Programming II

第10回 多態性(第12章)

塩澤秀和

9.1 継承

- オブジェクト指向の継承（p.370）
  - 「親」になるクラスのすべての機能（フィールドとメソッド）を引き継いだ「子」のクラスを定義する
  - ⇒ 既存の部品プログラムから，新しい部品を開発する
- よい継承の原則（p.395）
  - is-a関係（「子クラス is a 親クラス」の関係）で作る
  - 例) 猫 is a 動物，トラック is a 自動車，1年生 is a 学生
- 継承関係の用語（p.376）
  - 親クラス ＝ スーパークラス ＝ 基本（基底）クラス
  - 子クラス ＝ サブクラス ＝ 派生クラス

プログラミング言語等によって多少用語が違う

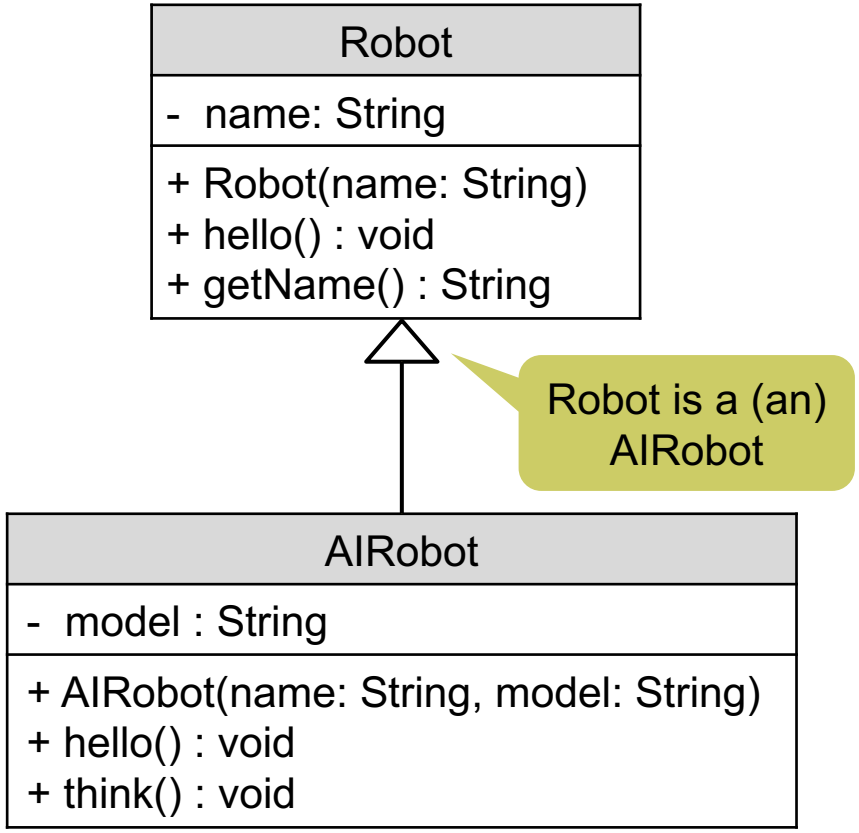
9.2 子クラスの定義方法

- Javaにおける継承の文法（p.374）
  - class 子クラス名 extends 親クラス名 { … }
  - 例）public class AIRobot extends Robot { … }
- フィールド/メソッドの追加
  - 子クラスでは，新しくフィールドやメソッドを追加できる
  - つまり，親クラスの機能を拡張（extend）することになる
- オーバーライド（p.377）
  - 親クラスのものと同じ名前のメソッドを子クラスに作ると…
  - 親クラスから継承したメソッドは覆い隠される
  - つまり，親クラスから継承した機能を修正（上書き）できる

