

# プログラミングII

<http://bit.ly/Prog3i>

## ソフトウェア開発方法論（1）

後期 第14週

2020/1/8

# 本日は・・・

- ▶ ソフトウェア開発の現状と問題
- ▶ ソフトウェア開発工程・方法論
- ▶ UMLとは
- ▶ クラス図について

# 本日は・・・

- ▶ ソフトウェア開発の現状と問題
- ▶ ソフトウェア開発工程・方法論
- ▶ UMLとは
- ▶ クラス図について

# ソフトウェア開発の現状

## ▶ 大規模化

必要なソースコードの行数が増大

→ 多くの開発者が共同で開発する必要あり

## ▶ 複雑化

社会が複雑化するのに合わせ、ソフトウェアへの

要求も複雑化

# ソフトウェア開発の問題

## ▶ 大規模化

必要なソースコードの行数が増大

→ 多くの開発者が共同で開発する必要あり

→ 共同開発は難しい

共同開発によるバグの増大

## ▶ 複雑化

社会が複雑化するのに合わせ、ソフトウェアへの要求も複雑化

→ 複雑化により設計, 実装がより難化

大規模化/複雑化したソフトウェアを完成させる方法論が必要

# ソフトウェア危機

ソフトウェアへの依存度が大きくなっていることに対する危機感を以下の主要な5項目で表す

- ▶ ソフトウェアの巨大化、複雑化に伴う開発費用の増大
- ▶ ハードウェア対ソフトウェアのコスト比の変化（「ハード > ソフト」から「ハード < ソフト」）
- ▶ ソフトウェア保守にかかる工数の増大
- ▶ ソフトウェア需要に対する供給能力の低下
- ▶ ソフトウェアトラブルの社会的問題化

# ソフトウェア工学について

次のような目的をもった**工学分野**

- ▶ ソフトウェアの生産性、信頼性の低下を防ぐ
- ▶ 学術的(理論や方法論)および、実践的(技術や技法)な分野を目指している

ソフトウェア工学が確立した経緯

- ▶ 1968年：ソフトウェア工学と**ソフトウェア危機**が提唱される
- ▶ 1970年代後半：ソフトウェア工学の分野が定着していく  
(ソフトウェア工学国際会議が開催される)

