Software Design
2. Modeling of Structure

Natsuko Noda
nnoda@shibaura-it.ac.jp

ソフトウェア設計論
2. 構造のモデリング

野田 夏子
nnoda@shibaura-it.ac.jp

Today's Topics

- Overview of modeling
- Modeling of structure
  - Class diagram
  - Object diagram

本日のお題

- モデリングの概要
- 構造のモデリング
  - クラス図
  - オブジェクト図

Overview of modeling

モデリング概要

cf. SE-J, 2.1

What is software modeling?

- Software modeling is the task of abstracting an information system and its related things and expressing them in a formal notation.
- Various targets of software modeling:
  - Real world related to an information system
  - Specification of an information processing system
  - Anything related to software design
  - Anything related to software implementation
  - etc.
- Software modeling is utilized in various processes of software development.
  - It is important in designing software.
    - But, software modeling ≠ designing software.

ソフトウェアモデリングとは？

- 情報システムに関わる対象を抽象化し、それを形式性のある記法で表現する作業
- ソフトウェアモデリングの対象は様々
  - 情報システムが関わる現実世界
  - 情報処理システムの仕様
  - ソフトウェアの仕様に関わるもの
  - ソフトウェアの設計に関わるもの
  - ソフトウェアの実装に関わるもの
  - etc.
- ソフトウェア開発の様々な工程で利用される
  - 設計においても重要
    - ただし、モデリング＝設計ではない

Why is software modeling useful?

- It provides basic viewpoints and description methods.
  - Although many types of artifacts of software development (programs, specification, design, ...) exist, there are patterns or common types of viewpoints and description methods that are used.
- It lightens scale and/or complexity of software.
  - Software is getting larger and more complicated. Using modeling, that is the abstraction of such software, we can describe specific aspects of software and make the software easy to understand.
- It provides an accurate way to describe software.
  - We can describe software accurately using formal notation.
  - When we use such formal description, we may find the parts that are inaccurate and/or vague.

ソフトウェアモデリングが有用な理由

- 基本的な視点や記述方法の提供
  - ソフトウェア開発中に扱うものは、プログラム、要求、仕様、設計など様々であるが、用いられる視点や記述方法には類型がある
- 規模や複雑さの軽減
  - ソフトウェアは大規模、複雑化しているが、モデリングにより特定の側面をコンパクトに表現し、理解を容易にすることができる
- 正確な表現方法の提供
  - 形式を決めた表現により、正確な記述が可能になる
  - モデルを書こうとすることにより、不正確な部分、あいまいな部分に気づく

Software modelling

- model – verb and noun
  - verb: make models in the meaning in the previous page.
- modelling: activities of making models.

- We mainly focus on object-oriented modelling.
- We use UML as modelling language.

ソフトウェアモデリング

- 英語の"model"は、動詞として、また名詞として使われる
  - 動詞の"model"は、前のページで説明した意味でモデルを作ることを意味する
- "modelling"はモデルを作る作業を意味する。日本語では「モデリング」あるいは「モデル化」と言う

- ここでは主にオブジェクト指向のモデリングに注目
- UMLをモデリング言語として使用

UML

- Unified modeling language
  - *The OMG's Unified Modeling Language* ™ (UML®) helps you specify, visualize, and document models of software systems, including their structure and design.
    - can be used for business modeling and modeling of other non-software systems too.
  - not programming language, but modeling language
- **Unifies notations and concepts of three major object-oriented development methods.**
    - Not unify methods themselves.
  - OMT method (Rumbaugh): information modeling
  - Booch method (Booch): design level techniques
  - OOSE method (Jacobson): usecase

UML

- Unified modeling language 統一モデリング言語
  - *OMG's Unified Modeling Language*™ (UML®) はソフトウェアシステムおよびその構造や設計のモデルを、仕様化し、視覚的にし、文書化することを助ける
    - これはビジネスのモデリングやその他ソフトウェア以外のモデリングにも使われる得る
  - プログラミング言語ではなく、モデリング言語であることに注意
- **以下の3つの主要なオブジェクト指向開発手法で使われている記法と概念を統一**
    - 手法そのものを統一したわけではない
  - OMT法 (Rumbaugh): 情報モデリングに注力
  - Booch法 (Booch): 設計レベルの技法に注力
  - OOSE手法 (Jacobson): usecaseを導入

