学習目標

この学習モジュールを受講すると，クラス図についてよく理解し，開発プロセスの中での分析レベルで，情報システムの対象をモデリングできるようになります． 具体的には次のことができるように学習しましょう

- クラス図の目的を説明できる
- クラス図におけるクラスの表記法、可視性の記号とその意味を説明できる
- 派生属性とは何か説明できる
- クラス間の関連の表記法と「多重度」の記法の意味を説明できる
- クラス間の関係で，「集約」「コンポジション」の記法と意味「依存関係」「汎化」を説明できる
- astahを使ってクラス図を描くことができる与えられた情報システムの説明に適合するクラス図を描ける
- 簡単なクラス図に対応するJavaのコードを記述できる

クラスとクラス図

クラスとは，現実に存在するもの，つまり具体例であるオブジェクトの属性や振る舞いの共通性に着目して抽象化したものです．オブジェクト指向のプログラミングにおいて，そのオブジェクトがどのようなはたらきをするかといった「機能」と，どのような性質を持っているかといった「属性」を定義しています．図１に，具体例であるオブジェクトと抽象化したクラスの関係を示します．

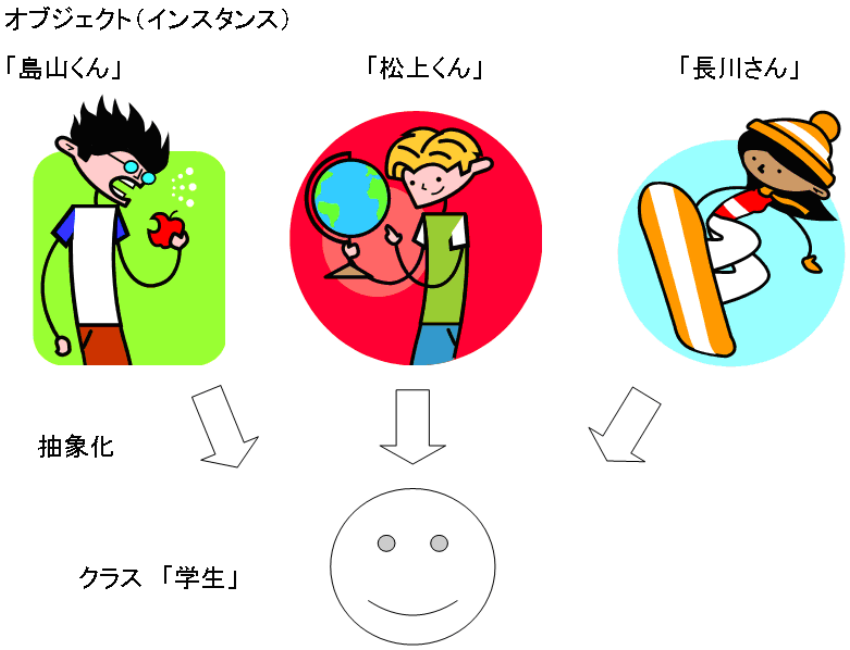


図1 オブジェクトとクラス

「島山くん」「松上くん」「長川さん」という個々のオブジェクトが持つデータはいろいろありますが，「TE大学の学生である」という共通点に注目すると，「学生」というクラスに抽象化して表現することができます．クラスは，JavaやC++などのプログラミング言語に出てくるプログラムの単位のクラス（class）と同じものです．UMLで設計を行う場合は，作成したものは最終的にプログラムコードへと移行し，UMLのクラスはそのままオブジェクト指向言語のクラスとなります．

クラス図は，クラスとクラスの静的な関係を表現するもので，いろいろな場面で使うことができます．

開発の初期段階の，仕様作成フェーズの場合には，プロジェクトの参加メンバーにシステムを理解してもらうために概念的なモデルとしてのクラス図を作成することがあります．これから作ろうとしているシステムや，そのシステムが利用されるところにおいて，どのような概念があってどのような構造を持っているのかを表現します．例えば，これから作ろうとしているシステムに「学生」「履修登録」「図書貸出」といった概念があることなどを表現します．これによってシステムの大まかな構造を理解することができます．

