# Day 14: The singleton design pattern

Singletons are lazily-instantiated and global. They're normal JVM heap objects. They’re good for:

- Global state (thread-safe, but not immune to races)
- Static methods
- ADTs (see prior day)

The book claims that they're considered an anti-pattern, and I kinda agree — shoving a whole bunch of state / static methods in a singleton is extremely monolithic and unfriendly to design. A singleton is forever.

We use singletons for:

- Entry points
- Global config / functionality / static methods
  - Think SharedKafka, CassandraConfig, ParseUtils (stateless), look-up files
- Essentially "boxes" for ADTs, implicit classes, etc.
- `apply` methods - essentially treating a singleton as a factory for some related type; companion objects are a classic use case.

```scala
// a simple box for an ADT
// case objects are singletons with `unapply`
object MystiqueDataLayer {
  sealed trait MystiqueDataLayer
  case object Cassandra extends MystiqueDataLayer
  case object Cosmos extends MystiqueDataLayer
}
```

Remember: "Accessing the singletons in an application, no matter if it's Scala or not, also needs to be done in a thread-safe way or the singleton should take care of this internally."