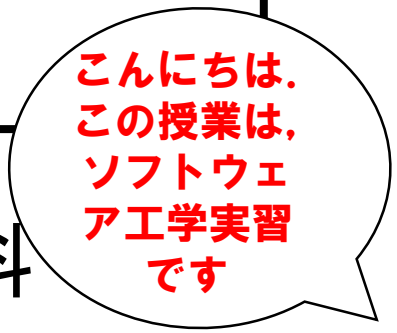
ソフトウェア工学実習
Software Engineering Practice
(第08回)

SEP08-001 UML[1](構造/静的モデリング)



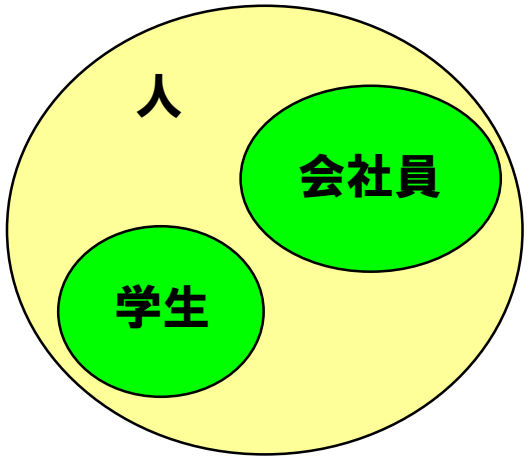
慶應義塾大学・理工学部・管理工学科
飯島 正
iijima@ae.keio.ac.jp

継承（復習）とクラス図

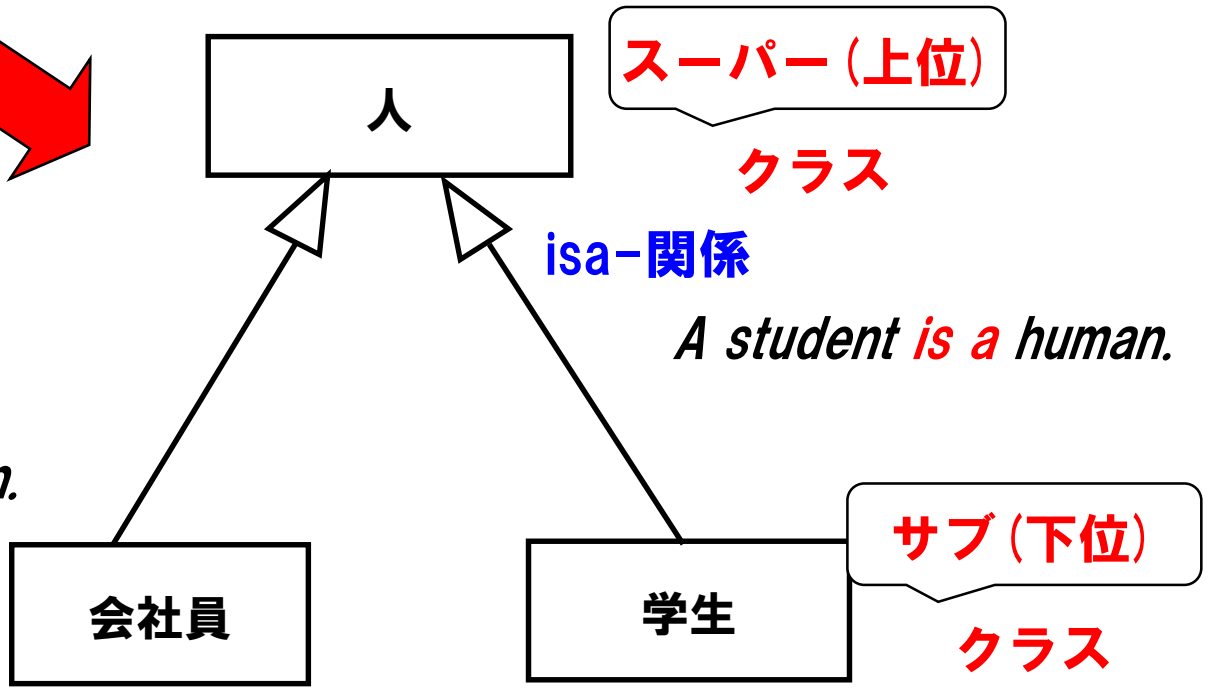
…

まずは，概念編です．
実習で，具体例を通して理解しましょう．

クラス階層（概念階層）

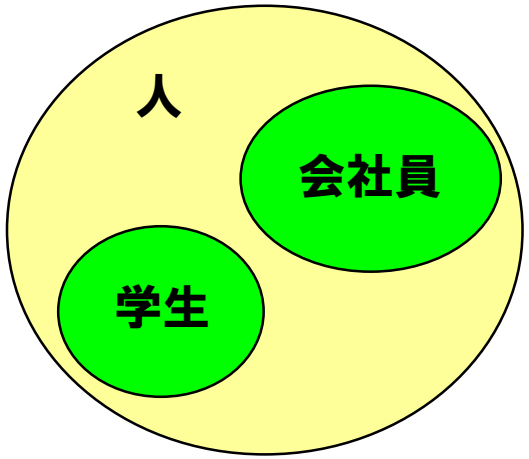


概念の包含関係を
木構造で表現する



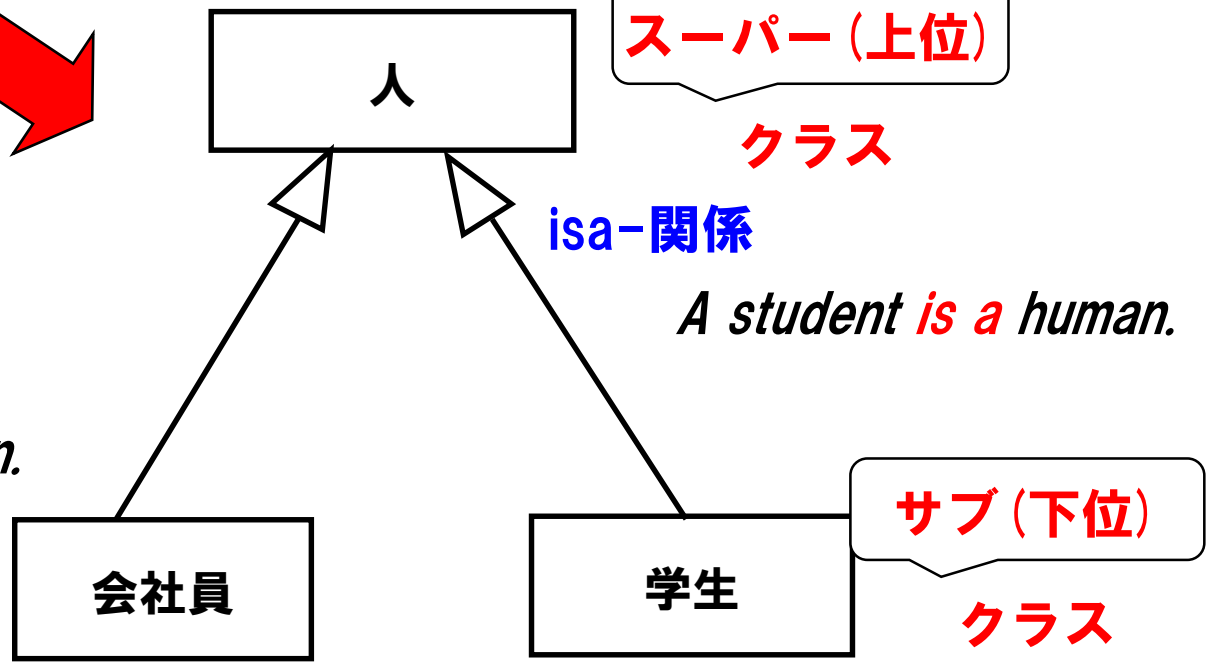
iijima@ae.keio.ac.jp

クラス階層（概念階層）



概念の包含関係を
木構造で表現する

※「会社員」でありながら「学生」でもあるという，
ケース（たとえば，社会人博士課程学生などもいます）が
ここでは，そういった共通集合はないものとしましょう

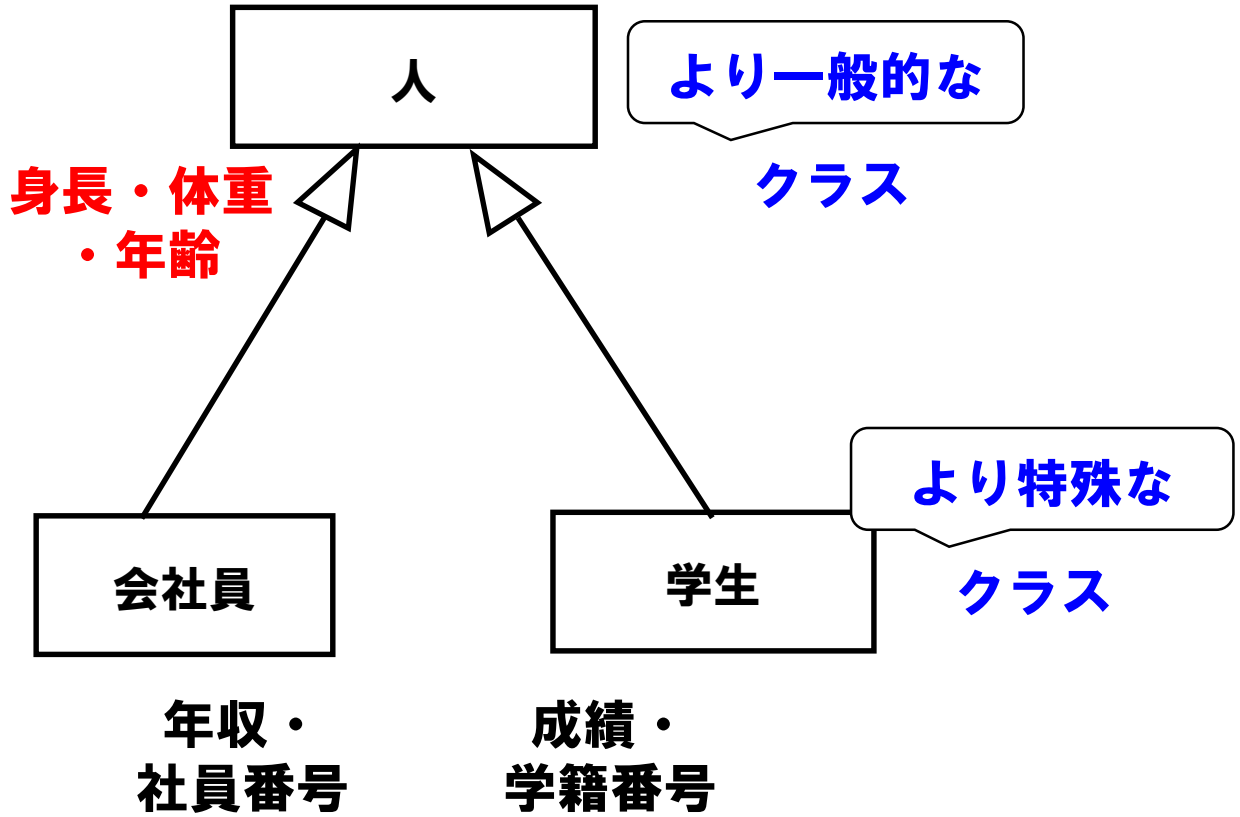


※厳密にいえば，「会社員」や「学生」は．クラスというよりも
ロール（役割）に相当する概念といえるが，わかりやすいので，この例で説明します．

クラス間の共通要素（属性，メソッド）

**上下関係にあるクラス同士には共通の要素がある．
通常，より一般的なもの（上位クラス）が要素の追加によって
特殊化される（下位クラスが作られる）**

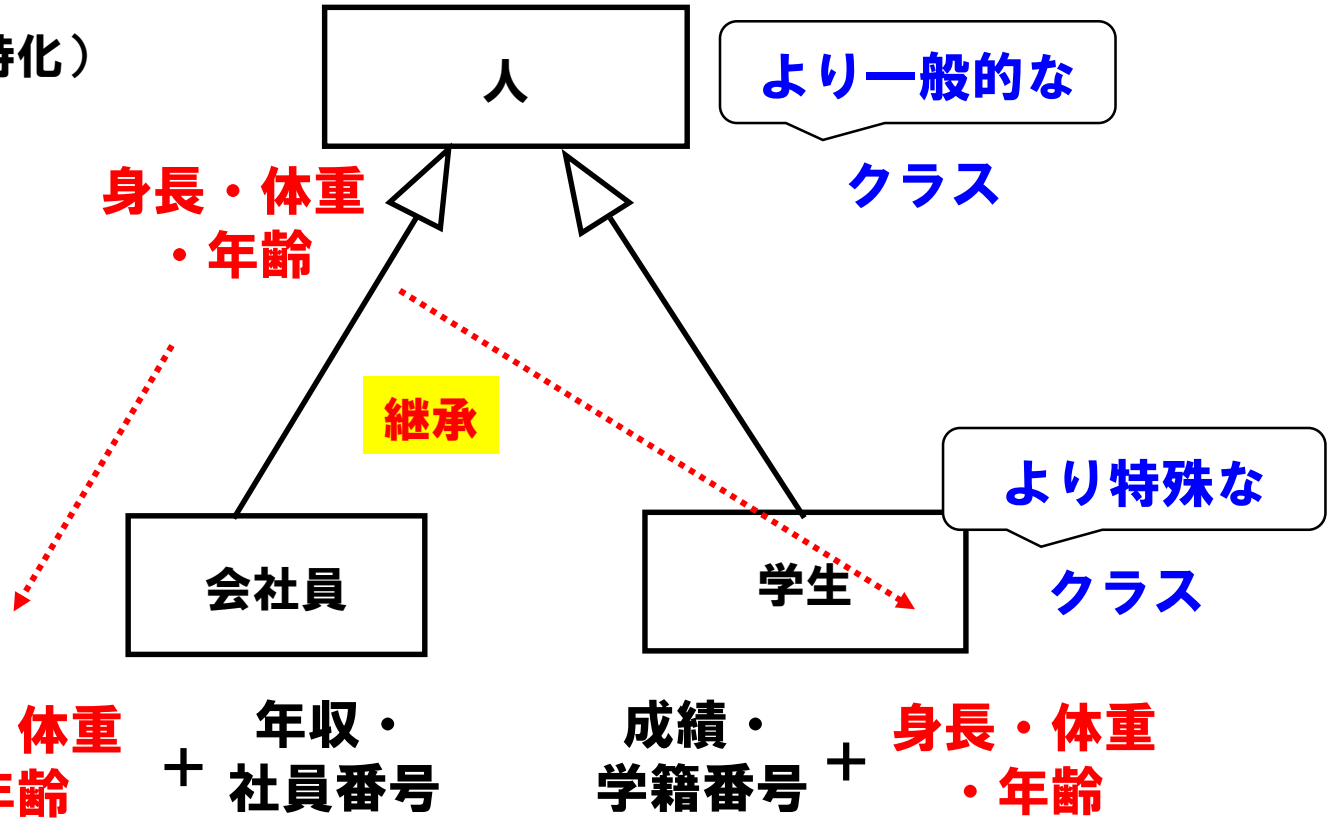
Gen-Spec（汎化 - 特化）
階層ともいう
Generalization
-Specialization



クラス間の共通要素（属性，メソッド）

**上下関係にあるクラス同士には共通の要素がある．
通常，より一般的なもの（上位クラス）が要素の追加によって
特殊化される（下位クラスが作られる）**

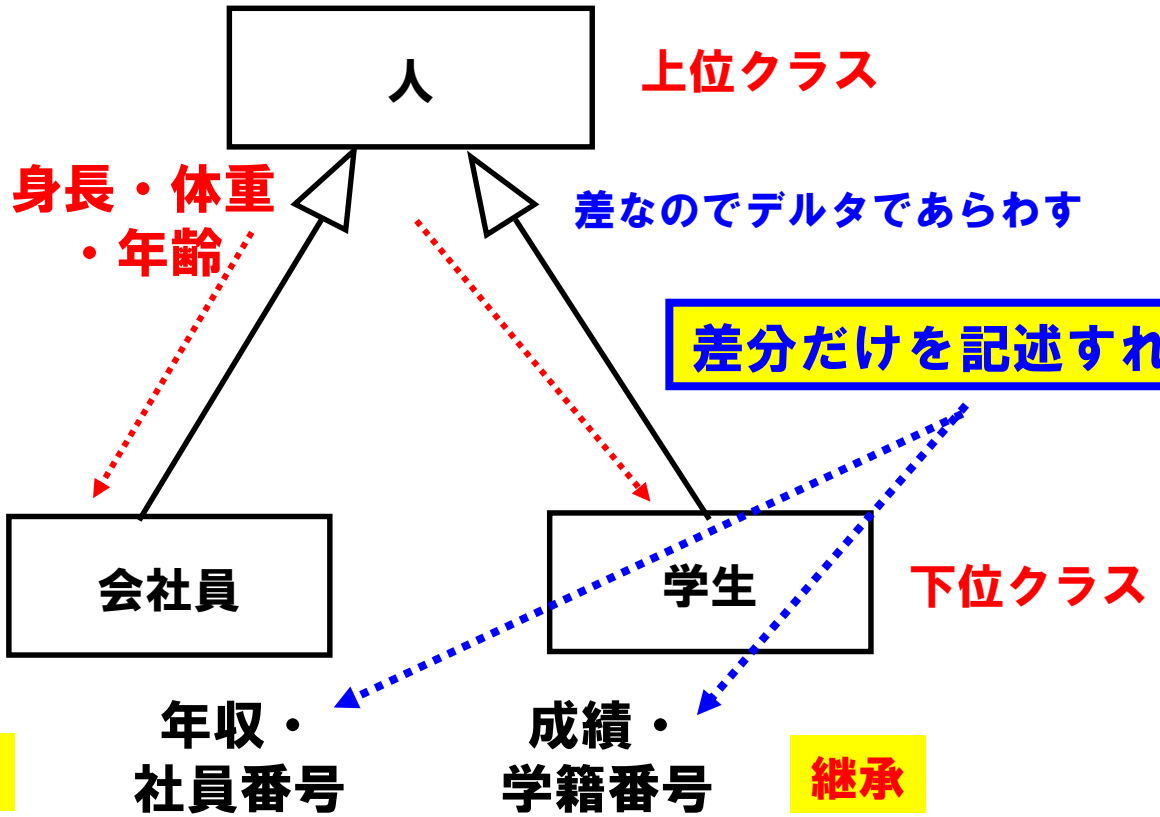
Gen-Spec（汎化 - 特化）
階層ともいう
Generalization
-Specialization



iijima@ae.keio.ac.jp

性質継承と差分プログラミング

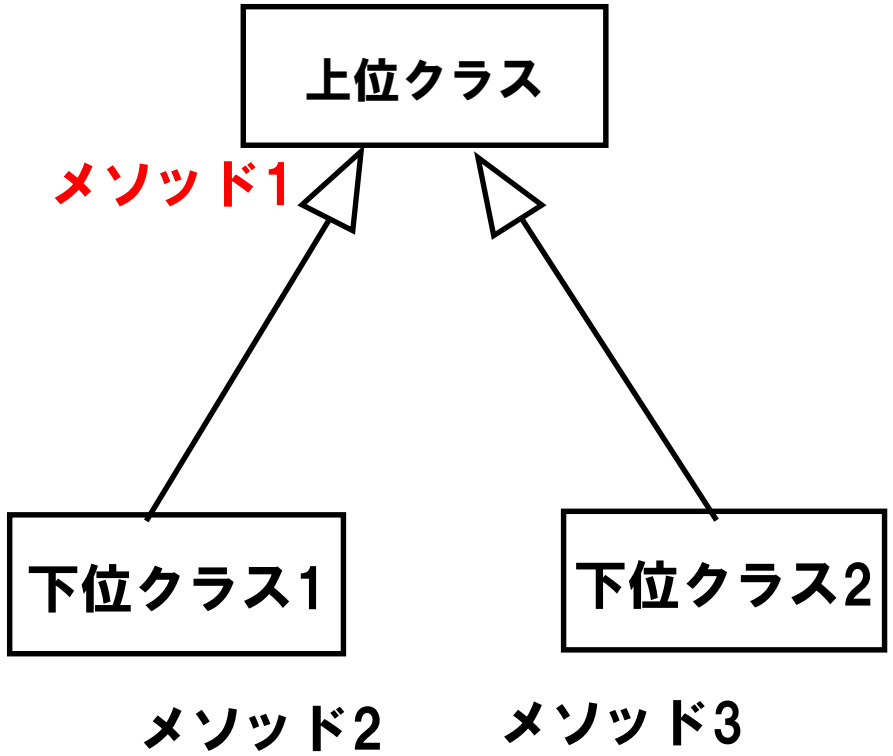
上位クラスから
下位クラスに向かって
属性とメソッドが
継承される



iijima@ae.keio.ac.jp

差分プログラミングにおける継承の打ち消し（オーバーライド）

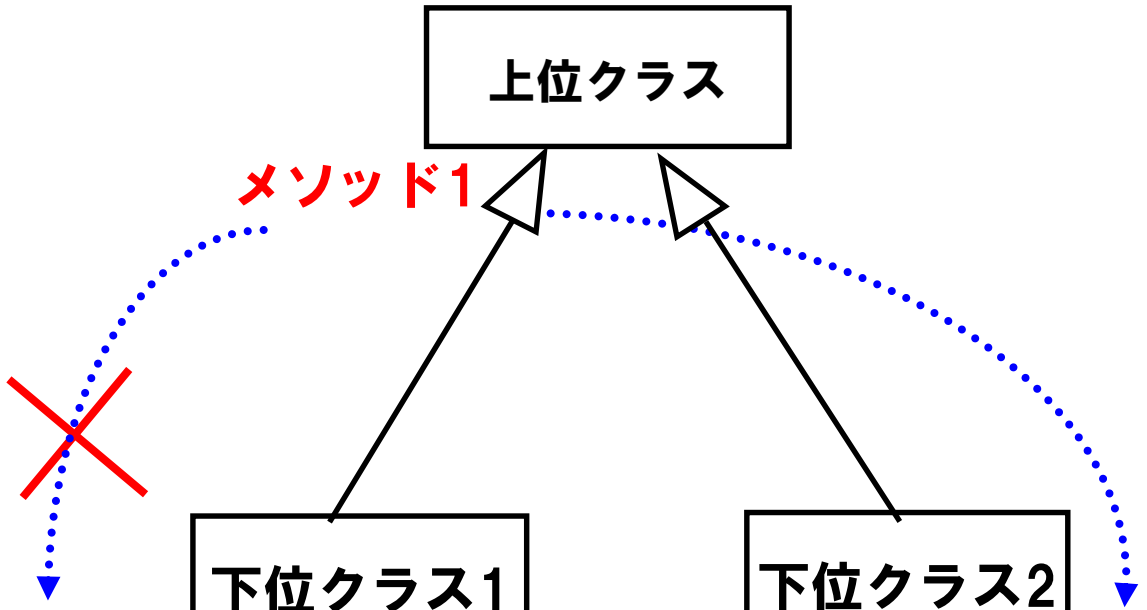
同じ名前
（同じシグネチャだが）
定義が互い
異なる場合
継承しない



iijima@ae.keio.ac.jp

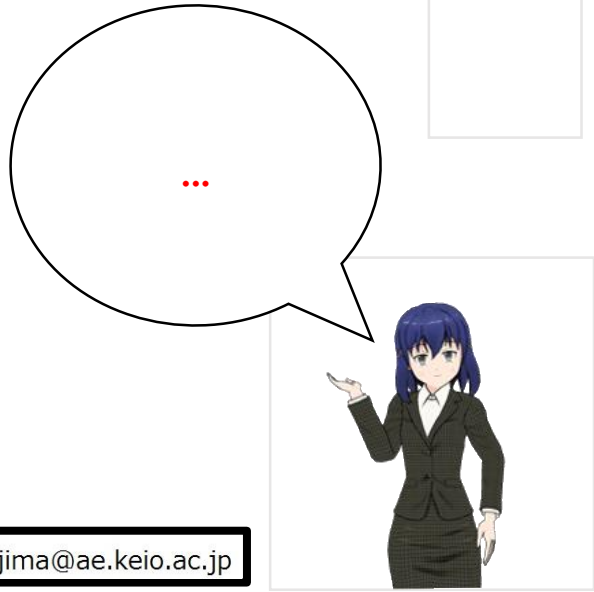
差分プログラミングにおける継承の打ち消し（オーバーライド）

同じ名前
（同じシグネチャだが）
定義が互い
異なる場合
継承しない



メソッド1 ‘ ＋ メソッド2　　メソッド3 ＋ 継承する

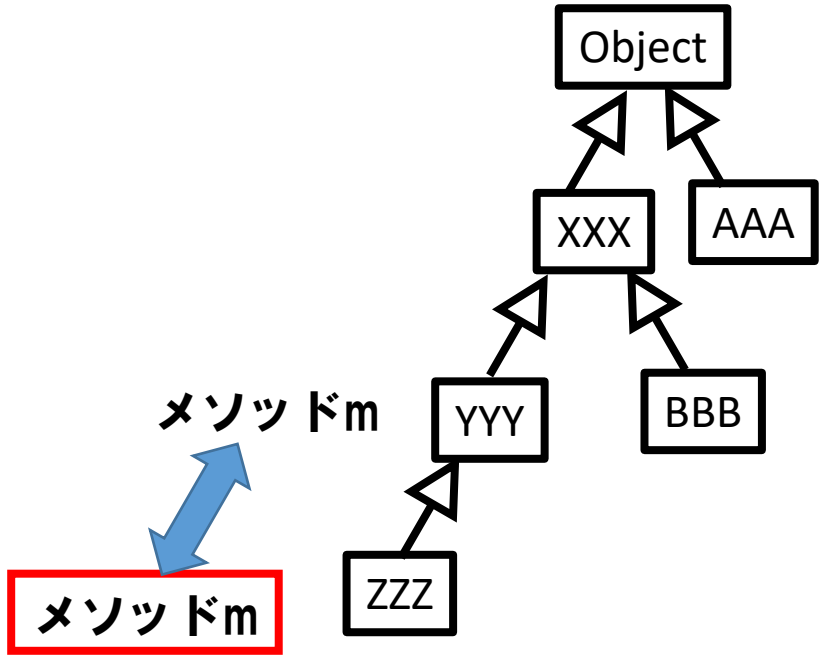
差分 ＝ 再定義 ＋ 追加



iijima@ae.keio.ac.jp

継承の打ち消し（オーバーライド）

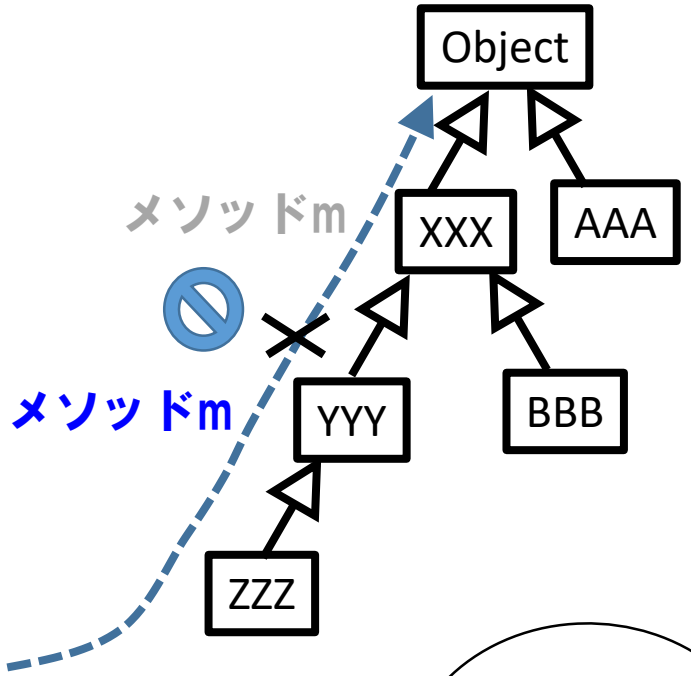
- 継承の打ち消し（オーバーライド）
  - 上位クラスで定義されているメソッドと，
    同じシグネチャ（メソッド名，~~返戻値の型~~，
    引数の個数・型・順序）を持つメソッドが，
    下位クラスで定義されていたら，
    上位クラスのメソッドを継承せず，
    より下位クラスでの定義の方を優先する．



