□ 5. 深い継承階層を避けた方がよいという理由は何でしょうか？

5.7.3 あなたの意見

□ 1.「死んでいる」と感じる建物や構造を思い浮かべてください。それに存在しておらず、より「活き活きしている」と思える類似の構造に共通して存在しているものは何でしょうか？

□ 2.「パターンの真の力とは、あなたの思考レベルを引き上げる能力です。」これが真実であると感じた経験はあるでしょうか？ 例を挙げてください。

第6章

Facadeパターン

6.1 概 要

章の概要

それでは、Facadeというパターンから学習を始めることにしましょう。このパターンは、おそらく名前を知らなかっただけで、あなたも過去に実装したことがあるはずです。この章では、以下のことを解説しています。

- Facadeパターンとは何か、そしてその使用局面
- このパターンの鍵となる特徴
- Facadeパターンのバリエーション
- CAD/CAMの問題とFacadeパターンの関連

6.2 Facadeパターンの紹介

目的：統一された高レベルインタフェースの提供

GoFによれば、Facadeパターンの目的は、以下のようなものとなっています。

> **サブシステム内に存在している複数のインタフェース群に対する、統一したインタフェースを提供する。Facadeは高レベルのインタフェースを定義することにより、サブシステムを容易に使えるようにするものである**[*1]。

基本的に、このパターンは現在の手法よりも簡単な方法でシステムとやり取りを行う必要がある場合や、特定の方法でシステムを使用する必要がある場合（2D描画を行うために3D描画プログラムを使う等）に使用するものです。何らかの機能を実現するシステムの部分集合のみが必要となる場合、そういったやり取りの方法だけを取り出したものを作るわけです。

[*1] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Boston: Addison-Wesley, 1995, p. 185
邦訳は『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』（ソフトバンクパブリッシング刊）ISBN-4-79731-112-6

6.3 Facadeパターンの学習

動機の例：複雑な既存システムの使用方法を学習する！

私は過去に、機械製造関係の会社で仕事をしていたことがあります。仕事を始めた最初の日、プロジェクトの技術担当者は休みでした。このため、私はすることがなかったのですが、会社としては私に無駄な時間を費やさせたくなかったようです。後々役に立つ必要はなくてもいいから、何か作業をやらせておけという指示が飛んでいたのです！ あなたにもこんな経験はありませんか？

このため、プロジェクトメンバの1人が私のためにやることを探してくれました。彼女は、「今後の作業では、私たちがたまに使っているCAD/CAMシステムも関係してくるので、今からその勉強を始めておいた方がよさそうですね。じゃ、まずはここにあるマニュアルから読み始めてください。」と言い、私をドキュメント棚まで案内してくれました。そこには、積み上げると2.5メートルにはなりそうなマニュアル群が私を待ち受けていたのです。しかも、マニュアルは21cm × 28cmと大きい上、印刷されている字はとても小さなものだったのです！ それは明らかに巨大で複雑なシステムでした。

こういったことと手を切りたい

こういったシステムをプロジェクトのメンバ4〜5人に使用させる必要が出てきた場合、どうすればよいのでしょうか？ 使用対象者全員が、このシステムのことを学習しないといけないのでしょうか？ あるいは、くじ引きで**負けた人**が、他の人たちのためにシステムとのやり取りを行うルーチンを記述しなければならないのでしょうか？

くじ引きで負けた人は、他のメンバがこのシステムを利用する方法と、特定の目的に最適となるAPI（*Application Programming Interface*）を決定することになります。その後、メンバが使うインタフェースを保持した新規クラスやクラス群を作成することになるわけです。これにより、メンバ全員が複雑なシステム全体を学習することなく、新たなインタフェースを使えるようになるのです（図6.2）。

図6.1 高さ2.5メートルのマニュアル＝複雑なシステム

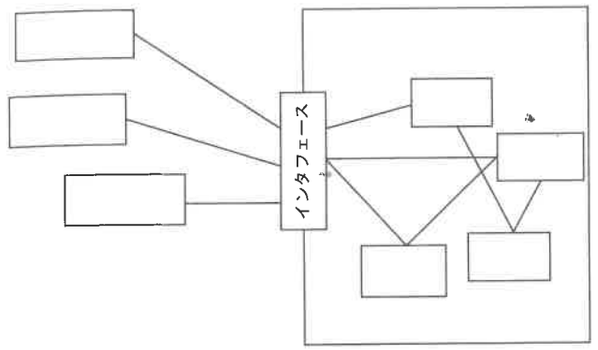

図6.2 サブシステムとクライアントを分離する

一部だけを用いて作業する

このアプローチがうまくいくのは、システムの持っている機能の一部だけを用いる場合や、特定の方法でやり取りを行う場合だけです。ただ、システムが提供している機能をすべて使用する必要がある場合でも、最初のうちは一部の機能だけを使用することで、設計の質を劣化させないようにできるはずです。

