



# オブジェクト指向の考え方

オブジェクト指向プログラミング特論

只木進一:総合情報基盤センター

# PROGRAMMING FRAMEWORK: PROCEDURE ORIENTED

- データ処理手順を中心に考える
  - データ処理のためのサブプログラムや関数を組み合わせる
- 流れ図による整理
- 手続き型プログラミング言語
  - C、Fortran
- 対象の操作・動作を手続きに翻訳しなければならない。
- アルゴリズム最適化などで高速化可能。



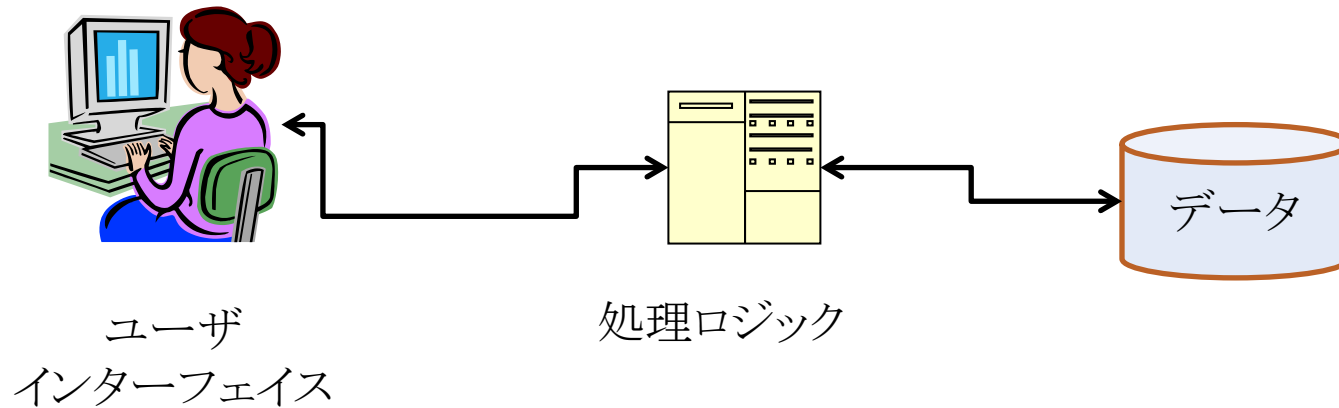
# PROGRAMMING FRAMEWORK: DATA ORIENTED

- データを中心に考える
  - データがどのように変化していくか
- データフローによる整理
- データベース処理
- 対象をデータとして整理しなければならない。
- 巨大なデータを扱える。

