Day 12: The static factory method

This is a simple, kinda limited pattern, but it's interesting. It allows for initialization via a singleton and pattern matching. I imagine it'd be good if you had radically different construction demands for some subtypes that can be differentiated "intentionally". Or maybe if you've got a clashing constructor on type, but clearly-delineated differences in use case within that type (like the String case below). Example:

```scala
trait Animal
class Bird extends Animal
class Mammal extends Animal
class Fish extends Animal

object Animal {
  def apply(animal: String): Animal = animal.toLowerCase match {
    case "bird" => new Bird
    case "mammal" => new Mammal
    case "fish" => new Fish
    case x: String => throw new RuntimeException(s"Unknown animal: $x")
  }
}
```

The clear problem here is the same with any pattern match on an ADT — any new type members need to be handled within the pattern match for completeness. Still, if the parent type is `sealed`, this could be useful.

This stands separate from the original factory method, as that was concerned with subtype polymorphism and how different classes can coordinate their subtype polymorphism. This is just straight-up pattern matching for constructors.