Contents

[**Course Overview** 3](#_Toc76048388)

[**Course Overview** 3](#_Toc76048389)

[**Setting up Gradle in Your Environment** 3](#_Toc76048390)

[**Introduction** 3](#_Toc76048391)

[**Comparing Ant and Maven** 3](#_Toc76048392)

[**Installing Gradle** 4](#_Toc76048393)

[**Running Gradle for the First Time** 5](#_Toc76048394)

[**Initial Use of Plugins** 6](#_Toc76048395)

[**Using the Gradle Wrapper** 7](#_Toc76048396)

[**Review** 8](#_Toc76048397)

[**Understanding Projects and Tasks** 8](#_Toc76048398)

[**Introduction to Build Files** 8](#_Toc76048399)

[**Writing Tasks** 9](#_Toc76048400)

[**Dependencies** 10](#_Toc76048401)

[**Adding Plugins** 11](#_Toc76048402)

[**Review** 11](#_Toc76048403)

[**Building Java and Kotlin Projects** 12](#_Toc76048404)

[**Introduction** 12](#_Toc76048405)

[**Basic Java Projects** 13](#_Toc76048406)

[**Java Build Tasks** 13](#_Toc76048407)

[**Using Source Sets** 15](#_Toc76048408)

[**Extending Java Builds** 16](#_Toc76048409)

[**Building Kotlin Code** 17](#_Toc76048410)

[**Review** 18](#_Toc76048411)

[**Understanding Gradle Dependency Management** 18](#_Toc76048412)

[**Introduction** 18](#_Toc76048413)

[**File-based Repositories** 20](#_Toc76048414)

[**Configuration Scopes** 20](#_Toc76048415)

[**Using Remote Repositories** 22](#_Toc76048416)

[**The Gradle Cache** 23](#_Toc76048417)

[**Review** 23](#_Toc76048418)

[**Creating and Managing Multi-project Builds** 24](#_Toc76048419)

[**Introduction to Multi-project** 24](#_Toc76048420)

[**A Simple Multi-project Build** 24](#_Toc76048421)

[**Configuring Multi-project Builds** 25](#_Toc76048422)

[**Configuring Multi-project Builds with the Kotlin Build Script** 26](#_Toc76048423)

[**Using the java-library Plugin and API Scope** 27](#_Toc76048424)

[**Review** 28](#_Toc76048425)

[**Using Gradle to Manage Testing** 28](#_Toc76048426)

[**Introduction** 28](#_Toc76048427)

[**Testing with JUnit 4** 29](#_Toc76048428)

[**Improving Logging** 29](#_Toc76048429)

[**Testing with JUnit 5** 30](#_Toc76048430)

[**Filtering Tests** 30](#_Toc76048431)

[**Review** 31](#_Toc76048432)

[**Understanding the Gradle Wrapper and How to Use It** 32](#_Toc76048433)

[**Introduction** 32](#_Toc76048434)

## **Course Overview**

### **Course Overview**

Hi, everyone. My name is Kevin Jones on Welcome to my course Grade Old Build Tool fundamentals. I am a developer and owner of Rock Salt. Acknowledge a soft development company based in the United Kingdom. Jarvis supports many build tools, most famously aunt and Maven. Grade A ll, in my opinion, is better than either of them. It uses a D S L. In fact, to do your cells, let your script builds that make it easy to both create, maintain and manage builds for projects. Some of the major topics that we will cover includes using different techniques to install Grable creating builds for Java and Colin projects using a managing dependencies within grade A ll creating multi project builds on managing. Jaded for James, five tests from within a greater builds. By the end of this course, you know how to use Grable to manage projects with one or more than one builds. I hope you'll join me on this journey to learn cradle with a grade A ll build to a fundamentals course at little sites

## **Setting up Gradle in Your Environment**

### **Introduction**

Hi. Welcome to the Great Hall built tool Fundamentals Class. My name is Kevin Jones. This is the setting up grade Aled in your environment chapter. So why this course? So as you're aware, Java has multiple build tools. Historically, there was aunt. We also have maven, which is the predominant built with the Java now Grade All, in my opinion, of course, is better than both of these through this course will understand why I think that is on how to use grade all so doing this course we'll cover setting up news and griddle will use grade all to build java on cotton applications. Crit multi project builds in Grady will show how to do testing with grade A ll. I will also show how great deals with dependency management. So first of all, then what is Grade A ll So if like me in the first build. Two years for Java was aunt and had no conventions. We had a code everything into one build script by hand. Then along came maven Maven does have conventions and grade A ll like maven is convention based. There's conventions such as where to find source files Where to find resource is where to find test source files and so on. The cradle is also highly configurable. I'm going to see doing this class how we can change those conventions if we need to. So Aunt on Maven are XML based tools Grade A ll comes to the D S L domain specific language. In fact, it has to do your cells. There's the original groovy language based E s L. And there's a new Katelyn base to your cell. I receive both of these right the way through this class. So all the builds that we do we'll use both the groovy on the cotton in D S. L's so in grade A ll, we can build a single project and set up a build for that with a build file. But it also supports multi project builds and we'll see how to do that. You're in this class. It's easily customizable. It has support for compiling many languages. So obviously Java based languages such as Java and Katelyn, but other languages as well such a C plus plus. Unlike maven, Greater also supports dependency management

### **Comparing Ant and Maven**

So what about the other Build tools we all know and love? Firstly, there's Aunt, which stands for or at least used to stand for another neat tool, probably the first Java build to written when people got very frustrated with using Make a guy named James Duncan Davidson came along was a Sun employee at the time I realized that make was failing us and decided to write a build tool. At the time, XML was used for everything. So we used XML to create the tool in the truly come up with was Aunt. So Aunt provides us with an XML build script that's hard to read and difficult to maintain. So this is an end build script for a fairly simple application. Consists essentially of \_\_\_ \_\_\_ fell that we're going to build. There are some library job falls to include. One of the big issues with aunt is the lack of conventions. So we have to specify everything from where I was sourced directly lives to where the classes will be built. Where we going to get libraries from toe, where we going toe output, Anything that we create here. If we look at this file would find there are many, many properties here specified that this information and then here a set of targets. So, for example, it's prepare target. The creator directors that we need is a build target that allows us to build a code. But notice again. There's no conventions, all the targets we have to specify. Absolutely everything that we need on files can get very large, very quickly become very un maintainable as we can see from this very unreadable. So what about Maven? I think Maven has several advantages over Aunt. Firstly has many, many conventions. We don't have to specify where we were going to get our sources from or where to build the open, too. May even already understands that maven also supports dependency management. See here, for example. We can see we're specifying We have a dependency on surveillance on it will go and find this dependencies and download them for us. We don't need to worry about carrying those jar files along with us. The Maven Palm allow us to specify the dependencies and then resolve the dependencies when we need them. Maven is also highly extensible, so we can configure maven with plug ins unlike Aunt, where everything has to be defined by us. However, like Aunt Maven is written in XML unlike, and it's hard to read and can be on maintainable. So what about great Earl? This is an example of a greater build file or build a cradle. Here we can see we write the build file in a language that specific to this domain to the build domain. So this is an example of our domain specific language. You can see we especially flying repositories to use plug ins to use dependencies to use. We also have some extra stuff here. We need some set up to do tests on again. We can see how we apply that within the build file. Now, at this point, you might be looking at this and wondering what on earth is going on? It's like nothing you may have seen before. It's certainly no XML. However, we'll see as we work through the rest of the class that build files because of the D s. L become largely self explanatory on relatively easy to maintain and manage on this D s. L is one of grade A ll strengths

### **Installing Gradle**

so how to be installed? Grable. Well, there are various options depending on the environment that you're using. We can install it from the website so we can go to greater home page, click on download and grab a zip file on the story from there. And this will work regardless of the platform which were on syphon on windows from Mac. No more the limits platforms. Then we can always install from the website. You can also use a tool called the SDK manager. This is available from sdk man dot io. I'm using the SDK manager. You can install Java developer kits for various places such as the Open J. D. K. You could also install various sdk such as grade A ll, maven cotton and many, many other things. However, the SDK manager really only works on UNIX platforms. Little work on Lenox. It'll work on the Mac. You can get us to work on windows but to do that you need to use the clinic substantive windows or something like Seguin. So what is possible to use the sdk manager on windows typically to install great along windows? You do it from the website in the final option. If you don't, the Mac is to use home brew. There's a brew recipe to install grade all so the primary way to install grade all you should do it through the website. I'll show you how to do this on Windows, but the same thing applies whether on Windows or the Mac or Lennox. So if we hit the great all home page a grade A dork from here there's a link to install Grader. And if I click on that, it takes me to the installation page. And here walk me through the prerequisites, such as having Java installed there was a version greater than job eight. Some additional resource is how to install using sdk man, which will take a look at soon how to install using homebrew again. We'll take a look out soon and then how to install manually. So from here I can download one of two versions of grade A ll, one with only the binaries or one with also the documentation on the sources. We just click on finery only. I'm gonna download this into my 10 folder and then once we have that, I've got a copy. This falls into my home Kevin J folder. Once we have that for looking here, we have a BIN folder inside the bin folder. We have a bad fall while I run. Great Alice, the bat file that runs three to make sure this location home, Kevin J. Gretel 63 Ben is on my path. So if I copy that, go to advanced system settings on environment variables and edit my path. Do new paste in the valley for the bin folder. Say okay. Okay. And then close these windows. If I run a command prompt, then from here, if I type in grade A ll short of a Gretel starts to run two Greatest not been installed on my windows machine. So what about installing on Lennox and Lennox like environment such as the Mac? Off With this, we can use something called the SDK manager. The idea behind the sdk manager is that it makes it easy for us to install Java based development environment such as JD Kay's on also Java based SD case such as May even groovy grade. Let's Evra, We get the sdk manager from https sdk man dot io We can download it with cool script. Once you've done that, we then add the sdk manager to the path. Once we have it on the path, we can then install our sdk So you want to install cradle? We can run sdk install cradle So let's see how we do this insight you been to So here I am inside you bun too And in here from the chromium Web browser When I'm at S t k man dot io This is the home page the sdk manager to install this If I click on the install link, it shows me the curl script And if I copy this and go to eternal window paste in the link it downloaded installs SD Cayman by then Run this script to make sure the sdk manager is on my path They can now run sdk Help ensure life is there back in the browser window If I go to s t case, scroll down and find grade A ll It tells me how to install grade all So just sdk installed Grader So s decayed installed cradle in the terminal window against installing the latest version just 6.3 If I run grade A ll gain. It's now installed this time inside my linens environment. We're finally on the Mac. Yeah, I am again instead of terminal window. And here I'm going to use home brew. So I first run brew update to make sure all the formless after date and then run through install cradle the gay notes. It grabs grade all 6.3, which is the latest version of time of recording. And again, if I run grade A ll from the command line tells me 6.3 is installed on running.