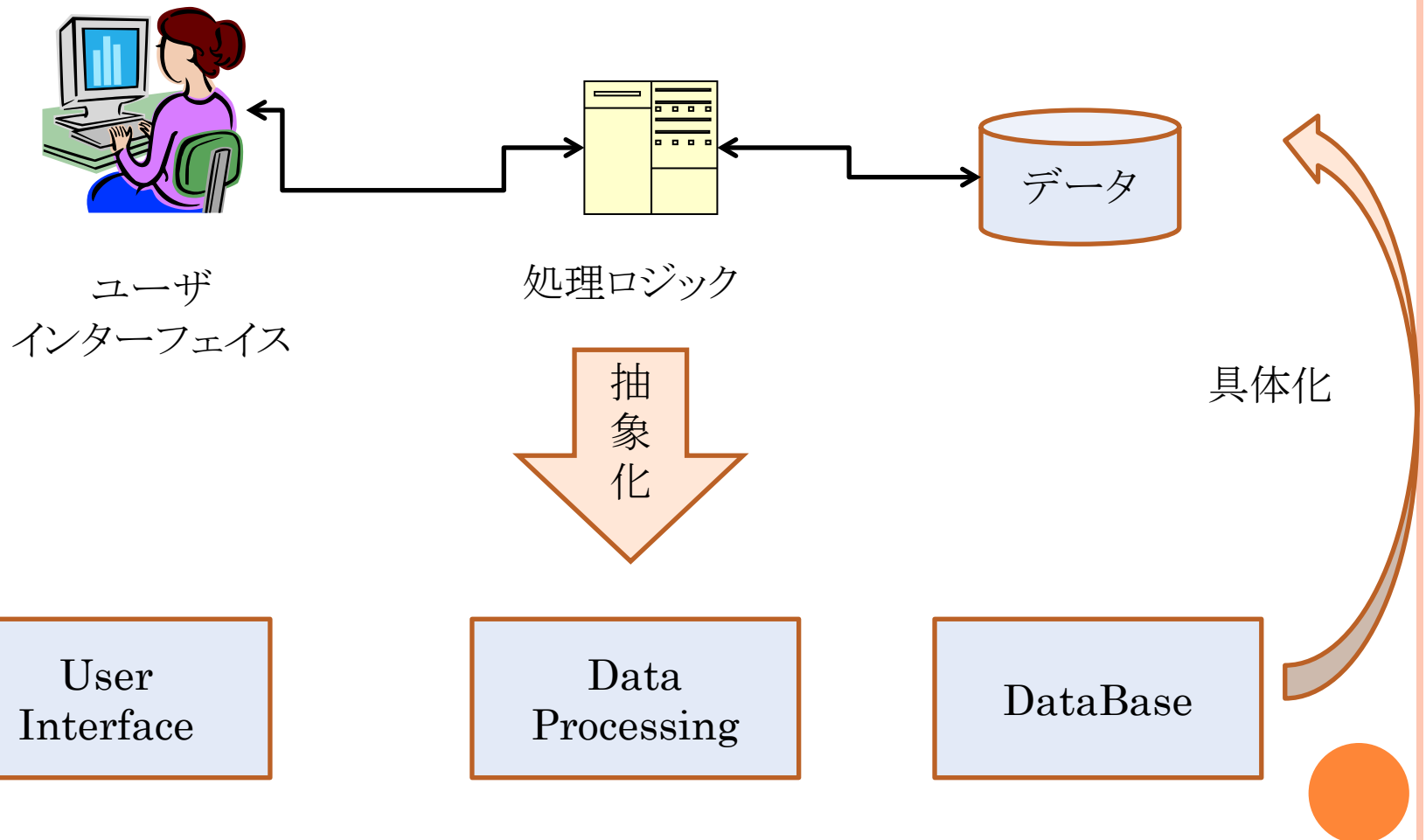


# PROGRAMMING FRAMEWORK: OBJECT ORIENTED

- オブジェクトとして対象を捉える
  - データとその動作・処理の組
- オブジェクトの相互作用としてシステムの動作をとらえる。
  - 日常の考えに近い



# OBJECTS, CLASSES AND INSTANCES

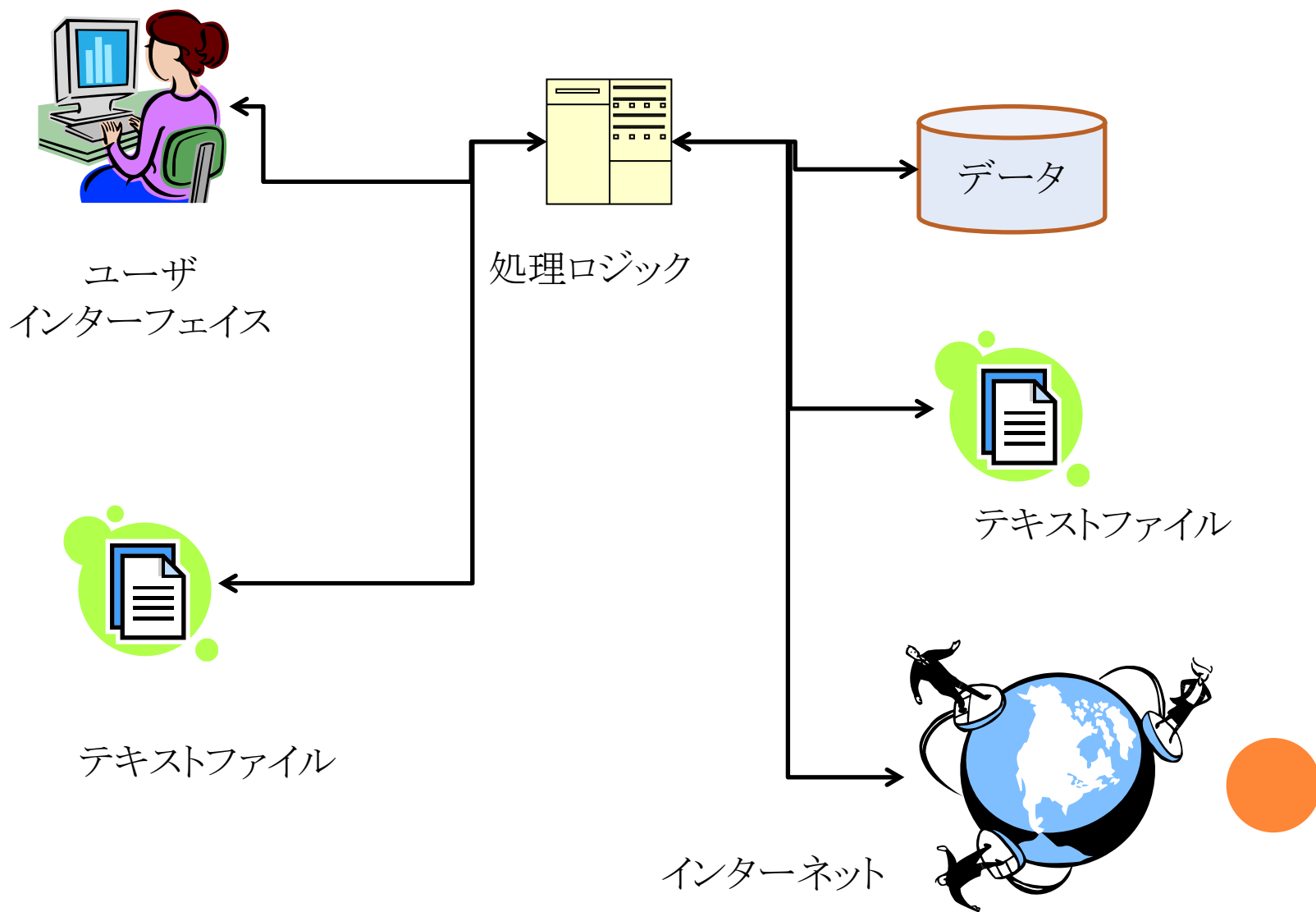


# OBJECTS, CLASSES AND INSTANCES

- オブジェクト
  - 取り扱い対象
- クラス
  - オブジェクトを抽象化したもの
  - データの型に相当
- インスタンス
  - クラスを実体化したもの
  - 具体的な値が入っているデータ



# 実装とインターフェースの分離



# オブジェクト指向で得られるもの

- 様々な実装に対応したい
- できるだけプログラムをいじらない
- 他人の書いたプログラムを活用する
- 開発コスト・保守コストの削減





# FIELD

- クラス内のデータ
  - Fieldと呼ぶ
- アクセス制限
  - public:他のクラスから参照
  - private:クラス内限定
  - protected:クラス内と継承クラス限定
  - 何も書かない:同じpackage内



- FieldへのアクセスはMethodを通じて行う
  - setterとgetter
- クラスに属するfieldとインスタンスに属するfield
  - static:クラスに属する
    - インスタンスが無くても存在する



# METHOD

- fieldへアクセスするmethod
  - クラスのデータへアクセス:Getter
  - クラスのデータを変更:Setter
- オブジェクトの操作を行うmethod



# JAVAでは全てがクラスに属する

- 主語を中心に問題を記述する。
  - システムの概要、動作を文で記述する:ユースケース
- どうしても述語になってしまうものもある。
  - 数学関数など汎用的な処理類
  - その述語を管理するべきクラスを考える
  - クラスに属するmethodとして定義



