# オブジェクト指向設計

FU14 ソフトウェア工学概論

第9回

吉岡 廉太郎

# 前回の内容

- モジュール設計は、ソフトウェアユニットの詳細を決めるプロセス
- アーキテクチャー設計のソフトウェアユニットをモジュールに分解
- アーキテクチャーと異なり、設計指針となる様式がない
  - 解決策はより個別的
- 設計の原則：モジュールを検討する際のガイドライン
  - 6つの原則
- 継続的に見直しながら設計を進める
  - 設計書を書きながら
  - リファクタリング

# 今日の内容

- オブジェクト指向設計
  - 最も一般的な設計手法の一つを学ぶ
  - 重要な用語と概念を知る
  - 設計のガイドラインを知る

- UML
  - 最も一般的なオブジェクト指向設計の記法を知る

# オブジェクト指向設計

5.6.1

- 設計の方法論の一つ
  - 広く普及している最も洗練された設計手法の一つ
- システムを、オブジェクトという実行レベルのコンポーネントに分解する
- オブジェクトとは
  - データや機能ごとに分離された実行単位
  - 一意に識別できる単位
  - メッセージや要求の対象
  - 入れ子にすることで合成できる
  - 実装は継承により再利用・拡張できる
  - 同じコードを異なる型に対しても実行できる（多態性）

# 用語と概念：クラス関連

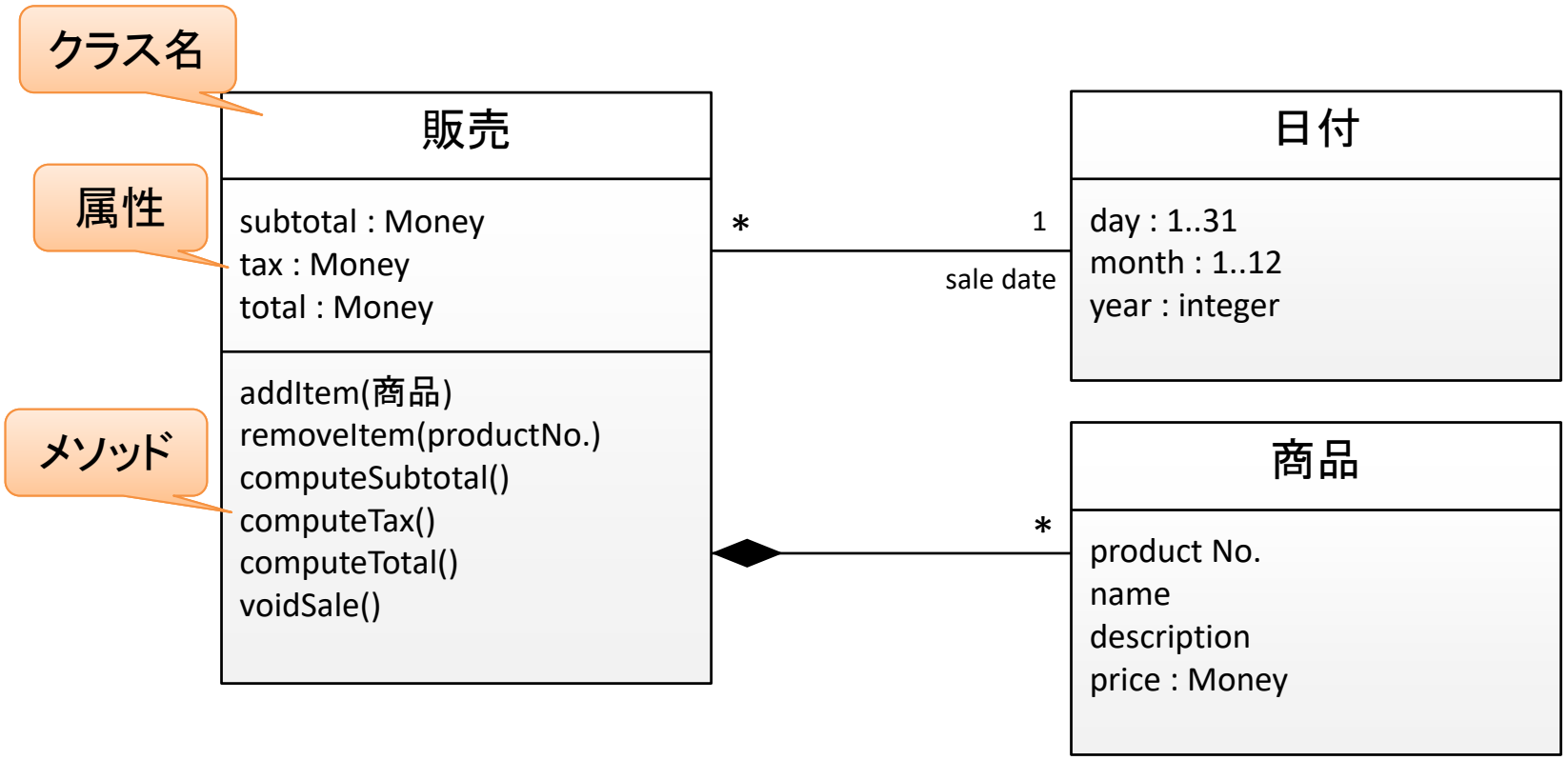
| | |
|---|---|
| クラス | • 抽象データ型を実装したモジュール |
| 抽象クラス | • 全てのメソッドを実装していないクラス |
| コンストラクター | • オブジェクトの新たなインスタンスを生成するメソッド |
| インスタンス変数 | • 計算に用いる変数で、オブジェクトを参照する<br>• オブジェクトはそのクラスの特定の“値” |

# オブジェクトでシステムを表す

- オブジェクト指向の設計では、システムはオブジェクトの集合
  - あるオブジェクトの抽象データ型をクラスという
- システムを構成するデータと操作を表現する
  - データ：性質、状態など
  - 操作：生成、読取り、変更、削除など
- 用語
  - オブジェクトのデータ：クラスの属性
  - オブジェクトの操作：クラスのメソッド

