設計の品質向上

FU14 ソフトウェア工学概論

第10回

吉岡 廉太郎

前回の内容

- オブジェクト指向設計
  - 課題のとらえ方・分析の仕方
  - 実績のある効果的な手法
  - オブジェクト指向設計の用語と概念
    - オブジェクト、クラス、インタフェース、インスタンス変数
    - 継承と合成
    - 代替可能性、デメテルの法則、依存関係逆転の法則
- UML
  - オブジェクト指向設計で用いる記法
  - 多数の図の集まり
  - 開発工程と各図の関係を理解する

今日の内容

- デザインパターン
  - 設計の“定石”（もっとも一般的な設計）
  - 再利用で効率と品質を高めることができる
- その他の設計要素
  - データの設計、例外処理の設計、UIの設計
- 設計書
  - プログラミングに必要な情報を明らかにする

デザインパターン

9.4.2

- 頻繁に直面する設計上の課題に対して、繰り返し使われる実績のある理想的な解法
  - 設計のテンプレート・ひな形
  - 対象に応じて変更、適応させて利用する
  - 部品ではない（＝そのままは使えない）

- 効果
  - 設計の原則に則った設計を促す
  - 理解性、拡張性、再利用性が良い
  - 開発経験の少ない人と、経験が豊富な人の差を埋める

デザインパターンの種類

- GoFが最初に提案し、その後追加・拡張されている
  - 当初提案されたパターンは23個

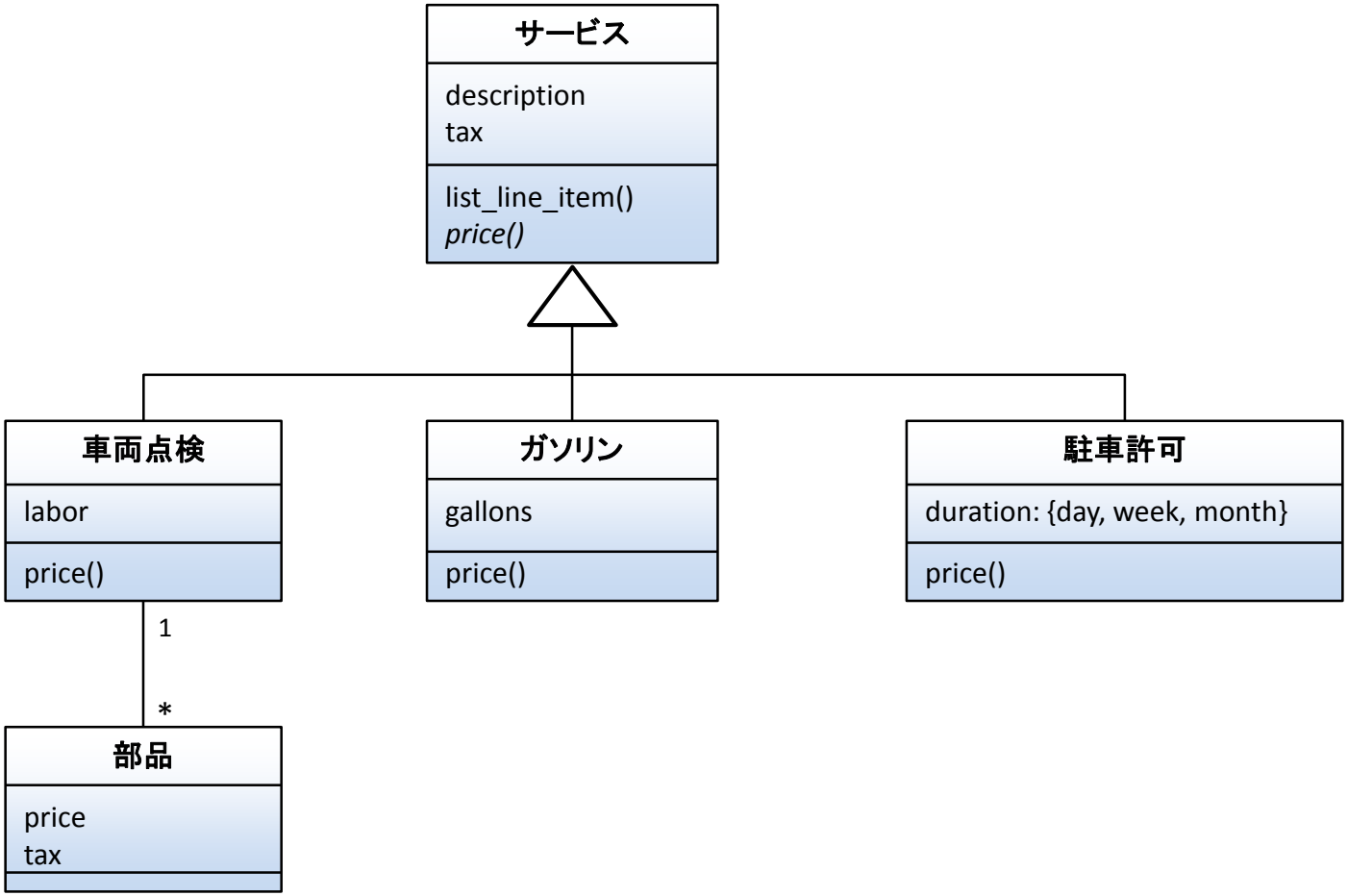
- 用途別に多くの種類がある
  - Template Method パターン
  - Factory Method パターン
  - Strategy パターン
  - Decorator パターン
  - Observer パターン
  - Visitor パターン、など

