慶応義塾大学文学部
応用情報処理V 2018

クラスとプログラム全体の構造

Chapter 9
2018-11-30

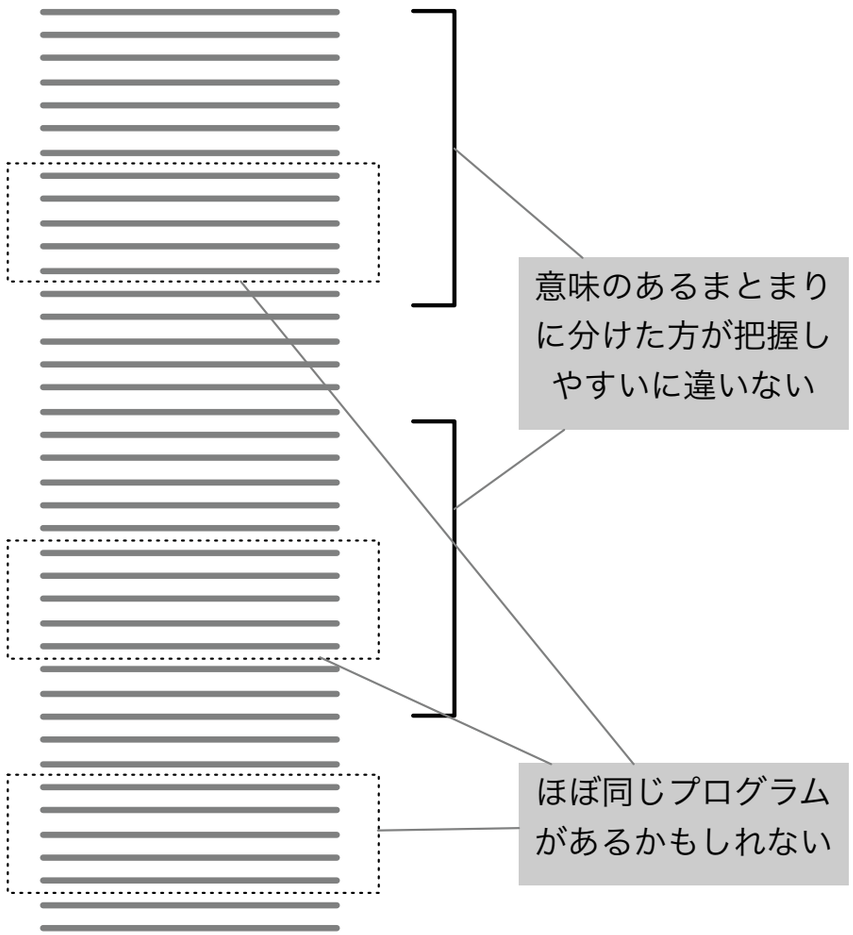
新居雅行 Masayuki Nii Ph.D. in Engineering
nii@msyk.net msyknii msyk_nii

