Setup VSCode for Kotlin Development

This tutorial will cover how you can set up your Visual Studio Code Editor for Kotlin projects. By now you should have installed your version of vscode. If this is not the case you can look at our post on how to install it on Windows, Linux, or Mac.

Visual Studio Code is a perfect choice to create Kotlin projects. It offers IDE functionality such as AutoComplete, Text/syntax highlighting, text formatting, and linting. Not only your code will well be displayed, but vscode also gives you the means to refactor and control your project. This is possible with the rich ecosystem of extensions and plugins for Kotlin.

The first part of this tutorial will focus on how to set up the extensions and plugins. Once this is done we will show you how to create a Kotlin project in vscode or how to import an existing one. In the end, we will focus on the development, including building and debugging a Kotlin application and how to run tests.

As a new part of this tutorial, we will discuss how to add **Gradle** support to your Kotlin project in Visual Studio Code.

Content [Hide]

Operating System

The development of Kotlin projects is not really dependent on your operating system. All you need is to have Visual Studio code installed (see our post). Also depending on the compile target, you may need the Java Virtual Machine or Java Runtime Environment.

Besides of that the development is platform-independent and can be done in all major OS, including Linux (e.g. Ubuntu), Windows, and Mac.

Use Kotlin with vscode in Windows

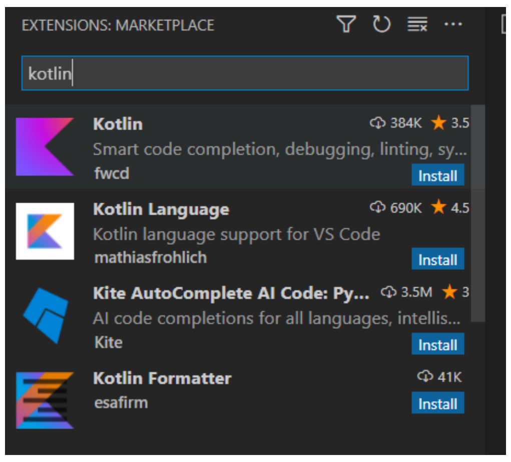
Our tutorial mainly focuses on the windows operating system. We have manually installed the Java Runtime Environment and the Kotlin Compiler. Also we use the standard version of Visual Studio Code.

Command line compiler – kotlinc

It is mandatory that you have a valid compiler available in order to build Kotlin projects. This step heavily depends on how you have installed Kotlin and/or Java on your machine. In case of doubt we recommend that you check out the official page from Jetbrains.

Extensions and Plugins

You can work on Kotlin projects without Visual Studio Code Extensions and Plugins. However, we recommend that you install them, as they give you a better coding experience.


EXTENSIONS: MARKETPLACE
kotlin
Kotlin
384K ★ 3.5
Smart code completion, debugging, linting, sy...
fwcd
Install
Kotlin Language
690K ★ 4.5
Kotlin language support for VS Code
mathiasfrohlich
Install
Kite AutoComplete AI Code: Py...
3.5M ★ 3
AI code completions for all languages, intellis...
Kite
Install
Kotlin Formatter
41K
esafirm
Install

On the right side of VSCode, you will find the tab for the marketplace. In there you can search all kind plugins which enhance the default behavior of the editor.

In our case we will install the extension "[Kotlin Language]".

Additionally we install the extension *vscode-runner*. This plugin gives the functionality to run simple programs directly from the editor.

Create Projects

We are now ready to create our first Kotlin project in Visual Studio Code.

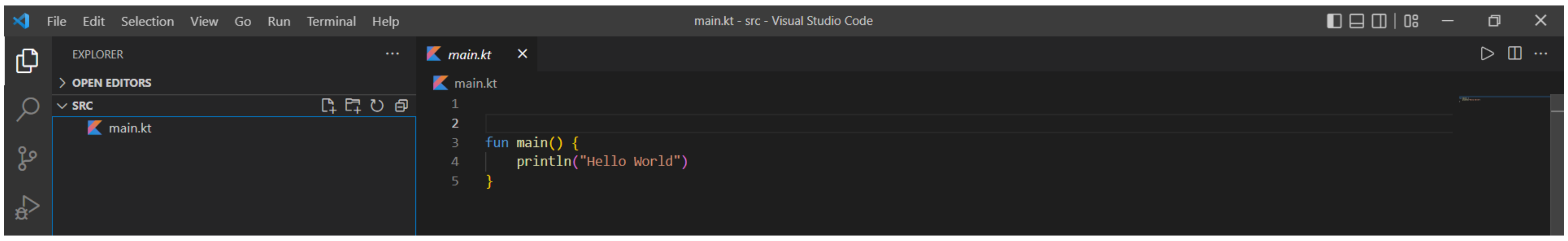
Hello World

First we create a main.kt file with a simple hello-world main function.