サブクラスでの定義を優先する

継承の打ち消し（オーバーライド）

- 継承の打ち消し（オーバーライド）
  - 上位クラスで定義されているメソッドと，
    同じシグネチャ（メソッド名，~~返戻値の型~~，
    引数の個数・型・順序）を持つメソッドが，
    下位クラスで定義されていたら，
    上位クラスのメソッドを継承せず，
    より下位クラスでの定義の方を優先する．

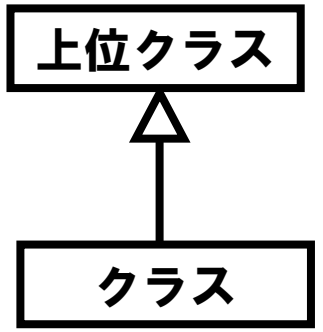


  - もう少し正確には，
    インスタンス化するときに使ったクラスから見て，
    継承の木構造中で根（ルート;JavaではObjectクラス）
    まで遡っていく際に，最初に見つかった，
    より手前にあるクラスのメソッドを優先する
    （単一継承　⇒　単一ルートの場合）．

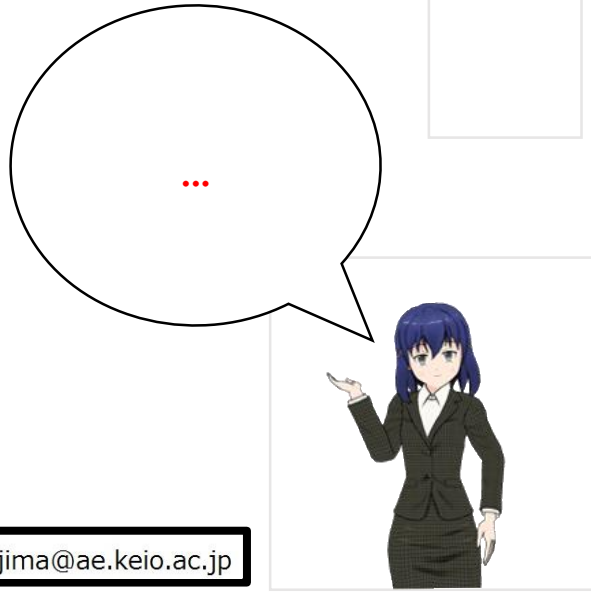
【参考】Javaでの継承の指定方法

- 実は，CounterFrameは，JFrameのサブクラス

```
public class クラス名 extends 上位クラス名 {
                    …
}
```

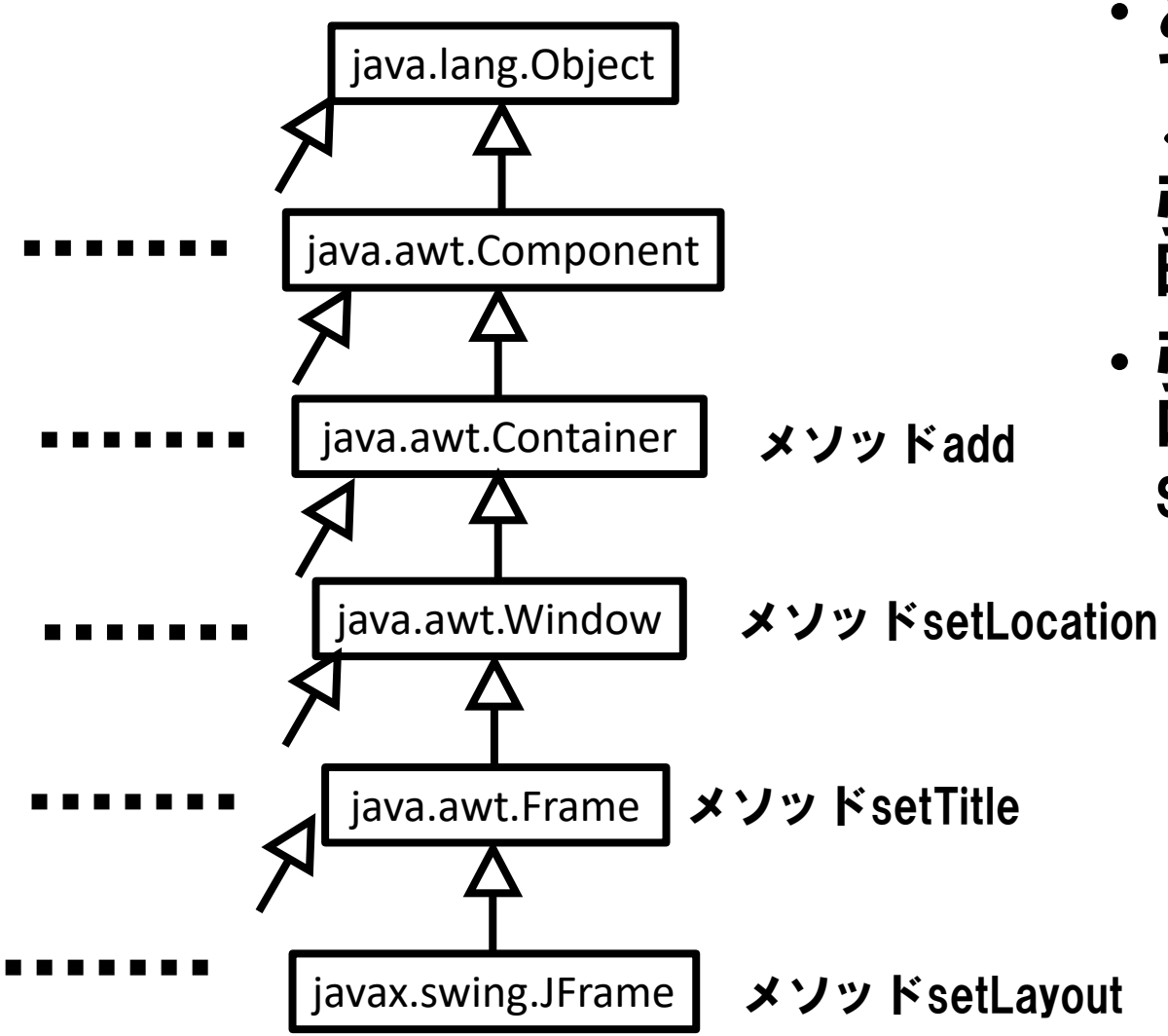


- 単一継承
  - 上位クラスは一つだけ指定できる
- オーバーライド（継承の打ち消し）
  - 上位クラスとシグネチャ（名前，引数/返却値の型と個数）が同じメソッドは，下位クラスで定義されている方を優先する
- 動的束縛とポリモルフィズム

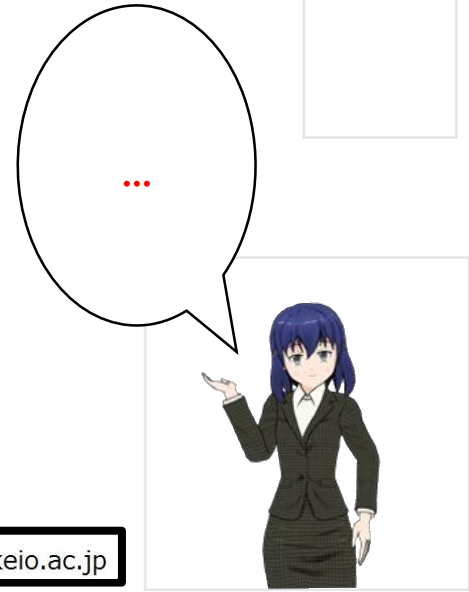


iijima@ae.keio.ac.jp

コンストラクタ

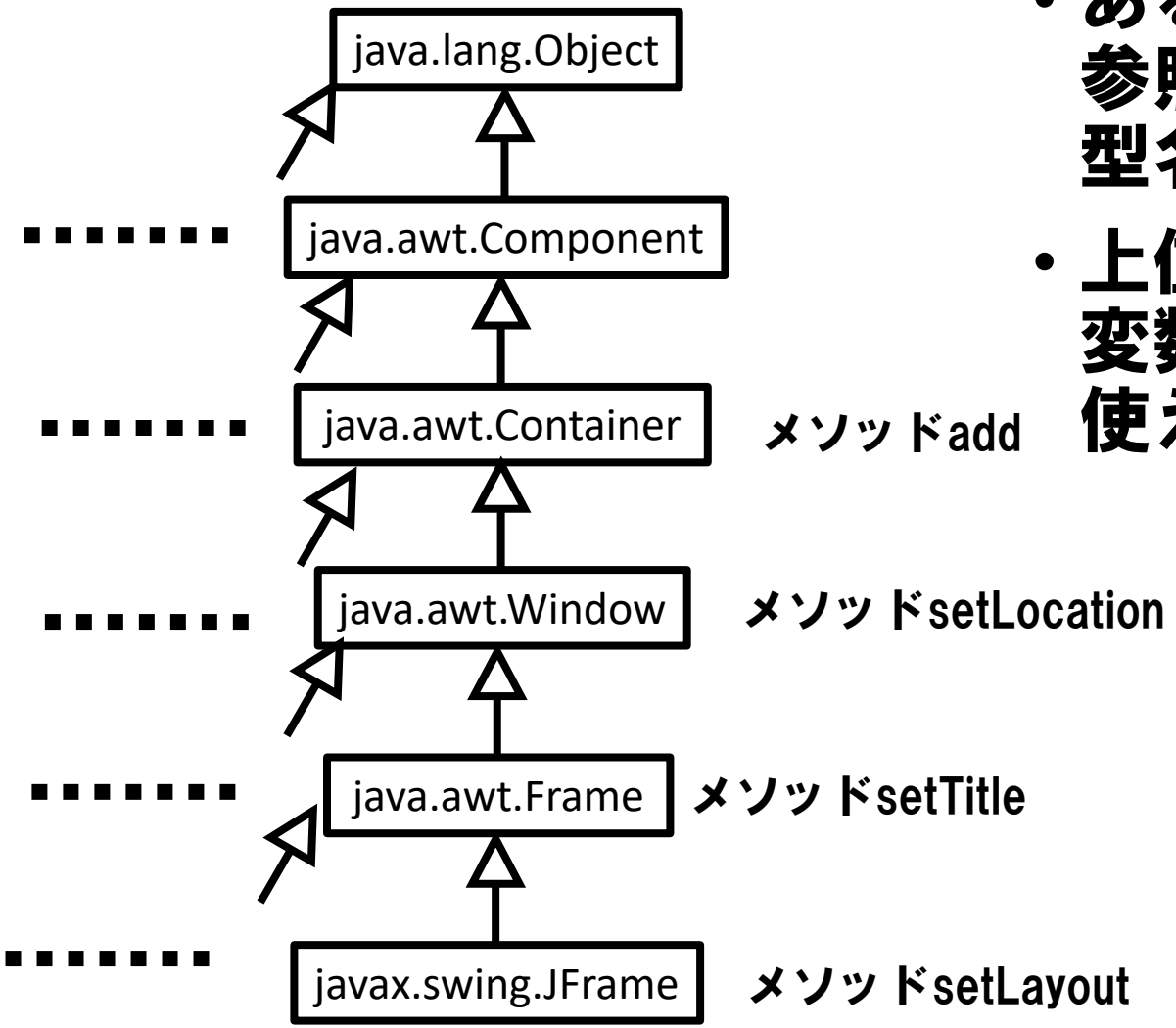


- あるクラスのコンストラクタでは１行目にsuper()が書かれていなくても，上位クラスの引数なしのコンストラクタが暗黙的に呼ばれる．
- 引数のあるコンストラクタを呼び出したいときには，明示的にsuper(…引数並び…)を明記する

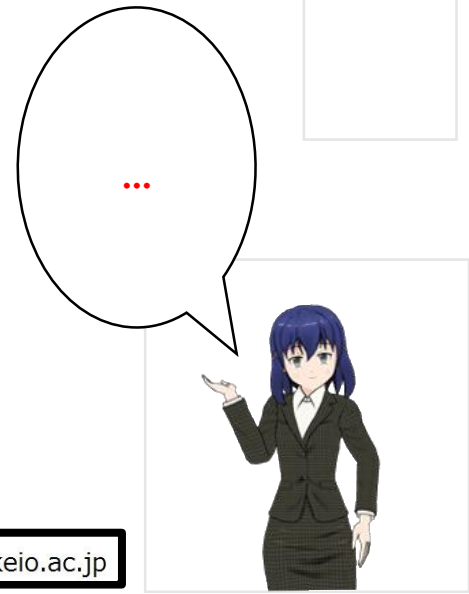


iijima@ae.keio.ac.jp

参照型変数の互換性

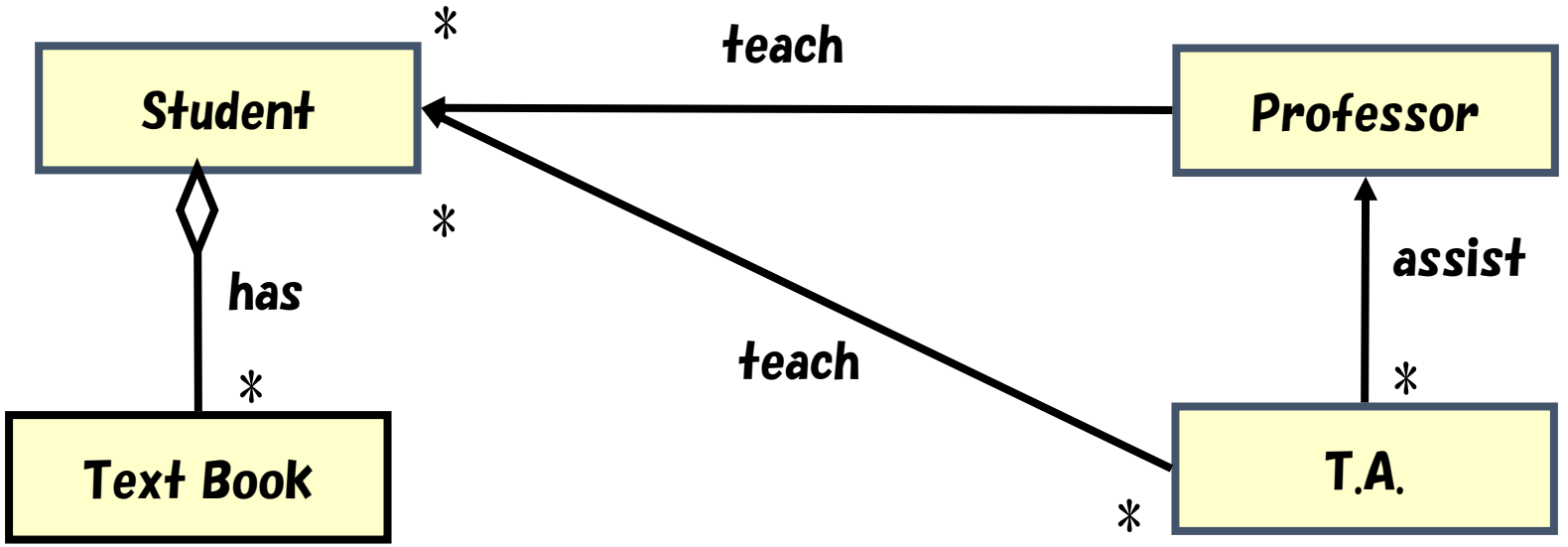


- あるクラスのインスタンスへの参照は，そのクラス名を型名とする変数に代入できる
- 上位クラスを型名とする変数にも代入できるが，使えるメソッドが限定される

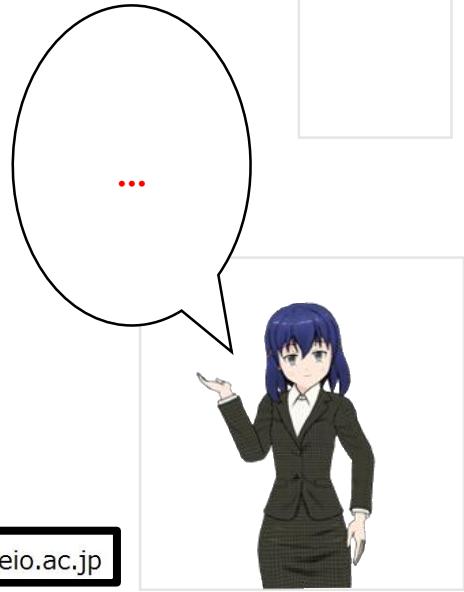


iijima@ae.keio.ac.jp

オブジェクト間の関連

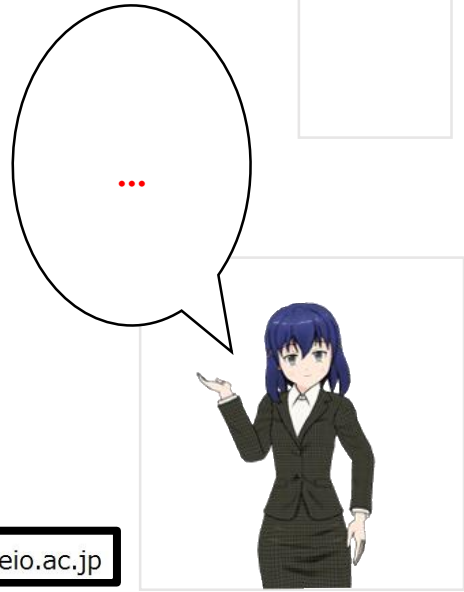


⇒グラフ構造
　　ノード集合×アーク集合



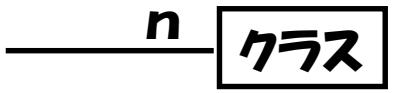
インスタンス間の関連

- インスタンス間の関連
  - プログラミング段階ではクラス中に埋め込まれる
    →リファレンス（後述）を値とするインスタンス変数
    ←→分析／設計段階での属性ではない
- インスタンス間の関連の分類
  - 構造に関するもの
    - 部分-全体関係
  - サーバ／クライアント間関係
    - メッセージ授受関係
    - 知人関係 (acquaintance)

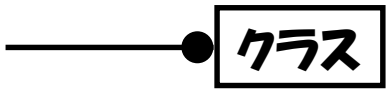


関連の方向，多重度，次数

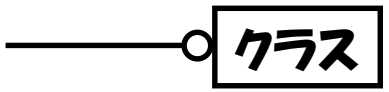
- 方向
  - 単方向，双方向
- 多重度
  - １対１関係，１対多関連，多対多関連



厳密にn個
（n=1なら省略可）

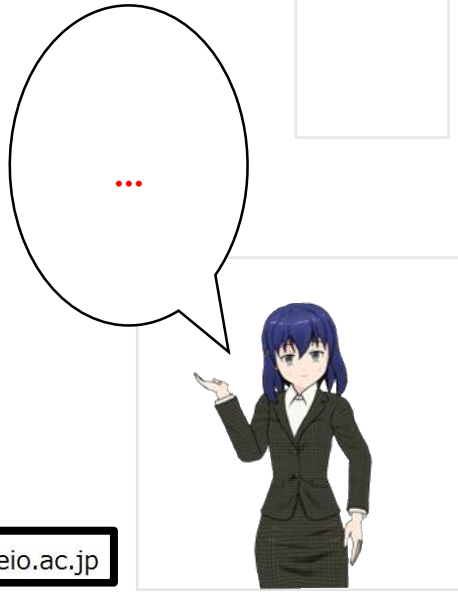


多数（0以上）



選択（0か1）

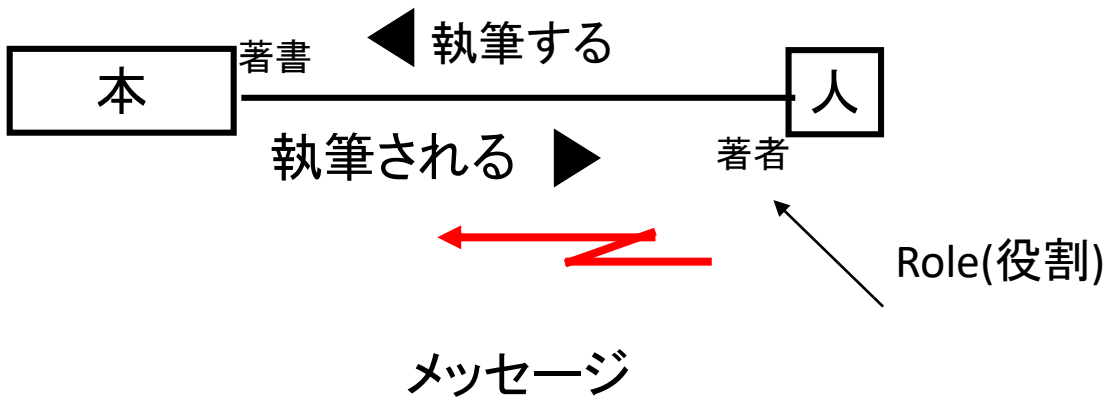
- 次数
  - 2項関係，多項関係

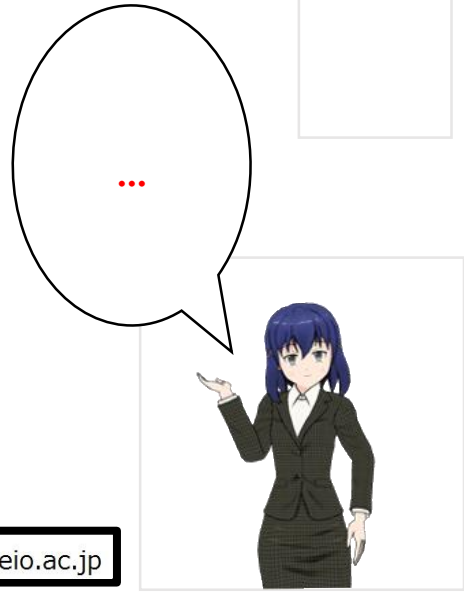


iijima@ae.keio.ac.jp

関連（メッセージ授受関係）

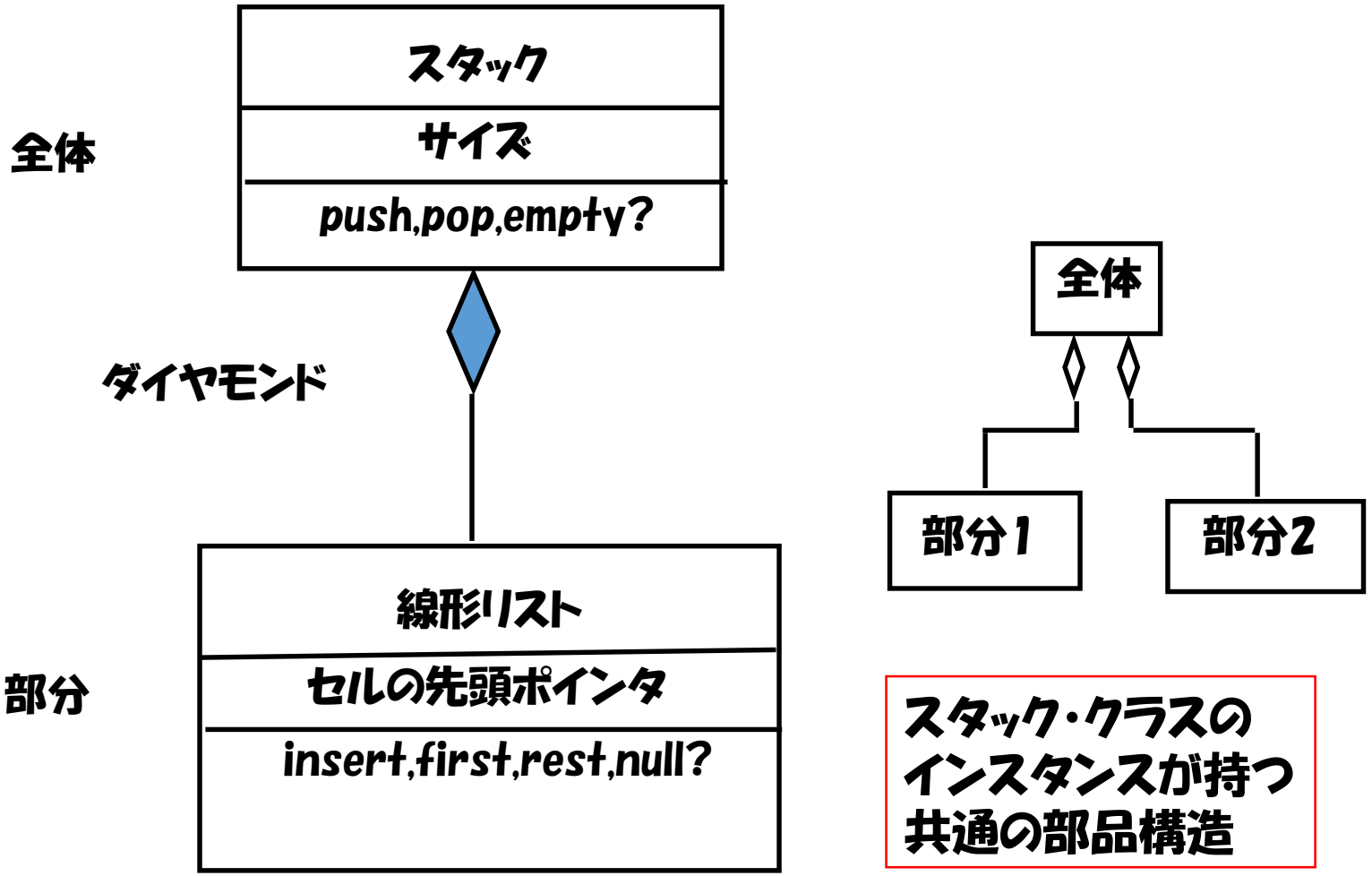
- メッセージ授受関係
  - メッセージを送るためには，送り先を指定しなければならない．
- サーバ／クライアント間関係
  - 一方，分散オブジェクトのための基盤環境(CORBA)の整備により，C/S(Client/Server)プログラミングを包含しつつある．



