オブジェクト指向技術 第1回
― オブジェクト指向概説 ―

立命館大学　情報理工学部
槇原　絵里奈
Mail: makihara@fc.ritsumei.ac.jp

講義の流れについて

- 前半：オブジェクト指向の基礎(座学)
  - 担当：槇原絵里奈
- 後半：オブジェクト指向の実演
  - 担当：丸山勝久先生
  - 対象のプログラミング言語：Java
  - 各自Javaが実行できる環境を構築しておいてください

講義評価

- 定期試験：60%
  - 選択式、記述なし
- 平常点：40%
  - 出席点：1点*15回
  - レポート：5点*5回の予定
    - レポートは後半の丸山先生の回で予定しています

講義計画

1(5/22) 月1限	概説	9(6/1) 木1限	プログラミング(1)
2(5/23) 火1限	開発プロセス	10(6/2) 金2限	プログラミング(2)
3(5/24) 水4限	モデリング：静的モデル(1)	11(6/5) 月1限	プログラミング(3)
4(5/25) 木1限	モデリング：静的モデル(3)	12(6/6) 火1限	プログラミング(4)
5(5/26) 金2限	モデリング：静的モデル(3)	13(6/7) 水4限	オブジェクト指向実践(1)
6(5/29) 月1限	モデリング：動的モデル(1)	14(6/8) 木1限	オブジェクト指向実践(2)
7(5/30) 火1限	モデリング：動的モデル(2)	15(6/9) 金2限	後半まとめ
8(5/31) 水4限	前半まとめ		

授業の目的

- オブジェクト指向(object-oriented)の基本概念の理解
- オブジェクト指向モデリング(object-oriented modeling)技法の基本を理解
- オブジェクト指向プログラミング(object-oriented programming)の基本を理解

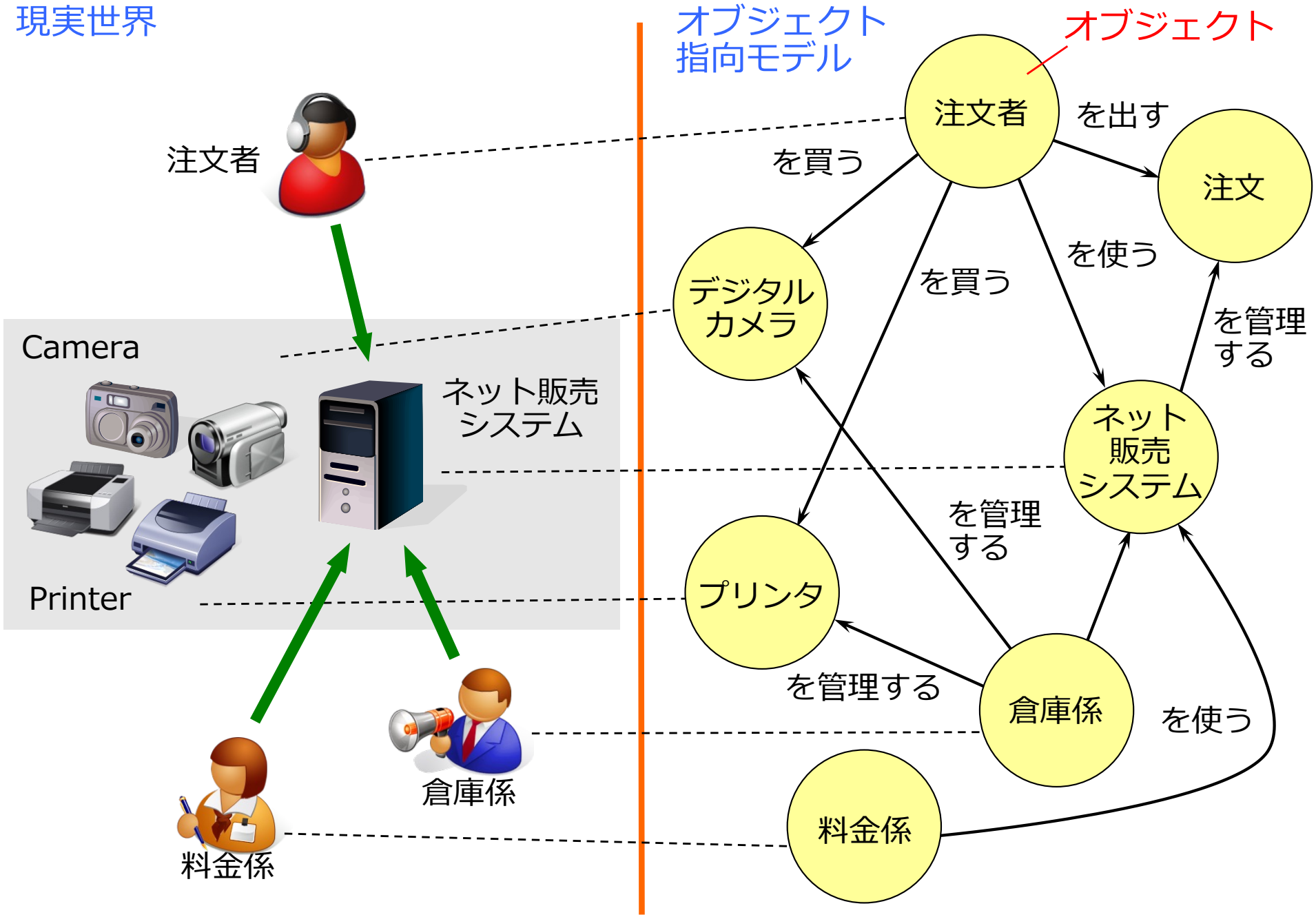
講義内容

- オブジェクト指向概説
  - オブジェクト指向とは
  - クラス、インスタンス、オブジェクト
  - 関連、リンク、継承、集約
- オブジェクト指向プログラミング言語概説
  - オブジェクト指向プログラミング言語
  - プログラミング言語の歴史

