SOLIDの原則

SOLIDの原則とは

・オブジェクト指向プログラミングをする上で、重要となる５つの原則の頭文字をとったものです。
・この原則を守ることで、プログラムを可読性が向上し、保守・拡張が容易になります
・有名なソフトウェア開発者のRobert C. Martinの2000年の論文で発表されました
https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf
・ソフトウェアの開発者なら是非知っておきたい内容です。

1. 単一責任の原則(Single responsibility principle)

2. 解放閉鎖の原則(Open/closed principle)

3. リスコフの置換原則(Liskov substitution principle)

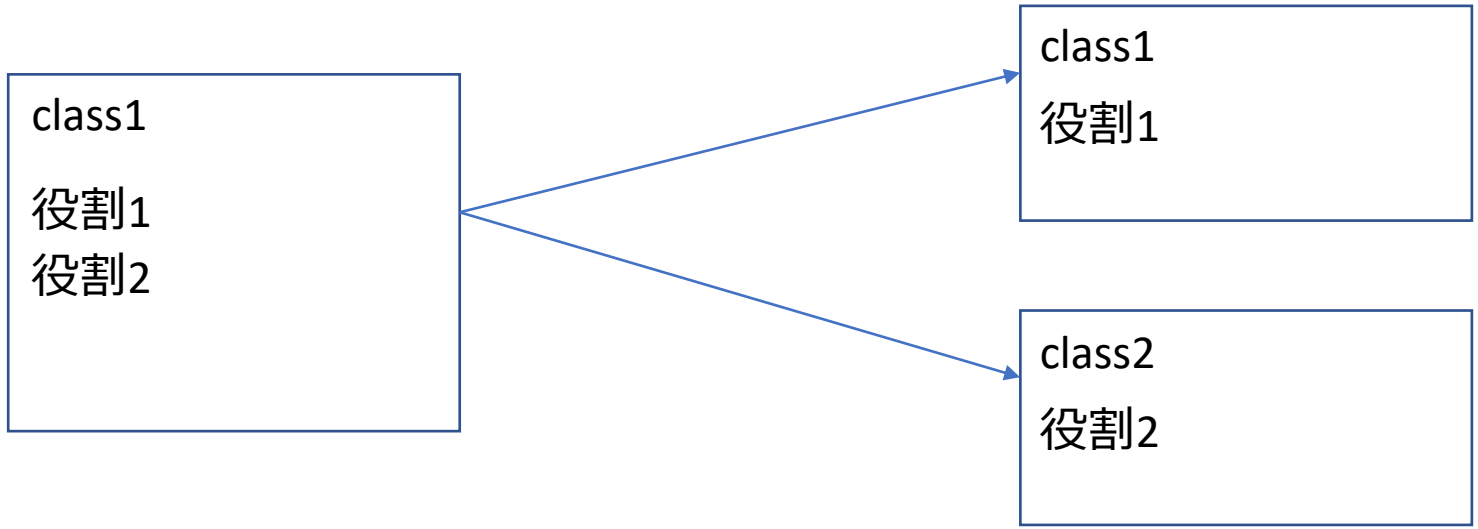
4. インタフェース分離の原則(Interface segregation principle)

5. 依存性逆転の原則(Dependency inversion principle)

単一責任の原則(Single responsibility principle)

全てのモジュールとクラスは1つの役割を提供して責任をもつべきとする原則
・役割が複数ある場合、各クラスが何なのか複雑化してしまう。
・１つの役割に別の役割が複雑に依存してしまい、１つの役割の変更が別の役割に影響を与える可能性があり、保守性の低下を招く

[単一責任の原則を守った場合]
・クラス、モジュールを分けることで、可読性が向上する
・役割を分けることで、他のクラスから利用できやすくなり拡張性が向上する



開放閉鎖の原則(Open/closed principle)

クラス、モジュール、関数等のソフトウェアの部品は拡張に対しては開いており、修正に対しては閉じていなければならない
・アプリケーションに変更があった場合、コードを追加して拡張し容易に対応できる
・モジュールの動作を変更する場合には、ソースコードを直接修正するようなことはないようにする

【メリット】
・機能拡張が容易になりソフトウェアの拡張性が向上する
・機能拡張時にテストをする際に、拡張機能だけを実施すればよく、元のソースコードはテストをしなくてよいため、保守性も向上する

リスコフの置換原則(Liskov substitution principle)

サブクラスは、そのスーパークラスの代用ができなければならない

φ(x) という関数に対してTクラスのインスタンスxで実行できるなら、
φ(y) という関数に対してTクラスのサブクラスのSのインスタンスyで実行できなければならない

・スーパークラスの仕様を理解すれば、それを継承したサブクラスの中身をすべて確認せずに利用することができ、拡張性、保守性が向上する。
・サブクラス、スーパークラス間で実行できるものと実行できないものがあると、サブクラスのコードをすべて理解する必要が生じる。

インタフェース分離の原則(Interface segregation principle)

