SOLID

Design Principles

Powerful - yet underrated

But WHY?

because everybody

- Readable
- Maintainable
- Testable code

* The only constant thing in life is CHANGE - you can't AVOID it though you can DESIGN for it - with SOLID principles.

S - Single Responsibility Principle

O - Open and Close Principle

L - Liskov's Substitution Principle

I - Interface Segregation Principle

D - Dependency Inversion Principle

S - Single Responsibility

- logical unit - module, class, method - should

perform only one desired action

- avoid if-else, switch statements: cause of code smell

- Name - what you want to do!

Do - what you named it!

O - Open and Close

open for extension, closed for modification

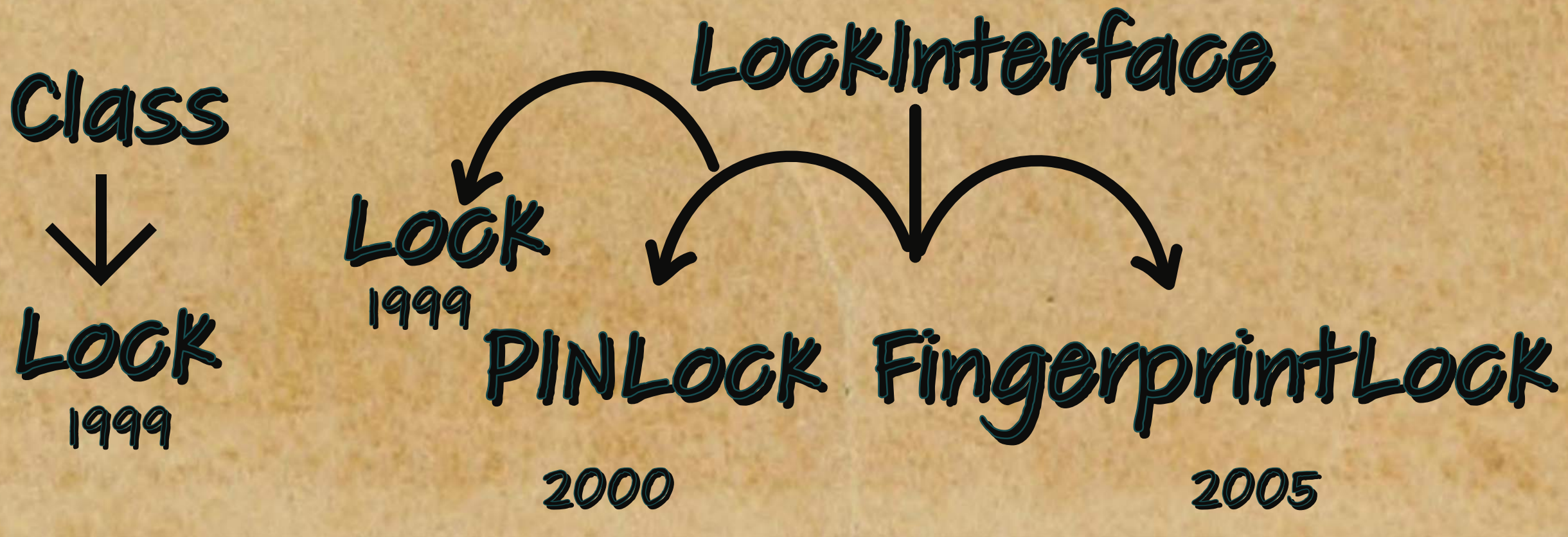
- everybody secures their home with locks: a good thing

- the year 1999 - wrote a class called Lock - job done

- in the year 2000 - digital pin locks arrived. Ah! what to do now?

- the year 2005 - fingerprint locks arrived.

Not an issue I followed SOLID?