オブジェクト指向(object-oriented)とは

- ソフトウェア構築の際の**考え方**の一つ
  - オブジェクト(object)：物
  - -oriented：～指向の、～を重視する
  - オブジェクトによりソフトウェアを構成
- オブジェクト指向技術
  (object-oriented technique)
  - オブジェクト指向でソフトウェアを作る際の関連技術
  - e.g., -設計(OOD)、-プログラミング(OOP)
- オブジェクト指向プログラミング
  (object-oriented programming)
  - プログラムを「オブジェクト」により構築する

オブジェクト指向モデル



オブジェクト指向プログラミング

- オブジェクト指向モデリング
  - 業務分析、要求定義、設計
- UML(unified modeling language)
  - ソフトウェアの構成や振る舞いを**図**で表現
- プログラミング言語
  - Smalltalk、C++、Objective-C、Java、C#、Ruby
- 再利用部品群
  - クラスライブラリ、フレームワーク、コンポーネント、デザインパターン

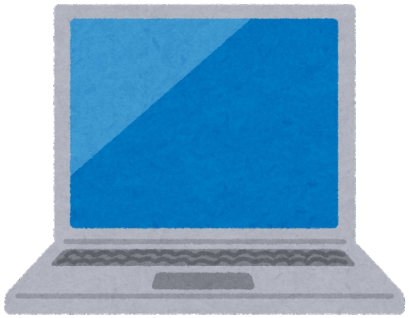
開発者全員で
共通認識を持つ

オブジェクト

- object　＝(辞書的には、)物
  - 人間が認知できる具体的あるいは抽象的な「物」
  - 実世界の物体や役割、概念を抽象化した「物」

- オブジェクトの持つ特性
  - 性質 (state)
    - **オブジェクトの現在の性質や状態**
    - プログラミング言語により属性(attribute)、プロパティ(property)等
  - 動作 (behavior)
    - **オブジェクトが実行できる動作や振る舞い**
    - プログラミング言語により操作(operation)、メソッド(method)等
  - 識別性 (identity)
    - 個々のオブジェクトを区別する手段

属性と操作1



ラップトップA

ラップトップB

オブジェクト
が持つ性質
現在の状態

属性	操作
メーカー	電源を入れる
識別番号	キーボード
型	マウス操作
価格	文字を入力
スペック	文字を表示
…	…

オブジェクト
が実行できる
こと

属性	操作
メーカー	電源を入れる
識別番号	キーボード
型	マウス操作
価格	文字を入力
スペック	文字を表示
…	…

属性と操作

教員

属性
- 教員番号
- 氏名
- 住所
- …

操作
- 教員番号?
- 氏名?
- 話す
- 授業を行う
- …

学生

属性
- 学籍番号
- 氏名
- 住所
- …

操作
- 学籍番号?
- 氏名?
- 立つ
- 座る
- 話す
- 宿題をする
- …

学生

属性
- 学籍番号
- 氏名
- 住所
- …

操作
- 学籍番号?
- 氏名?
- 立つ
- 座る
- 話す
- 宿題をする
- …

練習問題

- 「猫」や「犬」の属性と操作を考えてみよう

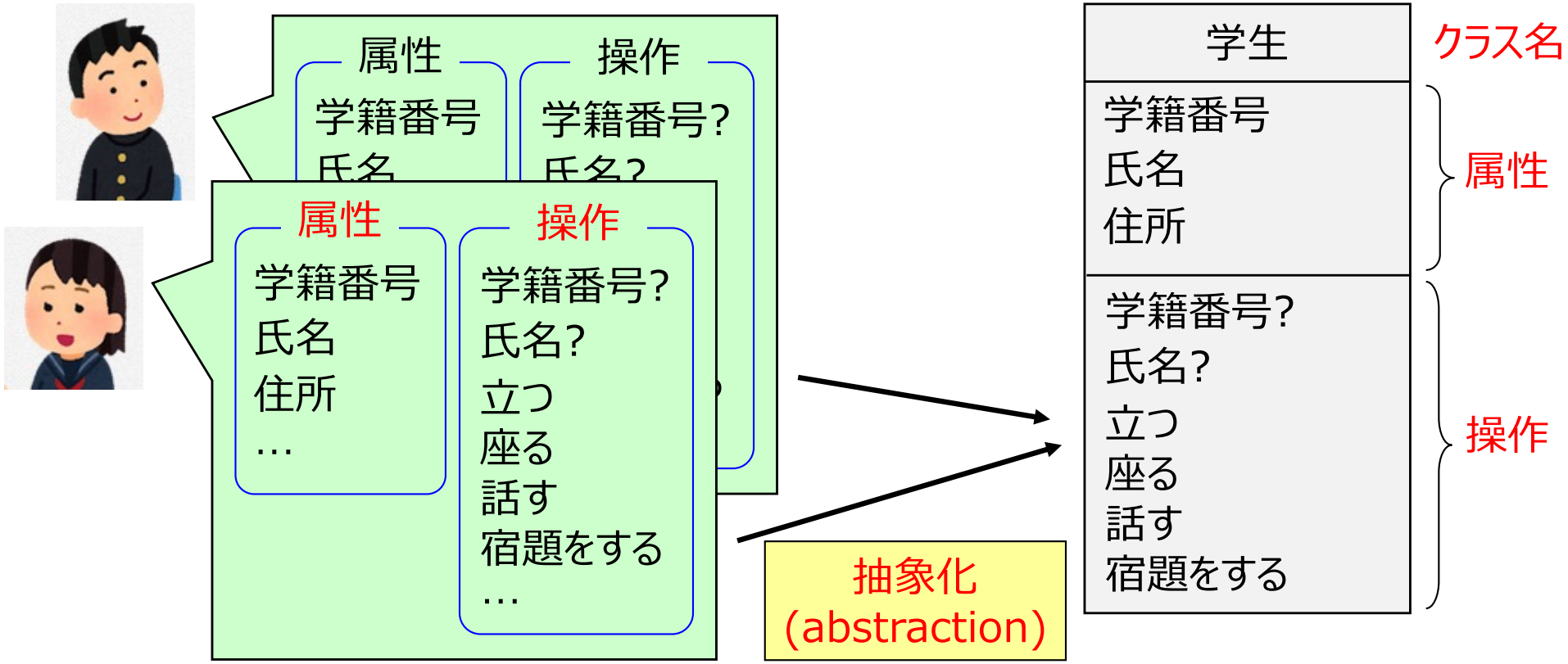
- 身の回りにある「もの」を1つ選択し属性と操作を考えてみよう

オブジェクト指向の基本概念

- **クラス**と**インスタンス**(class, instance)
- **カプセル化(encapsulation)**
  - データとそれに対する処理をまとめてモジュール化
  - オブジェクトの状態を外部から隠蔽（情報隠蔽）
- **メッセージパッシング(message passing)**
  - オブジェクトに処理を依頼する仕組み
- **関連(association)**
  - あるクラスが別のクラスを利用することを表す関係
- **継承(inheritance)**
  - 既存のクラスに属性や操作を追加して新しいクラスを定義する仕組み
- **集約(aggregation)**
  - オブジェクトを構成する部品（部分オブジェクト）を束ねて扱う仕組み

