オブジェクト指向論

丸山勝久

立命館大学 情報理工学部

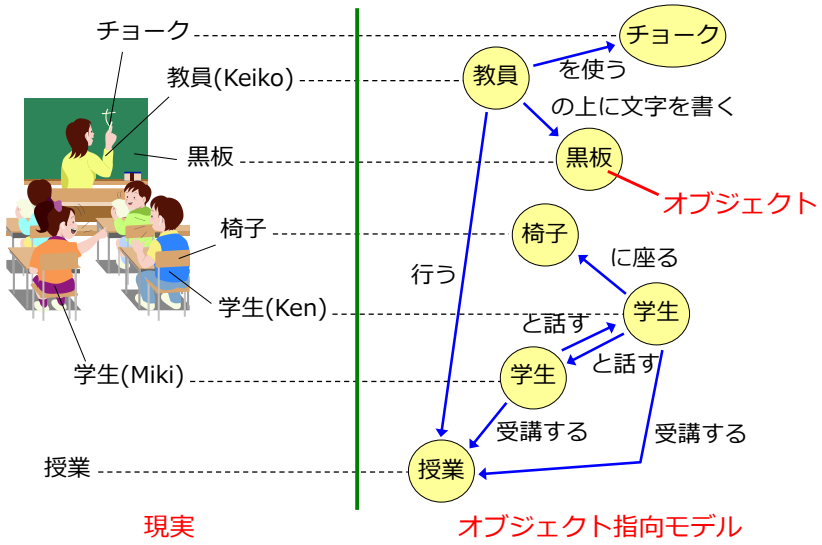
2019年度 秋学期

オブジェクト指向

オブジェクト

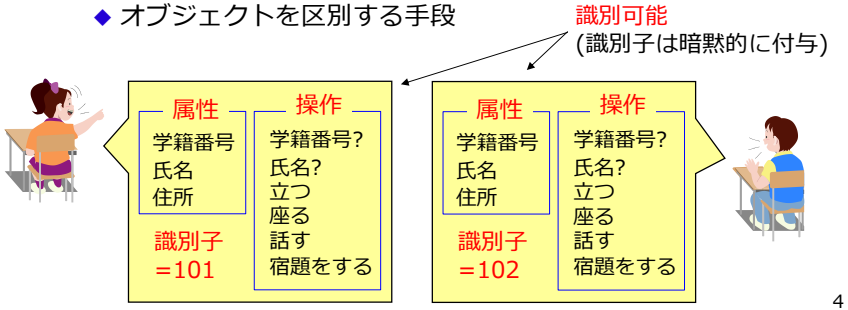
- オブジェクト指向(object orientation)
  - 実世界モデルをソフトウェアで直接的に表現する方法
  - オブジェクト(object)を構成単位としてソフトウェアを構築する枠組み
- オブジェクト(object)
  - 人間が認知できる具体的あるいは抽象的な「もの」
  - 実世界の「もの」や「役割」などの事柄(thing)を抽象化した「もの」
  - 物理的な「もの」, 概念的な「もの」
  - 分析/設計者や開発するシステムに依存

オブジェクト指向によるモデリング



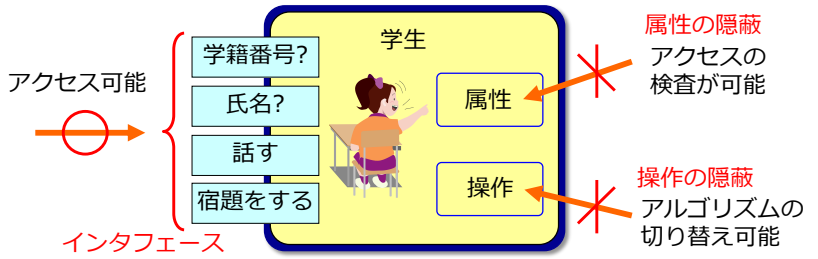
オブジェクトの持つ特性

- 状態(state)
  - オブジェクトの現在の性質
    - 属性(attribute)，プロパティ(property)
- 振る舞い(behavior)
  - オブジェクトが実行できる動作
    - 操作(operation)，メソッド(method)
- 識別性(identity)
  - オブジェクトを区別する手段



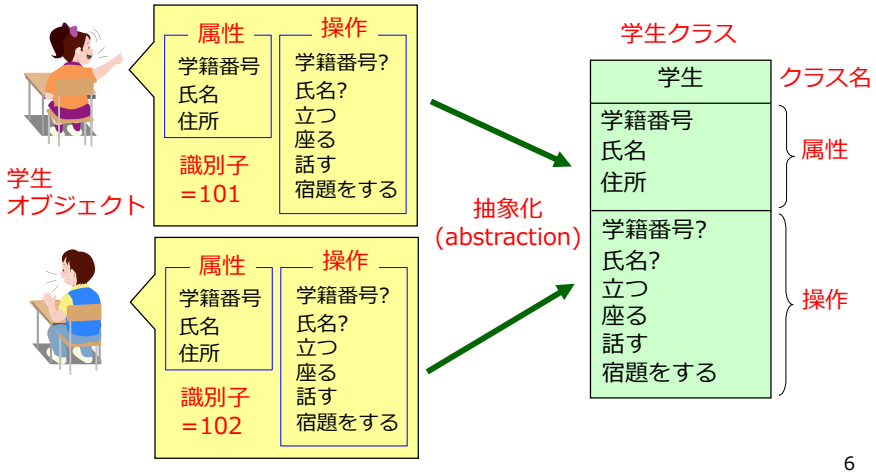
カプセル化

- 属性(データ)とその操作(手続き)をグループ化
  - オブジェクト内部をブラックボックス化
- インタフェースと実装(属性と操作)の分離
  - 抽象データ型(ADT: abstract data type)
    - オブジェクトの中身(実装)を知る必要がない
  - 情報隠蔽(information hiding)
    - オブジェクトの中身(実装)を知ることができない



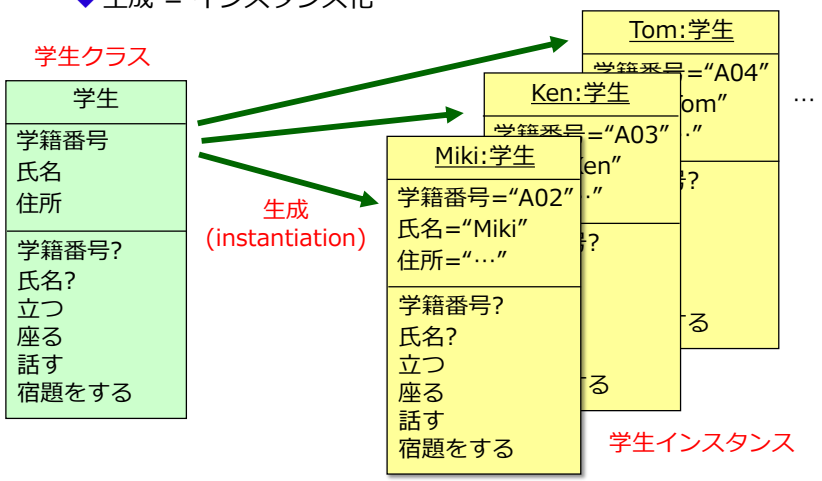
クラス

- 共通の属性と操作を持つオブジェクトを抽象化した雛形
  - オブジェクトの設計図

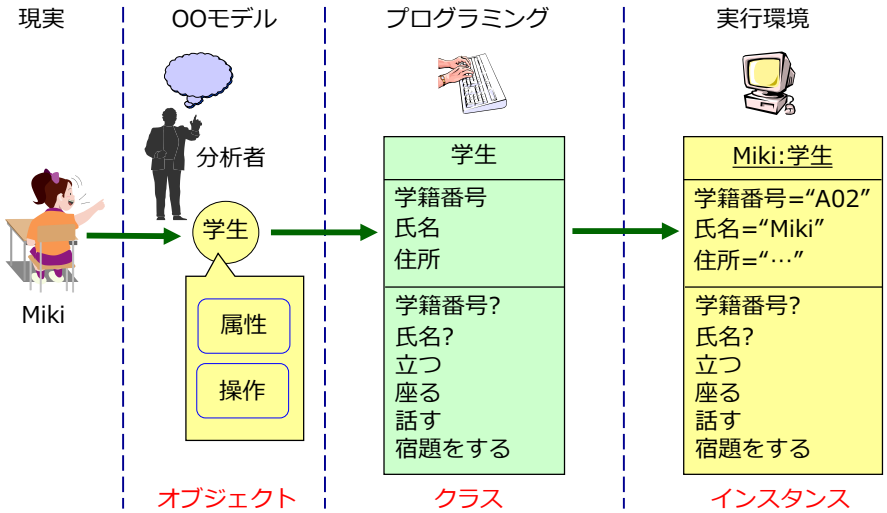


インスタンス

- クラスから生成されたオブジェクト
  - 生成 = インスタンス化



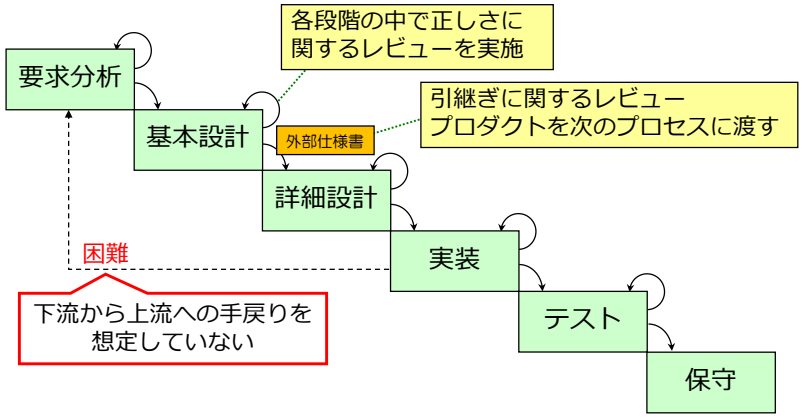
オブジェクト，クラス，インスタンス



オブジェクト指向開発プロセス

ウォーターフォールモデル

- トップダウンな開発プロセス: 滝(waterfall)



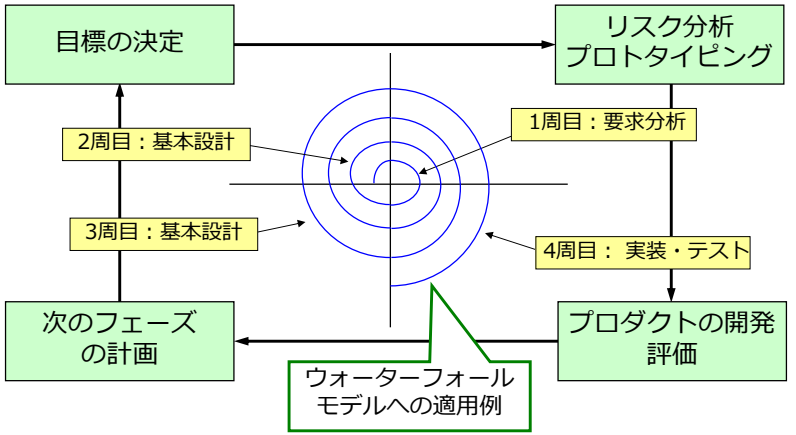
利点: 工数見積もりや進捗管理が容易(開発計画が立てやすい)

欠点: 仕様変更に弱い，誤り発見の遅れ，修正コストの増大

要求仕様が明確な大規模ソフトウェアの開発に向いている

スパイラルモデル

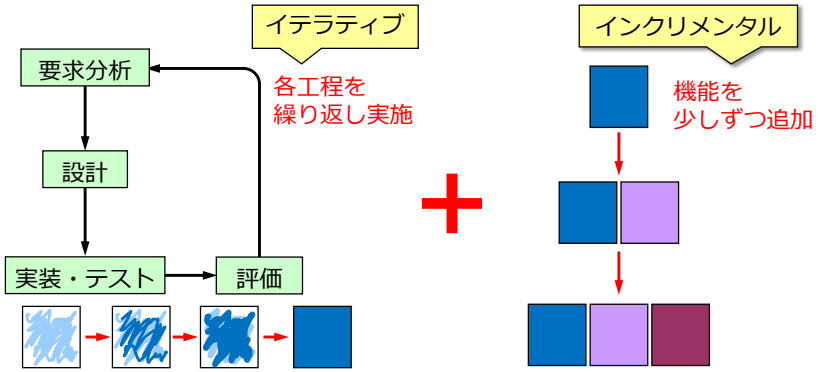
- プロトタイプなどで利用者からのフィードバックに対応しながら，徐々にシステムを作成
  - リスク(開発の失敗)を分析することで，リスクを軽減



進化型プロトタイピング

オブジェクト指向開発で採用

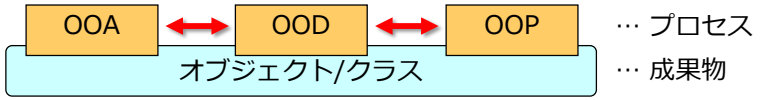
- プロトタイプを徐々に変更し，そのまま完成品として利用
  - 機能が明確かつ不確定要素が少ない部分を優先して開発
  - イテラティブ(iterative)開発
  - インクリメンタル(incremental)開発



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する作業
  - 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作業
  - 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(OOP: OO programming)
  - プログラムの記述とテスト

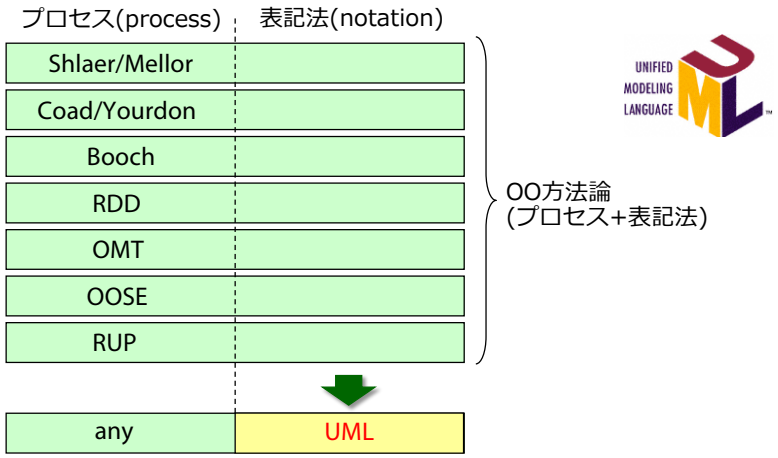
シームレス(seamless)な連携



オブジェクト指向ソフトウェア開発 = クラスを作成すること

Unified Modeling Language (UML)

- グラフィカルな記法とそのメタモデル(言語の概念)を定義
- 方法論とは独立(OMGにより管理)



(Booch, Rumbaugh & Jacobsonが提唱)

UMLダイアグラム(1/2)

構造(structure)に関するダイアグラム(6種類)

- 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ダイアグラム(2/2)

