Modern Application Development
Mr. Aamod Sane
FLAME University and Persistent Computing Institute
Indian Institute of Technology – Madras

Lecture – 24
Introduction of Modern Application Development

Hello. Welcome to week 8 of introduction to modern application development.

What we have learnt till now?

1. We discussed logins, but we have not implemented them yet.
2. We have servlets which allow us to maintain a session and last time, we saw how cookies can be used both as identification and mechanism to maintain continuity across screens which gives us one use which is sessions and another use which is logins.

Goal of this week:

1. How the view is put together.
2. How servlets are created and deployed.

In this session, we are now coming very close to finally having a full web application that is quite similar to a GUI application, but which will have multiple users and other facilities.

Servlets : A disciplined Way To Build Web Apps

The creation and deployment of servlets is a disciplined way of a general problem which we have repeatedly mentioned during this course which is, how to maintain and deploy a web app as you go ahead.

Interesting observation about most real life Web applications

Notice that web apps never have down times, they always seem to be working, the method that we are going to see in use with servlets is one of the key pieces in achieving this kind of **always-on type of development**. With web applications you are almost never aware that

anything like an update has happened. The things we are going to see today is one of the major pieces in how Web Apps achieve such continuity of behavior.

NOTE: This is very different from how desktop systems used to be, where people had to create new versions of programs and download them. However this is better now as we get operating system updates and the updates although we are somewhat more aware of it in the case of desktops and also to some degree in the case of mobiles.

(Refer Slide Time: 03:05)

Goal of the session:

1. Look at our own servlets' deployment
2. Take the first steps towards template need for templates and JSP - a particular template language

Demos:

First servlet : HelloServlet (last time we looked at HelloServlets and FormServlets as they came in Tomcat.)

This time we are going to look at what it takes to do our own, which is somewhat more involved. We build our own HelloServlet, we will use the patterns we learn while creating the HelloServlet to create a Form servlet , finally create a JSP servlet.

PART - I

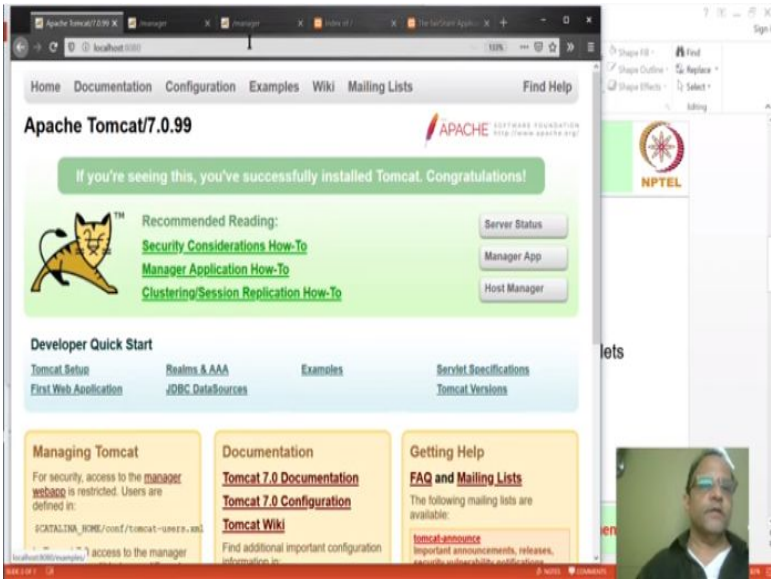
1. Understand the servlet structure.
2. Understand occasional problems that can arise while creating servlets.

(Refer Slide Time: 04:52)

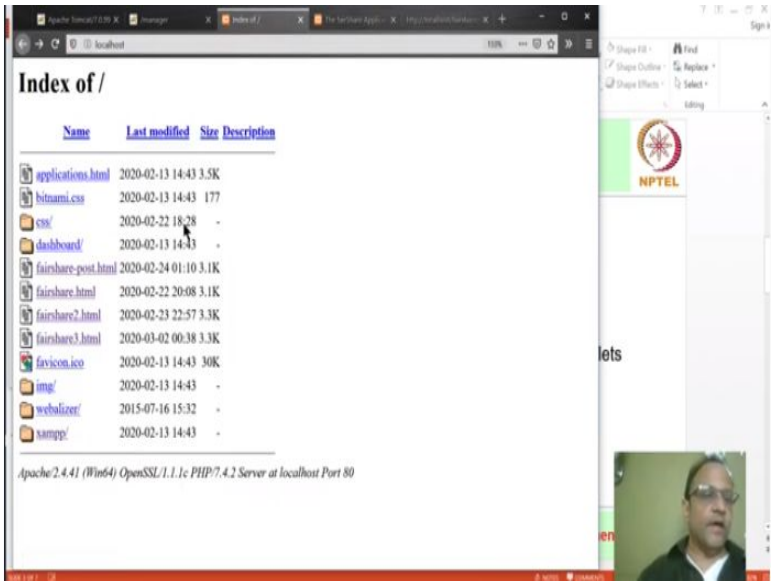
Quick review of what we have seen for Apache applications, the layouts, URL mappings and compare that with how it changes for servlets.

- Go to xampp control -> start Apache
- start Tomcat
- Open http://localhost:8080 to Run Tomcat in browser.
- Start Manager app

(Refer Slide Time: 05:21)

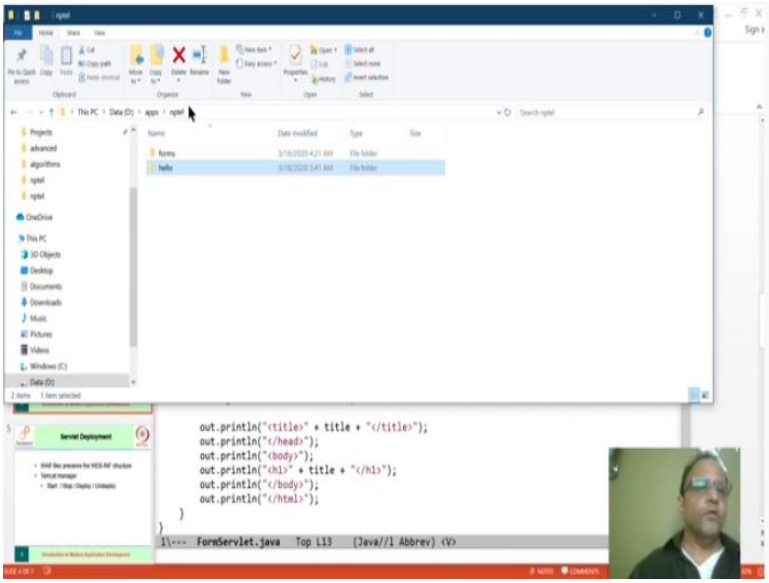


(Refer Slide Time: 06:25)



REMARK: Fairshare application consists of fairshare.html and fair share.sh, plus a fairshare.java code. Java code is usually not visible to the front-end, the front-end only shows HTML files but we know that there are 2 components to it, one of which is fairshare.sh and then the htmlInteractive.java files and class files which reside in cgi-bin. This is a disorganized system in which some files are in one place, some files are in another place. It becomes difficult to manage and so, the first thing that servlets do is, they impose a greater structure on what constitutes an application and how we deploy it.