# 本日は・・・

- ▶ ソフトウェア開発の現状と問題
- ▶ ソフトウェア開発工程・方法論
- ▶ UMLとは
- ▶ クラス図について

# ソフトウェア開発手法

ソフトウェアを開発・運営する過程→**ライフサイクル**  
ライフサイクルは**プロセスモデル**で表現される

## 代表的なプロセスモデル

- ▶ ウォーターフォールモデル
- ▶ スパイラルモデル
- ▶ アジャイルソフトウェア開発

# ウォーターフォールモデル

いくつかの工程に分けて開発を進める

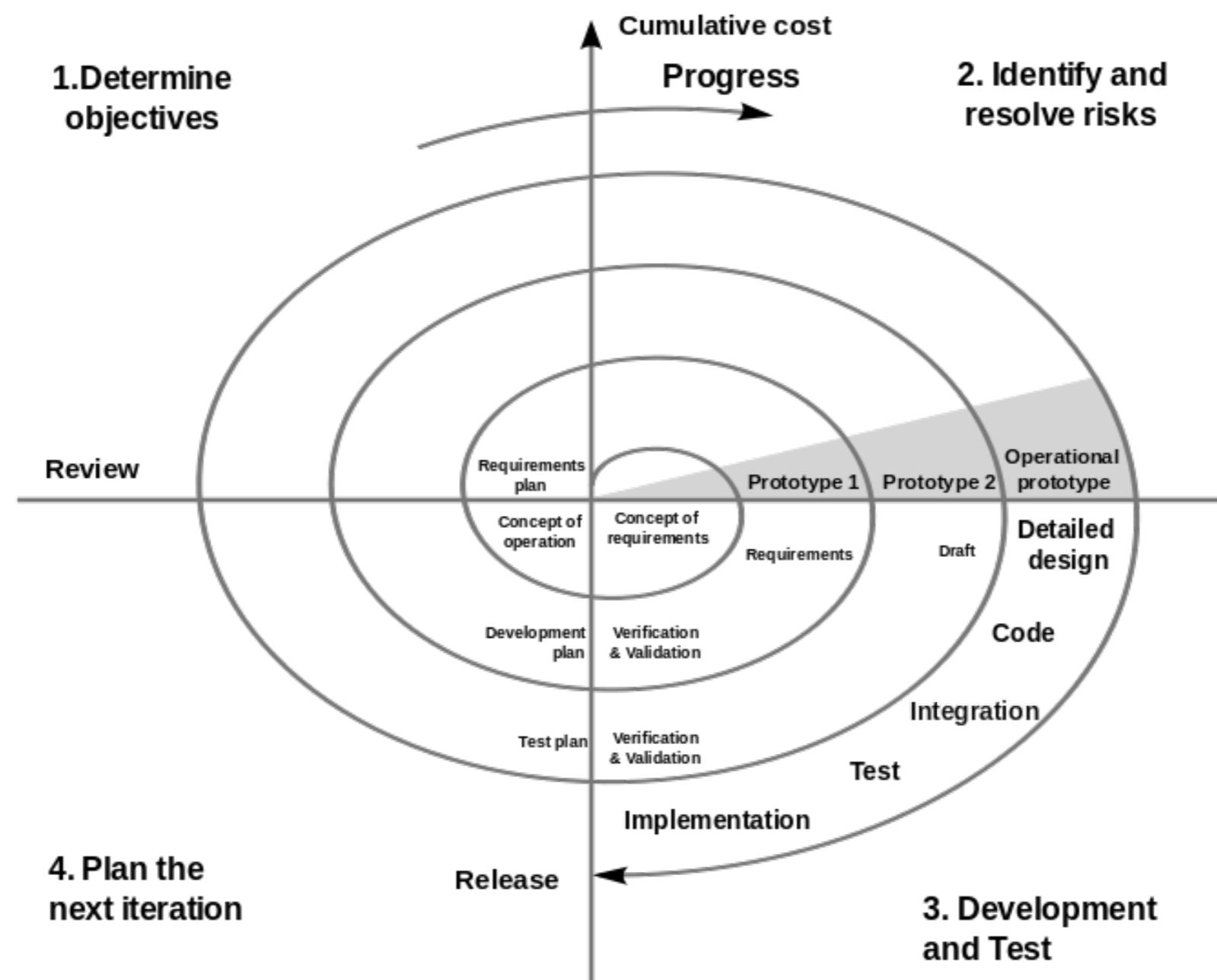


**上流工程**：利用者側の視点（要求定義, 基本設計）

**下流工程**：開発者側の視点（詳細設計, 実装, テスト, 保守）

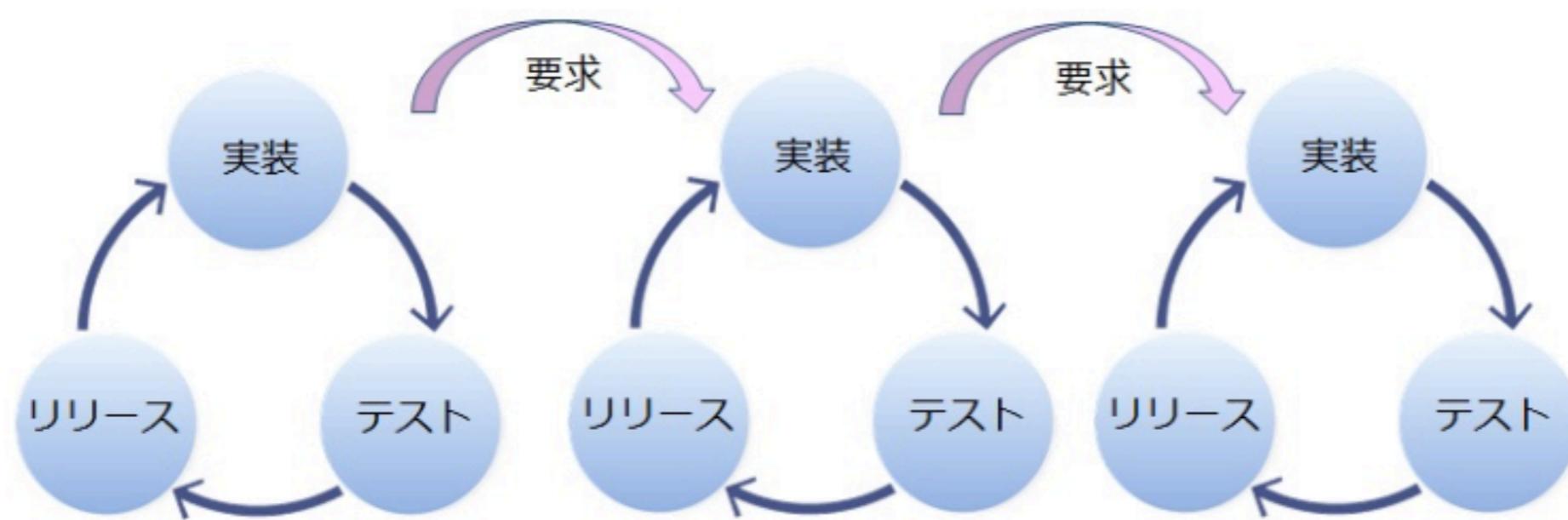
# スパイラルモデル

- ▶ 時間の経過とともに、ある方向へ向けて繰り返しながら成長する
- ▶ 4つのフェーズを1サイクルとし、**プロトタイピング**を繰り返す



# アジャイルソフトウェア開発

- ▶ Extreme Programmingが有名
- ▶ 軽量で短期間の開発に適している
- ▶ リリース（ある程度の形になって動作するソフトウェア）
  - 2~3ヶ月間隔
- ▶ イテレーション（ソフトウェアの部分的な設計・実装・テスト）
  - 2~3週間



いくつかのイテレーションを完了することでリリースを達成し、リリースを繰り返すことで完成度を上げる

