オブジェクト指向論(Q)

オブジェクト指向概論(B1)
オブジェクト指向(K1)

第1回講義
2023/4/10

來村 徳信

今日の講義のトピックと流れ

冒頭：基礎概念の概要

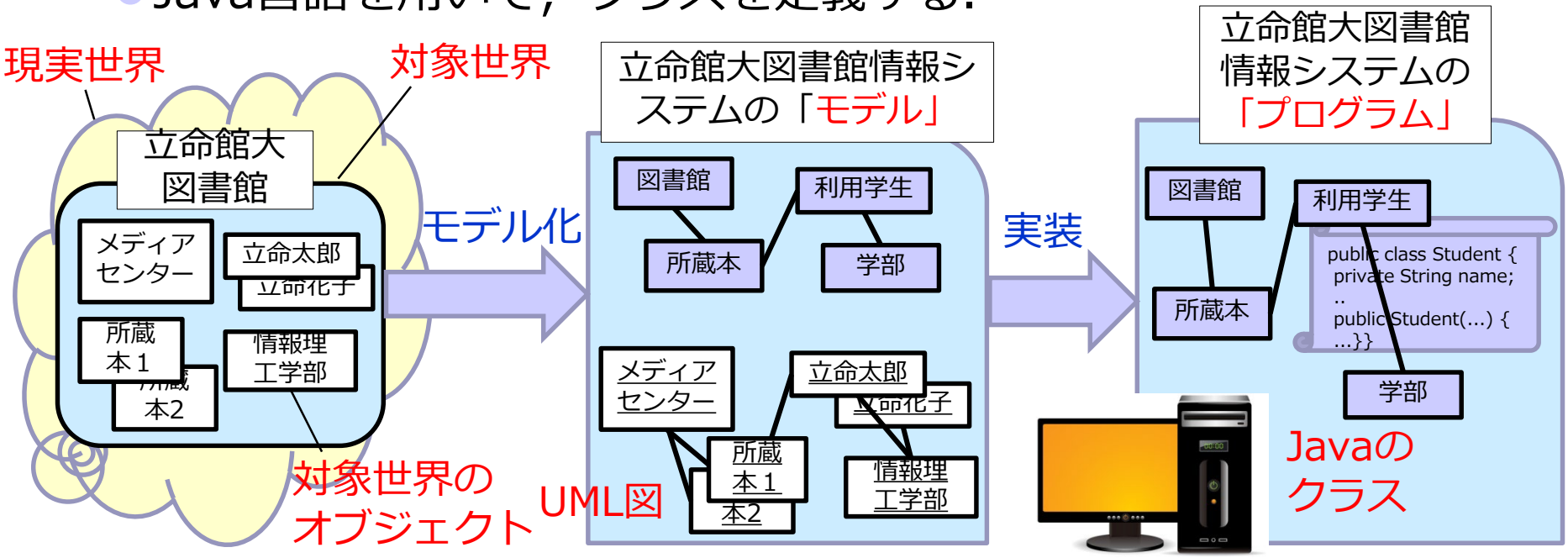
- オブジェクト指向とは？
- オブジェクトとは?
- クラスとインスタンス
  - 次回で詳しく述べるが概要を説明する

後半：オブジェクト指向モデルとは？　なぜ必要なの?

- (1) ソフトウェア開発の難しさ
  - ウオーターフォール型開発における難しさとその原因
  - 解決策の１つが「モデル」の記述
- (2) モデルとは？
  - モデルの種類
- (3) プログラミングにおける役割
  - プログラムとの対応関係

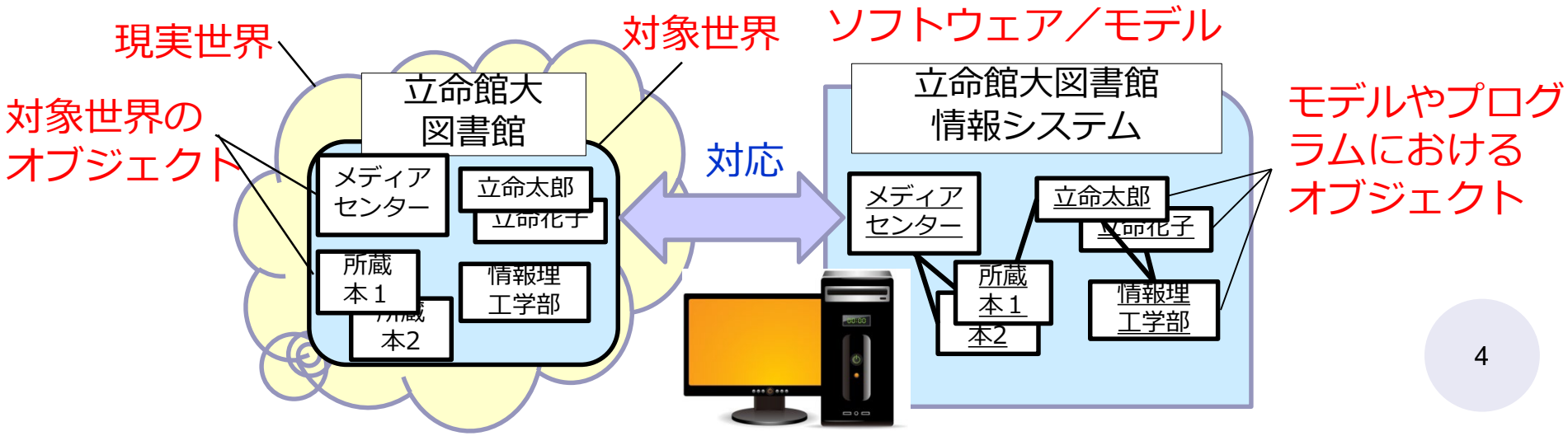
オブジェクト指向とは?

- オブジェクト指向モデリング
  - ソフトを作る前に，設計図として，対象世界のオブジェクトを表すような「モデル」を記述する.
  - UMLの図（ダイアグラム）として記述する.
- オブジェクト指向プログラミング
  - モデルに基づいて，オブジェクトを表す部品の集まりとして，「プログラム」を実装する.
  - Java言語を用いて，クラスを定義する.



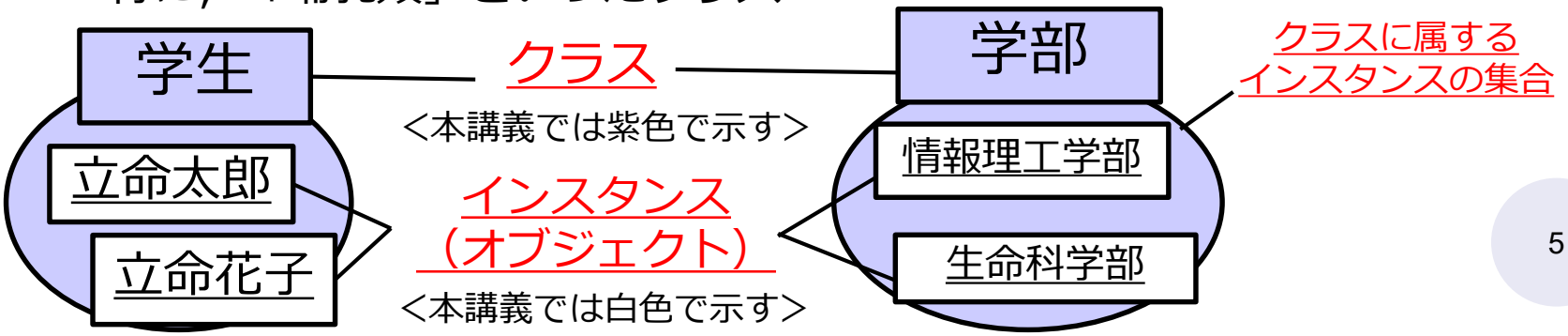
オブジェクト(object)とは

- オブジェクト ≒ 対象世界に存在する「モノ」または「コト」
  - 物理的に触れる「モノ」，例：本1(図書館にある本), 立命太郎(学生)
  - 物理的に触れない「モノ」，例：情報理工学部，立命館大図書館
  - 物理的に触れない「コト」，例：立命太郎が本1を借出した，この講義
  - 「性質」を持つ．例：本1の題目は「やさしいJava」
- 対象世界＝ソフトウェアが対象としている「現実世界の範囲」
- モデルやソフトウェアのオブジェクト
  - モデルやプログラムで，対象世界のオブジェクトに対応する要素



