

ソフトウェア工学 第7回

— モジュール設計 —

大連理工大学・立命館大学 国際情報ソフトウェア学部

大森 隆行

講義内容

■ モジュール設計

➡ ■ 概説

- モジュール分割の評価基準
- モジュール分割技法
- オブジェクト指向におけるモジュール設計

モジュール設計

■ ソフトウェアをモジュールに分割し構造化する作業

- ソフトウェア = 複数のモジュールで構成

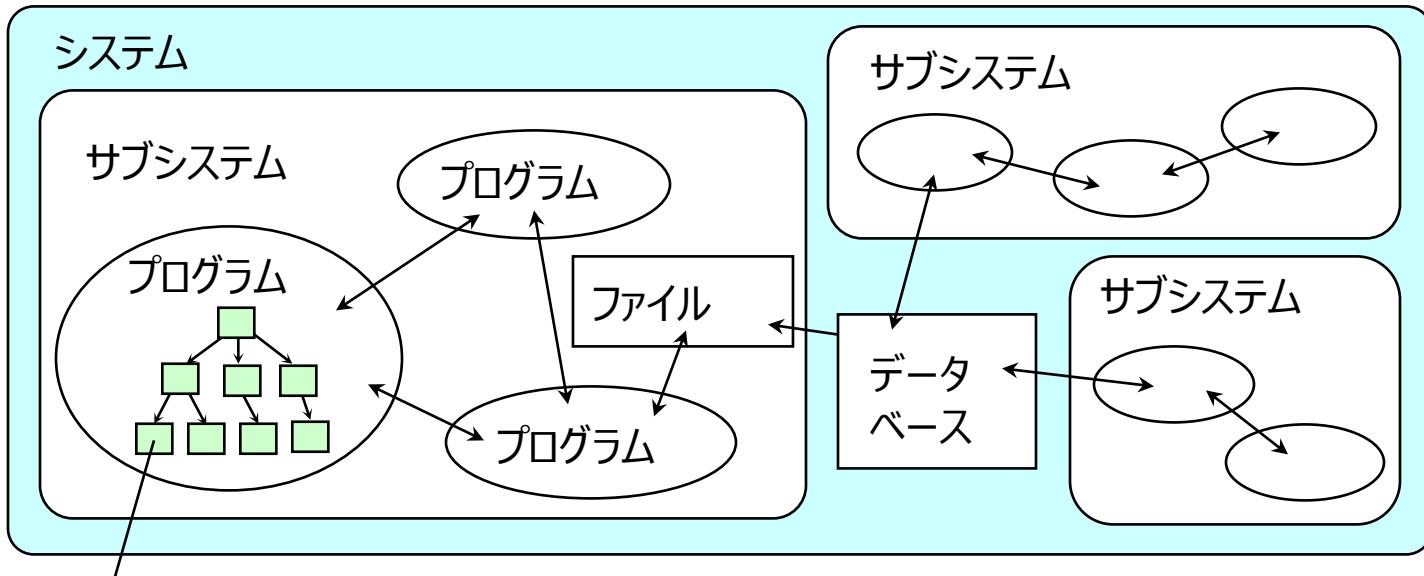
■ モジュール(module)

- 独立した機能あるいは関連する機能をひとまとめにした
プログラム単位
- サブルーチン、関数、手続き、クラス等 Subroutines

■ モジュール分割の利点

- 抽象化 : 詳細を把握せずとも利用可能
- 開発効率 : 並行して開発可能
- 再利用 : 既存のモジュールを再利用可能
- 変更容易性 : 変更範囲を局所化

モジュール



モジュール

- 複数の文で構成され、独立して識別可能な名前をもつ
- 決められたインターフェース(interface)を通してのみ呼出可能である

(例)

