

Introduction to Modern Application Development
Prof. Aamod Sane
FLAME University and Persistent Computing Institute
Abhijat Vichare
Persistent Computing Institute
Madhavan Mukund
Chennai Mathematical Institute

Lecture-19
Week6_Session1

Hi, welcome to modern application development. This is week 6, and session one. In this session, we are going to take a look at following up with the study of the logs and the browser tools that we did in the last few sessions. Here is a summary of what we have seen so far.

1. We began by looking at the development of an application that provides an interesting service which is the ability for people to manage their money given certain expenses.
2. We first saw how to create a simple command line version.
3. Then we change that into a CGI bin and in doing CGI bin we now see that in order to understand these applications where a standard debugging system does not actually work, we have to become familiar with new tools. These tools include the browser tools, which tell you what the browser is doing, which tells you how the view layout works.
4. And we saw that the the second set of tools is on server side, which show you what the server is doing. And then there is the intermediate set, which is how the protocol is working, which we do with the aid of both the browser and the server. And there are also intermediary tools available such as socat and all which we saw a simple example. But you can do many other things. For example, watch requests as they go by and so on. But if we need to, we will take a look at that.

But for now, it is enough for you to know that such things are possible okay. So, this time we are going to take a deeper look at how logging of an application is managed.

(Refer Slide Time: 02:14)

Persistent Computing Institute **NPTEL**

Week 06, Session 1: Plan

Session Plan

- Understand application logging
- Using Curl to test applications: submit forms

Demos

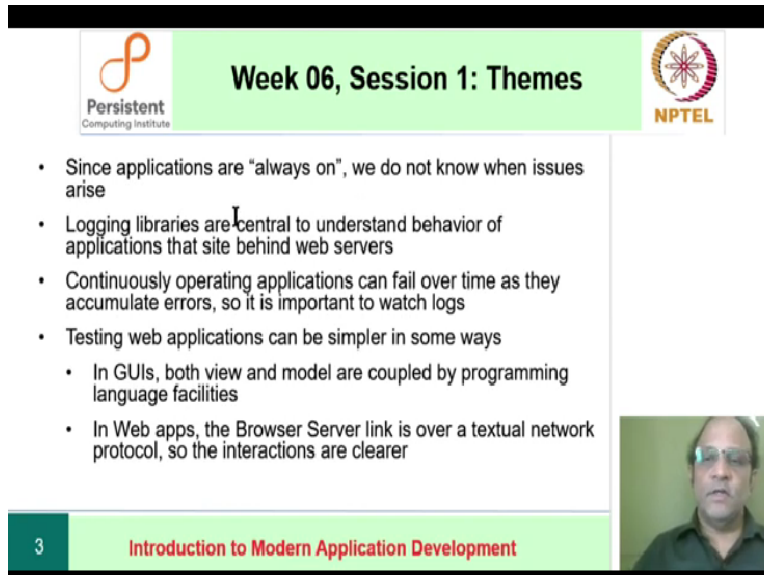
- We will study the details of the log4j library
- To automate testing, instead of relying on the browser we can use Curl
- Several other browser automation libraries exist, we will go over some of them.

2 Introduction to Modern Application Development

So, the session plan is to understand **application logging**. And to understand how together with the curl tool which we have used before, we will be able to debug web applications and understand them in the greater detail. It is important to understand the facilities such as logging in the context of a very simple application, so that by the time we add other details such as servlets we are not overwhelmed by the complexity of studying both of these things at the same time.

Therefore, we are going to study logging in isolation. We will use a library called **log4j**. And we will see how you can use curl instead of relying on the browser to automate some of the testing. Several other browser automation libraries exist and in time, we will go over them.

(Refer Slide Time: 03:21)



The slide is titled "Week 06, Session 1: Themes". It features the Persistent Computing Institute logo on the top left and the NPTEL logo on the top right. The main content is a list of bullet points:

- Since applications are "always on", we do not know when issues arise
- Logging libraries are central to understand behavior of applications that sit behind web servers
- Continuously operating applications can fail over time as they accumulate errors, so it is important to watch logs
- Testing web applications can be simpler in some ways
 - In GUIs, both view and model are coupled by programming language facilities
 - In Web apps, the Browser Server link is over a textual network protocol, so the interactions are clearer

At the bottom left, there is a green bar with the number "3" and the text "Introduction to Modern Application Development". On the bottom right, there is a small video inset showing a man speaking.

In the case of Desktop applications, and even many of the command line application, the person using it is present when the application is running. On the other hand, applications like Web Servers are always on and they are used by a multiplicity of users. So, we can't know when issues arise, or what issue arose if we are not storing the information about the errors etc. in the system. This is exactly what we achieve by using *logging* libraries.

And the act of logging becomes central to understand the behavior of applications that sit behind web servers. Here are some other points along this theme. One thing is that continuously operating applications can fail in one other way, which is different from applications which are started and stopped. They can accumulate errors over time as many, many different people use the application. And the failure of the application can be more gradual.



It might seem difficult at the first, but you should realize that testing web applications and understanding these parts can actually be simpler in many ways than understanding a simple graphical user interface application.

Because in GUIs both view and model are coupled by programming language facilities. And if your GUI happens to use what is called event-based programming, then the control flow of a GUI is rather non trivial. It is not amenable to the usual stack discipline. And you have to be aware of exactly how the writer has arranged the interaction before you can start making sense of it. On the other hand, the style used in web apps certainly of the kind that we are looking at is very straightforward in many ways.

For every single interaction, there is a request that gets sent out from the browser and a response comes from the server. And for every interaction, the result is a new webpage. This style of writing applications has created a great deal of simplification in what people expect from applications. Though, many people might not agree, but there is some weight in the fact that the way the web works is a preferred way of writing applications.

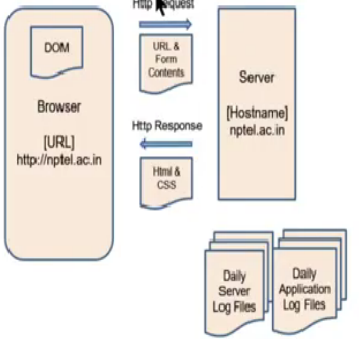
So, in this session, in particular we will see that *the browser server-link over a textual network protocol* helps us understand the functioning of the two parts of the application much better than *a tightly coupled view and model within a GUI application.*

(Refer Slide Time: 07:19)




Week 06, Session 1: Big Picture

- In this session, we will focus on logs
- We will also see how we can automate the behavior of a user to enable testing
- The overall picture is the same as for the last session.



4

Introduction to Modern Application Development



The big picture for this session is the same as the one we used last time. So, there is the browser with its URL, it has a DOM, it has form contents, HTML, CSS, etc. But now our interest lies in applications which store logs.