クラスとインスタンス（概要）

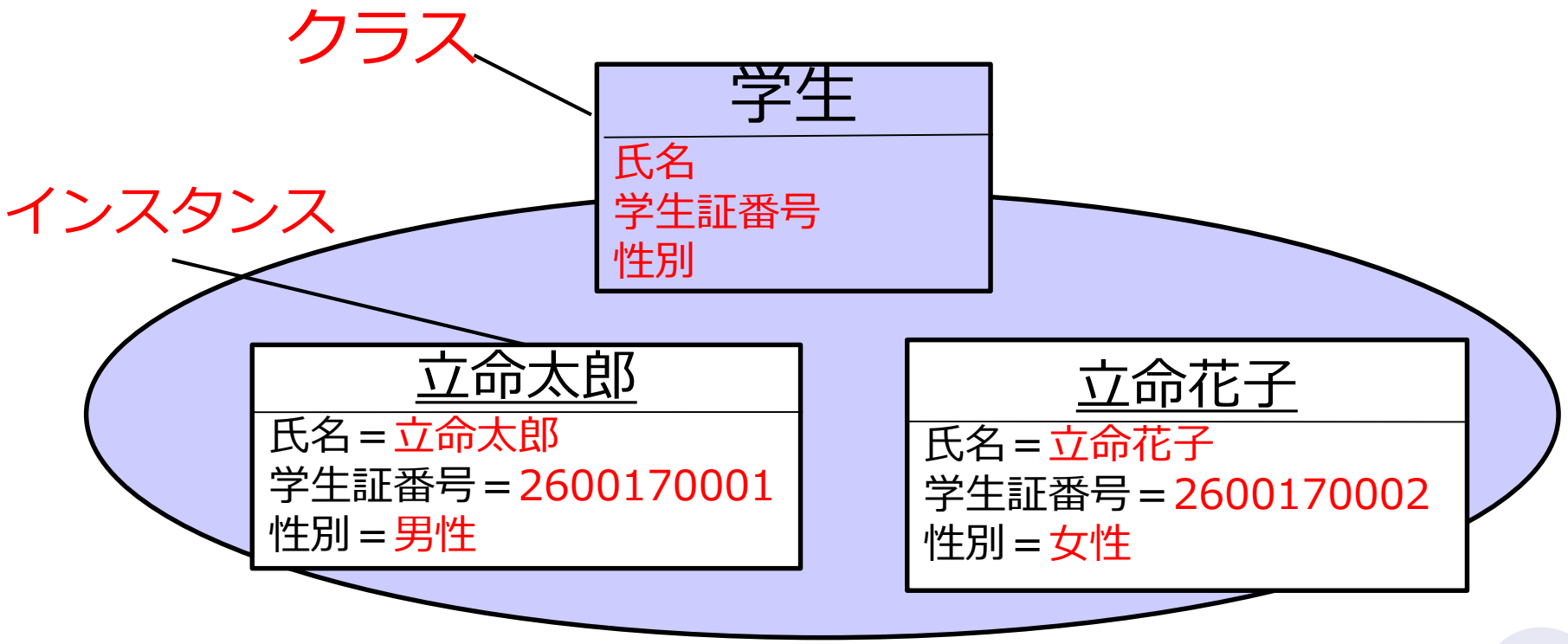
- インスタンス(instance) ≒ 個物
  - 現実世界に存在する個別のモノ・コト．実例．具体的．
  - オブジェクト ≒ インスタンス
    - 単にオブジェクトといった場合，インスタンスを指す（ことが多い）
    - 特にUML図の「オブジェクト図」はインスタンスの図を表す
- クラス(class)
  - 「共通の性質」を持つインスタンスの「集合」に対応する．
    - あるインスタンス e_1 はあるクラス C に「属する」という．「$e_1 \in C$」
    - 集合論的には，インスタンスは「集合の要素」
  - 「型(type)」≒「種類(kind)」≒「概念」を表す．
    - 型 ≒ インスタンスを作るときの鋳型 (cf. たいやき器の鉄板型)
  - 抽象的：実例ではない．物理的には触れない．
    - 特に，「哺乳類」といったクラス



クラスとインスタンス(2)

- クラスは「共通」な性質（例：属性）を持つ.
- インスタンスは「個別」の性質を持つ（例：属性値）
- というより…,「共通な性質を見いだしたものがクラス」

→詳しくは次回講義で



クラスとして捉える意義：概要

- 一般的に：
  - 個物をグループ化して，まとめて性質や処理を書きたい.
- グループ化（集合化）の意義
  - 人間の認知において：（本講義のメインテーマではないが）
    - 一般的な概念（例：哺乳類）とはそもそもそういうもの.
      - 例：「学生」という概念は「教育機関に所属して，学んでいる人間」というグループに概ね対応する.
  - モデリングにおいて：（本講義の前半部）
    - 対象世界の個物をどのようにグループ化するかをモデルとして明示化することで，対象世界の理解を共有できる（今回講義）.
  - プログラミングにおいて：一度でまとめて書ける（本講義の後半部）
    - あるインスタンス集合に適用可能なようにプログラムを書ける.
    - プログラムの記述量が減る（労力の削減）
      - 特に，上位ー下位クラスの階層があると，上位クラスで記述しておけば継承されるため，記述量が減る

  → オブジェクト指向プログラミングの動機のひとつ

今日の講義のトピックと流れ（再掲）

冒頭：基礎概念の概要

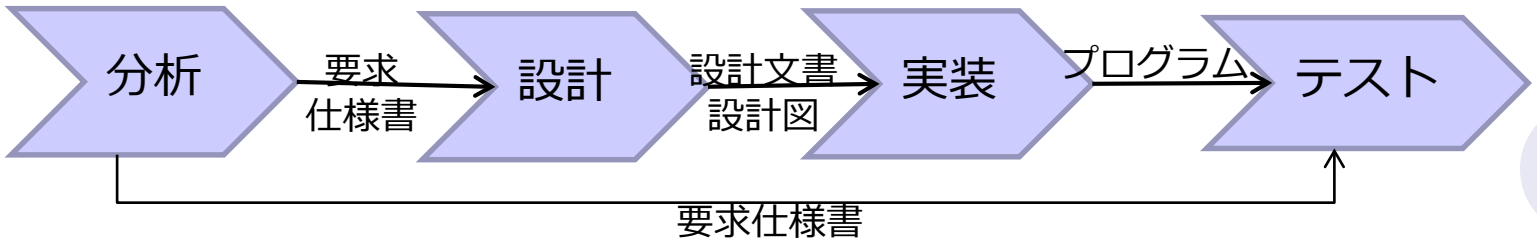
- オブジェクト指向とは？
- オブジェクトとは?
- クラスとインスタンス
  - 次回で詳しく述べるが概要を説明する

後半：オブジェクト指向モデルとは？　なぜ必要なの?