プログラミングでの様々な考え方

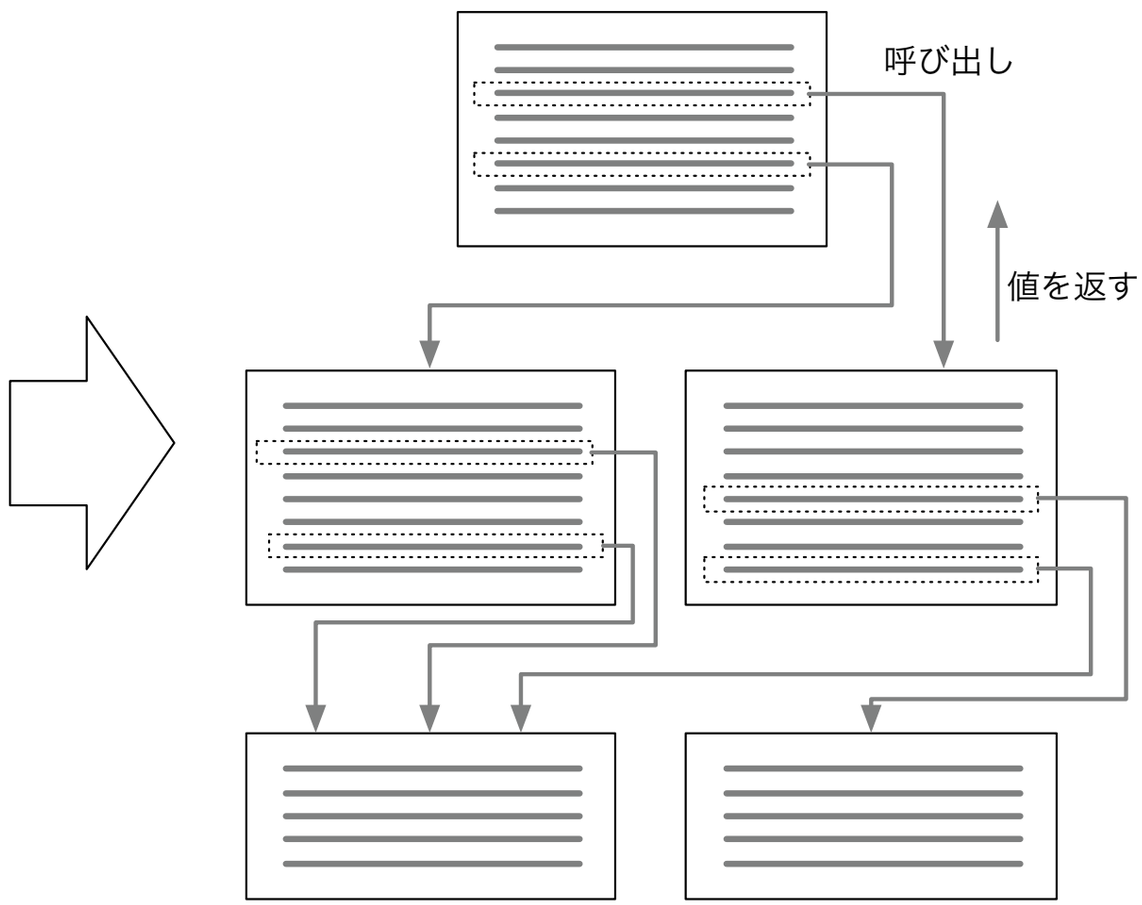
- **プログラムは「ステップの集まり」という見方**
  - プログラムの表現力を高めるために、条件分岐、繰り返し、呼び出しなどの仕組みが古くから追加された
  - 作成可能なプログラムが巨大になって来たとき、全体を把握することはどうすればいいのかという点が問題になる
- **サブルーチンと構造化**
  - 共通するプログラムを1箇所に記述して、各所からそれを呼び出すという「ライブラリ」的な考え方
  - プログラムの1つの塊を、論理的な意味のあるものと対応づける考え方
  - 階層的に整理するという考え方
- **アルゴリズムによる開発の効率化**
  - 最適なステップ作成の方針、あるいは例（こうすれば確実にソートができる、など）

プログラムの構造化に至る考え方

長いプログラムがあったとする



分割し、階層的に理解できるようにする



- **構造化プログラミング**
  - 呼び出しに加えて、逐次処理、繰り返し、分岐での記述であれば、プログラムの正しさを証明できるという手法
  - 「GoToがなくてもプログラムが作成できる」という点だけに注目される場合もある

オブジェクト指向

- **プロセスではなく、扱うデータを主体にする考え方**
  - データが処理機能を持つという考え方
- **基本ルールは極めてシンプル**
  - ある対象はいくつかの属性[プロパティ]を持つ。プロパティには値を割り当てることができる＝変数と同様な性質
  - ある対象は、自分自身を処理する仕組み[メソッド]を持つ
  - どんなプロパティやメソッドがあるのかといった対象に関する仕様を[クラス]と呼ぶ
  - クラスによって記述された対象を[オブジェクト]と呼ぶ
  - これだけの簡単なルールで森羅万象を記述しようという考え方

物事をオブジェクトとして捉える

- **例えば、自動車1台1台を捉える**
  - 属性にはその自動車特有のものもあるが、全ての自動車が持つ属性もある
    - 全自動車が持つ属性＝車種、車輪数、所有者*、ナンバー* (*: 値は自動車ごとに異なる…所有者は微妙か)
    - 一部の自動車が値を持つ属性＝搭載無線機、チャイルドシートの商品名
  - 自動車にある機能を「メソッド」として捉える
    - 例：発進する、停止する、点灯する
- **あらゆるものをオブジェクトとして捉える**
  - なんでも、プロパティとメソッドを持つクラスで記述できる
  - クラスを記述するプログラミング言語は、汎用性が高い
  - 現実的には、用途に応じたプロパティやメソッドだけが記述できれば事は足りる