### **Running Gradle for the First Time**

So how do we use grade? All the first thing we have to do is to create a build script we mentioned earlier. The cradle has a D s l If I could have to d S l's when we use that d S l to create the build script and the two flavors of DSLR Cartman and Groovy is the D s l has been around the longest since that since the early days of grade A ll and cotton is the newcomer. The groovy D S l is the advantage that it's used everywhere. If you search for help on grade on the Web, you'll find loads of examples of using the groovy D s L The court Lady sl is newer and the nice thing about the cotton in D s l Is that because cotton is a type language, we can get help with insight. I d e is when we use the d s. L so, for example, inside intelligent idea. If I write my build's scripts in cotton in using the cotton TSL intelligent idea understands the d S L s Oken, Give me help us. I'm using it. We'll see examples of both of these \_\_\_\_\_\_. This class we'll see as well that in some cases the groovy TSL is probably slightly cleaner in the cot Lindy S L. Although they both have the same degree of flexibility. So the build script you right defines tasks. I was seeing a moment how we can define a task within the build script itself and these tasks that we run these be tacit has builded, for example, to build a project or clean to clean a project and many, many other tasks. We'll also see that Build's scripts typically contained plug ins and by adding of plugging to a build script that plugging defines many tasks. And we'll see an example this soon with a Java. Plug it once we have the build script and we have all the task to find that we need. We can then run the task, but we can do this from the command line. What we could do this room inside an I. D. So all the major ideas so certainly intelligent eclipse Net beans all understands grade A ll and allow us to import cradle files as projects and then run the task from those files so stop looking at some simple build files, some simple build scripts and see how to use those before we go into more detail in later chapters. So this is a very simple, greater build. Far in fact, one of the simplest build falls that we could look at. So this Build phone is written using the groovy D S L. And then here we define a task that email a task is hello. And they also have an example here of a build phone written using the cop Cling to your cell, and in this case, it's starting, Morpher boasts. We have to register the task with the tasks collection. Again, the task is called Hello. After that, though, the Codex very, very similar. So tasks have a life cycle. They have various faces. There's a configuration phase under initialization phase, and then when you define a task, you could provide two methods one called do first on one called Do Last, and these run at various parts of the task life cycle. Now we only really care about this if we're writing tasks. There's an example of that here I was Seymour examples of this in the next module So what is our task to while in the greater build fart the do last method caused print line and have prints out Hello, Grady. The code inside do last is just groovy coat and similarly in the cotton in build file. Or do last method calls print line in a game Prince out. Hello. Grade. All in a game within the do last method here we can add any coddling codes that we'd like to use. So we've already installed greater here. We know we can run greater from the command line. So here I am on my my inside eternal window inside here. If I look inside my directory, then there's my builds, Doc Cradle files a build our cradle the typical names we give to our groovy based build files. So remember that our task was called Hello? So from here, I can run grade A ll Hello, which is the name of the task of ahead return. It tells me which task is executing. That's the hello task, but it runs the task. And we saw that task was due last meth od and that do last method we print out Hello, Grady. And similarly for the Katelyn task. So if I change to the directory that contains the cotton in build file Last build, Doc Cradle Katie s. So Katie s is a Coplin script far and that I could do the same thing. I can run grade all Hello. It shows me the task. This run and again Prince out. Hello grade All. So what we can see here is that grade A LL is built on tasks and it runs those tasks for us.

### **Initial Use of Plugins**

So he said that the build far we just looked at one of the simplest build fires that we could have. Well, this one's even simpler. So all we're doing here is applying the Java plug in supply gins our way of extending grade A ll and introducing tasks to grade all. And we'll see throughout this there are a couple of sin taxes for using plug ins. This is one of them will see the other way in a moment because I have this plug in. If I go back to the terminal window when the director that contains that directly contagious, disease riddled Doc cradle and our sources directly connects my job resources, I'll talk more about that in a moment. The 1st 4 What about this? Plug it. So if from here I run grade all tasks, task is like a built in greater task. What this will do is list all the tasks available to me through this build file. So if I hit enter here, it was served the tasks. If I scroll back up, we'll see that I know many tasks available for us and these tasks are categorized. So, for example, we have build tasks, and in here we have a task or build a tactical clean. We have a task called JAR. So that will build, For example, is Java project that will clean the directors of a Java project. That jar, far from the Java project, noticed we have a job doc task on the documentation tasks, and then, if I scroll will see that some verification tasks as well. So, for example, test and that will allow me to run my unit tests. So if I run the build task here, it's going to build the project for me. However, to do that, it needs to know where to find the Java files I noticed within the build script. I'm not telling it anything about where to find these files, so it's got to use its conventions. Locate these files so I look inside the equivalent of find a window. Here inside the source folder, there's a main folder, and inside the main folder, there's a Java folder and inside there that I will find the files season down. Based on package, the package is complete. Our site and there's my Hello Java. This is the convention. So by convention greater will look for files inside. Source. Maine Never look for tests inside source tests. Let me take a look at the driver. Plug in more detail. A little later, we'll talk about how we can change this. If we need to sew back at the command prompt, I can now build my Java code. I do that by running Cradle built so it doesn't show too much other than it's executed two tasks, but we know how some output. So, for example, we now have a build folder. So if I look in that build folder would find a classes folder on inside here will find Java main calm, clear sight. Hello, Doc Class and we have a lips fuller and in here we'll find First Java Dodge are to the build task Chris the classes for me and put him in the right place and also Chris a jar file form. It is based on the name of the project, so notice we also have a grade A folder and that great a folder contains information that great old users and notice that we're running version 6.3 and gold run this task like delete both of these folders, but this time we'll use the minus I flag minus. I just house cradle to print out some more logging information spire on grade A ll minus I build. We can see the tasks that get executed and some extra information about those tasks. If I scroll back to the top here, we'll see the tasks that run on the order in which they run. So, for example, there's a compiled Java task that runs, not Build. Was the Java code notice for process. Resource is, it says no source said. There are no resource is there's a talent, it doesn't run the classes task, and that's because the compiled Java task has already run. It does run the jar task to create the jar far for us, and the assemble task is essentially just a synonym for jaw. There's no need to run the assemble task and so on and so forth. We have no tests. We have no test. Resource is we have no test classes and then everything is done. If I clear the screen here and run this again, notice this time the jar task on the classes task say up to date to greater recognizes that we've already compiled this code and we've already created the JAR file. There's no need to run these tasks again. So I said that there's two ways of applying these plug ins and they apply plugging mechanism. There's no being superseded in many cases by the plug ins block, but in so here we say plug ins, plug ins, have an I. D in this case, the Java plugging. This is a well known plug. Integrated knows how to apply this automatically to our build script. So I use this syntax and go to my terminal window. And again I clear this and run grade all tasks. We get the same list of tasks, so in this case, we can use either syntax for a play in the Plug it on again. If I run cradle build here again, we noticed that the code is not compiled on the jars, not Rome, because the classes are up to date. Let's try this. So this is the job of file I'm compiling at the moment. Is printing out Hello world. Let's run the code just to show that so I run Java minus C. P at the class path and then calm. Doctor \_\_\_\_\_\_\_ dot hello When we get Hello World So what happens if I change this coat? So rather than saying hello world, let's say hello, comma world make sure this finest saved go back to the terminal window. And now if I do a great will build on, scroll up, we notice that the compiled Java task now runs. This is just not the data's. The sources have changed. The jar task runs again because the classes of no changed so great Oh does change detection. It notices that the source files have changed because they change. It knows which tasks that it needs to rewrite and if ever, on the application again. Now it says hello, calmer world.

### **Using the Gradle Wrapper**

so I want to talk about one last thing here. We've seen several ways that we can install Gretel. But this one other way, we can use something called the Grade or rapper. So from within this project around grade, all tasks noticed that one of the tasks we see here is a test called Rapper. And if I run that and then look at my directories, we see a few things have changed here. I now have a new direct record cradle that contains another direct recall rapper on inside. Here we have a jar file on a Properties farm. There's also these two extra Farsi greater W, which is a shell script for UNIX platforms in greater w dot Bad, which is a bachelor for Windows platforms. If I go back to my terminal window, so now rather than running griddle directly, I can use the rapper. So from this directory, that makes a great deal. W give the name of the task to run, which we build. Another first thing it does is done was a version of cradle to my machine and then uses that version of grade A ll to execute this code on on the Mac. It stores that version of greatly my home folder, Doc Cradle Inside Here there's a rapper folder, Distance Running Cradle 63 and its stores that version in this hidden folder, and I want to do a build. It will always use that this particular version of Grade A ll. So if I go back to the terminal window and here's grade all builds rather than greater W build notice, it tells May there's using cradle W instead of grade A ll. So I do this well, we'll talk about this in a bit more detail later in the class. But the idea now is that within my project, we can specify the exact version of grade all to use on. We'll see later. The rapid task is configurable speaking, specified within that task, the version of great I want to use for this project.

### **Review**

So in this module we've taken a look at various things we've mentioned. The Java has multiple build tools and amazement of the best known ones. Apart from grateful, as I've said, in my opinion, I think cradle is the better, too so out of the three that were always these cradle to manage my projects. Given the choice, of course we want to see initially how to use grade all. We've seen how to install grade all in various ways, using things like home brew on the SDK manager on different platforms. We've taken a look at how we create an initial project. They contain one task that we defined. We've taken a look at a project with the Java plug in when we showed you this here just to show the plug ins at tasks to projects. We've also looked at the fact that grade all comes with some built in tasks such as the tasks task on the wrapper task. So next up we'll take a look at tasks. We'll create several projects when we land tasks to those projects in various ways. We look a task dependencies when we look at plug ins in a little more detail will show how to add built in plug ins to a project such as the Java Plug in. We'll also show how to add community plug ins into a project as well.

## **Understanding Projects and Tasks**

### **Introduction to Build Files**

Hi and welcome back to the griddle. Build Tools, fundamentals. Course. This is the Understanding, Projects and Tasks module. And my name is Kevin Jones. So, as we said in the introduction grade, A ll mostly consists of projects projects of defying with a little greater file on tasks on those tasks are executed as part of the project. So, for example, we may have a built task that I just build a project in some way or a clean task that understands how to clean up the project. Now a build may have Maur than one project, so a simple build would have a single project. Maybe maybe we're building a library. But more complex build will consist of many projects, and we'll see how to handle multi projects builds later. As a build has more than one project, the project will have more than one task. All right, take a look out in this module is how to add tasks to the project. I'm going to see that we can't ask by hand, but we also see it as we saw in the direction that we can use plug ins and plug ins. He used to extend the project by adding tasks, among other things. So what is the project? So Project has a build file on this build file will either be a groovy build file called Builds Doc Cradle or Katelyn Base Build file called Build Dark Cradle Doc eight. Yes, optionally. You could have a settings file on the Settings file can hold information that the build found might need, for example, version numbers continually going to a settings file. So what's in the build phone? So Build file defines the tasks that he used for this project. I like we said the task could be, for example, a build task or a jar task or a clean task or any other sort of task you might defined That's necessary for this particular type of project. So grateful comes with some built in tasks such as rapper on the Tasks task. You can also define the tasks directly within the build file. When we saw how to do this in the introduction, I will cover more of that in this module, or you can introduce tasks indirectly by adding a plug in. So, for example, the Java Plugin, as we saw in the introduction to this course So before we get into the details of tasks, let's take a look. A really simple build file. So we said the grade A ll comes with some built in tasks. How do we know what those tasks are with the Aegis way would be to create an empty build file. So in my editor here, I'm using visual studio code. I create a new file and save it. Our critic ate yes, fall. But the same thing will be true for the standard Build Dog Cradle fall. If I go into the terminal window, I mean the right folders. If I take a look at the injuries in this directory, there's the empty build file. I'm from here. If I run grade all tasks, we get to see all of the built in tasks so you can see these really consists of two things with some set of tasks such as rapper we've seen already in it, which will take a look at a little later. This allows us to initialize the project for a given runtime environment. So, for example, we can use this to quit a project for a job application for a job, a library and then there are some help tasks. So the help task display some help messages and there's the tasks task that we've just run. Let's just display these tasks. So these are all that's asked to come built into grade all.

### **Writing Tasks**

we've seen that a project fall consists of tasks when we run a given task, greater run through it several phases for that task. There are three main faces. There's the initialization face is the configuration face. And then there's the execution face, sir, tearing the initialization phase greater works at which projects will be part of this particular build. So it's a single project build on that projects part of the builds, give us a multi project build, then it determines which projects. But then that money project build become part of this execution. The project that configured at the configuration stage. And this is done by executing the build's scripts of all the projects that a part of this build and then finally in the execution phase, greater works out which tasks will be executed. Mrs. Based on the task name this past is the argument on the greater command line, so given task as multiple faces of execution, they may be the You want things to run at the start of the task execution things you want to run at the end of the task execution of things you want to run only on certain conditions, so within the test, we can add a do first closure to specify the actions that execute. But the start of the task execution gonna do last closure to special things that execute at the end of the task execution. There are also other things. So, for example, you can specify only if something is true or false is part of the task execution as well. But you won't go into that is part of this class. So that's right. A task and see this in operation. So how do we have a task to a build? So here I have an empty build dark cradle file. Well, firstly, how we add tasks using the groovy syntax. And then I'll do the same thing using the cotton syntax. So for the groovy syntax, we use the task command. We give the task in name and then the task has a closure on inside here. And remember tasks apart the execution phase. Then the execution faced. Two things happen. Do First will be called and do last will be caught. Let's try this. Let's other method called Do First on inside here. That's a print. Hello. And then similarly yes, l a method called Do Last on Inside here, Lester a print line off comma world. So if I save this file and go to my eternal window So in here this is the build for called build a cradle we call the task Hello. So if I run grade all Hello then the do first ex capes and prince out Hello The do last ex kids and prints out Comma World We could do the same thing for our Kauffman's script. So the center just like different here Coffin has a tasks collection my ad tasks to that collection So it's a task Start register like give the task and name the game. We have a closure and inside the closure again, I can call do first and print out hello and do last and then print line, Let's say comma grade all this time The game Making sure the finest saved. Making sure we're in the right directory In the terminal there's our build Dark cradle Dark 80 s file and again running grade All Hello. Sure enough. It now prints out Hallow Grader. So we see we can write tasks in line within our codes using either the task keyword in groovy or adding a task to the tasks collection in court lit. And then we can execute those tasks. A case. This is a very straightforward task, a slightly more complex in this. So, for example, one task may depend on another, so let's see what that would look like.

