Avoid Throwing Exceptions

Why Throwing Exceptions Are Bad

Exceptions Are Like GOTOs To Somewhere.... Anywhere...Or Nowhere

At least GOTOs go to known places in the code. Well.. it ain't that bad. Exceptions also carry data describing the nature of the exception.

But just like GOTOs exceptions break the normal logical flow of your code.

Exceptions Should Be Used For Exceptional Reasons

Let's be honest: Is an invalid email considered an **EXCEPTIONAL** condition in your code? I doubt it. In reality, **an invalid email is just that, an invalid email and your application should be able to handle it harmoniously**. Treating mundane conditions like an invalid email should never be an **EXCEPTIONAL** situation. Same goes for any other invalid data handled in your application. Your code shouldn't be hard to understand or maintain because your fields require validation or don't conform to the expectations of your application.

Conditions like running out of memory or not having enough CPUs to run your code successfully are exceptional conditions. And in those cases, perhaps we should just exit the program or thread instead, not just throwing an exception.

You Lose Referential Transparency Thus Functional Purity

From Avoid Throwing Exceptions In Medium

Throwing an `Exception` breaks referential transparency.

This can be demonstrated fairly easily. If `throw` was referentially transparent, by definition, the two following methods would be equivalent:

```
def foo1() = if(false) throw new Exception else 2


def getA(): Int = throw new Exception
def getAOr2(a: Int): Int = if (false) a else 2
def foo2() =
val a = getA()
    getAOr2(a)

// foo3 and foo2 should be identical but aren't/
```

But they aren't foo1() will return 2 and foo2() will throw an exception.

If we need to factor out code (imagine we want to factor out foo1() to improve testability) the code will suddenly behave differently!

Lacking referential transparency makes refactoring/ testing and debugging more difficult. It also makes it harder to use functional libraries such as ZIO which relies on your functions being referentially transparent.

Since scala is a functional language. When we write functional code, we want our functions to be as pure as possible, making our code more resilient and testable. Functions that throw exceptions are not pure functions (because it may not return an output): Method getA() returns an Int, but in reality it doesn't return anything during execution. Having functional purity provides clear benefits (see https://alvinalexander.com/scala/fp-book/benefits-of-pure-functions/)

It Encourages Code Duplication For Data Validation

When throwing exceptions for data validation, It's common practice to validate arguments at the beginning of a method. We like to keep things tidy eg

```
def validateEmail(string: String): Unit =
  if (string == null)
    throw new Error("Email is null")
  else
    val split = string.split("@")
    if (split.size != 2)
      throw new Error(s"Email ${string} is malformed")
    else ()

case class Email(user: String, domain: String)

def produceEmail(email: String): Email =
  validateEmail(email)
  // But we already did the same split in validateEmail(email)!
  val splitEmail: Array[String] = email.split("@")
  val emailUser = splitEmail(0)
  val emailDomain = splitEmail(1)
  Email(emailUser, emailDomain)
```

Many of us do this separate validation so we can then have a clean "happy path" code afterwards.

This replication is not specific to the example above. Validation often requires decomposing data to inspect for it's validity, the same goes for data parsing. Therefore, it often occurs that clean code may mean duplicated code when throwing exceptions for data validation.

Exceptions Are Slow

According to this article throwing a freshly created exception has been benchmarked to be more than 100 times slower than just returning an exception object. Imagine a service running 150 times slower just because most of the requests have invalid fields. It will create a chain effect where bad data lowers the performance of your application, apparently, for no good reason!

Revision #11

Created Wed, Aug 9, 2023 7:22 PM by hernan saab

Updated Sun, Aug 13, 2023 1:06 AM by hernan saab