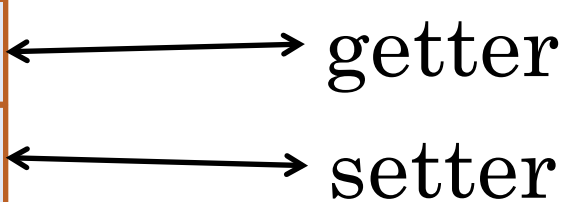
# クラス設計例: 文書管理

- ユーザがシステムへアクセス
  - ユーザがファイルを登録
  - ユーザが登録済ファイルを更新
  - ファイルの情報を登録
  - ユーザがファイルを閲覧
  - システムは、ユーザの権限に応じてファイルを表示
  - ...
- クラスと属性
    - ユーザ
      - 権限
    - 閲覧権限
      - ユーザー一覧
      - 権限一覧
    - ファイル
      - 閲覧権限
      - 登録ユーザ



## クラス宣言とSETTERとGETTER

```
public class Class{  
    private int field=0;  
    public int getField(){return field;}  
    public void setField(int field){  
        this.field=field;  
    }  
}
```



getter

setter

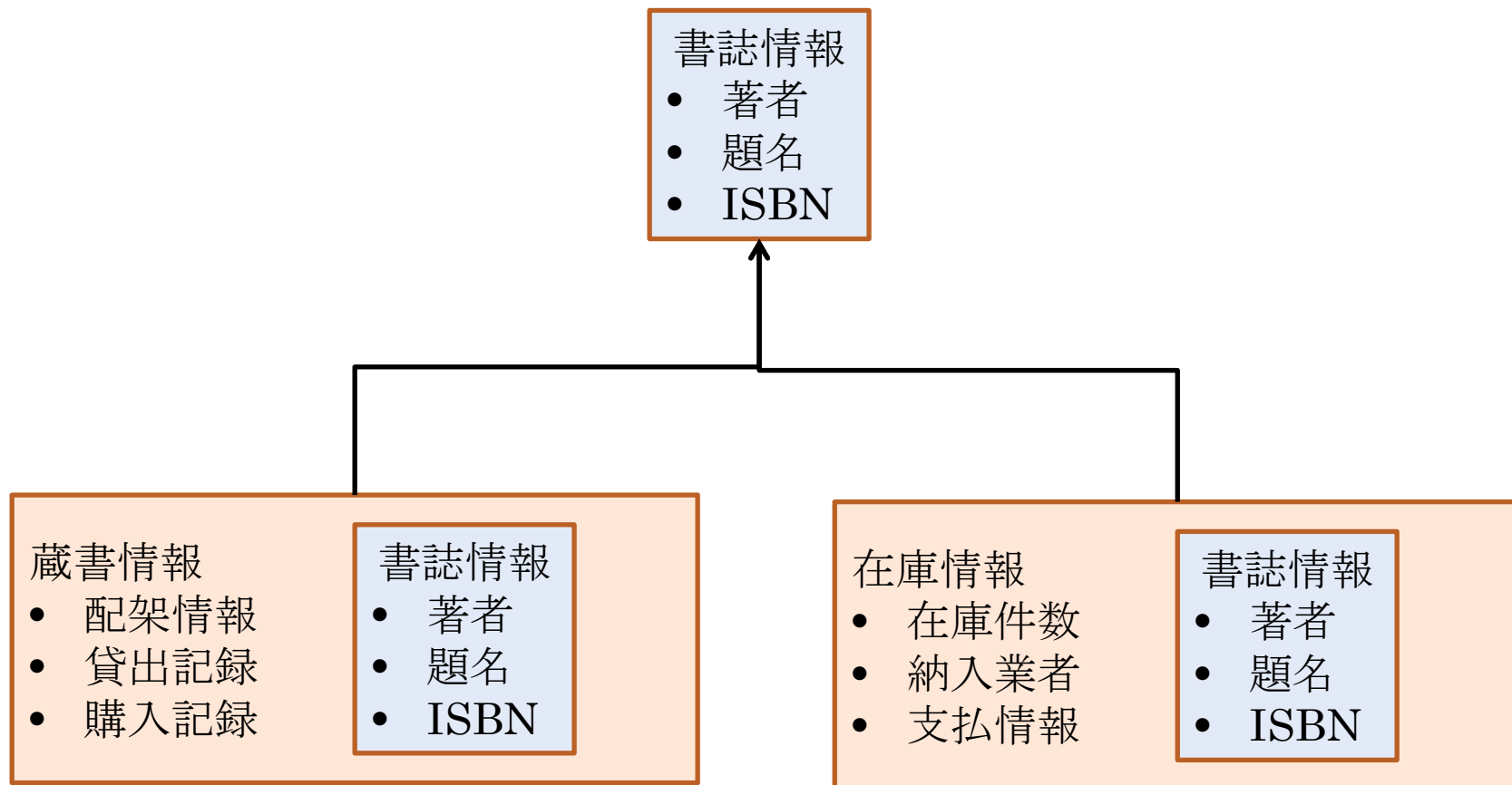


# クラスの継承 (INHERITANCE)

- 既存のクラスを引き継いで新しいクラスを定義
  - より具体化する
  - 対象に近づける
- 元のクラスのFieldとMethodを引き継ぐ
- 新しいFieldやMethodを追加する
- Methodを実装する: 抽象クラスの継承