属性をプログラムで記述する

- **ある対象が持つ値をプロパティと呼ぶ**
  - 「人間」は、「姓」「名」「年齢」という属性を持つとする
- **属性を記録するために、変数を用意する**
  - 変数名は、familyName、firstName、ageとする
- **つまり、プログラムでこのように記述できる**

```
String firstName;
String familyName;
int age;
```

  - この3つの「変数」があれば、1人の人間の情報を記録できるという考え方
  - データの内容に応じて、変数の型を選択する

属性を持つ対象をクラスとして記述

- **姓、名、年齢をひとまとまりのものとして区別した**
  - クラスとして1つにまとめる
  - 名前はPersonalityとする
- **次のように記述できる**

```
class Personality{
    String firstName;
    String familyName;
    int age;
}
```

  - 属性＝プロパティ、フィールドなどいくつかの呼び方がある
  - プロパティとなる変数＝メンバー変数

クラスに機能を追加する

- **フルネームが欲しい**
  - フルネームを構成する姓と名はプロパティとして定義済み
  - それらをつなげれば良い＝処理方法が確定した！
  - メソッドとして定義すれば良い（仕様は表の通り）

メソッド名	getFullName
返り値	姓と名を半角スペースでつなげた文字
引数	なし

- **実装結果**

```
class Personality{
    String firstName;
    String familyName;
    int age;

    String getFullName() {
        String fullName = firstName + " " + familyName;
        return fullName;
    }
}
```