分析・設計のフェーズでは，アーキテクチャを記述することができます．例えば，「図書貸出」には「資料名」や「学籍番号」が必要ですし，「学生をユーザーとして登録する」，「貸出期間の設定」など，システムを実現するために何が必要か，何を実現すればシステムが動くのかを表現します．また開発のフェーズでは，クラス図とソースコードの同期を取りながら，開発を進めることができます．

分析・設計のフェーズで用いられるクラス図は，システムの作り方を表現するもので，ここに描かれ

ていることはそのままの形で実装されることになります．つまり，「クラス図に描かれている構造」が「ソフトウエアの構造」となるので，どの機能がどのクラスにあるとか，クラス間の依存関係が分かりやすくなります．このようにクラス図は，仕様作成のフェーズから開発のフェーズまで幅広く利用されます．

クラスの表記

クラスは長方形のアイコンで表記します．クラス名のほかに，必要に応じて属性や操作を記述します．クラス名は一番上のアイコンの中に書きます．設計に用いる場合にはクラス名を日本語で書くことがありますが，プログラム開発に用いる場合は，使う言語にあったクラス名をつけることが必要です．その下に長方形のアイコンを加え，上から2番目のアイコンの中にはクラスの属性を記述します．3番目には操作を記述します．

図2にクラスの表記方法を示します．左側の図では，それぞれのアイコンの中に何を書くかを示しています．上から順に「クラス名」と「属性」と「操作」をそれぞれ記述します．右側には，その具体的な例を示しました．「学生」クラスを表す場合は一番上のアイコンにクラス名の「学生」と書き，学生クラスがその属性として「氏名」と「学籍番号」と「連絡先」を持っている場合は，2番目のアイコンに「氏名」「学籍番号」「連絡先」と書きます．学生クラスに，「氏名取得」「連絡先登録」といった操作がある場合は，3番目のアイコンに「氏名取得」「連絡先登録」と書きます．属性と操作は1行に１つ配置します．

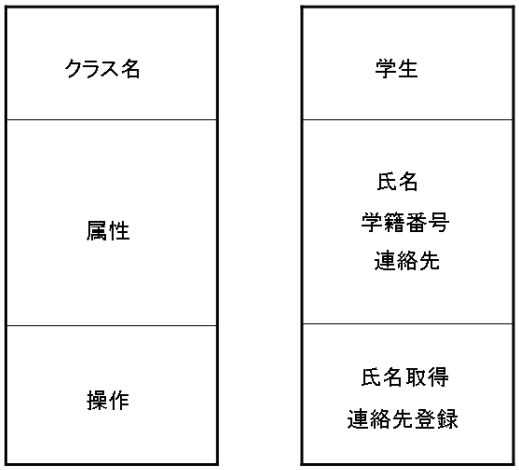


図2 クラスの表記

属性と操作の記述は省略することができます．省略したクラスの表記の例を図3に示します．一番左の図は，属性と操作の両方を省略し，クラス名のみを記述している例で，クラス名のアイコンだけが表示されています．真ん中の図は，操作の記述を省略し，クラス名と属性を記述しています．操作のアイコンは表示されず，クラス名と属性のアイコンだけが表示されている状態です．右側の図は，属性の記述を省略し，クラス名と操作を記述しています．この場合，属性のアイコンが表示されず，クラス名と操作のアイコンだけが表示されています．このように属性や操作の記述が省略されている場合は，属性・操作のアイコンそのものを隠すことができます．

こう描いても OK

図3 クラスの表記

クラスの属性

属性は，クラスを構成している情報の一つで，クラスの静的なデータを表現します．クラスの性質を現すもので，プログラム中ではメンバ変数となります．クラスに属性が定義されていると，そのクラスから生成されたオブジェクトはすべてクラスで定義された属性を持ちます．さらに，各オブジェクトのそれぞれの属性には，オブジェクト独自の属性値を持たせることができます．

属性は，以下のフォーマットで記述します．

属性名：属性の型＝初期値

属性の記述例を，図4に示します．左側の図では，クラス名は「学生」で，「学生」クラスは，属性として「氏名」「学籍番号」「連絡先」を持っています．さらに，属性である「氏名」はString型のデータを持つこと，「学籍番号」と「連絡先」も同じくString型であることが示されています．