# 本日は・・・

- ▶ ソフトウェア開発の現状と問題
- ▶ ソフトウェア開発工程・方法論
- ▶ **UML**とは
- ▶ クラス図について

# UMLとは

UML (Unified Modeling Language) は、OMG という団体により標準化された、**ソフトウェアの仕様や設計を図として表現するために使用する言語**です。

「クラス」の概念を用いたオブジェクト指向開発技法で利用されます。

現在はUML 2が利用されています。

# ダイアグラムの種類

UML 2では、**13種類の図（ダイアグラム）**が定められています。

本講義では、その中でも**使用頻度の高い**ダイアグラムを数種類を扱います。

# 参考文献

- ▶ 竹政照利 「はじめて学ぶUML」(ナツメ社)
- ▶ マーチン・ファウラー 「UMLモデリングのエッセンス」(翔泳社)
- ▶ 河村一樹 「ソフトウェア工学入門」(近代科学社)
- ▶ 高橋 麻奈 「やさしいJava オブジェクト指向編」(SBクリエイティブ)

# 本日は・・・

- ▶ ソフトウェア開発の現状と問題
- ▶ ソフトウェア開発工程・方法論
- ▶ UMLとは
- ▶ クラス図について

# この授業では・・・

## 一番利用頻度の高いクラス図を学びます

クラス図とは、**クラスの構造**（フィールド, メソッドなど）と**クラス間の関係**（集約, 繙承など）といったソフトウェアの**静的構造**を表すためのダイアグラム

