オブジェクト指向技術 第2回
― オブジェクト指向開発プロセス ―

Keyword: クラス、インスタンス、カプセル化、
オブジェクト指向分析、UML、ユースケース図

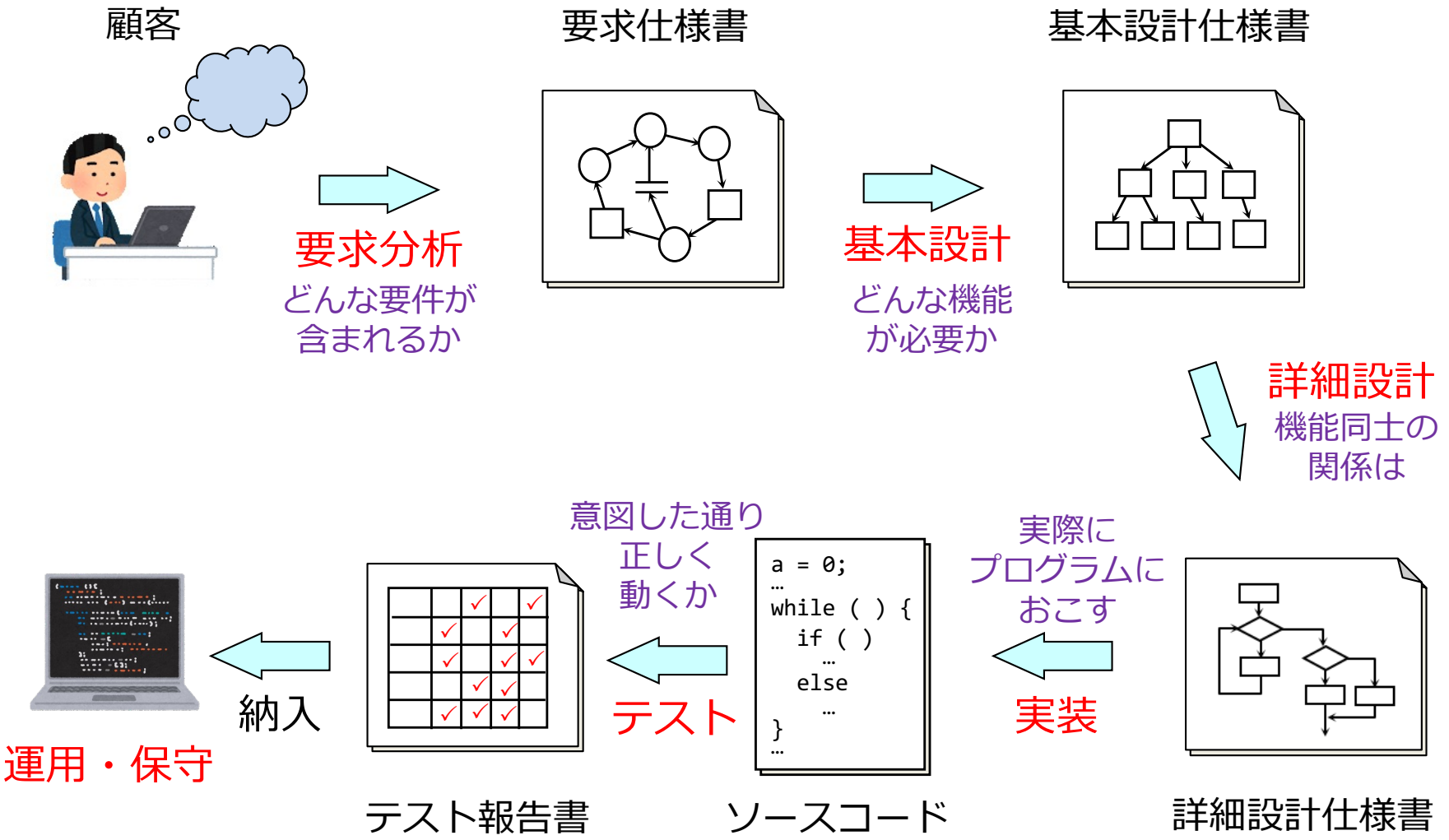
立命館大学　情報理工学部
槇原　絵里奈
Mail: makihara@fc.ritsumei.ac.jp

講義内容

- オブジェクト指向開発プロセス
  - UML
    - ユースケース図
    - クラス図
    - パッケージ図
    - アクティビティ図
    - シーケンス図
    - 状態機械図