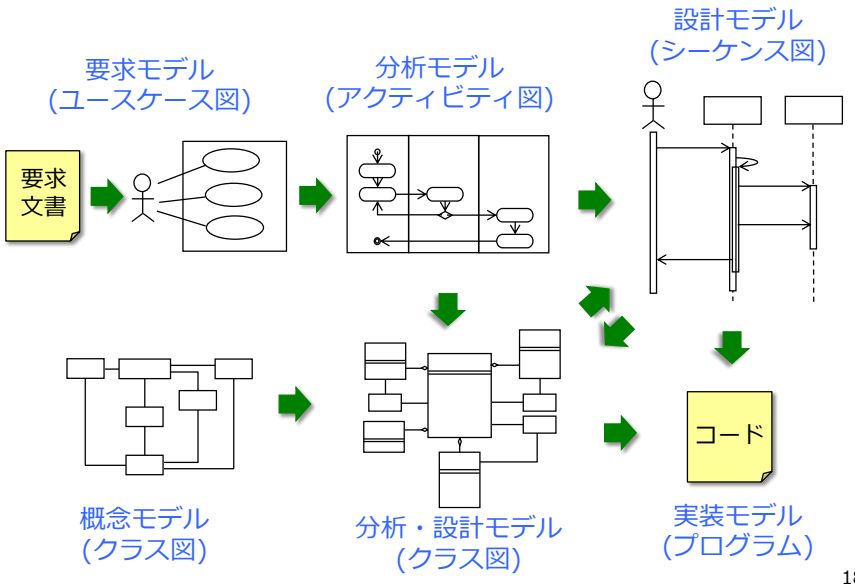
振る舞い(behavior)に関するダイアグラム(7種類)

- 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

オブジェクト指向モデリング

- 概念モデル
  - ドメイン分析(業務分析)により作成
  - 対象業務の世界を構成する概念と概念間の関係を表すモデル
- 要求モデル
  - 顧客がシステムに望む事柄を表すモデル
- 分析モデル
  - 実装方法に関知せずに，対象業務についてシステム化する方法を表すモデル
- 設計モデル
  - システム化する事柄と，その実装方法を詳細に表すモデル
- 実装モデル
  - プログラムソースコード，バイナリプログラム
  - プログラムの計算機上の配置

オブジェクト指向モデリングとモデル



オブジェクト指向モデリングにおける観点

- 機能的観点
  - システムの機能を定義
    - ユースケース図
- 静的観点
  - システムの構造を定義
    - クラス図，オブジェクト図
- 動的観点
  - システムの振る舞いを定義
    - シーケンス図，コミュニケーション図，アクティビティ図，状態機械図
- 物理的観点
  - システムの物理的制約を定義
    - コンポーネント図，配置図

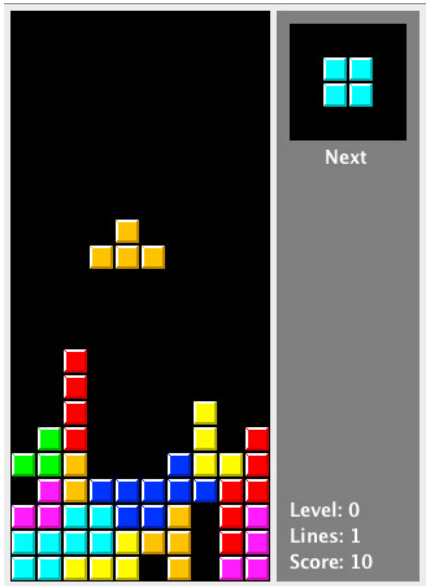
オブジェクト指向分析

- 静的モデリング(static modeling)
  - クラスの構造を定義
    - クラスの抽出
    - クラス間の関係の抽出
    - 継承，集約，多相性の認識
  - クラス図
  - オブジェクト図
- 動的モデリング(dynamic modeling)
  - クラスの振る舞いを定義
    - オブジェクト間の相互作用の認識
    - オブジェクトの状態遷移の定義
  - 相互作用図(シーケンス図，コミュニケーション図)
  - 状態マシン図

オブジェクト指向モデリング

静的モデル（１）

例題: テトリスゲームの開発



テトリス(Tetris)のルール説明

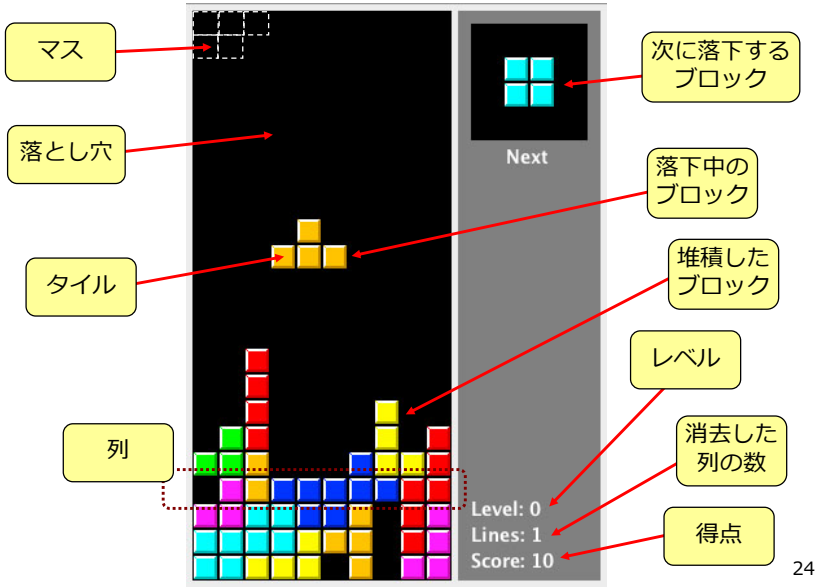
- 画面上部から落ちてくるブロックを左右に動かしたり，回転しながら積んでいく
- ブロックは横一列に揃うと消え，消した列の数により得点が増える
- ブロックが画面上部まで積みあがった時点で，ゲーム終了

テトリスゲームの開発(要求仕様)

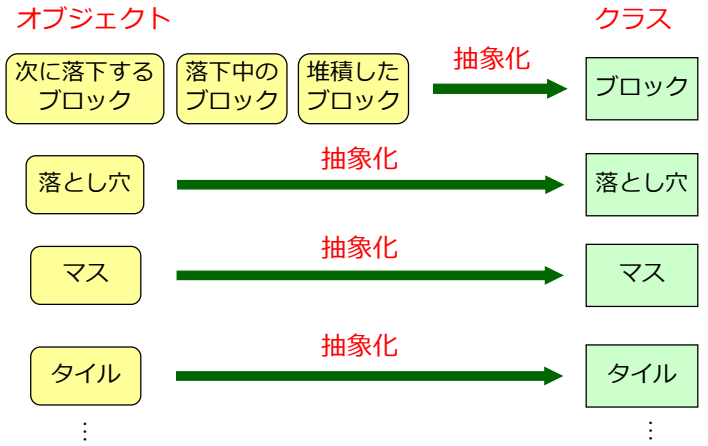
テトリスゲーム

- ブロックが落とし穴の上部から落ちてきて，順番に積み重なる．
- 落とし穴は縦に22個，横に10個の正方形のマスで構成されている．
- ブロックは7種類あり，それぞれ4つの(マスと同じ大きさの)タイルで構成されている．
- プレーヤーは，レバーで落下中のブロックを左右に移動させることができる．また，ボタンを押すことでブロックを回転させることもできる．
- タイルは横一列に揃うと消去され，その列より上部のタイルが消えた列の数だけ下がる．
- ブロックが落とし穴の最上部まで積み重なるとゲームが終了となる．
- タイルを消去すると得点が加算される．複数の列のタイルを同時に消去すると，それだけ高得点となる．
- 消去したタイルの数に応じてゲームのレベルが上がり，ブロックの落下速度が速くなる．
- 次に落ちてくるブロックは落とし穴の横に表示される．
- レベル，得点，消去した列の数が落とし穴の横に表示される．

オブジェクトの抽出



クラスの抽出



名詞抽出法(クラス候補の抽出)

- 名詞と名詞句に着目して，クラス候補を抽出

クラス候補

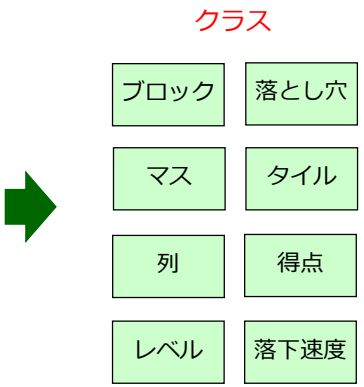
テトリスゲーム
- ブロックが落とし穴の上部から落ちてきて，順番に積み重なる．
- 落とし穴は縦に22個，横に10個の正方形のマスで構成されている．
- ブロックは7種類あり，それぞれ4つの(マスと同じ大きさの)タイルで構成されている．
- プレーヤーは，レバーで落下中のブロックを左右に移動させることができる．また，ボタンを押すことでブロックを回転させることもできる．
- タイルは横一列に揃うと消去され，その列より上部のタイルが消えた列の数だけ下がる．
- ブロックが落とし穴の最上部まで積み重なるとゲームが終了となる．
- タイルを消去すると得点が加算される．複数の列のタイルを同時に消去すると，それだけ高得点となる．
- 消去したタイルの数に応じてゲームのレベルが上がり，ブロックの落下速度が速くなる．
- 次に落ちてくるブロックは落とし穴の横に表示される．
- レベル，得点，消去した列の数が落とし穴の横に表示される．

名詞抽出法(クラスの選別)

- 不適当な候補を除去

ブロック(Block)
落とし穴(Pit)
× 正方形 … マスの形を表現
マス(Box)
タイル(Tile)
× プレーヤー … 開発対象外
× レバー … 開発対象外
× ボタン … 開発対象外
列(Line)
× 数 … 程度を表す
× ゲーム … 開発対象全体
得点(Score)
× 高得点 … 高い得点のこと
レベル(Level)
落下速度(Speed)

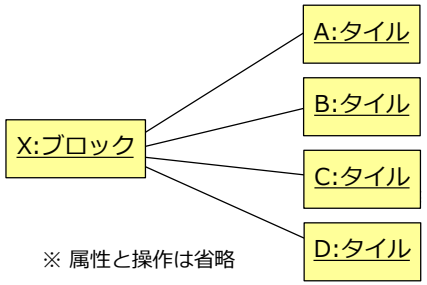
×: クラスでない(と思われる)もの



オブジェクト図(object diagram)

- オブジェクトの属性と操作，オブジェクト間の関係を表現

「ブロックは7種類あり，それぞれ4つのタイルで構成されている」



※ 属性と操作は省略

O:C のOはオブジェクト名
Cはクラス名

オブジェクトには下線を付与

「O:」，「:C」の場合もあり

クラス図(class diagram)

- クラスの内部構造(属性と操作)とクラス間の静的な関係を表現
- オブジェクト指向分析・設計・実装をつなぐ中心的な図
- 3つの観点を持つ
  - 概念の観点，仕様の観点，実装の観点
- クラス図に関する分析順序(必ずしもこの順番でなくとも良い)

(1) 関連(association)
- インスタンス間に成立する一時的な関係を抽象化したもの

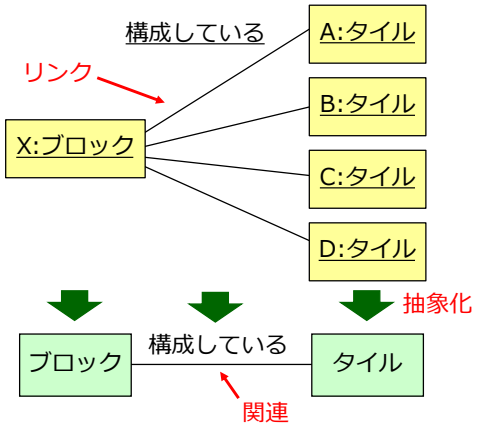
(2) 操作(operation)
- インスタンスが受け付けるメッセージ

(3) 属性(attribute)
- インスタンスが持つデータ

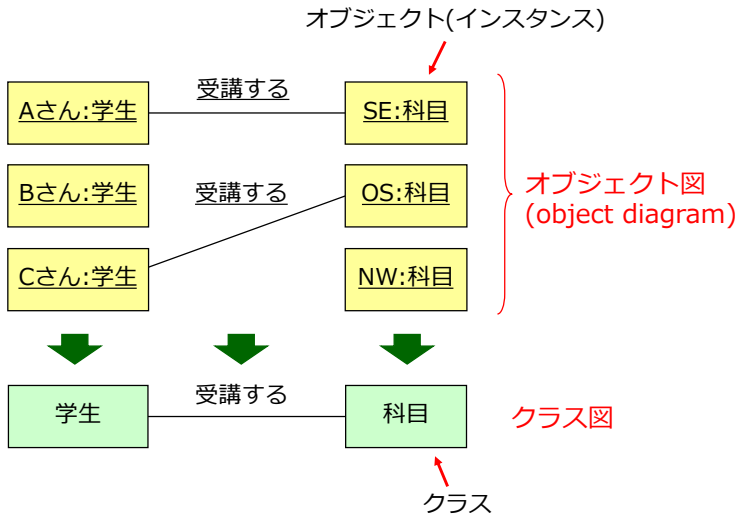
インスタンス： クラスから生成されたオブジェクト

関連とリンク

- リンク(link):
  - インスタンスどうしの一時的な協調関係
- 関連(association)
  - インスタンス間に成立する一時的な関係を抽象化したもの



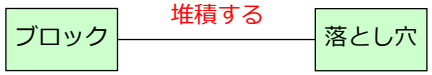
関連とリンク(補足)



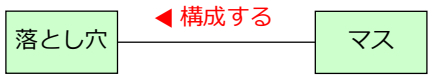
関連の抽出

- 要求仕様中の動詞に対応することが多い

「ブロックが落とし穴の上部から落ちてきて，順番に積み重なる」

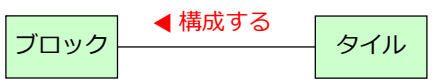


「落とし穴は縦に22個，横に10個の正方形のマスで構成されている」



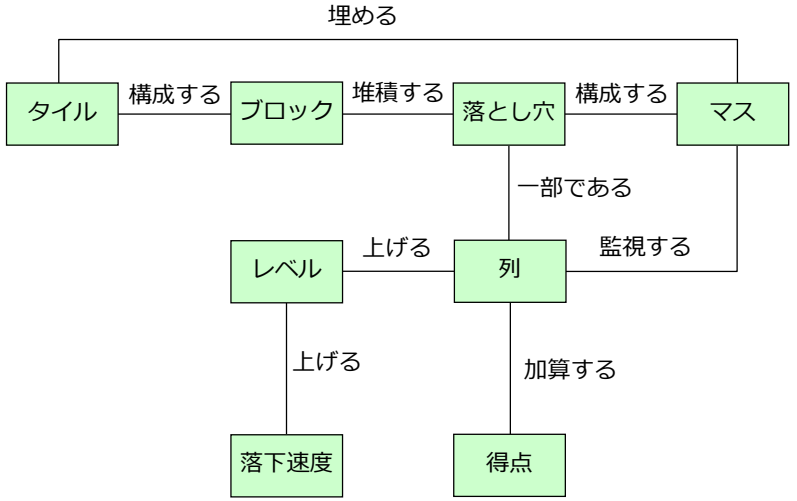
(関連の方向を明記)

