Software Design
11. Class Diagrams: Advanced Concepts

Natsuko Noda
nnoda@shibaura-it.ac.jp

ソフトウェア設計論
11. クラス図発展

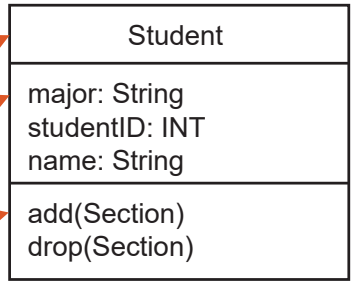
野田夏子
nnoda@shibaura-it.ac.jp

Class diagram: Advanced concepts and notations

クラス図：発展的な概念と記法

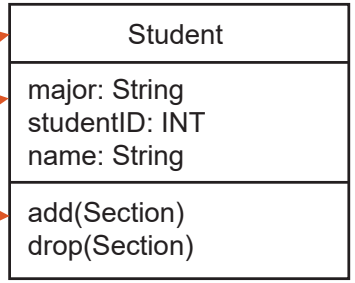
Class (Review)

- 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, or to divide features.



クラス (復習)

- 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 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

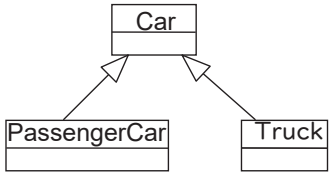
- Visibility refers to whether an element of the class is visible from outside the class.
- NOTICE:
  - Basically, attributes have to be hidden (=not visible) from outside. Operations are visible.
  - But detailed access control would be needed in some cases (especially in programming phase).
  - → Visibility 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

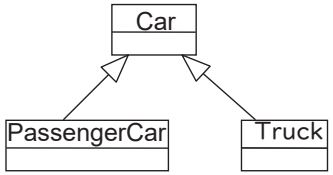
Generalization (Review)

- 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 (復習)

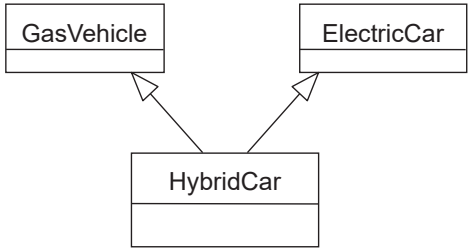
- 汎化は、一般化された要素とより特殊な要素の間の関係を示す
  - 白抜き三角形がついている方が一般的なクラス
  - (オブジェクト指向プログラミングでの)継承階層の表現と同じ
    - "is-a" 関係



cf. SE-J, pp. 28-29

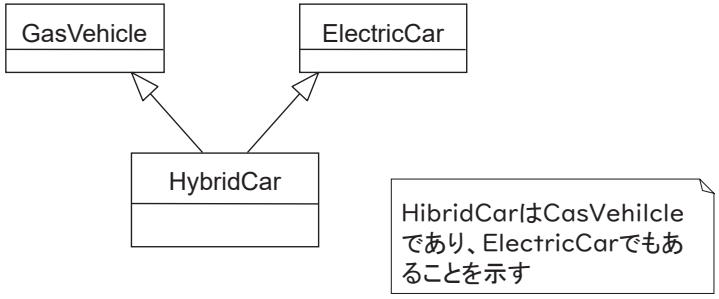
Generalization (Cont.)

- Multiple inheritance is allowed and can be described as below.



汎化について

- モデリングにおいて、多重継承は許されており、以下のように記述する
  - 参考) Javaでは多重継承は許されていない



Association notation

- Association is described by a direct line.
  - Has a label
  - Has multiplicities
  - Has association end names
  - Is directional – the navigation

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
  - 方向性がある – 誘導

Navigability

- Associations are directional.
  - Think about which class needs to know about the other; which class is an attribute of the other? Why/when do I care?

The source knows the target.

  - At first, you don't have to consider navigability too much.

誘導可能性

- 関連は向きを持ちうる
  - 関連がある時に、関連に参加するどちら側のクラスがどちら側のクラスを必要とするか？それを知っているか？
  - このような向き(誰が誰を知っているか)を示すために、関連に矢印を用いることがある

上の例ではSourceがTargetを知っている

  - しかし、誘導可能性については、特にモデリング学習の初期においては、あまり気にしなくて良い

Aggregation (Review)

- 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 (復習)

- 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

Composition

- A composition is similar to an aggregation. But in the case of composition, the component cannot survive on its own.
  - Strong aggregation.
- symbolized by a black diamond.