# オブジェクトの構成要素

- 例)「販売クラス」の設計

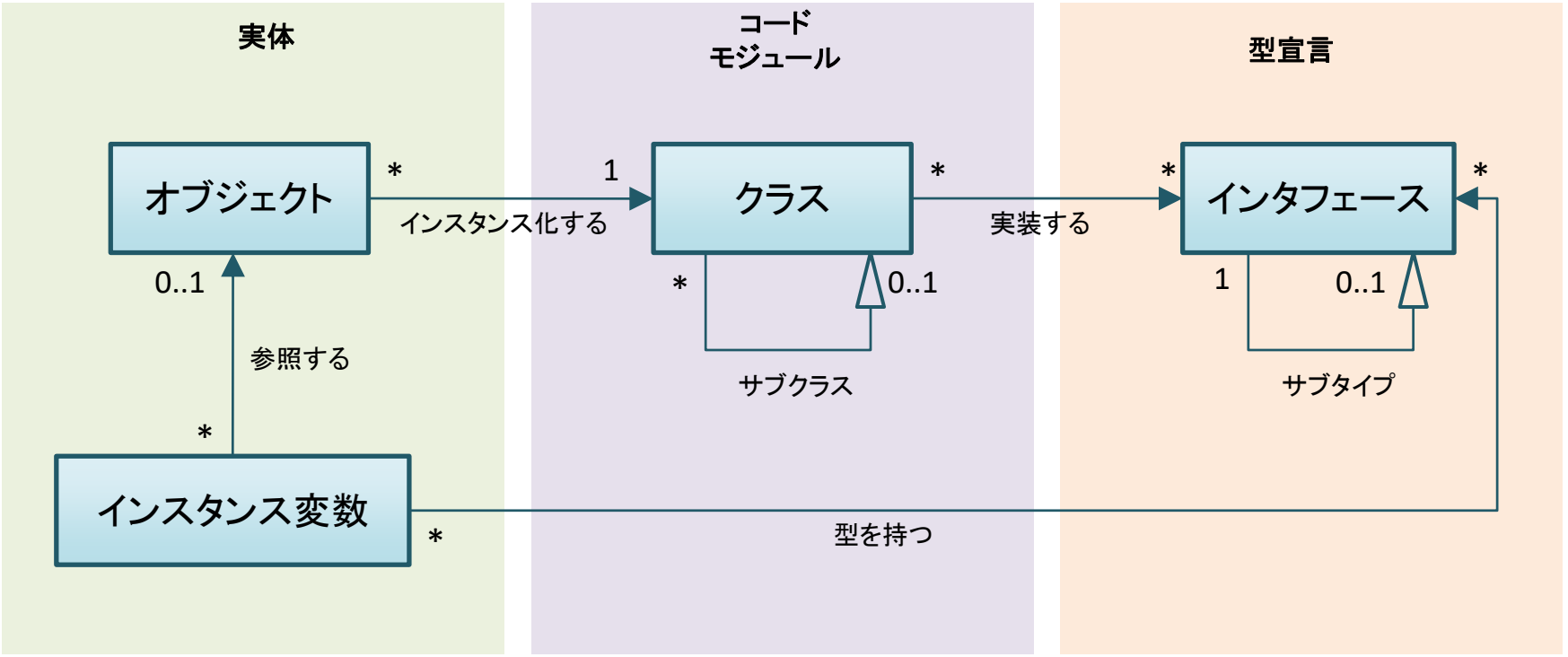


# インタフェース

- オブジェクトのデータ/メソッドに対するアクセス範囲を表す
  - 具体な実装を持たない＝抽象（型宣言）
- インタフェースのサブタイピング
  - インタフェースを継承して新たなインタフェースを定義すること
  - もとのインタフェースをスーパータイプと呼び、それより制限されたアクセスを提供するインタフェースをサブタイプという
- 動的バインド
  - 変数が参照するオブジェクトの型を実行時に変更できる
  - 共通のインタフェースをもつオブジェクトで可能