図４の右側の図は，具体例であるオブジェクトに，属性値を持たせた例です．「学生」クラスのオブジェクトである「島山」くんは，属性「氏名」の値として「島山正」を持ちます．属性「学籍番号」「連絡先」もそれぞれ「077177」「03-35XX-1234」という属性値を持っています．「氏名」「学籍番号」「連絡先」のデータの型はStringなので，これらの値である「島山正」「077177」「03-35XX-1234」もString型です．

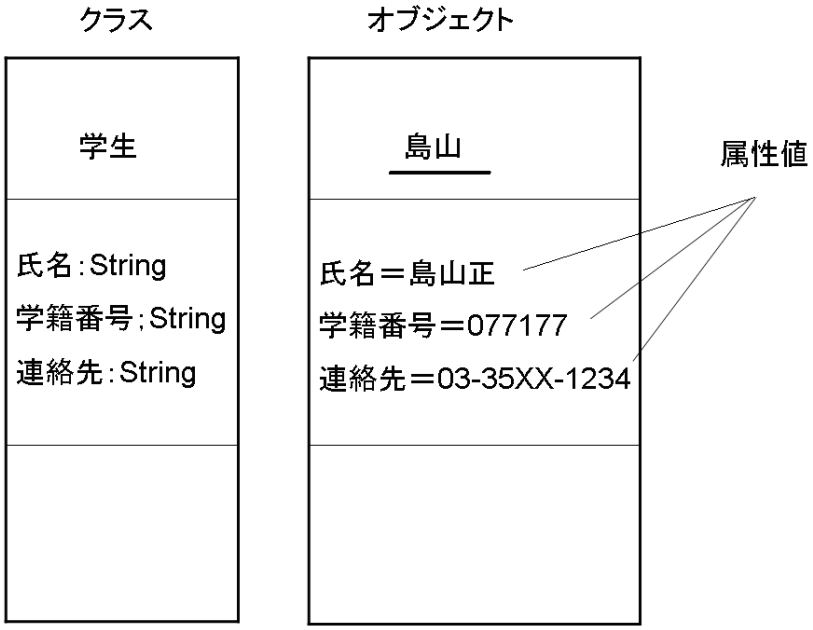


図4 属性の表記

クラスの操作

操作は，クラスを構成している情報の一つで，動的な振る舞いを表現します．操作がクラスに定義されていると，そのクラスから生成されたオブジェクトはすべてクラスで定義された操作を持ちます．また，そのオブジェクトに対して，その操作を呼び出すことができます．

操作は，以下のフォーマットで記述します．引数名，引数の型，戻り値の型は，省略することもできます．

操作名（引数名：引数の型）：戻り値の型

操作の記述例を図5に示します．「氏名取得」という操作は，引数を持たず，戻り値の型は，Stringです．「連絡先登録」という操作は，String型の引数「連絡先」を持ち，戻り値はありません．

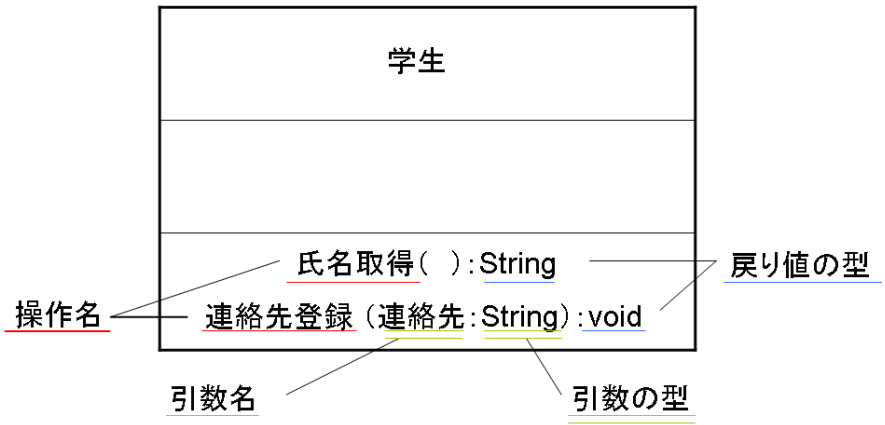


図5 操作の表記

可視性

属性名や操作名の前に可視性を指定して，その属性や操作が他のクラスから参照可能かどうかを示します．可視性の表記方法を表１に示します．

表1 可視性の表記方法

+	public	すべてのクラスから参照可能
−	private	自分自身のクラスからのみ参照可能
#	protected	自分自身およびサブクラスから参照可能
~	package	自分自身および同一パッケージ内のクラスから参照可能