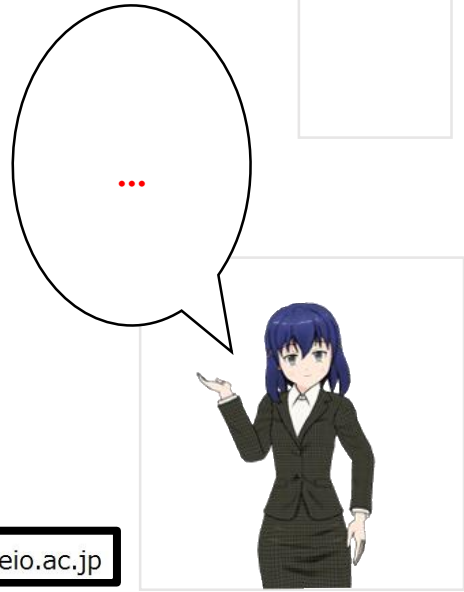


iijima@ae.keio.ac.jp

部分-全体構造 (part-whole,has-a)



スタック・クラスの
インスタンスが持つ
共通の部品構造



部分-全体関係の種類

◆——— 組立/合成

ライフタイム一致

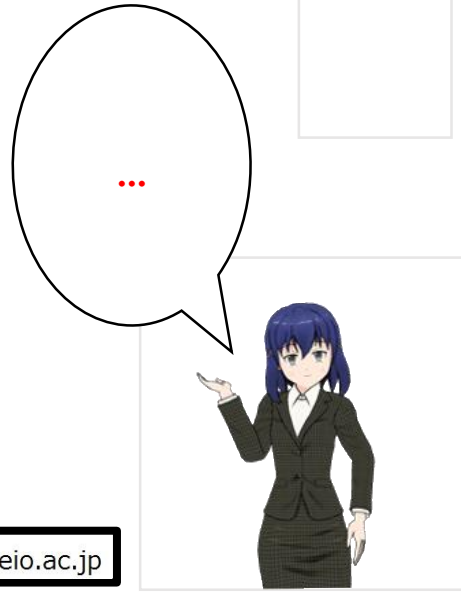
全体と部分 (フレームワーク)	車とタイヤ
容器と内容 (コンテナ)	飛行機と乗客
集合と要素 (コレクション)	チームとメンバ

静的
(組立構造)

動的
(中身が
増減したり,
入れ替わる)

集約 ———◇

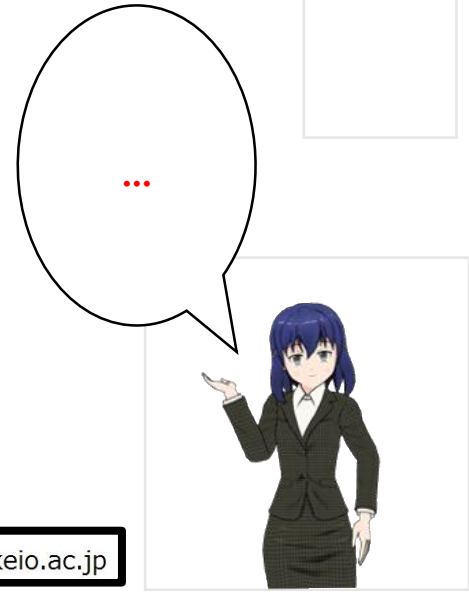
ライフタイム不一致/共有可



iijima@ae.keio.ac.jp

部分-全体関係の意義

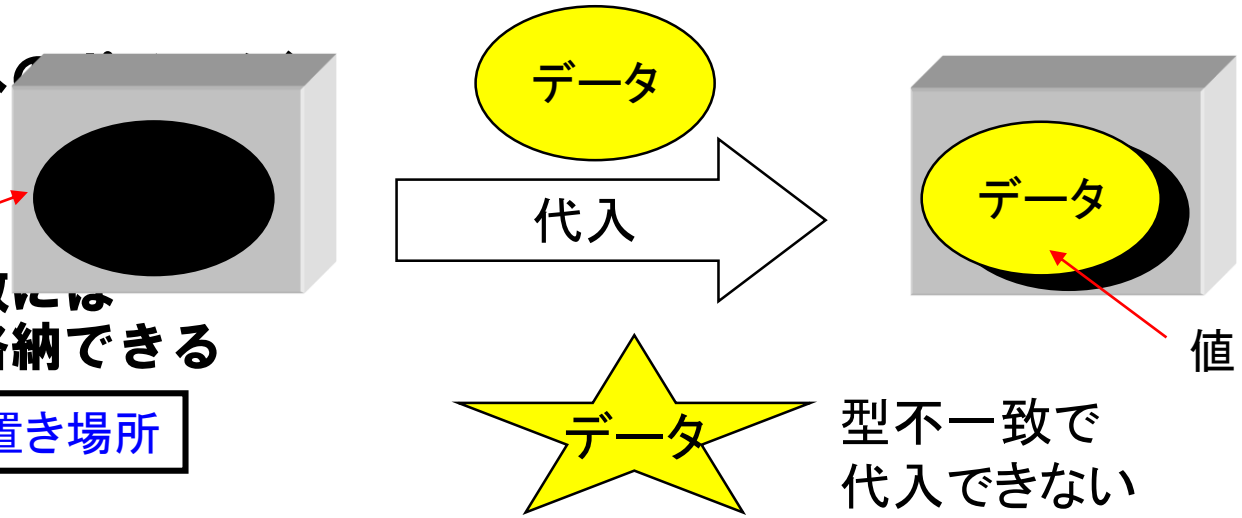
- 一般の関連の特殊ケース
  - 静的な組立構造
    - 「全体」が，「生成」と「消滅」の責任を持つ
      - 生成→「全体」のコンストラクタの中で
        「部分」のコンストラクタを呼ぶ
      - 消滅→javaにはデストラクタはない．
        メモリ管理は自動GC（garbage collector）．
        GCで回収される前にfinalize()メソッドが呼ばれる
    - ライフタイムが一致することが多い
      - 一致しない場合もある．
        廃車のタイヤだけが他の車に再利用される場合など
  - 動的なもの
    - 集合が要素の挿入,削除の際に制約チェックする
      - 挿入や削除のメソッドの中に制約を埋め込む

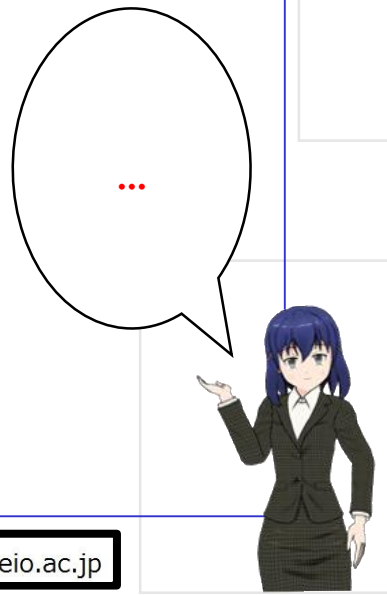


Javaでの（型付き）変数

- **値**は，
  - 基本データ型のデータ値か
  - リファレンス（オブジェクトへ
- **型**は一つ
  - 但し，リファレンス型の場合，
    クラス階層で上位クラスの変数には
    下位クラスのインスタンスを格納できる

変数：データの置き場所

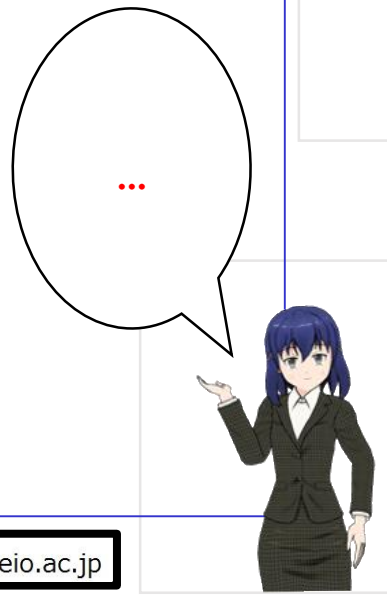




Javaの基本データ型

- ほとんど，C言語と同じと思えばよいが，若干異なる

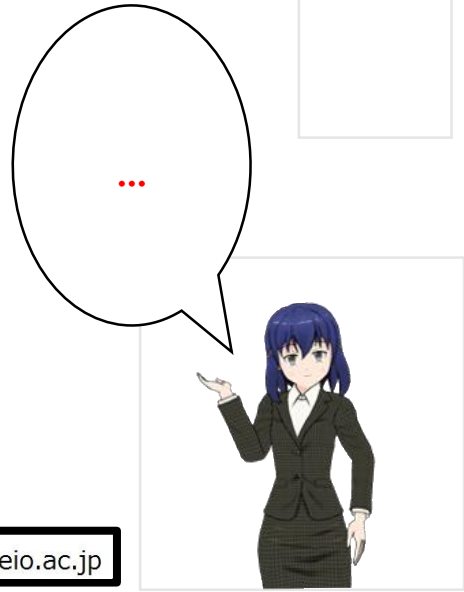
整数	byte short char int long	8bits 16bits 16bits(Unicode) 32bits 64bits
浮動小数点数	float double	32bits 64bits
論理値	boolean	1bits(true,false)



Javaでのインスタンス変数の使い方

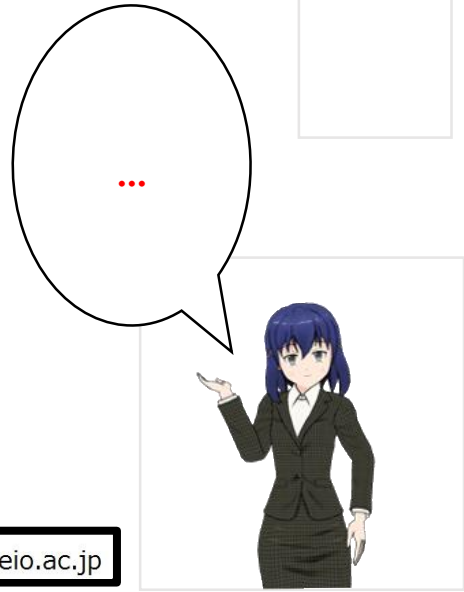
属性	基本データ型変数
関連	クラス・リファレンス型変数 （但し，基本は単方向かつ1対1）

クラス参照（リファレンス）型を，
単にクラス型とよぶこともある



Javaでの関連における個数の対応関係の表現

1対1関連	クラス参照(リファレンス)型変数で表現 (双方向なら相互にリファレンスを持ち合う)
1対多関連	リファレンスのコレクションで表現 (コレクションには，配列やVectorクラスの インスタンスが使える)
多対多関連	関連付けオブジェクト



【参考】Javaでの関連における個数の対応関係の表現

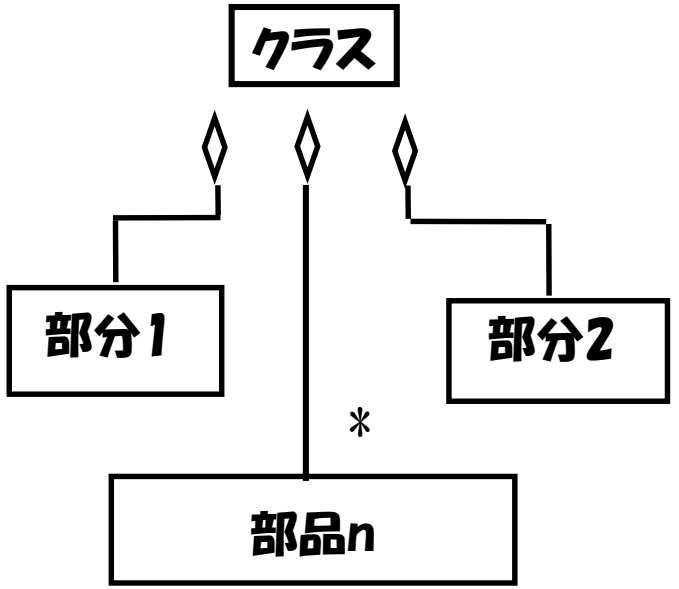
クラス参照(リファレンス)型変数

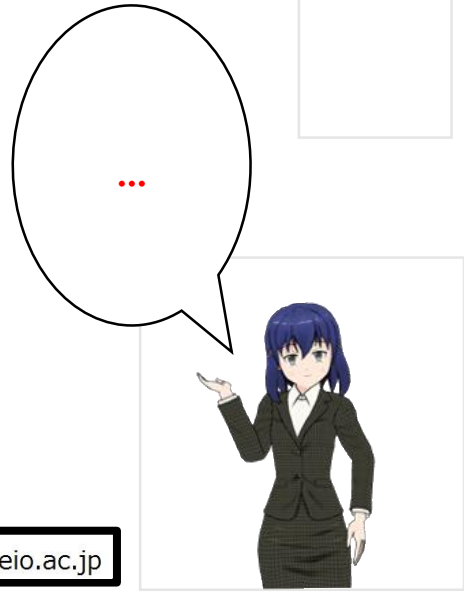
```
class クラス名 {
        属性の型名前;
        部品クラス1        名前;
        部品クラス2        名前;
        Vector             v;
}
```

属性

部分-全体関係

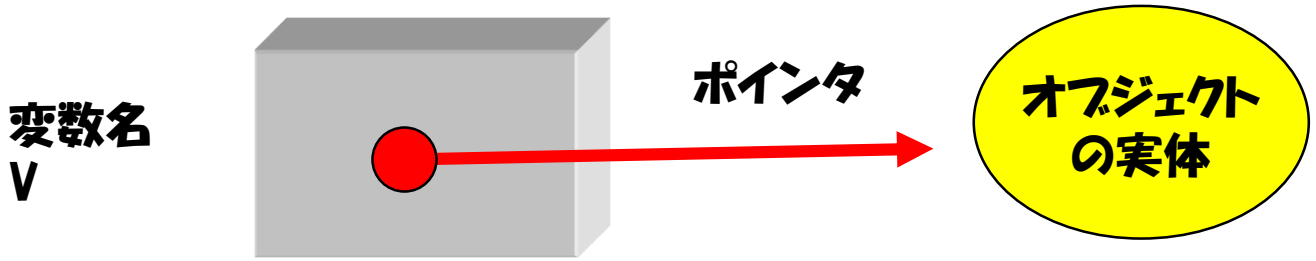
個数が不定個の(インスタンスによって異なる)場合，
1対多関連の場合





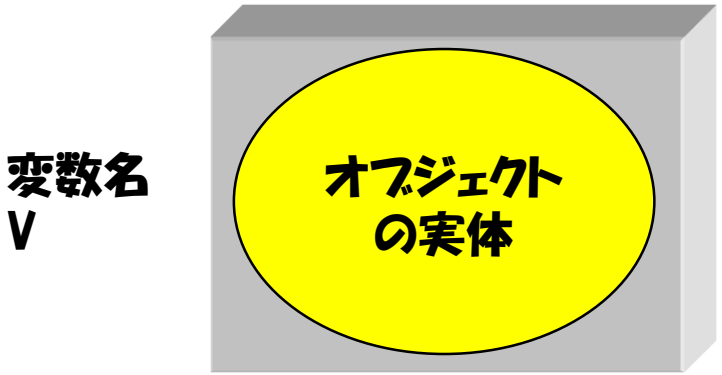
型の互換性：Javaにおけるリファレンス（参照）

- 参照は「ポインタ」と「名前付け」の両方の機能を持つ

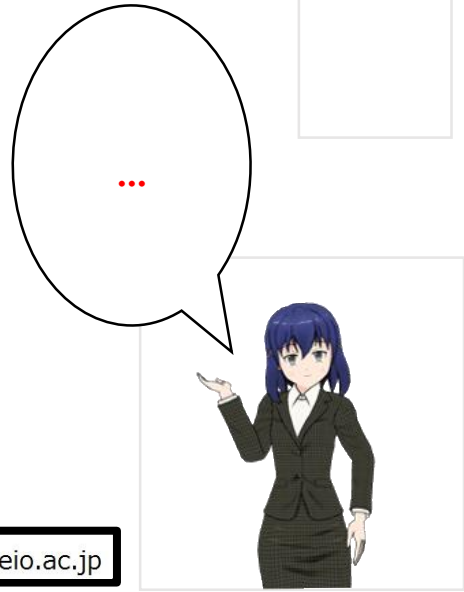


- つまり概念的には．．．

このオブジェクト自身の
名前がVであるかのように扱える

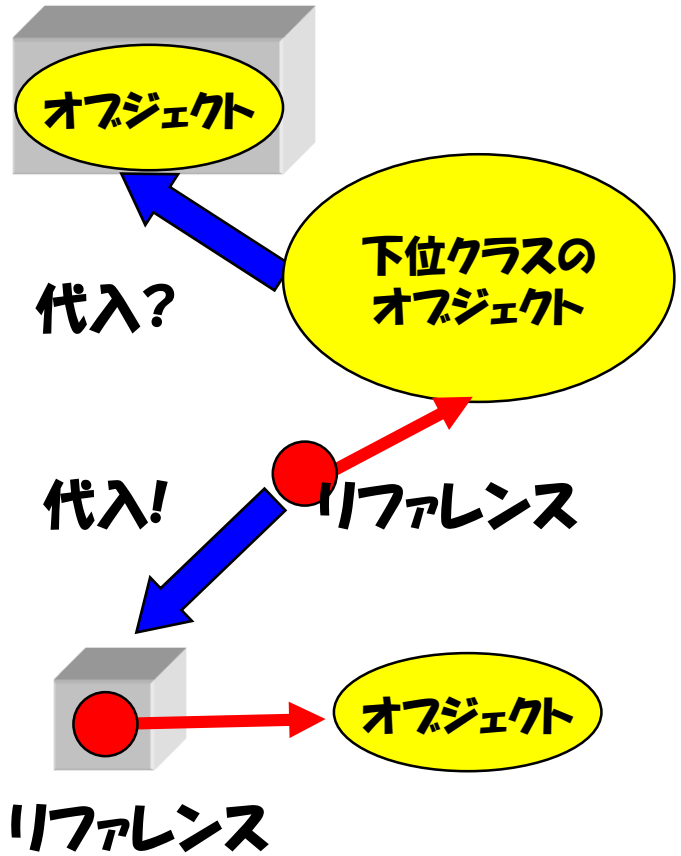


V.メソッド名(...)