メンバー変数を参照できる

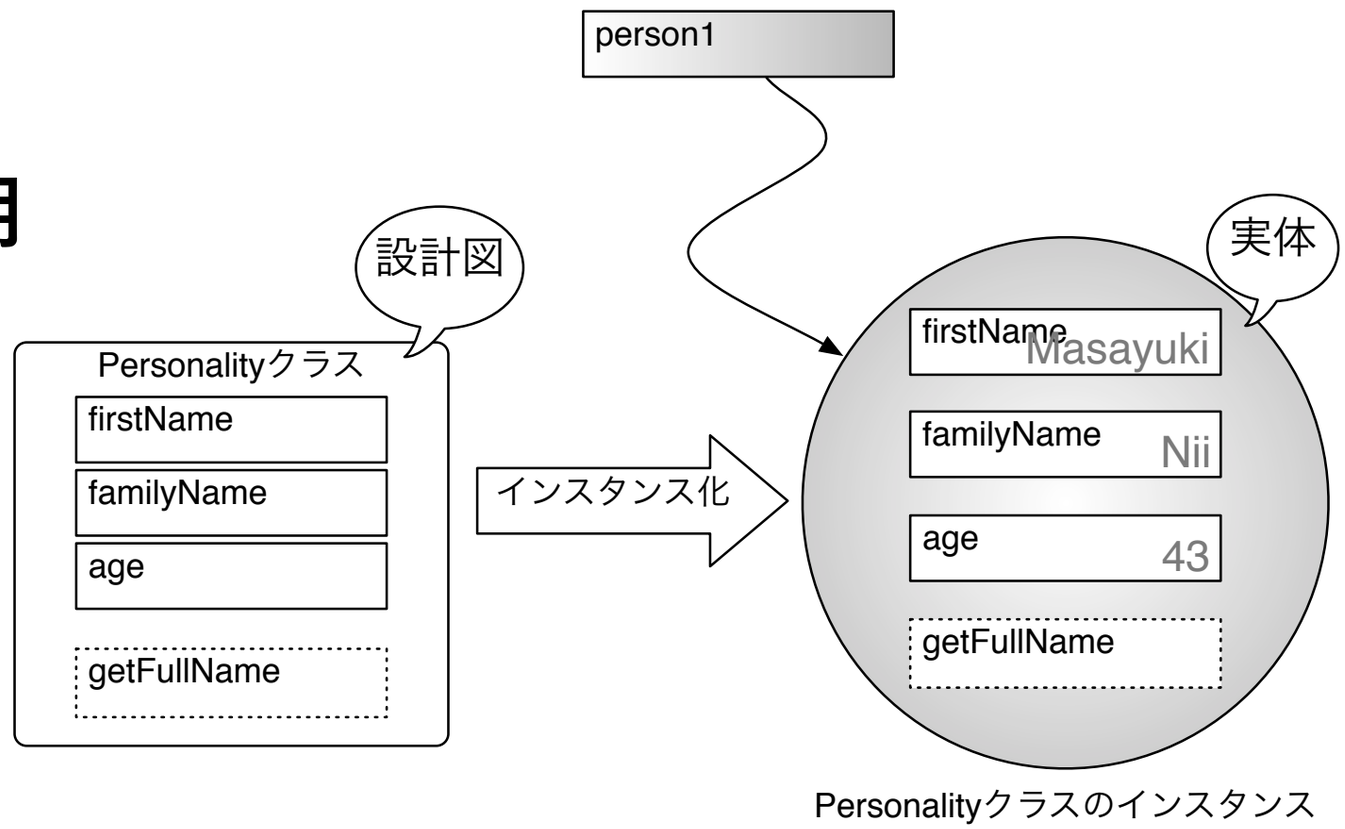
クラス（オブジェクト）の塊の中にあるので、すぐに手が届くというイメージ

オブジェクトはクラスを元に生成される

- **クラスは設計図、それを実体化(インスタンス化)したものをオブジェクトと呼ぶ**
  - 実体とは、「メモリ内に確保された領域」と理解しても良い
  - 実体化するには「new クラス名()」。newは一種の演算子
- **作った実体は、後々使いたい**
  - 実体が「どこにあるか」を記述した、「参照」というデータがある
  - 参照を変数に代入しておけば、変数からオブジェクトの実態にたどり着ける

プロパティやメソッドの利用

- 「参照.プロパティ」は変数と同様に扱える
- 「参照.メソッド(引数)」は値を返す



プログラムでのオブジェクトの生成と利用

- **例えば、次のプログラム**

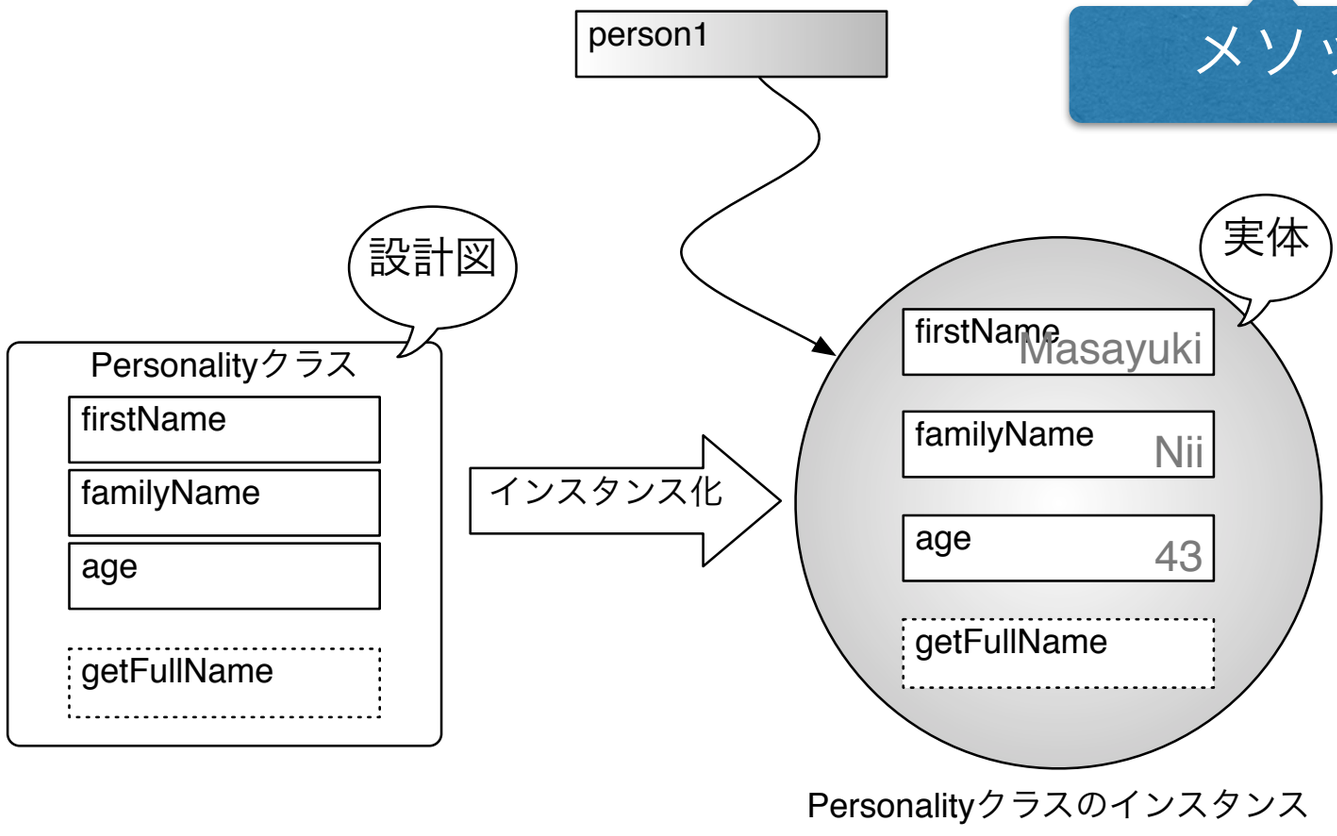
```
Personality person1 = new Personality();
person1.firstName = "Masayuki";
person1.familyName = "Nii";
person1.age = 43;
System.out.println("フルネームは"+person1.getFullName());
```