Diagrams in UML

Structure diagrams	Class diagram	
	Component diagram	
	Object diagram	
	Composite structure diagram	
	Deployment diagram	
	Package diagram	
Behavior diagrams	Activity diagram	
	Interaction diagram	Sequence diagram
		Communication diagram
		Interaction overview diagram
		Timing diagram
	Use case diagram	
	State machine diagram	

UMLで規定されている図

Structure diagrams (構造を表す図)	クラス図	
	コンポーネント図	
	オブジェクト図	
	合成構造図	
	配置図	
	パッケージ図	
Behavior diagrams (振る舞いを表す図)	アクティビティ図	
	インタラクション図	シーケンス図
		コミュニケーション図
		インタラクション概観図
		タイミング図
	ユースケース図	
	ステートマシン図	

Class diagram and object diagram

クラス図, オブジェクト図

cf. SE-J, 2.2.2

Class and object

- A class is an abstraction of similar entities.
  - In other words, it describes a set of multiple similar entities.
- An object is a construct of a specific class.
  - It is each entity of the set described by a class.
- "Class" and "object" are the concepts of object-oriented programming (programming with, e.g., Java).
  - An object is an entity that encapsulates data and methods. Only its operations are open to others.
  - A class is a description of multiple objects that have the same structure/methods.
- We can use these concepts to describe the structure of our description target; real world entity, business concept, information system, software, etc.

クラスとオブジェクト

- クラスとは、類似した対象を抽象化したもの
  - 別な言い方をすれば、類似した複数の対象の集合
- オブジェクトとは、クラスが示す集合の構成要素
  - 個々の対象事物
- クラスやオブジェクトは、オブジェクト指向プログラミング(例えばJava等を用いたプログラミング)での概念
  - オブジェクトは、データとメソッドをカプセル化したもので、操作のみ公開される
  - クラスは、同じ構造とメソッドを持ったオブジェクトの集合
- これらの概念を、様々な対象の表現にも用いることができる
  - 対象：実世界の実体、ビジネス上の概念、情報システム、ソフトウェア、...

Class diagram and object diagram

- A class diagram describes the structure of the modeling target using classes.
  - classes that consist the target.
  - various kinds of static relationships that exist among them.
  - A class diagram describes an abstract structure.
- An Object diagram describes the structure of the modeling target using objects.
  - objects that exist in the modeling target at a certain point in time.
  - links that exist between objects.
  - An object diagram describes a concrete structure.
    - It shows one concrete example of the abstract structure the class diagram describes.

クラス図とオブジェクト図

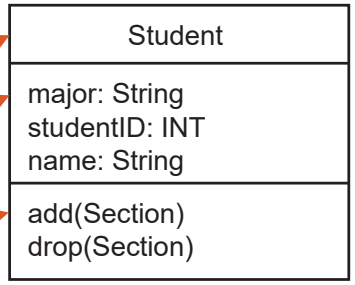
- クラス図は、クラスを用いて対象の構造を表す
  - 対象を構成する要素としてのクラスと
  - 要素間に存在する様々な関係を用いる
  - クラス図は抽象的な構造を表す
- オブジェクト図は、オブジェクトを用いて対象の構造を表す
  - モデリング対象について、ある時点で存在するオブジェクトと
  - オブジェクト間に存在するリンクを用いる
  - オブジェクト図は具体的な構造を表す
    - クラス図が示す抽象構造のひとつの具体例を表す

Notation of class Diagram and object diagram

クラス図、オブジェクト図の記法

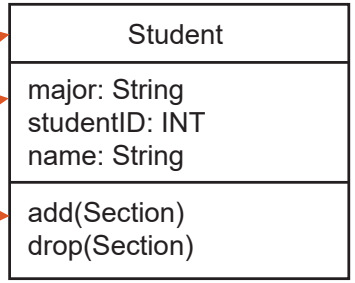
Class

- Class notation: A box. It has often three compartments:
  - Class name
  - Attributes
  - Operations
  - Additional compartments may be supplied to show other details, such as constraints.



