

7.7 開発設計

- UML 概要

- UML 図の種類

- Open API



1

UML 概要

2

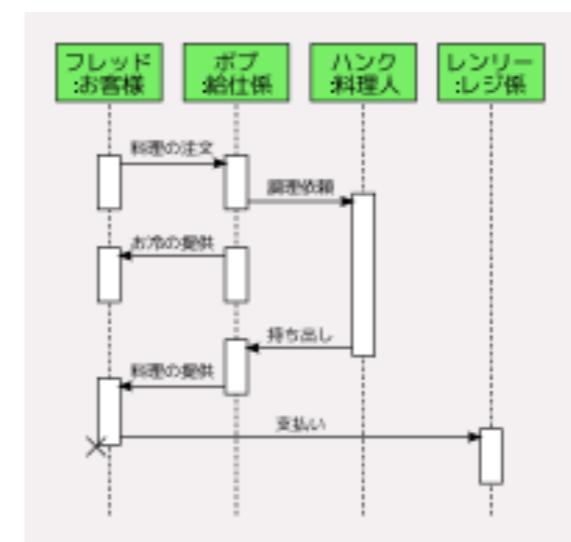
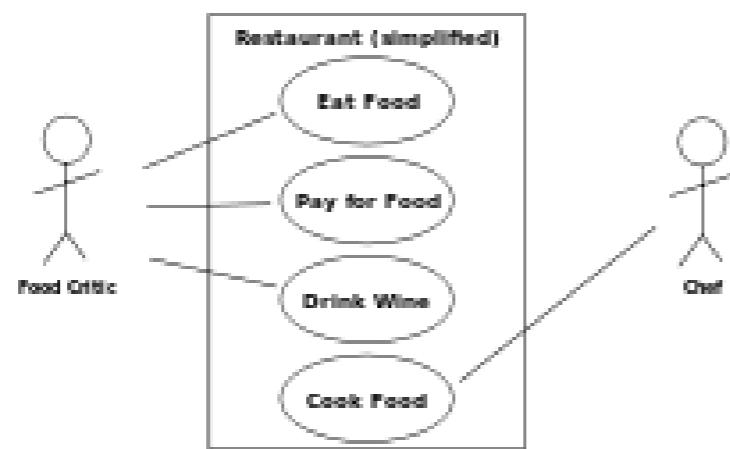
UML 図の種類

3

Open API

UML 概要

- **統一モデリング言語**[Unified Modeling Language, UML]は、ソフトウェアエンジニア開発の世界において、ビジネスユーザーを始め、システムを理解しようとするあらゆるユーザーが理解できる視覚化のための共通言語を打ち立てることを目的のために作られたモデリング手法です。
- UML は、ソフトウェアの可視化、統一を実現させ、説明書に対する理解力を向上させる効果があり、システム分析の可視化によりコミュニケーションが効率化できる効果があります。



UML 2.0 における変更

- UML は継続的に改良されています。アジャイルを含め、開発のより多くの側面に対応するよう、UML の仕様を拡張したものが UML 2.0 で、使いやすさや実装、導入を簡素化するため、UML を再構築し、改良することを目的として作られました。UML 図については、以下のような点が変更されています：
 - 構造モデルと行動モデルの間の統合強化、
 - 階層の定義とソフトウェアシステムのコンポーネントとサブコンポーネントへの分類が可能に、
 - UML 2.0 では図の数が 9 から 13 に拡大。

UML の視点

- システムの開発においては、以下の 3 つの全体的なシステムモデルに重点が置かれます：
 1. **機能**: ユーザーの視点からシステムの機能性を記述するユースケース図が該当します。
 2. **オブジェクト**: オブジェクト、属性、関連と操作の観点からシステムの構造を記述するクラス図が該当します。
 3. **動的**: 相互作用図、ステートマシン図やアクティビティ図は、システムの内部動作を記述するために用いられます。

UML 図一覧

| 名 前 | 説 明 |
|----------------|-----------------------|
| ユースケース図 | システムのユーザ要求を表現 |
| クラス図 | 部品(クラス)の内容と関係を表現 |
| オブジェクト図 | 部品(オブジェクト)の関係を表現 |
| シーケンス図 | 時間軸のオブジェクトの流れを表現 |
| ステートマシン図 | オブジェクトの状態や遷移を表現 |
| アクティビティ図 | 処理の流れを表現 |
| パッケージ図 | グループ化した単位の関係を表現 |
| コミュニケーション図 | オブジェクト間のメッセージのやり取りを表現 |
| タイミング・チャート | 時間軸の変化を表現 |
| コンポーネント図 | 部品の構造を表現 |
| コンポジット・ストラクチャ図 | クラスやコンポーネントの内部構造を表現 |
| 配置図 | システムの物理的な配置を表現 |



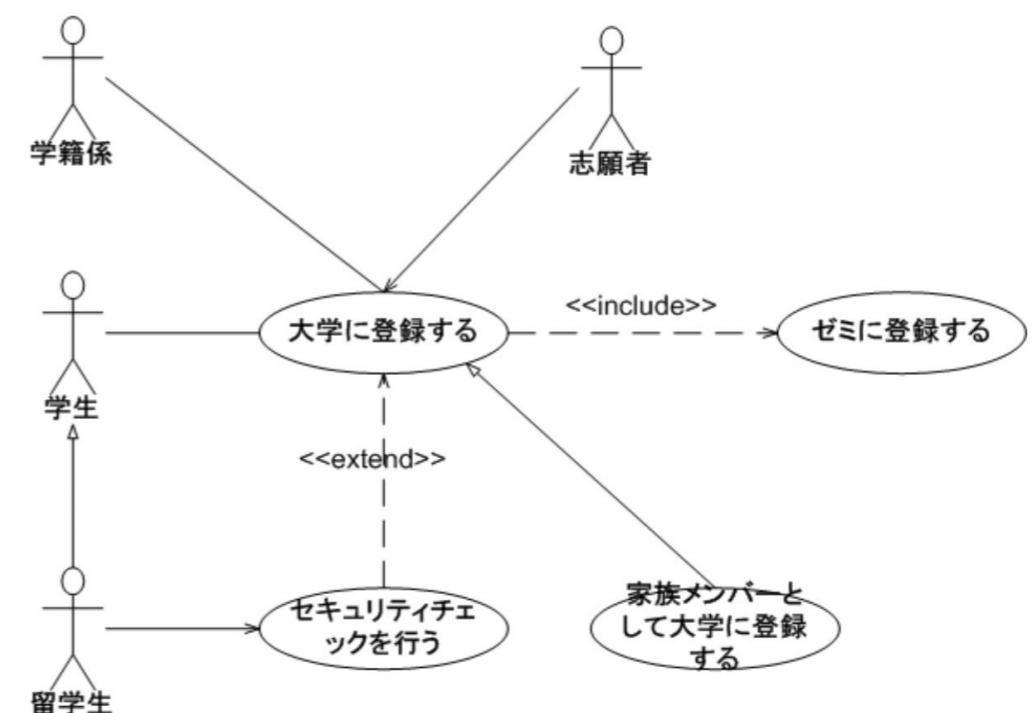
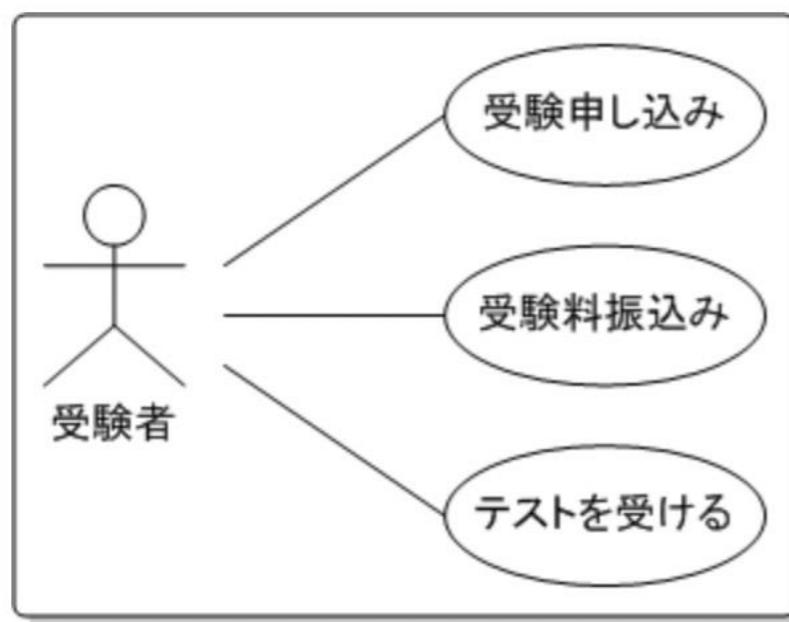
Question and answer