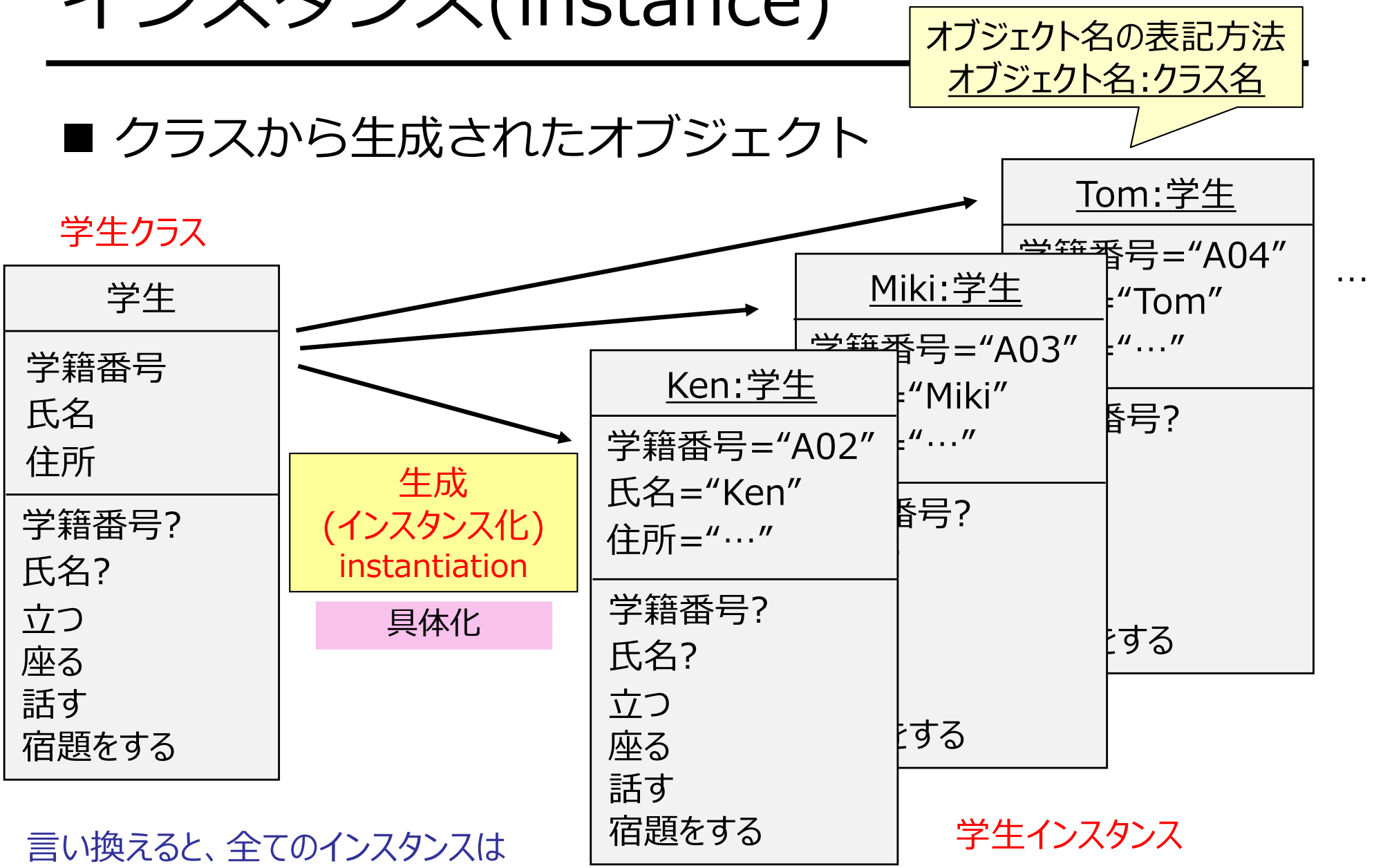
クラス(class)

