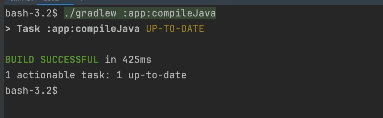
<https://dpeuniversity.gradle.com/courses/012de84f-fcd3-45d4-9c4c-284382eb3f3f/activities/6f37fa1e-578e-4493-8a06-dc6fcef84f43>

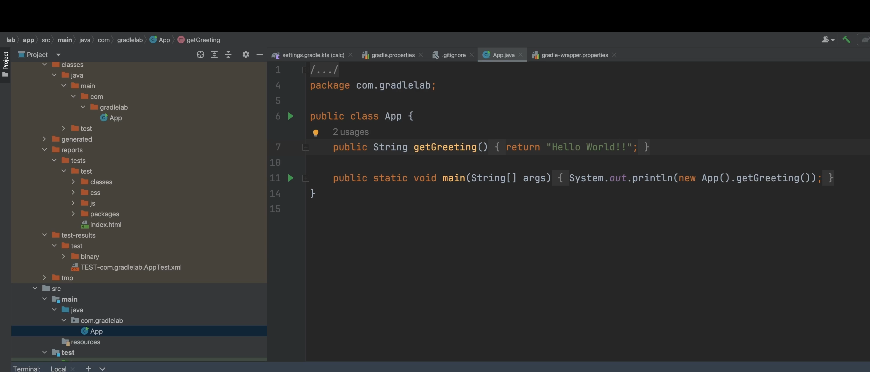
Configuring task properties – 5:44 minute

Learn how to configure tasks to customize their behavior

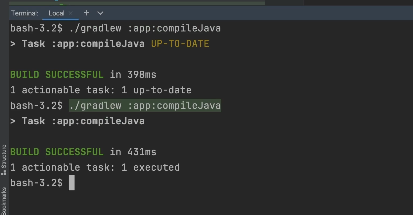
Now let’s explore the concept of input a little further. As we mentioned, inputs can be filed such as source code. If we run the compileJava task on the app subproject,



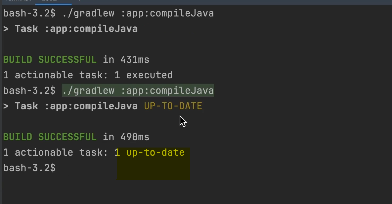
we can see the task is up to date, and we can see the output in the build directory. Now let’s change an input by editing a source file.



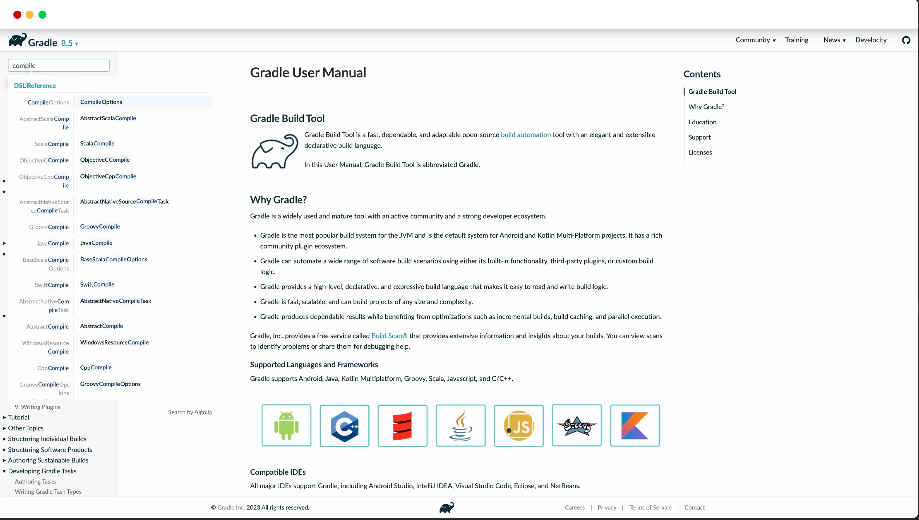
We add an extra exclamation mark to the Hello World String!! If we run the task again,