「ブロックは7種類あり，それぞれ4つのタイルで構成されている」



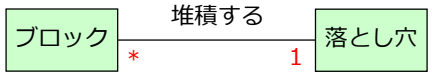
(関連の方向を明記)

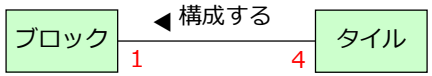
クラス図(1) [関連の追加]



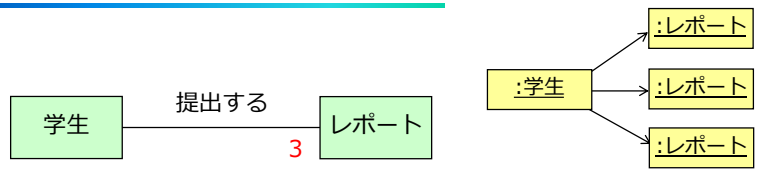
多重度(multiplicity)

- 関連に関与するインスタンスの数を表現
  - 特定の値: その数値を記述
    - 1: 1個
  - 値の範囲: 下限値と上限値を“..”でつないで記述
    - 1..5: 1個から5個の間
    - 0..1: 0個か1個
  - 任意の値: “*”を記述
    - 0..*, *: 0個以上
    - 1..*: 1個以上

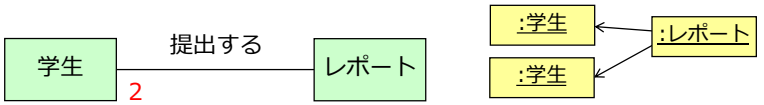




多重度(補足)



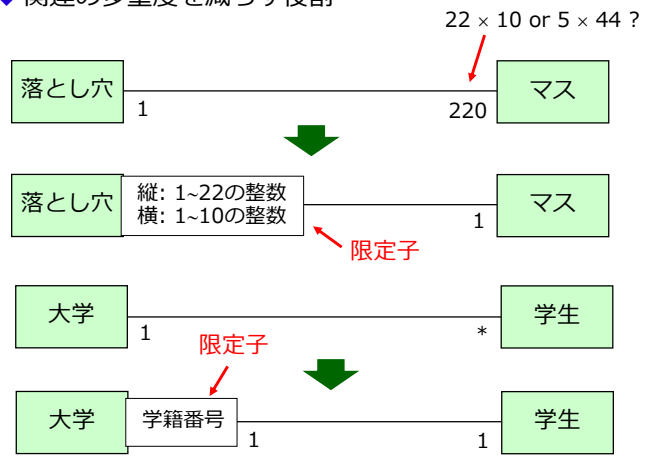
1つの「学生」インスタンスが3つの「レポート」インスタンスと関連を持つ
→ 1人の学生は2つのレポートを提出する



1つの「レポート」インスタンスが2つの「学生」インスタンスと関連を持つ
→ 1つのレポートは2人の学生によって提出される

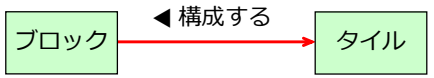
限定子つき関連

- 限定子(qualifier)
  - 関連の相手を特定する属性
  - 関連の多重度を減らす役割



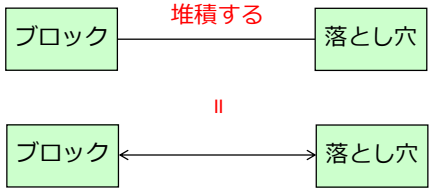
誘導可能性(navigability)

- 責任(参照)の方向を限定する



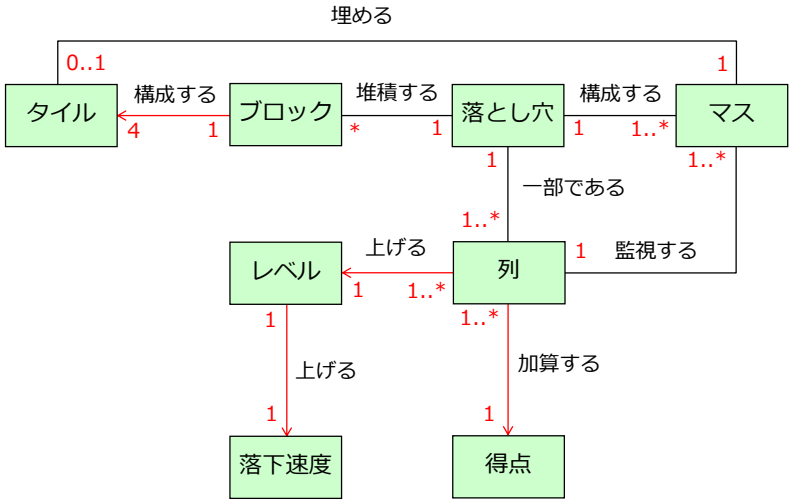
単方向関連
(関連の方向と無関係)

「ブロック」から「タイル」への参照あり(知っている)
「タイル」から「ブロック」への参照なし(知らない)



双方向関連

クラス図(2) [多重度と誘導可能性の追加]



オブジェクト指向モデリング

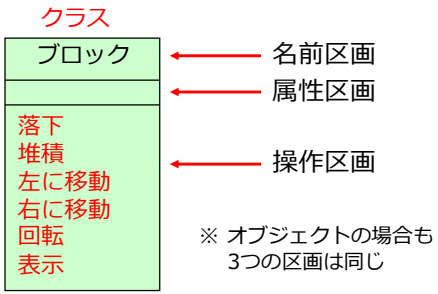
静的モデル（２）

操作の抽出

- 要求仕様中の動詞に対応することが多い

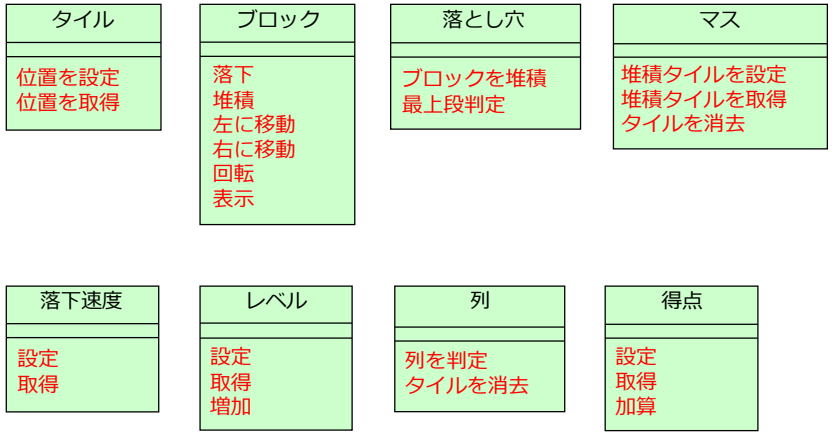
テトリスゲーム(「ブロック」に関する部分の抜粋)
- ブロックが落とし穴の上部から落ちてきて，順番に積み重なる．
- ブロックは7種類あり，それぞれ4つの(マスと同じ大きさの)タイルで構成されている．
- プレーヤーは，レバーで落下中のブロックを左右に移動させることができる．また，ボタンを押すことでブロックを回転させることもできる．
- ブロックが落とし穴の最上部まで積み重なるとゲームが終了となる．
- 次に落ちてくるブロックは落とし穴の横に表示される．

操作の候補
- 落ちてくる
- 積み重なる
- 左右に移動する
- 回転する
- 表示する

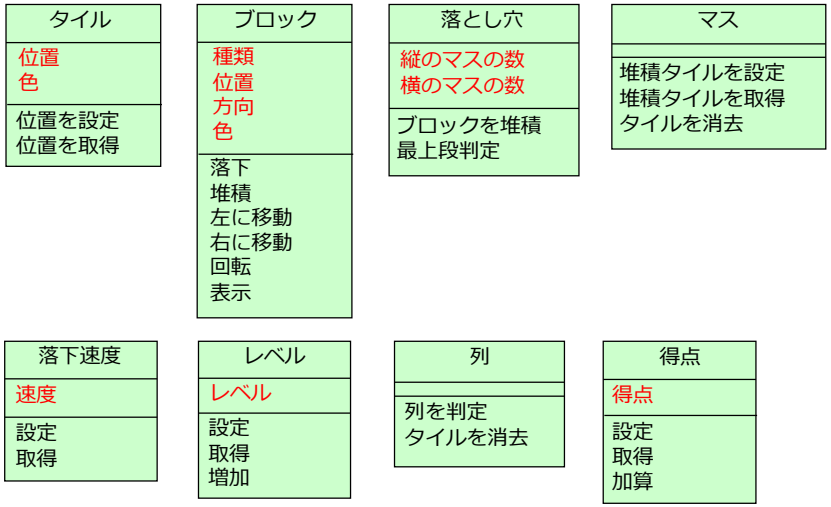


※ オブジェクトの場合も3つの区画は同じ

クラス(操作の追加)



クラス(属性の追加)



操作と属性の構文

- 属性の構文(シグニチャ)

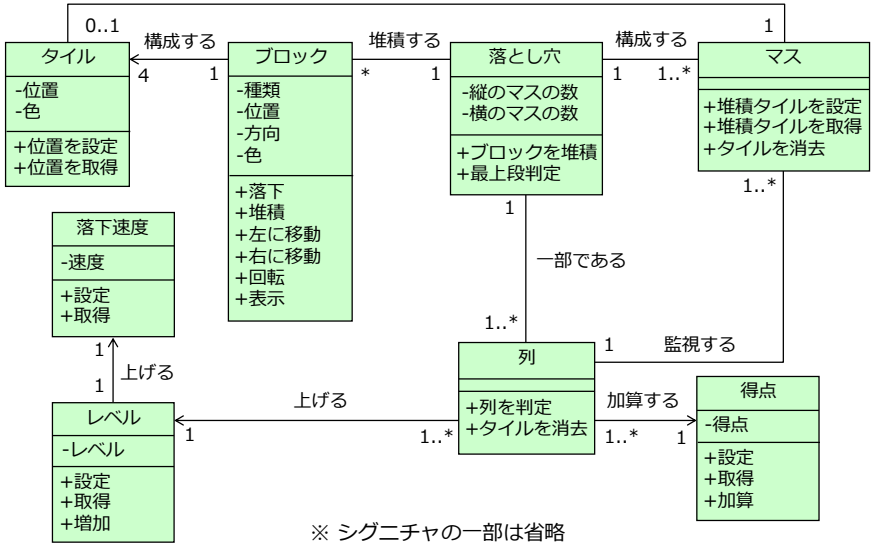
  可視性 名前:型=デフォルト値

  - デフォルト値は省略可
    -得点:Integer=0
- 操作の構文(シグニチャ)

  可視性 名前(引数のリスト):戻り値の型

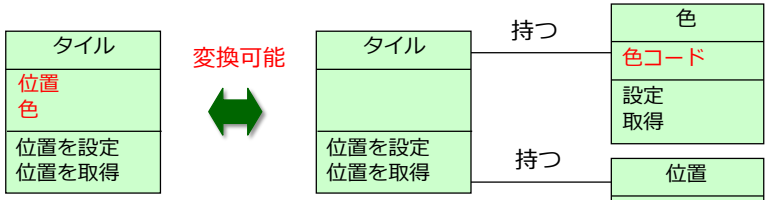
  - [可視性] +(public), #(protected), ~(package), -(private)
  - [引数のリスト] 引数をコンマで区切って並べたリスト
  - [引数] 方向 名前:型=デフォルト値 (デフォルト値は省略可)
  - [方向] in, out, inout (省略時はin)
  - [型] 型名[*] ([*]は多重度を表す．多重度1のとき省略可)
  - 戻り値がない場合は，戻り値の型は省略可
    ~得点の設定(s:Integer)
    +得点の取得():Integer

クラス図(3) [操作と属性の追加]



※ シグニチャの一部は省略

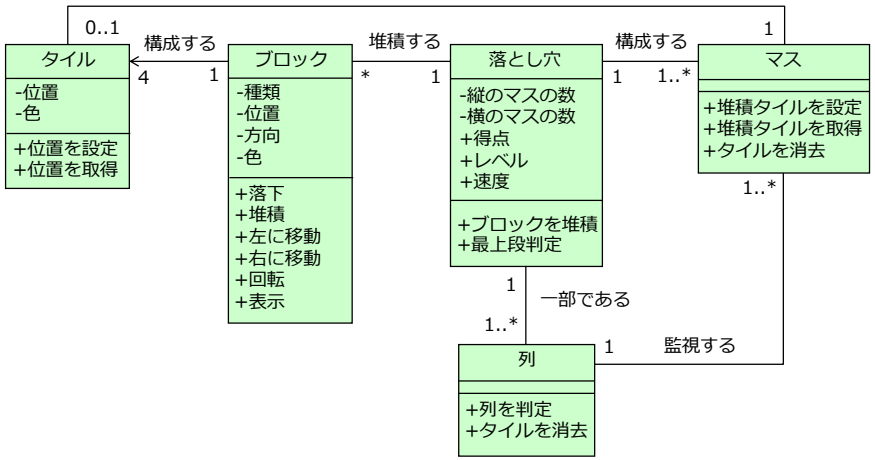
関連と属性の変換



属性
- クラスの本質的な特徴
- クラスとして扱うほど重要でない

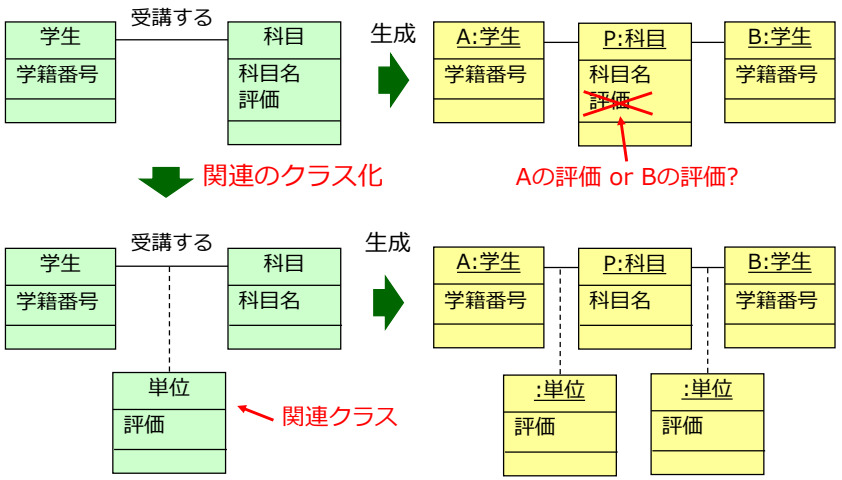
関連
- クラスの本質的な特徴でない

クラス図(4) [操作と属性の追加]



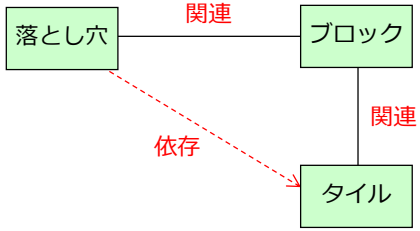
関連クラス

- 属性や操作を持つ関連



依存(dependency)

