j.xml] using ClassLoader.getSystemResource().
log4j: Trying to find [log4j.properties] using context classloader sun.misc.Launcher$AppClassLoader@a
log4j: Trying to find [log4j.properties] using sun.misc.Launcher$AppClassLoader@a39137 class loader.
log4j: Trying to find [log4j.properties] using ClassLoader.getSystemResource().
log4j: Could not find resource: [null].
log4j:WARN No appenders could be found for logger (test.Log4JPropertiesTest).
log4j:WARN Please initialize the log4j system properly.
```

How do I configure Log4J with properties?

As a next step, let's create a log4j.properties file and put it in our source directory so that log4j can find it. Note that you can use a log4j.xml file too, but .properties files are a little simpler so I'll use them in this example.



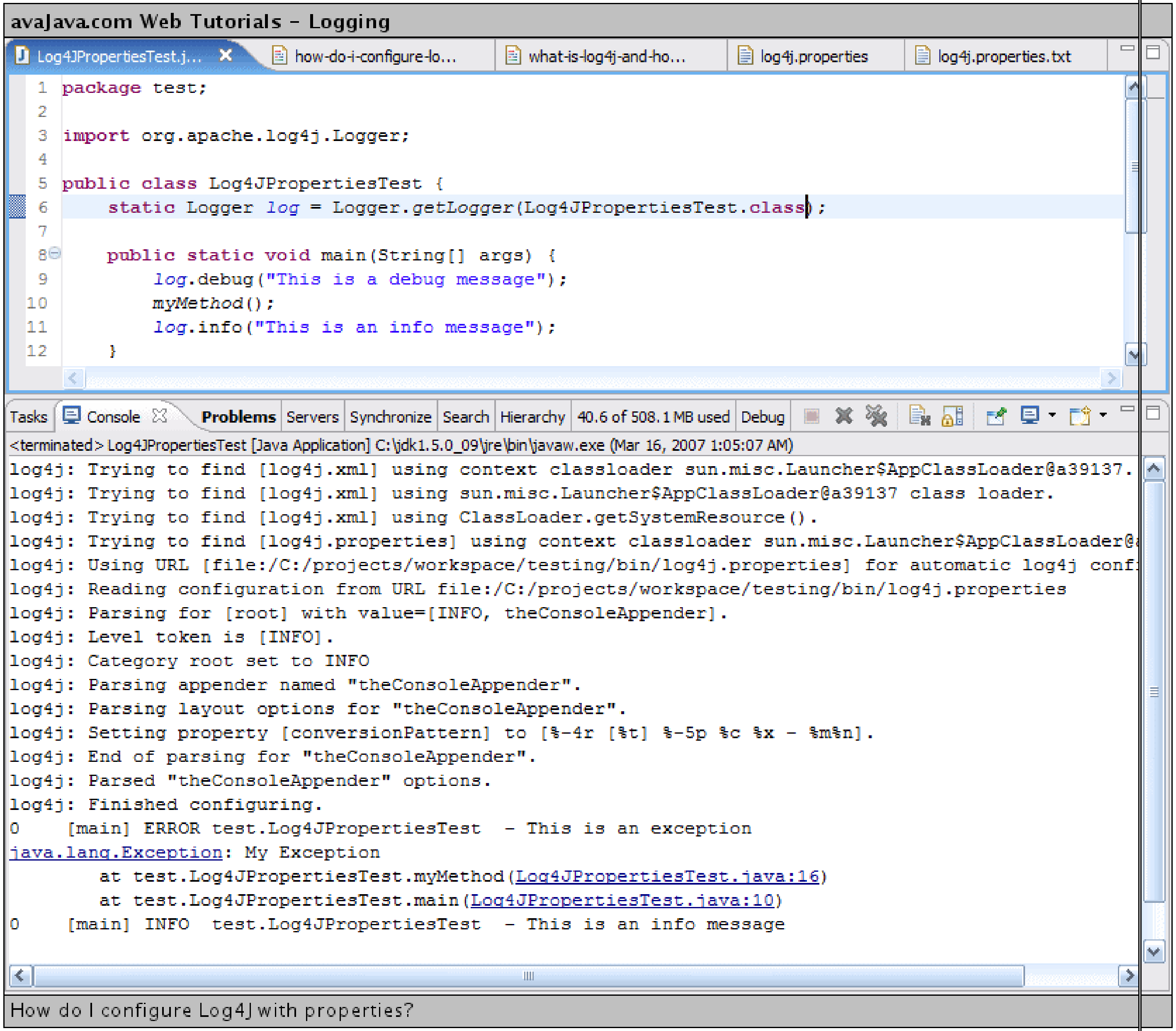
log4j.properties

