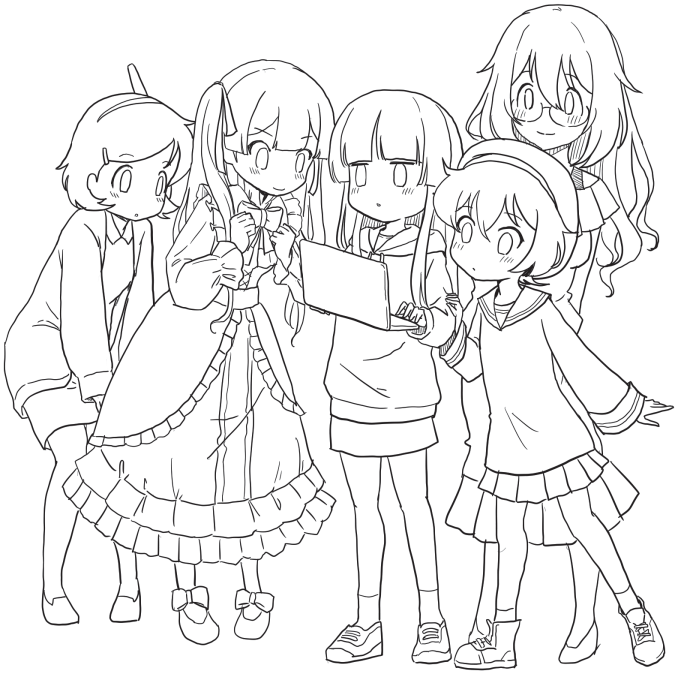


**●免責**

本書に記載された内容は、情報の提供だけを目的としています。したがって、本書を用いた運用は、必ずお客様自身の責任と判断によって行ってください。これらの情報の運用の結果について、技術評論社および著者はいかなる責任も負いません。

本書記載の情報は、2022年11月現在のものを掲載していますので、ご利用時には、変更されている場合もあります。

また、ソフトウェアに関する記述は、特に断わりのないかぎり、2022年11月現在でのバージョンをもとにしています。ソフトウェアはバージョンアップされる場合があり、本書での説明とは機能内容や画面図などが異なってしまうこともあり得ます。本書ご購入の前に、必ずバージョン番号をご確認ください。

以上の注意事項をご承諾いただいたうえで、本書をご利用願います。これらの注意事項をお読みいただかずに、お問い合わせいただいても、技術評論社および著者は対処しかねます。あらかじめ、ご承知おきください。

**●商標、登録商標について**

・本書に登場する製品名などは、一般に各社の登録商標または商標です。なお、本文中に™、Ⓡなどのマークは特に記載しておりません。

# はじめに

この本は、プログラミング言語の入門を済ませたけれど、もっと良く書きたいと思っている、中級者へのステップアップを目指す方のための本です。

プログラムというものは不思議なもので、習熟して思いどおりに書けるようになってくると、最初の「自分はこれで何でもできるんじゃないか」と思えた頃と打って変わって、だんだんと自分の無力さが見えてきます。

安心してください。自分が初心者の域からいつまでも出られないときちんと思えるのは、成長の証拠です。諦めずに考え続ければ、いつしか気づかないうちに中級者に手が届くようになってきます。そして、自分の頭で「プログラミング言語を覚えることよりずっと難しくて厄介な問題は、自分たちが作った設計の方にあるんじゃないか」といった考えが湧き上がってくるようになれば、その時こそ、大きく階段を踏み上がるチャンスです。

「プログラミング言語を正しく書くだけなら、文法エラーをなくして動作が間違ってなければよいとわかるんだけど、うまくプログラミングする方法＝ソフトウェア設計って話になると、何が正しいのかがわからないよ」「けれど、必要以上にややこしくなってすぐバグるのは、単にコーディング技法があるかないかじゃなくて、たぶんもっと大枠の設計がまずいんだろうというのはわかる」そんな悩みを抱えながらも、大きな一歩を踏み出せず、悶々とした気持ちで仕事をしている、あるいはこれから仕事をしようとしているプログラマーの方は、（はっきりそう意識できていなくても）とても多いと思います。自分がそんな悩みを抱えていた頃に、「これを先に知っていたら早かったのに」と思ったことを、この本の筋書きの中に入れ込みました。

本書の筋書きは、いまや当然となった、ごくありふれたソフトウェア工学の王道です。それぞれの専門書に比べると、決して網羅性が高いとは言えません。しかし、知識のとっかかりとして、そして何より、取り上げた各技法のつながりを理解して、あるひとつのソフトウェア設計の体系を認識するのに、本書がとても役立つと思います。

本書の中心を貫くのはたったひとつの価値観です。便宜上それはオブジェクト指向と呼ぶしかないのですが、まだ内容を読み進めていない方が想像するオブジェクト指向とは、少し違うかもしれません。回りくどい言い方をすると、「ソフトウェア工学のうち、かつてオブジェクト指向というトレンドをベースとして発達してきた領域」です。

フレデリック・P・ブルックスの名著で「狼男を倒す銀の弾丸はない」と主張

した『人月の神話』の第2版（1995年）に書き加えられた章に、「オブジェクト指向は真鍮の弾丸かもしれない」という言葉が記されています。現代のプログラマーは、オブジェクト指向言語を使いこなせば無条件にソフトウェアが作りやすくなるなんてことを信じる人はいないでしょう。ただ漫然とオブジェクト指向であるだけでは意味がありません。ブルックスはなぜオブジェクト指向は真鍮（銀の代わりに使われる庶民的な金属）になると予測したのでしょう？この真鍮を妥当な効果のある弾丸にするとは、いったいどういうことなのでしょうか？　それ自体に明確な言葉はないけれど、オブジェクト指向で語られる「原則」と「技法」と「パターン」を通して、「それ」をひとつ、この本から見つけ出してください。

　各セクションを理解できると、思わずクスッとなるかわいい挿絵が待っていますよ。がんばって読み進めましょう。

2022年秋
田中ひさてる

# 謝辞

私がこの本を書くにあたって、多くの方にチャンスと助力をいただきました。みなさんのおかげで、より良い（と私が思っている）知見を世に広めたいという願いが形になりました。

まずは、冗談のつもりで描いた絵を気に入ってくれためもりー（@m3m0r7）さん。「趣味でPHPを使ってJavaのバイトコードを動かす若き女性プログラマー」って……こんなすごい人が開発者コミュニティに現れたんだと、その活躍に感銘することがなかったら、めもりーちゃんというキャラクターは生まれてもいませんでした。ありがとうございます。

私の「Qiitaアドベントカレンダー1人で24日マンガ連載」なんていう冗談みたいなネタきっかけで、雑誌に誘っていただいたり、執筆を提案してくださったり、そしてなにより、入れたい内容が後から増える間、制作を待ってくださった、技術評論社の池本さん、ありがとうございます。

東京工業大学で研究室を発足され、京都大学に移られた首藤先生に一読していただき、文章表現や誤りをたいへん細かく指摘していただきました。ありがとうございます。大学の先生の目から見て、主張そのものが大きく事実に反するような点を指摘されなかったことで、自信を持てました。さらに、本書のところどころに、ここはいいことが書いてあるなと感想を持ってもらえたのが、とてもうれしかったです。

この本の原稿は、まだ紙面になっていない段階で、私が知りうるかぎりもっとも厳しく見てくださる方に、目を通していただきました。『プロになるJava』（弊社刊行）の著者の一人で、「（軽率に）オブジェクト指向（をありがたがるのを）やめろ」といった批判的なスピーチをしておられる、きしだなおき（@kis）さん。問題意識をともに議論してくださったこともですが、何より、この人にこの本の内容を否定されなかったということが、もっともうれしかったです。

子育てに忙しいママさんプログラマーでありながら、コミュニティイベントではしっかりとした設計哲学を語ってくれる、国内Symfonyユーザーとしてたいへん尊敬するななうぇぶ（@77web）さん。お願いしたコードの「らしさ」チェックだけでなく、解説文への疑問も遠慮なく出してくださって、ありがとうございます。

お名前は差し控えさせてもらいますが、私の事実誤認に気づかせてくださったあるベテラン開発者の方にも感謝を述べたいです。大きな誤解につながる可能性から、望まない不毛な議論につながってしまうところでした。

ほか、執筆中にTwitterに漏れ出てしまう私の思いにリアクションをくだ

さった、数多くの開発者コミュニティのみなさん。ペースダウンすることもあったけど、ほぼ毎日めもりーちゃんと仲間たちを見てくださったフォロワーのみなさん。本当にありがとうございます。

# 本書の読み方

## 「ちょうぜつ」について

本書のタイトル「ちょうぜつ」は、「ちょうぜつエンジニアめもりーちゃん」に由来します。めもりーちゃんは、著者が1コマ漫画として日々 Twitter連載しているシリーズ（数えてみたら1000回を超えていました）のシリーズタイトルです。カラーページにあったものがSoftware Design誌で連載されていたのを知っている方もいるかもしれません。

平仮名で「ちょうぜつ」と表記しているのは、「他の職種から見ると不思議に見えるけど、本職のITエンジニアにはありがちな」という意味合いです。

本書で扱うソフトウェア設計のアイデアも、慣れていないと驚くかもしれないけれど、本職のエンジニアにとっては、よく知られた普通の内容にすぎません。この本は、誰も知らない目新しい技法を期待した方には申し訳ないぐらい、枯れた方法論の掘り下げを徹底した内容の本になっています。

## ソフトウェア設計について

システム開発には、じつにさまざまなところに「設計」（アイデアの取捨選択に対する最適な判断）という言葉が登場します。設計と言うと、ハードウェアとソフトウェアと運用をすべて含んだシステム設計から、データベースやネットワークの設計、ユーザーに提供する機能を考えるアプリケーション設計まで、扱う広さもさまざまです。プログラムコード中の変数名を決めることさえ、設計と言えるかもしれません。

本書では、「ソフトウェア設計」のうち、どんなアプリケーション機能を提供するかと、具体的にどんなミドルウェアとアルゴリズムで機能を実現するかを除いた領域、ソフトウェアのアーキテクチャ作りに着目します。

ソフトウェア設計にとって、プログラミングはたいへん大きな影響を持っています。プログラムの構成は、下手を打ったときの周辺ダメージがとても大きいけれど、逆に、うまくやれば、他の意思決定をスムーズに押し進める、良き

媒体にもなります。

安定して良いプログラムを書き続ける支えとなるアイデア、広く知られたコード構成ノウハウ、といったものを通じて、現在のソフトウェア設計（アジャイル登場以後）が共通して持っている、前提知識の感覚を得るのが本書のねらいです。

## 使用しているプログラミング言語について

サンプルコードの記述にはPHPのバージョン8.0以上を使用します。PHPを使うといっても、とくに固有の言語機能に依存する記述はしません。なるべく他のプログラミング言語でも読み替えができるように表記しています。以下、一部PHPであるがゆえの注意点です。

- 現在のPHPは入出力変数の型宣言が推奨される言語ですが、型を宣言しない記述も可能です。本書では、型の重要性が低く、紙面上の可読性を優先するところで、型を省略して書くことがあります。

- PHPはPythonやJavaScriptと同様**$this**を省略できない言語です。C++, Java, Ruby, に慣れているとうるさく感じますが、ご容赦ください。

- PHP 8.0以上には、Constructor Property Promotionという言語機能があります。これは、クラスのプロパティ宣言とコンストラクタメソッドの引数の宣言を同時に行う記法です。コードを短く書くために、本編内のサンプルコードはすべてこの記法で記述します。

```
class Foo
{
    protected Bar $bar;
    public function __construct(Bar $bar)
    {
        $this->bar = $bar;
    }
}
```

↑↓ **同じ**

```
class Foo
{
```

```
    public function __construct(
        protected Bar $bar
    ) { }
}
```

- PHP自体の型にはテンプレートやジェネリクスといった機能がありません。PHPには「文字列の配列」といった文法上arrayとしか表現できない型について、決まった書式のコメントアノテーションで表す習慣があります。

```
class Parent
{
    /** @var Child[] */
    private array $children = [];
    /**
     * @param string[] $childNames
     */
    public function __construct(
        protected array $childNames
    ) { }
}
```

- アクセス指定子を持つ言語の間には、その意味に微妙な違いがあり、用法について議論が分かれます。本書ではメソッドとプロパティのアクセス指定子を以下のポリシーで使い分けます。

  - public：外部からアクセス可能な設計単位
  - protected：設計単位として存在はするが外からはアクセス不可（継承関係は除く）
  - private：将来設計単位として存在する保証のない一時的なもの

なお、PHPのクラスのアクセス指定には現在（提案が未採択で）、パッケージスコープでの区別がないため、他の言語に読み替える場合ご注意ください。PHPのクラスはすべて**public class**、あるいは**export**付きになってしまいます。内部利用に制限する場合は、コメントで**@internal**アノテーションを付ける習慣になっています。

PHPは決してプログラミング言語として最初から美しく設計された汎用言語ではありません。が、長い歴史の中で、時代に応じて必要な言語機能を徐々に獲得してきました。現在のPHPは「なぜかオブジェクト指向だけ一人前にできる」不思議な言語になっています。それしか上手にできないかもしれないけれど、だからこそ本書のテーマとの相性が際立ちます（めもりーちゃんの生まれたきっかけがPHPコミュニティだったというひいき目は、ええ、ちょっとあります……）。

## 構成について

本書は、セクションごとに独立したトピックを解説するスタイルの技術書とは少し異なり、全体で1本の物語のようになっています。個々の解説は個別に読んでも役立ちますが、すべてをつないだときに、大きなひとつの意味が見えてきます。

本書の流れは全体的に、歴史的な順序、あるいは一般的に「簡単」から「難しい」に流れる説明の順序とは、逆順になっています。過去のソフトウェア工学の偉人たちは、読者が、十分な経験者と同じだけの目的観を持っていると考えて、かなり高度なところから説明をする本を書いてきました。しかし実際には、読者との間に、より基礎的な意識の食い違いがあり、いつも正しく理解されないままになってしまいました。時とともに入門者が何を理解できないのかが明らかになることで、達人の感覚の中にしかなかった原理は、改めて少しずつ明文化されていきました。

本書では、そうして積み上がった知見を、最速で理解できる順序に並び替えてあります。ぜひ、順番に読み進めていってください。

### ■ 第1章　クリーンアーキテクチャ

ここでは、より良くソフトウェアを作るとき重要なスローガンと、必須キーワードを説明します。ソフトウェアの全体的な設計イメージをつかむのが目的です。どうやってそんなものを作るかという課題には、まだ踏み込みません。

### ■ 第2章　パッケージ原則

ソフトウェア部品をどのようにまとめて整理していくかに関して、定番となっている基本原則を紹介します。この原則を前提にしなければ、後の内容の必然性がわからなくなります。すべて理解できなくても、必ず目を通して、どういうことなのか疑問を持ち続けながら、次へ読み進めてください。後の内容は巨大な答え合わせになってきます。

### ■ 第3章　オブジェクト指向

本書で言う「オブジェクト指向」というキーワードが意味する範囲を明確にします。人によって同意できるかどうかに差はあるかもしれませんが、そういうものだと前提を置くことで、後の説明がしっくりきます。

