Programming II

第3回 オブジェクト指向の概念（第7章）

塩澤秀和

3.1 ここからは，オブジェクト指向

- オブジェクト指向とは（p.277）
  - プログラムを，オブジェクト（「もの」という意味）と呼ばれる部品の組み合わせで設計し，部品ごとに開発する技術

- オブジェクト指向の歴史
  - Simula（1960年代）
    プログラムの中に，「工場の製造装置」や「お店の客」などの構成要素を想定し，計算によって状況を予測するシミュレーションのために開発された言語
  - Smalltalk（1970年代）
    人間にコンピュータを扱いやすくするために，「ウィンドウ」や「ボタン」などを含むソフトウェアの構成要素同士がメッセージをやり取りして動作するという思想で作られた言語

要するに，プログラムの中に何か“もの”と関係を想定して“動かす”

3.2「オブジェクト」の特徴

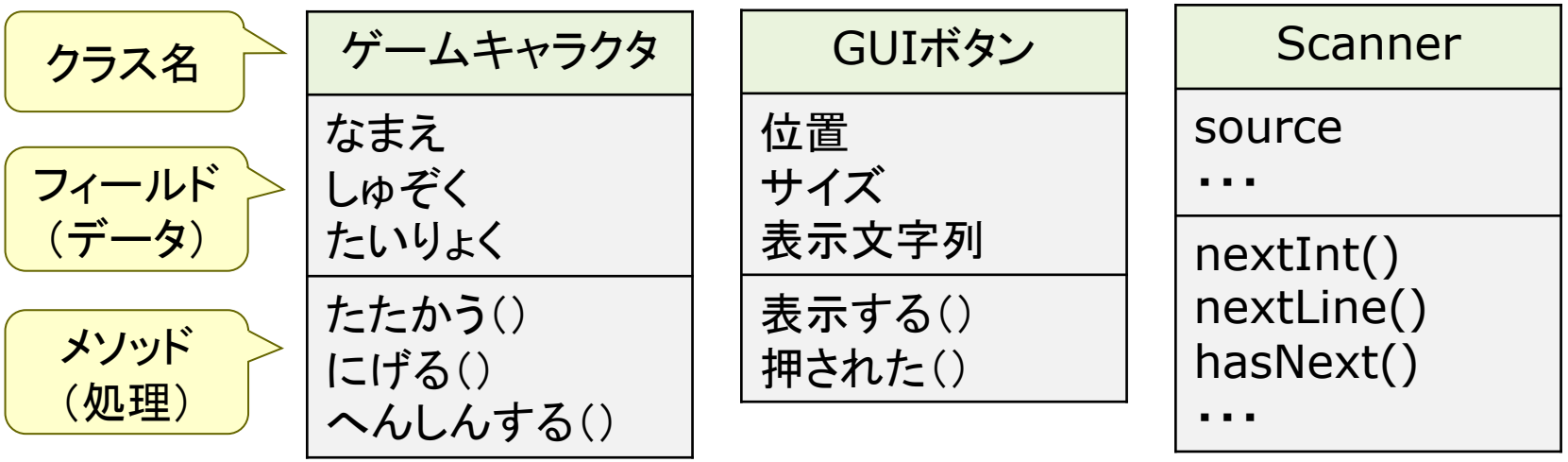
- オブジェクトの例
  - ゲームのキャラクタ，GUI画面のボタン，文字列（String）
  - 電話帳アプリの項目，乱数発生器（Random），Scanner
- オブジェクトの構造（p.289）
  - 状態を表すデータと，そのデータに関連した処理を持つ
  - 言語等によって呼び方が違う / 両者を「メンバ」ともいう

	UML	Java	他に使われる用語
データ	属性	フィールド	インスタンス変数，メンバ変数
処理	操作	メソッド	メンバ関数

3.3 まず，「クラス」を設計する

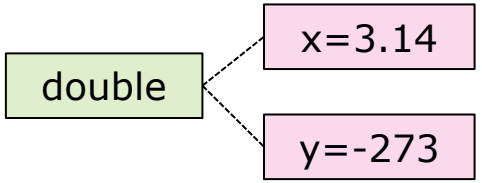
- クラスとは（p.302）
  - オブジェクトの“種類”（class）ごとに定義する“設計図”
  - クラス定義の記述だけでは，動く“実体”は生成されない
  - 「int」や「double」などのデータ型の拡張ともいえる

