S.O.L.I.D. Principles

Martin Stolle, 25.04.2018

S.O.L.I.D.

- ➔ Single-responsibility principle
- ➔ Open-closed principle
- ➔ Liskov substitution principle
- ➔ Interface segregation principle
- ➔ Dependency inversion principle

Single Responsibility Principle

A class should have one and only one reason to change, meaning that a class should have only one job.

Single Responsibility Principle

```
class Book
  def get_author:
   return author

  def get_title:
   return title

  def export(format):
   # Export logic
   # pdf / html / epub export
```

```
class Book
  def get_author:
   return author

  def get_title:
   return title

class Epub
  def export(book);

book = Book()
epub= Epub()
epub.export(book)
```

Open-Closed Principle

Objects or entities should be open for extension, but closed for modification.

Open-Closed Principle

```
class Report
  def body():
    return { 'a': 'Alf', 'b': 'Bob', 'c': 'Joe' }

  def print_json():
    body.to_json()
```

```
class Report:
  def body:
    return { 'a': 'Alf', 'b': 'Bob', 'c': 'Joe' }

  def print(formatter):
    formatter.format(body)

report = Report()
report.print(formatter=XMLFormatter())
report.print(formatter=JSONFormatter())
```

Liskov Substitution Principle

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

Liskov Substitution Principle

```ruby
class Animal
  def walk
     # do some walking

class Cat < Animal
  def run
    # do some cat style running
```

```ruby
class Animal
  def walk
     # do some walking

  def run
     raise NotImplementedError

class Cat < Animal
  def run
    # do some cat style running
```

Interface Segregation Principle

A client should never be forced to implement an interface that it doesn’t use or clients shouldn’t be forced to depend on methods they do not use.

Interface Segregation Principle

```
class Car
  def open;
  def start_engine;
  def change_engine;

class Driver
  def drive
    # use `Car.open` and `Car.start_engine`

class Mechanic
  def do_stuff
    # use `Car.change_engine`
```

```
class Car
  def open;
  def start_engine;

class CarInternals
  def change_engine;

class Driver
  def drive
    # use `Car.open` and `Car.start_engine`

class Mechanic
  def do_stuff
    # use `CarInternals.change_engine`
```

Dependency Inversion Principle

Entities must depend on abstractions not on concretions. It states that the high level module must not depend on the low level module, but they should depend on abstractions.

Dependency Inversion Principle

```
class Car
  def start_engine;
  def stop_engine;

class Driver
  vehicle = Car()
  def drive
    # Car.start_engine/Car.stop_engine
```

```
Interface Vehicle
    def start_engine;
    def stop_engine;

class Car < Vehicle
    def start_engine;
    def stop_engine;

class Driver
  vehicle = Vehicle()
  def drive
    # Vehicle.start_engine/Vehicle.stop_engine
```