# SOLID

Software Development is not a Jenga game

# ПРИНЦИПЫ SOLID

5 принципов объектно-ориентированного программирования, описывающих архитектуру программного обеспечения

# Что такое SOLID

- **SOLID** - это аббревиатура пяти основных принципов дизайна классов в объектно-ориентированном проектировании.
- Аббревиатура была введена Робертом Мартином в начале 2000-х.

- Рекомендую: **Чистый код. Роберт Мартин**

# Принципы объектно-ориентированного дизайна

# Single responsibility
## Принцип единственной обязанности

- Класс или модуль должны иметь одну и только одну причину измениться.

- Все члены этого класса должны быть связаны одной целью. Наш класс не должен быть похож на швейцарский нож, в котором при изменении одного из членов нужно изменять весь инструментарий.


# SINGLE RESPONSIBILITY PRINCIPLE

Just Because You Can, Doesn't Mean You Should

# Open-closed
# Принцип открытости/закрытости

- Объекты проектирования должны быть открыты для расширения, но закрыты для модификации.

- Это означает, что новое поведение должно добавляться только добавлением новых сущностей, а не изменением старых.

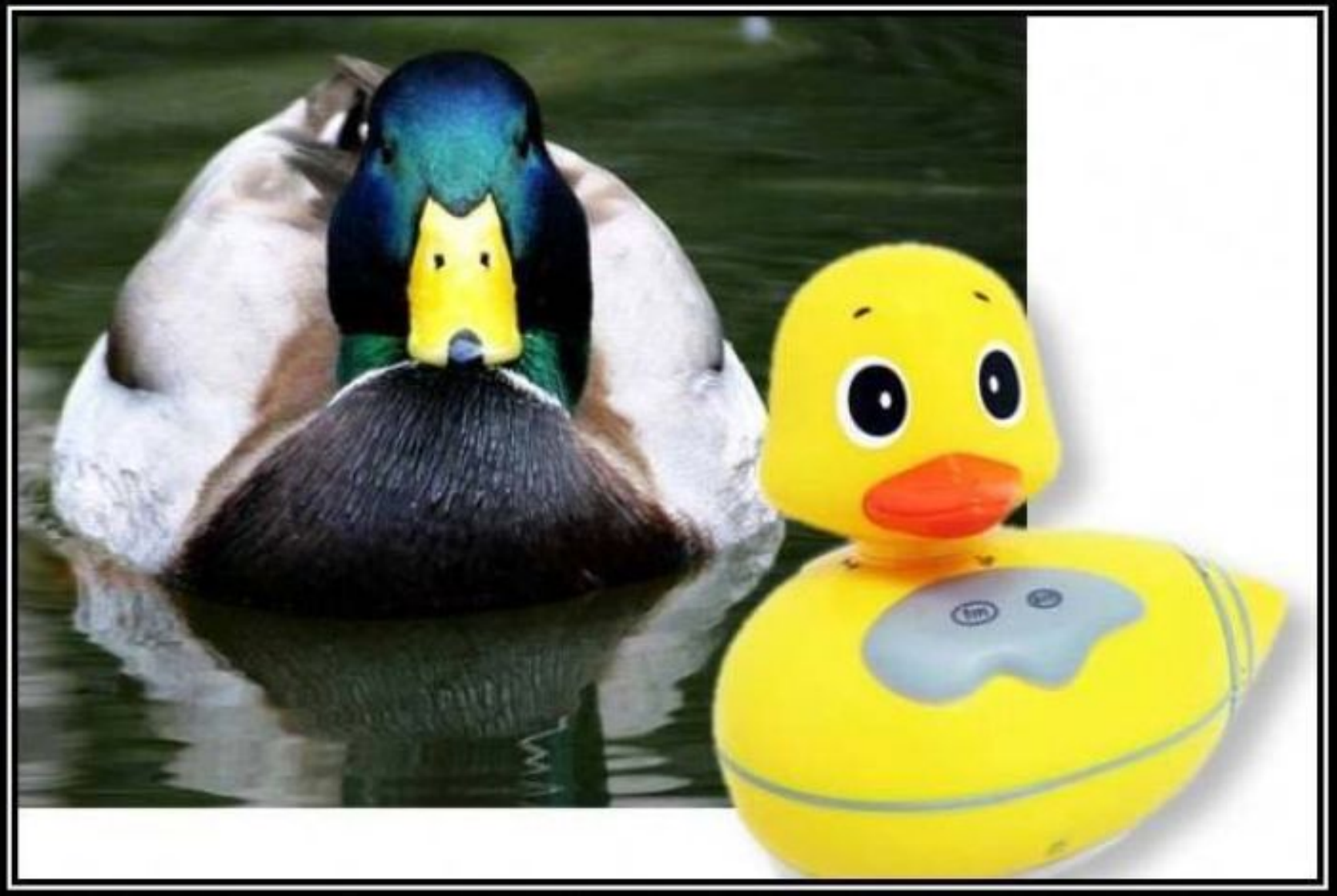
# OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat

# Liskov substitution

## Принцип подстановки Барбары Лисков

- Данный принцип гласит, что «вы должны иметь возможность использовать любой производный класс вместо родительского класса и вести себя с ним таким же образом без внесения изменений».



# LISKOV SUBSTITUTION PRINCIPLE

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

# Interface segregation
## Принцип разделения интерфейса

- Слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические, чтобы клиенты маленьких интерфейсов знали только о методах, которые необходимы им в работе.

# INTERFACE SEGREGATION PRINCIPLE

You Want Me To Plug This In, Where?

# Dependency inversion
## Принцип инверсии зависимостей

- Все взаимосвязи в программе должны поддерживаться с помощью абстракных классом или интерфейсов.
- Данный принцип гласит, что, во-первых, **классы высокого уровня не должны зависеть от низкоуровневых классов**. При этом оба должны зависеть от абстракций. Во-вторых, **абстракции не должны зависеть от деталей**, но детали должны зависеть от абстракций.


# DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

# Конфликты S.O.L.I.D. с другими подходами проектирования

- **Active Record** нарушает принцип единственной обязанности
- **Singleton** нарушает инверсию зависимости
- **Decorator** – декорирование метода Delete нарушает контракт метода
- **DDD** – МФУ с точки зрения DDD объект предметной области, с точки зрения ***Interface Segregation –*** антишаблон

# Вывод

- Слепое следование каким либо принципам может расходиться со здравым смыслом
- S.O.L.I.D. – мощный инструмент проектирования
- S.O.L.I.D. – позволяет создавать гибкие программные модули

**Изменения неизбежны! Будьте гибкими.**