- (1) ソフトウェア開発の難しさ
  - ウオーターフォール型開発における難しさとその原因
  - 解決策の１つが「モデル」の記述
- (2) モデルとは？
  - モデルの種類
- (3) プログラミングにおける役割
  - プログラムとの対応関係

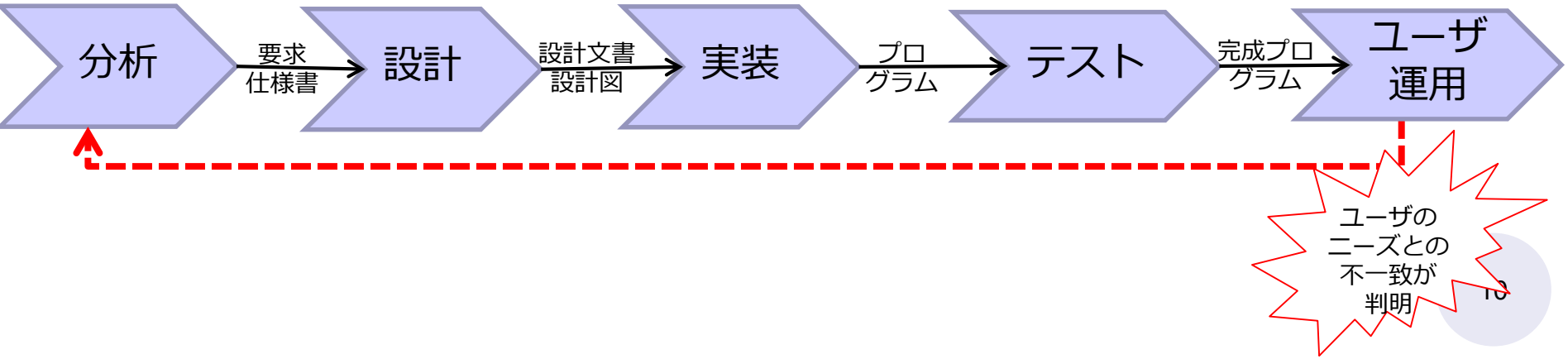
ソフトウェア開発のフェイズ

- 分析フェイズ
  - 要求分析：ソフトウェアの「要求仕様」を決定する
    - どのような機能を持てばよいか．ユーザと開発者の共同作業．
- 設計フェイズ
  - 要求仕様を満たす「ソフトウェア構成」を決定する
  - 出力物：ソフトウェア構成を表す文書や図（設計書／図）
    - モジュール表，画面インターフェイス仕様，フローチャートなど
- 実装フェイズ
  - プログラムを記述する
- テストフェイズ
  - 仕様を満たしていることを確認する



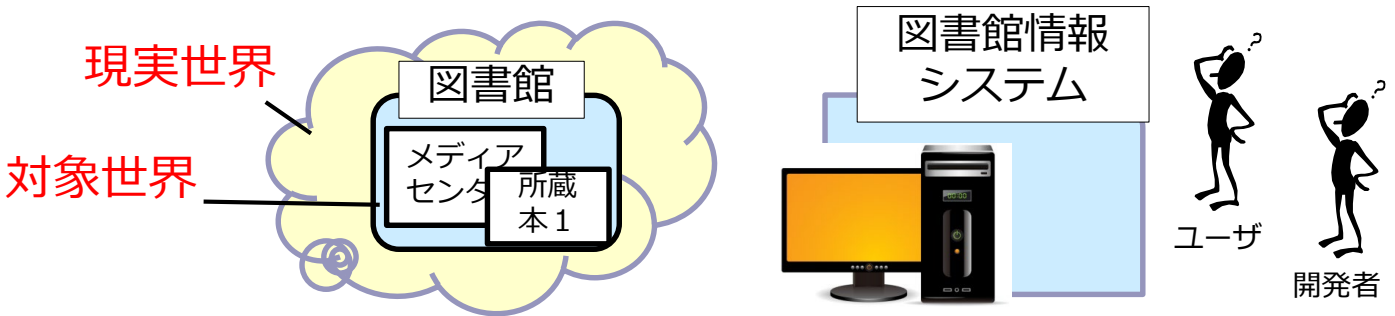
典型的なソフトウェア開発プロセス

- ウォーターフォール型プロセス
  - 分析-設計-実装-テストの各フェイズを順番に1回で行う
    - 各フェイスの出力物に基づいて次のフェイズが行われる
  - うまく行けば理想的
  - 現実にはうまく行かないことが多い.
    - 例：完成したプログラムをユーザに使わせたら，こんな機能が欲しかったんじゃない，と言われる．分析フェイズからやり直すことになり，開発時間／費用が増大．（要求変動）



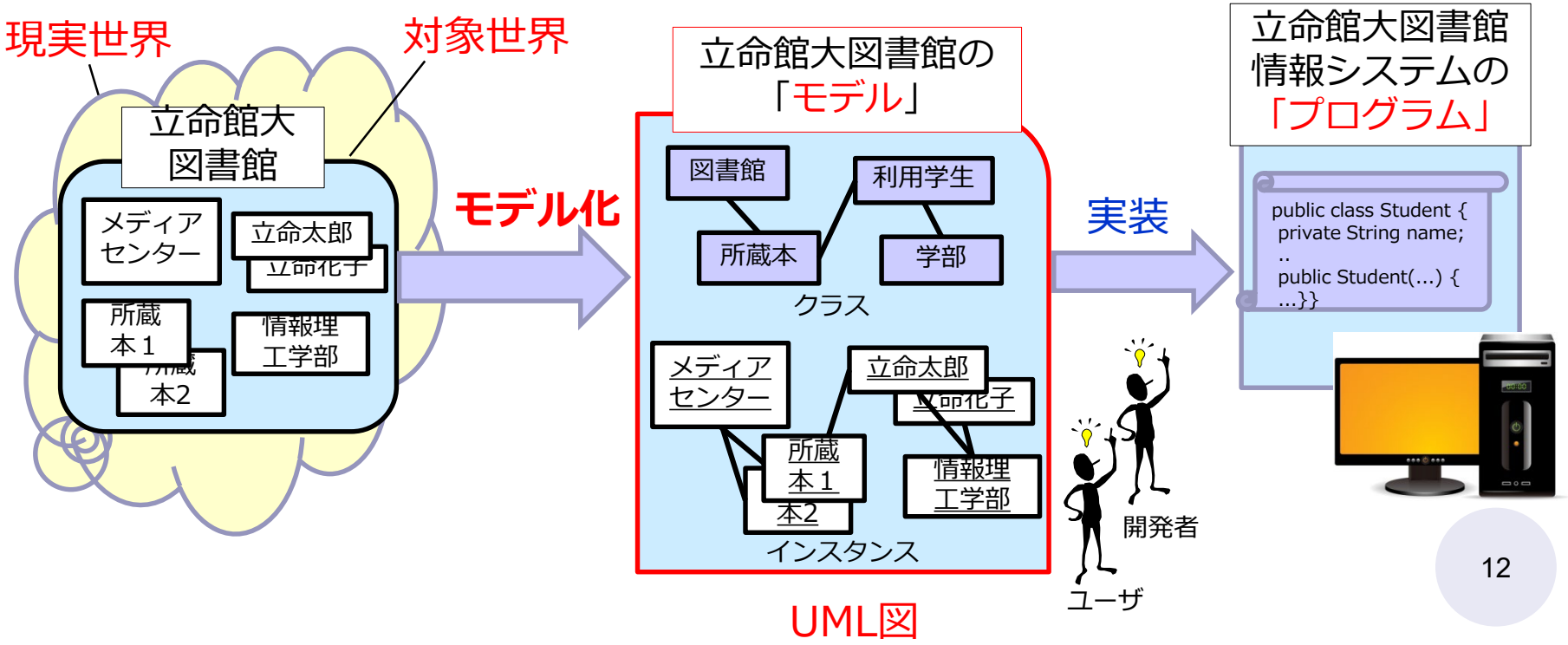
困難さの原因

- ユーザとの「意思疎通」が難しい
  - ユーザのニーズが不明確
    - ユーザもうまく表現できない．すべてを文章化するのは無理．
    - 文章に曖昧さがあり，開発者が理解できない
  - ユーザの（業務上の）ルールが暗黙的
    - ユーザには当たり前なので他人に伝える必要性が分からない
    - 開発者は理解していないのでプログラムもそれを満たさない
- 開発者間の「意思疎通」が難しい
  - 大規模ソフトウェアは複数会社のチーム（大人数）で開発．
  - オフショア開発（海外の会社に開発を委託）も多い．
- 根源的原因
  - 対象世界と情報システムの「理解の共有」が難しい



