
Table of Contents

Introduction	1.1
Let's Go!	1.2
S: Single Responsibility	1.3
O: Open/Closed	1.4
L: Liskov Substitution	1.5
I: Interface Segregation	1.6
D: Dependency Inversion	1.7

SOLID Design Principles In Common Lisp

Learn how to apply SOLID design principles with Common Lisp and the powerful CLOS system.



This book is released under the [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/) by Momozor.

If you find any problem, want to suggest an improvement or commit changes to this book, please visit this Github repository <https://github.com/common-lisp-reserve/solid-design-principles-in-common-lisp>

Let's Go!

What is SOLID?

- Single Responsibility Principle
- Open/Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

S: Single Responsibility

A class should have one, and only one, reason to change.

Bad

```
(defclass truck ()
  ((brand
    :initarg :brand
    :accessor brand)))

(defmethod get-brand ((self truck))
  (brand self))

(defmethod set-brand ((self truck) new-brand)
  (setf (brand self) new-brand))

(defmethod send-detail ((self truck) customer-id)
  "send truck's brand detail to customer..")
```

Good

```
(defclass truck ()
  ((brand
    :initarg :brand
    :accessor brand)))

(defmethod get-brand ((self truck))
  (brand self))

(defmethod set-brand ((self truck) new-brand)
  (setf (brand self) new-brand))

(defclass detail-sender ()
  ((customer-id
    :initarg :customer-id
    :accessor customer-id)))

(defmethod get-customer-id ((self detail-sender))
  (customer-id self))

(defmethod set-customer-id ((self detail-sender) new-customer-id)
  (setf (customer-id self) new-customer-id))

(defmethod send-detail ((self detail-sender))
  (send (customer-id self)))
```

O: Open/Closed

Software entities (classes, modules, functions, etc) should be open for extension, but closed for modification.

L: Liskov Substitution

Let $\Phi(x)$ be a property provable about objects x of type T . Then $\Phi(y)$ should be true for objects y of type S where S is a subtype of T .

.

L: Interface Segregation

Clients should not be forced to depend upon interfaces that they do not use.

D: Dependency Inversion

- High level modules should not depend upon low level modules. Both should depend upon abstractions.
- Abstractions should not depend upon details. Details should depend upon abstractions.