クラス

- 記法: 箱. 多くの場合、３つの部分から成る
  - クラス名
  - 属性
  - 操作
  - 制約等の他の詳細を示すために、さらに部分が追加されることもある



Attribute and operation

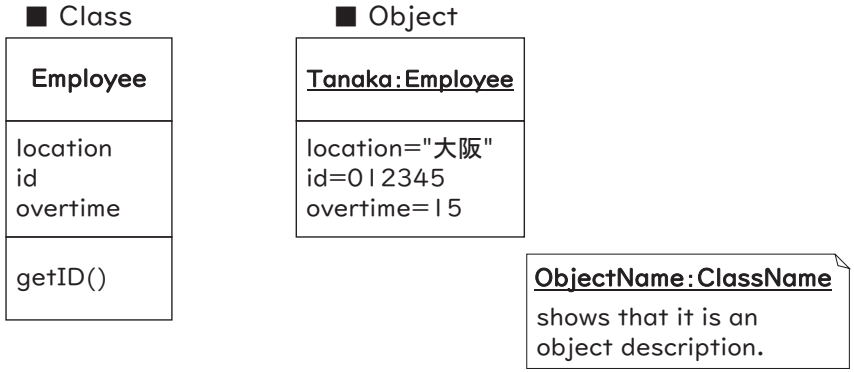
- Attribute
  - Data that every object of the modeling target (class) has.
    - Ex. Every student has major, studentID, and name.
- Operation
  - A service that the class provides to the outside.
    - A role that the class plays in the whole.
  - Roughly speaking, what the class can do.

属性、操作

- 属性
  - モデル化対象(つまりクラス)の全てのオブジェクトが持つデータ、情報
    - 例：すべての学生は、major(専攻)、studentID(学生番号)、name(氏名)を持つ
- 操作
  - クラスが外部に対して提供するサービス
    - クラスが全体の中で果たす役割
  - 直感的に言えば、「クラスができること」

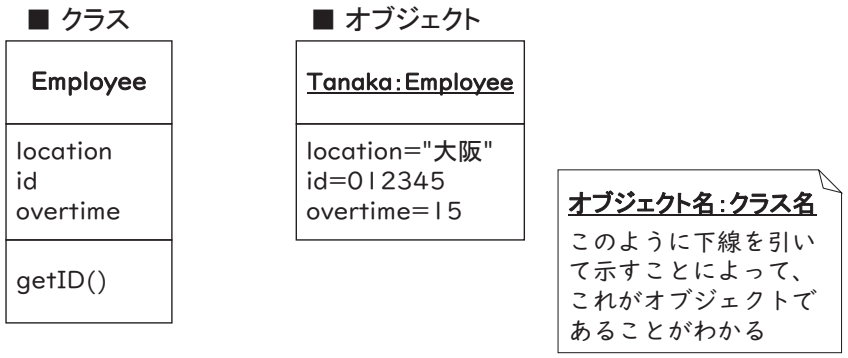
Object

- A concrete example of a class
  - An instance of a class
  - shows concrete example of data structure (attribute).



オブジェクト

- クラスの具体例
  - クラスのインスタンス
  - データ構造(属性)の具体例を示す



Notation option

- Class name (of class in a class diagram and of object in an object diagram) is mandatory.
- Object name of an object may be omitted.
- Attributes and operations (of class and object) may be omitted.

