Singleton Objects: Companion

In this lesson, you will be introduced to singleton objects and learn how to write a singleton companion object.

We'll cover the following

- Companion Objects

A singleton object is defined the same way a class is defined with the difference that instead of the `class` keyword, we use the `object` keyword.

Unlike a class, which can be instantiated multiple times, a singleton object only has a single instance. This is why we cannot use `new` to create an instance of a singleton object.

There are two types of singleton objects:

1. Companion Objects
2. Standalone Objects

Companion Objects

A singleton object which has the same name as a class is known as the *companion object* of that class and the class is known as the *companion class*. Companion objects and classes can access each other's *private* members.

Let's define a companion object of the `ChecksumAccumulator` class defined in the previous lesson.

This code requires the following environment variables to execute:

LANG	C.UTF-8

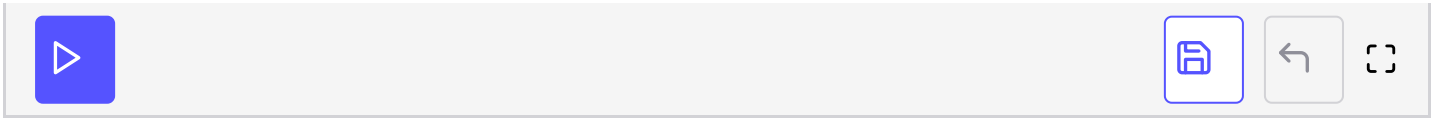
```
import scala.collection.mutable

class ChecksumAccumulator {
  private var sum = 0
  def add(b: Byte) = sum += b
  def checksum() = ~(sum & 0xFF) + 1
}
```

```
//companion object of ChecksumAccumulator

object ChecksumAccumulator {
  private val cache = mutable.Map.empty[String, Int]
  def calculate(s: String): Int =
    if (cache.contains(s))
      cache(s)
    else {
      val acc = new ChecksumAccumulator
      for (c <- s)
        acc.add(c.toByte)
      val cs = acc.checksum()
      cache += (s -> cs)
      cs
    }
}

// Driver Code
val result = ChecksumAccumulator.calculate("hello")
print(result)
```



The `ChecksumAccumulator` singleton object has one property, named `cache`. `cache` is a `Map` collection in which a single element consists of a key-value pair. The key in our case is the string whose checksum we want to calculate, and the value will be the corresponding checksum value. Initially, `cache` is empty.

The `ChecksumAccumulator` singleton object also has one method, named `calculate`, which takes a string and calculates a checksum for the characters in the string. It also has one private field, `cache`, that is a mutable map in which previously calculated checksums are stored. The strings are stored as keys and the checksum of those strings are stored as values.

The first line of the method, `if (cache.contains(s))`, checks to see if the passed string already exists as a key in `cache`. If so, it just returns the value mapped to that specific key, `cache(s)`. Otherwise, it executes the `else` expression.

The `else` expression starts off by creating an instance of the `ChecksumAccumulator` class. It then takes the passed string and runs `for` on it. `for`, one-by-one, stores each character into `c` and converts it into its binary form using `toByte` and adds the bytes together using the `add` method of the `ChecksumAccumulator` class. The `checksum` method is called on the final sum which calculates the checksum. Finally, `cache += (s -> cs)` stores the passed string and its corresponding checksum in `cache` and the checksum `cs` is returned.

The important thing to notice is here is how the singleton object is accessing the members of its companion class.

In the next lesson, we will define a standalone object which will help us