- 1 UML 図概要
- 2 UML 図の種類
- 3 Open API

ユースケース図

- **ユースケース図**[Use Case Diagram]は、システムのユーザー（別名アクター）とシステムとの相互作用を表すことができます。
- ユースケース図を作成するためには、一連の特別な記号とコネクターが必要となります。



ユースケース図の役割

- 効果的なユースケース図は、チームにおける以下の内容の議論や表現に役立ちます：
 - システムやアプリケーションが人、組織や外部システムと相互作用するシナリオ、
 - システムやアプリケーションがこれらの実体（アクター）の達成する内容を支援する目標、
 - システムのスコープ。

ユースケース図の目的

- UML ユースケース図は、以下の目的に適しています：
 - システムとユーザーの相互作用の目標を示す。
 - システムの機能要件を定義し、整理する。
 - システムのコンテキストと要件を指定する。
 - ユースケースにおけるイベントの基本的な流れをモデル化する。

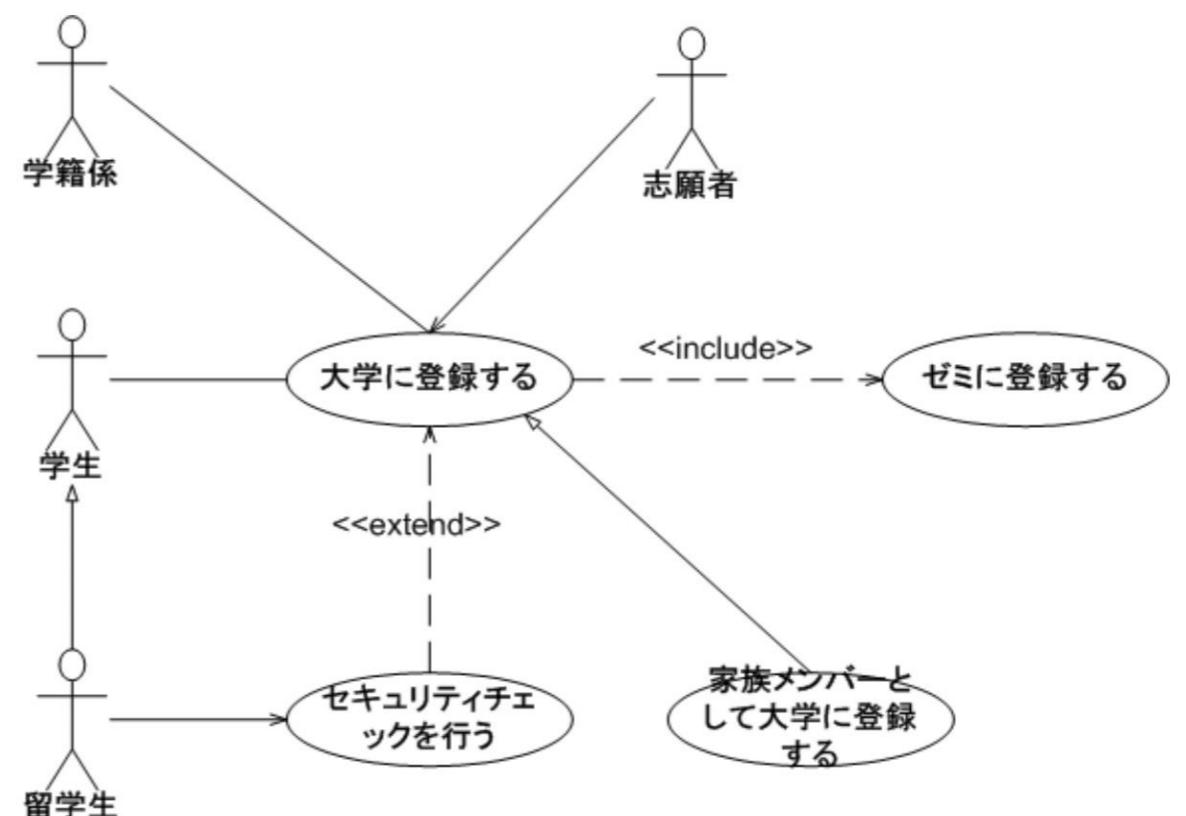
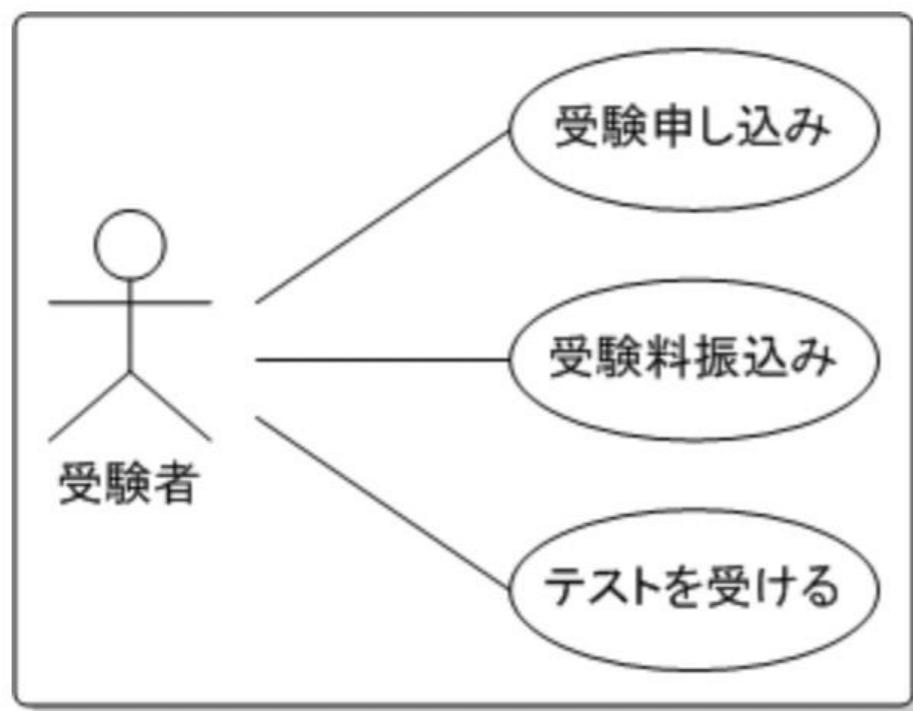
ユースケース図の構成要素

