

# 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)

講義資料#6

2023/5/8～15

來村 徳信

# モデルの分類の観点

## ○ 「時間的」観点からの分類：

これまでの  
講義



### (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

前回から  
の講義



### (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。


# 本資料のテーマと流れ

- 外部的観点からの動的モデル

- 外部的観点：システムを外部から捉える
- 動的：時間を明示的に扱う

- ユースケース図


- 機能的要求, 使用シナリオ



## 内部的観点からの動的モデル

- アクティビティ図

- 内部的観点から「処理（オブジェクトの動作）の流れ」を表す
- 外部的観点からの「ワークフロー」の表現にも使われる



## シーケンス図

- アクターやオブジェクト間の「対話」を表す
- メッセージパッシング

# 本資料のテーマと流れ

- 外部的観点からの動的モデル

- 外部的観点：システムを外部から捉える
- 動的：時間を明示的に扱う

- ユースケース図

- 機能的要求, 使用シナリオ

## 内部的観点からの動的モデル

- アクティビティ図

- 内部的観点から「処理（オブジェクトの動作）の流れ」を表す
- 外部的観点からの「ワークフロー」の表現にも使われる

## シーケンス図

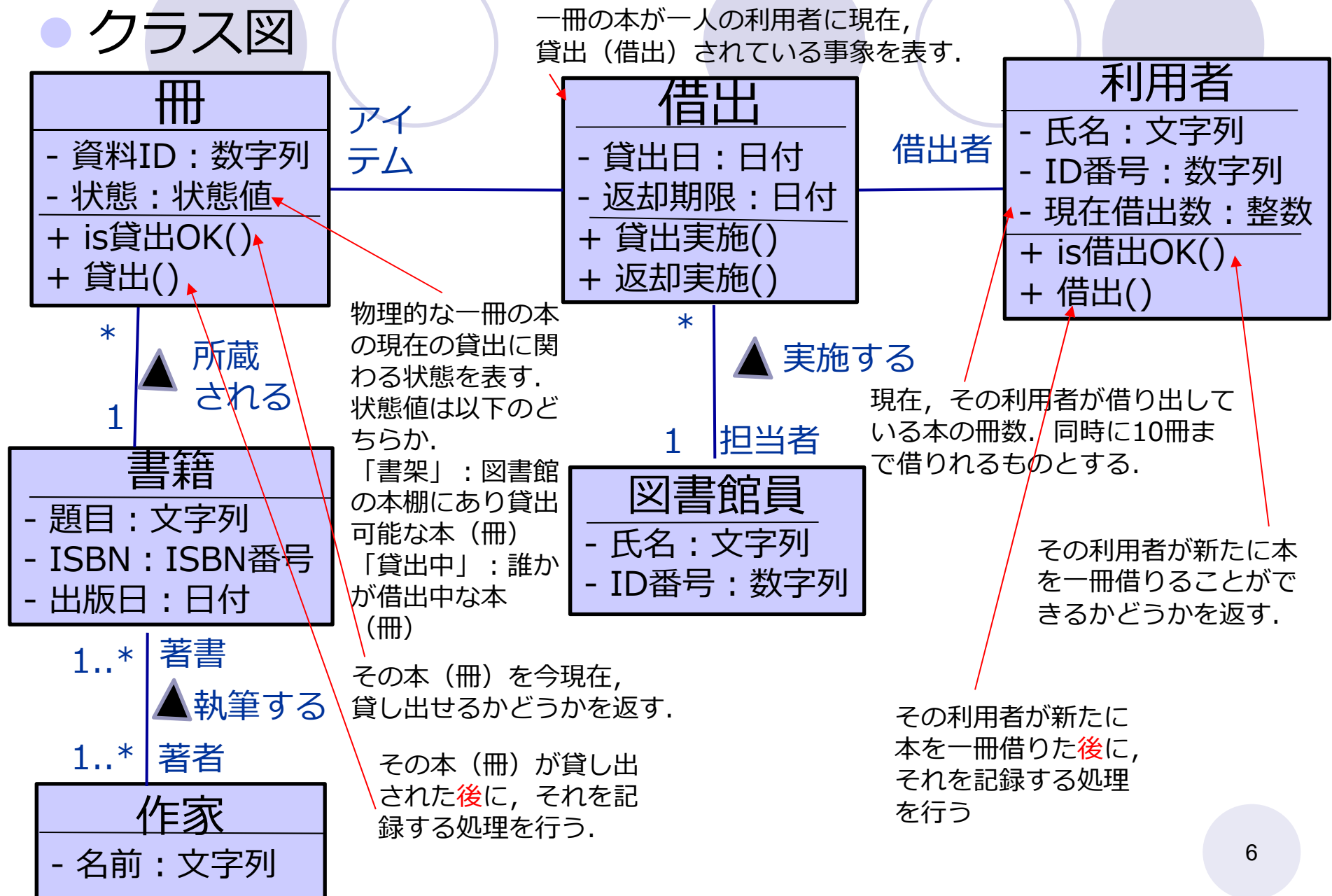