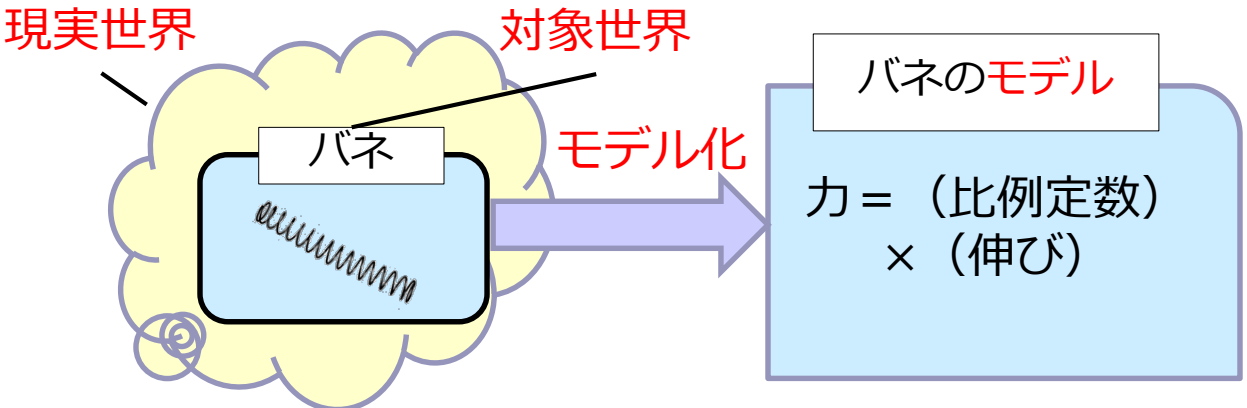
解決策: モデルの記述と利用

- 関係者全員が「分かる」モデルを作る
  - 「直感的」かつ「明確」に全員が分かる.
    - 文章ではなく「構造化」（要素とその関係を明確化）する.
    - 統一された方法で → Unified Modeling Language (ISO/JIS)
  - 「理解を共有」できる



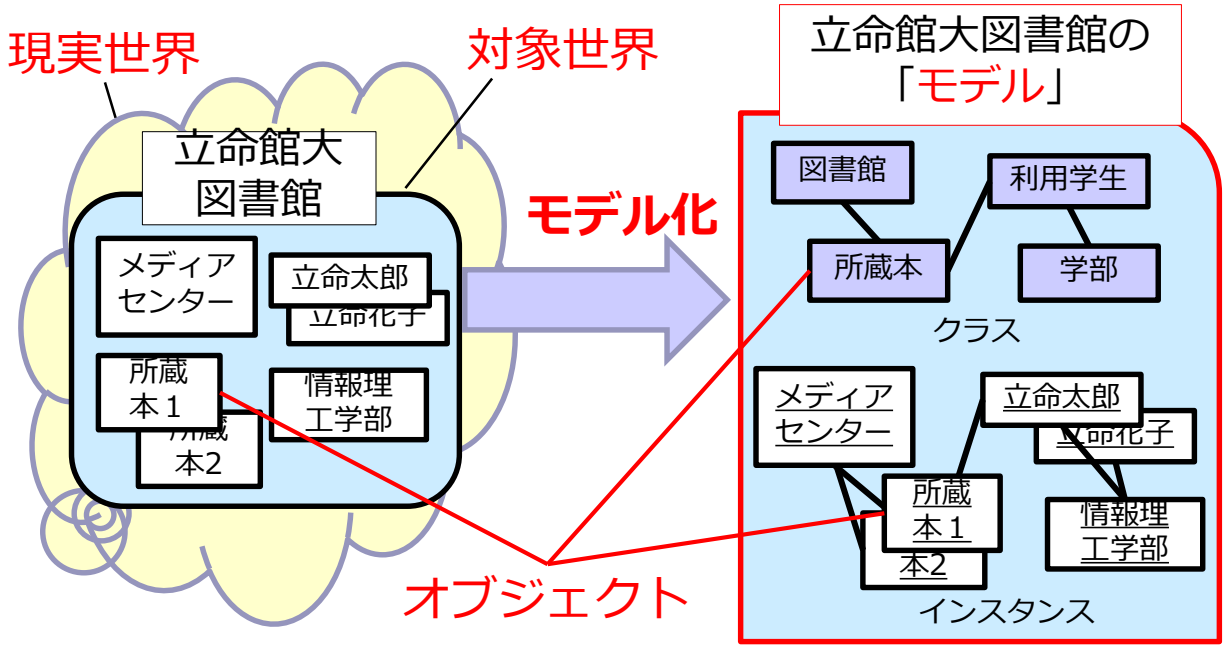
モデルとは? （自然科学における）

- 簡単には「対象世界の本質を表す簡単化した記述」
  - (1) 対象世界の「本質的な」性質や構造などを表している.
  - (2)「簡単化」(範囲限定, 単純化/近似, 規模縮小）されている
  - (3) 対象世界の要素とモデルの要素の「対応が付く」
  - 例：
    - DNAの構造は二重らせんモデルで表される
    - バネの伸びはフックの法則でモデル化できる（数理モデル）
    - プラモデル（模型）にも近い（三次元構造だけ．縮小）
  - 違う意味の「モデル」
    - モデル校（手本となるもの），キャリアモデル（模範），ファッションモデルさん，iPad 2018年モデル（型式）



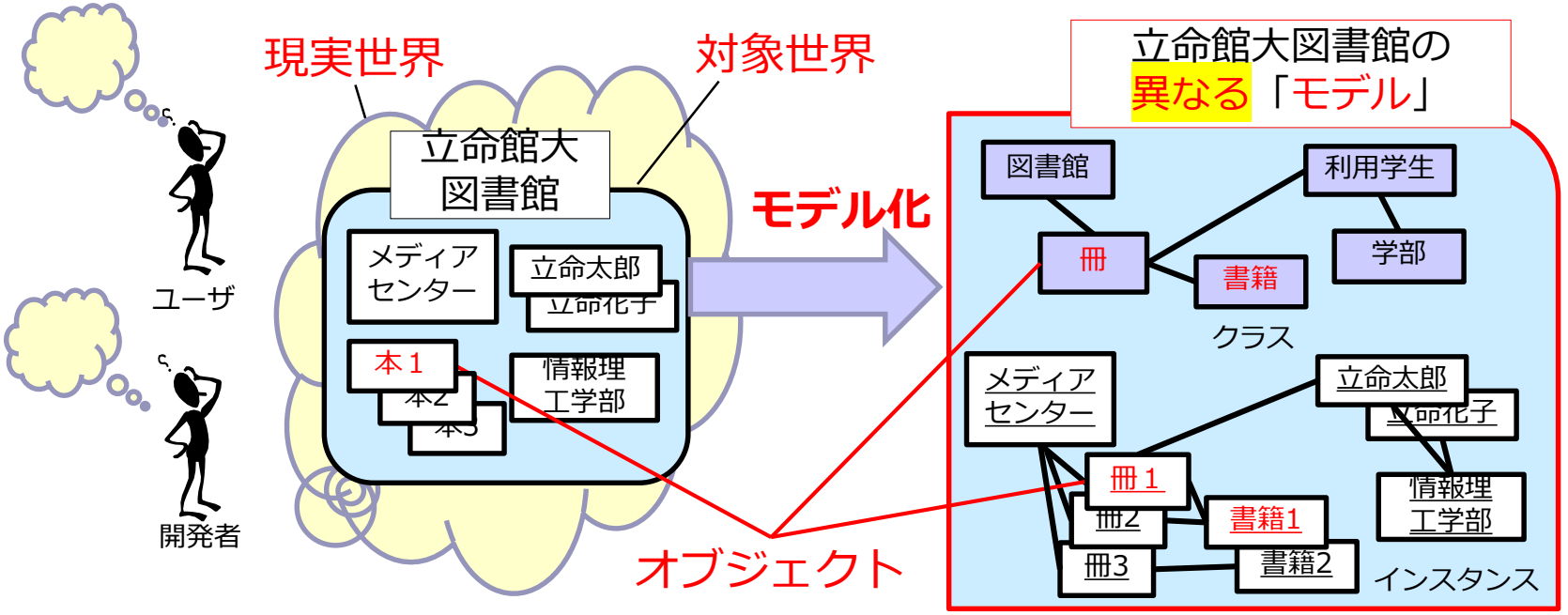
オブジェクト指向モデルとは?