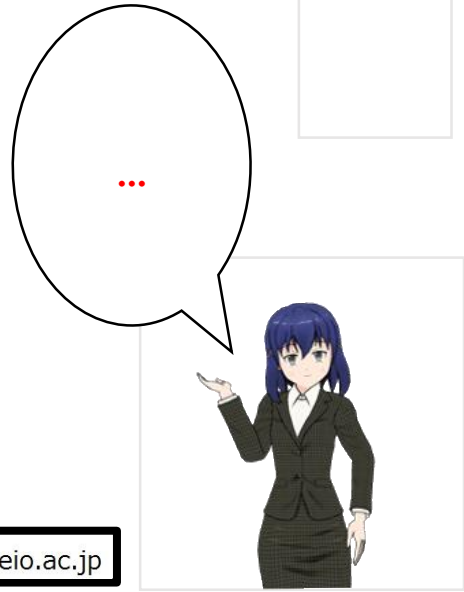
型の互換性：Javaにおけるリファレンス(参照)

- あるクラスを型とする変数には，
  その型のサブクラスのインスタンスを代入できる



一般に下位クラスのインスタンスの方が，
上位クラスのインスタンスより
サイズが等しいか大きい
(インスタンス変数が追加されるので)．
変数にオブジェクト自身を格納すると
箱の大きさが不定となってしまう．

現実にはリファレンスだけを格納する．
リファレンスのサイズは
オブジェクトの大きさによらず
同じなので問題は無い

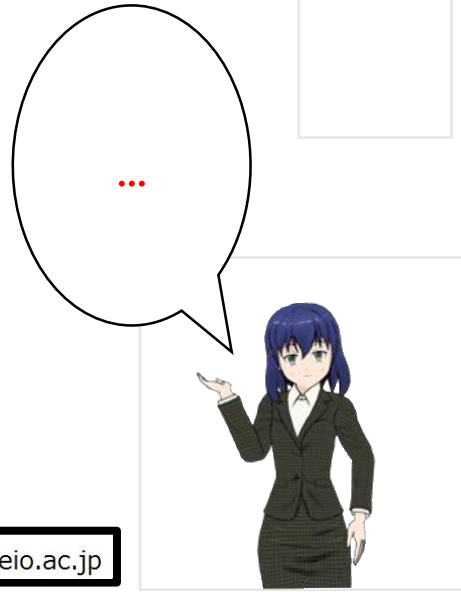


III. UMLの表記法の基礎 (Unified Modeling Language)

- ***静的構造図*** (Ver.1.4の範囲のみ)

- ***ユースケース図***
- ***相互作用図***
- ***振る舞い図***
- ***実装図***

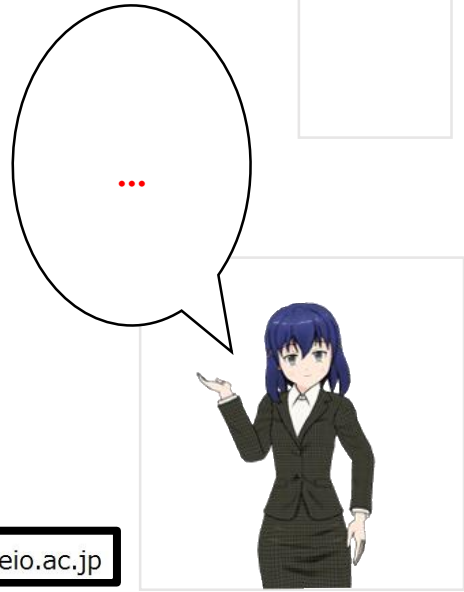
} 今日は扱わない



iijima@ae.keio.ac.jp

UML

- Unified Modeling Language (UML)
  - 表記法だけの統一化
  - UML 1.1
    - 1997.09.01公開
    - http://www.rational.com/uml/1.1
    - OMG (Object Management Group) へも提案
  - UML 1.3
    - 800ページ弱の仕様書の翻訳も出版されている.
      OMG Japan SIG 翻訳委員会UML作業部会,
      UML仕様書, ASCII, 2001.
  - UML 1.4
  - UML 2.0
    - 2003年6月
  - 現在は UML2.5
    - 2015年6月
    - http://www.omg.org/spec/UML/2.5/



UML関連のURL

- http://www.uml.org/
- http://www.omg.org

[1]UML 2.0 Superstructure Specification: formal/05-07-04
http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf
[2]UML 2.0 Infrastructure Specification: formal/05-07-05
http://www.omg.org/cgi-bin/apps/doc?formal/05-07-05.pdf
[3]UML 1.4.2 (ISO/IEC 19501でもある): formal/05-04-01
http://www.omg.org/cgi-bin/apps/doc?formal/05-04-01.pdf

UML 2の図面

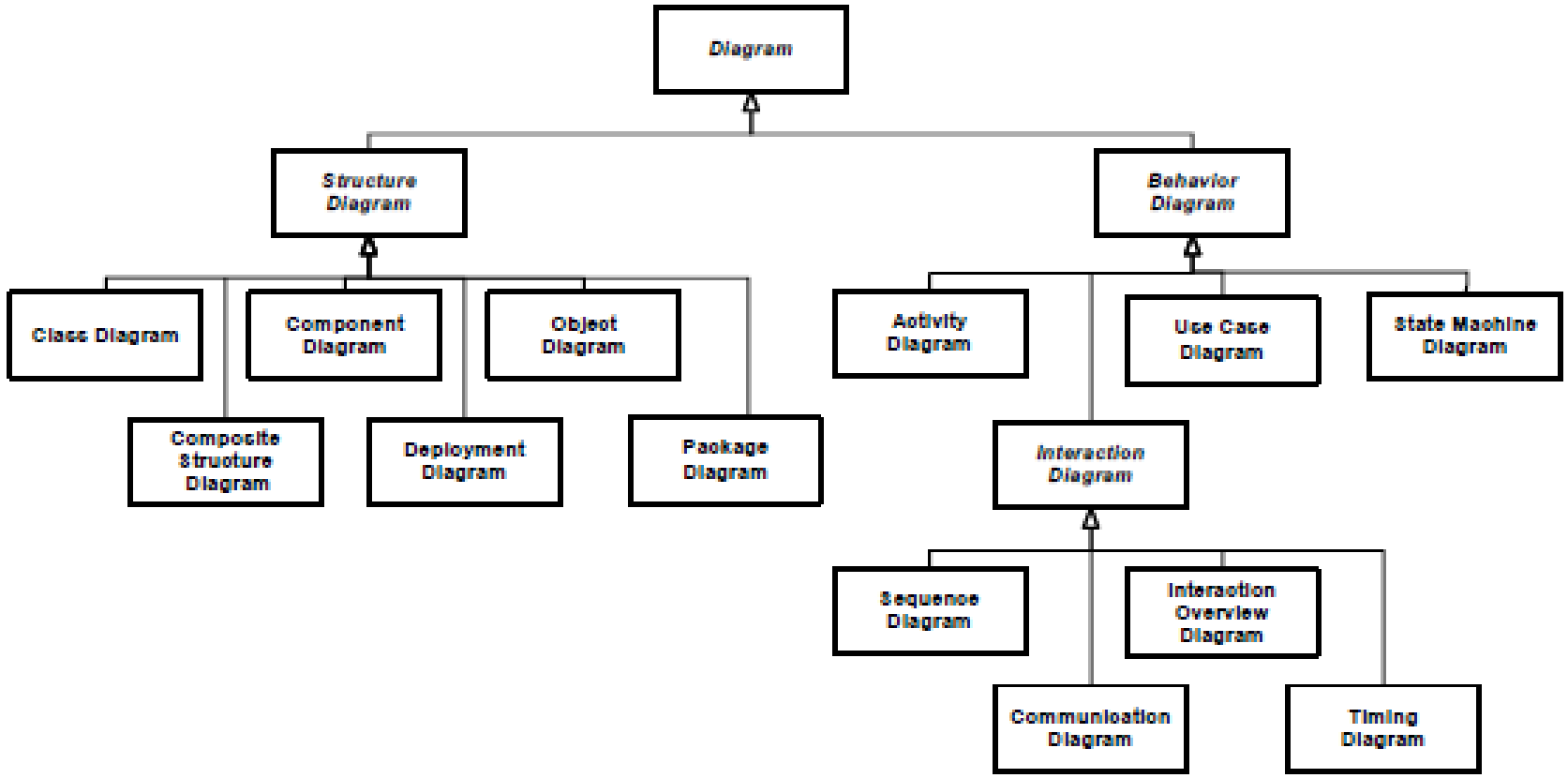
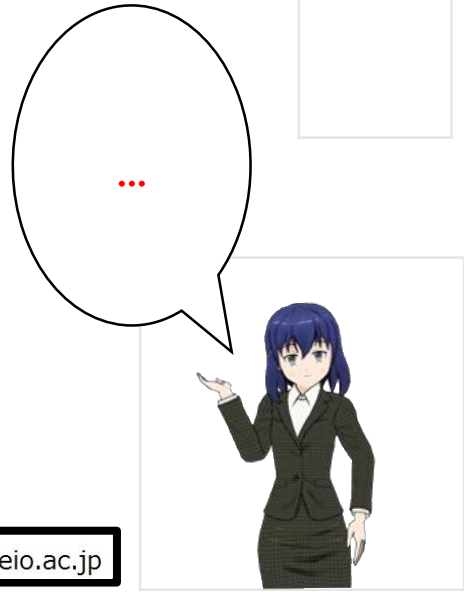


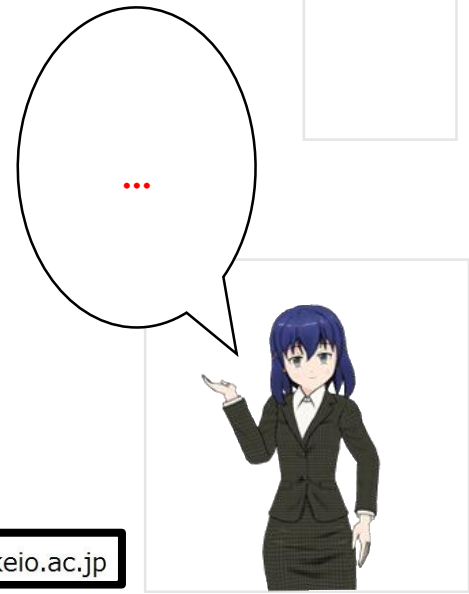
Figure A.5 - The taxonomy of structure and behavior diagram



UML 2の図面

- 構造図
  - (1-a) クラス図 :クラスの仕様（属性，操作など）やクラス間の関係（関連，汎化など），などを定義する図
  - (1-b) オブジェクト図：インスタンスとその関係を定義する図
  - (1-c) パッケージ図：モデルの要素を分類するパッケージを定義する図
  - (1-d) コンポジット構造図（UML 2.0で追加）：クラスの内部構造を定義する図
  - (1-e) コンポーネント図：コンポーネントを定義する図
  - (1-f) 配置図：システムの物理的な構成を定義する図

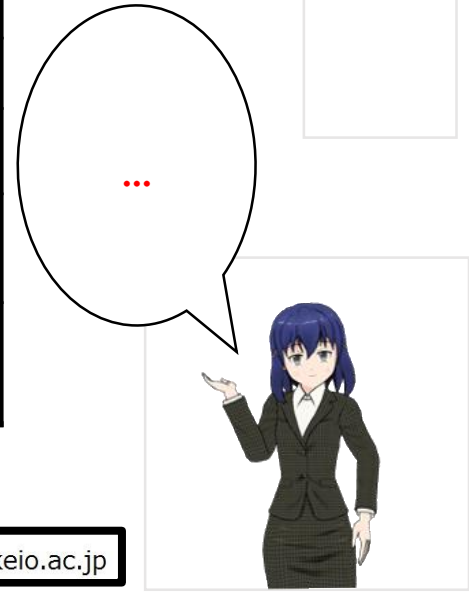
- 振る舞い図
  - (2-a) ユースケース図：システムが提供する機能を定義する図
  - (2-b) アクティビティ図：アクションの実行順序を定義する図
  - (2-c) 状態マシン図：状態と状態遷移を定義する図
  - (2-d) シーケンス図：クラス間の相互作用を時系列に定義する図
  - (2-e) コミュニケーション図（コラボレーション図）：クラス間の相互作用をクラス間の関係に着目して定義する図
  - (2-f) 相互作用図(UML2.0で追加)：相互作用の実行順序を定義する図
  - (2-g) タイミング図(UML2.0で追加)：相互作用と状態遷移に関する時間制約を定義する図



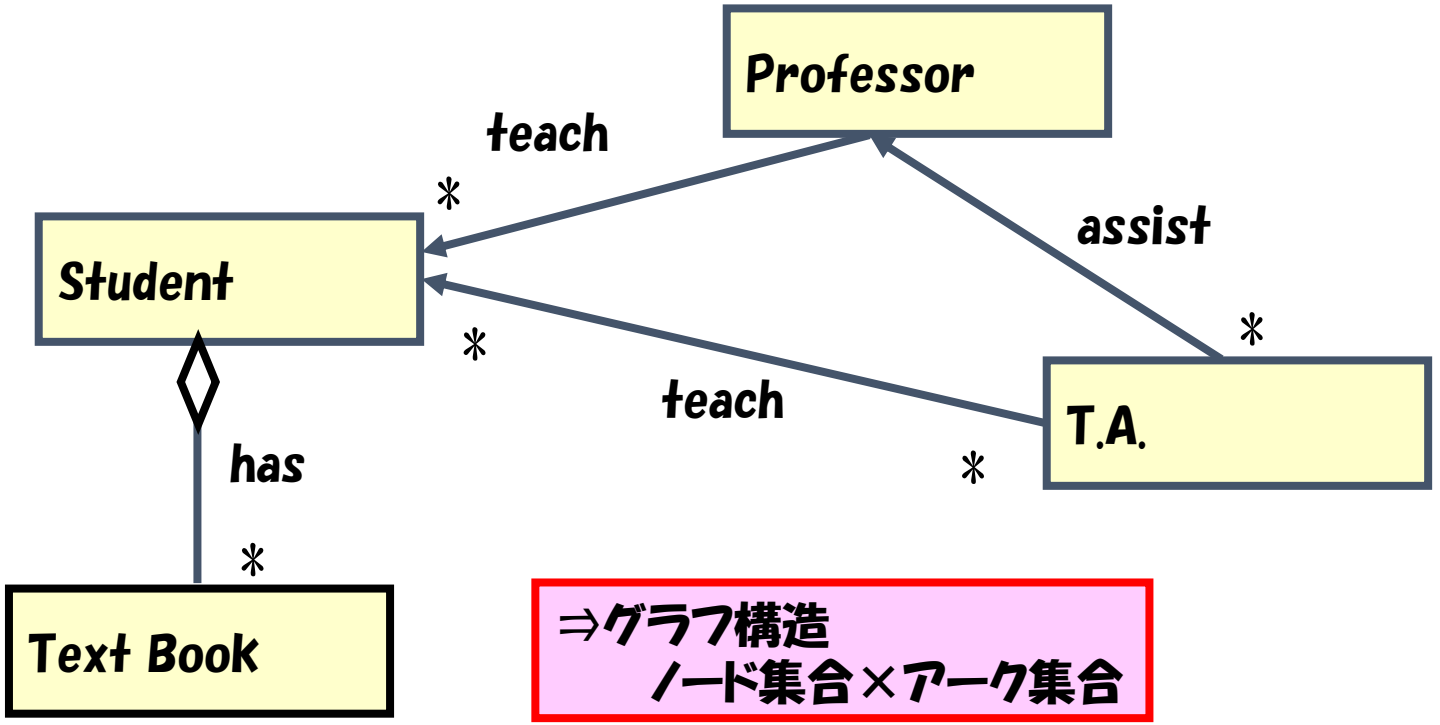
iijima@ae.keio.ac.jp

III－1. 静的構造図

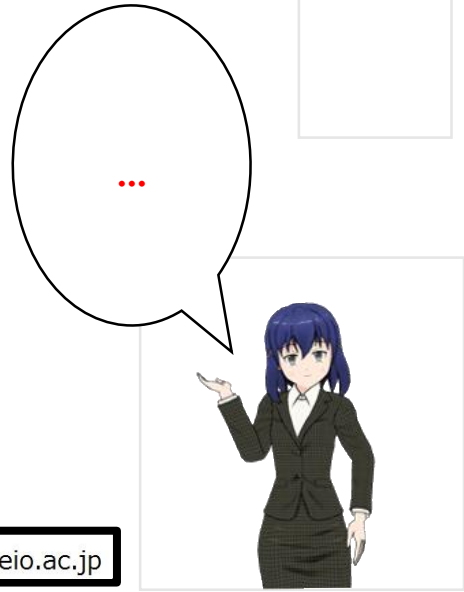
	名称	分析	設計	
	ユースケース図	◎		システムの機能を利用者の視点から示す（システムの振舞い，アクタ／外部システムとの関係）
静的構造図	クラス図	◎	◎	クラス間の関係（システムの静的な構造）
	オブジェクト図	◎	◎	クラス図中のクラスのインスタンス（ある時点を取り出したスナップショット）
相互作用図	シーケンス図	◎	◎	時系列的なメッセージのやり取り
	コラボレーション図	◎	◎	構造に着目した相互作用
振舞図	ステートチャート	◎	◎	状態遷移図（状態変化やイベントへの応答に関して，条件を含む）
	アクティビティ図	◎	◎	業務分析図（ビジネスプロセスのフローチャート）
実装図	コンポーネント図		◎	コンポーネント間における依存関係
	配置図		◎	ハードウェア環境と，そこで実行するコンポーネントの割り当て
	パッケージ図		◎	パッケージ（クラスのグループ）の階層構造と依存関係



オブジェクト間の関連



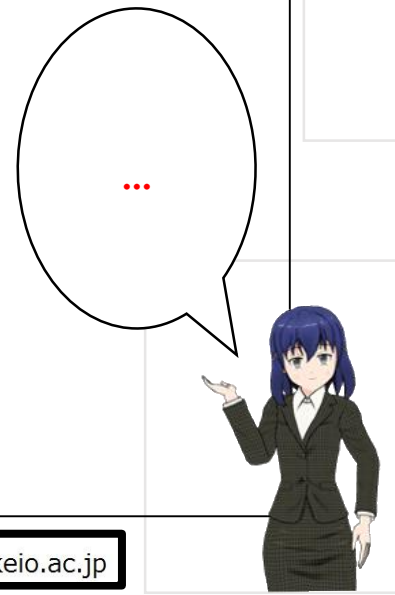
⇒グラフ構造
　ノード集合×アーク集合



クラス（実体の型）の表記

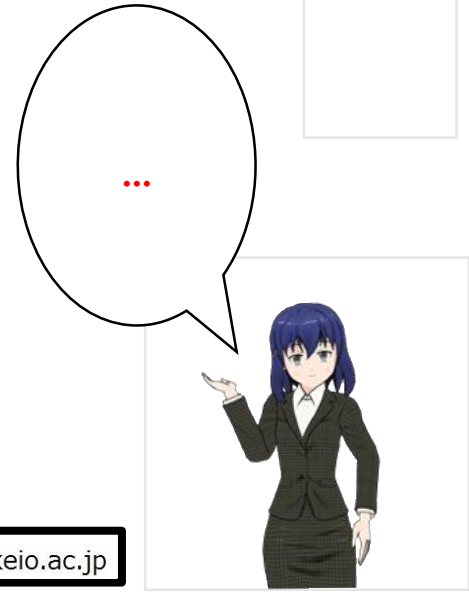
- クラス
  - クラス名
  - 属性：型
  - 操作（引数：型...）：型
- 属性と操作は省略可能
- カテゴリ名::クラス名

多角形
属性 中心：点 頂点：点のリスト 境界色：色 塗りつぶし色：色
操作 表示(on:面) 回転(angle：角度) 表示を消す() 消滅() 選択(p：点)：論理値



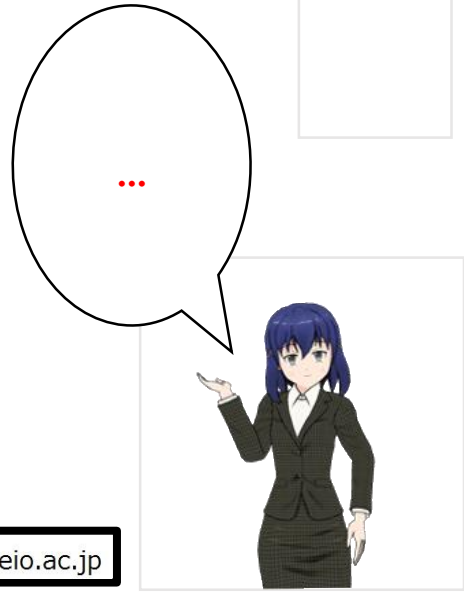
クラスの属性と操作の可視性とスコープ

- 属性（もしくは，変数，内部状態）
  - 名前:型＝初期値
- 操作（もしくは，メソッド）
  - 名前（引数：型＝デフォールト値, ...）：結果の型
- スコープ（有効範囲，可視性）
  - クラス・スコープ（C++ではstatic）
    - $名前
  - 可視性
    - public (+) , protected (#) , private (-)
- 言語依存の他の注釈は {} で囲む
    - {const} , {abstract} ,...



クラス（実体の型）とインスタンス（実体）

	オブジェクト	
	クラス	インスタンス
	型，仕様，テンプレート	値，実体，具体例
名前(name)	大学	慶応大学
属性，変数， もしくは 内部状態 (Attributes)	名前：　文字列型 所在地：文字列型 電話：　電話番号型 代表者：氏名型 学生数：整数型	名前＝慶応義塾大学 所在地＝東京都港区三田 2-15-45 電話＝03-3453-4511 代表者＝ 学生数＝・・・
メソッド， もしくは操作 (Methods or Operations)	入学させる 卒業させる 授業する	（同左）