Hello World

```
fun main() {
    println("Hello World")
}
```

In the image below you see that the file system is fairly simple. The editor, icons and texts are formatted and highlighted thanks to the first extension which we have installed. On the top right, you can find a small arrow icon. This icon comes from the second plugin. When clicking on the icon it will compile the main.kt file and run the application.


File Edit Selection View Go Run Terminal Help
main.kt - src - Visual Studio Code
EXPLORER
OPEN EDITORS
SRC
main.kt
main.kt
main.kt

```

fun main() {
    println("Hello World")
}
```

If everything went well, which means that the compiler was found and no compiler errors occurred, the editor will print "Hello World" in the terminal.

Import Existing Projects

As it was shown in the hello-world project, you can create or import any project.

Development

Check: Part_1_DevSetup

Build

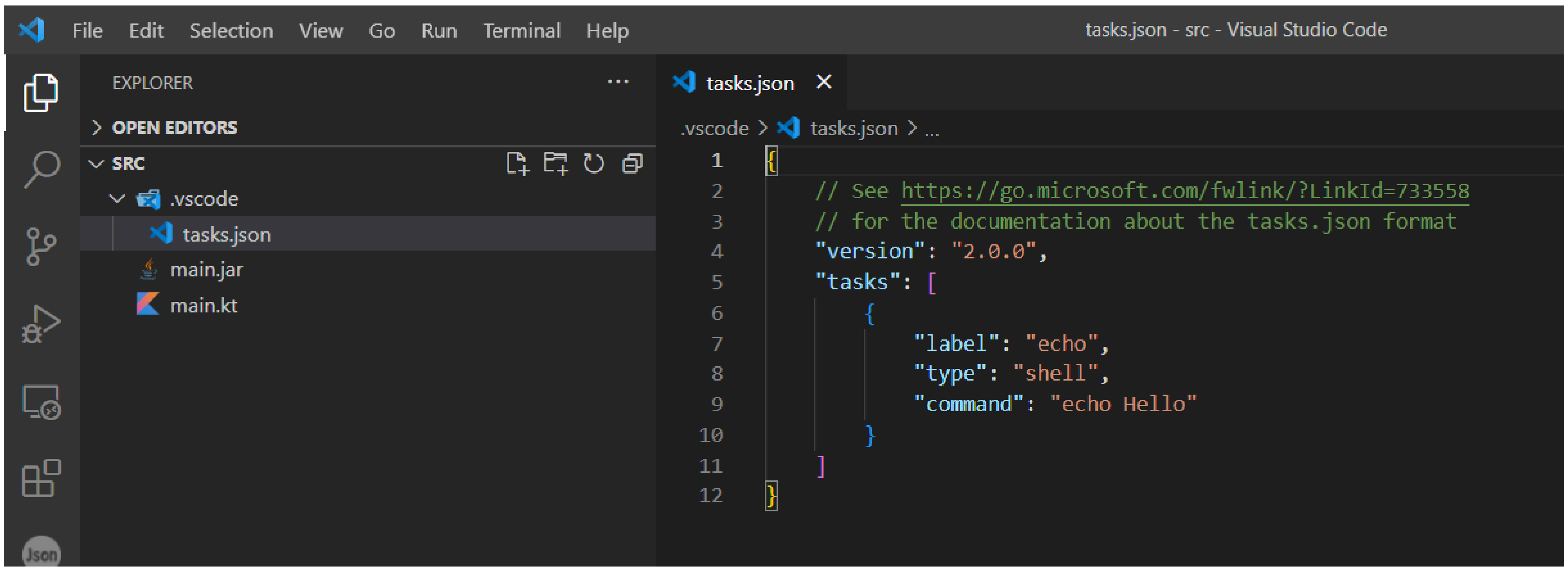
In order to build a Kotlin project, you will need of course the Kotlin compiler. You can download it manually from the Jetbrains page, which we have pinned above.

