Transpiling to JavaScript

We'll cover the following

- Compiling in the command line
- Running in Node.js
- Running in browsers

Front-end development is gaining momentum, and there's an increasing demand for full-stack developers. JavaScript has evolved significantly in recent years, but it's a dynamically typed language and also a weakly typed language——see Rediscovering JavaScript. In recent years, some programmers have gravitated toward statically typed languages, for example TypeScript, that will transpile to JavaScript. Statically typed languages can help find errors early, during compile time, and remove certain problems that may creep up during runtime if the code was written directly in JavaScript.

Languages like TypeScript offer static typing, but languages like Kotlin bring in that and a lot more. If you're going to choose static typing for front-end development, why stop at languages like TypeScript, when you could go all the way and benefit from the wealth of features of a powerful language like Kotlin? Kotlin makes it easier to create common code and to reuse code between platforms. You can also use Kotlin DSLs for HTML and CSS and thus use Kotlin for the entire front-end development.

Though the focus of this book isn't on creating front-end applications with Kotlin, let's explore this idea of Kotlin to JavaScript a bit here. If you're using IntelliJ IDEA, it'll extract the necessary Kotlin JavaScript runtime library for you automatically when you compile to JavaScript. If you're using other IDEs or a build tool, you have to do some extra steps. Here, you'll see how to use the command line to compile and run the generated JavaScript code both in Node and within a browser.

Compiling in the command line

We'll write a Kotlin source file, transpile it to JavaScript, and also extract the necessary Kotlin JavaScript runtime library files from the Kotlin distribution.

Let's start with a Kotlin file first.

```
package com.agiledeveloper.jsexample

fun greetNames(names: List<String>) = "hello ${names.joinToString(", ")}"

fun main() {
  println(greetNames(listOf("Jack", "Jill")))
}
```

greetnames.kts

There's no sign of JavaScript here. If you like, you may compile the file `greetnames.kt` using `kotlinc-jvm` and run it within the JVM. But here, we'll transpile that into JavaScript instead.

To transpile to JavaScript, we have to use `kotlinc-js` instead of `kotlinc-jvm`. The `-help` option will give you all the available command-line options. We'll use the `-output` option to specify the JavaScript file to place the transpiled JavaScript code in.

Also, we need to extract the Kotlin JavaScript runtime library from the file `kotlin-stdlib-js.jar`, which is present in the Kotlin distribution under the `lib` directory.

Here are the commands for those two steps:

```
kotlinc-js -output greet.js greetnames.kt
unzip $KOTLIN_PATH/lib/kotlin-stdlib-js.jar -d lib
```

For Windows, remember to use `%KOTLIN_PATH%` instead of `$KOTLIN_PATH`. Also, you may use `jar` with `xf` option to extract the files from `kotlin-stdlib-js.jar` or use any program that will allow you to unzip the contents of that jar file. If you're using `jar`, then create a `lib` directory, `cd` into that directory, and then use the `jar` command to extract into that directory.

After running the above two commands, in addition to `greetnames.kt`, you'll find the `greet.js` file and a `lib` directory. Within the lib directory, among other files, you'll find `kotlin.js` and `kotlin.meta.js`.

We're all set to run the transpiled JavaScript. You may run it within Node.js for server-side JavaScript or within a browser, like Chrome, Firefox, and so on, for

client-side JavaScript.

Running in `Node.js`

For this section, you'll need Node.js, or Node for short, installed on your system. If you don't already have it, download it from the Node website and verify that it's in your system path by running the command node --version on the command line.

The transpiled JavaScript code needs the content of the Kotlin JavaScript library that we extracted into the lib directory. Thus, we have to require that first. Then we can load and execute the transpiled file. To make these steps easy, let's create a JavaScript file:

```
// run-in-node.js
kotlin = require('./lib/kotlin');
require('./greet')
```

To run this code, use the command:

```
node run-in-node.js
```

Node will load the Kotlin JavaScript runtime file kotlin.js from within the lib directory, and also the transpiled file greet.js, and display the following output on the console:

```
hello Jack, Jill
```

The Kotlin-to-JavaScript transpiler converted the call to `println()` to `console.log();`, and thus the output appears on the console when JavaScript is executed.

Running in browsers

To compile and merge scripts, to eliminate unused code, and to generate smaller JavaScript files, we typically use tools like WebPack. It's common to use such tools when using Kotlin for the front end as well. Since we're creating a very small example and our focus isn't on the front-end development, we won't delve into that here. Let's focus on running the `greet.js` file that we compiled from Kotlin in a browser.

To see the file `greet.js` run within a browser, we will first need to create an `index.html` file:

```
<!DOCTYPE html>
<html>
  <body>
    <p>Please see the output in the browser console</p>
  </body>
  <script src="./lib/kotlin.js"></script>
  <script src="greet.js"></script>
</html>
```

index.html

In the HTML file, remember to load the `kotlin.js` script first, before loading the `greet.js` file.

Then open the file `index.html` within your favorite browser and take a peek at the browser's console. Here's an excerpt of the file being executed within Chrome.