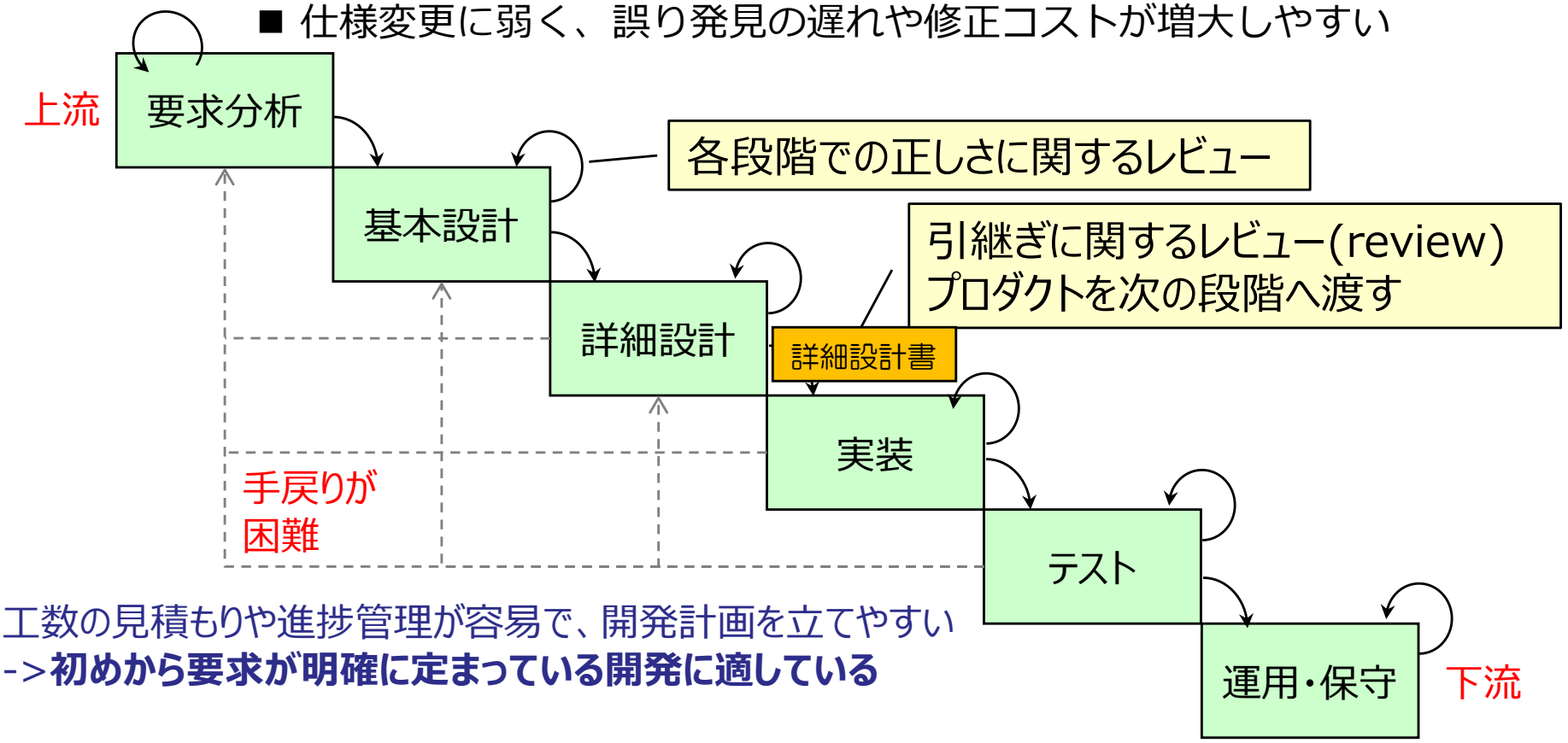
ソフトウェア開発プロセス

■ソフトウェア開発の作業工程



ウォーターフォールモデル

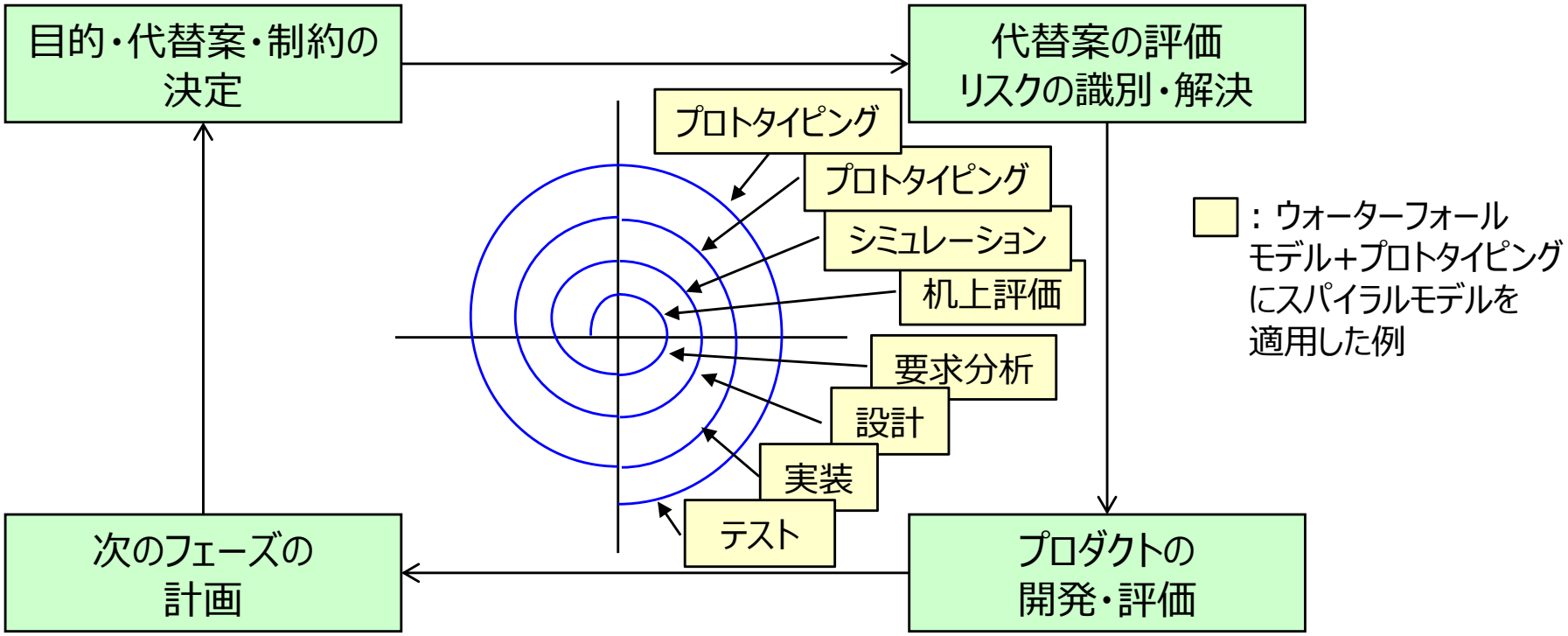
- 滝(waterfall)に例えられるプロセスモデル
- トップダウン(top-down)な開発プロセス
  - 要求分析から運用・保守まで一直線に進む
  - **工程の後戻り(手戻り)は原則想定しない**
    - 仕様変更に弱く、誤り発見の遅れや修正コストが増大しやすい



工数の見積もりや進捗管理が容易で、開発計画を立てやすい
->**初めから要求が明確に定まっている開発に適している**

スパイラルモデル

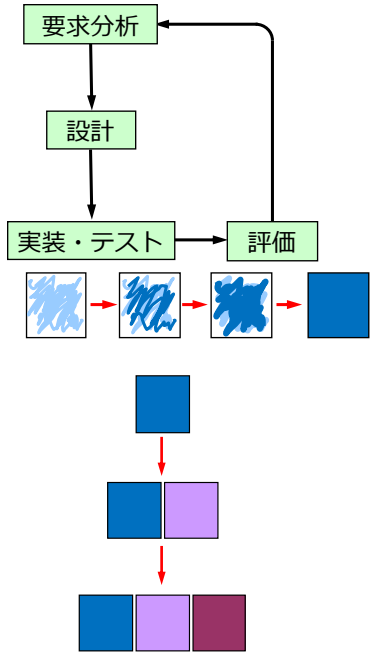
■ 下の4フェーズを繰り返して、段階的に開発を進める
- フィードバック(feedback)を元に徐々にシステムを開発



- スパイラルモデル自体はプロセスモデルではなく、他のプロセスモデルと組み合わせて用いる(既存のプロセスモデルを改善する)

進化型プロトタイピング(prototype)

- **プロトタイプ(施策モデル)**に修正を加えていき、最終的なソフトウェアとする手法
  - 機能が明確な部分から開発
  - すべての仕様を確認しながら作るため、大幅な手戻りを予防できる
  - 分析・設計・実装を繰り返す
    - iterative (反復的)
      - すべての部分を一度完成させ、プロトタイプとして提供
      - リリース(release)ごとに各部の完成度を高めていく
    - incremental (漸進的)
      - ソフトウェアを独立性の高い部分に分割
      - リリースごとに機能を追加する
    - 両者を組み合わせて使うのが普通



オブジェクト指向ソフトウェア開発

- オブジェクト指向分析 (OOA: OO Analysis)
  - 開発するシステムを定義
  - 作成するモデルは環境に非依存
- オブジェクト指向設計 (OOD: OO Design)
  - 定義したシステムをソフトウェアとして実現する方法を決定
  - 環境に依存する部分を考慮
- オブジェクト指向プログラミング (OO Programming)
  - プログラムの記述、テスト