このように，クラスなどを公開部分と非公開部分に分けることをカプセル化（情報隠蔽）と呼び，カプセル化されたものは，外部に公開された操作を通じてのみ，外部からのメッセージの送信に反応します．公開されている操作には呼び出し名，引数，戻り値などが明確に定義されていて，この公開されている操作を呼び出す以外に内部にアクセスする手段はありません．

可視性を表記したクラスの例を図6に示します．学生クラスの属性「氏名」「学籍番号」「連絡先」の可視性としては，privateが指定されています．「氏名取得」「連絡先登録」といった操作にはpublicが指定されています．属性の「連絡先」はprivateと指定されているので，publicと指定されている操作「連絡先登録」を通じて外部からアクセスが可能です．

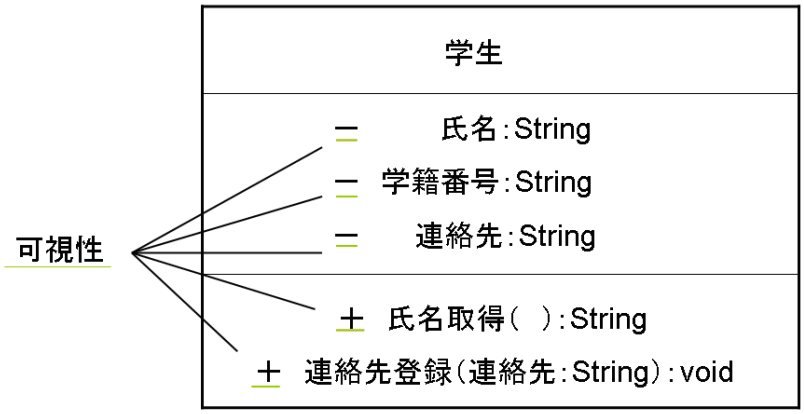


図6 可視性の表記

【コラム】 省略可能とは？

属性と操作の記述は省略可能ですが，属性・操作がないということではなく，表示を省略しているということになります．また，強調したい属性や操作のみを表示することもあります．

可視性や引数，戻り値についても，表示を省略できます．システムの仕様や概念を検討している段階で作成したクラス図では，引数の型などは省いて大まかなところを記述し，開発の段階のクラス図では，細部まで記述するというように，クラス図の作成の目的に合わせて省略することができるのです．

セルフテスト クラス図の概要

次のページに移動し，「クラス図の概要」に関するセルフテストを受験して，理解を確認しましょう．

クラス間の関連

クラスは，属性や操作以外に，関係を持つことができます．他のクラスと関係を持つことで，構造化された情報を表現できるようになります．

関連・関連名・関連端名

クラスは，そのクラスが持つ，属性や操作を定義するだけでなく，他のクラスとの関係を表現することで，さらに構造化された情報を表現することができます．構造化することでより正確なモデルを作成することができます．関係の表現には，「関連」「集約」「コンポジション」「依存」「汎化」などがあります．

関連は，クラス間に構造的な関係があるときに使います．関連を表すには，2つのクラスの間を実線で結びます．関連の表記例を図7に示します．「大学」と「A教室」，「大学」と「教員」，「大学」と「学生」に引かれた実線がそれぞれ関連を示しています．

関連には，必要に応じて関連名をつけることができます．関連名には，黒塗りの三角形をつけて，関連名を読む方向を指定することもできます．図７では，「大学」から「教員」にひかれた関連に「雇用する」という関連名をつけています．

また，あるクラスとあるクラスが関連を持っているとき，一方のクラスから別のクラスをみたときに，別のクラスがどのようにみえるかを関連端名（かんれんたんめい）で定義します．関連端名は，関連の両端につけることもできます．図７の，「大学」と「学生」に引かれた関係の場合，「学生」から「大学」をみると「在学先」であり，「大学」から「学生」をみると「在学生」であることが関連表されています．関連端名は，以前「役割名」「ロール名」と呼ばれていました．