- **アクター**: システムと相互作用を行うユーザーを指します。検討対象のアプリケーションやシステムとやり取りを行う人、組織や外部システムなどがアクターとなり得ます。アクターは、データを生成または消費する外部オブジェクトである必要があります。
- **システム**: アクターとシステム間の特定のアクションや相互作用のシーケンスを指します。システムはシナリオと呼ばれる場合もあります。
- **目標**: 大半のユースケースの最終結果を指します。この目標に到達するまでのアクティビティとバリアントを表現できるのが優れたユースケース図の条件となります。

ユースケース図の基本コンポーネント

- **ユースケース**: 水平方向に長い橢円形で、ユーザーの有するさまざまな利用例を示します。
- **アクター**: 棒人間の形状で、実際にユースケースを利用する人を表します。
- **関連**: アクターとユースケースを接続する線です。
- **システム境界枠**: ユースケースにつきシステムのスコープを設定するボックスです。このボックス外のユースケースはすべて、対象のシステムのスコープ外であるとみなされます。
- **パッケージ**: 異なる要素をグループ化するために使える UML 図形。コンポーネント図と同様、これらのグループはファイル フォルダーとして示されます。

ユースケース図の例

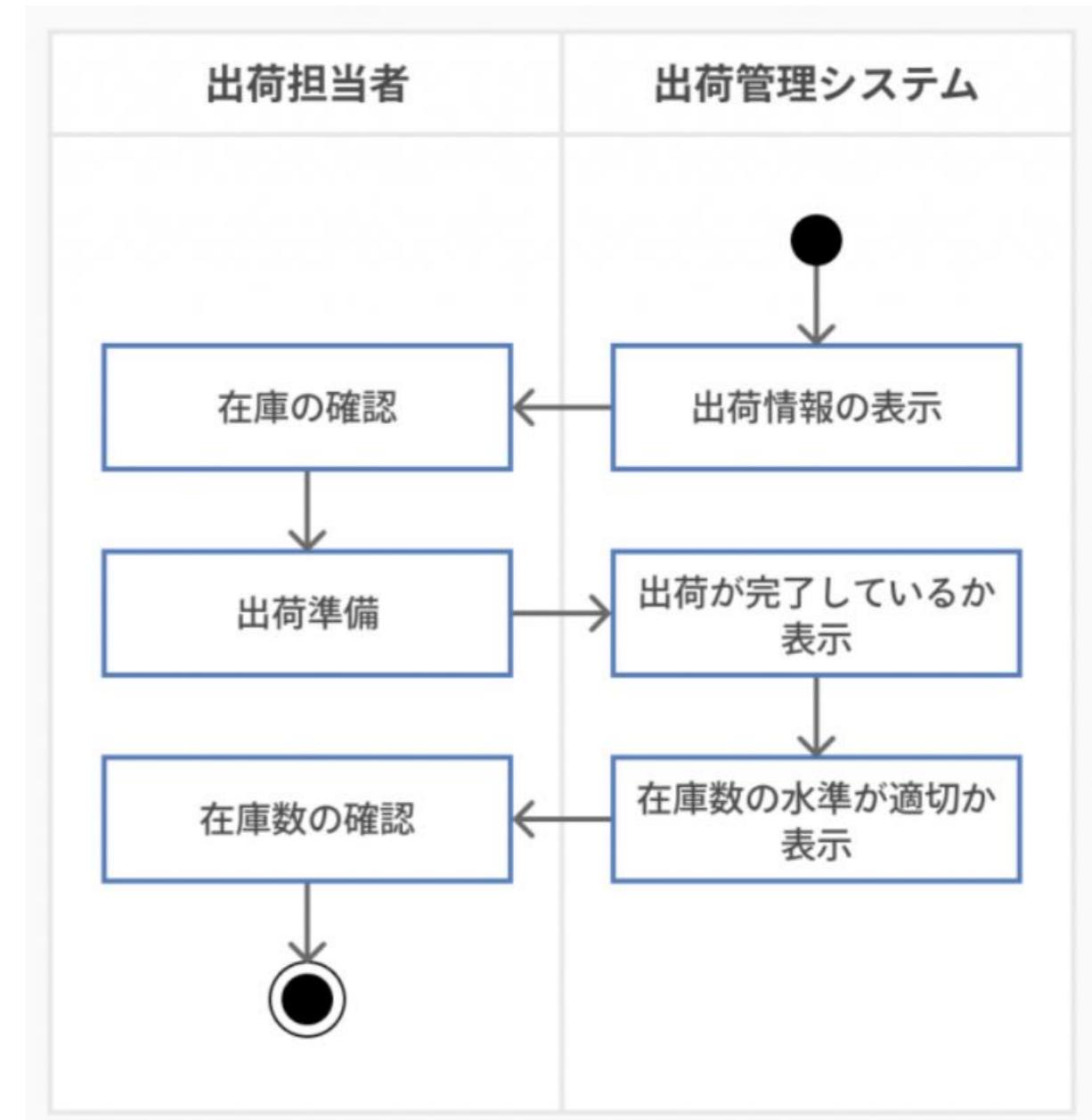




Question and answer

アクティビティ図

- アクティビティ図 [Activity Diagram] とは、連続する「実行」の遷移、つまり一連の「手続き」を表現するための図です。
- ある事象の開始から終了までの機能を実行される順序にしたがって記述します。
- 状態マシン図が実体の状態遷移を表すのに対し、アクティビティ図では実体の制御の流れを描写します。



