オブジェクト指向技術 第9回
― オブジェクト指向プログラミング ―

立命館大学 情報理工学部
丸山 勝久

maru@cs.ritsumei.ac.jp

講義内容

- オブジェクト指向プログラミング
  - Java
  - クラス，メソッド，フィールド
  - パッケージとアクセス制御
  - インスタンスの生成とアクセス
  - 継承，抽象クラス，インタフェース
  - オーバーライド，オーバーロード
  - 配列とコレクション
  - 例外処理，入出力処理
  - GUIプログラミング
  - イベント駆動モデル

Java

- クラスベースオブジェクト指向プログラミング言語
- クラスが必ず必要(クラスを第一級として扱う)
- 特徴
  - C言語やC++に似た構文
  - 仮想マシン上で動作することで
    Write Once, Run Anywhere を実現
  - 静的な型付け(static-typed)
    - 基本型の厳密な規定
    - 基本型以外はすべて参照値(ポインタ)渡し
  - GC (garbage collection)を提供することでメモリ管理不要

- Java 20(最新版)と Java 17(長期サポート版)が存在
  - この講義ではJava 11をベースに説明

Java環境の整備

- JDK (Java Development Kit)のインストール
  - https://www.oracle.com/java/technologies/downloads/
  - 実行環境JRE (Java Runtime Environment)もインストールされる
- テキストエディタ + コンパイラ で開発・実行する場合
  - Windowsコマンドプロンプトでコンパイルするには、予め環境変数PATHに、コンパイラの実行ファイルのパスを設定する必要がある
  - インストール済みのJavaのバージョンを確認

```
% java –version
java version "11.0.18" 2023-01-17 LTS
...
```

JDK 17をインストールしてもよい

プログラムの構成要素

- 識別子
  - クラス，メソッド，変数などの名前
- 予約語
  - 識別子として使用不可
- 演算子
  - + - * / % = == < <= () [] && || . new 等
- 文と式
  - 選択 (if/else/switch)，繰り返し (while/for/do)，return 等
  - 条件，メソッド呼出し，フィールドアクセス 等
- コメントは3種類
  - ラインコメント：　　// から行末
  - ブロックコメント：　/* から */
  - 文書化コメント：　　/** から */

変数

- 値を格納する入れ物やインスタンスを指す名前
- 使用前に宣言が必要
  - 型を持つ
- スコープ
  - 単純名(名前のみ)で参照可能な範囲
  - インスタンス変数(フィールド)
    - クラス内部
  - メソッドの引数
    - メソッド内部
  - ローカル変数
    - 宣言された位置からそのブロックの終わりまで

データ型(data type)

- 基本型(primitive type)
  - 整数
    - byte (8ビット), short (16ビット),
      int (32ビット), long (64ビット)
  - 浮動小数点数
    - float (32ビット), double (64ビット)
  - 文字
    - char (符号なし16ビット)
  - 真偽(ブーリアン)型
    - boolean
- 参照型(reference type)
  - 基本型以外のクラス，利用者が作成したクラス
  - Object (すべての参照型は暗黙的に継承している)
  - String (文字列を格納するためのクラス)

リテラル(literal)

- 整数
  - 123 (int型)，123L (long型)
- 浮動小数点数
  - 1.23 (double型)，1.23f (float型)
- 文字
  - 'a', '漢' (char型)
- ブーリアン型
  - true, false (boolean型)
- 文字列
  - "abc" (String型)
  - new演算子を省略して生成可能
- null
  - 参照するオブジェクトが存在しないことを意味

```
int a = 123;
```

```
double b = 1.23;
```

```
char c = 'a';
```

```
boolean b = true;
```

```
String s = "abc";
```

=

```
s = new String("abc");
```

```
Object o = null;
```

変数の初期値

- ローカル変数の初期値 → 不定
  - 初期化せずに参照するとコンパイルエラー
- フィールドと配列要素の初期値
  - 基本型
    - 整数(byte, short, int, long) → 0 (それぞれの型の)
    - 浮動小数点数(float, double) → 0.0 (それぞれの型の)
    - 文字(char) → 空文字(¥u0000)
    - 真偽(boolean) → false
  - 参照型 (Stringも) → null
- 定数
  - 宣言にfinal(変更不可)を付ける

