Hello there and welcome to this course on Apache Maven, where we'll get to see exactly how the Apache. Maven build tool works completely from the ground up in the time that you have been around which is well over 20 years. Now we've had lots of changes to the Java platform itself, but not just that changes to the tools. We use too. And this includes of course, the tools were used to build our Java applications. So called build tools. Now there are 3 predominant build tools in the Java World, Apache and Apache Maven and Gradle and was the main goal to build tool. 3 years earlier in Java's lifetime, but now it's pretty much dead in the water. These days people have moved across to Apache Maven. The new kid on the block is greater, which you may have heard of and while it's true that it's gaining extraction. These days in spite of this. Maven is still the most popular tool used to build Java projects in industry and I don't see that changing anytime soon. It's the most commonly encountered build tool for Java projects a used all over the world and because of this is a tool most often run into so this makes it super important to understand how this tool works and how you use it effectively. And that's handy because in this course that's what we're going to learn exactly how it all hangs together so that's the context of the course the big takeaway here is not that you're learning, something useful relevant and that's highly in demand, with employers now the point of this course is to give you as much theory as you need to know as much hands-on experience as possible through me showing you how to use may have been through the command line. This really is the best way to learn the tool later on. We'll see how to use mirror in the IDE ideas like Eclipse Intellij idea but by learning how Melvin works at the command line. Will be well placed to understand how the Maven Plugins work in the IDE itself by the time we get there. If we just jump straight into that it can get confusing so even though you might not necessarily be a command line person don't worry, it will be easy to do and I'll show you how to do it step by step, so let's talk briefly about how the course is structured. The first section is the course introduction. This is the one we're in now here I'll give you a quick high level overview of Maven and show you how to generate a Java projects completely from scratch. The next section. Installation basics covers how to install Maven on Windows Linux and Mac so no matter what. type of machine you have will get you up and running quickly. My favorite section is next. Fundamentals of Maven. This really gets into the meat of what may have been really is. And what makes it tick. I don't just give superficial points, about what it does, I actually show you what may happen is doing as it's doing it on my machine. Seeing as the best way of learning for some things and so we'll get some good demos in there, too of how Maven does its thing while doing this will get an understanding of all the major concepts, which you need to know to be able to use it effectively to next there's a section on working with Maven and IDE at this point in the course. Now we've got a good grounding in how may have actually works. We can use it in popular IDEs like intelligent IDE an eclipse and that's what we cover here in this section. I've also put some bonus resources into some Cheat Sheets, which servers are handy quick reference to have at your desk or by your side, while you're learning. So lookout for those as you go along last point. This course is 100% focused on just the stuff. You need to get up and running so you can feel confident and use Maven on your own. There's no fluff or filler like you might see with other courses. I just focus on what you need to know to get productive quickly so that's a good intro to the course we can get to the good stuff now, so let's jump in now and learn all about Apache Maven. Good luck and I hope you enjoy the course.

Before we get started properly with Maven at the command line. I just want to explain exactly what that means and exactly how we do it if you know what the command line is then please. Skip this section. This is just put in for those of you who aren't sure what it is I only have a vague notion of it without knowing how to access it. So the command line is the place where we can type commands at the operating system level. If you've ever seen people listing files and copying folders around or doing any other things like that in a text based window then that's the command line, you get to the command line in different ways depending on whether you're using Windows. Linux or Mac. In windows, you can go to the start menu. Click run and then type CMD and hit enter you should get a black box where you can type commands in for testing type die are followed by enter if you get back a directory listing In other words, a list of some files then you're in an all good on Linux. There are various ways, but typically you right click on the desktop and they'll be an option like knew terminal session or something similar click that and you're good to go. You can side LS and enter to get a directory listing just to test out if it works. If it does again all good. Finally, on Mac. Just go to applications and inside the utilities folder in there, you'll see terminal click that and a box should appear type. LS Mac is pretty much the same commands of learning systems and again if the directory contents appear that list of files you are in and all set so armed with this knowledge. Now it should prevent you getting to the next video is thinking. Where did the big black box come from and what you're typing in what is that you know now it's the terminal app on my Mac otherwise known as The command line now get the command line open and follow along with me in the next video is as we learn all about the fundamentals of Maven together.

In this video, we're going to see how we use the mapping tool itself. We're going to take some first steps to get us familiar with how we actually invoke moving and work with it as a tool and the best way of seeing how to do this is to actually create a project so let's create a brand new Maven project to do this we're going to use the Maven archetype plug in and we're going to use its generate goal. So I'll explain what that means in a second, but first of all let me create the project so if we type Maven. OK type generator Then never more basically go away and get all of the archetypes out of which it can create maven projects so that I can type in Maven is basically a template for a project consisting of various files and folders, which can be copied across and token replaced and form the basis of the new project so as you can see here there are quite a lot of archetypes available. There's 1818 so that's a bit overwhelming from the outside to look at for an introductory video so you can actually filter. This list, so we're going to filter it with the command line switch so if I just. Came out of that and clear the screen, then we can filter the list. This time to just give the what I call the standard Maven Archetypes sort of the words does supplied by the Maven project itself, so like the out of the box ones if you will see if we type Maven archetype. Generate again and this time with a minus D switch. Whenever you want to pass switches to Maven. You pass them with a minus D and so it's just like passing a standard Java system property in that respect so it minus D filter equals and then this is going to be the circle group ID of the Archetypes. We want to filter on so this is auger patchy. Maven Archetypes and to tell Maven that where filtering on a group ID we put a column. Afterwards. Whenever you are referring to artifacts in Maven, you have 3 constituent parts to the artifact. You have a group ID an artifact ID in a version will cover more of this later on, but basically typically these are separated by colons when they become parameters to certain commands so if we just hit enter there. This should filters down to a much more manageable list. And that's a lot better, yeah, we have 12 to choose from, which is much better, but either way. Whether we get the large list or whether we get the small filtered list. You'll find that the Maven system defaults to what's known as the quickstart archetype Which is the most basic archetypes that Maven offers to start projects out of so if we select number 9 as the default. It's asking us, which version we want to use so we're just going to use the latest version. So it defaults. They're ready to number 6, which is version 1.1, which is the latest version of that particular archetype. Next we define a value for our group ID of the project. So here we're just going to create a group ID of com.zention.training.demo The artifact ID is the name of the artifacts were going to create so let's just create an artifact called my app. And the version will leave this at one zero snapshot. This is fine. For now, and finally the package will leave the package as the same as well. For now, so if you say yes to all that it shows you the settings. It's going to build the project from just the yes and there, you have it. It's created you a new project named out of the artifact ID that you specified called my app and that's available for further development, so will check out what's in there in subsequent videos and get signed developing but that's how you create a project in Maven with Maven Archetypes.