- アクターやオブジェクト間の「対話」を表す
- メッセージパッシング

- オブジェクト指向の原則と利点

- 原則：情報と処理の一体的分散化による局所化
- 利点：大規模システムが開発／変更しやすい

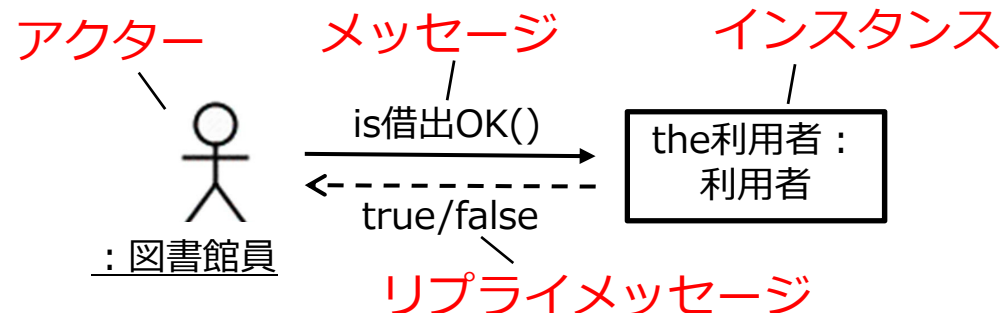
# 例題：図書館情報システム

## クラス図



# 相互作用図

- アクターやオブジェクト間の「対話」を表す
  - 「メッセージ」：
    - 「問い合わせ」 / 「更新」 / 「処理の依頼」などを表す
    - (通常) インスタンスの間で送られる
      - 特定のインスタンスではなく、仮のインスタンスを考える場合が多い.
      - 本講義での例: 「theクラス名」という名前のインスタンス
    - メッセージを受け取ったインスタンスは、それに対応する自分の「操作」(メソッド)を起動して、処理を行う. 返事(戻り値 / リプライメッセージ)を返す.
  - オブジェクト間の「相互作用」を表す
    - システムのすこし内部を表す
    - システム内部のオブジェクト間の相互作用



# シーケンス図

- 相互作用図の1つ

- 1つのクラス/アクターのインスタンスを「**縦線**」で表す

- 「**ライフライン**」と呼ばれる

- 上部：「ライフライン名」（インスタンス名 **：** クラス名） 下線は不要.

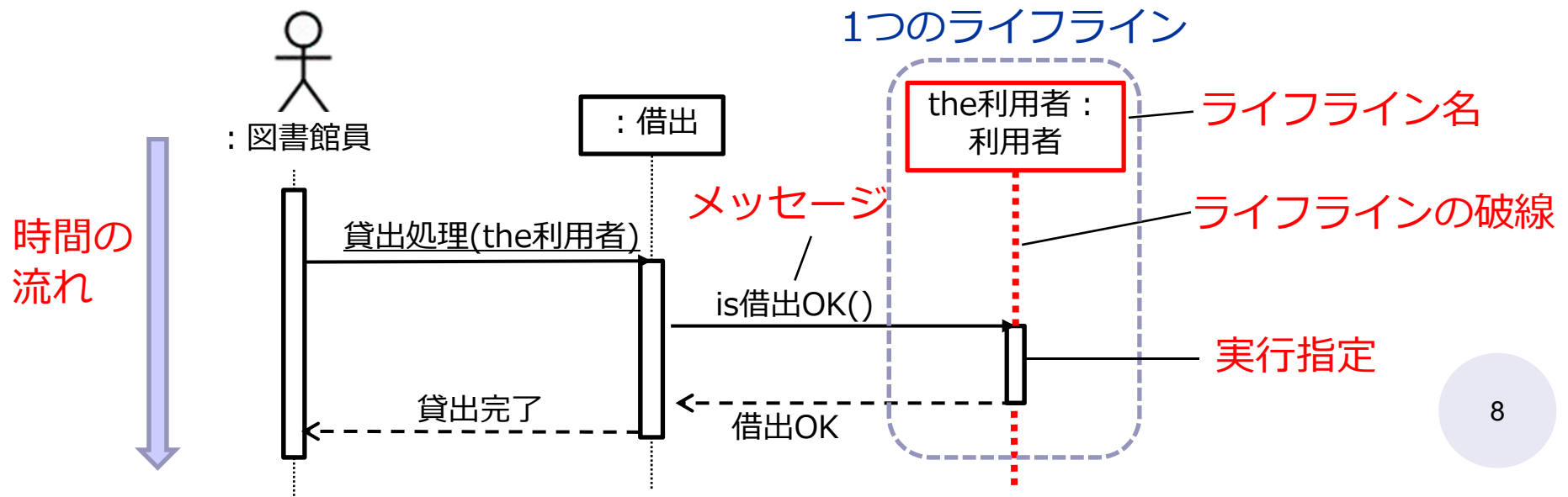
- **破線**：そのインスタンスが存在している期間を表す.

- 下の例：the利用者は「利用者登録」後、ずっと存在している.