(Refer Slide Time: 07:22)



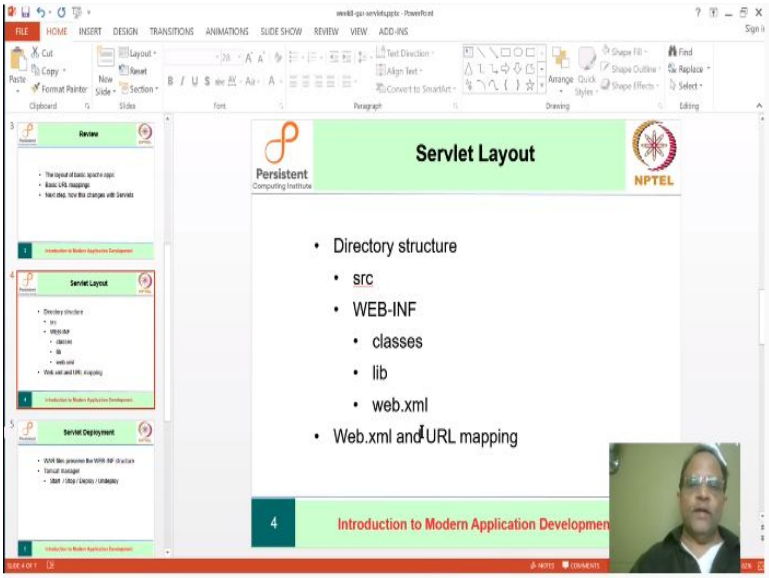
A web application in servlets is well structured using **WAR** format (Web ARchive).

Structure of a basic Hello application In servlets.

Hello application has a directory structure.

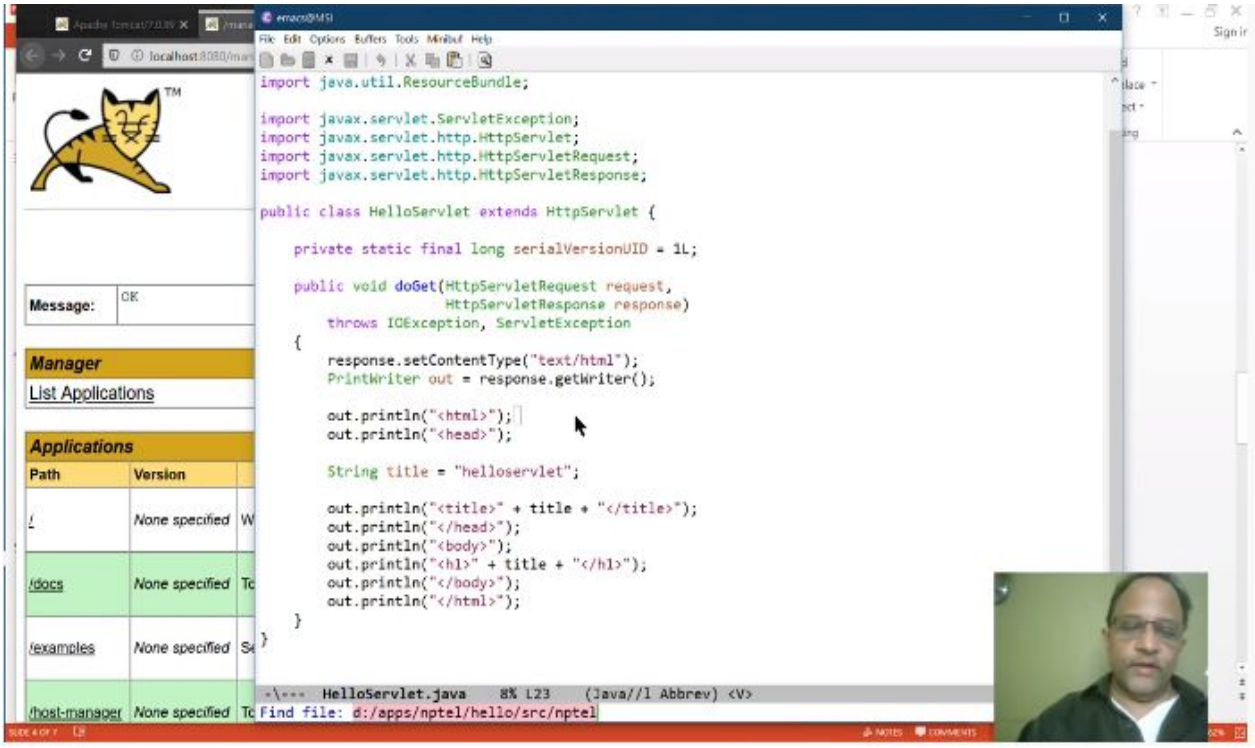
1. There is a top-level directory called nptel.
2. Inside nptel, we have created a directory called “hello”
3. Inside “hello”, there is “helloservlet.war” and two directories called “src” and “ WEB – INF ”.
4. src will contain sources.
5. WEB-INF has 2 directories: classes and lib and a file called web.xml.
6. web.xml does URL mapping : serves as a general description of how a Java file and a front-end sort of file, if any, is put together and made available to the server.

(Refer Slide Time: 08:17)



Basically, WEB-INF will contain whatever is necessary for the web application to function, which includes these 3 pieces.

(Refer Slide Time: 09:12)



Hello Servlet

1. src contains NPTEL

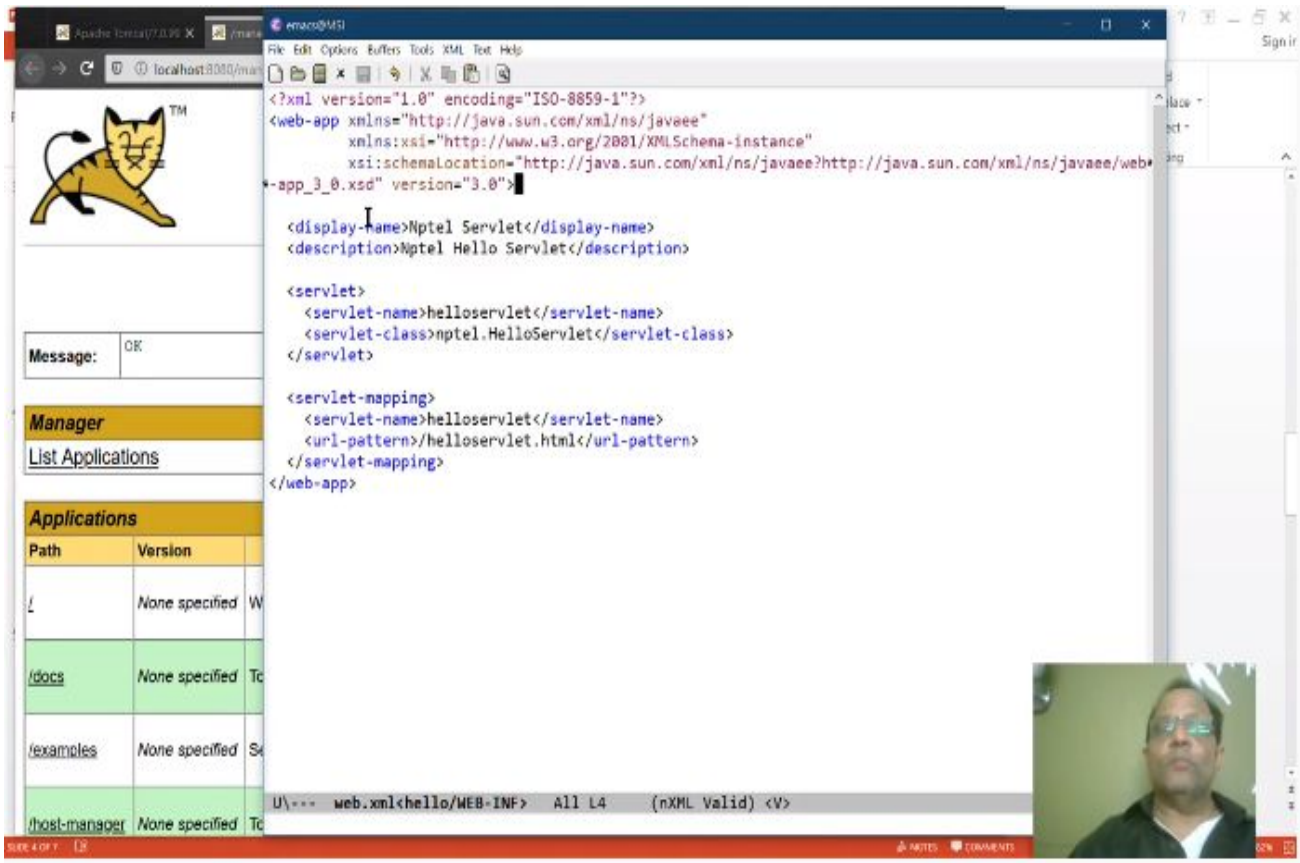
2. inside the nptel directory we have HelloServlet.