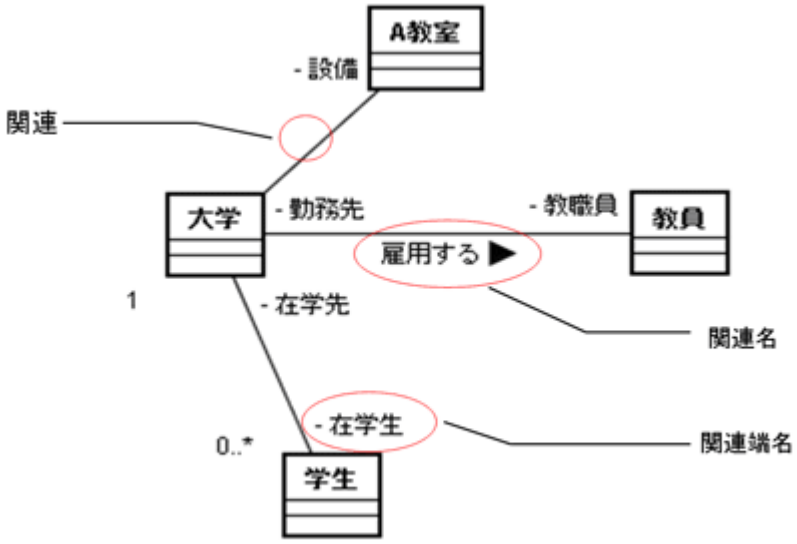


図7 関連と関連名

多重度

２つのクラス間に関連が存在しているとき，一方のクラスのオブジェクトからみて，他方のクラスのオブジェクトがいくつ接続されるかを多重度で表現します．多重度は「下限値．．上限値」のフォーマットで記述します．表２に多重度の表記方法を示します．

表2 多重度の表記方法

記述	説明
0..*または*	0以上無限大
1..*	1以上無限大
1	1
0..1	0か1
2..40	2から40

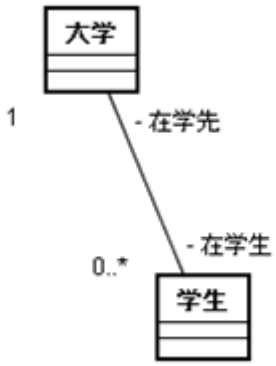


図8 多重度の例1

図9では，「理工学部」には4つの「学科」があるので，「理工学部」から見た「学科」の関係は4となり，「学科」からみた「大学」は1となります．

図9 多重度の例2

集約

あるクラスとあるクラスが，全体と部分の関係になっている場合に，2つのクラスは集約関係にあるといいます．集約は，全体側のクラスの関連の端に白抜きの菱形をつけた実線で表します．集約の表記の例を図10に示します．「机」と「黒板」は「教室」の部分なので，全体側のクラス「教室」の関連の端に菱形をつけ，「教室」と部分側のクラス「机」「黒板」が集約の関係になっていることを示しています．

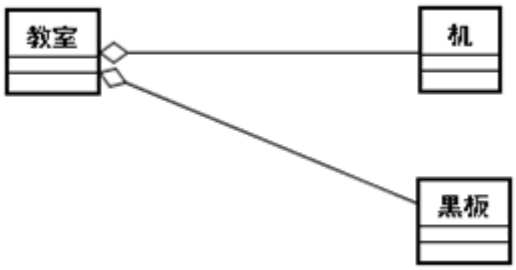


図10 集約

コンポジション

コンポジションは集約の一種で，全体側のクラスと部分側のクラスが同時に生成されて同時に消滅するといった，クラスのライフサイクルがほぼ同一の場合に使います．全体側のクラスの関連の端に黒塗りの菱形をつけた実線で表します．

コンポジションの表記の例を図11に示します．「主翼」は「飛行機」の一部で，「飛行機」と「主翼」は，ほぼ同時に製造されるので，この2つのクラスはコンポジションの関係にあるといえます．全体側のクラス「飛行機」の関連の端に黒塗りの菱形をつけ，部分側のクラス「主翼」「尾翼」とはコンポジションの関係であることを示しています．