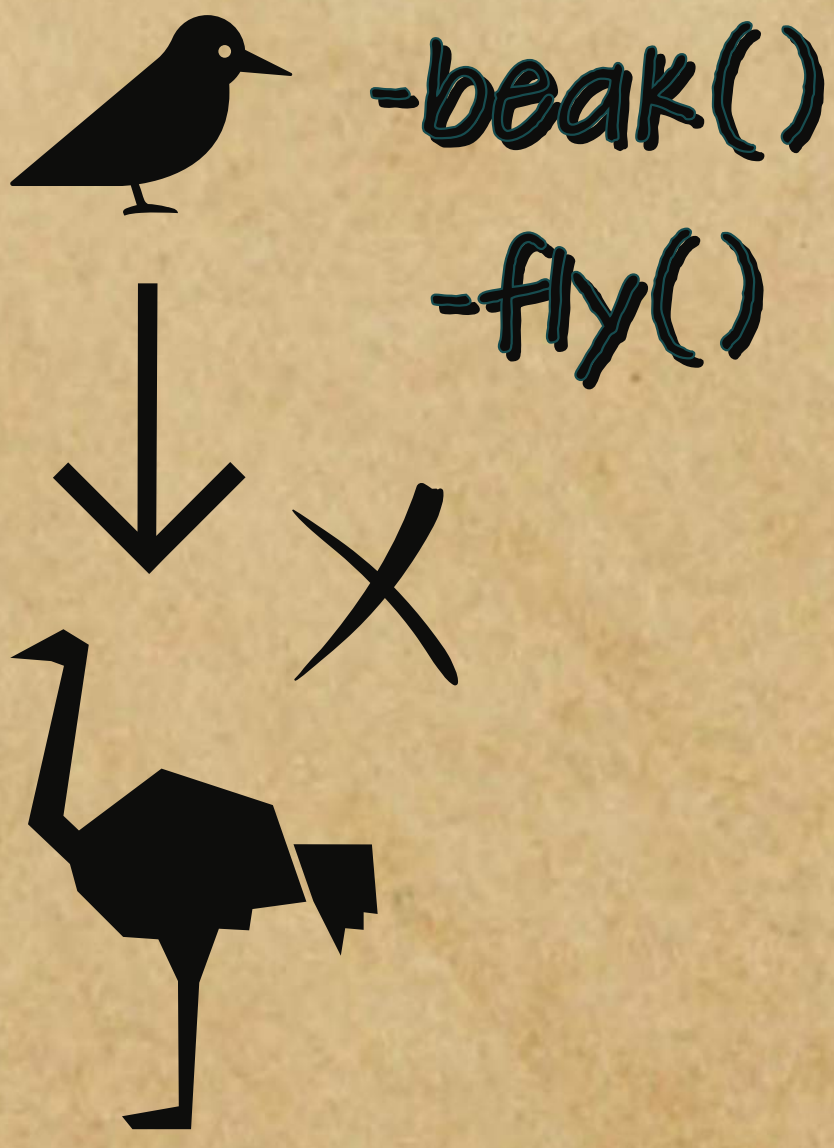


L - Liskov's Substitution

Don't ask about the "IS-A" relation

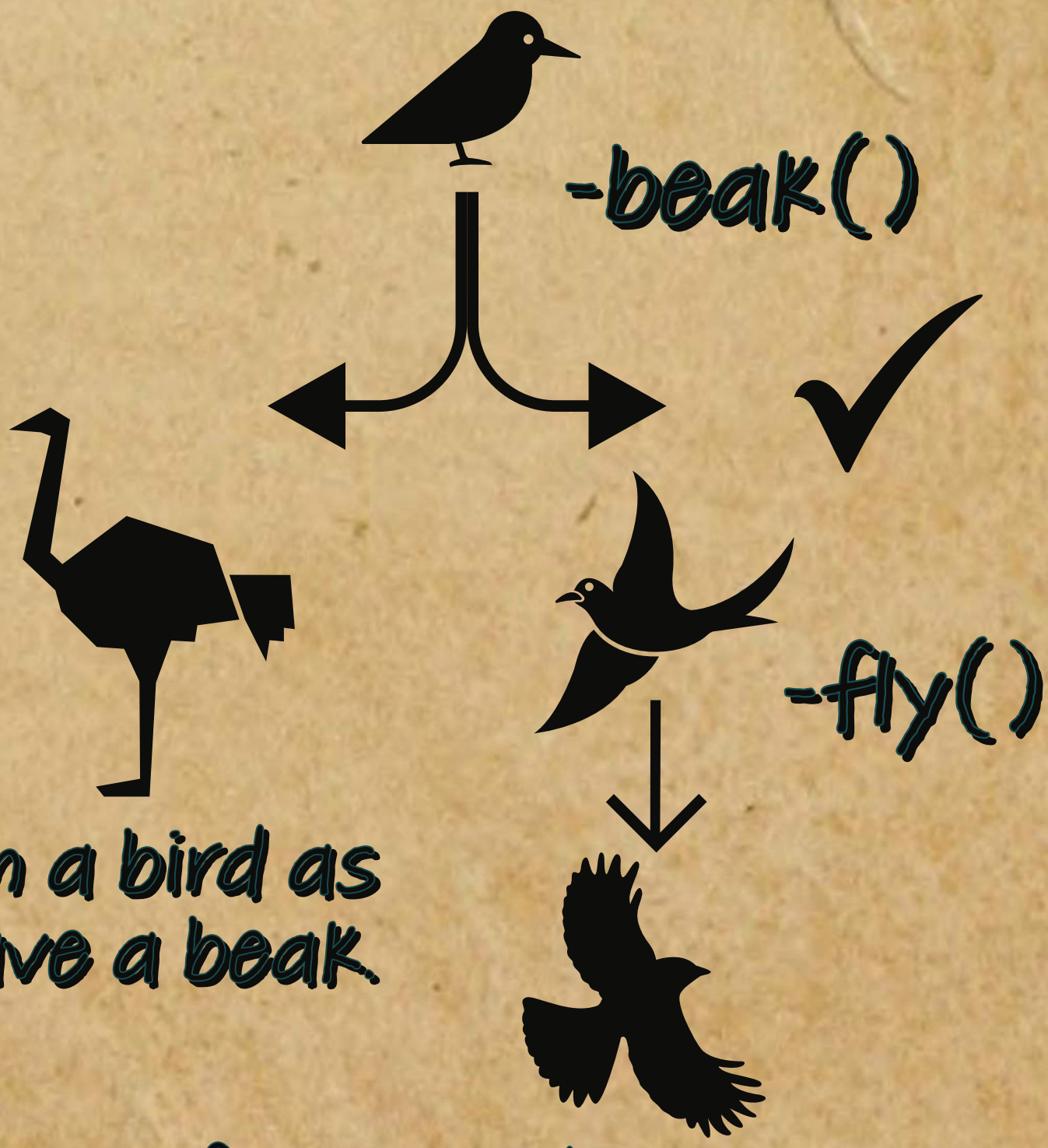
instead check if "it is substitutable by"