This is how as we have seen last time that the GET request is mapped into Java code and the servlet request and servlet response object contain all the data that we want.

Here is something interesting to remind you from last time, one is that in the response we set a content type and in a response, we create a printWriter and write to the printWriter all the HTML text so that it goes out in a single stream is all seen in this file.

src is where java code begins, and java packages are all structured inside src.

(Refer Slide Time: 10:51)



The structure of code in web.xml is by and large standard. They have not changed for a very long time and unless you are doing something special, the code can just be reused. So, only if you have a program that validates the xml will some of these kinds of details start mattering. Otherwise for the most part, most uses of xml are just based on tag names.

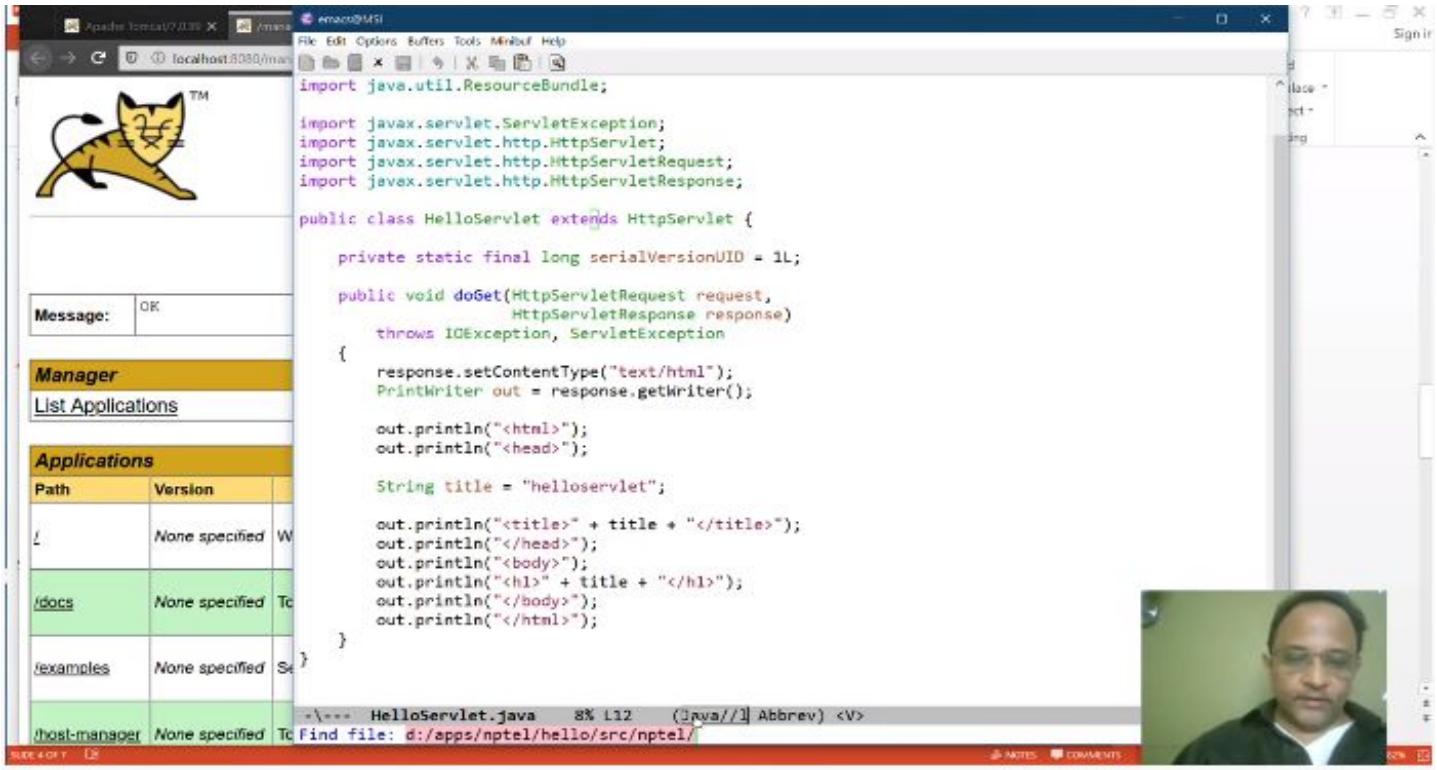
Remember : Components of a web application for the Apache case anyway included HTML, a shell file and a java program.

In servlets we just have a Java program. There is **no need for the shell file** because that is exactly what the servlet container which is Tomcat does.

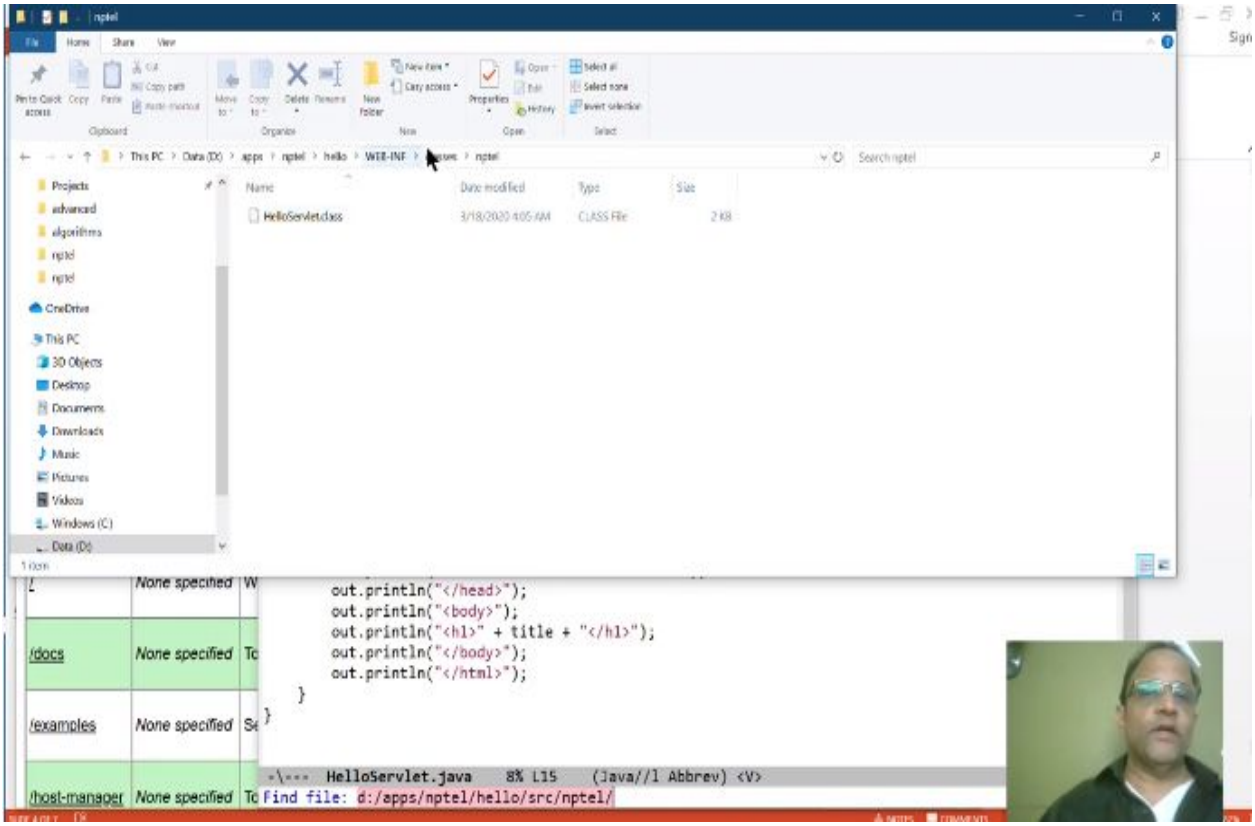
A servlet container runs the equivalent of a complete command line program but without having to deal with details of command lines. Instead, we just use a standard HTTP program which follows a standard interface called the servlet API.