- **白い長方形（実行指定）**：メッセージに反応している期間を表す.

- インスタンスの間のメッセージのやりとりを時間順に表現

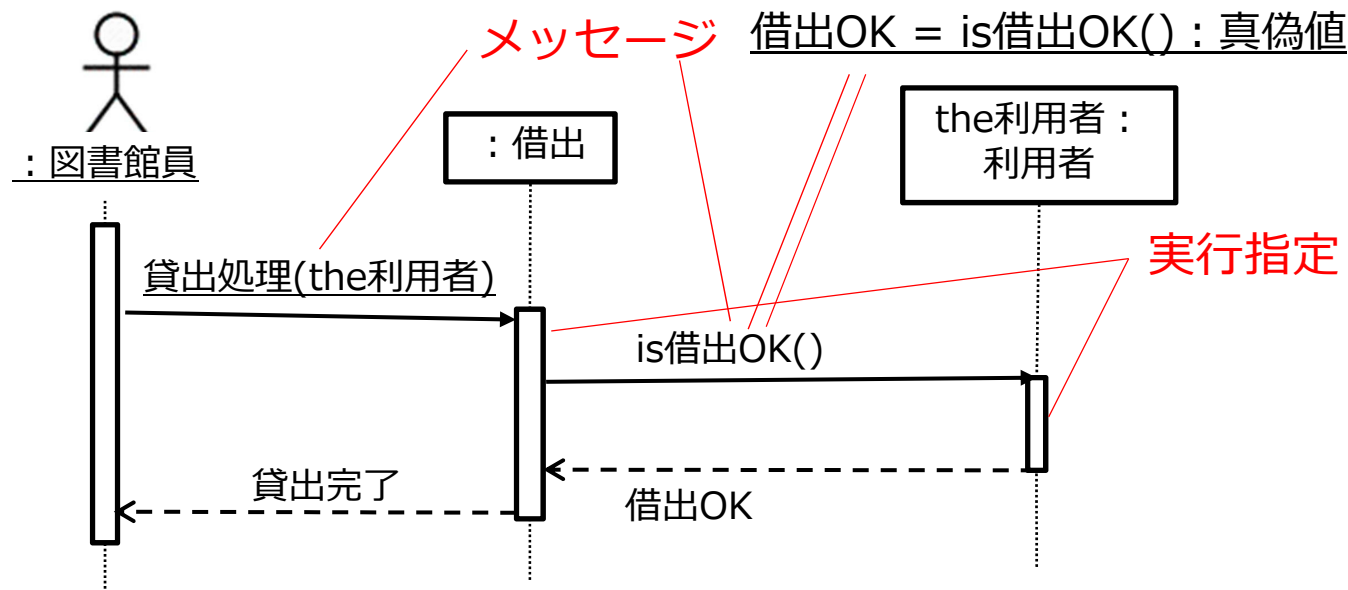
- 上から下に時間が流れる





# 「メッセージパッシング」

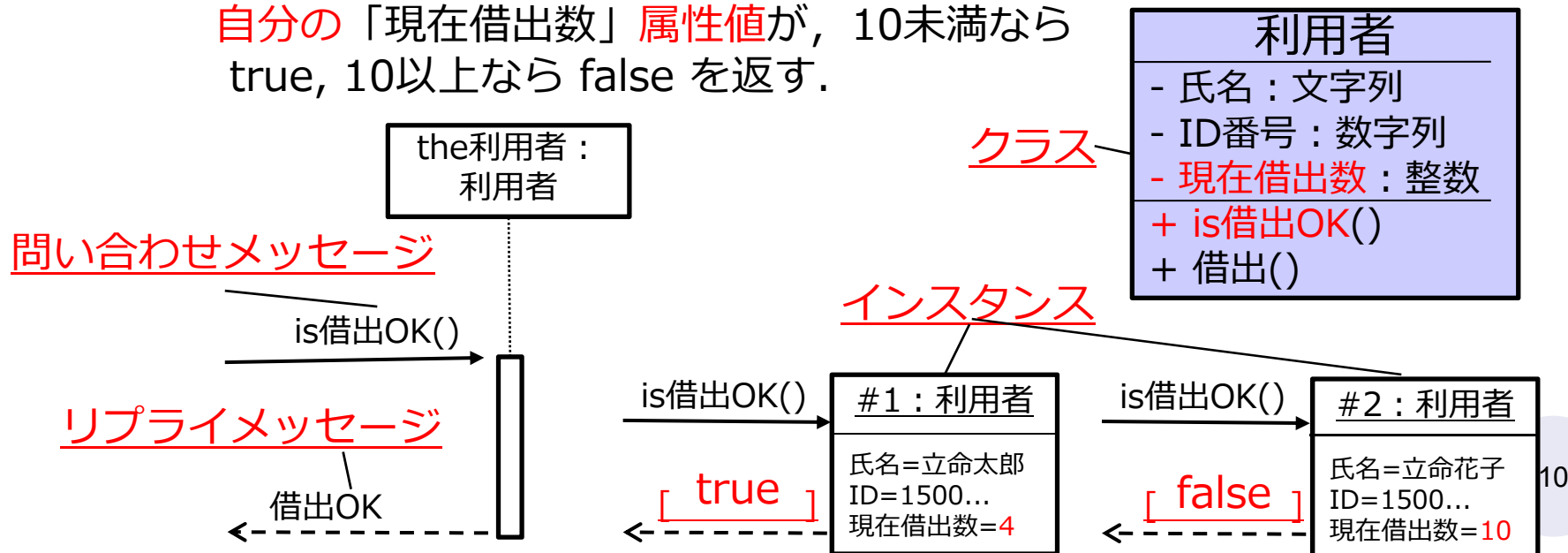
- オブジェクトにメッセージを送ること
  - オブジェクト = 「**インスタンス**」にメッセージを送ることが多い
  - 受け取ったオブジェクトでは、対応した**操作**（メソッド）が「**実行**」される（**実行指定**の白長方形が表す期間）
    - プログラムにおける**メソッド呼び出し**と**戻り値**に対応する。
  - メソッドはオブジェクトが属するクラスで定義されている必要がある。
- メッセージの表現
  - ライフラインの間に矢印を引き、メッセージ名（と引数）を書く。
    - 戻り値格納変数 = メッセージ名（引数名） : 戻り値



