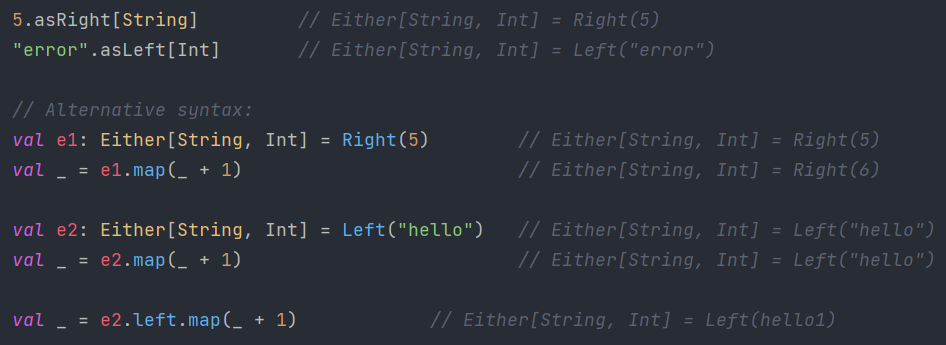
**EITHER:**

A way to handle errors, instead of throwing exceptions (which don’t carry as much information as we might want)

Either contains 2 types:  
- Right side (accessed by *asRight* method), which contains the value if operation/process successful  
- Left side (accessed by *asLeft* method), which contains the value if operation/process failed with an error

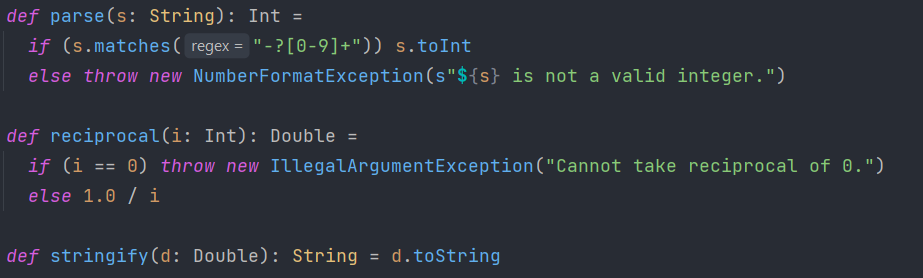
Either is right-biased, so when using map or flatMap on Either, it only transforms it if the value in the Either is on the right side (i.e., the value is a success).  
If the value is on the left, map & flatMap will just let the value ‘pass through’ without changing it. However, if *left* method, then this will return the value if it is a left-value, to then be mapped.

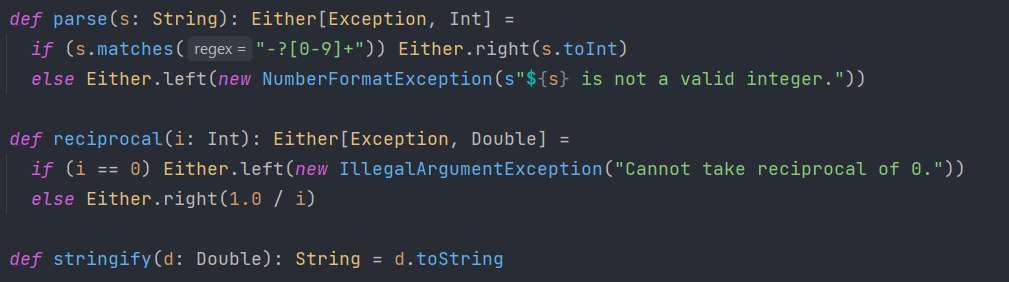


**Either vs Validated**  
In general, Validated is used to accumulate errors, while Either is used to short-circuit a computation upon the first error.