OOA、OOD、OOPの過程で一貫して
クラス、オブジェクトを中心に考える
→ 開発工程の移行がシームレス(seamless)

確認問題

■ 以下の各文は正しいか。○か×で答えよ。

- オブジェクト指向ソフトウェア開発では、開発の比較的初期からクラスやオブジェクトを扱う。
- オブジェクト指向ソフトウェア開発では、要求分析や設計は重要ではない。
- オブジェクト指向ソフトウェア開発は、ウォーターフォールモデルに従うのが一般的である。
- オブジェクト指向ソフトウェア開発では、進化型プロトタイピングは適用できない。

講義内容

■オブジェクト指向開発プロセス

■UML

- ●ユースケース図
- ●クラス図
- ●パッケージ図
- ●アクティビティ図
- ●シーケンス図
- ●状態機械図

UML

- Unified Modeling Language
  - ソフトウェアのモデルを表現するための**図式言語**と**そのメタモデル(言語の概念)**を定義
  - 1997年にOMG(object management group)が発表
    - その後、改訂が繰り返されている (2017年 UML 2.5.1)
  - 全世界的に使われる
    → ソフトウェア開発時の**共通語彙**
    設計の意志決定の伝達が容易

    何の機能が必要か
    どの機能とどの機能が関わっているか
  - 開発プロセスとは独立 (オブジェクト指向との親和性が高い)
- モデル：検討すべき項目に関連する重要なものを抜き出して記述したもの
  → ソフトウェアのモデル：ソフトウェアの構造・振る舞い等の重要な部分

  ・どのようなクラスで構成されるか
  ・どのような順番で処理を実行するか
  ・どのようにオブジェクトが状態遷移するか　等々

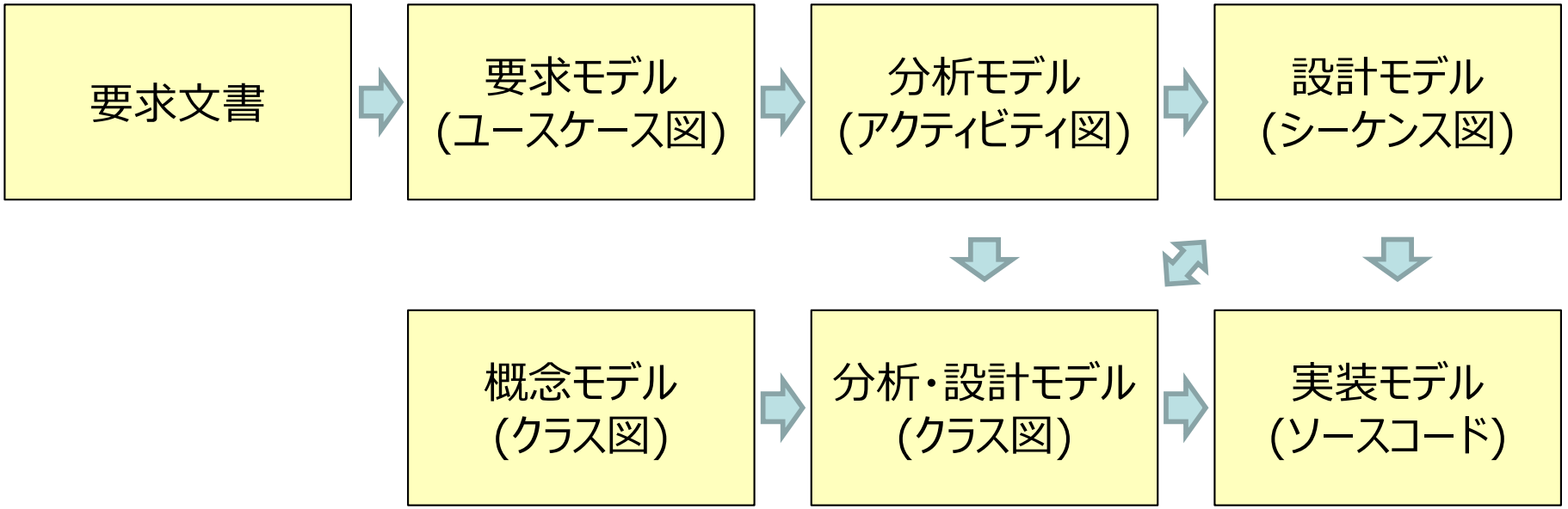
UML diagram

- 構造(structure)に関するダイアグラム
  - クラス図 (class diagram)
    - クラスの構造(属性や操作)とクラス間の静的な関係
  - オブジェクト図 (object diagram)
    - ある時点でのオブジェクトの状態とオブジェクト間の関係
  - パッケージ図 (package diagram)
    - パッケージの構成とパッケージ間の依存関係
  - 複合構成図 (composite structure diagram)
    - 実行時のクラスの内部構造
  - コンポーネント図 (component diagram)
    - コンポーネントの構造と依存関係
  - 配置図 (deployment diagram)
    - システムにおける物理的な配置

UML diagram

- 振る舞い(behavior)に関するダイアグラム
  - ユースケース図 (use-case diagram)
    - システムの提供する機能と利用者の関係
  - アクティビティ図 (activity diagram)
    - 作業の順序と並行性
  - シーケンス図 (sequence diagram)
    - オブジェクト間の相互作用の時系列
  - コミュニケーション図 (communication diagram)
    - オブジェクト間の相互作用のリンク
  - タイミング図 (timing diagram)
    - オブジェクトの相互作用のタイミング
  - 相互作用概要図 (interaction overview diagram)
    - シーケンス図とアクティビティ図の概要
  - 状態機械図 (state machine diagram)
    - オブジェクトの状態とイベントによる状態遷移