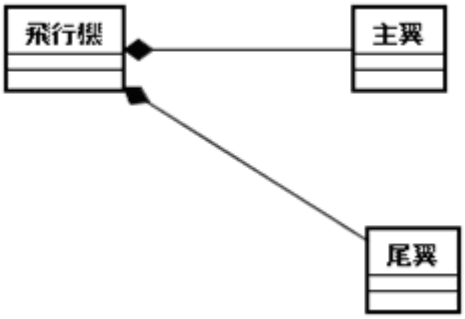


図11 コンポジション

汎化

汎化とは，あるクラスとそのクラスの性質を引き継ぐ別のクラスとの関係です．汎化関係は白抜きの三角がついた実線で表します．汎化関係にある上位のクラスをスーパークラス，下位のクラスをサブクラスと呼びます．

汎化の表記の例を図12に示します．「教室」は「普通教室」「CL教室」のスーパークラスであり，「普通教室」「CL教室」は「教室」のサブクラスです．また，「普通教室」「CL教室」は「教室」を継承しているともいいます．

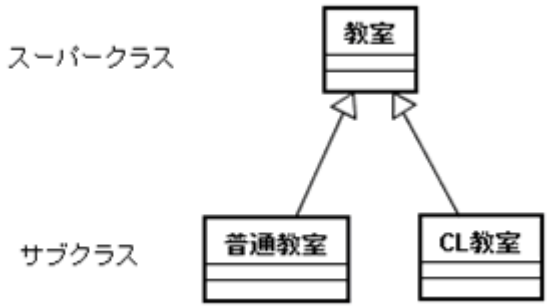


図12 汎化

依存関係

依存関係は，2つのクラス間の関係の中でも利用するとか参照するといった関係を表現するときに使用します．依存関係は，点線の矢印で表記します．依存関係の表記の例を図13に示します．クラスAはクラスBを利用・参照しています．

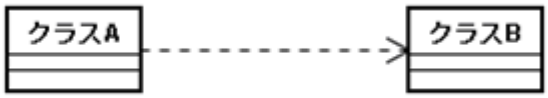


図13 依存関係

依存関係は，次の場合に使用します．

- 引数として参照するとき
- ローカル変数として参照するとき
- グローバル変数として参照するとき

【コラム】 派生属性

派生属性は，他の属性から計算できる属性です．独立した属性として持たせる必要はありませんが，計算に時間がかかるなどの理由で，主にパフォーマンスを向上させたいときに持たせることがあります．属性名の前に ／ を置いて，派生属性であることを示します．例えば，属性として生年月日が与えられている場合，年齢は計算することができますが，派生属性とすることもできます．

セルフテスト クラス図における関連

次のページに移動し，「クラス図における関連」に関するセルフテストを受験して，理解を確認しましょう.