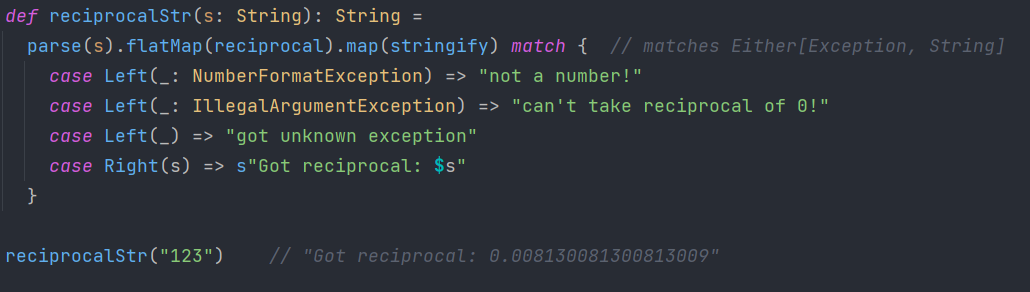
**EXAMPLE #1:**

Want to receive a String as input -> pass String to Integer -> take its reciprocal -> return reciprocal as String

Using throw exception style, we have…  


We modify this to use Either[Exception, String] instead…  


Then we chain these functions together and then match-case the resulting Either, for error-handling…



Modify further, by using custom exceptions. This allows us to store more information in the error message (e.g. what char has caused the issue) and allows us to easily pattern-match errors…  


**EXAMPLE #2:**

Getting errors from multiple different systems and want to keep these errors separate

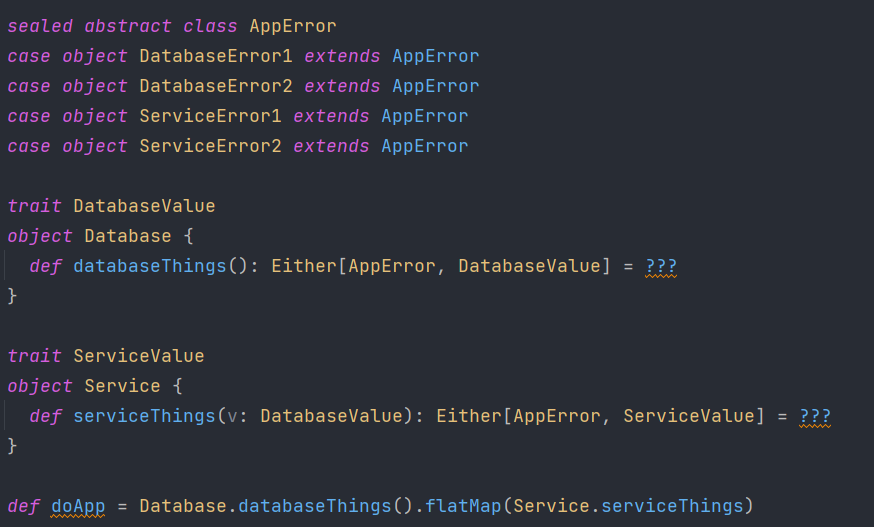


This flatMap is different from the ones you'll find on List or Option, for example, in that it has two type parameters, with the extra A1 parameter allowing us to flatMap into an Either with a different type on the left side.  
This behaviour is consistent with the covariance of Either, and in some cases it can be convenient, but it also makes it easy to run into nasty variance issues - such as Object being inferred as the type of the left side, as it is in this case.