決められた関数名・引数で呼び出し

```
void func(int x){...} ← func(123);
```

実際にモジュールという言葉が指すものは言語等により異なる

講義内容

■ モジュール設計

- 概説
- ➡ ■ モジュール分割の評価基準
- モジュール分割技法
- オブジェクト指向におけるモジュール設計

モジュール分割の評価基準

■ 分割の観点

■ モジュールの大きさ

■ 例：モジュールを構成する文の数

■ モジュールの簡潔さ

はんよう

■ 簡潔 \Leftrightarrow 単一の機能のみ含む \Leftrightarrow 汎用化しやすい

■ 独立性の観点

■ 機能独立性

模块的独立程度可以由两个定性标准来度量，
这两个标准分别称为耦合(Coupling、結合度)和内聚(Cohesion、强度)
耦合衡量不同模块彼此间互相依赖(连接)紧密程度
内聚衡量一个模块内部各个元素彼此结合的紧密程度

1つの目的に沿った機能のみ提供し、他のモジュールとの相互作用が少ない

■ 情報隠蔽（カプセル化）

モジュールのインターフェースと実装を分離

■ モジュール強度（凝集度）(module strength/cohesion)

同じモジュール内に存在する構成要素の関連の程度

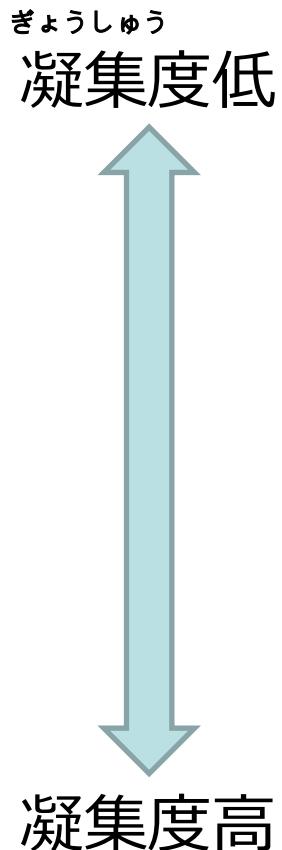
■ モジュール結合度(module coupling)

異なるモジュール間に存在する構成要素の関連の程度

モジュール強度(凝集度)

设计时应该力求做到高内聚。

- 暗号的強度、偶発的強度
(coincidental cohesion)
- 論理的強度(logical cohesion)
- 時間的強度(temporal cohesion)
- 手順的強度(procedural cohesion)
- 連絡的強度(communicational cohesion)
- 情報的強度(informational cohesion)
- 機能的強度(functional cohesion)



※モジュール強度の強さの順番は解釈によって変わることもある

モジュール強度(凝集度)

(1) 暗号的強度、偶発的強度(coincidental cohesion)

特定の機能を持たず、偶然に集められたモジュール

(2) 論理的強度(logical cohesion)

見かけ上は同一の機能を持つが、実際には多様な機能を集めたモジュール

(3) 時間的強度(temporal cohesion)

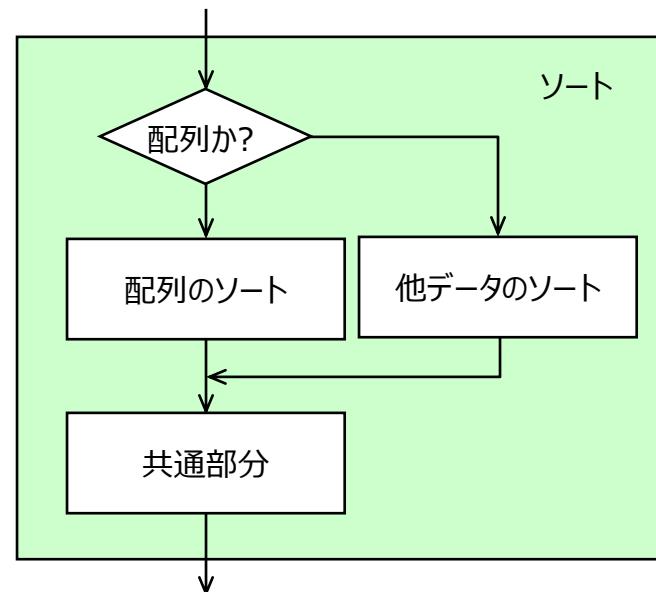
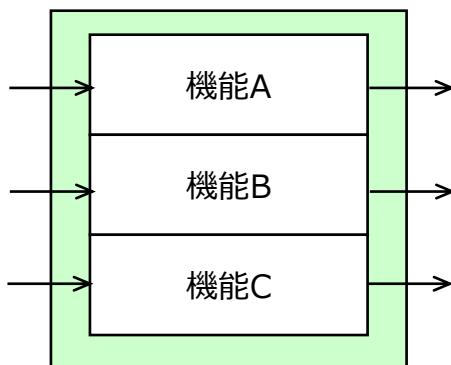
実行されるタイミングが近い機能を集めたモジュール