- 同じ属性と操作を持つオブジェクトを抽象化したひな形(template)
- オブジェクトの設計図



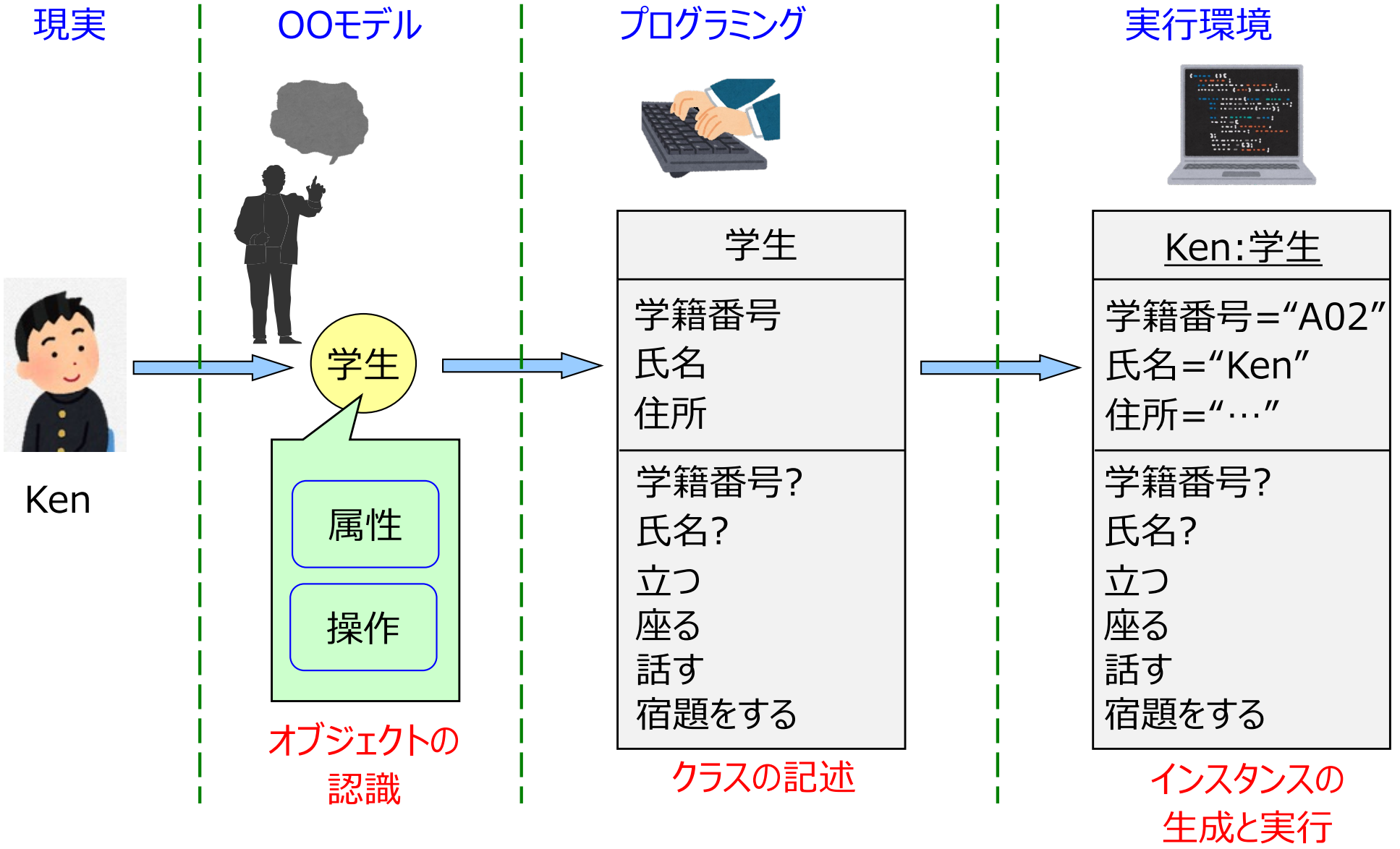
インスタンス(instance)

■ クラスから生成されたオブジェクト



言い換えると、全てのインスタンスは
何かしらのクラスに属する

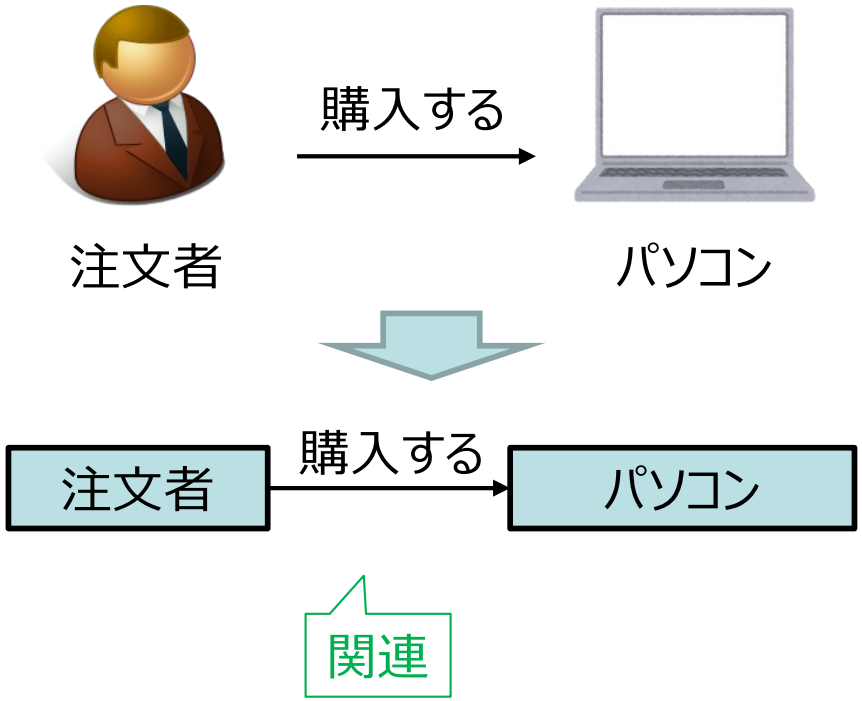
オブジェクト、クラス、インスタンス

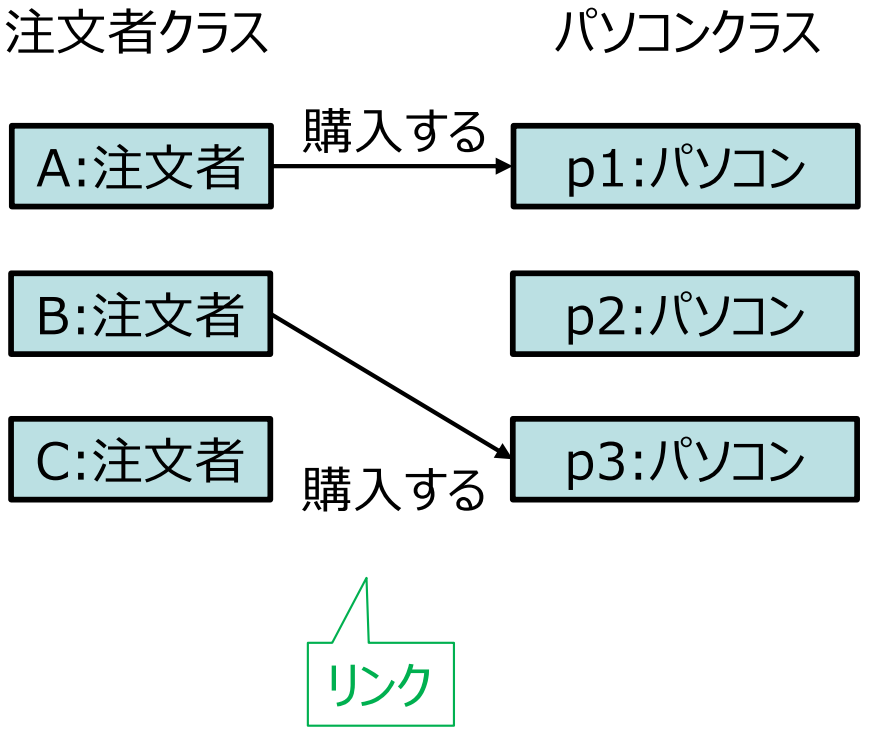


関連 (association)とリンク(link)