# 【課題の準備】

演習室で作業する前に、以下のコマンドを入れるだけで準備が完了する

```
$ mygitclone 「自分のGitHubユーザ名」  
$ cd prog3i-ユーザ名  
$ ./myconf
```

※本体をシャットダウンするまでは、  
上記「mygitclone」と「myconf」の設定は有効です

# 【課題の準備】

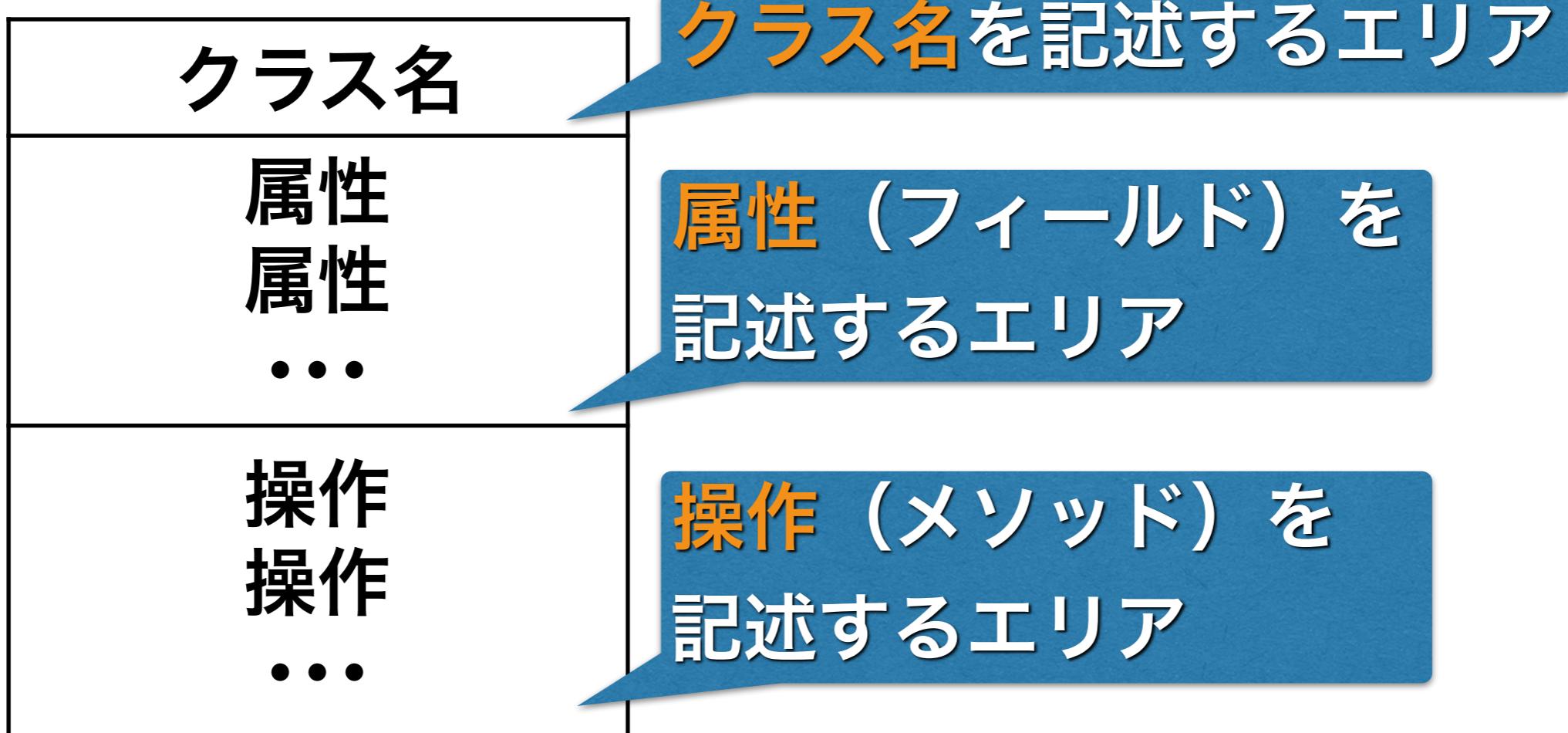
以下の流れで、課題のプログラムを作るためのフォルダを準備しましょう。

1. 端末を起動して、以下のコマンドを実行して後期第14週のフォルダを作る

```
$ cd prog3i-ユーザ名          ←既に移動しているなら不要)  
$ mkdir week214  
$ cd week214
```