In this section, we're going to see how to install Maven onto your development machine now installing software isn't the most interesting thing, but it is important to us so hang in there will get to the good stuff soon enough. We just have to get Maven up and running first before we can get to that. Now, the most common way to install Maven used or what I call the zip install method this is where you download a zip file from Maven website on your local machine unzip it and configure environments. A point to it, so that you can use it. This method can be used on all systems. So Windows Linux and Mac and it's pretty much guaranteed to work. I'm going to show you how to install it using the traditional method for Windows and Linux systems since we can install Maven pretty much in the same way on both of these environments without getting involved with the many different package managers available on the various Linux distributions. But on Mac just to mix it up a little. I'm going to show you how to install it using a cool tool called SDK Man, which allows you to install all different types of tools for the Java platform may have just one of these you can then use SDK management salt other Java packages later down the line. If you like too. So it's a good thing to have installed and personally, I wouldn't be without it. So all good stuff anyway enough talk. Let's get may have an installed onto our systems now.

In this video we're going to see how we can check if Maven is being installed onto a system, which is really straightforward to do before we do that will just double check. We have Java installed so it Maven runs on Java, which is the reason that Maven is a platform agnostic build tool in the sense that it can run on Linux Windows, Mac or Unix due to Java's write once run anywhere ability so First off. Let's just check if we have Java installed so if we type Java minus version and hit enter then this tells us that we have the 1.8 version of the JDK installed. So the next thing to check is that we have Maven so the way you interact with Maven is the same regardless of whatever platform you're running on an it's basically to use the mvn Command. So mvm in is the command line interface to Maven and and if we type minus minus version afterwards and hit enter then we should see the version information message, which tells us that Maven is being installed on this system so just a couple of things to point out here it tells the version of Maven. We're using which is 339, which is the latest version. As of this recording. It also tells us the Maven Home Directory. This points to the directory where the Maven tool is installed to so in this case. I'm running. SDK man and it's in that particular directory there. It also tells us which version of the JDK. We have which is 1.8 as we've just seen. And tells us where the Java Home for that. In other words, what version of Java home. It's going to use as well as some information about the platform settings. So the local character Encoding and the actual operating system that were on this is how we determined that mirrors installed on this system so at this point, you can actually create your own projects and start to actually developing projects from Maven.