オブジェクト指向分析とUML

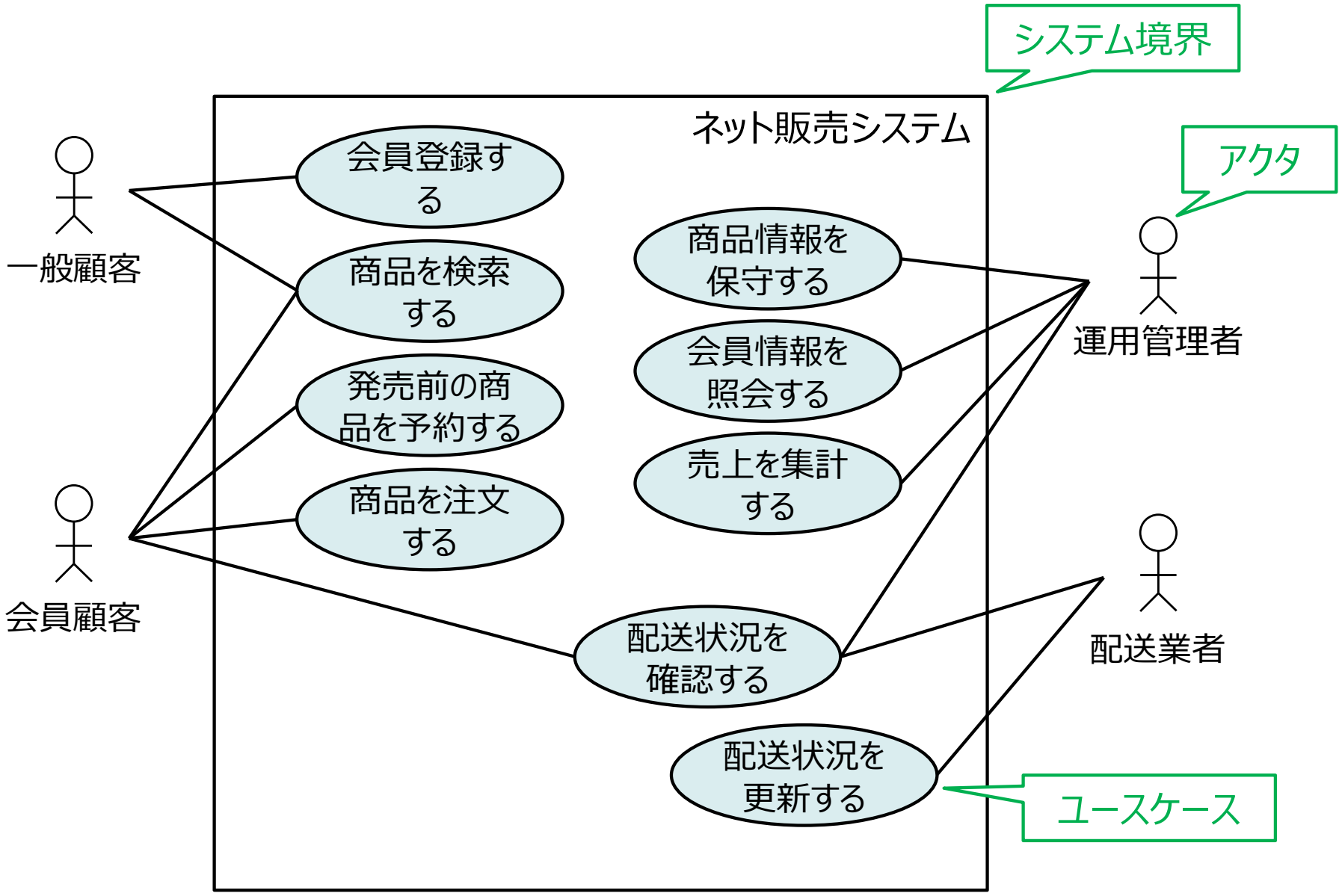


概念モデル： ソフトウェアとして実現するべき対象に含まれる概念を明らかにし、
どのようなクラスが必要かを確定。
分析モデル： ソフトウェアの挙動を機能ごとに整理し、クラスの責任範囲を確定。
設計モデル： クラスの名前、属性、振る舞いやクラス間の関連を確定。
拡張性や再利用性を考慮する必要。

ユースケース図(use case diagram)

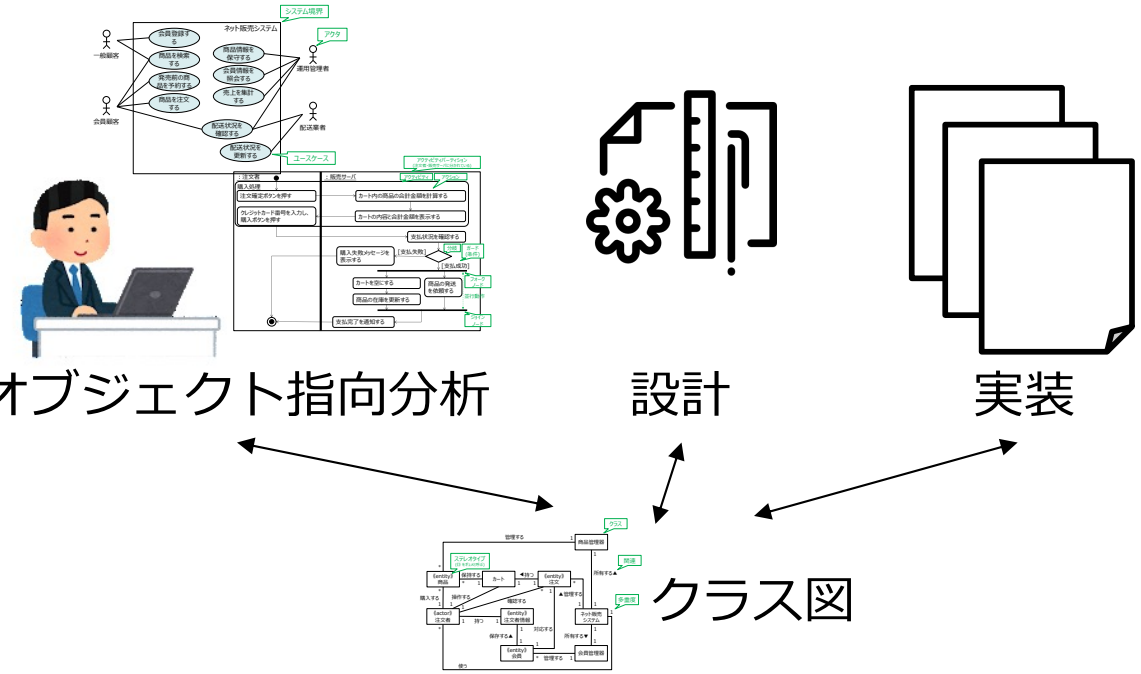
- 利用者がどのようにシステムを使用するのかを表す
  - システムの**内部と外部の境界を定める**
  - システムの機能ごとに作成
- 主な構成要素
  - **アクタ(actor)**
    - システムに対して利用者が果たす役割を表現
    - 役割ごとに異なるアクタが存在
    - 外部システムでもよい
  - **ユースケース(use case)**
    - 外部から見たシステムの振る舞いを表現
  - **ユースケース記述**
    - ユースケースの詳細を記述
    - 一般的に、シナリオ(scenario)により記述
      - 利用者とシステム間の対話を表す一連の手順
    - 正常系だけでなく異常系(例外処理等)も記述