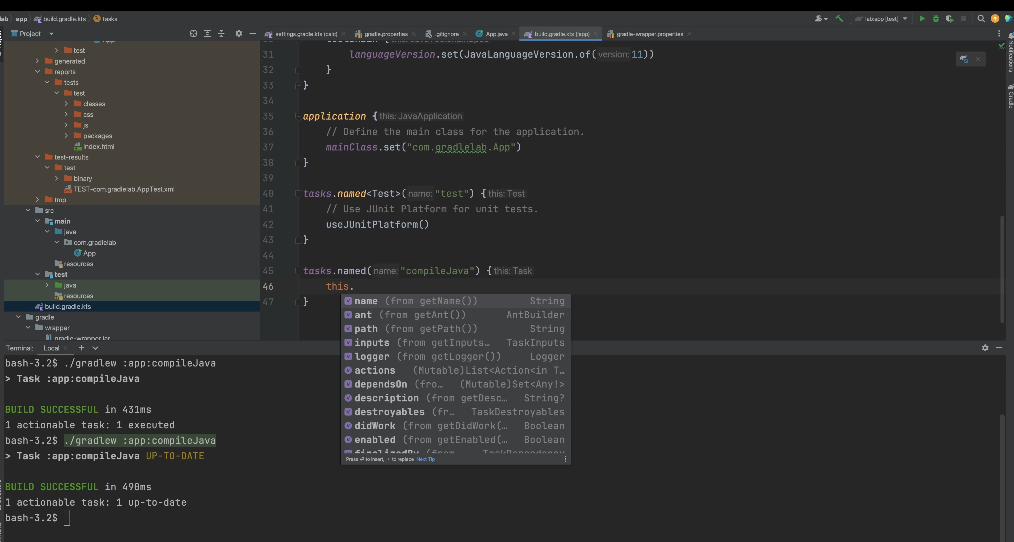
we can see it’s no longer up to date, because we changed change around the input a source file. If we run the task again,



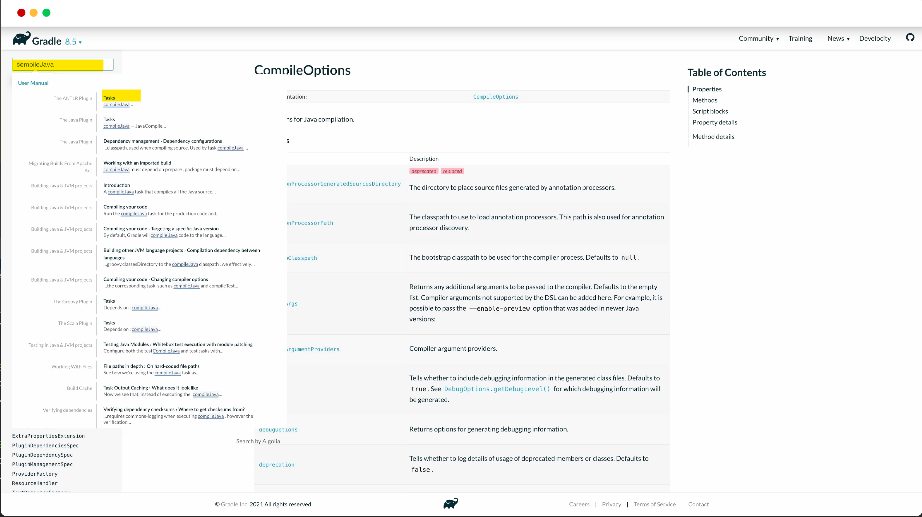
the task is once again up to date because none of the inputs changed. We set said that inputs can be more than just files, they can also be configuration options. Let’s see an example of this. Let’s change the compiler options for the compile Java task.