例題実習 astahを使ってクラス図を描く

次のクラス図と同じものを，それぞれastahで描いてみましょう．

例題1 基本的なクラス図

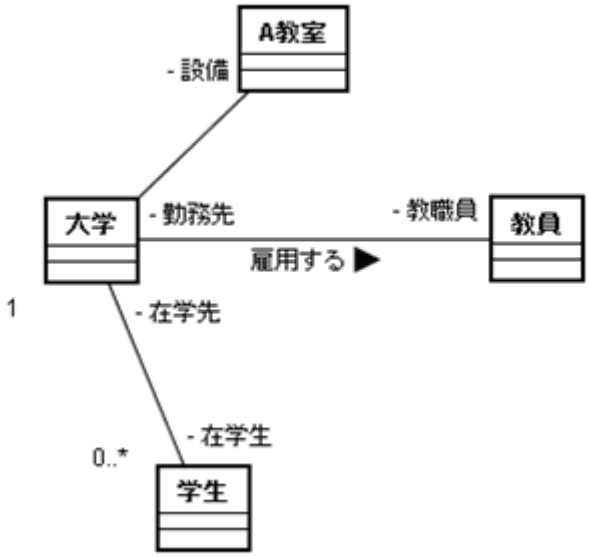


図14 例題1

例題2 少し複雑なクラス図

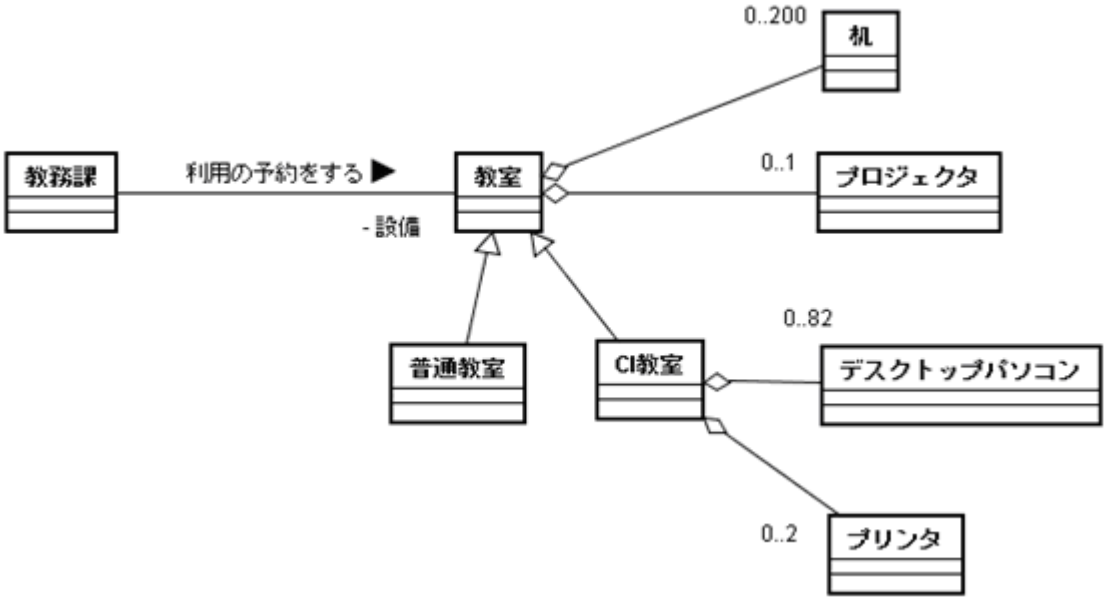


図15 例題2

それぞれのクラス図の意味を，友達と説明しあいましょう．

クラス図を描く際のポイント

(1) 目的にあったクラス図を描く
分析レベルのモデルと設計レベルのモデルの間には「システムとして作るべきものを明確にする」「実装可能なレベルまで明確に記述する」といった違いがあります．どちらのレベルが要求されているのか，目的をはっきりさせてクラス図を描きます．

(2) クラス名と中身を合わせる
クラス図を見直していくうちに，クラスに新しい役割や属性を割り当てることがあります．クラス名と中身の間にずれができると，分かりにくいクラス図になってしまいます．中身にふさわしいクラス名にしましょう．

(3) 一つのクラスに持たせる役割は１つか２つ
多くの役割が一つのクラスに集中すると，そのクラスが多くの属性や操作を持つことになり，そのクラスのメンテナンスが大変です．他のクラスに役割を振り分けることができないか，もう一度クラス図を見直しましょう．

(4) 関連に情報をつける
なるべく，多重度や関連端名をつけるようにしましょう．インスタンスが持つ役割を明確にしておかないと，関連がひかれた意味が伝わらないことがあります．図16の例では，関連名や関連端名が書かれていなければ，２つのクラスの関係の表現が十分ではなく，誤解を招きやすくなります．

図16 関連に情報をつける

(5) 分かりやすくレイアウトする
作成者の意図の伝わるようなレイアウトに注意しましょう．具体的には，以下のような注意点があります．

- 意味的な結びつきの強いクラスは，近くに配置する
- 左から右，上から下の方向にクラス図が読めるようにクラスを配置する
- なるべく，関連が交差しないようにクラスを配置する