### **Dependencies**

so tasks may have dependencies. So, for example, in the first module we saw that the jar task depends on the classes. Task. You can't create a jar file unless you have classes. Tusk may have more than one dependency. Sub tasks may also have dependencies. So, for example, let's suppose we have a task called Task Eh? Task A. May be dependent on multiple other tasks. Let's call these tasks be C and D. And then he may also be that both tasks e onda task t depend on another task. Task E. So what does this tell us? What tells us that when we run task A. We know we have to run tasks B, C, D and E. We also know that E must run before task Si and tasked the runs task B, C and D must run before task Aaron's. There's a hierarchy here, dependencies. So let's see how we set that up with our tasks. So let's do this again, both with Cotman on with the groovy D s. L's so again here we have the hello tasks registered. This is using the cotton into your cell on notice. I've simplified this. I've left the do last method in and here we gonna print out just Hello. But if I run this task you see a prince out the message. Hello? So let's how dependent asked this So to do that I need to add another task. So I say task stopped register and again I give this task of night Let's call this second task world. This will also have a do last and inhale print line world the company If I run this by saying grade A ll world three prints out space world However, what I'd like the world has to do is to execute the hello task first and the way we do that in the Katelyn TSL added dependency To do that we say depends on into depends on We passed the name of the task So world depends on hello. So if I go back to the eternal window and run grade all world now notice the hello task executes and prince out Hello then the world's task executes on Prince Out World We could do the same thing using the groovy d s l so here I'd say task world might do lost would say print line world and then I can say depends on hello, and that was the syntax is a little different here, perhaps a little cleaner. I don't need to enclose the helo in quotes, for example. So if I go to my terminal and make sure I'm in the right directory when the build file is in place on Run Greater World again, then again, Short Life of Princess Hello World. So both tasks execute the world. Task depends on the helo task. So what about the build's life cycle? Let's just take the cotton in script as an example here and how they do first as well as I do last and in the do first, Let's print out H E l and in the do last, print out the rest of the message. Well, then, let's do the same thing for world. So in due, first, we print out space W O. And in due last printout rld and go back to the eternal window game. Within the cotton and directory. There's the build file. If I run grade a world, what happens? We see the fridge task to do first method excuse and then the do last method so for her low we exclude, do first and print out \_\_\_\_ and then Lo and then for world execute. Do first and print out wooo and then do last. We print out R L d to do first and to last execute if you like. Within the tasks reach tasks, do first executes and then it's due last executes. But remember in between, these other things could execute as well.

### **Adding Plugins**

So we've seen that we can add tasks to the project either by adding tasks to the task collection explicitly inside Katelyn or using the task function inside the groovy scripts. We've seen that doing that we can build up dependencies attacks as well. But there are other ways of adding tasks to a build file in the most common way is to use plug ins, so plug in extends the project capabilities in some way. So here on the slide, we can see examples of how we apply the Java plug in to our build script. We could see that are a few ways of doing this so we can use the plug ins block in both the coddling script. Mark 80 s here on the groovy script. Well, we can use the apply plug in. We see an example here from Groovy, and as we go through this course, we'll see these examples many times. Let's take a quick look Now what happens when we apply the Java plug in to one of our scripts? So we saw in the previous module we can introduce plug ins, but using the plug ins block and here we have the Java plug in and this is using the groovy syntax. And similarly, for the cottons in taxes, Wells is in the K T s fall. I se plug ins Java. So Java is a well known plug in if you like. So Cradle understands this plague in Just give it the name inside the plug ins block that plug in is added for me into the buildfold. And we know that if we go to my eternal window and type cradle tasks, it lists all the task that are plaguing provides for us. We know, for example, here we have the extra verification tasks like check and test. Then further up, we have the build tasks in the documentation tasks. But what if you want to add other plug ins to our build Script plug ins The grader doesn't know about ahead of time. So, for example, there's a library and two that you can use in Java to build Winona's database migrations. This provides a way of easily updating and downgrading databases that you might want to write. This is called Flyweight. This is the Flyway website on Flyway comes with a plug it, but the flyway plug in is what's known as a community plug in. So I can't simply add this by name into my greater file. I also have to give this aversion. So for the Flyway, plug it. The current version as of recording is 6.3 point two and we had the plugging into a groovy build script like this. So we specify the I D, which is thief. You like the fully qualified name of the plug it and then the version. And then it looks for this. Plug it on the plug ins portal. If I go to my terminal window and now run and now run great all tasks, then the first thing that will happen is that great herbal. Try and download all the libraries that needs for this plug in and indeed the plug in itself and then list all the tasks we've added for this plug in there, we can see the group of flyway tasks so it doesn't matter in this case what these tasks do. It's just important to understand how to add community plug ins into our build script. We could do the same thing in the coddling d s. Ella's well, so if I got my builds. Duck cradled are Katie s file. The syntax is similar. It's no, we say I d and we give it the string the parameter to the I. D. Call that also passing the version number. And again, if I go to my terminal window, I'm now in the Coplin Directorate. And again, if I run great tasks the plugin was proves he downloaded. When I run this from the groovy folder, the game will list the flyway tasks in here. So for community plug ins, we have to use the fully qualified name to install the plug in when we see a little more of this as we go through and build various Java projects over the next few modules.

### **Review**

So in this chapter we saw the cradle mostly consists of projects and tasks we talked about but really have uncovered yet how build may have more than one project, and we'll see that in detail in a later module. However, we did see that a project can't have more than one task. We saw her toe add tasks quite simply into a project when we saw out too hard tasks by a plug ins. When we added tasks we talked about, Ah, one task may be dependent on another task such that when we run one task, it may also run Other tastic needs to to complete its work. I'm talking about plug ins. We saw how plug ins extend projects when we talked about the idea of well known plug ins. In this case, we saw the Java plug in and also the idea of community plug ins. We looked at the flyway Plug in. We'll talk about how we can add community plug ins into our projects as well. So next we'll take a look at a real project. We'll take a look at how we build Java project and also how we build Katelyn projects using grade all. I will see the various play games that we can apply for Java on the plug in. We need to apply to Build Cotton in projects.

## **Building Java and Kotlin Projects**

### **Introduction**

Hi. Welcome back to the Great Build Tools. Fundamentals. Course, This is the building Java and Katelyn projects on my name is Kevin Jos. So up until now we've looked at how to install grateful and how great it relies on projects and tasks. We've seen how to write a simple task and how to apply plug ins. So this module will take a look in more detail at Java plug ins. We gotta build Java projects in this module and we'll see that there are three Java plug ins that we could use for these plug ins will take a look at the tasks that are added on how to use those tasks and how those tasks are related to each other. We look at something called a library versus an application, and also we take a look at how we build Katelyn projects. So we look at the coddling plug in a swell. So to do this, we need to build a project. So how do we create a build? So first of all, we create a build file and I could either be using the groovy the SL all the \_\_\_\_ Lynne D s l must see examples of both during the module. Once we have the build fall, we are the appropriate plug in. We may need more than unplugging depending on what we're building. We'll see that plug ins as well as providing tasks, have specific properties that they set within the build. Far we can override those tasks and we can override those properties, and we'll see how to do that during this module. So I said this more than one Java Plunkett. We've seen one already. We've seen the Java plug it. This is the base plug in for Java that our tasks such as clean build and jar to our project. On top of this, there's a Java library plug in when we use this plug in. If we're building libraries as opposed to, say, applications, the library plug, it has some subtleties around how we use this in a later module. When we talk about multi project builds, we'll see some of the subtleties of this library in particular in the way that we can specify a P I is the labor users. And then finally there's the application plugging the application plug in life. The library plugin extends the Java plug in the application plug in gives us the ability to run the application from inside cradle. And this also implicitly applies another PLUGIN called a distribution plug in which applies tasks that help us when we came to distribute applications. So when we use the Java plug in one of the main things, it's not. The main thing we're going to do is to build our Java code to build the Java code. I need to find out the source code on the Java PLUGIN expects to find their source code in standard locations. Now we can change this. I must see how to do that in this module when we change it using something called a source set. This is a standard part of cradle. The standard code layout looks something like this. You have a source directory and in there you have two subdirectories main and test. And within each of these subdirectories, you have a job. A directory for your Java coat loving resource is directory that contains any resource. Is this project needs if you doing, Cartman, you also have a cop in subdirectory within each of Maine and test that contains your coddling code this is the same structure that may even uses. So if you have an existing maven project me moving across the grade A LL, you won't need to change a source code structure. If you do need to change the structure and you can change it by using source sets at the source after you pass a closure on inside the closure, you can specify the directories that you need for Java on four. Resource is, for example, so if you have a non standard layout, then you can specify that layout using source sets.

### **Basic Java Projects**

So we got to start building a driver project when we're going to start fairly simply when we got to see something we've seen before, which is just adding a plugging into a greater build file on building the project that way. And then we'll take it a step at a time and see how we can extend this project to our properties and configure the project in various ways. And then we'll end up by building the same functionality. But the code written in Katelyn rather than written in Java so we can see how to play the various Colin plug ins and configurations that we need to project. I would do this using both groovy build scripts on the cotton in Build's scripts so we start here. So we have a project source folder that follows the standard naming conventions, so there's a main with the Java and resource is on my code. In here we have two falls. We have a base, reportedly Java and reportedly Java. It doesn't really matter what in these falls will build on this as we go along. Essentially, these files just follow the repositories pattern to do database access. So we're going to start where we started before. We have an empty build far on in here. We're going to add the Java plug in Andre using the newest in tax for this with a plug ins block when we specify Java as the plug in saying I d Java. So if I go to my eternal window here, there's the build greater far. And there's the source folder. When you see already that if I run great all tasks, the plug in as number of tasks for me and we have seen this a couple of times now when we can see the verification tasks on the build tasks, I would talk in more detail through what these mean and what they do as we go through this module. So I know. But if I run great, I'll belt. We can see it's executed two tasks when we can see the output from this build. So if I go back into this fine David, we now have a build folder in here. We have a class is fuller, We have Java classes, we have complicit repositories and the two class files have been built for us. If I go back get inside the build folder. We have a Libs folder and in here we have repositories dot jar. The name is based on convention. So the name of the project the name of the directory we were running this from is called Repository. So it names the jar far repositories, Doctor. So let's take a look at this. Build in little more details if I clear the screen hit and I want to do a clean followed by a builds and I want to print out information about the belt. So first of all, let me show you this. So what I can do rather than running clean and then build I can run both these tasks one after the other so I can say grade a ll clean belt that runs the clean task followed by the built us coming to see that Nothing. Three tasks of run. So to see what's actually happened, let me run the clean task, clear the screen and then run the build task with the minus I flag in the minus. I flag uses informational level logging to print more detailed information to the screen so we can see the tests that have run on the task that haven't run. Let me pipe this output to a tax file. We can then remove some of superfluous information here and try and get a few on what's happening.