低内聚

偶然内聚(Coincidental Cohesion/Strength、偶発的凝集/强度): 一个模块任务间关系松散或者没有关系

逻辑内聚(Logical Cohesion/Strength、論理的凝集/强度): 一个模块完成的任务在逻辑上属于相同或者相似的一类

时间内聚(Temporal Cohesion/Strength、時間的凝集/强度): 模块包含的任务必须在同一段时间内执行 (如初始化模块)



暗号的

論理的

時間的

モジュール強度(凝集度)

(4)手順的強度(procedural cohesion)

逐次的に実行される関連のある機能を集めたモジュール

(5)連絡的強度(communicational cohesion)

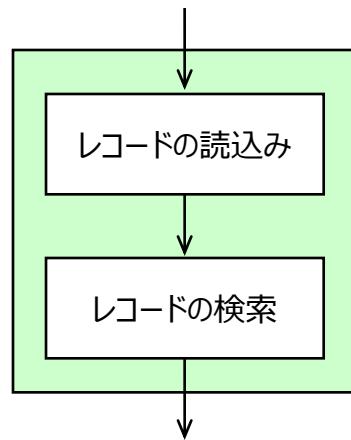
手順的強度で、

同じデータをあるいは出力する機能を集めたモジュール

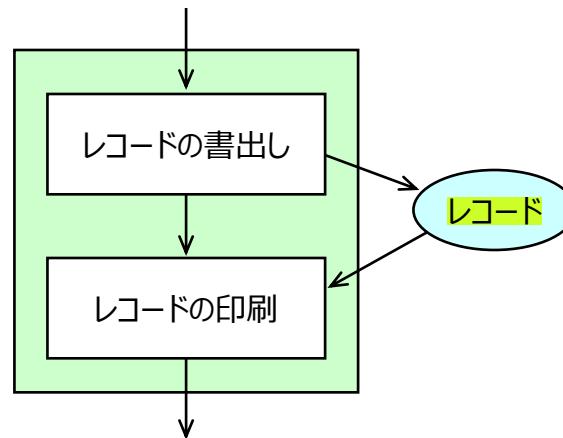
中内聚

过程内聚(Procedural Cohesion/Strength、手順的凝集/强度): 一个模块内的处理元素相关，并且以特定的次序执行,如将程序流程图中循环,判定部分划分成模块

通信内聚(Communicational Cohesion/Strength、通信的/連絡的凝集/强度): 模块中所有元素都使用同一个输入数据和/或产生同一个输出数据



手順的



連絡的

モジュール強度(凝集度)

(6)情報的強度(informational cohesion)

同じデータにアクセスする複数の機能を集めたモジュール
(オブジェクト指向のクラスが該当)

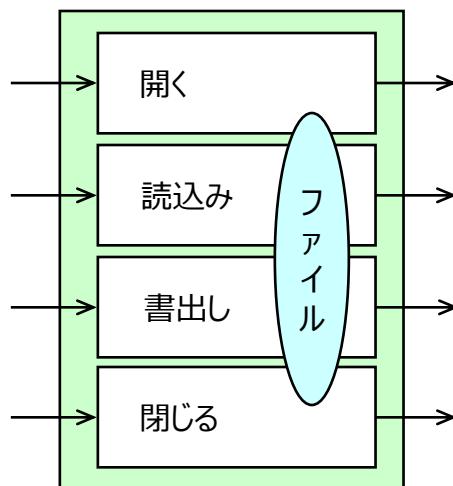
(7)機能的強度(functional cohesion)

単一の機能を実行するモジュール

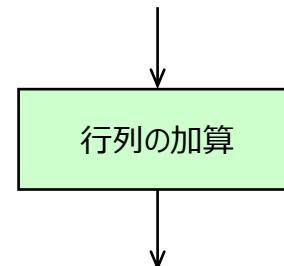
高内聚

順序内聚(Sequential Cohesion/Strength、順次的凝集/强度): 一个模块内的处理元素和同一个功能密切相关，并且必须顺序执行（前一个处理的输出是后一个处理的输入）