# 四大要素の関係

- クラス、オブジェクト、インタフェース、インスタンス変数

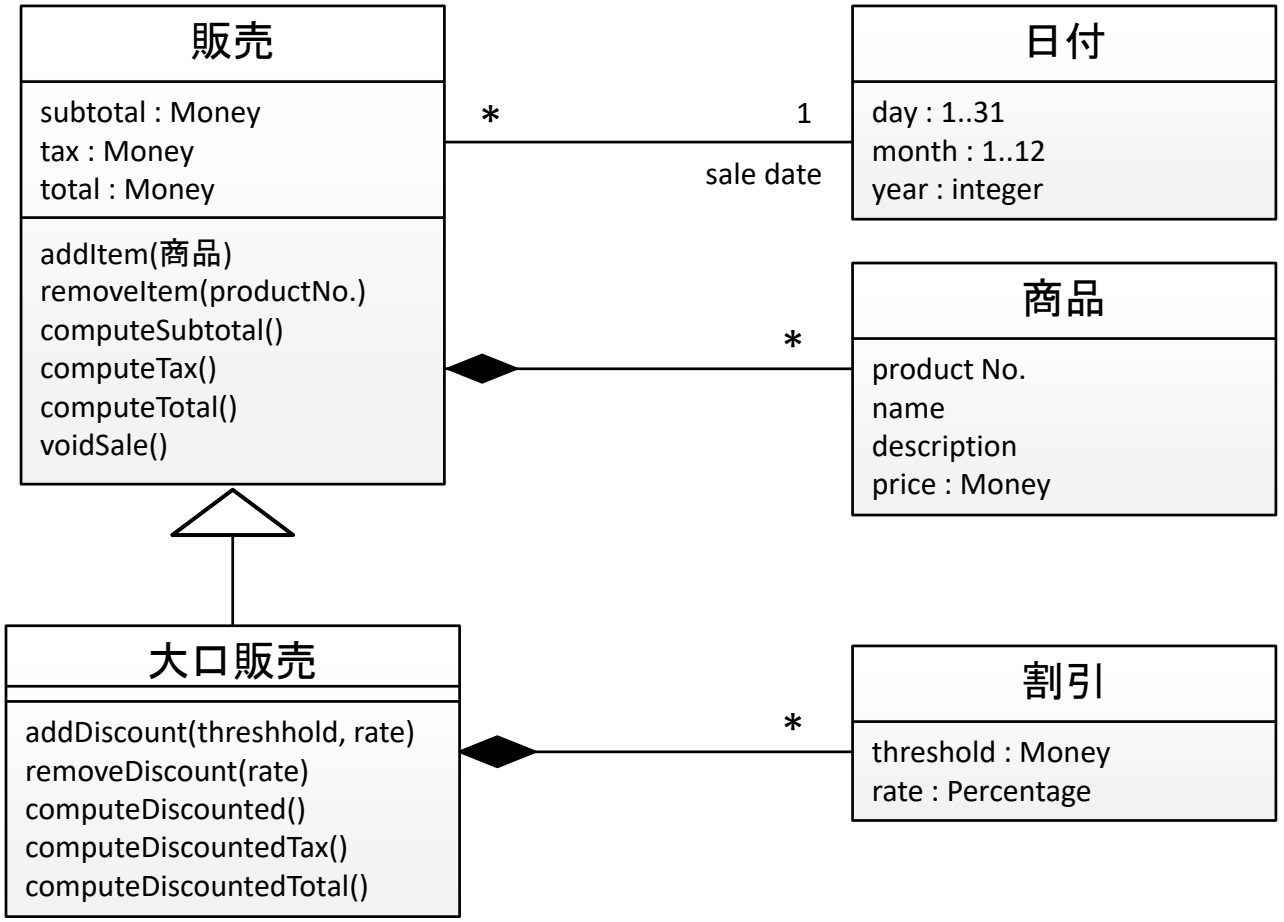


# クラスから新たなクラスを作る

- 継承
  - 既存のクラスを変更、拡張することで新たなクラスを構築すること
- 合成（composition）
  - クラスを組み合わせて新たなクラスを構築すること

# クラスの継承

- 例)「大口販売」のクラスを追加



# インタフェースの活用

- 多態性（ポリモルフィズム polymorphism）
  - 動作を実行時に変更する方法
  - インタフェースに合わせてコードを書いておく
  - 実際の動作は、実行時に関連づけられる（サブタイプを実装した）オブジェクトとそのメソッドによって決まる