Last time, we also saw how web servers maintain their own logs. And this time, we are going to focus on logs that are generated by the application. We will see how we should write them and how they are generated.

And as a side effect, we will also find out a way to automate the testing of the behavior of a user certainly for the simple application that we are using. So, you can think of this session as a continuation of the last one where we will be looking at a part that we did not study so much last time.

(Refer Slide Time: 08:17)



The slide is titled "Part I" and features logos for "Persistent Computing Institute" and "NPTEL". The main content is a bulleted list:

- Review of ideas so far
 - Logs used by servers like apache
- Where should application debug information go?
- Managing application logs

A small video inset in the bottom right corner shows a man speaking. The footer of the slide displays the number "5" and the text "Introduction to Modern Application Development".

Here is how we will proceed next, we will first take a look at the logs which are used by apache as a refresher again, and then we will think about where should application information go. Once we figure that out, we can understand how to manage application logs and we will be doing so, using a library called **log4j**.

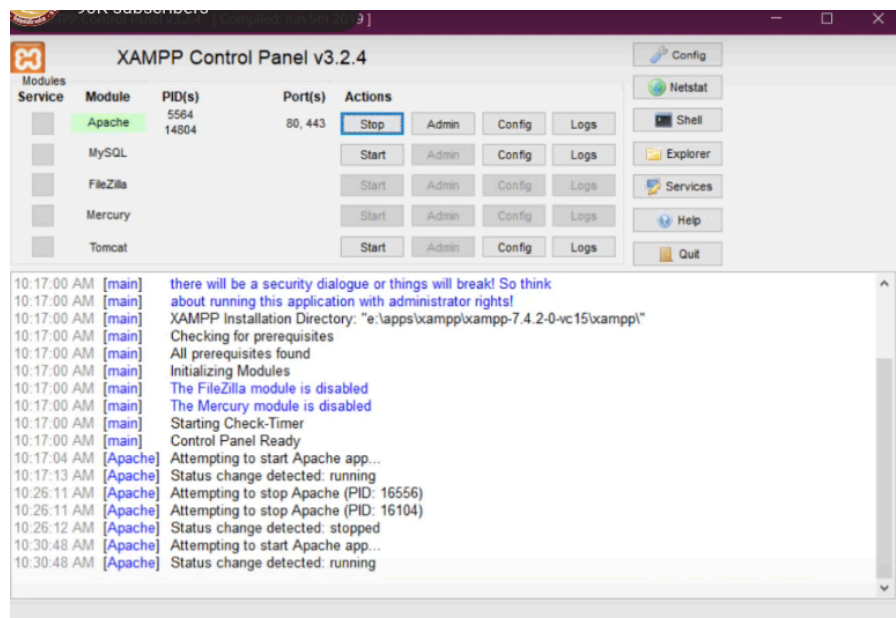
(Video Starts: 08:43).

Let us get started.

```
Aamod Sane@MSI /cygdrive/e/Apps/xampp/xampp-7.4.2-0-VC15/xampp/apache/logs
$ ls -al
total 17K
drwxrwx---+ 1 Aamod Sane None    0 Mar  2 10:26 ./
drwxrwx---+ 1 Aamod Sane None    0 Feb 13 14:43 ../
-rwxrwx---+ 1 Aamod Sane None    0 Mar  2 10:26 access.log*
-rwxrwx---+ 1 Aamod Sane None    0 Mar  2 10:25 error.log*
-rwxrwx---+ 1 Aamod Sane None    7 Mar  2 10:23 httpd.pid*
-rwxrwx---+ 1 Aamod Sane None  3.1K Feb 13 14:43 install.log*
-rwxrwx---+ 1 Aamod Sane None  3.2K Feb 16 15:51 ssl_request.log*

Aamod Sane@MSI /cygdrive/e/Apps/xampp/xampp-7.4.2-0-VC15/xampp/apache/logs
$ |
```

In the Apache Log directly, you can see (see the screenshot given above) that we have **access log** and **error log**, and both of them are empty at start. To begin with, we are going to start apache in our Xampp control panel. As soon as it starts, it prints some information to the logs. So, let us see what this information is.



We got quite a few things in the error log and nothing in the access log. This says that there has not been any URL access.

```
Aamod Sane@MSI /cygdrive/e/Apps/xampp/xampp-7.4.2-0-VC15/xampp/apache/logs
$ ls -al
total 21K
drwxrwx---+ 1 Aamod Sane None 0 Mar 2 10:26 ./
drwxrwx---+ 1 Aamod Sane None 0 Feb 13 14:43 ../
-rwxrwx---+ 1 Aamod Sane None 0 Mar 2 10:26 access.log*
-rwxrwx---+ 1 Aamod Sane None 2.3K Mar 2 10:30 error.log*
-rwxrwx---+ 1 Aamod Sane None 6 Mar 2 10:30 httpd.pid*
-rwxrwx---+ 1 Aamod Sane None 3.1K Feb 13 14:43 install.log*
-rwxrwx---+ 1 Aamod Sane None 3.2K Feb 16 15:51 ssl_request.log*
```

But the error log does not necessarily mean there are errors, although it might, it also prints some things that happen at the start of the system. So, let us look at the error log here. So here what it says is some complaint about the example.com which I tried out, then something about, you know, unclean shutdown of a previous apache run, etc., etc. Some warnings about PHP startups, and something about ability to unload, not load a dynamic library, and so on and so forth.

