S.O.L.I.D.

...

About this talk

- Interactive! (coding exercises with prizes)
- What is S.O.L.I.D.?
- Why should we care about S.O.L.I.D.?
- Going to assume you all know the basics of OO, if you don't feel free ask me a questions throughout!

Single Responsibility

- https://github.disney.com/IPBT-DCP/DisneyFileExchange/blob/Integration_Nov_7/Services/InsightToIpmMigration/DatabaseTransferObjects/MatchingStatementPair.cs
- Matching statement pair in DFE as a bad example
- Should have a matching statement generator class
- This is important because you when you look at a class called “MatchingStatementPair” and it does something other than that you have to do a mental context switch
- Also if you’re looking for the functionality which is hidden in the MatchingStatementPair class, it’s more of a pain to find
- We can fix this by separating classes and methods into the smallest sensible actions

Open/Closed principle

- Classes should be open for extension not modification
- We build our code so it encourages extending our classes rather than modifying the existing code in methods (because you may break the code that's already written!)
- Examples:
- https://github.disney.com/IPBT-DCP/DisneyFusion/tree/master/Api/Contract
- https://github.disney.com/IPBT/IPBTBOT/blob/master/extensions/sean_bot.coffee

Liskov Substitution Principle

- A superclass must be completely substitutable with any of its sub classes.
- This is important because:
  - Say you have a user class which is inherited by two sub classes (free user and regular user)
  - If user has a class getBankDetails and free user's version of this method just throws an exception because there is nothing then this means you cannot get a generic list of users and expect all methods for that to work all the time.
    - You can not just loop through a list of users and call getBankDetails
- Create a separate super class that can be implemented by the classes that need the extra methods

Interface Segregation Principle

- Don’t update interfaces and force clients of that interface to implement methods they won't use/don’t need!
- Interfaces should be as skinny as possible, just create a new extended version of that interface

Dependency inversion principle

- High level modules should not depend on the details of low level modules
- Both should depend on abstractions