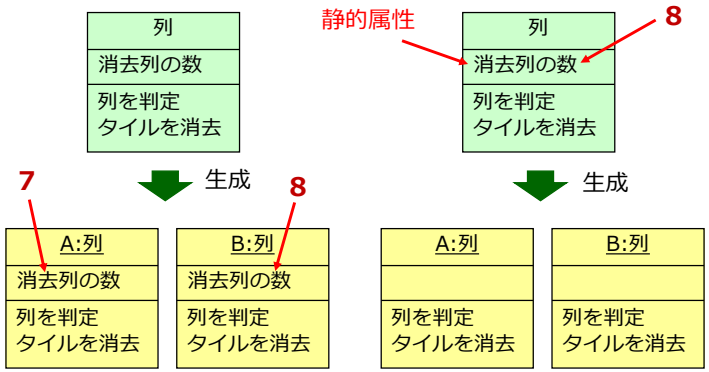
- クラスのインスタンス同士が一時的に協調する可能性を表現
  (関連より弱い)



落とし穴オブジェクトがある処理を実行中のときだけ，
タイルオブジェクトを参照

静的属性と静的操作

- インスタンスではなく，クラスに属する属性や操作
  - 同一のクラスから生成したインスタンス群で値を共有
  - 状態を持たないユーティリティ関数の実現
- インスタンスを生成せずに利用可能

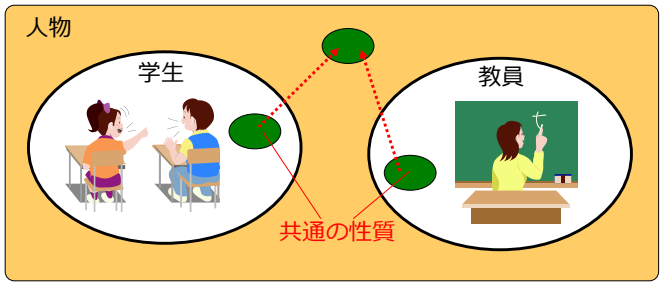


オブジェクト指向モデリング

静的モデル（３）

継承(inheritance)

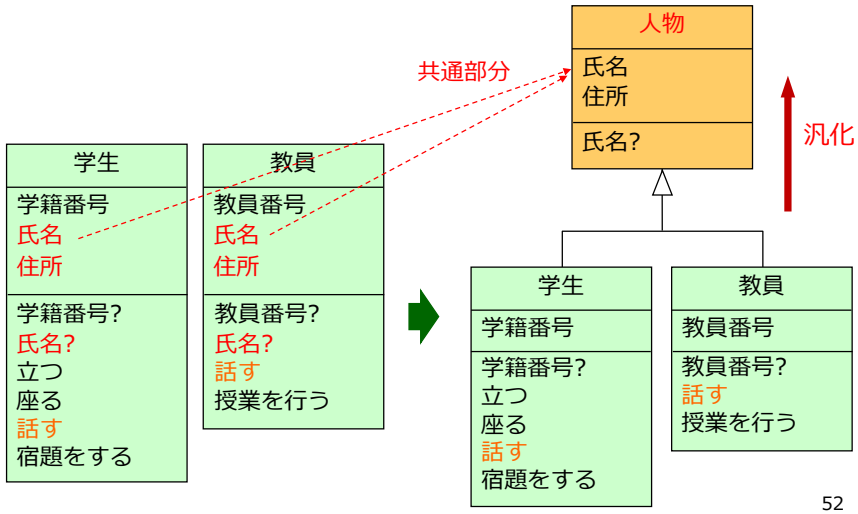
- 複数のクラスにおける共通の性質を共有する仕組み
- 他のクラスの性質を引き継ぐ仕組み
- is-a関係
  - 親クラスと子クラス
  - スーパクラスとサブクラス



学生 is a 人物 (学生は人物である)
教員 is a 人物 (教員は人物である)

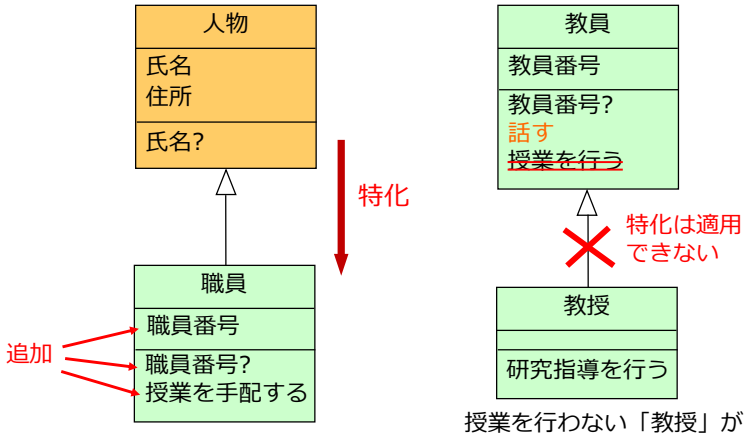
汎化(generalization)

- 複数クラスの共通部分を抽出して親クラスとして定義



特化(specialization)

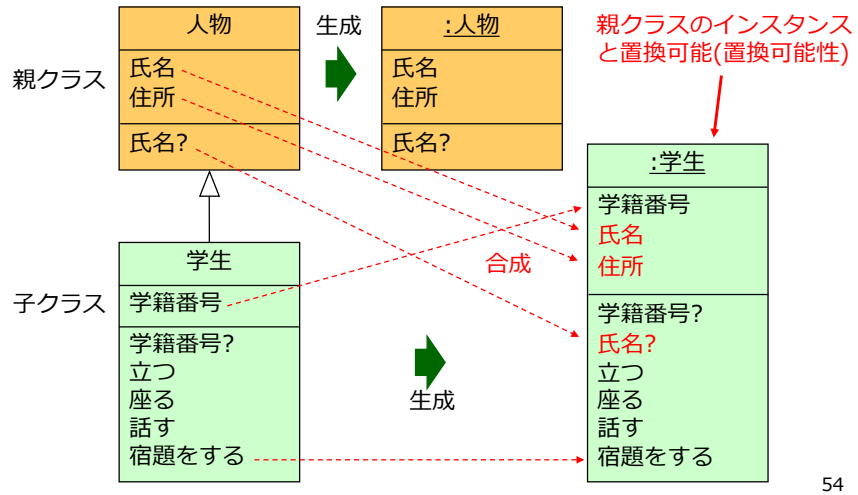
- あるクラスを基にして特殊化したクラスを定義



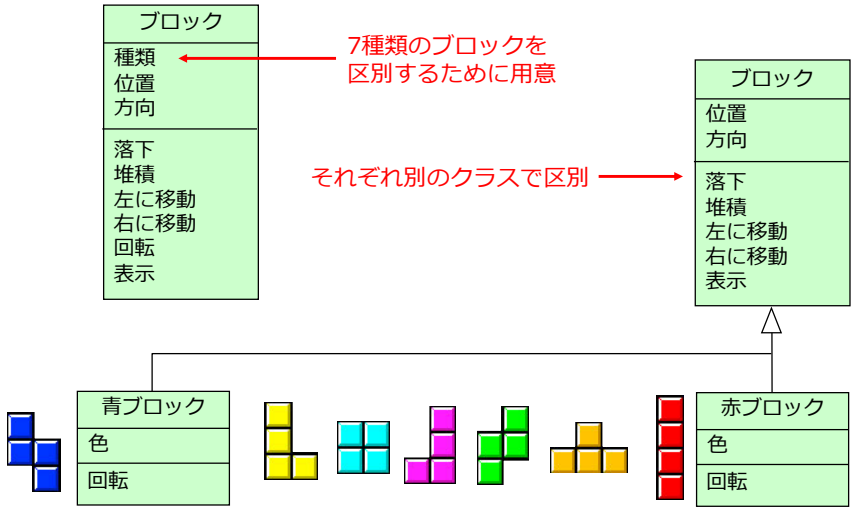
授業を行わない「教授」が
存在する場合

継承におけるインスタンスの生成

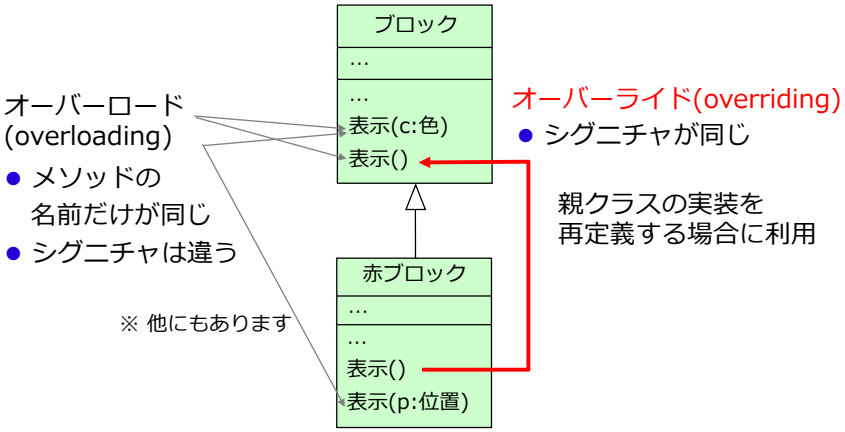
- 親クラスから属性や操作を引き継いで，自分と合成してインスタンスを生成



継承の例



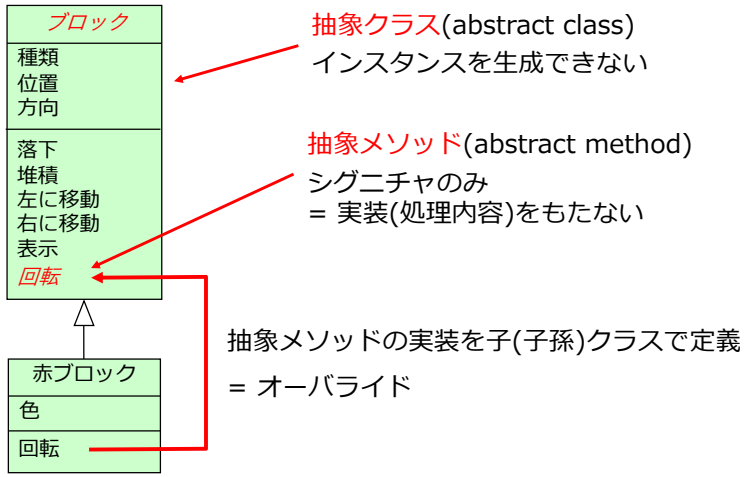
継承における操作のオーバライド



「赤ブロック」インスタンスに対して，「+表示()」を呼び出すと，
「赤ブロック」インスタンスの「+表示()」が呼び出される

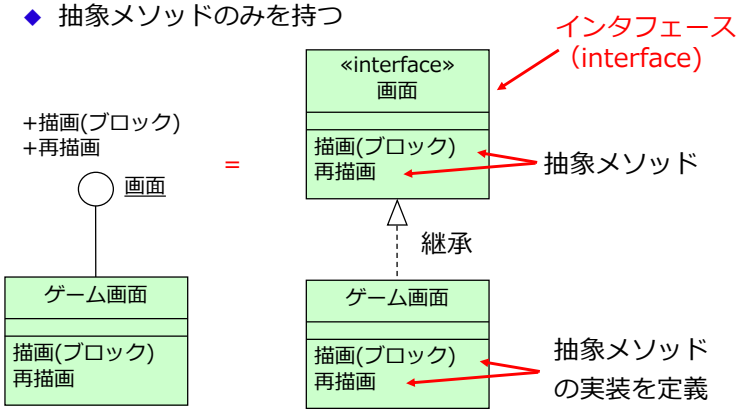
抽象クラス

- 抽象メソッドを持つクラス



インタフェース

- メソッドのシグニチャのみを定義
  - 抽象メソッドのみを持つ

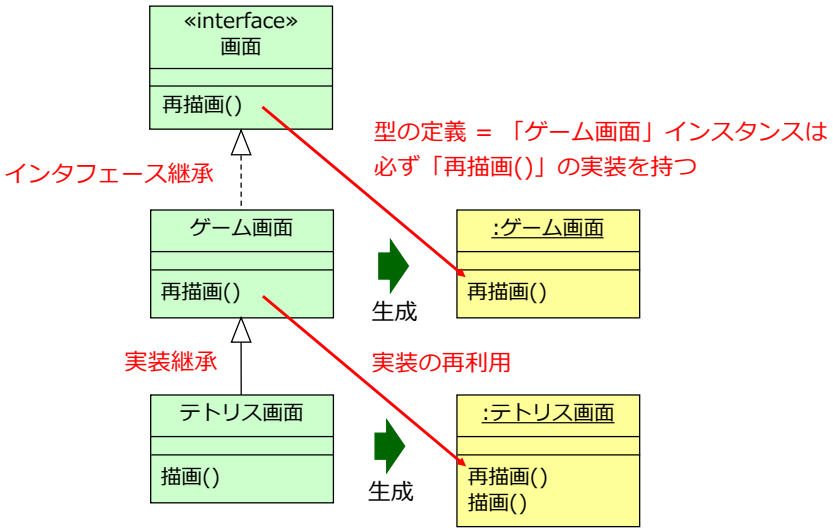


○ インタフェースがインタフェースを継承
○ クラスがインタフェースを継承
× インタフェースがクラスを継承

実装継承とインタフェース継承

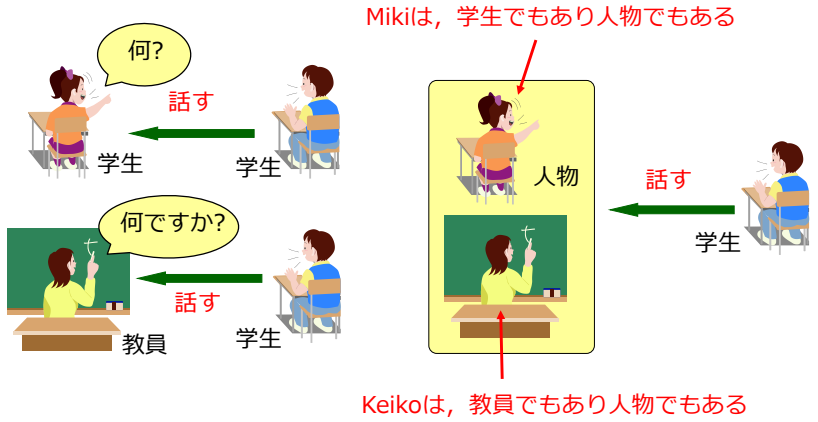
- クラスの役割
  - インスタンスのひな形という側面
  - 実装の再利用(差分プログラミング)の側面
  - 型定義の側面
- 実装継承(subclassing)
  - 実装の再利用の観点による継承
  - 親クラスの実装(属性と操作)を引き継ぐ継承
  - インスタンスを生成する際，実装の責任なし
    - 子クラスで実装しなくともよい
- インタフェース継承(subtyping)
  - 型定義の観点による継承
  - 親クラスの型(属性と操作のシグニチャの集合)を引き継ぐ継承
  - インスタンスを生成する際，実装の責任あり
    - 子クラスに実装を強要

実装継承とインタフェース継承の例



多態性(polymorphism)

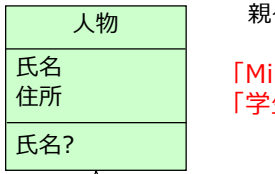
- 1つのインスタンスが複数のクラス(型)に属することを可能とする仕組み
- 多相性ともいう