アクティビティ図の役割

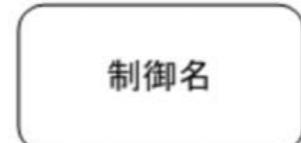
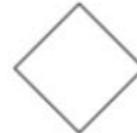
- アルゴリズムのロジックを示す。
- UML ユースケースで実行される手順を説明する。
- ユーザーとシステムの間の業務プロセスやワークフローを図示する。
- 複雑なユースケースを明確化し、プロセスの簡素化と改善を行う。
- メソッド、関数、操作などのソフトウェアアーキテクチャの要素をモデル化する。

アクティビティ図の基本コンポーネント

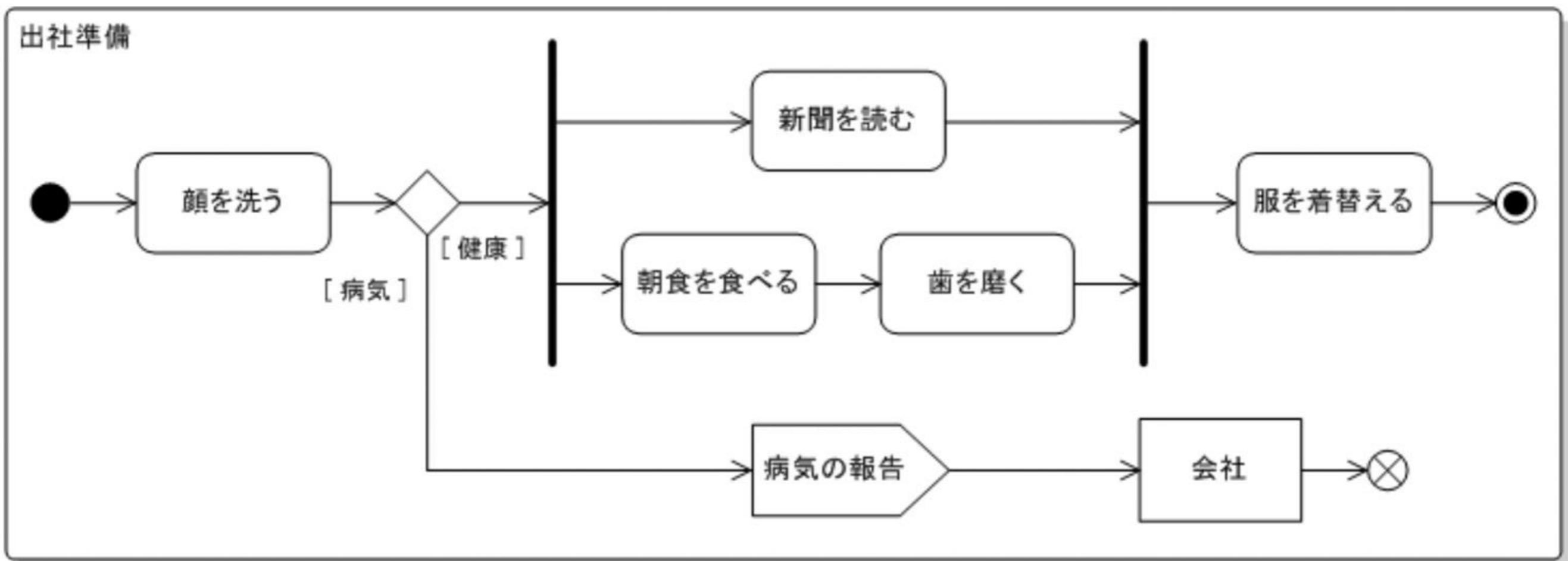
- **アクション**: ユーザーまたはソフトウェアが所定のタスクを実行するアクティビティの手順を指します。アクションは角丸の長方形で示されます。
- **判断ノード**: フロー内の条件分岐を意味し、ひし形で示されます。単一の入力と複数の出力を含みます。
- **制御フロー**: コネクターの別名で、図内の手順間のフローを示します。
- **開始ノード**: アクティビティの開始を意味し、黒い丸で示されます。
- **終了ノード**: アクティビティの最終段階を意味し、白い枠付きの黒い丸で示されます。

次へ 

 前へ

| 要素 | 表示形式 | 意味 |
|-------------------|--|---|
| 初期ノード |  | スコープ内で開始を表します。 |
| 最終ノード |  | スコープ内で終了を表します。 |
| アクションノード |  | 制御を表します。 |
| デシジョンノード / マージノード |  | 条件によるフロー分岐（デシジョンノード）もしくは、複数のフローの合流（マージノード）を表します。 |
| フォークノード / ジョインノード |  | 複数のフローが非同期に実行される（フォークノード）、もしくは、複数の非同期処理が終了する（ジョインノード）ことを表します。 |
| データストア |  | データーを保持するための記憶領域や装置を表します。 |