So, in this video we're going to see how to install Maven on a Windows system so at first up. We need to check if we have Java installed. The reason for this is that Maven runs on Java, which is what gives it its ability to run on multiple platforms. So we need to ensure that we have JDK installed on the system. We're going to install Maven So to check if we have Java installed will just go to a command prompt. I usually have a shortcut on my desktop to the command prompt you find quite useful, but if you don't have it, you can just go to the start menu. Programs. Accessories and then you'll find it there at the Top. But I have it so it cancel out of that. So if we go back to command prompt simply just type Java minus version. And here enter. Then it will come back with the version of the JDK, but we have installed as you can see on this system. We have 1.8 of the JDK. Installed so that's good never requires at least 1.7. of the JDk runs so we're good to go can install method now at this point. So I'll just cancel out with them and open up Chrome. So we just go through the address bar and type maven.apache.org that's the website from Maven itself. And hit enter. Then here we have the Maven website, so we'll see the download link. On the left NAV so let's just click that. And with the first things we see is that we've got the system requirements that we need to have a JDK. installed as we've just seen we have the JDK installed so we're OK for prerequisites. So we just Scroll down to the file section at this point, you'll see, there are 2 distribution types, we can download. Download the binary distribution or the source distribution, so we're going down with the binary distribution because that's the. Version we need to use if we just want to use my phone as a tool that source distribution has the actual source code of Maven itself. That's useful if you want to develop Maven or write your own plugins, but that's beyond the scope of this course. I'm just using Maven at also will download the binary distribution, so click on the zip file link to download that. So let's download it now, so just. Come out of that window. So once you've downloaded math and what you need to do. Next is to copy across into a directory where you're happy to have that insulation on your system? So we made what I tend to do is if I'm on windows systems. I like to put all of the job at tools. Inside the same Top level folder, which is usually in C:/Java. I think it keeps everything nice and self contained. And keeps the Java step separate from other programs that I've got on the system. So if we just open up windows Explorer now. Navigate to that C Java folder. Using then I will of course, the JDK that's OK, but we're going to put the Apache. Maven installation in here as well. So if we open up another window another Explorer window as well. Navigate to the downloads folder. Then will just drag the Apache Maven zip into the C:/Java folder that's copied across now. So I can close those to explore windows down. So let's go back into a command prompt. And if we just CD into the Java folder. And just double check that we've got the file there. Yeah. The deal now we've got the file there. So Next up We can unpack the archive with a Java command, so as long as you've got Java home set. And you've got Java home been on your system path will be able to use the Java command if not, you could just uncompressed with Windows decompression tool. Win zip or 7 zip or whatever anyway I've got it here, so I'm just going to type JavaXVF which is extra extract V for verbose In other words print out the listing And after specify the file. And the final here is Apache Maven, 350 pin.zip. Enter then you'll see extracting. And we'll just delete the zip file now 'cause we don't need that anymore, so just tell the name of the final. Enter. And we'll just double check that that's gone now, so. Serious and then just dealing all the directory. Yep, that's gone. And close the command prompt down. And next I'm going to go and set environment variable so empty windows. environment variable so if we just right click. On the computer on the desktop and select properties. Then next step just click the link which says advanced system settings. And from the pop up, which appears just go down to the environment variables button and just click that. And now at this stage we have 2 different types of environment variable we can create we can either create a user one. Or, a system 1. And so user ones. I just applicable for the current user and system ones are applicable for. I'm all uses I'd like to set system. One just in case I have any other accounts or any other. Show machine with any other people who are developers who want to use the same tools that I use. So I'm going to set as a system variable here. So if you just hit the new button to define a new environment, variable. And then here in the text box where it says variable name. We're going to give it M2\_home. In all capitals. This is the standard name for the environment. Roebling map and so do use that name 'cause of the tools. And sometimes require that specific environment variable so stick to the conventions and for the variable value. We're going to basically copy and paste. The The path to the Apache Maven. Fold away just don't work. Just unzip too. And I'm going to paste in there, so if we just go up to Explorer quickly. Navigate across to that folder. Which we just expanded the zip file into And we're just going to go into the title Bar of it straight. Just go into the address bar there. And select that to copy. And then go back over here and do a paste. To the value of section then just click OK. So now we can see we've got them to \_home environment variable set their next step. We need to basically append. The Bin Subdirectory of that onto the system path. So if we just Scroll down here. For other environment variables, we can see the path variable. So just click the edit button here 'cause I'm not defining a new environment, we now we're editing an existing one. And we're just going to append. To that path in the value text box I'm going to pin semicolon. Percent M2\_home percent. Backslash bin. And then hit OK. So this means we now have the bin subdirectory, which contains the scripts for Mevan and so the MVN.CMD. Script that it's got those defined and ready in the system path so we can execute them in command from. Whichever directory we happen to be in so just close that. And now let's go back into a command prompt and if everything is worked. We should just be able to type in MVN. Minus minus version. An here enter and if we have, maybe installed correctly. We should see the version message. So hit enter. And we do you can see it gives information about the Maven Home Directory Which it's got defined which is where we unpacked The zip file to which version of the JDK's running on a few other bits and pieces as well. At this stage, you're ready to start developing applications and doing builds with Maven on windows.

So, in this walkthrough. We're going to use Ubuntu Linux, which is at version 16.04LT S. But whatever Linux distribution, you use ultimately the process is going to be the same if I just log in here. So we're going to install Maven in the traditional way, which is pretty much to download the zip from the. Maven website extracted through directory were happy to have it on the local system separate environment variable to point to that. Directory and then ensure that that directory is been sub Directory is on the system path so we can use Maven tool And so it's probably easier to demonstrate visually than to explain it, so we'll just get started so if I go to the Maven website. Mavin.apache.org. Then just go to the download section. And from here, you can see that we need to have JDK installed which show we have already and you can just go down to the final section. And you'll see, there are 2 types of distribution. You can download so we're going to download the binary distribution. This distribution, you can also download at some point if you like but that contains the actual source code of Maven itself. We're just going to download the binary distribution. So we'll click on this target, she said fine here to download it. Just say OK to that. And you can see it's downloading. And it's downloading, which is great, so. So, just close this down. I'll go back to the terminal so now you have the option of installing to /opt over /user/local. directory path by preference with all Java tools is to have them pretty much in a /opt/Java folder. So this keeps all the tools nice together put them all in one place and it also means as well. If I want to just. Uninstall something all I gotta do is doing RN minus Fr on the particular subject, which contains that tool. So we're going to install it to /opt/Java so First off will check if we have that folder. And if not will create it so. Do we have opt? Yes, we have opt but we don't have the Java sub-directory so I'm just going to create that now with Pseudo. So now we should see we have that directory. Yes, we have the job of directory that's good, so we'll just CD into there. And clean the terminals get more space. So now we can just unpack that into the current directory. And we should now see the Maven folder there. Which we do so let's just CD into there and just have a quick look we should see a bin Sub-directory with in there and if we just have a look at the files in there. Yeah, then we can see here that the Maven shell scripts that we need all have the executable flag set. Which means that we don't have to do any setting up that flag to make them executable if we did we'd have to do a share more do plus X In the file itself to add that execute permission, but that'll looks good so that's fine. So the next thing to do is to add this path to the profile. Right, which runs when you log in this going to be a dot profile in your home directory or .partialC depending on how you have things setup. I'm going to put in the .profile. So we need to ensure that that path, which is op Java Apache Maven, 350 is assigned to an environment variable called M2\_home. And then we just make sure that empty\_home/bin is appended to the system path and that means that we should be able to run Maven from inside any directory on the system so let's just do that now clear that out. So if I just copy this. And then going to the profile. But jump to the end. I'm going to append 2 lines here. And export M2\_home. Equals and then paste in that path we just copied. And then finally. We're going to. Append to the system path. Enterhome/bin and then save that. So if we just log out and log back in again. Now I log back in. Now, if we go to a terminal. We should be able to type in MVN. --version. And we'll see them Maven tool has been installed, and it has. So from here now it says we're in Apache Maven 350. It shows us the Maven Home Directory, which is where we installed it too. And also shows the JDK that's running against which is 1.80131, one at this point, you're ready to start creating applications. And doing builds with Maven.