クラス図とJavaコードの対応

図17に示すクラス図は，図18のようなJavaのコードに対応することになります．

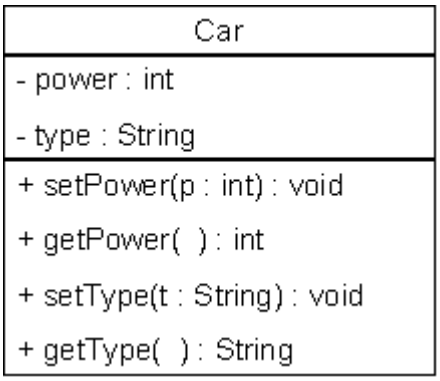


図17 クラス図の例

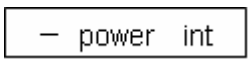
```
public class Car{                                   //1 行目
        private int power;                          //2 行目
        private String type;                        //3 行目

        public void setPower(int p){                //5 行目
                power = p;                          //6 行目
        }                                           //7 行目
        public int getPower( ){                     //8 行目
                return power;                       //9 行目
        }                                           //10 行目
        public void setType(String t){              //11 行目
                type = t;                           //12 行目
        }                                           //13 行目
        public String getType( ){                   //14 行目
                return type;                        //15 行目
        }
    }
```

図18 Javaコードの例

1行目では，クラスCarの宣言をしています．クラスの可視性は基本的にpublicとします．packageの外側から使う場合にはpublicとする必要があるためです．

2行目では属性powerを宣言しています．属性powerは，クラス図に，以下のように書かれています．

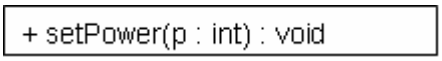


これをJavaの変数の宣言のときと同様に 変数の型 変数名 の順に記述します．最初の「–」は可視性としてprivateを指定していることを表しているので，一番最初には**private**と記述します．続いて属性の型名ですが，クラス図では属性powerの型名はintとされているので，**int**と書きます．また，属性

名はpowerとされているので変数名**power**とします．行の終わりに，セミコロン；を忘れずに．

3行目の属性typeも，クラス図の記述によると，可視性がprivate で，属性の型はString型です．変数名をtypeとし，同様に「**private String type;**」と宣言します．

5行目から７行目では，操作setPowerを定義しています．クラス図には以下のように書かれています．



setPowerの最初の記述は「+」となっています．可視性としてpublicが指定されているので，一番最初に**public**と記述します．すでに学んだように，クラス図においては操作を 操作名（引数名：引数の型）：戻り値の型 の順で書くので，戻り値の型は**void**で，引数の型は**int**，引数名は**p**となります．これをJavaの関数の宣言のときと同様に，戻り値の型 関数名（引数の型 引数名）の順で，記述すると以下のようになります．

```
public void setPower(int p){
        power = p;     <<手続きの部分
}
```

UMLのクラスでは，手続き（メソッド）を記述することができないので，手続きの部分は，Javaコードの記述するときに書き加える必要があります．setPowerは，車の排気量（Power）を属性powerに設定するという操作なので，ここでは，ｐをpowerに代入するという手続きを書いています．

8行目から10行目では，操作getPowerを定義しています．setPower と同様に，クラス図の記述を見て，getPowerの戻り値の型はint，引数はなし，とJavaコードで記述します．getPowerは，設定されている排気量の値を取得するという操作です．

11行目以降では，同様に操作setType，getTypeを定義しています．これらは，車のタイプ，セダンとかワゴンとかクーペなどの値を設定し，取得する操作です．

最後の行では，１行目のクラスの宣言の開き括弧に対応した閉じ括弧をつけ，クラスの宣言を終了します．

セルフテスト クラス図の作成とJavaコードの記述

次のページに移動し，「クラス図の作成とJavaコードの記述」に関するセルフテストを受験して，理解を確認しましょう．

セルフテストは二つありますので，二つとも受験しておきましょう．

まとめ

このモジュールでは，以下のことを学びました.

- クラス図は，クラスの構造やクラス間の関係を記述する
- クラスの構造としては，クラス名，属性，操作を記述でき，属性と操作の可視性は記号で表わす
- クラス間の関係には，構造的な関係を表す「関連」のほか「集約」「コンポジション」「汎化」「依存関係」がある
- クラス間のオブジェクトの接続数を多重度として表す
- クラス図を描く際には，クラスの役割に合ったクラス名をつける，関連にはできるだけ関連端名などの情報をつける，クラスの役割は分散させるなどの点に気をつける