1. There is a HelloServlet class.
2. So, nptel.HelloServlet here, by the way, this is a full Java class(so all the details of capitalization etc. matter). So, nptel/HelloServlet.java will have to be present.

(Refer Slide Time: 14:06)



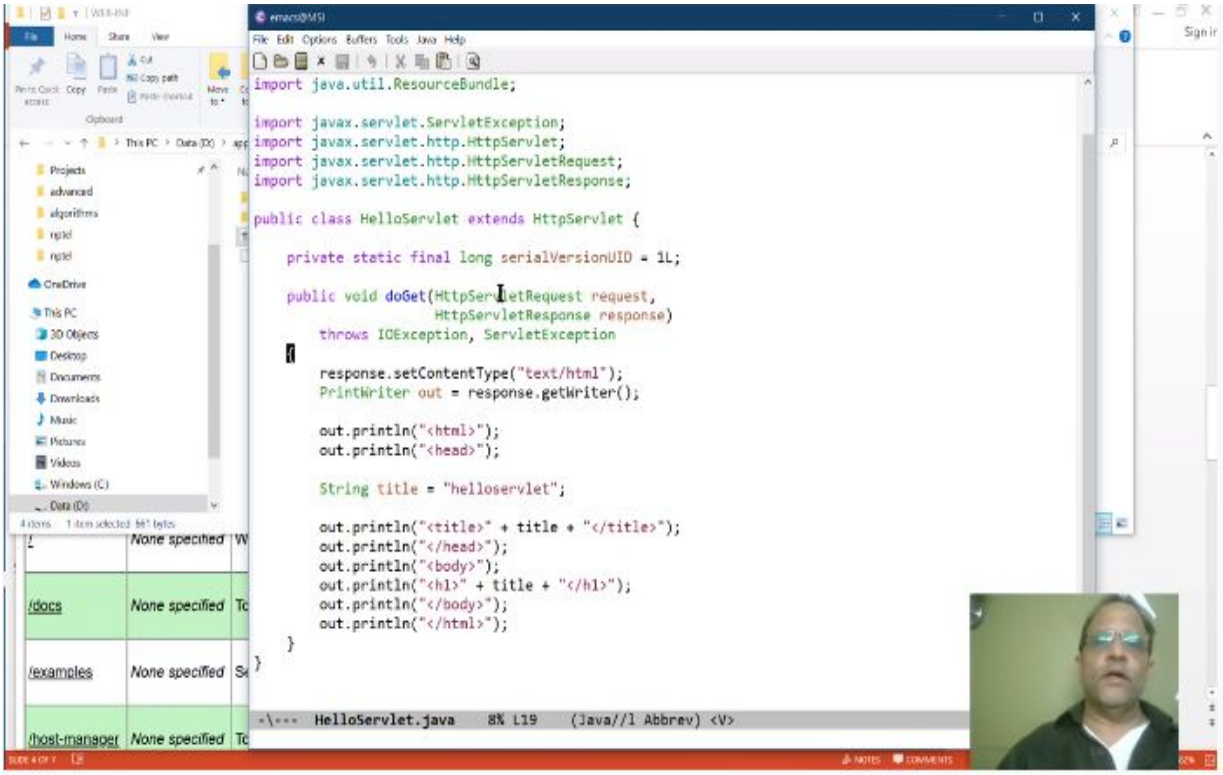
(Refer Slide Time: 14:20)



1.

2. Inside WEB-INF, you have classes and in classes there is nptel followed by HelloServlet class. So, inside WEB-INF, classes contain the overall directory structure for the distributed Java program.
3. lib in this case is empty.
4. lib is meant for jars and other standard libraries and classes.

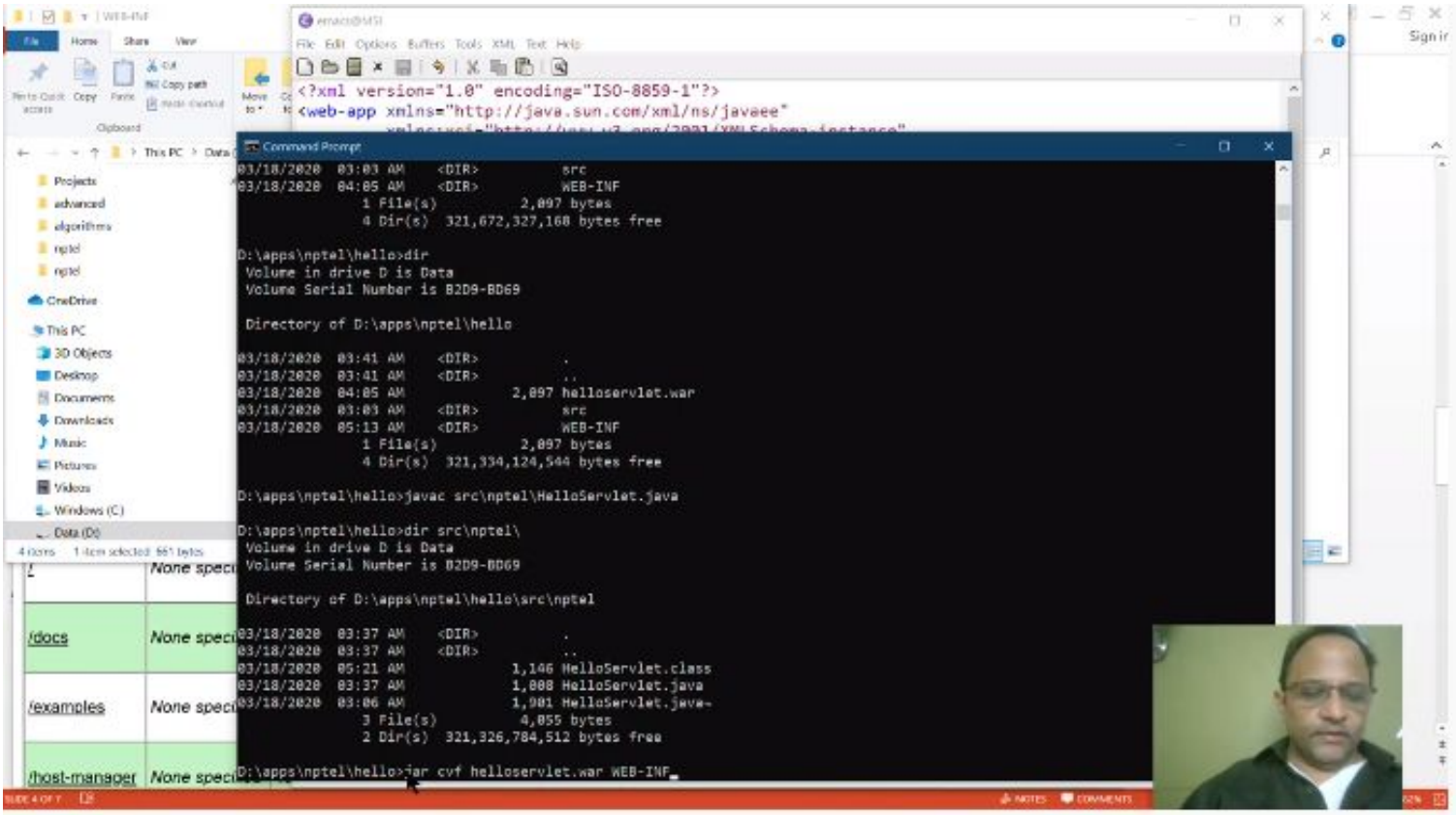
(Refer Slide Time: 14:58)