# クラスの表記

クラスは、以下の3つのエリアに分けて描きます。



# 属性の表記

属性はクラスが持つデータを表現します。必須なのは「名前」のみで、その他は省略可能です。

## 【書式】

可視性 属性名: 型 = 初期値

- ▶ **可視性**: 他のクラスから参照可能かを表す  
「+」(public), 「-」(private), 「#」(protected), 「~」(package)
- ▶ **属性名**: 属性を表す名前
- ▶ **型**: 属性の型
- ▶ **初期値**: 属性が最初に持つ値

# 操作の表記

操作はクラスが持つ振る舞いを表現します。必須なのは「名前」のみで、その他は省略可能です。

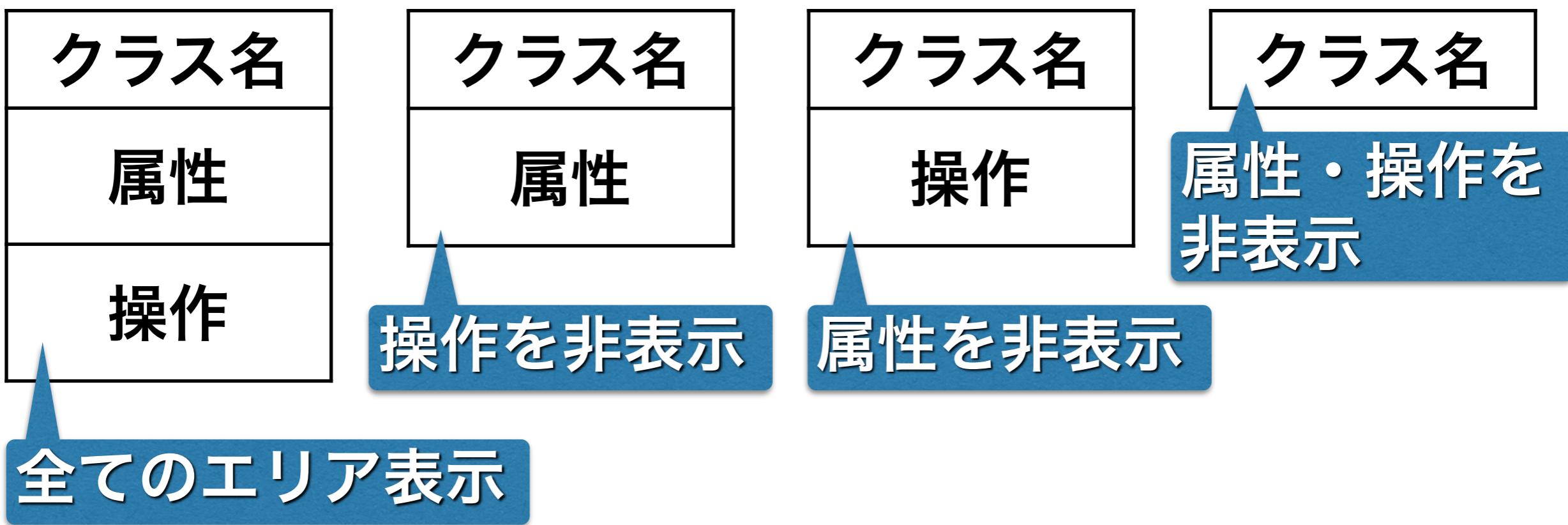
## 【書式】

**可視性 操作名(引数名: 引数の型, ... ) : 戻り値の型**

- ▶ **可視性**: 他のクラスから参照可能かを表す  
「+」 (public), 「-」 (private), 「#」 (protected), 「~」 (package)
- ▶ **操作名**: 操作を表す名前
- ▶ **引数名**: 引数を表す名前
- ▶ **型**: 引数と戻り値の型