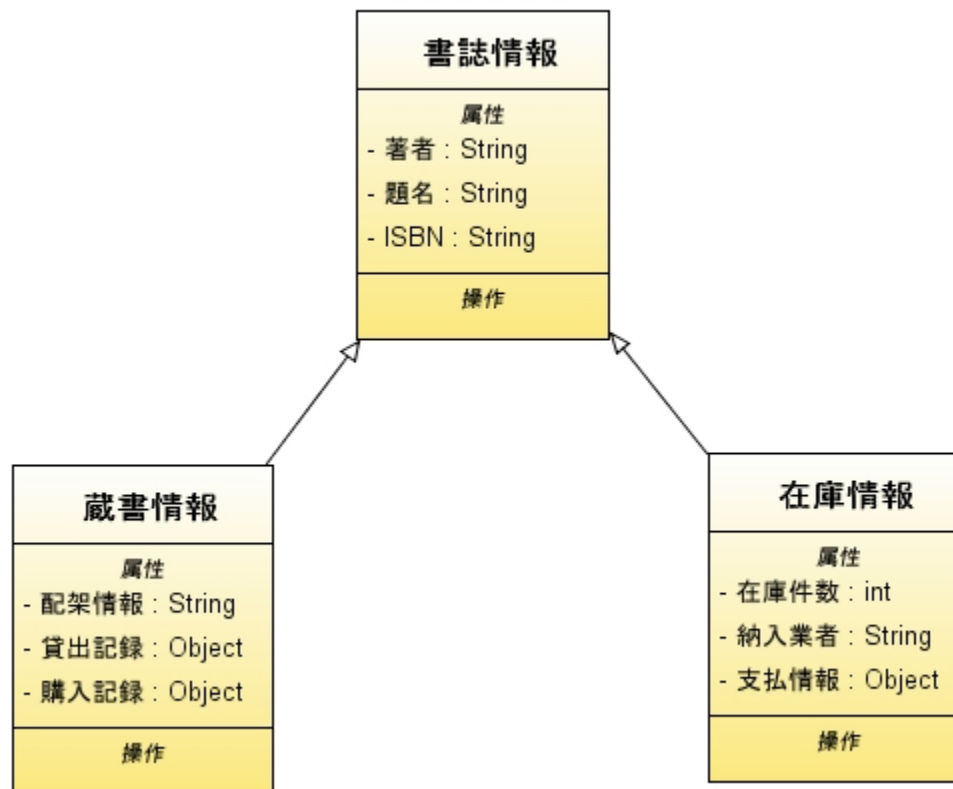


## クラスの継承例:FIELDの追加



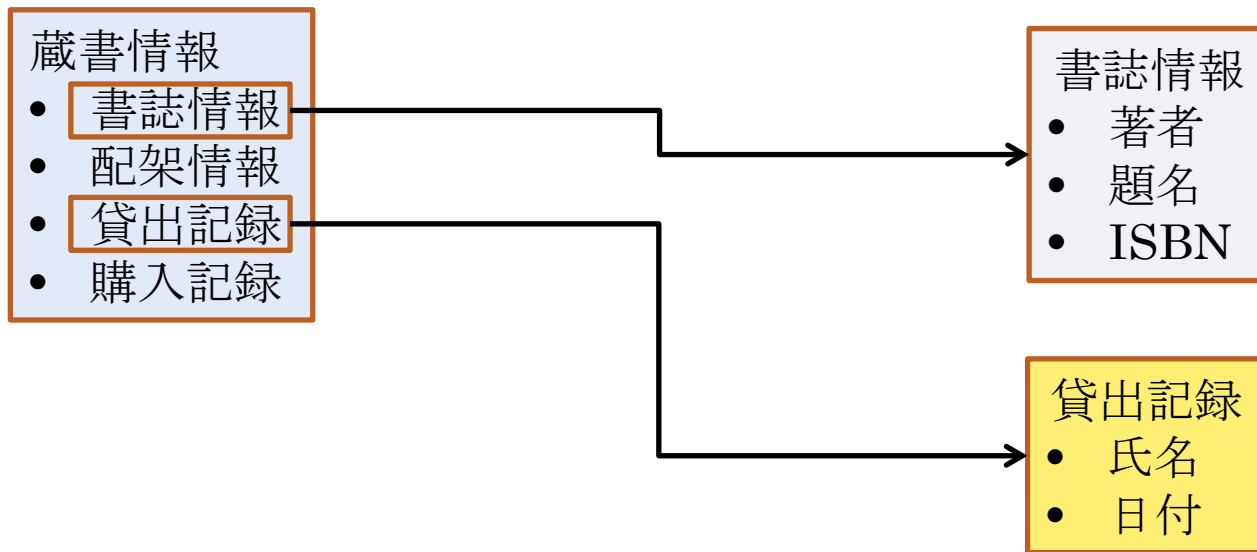