- NOTE: "Strong" is subjective. At first, you don't have to strictly distinguish between "aggregation" and "composition". If you are not sure, use "aggregation".

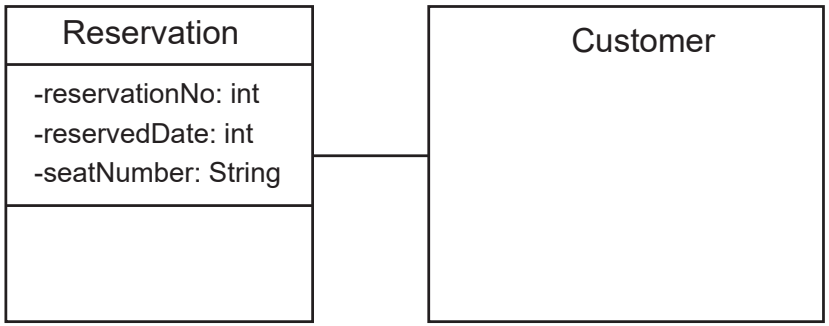
コンポジション

- コンポジションは、集約に近い概念であるが、コンポジションを使った場合には部分の側は単独では存在できないことを示す
  - 強い集約と言える
- 黒く塗りつぶしたひし形で表現

- 注意: 「強い」というのは主観的。最初は、集約とコンポジションを厳密に使い分けようとしなくて良い。もしはっきりしなければ、集約を使っておく

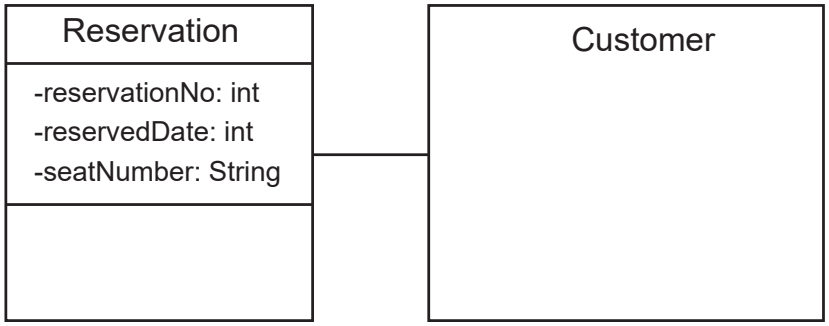
Java Example

```
class Reservation {
     int reservationNo;
     int reservedDate;
     String seatNumber;
     Customer reservationOwner;
     …
}
```



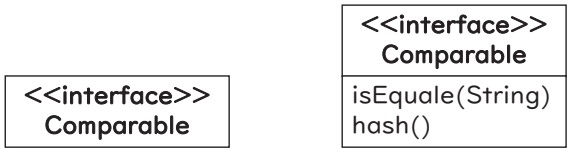
Java Example

```
class Reservation {
     int reservationNo; // 予約番号
     int reservedDate;       // 予約日
     String seatNumber;      // 席番号
     Customer reservationOwner;   // 予約者
     …
}
```



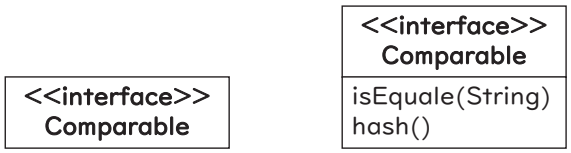
Interface

- An interface is a kind of class that has only public operations, with no method bodies.
  - defined by the signatures of the operations.
    - Signature: name, parameters, and return value.
  - corresponds to interface in Java.
- Shown using the class icon with the keyword «interface».



