Chapter 7 - Error Handling

Many code bases are completely dominated by error handling. When I say dominated, I don't mean that error handling is all that they do. I mean that it is nearly impossible to see what the code does because of all of the scattered error handling. Error handling is important, but if it obscures logic, it's wrong.

Use Exceptions Rather Than Return Codes

Back in the distant past there were many languages that didn't have exceptions. In those languages the techniques for handling and reporting errors were limited. You either set an error flag or returned an error code that the caller could check

Write Your Try-Catch-Finally Statement First

In a way, try blocks are like transactions. Your catch has to leave your program in a consistent state, no matter what happens in the try. For this reason it is good practice to start with a try-catch-finally statement when you are writing code that could throw exceptions. This helps you define what the user of that code should expect, no matter what goes wrong with the code that is executed in the try.

Provide Context with Exceptions

Each exception that you throw should provide enough context to determine the source and location of an error.

Create informative error messages and pass them along with your exceptions. Mention the operation that failed and the type of failure. If you are logging in your application, pass along enough information to be able to log the error in your catch.

Don't Return Null

If you are tempted to return null from a method, consider throwing an exception or returning a SPECIAL CASE object instead. If you are calling a null-returning method from a third-party API, consider wrapping that method with a method that either throws an exception or returns a special case object.

Don't Pass Null

Returning null from methods is bad, but passing null into methods is worse. Unless you are working with an API which expects you to pass null, you should avoid passing null in your code whenever possible.