功能内聚(Functional Cohesion/Strength、機能的凝集/强度): 模块内所有处理元素属于一个整体，完成一个单一的功能



情報的



機能的

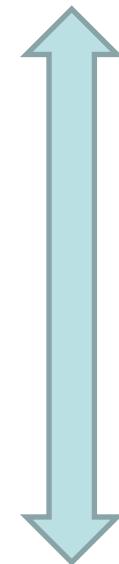
モジュール結合度

けつごう

在软件设计中应该追求尽可能松散耦合的系统。

- 内容結合(content coupling)
- 共通結合(common coupling)
- 外部結合(external coupling)
- 制御結合(control coupling)
- スタンプ結合(stamp coupling)
- データ結合(data coupling)

結合度高



結合度低

※モジュール結合度の強さの順番は解釈によって変わることもある

モジュール結合度

无耦合/非直接耦合(No Coupling/Indirect Coupling、非結合): 任何一个模块都独立工作 (现实系统中难以实现)

数据耦合(Data Coupling、データ結合): 一个模块访问另一个模块时, 彼此之间是通过简单数据参数 (不是控制参数、公共数据结构或外部变量) 来交换输入、输出信息的

标记/特征耦合(Stamp Coupling/Data-structured Coupling、スタンプ結合): 把整个数据结构作为参数传递, 而被调用的模块仅需使用其中一部分数据

(1) 内容結合(content coupling)

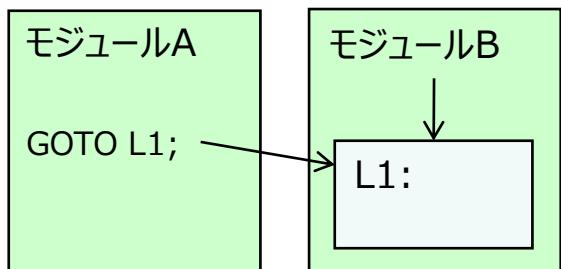
一方のモジュールが他方のモジュールの
内容を直接参照する

(2) 共通結合(common coupling)

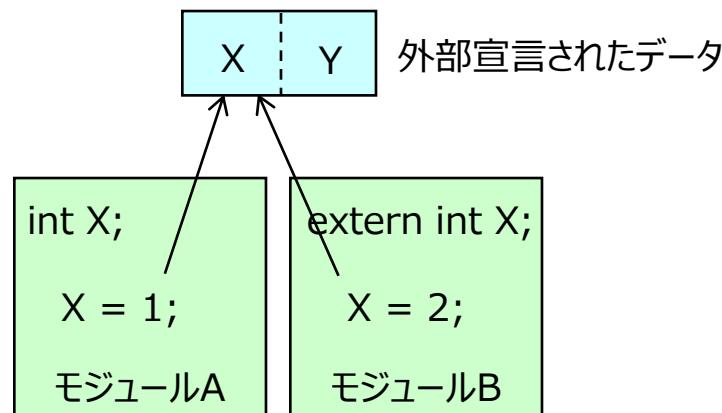
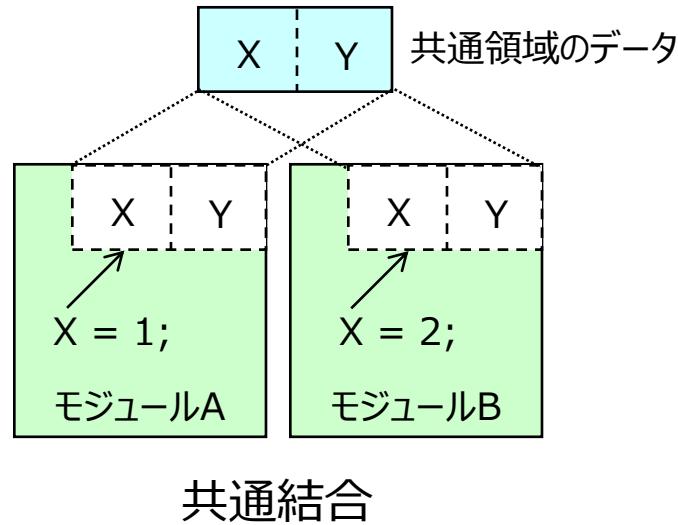
モジュール同士が共通データ領域にある
データを参照する

(3) 外部結合(external coupling)

モジュール同士が外部宣言された
データを共有する



内容結合



モジュール結合度

せいぎょ

(4)制御結合(control coupling)

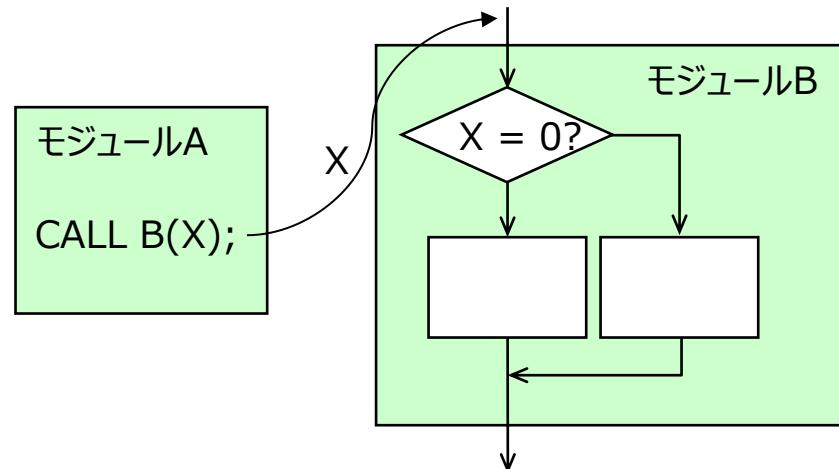
引数に基づいて条件判断を行う

(5)スタンプ結合(stamp coupling)

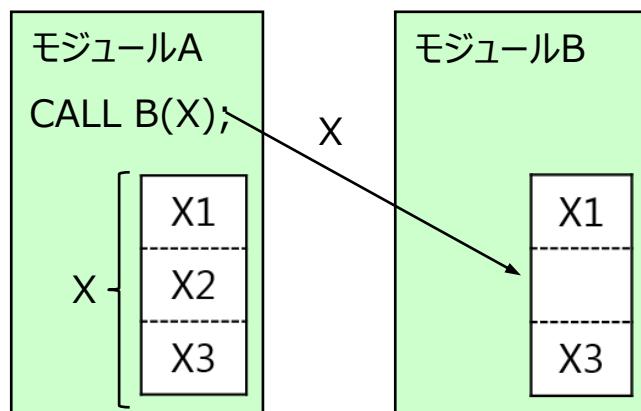
共通データ領域にないデータの構造体を受け渡す(不要なデータも含まれる)

(6)データ結合(data coupling)

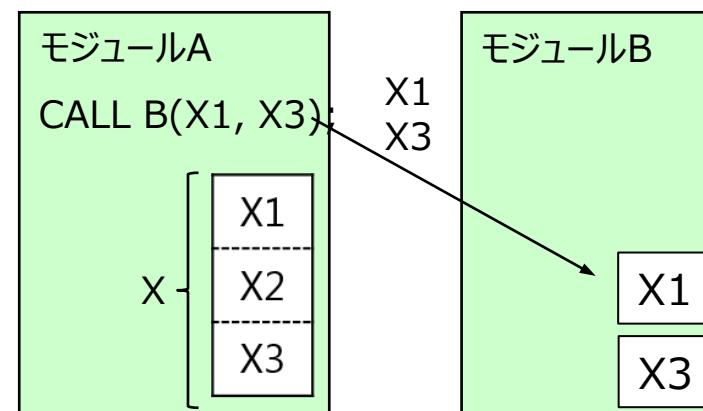
必要なデータだけを引数として受け渡す



制御結合



スタンプ結合



データ結合

確認問題

括弧内に選択肢があるものは正しいものを選択

- (1)は、独立した機能あるいは関連する機能をひとまとめにしたプログラム単位である。
- 一般的に、(1)は(2 複数・单数)の文で構成され、決められた(3)を通してのみ呼出し可能である。
- モジュール分割の利点としては、コードの(4 具体・抽象)化や、開発効率、再利用性、変更容易性の向上が挙げられる。
- 一般的に、モジュール結合度は(5 高い・低い)方が良いとされる。
- 一般的に、モジュール強度は(6 高い・低い)方が良いとされる。