In this video we're going to see how we can install again Maven set up on our local machines. Using a really cool tool called sdkman. sdkman tool, which enables you to install various different tools and technologies, which are related to Java technology. So for example, things like Griddle Apache Maven, Groovy, those kinds of things. Uninstall them onto your local machine. So I've got a Mac. SDK works nicely with Mac and also with Linux with windows. It's also possible to use it. But you have to do a bit of extra configuration and it's outside the scope of this course really. But you can look at how to do that, if you're in a Windows machine using Cygwin so SDK. The really great tool because it enables you to not only download and install the different tooling that use in Java. you can also maintain different versions of different packages as well. So we'll see what this means in a second but. It's kind of like a package manager for everything to do with Java, so anyway, so let's first of all go to Chrome. And we go to the webpage sdkman.io, you can see it's already in my bookmarks. And this is the web page for sdkman. So you can have a read about what it's all about down here. But we're going to do is we're going to go to install. And to install sdkman all you gotta do is to take this call command and execute it in a terminal session. So now I've got in the clipboard close down Chrome and get a terminal session up. Maximize that. I've got a useful tool called spectacle, which is a window manager for Mac, which enables you to arrange all the windows, so now I can just. Maximize that and if I paste that command, which was in the install web page. You can see here it's getting. It's doing HTTP get request. On get the sdkman dior and that it's running that through bash and what that's going to do is. The contents of that web page is basically a bash script, which is an executed by bash and inside that script. It has all the commands to be able to set up sdkman onto your machine, so if we. Enter. We can see. Now it's downloading sdkman onto the local machine. It searches for a few tools that it needs. First of all, so unzip zip, Colon set And now you can see it's been set up an also what it's done. If we look in the .bash profile file and we see it's also created this extra line here. And this is basically what establishes is the environment for sdkman. See you might have a .profile.bashRC. Or not much \_profile in your home directory and you can feel free to move this to whichever. Start file you want, but just know that this sets up the environment, which sdkman requires to function. So if I now command queue out of that and then go back into terminal so we're starting a brand new session. Maximize the screen again and now if I type SDK. This is the way you interact with the sdkman tool. Online tool to notify type SDK and hit enter you can see that the tools being installed, and there are several things you can do here. You can install so we can install a particular package can also uninstall one we can list packages that I'm going to local system. Indeed, we can list packages that are available to install on the system and we can do use which also means that we can. Configure a specific tool and a specific version to use on the system so if I want to see what can be installed. for example command key basically clears that and I do SDK and is or list. Now we can see all the. Different packages begin install so here we can see we've got and which is another build tool ask you talk to J, which is a documentation tool. And there's a few other bits and pieces be sail. Along crush and never heard of those you might get some kind of obscure ones. And that's all fine, but the main point is that you do get the main tools as well so if I search now. With a /slash I search for Melbourne. And hit enter we can see here we've got them. Evan tool so Maven and in brackets. This is the latest version, which is available for install. 6 or 2 I also shows us here, you'll notice. It also shows us how to install the tool as well. The command so SDK install Maven so I'm going to copy that and I hit Q to come out of here. Command key to clear again and then press that command in SDK install Maven. You can also do SDK LI Maven. Man has these short form L is for list. I for install and so on, and I hit enter. Now you can see it's downloading moving from the Apache Website. And it says it's now installing and it's also set merven 362 is the default, which means. If I now go MVN --version. I should expect to see the version message saying 362 is being installed. And as as you can see here, Apache Maven, 362, so I can then use Maven in the usual way to create projects and so on, and so forth will work with different maven projects that already exist. clears again. I can choose to list Maven. And we can do SDK LS without the tool, but it's basically showing you its listing the catalog but if you do. SDK ls with the tool and it's installed on your system then it's going to show you which version you have installed on your system And which of the versions are available to install so here for example, we've got 362 and there's a greater than symbol next to it, which means currently in use In other words that's the one that's the version. We're currently using now we also have the star. Which means it's installed so if you wanted to install a different version we could also do SDK I? for example Maven 360. That, it downloads 360 and it seemed we want it to be the default version are saying no to that. Command came now to clear and now we now list. Maven will expect to see 2 entries highlighted an we do 362. As a little greater than symbol next to it, so that's the one that's currently in use for 360. Is also installed so that means we can also use that so if we want to use just to recap So we do --version. Now it should show us 362, which it did before, but if we wanted to use version 360. But whatever reason, we could just do SDK use Maven 360 and now if we do Maven --version. We see that we're using the 360. OK do things to show you as well is basically where sdkman installs the various packages so sdkman has a home directory. Inside this .sdkman folder inside your home directory. And here we can see we've got a few folders, we've got pin, candidates, etc XT and a few others pin contains the scripts for sdkman. You never have to go into here. If there's only one script as it happens. But what is interesting is if you're going to candidates then you can see the different tooling All the different tools that have been installed or packages. If you will on your system so inside there. We can see we've only got maven here. Because we installed but inside Maven with and have the different versions, so we've got 360 and 362., which we have so if we look inside. 362 then this is the actual. Unzipped never installation from the Apache Website, so this is the actual Apache Maven version that will be used if you do SDK use. Maven 360. That should give you a good introduction to how to use sdkman. Feel free to have a look. Its website investigate some other packages and you'll find it's very useful for installing different Java tooling anyway. But it's a really great addition to your Java tool box. As you can see.