(例) ユースケース図

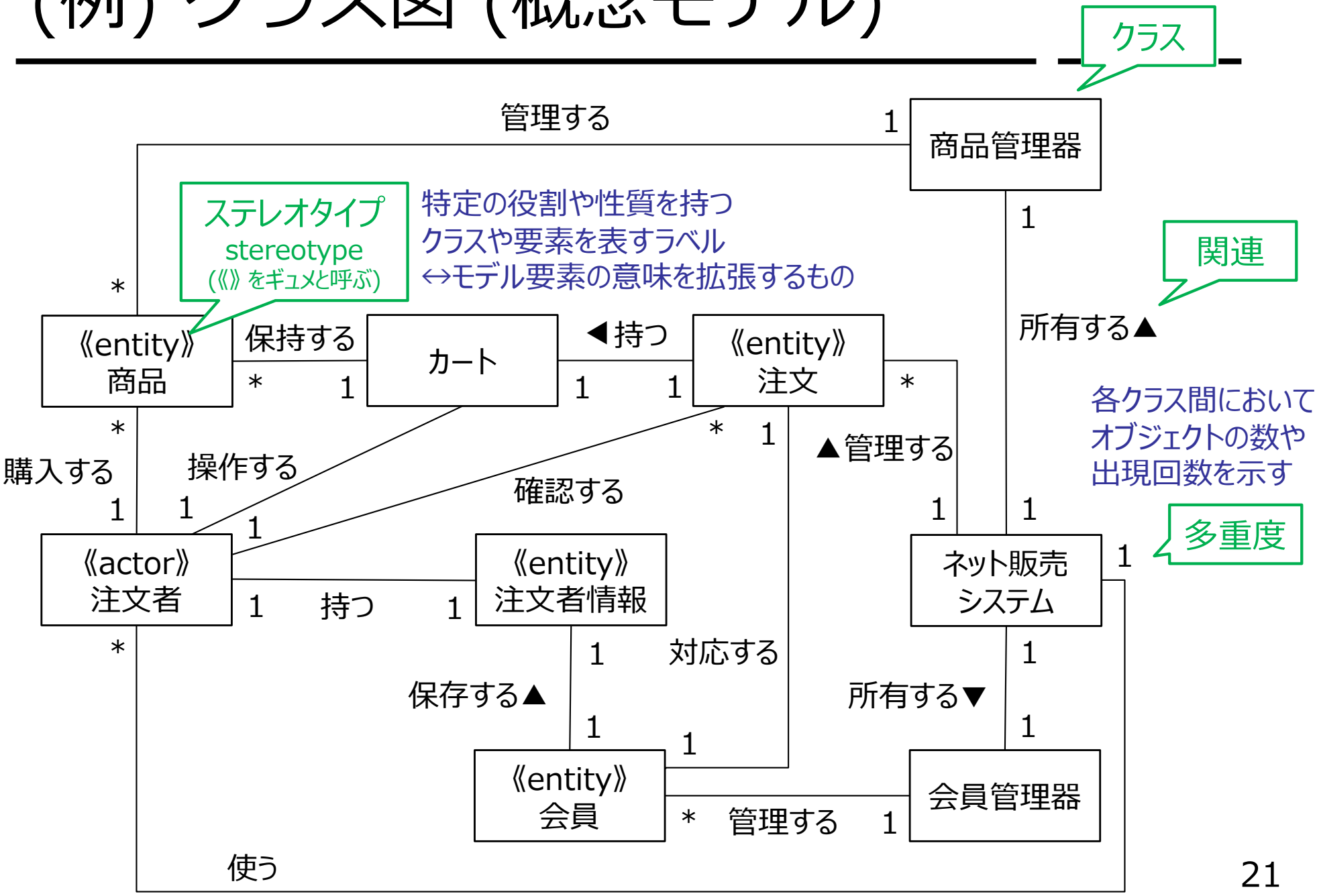

システム境界
ネット販売システム
アクタ
一般顧客
会員顧客
運用管理者
配送業者
会員登録する
商品を検索する
発売前の商品を予約する
商品を注文する
商品情報を保守する
会員情報を照会する
売上を集計する
配送状況を確認する
配送状況を更新する
ユースケース

クラス図(class diagram)

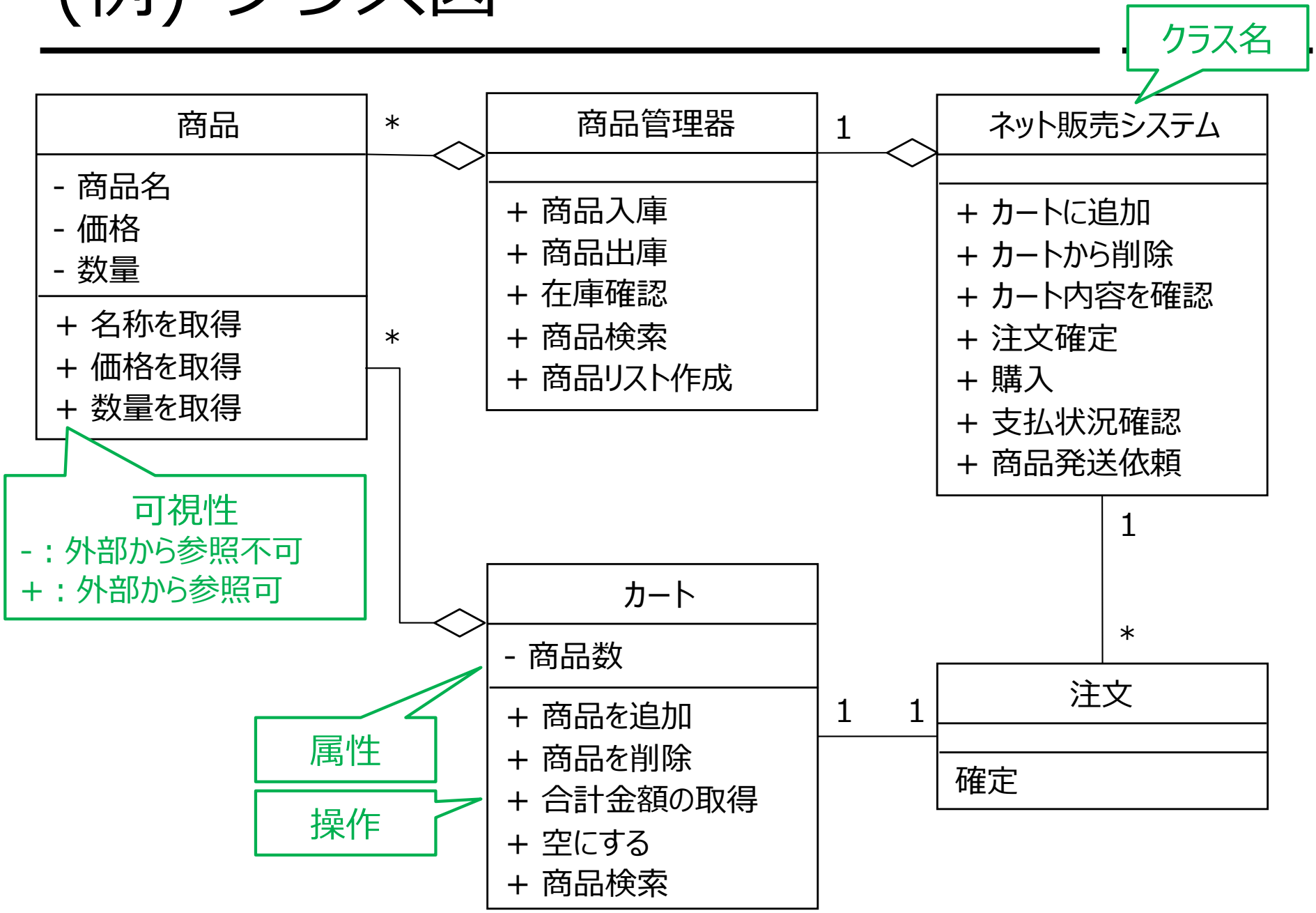
- クラスの内部構造(属性・操作)とクラス間の**静的(static)な関係**を表現
  - 稼働時に変化しない設定や属性など↔動的(dinamic)
- オブジェクト指向分析・設計・実装をつなぐ中心的な図
- 主な構成要素
  - **クラス**
  - **関連**