And something about starting these many worker threads. By and large, as long as the startup works, it is nice to have these things clean. But sometimes these warnings are spurious in with some experience you can know which matter and which do not. Some of them are simply announcements like this, something about the build time, and so on and so forth. But in any case, let us go ahead and see what happens. When we write the command: `tail error.log`, we get the following output.

```

Error Log
[Mon Mar 02 10:30:48.690262 2020] [ssl:warn] [pid 5564:tid 640] AH0190
9: www.example.com:443:0 server certificate does NOT include an ID whi
ch matches the server name
[Mon Mar 02 10:30:48.730486 2020] [core:warn] [pid 5564:tid 640] AH000
98: pid file E:/Apps/xampp/xampp-7.4.2-0-VC15/xampp/apache/logs/httpd.
pid overwritten -- previous shutdown of previous apache run
[Mon Mar 02 10:30:48.737466 2020] [ssl:warn] [pid 5564:tid 640] AH0190
9: www.example.com:443:0 server certificate does NOT include an ID whi
ch matches the server name
PHP warning: PHP Startup: Unable to load dynamic library 'pdo_sqlite'
(tried: \\Apps\\xampp\\xampp-7.4.2-0-VC15\\xampp\\php\\ext\\pdo_sqlit
e (The specified module could not be found.), \\Apps\\xampp\\xampp-7.4
.2-0-VC15\\xampp\\php\\ext\\php_pdo_sqlite.dll (The specified module c
ould not be found.)) in unknown on line 0
[Mon Mar 02 10:30:48.761403 2020] [mpm_winnt:notice] [pid 5564:tid 640
] AH00455: Apache/2.4.41 (win64) OpenSSL/1.1.1c PHP/7.4.2 configured -
- resuming normal operations
[Mon Mar 02 10:30:48.761403 2020] [mpm_winnt:notice] [pid 5564:tid 640
] AH00456: Apache Lounge VC15 Server built: Aug 11 2019 12:20:04
[Mon Mar 02 10:30:48.761403 2020] [core:notice] [pid 5564:tid 640] AH0
0084: Command line: 'E:/Apps/xampp/xampp-7.4.2-0-VC15/xampp/apach
e/bin/httpd.exe -d E:/Apps/xampp/xampp-7.4.2-0-VC15/xampp/apache'
[Mon Mar 02 10:30:48.809276 2020] [mpm_winnt:notice] [pid 5564:tid 640
] AH00418: Parent: created child process 14804
[Mon Mar 02 10:30:49.224282 2020] [ssl:warn] [pid 14804:tid 632] AH019
09: www.example.com:443:0 server certificate does NOT include an ID whi
ch matches the server name
[Mon Mar 02 10:30:49.225233 2020] [ssl:warn] [pid 14804:tid 632] AH019
09: www.example.com:443:0 server certificate does NOT include an ID whi
ch matches the server name
PHP Warning: PHP Startup: Unable to load dynamic library 'pdo_sqlite'
(tried: \\Apps\\xampp\\xampp-7.4.2-0-VC15\\xampp\\php\\ext\\pdo_sqlit
e (The specified module could not be found.), \\Apps\\xampp\\xampp-7.4
.2-0-VC15\\xampp\\php\\ext\\php_pdo_sqlite.dll (The specified module c
ould not be found.)) in unknown on line 0
[Mon Mar 02 10:30:49.282162 2020] [mpm_winnt:notice] [pid 14804:tid 63
2] AH00354: Child: Starting 150 worker threads.
error.log (END)
```

We will `tail` the error log as well as the access log. And this time, there is nothing in it yet. Now let us go to our fair share to application and within our standard example expenses to 200 or f1, something this far. So as soon as we submit it, you will see that first you get access log, which tells us that the application has become active. And second no error has happened. And so, no change in the error log.

```
Aamod Sane@MSI /cygdrive/e/Apps/xampp/xampp-7.4.2-0-VC15/xampp/apache/logs
$ tail -f access.log
127.0.0.1 - - [02/Mar/2020:10:32:57 +0530] "POST /cgi-bin/fairShare2.sh HTTP/1.1"
200 2306 "http://localhost/fairshare2.html" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0) Gecko/20100101 Firefox/71.0"
```

In this version of the fair share app, the app works successfully. And so, we did not see anything in the error logs. Now we will use such data on the form of FairShare which is known to have an error, and let us see what happens. We write following in the command box:

```
register f1 f2
expense f1 200
report f2
end
```

And as you will see, nothing new will shows up in the access log because I know there is an error. And now let us see what the log says. So, now if we check the the access log, it will simply show that there was an access (See below).

```
Aamod Sane@MSI /cygdrive/e/Apps/xampp/xampp-7.4.2-0-VC15/xampp/apache/logs
$ tail -f access.log
127.0.0.1 - - [02/Mar/2020:10:32:57 +0530] "POST /cgi-bin/fairShare2.sh HTTP/1.1"
200 2306 "http://localhost/fairshare2.html" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0) Gecko/20100101 Firefox/71.0"
127.0.0.1 - - [02/Mar/2020:10:53:30 +0530] "POST /cgi-bin/fairShare3.sh HTTP/1.1"
200 - "http://localhost/fairshare3.html" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0) Gecko/20100101 Firefox/71.0"
```

And in fact, there is something odd about this system because as far as the access is concerned, we are actually returning the HTTP success code. But if you go back to the browser, of course, you cannot see anything. And as far as this system is concerned, we still see HTTP success. And if you look at the contents, it looks like everything is okay. So, this tells us that when designing the application, we have to make sure that the error status is properly reflected in the application, which we will see how to do in a little while.