■関連：クラス間の利用関係

■リンク：オブジェクト間の利用関係

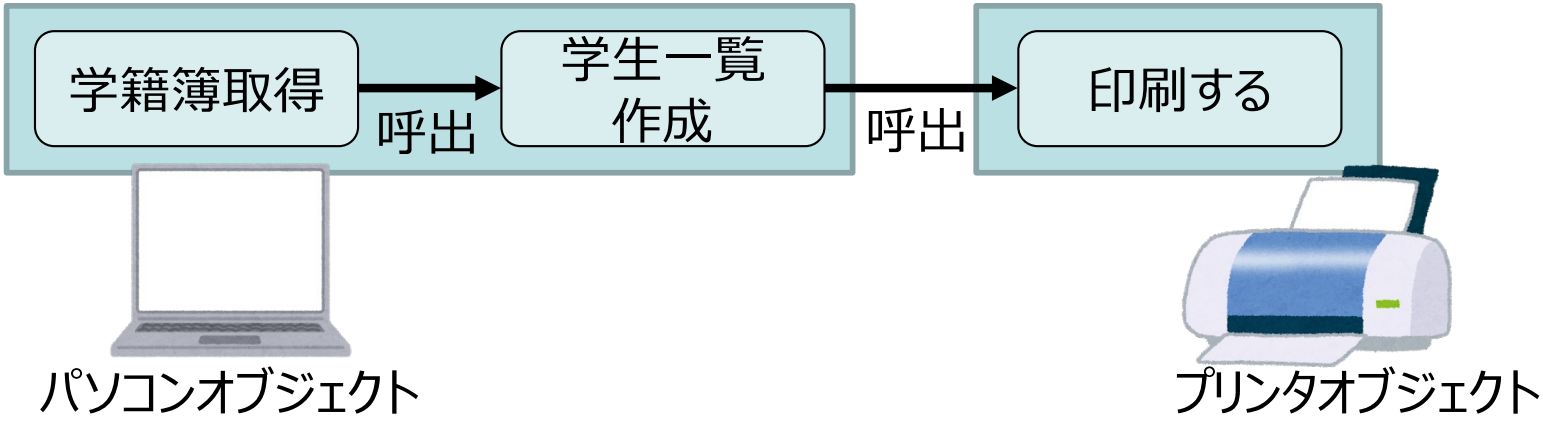




メッセージパッシング (message passing)

■オブジェクトに対して操作の実行を依頼する仕組み

●オブジェクトに対するメソッドの呼び出し



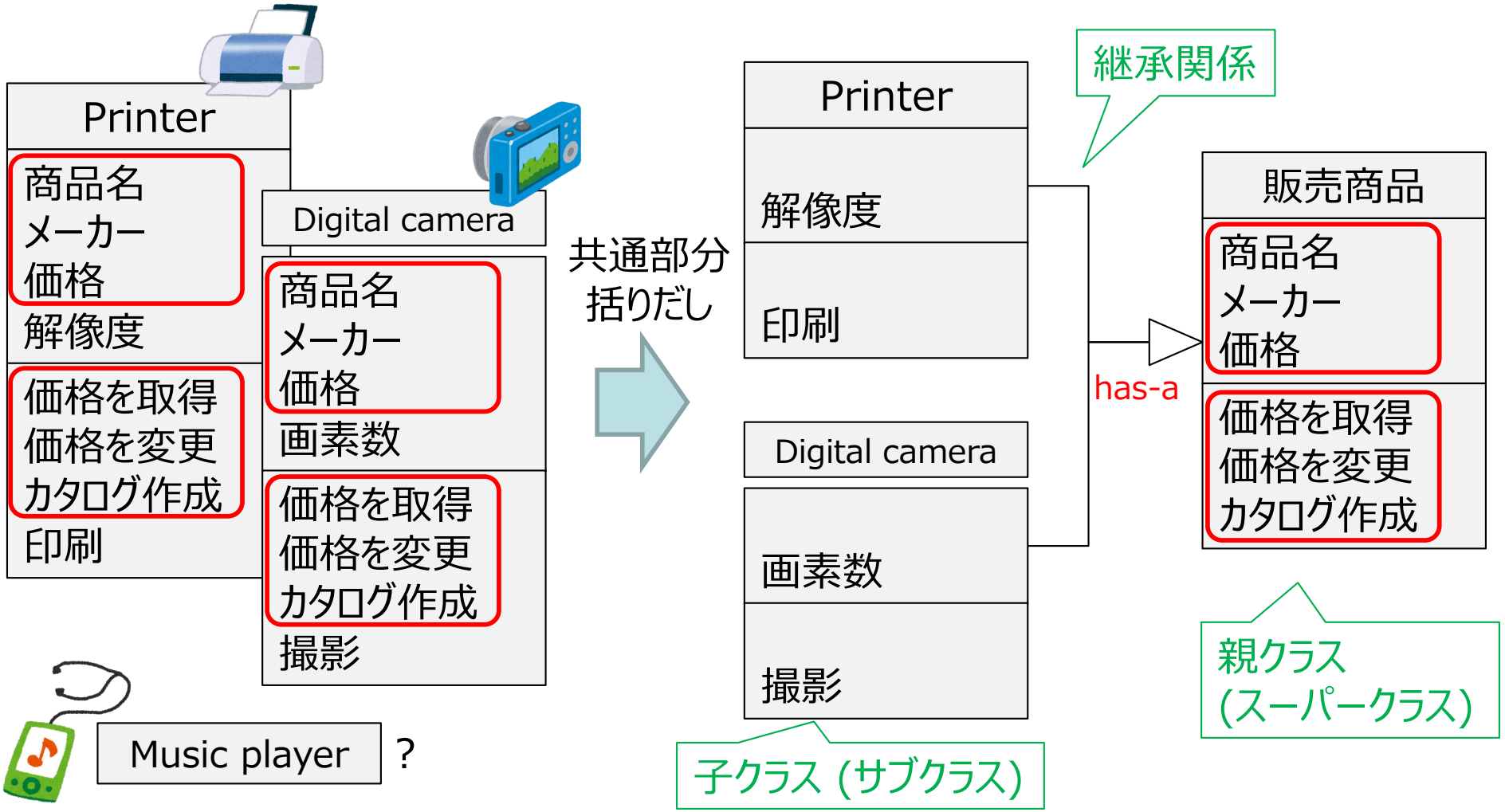
オブジェクトはそれぞれ異なる責任を持つ
(役割が明確に分かれている)