- クラスの例（UMLクラス図）



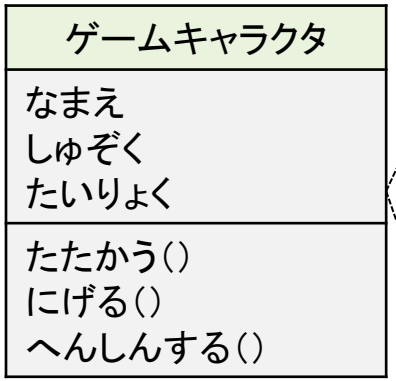
3.4 クラスとインスタンス

データ型と変数の関係に類似

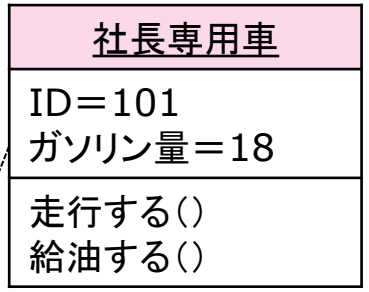


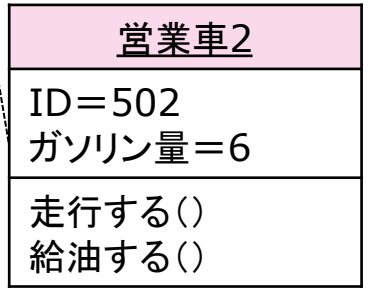
- インスタンスとは(p.305)
  - クラスに実際の値を入れて実体化したもの

クラス



インスタンス
(オブジェクト図)