# クラス表記のバリエーション

クラスは属性および操作の表示領域のどちらか、または両方を**非表示**にすることができます。（つまり、非表示になっていても、属性・操作が定義されている場合があるので注意して下さい。）



# クラス間の関係

クラス図におけるクラス間の関係について、以下の関係を学びます。

▶ 集約

▶ 繙承（汎化）

# 集約

2つのクラスが**全体と部分の関係**を表す。（例えば、商品リストと商品など）

【例】 クラスAがクラスBを集約している



全体となるクラス

部分となるクラス

# astahの補足

## 関係の作成について

1. ツールバーのボタンから関係の選択し、2つのクラスをつなぐ
2. 作成した関係をマウスで選択すると、画面左下のプロパティに関係の情報を見ることができる  
(「関連端A」「関連端B」のタブに、接続されているクラスの情報などが表示される)

## 関係の削除について

削除する際は、右クリックのメニューで「モデルから削除」を実行する。Deleteキーで削除してしまうと、見た目は消えるが、クラスには「関係でつながっている」ことが残る。その場合は、そのクラスを選択して、画面左下のプロパティから「関連」タブを開いて、関係しているクラスを削除する。

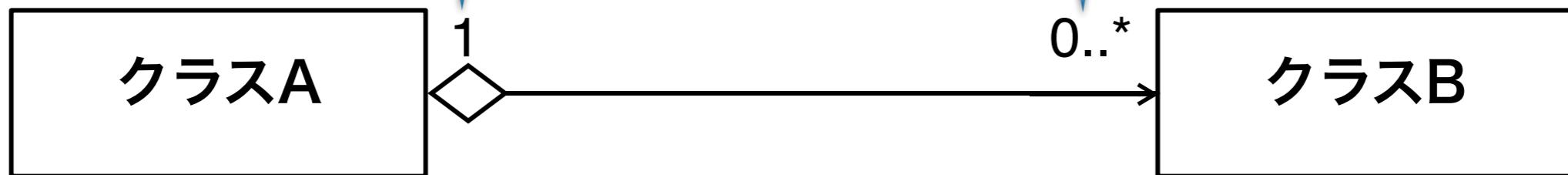
# 多重度

「クラスのオブジェクト（つまりインスタンス）が接続する可能性のある数」を多重度と呼び、関係の線にその情報を描く。

【例】 クラスAのオブジェクト**1個**に対して、クラスBのオブジェクトが**0個以上**集約する可能性がある場合

オブジェクトが1個

オブジェクトが0個以上



# 多重度の表記

「..」は「範囲」  
を表す

「\*」は「いくつ  
でも」を表す

表記	意味
0..1	0または1
1	1
0..* または *	0以上
1..*	1以上
3..10	3から10

# astahの補足

## 多重度の入力について

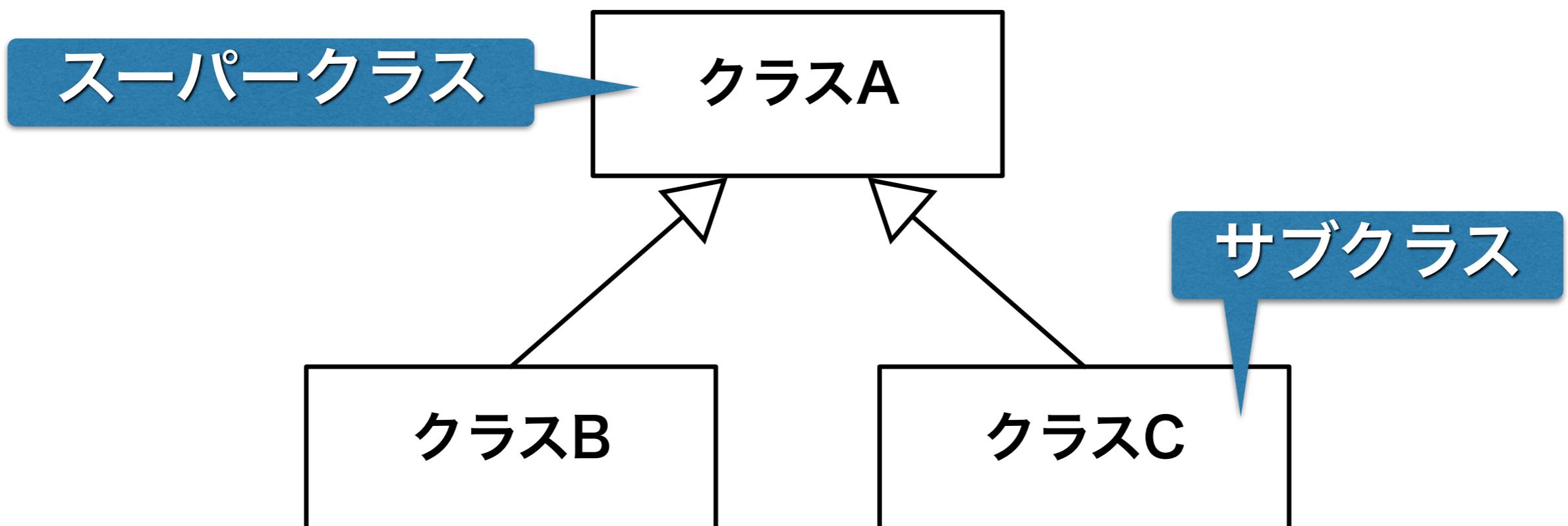
多重度は次の2つの方法で入力できる

- ▶ 関係のプロパティから「関連端」のタブで多重度を選択または入力する