まさにこれがFacadeパターン

これがFacadeパターンです。このパターンにより、一部の機能だけ、あるいは特定の方法だけを使用して、複雑なシステムをより簡単に使えるようになるのです。手元に複雑なシステムがあり、そのうちの一部の機能だけが欲しい場合には、そのシステムを単純かつ使いやすい形にカスタマイズできるぴったりの解決策となるわけです。

ほとんどの作業は、元となっているシステムに任せることになります。Facadeは、理解しやすい形でメソッドのコレクションを提供する役割を担うわけです。これらのメソッドは、新たに定義した機能を実装するため、元となるシステムを使用することになります。

Facadeパターン：鍵となる特徴

目　的	既存システムの使用方法を簡素化したい。独自のインタフェースを定義する必要がある。
問　題	複雑なシステムの一部だけを使用する必要がある。あるいは、特定の方法でシステムとやり取りを行う必要がある。
解決策	Facadeによって、既存システムを使用するクライアント向けの新たなインタフェースを作成する。
構成要素と協調要素	このパターンは、複雑なシステムの使用が容易になるよう、クライアントに対して簡潔なインタフェースを提供する。
因果関係	Facadeは要求されたサブシステムの使用方法を簡略化して提供する。しかし、Facadeはすべての機能を提供するものではないため、ある種の機能はクライアントから利用できない可能性がある。
実　装	必要なインタフェースを持つ新たなクラス（またはクラス群）を定義する。この新規クラス（群）から既存のシステムを使用する。

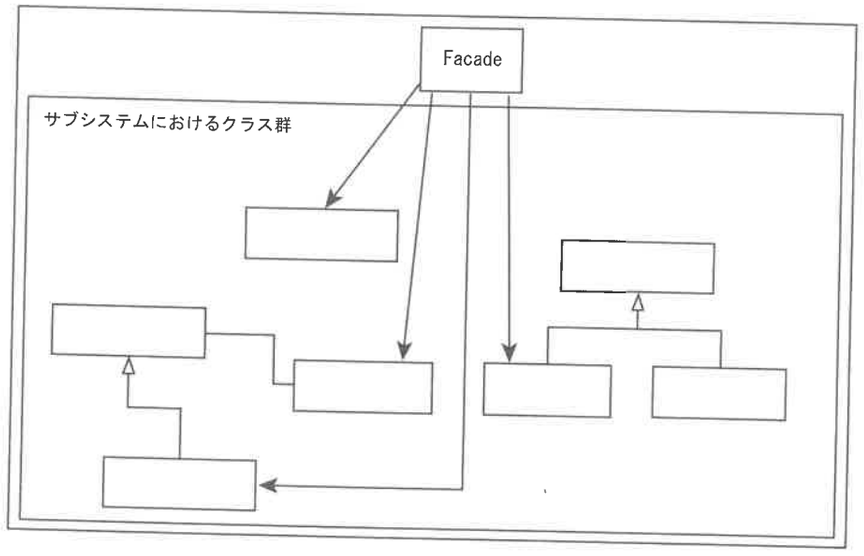

図 6.3　Facade パターンの一般的構造

6.4 フィールドノート：Facadeパターン

Facade のバリエーション：クライアントが作業する上で必要となるオブジェクトの数を削減する

Facadeは、メソッド呼び出しを使用してより簡潔なインタフェースを提供するという目的だけでなく、クライアントオブジェクトがやり取りする必要のあるオブジェクトの数を削減するという目的でも使用できます。例えば、**Database**（データベース）、**Model**（モデル）、**Element**（要素）とやり取りを行う **Client**（クライアント）というオブジェクトがあったと考えてください。**Client** は、まず **Database** をオープンし、**Model** を取得する必要があります。その後、**Model** に対してクエリを発行し、**Element** を取得します。最後に、**Element** に対して情報の照会を行うわけです。こういった場合、**Client** からのクエリを受け付ける **DatabaseFacade** を生成しておくと、話はずっとずっと簡単になるはずです（図 6.4 を参照してください）。