URL Mapping

Above figure shows nptel.HelloServlet. So, this tells the system that, in this servlet mapping the url HelloServlet.html corresponds to the class nptel/HelloServlet. So when you visit this url, Tomcat will interpret your visit as fetch the class and execute whichever method you have got.(In this case just the straightforward GET method). So, in Apache when a URL is mentioned usually, the mapping of the URL corresponds directly to the file system beginning with the root of the overall document hierarchy. This makes managing simple applications very easy but as applications get more complicated it becomes difficult to track 2 or 3 files in different directories, so a url pattern is instead mentioned and we say that there is a servlet name and the system takes care of mapping these 2 things for us.

So, that is why the structure becomes important. We shall see how this is reflected in the web.xml while learning JSP. **(Refer Slide Time: 16:52)**



HOW THIS WORKS IN PRACTICE.

- Run “javac src\nptel\HelloServlet.java” as shown above the compile the java file
- Go to src\nptel
- We see HelloServlet.class. If the compiling is sucessful, then a .class file gets created.
- Copy just the class to WEB-INF/classes and under the nptel directory to obtain the necessary directory structure.
- Bundle up using this command “jar cvf helloservlet.war WEB-INF”. (As shown in figure above)

  (jar is a program that will collect all the files that are under WEB-INF and put them inside helloservlet.war.)

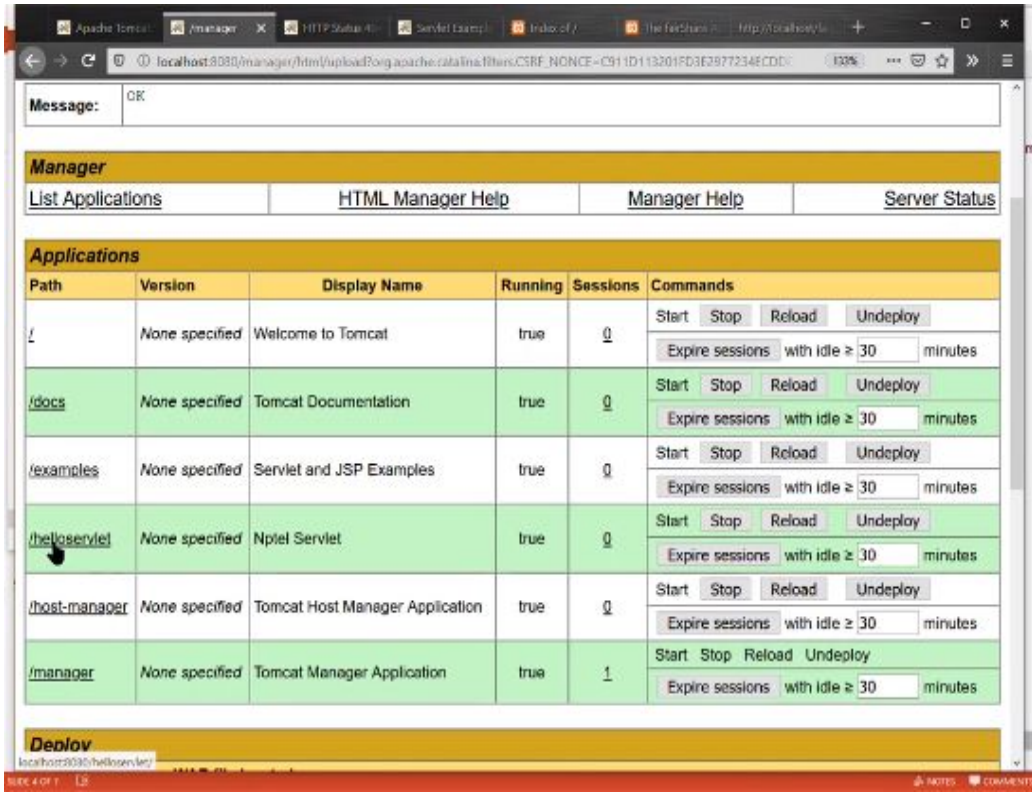
As the deployments get more complicated other directories like build and dist, but the core directories that are needed are src and WEB-INF.

(Refer Slide Time: 18:36)

If you want to see what a war file is like, go to: apps\nptel\hello\helloServlet.war.

These are just zip files. So, you can get a program like **7-zip** and look inside this thing. So, because we have just zipped a directory, this entire structure is present as is and what we get. We get meta information and manifest file created by the zip program. Deploy the entire WAR file without worrying about manually putting this file in this directory, that file in that directory and so on.

(Refer Slide Time: 19:29)



1. Go to the Tomcat application manager
2. There are 2 ways to deploy these systems.
3. Create a WAR file. (Since we have created it, i.e. helloservlet.war)
4. Deploy it. We get a new row and our helloservlet app is ready to use.

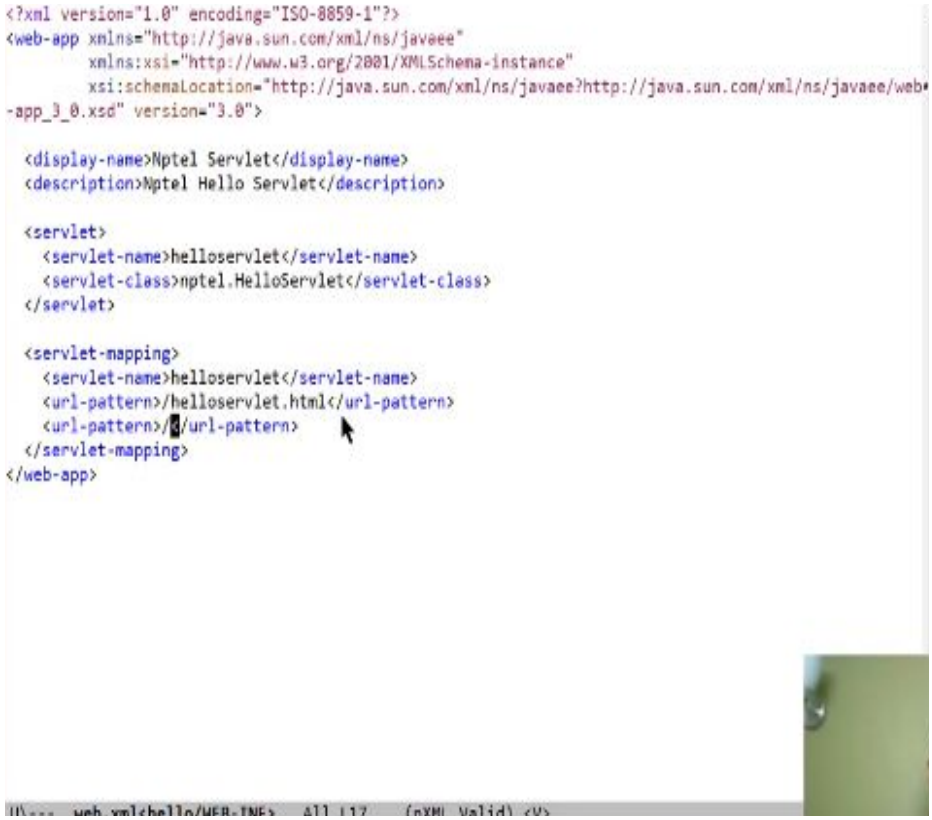
Error: HTTP status “404 - Not Found”.

