第1～2回
オブジェクト指向の基本概念

オブジェクト指向

- オブジェクト指向(object-oriented)
  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

オブジェクト指向のアプローチ

データ中心のアプローチ

認知科学における概念
- 属性(attribute): 何からできているのか．構造による認知
- 外延(extension): 何か，何と似ているのか．分類による認知
- 内包(intension): 何ができるのか．機能による認知

機能中心のアプローチ

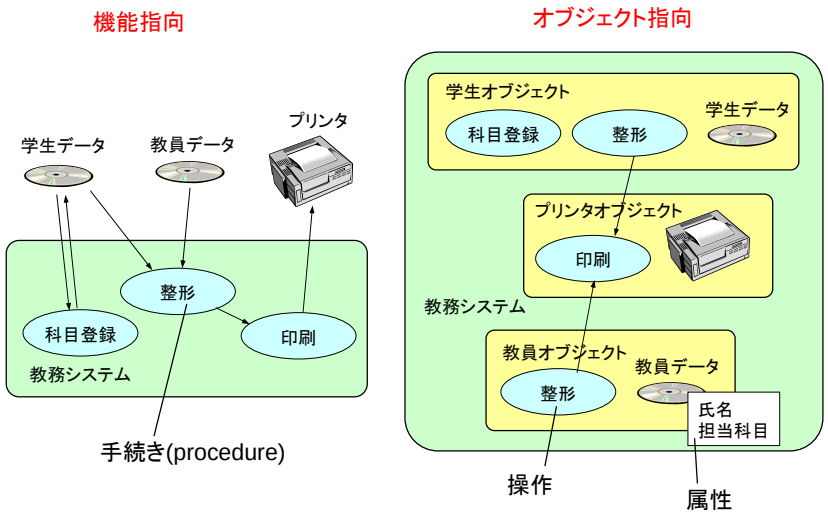
属性 = クラスの属性
外延 = クラスの継承
内包 = クラスの操作

- 人間の認知方法にできるだけ近づけたモデル化
- 実世界のシステムのモデルの直感的表現方法
  → 人間にとって理解しやすい

オブジェクト指向(cont'd)

- 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」
- 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，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
- オブジェクト指向ソフトウェア開発(OO software development)
  オブジェクト指向分析(OOA: OO analysis)
  オブジェクト指向設計(OOD: OO design)
  オブジェクト指向プログラミング(OOP: OO programming)
  – オブジェクト指向プログラミング言語
  (OOPL: OO programming language); Smalltalk, Java, C++, C#

機能指向とオブジェクト指向

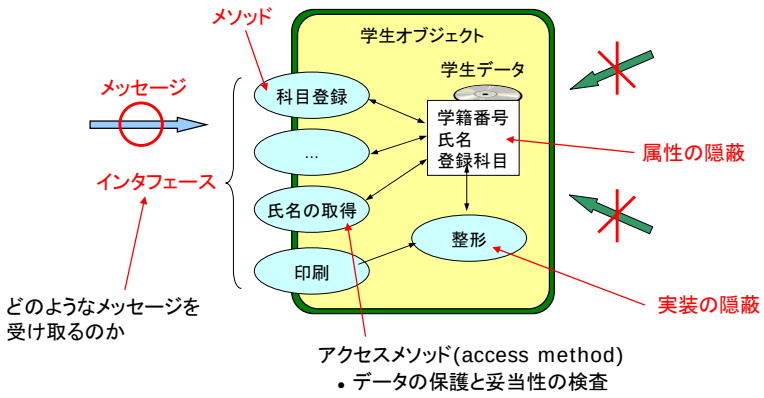


オブジェクト指向の基本概念

- カプセル化(encapsulation)
- メッセージパッシング(message passing)
- クラス(class)とインスタンス(instance)
- 継承(inheritance)
  複数のクラスにおける共通あるいは類似の性質を共有させる仕組み，is-a関係
  既存クラスの属性と操作の引継ぎと洗練 = 差分プログラミング
- 集約(aggregation)
  複数のオブジェクトをひとまとめにして扱う仕組み，part-of関係
- 関連(association)
  あるオブジェクトが別のオブジェクトを利用あるいは参照する関係
- 多相性/多態性(polymorphism)
  ひとつの「もの」がさまざまな姿をとることができる仕組み
  ひとつのオブジェクトが複数のクラスに属することが可能
  応答するオブジェクトは受信側で実行時に決定 = 動的束縛(dynamic binding)

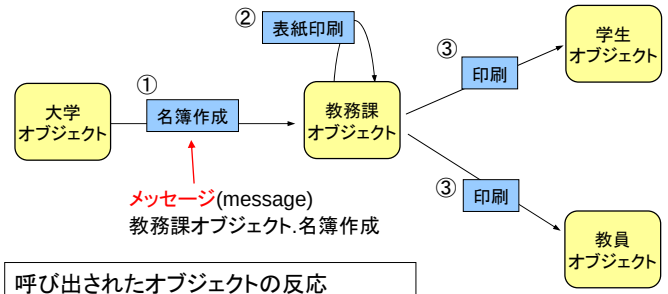
カプセル化

- データとその操作のグループ化，抽象データ型に基づくモジュール化
- インタフェースと実装の分離，インタフェースを介したデータ操作
  → 情報隠蔽(information hiding)



メッセージパッシング

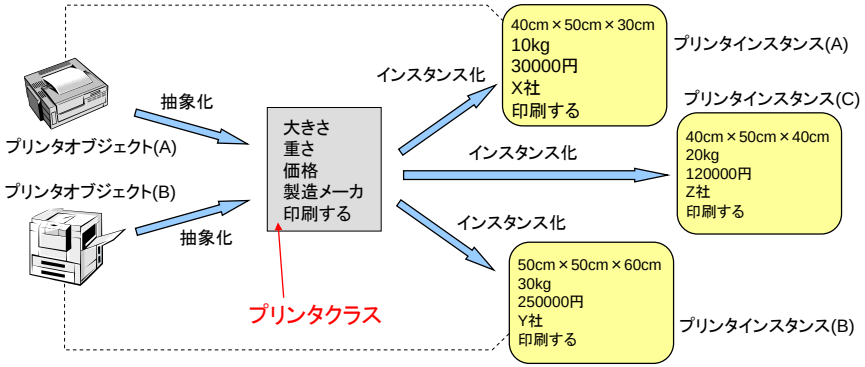
- 個々のオブジェクトに対して処理を依頼する仕組み
- オブジェクトに対する操作実行の依頼
  ＝ オブジェクトに対するメソッド呼出し(method invocation)



呼び出されたオブジェクトの反応
- 属性の値を変える(状態を変化させる)
- 属性値や計算結果を返信する
- 別のオブジェクトをさらに呼び出す

クラスとインスタンス

- 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(template, mold)
- 型(type) + 実装(implementation)
  型:どのような操作を提供するかを定義
- オブジェクトの設計図，生成されたオブジェクト = インスタンス

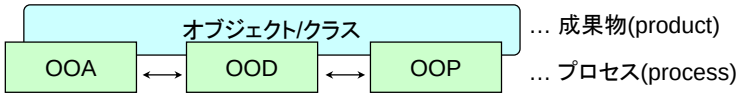


OOソフトウェア開発プロセス

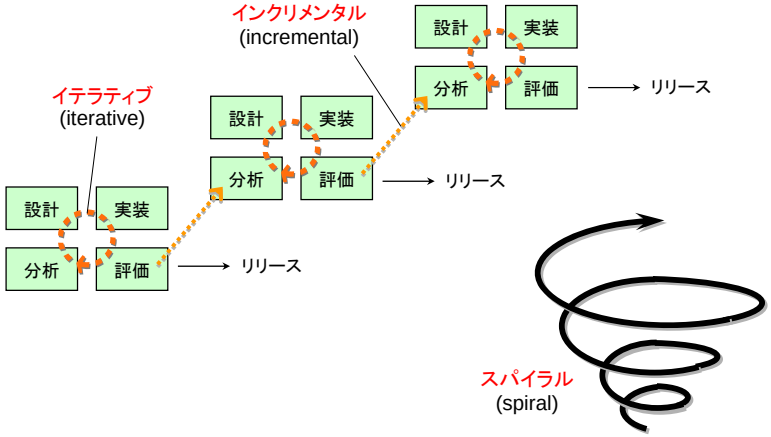
オブジェクト指向ソフトウェア開発 = クラスを作成すること

- OO分析(OOA: OO analysis)
  システムが対象とする領域の構造を定義
  - ✓ 要求分析，要求仕様の作成
  - ✓ クラス抽出，クラス間の関係の抽出，クラスの振る舞いの定義
- OO設計(OOD: OO design)
  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
  - ✓ コンピュータ領域のクラスの抽出，アルゴリズムの決定
  - ✓ アーキテクチャの決定
- OOプログラミング(OOP: OO programming)
  クラスの実装，テスト

シームレス(seamless)な連携



OOソフトウェア開発プロセス (cont'd)



OOソフトウェア開発の利点

- 拡張性(extendibility)
  カプセル化による変更の局所化と実装の隠蔽
- 再利用性(reusability)
  継承による差分プログラミング(programming by difference)
  抽象クラスと動的束縛によるフレームワーク(framework)の構築
  抽象化による部品のブラックボックス(black-box)化と
  インタフェースに基づく合成
- イベント駆動アプリケーションや分散アプリケーションとの親和性
  メッセージパッシングによるオブジェクトの協調

OO方法論(1)

- Shlaer/Mellor [OOSA] (1988):
  S. Shlaer and S. J. Mellor, *Object-oriented System Analysis*, Yourdon Press (1988)
  「オブジェクト指向システム分析」, 近代科学社 (1990)
  - ✓ 情報モデル(静的構造) + 状態モデル(イベントの実行順序)
    + プロセスモデル(データ変換)
  - ✓ オブジェクトの関係に重点
- Coad/Yourdon [OOA/OOD] (1990):
  P. Coad and E. Yourdon, *Object-Oriented Analysis*, Yourdon Press (1990)
  「オブジェクト指向分析(OOA) 第2版」, トッパン (1993)
  P. Coad and E. Yourdon, *Object-Oriented Design*, Yourdon Press (1991)
  「オブジェクト指向設計OOD」, プレンティスホール (1995)
  - ✓ 汎化-特化関係，全体-部分関係の導入
  - ✓ サブジェクト(関連の深いオブジェクトの集合)の導入
- Booch (1990, 1993):
  G. Booch, *Object-Oriented Design with Applications*, Benjamin/Cummings (1991)
  G. Booch, *Object-Oriented Analysis and Design with Applications*, 2nd ed., Addison-Wesley (1994)
  - ✓ 要求に基づくシナリオを導入
  - ✓ 実践的な手法

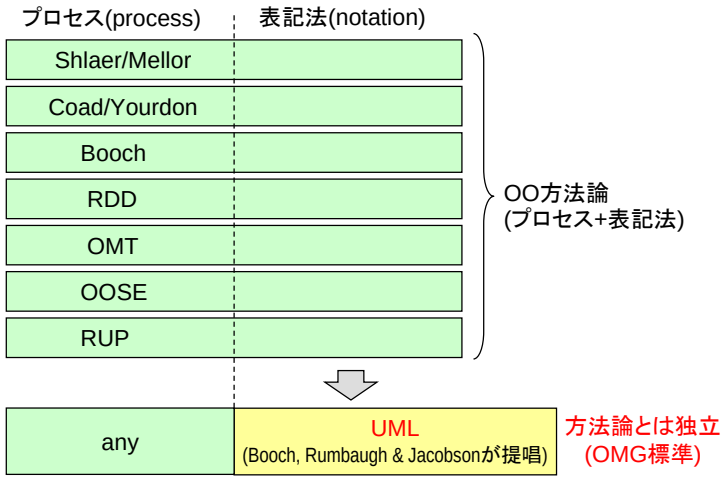
OO方法論(2)

- Wirfs-Brock [RDD: Responsibility-Driven Design] (1990):
  R. Wirfs-Brock et al., *Designing Object-Oriented Software*, Prentice-Hall (1990)
  - ✓ アプリケーションをクラス(class), その責任(responsibility), クラス間の協調(collaboration)によってモデル化
  - ✓ 契約(contract)/プロトコル(protocol)に基づくクラスの分解
- Rumbaugh [OMT: Object Modeling Technique] (1991):
  J. Rumbaugh, *Object-Oriented Modeling and Design*, Prentice-Hall (1991)
  「オブジェクト指向方法論OMT」, トッパン (1992)
  - ✓ 代表的な手法や指針を網羅
  - ✓ オブジェクトモデル: システムの静的な構造を拡張ER図(Entity-Relation diagram)で表現
  - ✓ 動的モデル: オブジェクトの時間的な動作を, 状態遷移図(state transition diagram)とイベントトレース図(event trace diagram)で表現
  - ✓ 機能モデル: オブジェクト間のメッセージ送受信によって発生するデータの流れをデータフロー図(data flow diagram)で表現
- Jacobson [Objectory, OOSE] (1992):
  I. Jacobson et al.,*Object-Oriented Software Engineering*, Addison-Wesley (1992)
  「オブジェクト指向ソフトウェア工学OOSE」, トッパン (1995)
  - ✓ ユースケース(use-case)中心

OO方法論(3)

- Martin/Odell (1992):
  J. Martin, *Principles of Object-Oriented Analysis and Design*, Prentice-Hall (1992)
  J. Martin and J. J. Odell, *Object-Oriented Methods: A Foundation: UML Edition*, Prentice-Hall (1994)
- Fusion (1993):
  D. Coleman et al., *Object-Oriented Development: The Fusion Method*, Prentice-Hall (1993)
- ROOM (1994):
  B. Selic et al., *Real-Time Object-Oriented Modeling*, John Wiley and Sons (1994)
- OOram (1995):
  T. Reenskaug et al.,*Working With Objects*, Manning Publications (1995)
- Catalysis (1998):
  D. F. D'Souza and A. C. Wills, *Objects, Components, and Frameworks with UML*, Addison-Wesley (1998)
- RUP [Rational Unified Process] (1995):
  I. Jacobson, G. Booch and J. Rumbaugh, *Unified Software Development Process*, Addison-Wesley (1999)
  「UMLによる統一ソフトウェア開発プロセス」, 翔泳社 (2000)
  P. Kruchten, *The Rational Unified Process: An Introduction*, Addison-Wesley (1999)
  「ラショナル統一プロセス入門」, ピアソン・エデュケーション (1999)
- XP [eXtreme Programming] (1996):
  K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley (1999)

UML(Unified Modeling Language)



OMG: Object Management Group
http://www.omg.org/technology/uml/index.htm

UMLダイアグラム(UML 2.0)

静的構造(structure)

◎ (1) 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
(2) 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
(3) 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
(4) 複合構成図(composite structure diagram): 実行時のクラスの内部構造
(5) 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
(6) 配置図(deployment diagram): システムにおける物理的な配置

動的振る舞い(behavior)

◎ (7) 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
(8) アクティビティ図(activity diagram): 作業の順序と並行性
◎ (9) 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
(10) 相互作用図(Interaction diagram)
◎ ー シーケンス図(Sequence diagram): オブジェクト間の相互作用の時系列
◎ ー 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
ー 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
ー相互作用概要図(Interaction overview diagram): シーケンス図とアクティビティ図の概要

第3～7回
オブジェクト指向分析(OOA)
要求分析
静的モデリング

オブジェクト指向分析

- オブジェクト指向分析(OOA)
  システムが対象とする領域の構造を定義
  1) 要求分析(requirements analysis): 要求仕様を定義
     - ユースケース
  2) 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
  3) 動的モデリング(dynamic modeling): クラスの振る舞いを定義
     - オブジェクトの状態遷移の定義
     - オブジェクト間の相互作用の認識

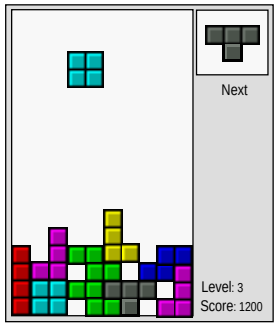
オブジェクト指向分析(cont'd)

- データ駆動型アプローチ(data-driven approach)
  1) システム中のデータを認識
  2) システムを構成するオブジェクトを抽出
  3) オブジェクト間の関係を抽出
  4) 個々のオブジェクトの振る舞いを決定

  Shlaer/Mellor法，Coad/Yourdon法，Booch法, OMT法
- 責任駆動型アプローチ(responsibility-driven approach)
  1) システムの振る舞い(責任)を認識
  2) システムを構成するオブジェクトを抽出
  3) オブジェクト間の関係を抽出
  3) 個々のオブジェクトの振る舞いを決定