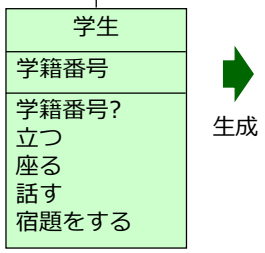


多態性の例(1)

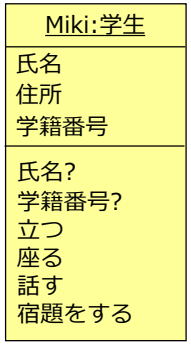
置換可能性より，子クラスのインスタンスは
親クラスのインスタンスとみなせる

「Miki:学生」は
「学生」クラスと「人物」クラスに属する



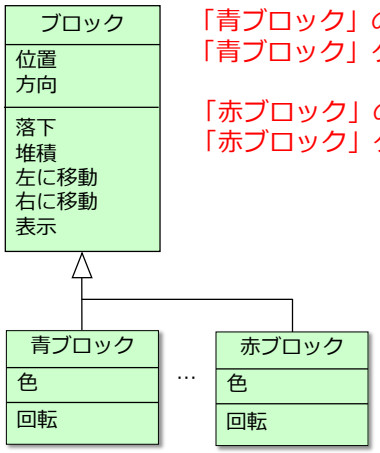


生成



人物型の変数personには
「人物」クラス
「学生」クラス
のインスタンスが代入可能

多態性の例(2)

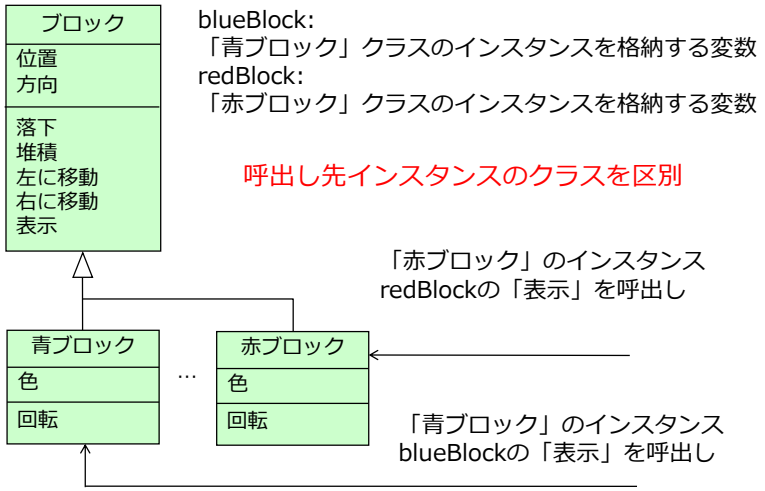


「青ブロック」のインスタンス
「青ブロック」クラスと「ブロック」クラスに属する

「赤ブロック」のインスタンス
「赤ブロック」クラスと「ブロック」クラスに属する

ブロック型の変数blockには
「ブロック」クラス
「青ブロック」クラス
「赤ブロック」クラス
のインスタンスが代入可能

多態性を活用しない場合



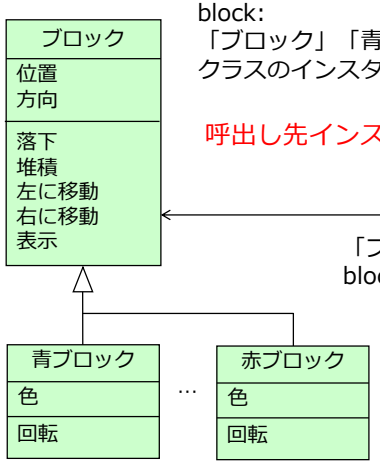
blueBlock:
「青ブロック」クラスのインスタンスを格納する変数
redBlock:
「赤ブロック」クラスのインスタンスを格納する変数

呼出し先インスタンスのクラスを区別

「赤ブロック」のインスタンス
redBlockの「表示」を呼出し

「青ブロック」のインスタンス
blueBlockの「表示」を呼出し

多態性を活用する場合



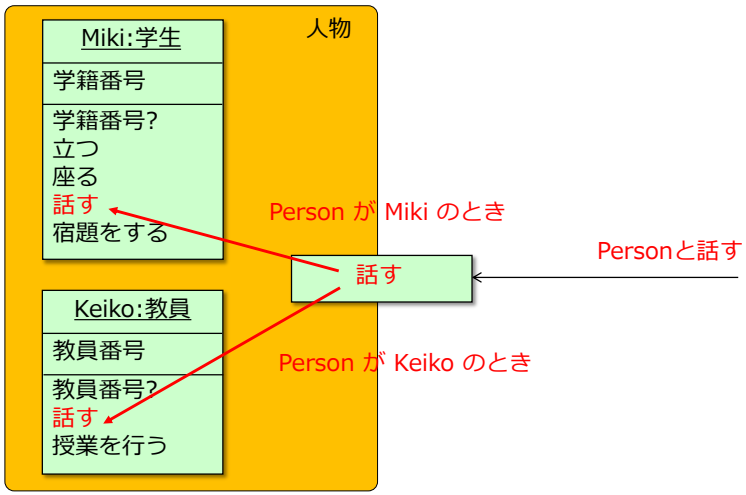
block:
「ブロック」「青ブロック」「赤ブロック」
クラスのインスタンスを格納する変数

呼出し先インスタンスのクラスを区別しなくてよい

「ブロック」のインスタンス
blockの「表示」を呼出し

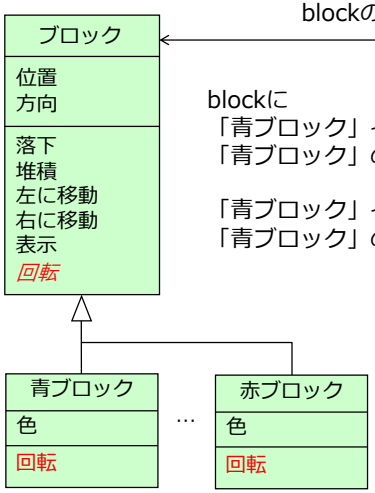
動的束縛(dynamic binding)

- 操作を実行する(応答する)インスタンスを実行時に受信側で決定
  - 呼び出され側のインスタンスを明示的に指定しなくてよい



動的束縛の例

「ブロック」のインスタンス
blockの「回転」を呼出し



blockに
「青ブロック」インスタンスが格納されているとき,
「青ブロック」の「回転」が呼び出される

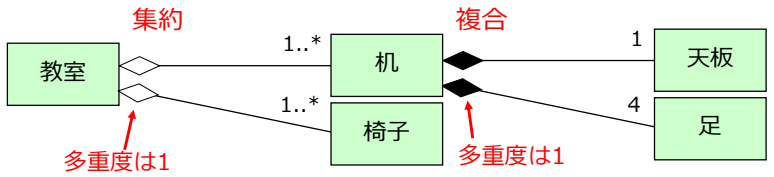
「青ブロック」インスタンスが格納されているとき,
「青ブロック」の「回転」が呼び出される

オブジェクト指向モデリング

静的モデル（４）

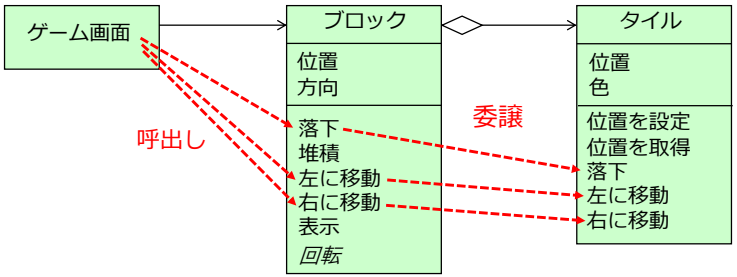
集約と複合

- 集約(aggregation)
  - 全体-部分(part-of, has-a)関係を表現する関連
  - インスタンスへの参照(ポインタ)を保持
    - 参照オブジェクトで実現
- 複合(composition)
  - 強い所有関係と同一の生存期間を持つ集約
  - インスタンスの値(実体)を保持
    - 値オブジェクトで実現



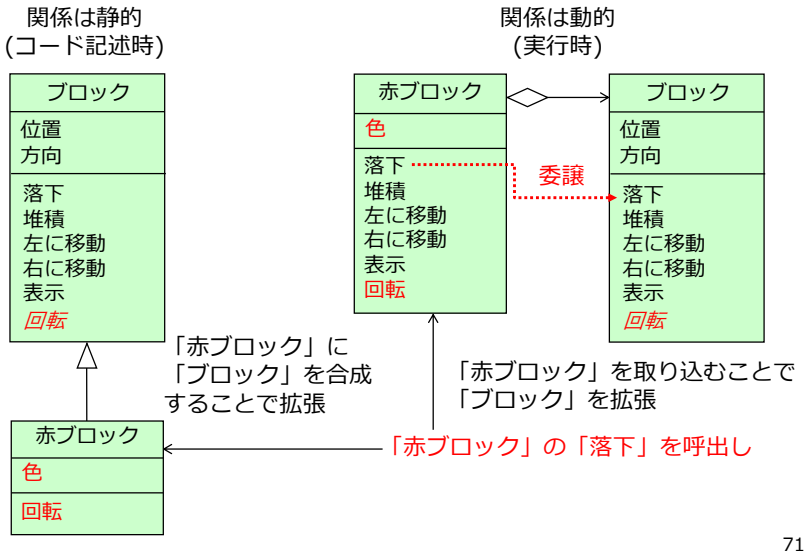
委譲(delegation)

- 別のインスタンスに処理の一部を代替わりさせること
  - 集約の場合，全体→部分の委譲が頻繁に発生

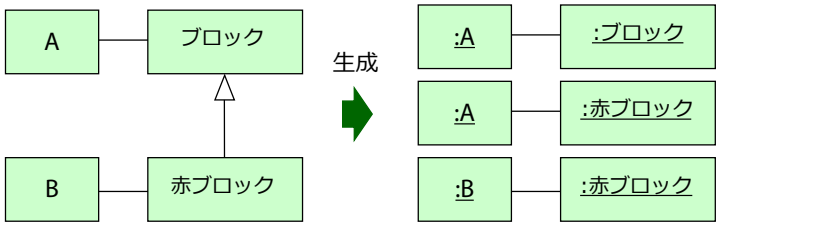


「ブロック」を「左に移動する」ための処理は，
4つのタイルを「左に移動する」ことで実現

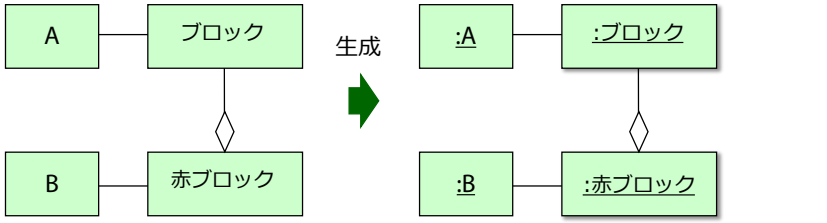
継承と集約による機能拡張



継承と集約の違い

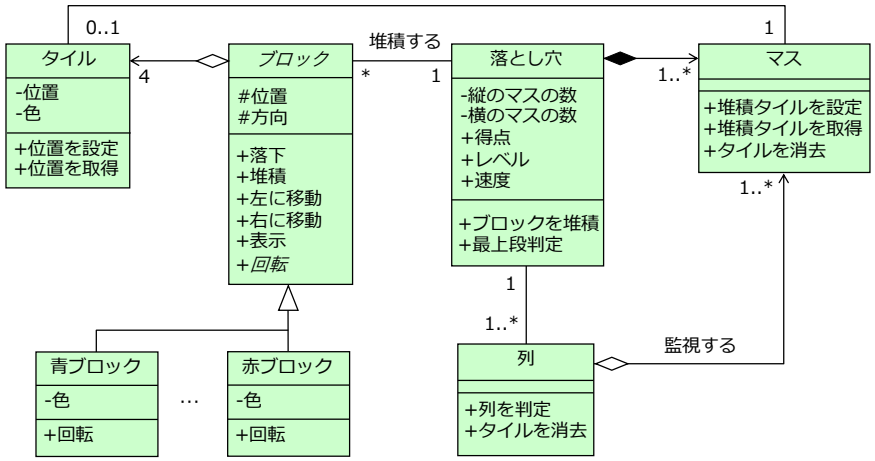


継承はインスタンス間に現れない
リンクが子クラスのインスタンスにも現れる



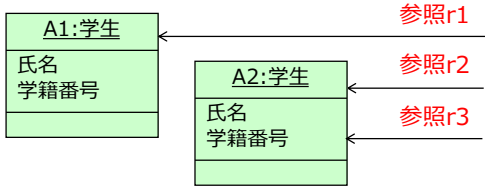
合成はインスタンス間に残る

クラス図(5) [継承と集約の追加]



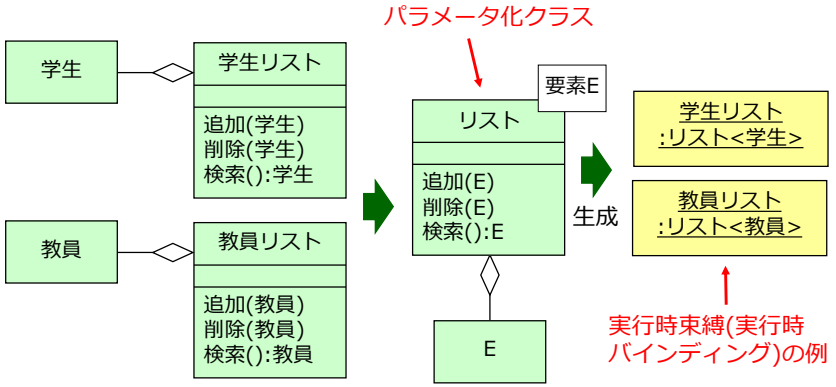
参照オブジェクトと値オブジェクト

- 参照オブジェクト(reference object)
  - 同じのものを1つのインスタンスで表現
    - 複製(コピー)は禁止
    - 識別性により同一性を判定 ---- r2 = r3
  - 関連で保持
- 値オブジェクト(value object)
  - 同じのものを指す複数のインスタンスが存在
    - 複製を許可
    - 値により等価性を判定 ---- r1.氏名 = r2.氏名 & r1.学籍番号 = r2.学籍番号
  - 属性で保持



総称(generics)

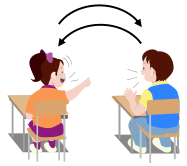
- 異なる型で同じ操作(アルゴリズムが同じ)を実現
  - パラメータ化クラス(parameterized class)
    - テンプレートクラス(template class)