記法のオプション

- (クラス図中のクラス、オブジェクト図中のオブジェクトの)クラス名は必須
- オブジェクトについて、オブジェクト名は省略されることがある
- 属性、操作は(クラスであってもオブジェクトであっても)記述しなくても構わない

Attribute syntax

- **[*visibility*] name [*multiplicity*] [:*type*] [=*initial-value*] [{*property-string*}]**
  - *visibility* : public "+", protected "#", or private "-"
  - *name* : (typical usage: capitalize first letter of each word that makes up the name, except for the first)
  - *multiplicity* : number, range, or sequence of number or ranges.
  - *type* : built-in type or any user-defined class
  - *initial-value* : any constant and user-defined object
  - *property-string* : e.g, changeable, addOnly, frozen

属性のシンタクス

- **[*visibility*] name [*multiplicity*] [:*type*] [=*initial-value*] [{*property-string*}]**
  - *visibility (可視性)*: public "+", protected "#", or private "-"
  - *name (名前)*: (英文表記の場合は、大文字で始める。複数の語からなる場合は、それぞれの語の先頭を大文字にして、空白を入れずにつなぐ)
  - *multiplicity (多重度)*: その属性値の範囲
  - *type (型)*: 既定のものもしくはユーザ定義のクラス
  - *initial-value (初期値)*: 個定値もしくはユーザ定義のオブジェクト
  - *property-string (プロパティ文字列)*: 変更可能、追加のみ、変更不可等

Operation syntax

- **[*visibility*] name [(*parameter-list*)] [:*return-type*] [{*property-string*}]**
  - *visibility* : public "+", protected "#", or private "-"
  - *name* : (typical usage: verb or verb phase, capitalize first letter of every word, except first)
  - *parameter-list* : coma separated list of parameters
  - *return-type* : primitive type or user-defined type
  - *property-string* : isQuery, sequential, guarded, concurrent

操作のシンタクス

- **[*visibility*] name [(*parameter-list*)] [:*return-type*] [{*property-string*}]**
  - *visibility (可視性)*: public "+", protected "#", or private "-"
  - *name (名前)*: (英文表記の場合は、大文字で始める。複数の語からなる場合は、それぞれの語の先頭を大文字にして、空白を入れずにつなぐ)
  - *parameter-list (パラメータリスト)*: コンマ区切りのパラメータのリスト
  - *return-type (返り値)*: プリミティブ型かユーザ定義型
  - *property-string (プロパティ文字列)*: 問い合わせか、連続、条件付き、並行等

Visibility

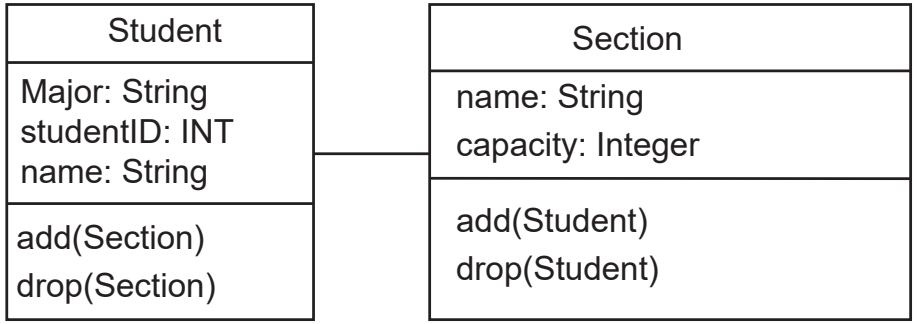
- Visibility refers to whether an element of the class is visible from outside the class.
- NOTICE:
  - Basically, attributes have to be hidden (=not visible) from outside. Operations are visible.
  - But detailed access control would be needed in some cases (especially in programming phase).
  - → Visibility can be depicted freely on attributes and operations.
- Depicting visibility is optional on a class diagram.

可視性

- 可視性は、その要素がクラス外からも参照できるかどうかを示す
- 注意:
  - 基本は、属性は見えない(参照できない)、操作は見える(参照できる)
  - しかし詳細な制御を追加したい場合もある (特にプログラミングの工程等において)
  - → よって、可視性を属性についても操作についても自由に書けるようになっている
- クラス図において可視性を示すことはオプションであって、必須ではない