  RDD手法，OOSE，CRCカード法

例題:テトリスゲームの開発



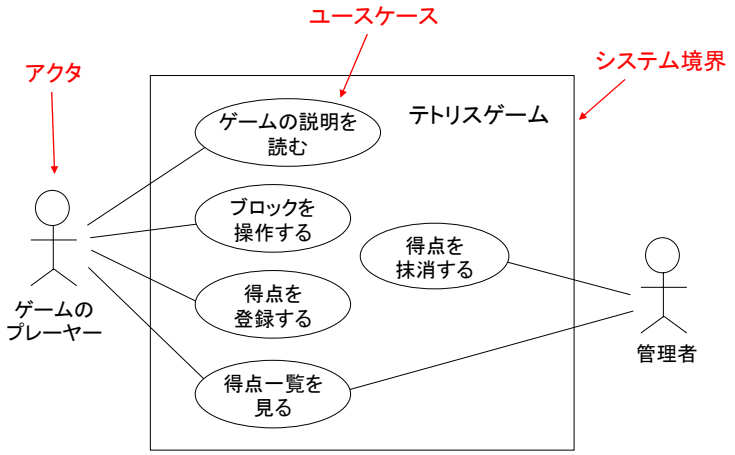
テトリス(Tetris)のルール説明

- 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．

ユースケース

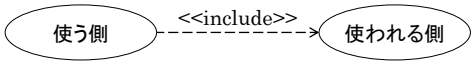
- ユースケース(use case) [Jacobson]:
  - ✓ システムの機能ごとに作成
  - ✓ システムの利用者側(アクタ)からみた使われ方を表現したもの
    - アクタ(actor): システムに対して利用者が果たす役割(role)
      役割ごとに異なるアクタが存在
      外部システムでもよい
      受益者(beneficiary)
  - ✓ 利用者の目的に照らして結び付けられた一群のシナリオ
    - シナリオ(scenario): 利用者とシステム間の対話を表す一連の手順
      ユースケースのインスタンス
  - ✓ 要求の獲得，反復計画，妥当性検査に適用
  - ✓ 図表現(ユースケース図，アクティビティ図)や文章表現

ユースケース図(1)

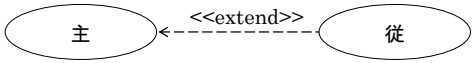


ユースケース間の関係

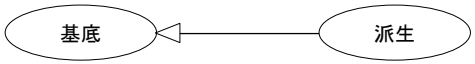
- 包含(include):
  複数のユースケースに現れる共通の振る舞いを括り出し
  既存のユースケースを再利用



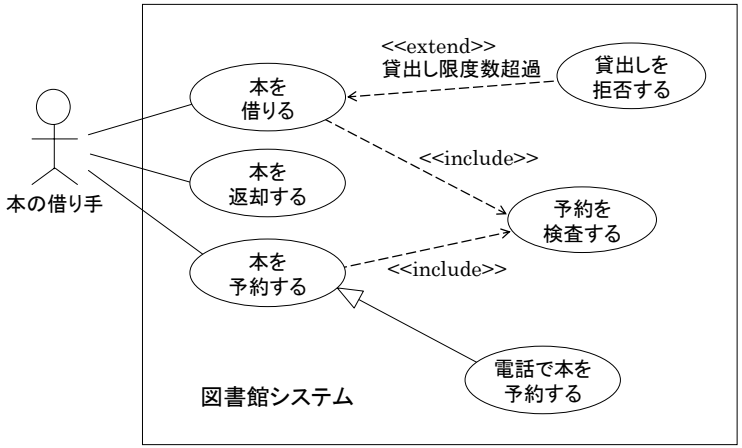
- 拡張(extend):
  異なる振る舞いを分離(例外ユースケースなど)



- 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(2)



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22個，横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．
- ゲームのプレーヤーは，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その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．
- レベルと得点は落とし穴の横に表示される．
- ~~プレーヤーはゲームの説明を読むことができる．~~
- ~~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．また，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を抹消することができる．~~

クラス候補の抽出

- 名詞抽出法(noun identification technique):
  (1) 名詞と名詞句に着目して，クラス候補を抽出
  (2) 不適当な候補(冗長，曖昧，イベントや操作，メタ言語，問題領域外)を除去

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22個，横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．
- ゲームのプレーヤーは，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その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．
- レベルと得点は落とし穴の横に表示される．

クラスの選別

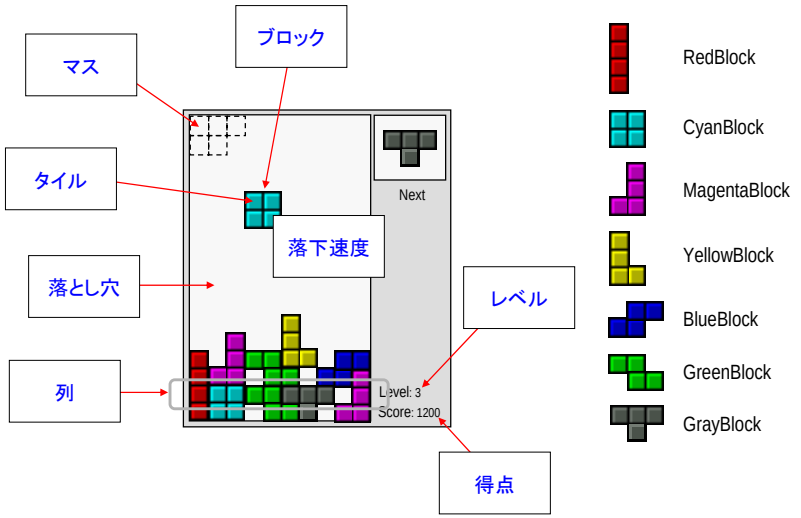
- ○ ブロック(Block)
- ○ 落とし穴(Pit)
- ○ マス(Box)
- ○ タイル(Tile)
- △ ゲーム … 開発するシステムを指す
- × プレーヤー … 問題領域外の概念
- × レバー … コンピュータ領域の概念
- × ボタン … コンピュータ領域の概念
- ○ 列(Line)
- × 数 … 程度を表す
- ○ 得点(Score)
- × 高得点 … 高い得点のこと
- ○ レベル(Level)
- ○ 落下速度(Speed)

○: クラス，×: クラスでない

抽出クラス

ブロック	落とし穴
マス	タイル
列	得点
レベル	落下速度

抽出クラスとシステムの関係



クラスの抽出(Exercise: Question)

図書館システム

- 図書館は本と雑誌を所蔵している.
- 1つの著書に対して，その本は複数冊ある. 雑誌は1冊づつだけある.
- 本および雑誌は，会員であれば3週間まで貸出し可能である.
- 図書館の会員は一度に6冊まで，職員の会員は一度に12冊まで借りられる. 雑誌は職員の会員だけが3冊まで借りられる.
- システムは本と雑誌がいつ貸し出されたのか，いつ返却されたのかをデータベースに記録する.
- 会員は端末から著書名や著者名により本および雑誌を検索することができる.

クラスの抽出(Exercise: An answer)

(図書館 … 開発するシステムを指す)
本(Copy of book)
雑誌(Journal)
著書(Book)
会員(Member)
職員の会員(Member of Staff)
著書名(Title of book)
著者名(Author of book)

クラス図

- 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
  - ✓ OO開発分析，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

(1) 関連(association)
  - ✓ クラスのインスタンス間の一時的な関係

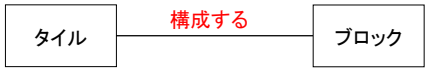
(2) 操作(operation)，メソッド(method) [Java]，メンバ関数[C++]
  - ✓ クラスのインスタンスが受け付けるメッセージ ≅ メソッド(method)

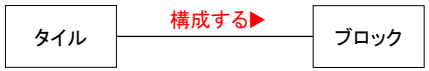
(3) 属性(attribute)，フィールド(field) [Java]，データメンバ[C++]
  - ✓ クラス(インスタンス)そのものが持つデータ ≅ 関連

関連の抽出

- 関連(association): 仕様中の動詞に対応することが多い

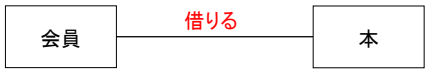
- ブロックは7種類あり，それぞれ4つのタイルで構成されている.





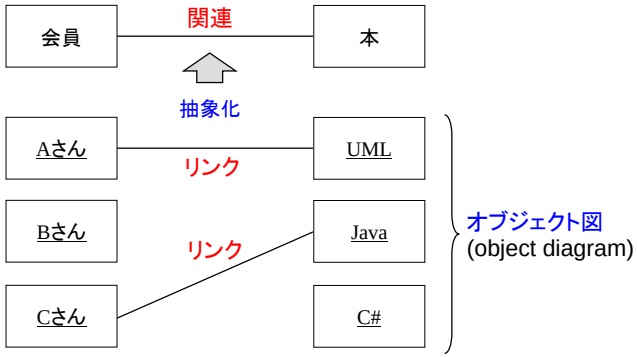
(関連の方向を明記)

- 図書館の会員は一度に6冊まで，職員の会員は一度に12冊まで借りられる.
(「冊」は「本」を指す)

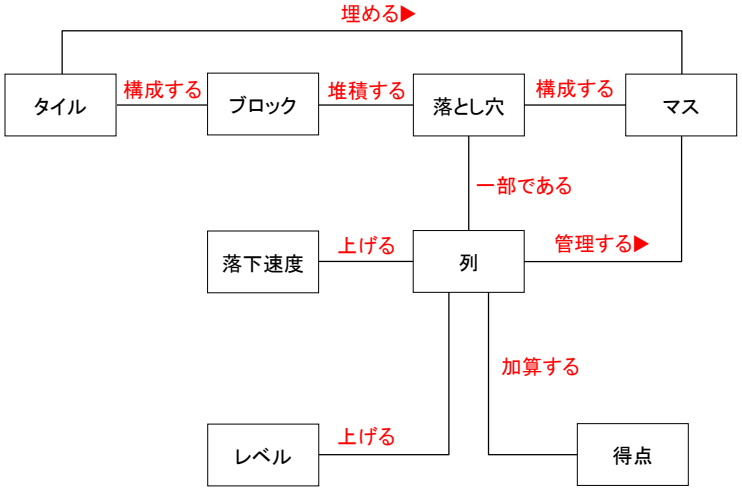


関連とリンク

- リンク(link): インスタンスどうしの一時的な協調関係

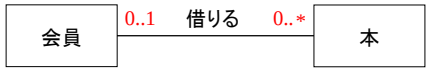


クラス図(tetris-1:関連の追加)



多重度

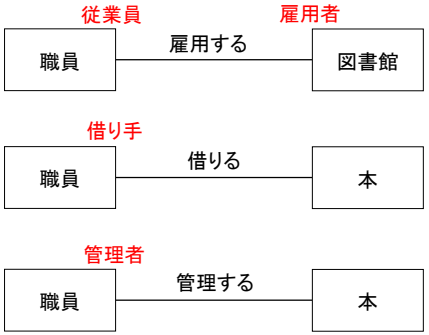
- 多重度(multiplicity): 関連に関与するオブジェクトの数を表現



- 特定の値: その数値を記述
  e.g., 1: 1個
- 値の範囲: 下限値と上限値を".."でつないで記述
  e.g, 1..5: 1個から5個の間
  0..1: 0個か1個
- 任意の値: "*"を記述
  e.g., 0..*, *: 0個以上
  1..*: 1個以上

ロール

- ロール(role): 相手クラスに対する位置付けを表現



誘導可能性

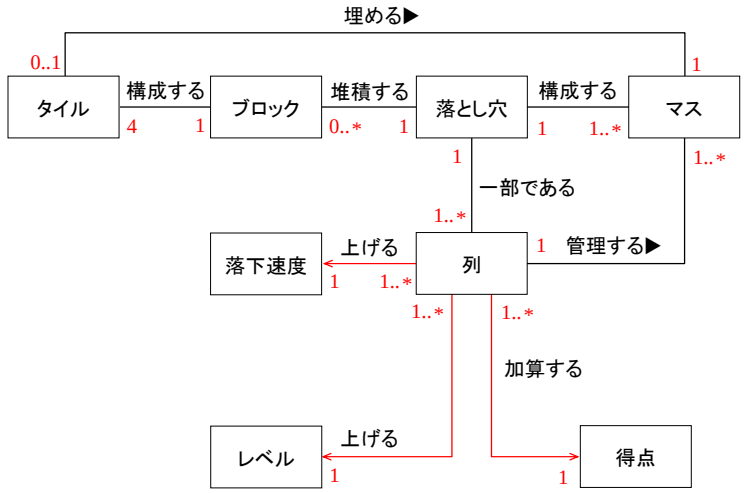
- 誘導可能性(navigability): 責任(参照)の方向を限定する

- 単方向関連(unidirectional association)

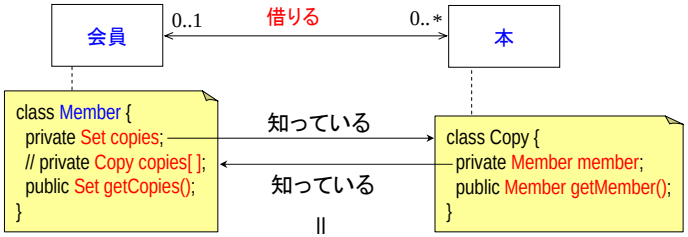
「著書」から「著者」への参照あり(知っている)
「著者」から「著書」への参照なし(知らない)

- 双方向関連(bidirectional association)

クラス図(tetris-2:多重度と誘導可能性の追加)



関連(実装例)

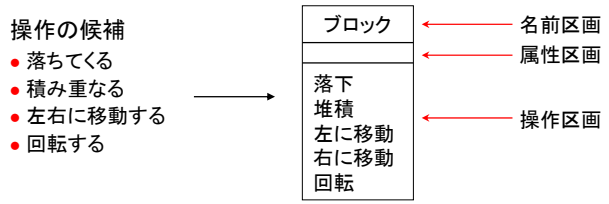


相手インスタンスへの参照を持つ

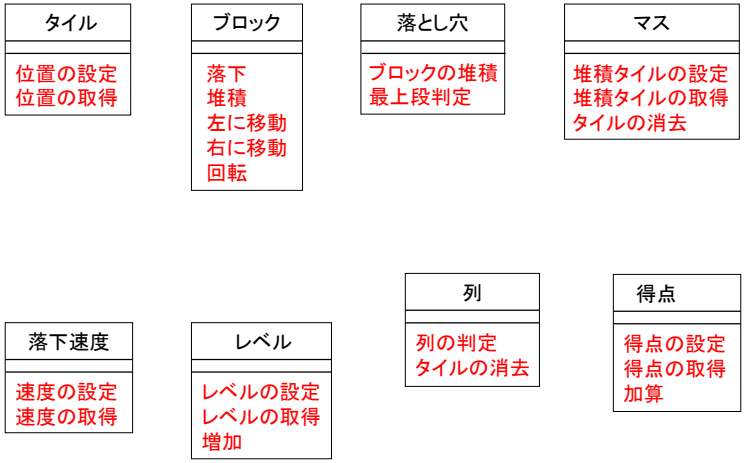
操作の抽出

「ブロック」の仕様(テトリスゲームの仕様から抜粋)

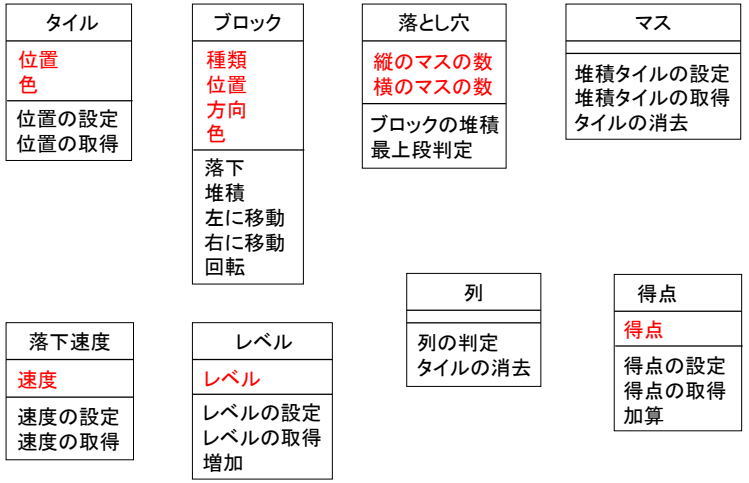
- ブロックが落とし穴の上部から落ちてきて，順番に積み重なる.
- ブロックは7種類あり，それぞれ4つの(マスと同じ大きさの)タイルで構成されている.
- ゲームのプレーヤーは，レバーで落下中のブロックを左右に移動させることができる.
  また，ボタンを押すことでそのブロックを回転させることもできる.
- ブロックが落とし穴の最上部まで積み重なるとゲームが終了となる.
- 消去したタイルの数に応じてゲームのレベルが上がり，ブロックの落下速度が速くなる.
- 次に落ちてくるブロックは落とし穴の横に表示される.