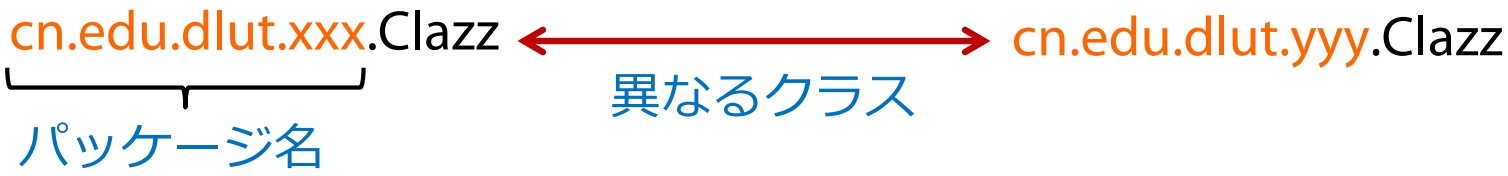
```
final int MAX_VALUE = 1024;
```

型エラー(type error)

- 代入文の左辺と右辺は同じデータ型でなければならない
  - × int x = 3.0;
  - × boolean b = 2;
- 異なるデータ型の値は演算できない
  - × 3 + true
- 例外
  - 整数は浮動小数点数に格上げされる
    - ○ double x = 3;
  - 基本型の値は文字列型と結合されると，文字列型に格上げされる
    - 明示的に変換する場合は，String.valueOf()を利用
    - ○ String s = “ABC” + 3;
    - ○ String s = “ABC” + String.valueOf(3);
  - 参照型の場合は，toString()メソッドが利用される

パッケージ

- クラスはパッケージごとにグループ化できる
  - 名前の衝突を回避
  - パッケージ名と同一名のディレクトリの中に，
    ソースファイルとクラスファイルを置く
- ソースファイルの先頭にパッケージ宣言を記述
  - パッケージを指定しない場合，
    パッケージなし(デフォルトパッケージ)とみなされる
- 完全修飾名(FQN: fully qualified name)
  - パッケージ名も含めたクラス名
    String → java.lang.String (FQN)
  - 慣用的に，インターネットのドメイン名を
    逆順に記述したパッケージ名を使用



Javaプログラムの実行手順

（１）ソースファイルの作成

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello Java");
    }
}
```

拡張子は
”.java”が一般的

Hello.java

（２）ソースファイルをコンパイルする

```
javac Hello.java
```

Hello.class が生成される

Hello.class

（３） Java実行環境(仮想計算機)で実行する

```
java Hello
```

Hello.class が実行される

Javaプログラムの実行(Hello.java)

Hello.class

Hello.classのプログラムを実行した場合
Hello.javaのmain()メソッドから実行が始まる

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello Java");
    }
}
```

文字列リテラルは
そのまま出力される

出力結果

```
Hello Java
```

コンソール画面に出力するためには,
System.out.print() (改行なし)や
System.out.println() (改行あり)を利用する

Javaプログラムの実行(Ex1.java)

```
public class Ex1 {
    public static void main(String[] args) {
        int a = 123;
        String s = "DUT";

        System.out.println(a);

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

出力結果

```
123
DUT
```

変数の値が出力

String型の場合は，
格納されている文字列が
そのまま出力

Javaプログラムの実行(Ex2.java)

```
public class Ex2 {
    public static void main(String[] args) {
        int x = 123, y = 56;
        String s = "abc", t = "def";

        System.out.println(x + y);

        System.out.println(s + t);

        System.out.println(s + x);

        System.out.println(s + x + y);
        System.out.println(x + y + s);
    }
}
```

出力結果

```
179
abcdef
abc123
abc12356
179abc
```

x+yの計算結果が出力

文字列同士の加算は結合
"abc" + "def" → "abcdef"

文字列に格上げされてから結合
"abc" + 123
→ "abc" + "123"
→ "abc123"

s+xが先に計算される

x+yが先に計算される

Javaプログラムの実行(Ex3.java)

