THE SOLID PRINCIPLES

OVERVIEW OF THE SOLID PRINCIPLES

SOLID is a mnemonic acronym for five principles

Single Responsibility Principle

Open/Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

Some argue that these are the ”first five” principles • That claim has not been justified or widely accepted • However, whether they are the “first five” principles is not very important • Following these principles can help ensure quality software, primarily from a developers' perspective

DESIGN PROBLEM

You've been contracted to build a Maze Generator

The program must be able to print the mazes using ASCII character or draw them in an image

SINGLE RESPONSIBILITY PRINCIPLE

This principle is vey closely related to the more general principle of Cohesion, which says that the responsibilities of any component (method, class, sub-system, etc.) should be tightly aligned and focused on a single purpose

This principle is also related to the principles of

• Localization of design decisions

• Encapsulation

Following this principle can help

• Increase Reuse and Maintainability

• Reduce Complexity, even though the number of classes might increase

OPEN/CLOSED PRINCIPLE

Core Ideas: Software entities (e.g., classes, generics) should be open for extension but closed to modification

Original definitions: A class is open if it is still available for extension , A class is closed if it is available for use by other class, and therefore should not be modified

Revised definitions:

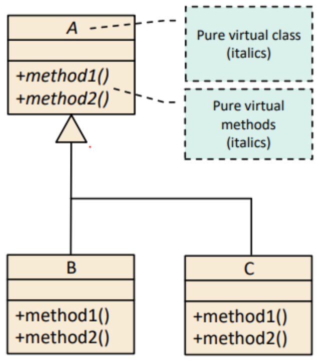
A system of classes is open for extension and closed for modification, if

Public methods (e.g., the abstractions) are declared using interfaces, or abstract classes (in Java)

Implementations or concrete classes inherit the public method declarations from the interfaces, abstract classes, or pure virtual classes

Users

INTERFACES, ABSTRACT CLASSES, PURE VIRTUAL CLASSES



OPEN/CLOSED PRINCIPLE

Ways to achieve the open/closed principle

Inheritance

Move public methods into their own abstractions, namely interfaces, abstract classes, or pure virtual classes

The public methods of one class can be grouped into multiple abstractions

Each abstraction should focus on a single purpose, as per the Single Responsibility Principle

Have concrete classes inherit from these abstraction
Java does not support multiple inheritance, so a class can have multiple base classes