クラス(操作の追加)



クラス(属性の追加)



操作と属性の構文

- 操作の構文

  可視性 名前(パラメータリスト):戻り値の型 { プロパティ文字列 }

  - 可視性(visibility): + (public), # (protected), - (private)
  - 名前(name): 文字列
  - パラメータリスト(parameter list): 方向:型=デフォルト値
    方向(direction): in, out, inout, 省略時in
  - 戻り値の型(return type): 各戻り値の型をコンマで区切って並べたリスト
  - プロパティ文字列(property string): 該当操作に対するプロパティ値

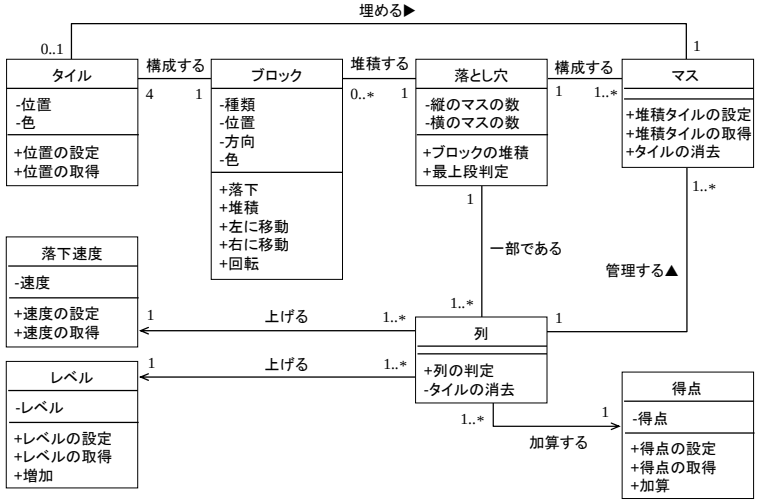
  e.g., +得点の設定(s:Integer)
  +得点の取得():Integer

- 属性の構文

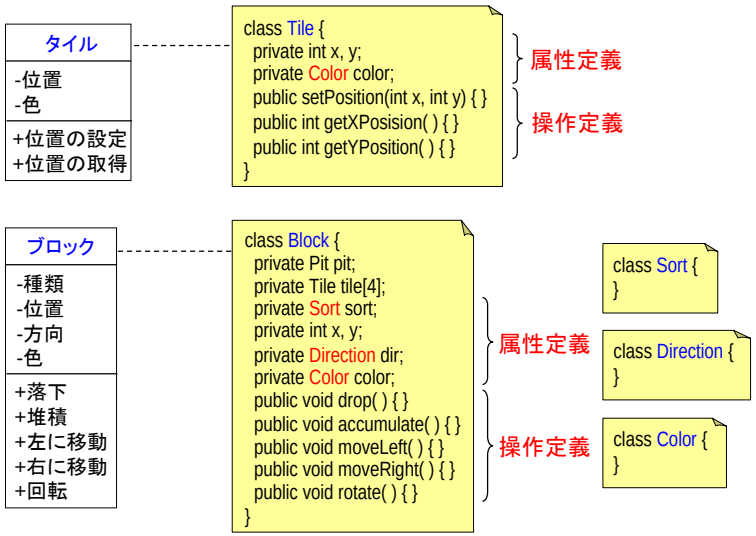
  可視性 名前:型=デフォルト値

  e.g., -得点:Integer = 0

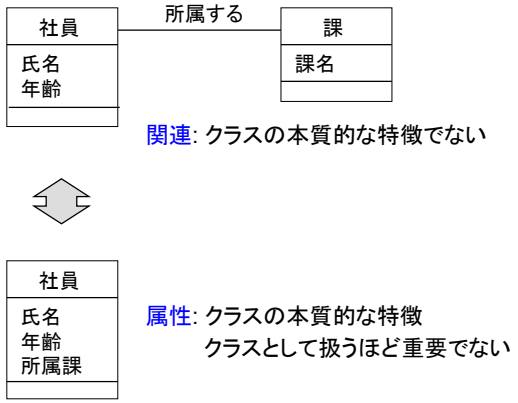
クラス図(tetris-3:操作と属性の追加)



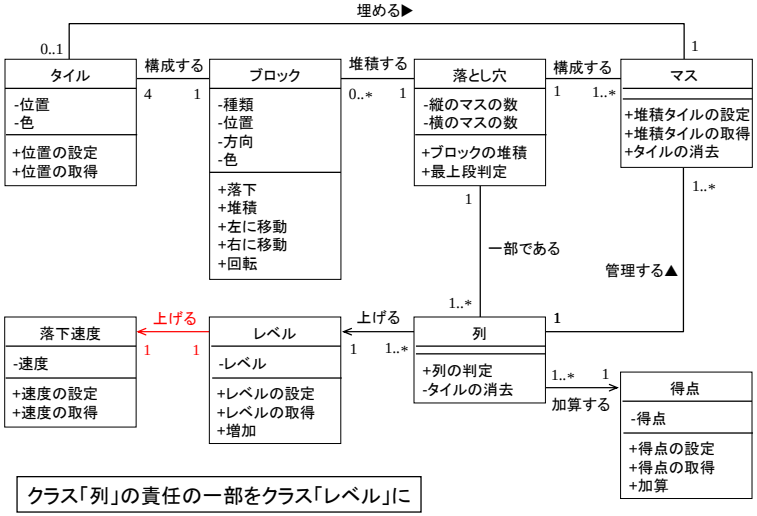
操作と属性(実装例)



関連と属性



クラス図(tetris-4:関連の修正)



クラス「列」の責任の一部をクラス「レベル」に

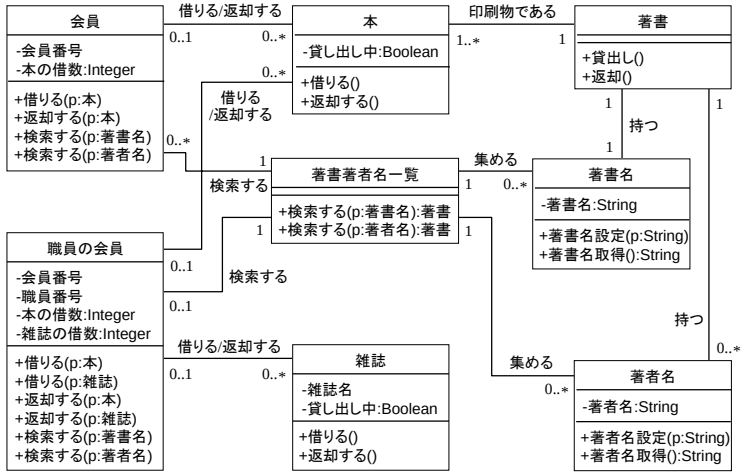
クラス図(Exercise: Question)

図書館システム

- 図書館は本と雑誌を所蔵している.
- 1つの著書に対して, その本は複数冊ある. 雑誌は1冊づつだけある.
- 本および雑誌は, 会員であれば3週間まで貸出し可能である.
- 図書館の会員は一度に6冊まで, 職員の会員は一度に12冊まで借りられる. 雑誌は職員の会員だけが3冊まで借りられる.
- システムは本と雑誌がいつ貸し出されたのか, いつ返却されたのかをデータベースに記録する.
- 会員は端末から著書名や著者名により本を検索することができる.

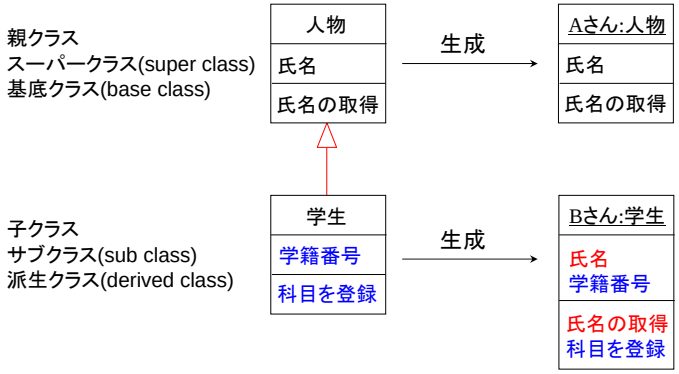
本	雑誌	著書	著書名	著者名
会員	職員の会員			

クラス図(Exercise: An answer)



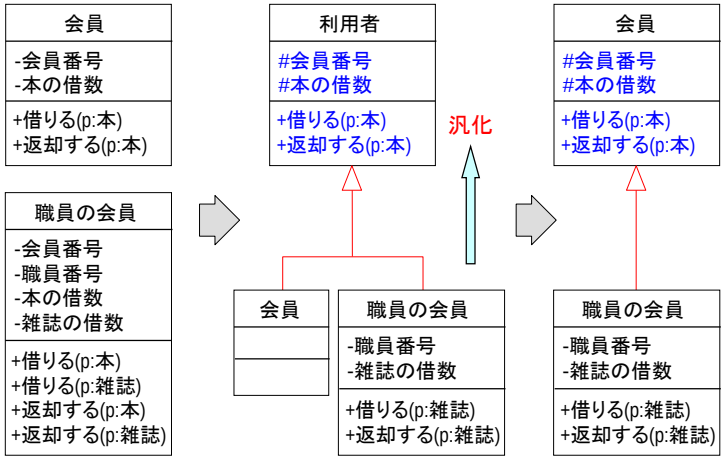
継承

- 継承(inheritance): is-a関係
  - ✓ 他のクラスの性質を引き継ぐための実装
  - ✓ 他のクラスから属性や操作をもってきて，自分と合成して インスタンスを生成する仕組み



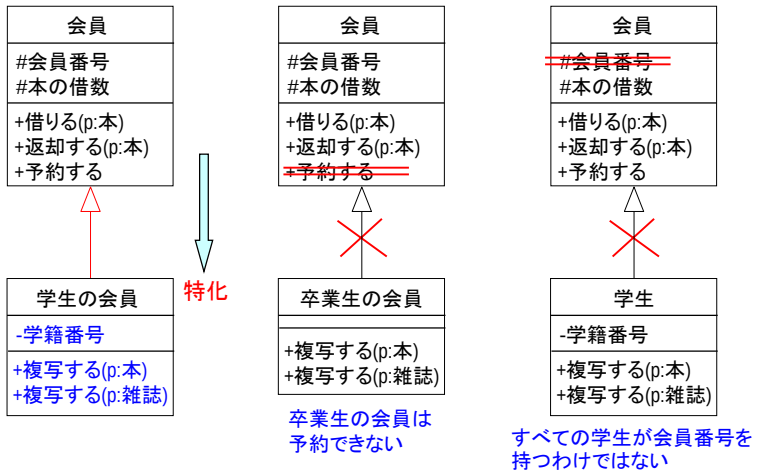
汎化

- 汎化(generalization): 複数クラスの共通概念を抽出して親クラスとして定義



特化

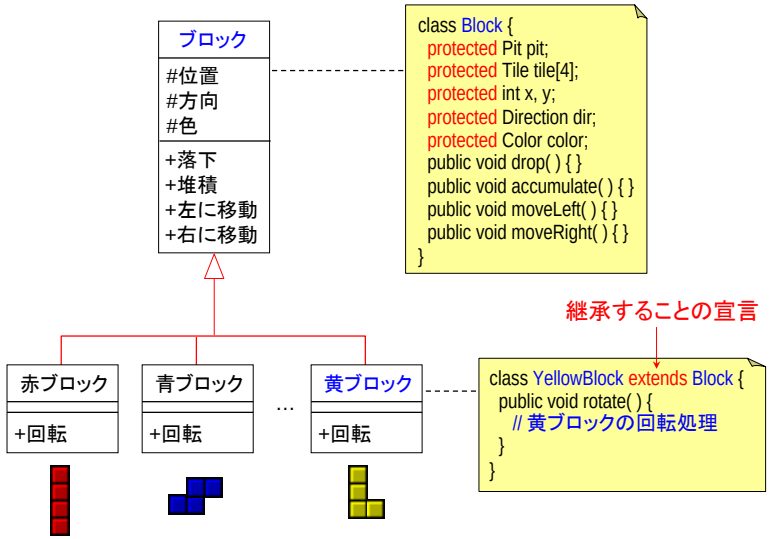
- 特化(specialization): あるクラスを基にして特殊化したクラスを定義



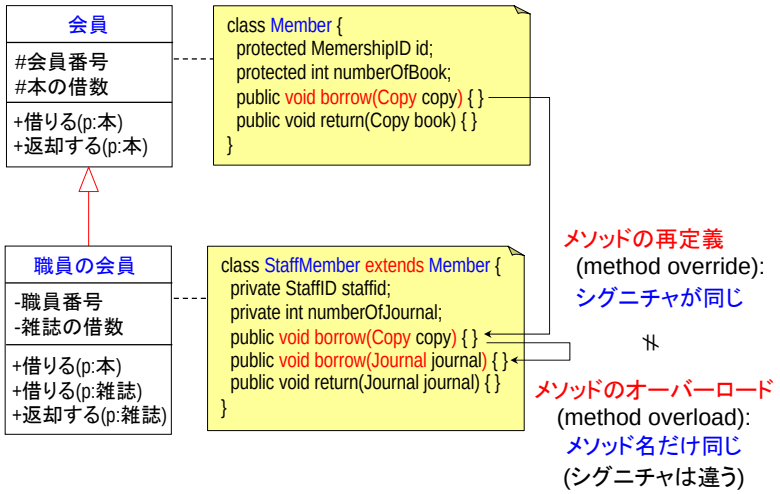
継承における適合

- 適合(conform):
  サブクラス(サブタイプ)のインターフェースに
  スーパークラス(スーパータイプ)のインタフェースの要素がすべて含まれる
  ≅ 置換可能性(substitutability):
  - ✓ スーパークラスのインスタンスはサブクラスのインスタンスに置換え可能
  - ✓ サブクラスはスーパークラスが結んだ契約(contact)を果たす
    → 契約による設計(design by contract)
    - サブクラスの事前条件 ＞ スーパークラスの事前条件 (より緩い)
    - サブクラスの事後条件 ＜ スーパークラスの事後条件 (より厳しい)
      事前条件(pre-condition): 操作実行前に成立する条件
      事後条件(post-condition): 操作実行後に成立する条件

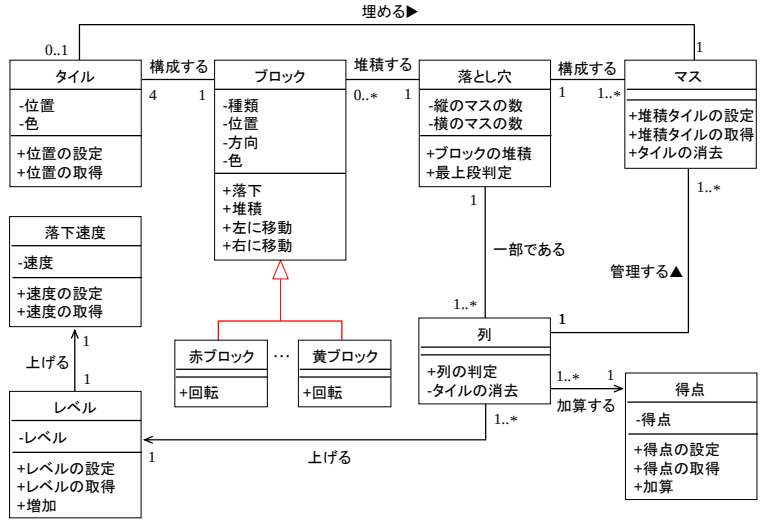
継承(実装例)



メソッドの再定義

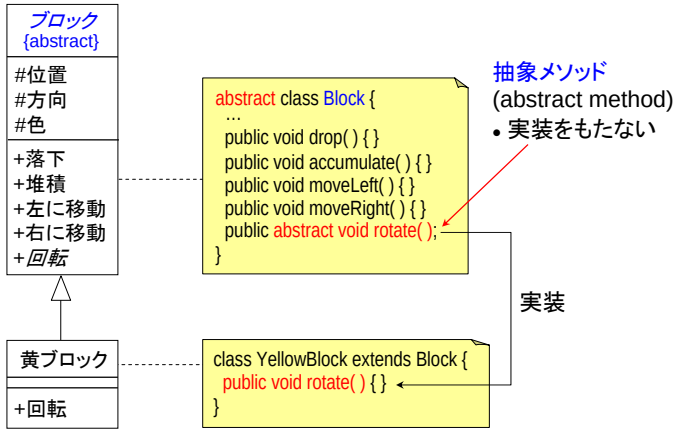


クラス図(tetris-5:継承の追加)