アクティビティ図の例



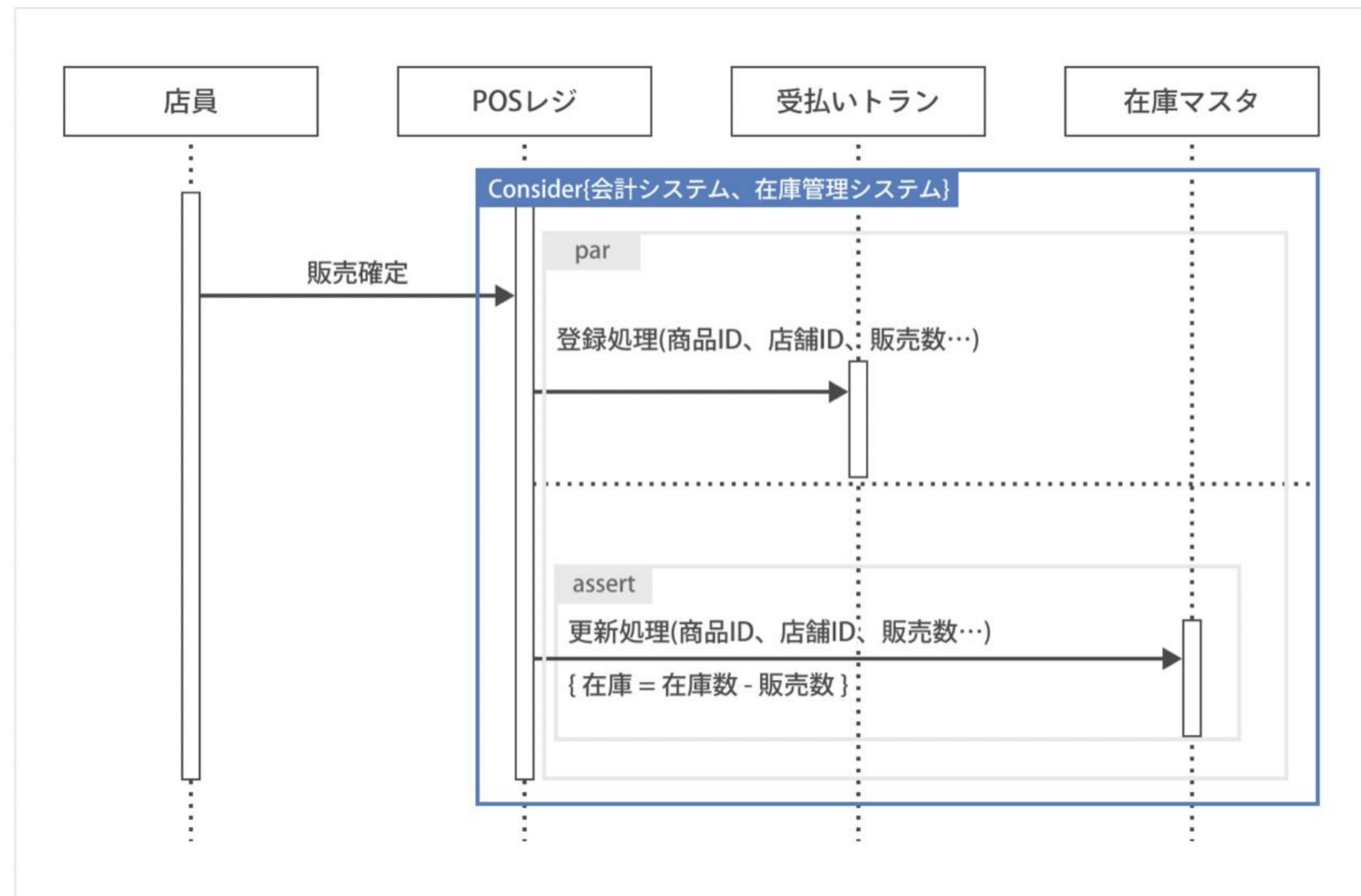


Question and answer

シーケンス図

- **シーケンス図**[Sequence Diagram]は、相互作用を経時的に示すものです。
- プログラムの処理の流れや概要について、具体的には**クラスやオブジェクト間のやり取り**を**時間軸**に沿って、図で表現します。
- UML の中では「相互作用図」の1つに位置付けられています。

シーケンス図の例



シーケンス図の役割

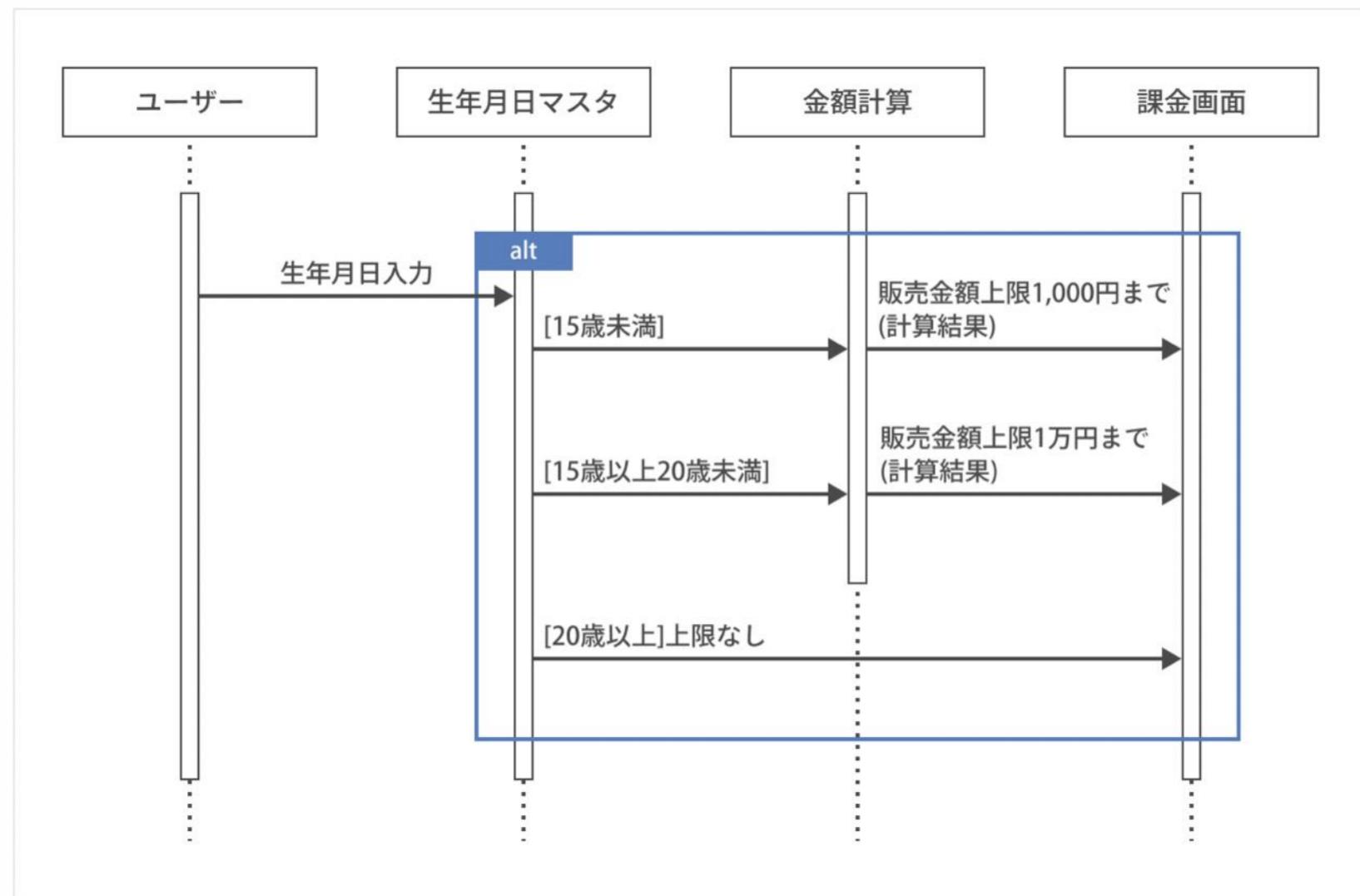
- UML のユースケースの詳細を示す。
- 高度な手順、機能や操作のロジックをモデル化する。
- プロセス完了までの過程におけるオブジェクトとコンポーネントの相互作用を確認する。
- 既存または将来のシナリオの詳細な機能を計画し、理解する。