オブジェクト指向モデリング

動的モデル（１）

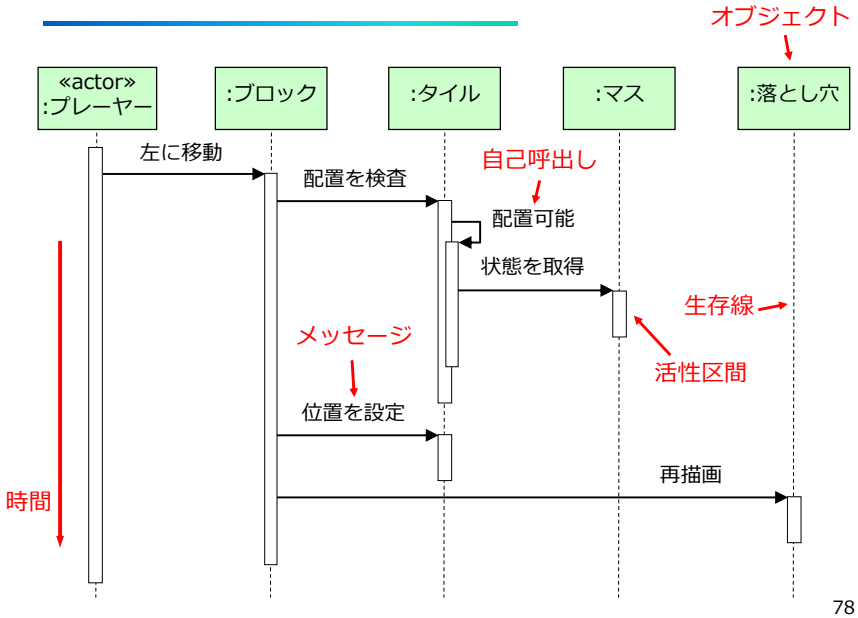
相互作用図(interaction diagram)



- 関連するオブジェクト群の振る舞いを表現
- どのオブジェクトが誰にメッセージを
  どのタイミングで送るのかを表現
- システムがどのようにユースケース(シナリオ)を
  実現するのかを記述

- シーケンス図(sequence diagram)
  - オブジェクト間のメッセージ送受信を時系列で表現したもの
  - メッセージの流れが直感的
- コミュニケーション図(communication diagram)
  - コラボレーション図(collaboration diagram) (UML.1.x)
  - 相互作用するオブジェクト間のリンクを表現したもの
  - レイアウト的に有利
- シーケンス図とコミュニケーション図はほぼ等価な情報を表現

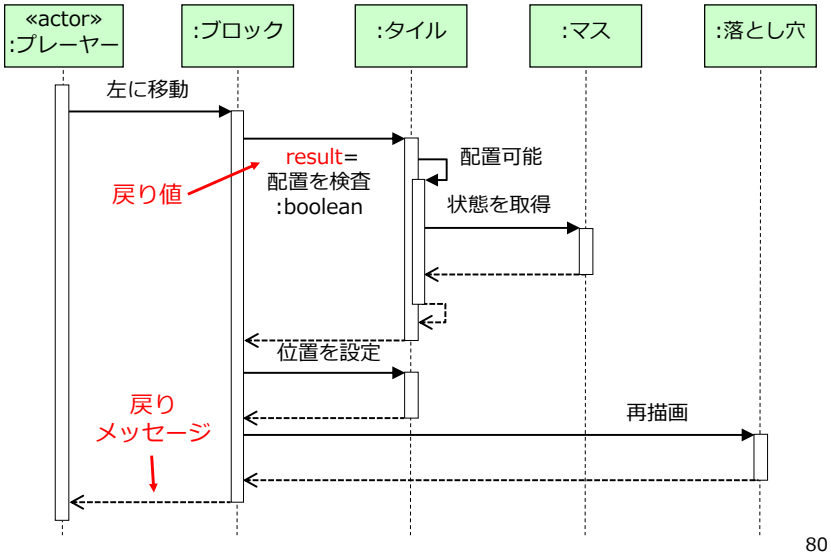
シーケンス図



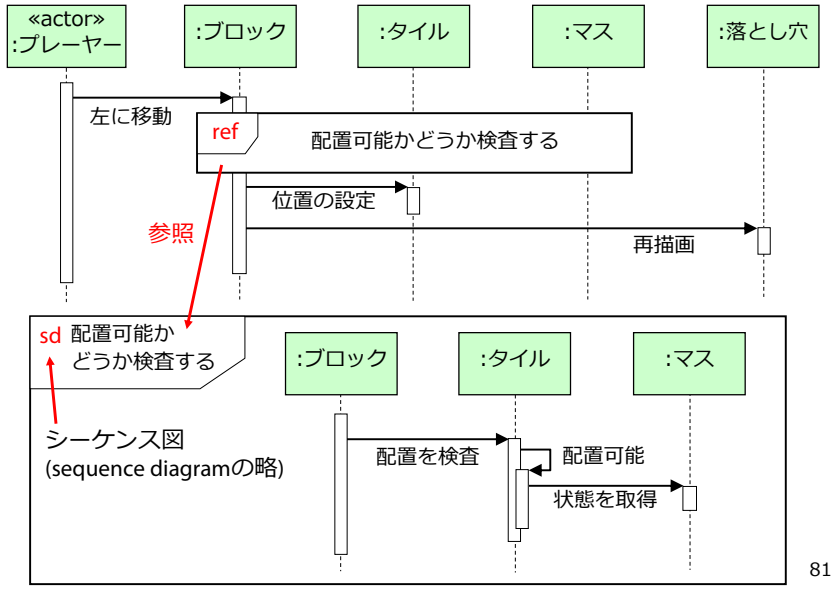
シーケンス図の構成要素

- オブジェクト
  - シーケンス図に参加する要素(インスタンスでなくともよい)
- メッセージ
  - 送信オブジェクトから受信オブジェクトへの矢印で表現
  - 同期メッセージ(⟶): 戻りを待つ
  - 非同期メッセージ(⟶): 戻りを待たない
  - 戻りメッセージ(⇢)
- 生存線(lifeline)
  - オブジェクトの存在を表現
- 活性区間(activation)
  - オブジェクトが活動中(実行中ではない)の期間を表現

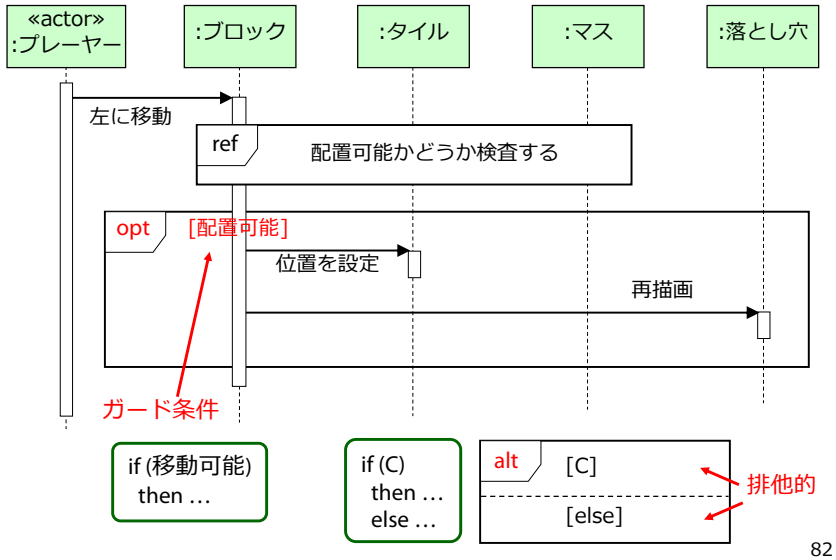
シーケンス図(戻りメッセージ)



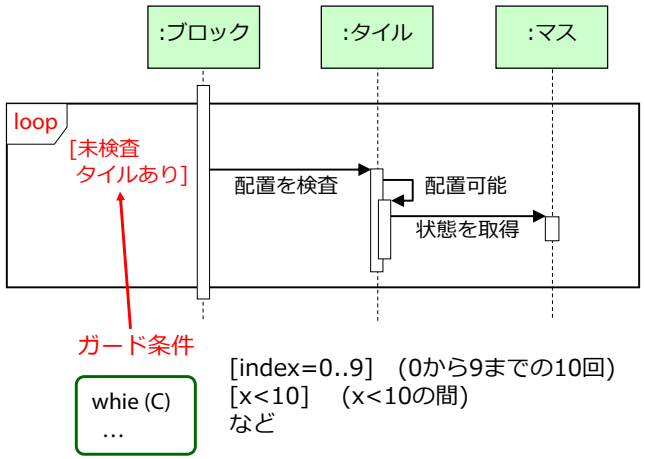
シーケンス図(入れ子の参照)



シーケンス図(選択)

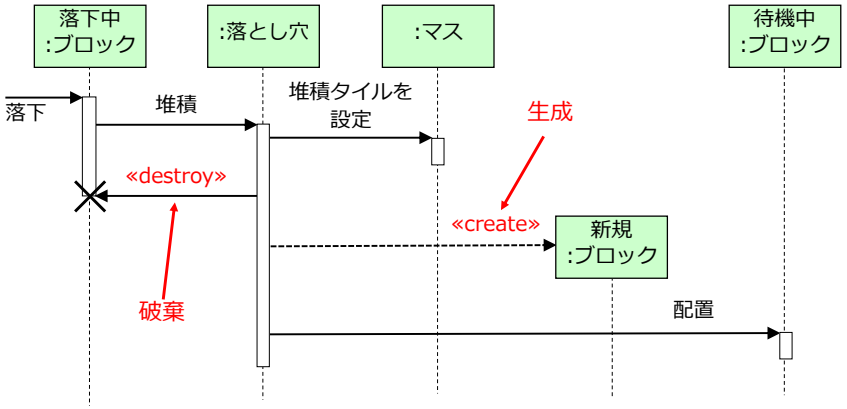


シーケンス図(繰り返し)

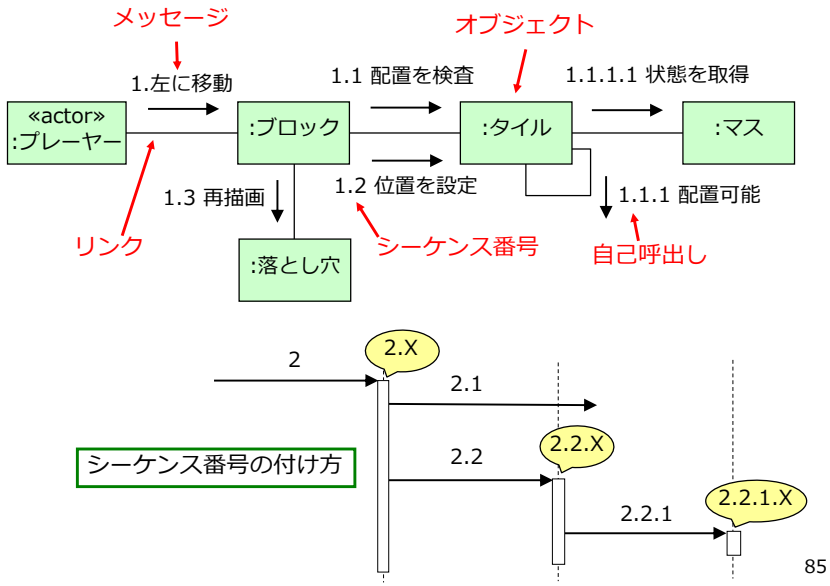


[index=0..9]　(0から9までの10回)
[x<10]　(x<10の間)
など

オブジェクトの生成と破棄



コミュニケーション図

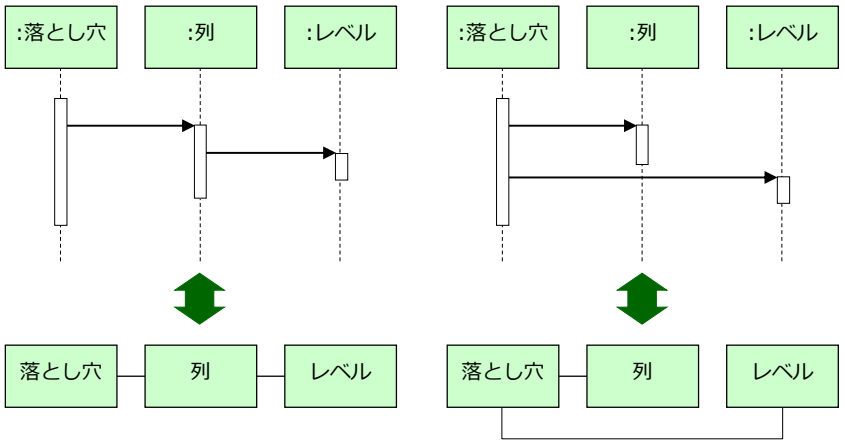


コミュニケーション図の構成要素

- オブジェクト
  - シーケンス図と同じ
- リンク
  - オブジェクト間の相互作用の存在を表現
- メッセージ
  - 送信オブジェクトから受信オブジェクトへの矢印で表現
  - シーケンス番号により，メッセージの順序を表現
    - 受信オブジェクトを活性化したメッセージの番号を，そのオブジェクトが送信するすべてのメッセージの前に付与
  - 同時性は「シーケンス番号+記号」で表現
    - 2aと2bなど
  - 選択
    - ガード条件で表現([移動可能]など)
  - 繰り返し
    - ループ制約で表現(*[index=0..9]など)

シーケンス図とクラス図の関係

- メッセージ送信には関連が必須
  - 矛盾がある場合は，図を修正して解消(一貫性の維持)



オブジェクト指向モデリング

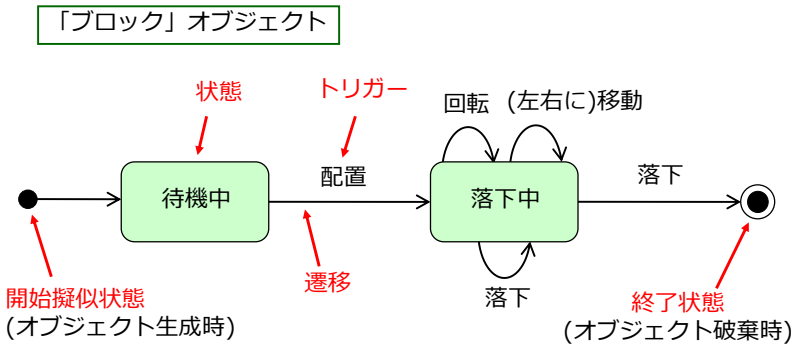
動的モデル（２）

状態機械図(state machine diagram)

- 特定のオブジェクトが取りうるすべての状態と，そのオブジェクトに到着したイベントによる状態の遷移を表現
  - 状態(state)
    - ある時点におけるオブジェクトの状況(受動的，能動的)
  - 遷移(transition)
    - ある状態(ソース状態)から次の状態(ターゲット状態)への変化
    - アクティブ(active): 遷移により状態に入ること
    - 非アクティブ(inactive): 遷移により状態から出ること
  - トリガー(trigger)
    - 状態の変化を引き起こすイベント(出来事)
- トリガーに対する振る舞いは現在の状態によって変化
- 通常，単一のオブジェクトに対して記述

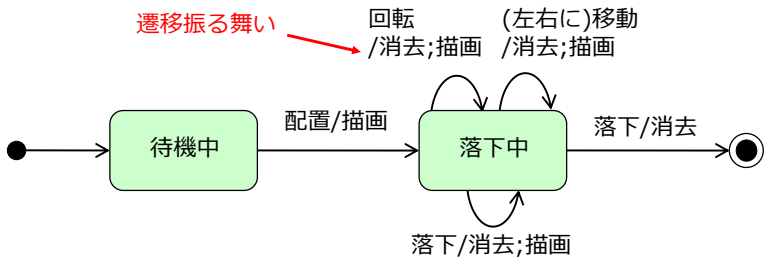
状態機械図(状態と遷移)