### **Java Build Tasks**

So we got to start building a driver project when we're going to start fairly simply when we got to see something we've seen before, which is just adding a plugging into a greater build file on building the project that way. And then we'll take it a step at a time and see how we can extend this project to our properties and configure the project in various ways. And then we'll end up by building the same functionality. But the code written in Katelyn rather than written in Java so we can see how to play the various Colin plug ins and configurations that we need to project. I would do this using both groovy build scripts on the cotton in Build's scripts so we start here. So we have a project source folder that follows the standard naming conventions, so there's a main with the Java and resource is on my code. In here we have two falls. We have a base, reportedly Java and reportedly Java. It doesn't really matter what in these falls will build on this as we go along. Essentially, these files just follow the repositories pattern to do database access. So we're going to start where we started before. We have an empty build far on in here. We're going to add the Java plug in Andre using the newest in tax for this with a plug ins block when we specify Java as the plug in saying I d Java. So if I go to my eternal window here, there's the build greater far. And there's the source folder. When you see already that if I run great all tasks, the plug in as number of tasks for me and we have seen this a couple of times now when we can see the verification tasks on the build tasks, I would talk in more detail through what these mean and what they do as we go through this module. So I know. But if I run great, I'll belt. We can see it's executed two tasks when we can see the output from this build. So if I go back into this fine David, we now have a build folder in here. We have a class is fuller, We have Java classes, we have complicit repositories and the two class files have been built for us. If I go back get inside the build folder. We have a Libs folder and in here we have repositories dot jar. The name is based on convention. So the name of the project the name of the directory we were running this from is called Repository. So it names the jar far repositories, Doctor. So let's take a look at this. Build in little more details if I clear the screen hit and I want to do a clean followed by a builds and I want to print out information about the belt. So first of all, let me show you this. So what I can do rather than running clean and then build I can run both these tasks one after the other so I can say grade a ll clean belt that runs the clean task followed by the built us coming to see that Nothing. Three tasks of run. So to see what's actually happened, let me run the clean task, clear the screen and then run the build task with the minus I flag in the minus. I flag uses informational level logging to print more detailed information to the screen so we can see the tests that have run on the task that haven't run. Let me pipe this output to a tax file. We can then remove some of superfluous information here and try and get a few on what's happening. So I quit. Build our text, so I've taken that output. It's in a text fault. I've stripped out most of the noise here so we can get some ideas. What's happening so we can see at the end you We've run the build task on the build. Task has all these dependent tasks. I'm working away. Back to the top. We see it the first task that execute is compiled Java. Underneath this, it says Compile Dr. It is not up to date because this is also because source files are later, if you like than the class falls so as to run. Compile Jarvis, which compiles the driver sources. It doesn't process any resource. Is was that on, and it skips the classes task because classes and compiled java and essentially the same thing. So the classes task is a life cycle task. The classes task depends on process. Resource is on compulsion over, so the classes tests simply forces those other two tasks to run. We then have the jar task. Now. The John task is a task, this part off the Java plugging on below that we have the assemble task, and the Java plugin depends on another plugging called the base plug it and the Assemble task is the general purpose task that's in the base plug in that assembles files together. Where's the jar task? Is the job a specific version, if you like of the Assemble task? So in this case, assemble in jar are synonyms, so the jar task runs. I'm generous the jaw fall for us and then under that we can see that no test sources to compile that are no test resource is to compile, and there's no test to run. And so the build task completes. So the Cartman syntax is equally a straightforward. So here we are. Plug ins, block miss a plug ins, java. And again, if I go to my terminal window, we have my coffee and build file hit and again a cradle clean built. I look in the folder. We now have our build folder, and in there we have the same output. We have our classes, files and we have the jar file. So the Congress into action, the grooviest inductive Very, very similar here. So how about configuring the build? Well, one thing we can do is our version Information for our outputs. So in my grade a file If I say version equals 1.0, dash snapshot Let's update the cotton and fire while we're here as well we save vision equals 1.0, Dash Snapshot Noticed Double cross here rather than single quotes in the groovy far If I go back to the terminal This is still my cotton in built on run grade all clean built If I look at the build for the cotton output lived this generated now says repositories Hyphen one Dachau hyphen Snapshot Dodger. I was taking my version number Gonna play that to my project name and generated the jar file name for me. This is one way that we can configure the Java build process. So if I changed directories and go to my groovy build folder there's not build a cradle fart and again run cradle clean belt And if I look at the groovy output again we've changed the name of the jar for that's been generated to include the version number. Okay, so now that we've seen these fairly simple builds, that's going through something a little more complex.

### **Using Source Sets**

So we've just seen how to configure a java build by using the Java plugin. Andi, how we can override some properties in this case, the version property. What we're not going to do is to see some other configuration We're gonna play to a built in this case. We gotta look at source sets. So we saw with the last build we did that. We had this repositories directory and in here saw straight tree with Maine and then Java on the automate and resource is so this project, we have the source directory, but notice there's no main directory. This doesn't have the standard project layout here to hear the father incomparable side security. These the fact that make up this project that just there just four Java files here. So if I look at the build file for this project, we have the Java plug in applied. Let me have some dependencies. So we haven't talked about dependencies yet. We'll cover these in more detail in a couple of modules time. But essentially, this Java library requires log for J J unit on the jacks, B A. P s and these fast you're just stored on my local farm system and these the father amusing within the project. So I'm just referencing those files for the class path here. So what happens if I try and build us? So this is the build file. Build a cradle. If I do, a grade will build, build executes. If I go and look in the directory, we see there's a build directory, but notice there's no classes directory built. However, there's a Lives directory, and then here. Sure enough, it puts a jar file. If I take a look inside here, then this just has the matter into director inside with the manifest. There's no classes inside the jar file, either, and this is because the Built has no longer knows where to find the source is. Remember, the Great relies on standard configuration here, so it expects the sources to be Insourced Main Java. But they're not just inside the source folder. We can fix this by specifying inside the build file something known as source sets. So inside my build file, I cannot a source set closure that inside here I can specify to further entries one for the main sources on one for the test sources and then for each of these for Java, specify the directory. So my Java source directory is sourced. SRC on the test if I had any, would be in the test SRC folders attest. Source. Fuller. So now that I have that, if I go back to my terminal window and run the build again notice now the compiled Java task has run. And if I go to the build directory, there's classes. Now there's the four Java files compiled into class files On the Lips directory, there's a jar file and again, if I renamed this to a zip file, look inside there. We now have the classes inside the jar file as well. Everything now works as expected. So I mentioned at the start of this module that there's more than one Java plugging this Java, this Java library and this also application in the application plugging derives from the Java plug in. So I change this plague in here from Java application. Go back to the eternal window on run grade. All tasks scroll to the top. There's a new task added cold run. What this has to do is to treat this Java coat is it was an application and run any main method it finds inside this application. To do that, I have to specify the main class name for the project. So in here my main classes, the hash class on a fully qualified name for that is calmed. Operas sight our security deal hash. So I say Main class name. This is a property that's brought in by the application plug. It set the value of that name. If I go back to the terminal window and do a grade A ll run, we'll look for that class name and execute that as if it was a main class for me and indeed just prints out. In this case, there's no Commander Log once passed to it. It just prints out the help that I've supplied. Within that application. This was all done with a groovy TSL. It's very similar with Katelyn D s L. The cotton in code would look like this. So the plug in at the top is application. The application closure as the main class name, which is the same in this case and compiling Java code hit source sets. I have a mane and a test or set on specifying Java directories for each. But to do that rather than using source does directly. I have to call set source stirs and then passed them a list of strings. So the job one we have source and further test one. We have test slash source other dependencies of the same and again will cover dependencies in more detail a little later in this course. So if I go to my terminal window and change to the coddling script directory here we have the build upgraded our Katie s file. What? I can do the same thing so I can do Gretel bells followed by grade A ll run and again we run the Java code, in this case, just built with a Carling style build script. But we saw a building Java coast. Yes. Using standard Java command line against the hash class to run this coat.

### **Extending Java Builds**

So the Java plug in is highly configurable. And in this clip will take a look at some of the ways in which we can configure it. So at the moment on this machine, if I run Java minus minus version, see that I'm running Java version 11. Now, it may be that when you do you compilation you don't want to use all the facilities of driver 11. What it may be that you want to target a specific version of Java to run on. We can do that from within the belt. So within my build script, I could add a Java block on that job. A block lets me configure the java plugging. Now here we use in the application plugin. But remember the application plug. It extends the Java plug. It supervise the same functionality. Like to say, a couple of things here that can set of source compatibility. I can set that to a specific version or I can use one of the Constance defined for me. So said this to be sourced compatible Java 1.8, for example. They could just say Java version. Don't version on the score. One underscore ate the same thing is true for the target as well. So as, well, a setting source compatibility, Aiken said. Target compatibility. If I don't set talking compatibility, it takes on the same value is the source compatibility. So if I go back to my terminal window run grade, all built, then the build is fine. They're not using any language features above Java eight. So if I edit this file this time inside intelligent idea and rather than declaring the type of the variable here, save our which is a Java version 10 language feature, then go back to the terminal window. Run great belt and I get an error. My God and error. Let's set the source compatibility to be Java eight. So I go back into the build file, remove the compatibility, go back to the terminal window and rebuild Everything now works so you could enforce Java compatibility both for the source on for the target from it in the build file. So what else can we do? What? We can also generate java docks. So from here, if I say grade a ll Java doc, remember that Java doc is one of the tasks added by the driver plug it this generous that drama docks for me. So if I go and look at the output from this that in the build folder, you know, have a doc's folder and in here we have the Java Doc. If I open this up, we get the standard Java doc created for us. So now that we have this, we can configure our job block even further. So at the moment when the generator jaw the jar contains the class files, the manifest any resource is. But we can start creating extra jobs as well as part of the build. So there's two things we can do here. We can create a Java doc jar on day sources job, which means that when we publish this project, we can also publish it with the Java docks on with sources if we need to. So again, if I go back to my terminal window and do a build, if I look at the output now that in the Lips folder as well as the standard jar file, we have sources jar far on the Java doctor offer. If I crack this open by renaming it looking inside here, we can see, we have the Java docks provided for us within a jaw fall. So the job of plugging is extensible. There are many other things that we can do with this if we need to. We did this using the groovy syntax. We could do the same thing with confidence intact as well. It's like the cotton syntax is exactly the same is the greatest syntax. It looks like this from the terminal window. If you change the director that contains the Katelyn Build file and run the build here in the build succeeds. And if you look at the output to this time from the K. T s folder, security was like Java build lives. Sure enough, you have the Java doctor on the sources jar in the same way we did with the groovy build. And similarly, if I change something in the Java code here to use a var rather news the actual type go back to my terminal window, do a build the build fails for the same reason we've targeted Java 1.8 hour code is using Java 11 The case. Nobody's seen this for Java code. Let's go and build some coddling code

### **Building Kotlin Code**

it's what we do now is we look at how to build coddling code using a greater build fall and in this part of the module will flip things around slightly. I'll write the build for use in the cotton in script syntax, and then at the end, we look at doing the same thing with the groovy syntax. Now note that we don't need to use the Cartman scripts in tax to build company files in the same way that we don't need to use the groovy syntax to build Java files. So here we can see the same project, essentially this security Tours project. But here we have the father's Kali Falls rather than Java files. There are four files. The hash provided security base and sign that we need to compile. Let's dive in and see how we do that. So this is the greater build phone we use previously to build a Java version. All the security tools code. Where you going to modify this to build a Katelyn version of this code? There's several changes that need to be made to this. No notice here. I'm going to replace all the Java adaptations with Kat in adaptations, some purely building cotton in code here. But if you're building Java code as well, we could leave the Java information in here as well. So the first thing to do is to add the Katelyn plugging to Coughlin supports various back ends such as JavaScript and native builds. Here we're building for the J V M for the Java virtual Machine version. A time of recording is 13 31. I'm not building Java coat so I can get rid of this Java section. For now, however, I am building coddling code, so I cannot a Coplin section into here, and it's inside this coddling section that was specified. The source sets. Let me get her to the source. That section and here was set up the source. That's for Main on the source. That's the test nursing this project. Currently we only have main sources. We have no tests, will cover how to testing a little later in this class. We also need to add another dependency, and the dependency we add is for Colin's J. D. K. Eight Standard Library. Again, let's not worry too much about dependencies. For now, we will cover those in more detail in a later module. However, now that we've added this dependency, I have to tell greater where to find it. And I do that by telling it which repositories to go insurgent and again we will cover this later. So what about versions we saw in a previous clip? We could set compatibility versions forward Java code so we could do the same thing for the coddling coat. But to do that, we have to override the coddling compilation tasks. When we do that by adding tasks, Block saying for the compile Katelyn Task set the J V M target at 1.8. And for the compiled test Katelyn Task set the J V M Target toe 1.8. This is the Build, \_\_\_\_\_ builds, Doc Cradle Dark A. T s. And from here, if I do a greater clean and build, we get some warnings. I'm gonna ignore those for now. If I look in my buildfold, we have my classes. This time it's in the coddling subdirectory. There's main calm little site security in the various compile classes. If I look in the Libs folder, we have my jar file security tours dark Katelin because that's the name of my directory. Here and again we saw earlier we could set our version string here and have this file named appropriately so. If I go back to the terminal window and do a great run, let me get the same output. So the calling code here essentially mimics the Java code. Doesn't exactly the same thing. But now we can see how to build Katelyn files using. Great. So what about the groovy version of the build? Fall? Well, it's very similar to the K T s version. We have a plug in section noticed that here we use the fully qualified name of the plug it or jetbrains cotton in Dodge A V M. As opposed to just saying Cartman and passing it the J V M strength but same version number. You are the repositories maven central again. You have the cotton in block B special with source sets. Here we just add source and test source to the appropriate collections. We have the same dependency. We have the cotton in Son of Live J. D. K. Eight library at it. That will need when we do a compilation when we also Update Tasks Collection For our compile Cotman compiled test coddling set the J V M target to be 1.8. And if I go to the terminal window and changed to the correct directory and run cradle clean belled, the code gets built with the same warnings. We know doing the groovy built. And if I look a security to a star Coplin in the build folder, you have classes and again cotton in Maine Comp little site security on the compile files. If I come back, look in the Libs folder. Sure enough, we have security tools not caught wind of jar again. So with your building using the Cartman script, all the groovy script open is the same on the build. Files are very similar.