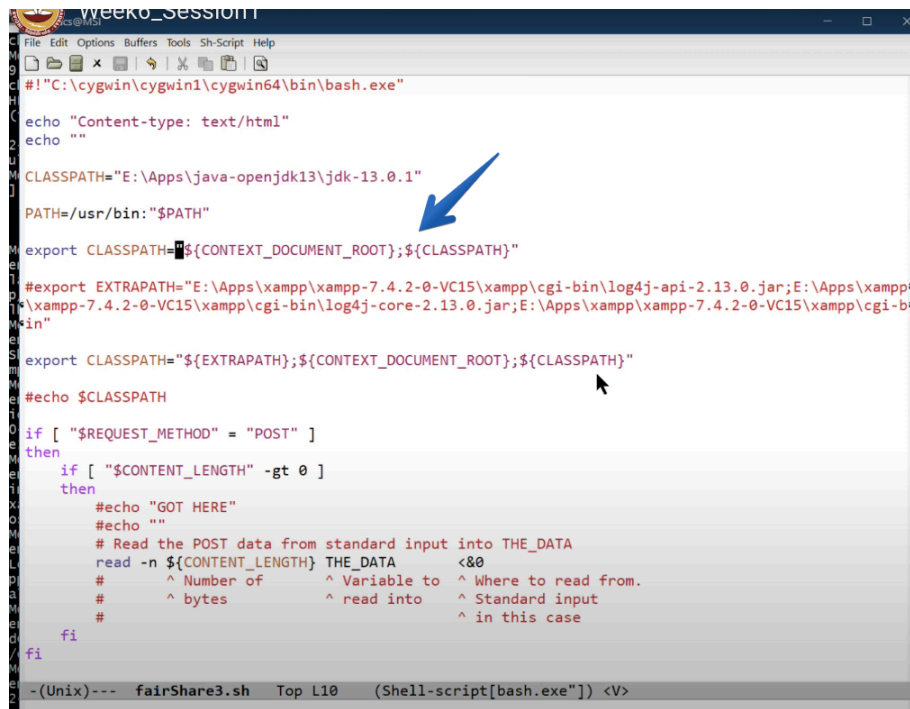
For now, we will ignore this part and instead see where can we find that there is an error, aside from the fact that there is a blank screen in the browser. So, let us see what the error was the access. So, this is what the error log has to tell us:

```
[Mon Mar 02 10:53:31.851312 2020] [cgi:error] [pid 14804:tid 1984] [client 127.0.0.1:59391] AH01215: Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/logging/log4j/LogManager\r: E:/Apps/xampp/xampp-7.4.2-0-VC15/xampp/cgi-bin/fairShare3.sh, referer: http://localhost/fairshare3.html
[Mon Mar 02 10:53:31.851312 2020] [cgi:error] [pid 14804:tid 1984] [client 127.0.0.1:59391] AH01215: \tat fairShareHtmlInteractive3.main(fairShareHtmlInteractive3.java:545)\r: E:/Apps/xampp/xampp-7.4.2-0-VC15/xampp/cgi-bin/fairShare3.sh, referer: http://localhost/fairshare3.html
[Mon Mar 02 10:53:31.852309 2020] [cgi:error] [pid 14804:tid 1984] [client 127.0.0.1:59391] AH01215: Caused by: java.lang.ClassNotFoundException: org.apache.logging.log4j.LogManager\r: E:/Apps/xampp/xampp-7.4.2-0-VC15/xampp/cgi-bin/fairShare3.sh, referer: http://localhost/fairshare3.html
[Mon Mar 02 10:53:31.852309 2020] [cgi:error] [pid 14804:tid 1984] [client 127.0.0.1:59391] AH01215: \tat java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:602)\r: E:/Apps/xampp/xampp-7.4.2-0-VC15/xampp/cgi-bin/fairShare3.sh, referer: http://localhost/fairshare3.html
[Mon Mar 02 10:53:31.852309 2020] [cgi:error] [pid 14804:tid 1984] [client 127.0.0.1:59391] AH01215: \tat java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:178)\r: E:/Apps/xampp/xampp-7.4.2-0-VC15/xampp/cgi-bin/fairShare3.sh, referer: http://localhost/fairshare3.html
[Mon Mar 02 10:53:31.852309 2020] [cgi:error] [pid 14804:tid 1984] [client 127.0.0.1:59391] AH01215: \tat java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:521)\r: E:/Apps/xampp/xampp-7.4.2-0-VC15/xampp/cgi-bin/fairShare3.sh, referer: http://localhost/fairshare3.html
[Mon Mar 02 10:53:31.852309 2020] [cgi:error] [pid 14804:tid 1984] [client 127.0.0.1:59391] AH01215: \t... 1 more\r: E:/Apps/xampp/xampp-7.4.2-0-VC15/xampp/cgi-bin/fairShare3.sh, referer: http://localhost/fairshare3.html
```

It says, first of all, that there was an exception in a thread and there was a Java class, which it was unable to find. Then, it kind of says the same thing over and over again.

There are many more errors but, in any event, what matters is that there is enough information for us to figure out that something is wrong, just exactly what is wrong. Since we have injected this error deliberately, we already know.

The point is that when a Java program is invoked, it gets a class path. And the particular class path pointed in the picture given below, as it turns out, is not what you need. Instead, what we need is a class path which I have already written below that line but is commented out. The point of this was simply to show how using errors, we can find the bug and then fix it so that the next time we run the program, we would not be having this particular problem.



```
weeko_session1
File Edit Options Buffers Tools Sh-Script Help
#!"C:\cygwin\cygwin1\cygwin64\bin\bash.exe"

echo "Content-type: text/html"
echo ""

CLASSPATH="E:\Apps\java-openjdk13\jdk-13.0.1"
PATH=/usr/bin:$PATH

export CLASSPATH=${CONTEXT_DOCUMENT_ROOT};${CLASSPATH}"

#export EXTRAPATH="E:\Apps\xampp\xampp-7.4.2-0-VC15\xampp\cgi-bin\log4j-api-2.13.0.jar;E:\Apps\xampp\xampp-7.4.2-0-VC15\xampp\cgi-bin\log4j-core-2.13.0.jar;E:\Apps\xampp\xampp-7.4.2-0-VC15\xampp\cgi-bin"
export CLASSPATH=${EXTRAPATH};${CONTEXT_DOCUMENT_ROOT};${CLASSPATH}"

#echo $CLASSPATH

if [ "$REQUEST_METHOD" = "POST" ]
then
    if [ "$CONTENT_LENGTH" -gt 0 ]
    then
        #echo "GOT HERE"
        #echo ""
        # Read the POST data from standard input into THE_DATA
        read -n ${CONTENT_LENGTH} THE_DATA <&0
        # ^ Number of ^ Variable to ^ Where to read from.
        # ^ bytes ^ read into ^ Standard input
        # ^ in this case
    fi
fi

-(Unix)--- fairShare3.sh Top L10 (Shell-script[bash.exe]) <V>
```

If we test this again, we shall see that it will work properly. And now success is actually success. And there were no new errors. As you can see, this is from the last time we created room here. So, if there were errors, the new message would have arrived at this point. So, we just saw another demo of how access log and error logs are used and **access log just record requests**, whereas **error log tells us whether there was an error and what the details were**. This part has been done for us by the apache server itself.

But our application will need to have logs of its own, we could decide to make it part of apache log. But there can be multiple applications. And generally, every application should have its own log. So, let us see what that looks like. Our applications are in CGI bin. And so, one good place for the logs to be could be in CGI bin in our case, usually though, all logs tend to be in the same directory.

And so, we will create typically logs in the same directory where the apache logs lie, in most production settings, but for now we will just do it inside the CGI bin directory itself. So, here is what we have in our CGI bin (see picture given below).

```
Aamod Sane@MSI /cygdrive/e/Apps/xampp/xampp-7.4.2-0-VC15/xampp/cgi-bin
$ ls -l
total 2.1M
-rw-rw-r--+ 1 Aamod Sane None 13 Feb 24 01:01 2
-rwxrwx---- 1 Aamod Sane None 930 Feb 23 23:52 fairShare2.sh*
-rwxrwx---- 1 Aamod Sane None 1.3K Mar 2 11:00 fairShare3.sh*
-rwxrwx---- 1 Aamod Sane None 1.3K Mar 2 00:52 fairShare3.sh~*
-rwxrwx---- 1 Aamod Sane None 16K Feb 22 23:12 fairShareHtmlInteractive.java*
-rwxrwx---- 1 Aamod Sane None 12K Feb 22 23:13 fairShareHtmlInteractive2.class*
-rwxrwx---- 1 Aamod Sane None 13K Mar 2 00:54 fairShareHtmlInteractive3.class*
-rwxrwx---- 1 Aamod Sane None 17K Mar 2 00:55 fairShareHtmlInteractive3.java*
-rwxrwx---- 1 Aamod Sane None 16K Mar 2 00:13 fairShareHtmlInteractive3.java~*
-rwxrwx---- 1 Aamod Sane None 633 Feb 23 13:01 log4j2.properties~*
-rwxrwx---- 1 Aamod Sane None 1.1K Mar 2 01:02 log4j2.xml*
-rwxrwx---- 1 Aamod Sane None 374 Feb 23 13:22 log4j2.xml~*
-rwxrwx---- 1 Aamod Sane None 280K Feb 23 12:38 log4j-api-2.13.0.jar*
-rwxrwx---- 1 Aamod Sane None 1.7M Feb 23 12:38 log4j-core-2.13.0.jar*
drwxrwx---- 1 Aamod Sane None 0 Mar 2 11:10 logs/
-rwxrwx---- 1 Aamod Sane None 1.2K Mar 2 01:09 logtest.class*
-rwxrwx---- 1 Aamod Sane None 841 Mar 2 11:08 logtest.java*
-rwxrwx---- 1 Aamod Sane None 825 Feb 23 16:03 logtest.java~*
-rwxrwx---- 1 Aamod Sane None 2.5K Feb 24 01:10 postdata.sh*
-rwxrwx---- 1 Aamod Sane None 2.5K Feb 24 01:03 postdata.sh~*
-rwxrwx---- 1 Aamod Sane None 1.3K Feb 23 23:35 postdatasimple.sh*
-rwxrwx---- 1 Aamod Sane None 1.2K Feb 23 23:34 postdatasimple.sh~*
```