変更が及ぼす影響範囲が
明確かつ最低限にとどまる

継承 (inheritance)

コードの再利用・拡張性
可読性を高める

■ 複数のクラスにおける**共通な性質を共有させる**仕組み

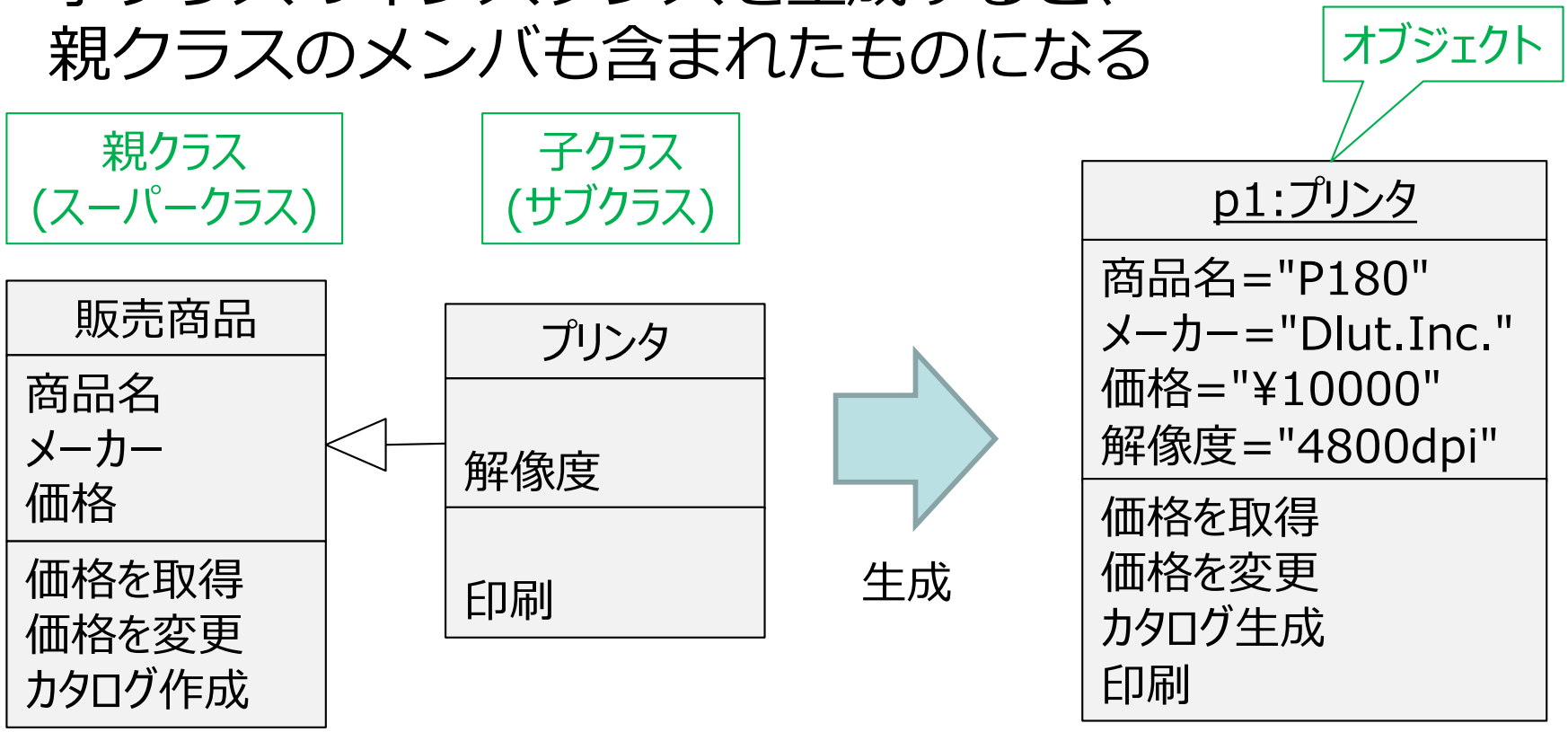


プリンタ:printer　デジカメ:digital camera

[オブジェクト指向]