Facade に多くのオブジェクトの面倒を見させる

Facadeをステートレス（つまり、内部に状態を保持しない）にできるのであれば、Facadeオブジェクトを1つ用意するだけで、複数のオブジェクトから使用できるようになります。第21章では、Singletonパターンと Double-Checked Lockingパターンを使用して、こういったことを実現する方法を見ていただきます。

Facade のバリエーション：新たなルーチンを用いて既存の機能を補う

また、既存システム内の機能を呼び出すだけでなく、特定機能の呼び出し回数を集計するといった、何らかの新規機能を提供する必要があったと考えてください。この場合、既存システムの機能を呼び出すだけでは実現できません。

しかし、Facadeクラス内のメソッドに手を加えれば、新たな機能を付加することも可能になるのです。これは、新たな機能の追加によって拡張されていますが、Facadeパターンの一種です。機能の追加があっても、それは Facade が呼び出すものであり、クライアント側からそういった機能を呼び出す必要はいっさいないため、この場

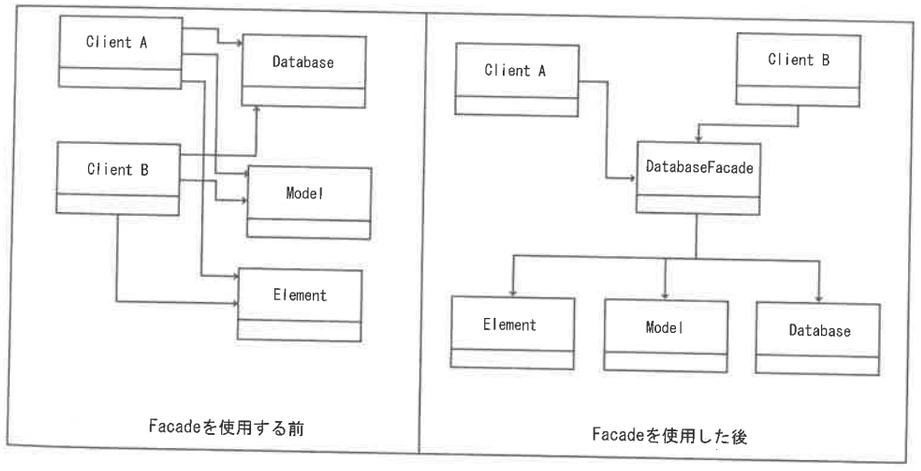

図 6.4　Facade はクライアントがやり取りするオブジェクトの数を削減する

合でも簡素化が主な目的であると考えられるのです。

Facadeパターンによって汎用のアプローチが用意されるため、それを開始点に考えていけばよいのです。Facadeパターンが Facade たるゆえんは、クライアントが既存システムのインタフェースを使用しなくてもよいよう、新たなインタフェースを生成するという点にあります。こういったことは、クライアントオブジェクトが既存システムにある全機能を必要としていないが故に可能となることなのです。

> **パターンとは汎用のアプローチを用意するものである**
>
> パターンは汎用のアプローチを用意してくれるものでしかありません。新たな機能を追加すべきか否かは、あなたが直面している状況に依存します。パターンは、あなたをスタート地点に立たせてくれるひな型であり、岩に刻みつけられた不変のものではないのです。

Facade のバリエーション：「カプセル化」階層

Facadeは、システムを隠蔽する、あるいはカプセル化するためにも使用することができます。つまり、既存システムを Facade クラスの `private` メンバとして保持すればよいわけです。このようにすることで既存システムは、Facade クラスと関連付けられているものの、Facade クラスのユーザからは見えなくなるのです。

システムのカプセル化が有効となるのは以下のような場合です：

- **システムの利用状況を追跡する**—システムに対するすべてのアクセスを Facade 経由にすることで、システムの利用状況を簡単に監視できるようになります。
- **システムを交換する**—将来的にシステムを交換する必要が出てくる場合があるかもしれません。そのような場合でも、既存システムが Facade クラスの `private` メンバになっていれば、システムへの影響を最小限に抑えられるわけです。それでも、それなりの修正作業は発生するでしょうが、少なくとも影響範囲は1ヶ所（Facade クラス）に絞り込めるのです。