### ■ 第4章　UML（統一モデリング言語）

この章は、すでに知っていれば、いったん読み飛ばしてもかまいません。知っているつもりでも、もしかしたら新しい発見があるかもしれないので、読み飛ばした場合は後で読んでみることをオススメします。

### ■ 第5章　オブジェクト指向原則 SOLID

パッケージ原則を支えるためのにオブジェクト指向の特徴を使うとき、絶対に欠かせない重要な原則、SOLIDを解説します。ここの内容でようやく、クリーンアーキテクチャの説明が完成します。また同時に、新たな疑問を保留するかたちで、次からの実技的な手法へ続きます。再度、後の内容で答え合わせをしていきます。

### ■ 第6章　テスト駆動開発

実際にPHPで単体テストを進めながら、単体テストの意味、テスト駆動開発、そして、テスト駆動開発を用いた設計プロセスを説明します。ここまでの原則をベースに、実際にプログラムコードで設計が書き上がっていく、たいへん面白いセクションになっています、

### ■ 第7章　依存性注入

依存性注入という考え方、DIコンテナと呼ばれる技術の使い方を通じて、なぜ原理原則に忠実なオブジェクトが大きなソフトウェア作りに役立つのかを、体験的に理解していきます。

### ■ 第8章　デザインパターン

原理原則として最短コースで進めてきた内容が、実際にどんな設計要素になるかを、いわゆるGoFのデザインパターンを通して確認していきます。この章はとくに、原典から大胆に構成を変えたり、省略したりしています。パターンそのものをマスターするのが目的ではなく、パターンを題材にした原則の応用方法を知るのが目的だからです。

### ■ 第9章　アジャイル開発

駆け足でのドメイン駆動設計、より深堀りしたアジャイル開発宣言、の二本立てで、本書の各技法を、そもそも何のために学んできたのかを振り返ります。プログラムコードの書き方には言及しませんが、非常に重要な事実が書いてあるので、締めくくりとして最後まで読んでみてください。

## 登場人物紹介

### めもりーちゃん

ねこになりたいプログラマー。量産型の3倍の速度で実装できる。時間が余るのでいたずらばかり考えている。

### ゆにっとさん

オブジェクト指向とガンプラが好き。見た目は子供、頭脳はおじさん。マリンセーラーは海軍の服なのでかっこいいと思っている。

### こみっとさん

めもりーちゃんの上司さん。バージョン管理とマネジメントがおしごと。全コードレビューしてくれるやさしいお姉さん。

### そけっとさん

インフラ担当のゴスロリさん。かわいいお洋服と配線が大好物。表の顔はシステム仮想化、裏の顔はアニメの仮装化！？

### くるみクン

AIの新人エンジニアちゃん。前向きでがんばり屋さん。でもAIなのでわりと天然(?)なところも……

# 目 次



## 第1章 ›› クリーンアーキテクチャ

## 第2章 ››

# パッケージ原則



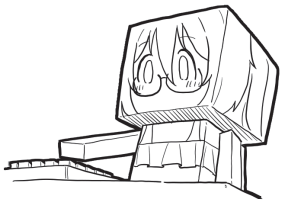
## 第3章 ›› オブジェクト指向



## 第4章 ›› UML（統一モデリング言語）

## 第5章 ››
# オブジェクト指向原則 SOLID





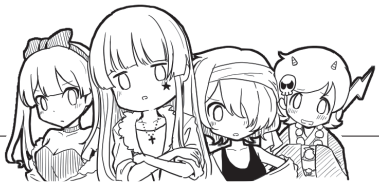
## 第6章 ››

# テスト駆動開発

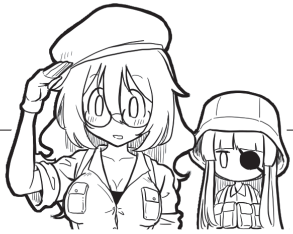


## 第7章 ›› 依存性注入





## 第8章 ›› デザインパターン



## 第9章 ›› アジャイル開発