# クラスの継承例: クラス図



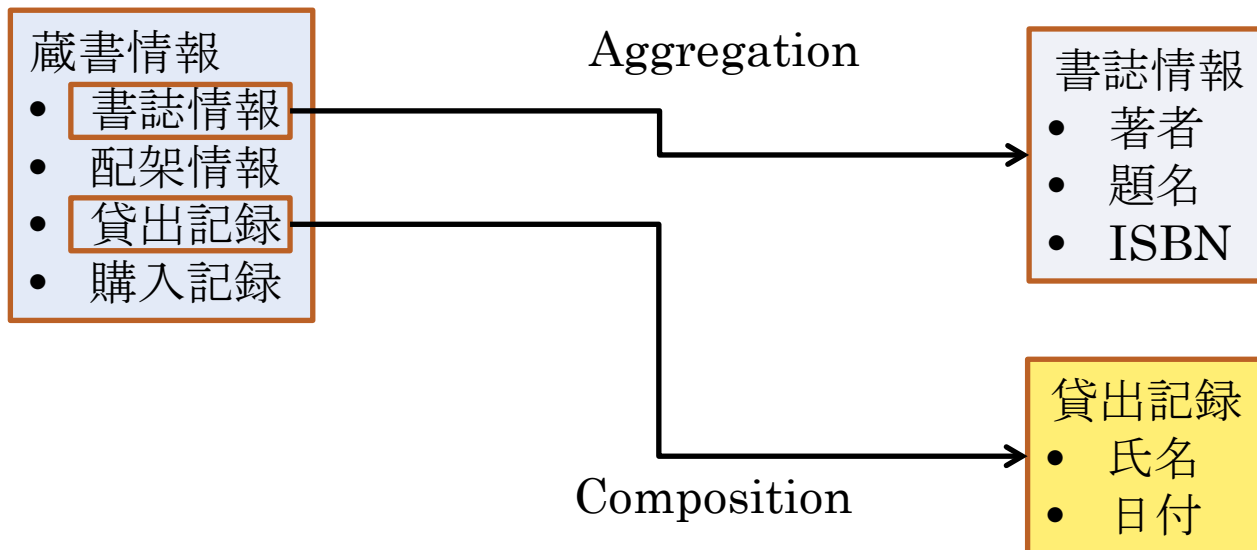
# 集約 (AGGREGATION)と合成集約 (COMPOSITION)