フィールドもオーバーライドできるが有害なことが多い

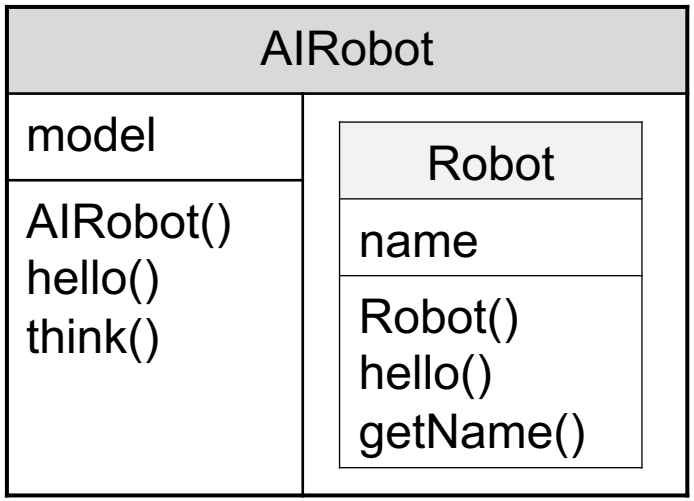
9.3/9.7 クラス図と内部構造

- 継承のクラス図



UMLのクラス図での書き方

- 継承の内部構造
  - 子クラスは親クラスの機能を持つ
  - ⇒ 内部に親クラス部分（super）を持つ



子クラスのインスタンスの内部構造

9.6 superの使用例

```
public class Robot {
  private String name;

  public Robot(String name) {
    this.name = name;
  }

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

  public String getName() {
    return this.name;
  }
}
```

```
public class AIRobot extends Robot {
  private String model;

  public AIRobot(String name,
                 String model) {
    super(name);
    this.model = model;
  }

  public void hello() {
    super.hello();
    System.out.println("（モデルハ "
      + this.model + "デス）");
  }

  public void think() {
    System.out.println(this.getName()
      + "ハ カンガエテイマス");
  }
}
```

適切なメインプログラムを
作成して動作を確認せよ

10.1 親子クラスの互換性

- 親子クラスの関係
  - 子クラスは親クラスの一種（is-a関係）であり，親クラスのすべての機能を持っている
  - 例）AIRobotは，Robotとしての機能を全て持っている
  - よって…

教科書の表現では，「ザックリあいまいにとらえる」

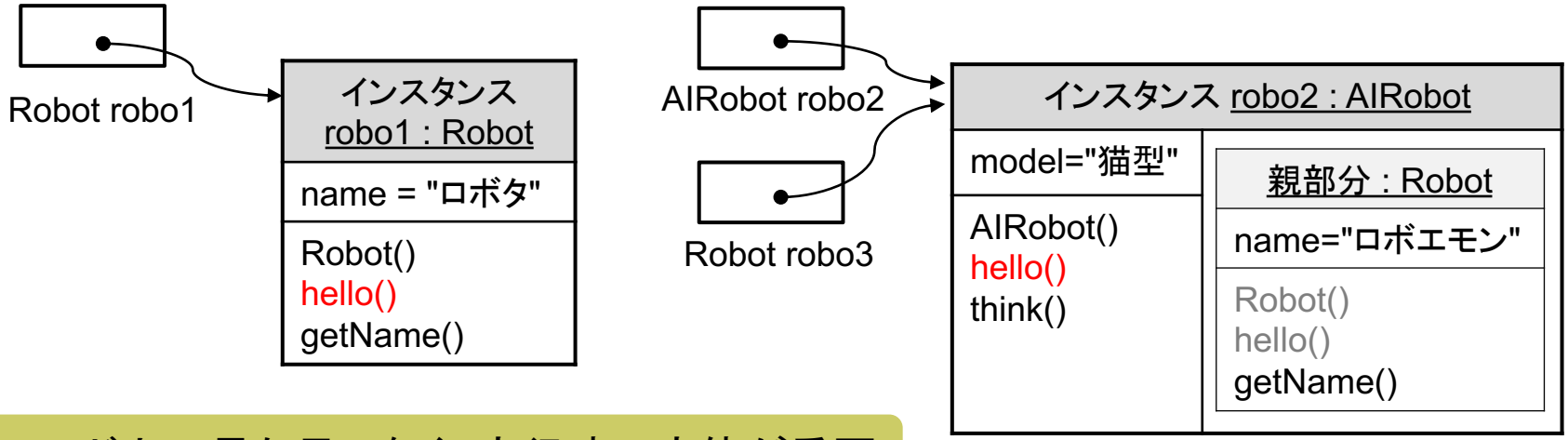
- 親子クラスの互換性（p.465，p.480，p.483）
  - 子クラスのインスタンスは，その親クラスのインスタンスが使える箇所なら，どこでも代わりに使える
  - 例）親クラス型の変数に，子クラスのインスタンスを代入
  - 例）親クラス型の配列に，子クラスのインスタンスを格納
  - 例）親クラス型の引数に，子クラスのインスタンスを渡す

10.2 多態性(ポリモーフィズム)

- 多態性(ポリモーフィズム)(p.475)
  - インスタンスの振る舞いは(親クラスとして参照されても), newで生成されたときの実体のクラスで決まる
  - オブジェクトの振る舞いは, そのオブジェクトが決める

```
Robot robo1 = new Robot(…);
robo1.hello();
```

```
AIRobot robo2 = new AIRobot(…);
Robot robo3 = robo2;
robo3.hello();  //← robo2.hello()と同じ
```



