SOLID Principles in Python
Designing software that scales

PyGeekle 22 Online Summit

Aldo Fuster Turpin
Software Engineer
Red Hat

Agenda

1. Introduction
2. The objectives of **SOLID**
3. The bases of software design
4. The **SOLID** Principles
    1. Single Responsibility (SR)
    2. Open-Closed (OC)
    3. Liskov Substitution (LS)
    4. Interface Segregation (IS)
    5. Dependency Inversion (DI)

“I'm not a great programmer;

I'm just a good programmer with great habits.”

— Kent Beck

1. Introduction

SOLID

Acronym created by Kent Bech based on the OOP principles developed by Robert C. Martin in 2000 in his paper “Design Principles and Design Patterns”.

2. The objectives of SOLID

Serve as a guide to create software:

- Scalable
- Maintainable
- Reusable
- Stable

In short, to create quality software.

3. The bases of software design

- Cohesion

Degree to which the different elements of a system are united to achieve a better result than if they worked separately.

How we can **bundle** multiple software units **together** to create a larger unit.

- Coupling

Degree of **interdependence** that two software units (classes, functions, modules…) have with each other.

4. The SOLID Principles

1. Single Responsibility (SR)

- What: *"A class should have one, and only one, reason to change."*
- How: *"Gather together the things that change for the same reasons. Separate those things that change for different reasons"*

2. Open-closed (OC)

- What*: “Software entities should be open for extension, but closed for modification.”*
- How: “Use polymorphism instead of hard-coding if/else behaviour in methods”

3. Liskov Substitution (LS)

- What*: “Derived classes must be substitutable for their base classes”*.

- How: *“Ensure classes just implement/extend from/what they do, NO from general behaviour”*

4. Interface segregation (IS)

- What*: “Many client-specific interfaces are better than one general-purpose interface.”*

- How: *“Use many interfaces that define few methods instead of being forced to implement methods that will not be used.”*

5. Dependency inversion (DI)

- What*: "Depend upon abstractions, not concretions"*
- How:
  - ✓ *"High-level modules should not depend on low-level modules. Both must depend on abstractions."*
  - ✓ *"Abstractions should not depend on details. The details should depend on the abstractions."*

Thank you!

You can find me at

- https://github.com/AldoFusterTurpin
- https://www.linkedin.com/in/aldo-fuster-turpin

The code repo

- https://github.com/AldoFusterTurpin/Solid_Principles_Python