Eg: Ostrich IS-A Bird but can Ostrich fly?



I have a beak

but I can't fly



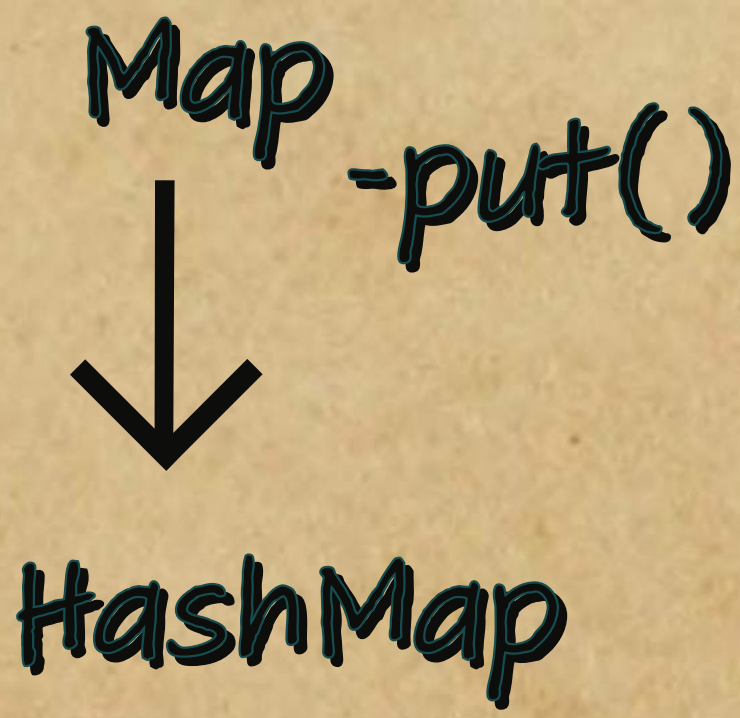
I am a bird as
I have a beak.