- ▶ 入力する関係の関連端付近に表示される「m」をクリックする

# 継承（汎化）

「スーパークラス（親クラス）とサブクラス（子クラス）の継承関係」を関係線で表す。

【例】 クラスBとクラスCがクラスAを継承している



# astahの参考サイト

- ▶ UML初学者向けチュートリアル

[http://astah.change-vision.com/ja/  
tutorial/tutorial-community.html](http://astah.change-vision.com/ja/tutorial/tutorial-community.html)

- ▶ 基本操作ガイド

[http://astah.change-vision.com/ja/files/  
astah\\_Basic\\_Operation\\_Guide.pdf](http://astah.change-vision.com/ja/files/astah_Basic_Operation_Guide.pdf)

# 【練習14-1】

astahを使って、次のJavaプログラムのクラス定義から、UMLクラス図を作成しましょう。

```
class MyClass {  
    private int num1;  
    private int num2;  
    public void setNum1(int n) {} // num1の値を設定する  
    public void setNum2(int n) {} // num2の値を設定する  
    public int sum() {} // num1+num2を返す  
}
```

MyClass
- num1 : int - num2 : int
+ setNum1(n : int) : void + setNum2(n : int) : void + sum() : int

完成したら、属性名と操作名のみの表示に変更してみましょう。



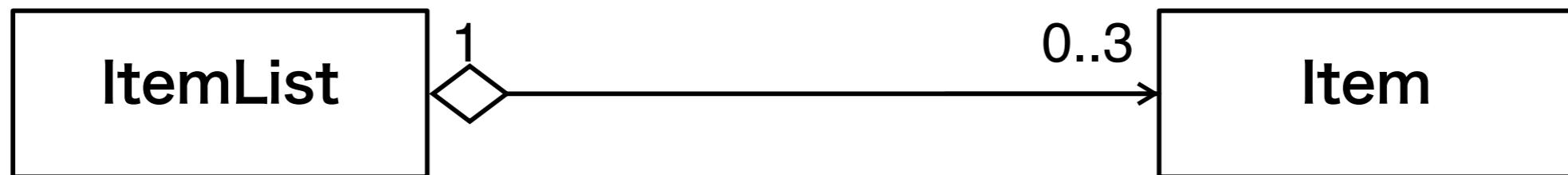
MyClass
num1 num2
setNum1() setNum2() sum()

