Prinsip SOLID pada Pemrograman JAVA

# Apa itu SOLID?

Prinsip SOLID adalah singkatan dari **Single Responsibility Principle, Open-Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, dan Dependency Inversion Principle**. Tujuan dari SOLID adalah memastikan bahwa software yang anda buat bersifat dinamis sehingga mudah untuk dimengerti, dikelola, dan diperbaharui, sehingga kode anda dapat memiliki jangka hidup yang berkepanjangan.

# Prinsip SOLID

## Single Responsibility

Masing-masing class pada software seharusnya hanya memiliki satu fungsi. Jika anda mengimplementasikan lebih dari satu fungsi ke dalam sebuah class, saat salah satu fungsi tersebut perlu diubah, akan dapat mempengaruhi fungsi lainnya dalam class tersebut. Single Responsibility membuat class anda lebih ramping dan mudah diperbaharui atau diperbaiki.

## Open-Closed Principle

Open-Closed Principle adalah prinsip saat anda perlu melakukan pembaharuan pada software anda seperti menambah sebuah fitur baru. Pada situasi ini, modul-modul yang sudah ada harus dapat mengimplementasikan ekstensi ke modul baru, namun tidak menerima perubahan di dalam modul itu sendiri.

## Liskov Substitution Principle

Liskov Substitution Principle (LSP) mendikte bahwa sebuah class yang berasal dari class lain harus dapat digunakan untuk menggantikan class asalnya tersebut. Sebagai contoh jika anda memiliki class A yang merupakan sub-type dari class B, maka, anda harus dapat menggunakan class A pada modul-modul yang mengimplementasikan class B tanpa mengganggu berjalannya software.

## Interface Segregation Principle

Jika software anda memiliki sebuah interface dengan banyak method, maka class-class yang mengimplementasikan interface tersebut harus juga mengimplementasikan semua method yang terdapat pada interface itu. Tujuan dari prinsip ini adalah memecahkan interface besar menjadi interface-interface kecil, sehingga class-class anda hanya perlu menerapkan method-method yang diperlukan saja.

## Dependency Inversion

Dependency inversion menyatakan bahwa anda harus menghindari penggunaan implementasi yang bersifat konkrit, dengan kata lain, anda sebaiknya menerapkan abstraksi (class dan interface abstrak). Modul yang bersifat high-level tidak boleh bergantung pada modul low-level, namun kedua modul tersebut harus mengarah pada abstraksi.