There is a bunch of implementations of fair share. And then here are a bunch of log related things. So, first, let us see what log related packages we have **log4star**.

So here we have a few important points.

We ignore the files whose name with tilde as those are created by the editor. We execute the following command:

```
ls -l log4* | grep -v '~'
```

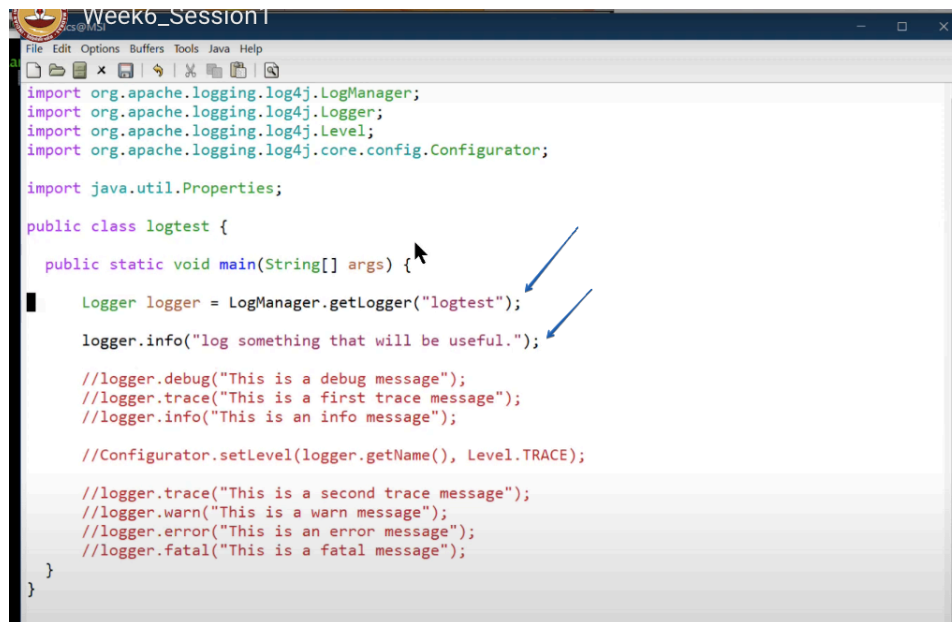
So, this will remove everything with that tilde okay and leaves the files we want. So, we have log4j2.xml, we have two jar files. And these two jar files, and the *config* is typically what you need for logging. So that is the as far as the library is concerned. But first before we look at the details of the library, let us take a look at what the general idea is behind organizing these logs.

So, here is what the **log4j** documentation tells us about logging versus ordinary printing.

1. The first and foremost advantage, they say, over `system.out.println` is that certain log statements can be disabled while certain other ones can be allowed to print unhindered. So, in normal system out print line, of course, whatever you insert is, whatever gets printed. Here, the main thing is that we can make distinctions between different statements that should and should not be printed, depending on our needs in the context in which we are working.

2. This capability assumes that the logging space that is the space of all possible logging statements is categorized according to some developer chosen criteria.

And what these criteria will be, we will see in a bit, here is the parts of the system that matter to us: In the library, there is an entity called a logger, which we are supposed to use instead of system or printf or similar. And this is how we use it. Let us take a look at the actual code.



```
Week6_Session1
File Edit Options Buffers Tools Java Help
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.Level;
import org.apache.logging.log4j.core.config.Configurator;

import java.util.Properties;

public class logtest {

    public static void main(String[] args) {

        Logger logger = LogManager.getLogger("logtest");

        logger.info("log something that will be useful.");

        //logger.debug("This is a debug message");
        //logger.trace("This is a first trace message");
        //logger.info("This is an info message");

        //Configurator.setLevel(logger.getName(), Level.TRACE);

        //logger.trace("This is a second trace message");
        //logger.warn("This is a warn message");
        //logger.error("This is an error message");
        //logger.fatal("This is a fatal message");

    }
}
```

Let us just take a look at these two lines pointed in the picture above. So, first of all, here we have logger. And here we have logger dot info, log something that could be useful. So, what is happening is that there is an object which is automatically getting created of the class *LogManager*. And when you ask the log manager to get a logger that is appropriate for the *logtest* class, we get it.

By the way this name is chosen by us, the logging system does not necessarily involve itself unless you ask it to be. And having acquired a logger object, we will now log something that will be useful. So, let us see what happens.

If we head over to CGI bin and compile `logtest.java` and let us run it. So, we have java log test. And here is what happens we get some information saying info log test log something that will be useful.

But a second thing has happened, in this directory logs, we have two logs, one for `logtest` and another for `fairshare`. And as you can see, something also got printed to `logtest`. And it was the same message there. We printed out here, which is `logtest - log something that will be useful`. So, the logger has indeed done something different from just standard out `printf`. How did this know where to put text message?

The answer is in the log configurator and the XML file that we just saw, which tells it what to do. So, the configuration says that we can change the receiver of the output. And you can even do so dynamically during the execution of the program. There is one other feature which we will talk about which is called log appenders, such that within a single log, you can have a multiple appenders. As a quick look, let us take a look at `log4j2.xml`

This XML file does the configuration and here is what I was talking about. First of all, there is something called the appenders, which is the targets where logs are appended to one such appender is `test app log`, which is our `log test dot log`. And it has this pattern layout. By and large, I just picked some layout that works. Sometimes there is value in designing this layout properly. But for now, we would not get into those details.