**Solution 1: Application-wide errors**

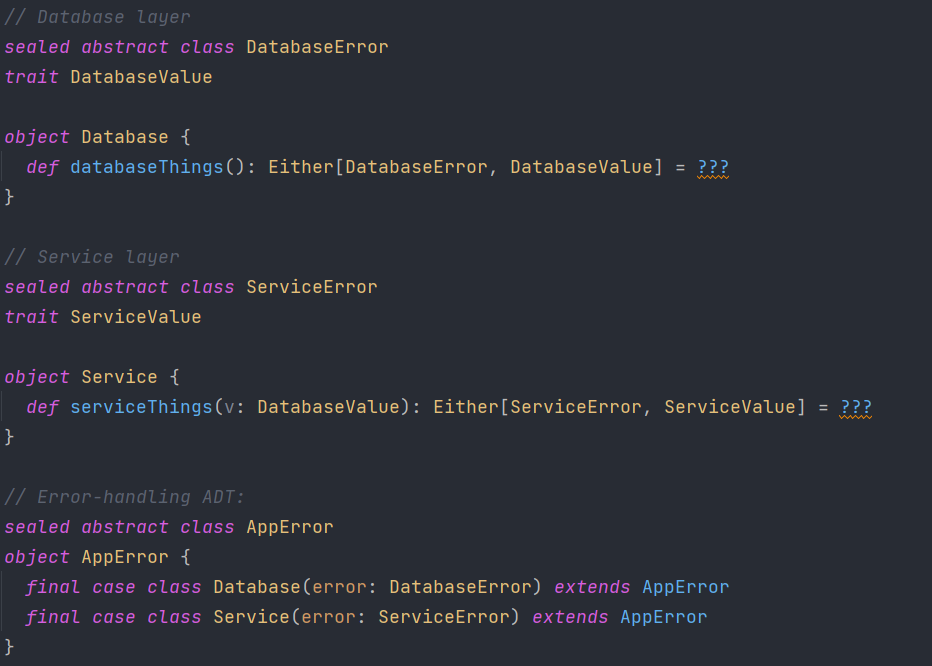
Make entire application share an error data type. This works, but:  
- When adding a new module to the system, would need to modify all existing errors  
- Means all layers/components see all errors in system (does database care about service layer errors?)

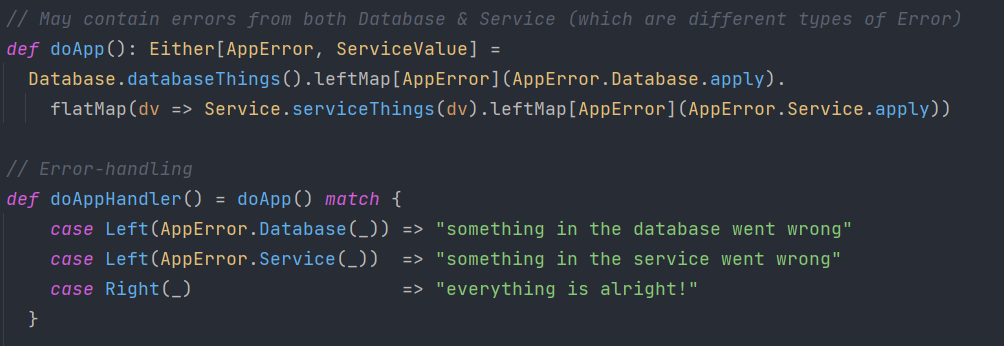
 **Solution #2: ADTs**

Instead of lumping all our errors into one big ADT, we can instead keep them local to each module, and have an application-wide error ADT that wraps each error ADT we need.

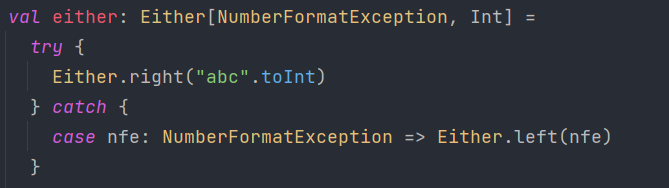
In application, we can wrap/lift each module-specific error into AppError and then call our combinators as usual. Either provides a convenient method to assist with this, called Either.leftMap - it can be thought of as the same as map, but for the Left side.

Each module only cares about its own errors and modules have their own error ADT that encapsulates each constituent module's error ADT. Doing this also allows us to take action on entire classes of errors instead of having to pattern match on each individual one.





**Example #3 - Working with Exception throwing code**



Instead of code like this, Either has a *catchOnly* method on its companion object (via syntax enrichment) that allows you to pass it a function, along with the type of exception you want to catch, and does the above for you.  
If you want to catch all (non-fatal) throwables, you can use *catchNonFatal*.