So, in this video we're going to have a quick look inside the Maven installation itself and just talk about a few files that are there, which are useful to know about to do this you need to find out where you've installed Maven too. So if you're not sure or if you have an existing installation If you type. Maven --version and hit enter then you'll see you have a Maven Home Directory, which is defined so if you take a look inside that Maven Home Directory, this is the actual installation directory where the Maven, in command is pointing to it. The minute so if we have a look inside here. Now, when my system. This is actually a symlink so if I follow the symlink with a capital L. It'll show me inside, What's actually inside that directory so this is the contents of the Maven installation itself, so this is the Maven tool basically so all the scripts are needed to run the configuration files. The libraries that kind of thing you don't normally have to look inside. Here, but sometimes it's useful just have a quick check to see what's in there, so if we have a look inside there. In the Finder were just a couple of things I want to point out which are useful to everyday use of Maven and most of this stuff here in the installation itself is the internals of Mavens and so we don't really care about it. However, stuff that's relevant to the user Maven and is in the bin that said directory so we have these 2 commands here, Maven and Maven.CMD the Maven command without the CMD suffix is a shell script, which launches the method and command line tool itself for Linux Mac and Unix. And the Maven.CMD. Script is about script, which performs the same function for Windows users so if you're on windows when you include this bin sub-directory on your path. It's the Maven.CMD file that's run and for Linux Unix and Mac users is the ambient command, which is run there. Also, MVM here. This is if you want to run the Maven tool with debug options such a PDA debug options You could establish remote debug session to step through the build few wanted to do that, so that can be useful sometimes if you're running. Unit tests and you want to step through some tests ordering the contain walking that kind of stuff. This is the thing. I want to talk about is the conference of directory. So this has a useful file in here called settings. .XML and this is basically the default settings file from Maven itself, so we just have a quick look at this in sublime text. Then we'll see that it's in regular XML file so we can see the settings tags here and there are various different sections, which you can uncomment to customize. Some of the default settings of movement so for example. There's a local repository thing here, so usually Maven creates a local repository cache in the home directory. .M2 repository. You can change it here. If you want. For example, there's other stuff as well relating to plug in groups service also proxies are quite interesting if you're behind a firewall are you in like a corporate environment. Then you may want to invite you may have to configure a proxy here to get outside of the network to the public Internet to the Maven repository, which is used where dependencies are downloaded from. So yeah, there's various different bits in there, you can have a look at and will cover how to customize those in another video, but for now, that's all I wanted to talk about just to give you a bit of a guide to show you what's in the method installation directory itself.

Now I have to say this really is. My favorite section because of the next few videos. We're going to learn from the ground how Maven works and how it's used you learn all about the project object model or pom as it's known which is how maven projects are described you'll learn about build life cycles and Plugins and how they work together and most importantly, this is really the most significant aspects of Maven, you learn about dependencies. How they declared and managed in Maven projects how you use them in the pom file to add additional libraries, plus we show you where to find these libraries, too. And then you learn all about dependency resolution. This is, how Maven works out, which job files you doing. The project and we cover dependencies scopes as well. There's a lot more to but I should probably stop talking now and just let you dig into the content so enjoy this section as you learn all about the fundamentals of Maven.

In this video we're going to take a look at the project structure, which Maven uses and a great way to do this is by looking at the project, which we created earlier from the quickstart archetype. I'll just open this up in sublime, so we can take a quick a quick look at what's in there, so let's just open up everything in the left hand NAV. So here you can see the structure which Maven has given us for this new project. There's a few things to point out here at the root of all Maven projects. We have a pom.XML file. This stands for project object model and this is basically the model of the project as Maven would like to see it. So it defines some properties about the project here, we can see the group ID artifacts, adding inversion. These are the so-called gap coordinates gap standing for the first initials of group ID artifacts version. These properties together define where the archetype will live in Maven repositories. So when you have a Maven project. Ultimately, you want to be able to publish that up to Maven repository so that it can be used by other people. For example, or the projects rather and when you do that. You'll find that the artifacts will live under these circled calf coordinates so they'll be available under a folder structure, which is comprising of the group. ID then the artifact ID and then the version will see that little later run. In fact, you also see dependencies which the project requires these are software components that the project needs to be able to run. And so we look at those a little bit later, but for now, just think of them as jar files that the project needs to be able to compile and run, although it's a little bit more complicated than that, but that's a good enough definition. For now, so aside from the pom.XML file. We have 2 main directory trees to go up here. You can see we have a source main Java and a source test. Java source main Java. This is the path where you put all of your production source code so In other words, the source code, which actually comprises the products do you developing once it's in Whatever distributor more former it might need to be in so for example, a jar file if it's used by other projects or war file if it's going to be deployed into web container and that kind of thing, so that's the source main Java directory structure, so that's production source code and in addition, you have a source test Java directory. And this is where you put all of your test code so all of the test that you write. These can be unit tests integration tests system tests container test. Whatever type of testing. You want to do the source code of this should be separate from the production source code and this is where the source files live. The reason they should be separate is because it's a separation of concerns so you don't want to mix up test code with production code in case you accidentally package it up into a war file. You wouldn't want to have for example, J unit jar files and such like making their way into a production war file so This is why I never makes the separation between the two directory structures, so source main Java production source code source test Java test source code. In addition, Maven also gives us 2 classes to start with. Which are App.Java. And. AppTest.Java. So after Java is basically a hello world application. If you like it's just a class, which emits hello world when it's run and the app test. Java is a test which doesn't really test anything in all fairness, but it's just an example of how you can create a test using the existing Junit dependency, which is provided in the pom. XML file so by default never includes actually a rather old version of JUnit, which is version 381. And this is used in the AppTest. File here you can see it's importing in JUnit.framework.test in this class. It's got a sweet method, which gathers together any methods, which are prefixed with the word test and executes those as a test suites In other words, it will run the TestApp. Method and just a certain true so these 2 classes in and of themselves aren't particularly interesting. But they do allow us to see how we can build a Maven project and also how we can test it as well, and will see those in the next videos. But for now. This was just a quick overview to show you what's generated in the project list generated from the Maven Quickstart archetype.