Now you have several options. First, you can use the code-runner, which we have installed as a Visual Studio Code extension. This code runner will automatically invoke the compile command and run the application afterward.

The second option, which is better in line with the editor's workflow, is to create a task.json file. To create a task.json file, do the following:

- Press F1 Press: shift + ctrl + p
- Write *Configure Build task*
- Select *Configure Default Build Task*
- Select *Create task.json file from Template*
- Select *Others*

Your filesystem should now look like in the following image.


File Edit Selection View Go Run Terminal Help
tasks.json - src - Visual Studio Code
EXPLORER
OPEN EDITORS
SRC
.vscode
tasks.json
main.jar
main.kt
.vscode > tasks.json > ...

```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            "label": "echo",
            "type": "shell",
            "command": "echo Hello"
        }
    ]
}
```

You can now change the specific task to create a default build.

```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            "label": "echo",
            "type": "shell",
            "command": "kotlinc main.kt -include-runtime -d main.jar",
            "group": {
                "kind": "build",
                "isDefault": true
            }, "options": {  "cwd": "${workspaceFolder}"  }        #  add the path to the working diectory
        }
    ]
}
```

If you now run “Ctrl + Shift + B” it will execute the build.

Run

To run your application you have different options as well. As in the previous section you can use the code runner, which we have installed as a plugin.

The second option is to configure Visual Studio Code to run your application from the command line. Since the build will create an executable jar file you can invoke it with the Java Runtime Environment.

Debug

You can attach a Java debugger to you application. This is possible as it was compiled towards the Java Runtime Environment

Gradle Project for Kotlin in VSCode

Check: Part_2_GradleSetup

In this section we will discuss how to setup a gradle project, which targets Kotlin and/or Java in Visual Studio Code. We will guide you step by step how to setup all files and what you need to install.

First of all you should install the following extensions.

Extensions

- Extension Pack for Java
- Gradle for Java

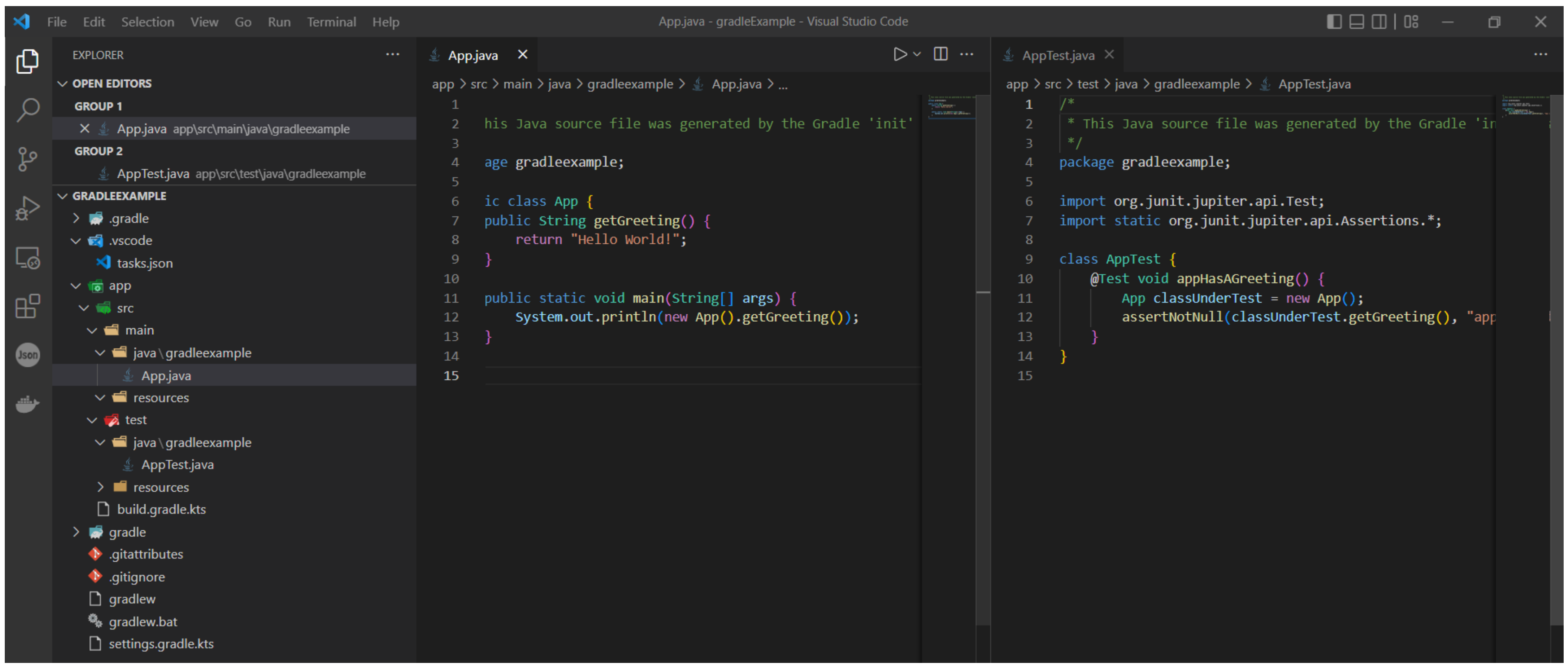
Both extensions are necessary to create and run a gradle project. Furthermore you should have installed the Java Run Time and also the Kotlin standalone compiler.