### **Review**

So in this chapter, we looked at how to do Java builds. We look at the various Java plug ins. So the Java plugging in, the application plugging in particular, we would like to see what tasks they've added. We've mentioned in passing the idea of library Mrs Application plug ins. We'll get more into the library plugging in the next module, see what that provides for us. We've seen how to modify these builds. We've seen how we can add properties to the script file. We've seen how we can do Java dark, for example. Never seen how we can produce extra output in the Bells, the various jobs that we can create. We've also taken a look at Colin. We mentioned how to create a column build by adding the Colin plugging. I'm providing some over ice for that plug in and that allows us to compile cotton in code. It's now that we've seen that we'll move on to doing multi project builds. Up until now, we've built one single project. Be a library or an application. In the next module will see how to combine project into a single build.

## **Understanding Gradle Dependency Management**

### **Introduction**

Hi and welcome back to the Great Hall build to his fundamentals. Course, This is the understanding Greater dependency management chapter on My Name is Kevin Jones. So we know that projects have dependencies and in this chapter would talk about how we set up and manage those dependencies. We'll see the baking, satisfy those dependencies from multiple places. So, for example, from the file system or from online repositories, we'll also see that Lincoln specify dependencies for different configurations. So, for example, we may have a comparative dependency. We may have a compartment dependency only for our tests. We may have run Tony dependencies and that other things that we can do as well. I will also see the dependencies a cashed. So when we download jar followed from, say, amazing Depository, those job falls a cached on the local false system, the idea being that when we use those jar files later, they'll already be stored locally. You don't waste time downloading them again, so our project may depend on other projects external, a breeze or internal. Avery's So, for example, in the next module will talk about multi project builds. We'll see in that setup dependencies between projects that has to be defined somewheres part of the belt. It may be that we use libraries that come from the public domain. I would almost certainly do this if you're building spring applications or Android applications are using Jamie's for testing than their external libraries. And it may be that we have other libraries that we developed internally money to use those as part of our project. So we need to be able to define dependencies that satisfy each of these things. So the dependencies, our project has can be satisfied from other projects. You could be satisfied from files on the local fire system. It could be satisfied from external maven repositories, and great also supports a V repositories as well. So if your company has an ivy repositories, we can set up dependencies to files within that repositories. Our project may also have many configurations, so we want dependencies that we use that compile type. We love dependencies that we may only use that run time. We may have dependencies for our tests or compile time. We may have dependencies for our tests, run time, and we need to be able to differentiate between each of these things on maybe other things as well. When the game was see that during the course of this module, We also know that the labours you depend on may themselves depend on other libraries. So, for example, if I bring in spring boot that spring boot core library itself is dependent on many, many other libraries, these are known as transitive dependencies. So what? I include a library in my project. If that library has transitive dependencies as well as downloading the library that I need, I'd like to get a download. Those transitive dependencies as well as you can imagine greater supports that one thing we can do in grade A ll is list of the dependencies. There are project users. You can do this in a couple of ways. We can ask if all the dependencies that were used or we can ask if brother dependencies that we use very given configuration. So let's see how we do that. So this is the build fall for the security tools project that we've used a couple of times in this course, and here I have some dependencies defined. I'm not gonna worry about the details of these now, in the next clip will talk about how we set up these dependencies what things like implementation and test implementation means. But just to notice, we have three dependencies you have longed for J. We have Java ex XML buying Jax. Be when we have J unit. And if I go to the TUNEL window on run grade A ll minus Q dependencies. If I scroll back up here, we can see we have a compile class path set up that contains log for J job ex xml dot bynes on DDE That trans defeat depends on job extra activation. If I scroll back down, we could see our test compiled. Class path has logged for J job Ex ex Mel Bind Samos to compile class berth, but also has J unit. So you can see here very quickly. We can set up different dependencies for different parts off the project that shows the dependencies for the entire project. We can show dependency is only for a specific configuration. So to do that, we can run greater minus Q dependencies and then specify configuration. You were saying, I just care about the dependencies for my standard compilation, so I'll run this, we get only the compound class mouth configuration, and this shows log for J. Jax Be and then the transit dependency on Java Ex activation. So let's show a slightly more complex project. Let's take a look at something with spring inside that so have a project here called Jacket Web. Let me clear this, Then. If I look inside here at the dependencies, scroll up. We could see many, many dependencies here. So, for example, we have other projects we dependent on, such as jacket service. Inside those projects, we have things like hibernate on the dependencies that hibernate has transitive Lee. We have things like Spring boot Start a J. D. B C. Then it's transit dependencies and so on and so on. So this is the spring put application. As we can see, there are many, many dependencies that we have here that have brought in forest by grateful by a dependency specification and then the transitive dependencies from that specifications

### **File-based Repositories**

so the satisfy dependencies. We have to specify a repositories. So repositories a location where jar files and other things are stored, and these repositories could be remote. So, for example, the standard maven depository. We also have what are known as local repositories, which you shouldn't use. This is essentially the local maven cash on your false system. But we can also low jar files from the far system. So from specified locations on the file system. And you might want to do this, for example, if you want to check in live, is that your project needs, rather than relying on remote repositories such as maven. So to specify repositories, we use the repositories entry in the build file on for file based repositories. We say flat, D R. And then specify the names of directories that contains those files. Let's take a look at this first, so this is a build be seen earlier. We're building security to his project, and I said I was security to his project, depended on a couple of jar Falls lot for J. Jax be and for the Test J unit, and I sort of said in passing that we can set up dependencies using this file section. So what we doing here? Saying Go look on the file system, look in the lib folder and go and find these jar falls. And when I do a build the code builds and uses these jars to satisfy these dependencies, we can see this. If I comment this out and go to my eternal window and do a clean built, the build fails and the build fails, because the various files that we need and no longer being referenced on the compiled class path and putting these back in and going back and running a build again. And now everything succeeds. However, this is not the best way to set up these file dependencies. Improbably, here is is that if I need another dependency, I have to go add up by hand as well. So what I can do instead is rather hard coding where they're going to find these things from the farm system. In the dependency section, I can add a depository section and in here if I want to use the fire system, I can say flat D I r mentality use the lib directory to go and satisfying my dependencies. Then what I can do is specify the qualified name of the file to use for the dependency. This is a specific format, so I can say implementation. Log for J Colon log for J Colon version number without the jar file. Here. This is one way off for mounting the version information. We can also do it another way. So you're sometimes see it. Then this way. You say group. This is Java. Ex ex Mel Bind name Just Jax. Be a P I on version number just 2.3 point one. I'm going to see more on this later. But essentially what? It means that we can group fast together with this case under the Java ex XML Bind Banner and Jax be a P. I is one of those files that comes as part of that group when we could do the same thing for J unit as well. So here we say J Unit J Unit 3.8 point one. So now what will happen when I do the build who were going look for these files in the lib directory on my local far system and a game will come back and cover what implementation and test implementation means in a little while. So again, if I go back and build us, it'll builds. Okay, Just approved this. Let me remove Jax, be from the build build again. And it now fails the same as we saw before to doing this. We're using the local far system and specifying depository on that local follow system. And as I said, this is useful. If you want to store all of your dependencies locally. Another good reasons for doing that. You might need all of these things checked into source control, maybe for auditing purposes, for example. So now we've started looking at repositories before we move on to looking at remote repositories. Let's take a look at what these scopes mean, So things like implementation and test implementation.

### **Configuration Scopes**

So I've said the dependencies, a result from repositories. We've looked at the file system and in a moment we'll take a look at maybe repositories such as J. Santa and Dave in Central. We've also said that when we resolve a dependency, we have to give it a configuration. We give us a call, a configuration scope so it given dependency exists within a given scope. No scopes are implementation. The implementation scope. If you like the main part of our project. This is our main coat and this is broken into two parts. You can have a compile only scope of runtime only scope. The implementation scope covers both of these. What this means is that if I add my dependencies, the implementation scope, that dependency will be used. Both that compile time on that run time for the main code within the application. If I'm off the implementation is compile only its use. Only a compile time for marketers. Runtime only. It's used only at runtime. Similarly, for test code, we have a test implementation scope. Nephew dependency is in that scope issues. Both were compiling the test coat on at runtime with a test code and then we also have a test compile only on a test run time only which is similar to compile only and run time only scopes for the main block of code. The test scopes derived from the non test scopes attest Implementation derives from implementation. So any dependency we have in the implementation scope is also used by the test implementation scope. Then the same is true for test compile only and test run time only. I was just going out some dependencies, the appropriate scopes. So this is the call transcript fall. We've seen some of this already in the groovy syntax fall. So here we have Ah, depository, which is J. Santa. We have some dependencies. We have the implementation dependencies and the test implementation dependencies so very similar to the way it's set up in the groovy falls. However, one thing we can do is to factor out diversion numbers here. So the way we do this for the Courtney Files is toe add a greater properties fall and in the Properties file set of properties, we going to use another script fall. The properties in this case could be the version numbers and we have three you have the log for J version and we're currently using one dot to 1.0.8. In this particular case, we have a Jax be version, which is 2.3 point one on old J unit version 3.8 point one. So to use these in our script file, remember, this is Cartman, then caught. Then we can declare, read only values we did up there using the file a key. Would you say Violet's lock for Jana school version and the type of this is string. And then we say by project. And if you know Colton in the bike key words means this is a delegate were delegating to something called Project just part of the cotton and support inside. Cradle on. What that says is go to the properties file, get the value from there, so lock for J underscore version here, we'll take on Lord for Jana school version for the properties files that'll become one dot to 1.0.8 and similarly for the jacks be version on the J une version. And then what we can do is in these implantation strings is rather than hard coding, the value is we can use these variables, it's going to say a dollar look for Jae Underscore version and similarly for the other two values. What you often find is that these versions are used in multiple places within a build. Far so in the future. If I need to update the version, I only changed in one place and that'll be the properties fall. So just to prove it still works, if I go to the terminal window, I mean the coddling Foley. Here's here's my build Doc cradled R K T s. If I do a clean belled, then everything works. Okay, so what we have that in place? Let's make another change. So here currently we have locked for J set as an implementation dependency. If I go to my terminal window, let's do a clean for by compiled Java, which succeeds and then a test and the tests exceed. So if I go back into, like a T s fault, let's change this dependency to be compile only go back to the terminal window on a game, do a clean followed by compiled java, then do a test. The test fail knows they failed because they're getting no class. They're found error. That's because the log for J class is no longer available on the run time class path. So in this case, we're specifying, I want that log for J version only available at compile time of the main classes. This is inherited by the test classes, so I can also add this to the runtime only class path for tests as well. Let's see if that succeeds enough for ungrateful test. The test succeed. It's a valid log for Jay to the runtime class path for tests. We could see how we could break this down if we needed to now not recommended you do. This is This is just an example. Sure, the various options that we have with these scopes. So just one last thing. So you see now used Constance is version numbers in the cotton in script. But what about in the groovy script fall? It's just done slightly differently. So in here, the way we would do this typically will be to add a build script section in the Build Script section is where we can figure things that we need during the running off the script not necessary during the building of the application of running of the application. One of the things we can do in here is that extended properties. It's like you say e x t that inside here. Say something like Look for J on a school version and set this equal to a value will use 1 to 17 is the log for J version here that property value is now usable right \_\_\_\_\_\_. This script so I can know do is here. It's a dollar log for Jay and the school version, but in groovy, we can't do string interpolation in double quoted strings. It's when he changes from a single quarters string to a double courted strength. And then if I go to my terminal window, I mean the groovy directory here. And if I do a great deal clean build, sure enough, everything is still built on. The test is still run correctly.