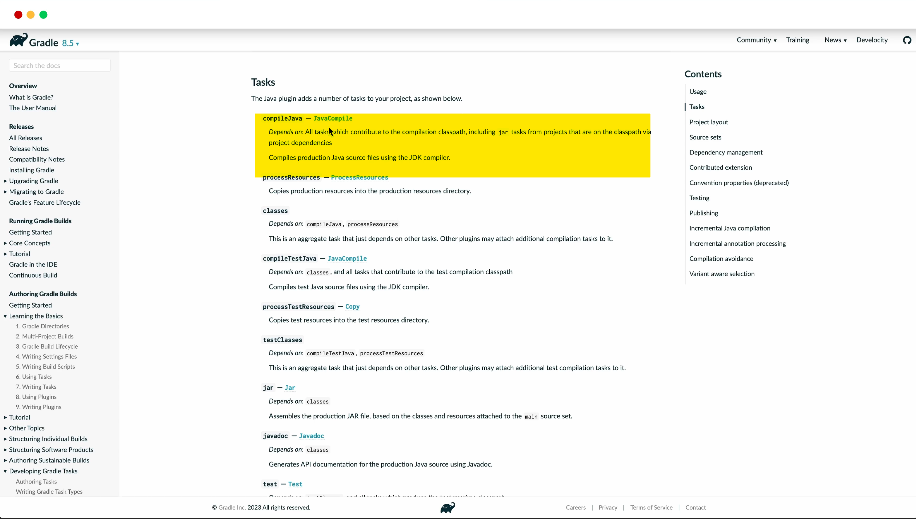
In the docs, we can search for compile options. We can see there’s configuration for whether to include debug information in the compile bytecode. By default, the compile Java task will include debug information in the compiled bytecode, which allows you to debug your application at runtime. Let’s change the configuration so debug information is not included in the compiled bytecode. We can open the builds file and access the configuration for the task by using the tasks variable and getting the task by name.



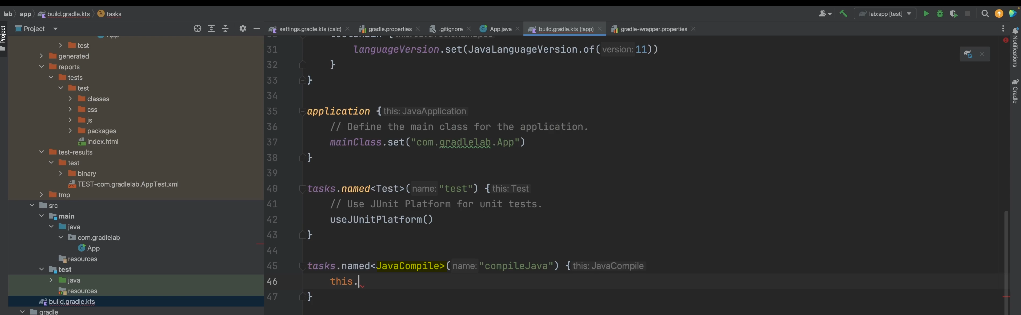
When you make changes to build files, some editors will have an icon pop up. The editor recognizes you’re changing Gradle build configuration, and when you’re done making the updates, you can click on the icon and the editor will re evaluate the files and update its understanding of the build configuration. If we type in the word **this**, we can see all the configuration options available.