And then there is something called loggers, which again for `log test` says the following that the content should go to the console and to the `test app log`. And this is what we are seeing in this system, there are a bunch of other things involving: levels, additivity, etc. which we will talk about. Additivity is something that we will set to false. And we will ignore the issue for now. It gets into things that are sort of irrelevant for the kinds of apps that we are looking at.

```


File Edit Options Buffers Tools XML Text Help
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
    <File name="FairShareLog" fileName="logs/fairshare.log">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </File>
    <File name="TestAppLog" fileName="logs/logtest.log">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </File>
  </Appenders>
  <Loggers>
    <Logger name="logtest" level="debug" additivity="false">
      <AppenderRef ref="Console" level="trace"/>
      <AppenderRef ref="TestAppLog" level="info"/>
    </Logger>
    <Logger name="fairshare" level="debug" additivity="false">
      <AppenderRef ref="Console" level="trace"/>
      <AppenderRef ref="FairShareLog" level="info"/>
    </Logger>
    <Root level="warn">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
U\--- log4j2.xml All L11 (nXML Valid) <V>

```

So, let us get back. So, as you can see from the picture give above, there is a configurator. And there is something called **appender**. And within a single log, there is an appender, which says send this thing two places to the console and to the test log okay. Here is how the apache side describes it. It says that there is a logger context.


(Video Ends: 23:44)

(Refer Slide Time: 23:45)



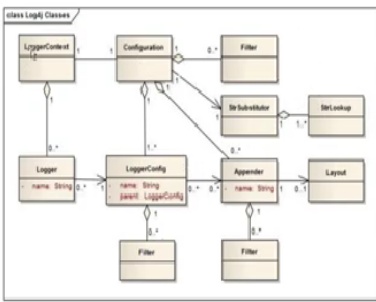
Persistent
Computing Institute

Logger Class diagram



NPTEL

- Log4J2 has many features, but we care about only a few
- In the code, use log statements
- In a config file, set up logs
- When needed, affect logger dynamically




```

classDiagram
    class LoggerContext {
    }
    class Configuration {
    }
    class Filter {
    }
    class Logger {
        name: String
    }
    class LoggerConfig {
        parent: LoggerConfig
    }
    class Appender {
        name: String
    }
    class Layout {
    }
    class Filter {
    }
    class Substitutor {
    }
    class StrLookup {
    }

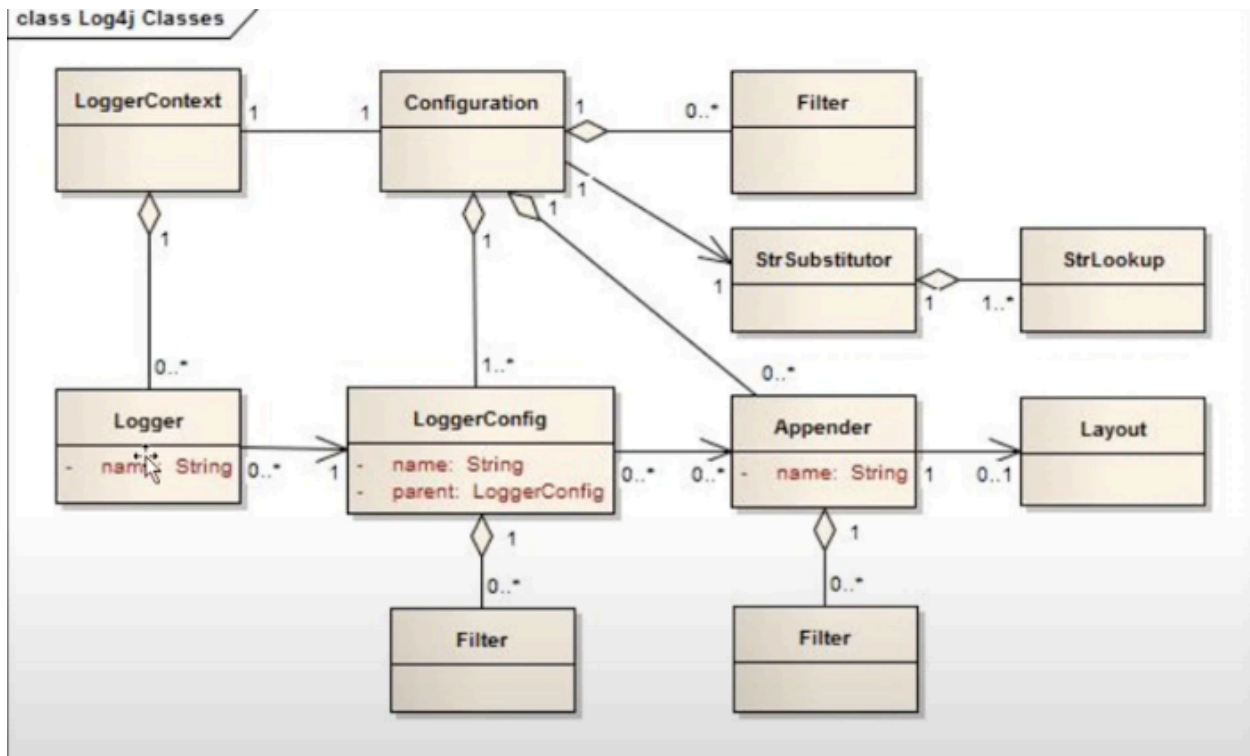
    LoggerContext "1" -- "1" Configuration
    Configuration "1" -- "0..1" Filter
    Configuration "1" -- "1" Substitutor
    Configuration "1" -- "1" StrLookup
    Logger "0..1" -- "1" LoggerConfig
    LoggerConfig "1" -- "0..1" Appender
    Appender "0..1" -- "0..1" Layout
    Appender "0..1" -- "0..1" Filter
    
```