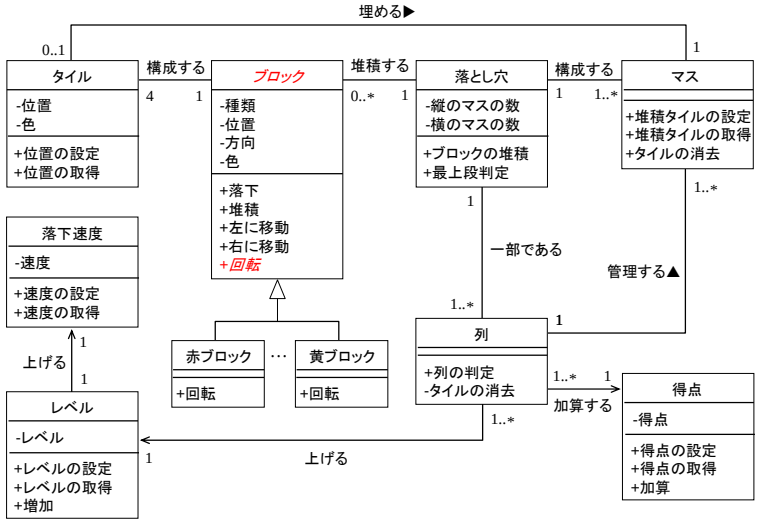


抽象クラス

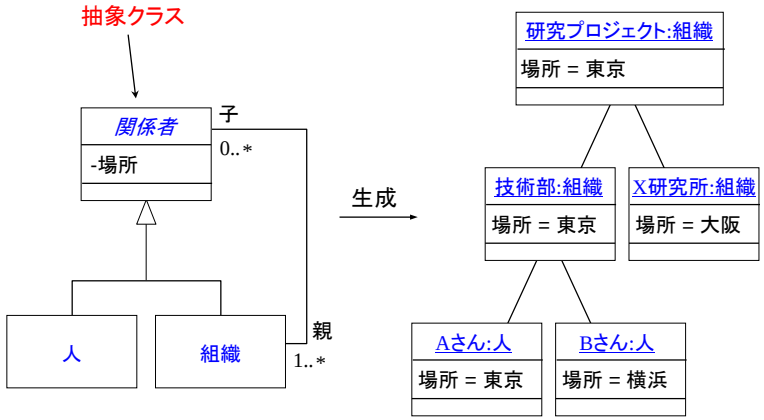
- 抽象クラス(abstract class): インスタンスを生成できないクラス



クラス図(tetris-6:抽象メソッドの追加)

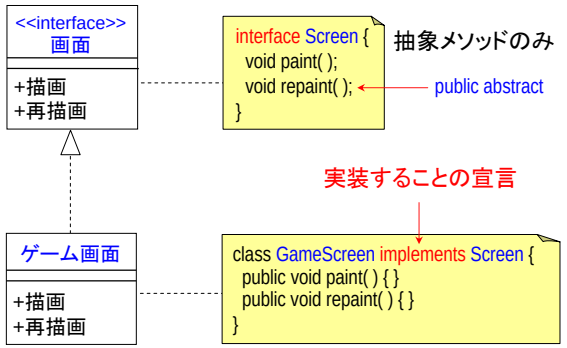


抽象クラスによる階層構造の表現



インタフェース

- インタフェース(interface): 実装を持たないシグニチャのみのクラス
  シグニチャ(signature): 操作の名前 + パラメータ + 型



インタフェース継承と実装継承

- インタフェース継承(interface inheritance, subtyping)
  - ✓ 仕様の観点における継承
  - ✓ 型(タイプ)のみの継承
  - ✓ サブクラスは実装の責任あり

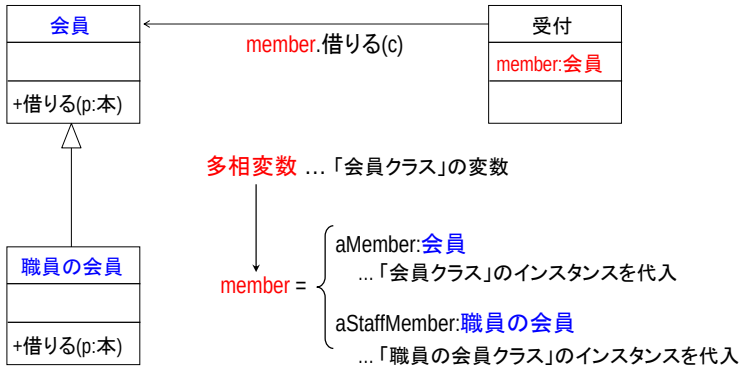
```
class ClassA implements InterfaceB {
}
```

- 実装継承(implementation inheritance, subclassing)
  - ✓ 実装の観点における継承
  - ✓ 型(タイプ)と実装(属性+操作)の継承
  - ✓ サブクラスは実装の責任なし

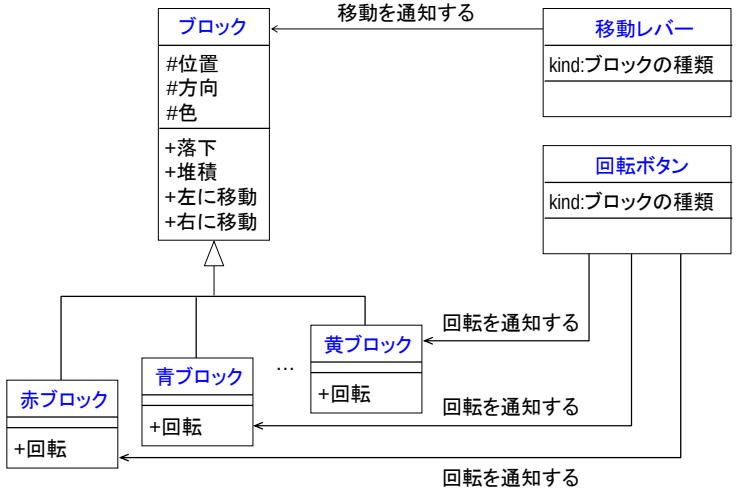
```
class ClassA extends ClassB {
}
```


多相性

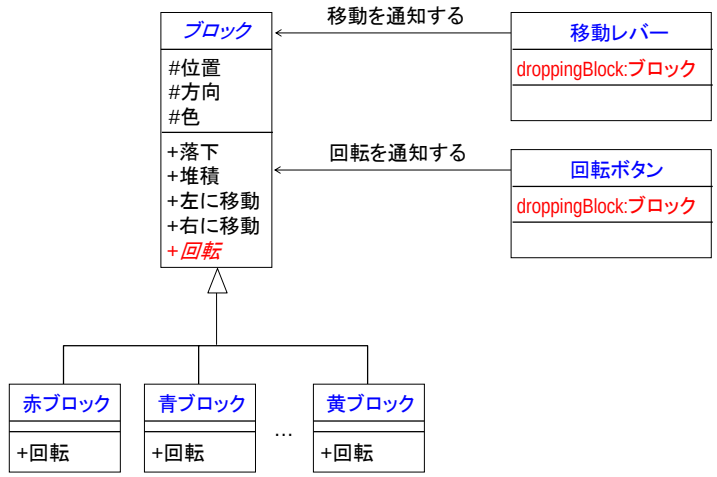
- 多相性/多態性(polymorphism):
  1つのインスタンスが複数の型のうちどれか1つを取り得ること



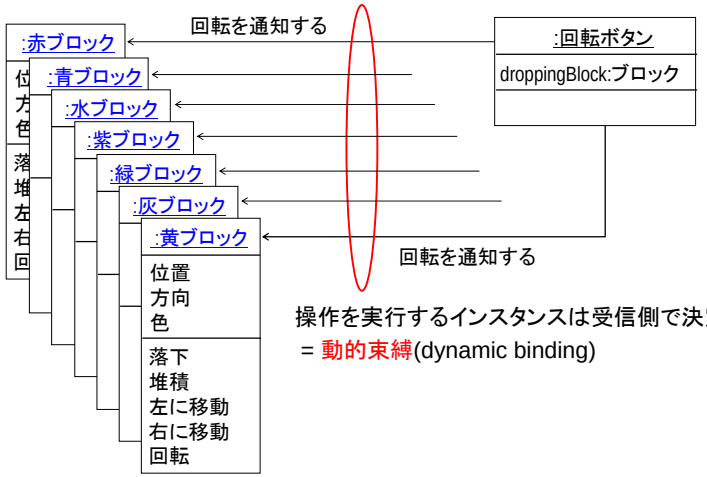
関連(多相性なし)



関連(多相性あり)

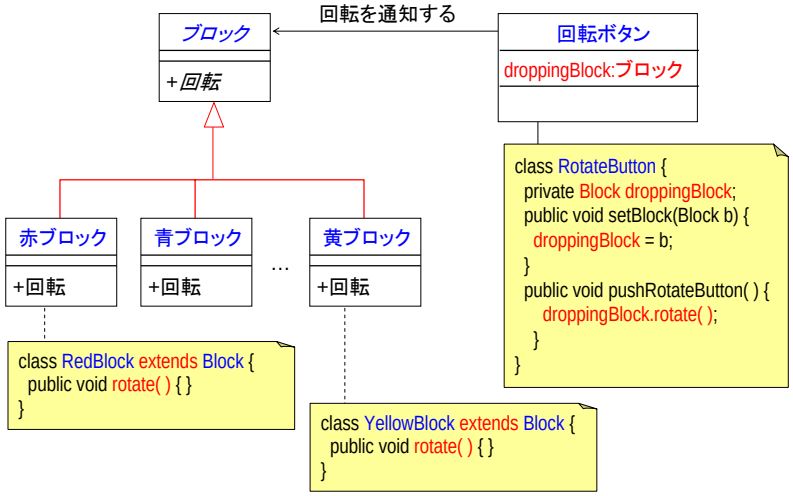


動的束縛

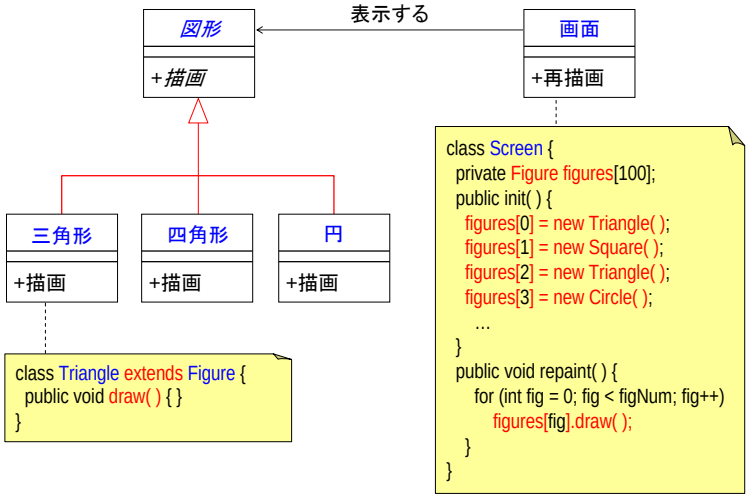


操作を実行するインスタンスは受信側で決定
= 動的束縛(dynamic binding)

多相性(実装例1)

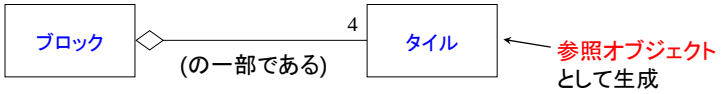


多相性(実装例2)



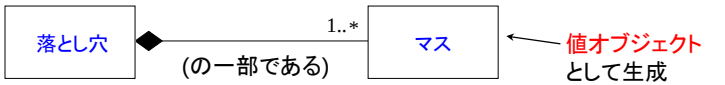
集約と複合

- 集約(aggregation): 全体−部分関係(物理的，情報分散)を表現する関連



  - インスタンスへの参照/ポインタを保有

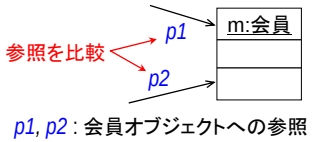
- 複合(composition): 強い所有関係と同一の生存期間を持つ集約



  - インスタンスの値(実体)を保有

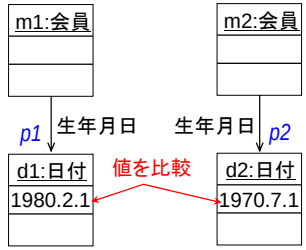
参照オブジェクトと値オブジェクト

- 参照オブジェクト(reference object):
  - ✓ 1つのものを1つのオブジェクトで表現
  - ✓ 参照あるいはポインタによる関連
  - ✓ 識別性(identity)により同一性(identical)を判定
  - ✓ 通常複製(コピー)は禁止



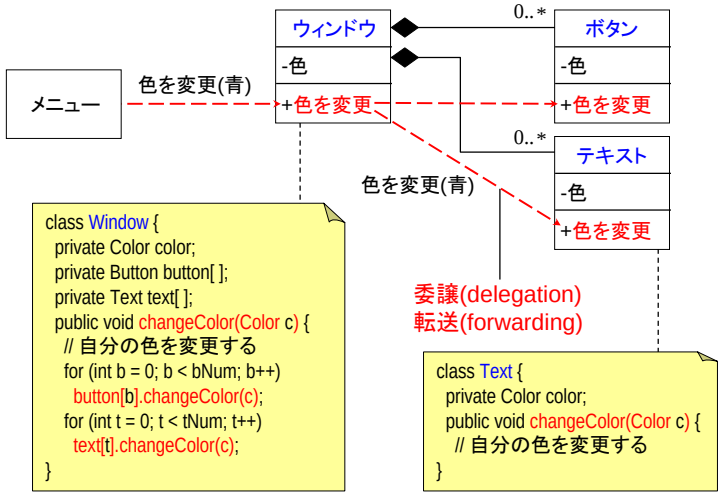
p1, p2 : 会員オブジェクトへの参照

- 値オブジェクト(value object):
  - ✓ 1つ(同一)のものを指す複数のオブジェクトが存在
  - ✓ 属性による値の保有
  - ✓ 値により同値性(equivalent)を判定
  - ✓ 複製(コピー)は頻繁

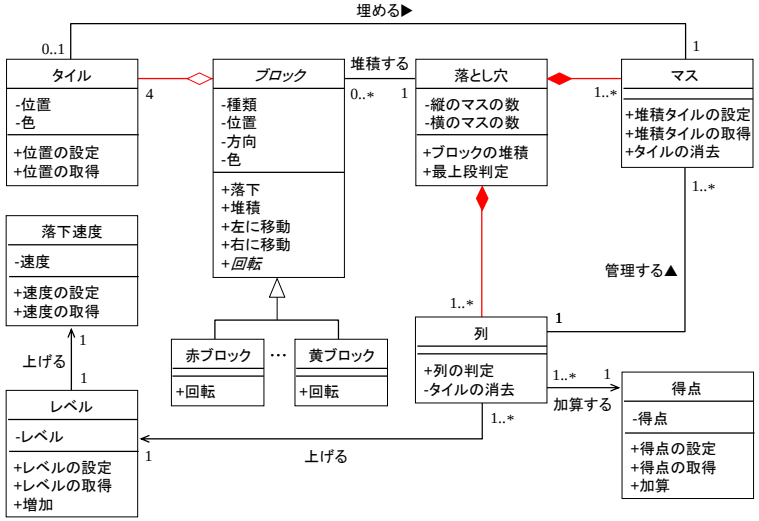


p1, p2 : 日付オブジェクトへの参照

委譲・転送

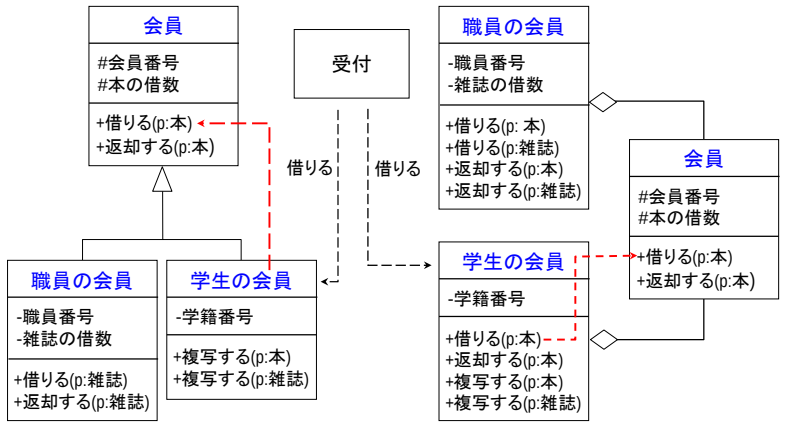


クラス図(tetris-7:集約と複合の追加)