- オブジェクトを単位として対象世界を捉えたモデル
  - (1) 対象世界の本質的な性質や構造などを表している
    - 例：「所蔵本」は特定の「図書館」に配架されていることが表現されている
  - (2) 簡単化（範囲限定, 単純化/近似, 規模縮小）されている
    - 例：図書館の外は表現されていない．本の表紙の色は表現されていない.
  - (3) 対象世界の要素とモデルの要素の対応が付く
    - 「オブジェクト」という単位で,
      対象世界の存在とモデルの要素の 対応が付く



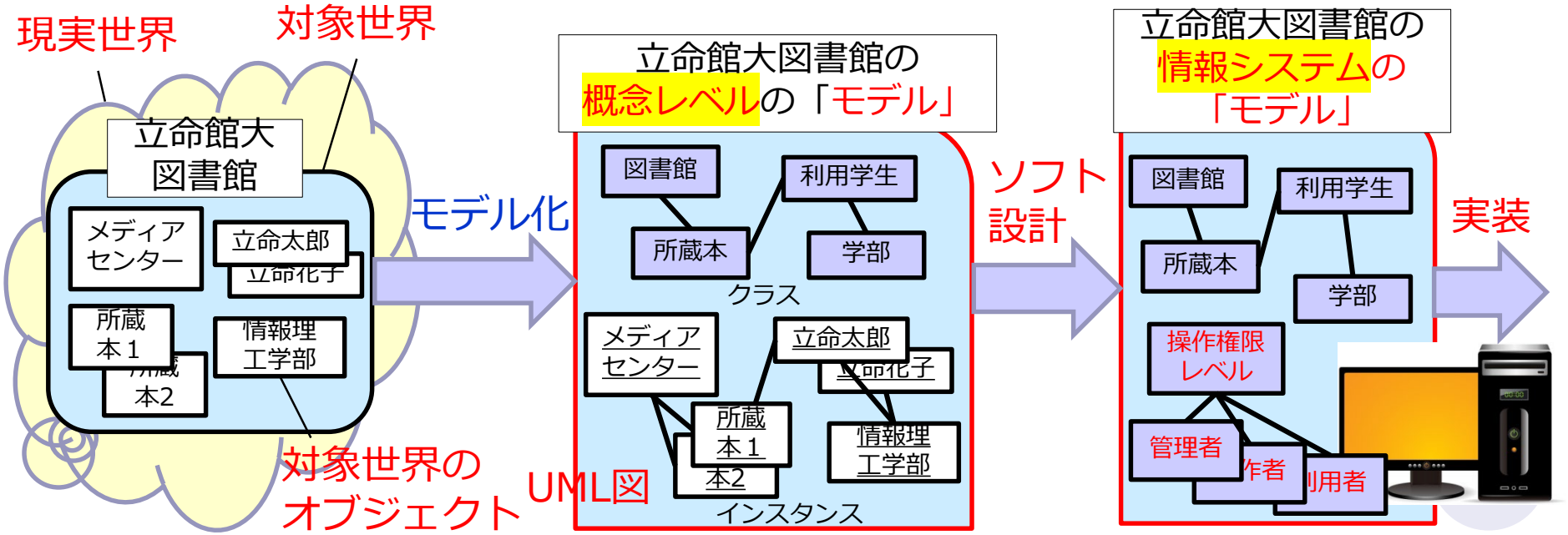
モデルの注意点

- 同じ対象世界でもさまざまなモデルがありえる
  - 例：運動と力のモデル
- 正確には，モデルは「ある人にとっての，ある状況，あるいはある状況についての概念の『明示的な解釈』」[Wilson] である
  - 人によって異なる．頭のなかにあるだけでは他人と共有できない．
  - 「解釈」を明示化する（モデル化する）ことで初めて共有できる．
  - 「オブジェクト指向」は解釈の仕方（対象の捉え方）の一つ
    - ≒オブジェクト指向は「考え方」である



モデルのレベル

- 同じ対象世界にはさまざまなレベルや観点のモデルがある
- 情報システム開発用のモデルのレベル：
  - (1) 概念レベル ≒ 対象世界のモデル
    - 対象世界の分析をするレベル．対象世界になにがあるかなどを表す．
  - (2) ソフトウェアレベル ≒ 情報システムのモデル
    - 情報システムの仕様や設計のレベル
- 本講義では簡単のためには両者を区別しない



オブジェクト指向開発プロセス

- 分析-設計-実装（Waterfall 型）

オブジェクト指向モデリング

オブジェクト指向プログラミング

分析フェイズ

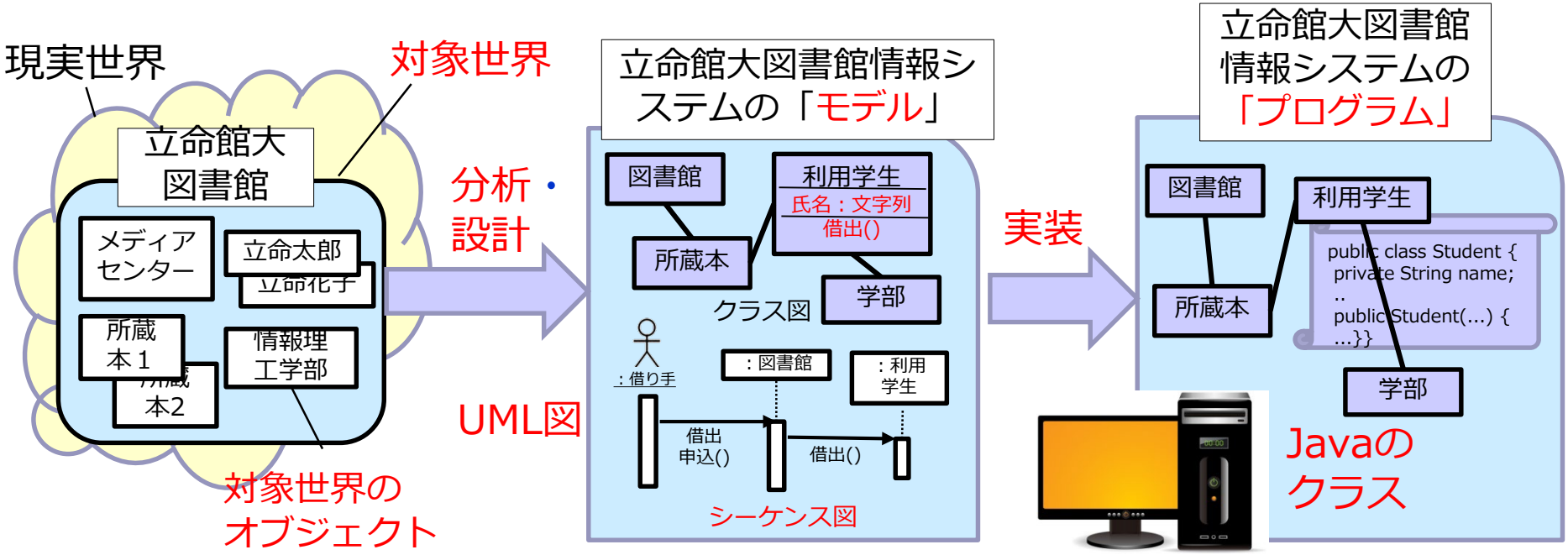
対象世界にどのようなオブジェクトの種類（＝クラス）が存在し，どのような処理が必要かを概念的に分析する．（要求分析を含む）．