- 継承、合成、多態性は、システムの使い勝手を良くするオブジェクト指向の強力な特徴となっている

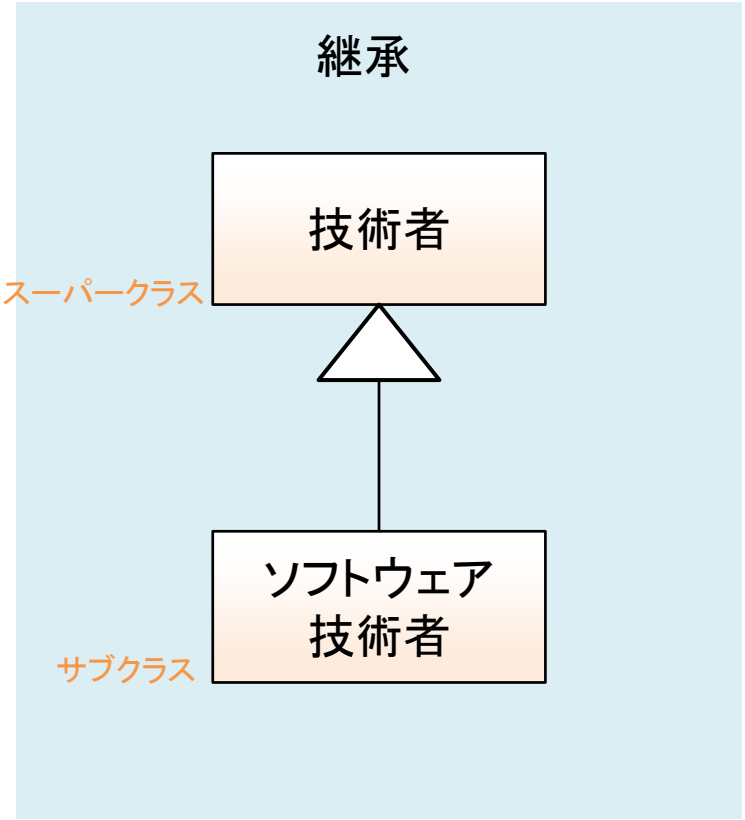
# 継承 vs. 合成

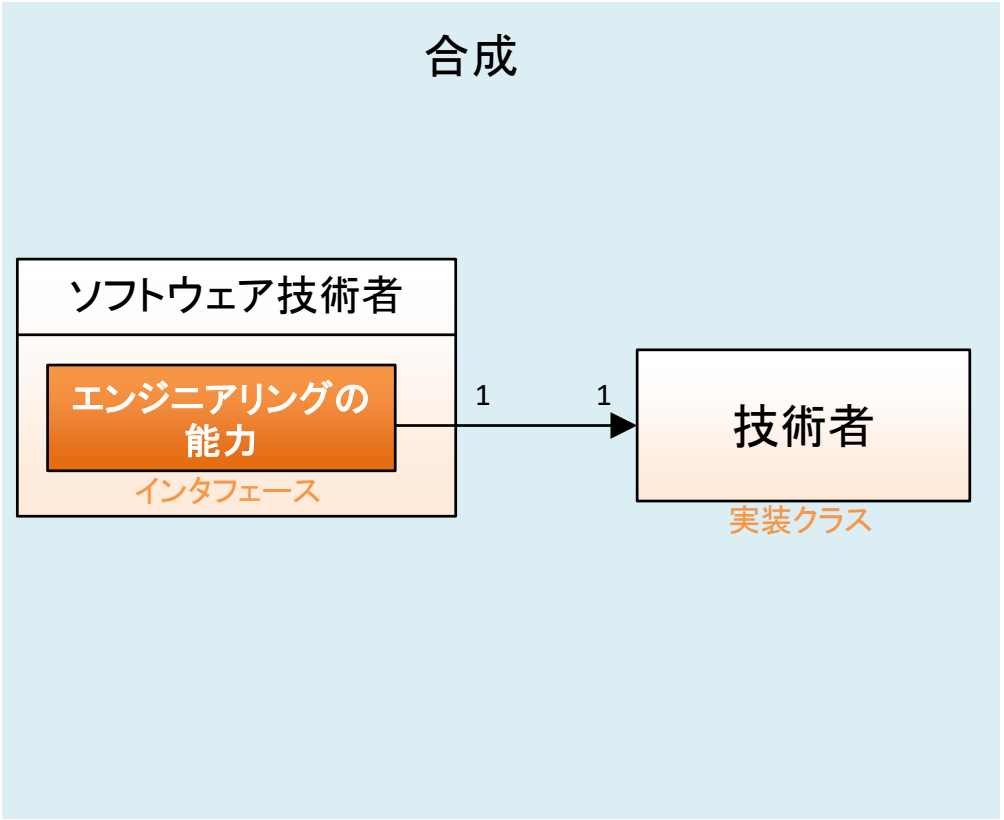
- 複雑なオブジェクトをどう構成するかは設計上の重要な判断ポイント

- 大きなオブジェクトを構築する2つの方法
  - 継承
    - 既存のクラスの振る舞いを拡張したり上書きする
  - 合成
    - 小さなクラスを組み合わせて複合クラスを作る

# 継承 vs. 合成

- 例)「技術者」と「ソフトウェア技術者」クラス



# 継承 vs. 合成

## ◇合成の利点

- 再利用するコードをカプセル化しやすい
- 組み合わされたクラスはインタフェースを通してアクセスする

## ◆継承の欠点

- 継承で作成されるサブクラスの実装は設計時に決定される
- 合成では、実行時の条件に応じて適合性のあるオブジェクトで置き換えができる

## ◇継承の利点

- 継承されたメソッドの動作を変更・特化することができる
- 動作を予測しやすく、理解しやすい

5.6.4

# 代替可能性

- サブクラスは親クラスの振る舞いに準拠しなければならない
  - コード上、サブクラスのインスタンスを親クラスのインスタンスと同等に扱かえる必要がある

- Liskovの置換原則
  - サブクラスは親クラスのすべてのメソッドを備え、引数や戻り値が同じである
  - サブクラスのメソッドは、親クラスの同一メソッドの仕様を満たさなければならない

> •事前条件　$pre_{parent}$ ⇒ $pre_{sub}$
> •事後条件　$pre_{parent}$ ⇒ ($post_{sub}$ ⇒ $post_{parent}$)

  - サブクラスは親クラスで定義されたすべてのプロパティーを備えなければならない

- この原則は、あるオブジェクトを別のオブジェクトで置き換えても問題がないかを確認するのに用いる
  - 原則を満たすように拡張設計を行えば、サブクラスを新たなに追加しても、これまでのコードは安全に動作する

# デメテルの法則

最小知識の原則

- 合成クラス利用における依存性を削減する方法

- デメテルの法則：合成クラスにおいて、含まれる子クラスにアクセスするメソッドを追加する

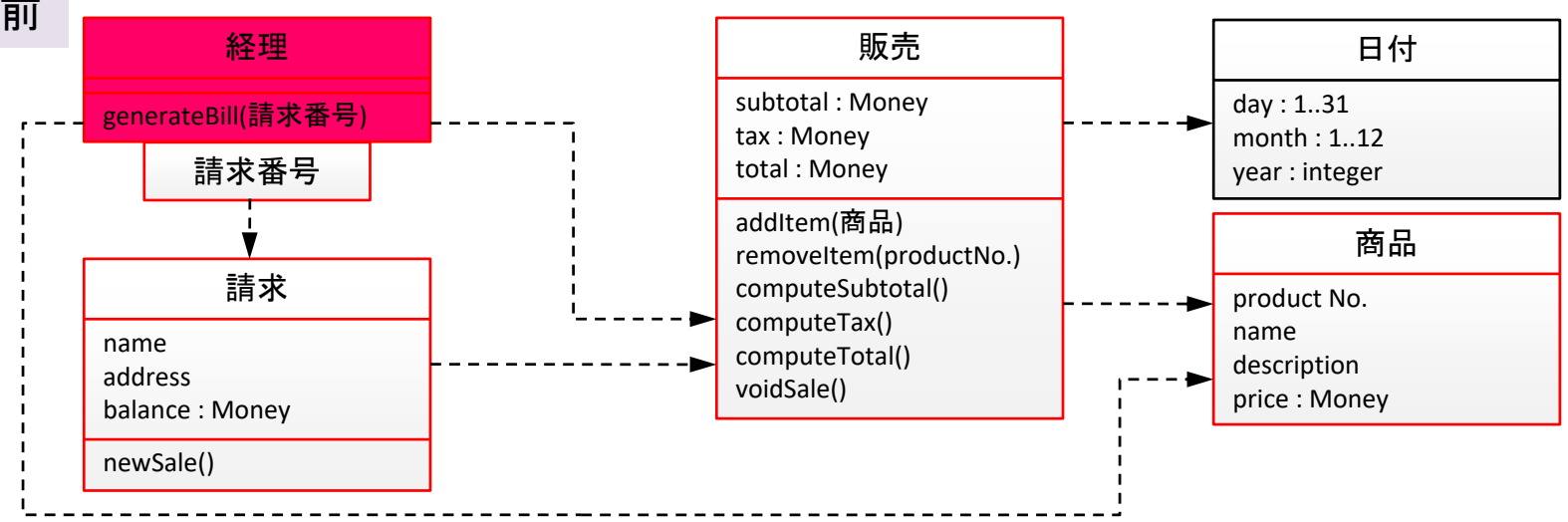
- 利点
  - 合成クラスを用いるコードでは、そのクラスのことだけ知っていれば良い
  - 合成クラス内の他コンポーネントについて知る必要がない