シーケンス図の適用場面

- **利用シナリオ:** システムを使用しうる方法を示す図が利用シナリオです。システムのあらゆる利用シナリオのロジックを想定できているかを確認する上で適した方法です。
- **メソッドのロジック:** UML シーケンス図でユースケースのロジックの検討に用いるのと同様に、あらゆる機能、手順や複雑なプロセスのロジックの検討にシーケンス図を利用することができます。
- **サービスのロジック:** 異なるクライアントが使用する高次のメソッドとしてサービスを捉える場合に、そのサービスをマッピングする上でシーケンス図が役立ちます。

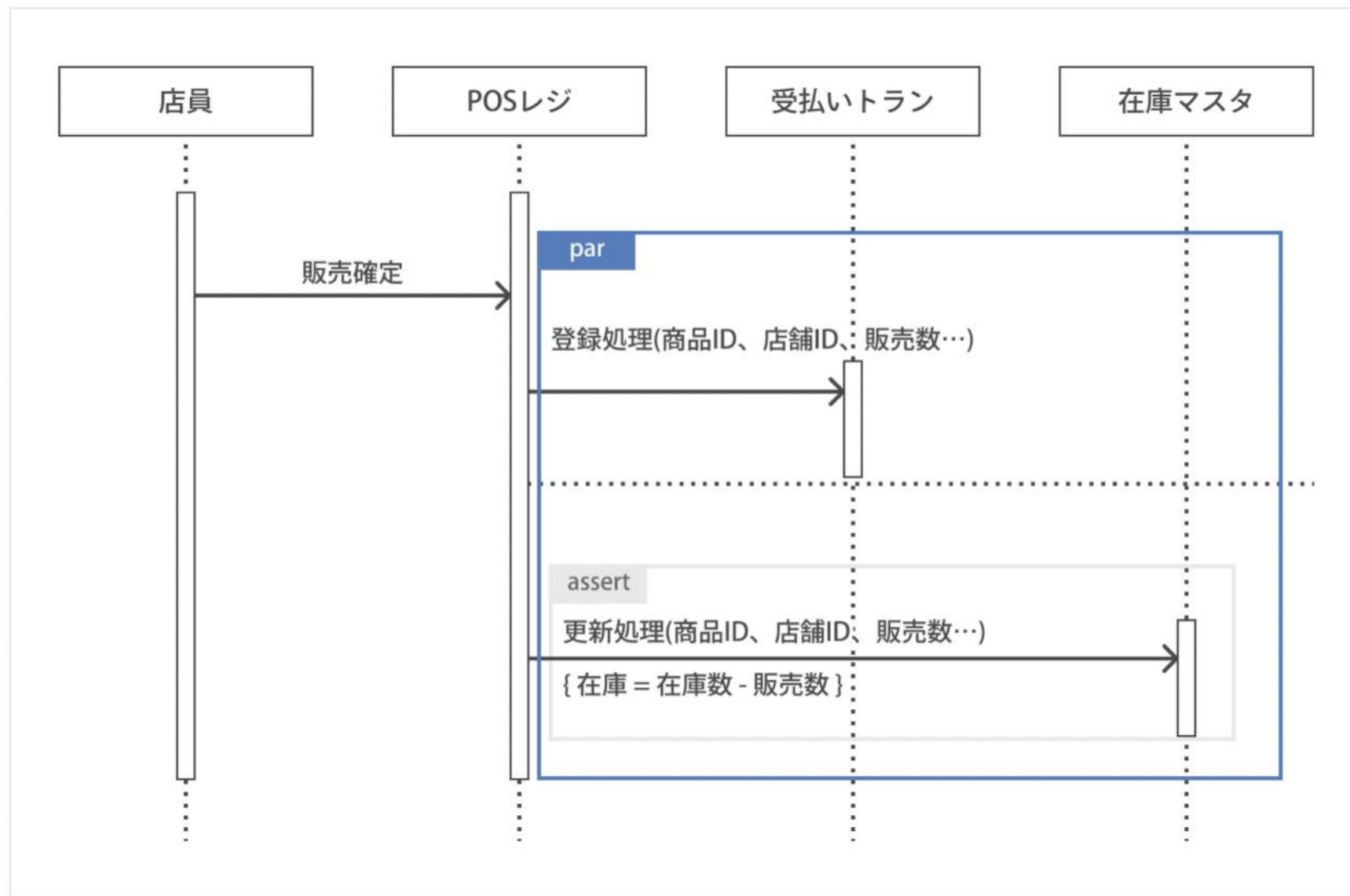
条件分岐

- 条件分岐（alt）は複合フラグメントを使用します。「[]」（ガードという括弧）内に分岐条件を記載し、各処理を点線で区切って記載します。



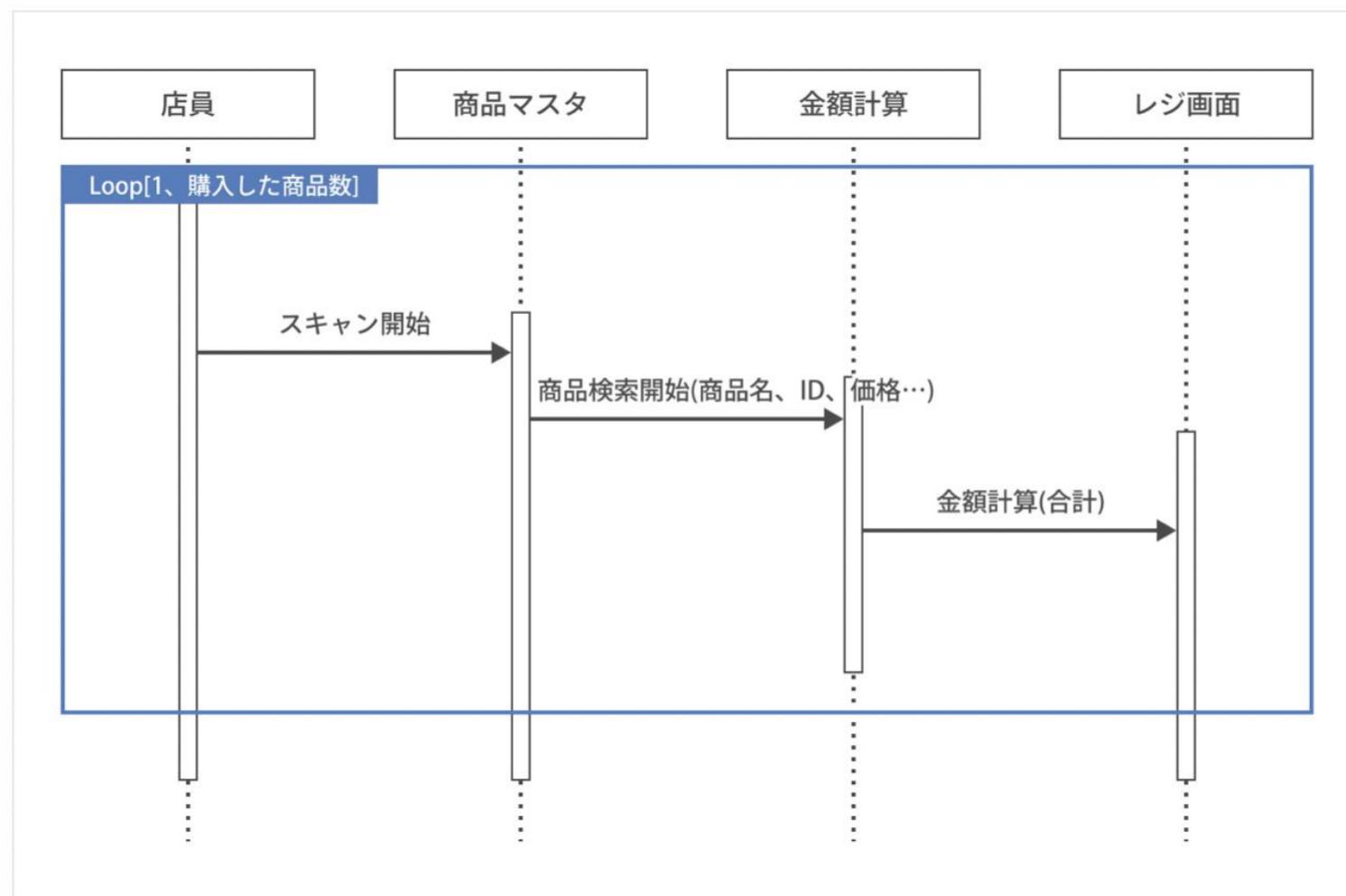
並列処理

- **並列処理 (par)** はその名の通り、1つのアクションに対して並列で処理が実行される場合に使用します。



ループ処理

