```
module Chapter2 where

import Debug.Trace

main = trace "Hello, World!"
```

This small sample illustrates a few key ideas:

- Every file begins with a module header. A module name consists of one or more capitalized words separated by dots. In this case, only a single word is used, but `My.First.Module` would be an equally valid module name.
- Modules are imported using their full names, including dots to separate the parts of the module name. Here, we import the `Debug.Trace` module, which provides the `trace` function.
- The `main` program is defined as a function application. In PureScript, function application is indicated with whitespace separating the function name from its arguments.

Let's build and run this code. Invoke the following command:

```
$ psc src/Chapter2.purs
```

If everything worked, then you will see a relatively large amount of Javascript emitted onto the console. Instead, let's redirect the output to a file with the `--output` command line option:

```
$ psc src/Chapter2.purs --output dist/Main.js
```

You should now be able to run your code using NodeJS:

```
$ node dist/Main.js
```

If that worked, NodeJS should execute your code, and correctly print nothing to the console. The reason is that we have not told the PureScript compiler the name of our main module!

```
$ psc src/Chapter2.purs --output dist/Main.js --main=Chapter2
```

This time, if you run run your code, you should see the words "Hello, World!" printed to the console.

2.6 Removing Unused Code

If you open the `dist/Main.js` file in a text editor, you will see quite a large amount of JavaScript. The reason for this is that the compiler ships with a set of standard functions in a set of modules called the Prelude. The Prelude includes the `Debug.Trace` module that we are using to print to the console.

In fact, almost none of this generated code is being used, and we can remove the unused code with another compiler option: