ソフトウェア工学

10: ソフトウェア再利用 — オブジェクト指向プログラミングとデザインパターン

理工学部 経営システム工学科
庄司　裕子

今回のテーマ

ソフトウェア再利用 — オブジェクト指向プログラミングとデザインパターン

- オブジェクト指向プログラミング
- デザインパターン
- デザインパターンの例

2

復習

これまでは開発プロセスのうち、

- 要求定義/分析(R)フェーズ
- 設計(D)フェーズ

に注目し、

- 分析/設計手法
- UML

について学んだ

☞今回は実装(C)フェーズ寄りの技術を取り上げる

3

ソフトウェア再利用 — オブジェクト指向プログラミングとデザインパターン

⇨オブジェクト指向プログラミング

- デザインパターン
- デザインパターンの例

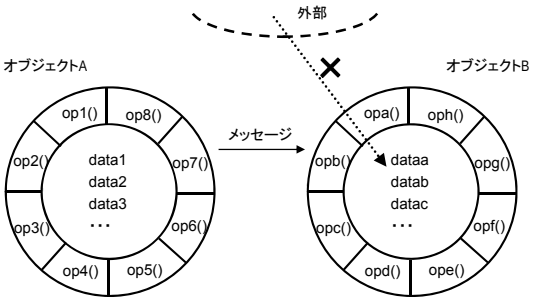
4

復習:オブジェクト指向のキーポイント

- カプセル化: オブジェクトにデータと手続きをひとまとめにする
- 継承: オブジェクトのクラスには階層構造があり、下位のクラスは上位のクラスからデータと手続きを受け継ぐが、再定義も可能
- 情報隠蔽: オブジェクト外部からは直接そのオブジェクト内のデータにアクセスすることはできない
- メッセージ送信: オブジェクトへのメッセージ送信でオブジェクト内部のデータの操作を依頼し、それに基づいたオブジェクト間の相互作用でシステムの機能を実現する

5

オブジェクト間の相互作用とデータの独立性



6

OOADとOOP

- オブジェクト指向プログラミング（OOP）のほうがオブジェクト指向分析/設計（OOAD）よりも歴史が長い
  - ☞ オブジェクト指向（OO）の考え方は元々プログラミング言語の研究から生まれたもので、それを上流工程に持ち込んだものがOOAD
- OOADで得られたモデル（設計）は、当然OOPで実装される
  - ⇨ 実装がスムーズ（モデル変換によるインピーダンス ミスマッチがない）
  - ☞ 上記のような歴史的経緯から、OOADモデルそのものがOOP言語の仕様にかなり引きずられている部分があるため、当然と言えば当然

7

OOADの結果からOOPへ

OOAD

- いわゆるシステム設計（場合によってはプログラミング設計の一部も）を提供
  - システム設計仕様書 ≒ クラス図＋相互作用図
  - ☞モジュール ＝ クラスまたはパッケージ

OOP

- システム設計をプログラム設計に詳細化
- コーディング
  - 個々のクラスの実装

8

OOPのカバレッジ

開発プロセスの観点では、OOPは

- 設計フェーズの一部（プログラム設計）
- 実装フェーズ（コーディング、単体テスト）

をカバーする

- オブジェクト指向プログラムのプログラム設計に有効なテクニックとして近年注目を集めているのがデザインパターン

9

ソフトウェア再利用 — オブジェクト指向プログラミングとデザインパターン

✓オブジェクト指向プログラミング

⇨デザインパターン

- デザインパターンの例

10

デザインパターン

- プログラム設計時によく現われる問題とそれを解決するクラス設計を再利用可能なパターンとして整理したもの
- Erich Gamma、Richard Helm、Ralph Johnson、John Vlissidesの4人が共著『Design Patterns: Elements of Reusable Object-Oriented Software』で提唱した23個のデザインパターンがきっかけとなって広まった
  - GoF（Gang of Four）パターンと呼ばれる

11

ソフトウェアにはパターンがある

- ソフトウェアが実現すべき機能や対象とする問題の構造には共通点が多いことがわかってきた

⇨共通する機能や問題を解決するクラス設計を、万人が利用できる明示的なパターンとして分類・整理すれば、クラス設計の再利用が図れ、生産性が向上する

12

パターンの主な有効性

- パターンが対象領域における重要な問題に対処するものであれば、それを利用することで、プログラム設計の複雑さが軽減される
- パターンがよく練られた汎用性の高いものであれば、それを利用することで、プログラム設計およびプログラミングの生産性が高まる
- パターンに一定の利用実績があり、十分安定に動作することがわかっている場合は、それを利用したプログラムの品質が高まる

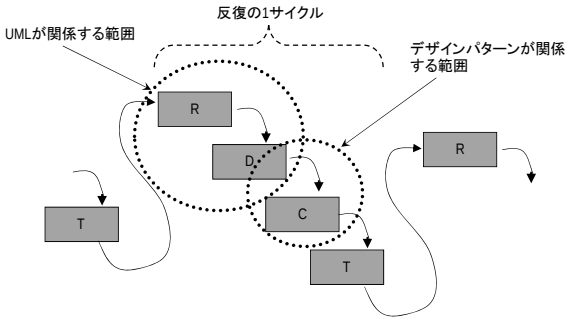
13

パターンの主な種類

- プロセス パターン
  - 開発プロセスのパターン
- アナリシス パターン
  - 分析工程で利用するパターン
- アーキテクチャ パターン
  - システム/アプリケーション全体の構造
- デザイン パターン
  - アーキテクチャより一段細かいレベル（マイクロアーキテクチャ レベル）のパターン
  - クラス設計までで、実装コードまでは定めない
- イディオム
  - 特定言語によるコーディング上のパターン