- 複合フラグメント「ループ処理 (loop)」も名前通り、ループ処理するシステムを表すシーケンス図に使用されます。
- ループ条件の書き方は、「loop[開始条件、終了条件]」です。



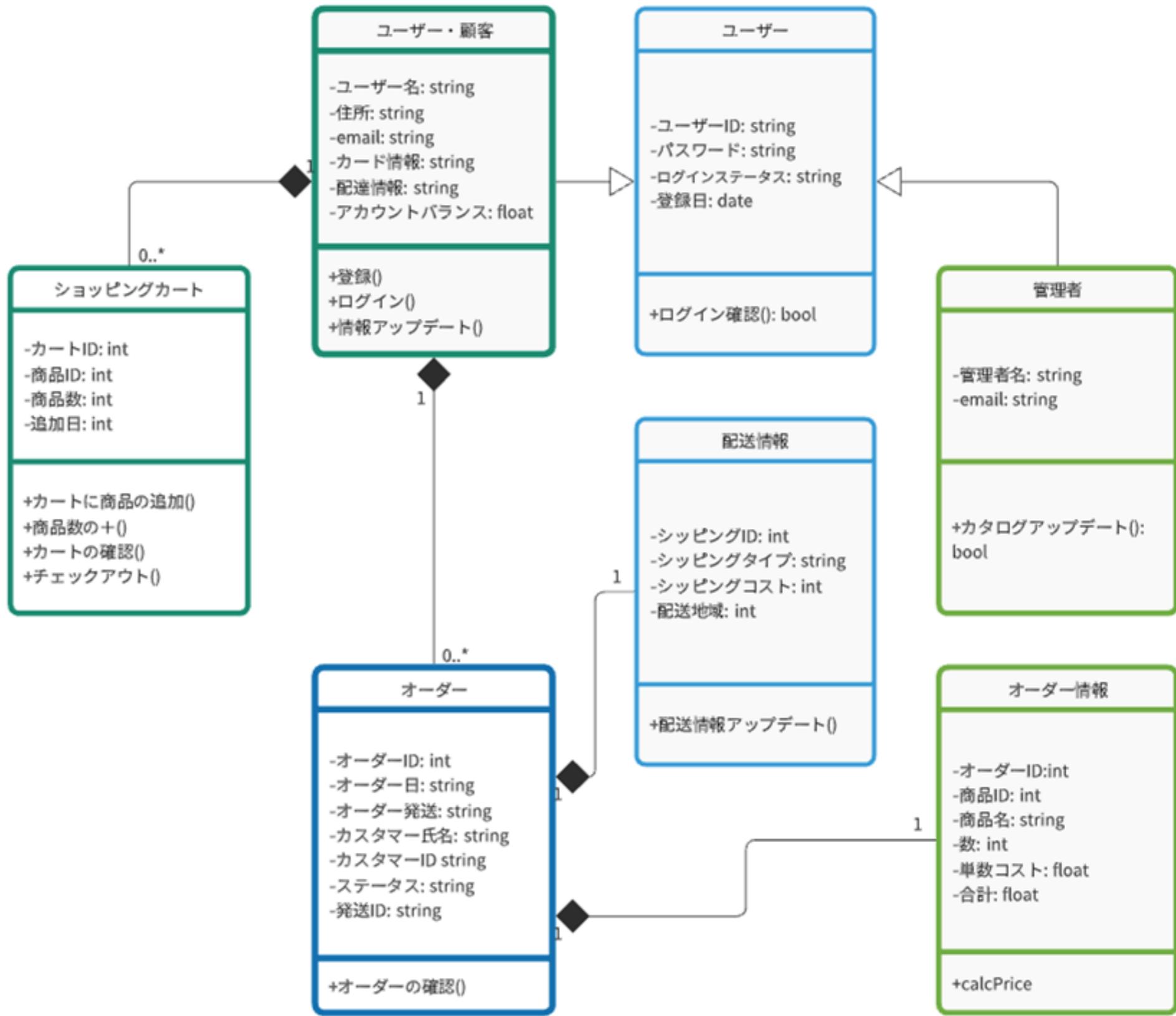


Question and answer

クラス図

- クラス図[Class Diagram]は、ソフトウェアアーキテクチャの文書化を目的としてソフトウェア技術者の間で広く用いられています。
- モデリングの対象となるシステム内に存在すべき構成要素を図式化したもので、構造図の一種です。

クラス図の例



クラス図の役割

- 複雑度にかかわらず、情報システムのデータモデルを図式化する。
- アプリケーションの概略図の全体的な概要について理解を深める。
- システムに特有のニーズを視覚的に表現し、その情報を組織全体に伝達する。
- 記述された構造を対象とした、プログラムや実装をする特定のコードをハイライトした詳細な図を作成する。
- システムで使用され、後に構成要素間で渡される実体型を実装に依存しない形で説明する。
- システム全体を可視化で表現できる。