参考「**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**, Erich Gamma 他, ソフトバンククリエイティブ 1999」

Template Method パターン

- 概要
  - 同じ親クラスを持つサブクラスが複数ある場合に、コードの重複を少なくする
- 用途
  - 複数のサブクラスで、同じではないが似たような実装を必要とする場合に用いる
- 解決法
  - 重複コードを抽象クラスにまとめる
  - 抽象クラスをテンプレートとして共通する処理を実装する
  - 実装が分かれる具体的な処理は抽象宣言する

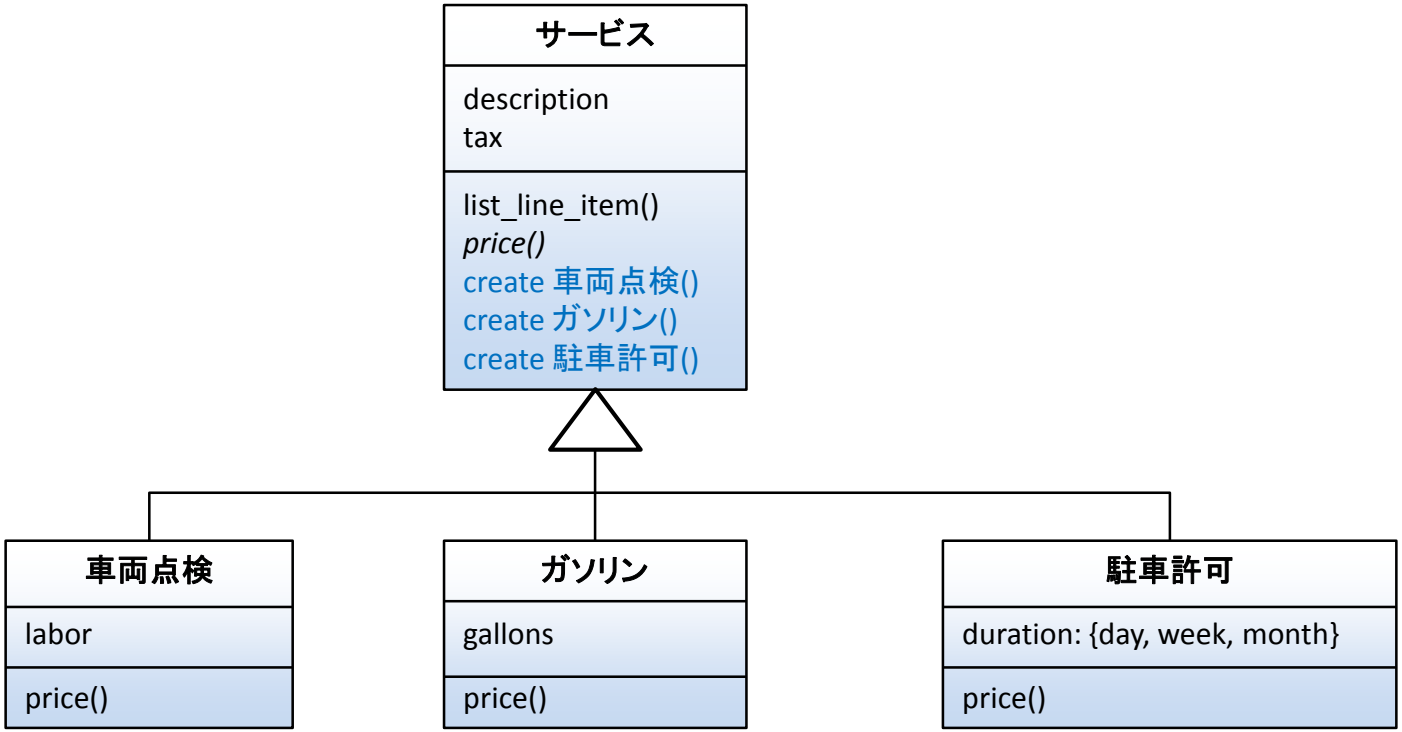
Template Method パターン



Factory Method パターン

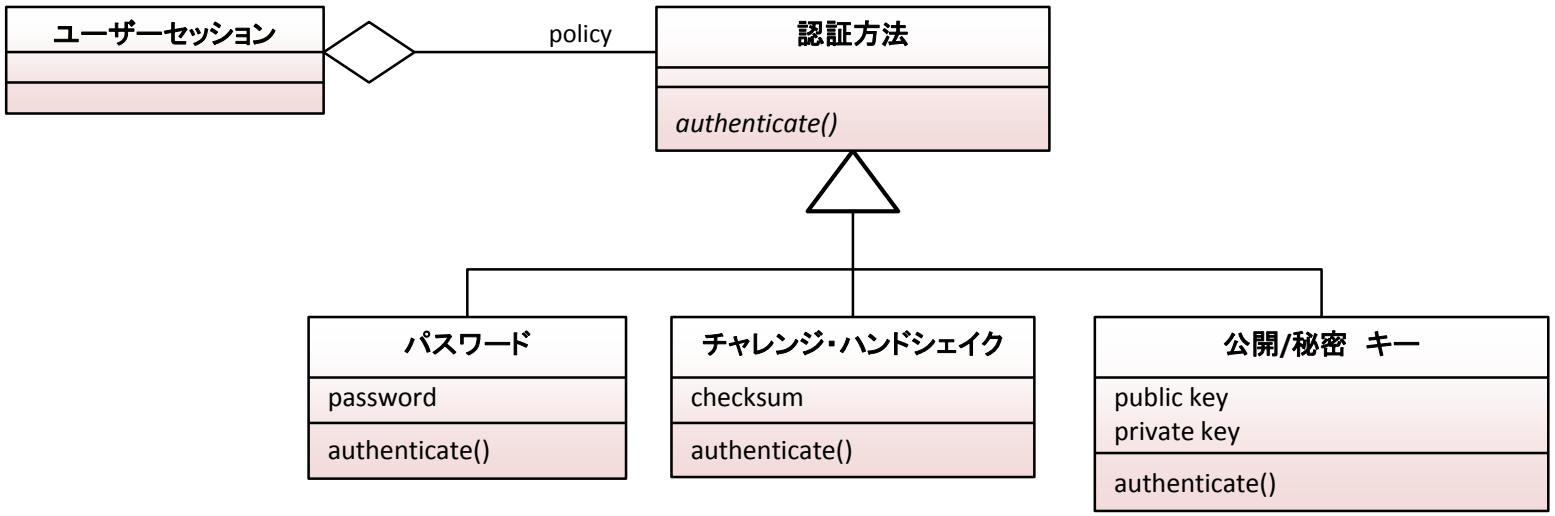
- 概要
  - オブジェクト生成用のコードをカプセル化し、直接見えなくする
- 用途
  - 具体的なクラス名を必要とするオブジェクト生成を整理する（抽象型やインタフェースが使えない）
  - オブジェクト生成に必要な知識をカプセル化する（知識境界）
- 解決法
  - 抽象クラスを作り、オブジェクトを生成するコンストラクタ・メソッドを宣言する
    （Template Method パターンに似ている）