In this video we're going to look at how we can build with Maven. So we've seen how we can create a project using Maven Archetypes. This generates a simple project. A basic project as a starting point for our own projects out of the Maven Quickstart archetype and we've seen that it has a source men. Java folder for production source code and a source test. Java folder for test source code and it also comes with a pom.XML, which is the main file in Maven, which describes the build customizations. which our project has pretty much in Maven. You are not really writing a build file per say. And you're more customizing an existing standardized build which may even comes with and we'll see more about that. When we look at build lifecycles a little later on, but first of all let's just see how we can invoke Maven and run a few build commands if I want to build the project, which I have so First off. Let's just take a look and will see that we don't have a target directory so this means that the project hasn't been built before. Maven will create a target directory to generate all its build artifacts in during a build so we don't have a target directory here, so we haven't built it before. Anyway, if we wanted to build this project then we can give the phase of the build that we want to run will talk more about phases. Later on when we look at the build lifecycle, but for now. For example, if I want to create the package, which is basically the jar file of this project I can just type mvn package and enter. So as you can see when I did this, the end output. You can see here that it's written on my up 1.0 snapshot dot jar file into the target directory. I can actually have a look at that directory. And see what's inside. And we can see the inside the jar file. We have an app.class. This corresponds to the app source folder. We saw earlier which was generated so this was the App class that was in the source men. Java folder notice that we don't have app test in here and we don't have app testing here because that's test code so Maven makes a distinction between production source code and test code and the build artifacts, which is generating which in this case, it's a jar file 1 include any of the test classes. This is because of a concept known as dependencies scopes again will cover that a little later on in the course. But for now, just know that it's created a jar file an we didn't really have to do anything. We didn't have to write a script to compile the source code. We didn't have to copy the class files into a temporary directory and then use some kind of jar task to be able to create the jar file may have just done it all for us out of the box, not only has made and created this jar file for us. It's actually executed. The prerequisite build steps required to get to the stage where it can even generate a jar file. If we look up what actually happened here, we see a few things that have gone on. These are really the steps, which Maven took to be able to generate the end jar file so just to point a couple out you can see here. It's used the compiler plugin to compile the source code and then let's use this surefire plugin to test the source code. Nice and run those tests and it's finally built the jar file itself, so this is how to create the jar file to create the end build artifact. Just clear this down a second. Also, another thing I can do is I can just decide to clean the project so if I clean the project. And as you can see it's deleted the target directory. So this means we don't have to target directory any longer, which is correct, so this clears out all of the temporary files that were used in the build and the end, resulting artifacts as well, which were held in the target directory. A few other things we can do as well, we can just decide to test the source code. So if we just type in test. Then this will just run the tests, we can just compile the source code. And this just compiles the source code in the production sources directory. And we can also compile the test source code too. So this should really demonstrate how easy it is to use Maven and the functionality. That's provided out of the box by the plugins such that we don't really need to write any build scripts like we used to do with ants or bash scripts and in the next videos will dig in further.

In this video I just want to talk about the different invocation modes that you can use in Maven when invoking Maven command line. So I'm never really has 2 different invocation modes, which you can use that is styles of command line that you use when you interact with Maven through its command line tool MVN we can either invoke Maven Fire a phase such as Maven compile or Maven package. Or maybe in clean and whenever you see a word on its own in Maven after the Maven command. This denotes a phase so this is a phase in the build lifecycle that's going to be invoked. Alternatively, we could also specify a goal, which is executed within a particular plugin that Maven supply is so we've seen this before we've seen this with the archetype. Generate plugin or go rather and in this format. You can see that the format is basically to have the plug in and then the colon and then the goal that you're executing within that plug in so if I see this command and it's not a single word, and then it's separated by a column that I know that this is basically invoking the generate goal from the archetype plug-in so that's basically the format of those commands so it's either Maven then affairs of the build lifecycle or it's Maven Plugin Colon goal so that's just the format there, which is worth pointing out.