(例) クラス図 (概念モデル)


クラス
管理する
1
商品管理器
1
ステレオタイプ
stereotype
(《》をギュメと呼ぶ)
特定の役割や性質を持つ
クラスや要素を表すラベル
↔モデル要素の意味を拡張するもの
関連
*
《entity》
商品
保持する
*
1
カート
◀持つ
1
1
《entity》
注文
*
所有する▲
各クラス間において
オブジェクトの数や
出現回数を示す
多重度
*
購入する
1
1
操作する
1
確認する
*
1
▲管理する
1
1
《actor》
注文者
1
持つ
1
《entity》
注文者情報
ネット販売
システム
1
*
1
対応する
保存する▲
1
1
所有する▼
1
1
《entity》
会員
*
管理する
1
会員管理器
使う

(例) クラス図


クラス名
商品
- 商品名
- 価格
- 数量
＋ 名称を取得
＋ 価格を取得
＋ 数量を取得
*
*
商品管理器
＋ 商品入庫
＋ 商品出庫
＋ 在庫確認
＋ 商品検索
＋ 商品リスト作成
1
ネット販売システム
＋ カートに追加
＋ カートから削除
＋ カート内容を確認
＋ 注文確定
＋ 購入
＋ 支払状況確認
＋ 商品発送依頼
1
*
可視性
-：外部から参照不可
＋：外部から参照可
カート
- 商品数
＋ 商品を追加
＋ 商品を削除
＋ 合計金額の取得
＋ 空にする
＋ 商品検索
1
1
注文
確定
属性
操作

カプセル化 (encapsulation)

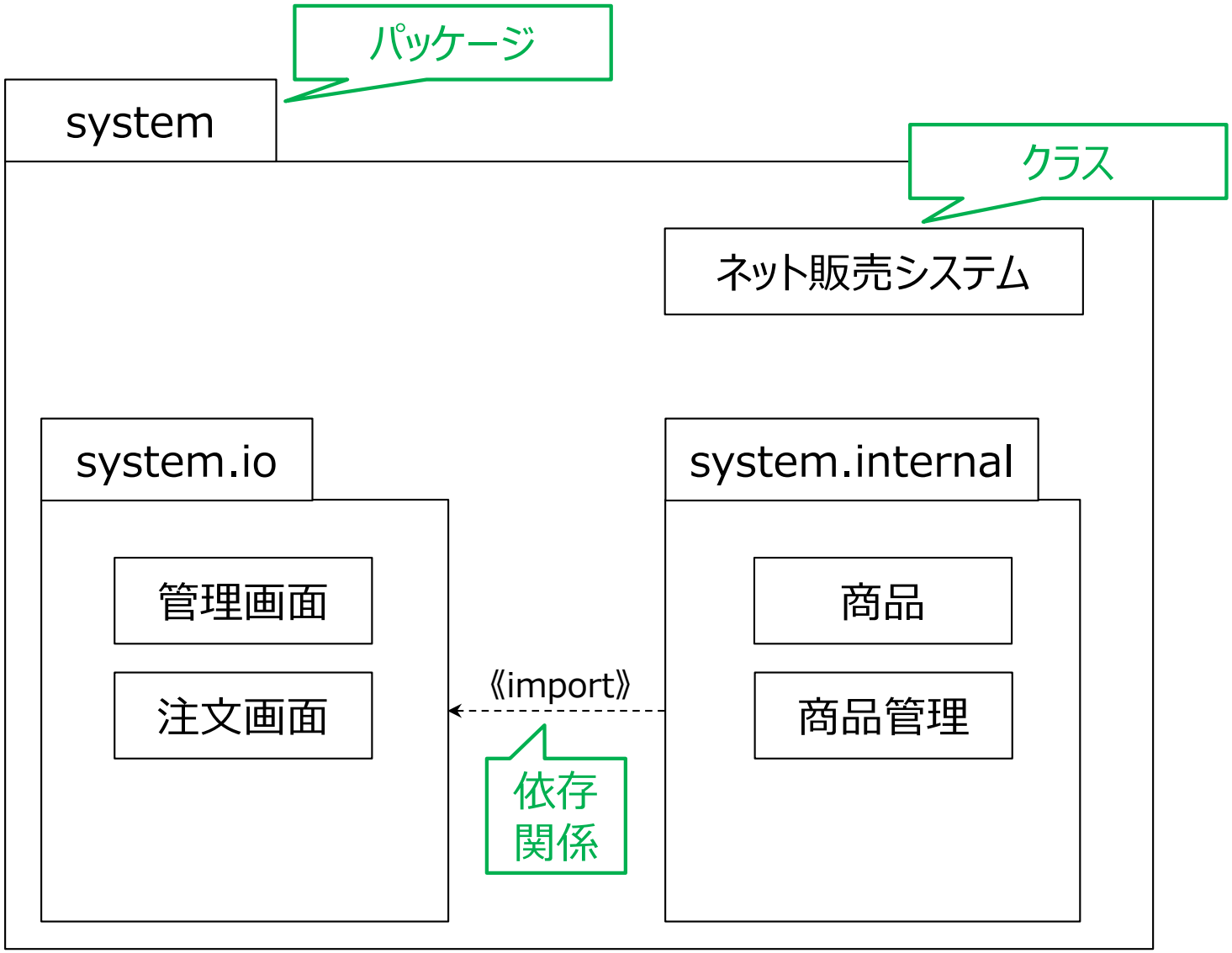
- 情報隠蔽ともいう
- クラスの一部を外部からアクセスできないようにすること(例：private)
  - 外部から利用する方法(インタフェース)を限定
    - この機能を使いたいならこれを使ってね
- 利点
  - どう使えば良いか・いけないかが明確
  - 変更の影響箇所が分かりやすくなる
    - 特定のクラスの機能はそのクラス内で修正
    - 他のクラスまで参照する必要がなくなる
  - 不正な操作や不適切な変更の防止

パッケージ図(package diagram)

- パッケージ(package)：モデル要素(クラス等)がどのようにグループ化されているかを示す

- 主な構成要素
  - パッケージ
  - クラス
  - 依存関係(import, merge)