コード上の見た目でなく, 実行時の実体が重要

10.3 多態性の例(9.6の続き)

```
public class Main {
    public static void main(String [] args) {
        Robot robo1 = new Robot("ロボタ");
        robo1.hello();

        AIRobot robo2 = new AIRobot("ロボエモン", "猫型");
        robo2.think();

        Robot robo3 = robo2;  // AIRobotのインスタンスはRobotとしても通用する
        robo3.hello();        // 表面上はRobotだが，処理内容はAIRobotになる

        // robo3.think();     // 変数の型はRobotなので，これは文法違反になる

        Robot [] robots = new Robot[2];
        robots[0] = robo1;
        robots[1] = robo2;    // 子クラスのインスタンスを親クラスの配列に入れる
        for (Robot robo : robots) {
            robo.hello();     // ★：各インスタンスは自分で振る舞いを決める
        }
    }
}
```

10.4 多態性に関する文法上の制限

- メンバ参照の制限（p.471）
  - インスタンスを親クラス型の変数で参照している場合…
  - 親クラスに存在しないメンバの使用は，文法エラーになる
  - 例）Robot robomi = new AIRobot("ロボミ", "猫型");
    robomi.think(); // エラー！Robot型からは見えない

- ダウンキャスト（p.477）
  - インスタンスの扱いを親クラスから子クラスに戻す
  - キャスト演算子「(クラス名)」による明示的な変換が必要
  - 実体が子クラスのインスタンスでなければ，実行時エラー
  - 例）AIRobot airobo = (AIRobot) robomi;
    airobo.think(); // 文法上の問題が解消され，OK

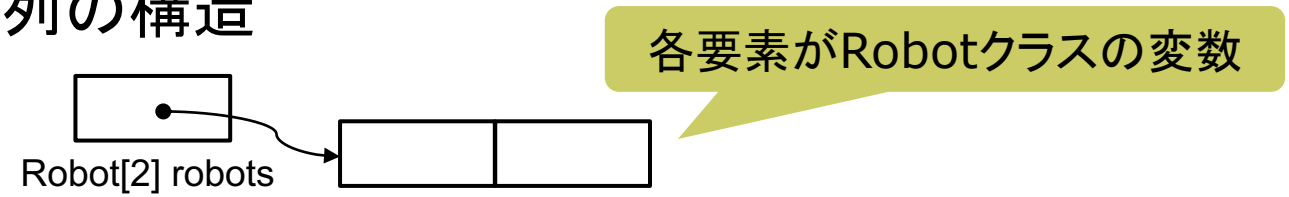
10.5 実行時の型判定

- ダウンキャストの危険性（p.478）
  - キャスト（強制型変換）の可否は，実行時にチェックされる
  - 失敗した場合，プログラムがエラーで強制終了してしまう
  - ダウンキャストの前には，実際に変換可能か確かめよう

- instanceof演算子（p.479）
  - x instanceof A : xが，クラスA（またはAの子孫クラス）のインスタンスならば真，そうでなければ偽となる
  - ⇒ 結果が真ならば，xをAにキャストしても大丈夫
  - 例）if (robomi instanceof AIRobot) {
    AIRobot airobot = (AIRobot) robomi;
    airobot.think();
    }

演習

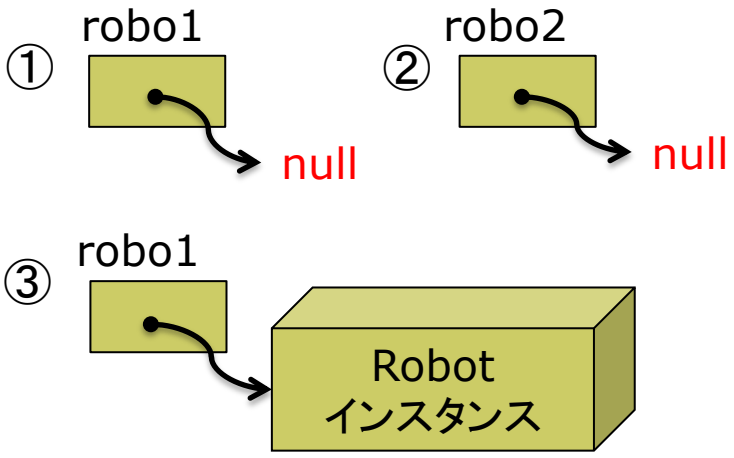
- 10.3
  - 入力して実行せよ(9.6も必要)
  - 4つの参照型変数robo1, robo2, robo3, robotsと，2つのnewで生成されたインスタンスと，1つのRobot型配列の構造の関係を次ページ(4.4)の図のように示してみよう
  - 参考：配列の構造