Create new gradle project

Thanks to the extensions you can create a brand new gradle project by typing the following:

- Ctrl + Shift + P
- Type Gradle
- Select “Create a gradle java project”
- Select Folder and project name
- Select Kotlin as DSL language

The extensions will create a new project as shown in the image below. Note that the .vscode folder might be missing at this point. If it is missing don’t worry about it. As you can see, the created project has only Java code. Before we are going to change / add Kotlin code, we will setup a tasks.json file to automatically build and run the project. This will help us to see if everything is setup properly.


File Edit Selection View Go Run Terminal Help
App.java - gradleExample - Visual Studio Code
EXPLORER
OPEN EDITORS
GROUP 1
App.java app\src\main\java\gradleexample
GROUP 2
AppTest.java app\src\test\java\gradleexample
GRADLEEXAMPLE
.gradle
.vscode
tasks.json
app
src
main
java \ gradleexample
App.java
resources
test
java \ gradleexample
AppTest.java
resources
build.gradle.kts
gradle
.gitattributes
.gitignore
gradlew
gradlew.bat
settings.gradle.kts
App.java
app > src > main > java > gradleexample > App.java > ...
AppTest.java
app > src > test > java > gradleexample > AppTest.java

```java

his Java source file was generated by the Gradle 'init'

age gradleexample;

ic class App {
public String getGreeting() {
    return "Hello World!";
}

public static void main(String[] args) {
    System.out.println(new App().getGreeting());
}

```

```java
/*
 * This Java source file was generated by the Gradle 'in
 */
package gradleexample;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class AppTest {
    @Test void appHasAGreeting() {
        App classUnderTest = new App();
        assertNotNull(classUnderTest.getGreeting(), "app
    }
}