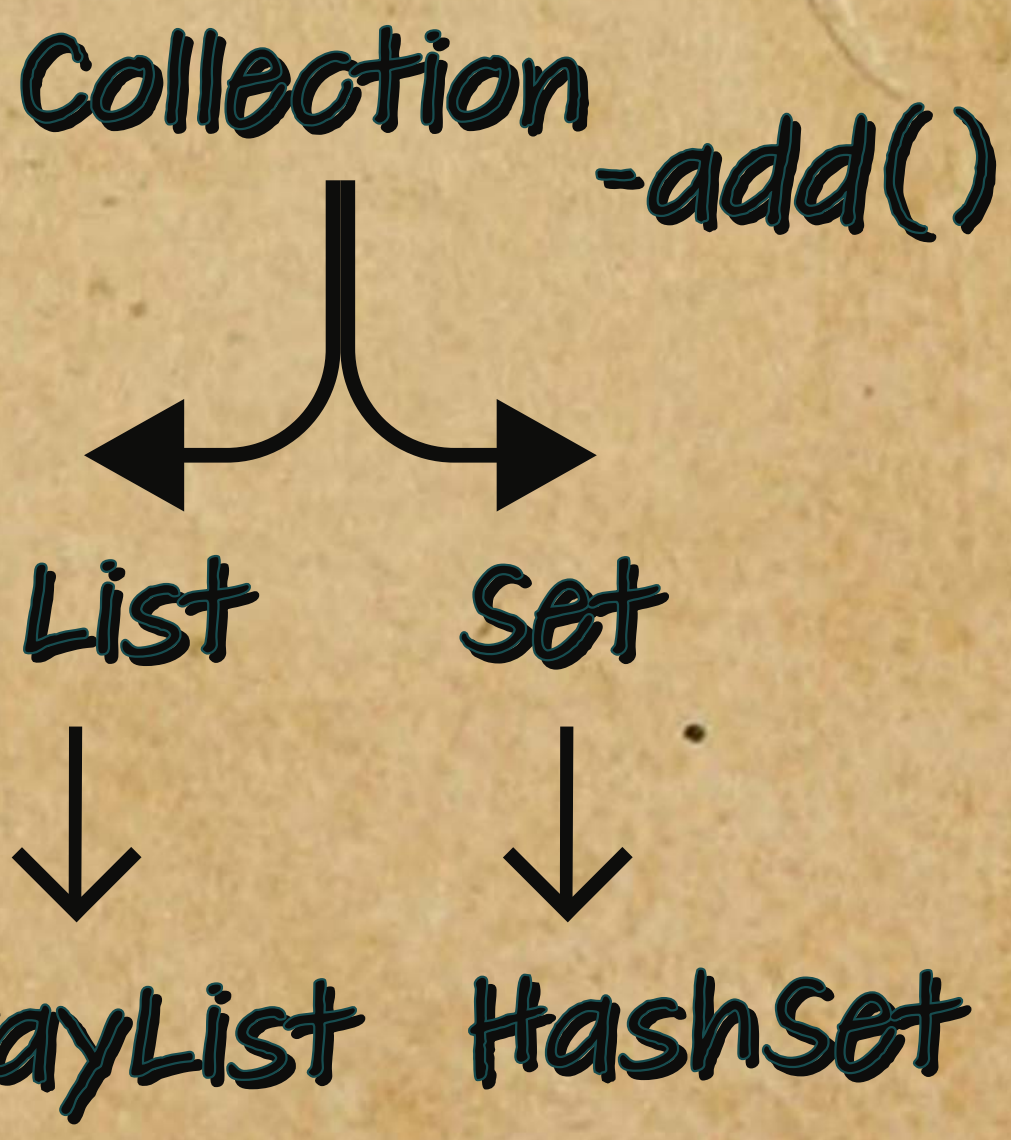
I am a flying bird as I have a beak and I can fly.

I - Interface Segregation

Break big interfaces into smaller focused interfaces

- reinforces LSP SRP
- eg.: Java collections - Map interface





put() was not forced to be part of the Collection interface otherwise all implementations would have empty methods

D - Dependency Inversion

- High-level modules shouldn't depend on low-level modules rather depend on abstraction

- Abstractions should not depend on details. Details should depend on abstractions.

- the idea is to have to lose coupling between the modules/classes.
- to put it we can achieve this with the help of two concepts
- Dependency Injection - DI
- Inversion of control - IoC
- introduce interfaces/abstract classes as abstraction layers
- do not create objects within consumer classes instead pass them in.
- Does it reminds you of something? Spring framework. Yeah, that's right!

Detailed video on SOLID Design Principles on my YouTube channel @YouCanCodeX

Amit Dhall

@YouCanCodeX