確認問題

括弧内に選択肢があるものは正しいものを選択

- (1)は、独立した機能あるいは関連する機能をひとまとめにしたプログラム単位である。
1. モジュール
- 一般的に、(1)は(2 複数・单数)の文で構成され、決められた(3)を通してのみ呼出し可能である。
2. 複数 3. インタフェース
- モジュール分割の利点としては、コードの(4 具体・抽象)化や、開発効率、再利用性、変更容易性の向上が挙げられる。
4. 抽象
- 一般的に、モジュール結合度は(5 高い・低い)方が良いとされる。
5. 低い
- 一般的に、モジュール強度は(6 高い・低い)方が良いとされる。
6. 高い

講義内容

■ モジュール設計

- 概説
- モジュール分割の評価基準
- ➡ ■ モジュール分割技法
- オブジェクト指向におけるモジュール設計

モジュール分割技法

■ STS分割

- 源泉(source) : データ入力部
- 変換(transform) : データ変換部
- 吸収(sink) : データ出力部
- S,T,Sを担当するモジュールをそれぞれ配置
 - データフロー上のデータ変換を追跡し、入力データ・出力データとはみなせなくなった所で分割
- それを制御するモジュールを配置

数据流上的数据转换将在不再被视为输入数据/输出数据的位置进行跟踪和划分



モジュール分割技法

■ TR分割(トランザクション分割)

■ トランザクション処理：入力データに関する一連の処理

- データベースの一貫性を維持
- 途中で終わらせたり、一部だけ実行することはできない

■ 機能をトランザクションごとに分割

- 入力データの種類が複数あり、それぞれ異なる処理を行う場合に行われる

但是在数据流具有明显的事务特点时，也就是有一个明显的“发射中心”(事务中心、Transaction Center、トランザクションセンター)时，还是以采用事务分析方法为宜



共通機能分割

- 他の分割方法で機能分割を行う過程で、共通する機能を別個のモジュールとして抽出
- 共通のデータに関する機能を1個のモジュールとして抽出

变换分析设计是一个顺序结构，由输入、变换和输出三部分组成，其工作过程有3步：取得数据、变换数据和给出数据。
事务分析设计是将它的输入流分离成许多发散的数据流，形成许多加工路径，并根据输入的值选择其中一个路径来执行。
二者区别：变换分析设计适用于具有明显变换特征的数据流图，事务分析设计适用于具有明显事务特征的数据流图。

確認問題

- 以下の説明に合うモジュール分割技法を答えよ。
 - 複数のモジュールに共通する機能を別のモジュールとして抽出する。
 - 機能を、途中で終わらせたり、一部だけ実行することはできない単位に切り分け、モジュールとする。
 - データの入力、変換、出力それぞれを担当するモジュールを作成する。

確認問題

- 以下の説明に合うモジュール分割技法を答えよ。
 - 複数のモジュールに共通する機能を別のモジュールとして抽出する。**共通機能分割**
 - 機能を、途中で終わらせたり、一部だけ実行することはできない単位に切り分け、モジュールとする。**TR (トランザクション) 分割**
 - データの入力、変換、出力それぞれを担当するモジュールを作成する。**STS分割**

講義内容

■ モジュール設計

- 概説
- モジュール分割の評価基準
- モジュール分割技法
- ➡ ■ オブジェクト指向におけるモジュール設計

オブジェクト指向設計における モジュール強度・結合度

■ クラス

- 情報的強度：
同一データにアクセスする機能をまとめている
- 理解容易性、変更容易性のため、
複数のデータを1クラスにまとめる場合もある
→ 情報的強度よりも低下
- 2つのクラスがデータ結合であっても、
クラス間のやり取りが多いのは望ましくない