```
# This sets the global logging level and specifies the appenders
log4j.rootLogger = INFO, theConsoleAppender

# settings for the console appender
log4j.appender.theConsoleAppender=org.apache.log4j.ConsoleAppender
log4j.appender.theConsoleAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.theConsoleAppender.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
```

This is a standard Java properties file. Comments are indicated by lines beginning with #. The first property (log4j.rootLogger) configures the log4j root logger, which is essentially the logger at the top of the log4j logging hierarchy. Since it's at the top of the hierarchy, setting the log level of the root logger applies to everything unless something lower in the hierarchy is specified to have a different logging level. In this example, the log level is set to INFO, meaning that log.debug() messages won't be logged, but log.info(), log.warn(), log.error(), and log.fatal() will be logged.
An appender is a class that sends the log4j logging messages to a particular place. The ConsoleAppender outputs log messages to our console window. I specified that we are using a console appender and named this appender 'theConsoleAppender'. If we wished to add another type of logging, such as to a log file or to an email server, we can specify these additional appenders on the same line separated by commas.

Notice that we can change the format of the log messages going to the console via the ConversionPattern property. This basically calls the setConversionPattern method on the PatternLayout class, so if you're interested in finding out the various conversion patterns, check out the PatternLayout javadocs.

Now that we've added the log4j.properties file to our source directory, let's run our class.


avaJava.com Web Tutorials - Logging
Log4JPropertiesTest.j... how-do-i-configure-lo... what-is-log4j-and-ho... log4j.properties log4j.properties.txt
Tasks Console Problems Servers Synchronize Search Hierarchy 40.6 of 508.1 MB used Debug
How do I configure Log4J with properties?

```
package test;

import org.apache.log4j.Logger;

public class Log4JPropertiesTest {
    static Logger log = Logger.getLogger(Log4JPropertiesTest.class);