- この法則に則った設計では、クラス間の依存性が減り、依存性を多く持たないクラスの方がエラーを含みづらく変更にも強い
  ＝結合性が弱まる

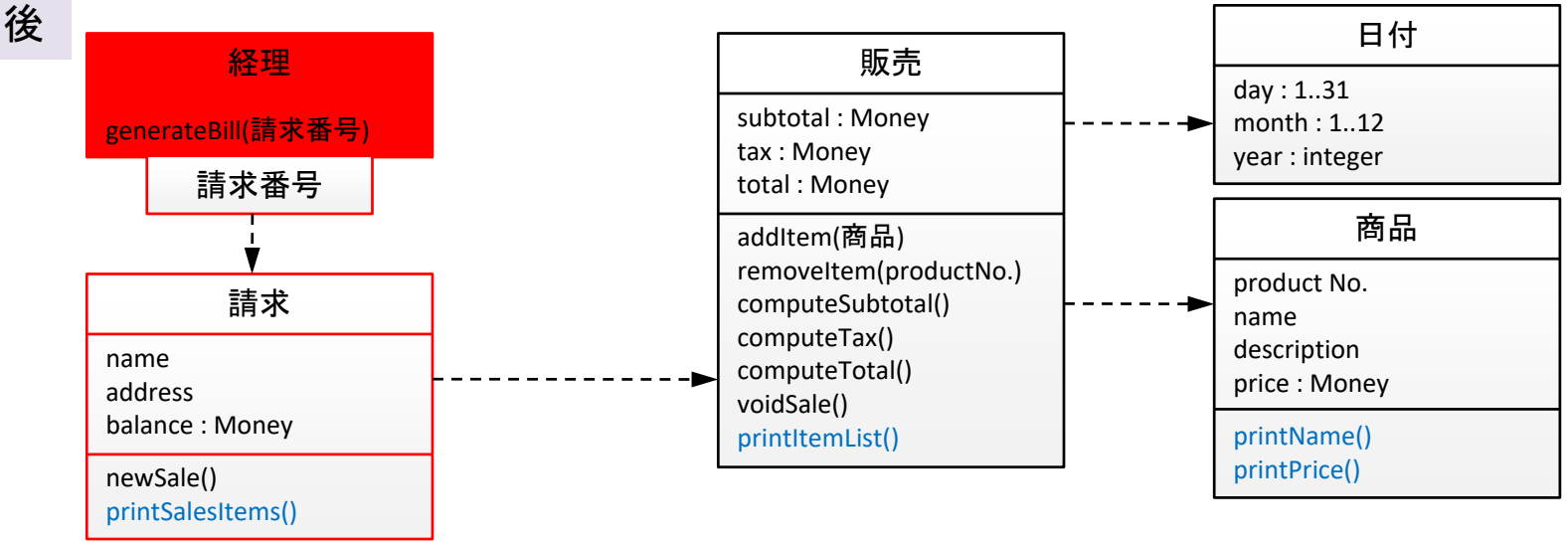
# デメテルの法則の例

商品一覧付き請求書を印刷したい

## 適用前



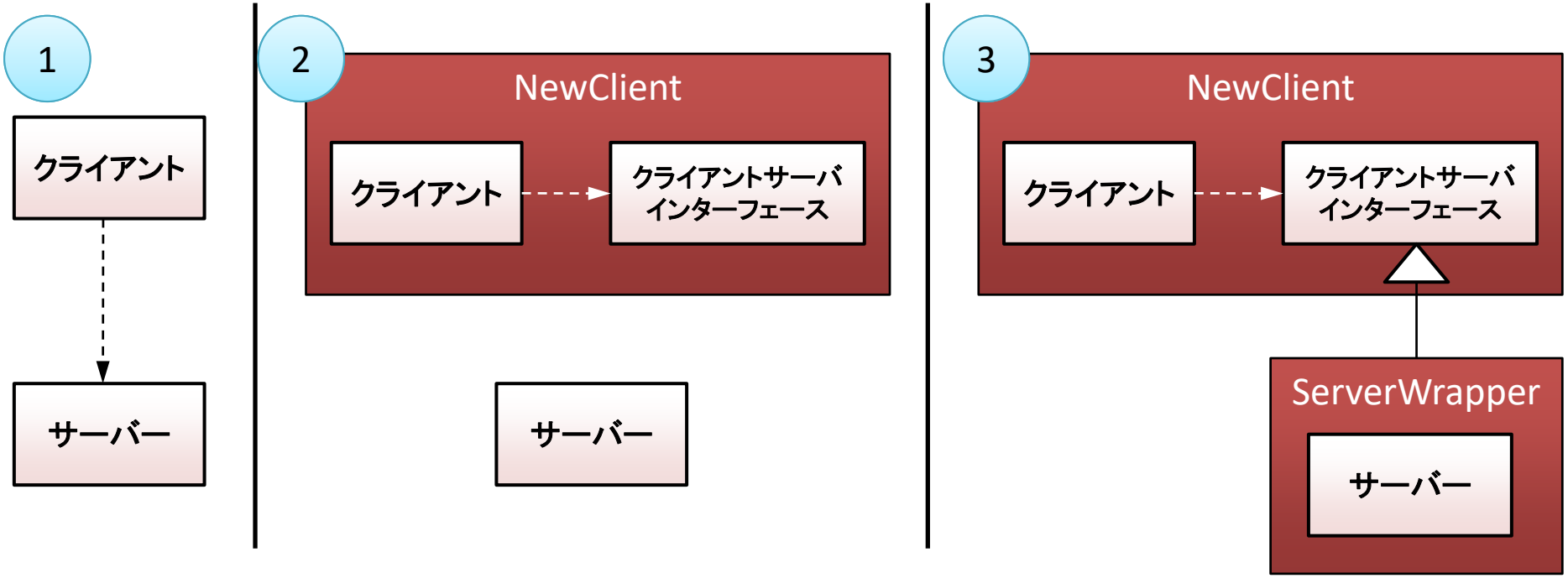

## 適用後



5.6.4

# 依存関係逆転の法則

- クラス間での依存関係を逆転すること
  - 具体(実装)に依存すると結合性が高まる
  - 抽象(インタフェース)に依存するようにする
  - 結果的に依存性が逆転し、結合性が弱まる