参照が変数に代入される

クラス名は型名でもある

プロパティに代入

メソッド呼び出し



自分自身を参照するthis

- **内部からはどうやって参照しているか？**
  - thisというキーワードが定義されていて、自分自身を参照
  - しかしながら、多くの場合、省略可能であるため、書かれないこともよくある
- **Personalityクラスをthisを使って記述した例**

```
class Personality{
    String firstName;
    String familyName;
    int age;

    String getFullName() {
        String fullName = this.firstName + " " + this.familyName;
        return fullName;
    }
}
```

コンストラクター

- **オブジェクト生成と同時に何かしたい事は多い**
  - クラスにコンストラクターという特殊なメソッドを実装できる
  - newで生成した時に、コンストラクターが定義されていれば、呼び出される
  - 引数を指定できる。言い換えれば、new クラス名(引数) において、引数の型のパターンに一致したコンストラクターが呼び出される
- **コンストラクターのルール**
  - 返り値は記述しない
  - メソッド名はクラス名と同一
  - メソッド内部では、returnを最後に書く必要なし。つまり、値を返さなくても良い
  - コンストラクターを1つでも定義したら、引数のないコンストラクターを通常は必ず定義しないといけない

コンストラクターと利用例

- **コンストラクターを追加したPersonalityクラス**
  - class Personality{

```
// メンバー変数の定義
String firstName;
String familyName;
int age;

// 以下のメソッドが「コンストラクター」
Personality( ){ }

Personality( String initFirstName, String initFamilyName, int initA
    this.firstName = initFirstName;
    this.familyName = initFamilyName;
    this.age = initAge;
}

// 通常のメソッド
String getFullName() {
    String fullName = this.firstName + " " + this.familyName;
    return fullName;
}
}
```