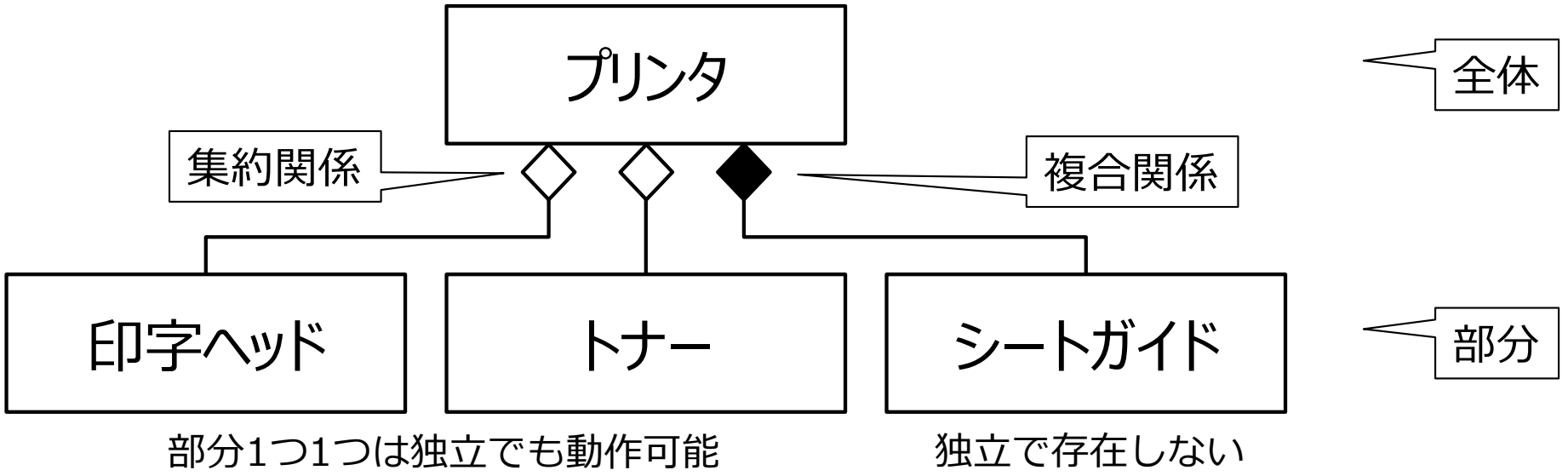
継承とオブジェクトの生成

■ 子クラスのインスタンスを生成すると、
親クラスのメンバも含まれたものになる



集約関係

- 集約(aggregation)：
  複数のオブジェクトを束ねて扱う仕組み
  - 全体とその構成部品の関係
- 複合(composition)：**強い所有関係**を持つ集約
  - 所有物と被所有物の生存期間が一致



確認問題

■ 以下の文の空欄を埋めよ。

- オブジェクト指向では、ソフトウェアを(1)によって構成する。
- オブジェクトの持つ特性として、(2)、(3)、(4)が挙げられる。

■ 以下の説明に合う語句を選択肢から選べ。

5. 同じ属性と操作を持つオブジェクトを抽象化したひな形
6. (5)から生成されたオブジェクト。
7. (5)から(6)を作ること。
8. オブジェクトに対して操作の実行を依頼する仕組み。
9. 子クラスが親クラスの性質を引き継ぐこと。また、その仕組み。
10. 全体とその構成部品の関係。

語群： 関数、ファイル、クラス、インスタンス、
継承、誕生、集約、生成、メッセージパッシング

講義内容

■オブジェクト指向概説
- ●オブジェクト指向とは
- ●クラス、インスタンス、オブジェクト
- ●関連、リンク、継承、集約

■オブジェクト指向プログラミング言語概説
- ●オブジェクト指向プログラミング言語
- ●プログラミング言語の歴史

オブジェクト指向プログラミング言語

- Simula67 (1967年)
  - クラスを導入した最初の言語
- Smalltalk (1972年)
  - アラン・ケイ(Alan Kay)らにより開発
  - すべてをオブジェクトと捉える
  - メッセージパッシングの実装
  - Squeak, Squeak eToys等の派生がある
- C++ (1979年)
  - C言語との互換性有り
  - 実行時性能を重視し、広く普及したが、本来のオブジェクト指向とは異なる

※各言語の誕生年は目安

オブジェクト指向プログラミング言語

- Objective-C (1984年)
  - C言語にSmalltalkのようなメッセージパッシング機構を導入
  - MacOSに標準付属していた(現在もXcodeがサポート)
- Ruby (1993年)
  - まつもとゆきひろにより開発
  - スクリプト言語。動的型付け(型チェックは実行時)
  - 開発者の生産効率向上を目指し、シンタックスシュガーを多数導入
    - シンタックスシュガー(構文糖衣)：コードの読み書きを容易にするための言語の機能

オブジェクト指向プログラミング言語

- Java (1995年)
  - Sun Microsystems社が開発 → Oracleが買収
  - 現時点で最も広く使われている言語の一つ
  - Groovy, Scala, Kotlin等に影響を与えた
- C# (2000年)
  - Microsoft .NET Frameworkの一部
  - Javaの影響を受けている
  - バージョンアップごとに関数型、命令型等多様な言語の機能を取り入れている

確認問題

■以下の各文は正しいか。○か×で答えよ。

- ●オブジェクト指向プログラミング言語として初めて作られた言語はJavaである。
- ●C言語はオブジェクト指向プログラミング言語である。
- ●C#はオブジェクト指向以外の様々な言語の機能を取り入れている。

確認問題

■以下の各文は正しいか。○か×で答えよ。

× ●オブジェクト指向プログラミング言語として初めて作られた言語はJavaである。

× ●C言語はオブジェクト指向プログラミング言語である。

○ ●C#はオブジェクト指向以外の様々な言語の機能を取り入れている。

プログラミング言語の歴史

- 人間が機械語で記述　　01010110...
- アセンブリ言語
  - 機械語を人間がわかるシンボル(記号)に変換　　ADD $s0 $s1 $s2
- 高級言語
  - 直感的なソースコード　　Z=X+Y
  - FORTRAN(1957年), COBOL(1960年)
    C(1972年)
- 1968年 第1回ソフトウェア工学会議
  - ソフトウェア生産性の重要性

人間が理解できる言葉で
機械を操作する

プログラミング言語の歴史

- 構造化プログラミング
  - プログラムを連接・選択・反復という構造で表現
  - 人間にとっての理解性を重視
  - Gotoレスプログラミング
    (C言語等ではGoto文を使わないのが普通)
- サブルーチンの独立性強化
  - 構造化分析等により適切に処理を分割

整備や維持のし易さ

異なるソフトウェアへ
の取り込み・活用

- 保守性と再利用性の改善へ
  - まだグローバル変数の存在と貧弱な再利用が問題
  - (初期のオブジェクト指向言語は計算機性能不足
    のため、あまり広まらなかった)