(It says “the origin server did not find a current representation of the target resource or is not willing to disclose one that exists”).

This means URL HelloServlet as is, does not really work.

web.xml says /helloservlet.html should map to nptel.HelloServlet.
When the manager gets a war file, if examples were deployed by a file called examples.war then this is used as a starting directory of the URL. If we want multiple applications, they should be in multiple parts of the subdirectories on the URL. But here we have just given slash at the beginning.

So what the manager does is, it takes the war file name as the base and extends your url in relation to the war file.
(Refer Slide Time: 22:55)



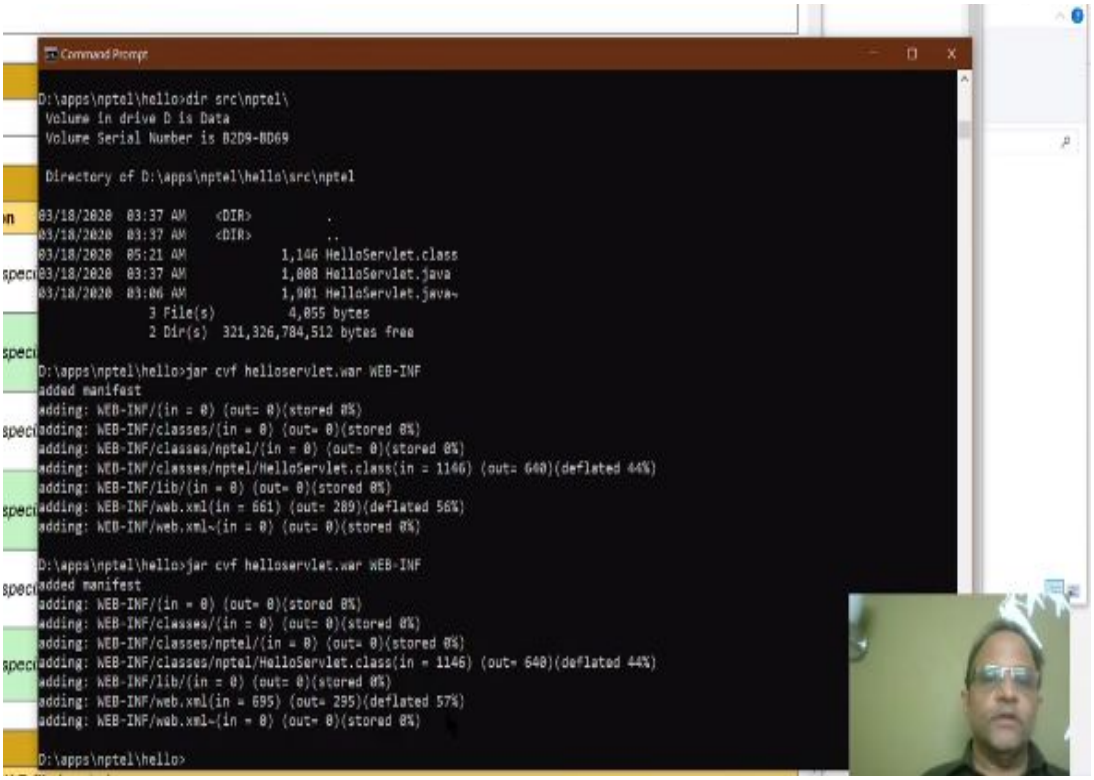
Is there any way for example to easily get helloservlet directly serve you the url?

Open web.xml, and in addition to the url(helloservlet.html), since anyway it is the final part of the name that matters, put an additional url which is simply "/" .

(look at code in figure above)

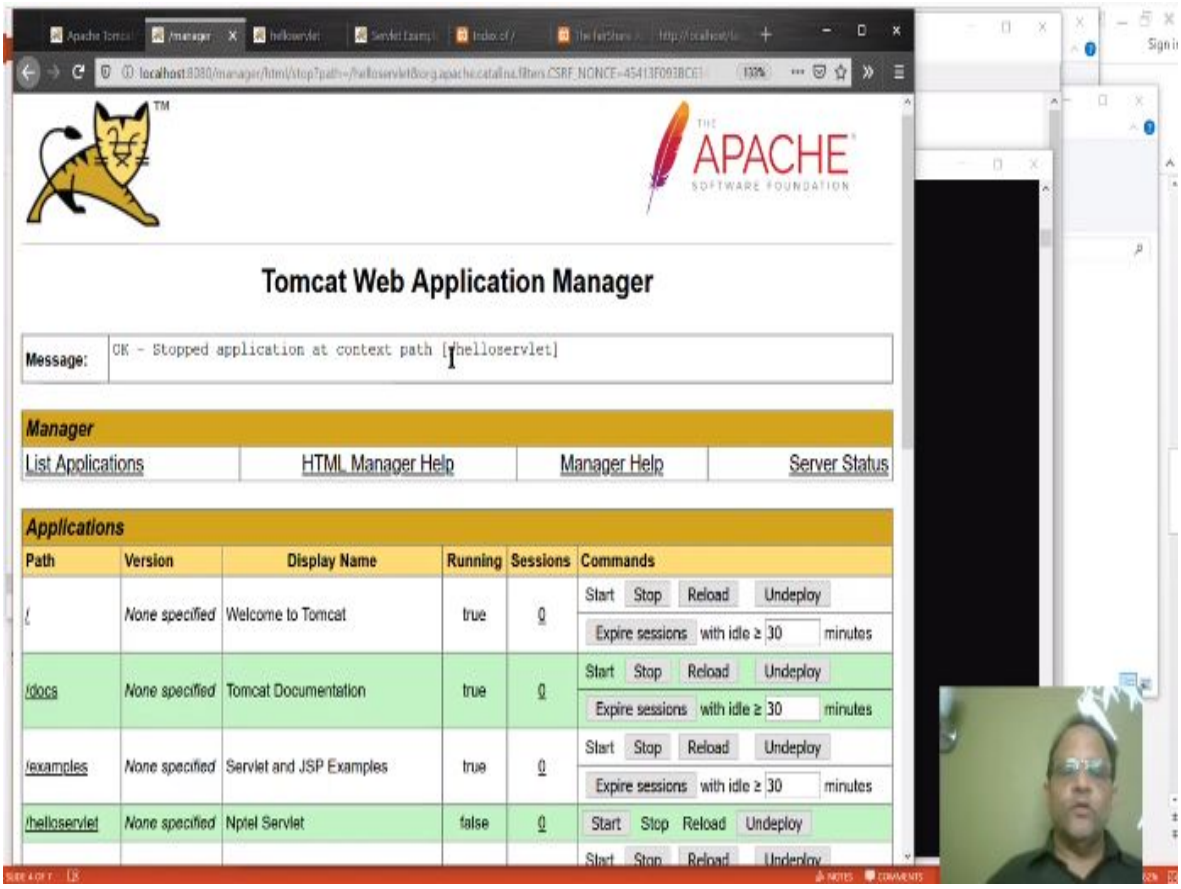
This time you will see that the deployment is a much simpler affair.

(Refer Slide Time: 23:51)



We have changed the configuration and web.xml so re-package (as shown in command prompt above) to create a new WAR file. So, this new WAR file has our new XML.

