



S.O.L.I.D. Principles of OOP

History of SOLID Principals

- The SOLID principles date back to March of 1995.
- The principles are from Robert “Uncle Bob” Martin.
- Started as writings, which ultimately were turned into the book “Agile Software Development: Principles, Patterns, and Practices.
- Michael Feathers is credited with coming up with the SOLID acronym.

Why Use the SOLID Principles of OOP?

- Object Oriented Programming is a powerful concept
- But, OOP does not always lead to quality software
- The 5 principles focus on dependency management
- Poor dependency management leads to code that is brittle, fragile, and hard to change
- Proper dependency management leads to quality code that is easy to maintain.



Single Responsibility Principle

Just because you *can* doesn't mean you *should*.

Single Responsibility Principle

- Every Class should have a single responsibility.
- There should never be more than one reason for a class to change.
- Your classes should be small. No more than a screen full of code.
- Avoid 'god' classes.
- Split big classes into smaller classes.

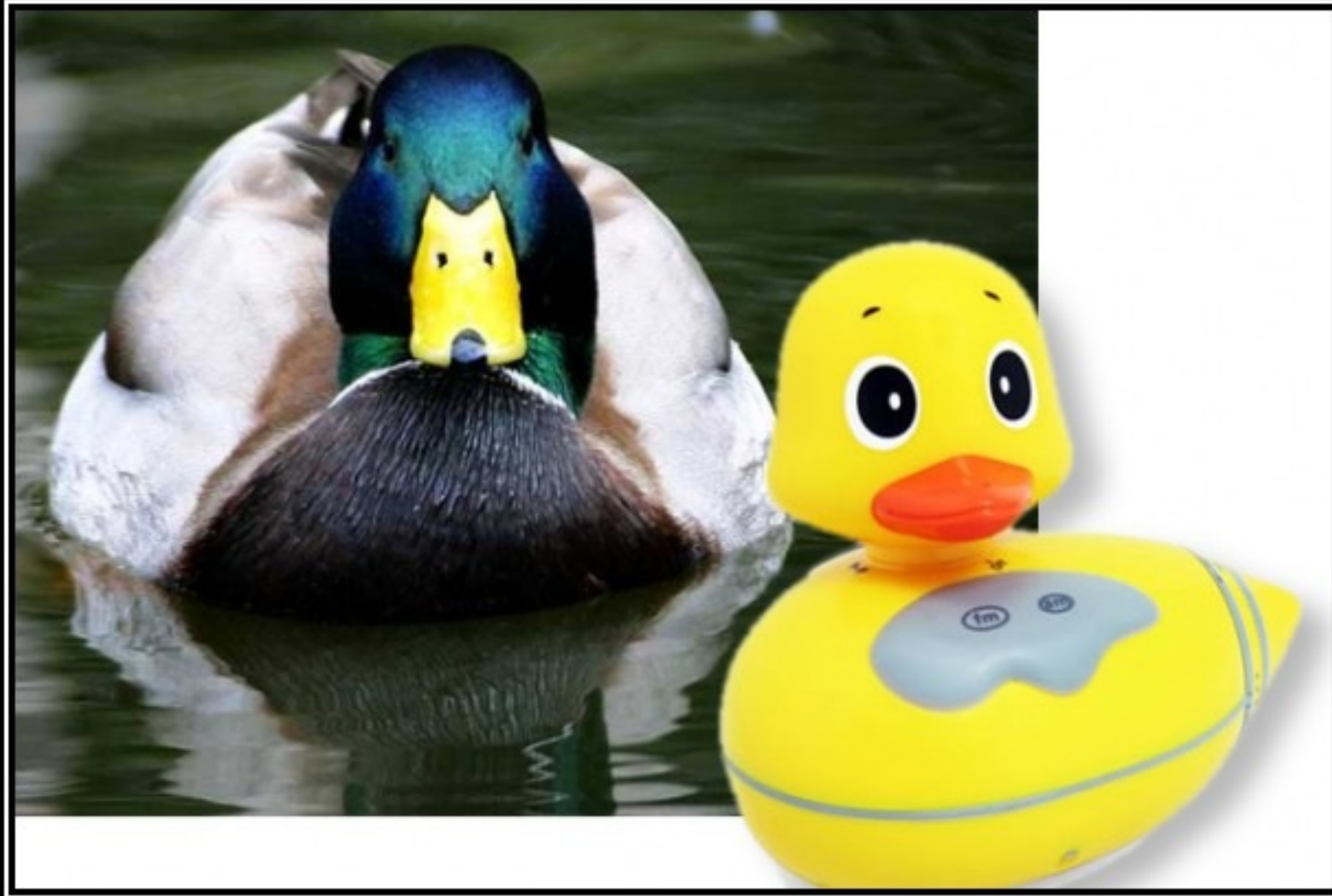


OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat

Open/Closed Principle

- Your classes should be open for extension
- But closed for modification
- You should be able to extend a classes behavior, without modifying it.
- Use private variables with getters and setters - ONLY when you need them.
- Use abstract base classes



LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

Liskov Substitution Principle

- By Barbara Liskov, in 1998
- Objects in a program would be replaceable with instances of their subtypes WITHOUT altering the correctness of the program.
- Violations will often fail the “Is a” test.
- A Square “Is a” Rectangle
- However, a Rectangle “Is Not” a Square



INTERFACE SEGREGATION PRINCIPLE

You Want Me To Plug This In, Where?

Interface Segregation Principle

- Make fine grained interfaces that are client specific
- Many client specific interfaces are better than one “general purpose” interface
- Keep your components focused and minimize dependencies between them
- Notice relationship to the Single Responsibility Principle?
 - ie avoid ‘god’ interfaces



Dependency Inversion Principle

Would you solder a lamp directly to the electrical wiring in a wall?

Dependency Inversion Principle

- Abstractions should not depend upon details
- Details should depend upon abstractions
- Important that higher level and lower level objects depend on the same abstract interaction
- This is not the same as Dependency Injection - which is how objects obtain dependent objects

Summary

- The SOLID principles of OOP will lead you to better quality code.
- Your code will be more testable and easier to maintain.
- A key theme is avoiding tight coupling in your code.