クラス（実体の型）とインスタンス（実体）の例

「会社」クラス

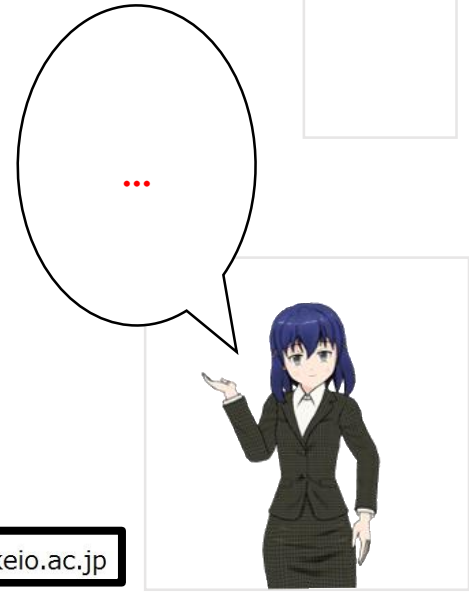
会社
名前 所在地 代表者名
雇う 解雇する 支払う 購入する

クラス図

「大学」クラスのインスタンス

慶応義塾:大学
名前 ＝ 慶応義塾大学 所在地 ＝東京都港区三田 2-15-45 電話＝ 03-3453-4511 代表者名 ＝

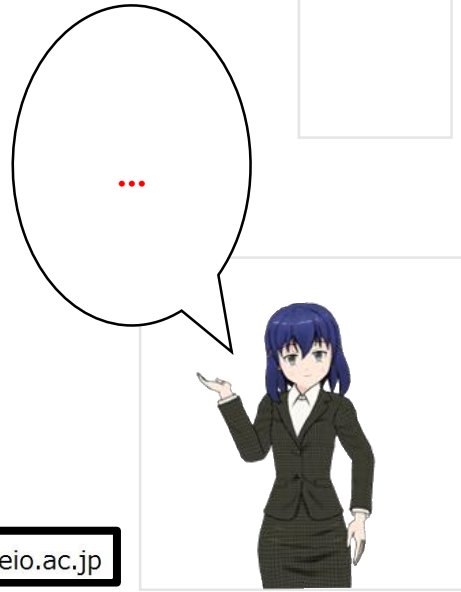
オブジェクト図



インスタンス(実体)の表記：オブジェクト図

- オブジェクト名：クラス名
  ＿＿＿(下線をひく)
- 属性名：型＝値

三角形1:多角形
中心=(0,0) 頂点=((0,0),(4,0),(4,3)) 境界色=黒 塗りつぶし色=白



オブジェクト間，クラス間の関連

一般の関連

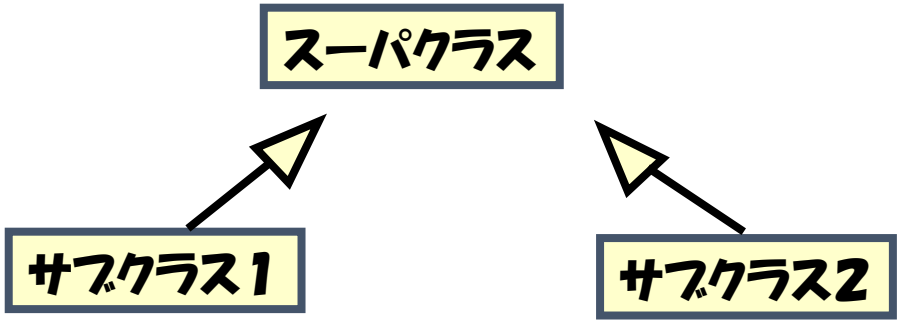
部分-全体関係

集約

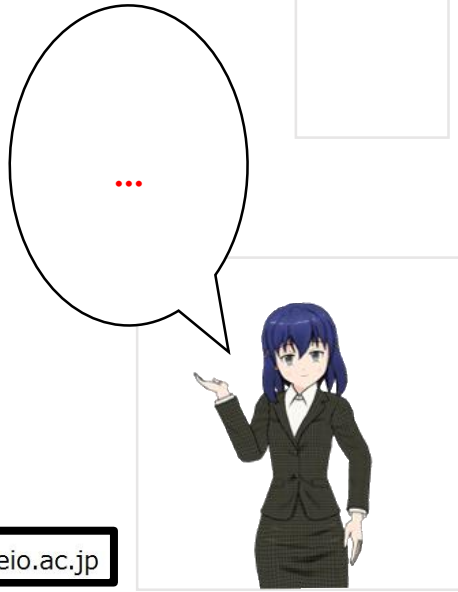
組立

あるクラスに属するすべてのオブジェクトがもつオブジェクト間の関係（実際に関係が結ばれるのはインスタンスの間）

汎化関係（継承関係）



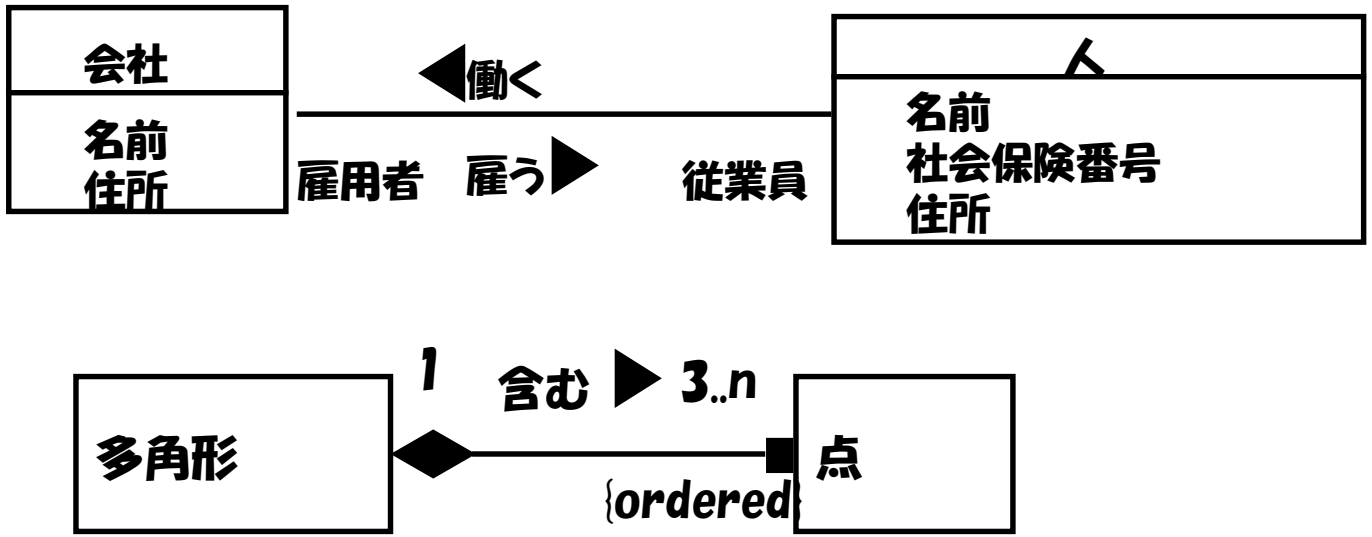
クラスの定義の間の関係

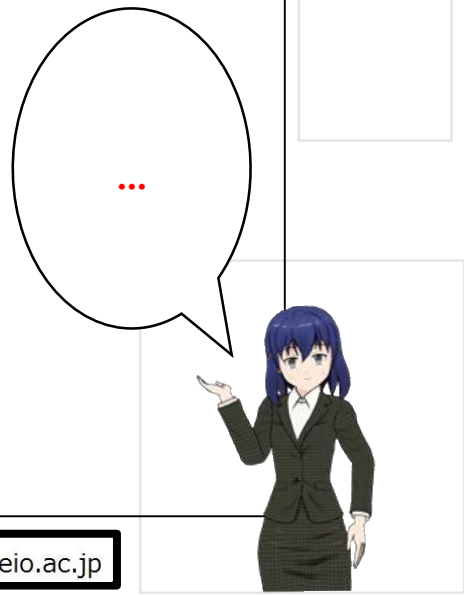


iijima@ae.keio.ac.jp

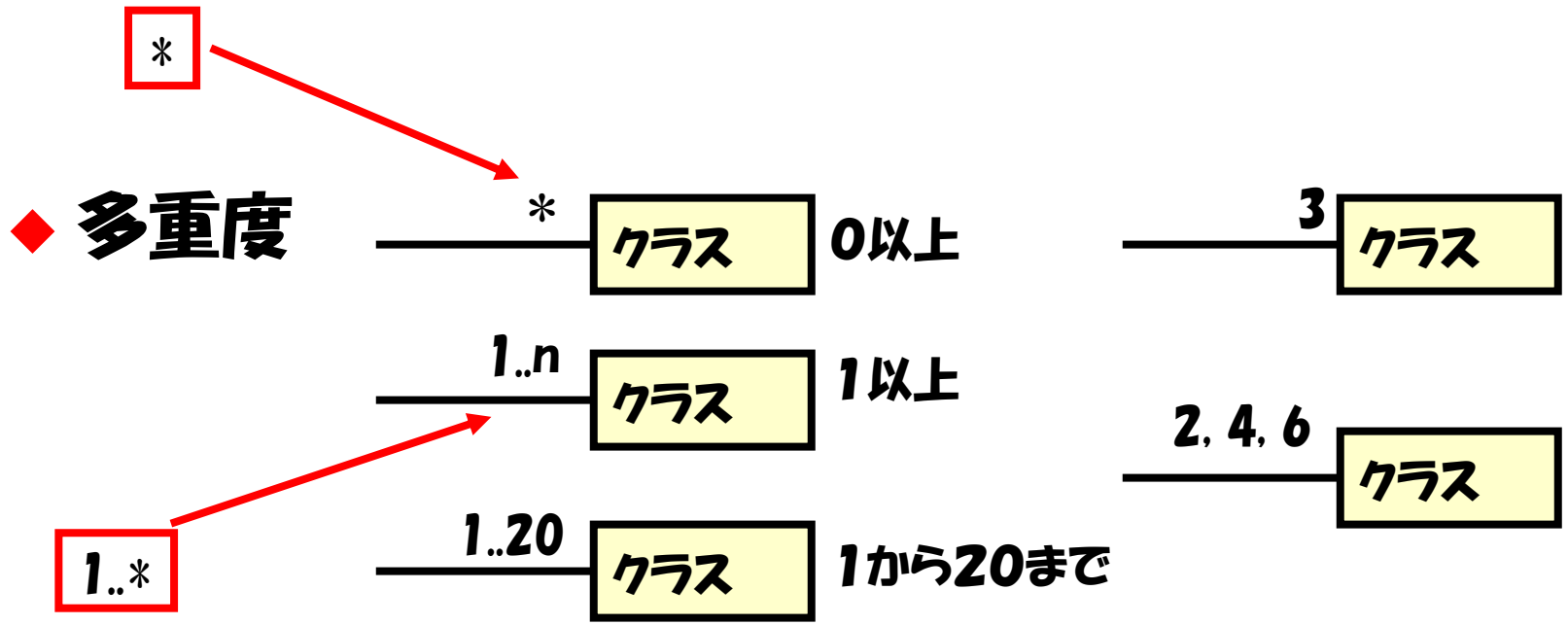
一般の関連 (Association)

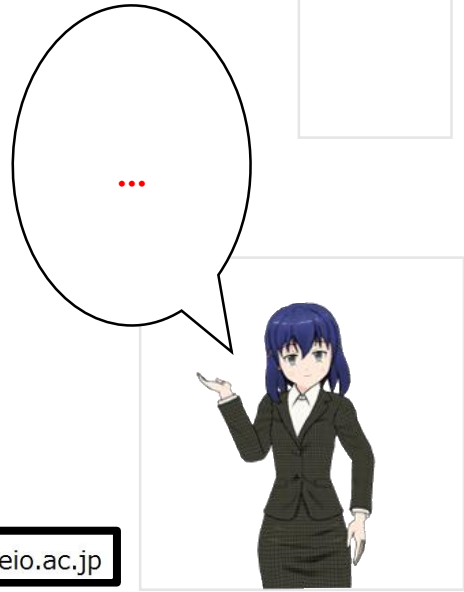
- 関連
  - 異なったクラスのオブジェクト間の関係
  - link: 関連の個々のインスタンス
  - 方向，役割（role)，多重度 (multiplicity），{ordered} 指定





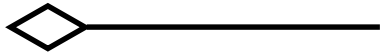
多重度：関連につける属性（対応関係）



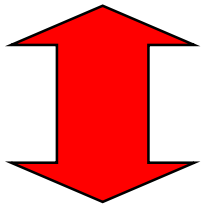


特殊な関連：部分-全体関係

部分-全体(part-whole)関係 { 集約関係(Aggregation)
組立構造(composition)



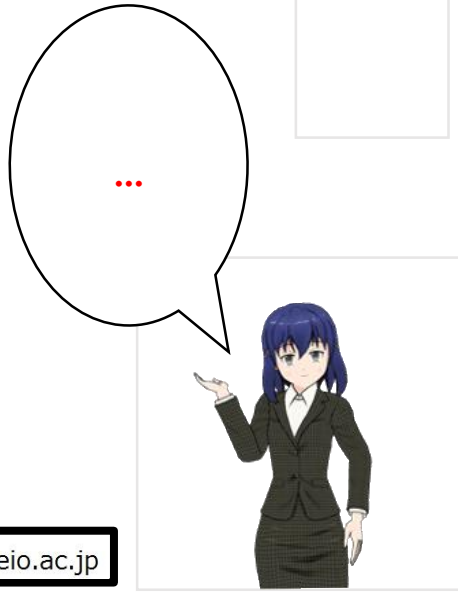
集約(aggregation)



容器と中身

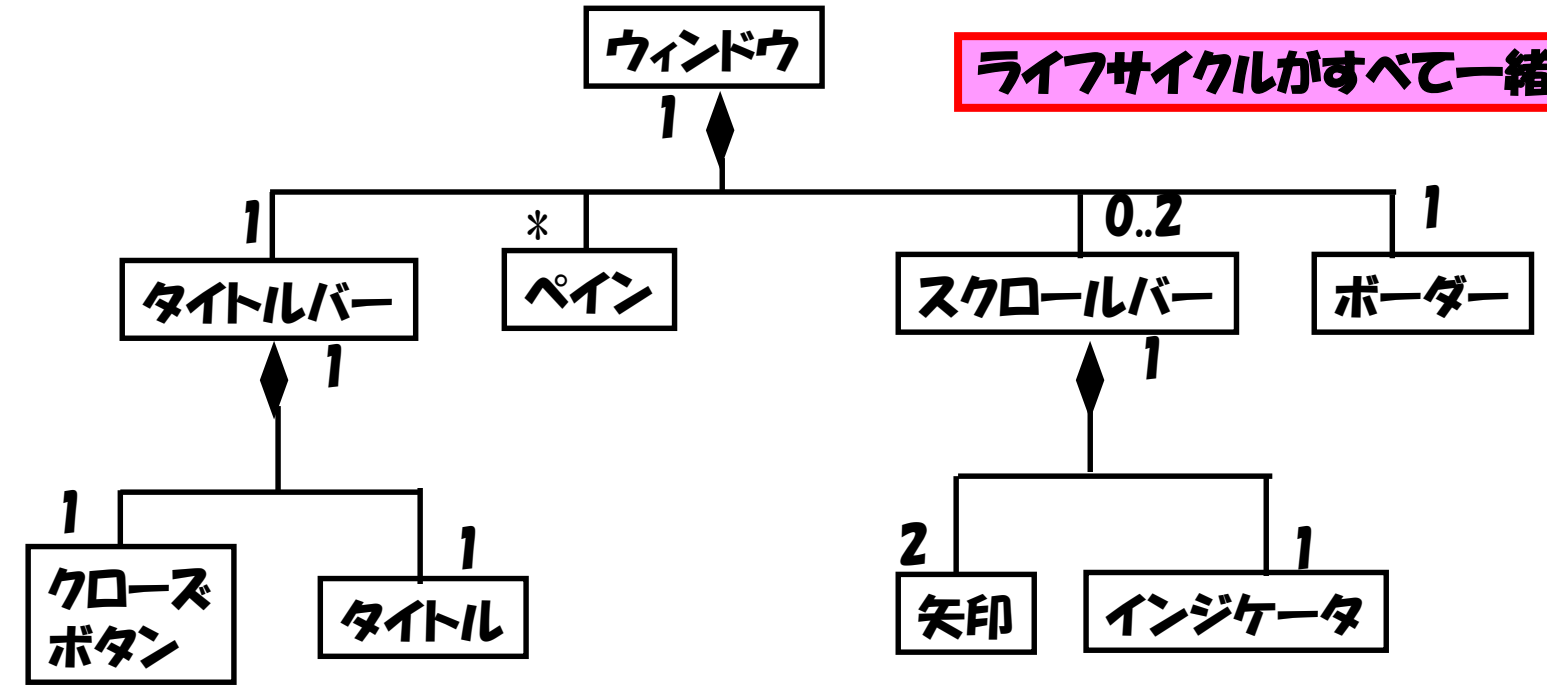
組立(composition)

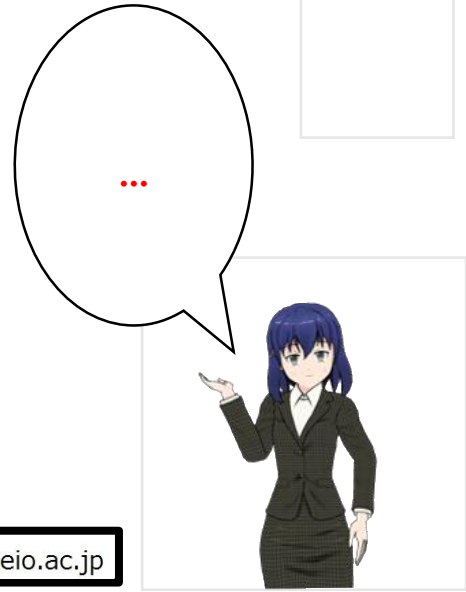
ライフタイム(生存期間)が一致
生成／削除の責任の所在



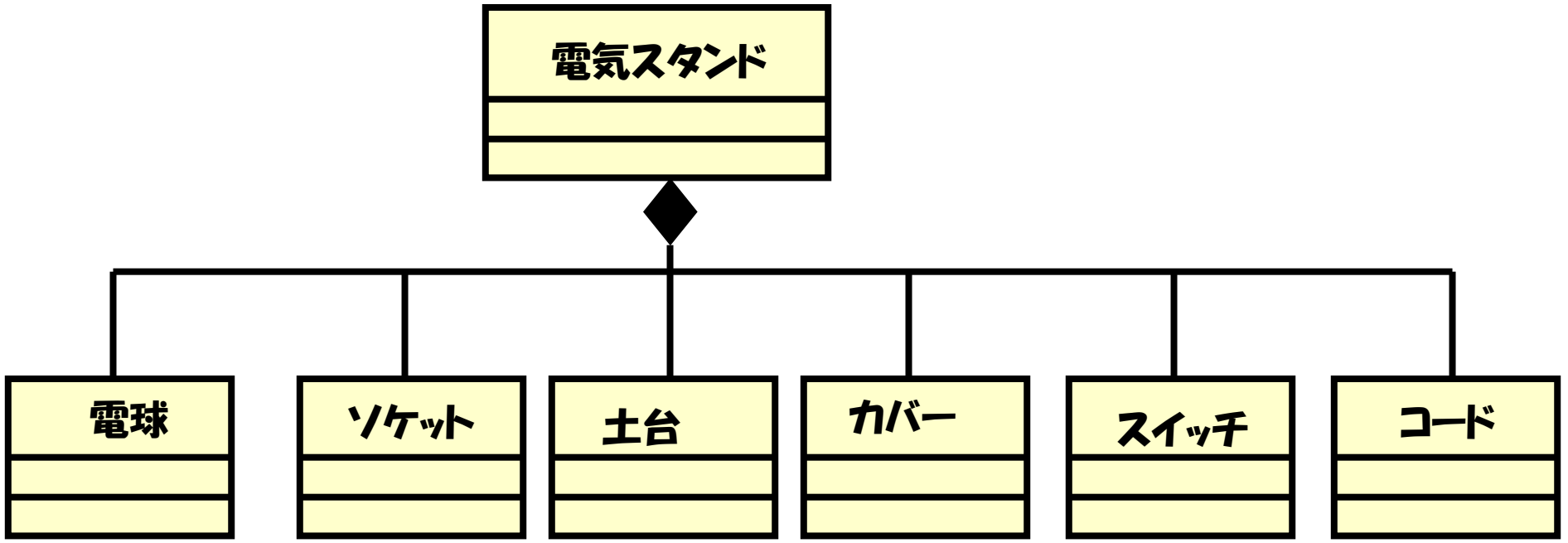
特殊な関連：部分-全体関係（組立構造）

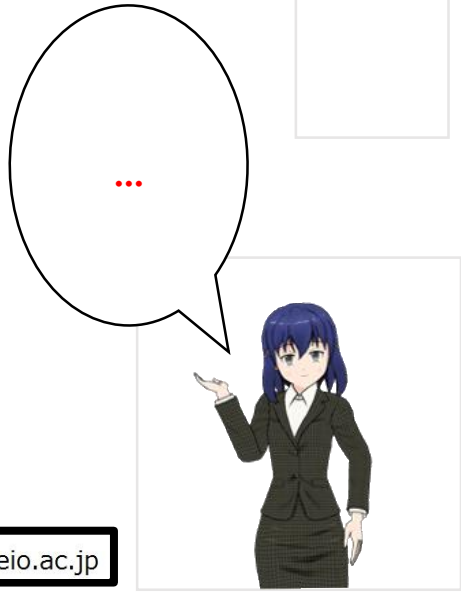
部分-全体(part-whole)関係 { 集約関係(Aggregation)
組立構造(composition)





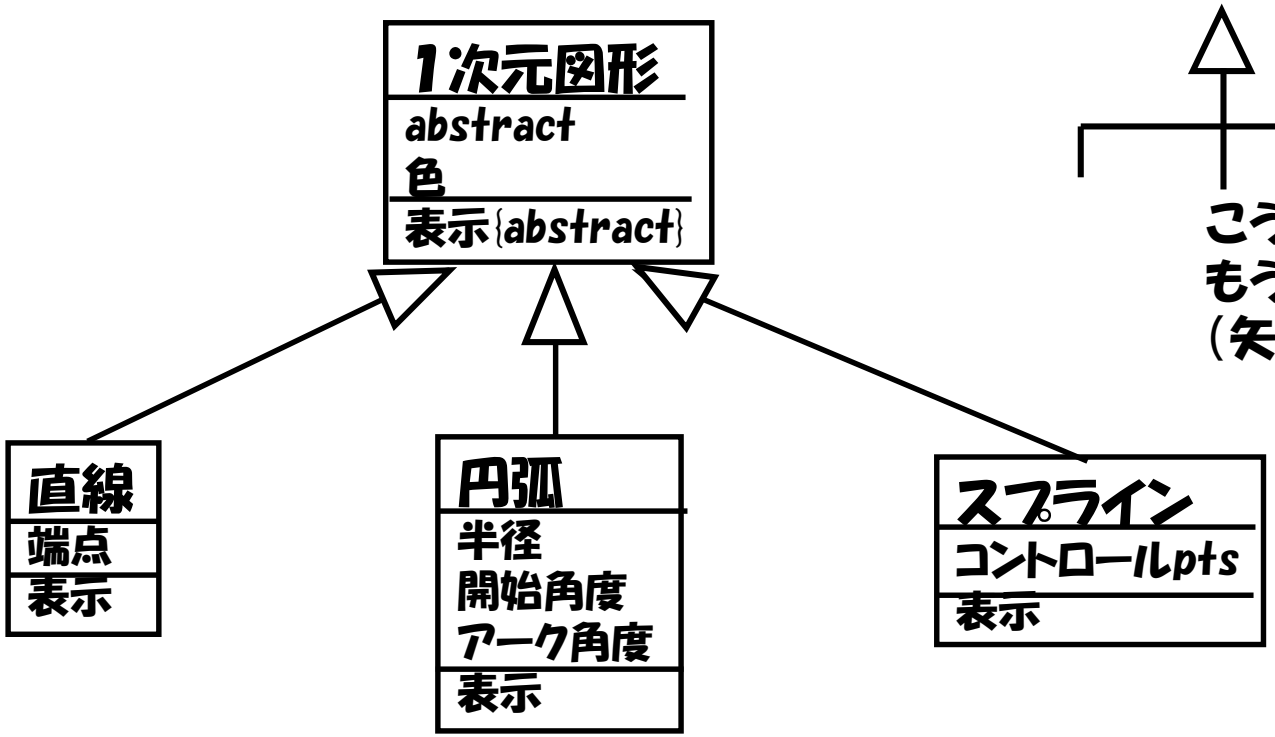
組立構造の例

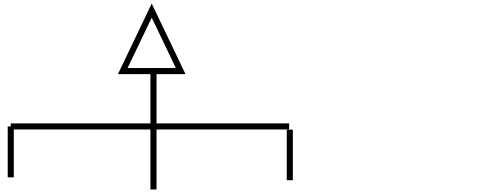




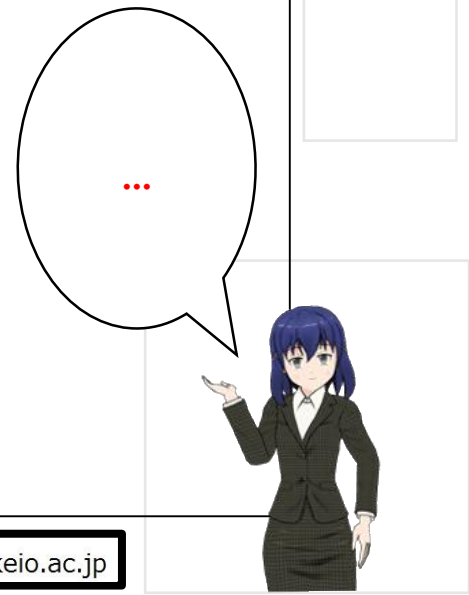
クラス間の関連：継承 (inheritance)

- 一般化ー特殊化　generarization-specialization
- 抽象クラス abstract
  - 直接はインスタンスを持たないクラス.
  - 具体的なサブクラスをまとめるのに使われる.