# メッセージの種類(1)

## ○「問い合わせ」(query)

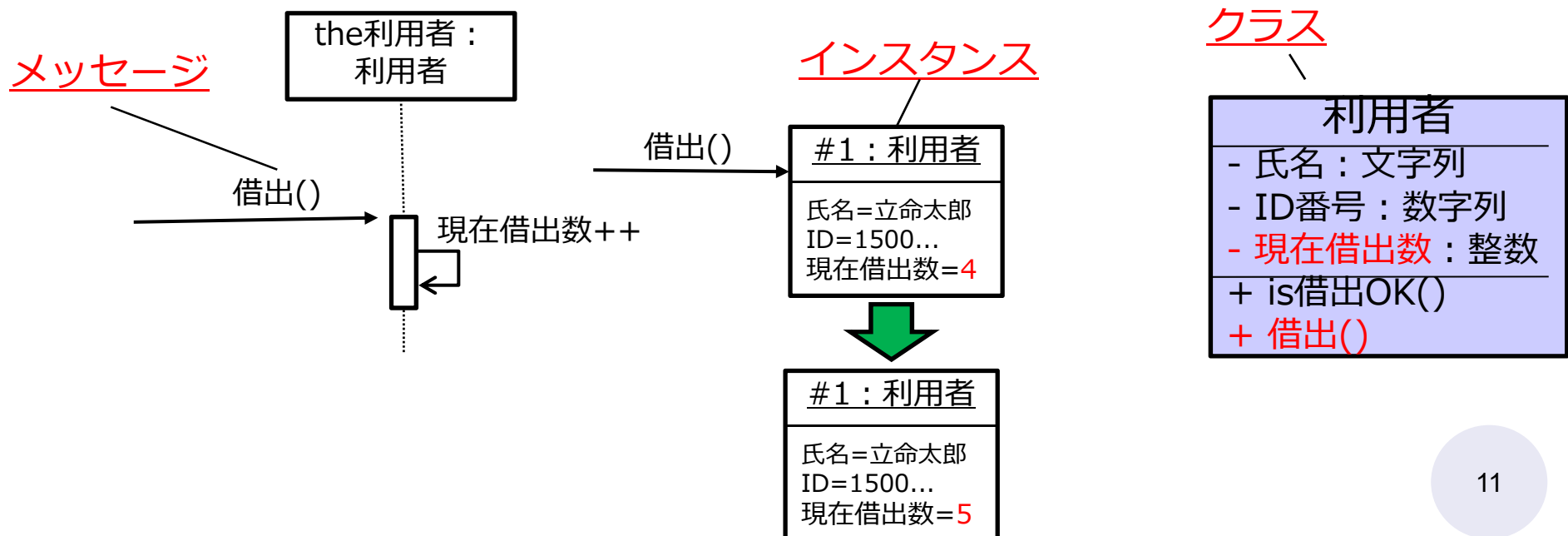
- オブジェクト (=インスタンス) の「属性値」を取り出す.
- オブジェクトに関する「判断結果」を得る.
  - オブジェクトは自分自身の属性値を参照して, 判断する
- オブジェクトの状態は変化しない
- 例: 図書館利用者オブジェクトへの「借り出しok?」メッセージ
  - その利用者が「本を新たに借り出す資格があるか」どうかの問い合わせ.
  - is借出OK() メソッドの処理の例:  
自分の「現在借出数」属性値が, 10未満なら true, 10以上なら false を返す.