6.5 CAD/CAM の問題と Facade パターンの関係

V1 システムのカプセル化

CAD/CAMにおける例を考えてみましょう。Facadeパターンは、**V1Slot**、**V1Hole** 等が **V1System** を使用する際にうまく利用できるはずです。第13章の「CAD/CAM問題をパターンによって解決する」でも、この手法を採用しています。

6.6 サマリ

この章のまとめ

Facade パターンは、既存システムのフロントエンドとして新たなインタフェース（見せかけ=*facade*）を置くため、この名前が付けられています。

Facade パターンは、以下の場合に適用することができます。

- 複雑な既存システムがあり、その全機能を使用する必要がなく、かつ、そのシステムの利用規則をすべて取り込んだ新規クラスを生成できる場合。必要な機能が既存システムの部分集合である場合、新規クラスの API はそのシステムの API よりもずっと簡単なものとなるはずです。
- 既存システムをカプセル化、あるいは隠蔽したい場合。
- 既存システムの機能を使用しながら、同時に新機能を追加したい場合。
- 新規クラスを開発するコストが、既存システムの使用方法を全員で学習するコストよりも安くつくか、将来の保守コストよりも安くつく場合。

6.7 練習問題

6.7.1 基 礎

□1. Facade を定義してください。
□2. Facade パターンの目的は何でしょうか？
□3. Facade パターンの因果関係とは何でしょうか？ 例を挙げてください。
□4. Facade パターンにおいて、クライアントはどのようにしてサブシステムと協調するのでしょうか？
□5. Facade パターンを使った場合、通常はシステム全体にアクセスできるようになるのでしょうか？

6.7.2 応 用

□1. GoF は Facade パターンの目的を「サブシステム内に存在している複数のインタフェース群に対する、統一したインタフェースを提供する。Facade は高レベルのインタフェースを定義することにより、サブシステムを容易に使えるようにするものである」と述べています。
  - この意味を説明してください。
  - 例を挙げてください。

□2. 以下はソフトウェアとは関係のない Facade の例です：アメリカ国内にあるガソリンスタンドの給油装置は、支払方法、給油する燃料の種類、広告表示等多くのオプションがあり、非常に複雑なものとなっています。このインタフェースを統一する方法の一つは、スタンドの係員に任せてしまうことです。州に

よっては係員の操作が義務づけられているところもあります。

- 実生活における Facade の例をもう一つ挙げてください。

6.7.3 あなたの意見

□ 1. 既存システムが提供していない機能を必要とする場合、Facade パターンを使用することができるでしょうか？

□ 2. Facade パターンを使ってシステム全体をカプセル化する理由は何でしょうか？

□ 3. Facade を用いて従来のシステムをカプセル化するのではなく、新たなシステムを作成するようなケースはあるのでしょうか？それはどういったものでしょうか？

□ 4. GoF がこのパターンを Facade と呼んだのは何故だと思いますか？これは行っていることに合った適切な名前でしょうか？その答えと理由を述べてください。

第7章

Adapterパターン

7.1 概 要

章の概要

では、次に Adapter パターンを見ていくことにしましょう。Adapter パターンは非常に一般的なパターンであり、他の多くのパターンとともに利用されます。

この章では、以下のことを解説しています。

- Adapter パターンとは何か、そしてその使用局面
- このパターンの鍵となる特徴
- このパターンを使用したポリモーフィズムの説明
- さまざまな詳細レベルに UML を使用する方法
- Adapter パターンと Facade パターンの比較を含む、今までの経験から見た Adapter パターンについての所見
- CAD/CAM の問題と Adapter パターンの関連

7.2 Adapter パターンの紹介

目的：新たなインタフェースを生成する

GoF によれば、Adapter パターンの目的は、以下のようなものとなっています。

> **あるクラスのインタフェースを、クライアントが望むインタフェースに変換する。Adapter によってクラス群は互換性のあるインタフェースを持つことになり、協調して動作できるようになる**[*1]。

この文章が基本的に意味していることは、オブジェクト自体の動作が望み通りであるものの、インタフェースが望み通りでない場合、新たなインタフェースを生成する必要があるということなのです。

[*1] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Boston: Addison-Wesley, 1995, p. 139
邦訳は『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』（ソフトバンクパブリッシング刊）ISBN-4-79731-112-6