呼び出された時点でオブジェクトは作られていると考えて良い

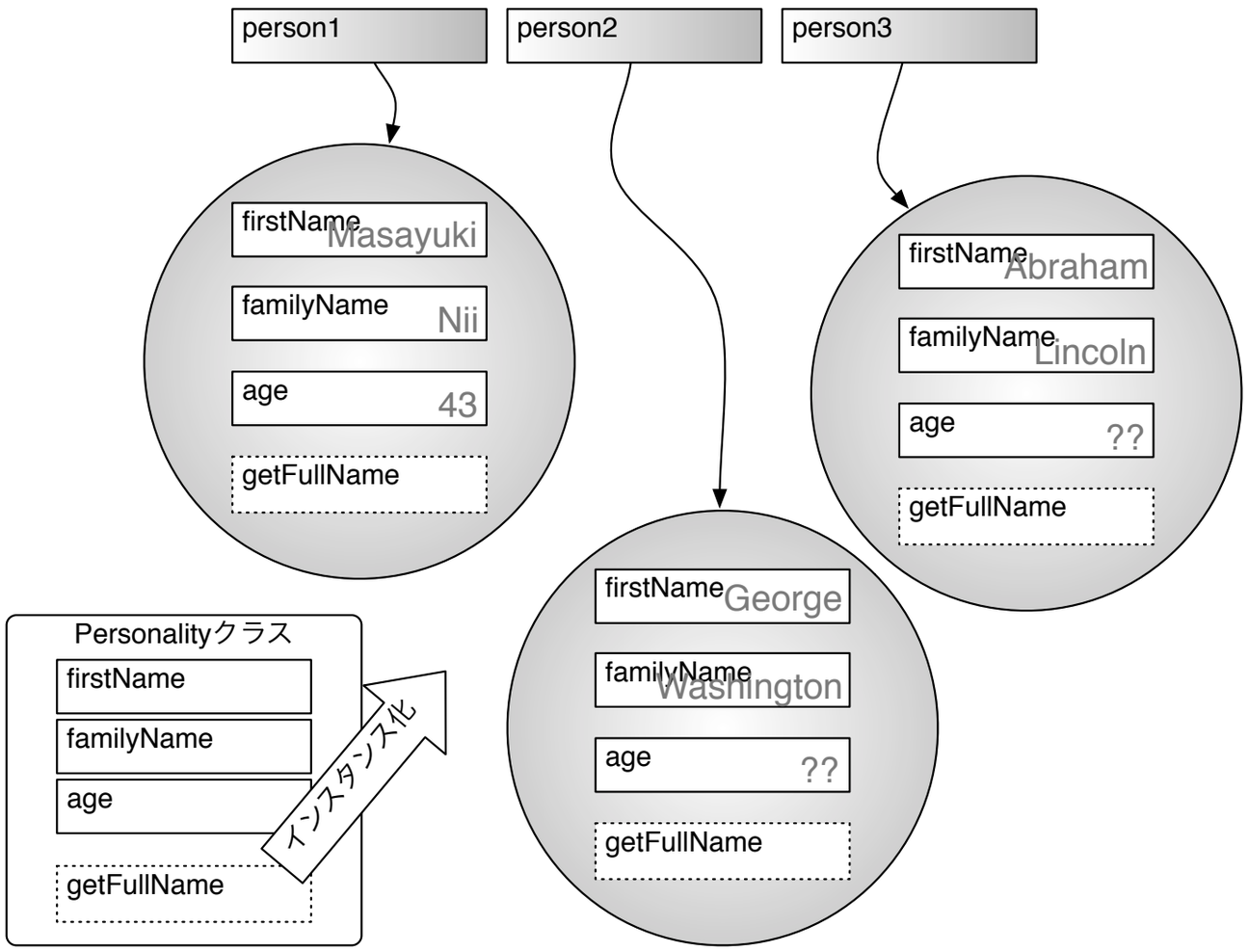
引数がString, String, intでクラス生成すると、このコンストラクターが呼び出される。生成側で指定したデータが引数に設定されて呼び出される

- **コンストラクターの利用例**
  - Personality person1 = new Personality("Masayuki", "Nii", 43);
    System.out.println(person1.getFullName() + "さんの年齢は" + person1.age

コンストラクターにより、初期化をまとめて行えるというのは、典型的な利用法

複数のインスタンス

- **複数のインスタンスを生成した…**
  - 当然のこととして、参照先はそれぞれ違う
  - 1つの変数では覚えられない。原則としてオブジェクトの数だけ参照は必要



講義のまとめ

- 属性としてのプロパティ、処理機能としてのメソッドとしてあらゆる対象を記述するオブジェクト指向の考え方
- 対象はプロパティとメソッドの集合であるクラスとして捉える
- クラスを設計図として、動くソフトウェアを生成して利用するのが基本的な考え方（インスタンス化）
- 生成結果は「参照」により、プログラム内で活用できる
- 自分自身を参照するthis
- 生成時にメソッド呼び出しができるコンストラクター