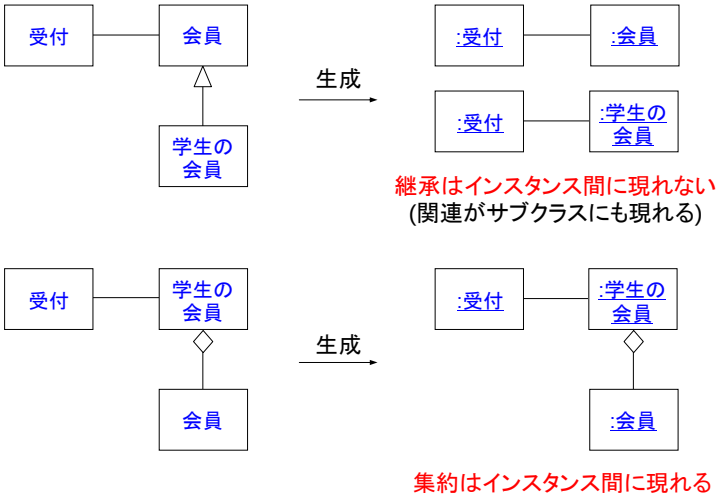


継承と集約

- 継承: 関係は静的(インスタンス生成時)
- 集約: 関係は動的(インスタンス生成後)

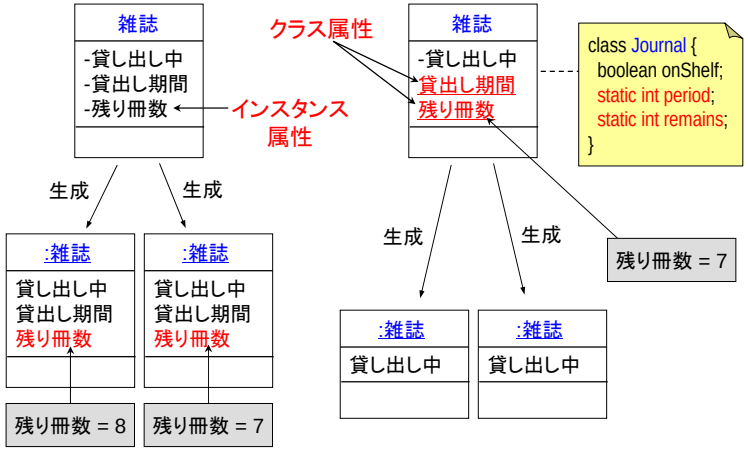


継承と集約(cont'd)



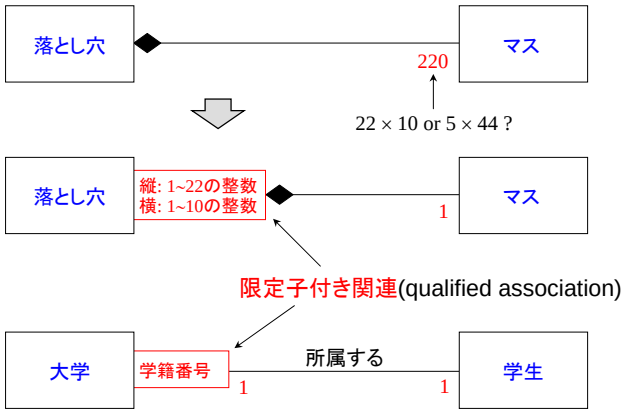
クラス属性

- クラス属性(class attribute): インスタンスではなくクラスに属する属性



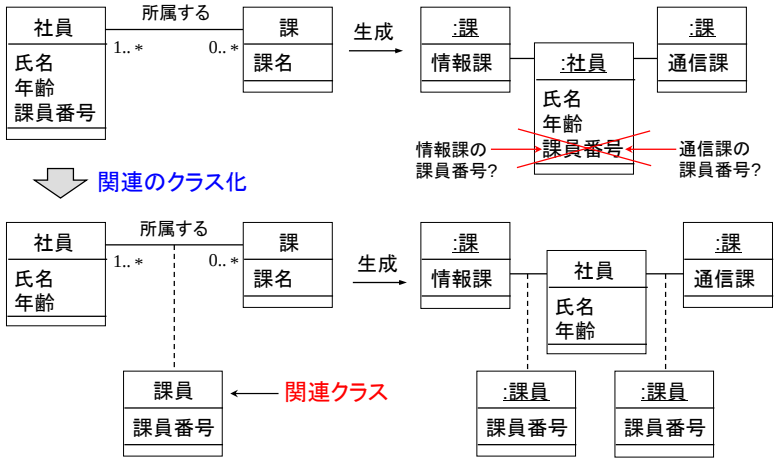
限定子つき関連

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



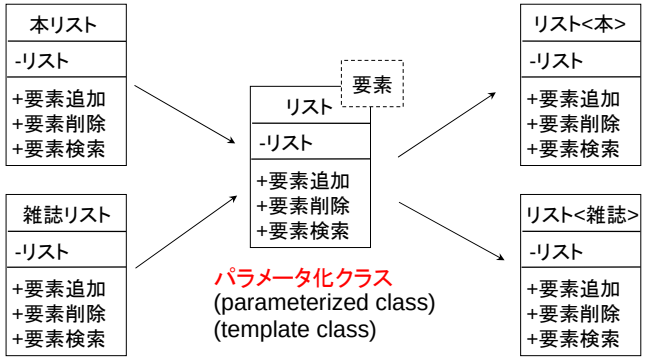
関連クラス

- 関連クラス(association class): 属性や操作を持つ関連



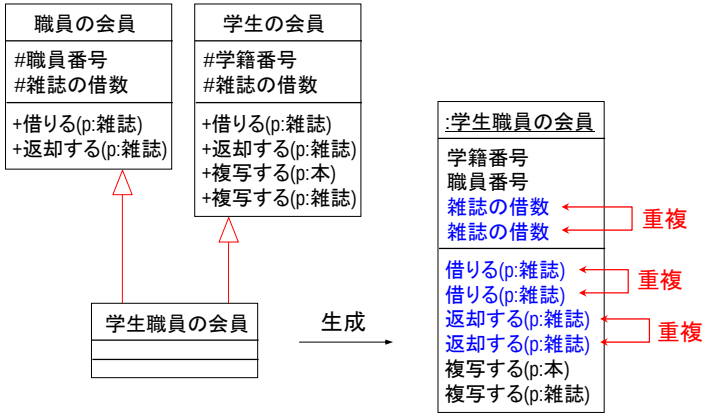
総称とパラメータ化クラス

- 継承: 異なる型で同じ操作．ただし，アルゴリズムは異なる
  e.g.,「職員会員」と「学生会員」の「借りる」操作は違う
- 総称(genericity): 異なる型で同じ操作．アルゴリズムも同じ．
  e.g.,「本リスト」と「雑誌リスト」の操作はすべて同じ

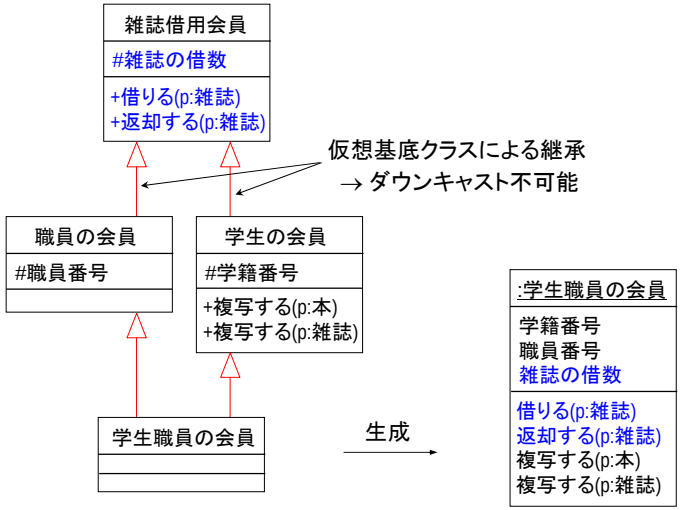


多重継承

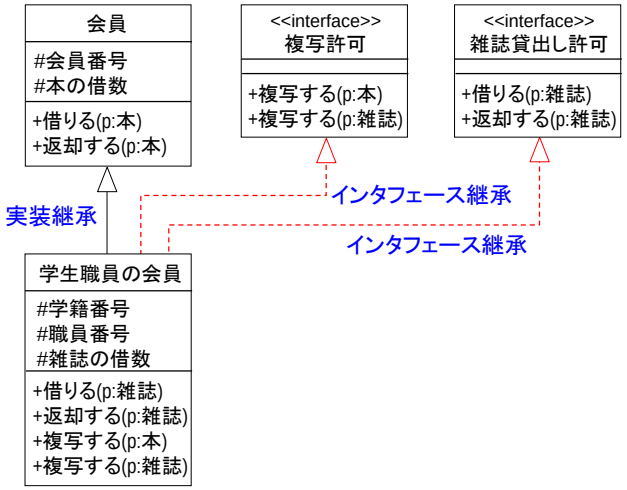
- 多重継承(multiple inheritance): 親クラスを複数持つ継承



多重継承(C++)



多重継承(Java)



第8～11回
オブジェクト指向分析(OOA)
動的モデリング

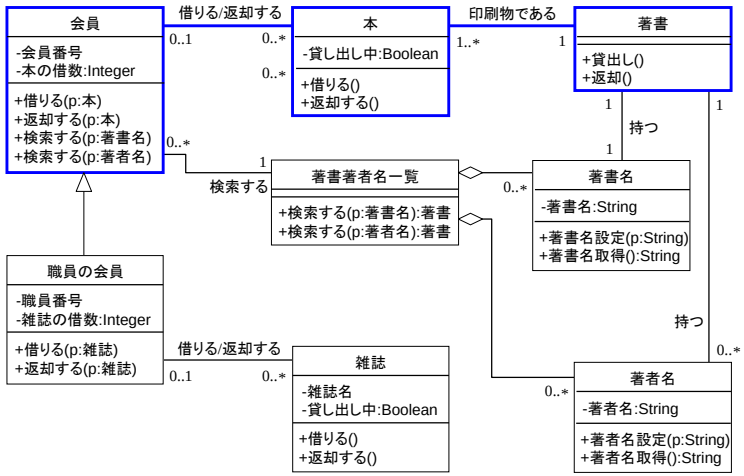
相互作用図

- 相互作用図(interaction diagram)
  - ✓関連するオブジェクト群の振る舞い(behavior)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

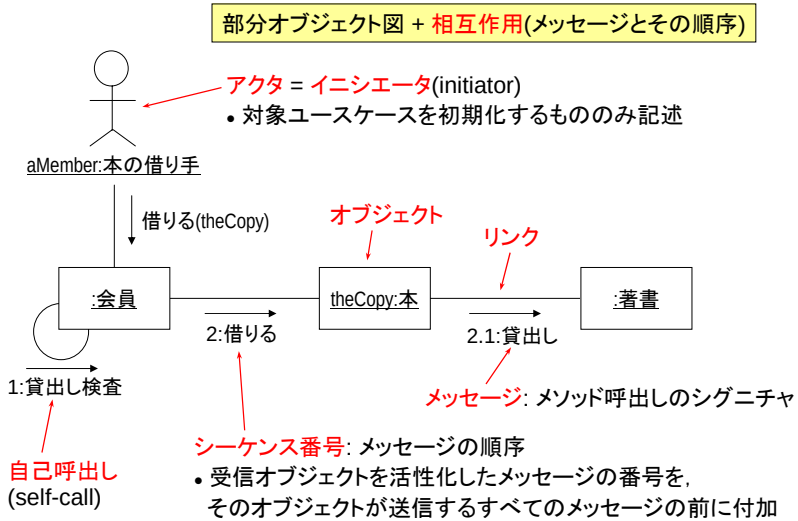
(1) コミュニケーション図(communication diagram) (UML 2.0)
コラボレーション図(collaboration diagram) (UML.1.x)
相互作用するオブジェクト間の関連を表現したもの
レイアウト重視

(2) シーケンス図(sequence diagram)
オブジェクト間のメッセージ送受信の時系列を表現したもの
メッセージの流れが直感的

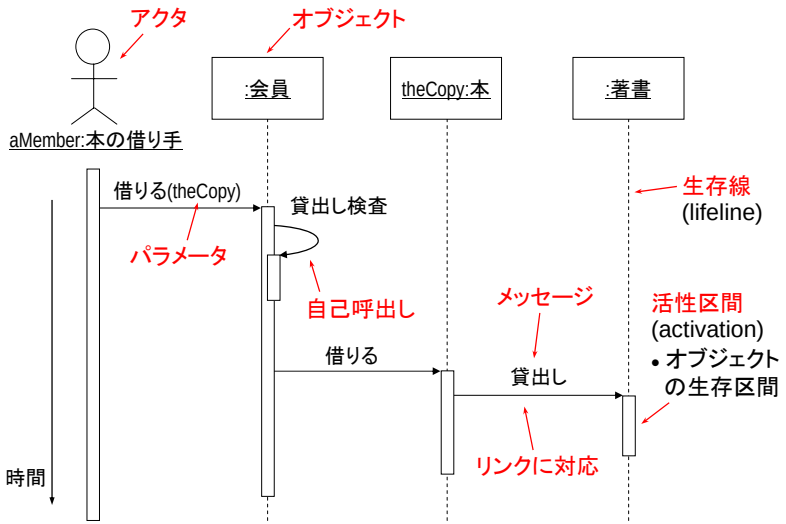
クラス図(図書館システム)



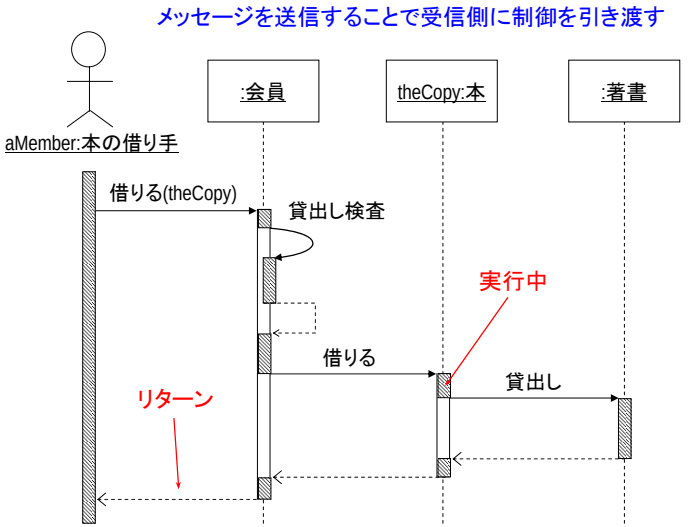
コミュニケーション図



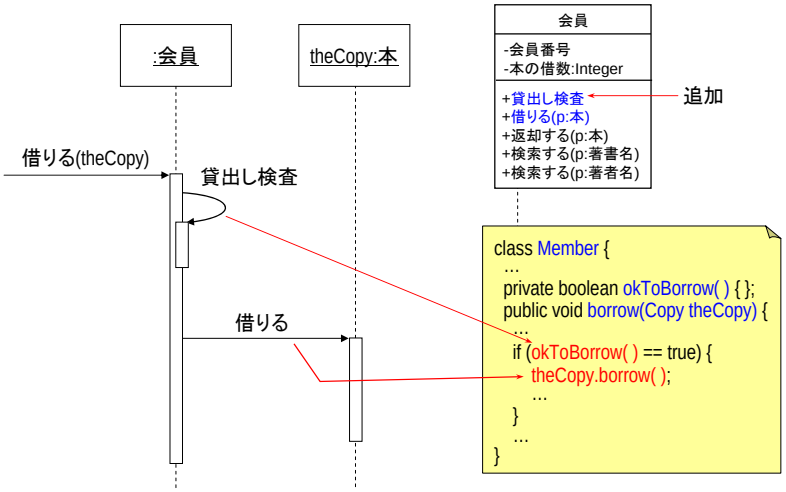
シーケンス図



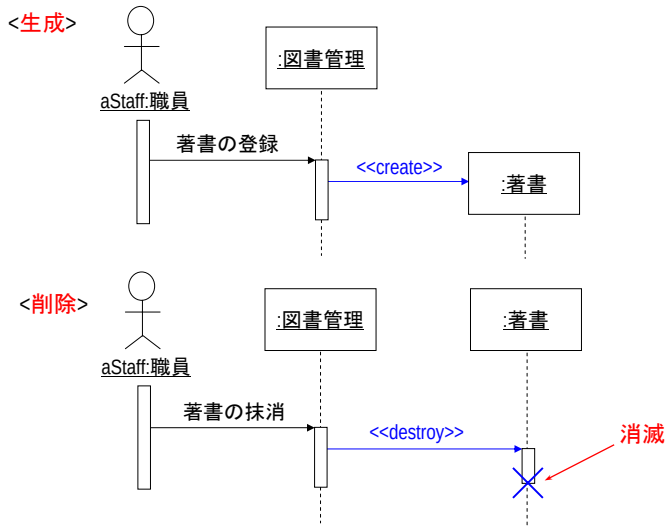
シーケンス図(cont'd)