# メッセージの種類( 2 )

## ○ 「更新」(modifier)

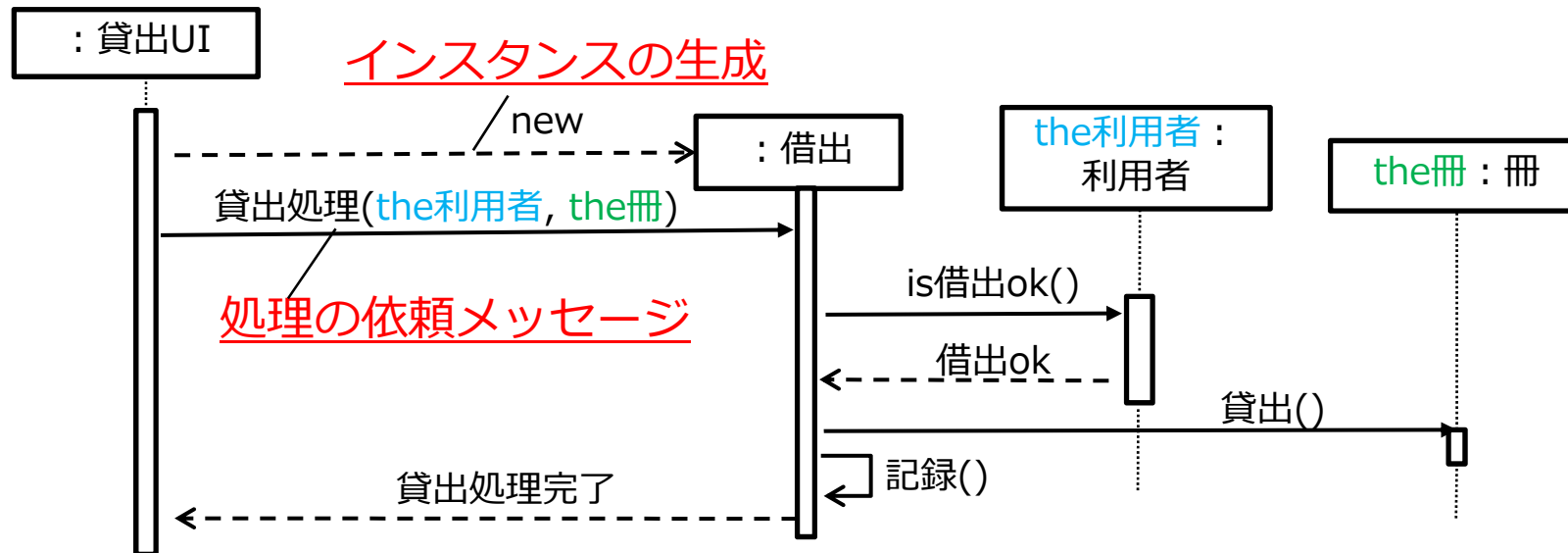
- 通知. 状態の更新の依頼.
- オブジェクトの状態が変化する. 属性値のセット/変更.
- 例: 利用者オブジェクトへの「借出」メッセージ
  - その利用者が1冊借り出しましたよ, という通知
  - 借出メソッド:
    - 「その利用者が新たに本を一冊借りた後に, それを記録する処理」
    - 処理例: 自分の「現在借出数」属性の値を1つ増やす



# メッセージの種類( 3 )

## ○「処理の依頼」(command)

- 他のオブジェクトに処理を行わせて、全体の制御を行う。
- 例：借出処理：「借出」インスタンス
  - まず「借出」クラスの新しいインスタンスが生成(new)される。
    - 点線の矢印で表現する。ライフラインが開始される。
  - その借出オブジェクトに「貸出処理」が依頼される。
    - 特定のインスタンス (the利用者, the冊) が引数として渡される。
  - 借出オブジェクトは、the利用者オブジェクトやthe冊オブジェクトとメッセージを交換しながら、処理を進める。最後に、自分に貸出日付などをセット(記録)して、処理が完了。