(例) パッケージ図

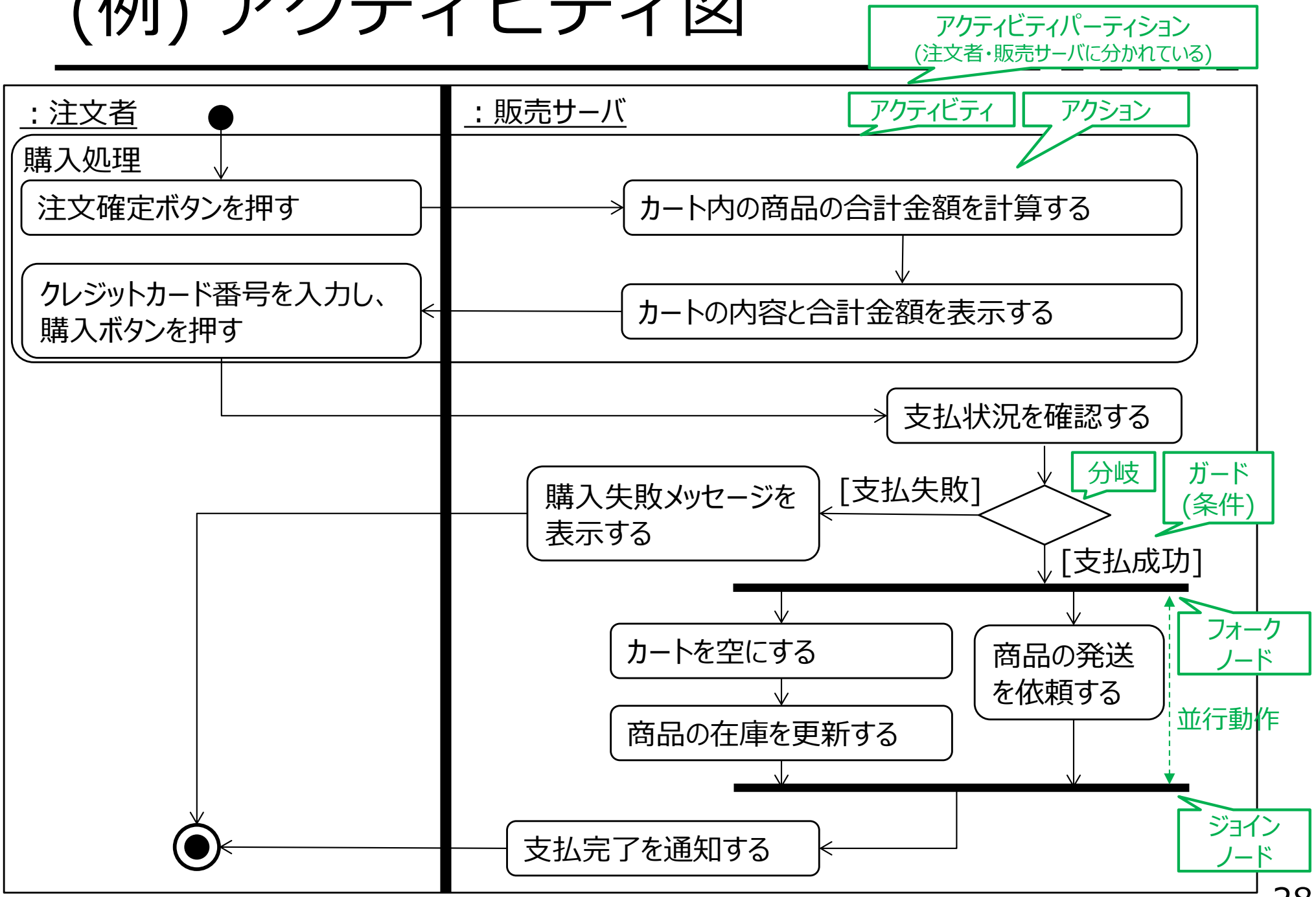

パッケージ
system
クラス
ネット販売システム
system.io
管理画面
注文画面
《import》
依存
関係
system.internal
商品
商品管理

アクティビティ図(activity diagram)

- 処理の**実行手順**を表す
- 主な構成要素
  - アクティビティ(activity)：一連の処理。複数のアクションで構成
  - アクション (action)：アクティビティを構成する処理の単位(他のアクションを内包できない)
  - 制御フロー (control flow)：制御の流れ。アクションの順序を示す
  - アクティビティパーティション (activity partition)：アクションを実行する主体やフェーズごとの切り分け。スイムレーン(swimlane)とも呼ばれる

ある機能を実現するために
必要な作業の順序を示す

(例) アクティビティ図

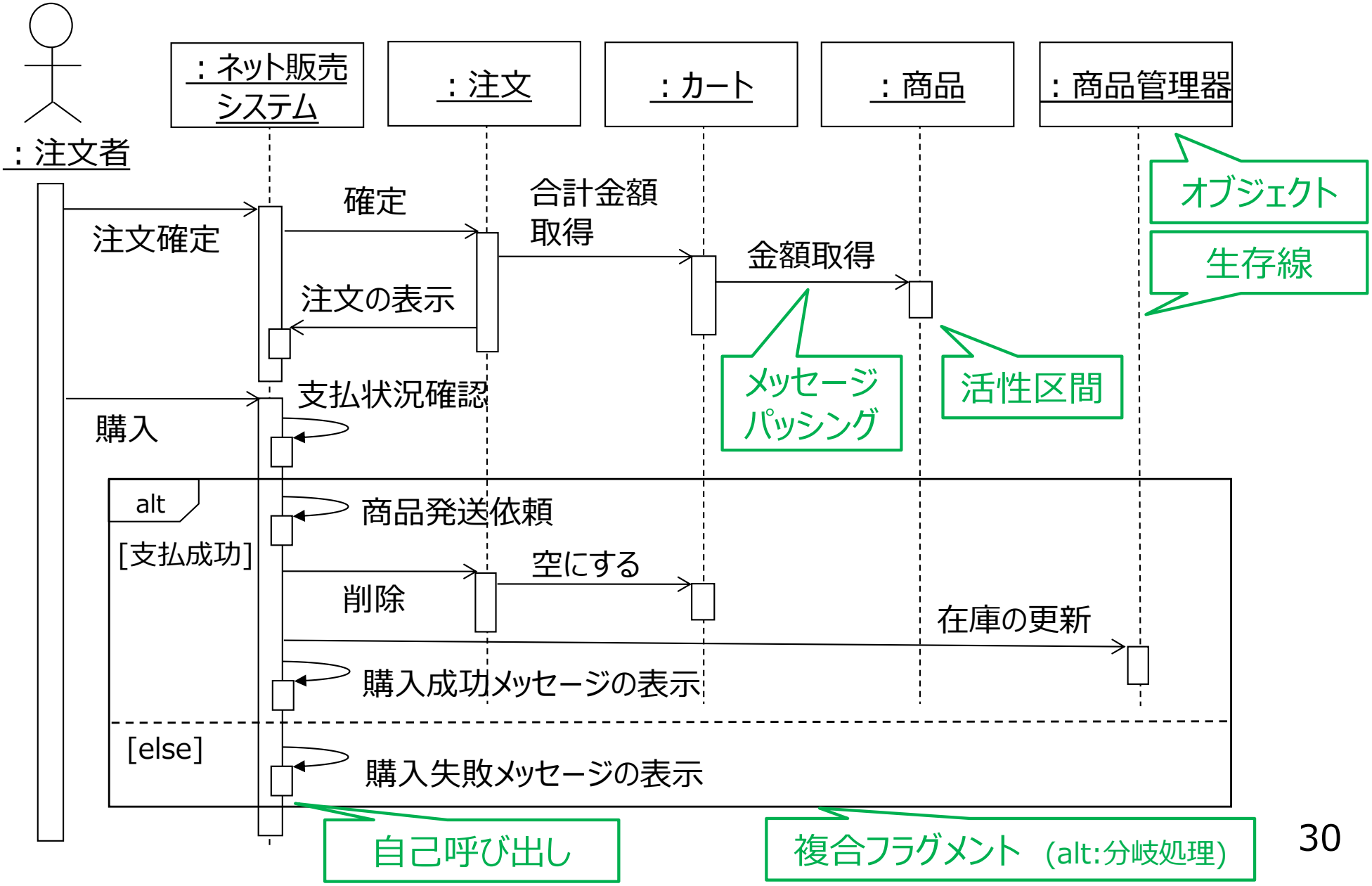


シーケンス図(sequence diagram)