この法則はデザインパターンで利用されている

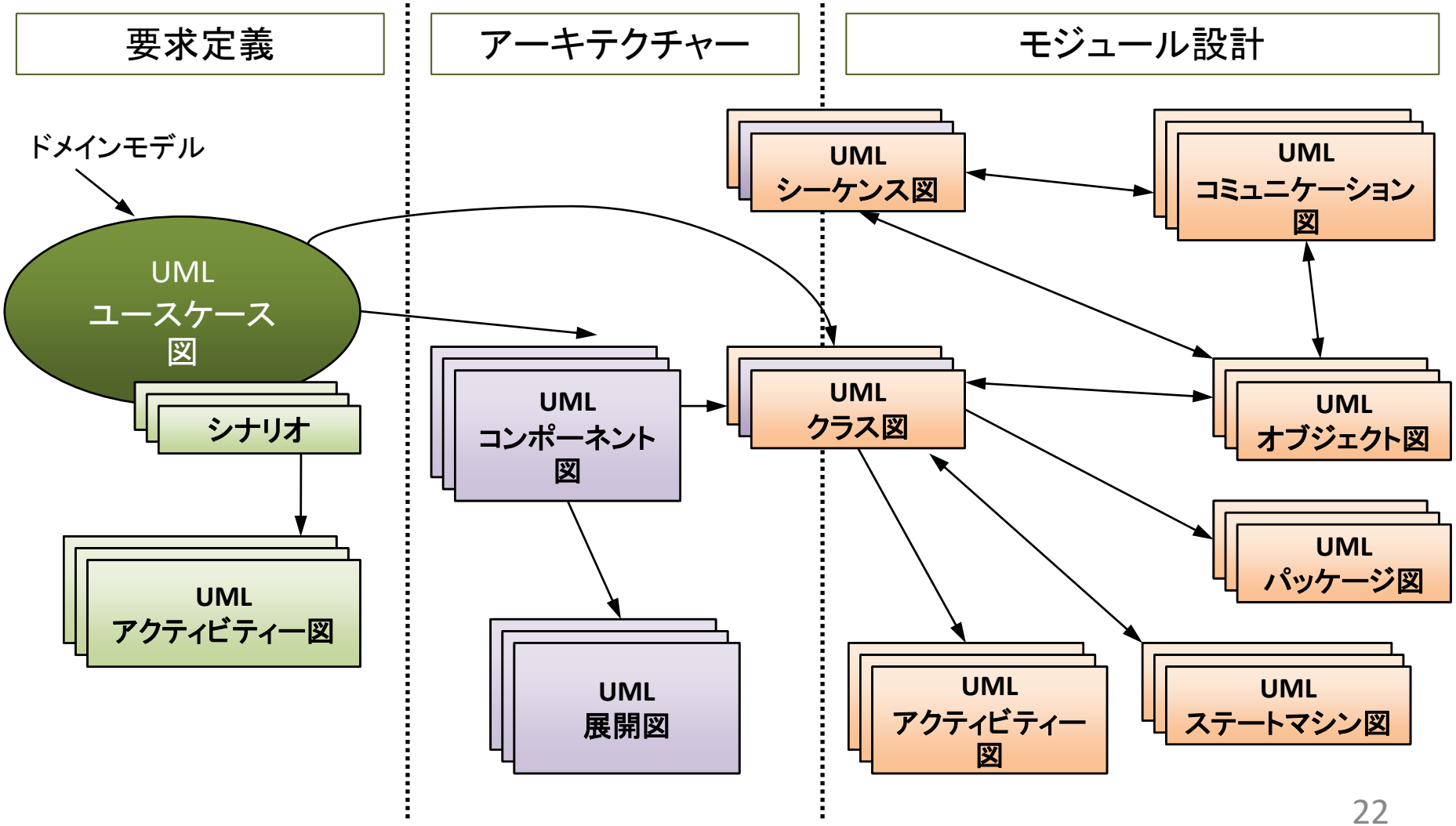
# UMLとオブジェクト指向設計

- UMLは、オブジェクト指向のシステムの記述として一般的な設計記法である
- UMLを用いて、ソフトウェアの設計を可視化、定義、ドキュメント化することができる
- UMLが特に役立つ点
  - 設計上の選択肢を比較する
  - 設計をドキュメント化する