- オブジェクトは必ず1つの状態に属する(複合状態を除く)
- トリガーが記述されていないとき，自動的に遷移
- トリガーは一瞬
- 遷移時間は無視



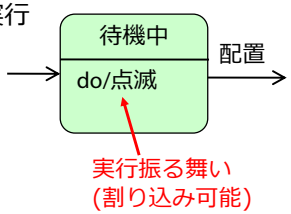
状態機械図(振る舞い)

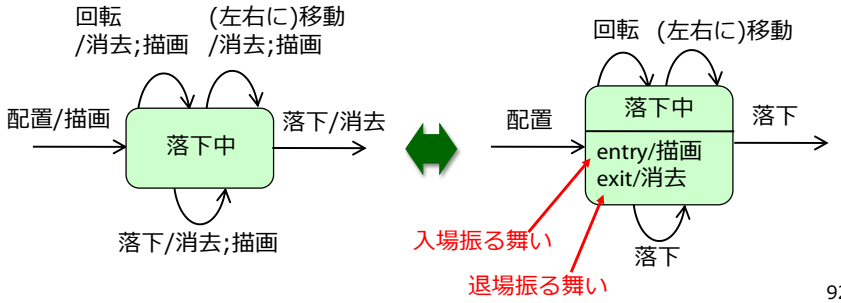
- 遷移振る舞い
  - 遷移が行われる間に実行される振る舞い
- 内部振る舞い
  - オブジェクトがある状態の間に実行される振る舞い



状態機械図(内部振る舞い)

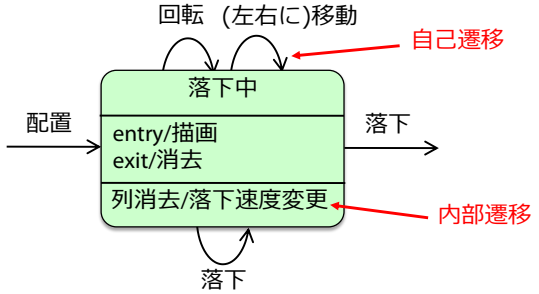
- 実行振る舞い/実行アクティビティ(do)
  - 状態がアクティブである間，継続的に実行
- 入場時振る舞い(entry)
  - 状態がアクティブになった直後に実行
- 退場時振る舞い(exit)
  - 状態が非アクティブになる直前に実行





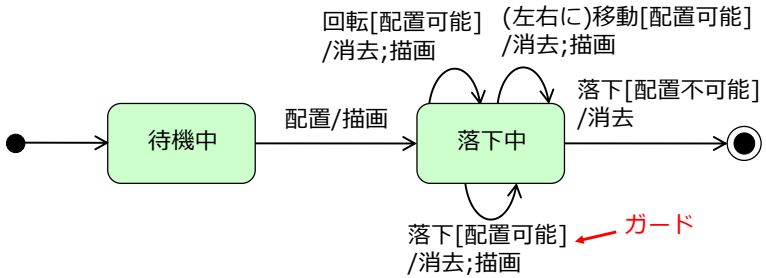
状態機械図(自己遷移と内部遷移)

- 自己遷移(self-transition)
  - 状態遷移が発生(たまたま同じ状態に遷移)
    - 状態の入場時振る舞いと退場時振る舞いが実行
- 内部遷移(internal transition)
  - 状態遷移なし
    - 状態の入場時振る舞いと退場時振る舞いは未実行



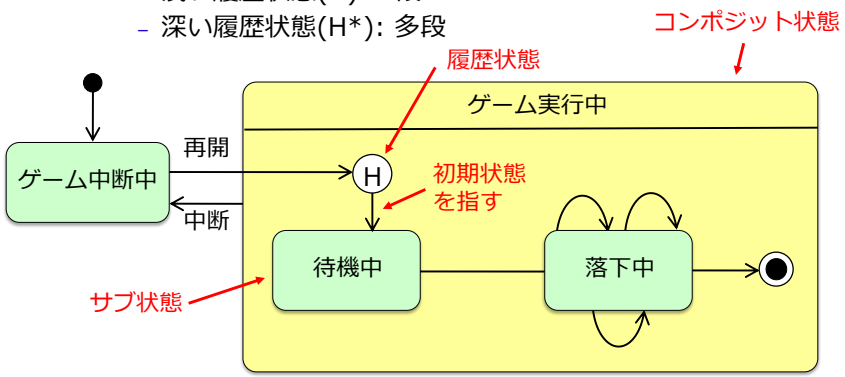
状態機械図(ガード)

- 遷移を許可あるいはブロックする条件
  - ガードが成立するとき，状態が遷移
  - ガードが成立しないとき，状態遷移なし
- トリガーとガードが記述
  - トリガーの発生によりガードを検査
- ガードのみ
  - 内部振る舞いの完了直後にガードを検査



状態機械図(コンポジット状態とサブ状態)

- コンポジット状態(composite state)
  - サブ状態(sub-state)の入れ子を表現
- 履歴状態(history state)
  - コンポジット状態を出る直前の状態を保持
    - 浅い履歴状態(H): 1 段
    - 深い履歴状態(H*): 多段



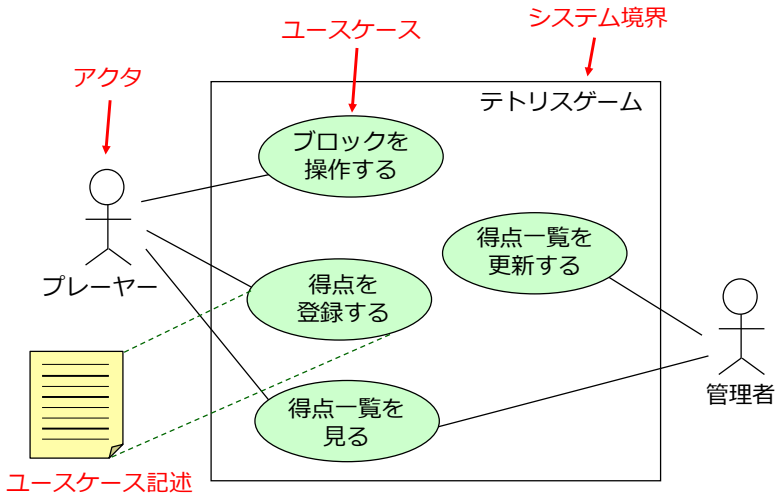
イベント

- シグナルイベント
  - シグナル(«signal»を利用)の発生(送受信)
  - 非同期的
    - キー入力，マウス移動
- 呼び出しイベント
  - オブジェクトの操作に対する呼び出しの受信で発生
  - 同期的
- 時間イベント
  - 特定の時間経過後や時刻に発生
    - after (2秒)，at (2010年10月1日午後1時)
- 変化イベント
  - 状況の変化(条件が偽から真になった瞬間)で発生
    - when (得点 >10000)

ユースケース図(use-case diagram)

- システムの機能ごとに作成
- システムの利用者側(アクタ)からみた使われ方を表現したもの
  - アクタ(actor)
    - システムに対して利用者が果たす役割(role)
    - 役割ごとに異なるアクタが存在
    - 外部システムでもよい
    - 受益者でなければならない
- 利用者の目的に照らして結び付けられた一群のユースケース記述(シナリオ)を持つ
  - シナリオ(scenario)
    - 利用者とシステム間の対話を表す一連の手順
    - ユースケースのインスタンス
- 要求の獲得，反復計画，妥当性検査に適用

ユースケース図(アクタとユースケース)



ユースケース記述

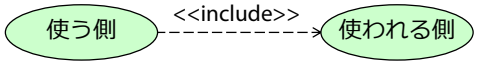
名前　得点を登録する
アクタ　プレーヤー
事前条件　ゲームが終了している(得点が存在する)
事後条件　登録した得点が一覧に存在する

基本フロー
1. ゲーム終了時に，プレーヤーは「得点を登録する」を選択する
2. プレーヤーは，名前(あるいはニックネーム)を入力する
3. テトリスゲームは，画面に得点一覧を表示する

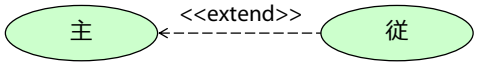
代替フロー
2. 得点が100位以下の場合は，テトリスゲームは得点の登録を中止する
   a. 失敗
3. すでに同じ名前が存在する
   a. テトリスゲームは，入力した名前がすでに存在することを通知する
   b. 2に戻る

ユースケース間の関係

- 包含(include):
  - 複数のユースケースに現れる共通の振る舞いを括り出し
  - 既存のユースケースを再利用

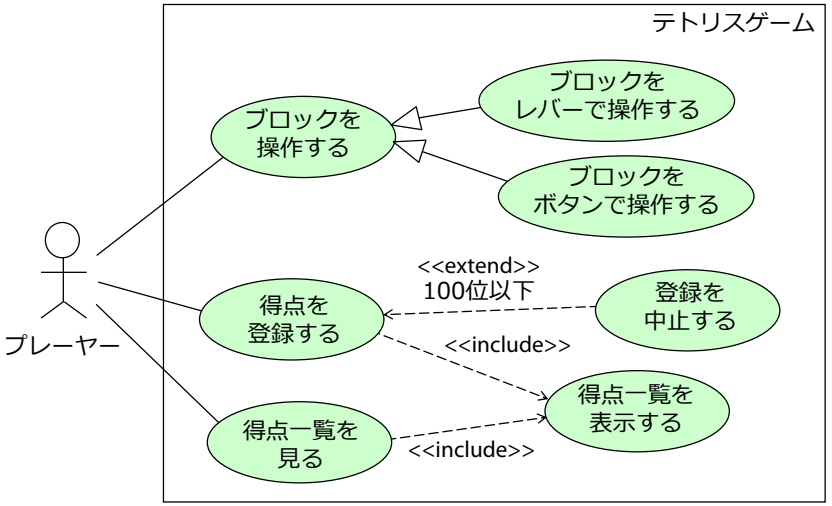


- 拡張(extend):
  - 異なる振る舞いを分離(例外ユースケースなど)



- 汎化(generalization):
  - 基本ユースケースの振る舞いのバリエーションを記述

ユースケース図(ユースケース間の関係)



テトリスゲームの開発(要求仕様の追加)

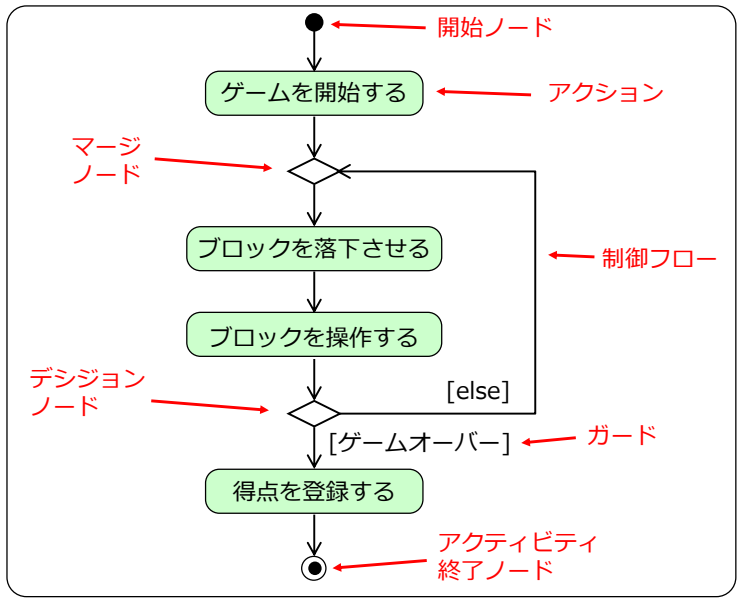
テトリスゲーム(追加)

- プレーヤーはインターネットを通じて得点を登録することができる.
- プレーヤーは登録されている得点の一覧を見ることができる.
- ゲームの管理者は得点の一覧を見ることができる．さらに，登録されている得点一覧を更新することができる.

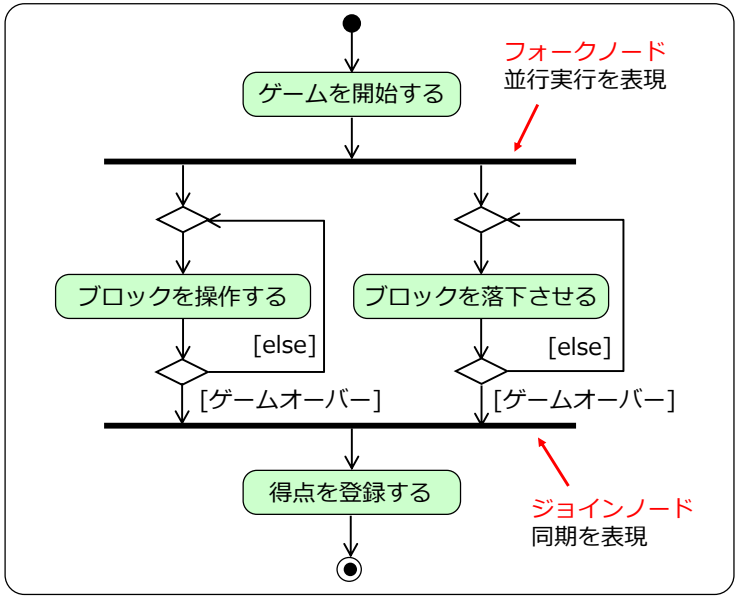
アクティビティ図(activity diagram)

- プロセスモデリングに利用
  - プロセスにおけるワークフローを表現
- 個々のアクティビティをどのように実現するかを表現
  - 必要なアクション群とそれらの依存性(順序，選択，並行性)を明記
    - アクティビティ(activity): モデリング対象のプロセス
    - アクション(action): アクティビティ全体の中で実行される計算や作業(タスク)
- データ(オブジェクト)の流れやシグナルの送受信も表現可能

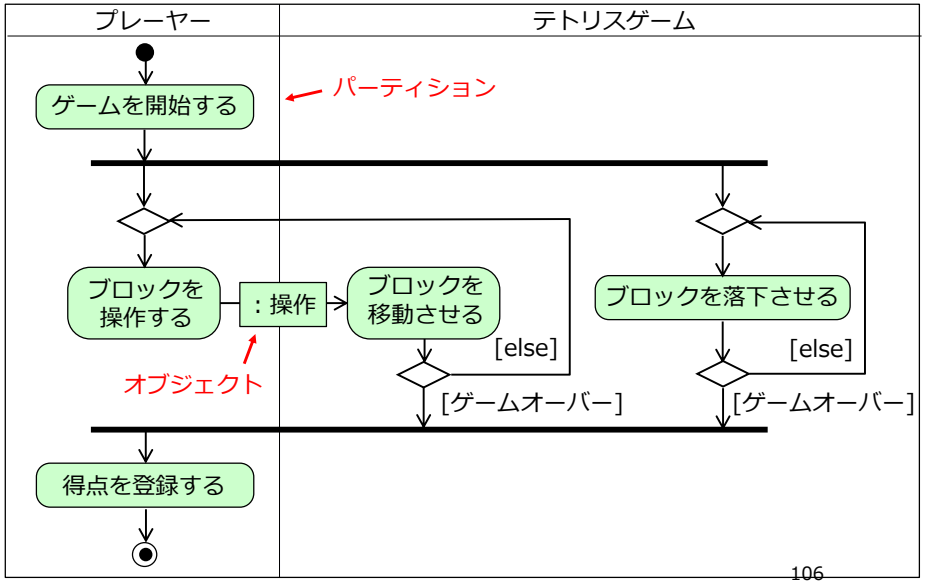
アクティビティ図(アクションと制御フロー)