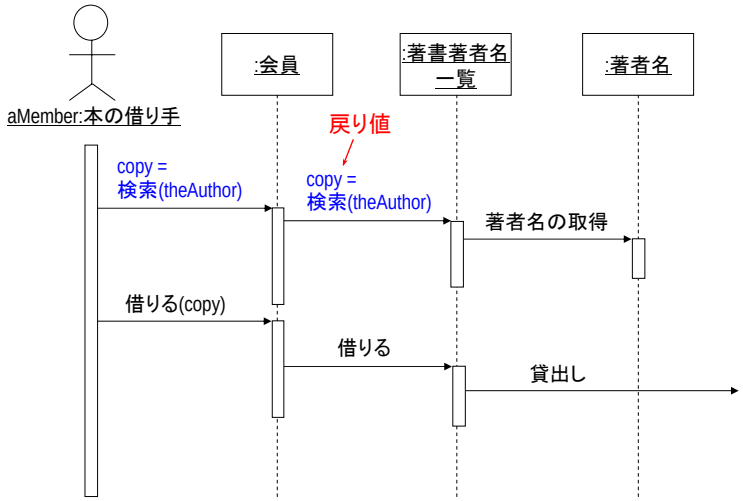
シーケンス図(実装例)



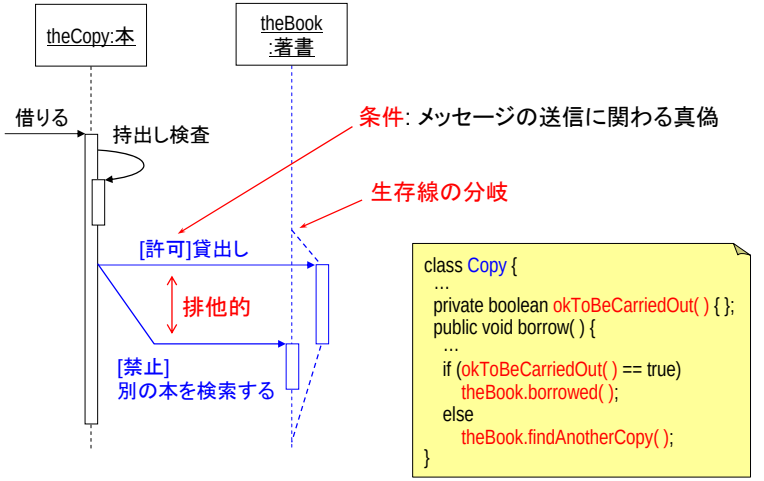
オブジェクトの生成と削除



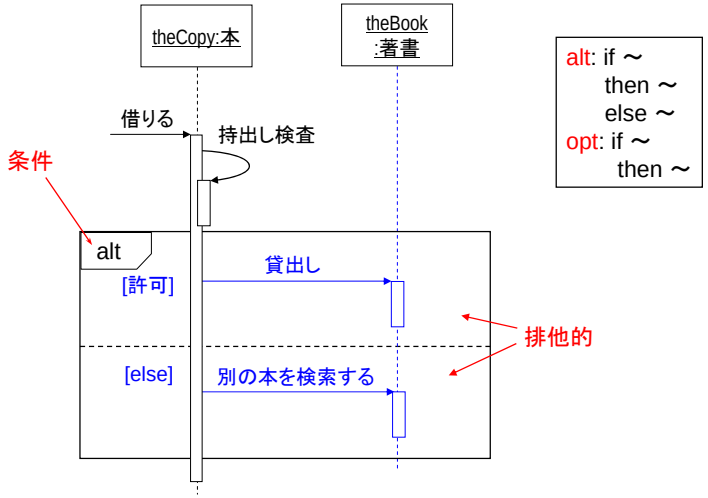
戻り値



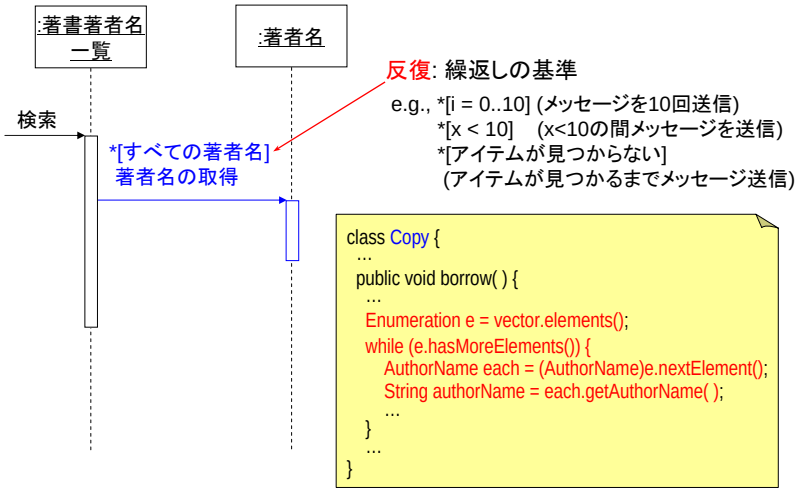
条件(UML 1.x)



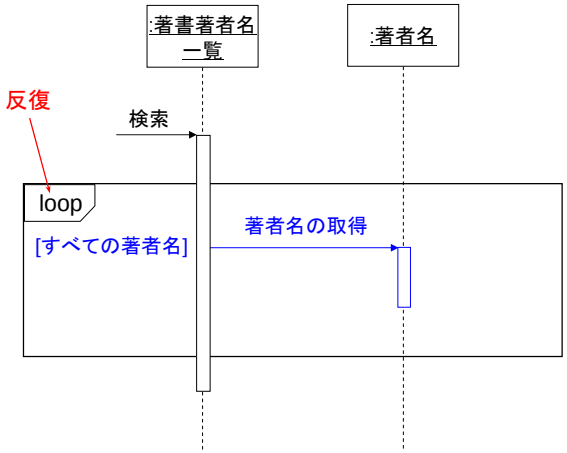
条件(UML 2.0)



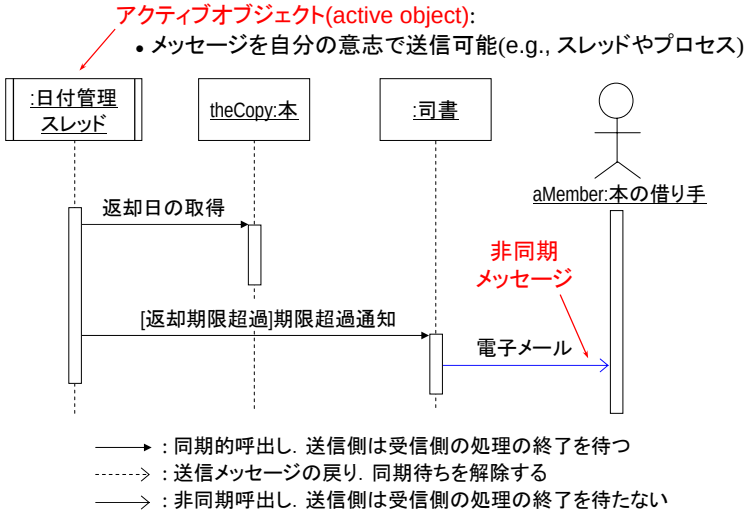
反復 (UML 1.x)



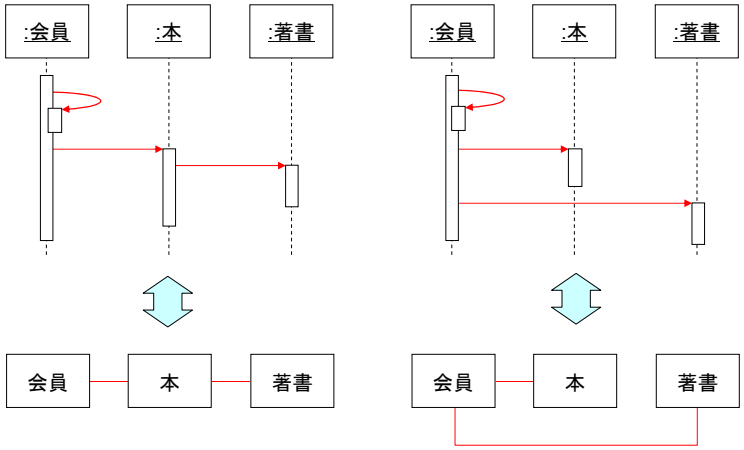
反復 (UML 2.0)



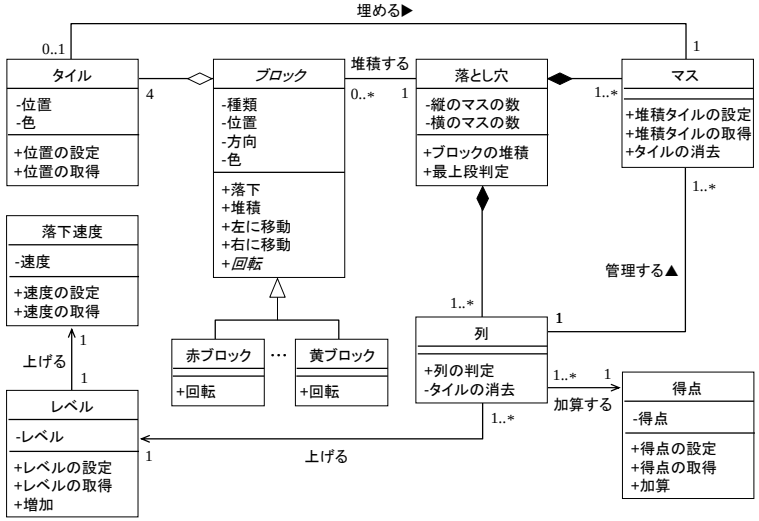
並列性



相互作用図とクラス図



クラス図(再)

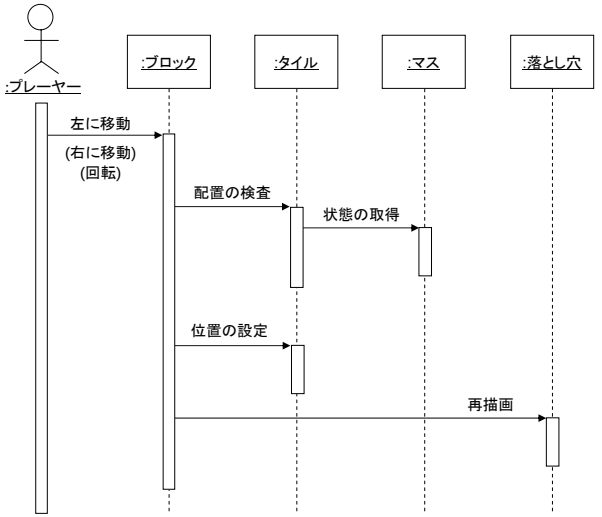


要求仕様(Revisited)

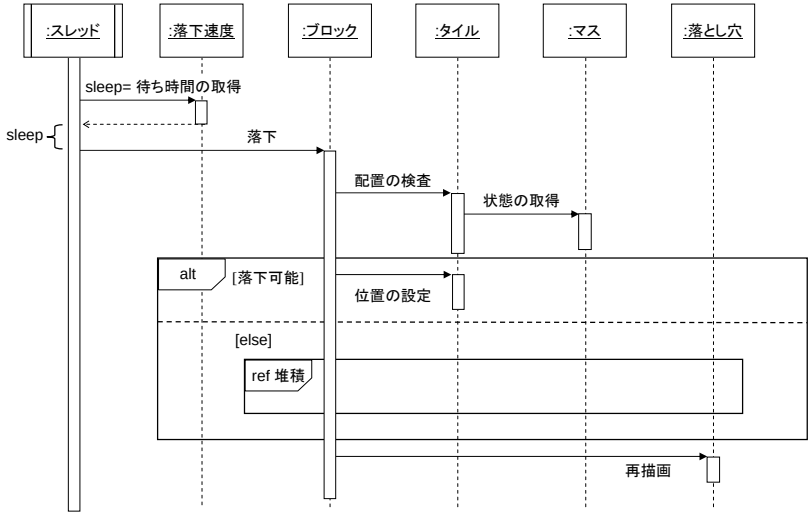
テトリスゲーム

- 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．(2)
- 落とし穴は縦22個，横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．
- ゲームのプレーヤーは，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そのブロックを回転させることもできる．(1)
- 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．(2)
- 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．(3)
- 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．(2)
- 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．(2)
- 次に落ちてくるブロックは落とし穴の横に表示される．
- レベルと得点は落とし穴の横に表示される．

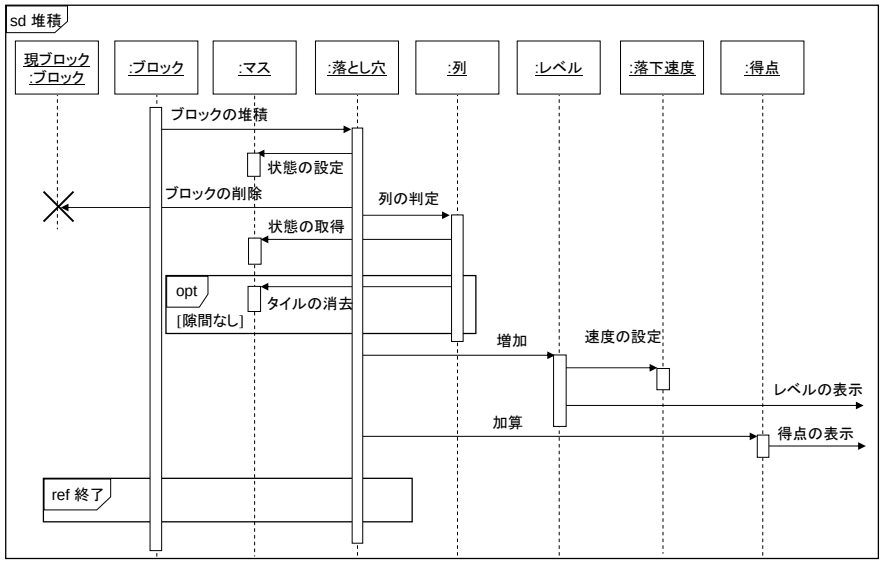
シーケンス図(テトリスゲーム)(1)



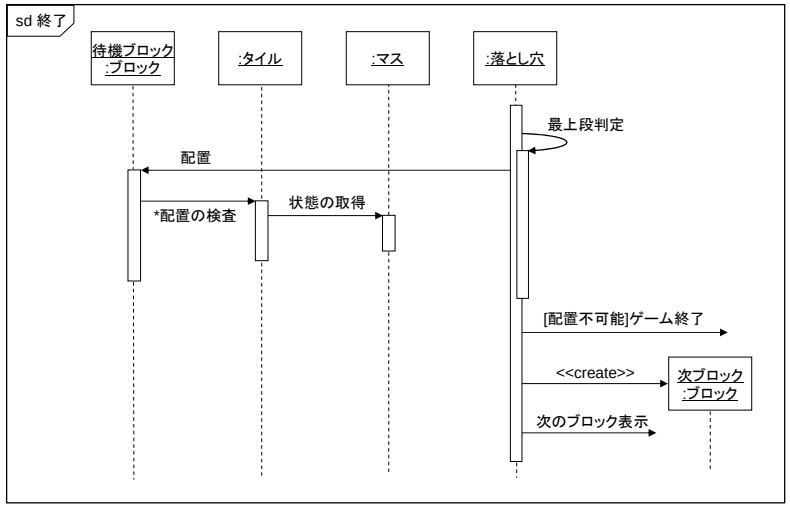
シーケンス図(テトリスゲーム)(2)



