デザインパターン

中原 孝輔，冨田 龍太郎
Kosuke NAKAHARA，Ryutaro TOMITA

1　はじめに

近年，オブジェクト指向プログラミングが広く利用されている．オブジェクト指向プログラミングは，コードを再利用することにより生産性の高い開発を行うことができる．そのため，オブジェクト指向では設計者が再利用しやすいコード設計をすることが重要である．過去の設計者達はコードの再利用性向上のための問題に取り組んだ．その際に同じ解決策を採択した者が多かった．それら過去の設計者達のコードは再利用性の高いコードを設計するにあたって良い参考となる．そして，その解決策のノウハウを集めたものがデザインパターンである．

2　オブジェクト指向プログラミング

デザインパターンはオブジェクト指向と深く関係がある．オブジェクト指向プログラミングとは，カプセル化，継承，ポリモフィズムの三要素のうちいくつかの要素を持つプログラミング手法である．その中でカプセル化のみオブジェクト指向に不可欠の要素である．カプセル化とはクラスにおけるデータや処理を出来るだけ非公開にし，インプットとアウトプットのみに着目する設計概念である．また，オブジェクト指向プログラミングは，システム上における構成要素を分析し，その特徴をまとめてクラスとして定義する．その特徴とは，オブジェクトの性質や，システム上での振る舞いのことである．そして，オブジェクト指向プログラミングでは，各オブジェクトを用いることでシステム全体の処理を進行させる．

3　デザインパターン

3.1　概要

デザインパターンとは，過去のソフトウェア設計者のノウハウを蓄積し再利用しやすいように特定の規約に従ってカタログ化したものである．1995 年に Gang of Four(以下，GoF とする) という四人の設計者が 23 個のデザインパターンを定義し，デザインパターンをソフトウェア開発に導入した．[1] ソフトウェア開発におけるデザインパターンは，GoF によるもの以外にも AWS（Amazon Web Servis）デザインパターンやモバイルデザインパターンなど複数存在している．ただし，本稿では最も広く使われている GoF によるデザインパターンを扱う．GoF によるデザインパターンはオブジェクトの生成，プログラムの構造，オブジェクトの振る舞いの三つに分類することが可能である．本稿では，GoF の 23 のデザインパターンの中から各分類から一つずつ，実用性の高い Facade パターン，Singleton パターン，Iterator パターンを紹介する．

3.2　Facade パターン

Facade パターンとは, インターフェイスとなるクラスを追加してクラス呼び出しをすることにより，既存のクラスを複数組み合わせて使う手順を簡素化するデザインパターンである．システムは構築していくと，最初は小さいものでも次第に大きくなる．クラス数もともに増加し，相互に作用し複雑な構成になる．大きなシステムでは，クラスの関係性を正しく理解し，関係し合っているクラスを正しく制御するすることが必要である．Facade パターンを使うことにより，上記の問題を解決する．クラス呼び出しを行う処理を実行するインターフェイスであるクラスを用意する．このクラスを Facade クラスと呼ぶ．Facade クラスが存在することで個別に多くのクラスを制御せずとも，インターフェイスとなる Facade クラスに対して要求するだけで済む．以下，図を用いて Facade パターンの簡単な例を挙げる．

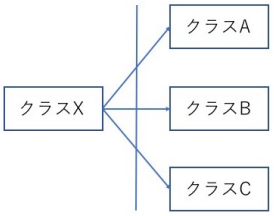

Fig.1　Facade クラスを用いない場合

Fig.1 のようにクラス X とクラス A, クラス B, クラス C があるとする．ここで，クラス X は目的を達成するためにはクラス A〜C を使用する必要があり，クラス A〜C の扱い方を熟知していなければならない．

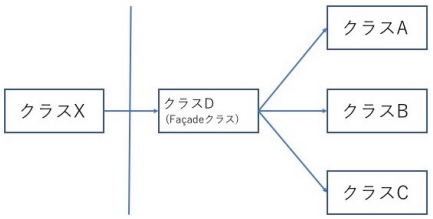

Fig.2　Facade クラスを用いた場合

しかし，FIg.2 のようにクラス D という Facade クラスを用意することで，クラス X はクラス D のみを扱うだけで良くなる．これが Facade パターンである．

3.3　Singleton パターン

Singleton パターンとは, クラスが生成するオブジェクトの数を一つに制限するパターンである．[2] 特定のクラスのオブジェクトが一つしか存在しないことを保証したい場

合に用いることで効力を発揮する．下記のソースコード 1 が，Singleton パターンの Java による実装例である．

ソースコード 1　Singleton パターンの Java による実装例

```
1   public class Singleton {
2     private static Singleton instance;
3     private Singleton(){};
4     public static ingleton getInstance(){
5       if(instance == null){
6         instance = new Singleton();
7       }
8       return instance;
9     }
10  }
```