Factory Method パターン



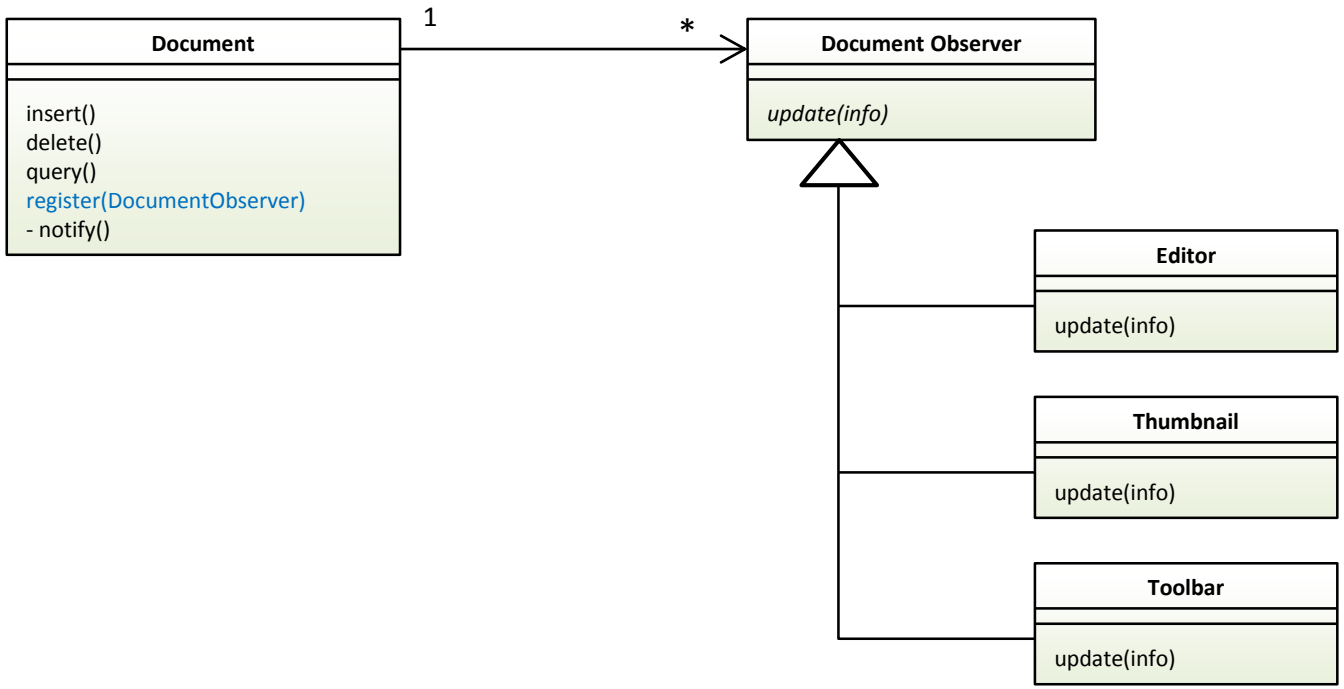
Strategy パターン

- 概要
  - 実行時にアルゴリズムの選択を実現する
- 用途
  - 状況に応じて、複数の選択肢から最適なアルゴリズムを選択する

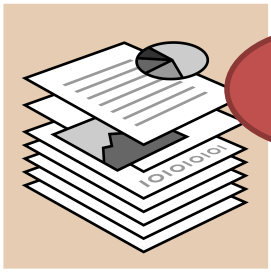


Observer パターン

- 概要
  - publish-subscribe のアーキテクチャを実現する
- 目的
  - 一つのイベントを複数のオブジェクトに通知する



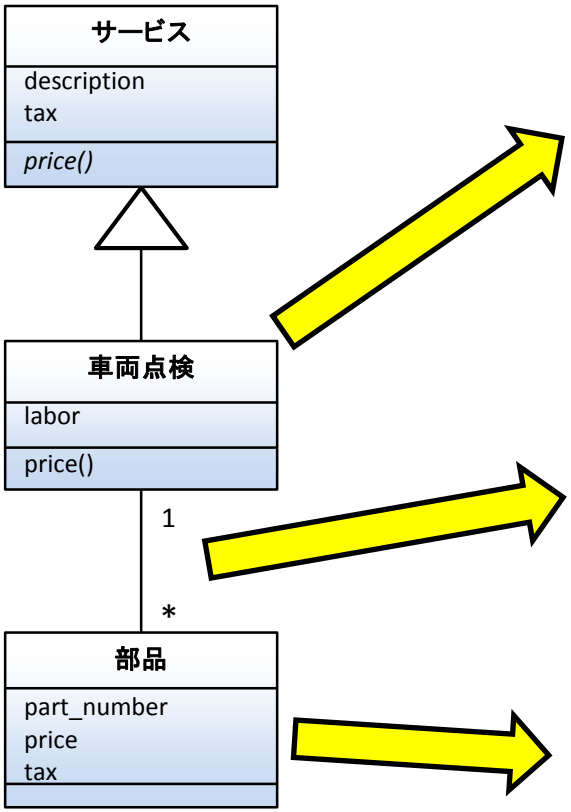
データ管理の設計



- 目的
  - データに関する性能と容量の要求に応える
- 要求と制約に留意して設計する
  1. データ、データ構造、データ同士の関連の明確化
  2. データ構造と関連を管理するサービスの提供
  3. データ管理に使うツール、システムの調査
  4. データ管理機能を制御するクラス群の設計

例：オブジェクト指向データベース

ガソリンスタンドの
アプリケーション



車両点検	ID	種類	税	作業
	1	オイル交換	0.25	5
	2	タイヤ交換	0.25	5
	3	タイミングベルト交換	5	100

点検X部品	ID	部品番号
	1	1X782
	2	—
	3	F895ERT

部品	部品番号	値段	税
	1X782	8	.40
	P3291	200	10
	E89WWY	34	1.70

例外処理の設計

- 目的
  - エラーチェックとエラー回復処理を主要な処理から分離する
  - 積極的にエラーを想定することで堅牢性を上げる
  - エラーからの回復方法を想定しておく

- 例外処理は、プログラミング言語の機能として多くの言語でサポートされている

例外処理の例（擬似コード）

- 手続き unsafe-transmit を用いてメッセージの送信を試みる
- unsafe-transmitは送信に失敗すると例外を投げるのでリトライする
- 100回続けて失敗するとあきらめて例外を投げる

```
attempt_transmission (message: STRING) raises TRANSMISSIONEXCEPTION
local
        failures : INTEGER
try
        unsafe_transmit (message)
rescue
        failures := failures + 1;
        if failures < 100 then
                retry
        else
                raise TRANSMISSIONEXCEPTION
        end
end
```