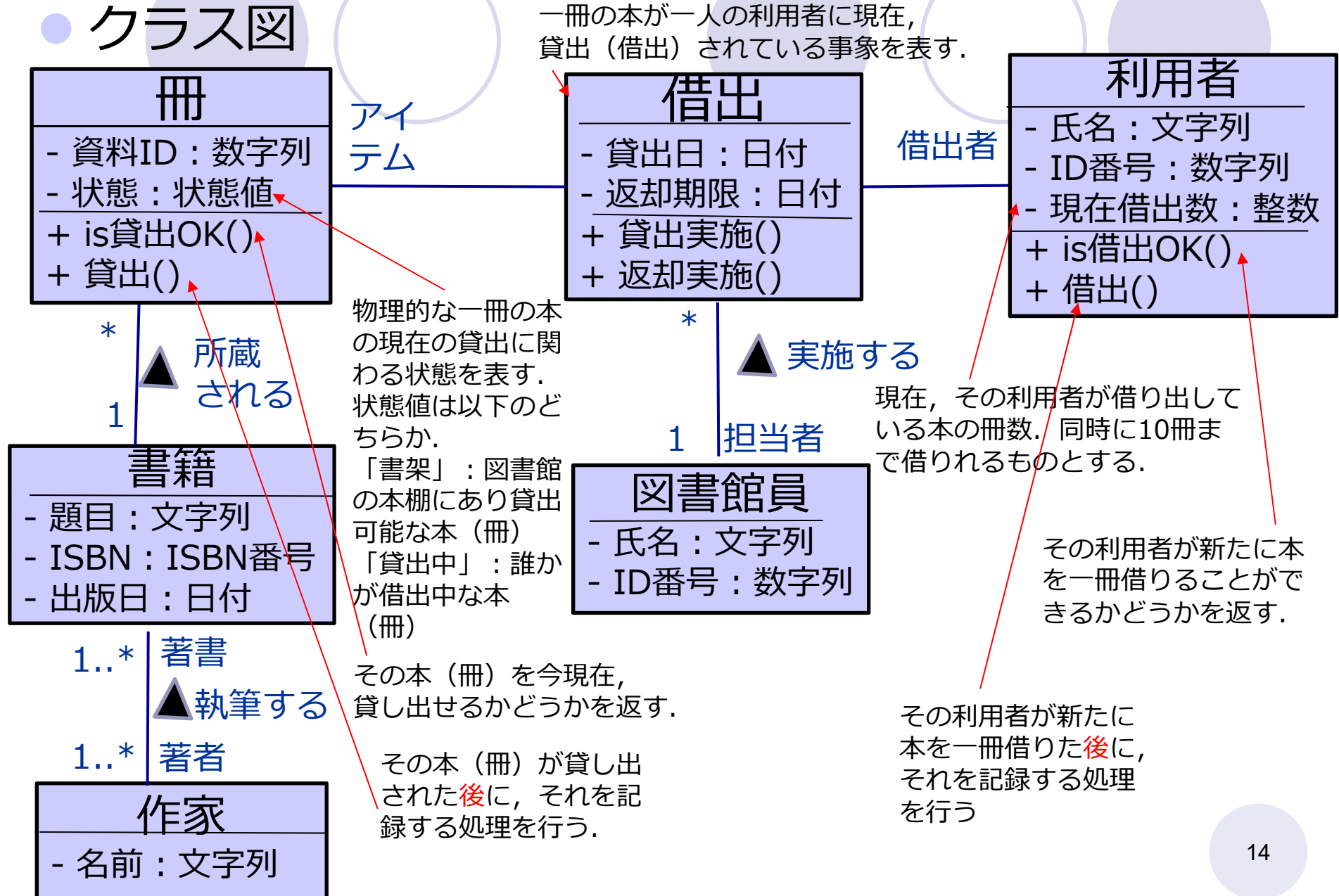


# 操作／メソッドの分類

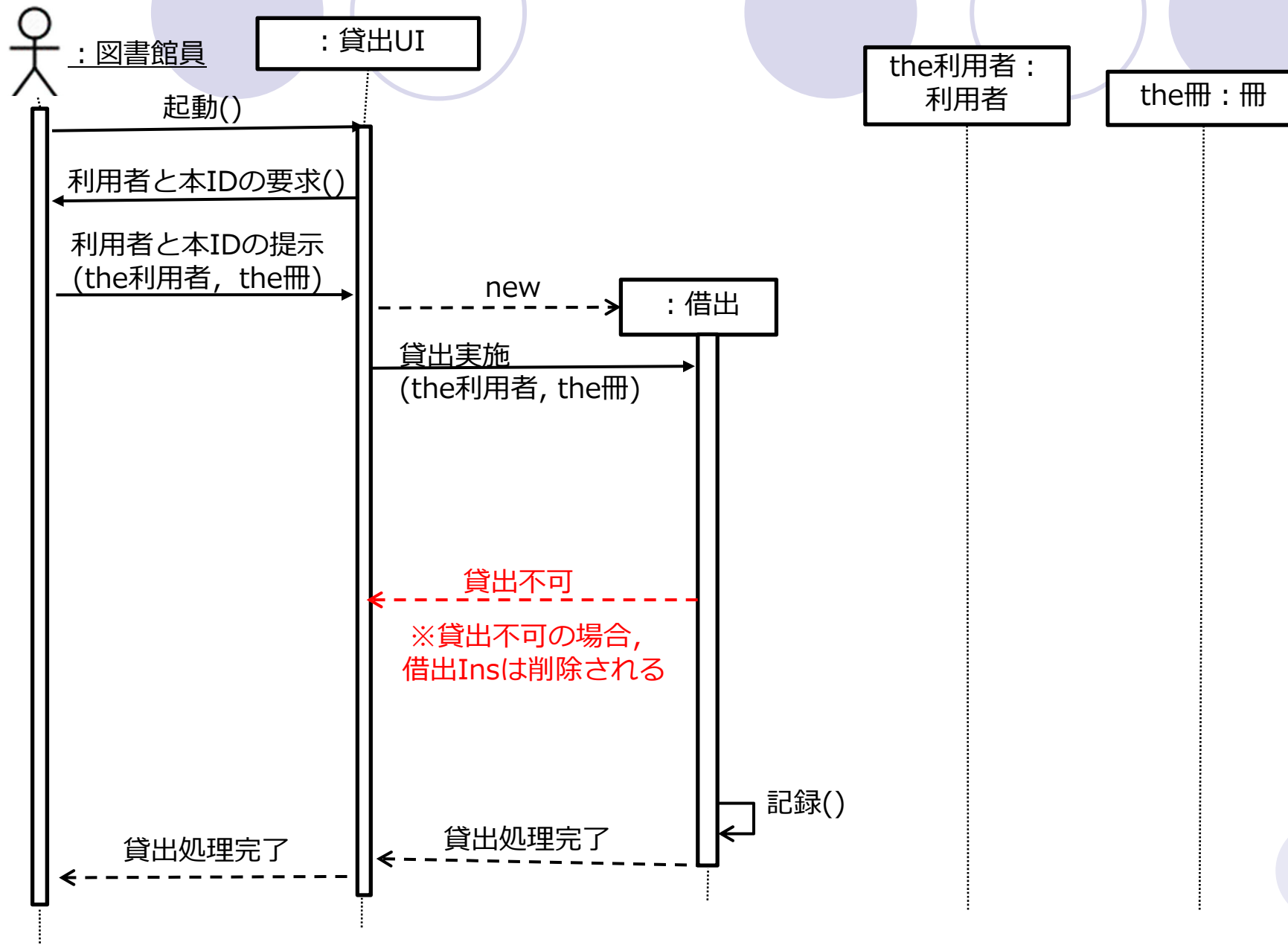
- 「副作用」の有無
  - 副作用 = オブジェクトの状態の変化
  - 副作用を伴わない操作
    - オブジェクト（主にインスタンス）の属性値が変化しない.
    - 「問い合わせ」メッセージの場合
  - 副作用を伴う操作
    - オブジェクト（主にインスタンス）の属性値が変化する.
    - 「更新」メッセージや「処理の依頼」メッセージの場合
- 操作とメソッド（厳密には違う）
  - 操作 = 操作名
    - オブジェクトに送られるメッセージで起動される処理
    - 操作の仕様（機能, 引数, 戻り値）の宣言
  - メソッド = 操作の実装
    - 処理のひとつの実装された本体（通常, クラスで定義される）
    - 1つの操作は複数のメソッドを持つことがある（オーバーライドなど）

