# Table of Contents

# SOLID Design Principles In Common Lisp

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



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

# 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..")

(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))
```

## Good

```
(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.**

## Bad

```
(defclass circle ()
  ((radius
    :initarg :radius
    :reader get-radius)))

(defclass area-calculator ()
  ((shapes
    :initarg :shapes
    :reader get-shapes)))

(defmethod total-area ((self area-calculator))
  (reduce #'+
          (mapcar #'(lambda (x)
                      (* pi
                         (get-radius x)
                         (get-radius x)))
                  (get-shapes self))))
```

## Good

```
(defclass shape ()
  nil)

(defclass circle (shape)
  ((radius
    :initarg :radius
    :reader get-radius)))

(defmethod area ((self shape))
  (* pi (get-radius self) (get-radius self)))

(defclass area-calculator ()
  ((shapes
    :initarg :shapes
    :reader get-shapes)))

(defmethod total-area ((self area-calculator))
```

```
    (reduce #'+
            (mapcar #'area
                    (get-shapes self))))
```

# L: Liskov Substitution

**Let *Φ(x) be a property provable about objects x of type T*. Then *Φ(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 asbtractions.**