### **Using Remote Repositories**

So we have a fair ways of defining remote repositories. We can use some well known names, for example, Maven central points at the Maven Depository. What J center points at the J frog. Been trade repositories. You can use either one of these just by name. You can also special fire depository via your own. So, for example, here we using J Center using an http or all. You can also specify a custom depository. So here we're using a maven depository. But this time this is our company depository rather than made in central. You can also use maven local that says, use your local maven depository, which essentially means the cash on your local far system. This is not recommended. We can also set up ivy repositories. If your company uses ivy and you have an I V repositories inside the company, you can set that up in Southern repositories block and finally, you can specify in multiple repositories. So here we have a major depository on an I V depository on any dependency will be searched in both of these places. Let's take a look and see how we use a remote repositories. So how do you use a remote repositories, we can use Maven or J. Sander, and we can specify that in the repositories section. But either saying Maven Central or J. Santa and they might be good reasons to be using one over the other. It may be that your company is standardized on one or the other, or it may be a personal choice. I tend to use J Center. You look at the J Center website. It tells us that generally it's faster support Cdn. At one time it was the only book depository that only supported https. Jason tells us there's a super set of maven, so anything you find in May even should also finding J. Sander. So, having specified repositories, how would we know which versions of libraries are available at that repository? How would we know how to specify the names for those libraries? Let's take a look at the maven sight to see how we can do this. So if I look here, for example, that may even depository and look ATT log for J, we can see that lock for J 1.2 point one sam is available. If I look in the grade all section of the tabs. Here we can see the fully qualified name of this artifact sweetness. See it as a group, which is log for Jay hears. The name is locked for Jay, and it has a version. The compiles part of the front of this is wrong for greater Now this has changed to implementation, so artefacts are grouped, and that's with a group named Come Strong. They have a name within that group, and then they have a version so we can download a specific version off the artifact so I can grab that definition. And I can use that in my build file. Well, as we can see on the line above, that can use a short and version of that definition. I can update our version number here for 8 to 17 for example, so I prefer the short time version. There's just less typing we can see here. We can mix and match for different dependency is a short on version or the long hand version. So for the jacks, be one Ali that, as is leave the long hand version off that in the game, if I go back to the to the terminal window. Do a clean on the build. Gonna get in the build of succeeded If I go back to the build, file on comment out the implementation section here, try this again. Then again, the build fails, but now it's satisfying the dependency from the remote repositories.

### **The Gradle Cache**

So cradle to save time on the download caches files. So any libras if it needs any murders, that we need a cash on the file system so the cash is filed based the metadata about the fire on the farmers themselves. The jar files, stored separately within the fire system in the depository, cashes their independence. If you're using both I've repositories and maven repositories, and it cashes follows from these repositories independently. This means that when we do a build for the first time, when it tries the sites where the dependencies you looked in the cash, this is the first time we've used these dependencies. There won't be there, so we'll download them. But in subsequent Bells, those files will be in the depository and so don't need to be downloaded. But it's get added to the compiled class path from the cash. When great will store something in the cash, it takes a hash off the file on the file. Him in the cash is based on that hash. So when Gretel checks to see if a family's re downloading, it looks at the fall remotely Recalculates the hash and checks that hash against the filing in the cash. Now we can't ask griddled to refresh dependencies. There's a minus minus refresh, hyphen, dependencies, flag. The idea here is that the cash may have got out of sync. Remote repositories. So, for example, somebody may have updated the remote depository with a new version of file. When Cradle does the build, it looks in the cash and finds that file some minus minus. Refresh dependencies says to Gretel, Go on, looking remote depository. Re calculate the hash. If the hash is different to the files in my cash, then re download that file and restore it. So I checked to see if anything needs to be re downloaded. Now, we can also delete the cash so I can go ahead and delete the entire cash. If I do that Grade A ll, but they just re download the files as it when he needs them. Let's take a look at some of this now, so I'm on my Mac. But in my users Kevin Jones Doc Cradle folder There's a direct record. Cash is if I looked out here and look in the modules to folder on the files to 0.1 folder, we can see the various cashes in place. For example, for look J unit these the cash is for J unit 3.14 point 12 4.13 And every dependency ever use for grade A ll is held somewhere in this cash so we can safely leave this cash is folder. If I do that and go to my two more window, I mean the security tools project here. If I do a griddle built, then go back and look at the file system. Let's see, the cash is directory has been recreated This modules two fast 2.1 that we've got things like J unit and log for J. Similarly, if I go back to the terminal window and changed to the jacket project, this is a full blown spring project. If I do a great will build here, this takes 41 seconds to run. And again, if you look in the file system and sigh cash is now we have many, many farms that are being downloaded on these. The dependencies for that project on the transitive dependencies for that project suggested sure that we do use the cash. If I go back to my eternal window do a clean built. This now takes three seconds to run. So it was taking something like 38 seconds to download all the dependencies that we need. So the cash saved us a lot of time in this case.

### **Review**

So in this module we looked at dependencies on almost every project we ever going to create will have dependencies. We need some way of managing those dependencies within a project we've seen. We can satisfy those dependencies from multiple places. We looked at Saks Fine appendices from the file system. When we said that a good reasons we're doing that you might want all the dependencies check it a source control, for example. We also have to satisfying dependencies from maven repositories such as made in Central and J Center. We mentioned also, if your company, as ivy repositories greater, can use those to satisfy the dependencies. We also talked about configurations to the fact that we can add dependencies for a compilation configuration say, a test compilation configuration that these two are separate things. And finally we talked about cashing. So when we downloaded tendencies there cashed little girl far system. The next time Cradle needs those dependencies, it checks the cash. If they're already there, that will use the dependencies from the cash. We can try and update the cash from the command line and we can also freely delete the cash if we need to and get greater to re download those dependencies again as part of the built. But be aware if you do that, Build could take much, much longer. Now that we've done that, let's go take a look. A testing inside cradle. I will take a look at both testing with J Unit for On with J Unit five.

## **Creating and Managing Multi-project Builds**

### **Introduction to Multi-project**

Hi and welcome back to the Cotton Biddle Tools. Fundamentals. Course, this is the creating and managing body Project builds module. My name is Kevin Jones. So in this module, we talk about applications. The consists of multiple projects each those projects will have its own build file. So what we can do in grade a ll is create what's known as a multi project built to do that we special fei a top level build file. We also need to specify the top level settings for that's settings. Far identifies the sub projects that make up this top level project. The build file is used to configure projects. We could do all the configures that we need to within this build fault. It's also used to set dependencies between projects. So if we have a Web project that depends on a service project, we can specify that within this top level build fall, each sub project will still have its own build fart at the project level. Within that weaken set project level properties most secret tasks, but a specific to that particular sub project. Let's see how we do that

### **A Simple Multi-project Build**

So here we have a direct recalled jacket, and this is going to be my top level project. Inside here we have two subdirectories, one called depository. On in there. There's a source folder. Now they build our cradle file. Let's build a greater fault. It essentially empty, least minimal. There's a play plug in Java on aversion string. We also have this jacket service project we'll see later. The jacket service depends on depository uses repositories. Again, there's a source folder on a build a cradle far and again that build our greater far it essentially empty. What we going to do is to marry these two together into a top level build. So within the Jacket folder will have the build's ghost. We need to make this a multi project built, and they would do the same thing for the continent's intact as well. So the first thing we'll do is to create a settings file, and in here we include the sub projects that are going to be built as part of this top level project. We're doing this in groovy. Initially, the satisfied looks like this. We include both the repository sub project on the jacket service sub project. If I say this, we say this a settings dot cradle and it goes in the top level projects directory. If you think back to an earlier module, we talked about the greater life cycle. The first phase of the great life cycle is the initialization phase. And during the initialization phase, Grable decides which project need to be built on. One way it does, that is, by looking in this settings fall. Once we have the settings file, the next step is to create the build, file itself on a multi project build fall consist of several parts, so we can add properties to the file. That effect, for example, all the projects we cannot \_\_\_ since the fall to define things that affect all the projects on. We cannot project specific sections to the fall. Let's start with the property so we can set the version string we want to use for all the projects. And here we have version equals 1.0 dot snapshots. This is our project file, and this will be build our cradle. Let me say that so we can add a project specific settings here. When we do that by calling the project function, specifying the name of the project noticed with Colon in front of it. And then the closure at the end is where we specify those project specific settings, which we don't really have any yet. This shows the syntax, one of the things we can do for these projects. Pacific sad things is to specify dependencies. So, for example, the jacket service will depend on the repositories service. Before I compile a jagged service, I needed compiled depository. First we do that by saying dependencies giving this a closure on inside here saying this is an implementation dependency in fact depends on the depository project. So each of these projects is a Java project. While I can go through every project, another Java plugging at the project level, I can also do things for all projects. So in here I can say all projects and give this a closure and then as information into here, that a place, all the projects so I could say apply plug in Java and this is using the old syntax. This gives me a build file like an hour run to build all the projects. Let me say this if I go to the jacket directory and here we have my build, a cradle, my settings Dark cradle and the two sub projects. I'm from here. I do a grade A ll build minus I. Then we'll see that it builds both projects. It's my scroll back up to see the jacket service being built. And if I scroll back down, we'll see the depository being built swat about doing the same thing. But with a cotton in build script, the idea is the same. We need a settings greater fault. And that is exactly the same for the cotton and stripped on for the groovy script. So we say this now in the K. T s folder and again the jacket top level project course settings Dark Cradle. Namely the builds, Doc Radar. Katie s file. So again very similar. We cannot project level configuration. We can specify dependencies between projects. We can specify configuration for all projects again if I save this jump off to the terminal window and changed the directory that contains the cotton and build script and again do a build minus I they gave me see, the depository is being built and the service has been built

### **Configuring Multi-project Builds**

So we've seen a multi project build with a couple of projects. Now let's take a look at a slightly more complex example. It's the same project, but with most projects added. And I just want to take a look at the sorts of things we can do inside the top level project to configure the builds. So here, for example, we have a build script. This is a groovy build fall. And in here we're setting some properties. We have this ham crest version, um, Cressida Testing Library, Commons logging. So who does some logging and fly away, which I've mentioned before? This is a tool to help build database migrations in Java. What we're doing inside the e x T block is setting version. So we got the time crest versions 1.3, for example. The point here is that we can use thes versions anywhere through the coat so I can use them in this top level build stripped. And if I stroll down here, we have my dependencies block. We have a test implementation scope. Are we saying home crest version that. But we can also use these values in other places. So if I look in this jacket service build file here. This is one of the sub projects. We have a reference to this job. Ex inject Lavery. We're hard coded version number here. If I said this to be a value, let's say job, ex inject, underscore version. Then go back to the main build file. Not at the top. Hair at Java X underscore. Inject version to be one that will get picked up in the sub build. And if I go to the terminal window and to a greater build that everything feels successfully So what else can we do? Well, notice. Here we have a plug ins block. When the plug ins block, we're playing the Java plug in that will apply the plague into this project and also projects. We're also identifying the flyway plug in here. But what we're saying apply falls. So here we don't want that plug in applied. We just bring a reference to it into the project. We'll see whether I play. Gin is used in a moment. We have a sub projects block previously read on all projects floor. Anything Is that all projects? This configuration is a plate. Every project, including the top level project in sub projects were playing these configurations only told the sub projects then So here we doing things like setting our version information. I would Java source compatibility and target compatibility adding repositories that applied to all the projects. Nothing dependencies, that applied to all the projects. We can do these things in a number of places. We can apply configuration within this file for all projects. Within this far all sub projects, we can apply configuration obviously on a project basis. In there Build files. We can also apply configuration on a per project basis from this file. So I scroll down here. It's the first thing to note Is that for things like the hydrate depository, we say this is dependent on the repositories project jacket repositories dependent on both the hybrid depository on depository project and so on and so forth within these dependencies blocks, I could also add per project dependencies in the sub project. Block it or we can do it in the Project Build files. This is either a matter of taste Or maybe your company has standards around this. We can also configured groups of projects. See here for example, these three projects depository the service and the where project. I need to add a plug in on some task to each of these projects. So I could duplicate this code and put this coat inside. Each projects build file, or here I can write some groovy code. So I'm doing here. I'm adding a project block for each name the project. So jacket, depository jacket, service, jacket, web And inside that project block, I'm applying the flyway. Plug it. I'm adding this new task call. My great test. Doesn't really matter what this test does. We can add tasks on a per project basis in this top level project for just a note. We're applying this or fly away D B plug in here. If I scroll back up when we referenced the plague in here, we had apply force. We don't want to play the plague in at the top level, but we do want to apply it on a pro project basis.