インタフェース

- インタフェースはクラスの一種であり、公開された操作だけを持ち、メソッド本体を持たない
  - 操作のシグニチャにより定義される
    - シグニチャ：名前、パラメータ、返り値
  - Javaでのインタフェースに対応する
- クラスと同じ記号を用いて、キーワード «interface» をつける



Interface and relations

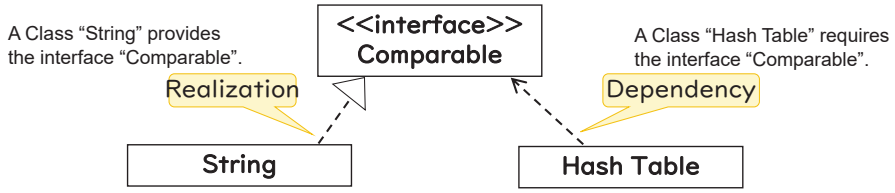
- Realization： Relationship between specification and implementation
  - Ex. Relationship between an interface and a class that implements the interface.
- Dependency： Relationship that shows that a class uses another
  - supplier/client relationship.
  - Ex. Relationship between an interface and a class using the interface.

インタフェースが持つ関係

- 実現関係： 仕様と実装の関係
  - Ex. インタフェースとそれを実装するクラスの関係
- 依存関係： クラスが他を使う関係
  - サプライヤ/クライアントの関係
  - Ex. インタフェースとそれを使うクラスの関係

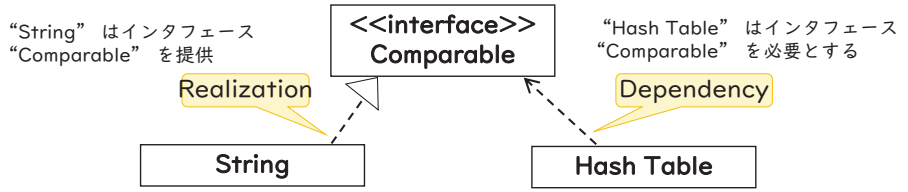
Interface and relations (Cont.)

- Notation:
  - Realization: Dashed line with a triangular arrowhead at the end that corresponds to the realized element.
  - Dependency: Dashed arrow. The element at the tail of the arrow (the client) depends on the element at the arrowhead (the supplier).



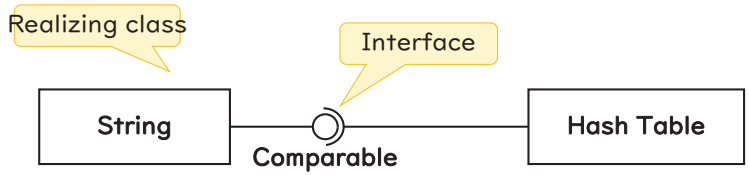
インタフェースが持つ関係 (Cont.)

- 記法:
  - 実現関係: 三角形のついた破線。三角形がついている方のものをついていない方が実現する
  - 依存関係: 破線矢印。矢印の根元が、矢印の先に依存する



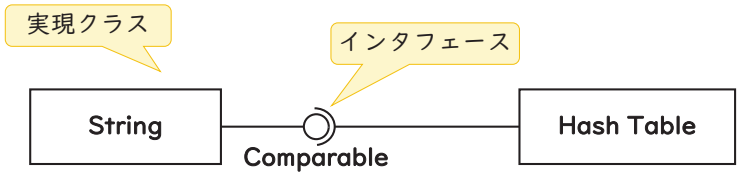
Interface and relations (Cont.)

- Compact notation.



インタフェースが持つ関係 (Cont.)

- 以下のようなコンパクトな記法を用いることもできる



Dependency

- Dependency can be defined not only between interface and class, but also between classes.
  - One class uses another class.
  - A weaker form of association.

依存関係

- 依存関係は、インタフェースとクラス間だけでなく、クラス同士の間ににも定義できる
  - あるクラスが別のクラスを使う場合
  - 関連の弱い形である

Abstract class and interface

- Abstract class
  - A class that cannot be directly instantiated.
    - Instead, you instantiate an instance of a subclass (that is not abstract).
  - Its one or more operations are abstract; that means they have no method bodies.
  - Shown by italicizing its name.

抽象クラスとインタフェース

- 抽象クラス
  - 直接インスタンス化できないクラス
    - そのかわりに、抽象クラスから継承した抽象でないクラスをインスタンス化する
  - ひとつ以上の操作が抽象である。
    - 操作が抽象であるとは、その操作がメソッド本体を持たないこと
  - クラス名を斜字体にすることにより表現