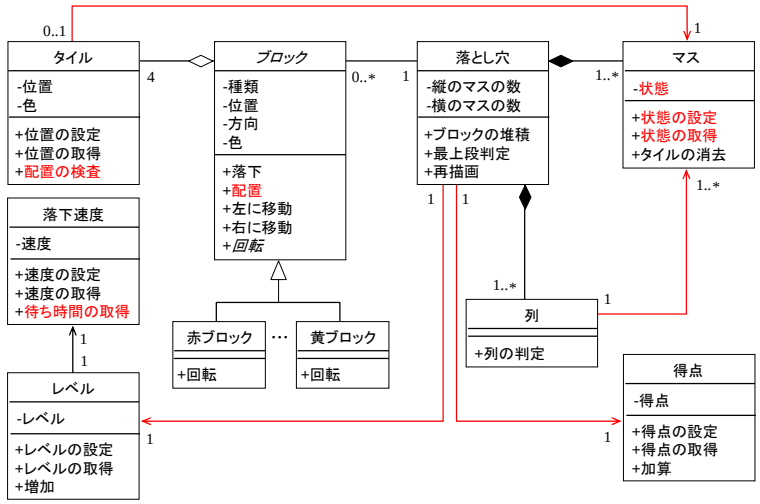
シーケンス図(テトリスゲーム)(2 cont'd)



シーケンス図(テトリスゲーム)(3)



クラス図(tetris-8:相互作用図の反映)

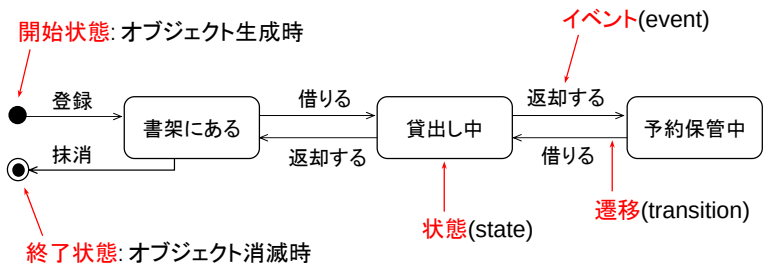


状態機械図

- 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変化を表現
  - ✓ 外部からのイベントに対するオブジェクトの応答を表現
  - ✓ 通常，単一のクラス(オブジェクト)に対して記述
  - 状態(state):
    オブジェクトの持つ属性の値や他オブジェクトとのリンクで特定

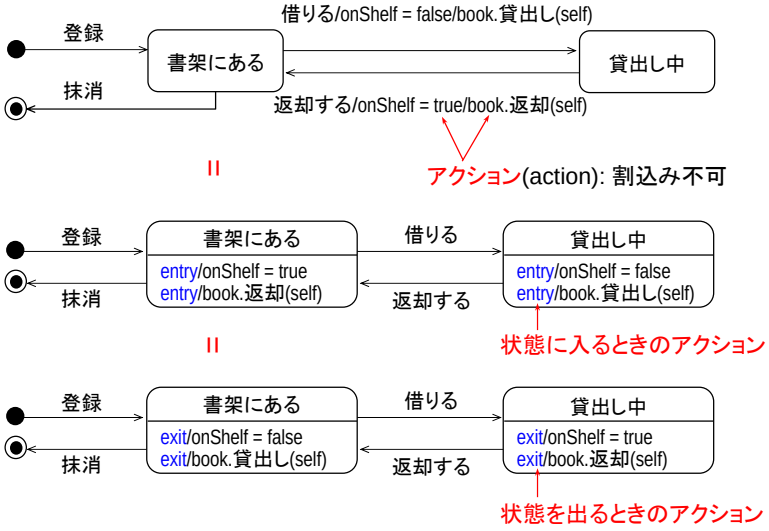
状態機械図(cont'd)

クラス「本」

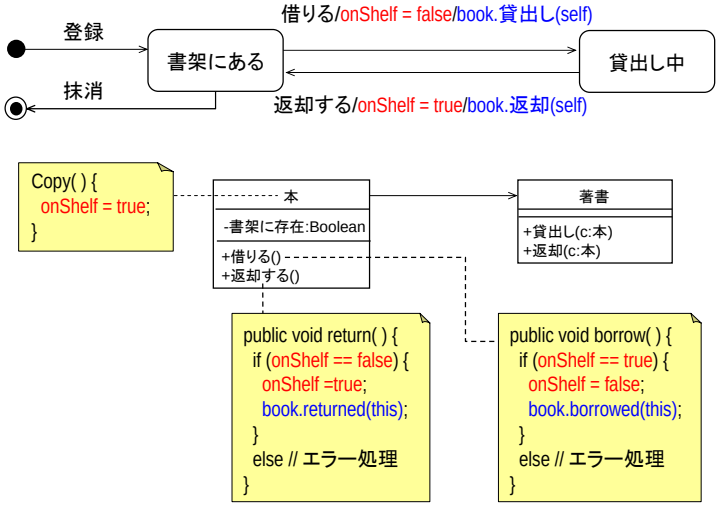


- 必ず1つの状態に属する(複合状態を除く)
- イベントは一瞬
- 遷移時間は無視
- イベントに対する応答は現在の状態によって変化

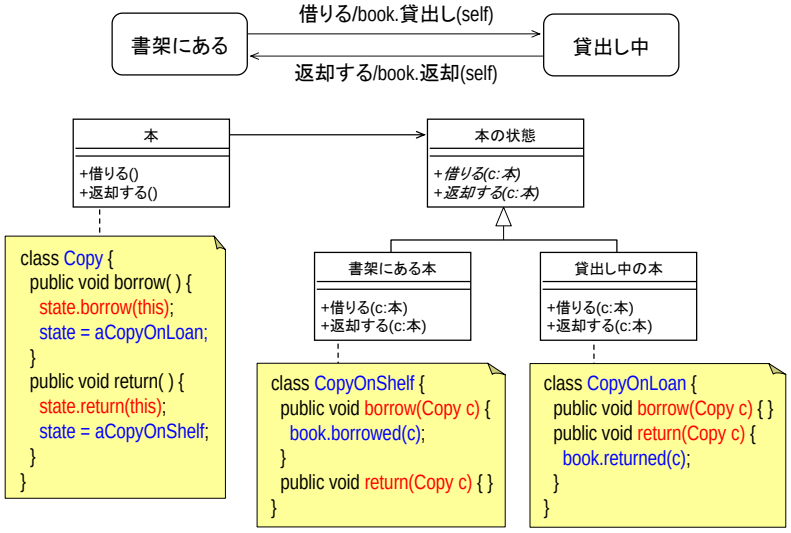
アクション



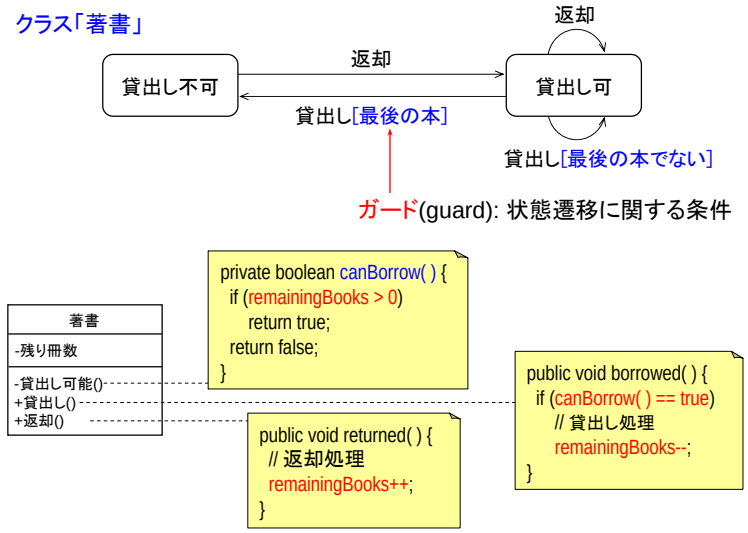
状態機械図(実装例1)



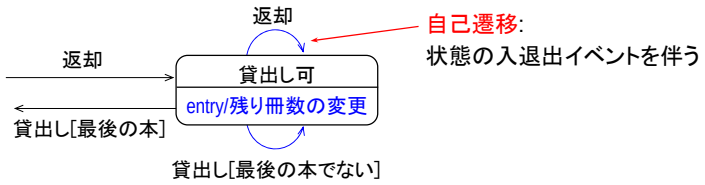
状態機械図(実装例2)



ガード

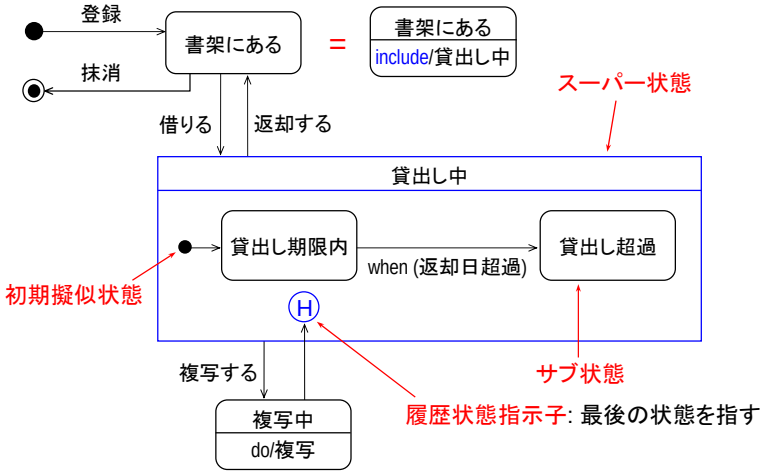


自己遷移と内部イベント

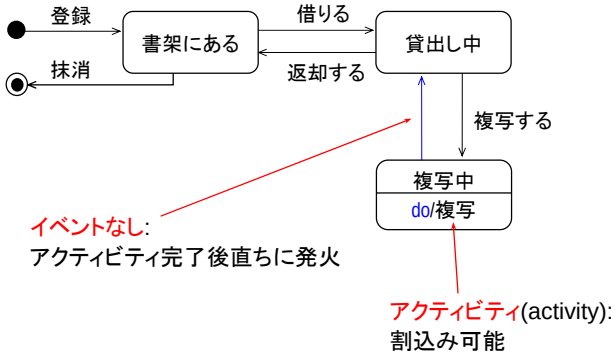


複合状態(入れ子)

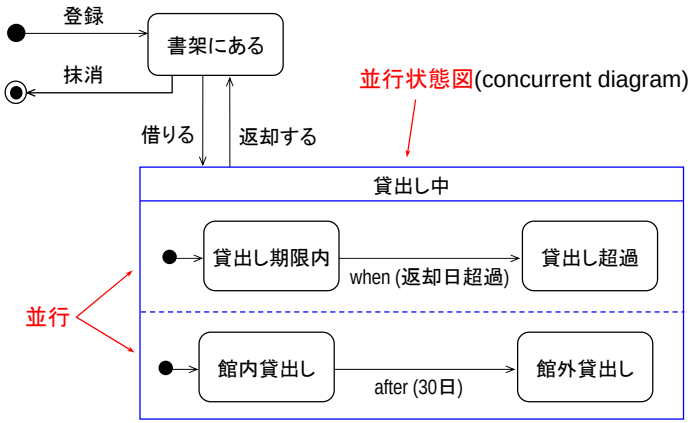
特定の状態の内部状態を表現 → 複合状態(compound state)



アクティビティ

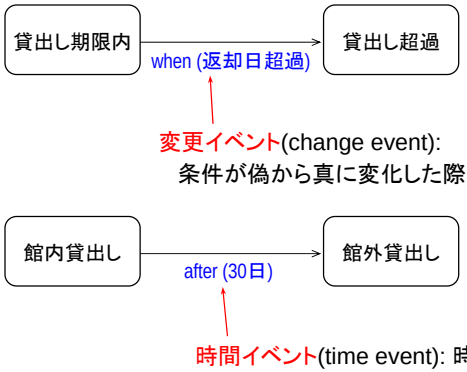


複合状態(並行)



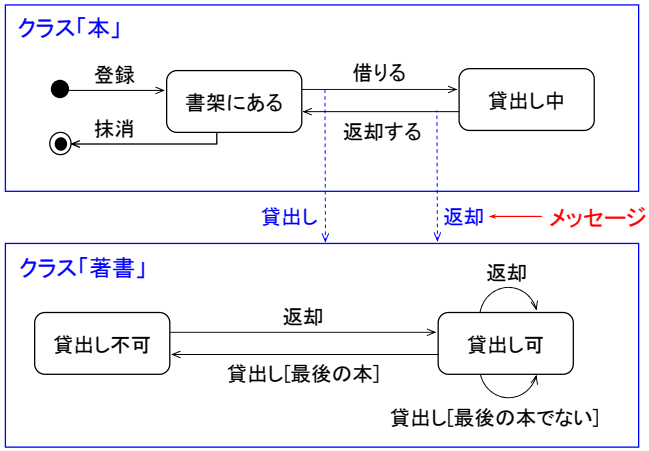
複数の状態に同時に属する(e.g.,「貸出し期限内」and「館外貸出し」)

イベント

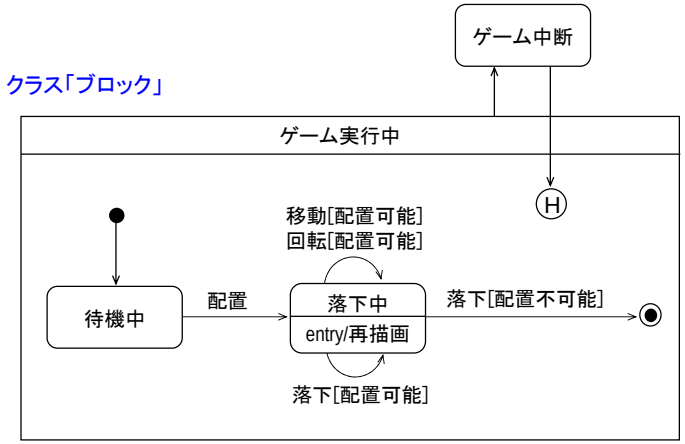


- 呼出しイベント(call event): 操作の実行を要求
- シグナルイベント(signal event): シグナル受信時(<<signal>>を使用)

状態図間のメッセージ送受信



状態機械図(テトリスゲーム)



第12～13回
オブジェクト指向設計(OOD)・実装(OOP)

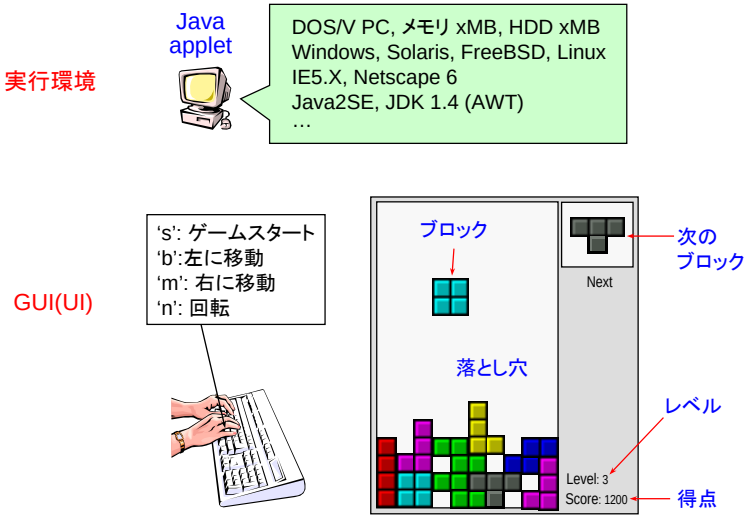
オブジェクト指向設計・実装

- 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
  - (1) システム設計
    - ✓ システムレベルの構成と振る舞いを設計
  - (2) 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

- システム設計
  - ✓ アーキテクチャの全体構造の設計(C/S，3-tier，DB)
  - ✓ ハードウェア環境(制約や条件)の決定
  - ✓ ソフトウェア環境(実装言語，ライブラリ，フレームワーク)の決定
  - ✓ 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
  - ✓ DB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)



