Defining Domain Objects

We'll cover the following

- Fetching and storing data
- Making asynchronous calls

Fetching and storing data

Our app will make requests to the FAA web service to fetch the airport data. For this, we'll need a variation of the `Airport` class we created in Chapter 19, Unit Testing with Kotlin. In Android Studio, in the Project pane, right click the package name `com.agiledeveloper.airports` under `app/java` and select New, and Kotlin File/Class. Change the value for Kind to Class and key in the name "Airport" in the text box next to the Name label.

Edit the generated file to create the `Airport` class, like so:

```
package com.agiledeveloper.airports

import com.beust.klaxon.*

class Weather(@Json(name = "Temp") val temperature: Array<String>)

data class Airport(
  @Json(name = "IATA") val code: String,
  @Json(name = "Name") val name: String,
  @Json(name = "Delay") val delay: Boolean,
  @Json(name = "Weather") val weather: Weather = Weather(arrayOf(""))) {

  companion object {
    fun sort(airports: List<Airport>) : List<Airport> {
      return airports.sortedBy { airport -> airport.name }
    }

    fun getAirportData(code: String) =
      try {
        Klaxon().parse<Airport>(fetchData(code)) as Airport
      } catch(ex: Exception) {
        Airport(code, "Invalid Airport", false)
      }

    private fun fetchData(code: String) =
      java.net.URL("https://soa.smext.faa.gov/asws/api/airport/status/$code")
```

```
      .readText()
  }
}
```

Airport.kt

The `Weather` class is used to store the temperature at an airport. An instance of this class is used within the `Airport` class, which comes next. The `Airport` class isn't much different from the one we created in previous chapters. The `getAirportData()` function uses the Klaxon parser to create an `Airport` instance from the data received from the FAA web service. If there's an error, it returns an `Airport` instance with the name `Invalid Airport`.

Making asynchronous calls

Let's now create the `AirportStatus` file that will contain the function to make asynchronous calls to the `Airport`'s `getAirportData()` function. Create a file named `AirportStatus.kt`, following the steps that we used to create the `Airport` class. Change the contents of the generated file, like so:

```
package com.agiledeveloper.airports

import kotlinx.coroutines.*

suspend fun getAirportStatus(airportCodes: List<String>): List<Airport> =
  withContext(Dispatchers.IO) {
    val airports = airportCodes
      .map { code -> async { Airport.getAirportData(code) } }
      .map { response -> response.await() }

    Airport.sort(airports)
  }
```

AirportStatus.kt

The `getAirportStatus()` function is the same as the one we implemented in the previous chapters. It makes the calls to `getAirportData()` asynchronously, from a thread within the `Dispatchers.IO` thread pool, instead of from the Main thread. Thus the UI thread won't block when the code makes network calls. Since we already added the dependency to the `kotlinx.coroutines` library, we should have no trouble compiling this code.

We're done with coding the part that gets data from the web service. In the next lesson, we'll turn our attention to the UI.