Javaの場合

```
try {
    ...
} catch(Exception e) {
    ...
}
```

ユーザインタフェース設計

- 主要な検討項目
  - システムを利用する人を明確化
  - システムの動作に結びつくシナリオの定義
  - ユーザのコマンド階層の設計
  - ユーザのシステムとのやりとりの手順を詳細化
  - UIを実装するクラス階層の設計
  - UIのクラス階層と全体のクラス階層の統合

画面設計の立案

既存の帳票

Akabekoガソリンスタンド
福島県会津若松市一箕町

請求書

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

日付：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

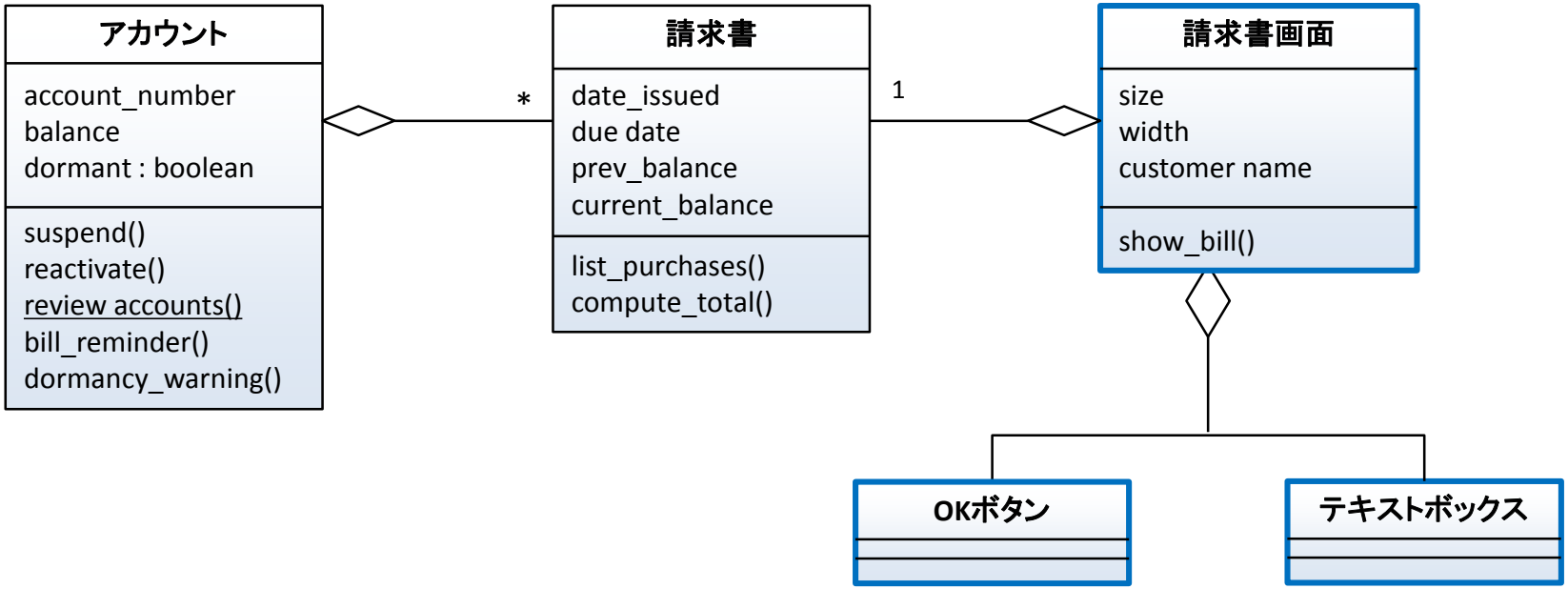
お買い上げ品

日付	品名	金額

合計金額：＿＿＿＿＿＿

対応する設計

対応するクラス階層の設計



開発フレームワーク

9.3.2(3)
9.3.3

- 再利用可能な「大規模な設計」
  - アプリケーション分野に特化している
  - GUI環境、Webアプリケーション、経理システム、などさまざま
- フレームワークは公開されたツールキット
  - 企業内利用を目的としたライブラリーとは異なる
  - 高レベルのアーキテクチャーであり、低レベルの詳細を埋めなければならない