- オブジェクト間の**相互作用** (メッセージのやり取り) を時系列に沿って表現
- 主な構成要素
  - アクタ・オブジェクト：メッセージやり取りの主体
  - 生存線(lifeline)：各アクタ・オブジェクトから伸ばした線。活性区間(処理を行っている時間)は太く(矩形で)示す
  - メッセージパッシング：オブジェクト間でやり取りされるメッセージ
  - 複合フラグメント：相互作用の一部に特別な意味を付加(e.g., 条件分岐、繰り返し)

(例) シーケンス図

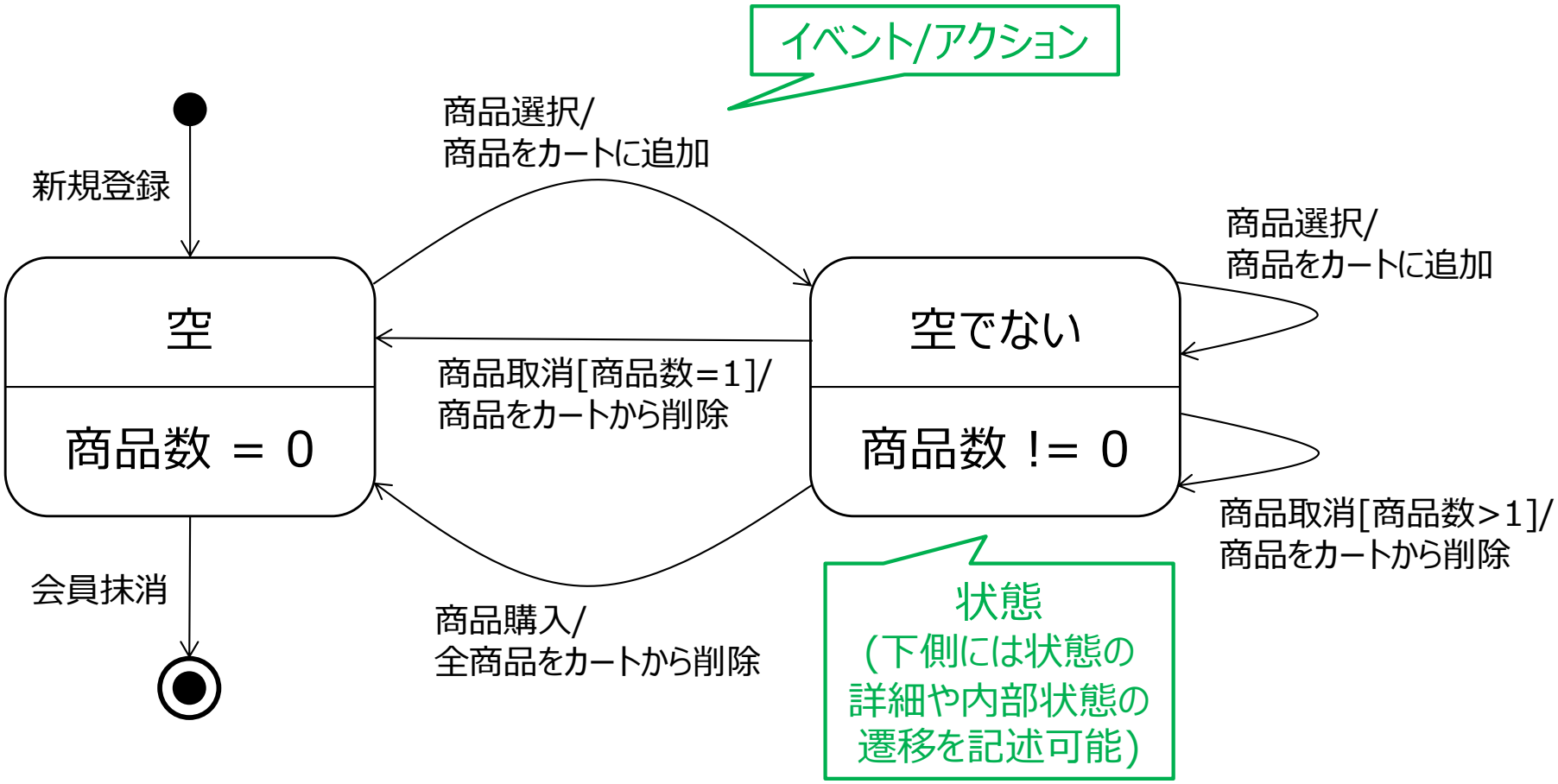
オブジェクト間のメッセージ
送受信の時系列



状態遷移図(State-Transition Diagrams)

- 特定のオブジェクトの状態遷移を示す
  - 1つの状態が内部状態を含むこともできる
  - ステートマシン図(state machine diagram)とも呼ばれる
- 主な構成要素
  - 開始擬似状態
  - 終了状態
  - 状態：オブジェクトの状態
  - 状態遷移：どのようなイベントが発生すると状態間の遷移が発生するかを示す

(例) 状態遷移図


イベント/アクション
新規登録
商品選択/
商品をカートに追加
空
商品数 = 0
空でない
商品数 != 0
商品選択/
商品をカートに追加
商品取消[商品数=1]/
商品をカートから削除
商品取消[商品数>1]/
商品をカートから削除
会員抹消
商品購入/
全商品をカートから削除
状態
(下側には状態の
詳細や内部状態の
遷移を記述可能)

確認問題

- 以下の各文は正しいか。○か×で答えよ。
  - UMLを使う利点として、設計の意思伝達が容易になることが挙げられる。
  - UMLはオブジェクト指向ソフトウェア開発においてのみ使用される。
  - カプセル化の利点として、コード変更の影響箇所が分かりやすくなることが挙げられる。
- 各説明に合う適切な語句を選択肢から選択せよ。
  - ソフトウェアの構造・振る舞い等の重要な部分を抽出したもの。
    選択肢：モジュール、モデル、言語
  - クラスの一部を外部からアクセスできないようにすること。
    選択肢：インスタンス化、カプセル化、モジュール化
- 以下の説明に合うUMLの図の名称を答えよ。
  - クラスの構造とクラス間の静的な関係を表現する。
  - 動的なオブジェクトの状態とオブジェクト間の関係を表現する。
  - システムの提供する機能と利用者の関係を表現する。
  - オブジェクト間の相互作用の時系列を表現する。
  - 作業の順序と並行性を表現する。
  - オブジェクトの状態とイベントによる状態遷移を表現する。