プログラミング言語の歴史

- オブジェクト指向プログラミング言語
  OOPL: Object-Oriented Programming Language
  - クラス(class)
    - 変数とサブルーチン(関数)をひとまとめに
      - 関数の実行に必要な変数は関数内で保持、守る
      - グローバル変数を使わなくて済む
  - 継承、多様性(Polymorphism)
    - 新しい再利用の単位
    - 重複する機能を1つにまとめる
      - 複数箇所を修正する必要がなくなる

poly: 多様な
morph(ism): 形

OOPLの機能 - まとめる

変数と関数
がバラバラ

```
char[] 氏名、住所;

void change_name(char[] name){
   氏名 = name;
}

void change_address(char[] adr){
   住所 = adr;
}
```

何の氏名？何の住所？

異なる関数で氏名や住所を
書き換えられる
->不適切な値が代入される
　恐れがある

*「学生」は
氏名と住所
のデータを持つ*
ことが分かる

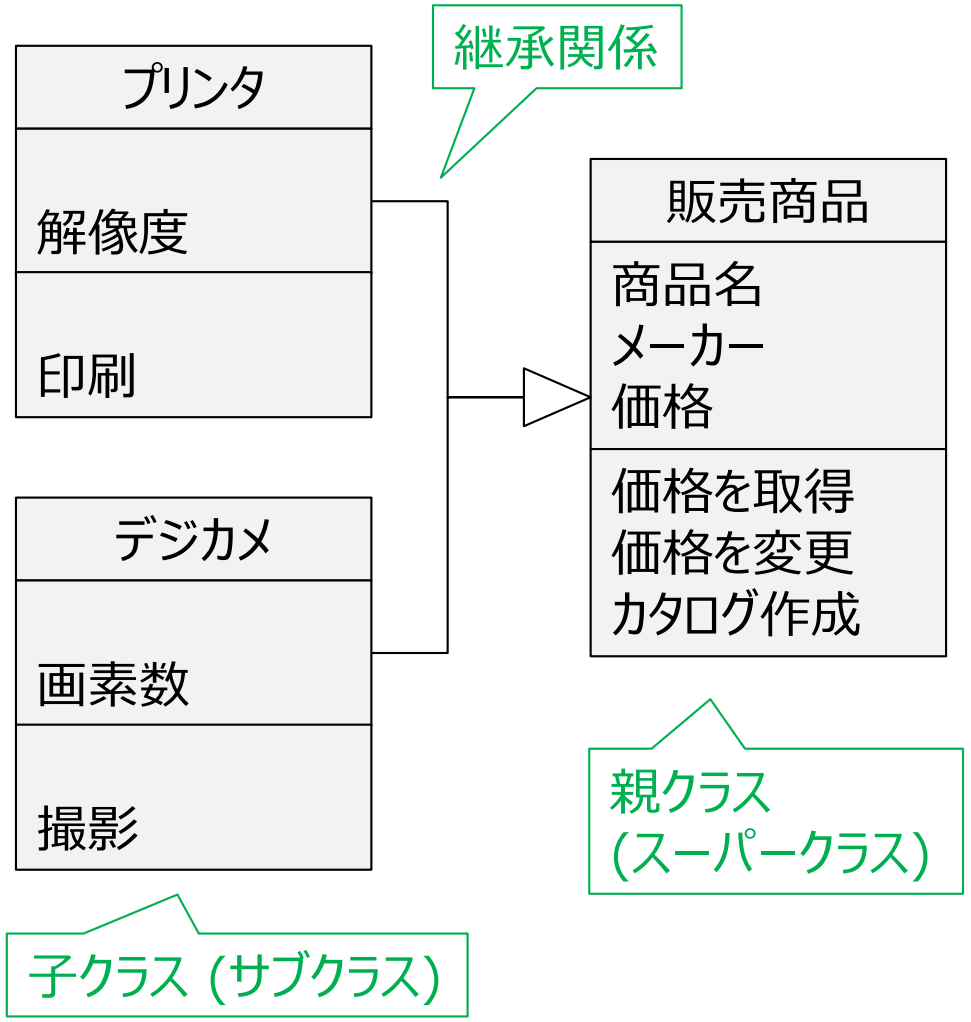
```
class 学生 {
char[] 氏名、住所;

   void change_name(char[] name){
      氏名 = name;
   }

   void change_address(char[] adr){
      住所 = adr;
   }
}
```

氏名や住所は学生クラス内で
操作される

OOPLの機能 - まとめる



プリンタとデジカメの共通部分を
1つにまとめて省略

コードをまとめることで
修正箇所も減らすことができる

OOPLの機能 - 隠す

```
class 学生 {
  private int 学籍番号;
  private char[] 氏名、住所;

  public void change_name(char[] name){
    氏名 = name;
  }

  public void change_address(char[] adr){
    住所 = adr;
  }
}
```

- 変数(フィールド)や関数(メソッド)はアクセスする必要がない場所では**使えなくする**
  - 勝手に中身を書き換えられたり、不適切な値が代入されることを防ぐため
  - 特に保守の際、見なければならない範囲を限定できる

- 可視性(visibility)
  - クラス内部のフィールドやメソッドを他のクラスから見えない(参照できない)よう指定
  - public(どのクラスからも参照可), private(そのクラスのみ参照可), protected(そのクラスとサブクラスは参照可)

確認問題

■以下の各文は正しいか。○か×で答えよ。

- クラスを使うことで、プログラムで扱うデータ(変数)と、そのデータを操作する機能(関数、メソッド等)を1つのモジュールとしてまとめることができる。
- 継承を使うことで、複数のクラスの共通部分を1つの親クラスに抽出することができる。
- プログラム中の変数(フィールド)や機能(関数、メソッド等)は常にプログラムのどこからでも利用可能であることが望ましい。

確認問題

■以下の各文は正しいか。○か×で答えよ。

○
- クラスを使うことで、プログラムで扱うデータ(変数)と、そのデータを操作する機能(関数、メソッド等)を1つのモジュールとしてまとめることができる。

○
- 継承を使うことで、複数のクラスの共通部分を1つの親クラスに抽出することができる。

×
- プログラム中の変数(フィールド)や機能(関数、メソッド等)は常にプログラムのどこからでも利用可能であることが望ましい。