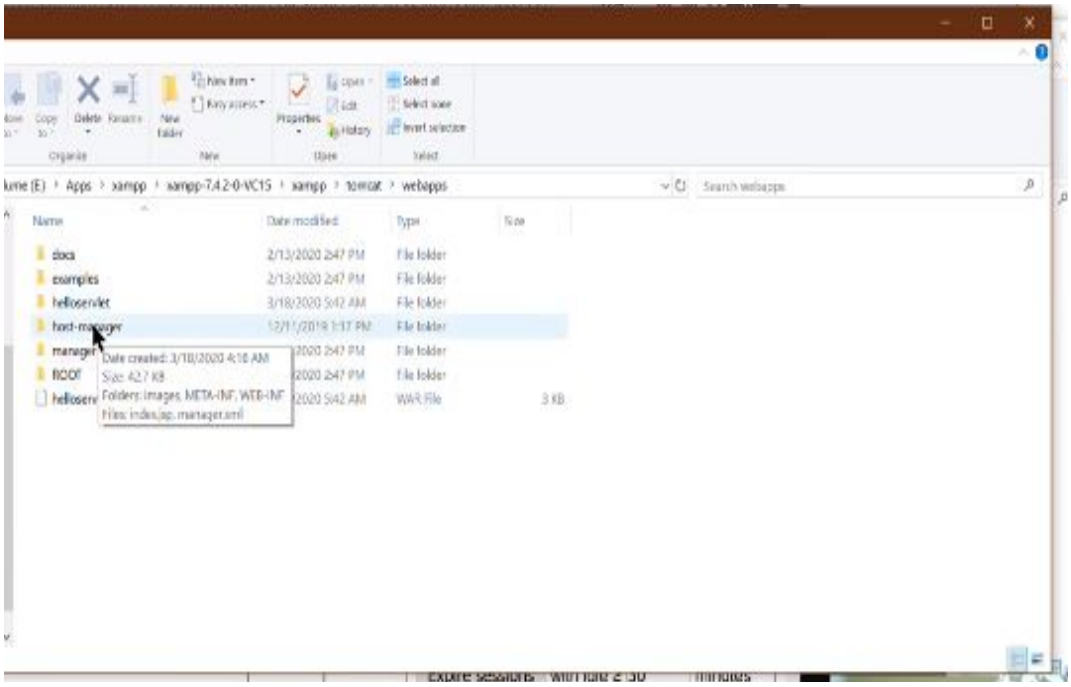
(Refer Slide Time: 24:11)



- Stop the application (by clicking Stop in the row corresponding to helloservlet).

  It returns a “context path” which is the name of the servlet because it gives you a context.
- Un-deploy the application by clicking on the Undeploy button of the row.
- So, at this point this file is gone from the system.

(Refer Slide Time: 24:37)

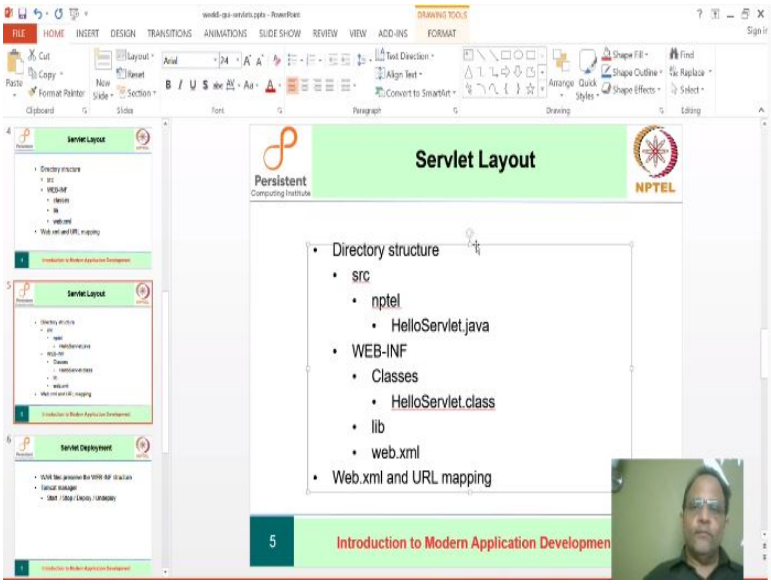


Go to xampp / Tomcat / webapps. We used to have our hello but it is no longer there.

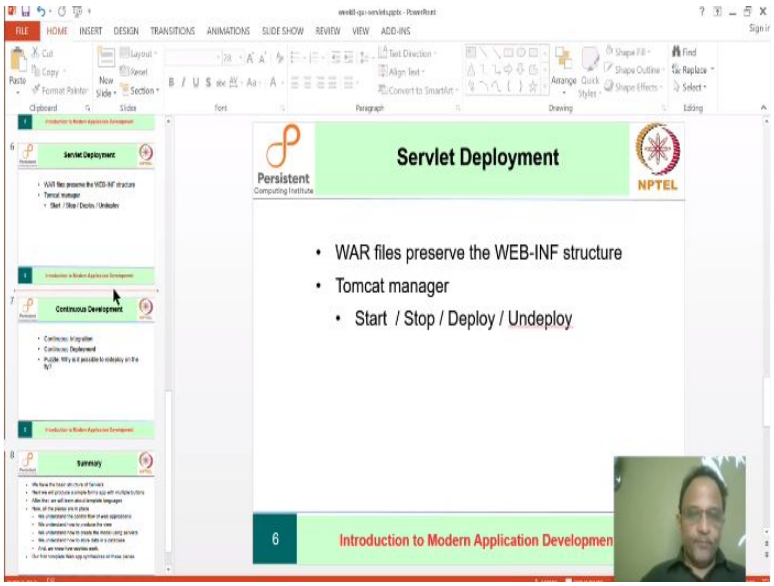
Re-deployment:

Deploy again as done earlier, helloservlet.war appears in webapps again

(Refer Slide Time: 25:54)



(Refer Slide Time: 27:28)



War files preserve the WEB-INF structure because those are just jar files and jar as we have seen is simply zip files which you can open with programs like 7-zip.

Tomcat manager does start-stop, deploy and undeploy.There isn't much copying to do. In complicated cases there are tools like ANT. ANT is a reasonably common build tool. There are newer ones that have come around which deploy, un deploy etc. and in a single shot produce the build, copy, create the WEB-INF structure, wrap it into a war and deploy, which simplifies a great deal of our life.

(Refer Slide Time: 29:10)

But the interesting part I would like to point out is that there is an explicit place where we take urls and map them.

Continuous Integration and Deployment

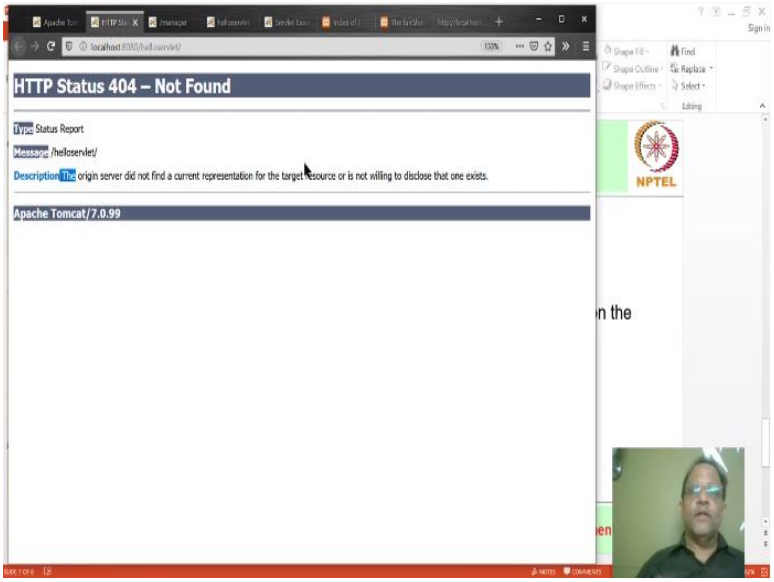
In normal applications, continuous integration and continuous development happens: Different parts of a team come together and as they produce features, the website gets refreshed (sometimes as many as every half an hour).

Not only do though all the teamwork is brought together, the entire deployment is completed very fast, this idea is called continuous integration and continuous deployment. Making this work one step is simple deployment.