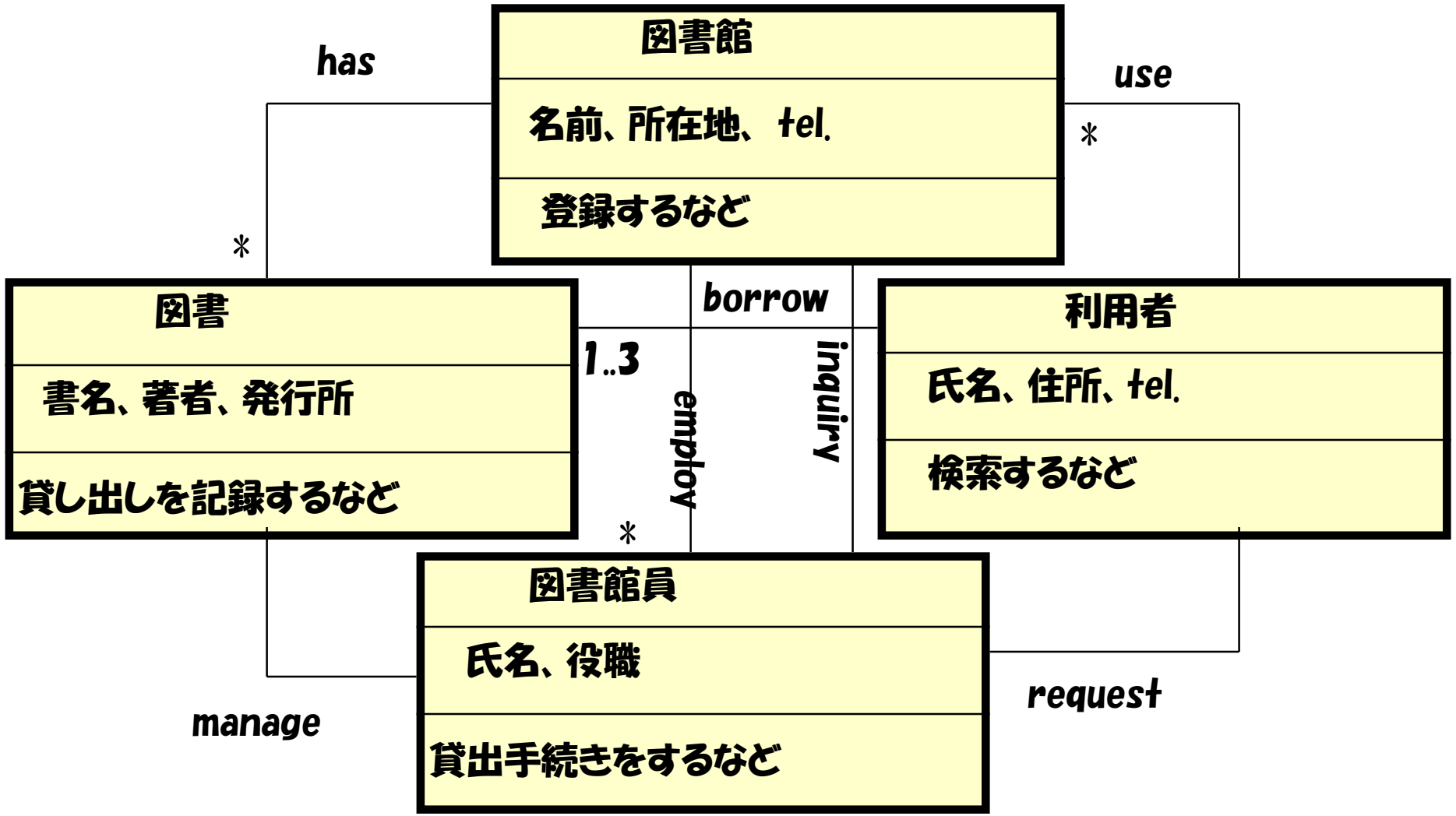


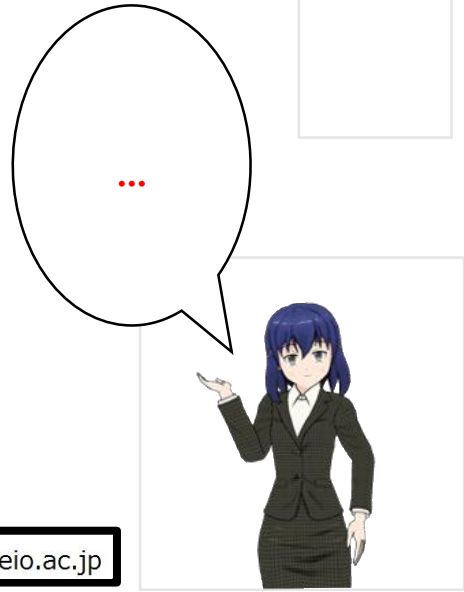
こういう書き方は
もうあまりしない
（矢線で統一）



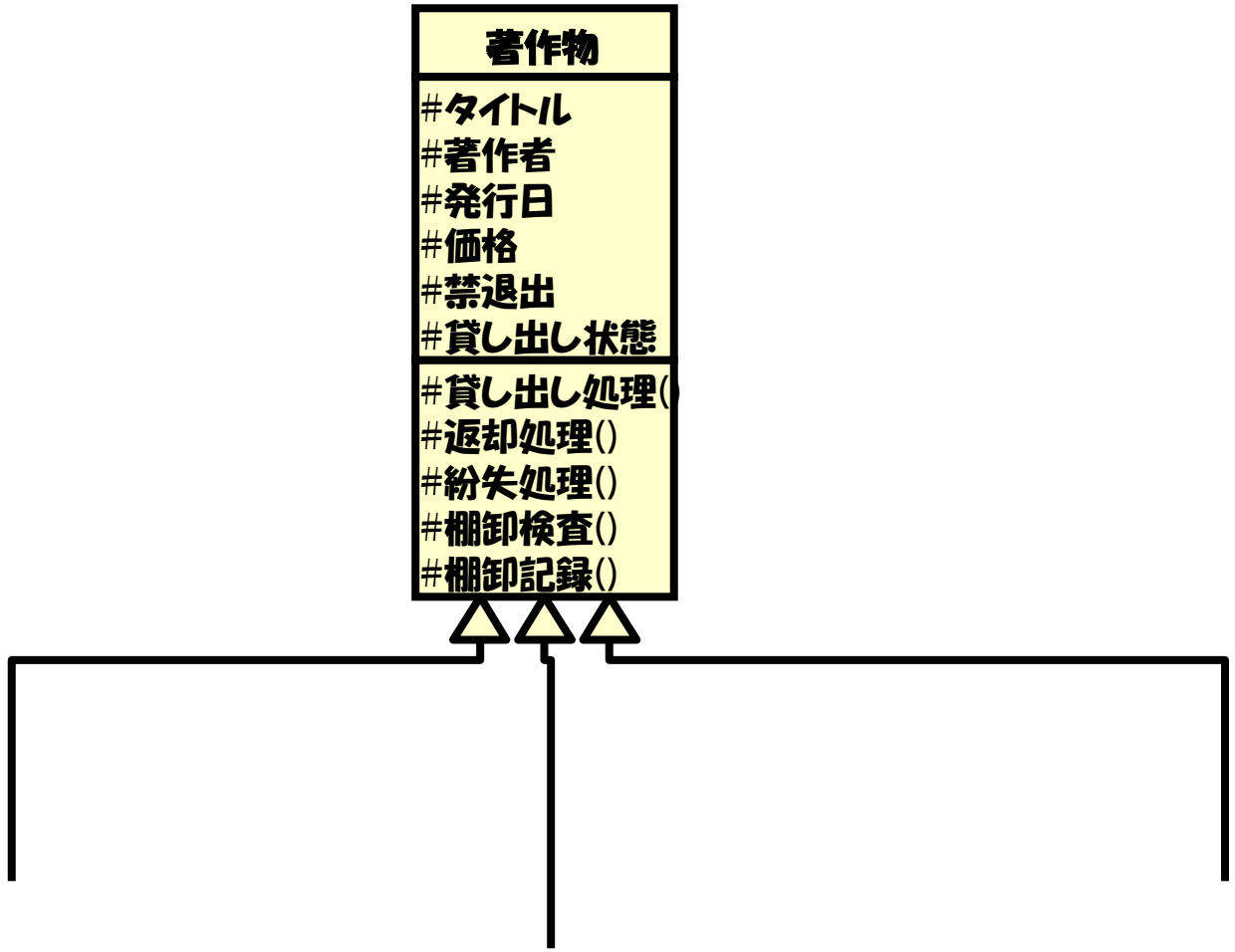
iijima@ae.keio.ac.jp

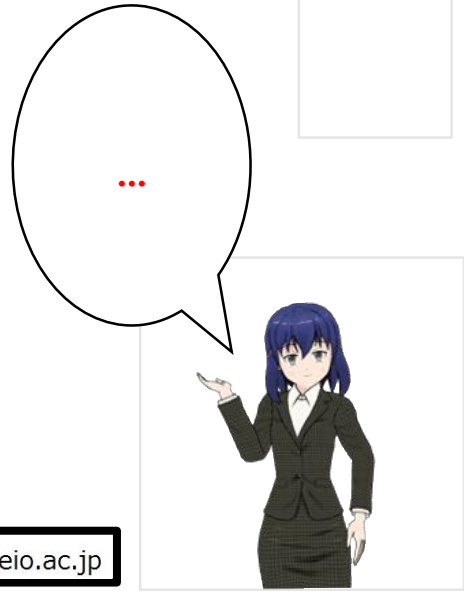
図書館のクラス図


図書館
名前、所在地、 tel.
登録するなど
has
use
*
*
図書
書名、著者、発行所
貸し出しを記録するなど
borrow
1..3
employ
inquiry
利用者
氏名、住所、tel.
検索するなど
*
図書館員
氏名、役職
貸出手続きをするなど
manage
request


…

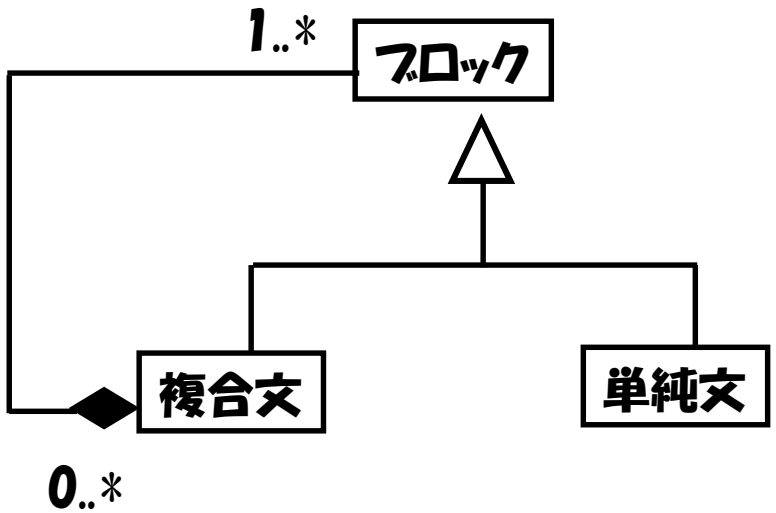
設計クラス図の例（一部）

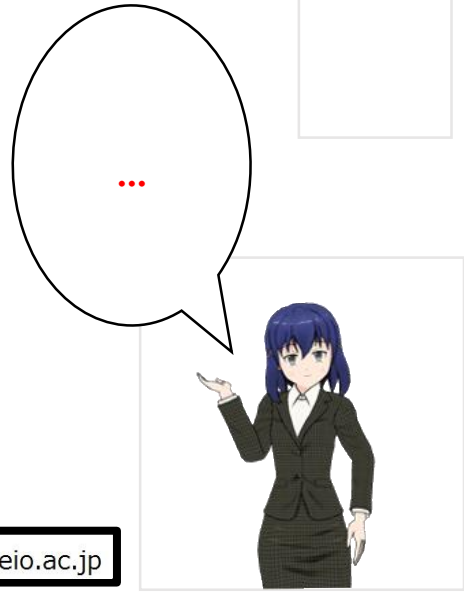




iijima@ae.keio.ac.jp

再帰的構造の例

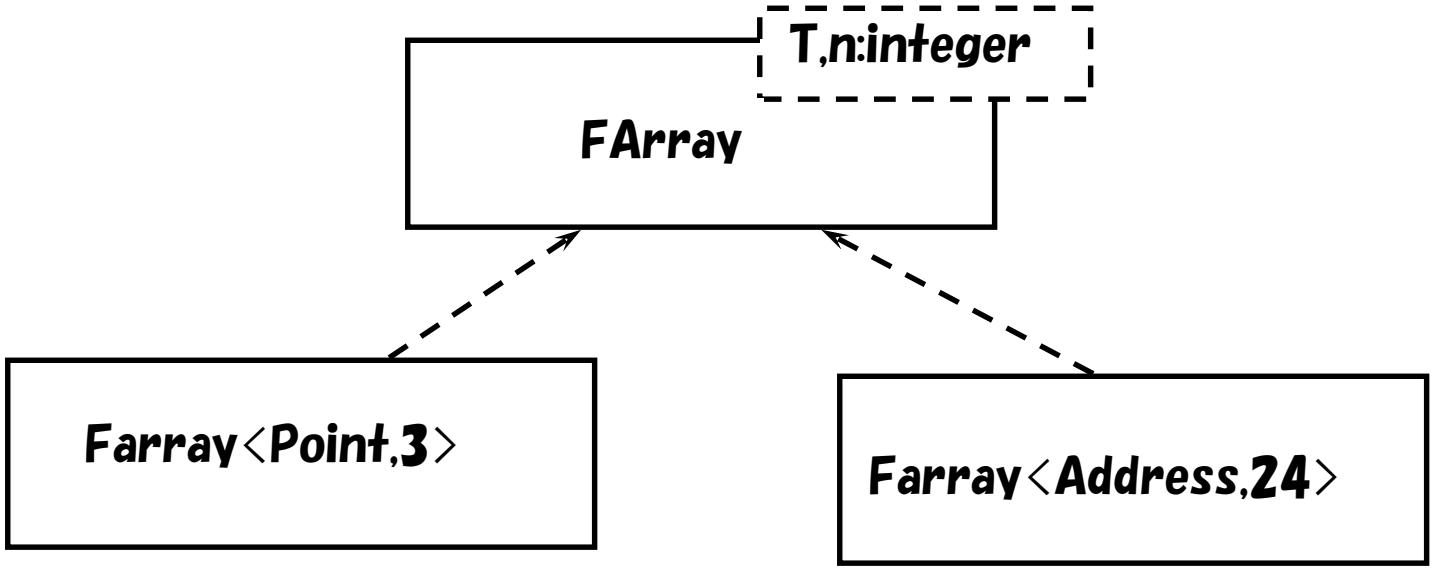


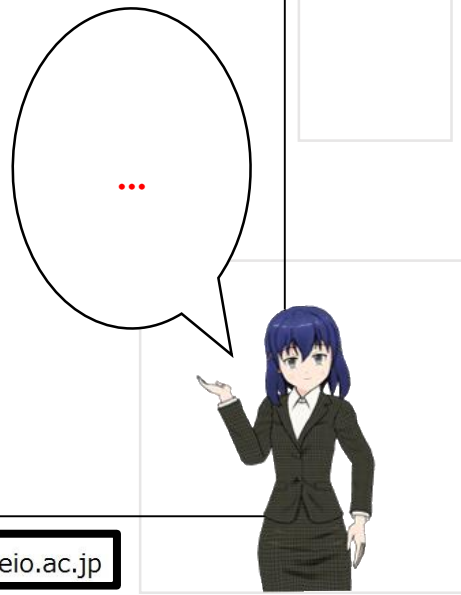


iijima@ae.keio.ac.jp

パラメータ化クラス

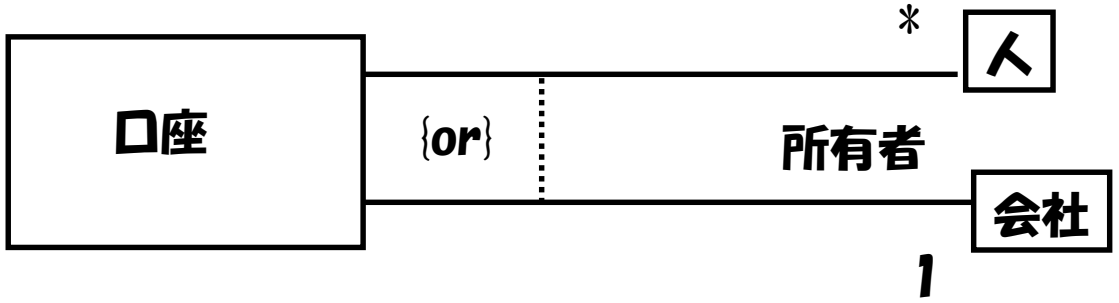
- パラメータ化クラス
  - C++にあるテンプレートのこと
  - インスタンス化関係