インタフェース上に必要のないメソッドを追加して、継承先が無駄なコードを作成することがないようにする代わりに、新しいインターフェースを作成してクラスは必要なインタフェースを継承するようにする

・継承するインターフェースの持つメソッドは必要最小限のものにする
・インターフェースの持つメソッドを増やしすぎることで、継承先のクラスに無駄なコードが増えてしまい、コードが複雑になり利用者は混乱する
・インターフェースの継承先が増えるため、インターフェースの修正による、継承先のクラスの修正量が多くなり保守性が低下する

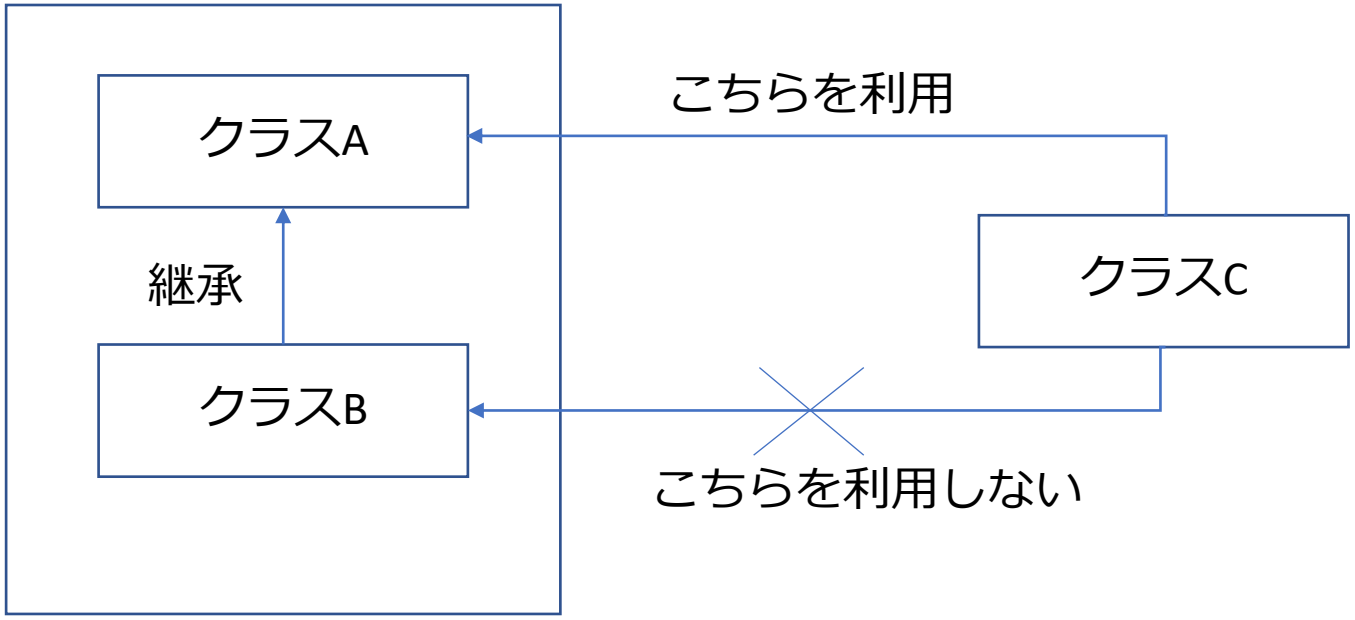
*) インターフェースとは、メソッドの中身を記載していない継承して利用するためのクラスのことです

依存性逆転の原則(Dependency inversion principle)

ソフトウェアモジュール間の依存関係を切り離すための方法

・高水準なモジュールは、低水準のモジュールに依存してはいけない。両者は抽象化に依存すべき
・抽象化は詳細に依存すべきでなく、詳細は抽象化に依存すべきである

依存性逆転の原則を守ることで、低水準モジュールを継承したクラスを利用した機能拡張が容易になる。



まとめ

・オブジェクト指向プログラミングをする上で、重要となる５つの原則の頭文字をとったものです。
・この原則を守ることで、プログラムを可読性が向上し、保守・拡張が容易になります

1. 単一責任の原則(Single responsibility principle)
   →全てのモジュールとクラスは1つの役割を提供して責任をもつべきとする原則

2. 解放閉鎖の原則(Open/closed principle)
   →クラス、モジュール、関数等のソフトウェアの部品は拡張に対しては開いており、修正に対しては閉じていなければならないとする原則

3. リスコフの置換原則(Liskov substitution principle)
   →サブクラスは、そのスーパークラスの代用ができなければならないとする原則

4. インタフェース分離の原則(Interface segregation principle)
   →インターフェースのメソッドが多すぎる場合には、インターフェースを分離する原則

5. 依存性逆転の原則(Dependency inversion principle)
   →高水準なモジュールは、低水準のモジュールに依存してはいけない。両者は抽象化に依存すべきとする原則

デザインパターンでは、これらの原則が利用されています