クラス図の基本的な構成要素

- 上段: クラスの**名前**が含まれます。分類子の場合でも、オブジェクトの場合でも、このセクションは常に必須です。
- 中段: クラスの**属性**が含まれます。クラスの性質の説明に用いるセクションです。クラスの特定のインスタンスを記述する際にのみ必要となります。
- 下段: クラスの**操作**（メソッド）が含まれます。リスト形式で表示され、各行に1つずつ操作を記述します。これらの操作は、クラスがデータと相互作用する方法を示します。



メンバーのアクセス修飾子

- 各クラスには、**アクセス修飾子**（可視性）によりレベルの異なるアクセス権が付与されます。アクセスレベルと対応する記号は以下のとおりです：
 - 公開 (+)
 - 非公開 (-)
 - 保護 (#)
 - パッケージ (~)
 - 派生 (/)
 - 静的 (_)



クラス間相互関係の表現

| 関係 | 線形 |
|-----------------------|--------|
| 関連 (association) | ————— |
| 集約 (aggregation) | ◊————— |
| コンポジション (composition) | ◆————— |
| 依存 (dependency) | ←----- |
| 汎化 (generalization) | ↖————— |
| 実現 (realization) | ↖----- |

多重重度の表現

| 表記 | 意味 |
|---------------|--------|
| 1 | 1 個 |
| '0..n' or '*' | 0 以上 |
| 1..n | 1 以上 |
| 2..5 | 2 から 5 |



Question and answer



- 1 UML 図概要
- 2 UML 図の種類
- 3 Open API

OpenAPI

- UML 以外に、API を設計する時よく使っているツールを紹介します。
- OpenAPI とは、RESTful API を記述するためのフォーマットのこと。
- [!\[\]\(5a45c1c718321bad967123653ebfc4ea_img.jpg\) https://spec.openapis.org/oas/v3.1.0](https://spec.openapis.org/oas/v3.1.0)
- [!\[\]\(37bb49cbe58706990568dfa8462d7de4_img.jpg\) https://petstore.swagger.io](https://petstore.swagger.io)

RESTful API

- **RESTful API** とは、REST の原則に則って構築された Web システムの HTTP での呼び出しインターフェースのこと。主に以下の 4 つの原則から成る：

1. アドレス可能性 (Addressability)

提供する情報が *URI* を通して表現できること。全ての情報は *URI* で表現される一意なアドレスを持っていること。

2. ステートレス性 (Stateless)

HTTP をベースにしたステートレスなクライアント・サーバプロトコルであること。セッション等の状態管理はせず、やり取りされる情報はそれ自体で完結して解釈できること。

3. 接続性 (Connectability)

情報の内部に、別の情報や（その情報の別の）状態へのリンクを含めることができること。

4. 統一インターフェース (Uniform Interface)

情報の操作（取得、作成、更新、削除）は全て HTTP メソッド（GET、POST、PUT、DELETE）を利用すること。

OpenAPI

| フィールド名 | 必須 | 概要 |
|--------------|-----|---|
| openapi | YES | セマンティックなバージョニングを記述する。今回は3.0.0を用いる。詳しくはドキュメントを参照。 |
| info | YES | API のメタデータを記述する。 |
| servers | | API を提供するサーバーを記述する。配列で複数記述可能(STG、PROD 等)。 |
| paths | YES | API で利用可能なエンドポイントやメソッドを記述する。 |
| components | YES | API で使用するオブジェクトスキーマを記述する。 |
| security | | API 全体を通して使用可能なセキュリティ仕様 (OAuth 等) を記述する。 |
| tags | | API で使用されるタグのリスト。各種ツールによってパースされる際は、記述された順序で出力される。タグ名はユニークで無ければならない。 |
| externalDocs | | 外部ドキュメントを記述する (API 仕様書等)。 |

info

example.yaml

```

info:
  title: Sample API
  description: A short description of API.
  termsOfService: http://example.com/terms/
  contact:
    name: API support
    url: http://www.example.com/support
    email: support@example.com
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
  version: 1.0.0

```

必須 概要 フィールド名

| | | |
|----------------|-----|--|
| title | YES | API の名称。 |
| description | | API の簡潔な説明。 CommonMark* シンタックス が使える。 |
| termsOfService | | API の利用規約。URL 形式で なければならない。 |
| contact | | コンタクト情報（サポート ページの URL やメールアドレス等）。 |
| license | | ライセンス情報。ライセンス ページの URL も記述可能。 |
| version | YES | API ドキュメントのバージョン。 |

* <https://spec.commonmark.org/0.30/>

servers

example.yaml

```
servers:  
  - url: https://dev.sample-server.com/v1  
    description: Development server  
  - url: https://stg.sample-server.com/v1  
    description: Staging server  
  - url: https://api.sample-server.com/v1  
    description: Production server
```

フィールド名 必須 概要

| | | |
|-------------|-----|----------------|
| url | YES | API サーバーの URL。 |
| description | | API サーバーの説明。 |

paths

example.yaml

```

paths:
  # paths オブジェクト
  /users:
    # path item オブジェクト
    get: # GET
      # Operationオブジェクト
      tags:
        - users
      summary: Get all users.
      description: Returns an array of User model
      parameters: []
      responses: # レスポンス定義
        '200': # HTTP status
          description: A JSON array of User model
          content:
            application/json: # レスポンスの形式指定
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/User' # 参照するモデル
        example: # サンプルデータ
          - id: 1
            name: John Doe
          - id: 2
            name: Jane Doe

```

フィールド名 概要

| | |
|-----------------------|--|
| summary | エンドポイントのサマリ。 |
| description | エンドポイントの簡潔な説明。CommonMark シンタックスを使用可能。 |
| get, post, delete ... | HTTP メソッドを定義。GET、PUT、POST、DELETE、OPTIONS、DELETE、PATCH、TRACE が使用可能。 |

components

example.yaml

```
components:  
  schemas: # スキーマオブジェクトの定義  
    User: # モデル名  
      type: object # 型  
      required: # 必須フィールド  
        - id  
    properties:  
      id: # プロパティ名  
        type: integer # 型  
        format: int64 # フォーマット(int32, int64等)  
    name:  
      type: string
```

フィールド名 概要

| | |
|----------------|-----------------------|
| schemas | User や Product 等のモデル。 |
| request Bodies | リクエストボディ。 |
| responses | API レスポンス。 |
| headers | リクエストヘッダ。 |
| parameters | リクエストパラメータ。 |



Question and answer

まとめ

Sum Up

1. UML 図:

- ① ユースケース図、
- ② アクティビティ図、
- ③ シーケンス図、
- ④ クラス図。

2. OpenAPI:

- ① RESTful API の定義。
- ② OpenAPI の基本文法。

THANK YOU!