設計書

- アーキテクチャー設計書に則ったプログラム開発を行うための文書
  - 詳細設計書、プログラム設計書という

- 設計書に含める内容
  - プログラムの作成に必要十分な情報

5.6.3

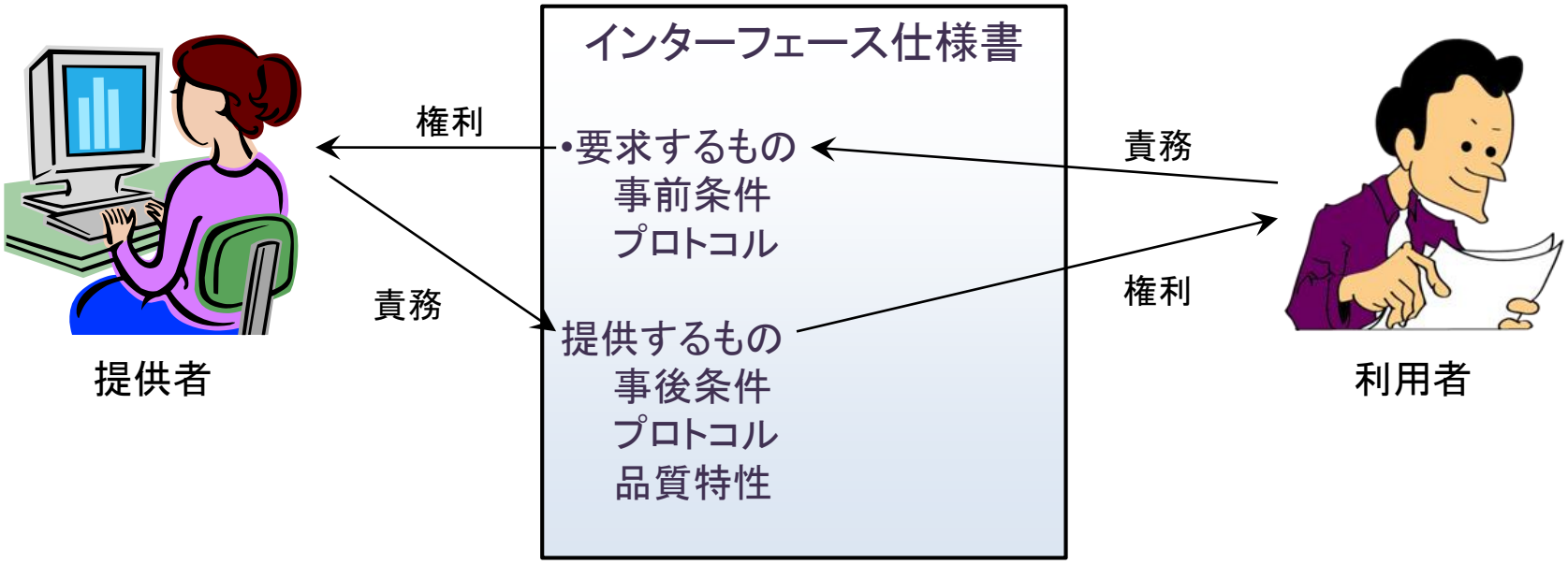
Design by Contract

Bertrand Meyer

- モジュールの動作をインタフェース仕様で詳細に定義する
- 仕様を“契約”と呼ぶ
  - 他モジュールとのやりとりの方法のみを定める
  - 処理内容の正しさは保証していない
- 契約の構成
  - 責任：利用する側が準備する→事前条件
  - 利益：利用する側が得るもの→事後条件
  - 不変式（invariant）：実行される処理
- 効果
  - モジュール間の連携を保証する
  - プログラムを書く際の責任範囲が明確になる
  - テスト、検証の規準になる