設計フェイズ

ソフトウェア的観点から，クラスの属性やクラス間の関連を定義し，処理の流れを明確に記述する．

実装フェイズ

オブジェクト指向型プログラミング言語のクラスを定義する．属性を宣言し，処理をプログラムする．



モデルの分類の観点

- 「時間的」観点からの分類：
  - (1) 「静的」モデル（UML：「構造図」）
    - 対象世界になにが存在し，どのような関係があるかなどを表す．
    - 時間を気にしない．存在しうるオブジェクトや関係を全部，記述する
    - オブジェクトの「性質」を表す
  - (2) 「動的」モデル（UML：「振る舞い図」）
    - 時間が流れている
    - 対象世界やソフトウェアがどのように動くか（振る舞い）を表す
    - オブジェクトの「動作／処理／相互作用」を表す
- 「内部／外部」の観点からの分類：
  - (1) 「外部的」観点からのモデル
    - 情報システムを外部（ユーザの観点）から見たモデル．「機能」を表す．
    - 情報システムが内部的にどう動くかから独立
    - 分析フェイズや設計フェイズで用いられる
  - (2) 「内部的」観点からのモデル
    - 情報システムが内部的にどう動くかを表す
    - 設計フェイズの後半の詳細設計で用いられる

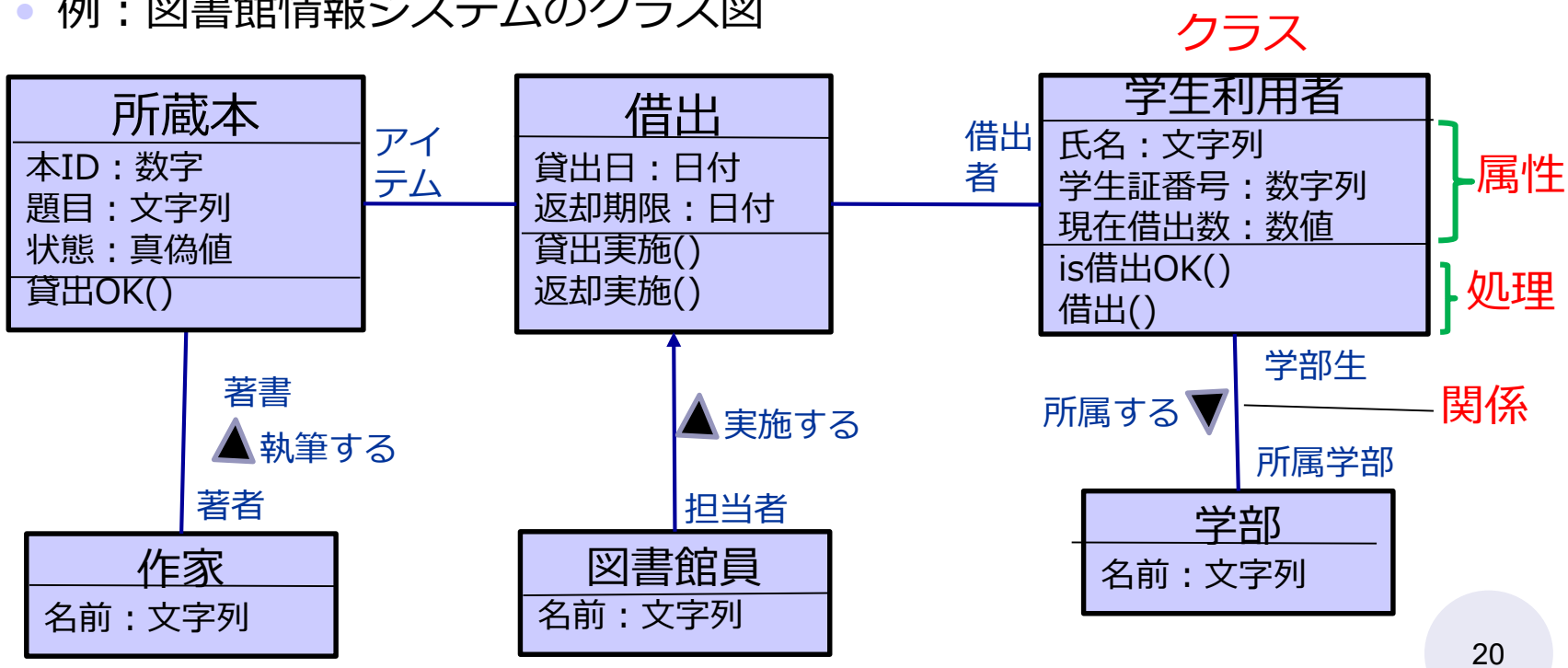
UMLのモデル図の種類

- 13種類ある
- 「構造図」　（＝静的モデル）
  - クラス図，オブジェクト図，パッケージ図，コンポーネント図，コンポジット構造図，配置図
- 「振る舞い図」　（＝ 動的モデル）
  - 「相互作用図」
    - シーケンス図，コミュニケーション図，相互作用図，タイミング図
  - ユースケース図，アクティビティ図，ステートマシン図

- 本講義では下線のモデル図（のみ）を扱う．

静的モデル

- 性質／用途
  - 対象世界に存在するオブジェクトとそれらの関係を表す．
  - オブジェクトの属性・関係・処理を表す．時間は気にしない．
  - 分析フェイズや設計フェイズで用いられる
  - 本講義では静的モデルの外部的観点と内部的観点は区別しない
- UML図：クラス図，オブジェクト図
  - 例：図書館情報システムのクラス図

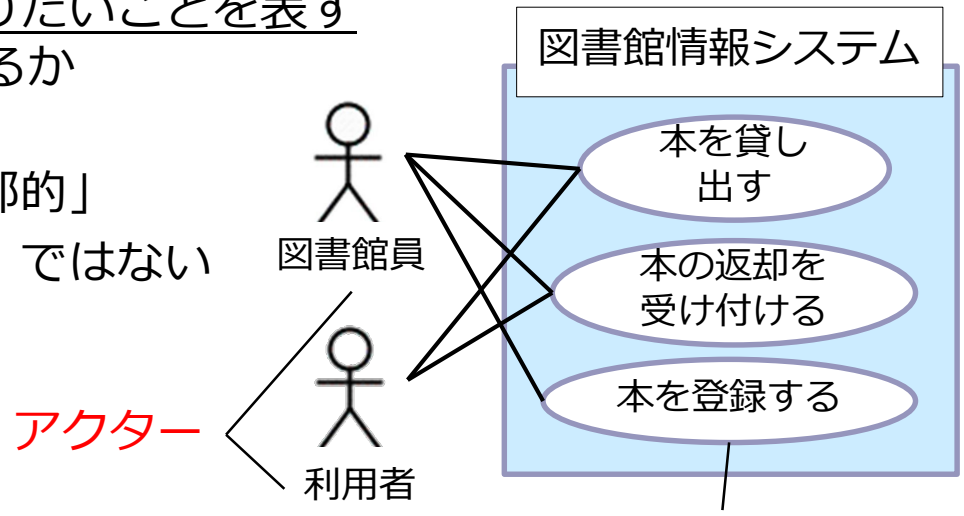


※イメージをつかんでもらうための例です．正確ではないし，今は理解できなくてOK

外部的観点からの動的モデル

- 性質／用途
  - 情報システムの「時間的振る舞い」を外部（ユーザの観点）から見たモデル
  - システムの「機能」を表す．分析フェイズや設計フェイズで用いられる
- UML図：ユースケース図
  - ユーザがシステムを「使用」する「シナリオ」を表す
  - 「アクター」　：システムの利用者（人間・組織・他のシステム）の役割
  - アクターがシステムを用いてやりたいことを表す
    ≒ システムがその際になにをするか
    ≒ システムの「機能」
    - 「なにが」できるか＝「外部的」
    - 「どのように」＝「内部的」ではない