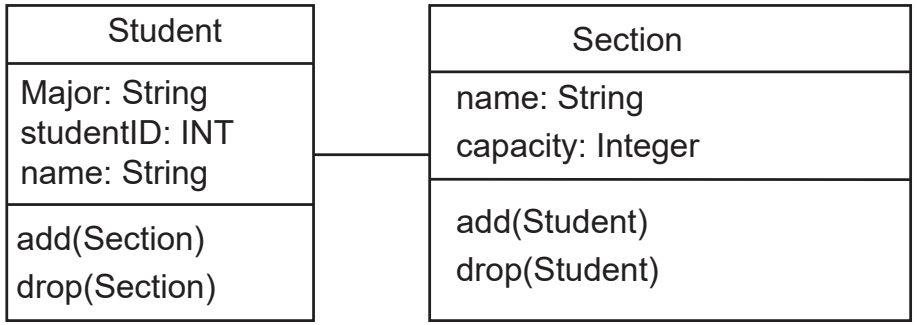
Association

- Each class can be associated with other classes.
  - Line between classes describes "association".
  - An association is a structural relationship that specifies that objects of class may be connected to objects of another class.



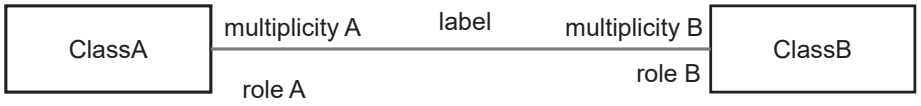
関連

- クラスは他のクラスと関連づけられ得る
  - このことをクラス間にひかれた線で表現し、この線が「関連」である
  - 関連は、クラスのオブジェクトが他のクラスのオブジェクトと何か結び付けられる得るという構造上の関係を示す



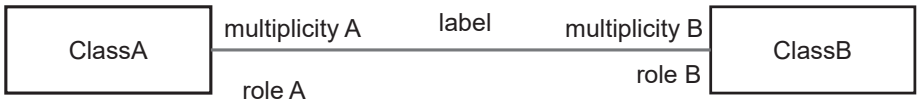
Association notation

- Association is described by a direct line.
  - Has a label
  - Has multiplicities
  - Has association end names



関連の記法

- 直線で表現
  - ラベルがつく
  - 多重度を持つ
  - 関連端名を持つ



Label (on association)

- Indicates association name.
- Optional; but it is recommended for you to name every association.
  - If you cannot give an appropriate name to an association, the association itself could be inappropriate. (There could be no association between the classes.)
  - Noun can be used.
  - Verb phrase can be used.

関連上のラベル

- 「関連名」を示す
- これはオプショナル(書かなくても間違いではない)が、すべての関連に関連名をつけることをおすすめする
  - もしも適当な名前をつけることができないとしたら、関連そのものが適当でないのかもしれない。つまり、クラス間には関連がないのかもしれない
  - 関連名として、名詞を使ってよい
  - 動詞も使える
    - ただし日本語でモデル化している場合には、動詞の意味を持つ名詞で書くことが多い。例えば、「所属する」ではなく、「所属」と書く等

Multiplicity (on association)

- Indicates how many objects may be related to an object of the class on the other end.

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where n>1)
0..n	Zero to *n* (where n>1)
1..n	One to *n* (where n>1)

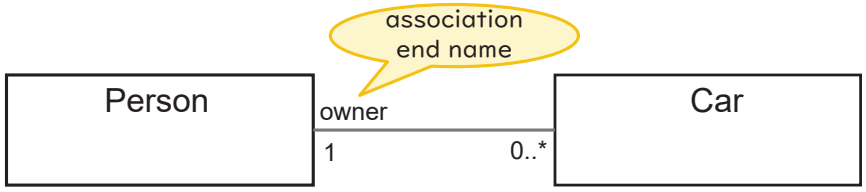
多重度

- そのクラスのあるオブジェクトが、関連する他方のクラスのいくつのオブジェクトと関係し得るかを示す

多重度の表現	意味
0..1	0か1
1	1
0..*	0 以上
1..*	1以上
n	*n* (ただしn>1)
0..n	0から*n* (ただし n>1)
1..n	1から*n* (ただし n>1)

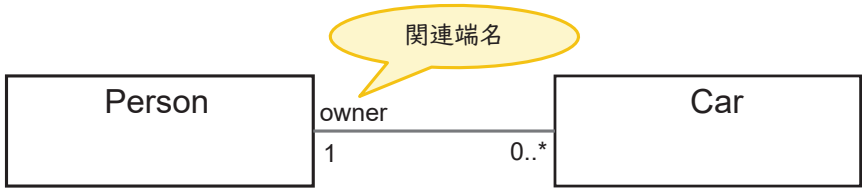
Association end name

- You can name both association ends.
- Typically, it is used to describe the role that objects play on the association.