In this video we're going to talk about the Maven POM or Project Object Model so let's just open up a sample project. Have a look at what's in the POM. So here's a sample project opened up. So all Maven projects have a project object model and this describes the view of the project as Maven would like to see it. So by this we mean it includes configuration an properties and Build Customizations, which customize the build for a given project. Let's just have a look at this project and see what properties we have in this is a project, which was created with the Maven. Quickstart archetype as a default project, but it's a good place to get started. So First off we have model version, so this is the version of the POM that's used in general. This isn't change between map and versions, though. Either way, it's generated by the archetype anyway. So you don't really have to worry about it the next we have 3 Very important properties in Maven, which are known as the Gaff coordinates which collectively Stanford Group ID Artifact ID. Inversion so we can see these 3 properties. The group ID is basically kind of a namespace for where this, particular project will live. Kind of synonymous with the idea of a Java package. So it's a way of being able to group in aggregate different projects together under the same location which corresponds to the same section in a Maven repository next that we have the artifact ID this is the name of the artifact that will be produced. By that we mean the end build artifacts. That's produced as the output of the Maven build so this is the best name for the artifact. Next we have the version, which in this case is one zero snapshot so these 3 properties taken together. The group ID artifact, and version specify the location that this artifact will be located in when its deployed to Maven repository. This is either an external repository such as one of the many public open source ones. There are like Mavne Central for example. Which is the main default repository for maven artifacts Although you won't be allowed to publish here unless you have a proper open source project. Or it could be an internal repository, which is normally repository, which is part of a corporate infrastructure. So here we're talking about tools like Nexes. Artifactory, which house all the Maven artifacts across an organization and 3rd party dependencies as well in most cases. Or the local maven repo cash, which is actually on your system will just quickly demonstrate how the group ID artifacts conversion, map to the location of the repository by going to the command line and building and installing the artifact into the local repo cash, let's just do that now. So let's just quickly recap this. pom.XML file and so this project should be located in a group ID, which is com.zention.training.demo This means that in the local maven repo cash, which is located here we expect to see a folder structure like this .com.zention.training.demo and we don't have that, so because we don't have that means it's not installed on this system currently? So if we quickly. Install. This will compile the source code makes the jar file and then install it into the local repository. Then we should see. So it's populated that particular directory and you can see here in the bottom or it's installed 2 items. We see my app 1.0 snapshot. jar and we also see the POM file which is deployed as well. Now, if we re execute that command. We will see the artifact, there, so the first part of the path after empty repository is the group ID that's .com.zention.training.demo The number that we have a subdirectory, which is named after the artifact ID, which is my app and if we go into my app. They will see 10 snapshots under the artifact ID directory. We see the versions, which are available. There's only one version here 'cause. It's the version we just installed. And then finally inside that version folder. We have the actual artifact itself, which is my app 1.0 snapshot.jar you'll notice as well. Also deployed in my app Windows 0 snapshot.pom so let's just quickly take a look at that. And you should recognize this because this is actually the pom. XML file, which was in the root of the project. So, whenever you do an install with Maven to install to the local Maven Repository Cash or you deploy to deploy external repository it also pushes along the pom.XML file such that when this particular artifact is used as a dependency. in other projects. Maven knows which other dependencies in these to download as well, and a few other aspects too. So we see the group ID artifact ID, inversion this so called Gaff coordinate is used both in the publishing. Or installing of an artifact in German repository to define its location, there and the graph coordinate can also be used to use an artifact as a dependency. In the Dependencies section of the pom.XML file too. So let's just go back to the pom.XML file. And continue reviewing this so whenever you create a Maven project at the very least you have a pom.XML file. Which is at the root of the project and this needs to include at least the first five properties which the model version. True party artifact alien version and the packaging type and this constitutes a so called Minimal Pom, In other words, it's the absolute bare minimum that you need in Maven to be able to describe a project, but you can add other sections as well. As we can see by looking at this page on the main website. So this page on the map in website gives you an overview of the Maven Pom structure for the project. Object model. If we Scroll down. This section about what is the POM But just leave there. We have a description of what else is in here, so we can see in the first section we have the basics. Which the group ID artifacts ID and the version we saw before the graph coordinates along with the packaging type which can be jaw. a war this determines what actual type of build artifacts. You want to generate so a war packaging type to generate a war file, which can be deployed into web container. A jar packaging type to create a jar file, which can be used as a dependency than in other projects. Or maybe, as a standalone executable jar file for example, and there's some of the bits and pieces there as well related to more advanced. Use cases of Maven relating to when you create multi module builds and how you manage dependencies within their. We also see the Dependencies Section. This is where you specify the dependencies to the build should use and then underneath there, you have the build settings. So the one where concerned about in this course is to build tag and it's inside the build tag. Do you actually define customizations of the current build so let's just dig into that little bit further So the build section within the POM file can house, a few different elements. But in this course we're concerned with the Plugins section. And this is where you define plugins you're going to use within the build or provide configuration and customize existing plugins, which are being used in the default build. We'll see more about that later when we look at build life cycles. But feel free to have a look at this particular section, which describes the POM to dig in for a few. More details but we've covered here should give us enough of an idea of what the POM does and what it is in this introductory course.