Design by Contract

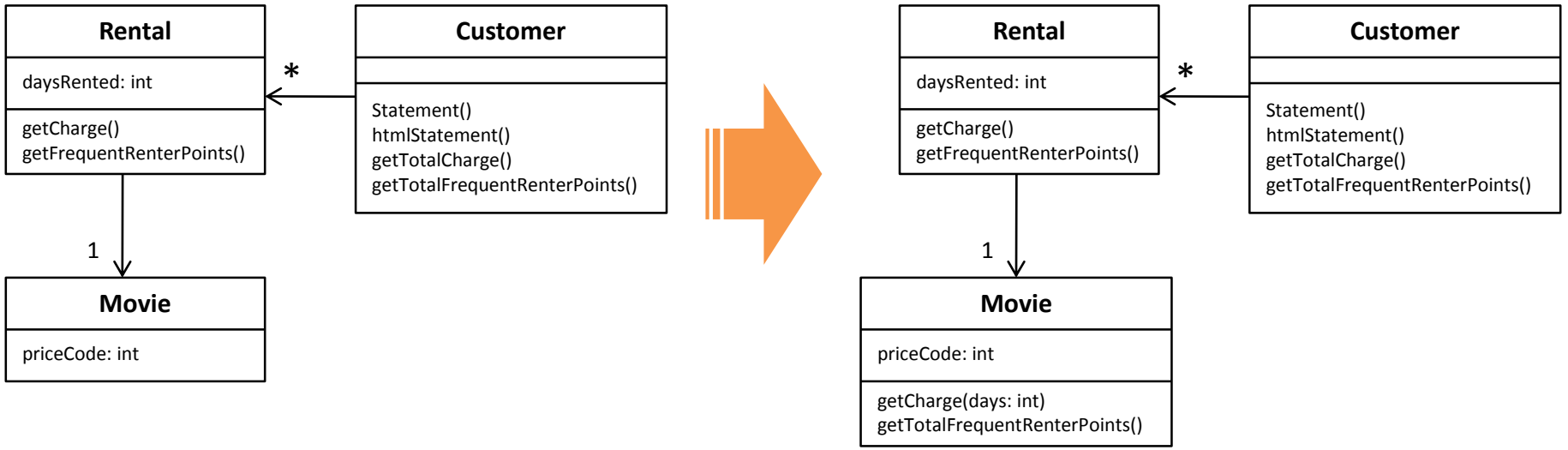
- モジュールの提供者と利用者間の設計契約



9.2.3

リファクタリング

- 設計を定期的に見直し、書き換えること
  - 複雑な解決策を単純化する
  - 設計を最適化する
  - アジャイル開発では特に重要



リファクタリング：プログラミングの体質改善テクニック、マーチンファウラー著、ピアソンエデュケーション

実際のプロセスと設計書の違い

- 理想
  - トップダウンに、一階層ずつ詳細にしながら、整理されたモジュールを設計する
- 実際
  - 行ったり来たりを繰り返し、モジュールの設計は段階的に完成に近づく

- 設計書は、トップダウンに、階層的に設計を整理し直して記述する
  1. ソフトウェアユニットをモジュールに分解
  2. モジュールのインタフェースを決定
  3. モジュール間の依存性を記述
  4. 各モジュールの内部設計を記述

- 判断を先延ばしする部分は、詳細が決定するにしたがって追記する

本日のまとめ

- デザインパターン：設計の定石
  - 再利用可能な部分的解法
  - 設計の原則に則った「設計のパターン集」
- その他の設計要素
  - データ管理の設計、例外処理の設計、UI設計
- 設計書
  - プログラミングに必要な情報を記述する

次回講義の事前学習：
講義資料