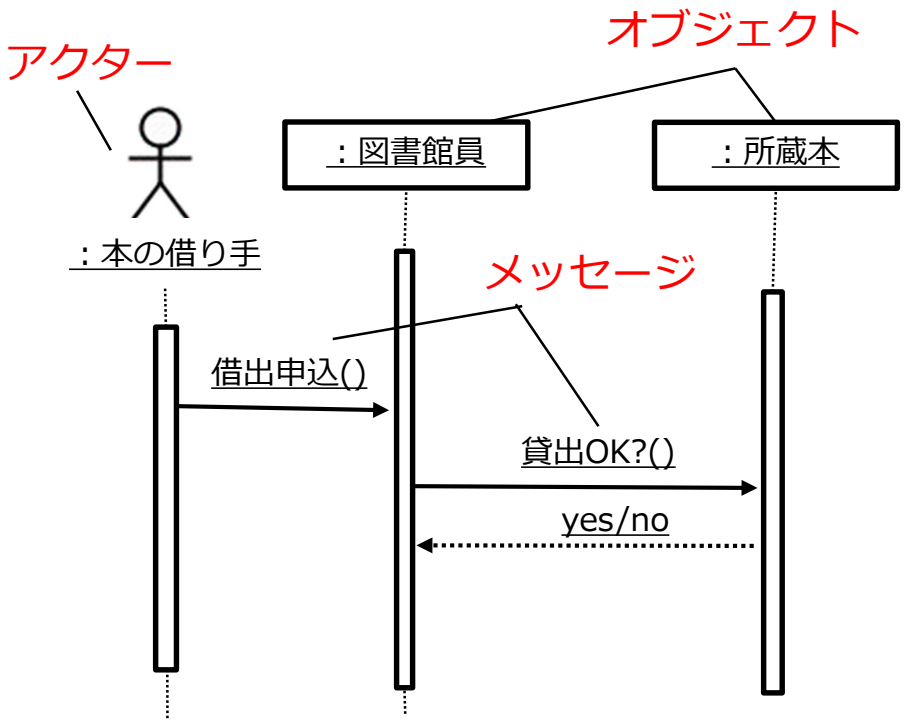
ユースケース図の例



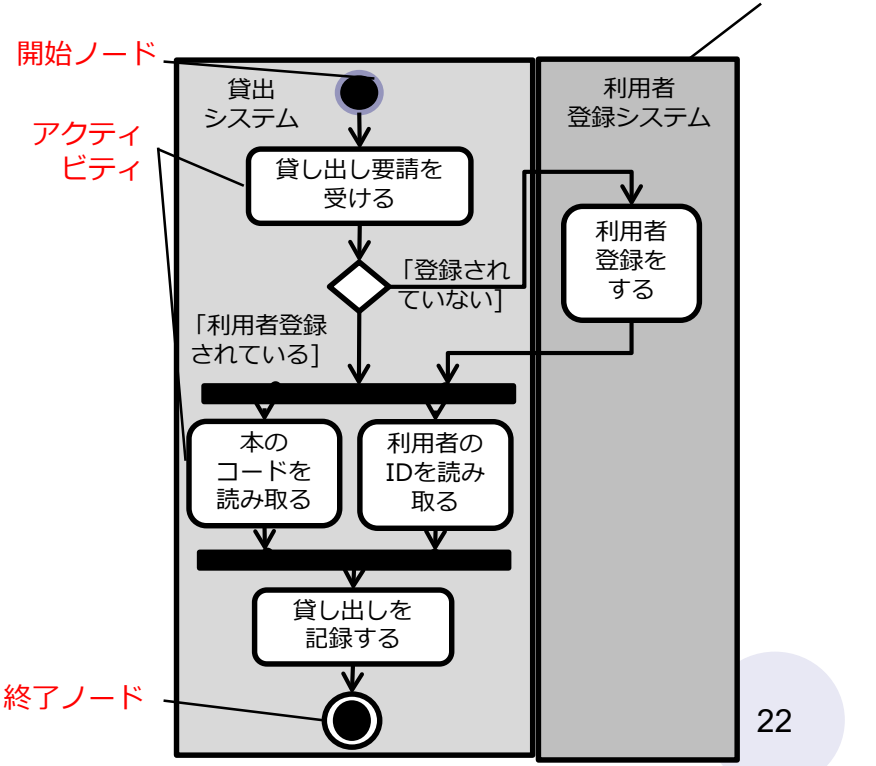
※イメージをつかんでもらうための簡単な図です
※児玉公信著「UMLモデリングの本質」の図を参考に改変

内部的観点からの動的モデル

- 性質／用途
  - 情報システムが「内部的」にどう動くかを表す
- UML図：シーケンス図
  - アクターやオブジェクト間の「メッセージ」のやりとりを表す
- UML図：アクティビティ図
  - 業務の「ワークフロー」などの外部的観点から用いることもある．



シーケンス図の例



アクティビティ図の例

オブジェクト指向開発プロセス

- 分析-設計-実装（Waterfall 型）

オブジェクト指向モデリング	オブジェクト指向プログラミング

分析フェイズ

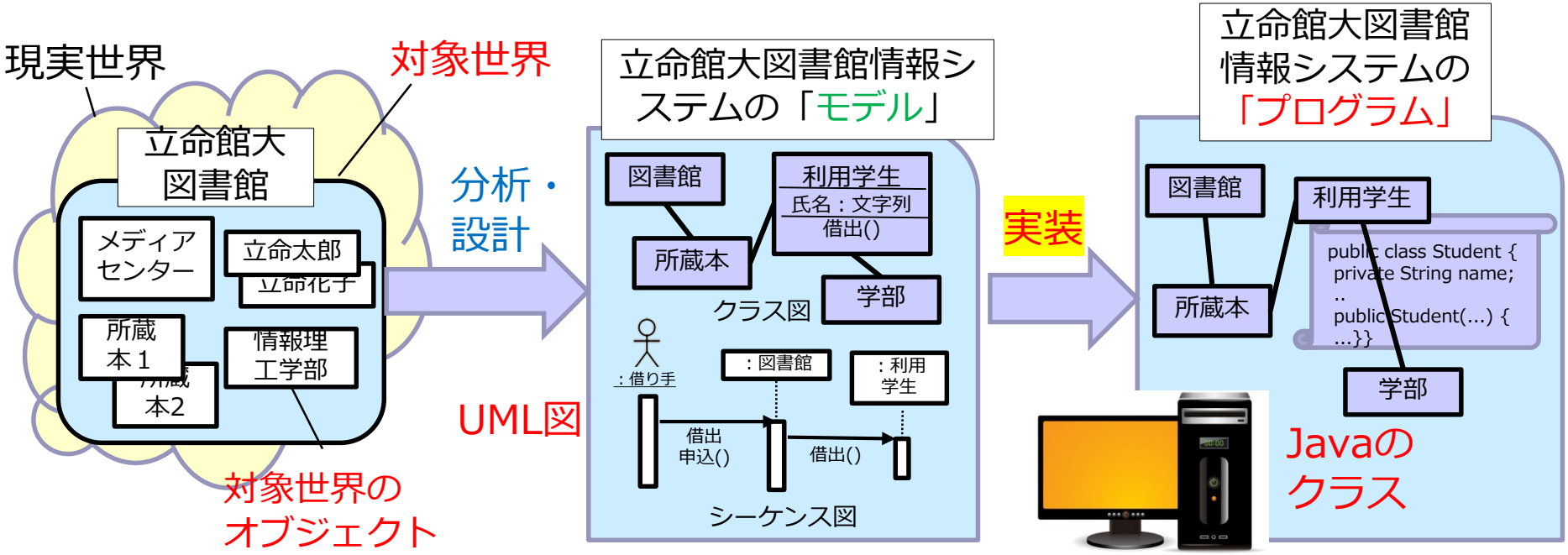
対象世界にどのようなオブジェクトの種類（＝クラス）が存在し，どのような処理が必要かを概念的に分析する．（要求分析を含む）．

設計フェイズ

ソフトウェア的観点から，クラスの属性やクラス間の関連を定義し，処理の流れを明確に記述する．

実装フェイズ

オブジェクト指向型プログラミング言語のクラスを定義する．属性を宣言し，処理をプログラムする．



モデル(緑字)とプログラム(赤字)の関係

- モデルは要求仕様と設計図を表す.
- 要求仕様をみたすプログラムを開発する.
  - ユースケース図が表す機能を実現する.
- モデルに従って，プログラムを実装する.
  - クラス図
    - 属性を表すフィールド変数を宣言する
    - 処理を実現するメソッド（関数）をプログラミングする.
  - シーケンス図
    - インスタンス間で，どのようなメッセージのやりとり（相互作用）＝メソッド呼び出しがあるかを表す.
  - アクティビティ図
    - 処理の時間的／論理的な流れを表す.