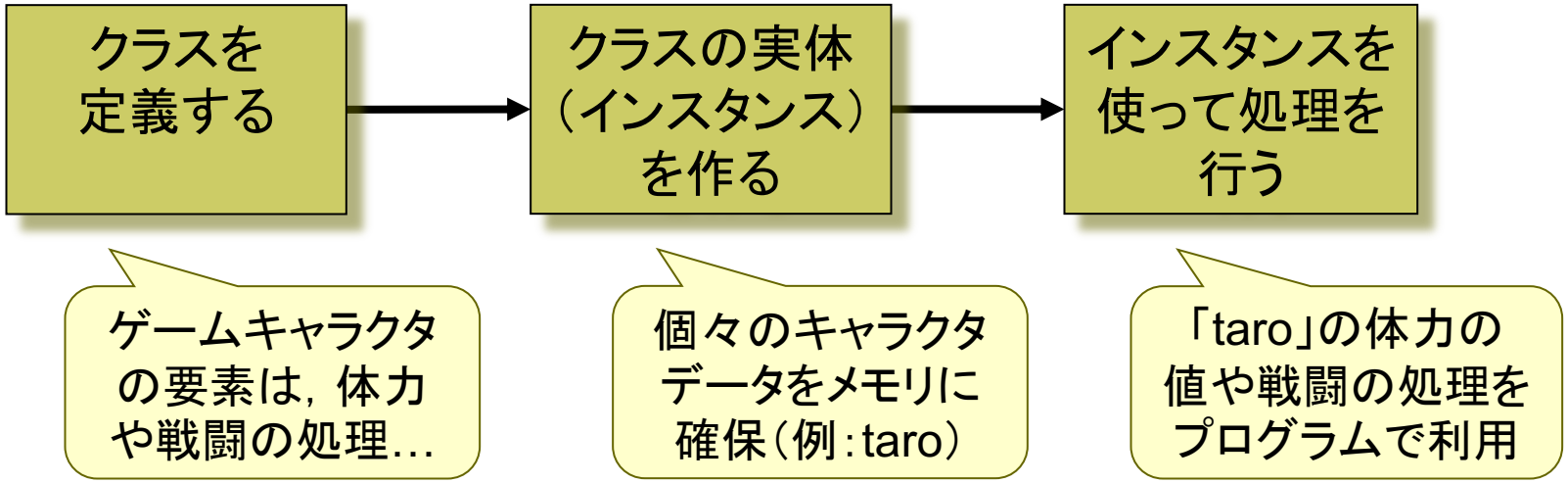




フィールドの
値は違うが
メソッドは共通

3.5 オブジェクト指向のプログラム

- クラスを使ったプログラムの流れ（p.302）



- インスタンスの生成方法（p.320）
  - クラスだけ書いても，インスタンスを作らないと動かない

    ※ ただし，前回説明した「メソッドの分類のためだけのクラス」は例外
  - Javaでは，new演算子でインスタンスを作る

```
GameCharacter taro = new GameCharacter();
```

3.6 定義済みクラスを使う（Scanner）

```
// Scannerクラスは，標準のjava.utilパッケージで定義済み
import java.util.Scanner;

public class Main {
	public static void main(String[] args) {

		// Scannerを使う準備
		// 新しい「実体」（インスタンス）を作って変数scに代入する
		Scanner sc = new Scanner(System.in);

		// 変数scに代入した「実体」を介して，Scannerのメソッドを使う
		int a = sc.nextInt();

		System.out.println(a);
	}
}
```

3.7 自作のクラスを定義する(p.310)

```
// ロボットを表すクラス
class Robot {
    String name;

    void hello() {
        System.out.println("コンニチハ " + this.name + " デス");
    }
}

// メインプログラム
public class Main {
    public static void main(String[] args) {
        Robot robo1 = new Robot();
        // ここに次ページのコードを書いてみる
    }
}
```

フィールド(属性)(p.311)

メソッド(操作)(p.312)
staticをつけないことに注意

thisは自分が持っているメンバから探すことを表す(p.313)

インスタンスの生成(new)(p.320)

3.8 インスタンスを使った処理

- 「.」演算子（p.321）
  - インスタンス（orクラス）のメンバにアクセスする演算子
  - 自分のインスタンス内へのアクセスは「this.〇〇」と書く
- フィールドを使った処理
  - 例: robo1.name = "ロボタ"; ←フィールドの値の設定
  - 例: String n = robo1.name; ←フィールドの値の取得
  - つまり，「robo1.name」で1つの変数だと思えばよい
- メソッドを使った処理
  - 例: robo1.hello(); ←インスタンスが持つメソッドを実行
  - robo1.nameに別の値を代入して結果を比較してみよう

3.9 メソッドに関するまとめ

- static修飾子がついたメソッド
  - インスタンスの準備が要らないメソッド(「クラスメソッド」)
  - クラス名.メソッド名(引数)
  - 例: Message.kaishin(n)
- newで生成したインスタンスのメソッド
  - オブジェクト指向における「操作」
  - インスタンス.メソッド名(引数)
  - 例: robo1.hello()

例えば, mainでインスタンスrobo1の機能を使うとき

- クラス定義内で自クラスが持つメソッド
  - this.メソッド名(引数)
  - 例: this.hello()

例えば, Robotクラスの中で自クラスの機能を使うとき

3.10 クラス定義の標準的な方法

- クラスは別々のJavaファイルに分ける
  - クラス名は大文字，メンバ名は小文字で始める（p.313）
  - 各クラスは，「クラス名.java」というファイルに記述する
  - その際，クラス定義にpublic（公開）修飾子が必要になる
  - 【注意】ここまでの学習範囲では，プログラムを複数のパッケージに分けると動かなくなるので，そうしないこと

Main.java

```
public class Main {

    public static void main(…) {
        // 省略
    }
}
```

外から利用可

Robot.java

```
public class Robot {

    String name;

    void hello() {
        // 省略
    }
}
```

演習

- 3.7～3.10
  - 最終的なソースコードと実行結果を提出

第3回 まとめ

- オブジェクト指向
- オブジェクトの構造
  - データ，属性，フィールド
  - 処理，操作，メソッド
- クラス
  - UMLのクラス図とオブジェクト図
  - クラスの定義方法（class）
  - クラス定義の標準的な方法（「クラス名.java」）
- インスタンス
  - インスタンスの生成（new演算子）
  - インスタンスのフィールドとメソッドの利用（「.」演算子）