```

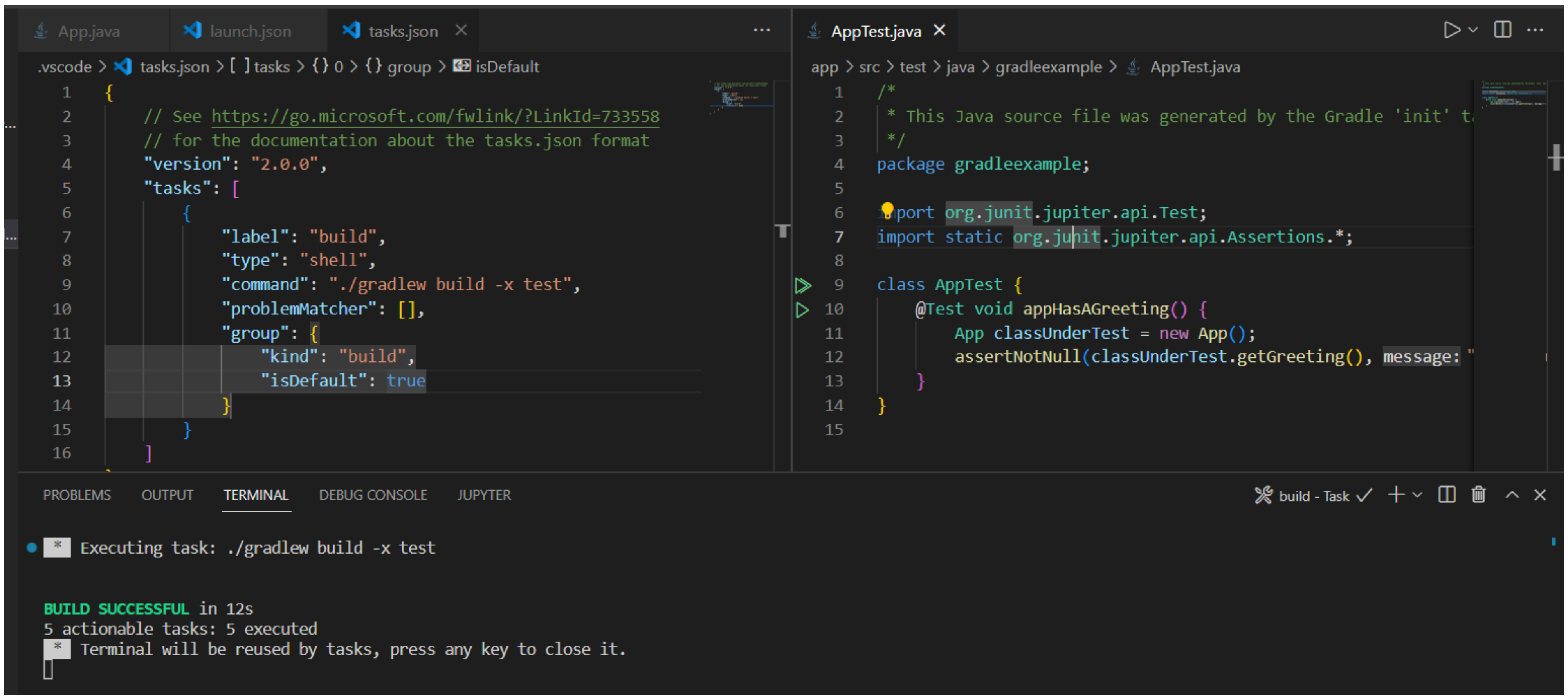
Create task.json for gradle

Setup environment

- Ctrl + Shift + P
- Type Tasks, Select Any (e.g. Configure Default Build Task)
- Create tasks.json from template (type others)

In the newly created task.json file we recommend to put at least the following tasks. The first one is the default build task which will build your application. The second is the run task, which will eventually open and run your application.

```
        {
            "label": "build",
            "type": "shell",
            "command": "./gradlew build -x test",
            "problemMatcher": [],
            "group": {
                "kind": "build",
                "isDefault": true
            }
        },
        {
            "label": "run",
            "type": "shell",
            "command": "./gradlew run",
            "problemMatcher": []
        },
```


App.java
launch.json
tasks.json
AppTest.java
.vscode > tasks.json > [ ] tasks > { } 0 > { } group > isDefault
app > src > test > java > gradleexample > AppTest.java
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
JUPYTER
build - Task

```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            "label": "build",
            "type": "shell",
            "command": "./gradlew build -x test",
            "problemMatcher": [],
            "group": {
                "kind": "build",
                "isDefault": true
            }
        }
    ]
```

```
/*
 * This Java source file was generated by the Gradle 'init' t
 */
package gradleexample;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class AppTest {
    @Test void appHasAGreeting() {
        App classUnderTest = new App();
        assertNotNull(classUnderTest.getGreeting(), message: "
    }
}
```

```
 *  Executing task: ./gradlew build -x test

BUILD SUCCESSFUL in 12s
5 actionable tasks: 5 executed
 *  Terminal will be reused by tasks, press any key to close it.
```

The build task can be triggered by running the shortcut "Ctrl + Shift + B". Both tasks can also be run by typing the following.

- Ctrl + Shift + P
- Tasks: Run Task
- select "run" or "build"

Once you have build and run the application you should have a similar out as:

```
> Task :app:run
Hello World!

BUILD SUCCESSFUL in 1s
2 actionable tasks: 1 executed, 1 up-to-date
 *  Terminal will be reused by tasks, press any key to close it.
```

Adding Kotlin code to gradle project

In the next step we want to add Kotlin Code to our Gradle Project in Vscode. To have that working correctly we have to adapt the *build.kotlin.kts* file. As indicated in the official Kotlin documentation (https://kotlinlang.org/docs/gradle.html) we must add the plugin to gradle.

Note: plugins is found in app/build.gradle.kts;   And to check for the kotlin version do: $ kotlin -version;

```
plugins {
    // Apply the application plugin to add support for building a CLI application in Java.
    application
    kotlin("jvm") version "1.7.10"
}
```

We are targeting the the java virtual machine. The indicated version might change in future. So please make sure to check the current Kotlin and jvm version.

