Programming II

第5回 クラスの基本機能（第9章）

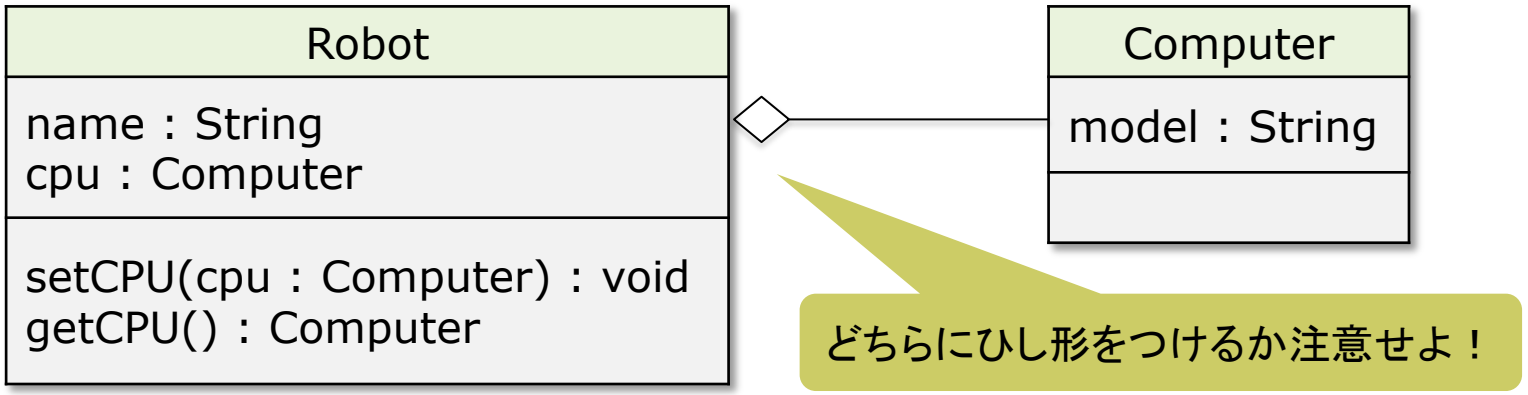
塩澤秀和

プログラミングの原則

- プログラムは正しく動くことが第一だが…
  - 正しく動くように見えるだけでは，「よいプログラミング」とはいえません
  - その方法が効率的なのか，どんな場合でも動くか，よく考えてください
- コンピュータに無駄な処理はさせない
  - 例えば，なるべく else if を使う / Scannerは1回だけnewする
- （実行例以外の場合でも）論理的に正しく動くようにする
  - 特定の入力例にしか使えないなら，プログラムを開発する意味がない
- 似たようなことは何度も書かない
  - コピペで解決せずに，なるべくループやメソッドを使って処理をまとめる
- プログラムは「人間が読むもの」でもある
  - 字下げや空白を使って見た目を整える / 意味のある変数名をつける

5.1 HAS-A関係

- クラス型のフィールド（p.342）
  - クラス（のインスタンス）が，クラス型のフィールドを持つ
  - 英語にすると，「Robot has a Computer」と表現できる
  - 「.」演算子の連鎖でフィールドが持つメンバを参照できる
  - 例：robot1.cpu.model = "M1";
- クラス図
  - UMLでは「集約」といい，ひし形を使ってクラスをつなぐ



5.2 メソッドの呼び出しとクラス型

- クラス型の引数（p.345）
  - メソッドにクラスのインスタンスを渡す
  - 例：robot1.setCPU(cpu)
- クラス型の戻り値（p.345）
  - メソッドがクラスのインスタンスを返す
  - 例：Computer cpu1 = robot1.getCPU()
- 「値渡し」と「参照渡し」の違い
  - Javaでは，引数が基本型（intやdouble）の場合，中身の値がコピーされてメソッドに渡される（値渡し）
  - 引数が参照型（クラスや配列）の場合，コピーはされず，同じ実体への参照（アドレス）だけが渡される（参照渡し）

5.3 クラス同士の連携の例

```
public class Computer {
  String model;
}
```

```
public class Robot {
  String name;
  // クラス型のフィールド
  Computer cpu;

  // クラス型のメソッド
  void setCPU(Computer cpu) {
    this.cpu = cpu;
  }

  // クラス型の戻り値
  Computer getCPU() {
    return this.cpu;
  }
}
```

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

    Robot robot1 = new Robot();
    robot1.name = "ロボタ";
    robot1.cpu = new Computer();
    robot1.cpu.model = "Z80";
    System.out.println(robot1.name +
        " (" + robot1.cpu.model + ")");

    Computer cpu1 = new Computer();
    cpu1.model = "M1";
    robot1.setCPU(cpu1);
    System.out.println(robot1.cpu.model);

    Computer cpu2 = robot1.getCPU();
    System.out.println(cpu2.model);
  }
}
```

後で，RobotとComputerにコンストラクタを追加してmainで利用してみよう

5.4 コンストラクタ

- コンストラクタ（p.349～）
  - インスタンスの生成時に自動的に実行されるメソッド
  - 初期化引数を受け取って，インスタンスを初期化する

```
Cat tama = new Cat();
tama.name = "タマ";
tama.age = 5;
```

初期化をシンプルに書けるようにしたい

```
Cat tama = new Cat("タマ", 5);
```

```
public class Cat {
    String name;
    int age;

    Cat(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

thisの有無による違いを理解せよ！

コンストラクタの定義例

- コンストラクタの定義方法
  - メソッド名をクラス名にする
  - 戻り値はない（voidでもない）

演習

- 5.3
  - Computerクラスに下記のコンストラクタを追加せよ
    - Computer(String model)
  - Robotクラスに下記のコンストラクタを追加せよ
    - Robot(String name)
  - Mainは上記2つのコンストラクタを使うように書き換えよ
  - 最終的なソースコードと実行結果を提出

5.5 コンストラクタの多重定義

- オーバーロード（多重定義）（p.357）
  - コンストラクタも，互いに引数が異なるものを定義できる
- 複数のコンストラクタで処理を共有（p.361）
  - コンストラクタは，内部で他のコンストラクタを呼び出せる
  - 「this(引数)」という形式で，他のコンストラクタを呼び出す
    - × this.コンストラクタ名(引数)
    - ○ this(引数)
  - これはコンストラクタの処理の最初に1回しか書けない
- デフォルトコンストラクタ（p.359）
  - クラスに1つもコンストラクタを定義しなかった場合には，引数のないコンストラクタが自動的に定義される

5.6 コンストラクタの多重定義の例

- コンストラクタの多重定義と「this(引数)」
  - 共通する処理は，なるべくまとめたほうがいよい

```
public class Cat {
  String name;
  int age;

  Cat(String name, int age) {
    this.name = name;
    this.age = age;
  }

  Cat() {
    this.name = "まだ無い";
    this.age = 0;
  }
}
```

```
public class Cat {
  String name;
  int age;

  Cat(String name, int age) {
    this.name = name;
    this.age = age;
  }

  Cat() {
    this("まだ無い", 0);
  }
}
```

第5回 まとめ

- has-a関係
  - クラス型のフィールド
  - クラス図（集約）
- メソッドとクラス型
  - クラス型の引数
  - クラス型の戻り値
- コンストラクタ
  - コンストラクタの定義方法
  - コンストラクタのオーバーロード
  - this(引数)
  - デフォルトコンストラクタ