9

Introduction to Modern Application Development



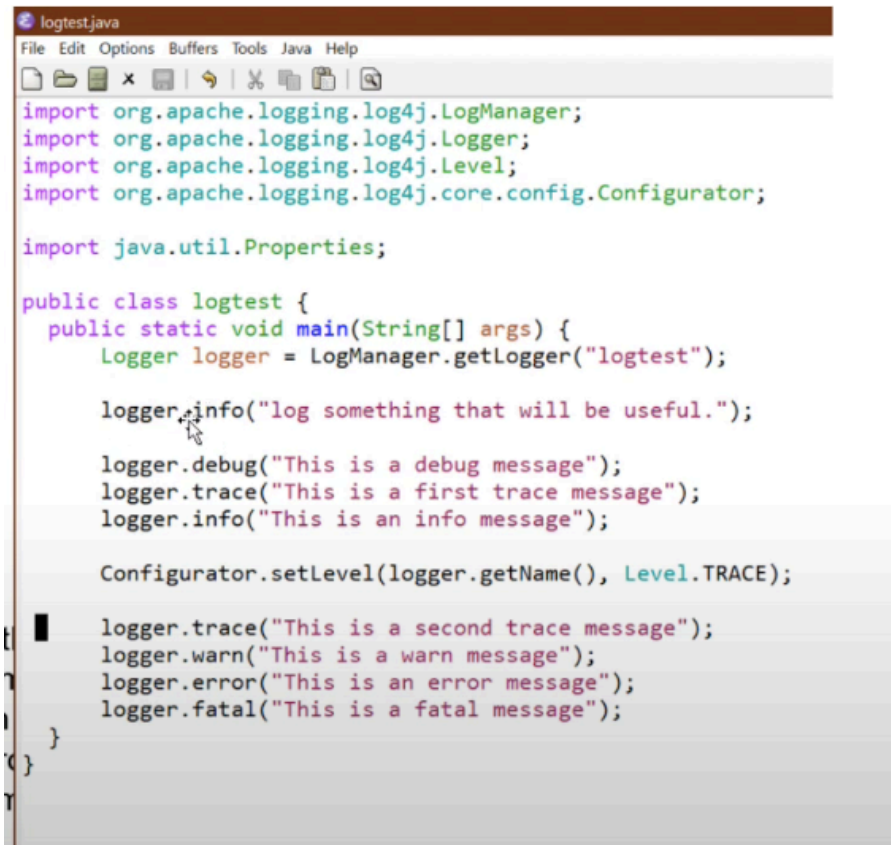
So, we have a logger context. We have a configuration is a bunch of things called filters. These two are abstract. What we care about is there is a logger, there is a log config and there is an appender. These are really the only 3 classes that we care about. It has many features, but I have picked out a few that are relevant to our types of apps. In the main point is in the code, use log statements in a config file setup log.



And when needed, we can affect the logger dynamically. And that is actually one of the more interesting parts about using loggers.

(Video Starts: 24:26)

So, let us see what is going on with that. This is an idea called **log levels**. So, if you look at the code shown in the picture given below, what it says is after the statements there is a bunch of false to the logger using levels called info, debug, trace, etc. Something else is happening. This is the dynamic setting and we will come to that in a bit. But here is the significant part first, every time in the log file has a level.



```
logtest.java
File Edit Options Buffers Tools Java Help

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.Level;
import org.apache.logging.log4j.core.config.Configurator;

import java.util.Properties;

public class logtest {
    public static void main(String[] args) {
        Logger logger = LogManager.getLogger("logtest");

        logger.info("log something that will be useful.");

        logger.debug("This is a debug message");
        logger.trace("This is a first trace message");
        logger.info("This is an info message");

        Configurator.setLevel(logger.getName(), Level.TRACE);

        logger.trace("This is a second trace message");
        logger.warn("This is a warn message");
        logger.error("This is an error message");
        logger.fatal("This is a fatal message");
    }
}
```

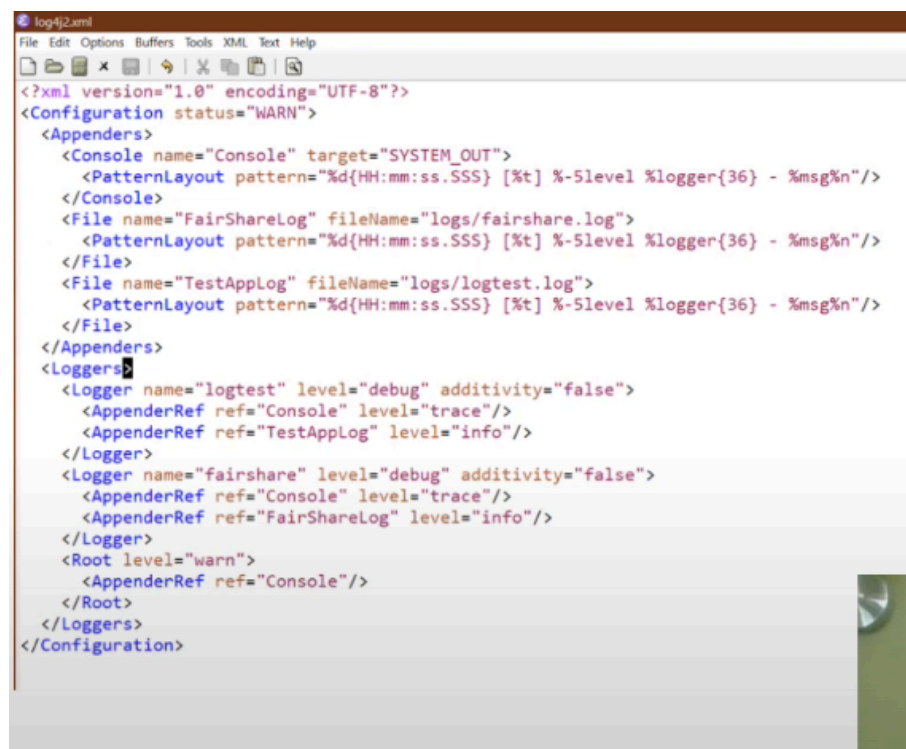
So, this is the level indication in the log. Here is how you choose levels. Throughout your application, you are going to put log statements. And the categorization works like this, you should use info to show that the application is working normally, a few important stages in the application are captured at the info level. So that it is kind of a health check. You can write log watchers which are processes that go around looking through the log to see whether the info material is flowing through as expected.

The next level is one which says that things are okay now, but you might be in trouble later because maybe, you know, the currently we found a workaround around some fault. For example, you are trying to contact some other server and it is not crucial, it may be gives you something useful, like how many visitors have visited or some such thing. And perhaps it is not crucial right now, but at some point, it may start mattering.

Error level says that there are problems right now, and you should act. And this is where some crucial service is failing and perhaps users can see the effect. So, error generally is a pretty severe situation.

Fatal, on the other hand is giving up completely, at least for the request that is currently working to other levels are debug and trace as you might imagine, these are for detailed information for debugging purposes.

And trace is even more detailed. So, look at it absolutely step by step. That is generally how this thing works. And now we will see some interesting interactions between these ideas. This log configuration here, as we can see in picture given below:

A screenshot of a text editor window titled 'log4j2.xml'. The editor shows an XML configuration for Log4j2. The root element is <Configuration status='WARN'>. Inside, there's a <Appenders> section with three appender definitions: <Console name='Console' target='SYSTEM_OUT'>, <File name='FairShareLog' fileName='logs/fairshare.log'>, and <File name='TestAppLog' fileName='logs/logtest.log'>. Each appender has a <PatternLayout pattern='<code>%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n</code>'>. Below the appenders is a <Loggers> section with two logger definitions: <Logger name='logtest' level='debug' additivity='false'> and <Logger name='fairshare' level='debug' additivity='false'>. Each logger has two <AppenderRef> elements: one for 'Console' with level 'trace' and one for the respective file appender with level 'info'. Finally, there's a <Root level='warn'> element with an <AppenderRef ref='Console'>.