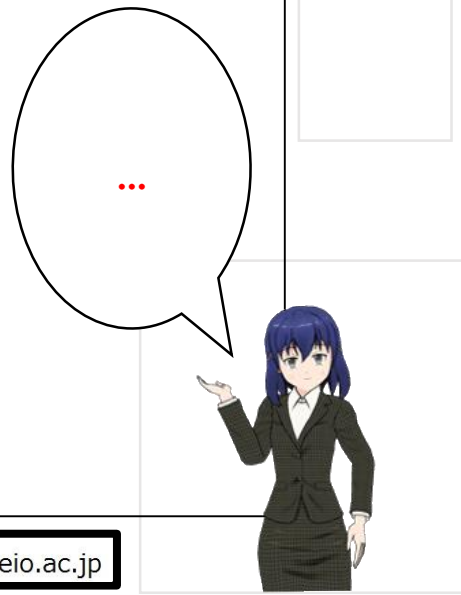




Or−関連

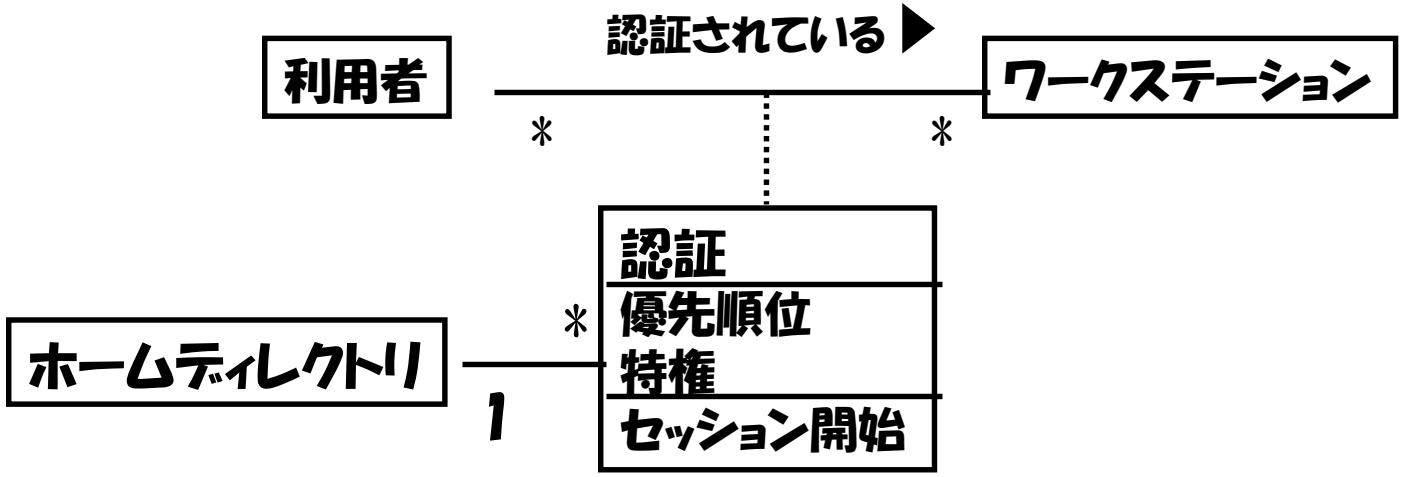
- Or−関連

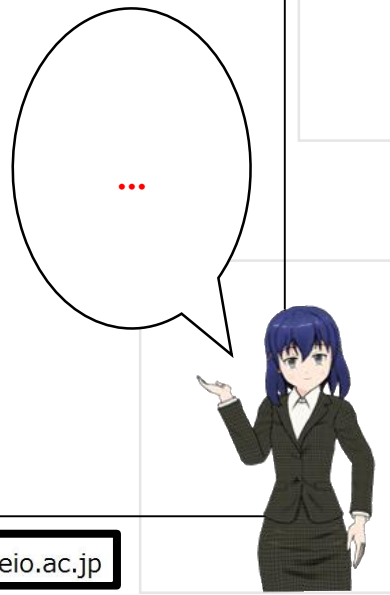




関連クラス

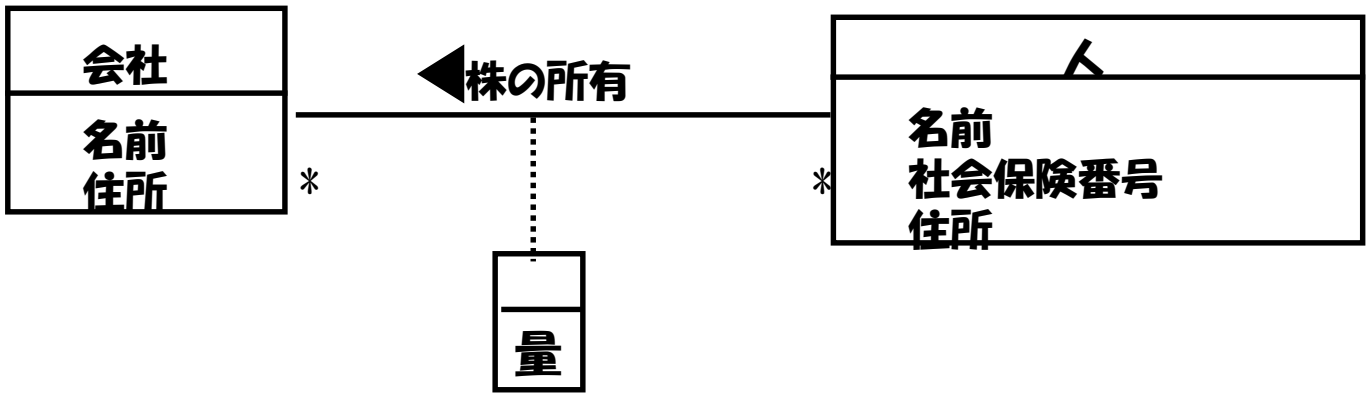
- 関連クラス
  - 属性を持った関連
  - 関連名とクラス名は同じ

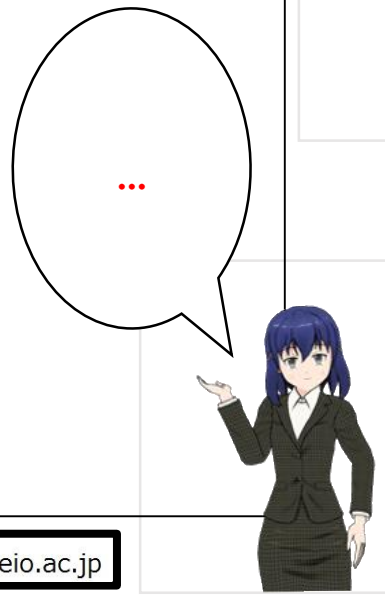




関連属性

- 関連クラスのクラス名は明記しなくてもよい

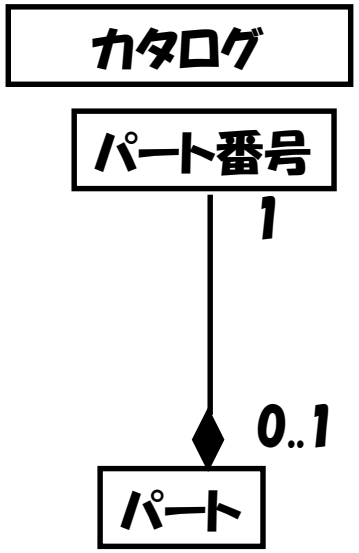


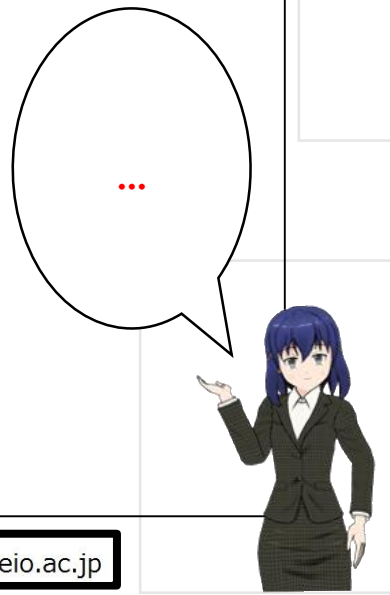


iijima@ae.keio.ac.jp

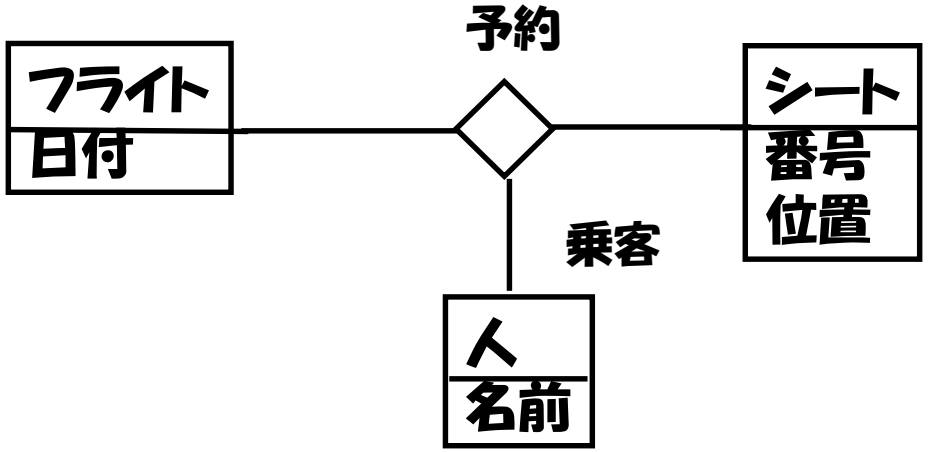
限定子付き関連

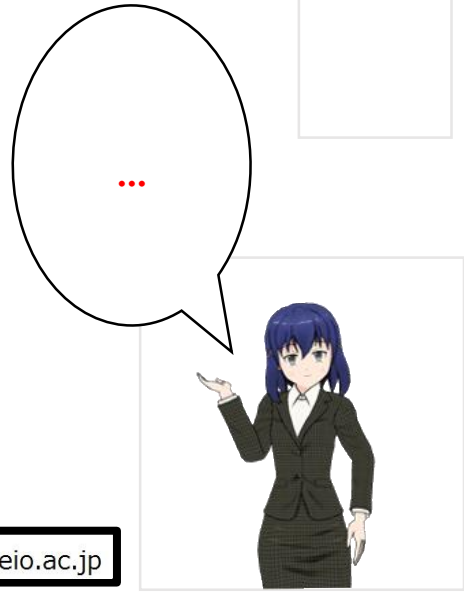
- 複合キーとなる





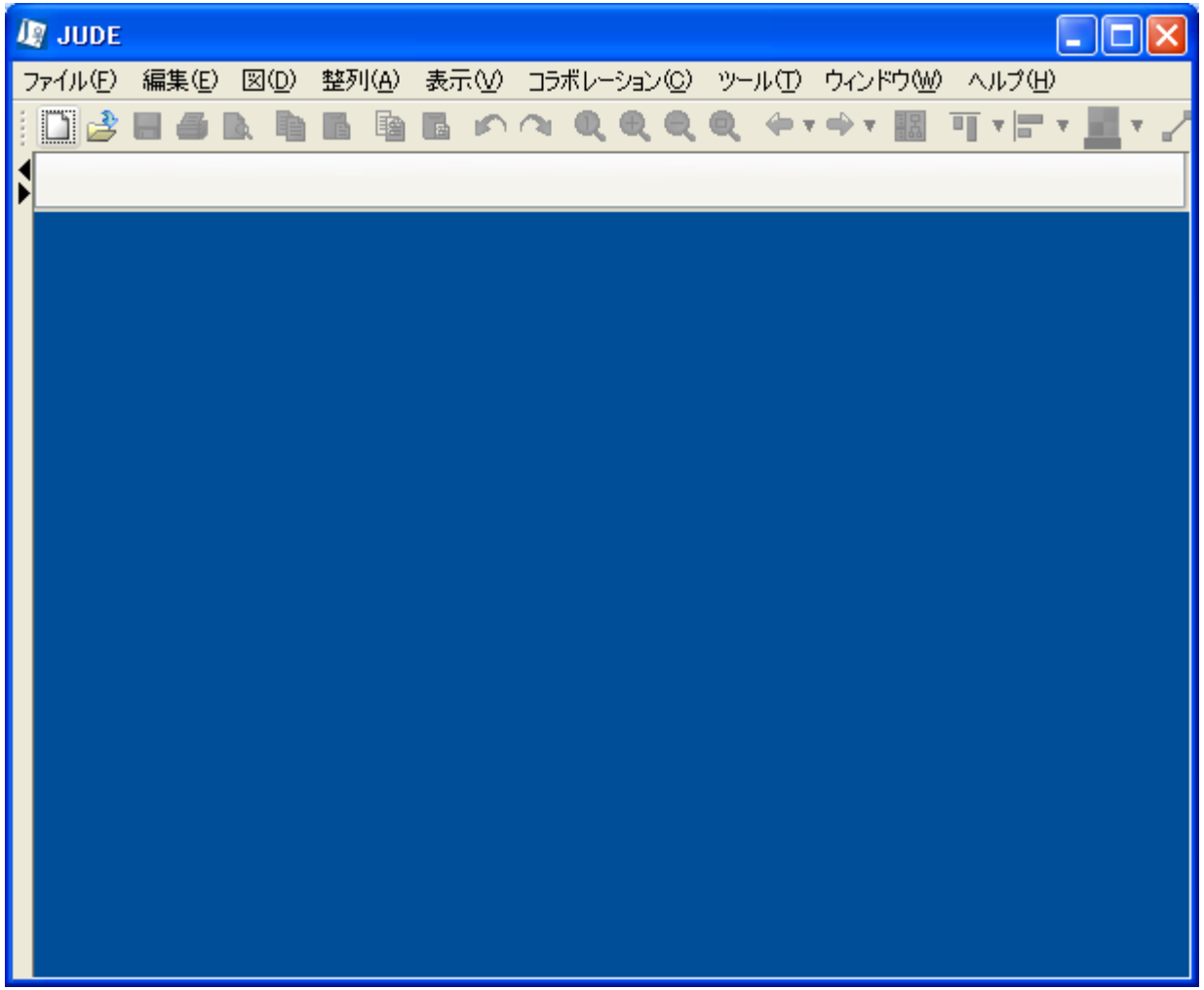
n 項関連

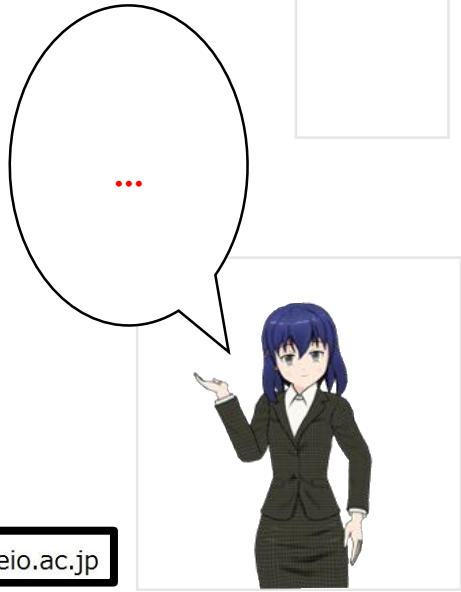




iijima@ae.keio.ac.jp

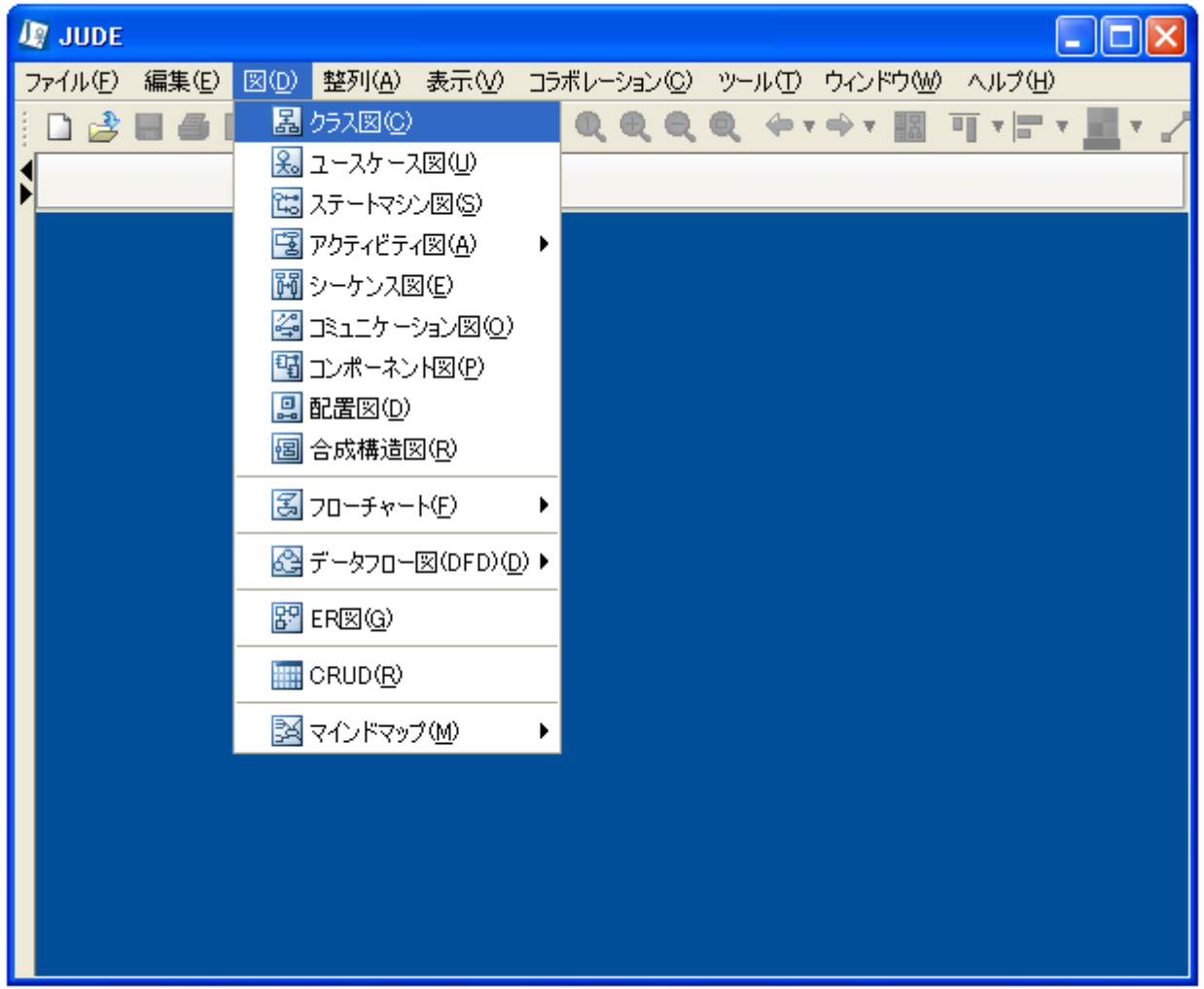
UMLエディタ: Astah*の使い方 (起動直後)

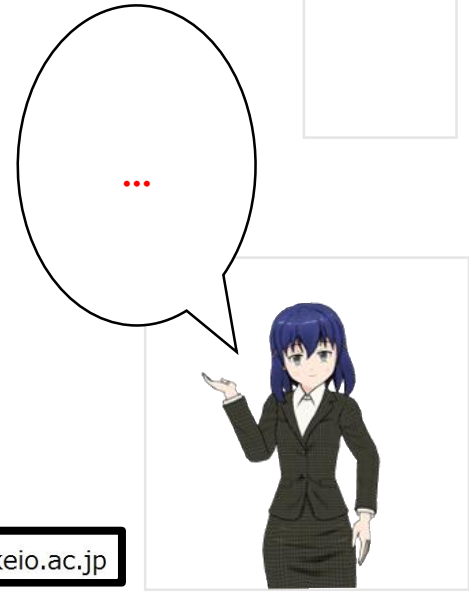




iijima@ae.keio.ac.jp

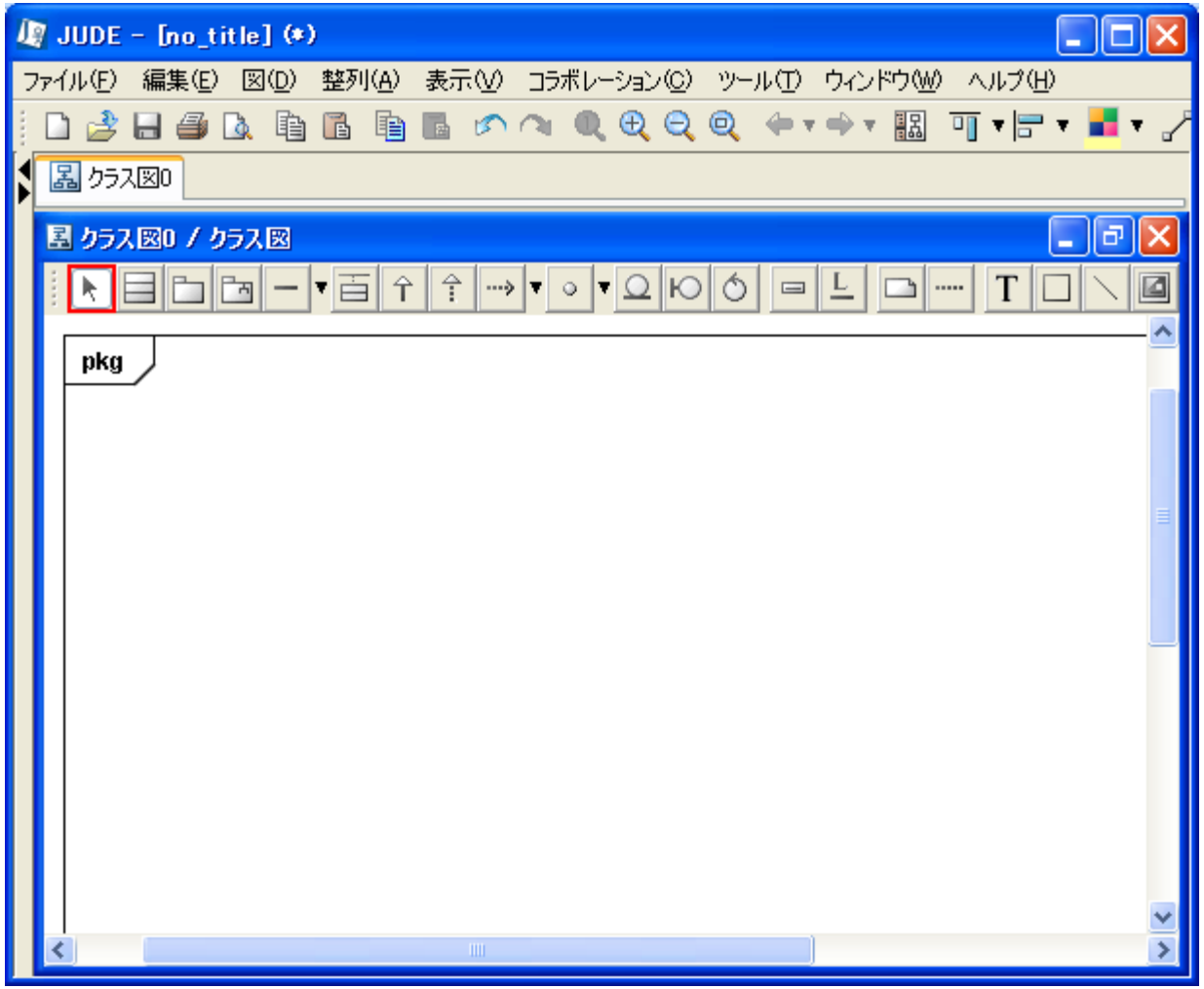
UMLエディタ: Astah*の使い方(クラス図)

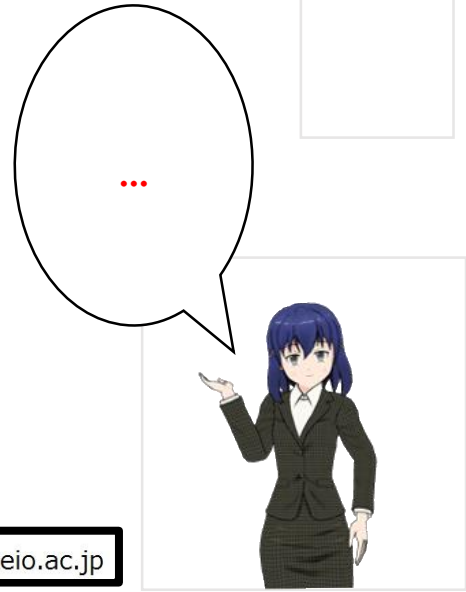




iijima@ae.keio.ac.jp

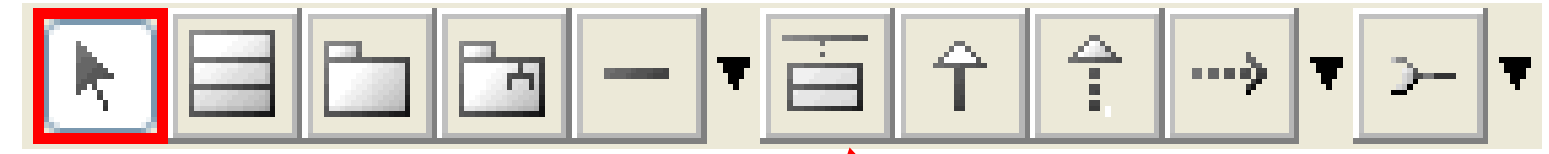
UMLエディタ: Astah*の使い方(クラス図エディタ)





iijima@ae.keio.ac.jp

UMLエディタ: Astah*の使い方（主な図要素コマンド）



選択

クラス

パッケージ

サブシステム

関連

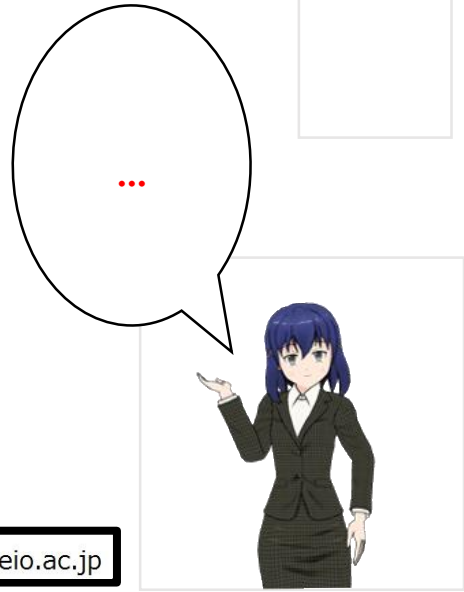
関連クラス

汎化

実現

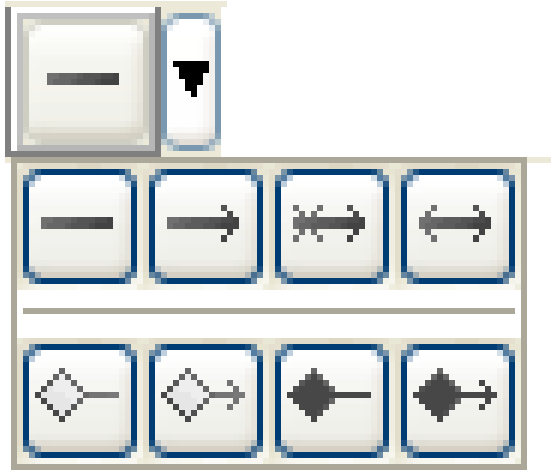
依存

要求I/F

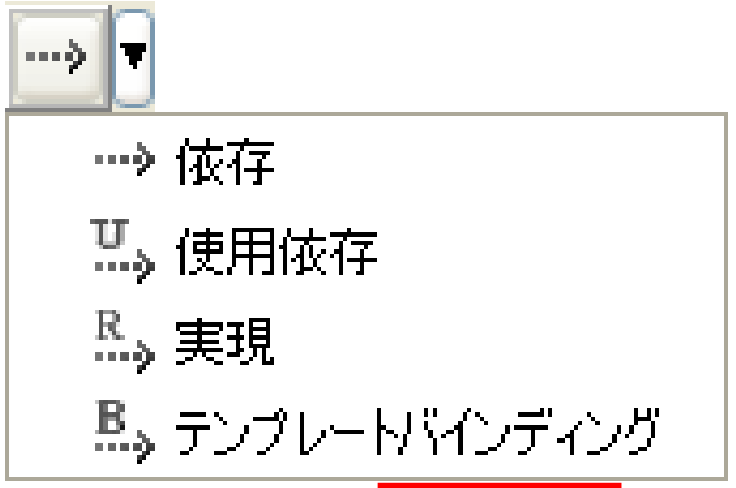


iijima@ae.keio.ac.jp

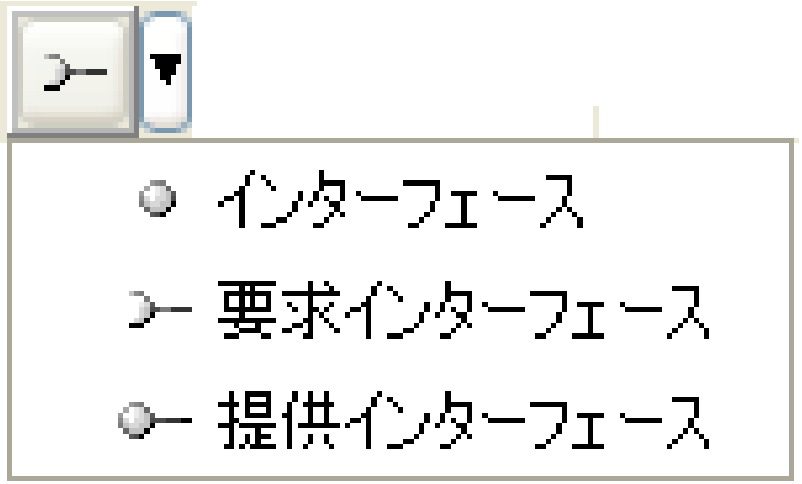
UMLエディタ: Astah*の使い方(サブメニュー)



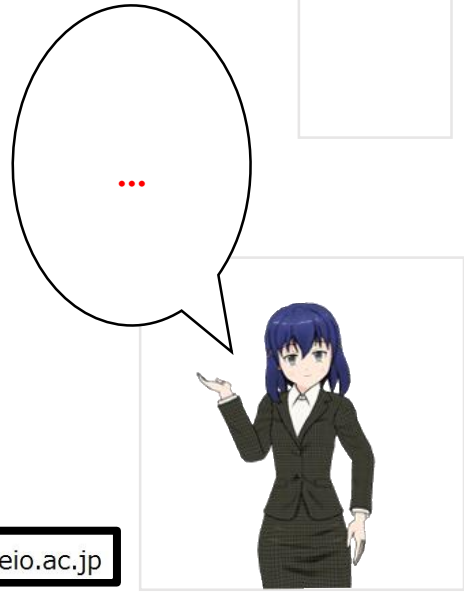
関連



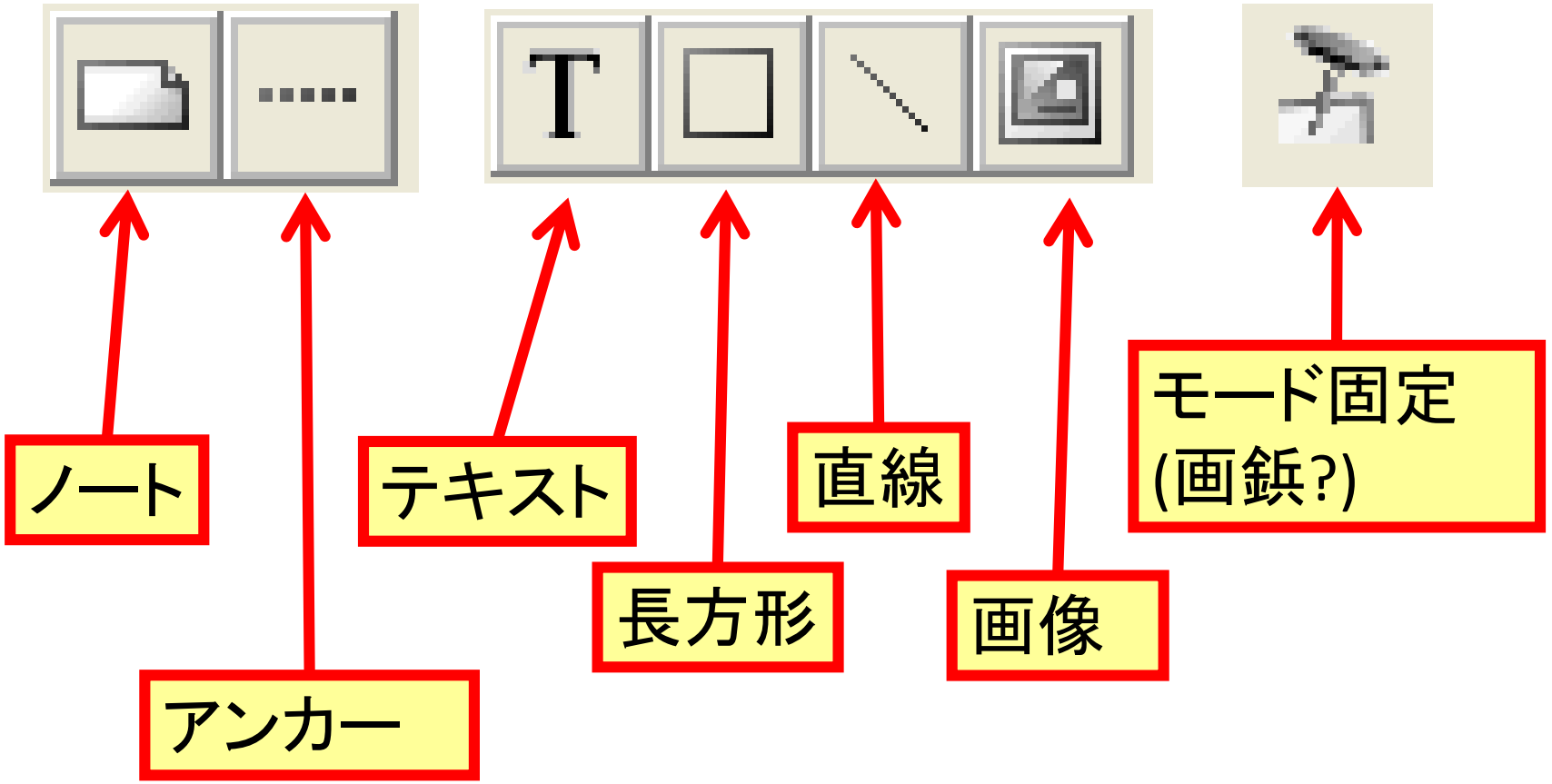
依存

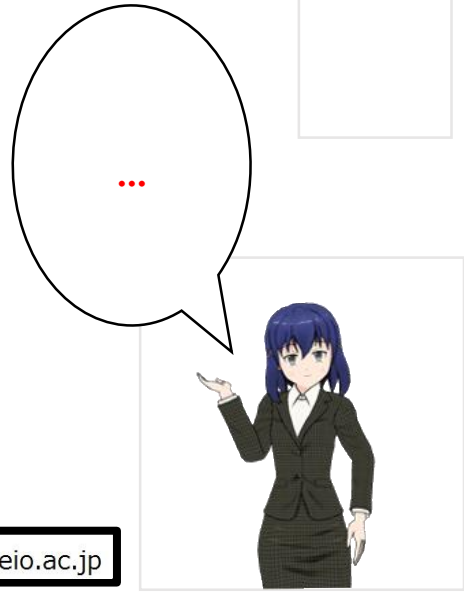


要求I/F

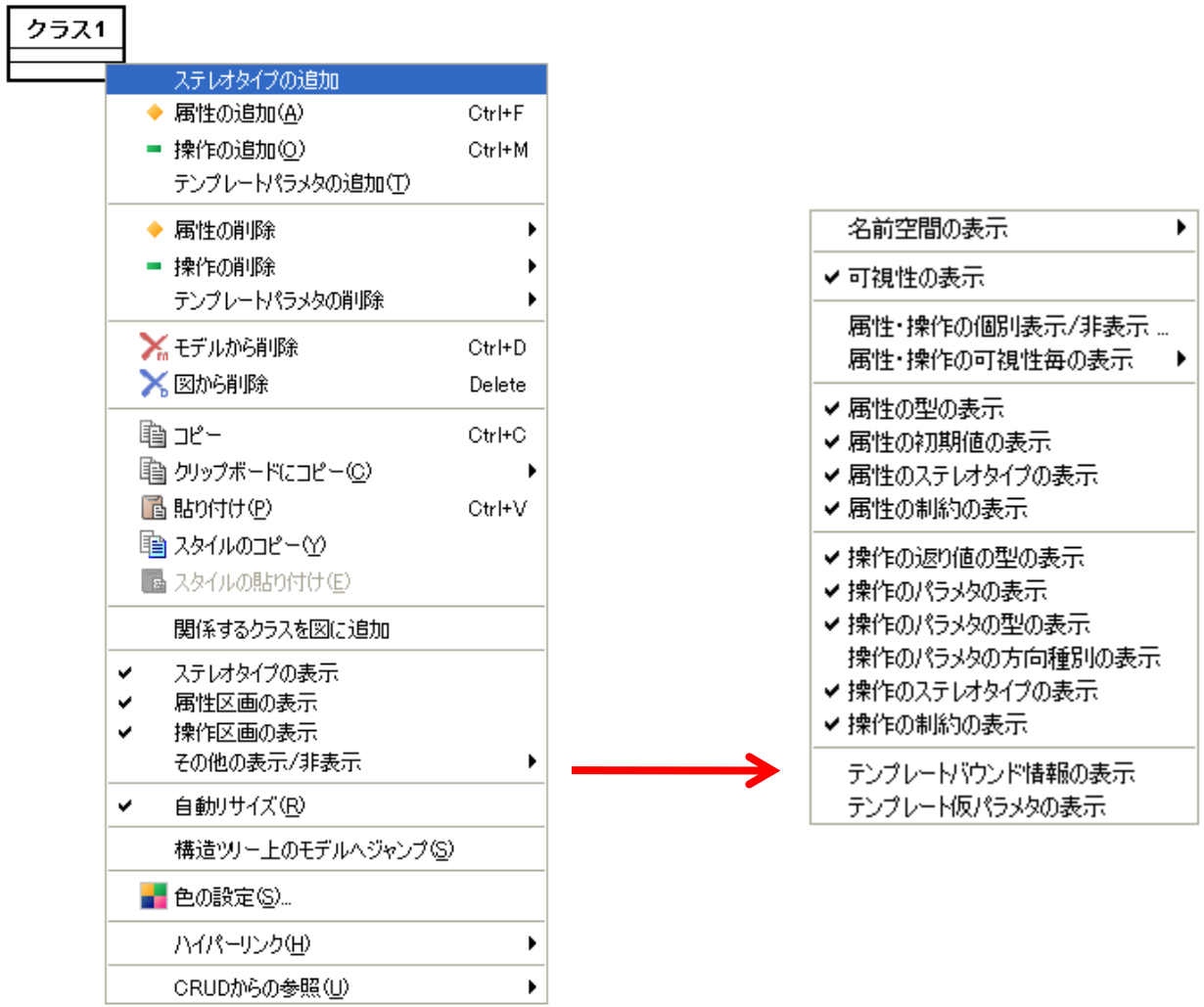


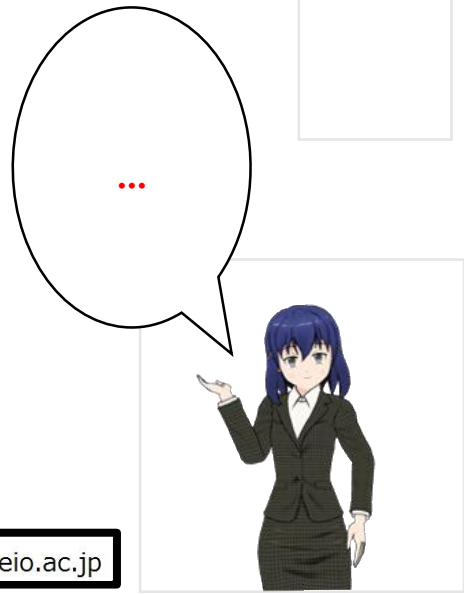
UMLエディタ: Astah*の使い方 (共通的な図要素等)