For continuous integration there is something much harder, which is a good test suite, which allows you to make small changes, tests, to make sure things have not been broken and go ahead and deploy.

So, CICD as it is called is a bigger topic but it can be motivated by the question: Why is it possible to redeploy on-the-fly?

(Refer Slide Time: 30:26)



We went to the manager, undeployed and redeployed and without any change, so to speak, **as far as users are concerned**, we were just able to refresh right away.

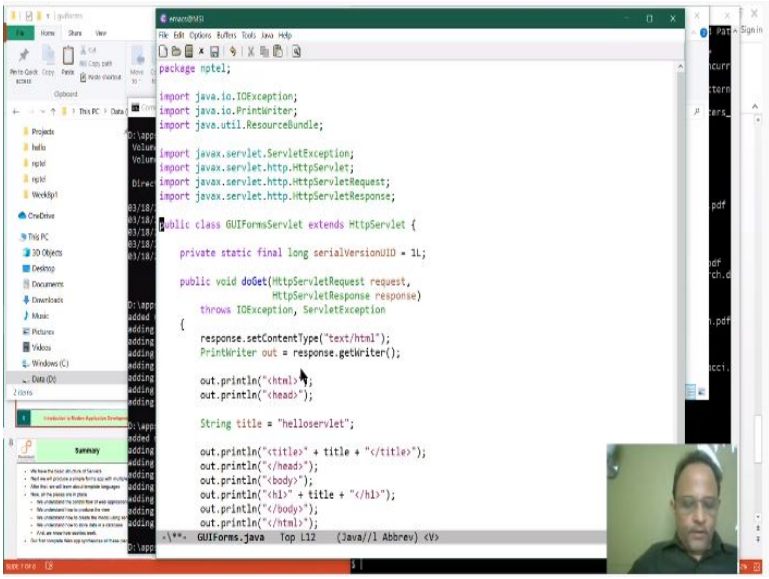
Observation: Since we are the ones deploying and using, in helloservlet above, there is no case where a request arrives at the same time that the application is undeployed.

But if we had undeployed this application, then we just cannot go to it and if we try running the file at the same time, it does not find the file. So, in that sense, there is no magical continuous deployment in our example, but if you want to do it, keep the old file as it is, before we make the new change. (We say that a certain percentage of requests only should go to the new file.) So, you simultaneously deploy the old file and the new file under a new name. Eventually, as you acquire more confidence in the new file released, increase the percentage of traffic that goes to the new file and slowly switch over. In case something goes wrong, at most a few people are affected and you usually find out that something is wrong very quickly. The simple web.xml that we have seen does not allow to do these sorts of changes.

But there are ways to do it and if we get time in this course, we will see how it is that this is achieved, it is usually achieved through things like Apache front ends and whatnot, so we may

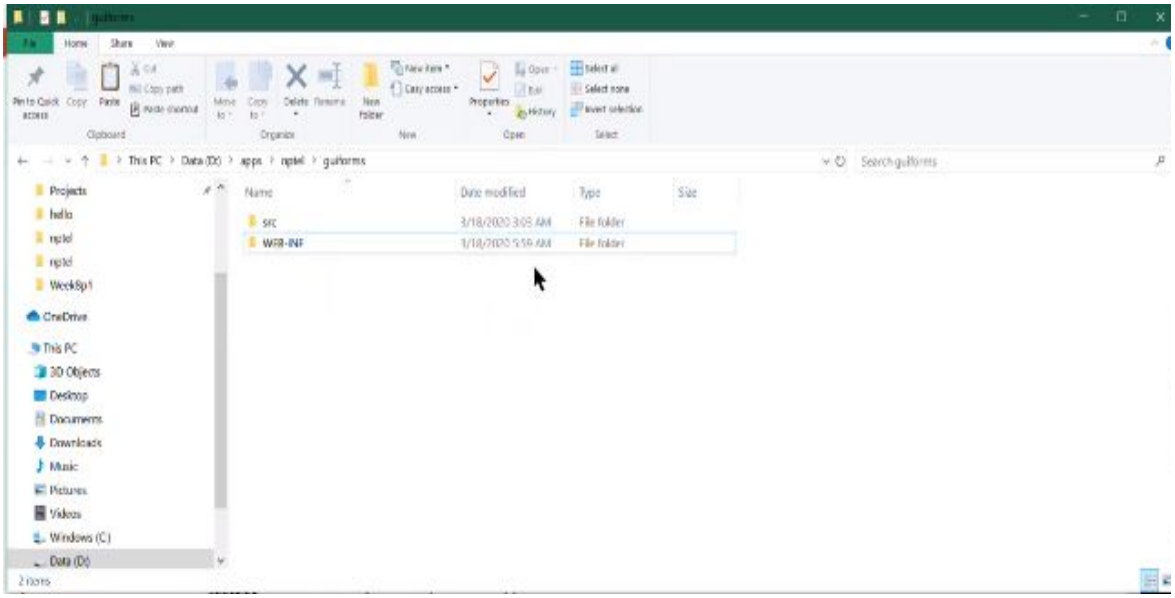
not get into those intricacies this time. But it is useful to know the general principle of how these sorts of, you know, evergreen fresh websites work. So, we have seen the overall structure.

(Refer Slide Time: 32:57)



When we develop our next app, replicate the overall structure.

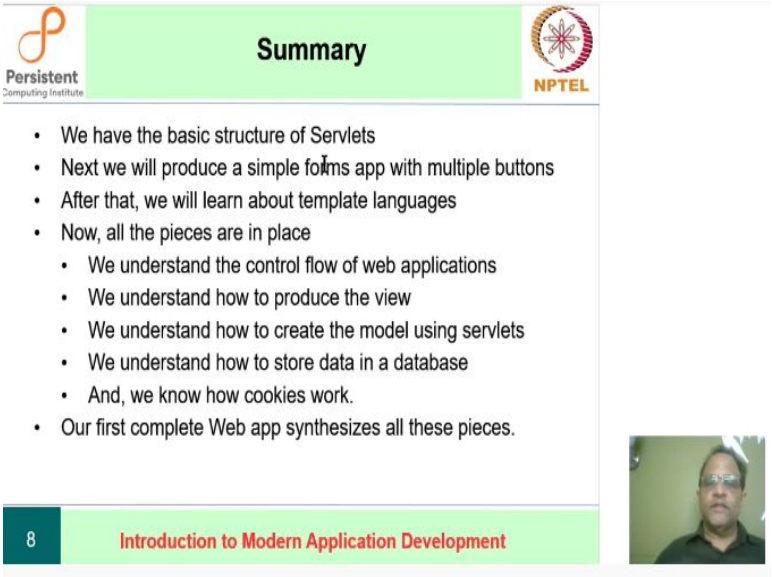
Eg: Create a directory called “guiforms” similar to “hello”, inside guiforms you follow the same structure as “hello”. **(Refer Slide Time: 34:25)**



Especially if you use IDEs like eclipse which are capable of generating their standard structures and populating them, which is usually how most people do these kinds of things when they are, they are doing new materials. However, aim of the course is to go every step so that you have a complete understanding.

These are examples of a particular way of organizing web systems. It would be just as easy to use some other tool, for example, nodeJS or elixir , but the only thing that would be different is the underlying tools that are used to put these systems together, the fundamental flow and the fundamental principles that lie within web systems does not change.

(Refer Slide Time: 36:14)



- We have the basic structure of servlets.
- Next, we will produce simple form apps with multiple buttons.
- Then we will see why something called template languages gets used and all our pieces at that point will be in place.
- We will understand the control flow of web applications, we understand how to produce the view.
- we understand how to create the model.
- We have seen examples of how the database is used
- We also have our first glimpse of how cookies and sessions work.

The complete web app synthesizes all these pieces.