■ パッケージ package

- 複数のクラスをまとめている
→ これもモジュールの一種



オブジェクト指向では、
従来のモジュール強度・結合度とは違った指針が必要

クラスの設計原則

- 単一責任の原則 (SRP: single responsibility principle)
 - クラスを変更する理由は1つでなければならない
- オープン・クローズドの原則 (OCP: open-closed principle)
 - 拡張に対してオープンで、修正に対してクローズでなければならない
- リスコフの置換原則 (LSP: Liskov substitution principle)
 - サブクラスはそのスーパークラスと置換可能でなければならない
- 依存関係逆転の原則 (DIP: dependency inversion principle)
 - 上位のモジュールは下位のモジュールに依存してはいけない
 - 抽象は実装の詳細に依存してはいけない

依赖倒转原则(Dependency Inversion Principle, DIP): 抽象不应该依赖于细节, 细节应当依赖于抽象。换言之, 要针对接口编程, 而不是针对实现编程。
- インタフェース分離の原則 (ISP: interface segregation principle)
 - 強い関連性を持つインターフェースのみをまとめてグループ化しなければならない

パッケージの設計原則

- 再利用・リリース等価の原則 (reuse-release equivalence)
 - パッケージはリリースの単位で再利用されなければならない
 - 閉鎖性共通の原則 (common closure)
 - 1つの変更理由は單一パッケージに閉じ込められなければならない
 - 全再利用の原則 (common reuse)
 - パッケージ内の全クラスが再利用されなければならない
 - 非環式依存の原則 (acyclic dependencies)
 - パッケージの依存関係は無閉路有向グラフでなければならない
 - 安定依存の原則 (stable dependencies)
 - より安定しているパッケージに依存しなければならない
 - 安定度・抽象度等価の原則 (stable-abstraction)
 - 抽象的なパッケージほど安定していなければならぬ
- 1, 发布/重用等价原则 (The Release/Reuse Equivalency Principle) (REP)
创建一个包是为了方便别人重用。
- 2, 公共闭合原则 (The Common Closure Principle) (CCP)
按预期的修改将类分组，使因为同一原因而被修改的类被放在同一个包中。参照：SRP原则。
- 3, 公共重用原则 (The Common Reuse Principle) (CRP)
尽可能将被一个客户使用的包和被多个客户使用的包分开。参照：ISP原则。
- 4, 非循环依赖原则 (The Acyclic Dependencies Principle) (ADP)
包与包之间不要循环依赖，可以使用JDepend工具处理。
- 5, 稳定依赖原则 (The Stable Dependencies Principle) (SDP)
包不能依赖于不稳定的包。
- 6, 稳定抽象原则 (The Stable Abstractions Principle) (SAP)
稳定的包应该是抽象的，保证稳定的包易于被控制。

確認問題

- 以下の説明に合うクラスの設計原則を下記の語群から選べ。
 - クラスを変更する理由は1つでなければならない
 - サブクラスはそのスーパークラスと置換可能でなければならない
 - 強い関連性を持つインターフェースのみをまとめてグループ化しなければならない
 - 拡張を受け入れ可能である一方で、修正の影響が最小限になるようにしなければならない
 - 上位のモジュールは下位のモジュールに依存してはいけない

单一責任の原則
オープン・クローズドの原則
リスコフの置換原則
依存関係逆転の原則
インタフェース分離の原則

確認問題

- 以下の説明に合うクラスの設計原則を下記の語群から選べ。
 - クラスを変更する理由は1つでなければならない **单一責任の原則**
 - サブクラスはそのスーパークラスと置換可能でなければならない **リスコフの置換原則**
 - 強い関連性を持つインターフェースのみをまとめてグループ化しなければならない **インターフェースの分離原則**
 - 拡張を受け入れ可能である一方で、修正の影響が最小限になるようにしなければならない **オープン・クローズドの原則**
 - 上位のモジュールは下位のモジュールに依存してはいけない **依存関係逆転の原則**

单一責任の原則
オープン・クローズドの原則
リスコフの置換原則
依存関係逆転の原則
インターフェース分離の原則

参考文献

- 「ソフトウェア工学」
高橋直久、丸山勝久 著、森北出版、2010
- 「効果的プログラム開発技法」
國友義久 著、近代科学社、1979
- R.C. Martin et al., Principles of object oriented design
<http://wiki.c2.com/?PrinciplesOfObjectOrientedDesign>