Using Map

We'll cover the following

What is a `Map`?

A map keeps a collection of key-value pairs. Much like the way Kotlin provides a read-only immutable interface and a read-write mutable interface for the JDK `List`, the language also provides two interfaces for the JDK `Map`. All methods of the JDK `Map` interface are available through the mutable interface, and the read-only methods are available through the immutable interface.

Creating maps in Kotlin

You may use `mapOf()` to create a map and get a reference to the read-only interface `Map<K, V>`. Alternatively, use `mutableMapOf()` to get access to `MutableMap<K, V>`. Also, you may obtain a reference to the JDK `HashMap` using `hashMapOf()`, `LinkedHashMap` using `linkedMapOf()`, and `SortedMap` using `sortedMapOf()`.

Let's create an example using the immutable/read-only interface `Map<K, V>` and look at ways to access the elements. Here's a piece of code to create a map of site names and their corresponding URLs, where both keys and values are `Strings`:

```
val sites = mapOf("pragprog" to "https://www.pragprog.com",
  "agiledeveloper" to "https://agiledeveloper.com")

println(sites.size) //2
```

usingmap.kts

The key-value pairs are created using the `to()` extension function, available on any object in Kotlin, and `mapOf()`, which takes a vararg of `Pair<K, V>`. The `size` property will tell you the number of entries in the map.

Accessing elements of a `Map`

You may iterate over all the keys in the map using the keys property or all the values using the values property. You may also check if a particular key or value exists using the `containsKey()` and `containsValue()` methods, respectively.

Alternatively, you may use the `contains()` method or the `in` operator to check for a key's presence:

```
println(sites.containsKey("agiledeveloper")) //true
println(sites.containsValue("http://www.example.com")) //false
println(sites.contains("agiledeveloper")) //true
println("agiledeveloper" in sites) //true
```

usingmap.kts

To access the value for a key you may use the `get()` method, but there's a catch. The following won't work:

```
// usingmap.kts
val pragProgSite: String = sites.get("pragprog") //ERROR
```

It's not guaranteed that a key exists in the map, so there may not be a value for it. The `get()` method returns a nullable type—see Nullable References. Kotlin protects us at compile time and wants us to use a nullable reference type:

```
// usingmap.kts
val pragProgSite: String? = sites.get("pragprog")
```

The `get()` method is also used for the index operator `[]`, so instead of using `get()` we may use that operator:

```
// usingmap.kts
val pragProgSite2: String? = sites["pragprog"]
```

That's convenient, but we can avoid the nullable reference type by providing an alternative, default, value if the key doesn't exist:

```
// usingmap.kts
val agiledeveloper =
  sites.getOrDefault("agiledeveloper", "http://www.example.com")
```

If the key "agiledeveloper" doesn't exist in the map, then the value provided as the second argument will be returned. Otherwise, the actual value will be returned.

Manipulating `Map` elements

The `mapOf()` function provides a read-only reference, so we can't mutate the map. But we may create a new map with additional key-value pairs, like so:

```
// usingmap.kts
val sitesWithExample = sites + ("example" to "http://www.example.com")
```

Similarly, we can use the `-` operator to create a new map without a particular key that may be present in the original map, like this:

```
// usingmap.kts
val withoutAgileDeveloper = sites - "agiledeveloper"
```

To iterate over the entries in the map, you may use the `for` loop we saw in Chapter 5, External Iteration and Argument Matching. Let's use the `for` loop here:

```
for (entry in sites) {
    println("${entry.key} --- ${entry.value}")
}
```

usingmap.kts

The variable entry refers to a `Map` entry, and we can get the key and value from that object. But instead of that extra step, we may use the destructuring feature we saw in Destructuring, to extract the key and value, like so:

```
for ((key, value) in sites) {
```

```kotlin
for ((key, value) in sites) {
    println("$key --- $value")
}
```

usingmap.kts

As the iteration progresses, Kotlin automatically extracts the `key` and the `value` from each entry into the immutable variables named `key` and `value`, thanks to the feature of destructuring.

We used the imperative style to loop here. Later in the course, when we explore internal iterators, we'll revisit this to see how to iterate over the values using the functional style—in Chapter 11, Functional Programming with Lambdas.

The `Map` interface also has two special methods, `getValue()` and `setValue()`, which enable us to use maps as delegates—a powerful concept we'll dive into in Chapter 10, Extension Through Delegation.

1 Which of the following functions checks for the key in a `Map` collection?

2 How would you correctly initialize an immutable `Map`?

Retake Quiz

The next lesson concludes the discussion for this chapter.