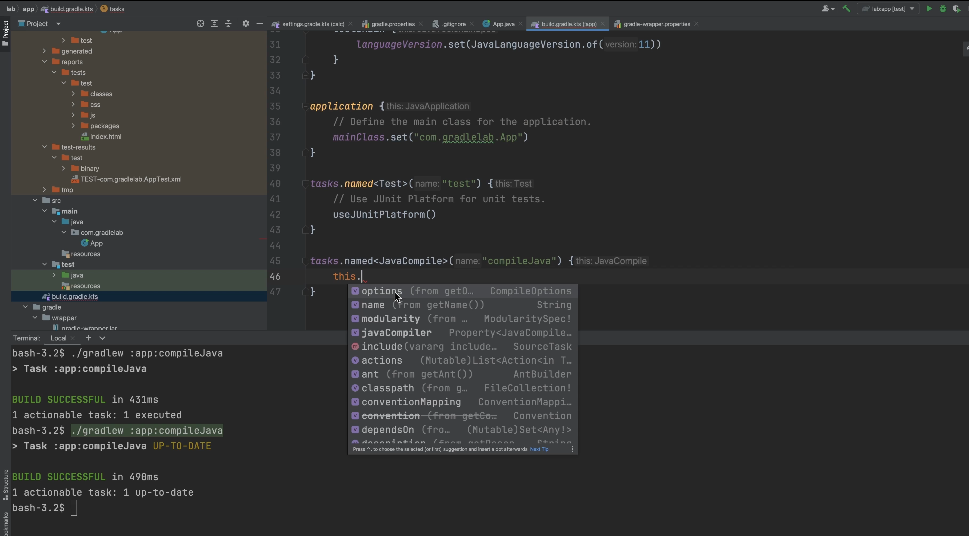
In the docs, if we look at the compileJava task, we can see subtype Java compile,



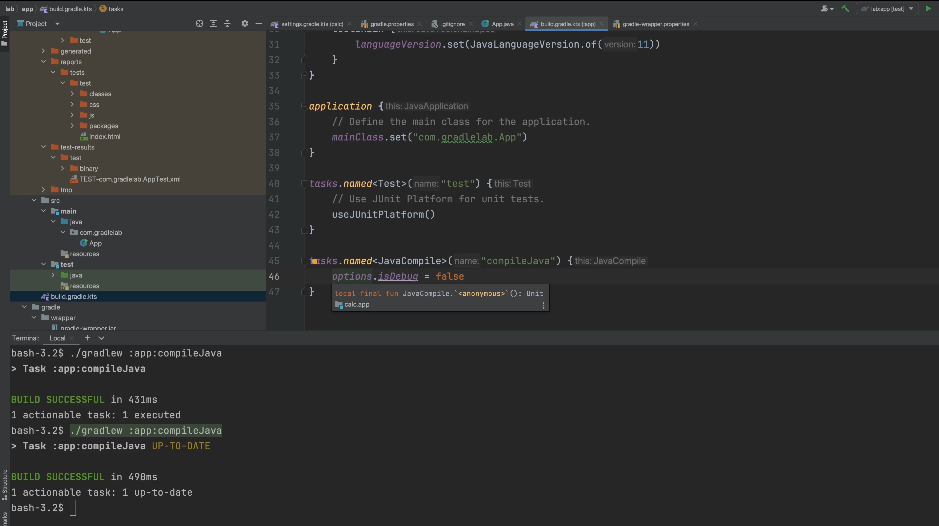
and if we click through, we can see to access the compiler options, we can use the options variable. If we go back to our build file and look at the available options, we do not see the options variable, even though we have access to configuration for the compile Java task, it is being treated as a default task type, not as a Java compile task. We need to tell Gradle build tool that it’s a Java compile task type. We can do this as follows.



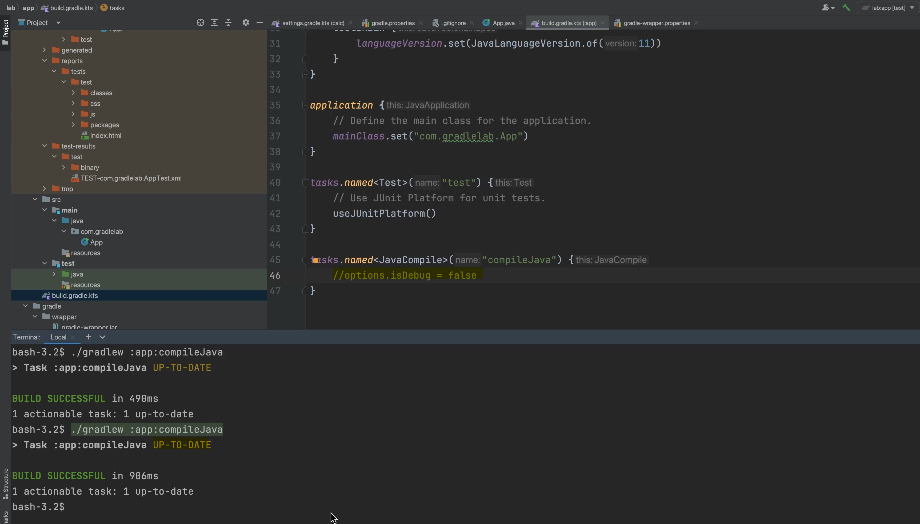
Now, when we look at the available options,



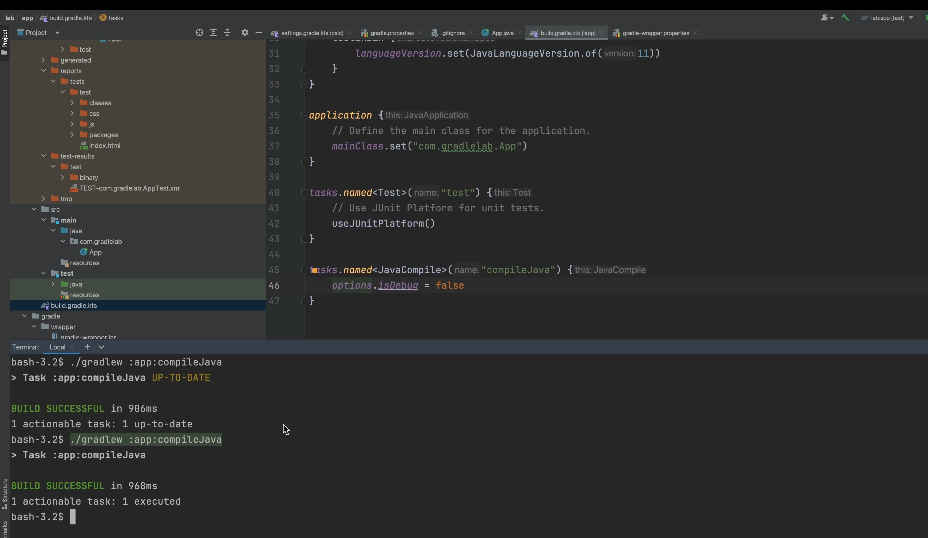
we can see the options variable, and in there we can see a debug flag, which we can set to false.



Just to confirm the task is you still up to date before making this change, let’s comment out the configuration and run the task.



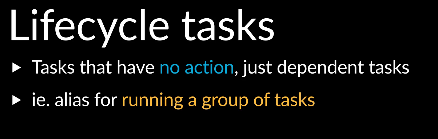
We can see it’s still up to date. Now, let’s uncomment the configuration and run the task.



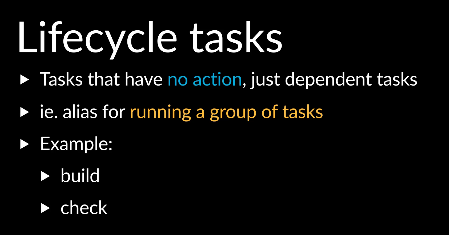
We can see there was no outcome label, meaning the task action executed successfully. If you think about it, changing the compiler option will result in different bytecode being produced without the debug information, therefore, the compiler option is an input that the compile Java task needs to consider and it does. When it changed, it recognized one of the inputs changed and executed the action. We discussed how tasks have inputs, they may execute an action and produce outputs.



However, there are some tasks that do not have actions and do not produce outputs, when you run them, all that happens is that a number of prerequisites tasks run, these tasks are like an alias or a shortcut to run a group of other tasks,



the only thing these tasks have configured is the prerequisite has tasks to run, these are known as lifecycle tasks, examples include the build and check tasks, which do not have actions, running them causes a number of other tasks to run.



The build lifecycle task runs other tasks that compile and produce a package such as a JAR file. The check lifecycle task runs tasks that validate your code, such as executing tests, code style and verifying code average. As a developer, the check lifecycle task is one you may often run.

Completed…