上記コードの getInstance メソッドを工夫することにより Singleton パターンはオブジェクトの個数制限をするパターンとしても利用することができる．五つまでは新しいオブジェクトの生成を許すが，六つ目以降は既存のオブジェクトを返すというようなコード組むことができる．

3.4　Iterator パターン

Iterator パターンでは，データを持つオブジェクトに対して別のオブジェクトからデータを取得する時，取得する側のオブジェクトはデータを持つオブジェクトのデータ構造を知る必要がある．しかし，Iterator パターンではデータを持つオブジェクトのデータ構造は公開せず，hasNext() と next() という 2 つのメソッドのみを公開する．[2] hasNext() は，未取得のデータが残っていると true を返し，残っていなければ，false を返す．next() は，順番にデータを取得して返す．オブジェクト A からオブジェクト B に，hasNext() でデータの有無を問い合わせ，返り値が true なら next() でそのデータを要求する．

ソースコード 2　配列の要素を取り出す for 文の例

```
1   sum = 0;
2   for (index = 0; index < MAXSIZE; index++) {
3     sum += array[index];
4   }
```

上記のソースコード 2 は，変数 index をループカウンタとした for 文を用い，配列 array のすべての要素の合計値を変数 sum に代入するプログラムである．例えば，オブジェクト A とオブジェクト B があるとする．オブジェクト B の中にデータが格納されている配列があり，配列を使う for 文がオブジェクト A にあるとする．このとき，オブシェクト A からオブジェクト B のデータを呼び出そうとするとオブジェクト B の持つデータは配列形式であるということを認識している必要がある．それに伴い，配列の名前や先頭と末尾のインデックスも必要となる．このような構造ではオブジェクト B のデータ構造を変更すると，オブジェクト A まで変更する必要性が生じる．しかし，Fig.3 に示すように Iterator パターンを用いると，オブジェクト B のデータ構造が何であれ，hasNext() と next() だけでデータを取得できる．また，例えオブジェクト B のデータ構造が変わったとしても，オブジェクト A のプログラムは変更が不要である．

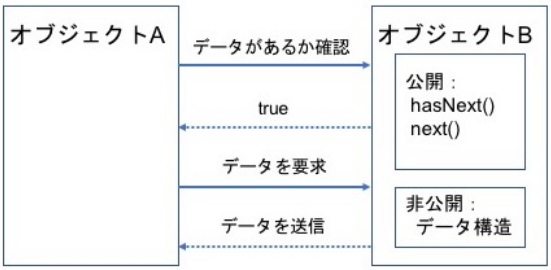

Fig.3　Iterator パターンにおけるオブジェクト間のやりとりのイメージ図

4　デザインパターンの利点・欠点

デザインパターンを用いると再利用性の高い柔軟な設計ができる．各パターンには多くのノウハウが凝縮されているため，これまで設計者の直感や経験などに依存していた設計が，デザインパターンを導入することで，初心者でも熟練者のノウハウを利用した再利用性の高い設計をすることが可能となる．

次に，技術者同士の意思疎通が容易になる．ソフトウェア開発におけるコミュニケーションを円滑にするには，問題に対する共通の認識を持つ必要がある．従来は，システム設計において意思疎通が難しかった．しかし，設計パターンに名前を付けることで，問題点や解決策を記会話の中で取り上げることができるようになる．

しかし，デザインパターンには利点だけでなく，欠点もある．デザインパターンは問題解決のためのノウハウ集であり，直面している問題に対する直接的な解決策を与えてくれるわけではない．無理やり適用しても効力は発揮せず，正しい使い方をしない限りはノウハウを教授することは出来ない．オブジェクト指向やプログラミングに対する基本的な知識と経験を持った上で，デザインパターンを深く学習し，何故これらがいい設計とされているのかを理解することが重要である．

5　今後の展望

GoF がデザインパターンを作り上げてから 20 年以上が経過した．現在，デザインパターンはこの GoF の思考を受け継ぎ，様々なデザインパターンが開発されている．スマートフォンのアプリケーション開発において用いられるモバイルデザインパターンや，AWS(Amazon Web Services) クラウドにおけるアーキテクチャの設計パターンを持つ AWS デザインパターンなどがある．今後も新しい技術の発展とともにソフトウェア開発は多様化すると予想される．また，その開発を支援するデザインパターンも生まれ，多様化するだろう．そのため，パターンの数は増大し開発においても頻繁に利用されるだろう．

参考文献

1) Erich Gamma,Richard Helm,Ralph Johnson ほか: 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, ソフトバンククリエイティブ株式会社 (2009).
2) 結城浩: Java 言語で学ぶデザインパターン入門, ソフトバンククリエイティブ株式会社 (2015).