### **Configuring Multi-project Builds with the Kotlin Build Script**

so things are similar for the Kaufman script version of the build File to notice. Here we have a bunch of values defined, and these are defining the greater properties fault as we saw previously and again, if we need to, we can use thes right throughout all project. So notice that, Hey, I have a build script section. Remember that little script is used to configure the execution of the script. So any dependencies, for example, we got into here He used by this script not by the job of Build processes. So here we're having a dependency to the class path on that dependency is around something called H two, which isn't in memory database we can use for testing notice. Here we have to redefine the H two versions strength, even though it's defined above. So if I comment that string out and try and run a build, the build fails with unresolved reference. Go back and put that back in again on the build. No succeeds. So there is some scoping going on here. Notice also, but in the plug ins block, I can't use these values. I have two hard code. The version numbers and That's because the plug ins block is what's known as Item Potent III. When we run this, the plug ins block has to reproduce the exact same output every time the script is evaluated. If we relied on values here, those values could change and therefore the plug ins block my produce different output each time it was run. Having said that is then very similar to the groovy based plug ins block. Now, the plug ins identified by I. D. We have a couple of apply forces here, so we don't want spring dependency management or the spring boot plug. It applied here really specifying them here. We got to use them elsewhere. Notice as well. We have a top level Java block here on. We use this to configure the source and target compatibility have used the fully qualified name of the Java version here. We don't need that. You can put that back to Java version and then we have a sub projects block and again inside here. We gonna play the Java plug in. We can specify any repositories that we need. We can specify the dependencies that he used by all our side projects and Similarly, we have the different projects. We have a report, every project. There's no dependencies. We have the hybrid repositories depends on depository jacket repositories that depends on hybrid, depository and depository, and so on and so forth. And then down here we have the jacket Web project, but we're applying this spring framework, plugging on this spring dependency management plug in as well as the wall plug in. And again, if I just go back to the top of the file here inside the plug ins block, we include the Spring Framework would plug in on the spring dependency management plug in, but neither of those is yet applied. He was just referencing them. We can use them later inside the script, and finally, we cant similar configuration to a list off project here. So a jacket, depository jacket service and jacket web all need the flyway plug in a place we all need. This new task on migrant test as well as configuring all projects, sir projects individual projects. We can't take a figure. Groups of projects

### **Using the java-library Plugin and API Scope**

So when we noted building Java project earlier, we mentioned three plug ins. We mentioned the Java plug in. You mentioned the application plug in. We mentioned the Java lively plug in when we said we talk about the job, A library plug in a little later on. Now is that time. So here we have a multi project build. We have a top level build phone, which we can see here. This is the groovy version we can see. Hey, we using the Java plug in. We're using J Santa as repositories when we have to sub projects. We have this J curl lib sub project on a terminal client some project. And the idea here is that we're writing a Java version of Cool. Now we're not going to implement that here. It's a gesture to show how we use the Java lively plug in on the A. P I scope in particular so we can see the tumult client. This would be the front end. Toa library relies on Jake her lip and Jake. Her lib is where the actual functionality would live. The idea is that we could have multiple clients built on top of this we could have a term a client, for example, maybe a swing client and other clients as we needed that. If I look at the build file for the terminal client, we can see it applies the application. Plugin doesn't do much more than that. This is to run the application. So if I look at the build file the Jake her lip client, we can see this is one dependency. It depends on the Apache. Http. Components http Core library This contains a \_\_\_\_\_\_ P bits that we'll use as part of our code. I notice it applies the Java Lavery plugging rather than just the Java plug it. I would see how to use that in a moment. So if I look at the code for this library, it was a client class here. The important thing here is the return value. So it returns something of type. Status line Status line is part of the Apache http court library Tow Any consumer of this client will also need access to the Apache. Http Core library. So I look at my term our client code we can see inside here we create a new interest of client. We call client dot get image. And if I hover over this, this fails because we currently don't have access to all the Apache HDTV status line. So I can fix this by going to the build file of turmoil clients, adding a dependency section and in here adding a dependency on http core If I go back to my terminal da Java, the air has gone away. However, if we think about this, I shouldn't we k about this implementation here. I shouldn't care that the client I'm using exposes something from 1/3 party library and then relies on me to know how to resolve this library. So the Java Library plugin provides a way to fix that. So if I under these changes I made Thio the term or clients build file, so notice that the era has now reappeared while I can now do is add it the J Kulaib build file Doing here rather than saying implementation, I can say AP I It's what implementation says that this project Jay Kulaib relies on an implementation of HDTV core and use that on my compilation class path. Let me compile the coat. What a P I says is do everything implementation says, but also expose this dependency Http core to anybody that uses J. Curlett. The thing that's using J Kulaib is the terminal client. So I go back into my term recline java file. The error goes away again, but I haven't. I did anything to my dependency section here, so I'm relying on things upstream if you like. J curl up in this case telling things downstream to my client in this case, the set of dependencies that we need to satisfy J curled lip. And that's where Java library comes in. The same is true using the Katie s Falls as well. So without going into detail here, if I look at the top level built the game, we have my J Kulaib project, my terminal client project. That depends on J curlett. If I look inside J curled lip, it uses the Java Lavery plug in on a P I to specifies that http court is used downstream of May. And if I look in my terminal client case Yes, well, there's no reference to the http core library, but if I look in the sources everything compulsive Fine. So that's how we'd use Java library on the a p I scope

### **Review**

So when this chapter we looked at the fact that applications generally consists of multiple projects, then we looked at how grateful supports multi project built. We looked at creating a top level project when we saw how we could configure projects at the top level. We looked at the settings far that defines which projects make up this overall top level project, and we then looked within this top level project file. We saw there are various ways but configuring or the projects. There's an all project section. We configure the top level project and all the sub project. There's also a sub project section. We only configure the sub projects. You saw that we can apply plug ins from the top level when we saw the Wigan apply Plug Ins Justice sub project from within this top level project. We also looked at the fact that dependencies applied within the top level project could apply to all the sub projects as well. When we also looked at variables to setting values or properties within the top level project, it can also be a played within the sub projects. Then we also stole the can override settings inside a specific project. And indeed, each project still has its own build file. You could settle usual settings within that build file. So next we'll take a look. A testing. We'll take a look at setting up J unit for unjaded five tests within a greater built.

## **Using Gradle to Manage Testing**

### **Introduction**

Hi. Welcome back to the Greater Build Tool Fundamentals Class. This is the using griddle to manage testing module on. My name is Kevin Jones. So most, if not all projects will have tests on those projects will typically be using either J unit for J Unit five. So in this module, we look at setting up J unit for to be used inside Gretel. We'll also be looking at how we set up J Unit five to be used inside Gretel. They're going to be times when you won't need to run all the tests in a project. It may be that you want to filter test on, only run a certain subset. Well, we'll see how we do that in this macho. We've talked previously about source sets and there is a test so set and by default, cradle books for unit tests in source. Test job. Just Java unit tests. Of course. And you can change that. Megan, we've seen our previously in this course. The outputs from the test build our to build classes test on any reports that generated are put to build reports, test speak, looking here, for example, for any HTML reports Associated with this testing to set up the tests we need to add a dependency on the scope of dependency is test implementation. And for Jay, Unit four, it looks like this. So the J Unit group, the J Unit Library, and in this case version for 12. Now, as we'll see, we'll also need to enable test logging. The way we do that is to use a test block. We can use this test block to configure anything about tests. We had a test logging block on in here. We specify the events that we want displayed when we run the tests. Okay, so I was seeing this. Let's go and write some J Unit Four tests for our code.

### **Testing with JUnit 4**

So here we have a project and this is just my security to his project. And in here we got to run tests using J unit four. So in here we have a source folder and this is the standard structure. So we have test Java, but inside here we have our test coat. If you keep going across, we have comparable size security, and then various tests classes. Are we going to run? So for looking my Build Grable file, this all looks fairly standard. But remember that we need to put into here the test implementation scope for J unit four. So we do test implementation J unit J Unit for 12 in this case, then, once I have that, I can go to my eternal window so far on greater clean builds that will build a code on dhe. Run the tests to remember that build as well as doing a compilation. Also does a check and check is running unit tests. We can see that more clearly. If I do minus minus console playing here, you just keep playing Consul Ope it on. As part of this will see the tasks and the order in which they run, we can see at the bottom we run. Build on Build depends on check and track. Depends on test classes and so on and so forth. What the top The code is compiled in the test code is compiled, and then the tests were run. And the odd thing here is we see no output from the tests other than the fact that the code is run successfully so we can fix that. So, back in my build, great. I'll file. I put a test block and in here we can configure the testing at a test logging block and in here, Gunn specify what logging I want to see from the testing. We can figure this by adding more called events so I can say events and list the events that I won't raised and therefore the law Go put that I want to see. So let's add the failed test event, the past test event on this skipped test event that will look something like this for this to work. I also need to add some imports to the script file. I'll add those to the top, so I import the test exception format on the test log event if I go back to my terminal window and then run the same again Now we get a list of all the tests and you thought what we get a list of test the past, a list of test that fail or a list of test that I skipped. So what happens when a test fails? So if I go into my coat, make this code wrong, the test will fail and then go back to my terminal, and this time I'll run the test. But without the consul plane on now we have one failing test. If I scroll back up here well through the failed test in red, so it tells us test have failed, and it shows us what the failing test is as well. And to make this easier to read like a gesturing move the past event from the greater file. So if I take this out on rerun the codes, then we now just get the list of failed tests on the fact that one test has failed. So if I look at the K. T s version of this the cotton script version, well, see, it's very, very similar almost exactly the same. So we import the test exception Former on the test log event. We set up the dependency for J units, but the difference is how we set up the events. So here we say, the test logging events is a set of these, the events that I want to use it again. It's test law give Ain't Failed past and skipped so similar, but not exactly the same. But here's the script. For months it will work in exactly the same way.

### **Improving Logging**

so that test open we saw isn't great. When we can't improve on that, there's a plug in and we can add to a greater build called the test lager. And this is the website here. If we use this, we get much nicer. Logging open. Let's add this trauma build. Fall the build. Fight County looks like this. So I'm gonna take out the test section. I'm going to add the plug it. And remember, this is a community plug it. So we use the fully qualified name of the plug in on the version is currently version 200 and then I'm going to configure this longer. We configure it by adding a test lager block to Arbil file and there are many different configuration options we can add to this. I'm showing all of these here. You can have as many of these you need two keys of the default values and you can change the values as you need them. So with this in place, if I go back to my terminal window and now do a clean built, we get much nice output for our tests. And again if I change the code to cause the test to fail. Rerun the test game. We see the failure. When we see the failed test in line with the rest of the tests here, it's obviously have to. And again your country may have policy on this. This is quite a nice little addition to our testing. And again, the cotton in code is very similar. So we had our plug in, and we add the test logger section. Notice the note here. For some reason, we can't say theme equals. To set the theme, we have to call either call \_\_\_ theme or we use this theme type. That theme type is imported at the top of the file. And again, if I run this with a Katelin build script Marca, rather than standard that the groovy build script had we got similar output. It's so different we got the text rather than the wording past. For example, here

### **Testing with JUnit 5**

So for J Unit five, things are slightly different. We have to tell grade A ll to use J unit. We have this year's J unit platform method that we call to do that. We can still use logging events. We consider this a test logging that events and special events we want to use. But also the dependencies are different. So here we have two dependencies. We have the J unit Jupiter a P I for the compile time scope. The Jupiter was the name that Jamot fivers developed under. Let me also see the runtime engines. We have this run time only configuration, so test run time only. And that's for the J Unit five Jupiter engine. So let's see how we do this in code. So for J unit, for we had something like this. We have a test implementation of dependencies of J unit. June it for 12. Version four of your unit for Jane. And five, we need to change that reference when we change it to all Jana Jupiter and then Jupiter. A PG version five. But remember, we also need to add a test run time only we need to add our runtime engine for J Unit five and then in the test block. We need to configure Grader to use J on it. And for that we call use J unit platform. So with that in place, if I go to my terminal window, I mean the directory that has tthe e j Unit five coded. If I run clean and belt. Then again, everything works. So to prove this, if I take out the use Jamie platform and go back to the terminal, then run clean and build a game. It looks like everything's succeeded, but notice There's no logging open here, so the tests haven't run. So be careful and be aware that you need used J unit platform in here. If you look at the car transcript version of this, then again it's very similar. We have our test implementation, which is J unit due to a P. I have our test run time only with Jupiter engine on. We have the used J unit platform. If I go to my terminal and changed the Katelyn directory, do a clean belled. Then again, the same output, the tests of passing in this one as well