  - ★印の箇所に配列要素がAIRobotのインスタンスである場合だけ，think()メソッドを実行する処理を追加せよ
  - 最終的なソースコードと実行結果を提出

4.4 クラス型の変数とインスタンス

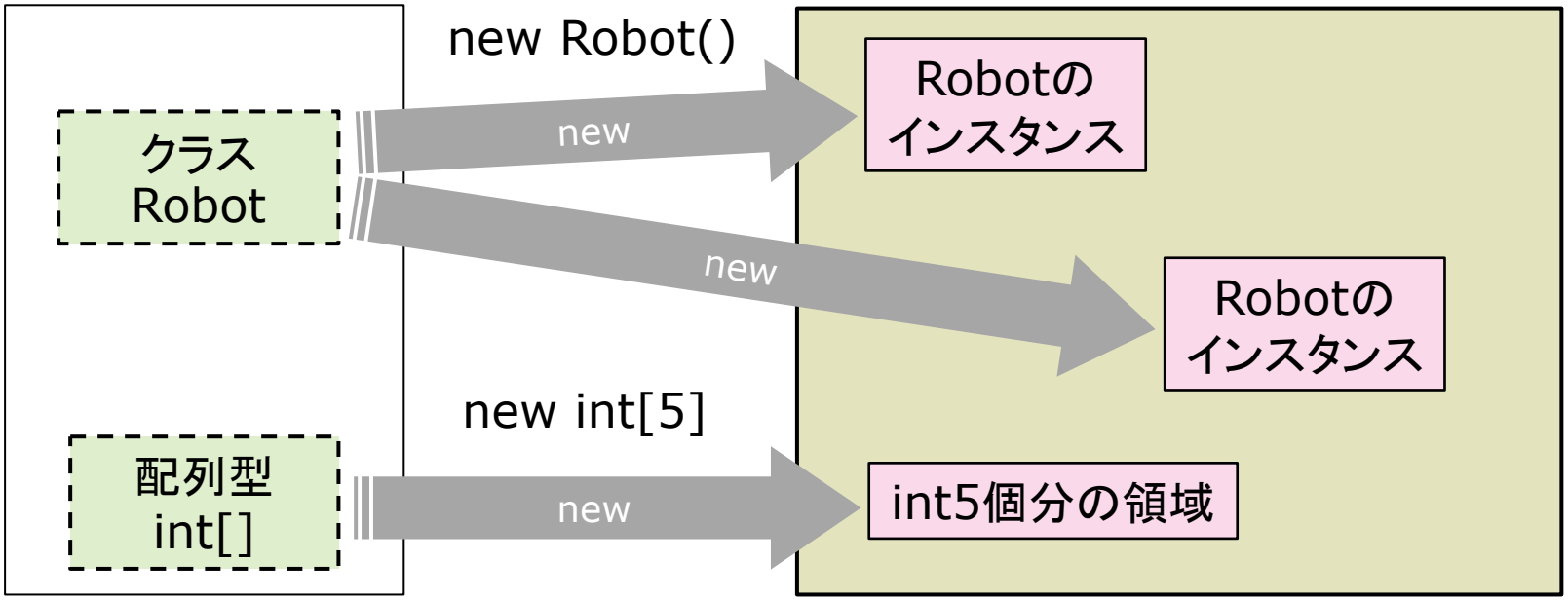
- クラスは参照型（p.334）
  - 実体（インスタンス）を代入するまで，変数は空（null）
  - newで作った実体を代入する
    - ① Robot robo1;
    - ② Robot robo2;
    - ③ robo1 = new Robot();
    - ④ robo2 = robo1;



- 復習：配列も同じ参照型
  - 以下を同様に図示せよ
    - ① int [] data1;
    - ② data1 = new int [5];
    - ③ int [] data2 = data1;

4.5 インスタンスの生成

- new演算子の働き（p.334）
  - コンピュータの中のメモリに領域を確保している
  - 1回のnewで1個のインスタンスができる 👈 これ重要！



第10回 まとめ

- is-a関係
- 多態性（ポリモーフィズム）
  - 親子クラスの互換性
  - 例）親クラス型の変数に，子クラスのインスタンスを代入
  - 例）親クラス型の配列に，子クラスのインスタンスを格納
  - 例）親クラス型の引数に，子クラスのインスタンスを渡す
  - インスタンスの振る舞いは，生成時のクラスで決まる
- ダウンキャスト
- instanceof演算子