# UMLで定義された図の種類

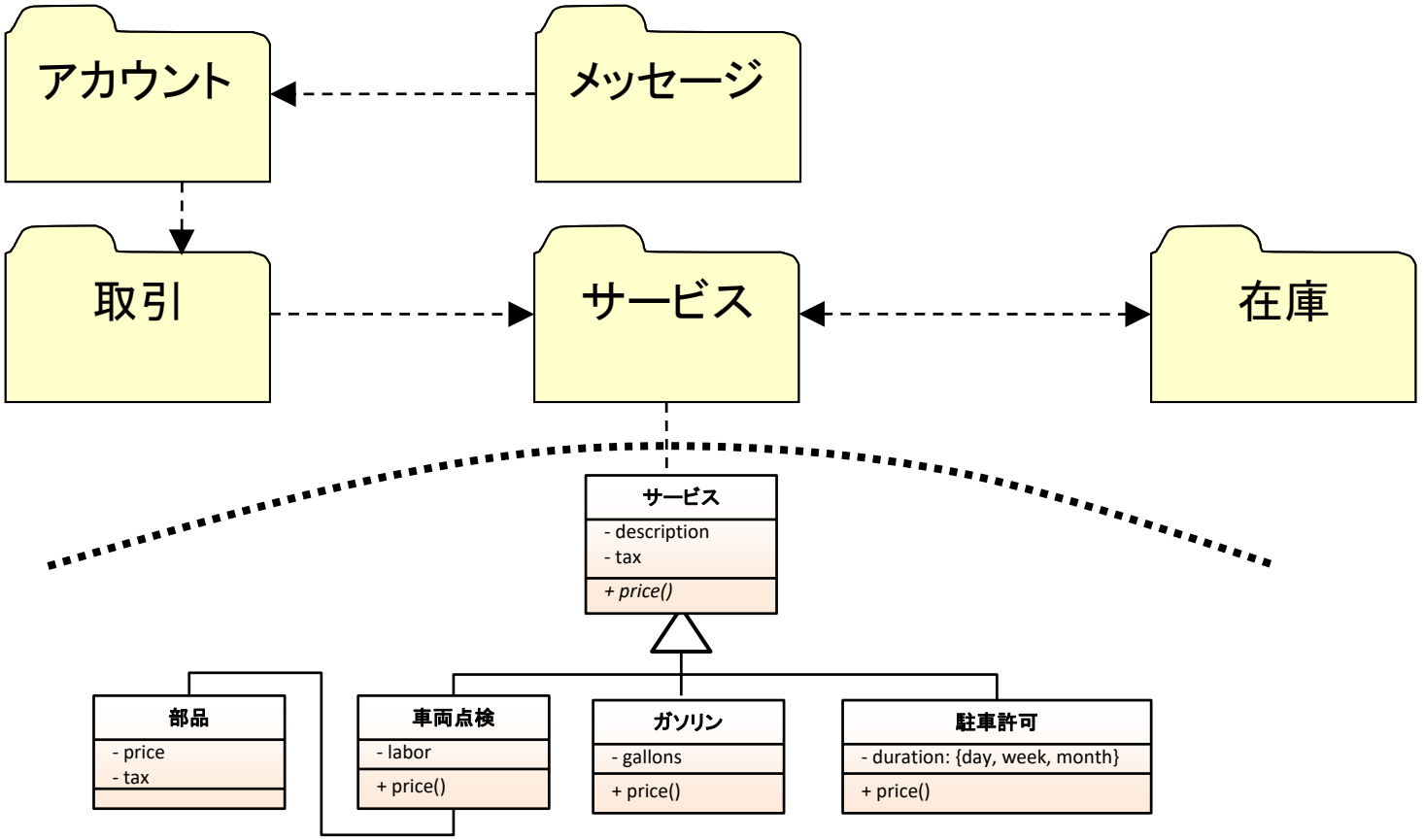
| | | | |
|---|---|---|---|
| ユースケース図 | アクティビティ図 | ドメインモデル | コンポーネント図 |
| 配置図 | クラス図 | 相互作用概念図 | シーケンス図 |
| コミュニケーション図 | アクティビティ図 | ステートマシン図 | パッケージ図 |