So so far, we've seen how dependencies resolved by Maven. We've seen the resolution of a simple dependency In other words, it dependency, which doesn't depend on anything else. It's just resolved as a simple jar file and we've seen the resolution of a more complex dependency, which does depend on other dependencies too. And this brought encircled transitive dependencies so in addition to being a top level Dependencies included in the project. It also brought in some other jar files to which it depended upon in order to function and we know that we have and achieves its dependency resolution mechanism. By inspecting the pom files that are diploid and packaged along with the artifacts that deployed to Maven repository. So just as you specify the graph coordinates in your project that is the group ID artifact only inversion, which tell Maven where the artifacts of the project is going to live inside Maven repositories. This project can be included interdependency by using those same graph coordinates and this is what makes moving very flexible in the mixer very sophisticated model for building software. We also have to look at the search front end for the Melbourne Central repository. As an easy means to be able to inspect those pom files and look at the Dependencies section and figure out how dependencies resolved by looking at a given artifact. Dependencies section in its POM file. However, there's a much easier way. And this is to use the dependency plug-in so the dependency plugin. Maven allows you to inspect dependencies on a project so if we have a look in our project here. This is the Web app project. We had from before in here we can see that we included spring web and we saw that brought in some module files. We can actually find out where those jar files come from in a much easier way by using the dependency plug in to inspect those. It will give us a dependency graph so let's do this we type in mvn. Dependency this is a dependency plugin. And then tree and this is going to give us a dependency tree. Effectively, a print out of the graph of Dependencies and then we can see where they come from so if I hit enter. So from here, we have the actual project artifacts at the very top which is the root of the dependency tree. This is my web app 10. Snapshot then we have 2 Top level dependencies directly under there. which are spring web and J unit J units are test dependencies on so that won't make its way into the war file so if we just focus on spring web for now, and under the spring web. We see that it required four dependencies itself. These were spring aop, spring beans, Spring context, and spring core and of those, 4 dependencies both of those each had a dependency on one other dependency in the case of spring context. It depended upon spring expression and in the case of spring core it depended upon Commons logging. And so all of those together when we inspected the contents of the war file and will do that again now. Resulted in these 7 jar files being included in the Web INF Lib Directory and this is how you inspect dependencies in Maven using the dependency plugin.

So, in this video we're going to see how we can use the search front end for the Maven Central Repository. To look up dependencies and include them into projects so if we go to search.maven.org. Then we see the search front end so from here, you can type any dependency that you're looking for for example. log4j. And then here it will come back with all of the different results, which include a log4j. You'll notice here this isn't necessarily restricted to just the artifact ID, but the log4j string actually in the group ID itself. So for example, the real log4j dependency, which we actually want is in this one here, which is log4j. Group ID log4j Artifact ID, but it includes a lot of other stuff as well, which maybe we want to see. I can get around that by doing the defense search instead so from here is much more sophisticated we can type in. the group ID, the artifact ID or version so if you just want to search for artifacts that only called log4j. Then we can specify that here and notice as well. When you do that. But in the text box above. It's actually giving a shorthand notation for the more sophisticated search so A stands for artifact ID that I call on the string. It's going to filter on. So we can get the real log 4j by, specifying the group ID on the artifact ID. And hitting search. Then we only have the one result that were actually after there. From this screen you can either click on the group ID to get back to all of the other projects, which are under the group ID is all of the other artifacts, which are under the same group idea log4j. In this case, there are only 2 there's log for chain and Apache-log4j-extras. So we'll come back out of that. Or you could click the artifact ID in which case you're pretty much filtering on just the artifact ID, but then, saying the group ID can be. Wild card match so this is basically a list of projects that produce a log4j artifact ID. Go back there. Here we see, there's a version section as well, so the latest version is 1.2.17 from here. If we wanted to get a list of all the versions. We can just click all and then we can see all the different versions that we can possibly include in a project. Say for example, wanted version 1.2.9 We could just click that number. Whenever you select the version itself. Because that's the most specific way of identifying a dependency at this point we're actually talking about an actual dependency itself actual artifact can be included so this, then brings us to the details section for that artifact. So here you can see it's got the group ID artifacts on the version listed it's also got in the dependency information section that dependency snippet you'd have to include a Maven From the XML file and also other formats as well, so for example, if you do review groups. This would be a way of including that dependency into a group project outside the scope of this course. Of course, but it's useful to note that you can include Maven dependencies from Maven repository in projects that use other build technologies. And then to the right you finally have the POM or the Project Object Model, which this dependency was deployed to the repository with. So, in this case, it's very simple. It's pretty much the minimal POM. It's just specifying the fact that it's a group ID an artifact ID and a version. And there's no dependency information doesn't require anything, it's just a very, very, very basic jar file which has been deployed. Let's go back out from there to the search results again. OK so it tells you when it was last updated the 26th of May 2012 in this case. That let's you download different artifact types, so you can either choose to download the POM itself, which is what we've just seen. Or the jar file which will actually allow you to download the binary jar file itself, all the zip, tar.gz side or the Javadoc, or sources.jar. So for example, if I downloaded the sources.jar. Then I could download that some way on local system and then I would have. A copy of the source code of that particular project so if we just do a jar TVF. On that sources.jar file we downloaded we can see here we see the source code for that particular version of log4j. You can also include this in projects there by specifying the minus D download sources equals true flag as well Or by also putting that in the POM file. Advance search also gives you another option, which is rather. Useful and this is to specify a class name so for example, if I go back to the log4j chase source code, which I have here so I know that this appears in I think it was version 1.2.9 of the log4j project if I copy. This part of the filename, which will correspond to the package and class names or the fully qualified class name of this particular class, which is the SAX error handler for all the Apache to log4j.XML.sax error handler. If I was to paste that into a class name section. Replacing. Slashes. With full stops ofcourse then we find out our search. This shows me all of the artifacts deployed to the Maven Central Repository, which include that specific file and then again, you could link back. And go to a specific version. And have a look there to further inspect dependencies or the POM and such like this particular class was it also included in a different dependency, but for the purposes of a simple course. It enables you to research which dependencies you might need for something so for example, imagine you're going to a blog website. You see some code that's there. We just has imports. It doesn't actually tell you, which jar file it needs to include. You could copy that fully qualified class name of the import statement into here and then they give you a clue. As to which dependencies you might want to download and including your project, yeah, so as you can see. The Maven Central Repository. The search went in for that very useful and a really great way to kick start new projects. By looking for dependencies and copying the dependency snippets into the pom.XML file and that's how we use them ever. Central repository search front end to find and use dependencies with Maven.