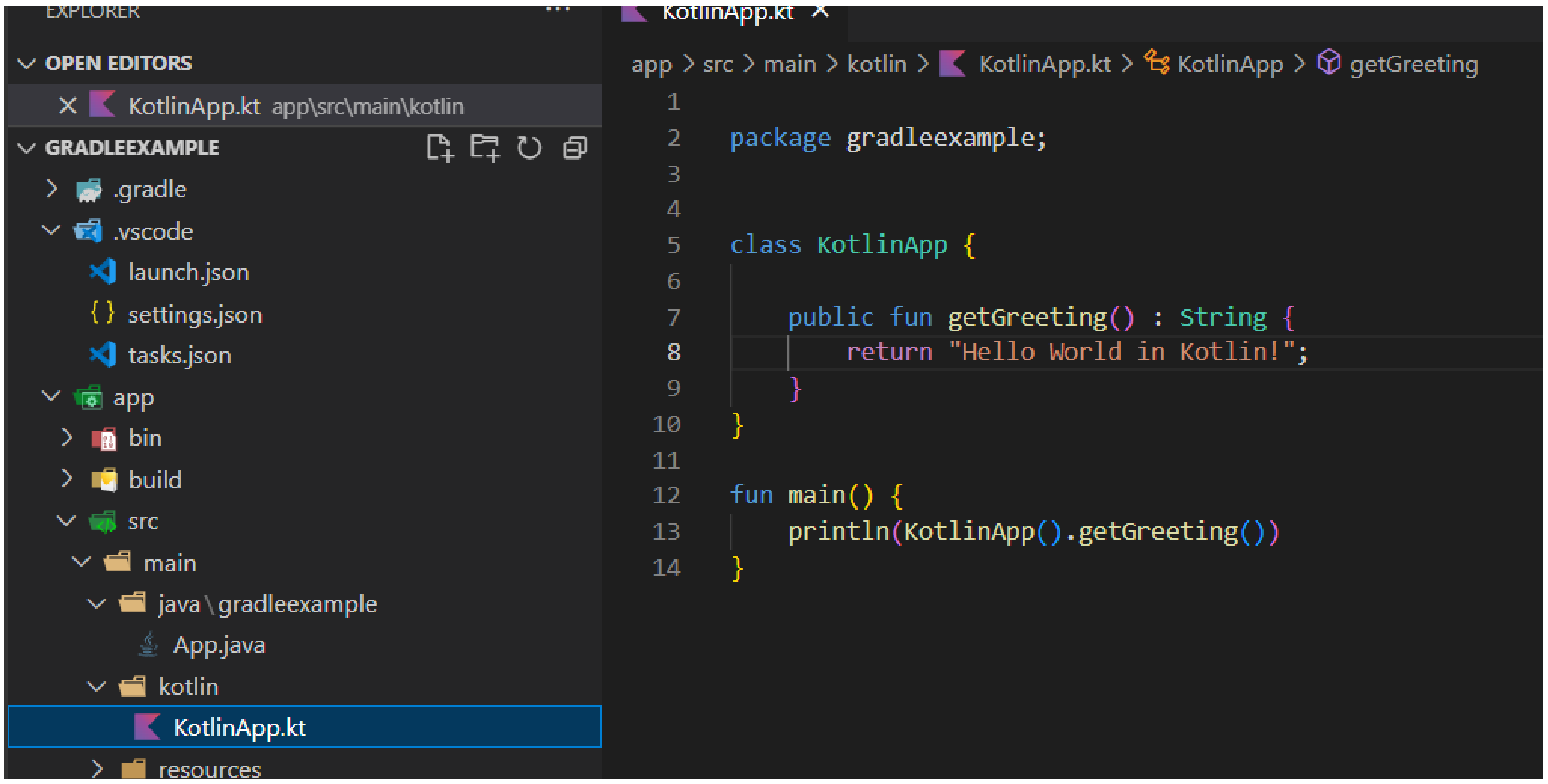
Project structure

By default you must follow the following project structure. This is important because otherwise gradle will have problems to correctly compile and match your classes and source files.

```
project
    - src
        - main (root)
            - kotlin
            - java
```