参考

OOPでのクラスとインスタンス

- OOPでのプログラムを書く＝クラスを定義する
  - クラスの定義
    - UMLの属性 → フィールド変数の宣言
    - UMLの処理・メッセージ → メソッド（関数）の手続きの記述
- OOPでのプログラムの実行
  - インスタンスの生成
    - クラスの定義（鋳型）に沿って，インスタンスを生成する
  - インスタンスによるデータの保持
    - 各インスタンスは属性（変数）の値を保持する．
  - インスタンス間のメソッド呼び出し
    - インスタンスに「メッセージ」を送ると，メソッド（関数）が実行される．
    - インスタンスの属性（変数）の値が変化する．

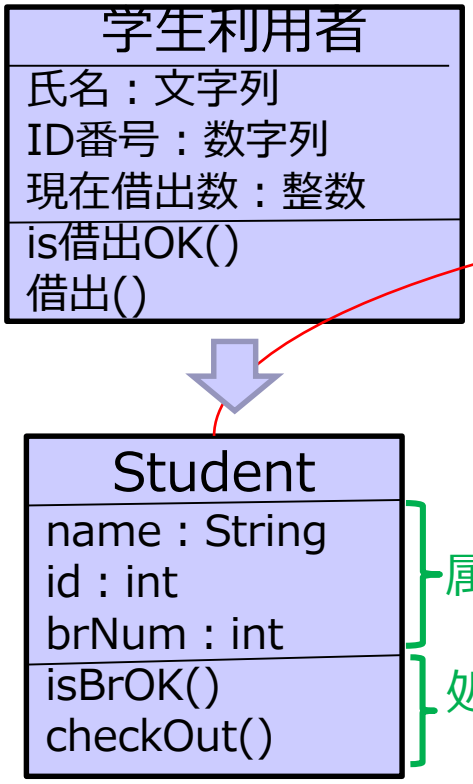
※もちろんまだ意味がよく分からなくてOK

参考

UML図とJavaプログラムの対応：イメージ例

- 図書館の(学生)利用者クラス

UMLのクラス図



Java言語のプログラム

クラスの定義

```
public class Student {
 private String name;
 private int id;
 private int brNum = 0;

 public Student(String name) {
    this.name = name;
    }
 public boolean isBrOK() {
    return (this.brNum <10);
    }
 public void checkOut() {
     this.brNum++;
    }
}
```

フィールドの宣言

(コンストラクタ)

メソッドの定義

※イメージをつかんでもらうための例です.
※もちろんまだ理解できなくていいです
※このままでは動きません

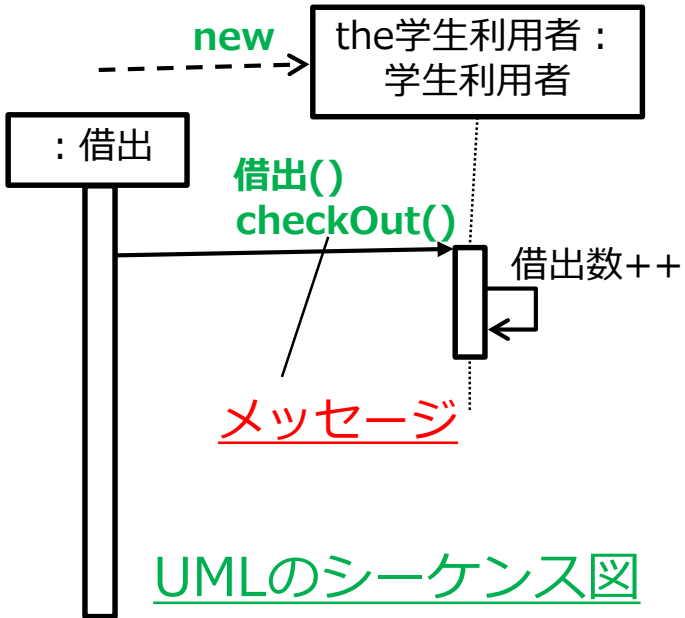
シーケンス図とプログラムの動作イメージ例

- まずインスタンスを生成する
  - new というメッセージ．コンストラクタという特別なメソッドが実行される．
- メッセージ例：「**借出**」メッセージ
  - 「１冊借り出しましたよ」という通知
  - checkout()メソッドが実行される
    - 自分の「現在借出数」(brNum) 属性の値を１つ増やす

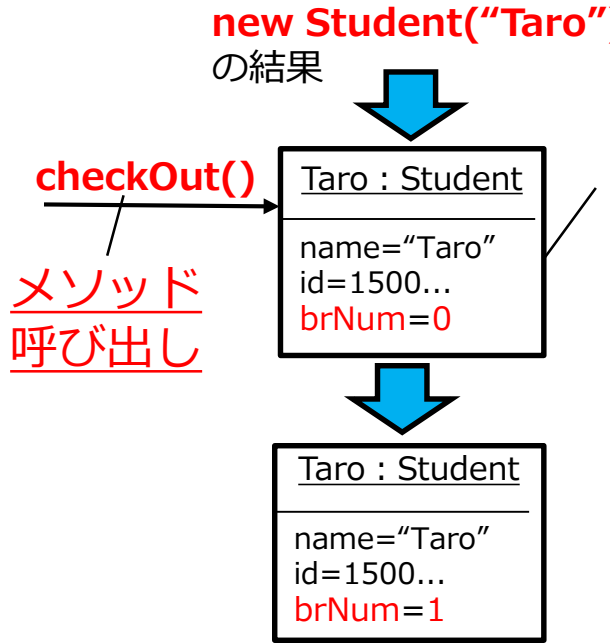
参考

```
public class Student {
 private String name;
 private int id = 0;
 private int brNum = 0;

 public Student(String name) {
    this.name = name;
    }
 public boolean isBrOK() {
    return (this.brNum <10);
    }
 public void checkOut() {
     this.brNum++;
    }
}
```



UMLのシーケンス図



※イメージをつかむための例です．まだ理解できなくていいです．

参考

プログラミングにおけるモデルの役割

- 全般的に
  - 設計図のようにモデルに基づいてプログラムを書く
  - 明確化された概念や設計に基づける
- 種類別の役割
  - 静的モデル（構造図）
    - クラスに対応させる.
    - データと手続き（処理）を一体的にプログラミングする
    - クラスの関係が明確になる
  - 外部的観点からの動的モデル
    - そのように振る舞うようにプログラミングする
    - 要求仕様と果たすべき機能が明確になる
  - 内部的観点からの動的モデル
    - 従来のフローチャートと同じように，明確化された流れに沿ってプログラミングできる

参考

別の解決策:異なる開発プロセス

- ⇔ ウォーターフォール型プロセス
- スパイラル型開発プロセス
  - ①計画＆分析，②詳細設計，③実装＆テスト，④評価のサイクルを，複数回繰り返しながら，開発する.
  - １サイクルごとに「プロトタイプ（試作品）」を作成し，ユーザに使ったもらう.
- 反復型開発プロセス
  - インクリメンタル：基礎となる機能モジュールから順番に，設計・実装・テストを行う.
  - イテレーティブ：計画＆設計を最初に行ったあと，スパイラル型のようにシステム全体を繰り返し開発する.
    - 例：アジャイルプロセス
- cf.「ソフトウェア工学」講義

まとめ

- (0) オブジェクト指向とは
  - クラスとインスタンス（次回で詳しく）
- (1) ソフトウェア開発のフェイズとプロセス
  - ウオーターフォール型開発の困難さ
  - 原因：対象世界と情報システムの「理解」が人によって異なるため，関係者間の意思疎通が難しい
  - 解決法：モデルの記述と利用
- (2) モデルとは？
  - モデルとは：対象世界の本質を表す簡単化した記述
  - オブジェクト指向モデルとは：オブジェクトを中心に対象世界を捉えたモデル
  - モデルの種類
    - 観点の違い：静的と動的，外部的と内部的
- (3) プログラミングにおける役割
  - 明示化された仕様／設計図を与える