# プロセスの中でのUML



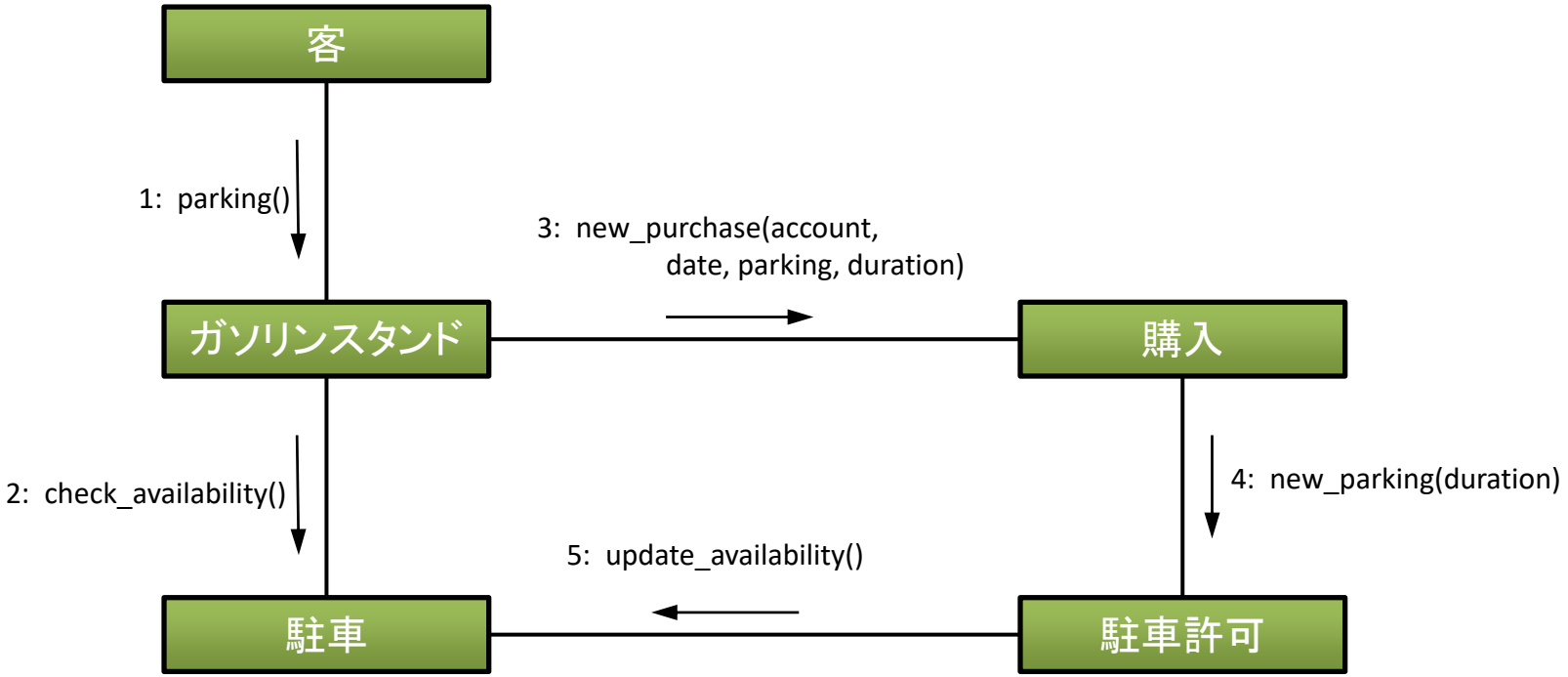
# UMLパッケージ図

- 将来、拡張が可能な単位でクラスを一単位（パッケージ）にまとめ、全体を把握しやすく見せる



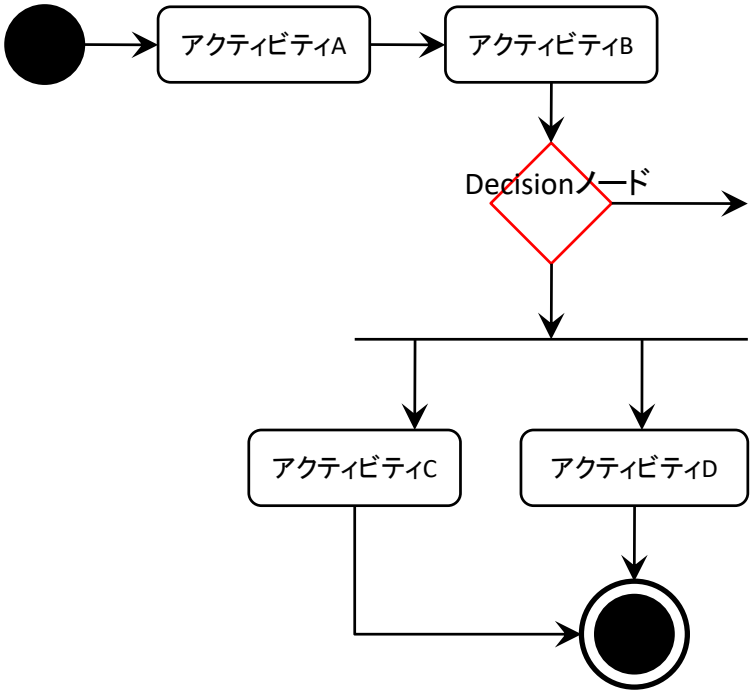
# UMLコミュニケーション図

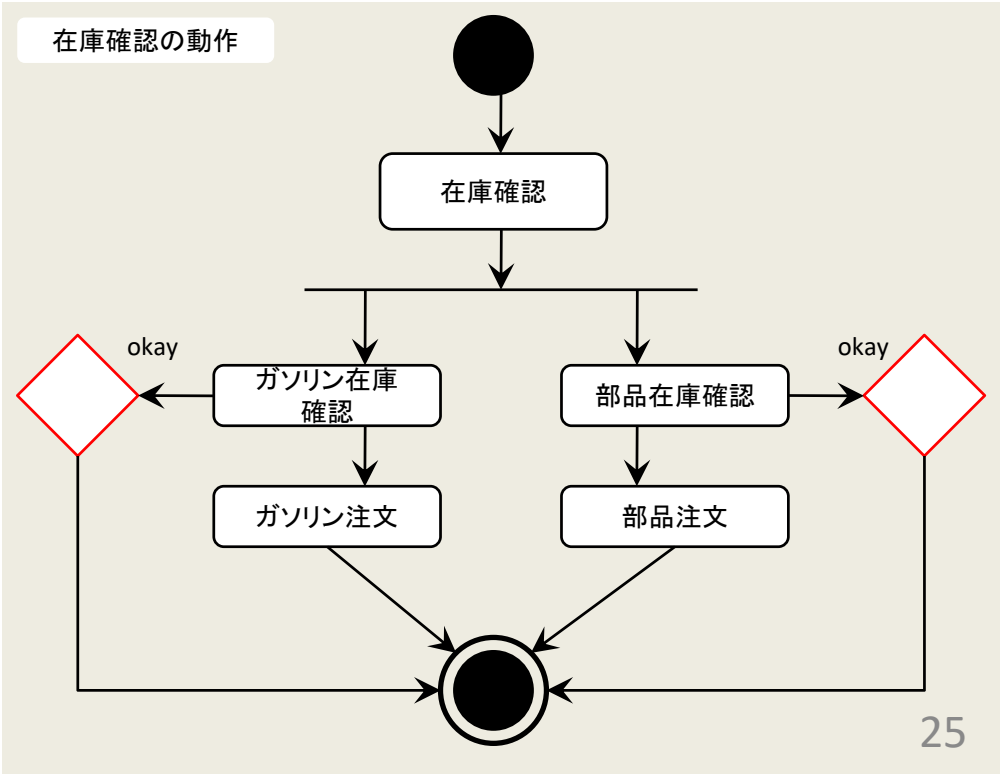
- オブジェクトの関係図の上でメッセージの一連の動きを表す



# UMLアクティビティー図

- クラス内での手続きの順序やアクティビティーの流れを表す
- Decisionノードで呼び出すアクションの選択を表す



# 本日のまとめ

- オブジェクト指向設計
  - 課題のとらえ方・分析の仕方の一つ
  - オブジェクト指向設計を象徴する用語と概念
    - 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、合成
    - 代替可能性、デメテルの法則、依存関係逆転の法則
- UML：記述・モデリング言語
  - オブジェクト指向設計で良く用いる表記記法
  - プロジェクトの各工程で活用できる多数の図を定義

*UMLモデリングのエッセンス（UML Distilled)*, Martin Fowler, Addison-Wesley

次回講義の事前学習：
5.4, 5.6.3, 9.2.3, 9.3.3, 9.4.2