### **Filtering Tests**

so we can also fill the tests. We can do this several ways. So within the test configuration, we cannot have filter. We could include specific tests will become wild card. This we can, aunt. A new test task in this case is called Single Test Weaken Special. The group this belongs to us when we list out all the tasks appears in the correct place in that list with a description and inside here we can do a filter. And, for example, here we can just specify an exact test to run on Dhe if I wanted to. The same thing for J. Unit five, the code is very similar, except inside my task. I have to call use Jane a platform to tell the task to make use of J Unit. What? We can do this from the command line so we can run great. I'll test and then say minus minus tests and again, wild card. The test we want to run. Let's see how we do this. So let's see how we add a single filter. And this is using the groovy syntax. So in here I'd say task and give the task in name Mason the name This single test here. When we have a task of type test, I can give grade a ll. Some information about this tasks when your special by the group is going to be in This is for you. I purposes. And I can also give the singer description again disappears on the terminal window. We can specify dependencies in here, so I can say depends on Miss Depends on test classes task. So we now over depends he set up. So if I try and run this task, it'll run tests classes first on all of its dependencies, Obviously before that on the inside here, I can add a filter on the filter I'm going to add Well, cause it's just to run one single test to be used include tests matching and give it a fully qualified name of the test. So if I go to my terminal window so first of all, if I run grade A ll tasks in the verification task section, there's my single test task and it's just gonna run a single test if from here I run greater clean, single test, we get an error and it tells me you can't find the test. And the reason for this is that here I'm using J unit five. If I try sort of a test type tasked with J Unit five then in the task I need a cord used J on that platform. I'll also need a set of the logging as well. Let's do that now. Now, if I go back to the terminal and run the test, then the test is no scene to pass. So the cotton in code for reaching a new task is very similar. We register, the task is of type test. We give it a name. We gave it a group and description which help grade A ll. What other task? This depends on. Then we call use June and platform, set the logging events and then add the filter for this particular test. We've just other tasks to run J Unit five tests and we've seen we need to call years Jane and platform within that new task with Jay. And four things are slightly cleaner. So when we run jailing four tests, we don't need a called used J minute platform. We still need a lot of the events, however, so here we have a similar task or single test. This is within the jail. Unit four project. It looks the same as the jail and five type task. But here we know calling used Jamie Platform. Just other test logging instead. Now, earlier, we saw that J Unit four tests run real. Our test loca. That's this build file here. So in here we have the lager configured and also in here. We have our new tests run a single test. I noticed No in here. If we use the longer this lager is configured for any type of test that we run, so I don't need to add any extra configuration into this task. It picks up the global configuration, if you like. That's already inside the build phone and the same will be true. If I was running J Unit five tests here, except I would still need to call use J unit platform within the task itself. I wouldn't need to set up any filters. I wouldn't need to add any events on. I wouldn't need to reconfigure this external lager

### **Review**

we've talked about testing. I've done this because these days all projects have tests or his whole project should have test. We've talked about how we set up Janet for essentially for jail before. We just need to add the dependency on Jane it for. We talked about setting up Jane and five for Jamie. At five. We have two dependencies. We have the compile time dependency on the run time dependency, which is the engine we also need to call. Use J minute platform when we have Jane it five tests. We also looked at how we show the output from the testing when we showed the default behavior where we can raise events and have those events General open for us. Oh, we can ID the next little test lager plugging that gives us, I think, nicer information, a nice output based on the tests and finally relative filtering. When we said that we kind of filter into the test set up to filter a specific set of tests, we can filter from the command line, and we saw how to add a new task. And inside that task, just filter a specific set of tests we talked with the differences with our task between Jane and four and Jaded five, but we have to call use J unit platform. So now that we've seen that, let's move on and talk about the greater rapper and see where the wrapper is used. Both for idee ease on if we want use external build servers.

## **Understanding the Gradle Wrapper and How to Use It**

### **Introduction**

Hi. Welcome back to the great. I'll build tools. Fundamentals. Course, This is the what is the greater wrapper on how to use it? Module on. My name is Kevin Jones. So in this module, we'll take a look at the greater wrap it and explain what it is, and they will show how to add the greater wrapper to a project. And then we show one of the uses of this will show how to use the greater rapper with a build server in particular. In this case, we'll use Team City, which is a build server provided by jetbrains. Will show how to build a project on Team City using the Griddle rapper. So first of all, then what is the greater rapper? The idea behind the greater rapper is that we can define a specific version of Grade A ll two years with the project. So wrath and using whichever version of grade A ll you have installed on your machine or other developers machines, we can say for this project I want to use this version of greater. The idea of this is that it gives us consistent builds. So we always know which version of great. Are we going to use whether it's 6.22 point six or something completely different? When you do this on Windows, we got a batch file called Cradle W dot bat known UNIX systems. You get a shell script called Cradle W When you run grade a ll from the command line, it will detect if the wrapper is in store for this particular project. You just look to see if Griddle w exists. And if it does, it runs the rapper version of Grade A ll rather than the installed version of Cradle. Okay, so let's see how we do this. Let's go add the rapper to one of our projects.

### **Adding the Wrapper**

So here we have a version of my security tourist project, and this is the one with the J Unit five testing it. I notice that currently this is Justice Standard Project. There's no rapper installed here. Notices. Well, this project is checking to get well. We'll see what that's important in a later clip. So if I go to the terminal window from here on grade a rapper and go back and look at the file system, we now see that we have a greater folder, which contains a rapper folder, which contains a rapid jar far on the properties file. I'm looking at properties fall in the moment. We also have these two script falls so great a W in greater W dot bats. And if I look at this property's file, we see in here that it points at a distribution of cradle appointed Gretel 6.3. This is the version of Greater We're running from the command line. Then if I go back to my terminal window on run griddle build, say yes, there's the plane consul hit notice, it says executing greater W instead of grade A ll, then the first thing it does is actually downloads the wrapper, and then it runs the build. It stores the wrapper in your dot Grader folder in your home folder. It's the next time I do a build, Gainous says. It's using great L W. Rather than cradle, but it doesn't download the rap of this time. It's already being download it so you can configure the wrapper from the command line with a couple of flags. So as well as running just a plain greater rapper, I can specify aversion to be counting on version 6.3. Let's say 6.2 my cancer specify a distribution type. So by default, when we download the wrapper, it download what's known as the being distribution. So I can specify the all distribution here. And if I look on the false system and take a look at my home folder dot Cradle and Rapper Fists Legacy. Currently I have great. All 63 been downloaded. That was a default. If I go back in, run this command. If I look in the properties fall now, we'll see. It's pointing a grade A ll 62 rather than 63 and it's pointing it all rather than bin and then back in the terminal window. If I do a build against time loading the distribution this time, it's downloading 6.2. And then if I look back on the file system, we'll see we have the great all six to all distribution downloaded just when we asked for.

### **Using a Build Server**

so there are a couple of reasons you're gonna want to use the rapper. One is if you're using an I. D. So, for example, if youse intelligent idea when you load a graded based project, it will use the greater wrapper for you. If we don't have one installed, it will install one, and it might install the wrong one. It's worth using the wrapper for that. But the other place you don't use Rapper is when you're doing builds, mrs. For the same reason you want to run the build using a known version of Cradle. And it could be that the build, severe running the build on doesn't have greatly installed, in which case using the wrapper that you have a version of Grade A ll that this Build Silver can use. So I'm talking about using a build server. But why do we use build service? So we tend to use Build servers to do what's known as continuous integration. So whenever we do a change to our code, we checked the code in on that code is built continually. When it's built, we run the build, we run the tests. We know that everybody has the latest version of the code that compiles on runs the test correctly. Not everybody doesn't build on every check in, but you will do nightly builds. You have the latest up to date version of your code than every night. And even if you don't do nightly builds, it's very good to have somewhere where you can do a clean build so we don't have. This works on my machine issue where the build runs fine on your machine, but somebody else checked out your code, and it doesn't run there. If we know the Builds server runs it correctly. At least we know the build is working and we can put a size any environmental issues we might have when doing the build. There's a build server we're going to use Team City. This is a build so provided by jetbrains the same people that give us intel. Ajay. It's available like Jeff brings dot com slash Team City, and you can use this for free on small projects. So what we're gonna do is to add a build Team City on that build is going to use the wrapper. So this is my team city server only Sisters. The Web page for the team City server. Come on here. We can see if a build set up this build is for a project called Cradle Fundamentals. I fight this project is the security tools project the J unit tests in what I'd like to do here is to set up a new build to show how easy there's a certain build using the greater wrapper on Team City. So to credit build, we need access to the sources. As I mentioned in the previous clip, this code was checked in to get within Team City. I've already added what's known as A. B. C s route to give me access to that repositories. And if I go to the administration for this project, look att, BCS roots. My Visa's roots called greater fundamentals. If I look at that and this is the route, and here we can see the u. R L's but might get repositories so it's get up or calm. Kevin are Joe's greater Fundamentals docket when we use that to create the built. So if I go back to the good adman window here, look at the little configurations I can create a new build configuration. We'll give this a name. Let's call this C. I built continuous integration, so I click on create to create that attached my source control roots, which is greater fundamentals. I noticed it says auto detecting build steps. So what team said he will do is look at the depository notice that we have great Earl installed and use that information to say, Do you want these build steps within the project? So we have a duplicate here, so I'm gonna pick one. You say grade A ll and you select it, and I just shows us we can now run the built. So if I click on run, take a look at the build lock while this is building, we do a check out of the source code. Let me see the tasks Running's were on the clean tastic compiled Java task and so on and so forth. The tests were all running and then the builders finished. If I go to the overview page, we can see that the build is completed on all 28 test of past

### **Extending the Build**

It's another the Builds completed. Let's just take a look and see what actually happened. If I go to the administration for this belt, looking great of fundamentals and see I built and look at the biddle steps, there's one build step. We can take a look at that. So if I edit this so we're using the teams. That is great All runner in here. This tells me the tasks that are gonna run. So it runs the clean task followed by the middle task. And then down here, it tells me that it's using the greater rapper to do the built I notice Hit that I'm using this. Build a cradle for Katie s faults. I'm running the cotton in script version of the build here. We could as easily run the groovy version of the builds. That wouldn't matter. In this case, let me do just one more thing here. This is supposed to be a continuous integration pills on. See, I builds run when we do a check in. So the force teams that you to do the build on shake it, I can add a trigger. We got our new trigger, the trigger. We want to add is a BCS trigger. Click on Safe triggers a build after the VCs check in is detected. Let me go back to the Home Directory for the project the C. I built that if I go back to my eternal window, we have changes here, but I've been playing with this so I can do it. Get ad toe at everything I can do it, get, commit and say updated rapper and then I get Push What has pushed? If I go back to Team City, we wait a while. What happens? The trigger kicks off. Detect that the sources have changed and I would read runs the build of their resources. Locally, there's only one step, which is the greater build. Step runs that step. Everything succeeds. So based on a check in, we will rerun. The build build runs to completion when we know we have a build that works with test the past. So if we can see here, it's relatively straightforward. To use grade. Aled to set would build on a build server in this case Team City

### **Review**

So in this module we've looked out how to add a greater wrapper to a project. Let me do that by simply running griddle space wrapper on using the built in rapid task. We've seen that configures the project, the properties far the points at a rapid distribution. And when you run the project, it downloads the rapper to your local file system, then uses that when it builds the projects. Don't lose the wrapper once there's no it'd download in multiple times as its cash locally on the file system, we said the reasons for doing this is that we now get consistent builds. We know exactly which version of great are we going to use to build this project. We then looked at how to use the rap with the build server. We set up a build on Team City. We set up a trigger to trigger that build. When we do this a game, we want a clean environment in which to do our belts. We've reached the end of the course. So what if we covered? We looked at how to install Grable on the various ways we can do that from things like sdk man installing by hand to using things like home brew on the Mac wilted writing tasks using simple tasks, then grade all so how we could set up a task dependencies building Java projects and, indeed, Katelyn projects with various Java plug ins such as Java and the application on the Java library pregnancy. We don't have to build multi project builds, so having a top level project and sub projects beneath that and now we can apply configuration of the top level or within the sub projects. We're testing both with Jay Unit 405 and finally we've looked at how to use the wrapper now is that we build server and use the greater rapid to run builds on the build server. So that's it for me. I want to say thank you. I hope you've enjoyed the course.