    public static void main(String[] args) {
        log.debug("This is a debug message");
        myMethod();
        log.info("This is an info message");
    }
```

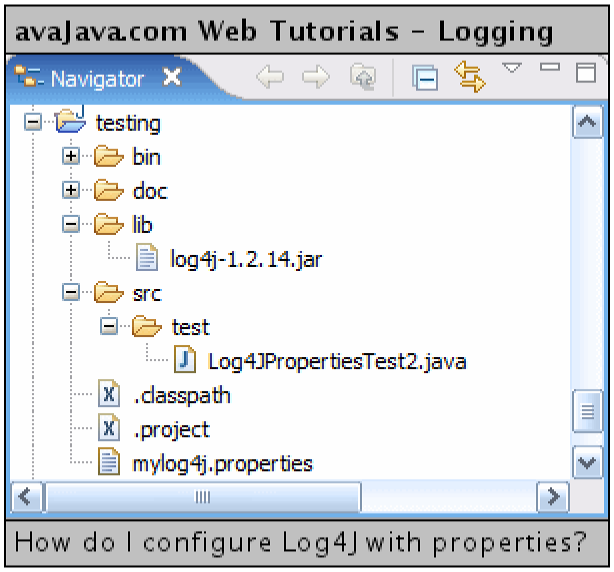
```
<terminated> Log4JPropertiesTest [Java Application] C:\jdk1.5.0_09\jre\bin\javaw.exe (Mar 16, 2007 1:05:07 AM)
log4j: Trying to find [log4j.xml] using context classloader sun.misc.Launcher$AppClassLoader@a39137.
log4j: Trying to find [log4j.xml] using sun.misc.Launcher$AppClassLoader@a39137 class loader.
log4j: Trying to find [log4j.xml] using ClassLoader.getSystemResource().
log4j: Trying to find [log4j.properties] using context classloader sun.misc.Launcher$AppClassLoader@
log4j: Using URL [file:/C:/projects/workspace/testing/bin/log4j.properties] for automatic log4j conf
log4j: Reading configuration from URL file:/C:/projects/workspace/testing/bin/log4j.properties
log4j: Parsing for [root] with value=[INFO, theConsoleAppender].
log4j: Level token is [INFO].
log4j: Category root set to INFO
log4j: Parsing appender named "theConsoleAppender".
log4j: Parsing layout options for "theConsoleAppender".
log4j: Setting property [conversionPattern] to [%-4r [%t] %-5p %c %x - %m%n].
log4j: End of parsing for "theConsoleAppender".
log4j: Parsed "theConsoleAppender" options.
log4j: Finished configuring.
0    [main] ERROR test.Log4JPropertiesTest  - This is an exception
java.lang.Exception: My Exception
        at test.Log4JPropertiesTest.myMethod(Log4JPropertiesTest.java:16)
        at test.Log4JPropertiesTest.main(Log4JPropertiesTest.java:10)
0    [main] INFO  test.Log4JPropertiesTest  - This is an info message
```

As you can see, log4j doesn't find log4j.xml, but it does find log4j.properties at file:/C:/projects/workspace/testing/bin/log4j.properties. Notice that log4j.properties was automatically copied from our src directory to our bin directory by EclipseSW. Since we still have -Dlog4j.debug=true as a VM argument, we can see what log4j is doing. We can see that it reads and parses the properties file to initialize log4j.

After logging is initialized, we can see that the ERROR and INFO messages that we set up are displayed in the console window. Notice that the DEBUG (log.debug()) message is not shown, since this is at a lower level of logging than INFO, which is the root logger logging level that we specified in log4j.properties.

Of course, placing a properties file at the top of your source directory does have its limits. **Thankfully, the PropertConfigurator class lets you solve this problem, since you can pass it the location/name of the log4j properties file that you would like to use.**

Let's create a mylog4j.properties file at the root level of our project.


avaJava.com Web Tutorials - Logging
Navigator
testing
bin
doc
lib
log4j-1.2.14.jar
src
test
Log4JPropertiesTest2.java
.classpath
.project
mylog4j.properties
How do I configure Log4J with properties?

The mylog4j.properties file is shown below. It is very similar to the log4j.properties file that we saw earlier. Note that the log level is set to DEBUG.

mylog4j.properties

```
# This sets the global logging level and specifies the appenders
log4j.rootLogger=DEBUG, myConsoleAppender

# settings for the console appender
log4j.appender.myConsoleAppender=org.apache.log4j.ConsoleAppender
log4j.appender.myConsoleAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.myConsoleAppender.layout.ConversionPattern=%-5p %c %x - %m%n
```

This project contains a Log4JPropertiesTest2 class that is just like the Log4JPropertiesTest class except that it features the following line:

```
        PropertyConfigurator.configure("mylog4j.properties");
```

This line initializes log4j with the configuration properties specified in log4j.properties.

Log4j2.java:

```
package com.sashi;

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

public class Log4j2 {

	static Logger logger = Logger.getLogger(Log4j2.class);

	public static void main(String[] args) {

		PropertyConfigurator.configure("mylog4j.properties");
		logger.debug("This is a Debug Message");
		myMethod();
		logger.info("This is a Info Message");
	}

	public static void myMethod() {

		String s = null;

		try {
			s.length();
		}
		catch(NullPointerException ex) {
			logger.error("NullPointer Exception is occured");
		}
	}
}
```

In the Eclipse debug configuration, I don't have the VM -Dlog4j.debug=true argument present in the earlier example. For normal use, typically this argument is not present. If we run **Log4j2**, we see the following:

Output:

```
DEBUG com.sashi.Log4j2  - This is a Debug Message
ERROR com.sashi.Log4j2  - NullPointer Exception is occured
INFO  com.sashi.Log4j2  - This is a Info Message
```

As you can see, the ConsoleAppender outputs our log messages to the console according to the pattern that we specified in mylog4j.properties.

The PropertyConfigurator can also take a Properties object as an argument. In the Log4jPropertiesTest3 class below, we create a Properties object called log4jProperties and then set several properties on the object. We then pass log4jProperties as a parameter to PropertyConfigurator.configure().

Log4j3.java

```
public class Log4j3 {
	static Logger log = Logger.getLogger(Log4j3.class);

	public static void main(String[] args) {

		Properties log4jProperties = new Properties();

	log4jProperties.setProperty("log4j.rootLogger", "ERROR, myConsoleAppender");
	log4jProperties.setProperty("log4j.appender.myConsoleAppender",
				"org.apache.log4j.ConsoleAppender");
	log4jProperties.setProperty("log4j.appender.myConsoleAppender.layout",
				"org.apache.log4j.PatternLayout");
	log4jProperties.setProperty("log4j.appender.myConsoleAppender.layout.ConversionPattern",
					"%-5p %c %x - %m%n");
		// Passing log4jProperties Object.
		PropertyConfigurator.configure(log4jProperties);

		log.debug("This is a Debug Message 1 ");
		myMethod();
		log.info("This is a Info Message 1");
	}
	public static void myMethod() {
		String s = null;
		try {
			s.length();
		}
		catch(NullPointerException ex) {
			log.error("NullPointer Exception is occured : " +ex);
		}

	}

}
```

Output:

```
ERROR com.sashi.Log4j3  - NullPointer Exception is occured :
                                                java.lang.NullPointerException
```

Notice that in the properties, we specified ERROR as the logging level. In the console output, we see that only the log.error message from our class is displayed.

As a final note, it's possible to initialize log4j by specifying a log4j properties file via a VM argument. This technique is illustrated here:

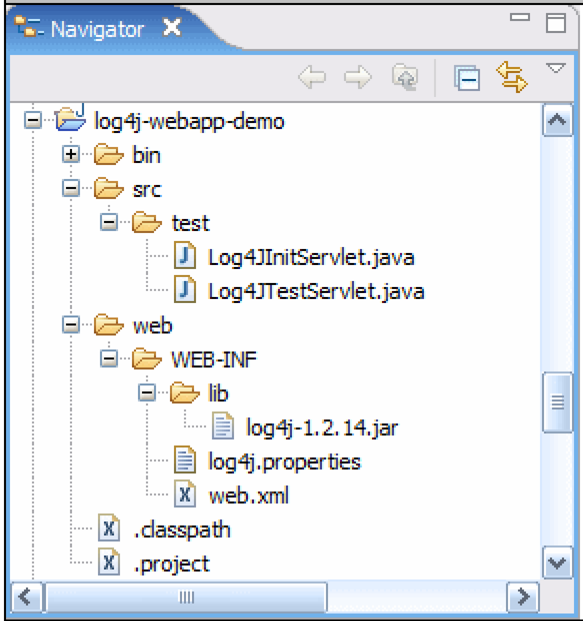
```
-Dlog4j.configuration=file:/C:/projects/workspace/testing/mylog4j.properties
```

3.How do I initialize Log4J in a web application?

In other tutorials, we saw that log4j needs to be initialized in order to be used, and we saw how log4j can be initialized with a BasicConfigurator and initialized with a PropertyConfigurator. This typically occurs when an application starts up. In a web application, we'd also like log4j to start up when the application starts up. One straightforward way of doing this is to put this log4j initialization in a servlet, and specify for the servlet to start up when the application starts up.

Here is a web application project to illustrate this. It contains the log4j jar file in its build path, a log4j.properties file, a web.xml file, a log4j initialization servlet, and a test servlet to try out our log4j initialization.

avaJava.com Web Tutorials – Logging



How do I initialize Log4J in a web application?

Let's start by examining our web.xml file. It contains references to our two servletsw, Log4JTestServlet and Log4JInitServlet. Log4JTestServlet gets mapped to /test so that it can be hit via a browser. The Log4JInitServlet starts up when the web application starts up because of the <load-on-startup> tag. Notice that it has an init-param and an init-value. This value gets passed to the servlet and specifies the location of our log4j.properties file.

web.xml