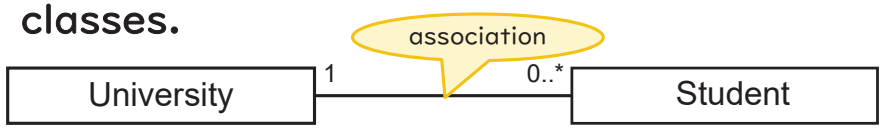
関連端名

- 関連端に名前を付けることができる
- 典型的には、そのオブジェクトが関連において果たす役割を示す

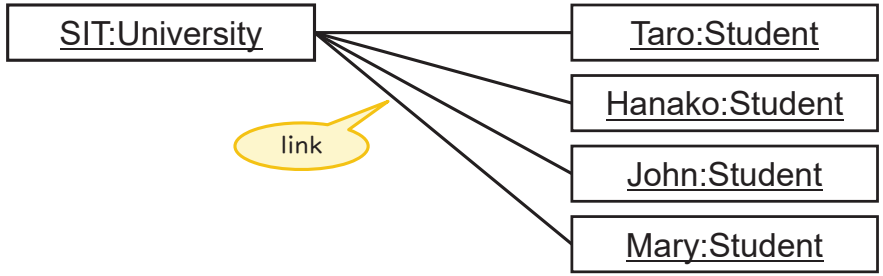


Association and link

- Association: structural relationship between classes.



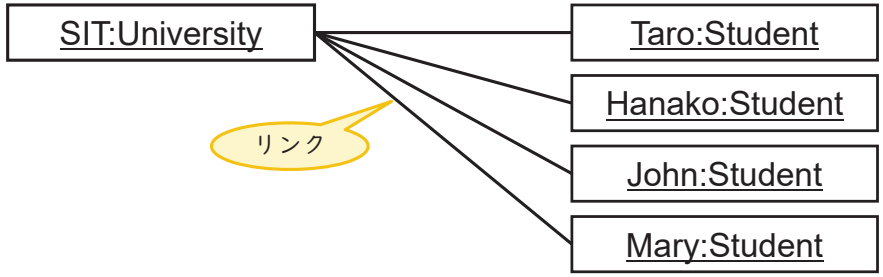
- Link: An instance of an association; between objects.



関連とリンク

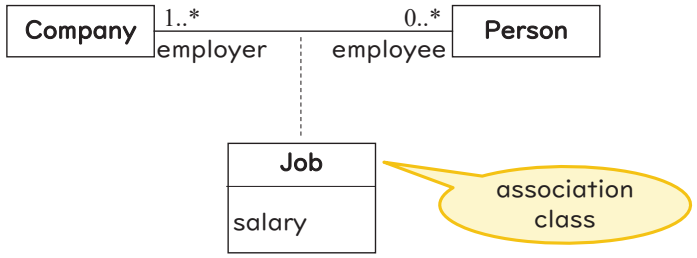
- 関連：クラス間の構造上の関係

- リンク：関連のインスタンス(具体化したもの)。オブジェクト間に存在



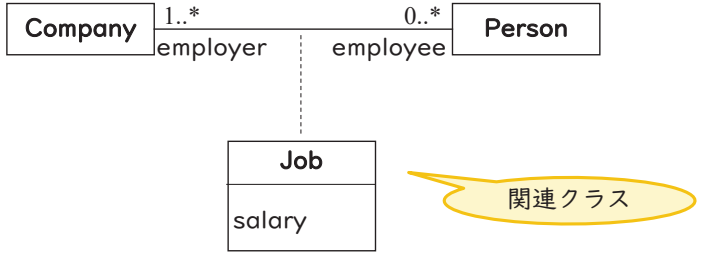
Association class

- An association that has a set of features (attributes and operations) of its own.
- Both an association and a class.