Add this directory and the kotlin files inside

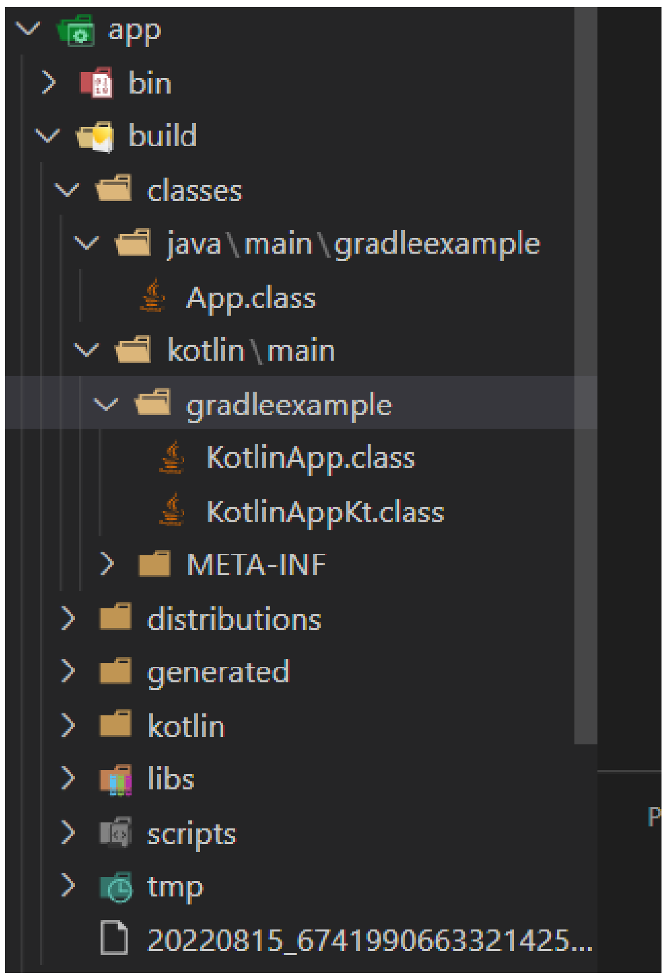
As an example we will add a new main class in the Kotlin section.



```
EXPLORER
OPEN EDITORS
  × KotlinApp.kt  app\src\main\kotlin
GRADLEEXAMPLE
  > .gradle
  v .vscode
      launch.json
      settings.json
      tasks.json
  v app
    > bin
    > build
    v src
      v main
        v java \ gradleexample
            App.java
        v kotlin
            KotlinApp.kt
        > resources
```

```
KotlinApp.kt ×
app > src > main > kotlin > KotlinApp.kt > KotlinApp > getGreeting

package gradleexample;


class KotlinApp {

    public fun getGreeting() : String {
        return "Hello World in Kotlin!";
    }
}

fun main() {
    println(KotlinApp().getGreeting())
}
```

Once you have built the application you should check your build-folder to see if the Kotlin files were correctly parsed and the binary files were correctly compiled.



```
v app
  > bin
  v build
    v classes
      v java \ main \ gradleexample
          App.class
      v kotlin \ main
        v gradleexample
            KotlinApp.class
            KotlinAppKt.class
        > META-INF
    > distributions
    > generated
    > kotlin
    > libs
    > scripts
    > tmp
      20220815_6741990663321425...
```

In the build folder two sections are created; one for Java and one for Kotlin classes.

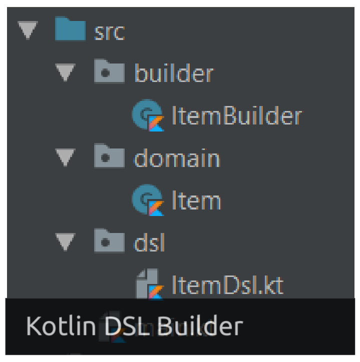
Kotlin gradle main class

With the new compiled Kotlin classes we can target it as the main entry point of the application. To do so, we must once again change the *build.gradle.kts* file.

```
application {
    // Define the main class for the application.
    mainClass.set("gradleexample.KotlinAppKt")
}
```

After this change you can build and run the application. The output should be the following:

```
> Task :app:run
Hello World in Kotlin!

BUILD SUCCESSFUL in 4s
3 actionable tasks: 1 executed, 2 up-to-date
 *  Terminal will be reused by tasks, press any key to close it.
```



Kotlin DSL Builder