First of all, we have two loggers. Currently we are only concerned with log test we will look at fair share in a little while. But the two things to watch it are that logs are going to console and to the log file. Okay, so let us see what happens with our logtest file as we keep making changes.

So, first of all, here we have logger info. And after this, we will put out a debug message. And we have to go and do the same thing that we did before which is to compile and run

logtest.java. Now we get one more message info and debug but this time something else interesting happens. So, if we execute:

```
cat logs logtest.log
```

we find that the second info message has arrived. But the debug message appears only on the console and not in the log file. Why did that happen? This was our choice in the configuration.

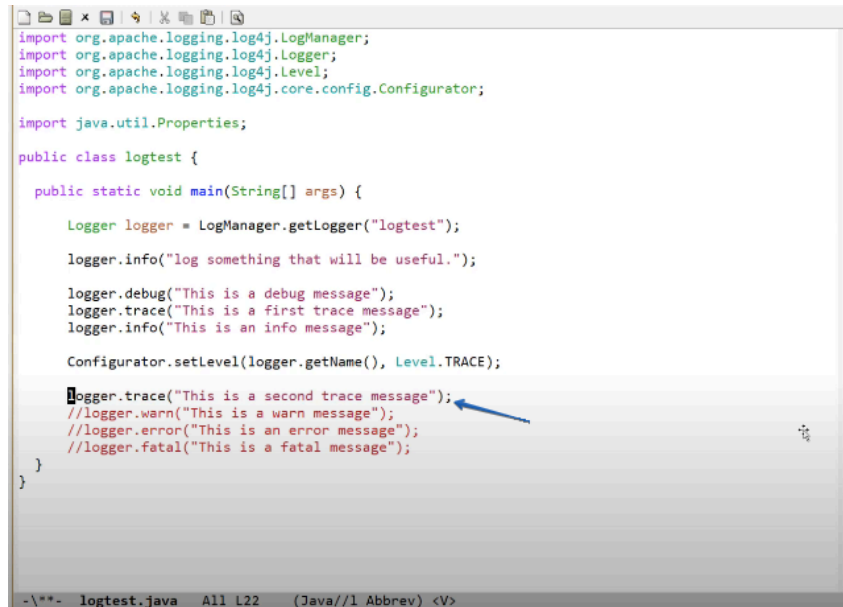
So, let us take a look at the configuration again. It says that overall this logger will accept any debug level and above messages, whereas trace and above will go only to the console and the test app log will only have info level messages. Here what we are doing is what we are using this idea of levels.

In the notion of levels, we there is more to what then what we saw, there is an implicit hierarchy of levels. So, we can say that info is a lower level in the hierarchy and debug is a higher level in the hierarchy. So the general idea is that we can say that levels higher than a certain level should not go inside a particular app vendor like we saw, such as the file system, whereas debug type messages should go to the console because the user has generated them, the developer has generated them to work that way.

We will also be able the same thing happens, for instance, for debug level messages as it is also not logged in the file as per our configuration

Because what we have said in the configuration is that the only debug level messages go due to this logger. Whereas inside it we have given a higher level called trace, but the message never makes it past the first filter.

But that is not always what you want. Sometimes you also want trace level messages. And you can do with the line pointed in picture given below:



```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.Level;
import org.apache.logging.log4j.core.config.Configurator;

import java.util.Properties;

public class logtest {

    public static void main(String[] args) {

        Logger logger = LogManager.getLogger("logtest");

        logger.info("log something that will be useful.");

        logger.debug("This is a debug message");
        logger.trace("This is a first trace message");
        logger.info("This is an info message");

        Configurator.setLevel(logger.getName(), Level.TRACE);

        logger.trace("This is a second trace message");
        //logger.warn("This is a warn message");
        //logger.error("This is an error message");
        //logger.fatal("This is a fatal message");

    }
}
```

So, we can tell the configurator instead of doing it in the config file that for the logger we are using now it should enable trace level messages. So, *the first trace message* did not make it. But by changing the level dynamically we will make sure that the second trace level message makes it. If we check again, we will see that the second trace level messages are being shown in the console. And in the log (file) on the other hand, we still have the info filter, so, nothing new happens. Notice that the log is of course, active cumulating all the messages, whereas here we see only things that happened during this particular era. This is the way we can decide that some things are useful only in the context of a single request. Whereas some things are useful in the context of the entire application lifecycle.

And accordingly, we can use the levels and filter out these messages. Since this is your first web app, and you have not really gotten into these kinds of issues before, though eventually you will understand the importance of making these kinds of distinctions as the complexity of your app continues to increase. We said that what happened for one particular level, which is the trace message.

Now, let us see what happens for something like **warn**. *Warn* is considered of greater importance than info. And one should therefore go to both destinations. But let us just verify this Java log tests. So, there is a warn message which showed up here. And if you go to the log of course warn shows up here as well. We will see that the same thing happens for error and fatal which are even more important than warn.

So, this is how a logging system helps us make important distinctions between message types, what to watch for. So, the log watcher thing which I talked about right before, that is a separate program which watches a log, and it can start doing things like count the number of warnings or the frequency of warnings, and any presence of error, fatal messages, and alert whoever is responsible for running the website.

Once upon a time, the running the website part was more or less the province of system administrators, but increasingly, the system administrators are being asked to develop some programming skills, and programmers in turn are getting asked to manage websites. Not always as it depends on the size of the organization, in something small, like a startup of course, everybody does everything.

And so, being on alert for something like failures, falls to the lot of developers and for administrators, if there are any, and then it is your skills, that your knowledge of the logging and what they are telling you, which can make a difference between success and failure in resolving problems. When I was involved in developing websites at one point long ago, there used to be things called pagers, and the pager would alert you when something went wrong.

And all of us had rotating pager duties. There is even a startup called or maybe now it is a full-fledged company I do not know it is called pager duty, which takes care of precisely this, of who should be paged when what happens. And I do not know if devices called pagers are available anymore, I imagine that people simply get calls on their phones or whatever it is that people do these days. But in any case, pager duty is one of the most interesting and important part of any developer who is responsible for a website.

Now that we have seen our simple log testing program, which generates a logger, which shows us how to make dynamic changes, and what the users of different levels are. We will use this in our application to figure out how the application can tell us what it is doing in the next part of this session.