- オブジェクトの集まりをオブジェクトにする。
- 例: 図書館の蔵書情報

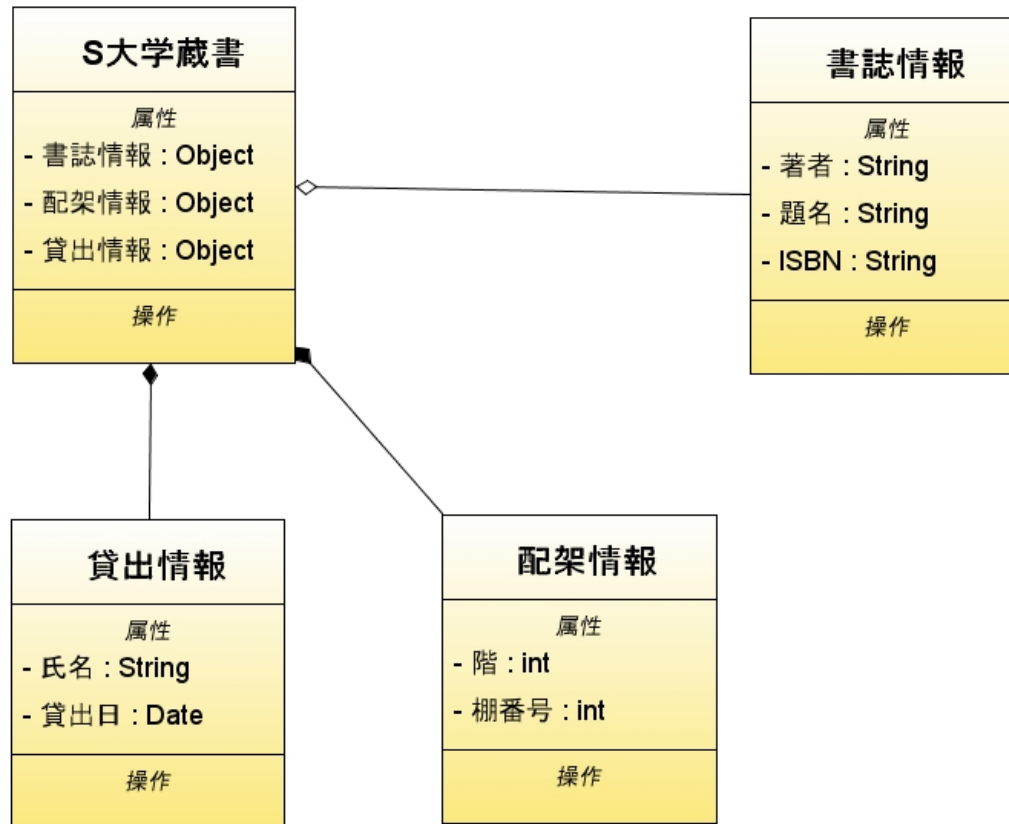


# 集約 (AGGREGATION)と合成集約 (COMPOSITION): 区別

- 集約: つながりの弱い場合
  - 書誌情報が外部DB:蔵書情報と独立に存在
- 合成集約: つながりが強い場合
  - 貸出情報は蔵書情報が無いと無意味



# 集約 (AGGREGATION)と合成集約 (COMPOSITION):クラス図



# 抽象クラス

- Methodが実装されていないクラス
- データの持ち方がほぼ決まっている
  - 一部methodが実装されていない
- 操作のされ方がほぼ決まっている
  - インターフェイスの名前が決まっている
  - 実装がされていない



# 多態性 (POLYMORPHISM)

- 同じメッセージに対して異なる処理を行う