# 例題：図書館情報システム

## クラス図



# 図書館：借出のシーケンス図の一部



# 講義のテーマと流れ

- 外部的観点からの動的モデル

- 外部的観点：システムを外部から捉える
- 動的：時間を明示的に扱う

- ユースケース図

- 機能的要求, 使用シナリオ

- 内部的観点からの動的モデル

- アクティビティ図

- 内部的観点から「処理（オブジェクトの動作）の流れ」を表す
- 外部的観点からの「ワークフロー」の表現にも使われる

- シーケンス図

- アクターやオブジェクト間の「対話」を表す
- メッセージパッシング



## オブジェクト指向の原則と利点

- 原則：情報と処理の一体的分散化による局所化
- 利点：大規模システムが開発／変更しやすい

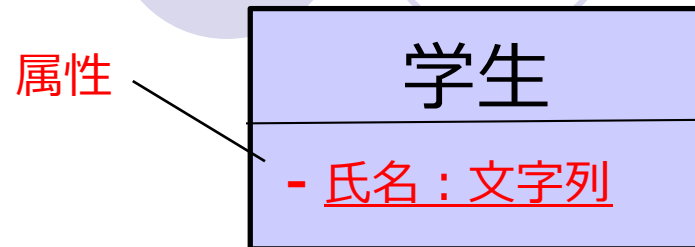


# オブジェクト指向の原則：分散と局所化

- 処理（操作）をオブジェクトに「分散」させる
  - 「責務(responsibility)の分担」という
  - 一連の処理はメッセージパッシングによって、複数のオブジェクトのメソッドを起動することで進んでいく
- 情報をオブジェクトに分散させる
  - 情報をオブジェクトの属性として「局所化」する。
  - 他のオブジェクトからは見えなくする。「情報の隠蔽化」
    - 属性の可視性：「-」プライベート (↔ 「+」パブリック)
- 処理と情報をオブジェクトが一体的に持つ
  - 情報をもつインスタンスオブジェクトにメッセージを送って、処理を実行してもらう
  - プログラム間でデータを流すのではない。
  - 情報の表し方や処理の詳細を依頼側は知らない／知らせない（「カプセル化」と呼ぶ）