14

デザインパターンと開発プロセスとの関係



15

GoF パターン

- デザインパターンはGoFパターンに限らないが、現在最もよく理解され最も普及しているデザインパターンと言えば、GoFパターンである
- GoFパターンでは、設計でよく現われる問題とそれを解決するクラス設計が23組識別されている
  - これらは、多数のオブジェクト指向開発者のノウハウを選別・分類して、パターンとして明文化したもの
  - クラス設計者の事実上の「共通言語」となりつつある
  - それぞれに「Visitorパターン」などと名前が付いている

16

GoFパターンの分類

		目的		
		生成	構造	動作
適用対象	クラス	Factory Method	Adapter（クラス）	Interpreter Template Method
	オブジェクト	Abstract Factory Builder Prototype Singleton	Adapter（オブジェクト） Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

17

GoFパターンの分類基準

目的

- 生成
  - オブジェクトの生成
- 構造
  - クラスまたはオブジェクトの構成
- 動作
  - クラスまたはオブジェクトの相互作用と責務の割り当ての方法

適用対象

- クラス
- オブジェクト

18

パターンの適用対象による比較

- クラス パターン（主にクラスに適用されるパターン）
  - クラスとサブクラスの関係を扱う
    - 関係は継承を通じて確立される
    - 静的（コンパイル時に確定）
- オブジェクト パターン（主にオブジェクトに適用されるパターン）
  - オブジェクト間の関係を扱う
    - 動的（実行時に変更可能）

19

ソフトウェア再利用 — オブジェクト指向プログラミングとデザインパターン

✓オブジェクト指向プログラミング
✓デザインパターン
⇨デザインパターンの例

20

Template Method パターン

- スーパークラスとサブクラスで、以下のような役割分担を行う

スーパークラス（抽象クラス）では

- フック（処理の細部）の呼び出しを組み合わせて、処理の枠組みを記述（テンプレート メソッドの定義）
- フックの仕様（抽象メソッド）を宣言

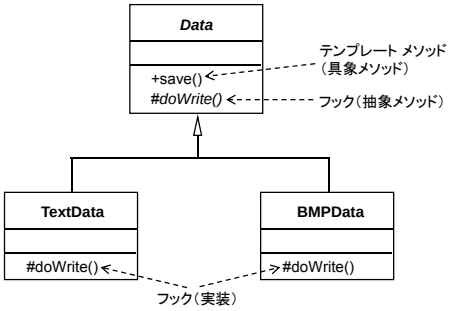
サブクラス（具象クラス）では

- フック（抽象メソッド）を実装

- 再利用性を保ちつつ、実質的なメソッド（フック）の実装をサブクラスごとに定義することで処理のバリエーションに対応できる

21

Template Method パターンの例



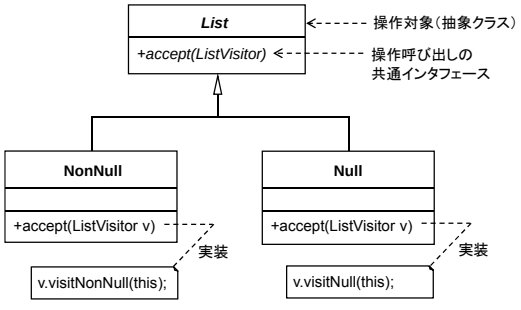
22

Visitor パターン

- あるデータ構造に対する操作を、そのデータ構造（操作対象）そのものから分離する
- データ構造の定義を変更せずに新しい操作を定義でき、データ構造の再利用性が高まる
  - 逆に、このような分離をしておかないと、新しい操作を定義するたびにデータ構造の定義に手を加えなければならず、再利用性が悪くなり、コードの安定性も損なわれるおそれがある
- 処理の分担
  - データ操作を表すビジター クラスを定義し、その処理の詳細をそのビジター クラスの処理メソッドとして定義する
  - 操作対象クラスの操作呼び出しインタフェース メソッドでは、渡されたビジター オブジェクト内の対応する処理メソッドを呼び出す

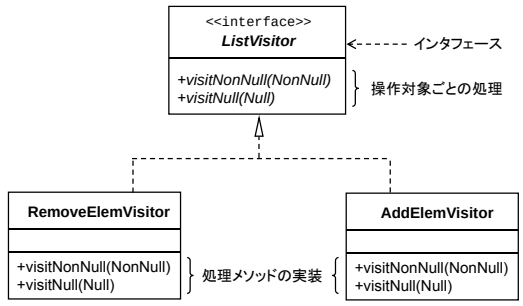
23

Visitor パターンの例（操作対象）



24

Visitor パターンの例（ビジター）



25

デザインパターンの利用法

- とりあえず、GoFパターンを出発点とする
- 各パターンの意味、用途、制限などをよく理解した上で、本当に適した使い方をする
  - 誤った適用は、適用しないことよりも悪い
- 汎用性やモジュール性を高めるために、かなり複雑なクラス構造になっていることが多いことをあらかじめ認識（覚悟）しておくこと
  - ハードルは高いが、それを越えたときに得られるものは非常に大きい
  - GoFの原著が難解であれば、巷に溢れるその他の解説本も参考になるだろう

26

参考文献

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, “Design Patterns : Elements of Reusable Object-Oriented Software”, Addison-Wesley Pub Co.
  - （邦訳:「オブジェクト指向における再利用のためのデザインパターン 改訂版」, ソフトバンク パブリッシング）
- 日経ソフトウェア（編）:「ゼロから学ぶソフトウェア設計」, 日経BP
- Perdita Stevens with Rob Pooley, “Using UML: Software Engineering with Objects and Components, Updated Edition”, Pearson Education
  - （邦訳:「オブジェクト指向とコンポーネントによるソフトウェア工学 — UMLを使って —」ピアソン・エデュケーション）

27