```
	<servlet>
		<servlet-name>Log4JTestServlet</servlet-name>
		<servlet-class>test.Log4JTestServlet</servlet-class>
	</servlet>

	<servlet>
		<servlet-name>Log4JInitServlet</servlet-name>
		<servlet-class>test.Log4JInitServlet</servlet-class>

		<init-param>
			<param-name>log4j-properties-location</param-name>
			<param-value>WEB-INF/log4j.properties</param-value>
		</init-param>

		<load-on-startup>1</load-on-startup>

	</servlet>

	<servlet-mapping>
		<servlet-name>Log4JTestServlet</servlet-name>
		<url-pattern>/test</url-pattern>
	</servlet-mapping>

</web-app>
```

Let's look at the Log4JInitServlet. In its init() method, it reads in the 'log4j-properties-location' init param value. If the init param doesn't exist, the application gets initialized via the BasicConfigurator. If is does exist, it gets the application's web directory location and appends the init param value to that. If the log4j.properties file exists at that location, it initializes log4j via the PropertyConfigurator using the log4j.properties file. If the log4j.properties file doesn't exist, the application gets initialized via the BasicConfigurator.

Log4JInitServlet.java

```
public class Log4JInitServlet extends HttpServlet {
	@Override
	public void init() throws ServletException {
		System.out.println("Log4JInitServlet is initializing log4j");
		String log4jLocation = getServletConfig().
					getInitParameter("log4j-properties-location");
		System.out.println("log4jLocation      :      " + log4jLocation);
		ServletContext context = getServletConfig().getServletContext();
		if (log4jLocation == null) {
			System.err.println("No log4j-properties-location init param,
				so initializing log4j with BasicConfigurator");
			BasicConfigurator.configure();
		} else {

			String webAppPath = context.getRealPath("/");
/*
			String getRealPath(java.lang.String path)

		Gets the real path corresponding to the given virtual path.

	For example, if path is equal to /index.html, this method will return
	the absolute file path on the server's filesystem to which a request of the form
	http://<host>:<port>/<contextPath>/index.html would be mapped,
	where <contextPath> corresponds to the context path of this ServletContext.

	The real path returned will be in a form appropriate to the computer and
	operating system on which the servlet container is running,
	including the proper path separators.

	Resources inside the /META-INF/resources directories of JAR files bundled in the
	application's /WEB-INF/lib directory must be considered only if the container has
	unpacked them from their containing JAR file,
	in which case the path to the unpacked location must be returned.

	This method returns null if the servlet container is unable to translate
	the given virtual path to a real path.
 */

			System.out.println("webAppPath   :      " + webAppPath);
			String log4jProp = webAppPath + log4jLocation;
			System.out.println("log4jProp    :      " + log4jProp);

			File file = new File(log4jProp);
			if (file.exists()) {
				System.out.println("Initializing log4j with   :" + log4jProp);
				PropertyConfigurator.configure(log4jProp);
			}
			else {
				System.err.println(log4jProp + " file is not found,
					so initializing log4j with BasicConfigurator");
				BasicConfigurator.configure();
			}

		}
		super.init();
	}
}
```

Here is the log4j.properties file. I described properties files in another tutorial so I won't describe them here.

log4j.properties

```
# This sets the global logging level and specifies the appenders
log4j.rootLogger = INFO, theConsoleAppender

