# Day 22: The flyweight design pattern

The flyweight pattern is meant to help minimize the memory footprint of your classes / objects.

In short, this is caching. When an object is instantiated that can be reused across multiple other objects, we can cache that object at instantiation time such that subsequent references simply retrieve the instantiated object. This is achieved via mutable Maps / ListBuffers and so on.

To be honest, I don't see a huge difference between this in just having lazy-initialized singleton objects, as long as your object domain is finite. If your object domain is arbitrary / determined at runtime, it could be good to store them in some sort of caching data structure like a Map.

Brief code example:

```
1  import scala.collection.mutable
2
3  class MySchema(schemaName: String, colNames: Seq[String]) {
4
5    println(s"Creating new schema $schemaName")
6
7    override def toString(): String = s"MySchema($schemaName)"
8
9    def printColNames: Unit = print(colNames.mkString("\n"))
10 }
11
12 object MySchema {
13   val schemaBuffer: mutable.Map[String, MySchema] = mutable.Map.empty[String, MySchema]
14
15   def apply(schemaName: String, colNames: Seq[String]): MySchema = {
16     schemaBuffer.getOrElseUpdate(schemaName, new MySchema(schemaName, colNames))
17   }
18 }
```

The clear advantage here is that you'll see upon repeated initialization calls to the same `schemaName` object, you'll only initialize once (`MySchema("a", Seq("one"))` will only print "Creating new schema a" once no matter how many times you call it). This is a nicer memory footprint. Always good to consider.