# UML：クラスの属性の表現

- 例：学生クラスの「氏名」属性



- 属性の記述形式（の一部）
  - 「可視性」「名前」：「タイプ」



## 可視性

- + : パブリック, - : プライベート
- 属性がクラスの外側からみえるかどうか
- 詳しくはプログラミング回で扱う。モデリング回では省略してもよい

## 名前

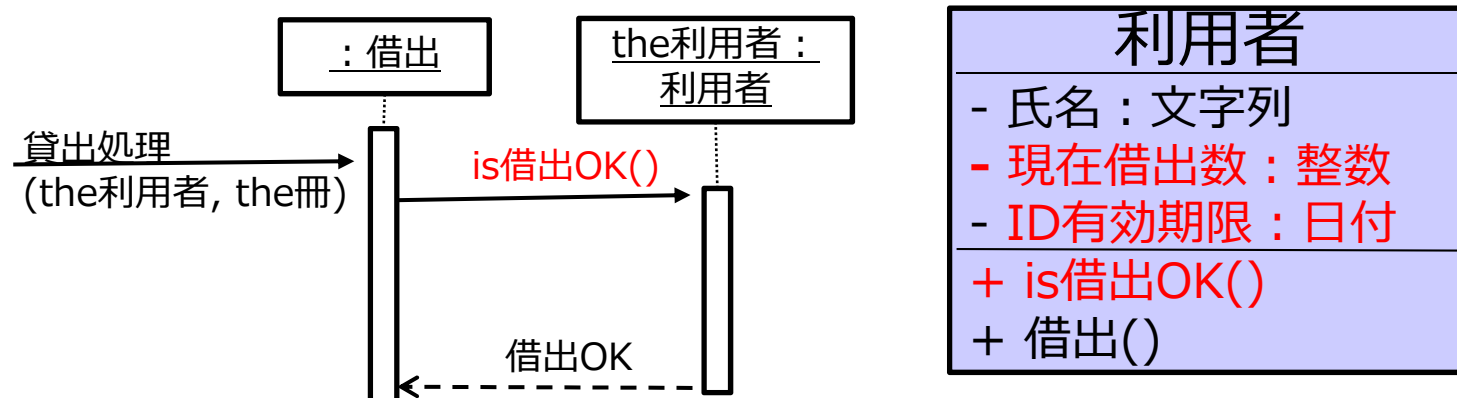
- 属性を表現する文字列。変数名に対応。（例：氏名）

## タイプ

- 属性の値を表すオブジェクトの種類やデータの型。
  - 例：文字列(String), 数値(Integer), 日付(Date), クラス名
- 変数の「型」に対応。

# 分散と局所化のメリット

- 他のオブジェクトは情報や処理の詳細に依存しない
  - 情報の持ち方や処理のやり方（ロジック）を変えても、他のオブジェクトに影響しない。  
→ 変更の範囲を「局所化」できる
  - 例：利用者が本を新たに借りてもよいかの判断
    - 利用者クラスで is借出OK() というメソッドを定義
    - 現在：利用者インスタンスが自分の属性である「現在借出数」の値を参照して、判断する。
      - 現在借出数の可視性を「-」プライベートに指定することで、他クラスから直接に参照・変更されないことが保証される。
    - 将来：他の属性(例：IDの有効期限)を参照するようになってても、利用者クラスの変更だけで済む。借出クラスは変更しなくてよい。



# 非オブジェクト指向との比較

非オブジェクト指向のプログラムは：

- 処理のプログラム単位が大きくなりがち
  - 例：貸出処理()メソッドに全てを書きがち
  - ↔ オブジェクト単位で小さい処理メソッドを書く
    - メインスタンス同士がメッセージをやりとりして処理が進む
- 情報と処理の対応関係が不明確になりがち
  - 例：利用者DBを参照している部分が、あちこちにある
  - ↔ 利用者に関する情報は利用者インスタンスが持つ.  
利用者に関する処理は利用者クラスのメソッドのみ.

# 非オブジェクト指向との比較(2)

非オブジェクト指向のプログラムは：

- → 分かりにくい
  - 大規模になるとどこになにが書いてあるか分かりにくい
  - ↔ 情報と処理が一体化しているため分かりやすい
  - ↔ 現実世界のモノ／コトと対応しているため分かりやすい
    - オブジェクトという単位で局所化する
- → 変更に伴う影響範囲が大規模になりやすい
  - 例：利用者の借出基準が変更されたとき，どこを直せばよいのかわかりにくく，箇所が多くなりがち。
  - ↔ 利用者クラスの属性とメソッドのみを修正すればよい。
- オブジェクト指向は解決方法の一つ

# まとめ

- 外部的観点からの動的モデル

- 外部的観点：システムを外部から捉える
- 動的：時間を明示的に扱う

- ユースケース図

- 機能的要求, 使用シナリオ

- 内部的観点からの動的モデル

- アクティビティ図

- 内部的観点から「処理（オブジェクトの動作）の流れ」を表す
- 外部的観点からの「ワークフロー」の表現にも使われる



- シーケンス図

- アクターやオブジェクト間の「対話」を表す
- メッセージパッシング



- オブジェクト指向の原則と利点

- 原則：情報と処理の一体的分散化による局所化
- 利点：大規模システムが開発／変更しやすい

# 参考文献

- [1] 児玉公信（著），UMLモデリングの本質，日経BP社，2004
- [2] マーティン・ファウラー（著），羽生田 栄一（翻訳），UMLモデリングのエッセンス（第3版），翔泳社，2005
- [3] かんたんUML入門 改訂2版，技術評論社，2017
- ※特に図書館情報システムのUML図は [1] を参考にして，改変した．