# settings for the console appender
log4j.appender.theConsoleAppender=org.apache.log4j.ConsoleAppender
log4j.appender.theConsoleAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.theConsoleAppender.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
```

Our test servlet is shown below. It contains a logger called log. Within its doGet() method, it displays the message 'Howdy' in response to a browser request, and it contains calls to log.debug(), log.info(), log.warn(), log.error(), and log.fatal().

Log4JTestServlet.java

```java
package test;

public class Log4JTestServlet extends HttpServlet {

	static Logger log = Logger.getLogger(Log4JTestServlet.class);

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse res)
	                                  throws ServletException, IOException {

		res.setContentType("text/html");
		PrintWriter out = res.getWriter();

		out.println("Howdy");
		System.out.println("Howdy");

		log.debug("debug message");
		log.info("info message");
		log.warn("warn message");
		log.error("error message");
		log.fatal("fatal message");
	}

}
```

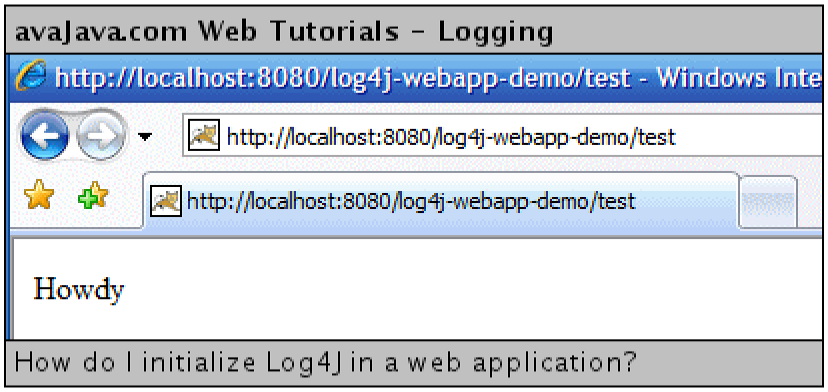
If we start up the project in Eclipse with the Tomcat bootstrap, we see the following console output:

Console Output:

```
INFO: Starting Servlet Engine: Apache Tomcat/7.0.12
Log4JInitServlet is initializing log4j
log4jLocation:      WEB-INF/log4j.properties
webAppPath    :
       D:\Sashi\Eclipse\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\log4j
_Dynamic_Web_Project_2\
log4jProp     :
       D:\Sashi\Eclipse\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\log4j
_Dynamic_Web_Project_2\WEB-INF/log4j.properties
Initializing log4j with          :
       D:\Sashi\Eclipse\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\log4j
_Dynamic_Web_Project_2\WEB-INF/log4j.properties
INFO: Server startup in 425 ms
```

In the console output, notice that, among the Tomcat startup messages, our log4j gets initialized, as evidenced it prints.

If we hit our test servlet in a browser window, we see the 'Howdy' message as expected.


avaJava.com Web Tutorials – Logging

http://localhost:8080/log4j-webapp-demo/test - Windows Inte

http://localhost:8080/log4j-webapp-demo/test

http://localhost:8080/log4j-webapp-demo/test

Howdy

How do I initialize Log4J in a web application?

If we now check our console window, we can see the logging messages that were output in the doGet() method of our test servlet. The output is formatted as we specified in the log4j.properties file. Notice that the log.debug() message in doGet() isn't displayed to the console, since our log4j.properties file specified that the log level is INFO.

```
INFO: Server startup in 425 ms
Howdy
0    ["http-bio-8080"-exec-3] INFO  test.Log4JTestServlet  - info message
2    ["http-bio-8080"-exec-3] WARN  test.Log4JTestServlet  - warn message
3    ["http-bio-8080"-exec-3] ERROR test.Log4JTestServlet  - error message
3    ["http-bio-8080"-exec-3] FATAL test.Log4JTestServlet  - fatal message
```

Initializing log4j via a servlet that loads on web application start-up is a handy technique for initializing log4j.