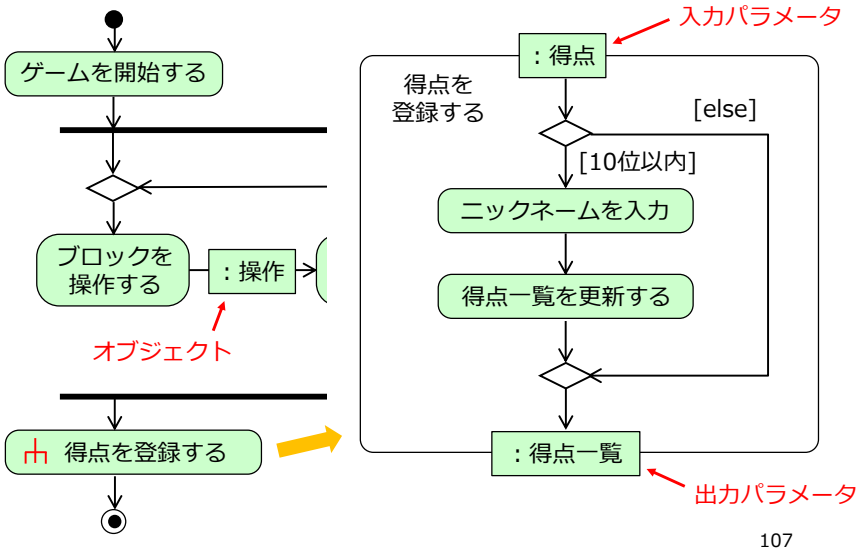
アクティビティ図(フォークとジョイン)



アクティビティ図(パーティション)



アクティビティ図(アクションの分解)



オブジェクト指向設計・実装

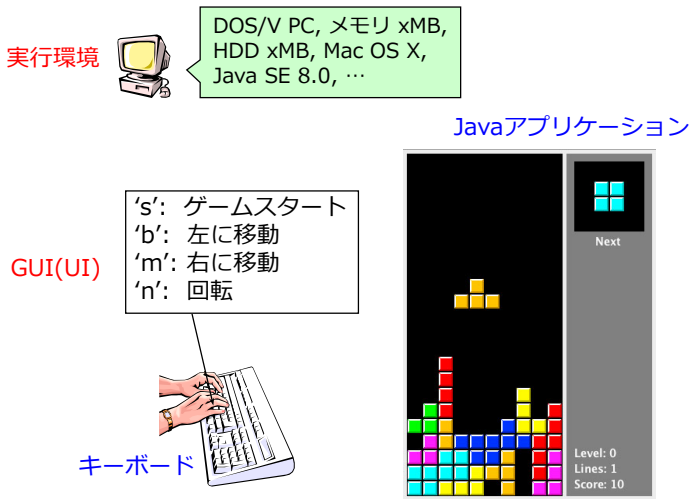
オブジェクト指向設計・実装

- 設計と実装に明確な区別なし
- オブジェクト指向設計(OOD)
  - ソフトウェアとして実行するために必要な実装方法を定義
  - システム設計
    - システムレベルの構成と振る舞いを設計
  - 詳細設計
    - システム内部の構造と振る舞いを開発者の視点から具体化
- オブジェクト指向プログラミング(OOP)
  - クラスの実装(コーディング)
  - テスト(単体テスト，統合テスト)

システム設計

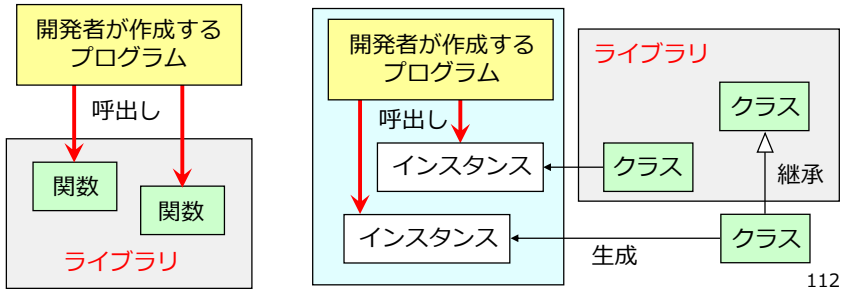
- アーキテクチャの全体構造の設計
  - クライアント/サーバ(C/S)
  - 3層(3-tier)アーキテクチャ
  - データベース管理システム(DBMS)
- ハードウェア環境の決定
  - 機器
  - 入出力デバイス
- ソフトウェア環境の決定
  - 実装言語
  - ライブラリ
  - フレームワーク
- サブシステム分割とインターフェースの定義
- GUI(UI)の設計
- データベースのスキーマの設計，データの正規化
- …

システム設計(テトリスゲーム)

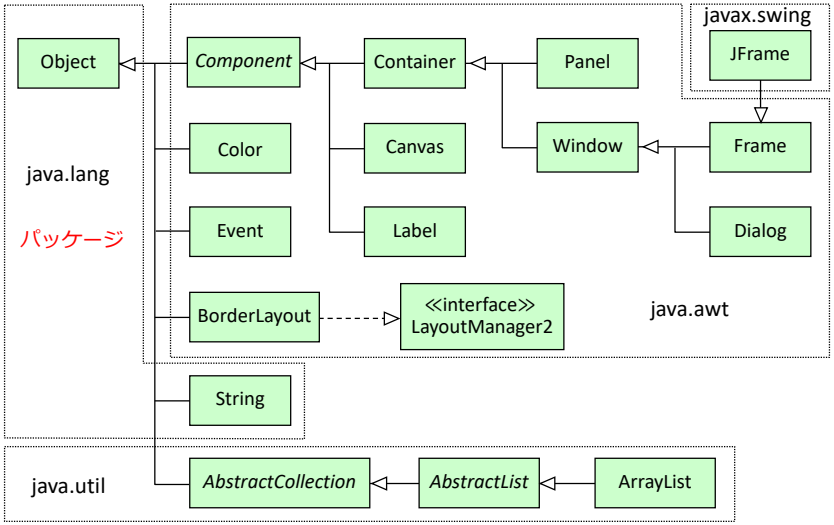


ライブラリ

- 再利用されることを意図して作成されたモジュールの集まり
  - 手続き(関数)の集まり
    - C言語ライブラリなど
  - オブジェクト指向におけるクラスライブラリ
    - クラス = 独立性の高いモジュール
    - 実装の隠蔽(カプセル化)による置換が容易
    - 既存クラスのインスタンス生成と継承による再利用



OOライブラリ(例:Java)

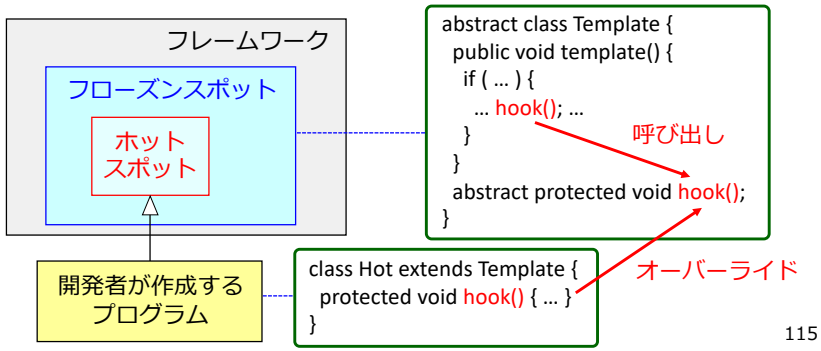


フレームワーク(framework)

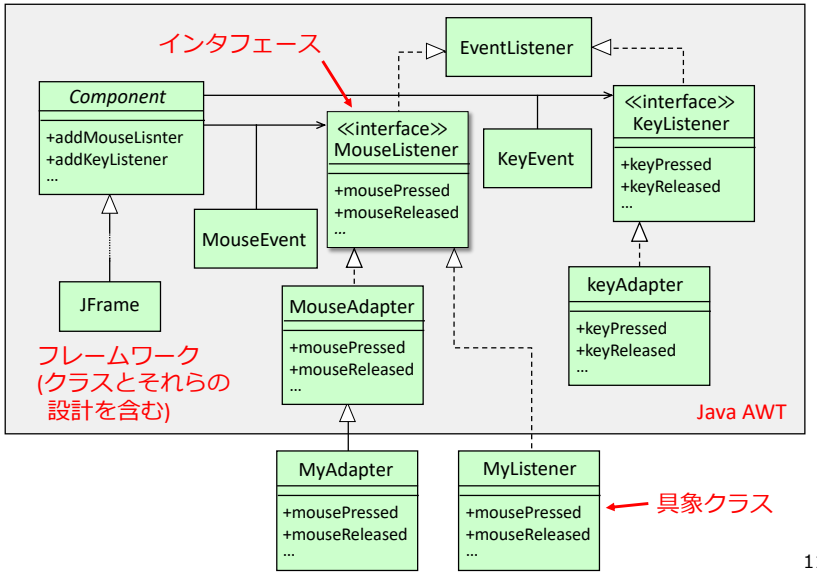
- (汎用)フレームワーク
  - 多くのアプリケーションに共通するアーキテクチャ
    - MVCフレームワーク
  - 特定のアプリケーションドメインを対象
    - 金融アプリケーションフレームワーク
- オブジェクト指向フレームワーク(OOFW)
    - Java AWT, Java Swing, .NET
  - 再利用されるクラスとそのインスタンス間の相互作用を内包する設計
    - コード(クラス)の再利用 + 設計の再利用
  - 開発者がカスタマイズ(拡張)できる半完成アプリケーション
    - 抽象クラス(インタフェース)を含む
    - 具象クラスを差し込むことでカスタマイズ

フローズンスポットとホットスポット

- フローズンスポット(frozen spot)
  - ドメインに対して標準(固定)的な部分
  - テンプレートメソッド(template method)で実装
- ホットスポット(hot spot)
  - 特定の要求に対応する柔軟な部分
  - フックメソッド(hook method)で実装

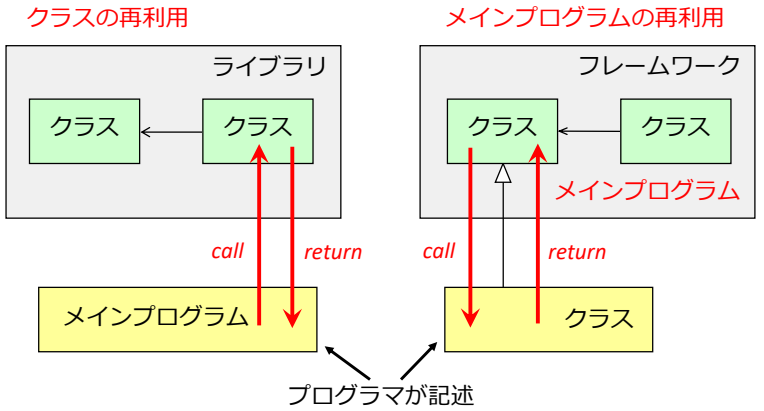


オブジェクト指向フレームワーク(例:Java)



制御の逆転(inversion of control)

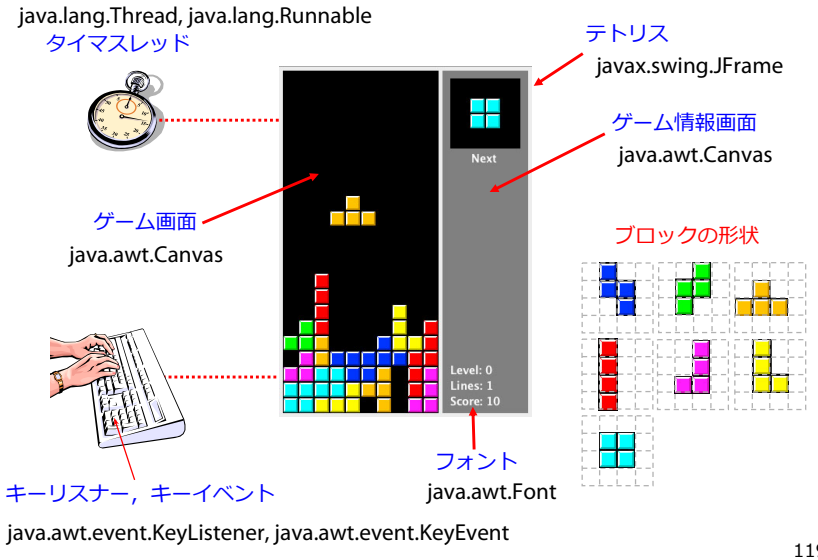
- フレームワークが制御の流れを管理
  - メインプログラムを内包
  - 呼び出されるクラスを作成



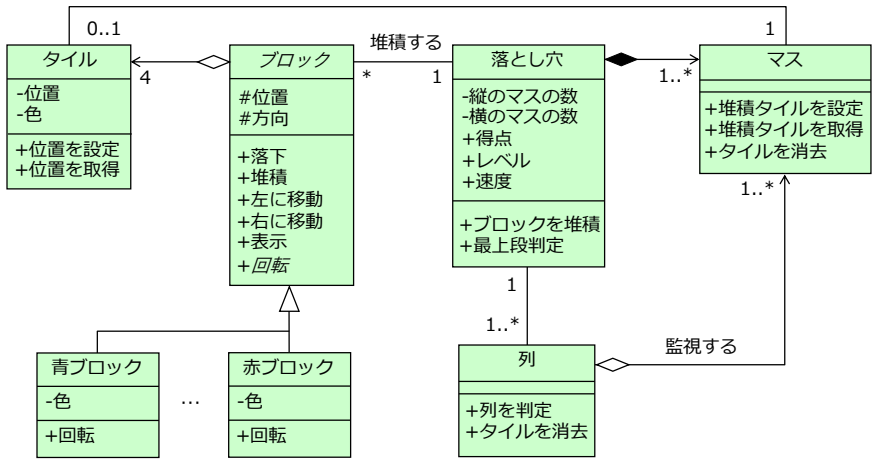
詳細設計

- コンピュータ領域のクラスの抽出
  - システムそのものを指すクラスの導入
  - インタフェースクラスの導入: システムとユーザの接点に存在するクラス
    - キーボード，マウス，画面，プリンタ
  - 制御クラスの導入: システムを実現する上で別途必要なクラス
    - イベント管理，ファイル管理，スレッド管理
- アルゴリズムの決定
- パターンの定義と適用
- 関連の実現方法の決定(方向，多重度，可視性)
- 属性の設計(可視性，アクセス関数，初期値)
- 機能の設計(操作の内部仕様の設計)
- 状態遷移の設計(状態の管理方法の決定)
- …

詳細設計(テトリスゲーム)



クラス図(5) [再掲]



クラス図(6) [詳細設計]