関連クラス

- 関連自身が属性や操作を持つ場合がある
- つまり、関連であり、クラスである
- こうしたものを「関連クラス」として記述



cf. SE-J, p.28

Aggregation

- Special kind of association.
- means "part of."
  - "has-a" relation
- symbolized by a white diamond.

The above diagram shows "A company has many departments." In other word, a department is a construct of a company.

An aggregation could be described as an association, because an aggregation is a special association.
When you want to emphasize "whole-part" relationship, use aggregation.

集約

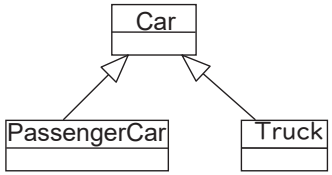
- 関連の特殊形
- 部分であること、全体に対する構成要素であることを示す
  - “has-a" の関係
- 白抜きひし形で示す

Company(会社)は多くのDepartment(部門)を持つ、あるいは部門は会社の構成要素である、ということを上の図は示している

集約は、関連の特殊形であるので、関連を用いても表し得る。関連の中で、「全体部分」の関係を強調したい時には、集約を使うと良い

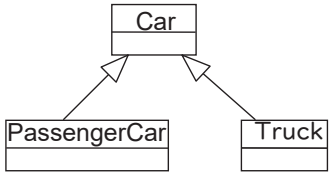
Generalization

- Generalization describes the relationship between generalized elements and more specific elements
  - A triangle on the side of the general class
  - Same as inheritance hierarchy (inheritance)
    - "is-a" relation



汎化

- 汎化は、一般化された要素とより特殊な要素の間の関係を示す
  - 白抜き三角形がついている方が一般的なクラス
  - (オブジェクト指向プログラミングでの)継承階層の表現と同じ
    - "is-a" 関係



cf. SE-J, pp. 28-29

Notes

- So far, we have learnt the basics of class diagram and object diagram.
- About class diagram, there are more advanced concepts and notations. If you are interested, see the specification documents (available from OMG's website).

補足

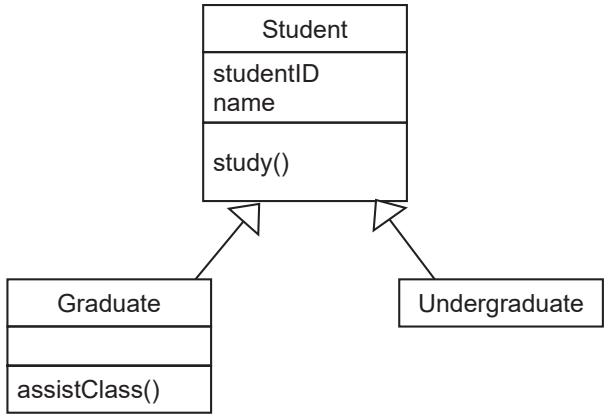
- クラス図、オブジェクトの基本をここまでで学修
- さらに発展的な概念およびその記法がクラス図にはある。興味があれば仕様書(OMGのサイトから入手可能)を参照してください

Exercise

- Draw a class diagram describing the following:
  - There are three classes：University, Teacher, Student
  - There are three associations:
    between University and Teacher,
    between University and Student,
    between Teacher and Student
    - add an appropriate association name
    - depict multiplicities on associations

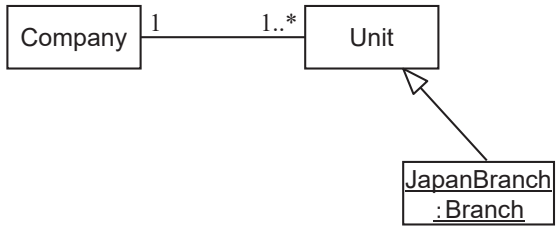
Exercise

- Write down all attributes and operations of each class.



Exercise

- What is wrong?



練習

- (これより前の3ページの練習問題については、簡単な英語しか使われていないので、日本語訳ページは省略)