# 【練習14-2】

astahを使って、次のJavaプログラムのクラス定義から、UMLクラス図を作成しましょう。

```
class ItemList {  
    private Item[ ] list;  
    ItemList() {  
        list = new Item[3];  
    }  
    public void show() {  
        // list[0]～list[2]の  
        // show( )を呼び出す  
    }  
}
```

```
class Item {  
    private String name;  
    private int price;  
    public void setName(String n) {}  
    public String getName() {}  
    public void setPrice(int p) {}  
    public int getPrice() {}  
    public void show() {}  
}
```

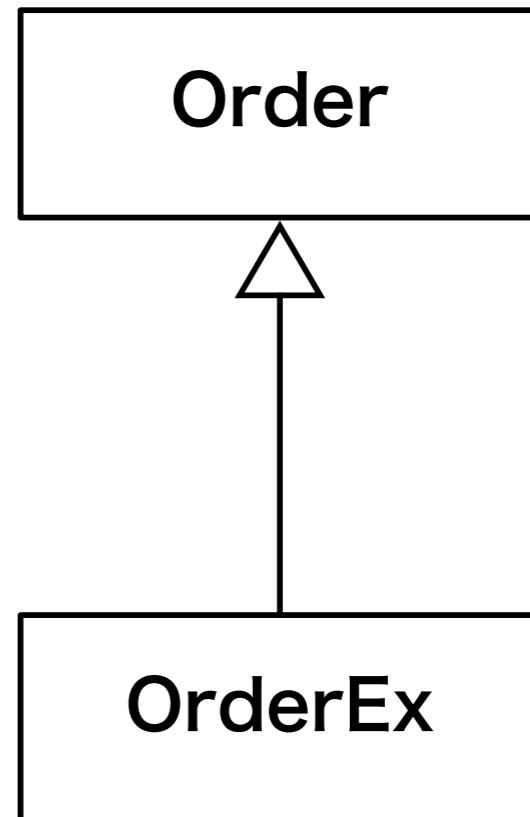


# 【練習14-3】

astahを使って、次のJavaプログラムのクラス定義から、UMLクラス図を作成しましょう。

```
class Order {  
    protected String name;  
    protected int price;  
    protected int quantity;  
    public void show() {}  
    public int calcPrice() {}  
}
```

```
class OrderEx extends Order {  
    private int off;  
    private String genre;  
    public void show() {}  
    public int calcPrice() {}  
}
```



# 【課題14-1】

別紙に示すプログラム中のクラス定義から、クラス Date, Time, Schedule, ScheduleMainを、UML のクラス図として作成してください。

なお、ファイル「2\_14\_schedule.ast」には、既にクラスDateが描かれていますので、残りのクラスを追記すれば完成します。

## 【補足】

mainメソッドの「static」は、astahの画面左下のプロパティ部分から設定可能です。

# 【課題の提出】

以下の流れで、作ったプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A  
$ git commit -m "課題14-1提出"  
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))

# 【課題14-2】

別紙に示したプログラム中のクラス定義から、クラス図の集約（多重度含む）と汎化の関係を追加してください。

ただし、この課題で新規に追加するクラスについては、クラス名のみで属性・操作は空欄で構いません。

クラスの関係を表すために宣言されているフィールド変数について

クラス図ではこれを関係線として描いた場合は、意味が重複するため、フィールド変数（つまり属性）には描かない。（このダイアグラムから生成されるスケルトンコードに影響するため）

# 【課題の提出】

以下の流れで、作ったプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A  
$ git commit -m "課題14-2提出"  
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))