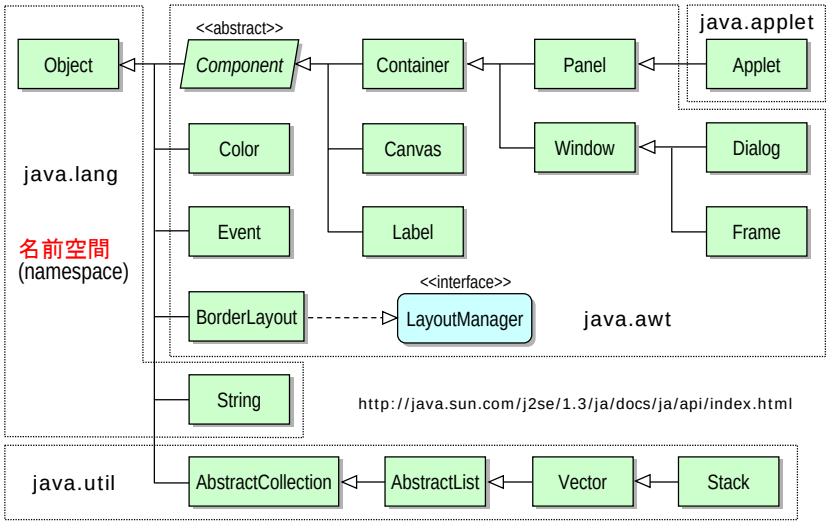
ライブラリ

- ライブラリ(library)
  - ✓ 再利用可能な部品(reusable components)を集めたもの
  - ✓ ソフトウェアを構成する汎用的なクラスのリポジトリ(repository)
- オブジェクト指向ライブラリ(OO library)

  e.g., JFC(Java), MFC(Microsoft), STLライブラリ(C++)
  - ✓ 階層的に整理されたクラス(属性+操作)の集合
  - ✓ クラス継承とオブジェクト合成(委譲や包含)によるコードの再利用
  - ✓ インタフェースを含む → インタフェースの再利用

  ↔ 手続き指向ライブラリ
  - ✓ 手続き(関数)の集まり
  - ✓ 階層なし

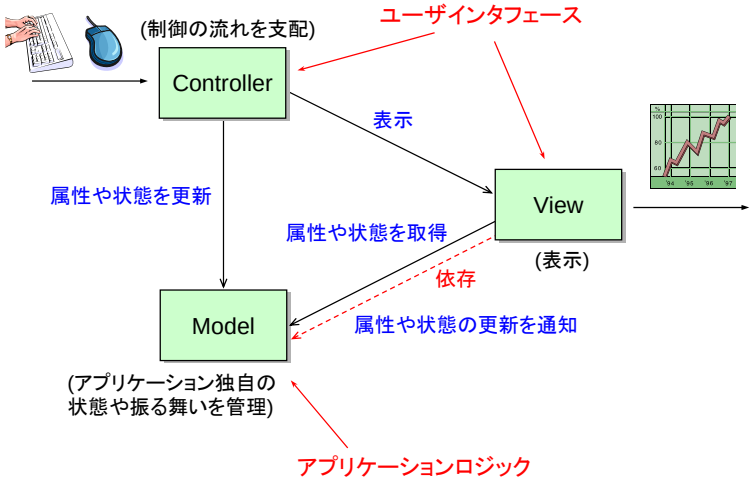
OOライブラリの例(Java)



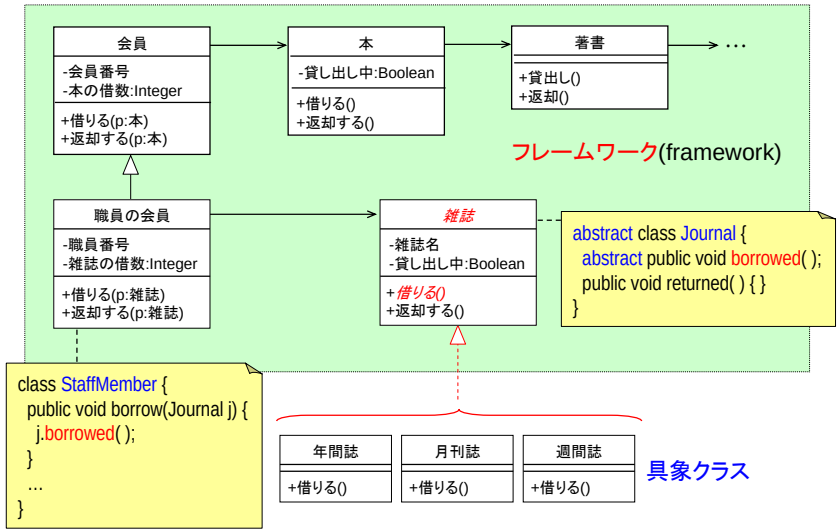
フレームワーク

- (汎用)フレームワーク(framework)
  e.g., MVC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
    cf. アプリケーションフレームワーク: 特定のアプリケーションドメインを対象
  - ✓ 再利用可能な高いレベルの設計
- オブジェクト指向フレームワーク(OOFW)
  e.g., Java AWT, EJB, Java Swing, MFC, .NET, IBM San Francisco, MacApp, Andrew, InterViews, ET++
  - ✓ 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
  - ✓ アルゴリズム(テンプレートメソッド)とその内容(フックメソッド)を分離
  - ✓ 制御の逆転(inversion of control): FWがアーキテクチャと制御の流れを内包
    = メインプログラムの再利用，呼び出される部品を作成
    ↔ 手続き型プログラミングパラダイム = 部品(手続き，関数)を再利用

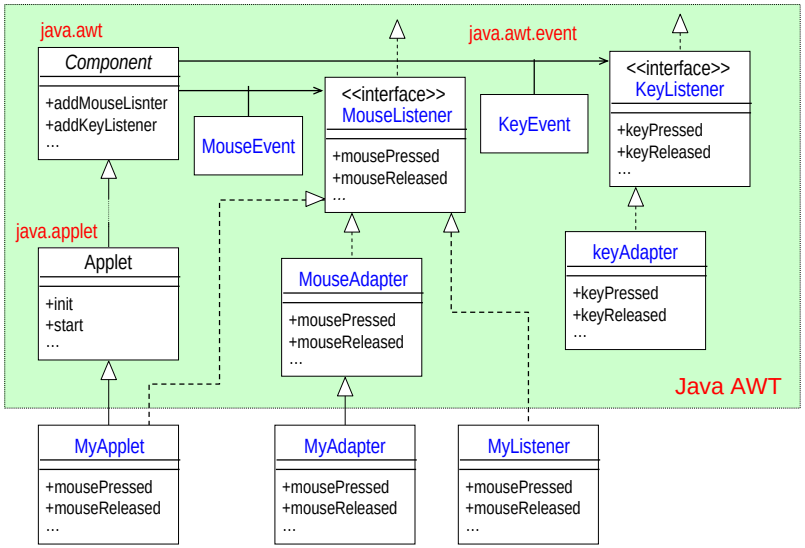
MVCフレームワーク(MVCアーキテクチャ)



OOフレームワーク

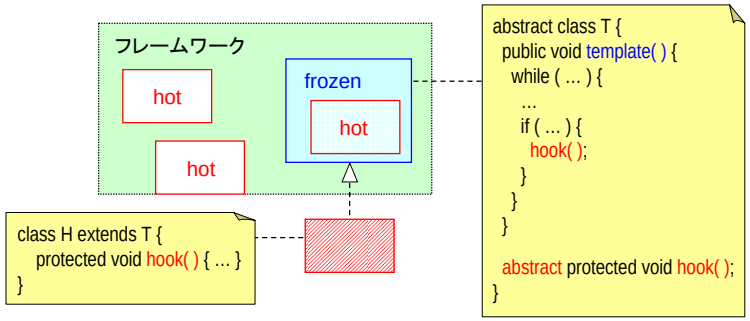


フレームワーク(JFC)



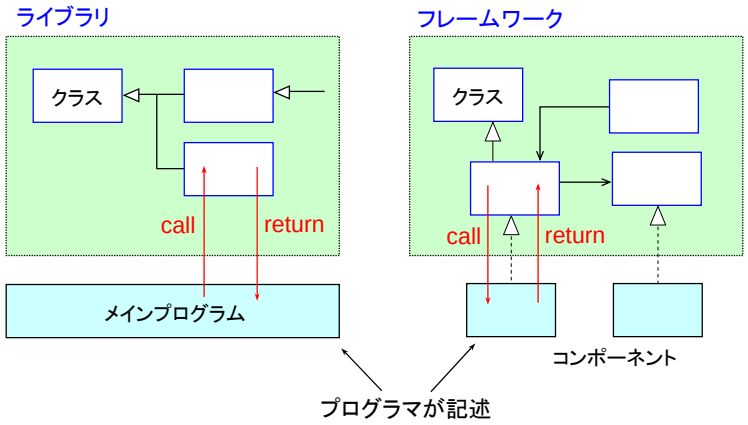
フローズンスポットとホットスポット

- 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



制御の逆転

- 部品(コンポーネント)の再利用
- メインプログラムの再利用



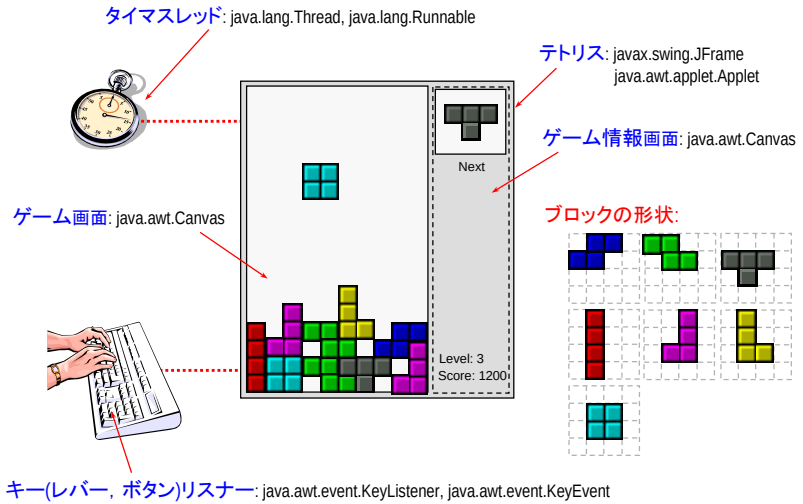
詳細設計

- 詳細設計
  - コンピュータ領域のクラスの抽出
  - 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，ロール名，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)

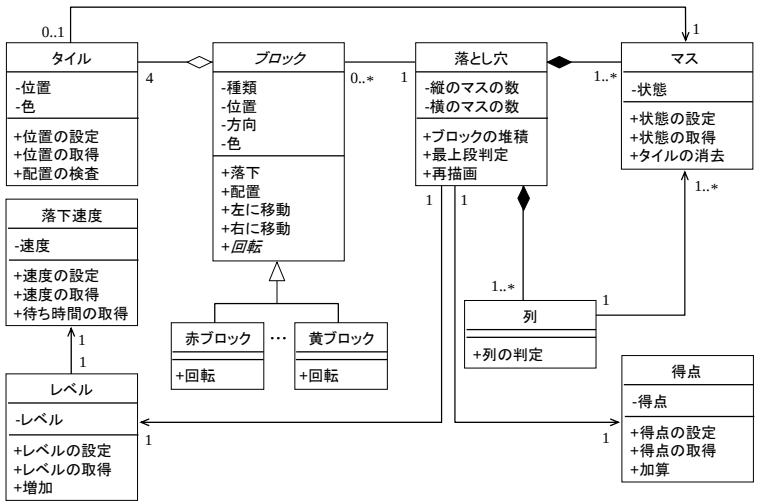
詳細設計

- システムそのものを指すクラス
- 実体クラス: システムの本質を表現するクラス(分析時に抽出)
- インタフェースクラス: システムとオペレータとの接点に存在するクラス
  - e.g, キーボード
    - プリンタ
    - 画面
    - マウス
- 制御クラス: システムを実現する上で必要となるクラス
  - e.g., イベント管理
    - 画面管理
    - プリンタ管理
    - データベース管理
    - ファイル管理
    - スレッド/プロセス管理

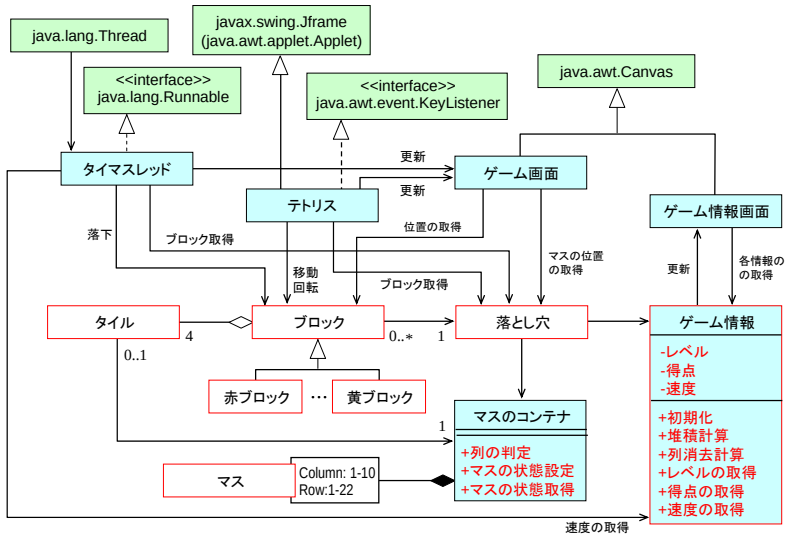
詳細設計(テトリスゲーム)



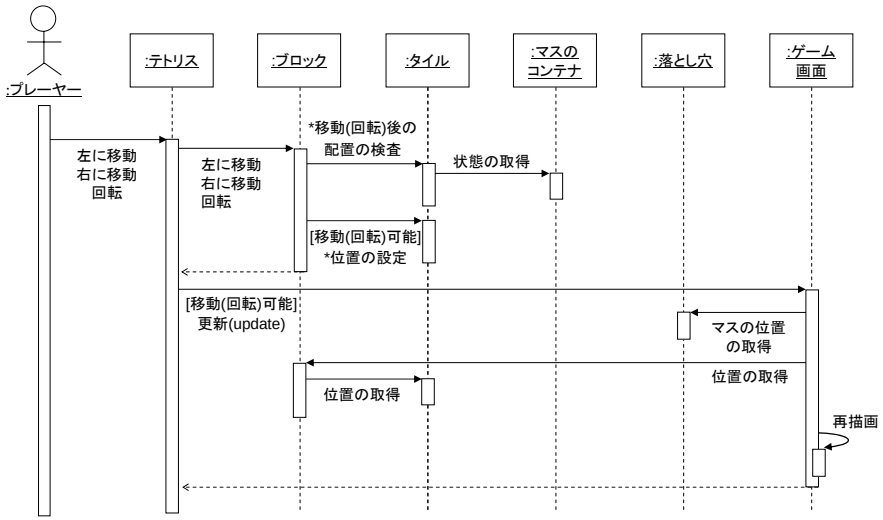
クラス図(テトリスゲーム: Revisited)



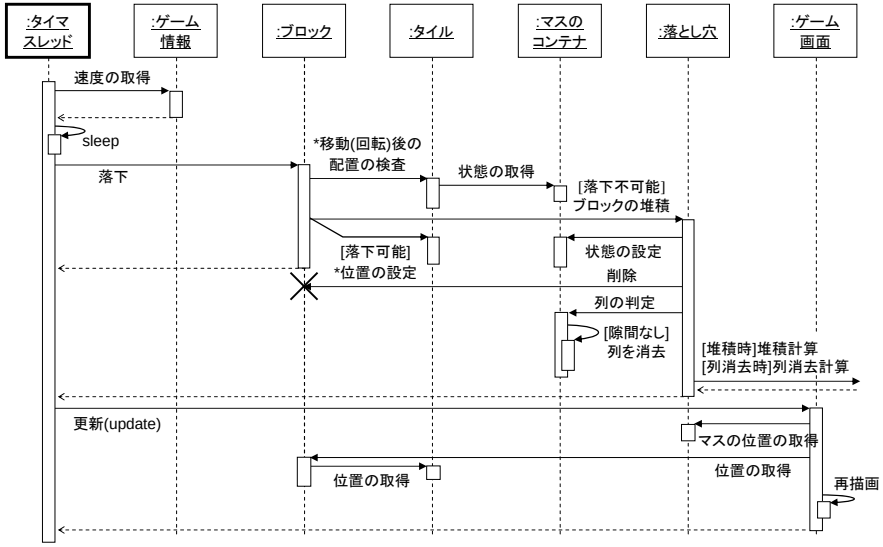
クラス図(テトリスゲーム: 詳細設計)



シーケンス図I(テトリスゲーム：詳細設計)



シーケンス図II(テトリスゲーム：詳細設計)



シーケンス図III(テトリスゲーム：詳細設計)