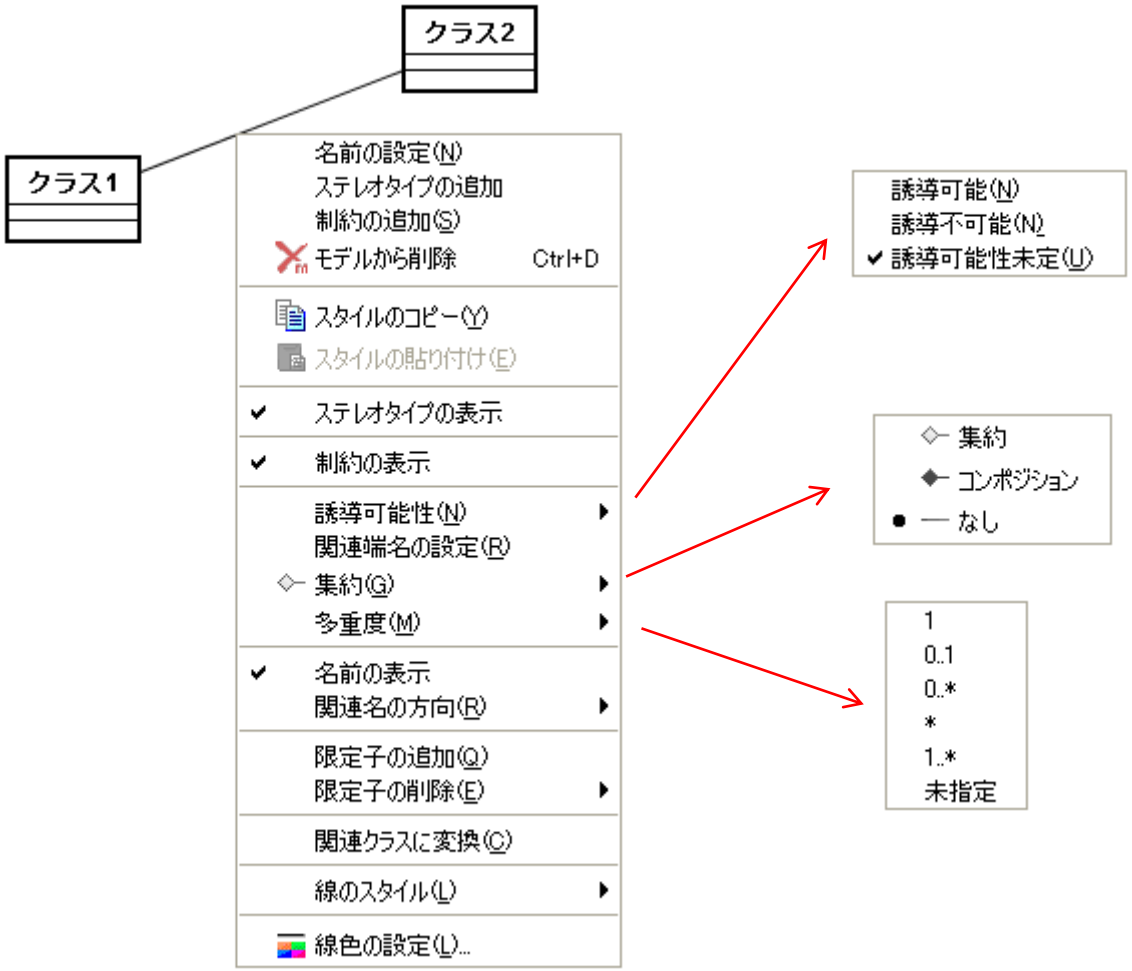


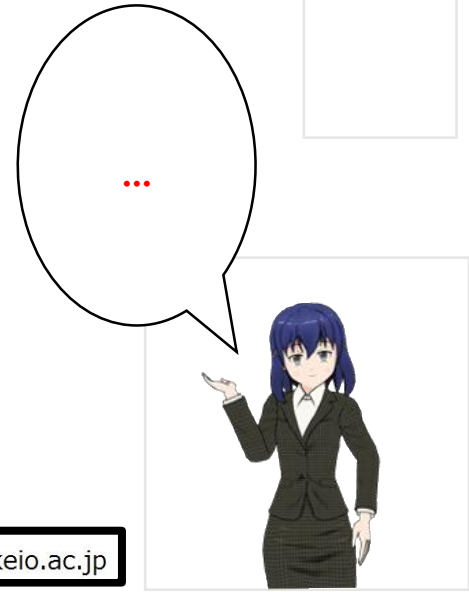
UMLエディタ: Astah*の使い方（クラス:ポップアップメニュー）





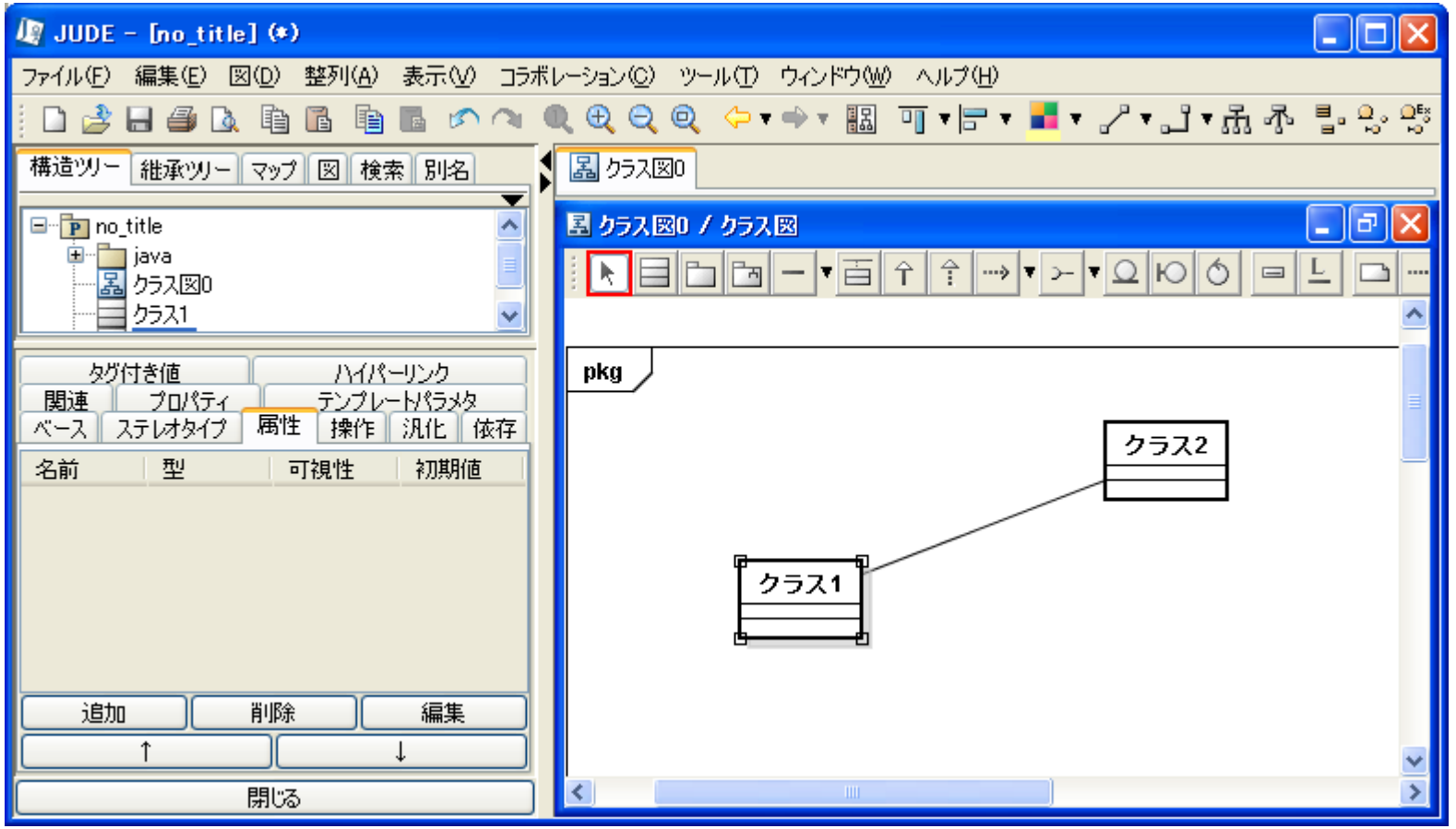
UMLエディタ: Astah*の使い方 (関連:ポップアップメニュー)

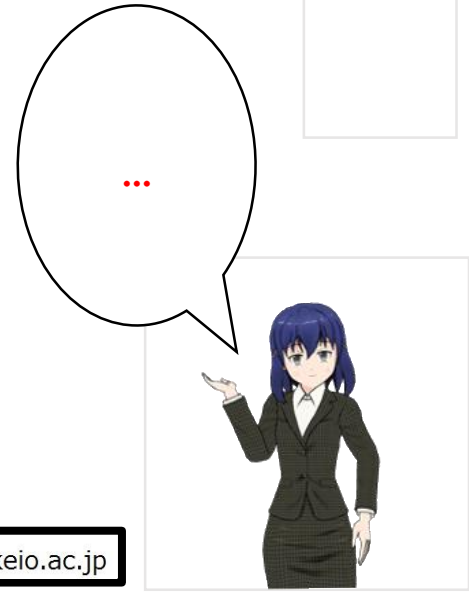




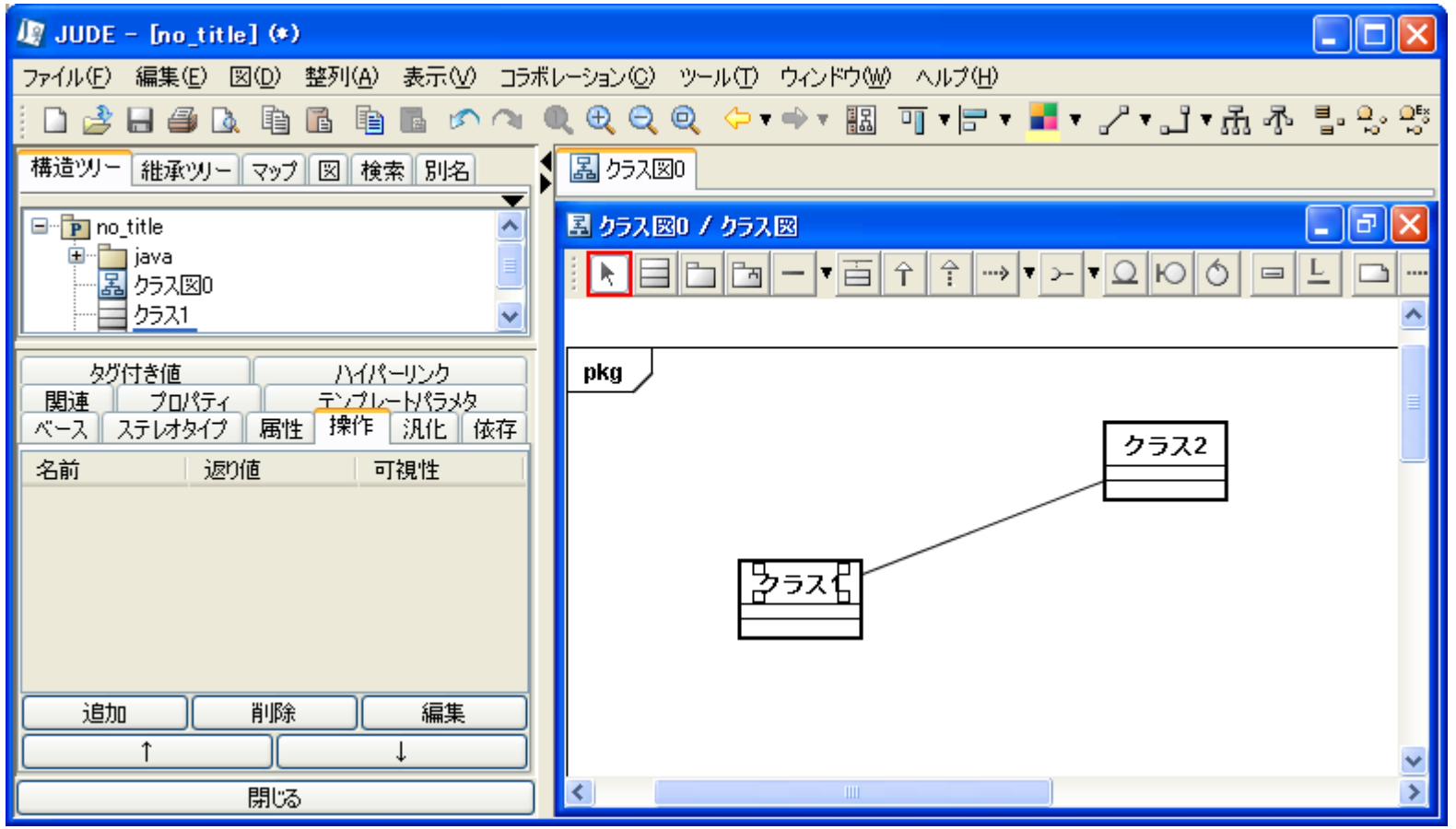
iijima@ae.keio.ac.jp

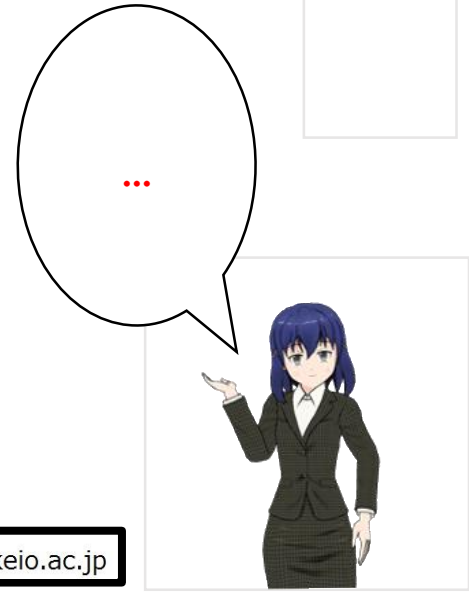
UMLエディタ: Astah*の使い方(クラスの属性)



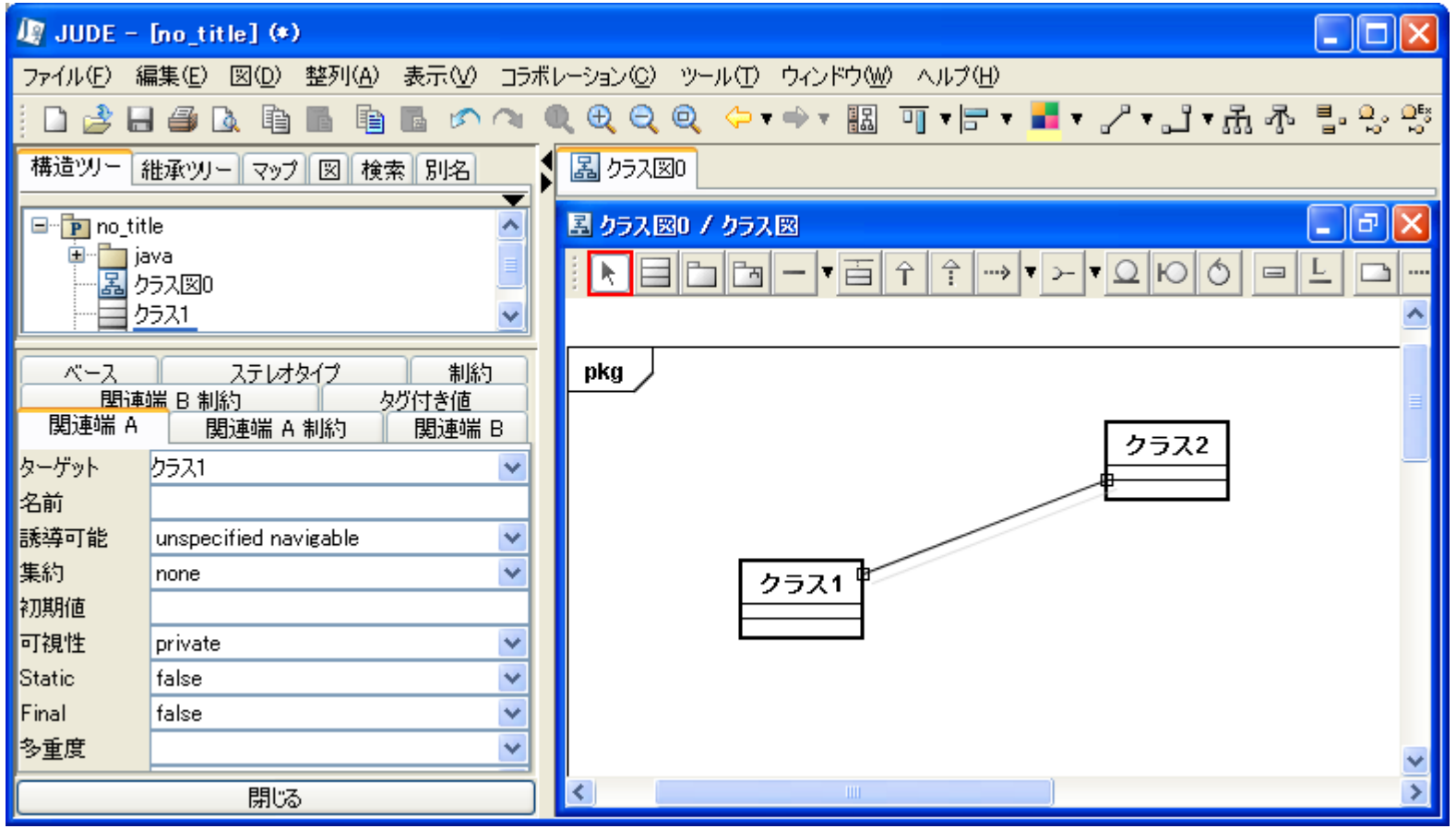


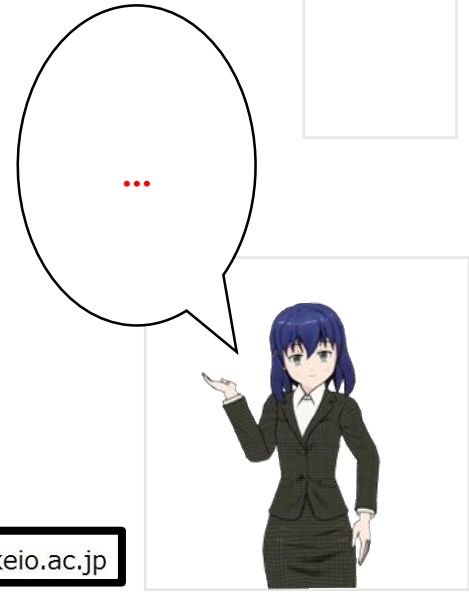
UMLエディタ: Astah*の使い方 (クラスのメソッド=操作)



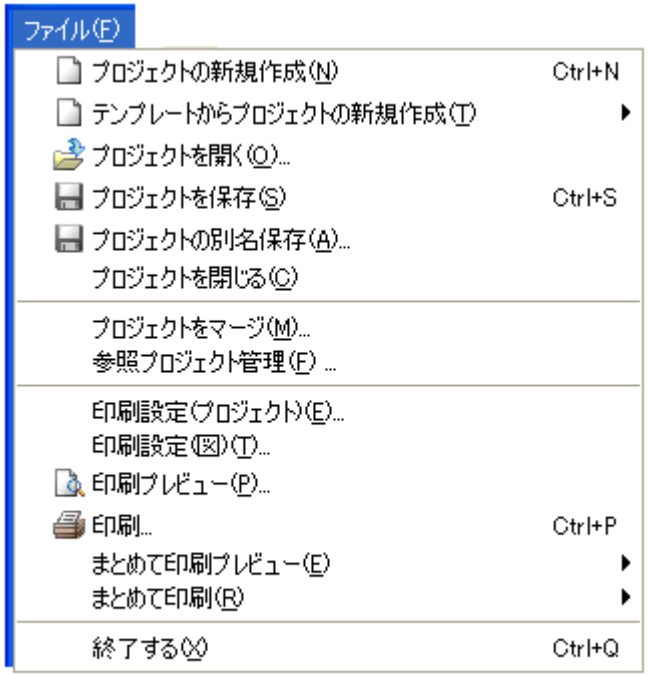


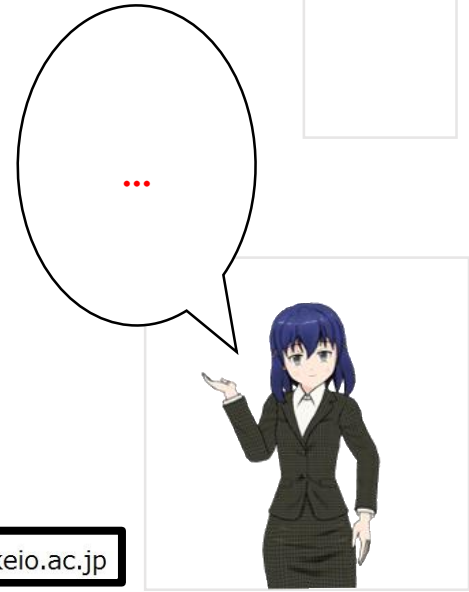
UMLエディタ: Astah*の使い方（関連のプロパティ）



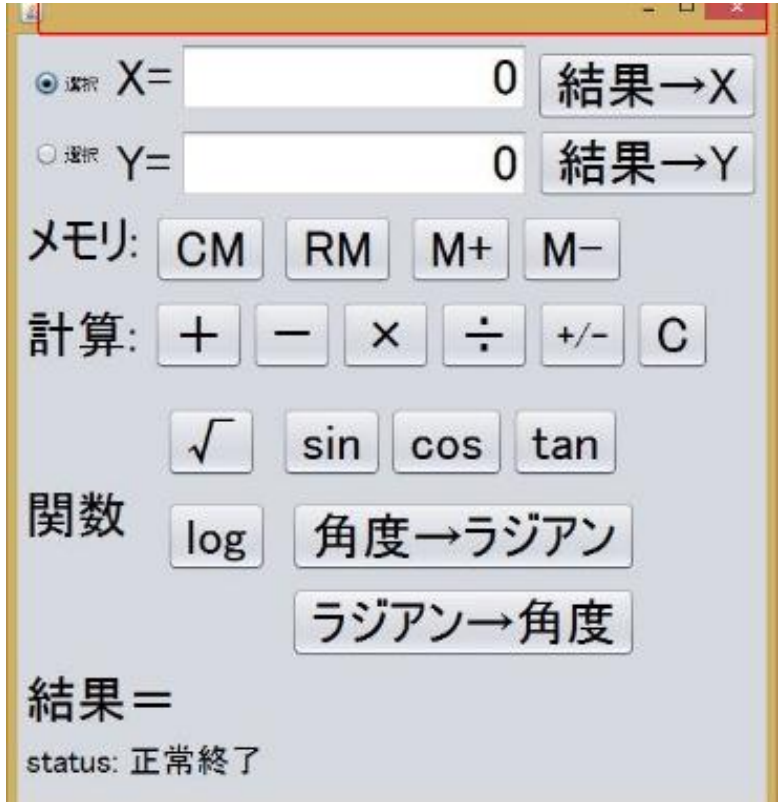


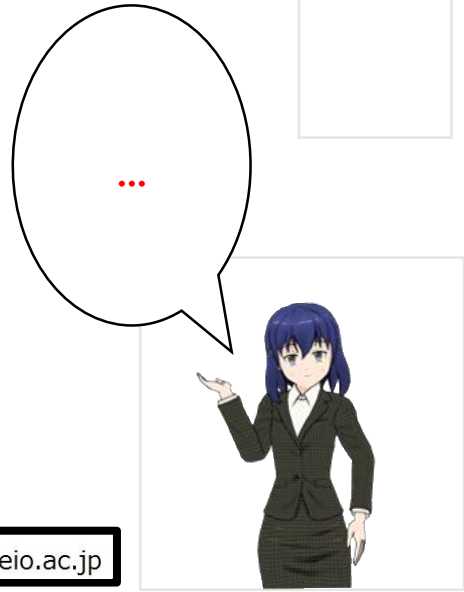
UMLエディタ: Astah*の使い方（ファイル・メニュー）





クラス図を作ってみよう





iijima@ae.keio.ac.jp