```
public class Ex3 {
    public static void main(String[] args) {
        int result = sum(2);
        System.out.println(result);
        System.out.println(sum(5));
        System.out.println(sum(-1));
    }

    static int sum(int n) {
        if (n <= 0) { return 0; }
        int sum = 0;
        while (n > 0) {
            sum = sum + n;
            n--;
        }
        return sum;
    }
}
```

メソッド呼び出し式

条件文

繰り返し文

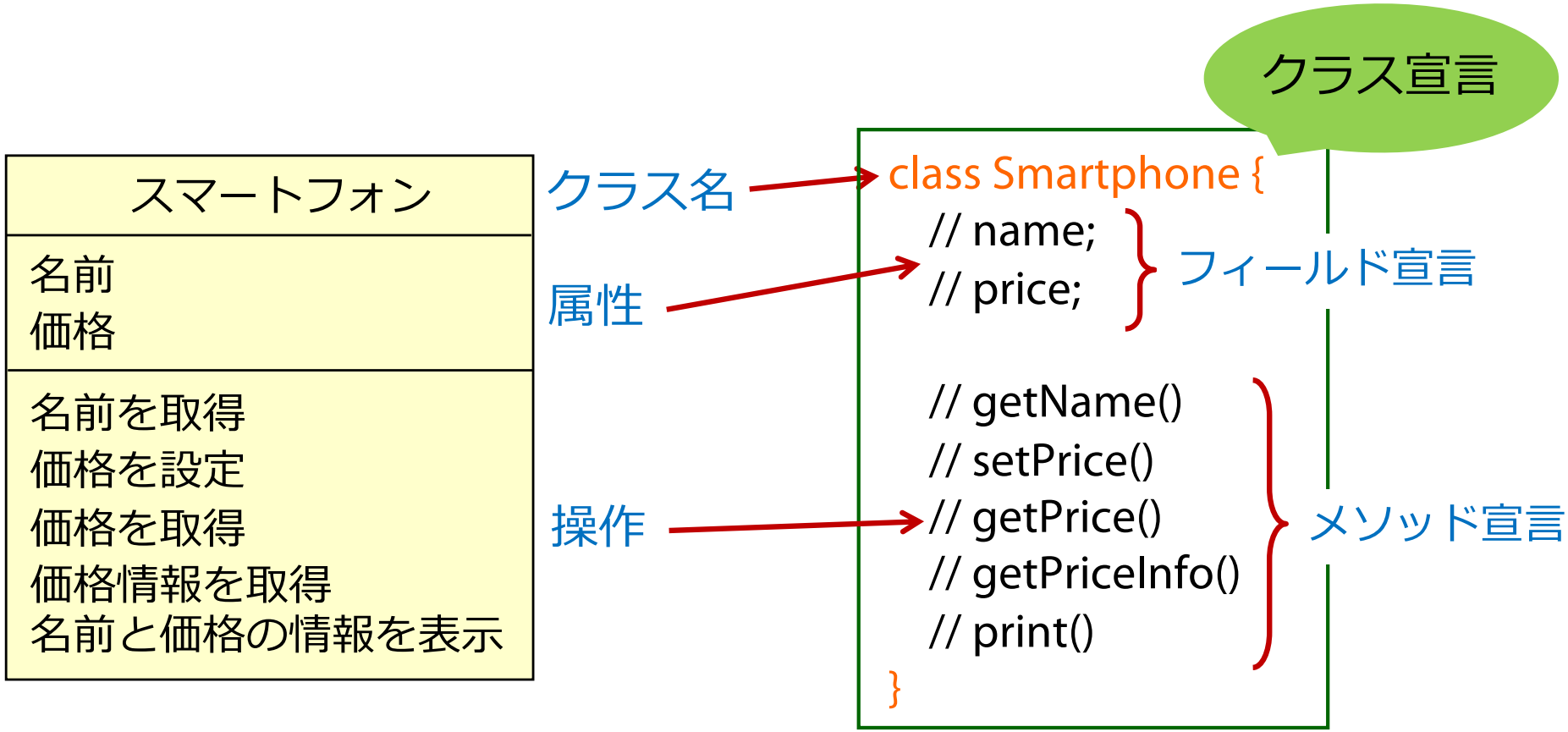
return文

出力結果

```
3
15
0
```

クラスの記述

- クラスの宣言
  - classキーワードを利用する
- フィールドとメソッドはクラス宣言の中に記述

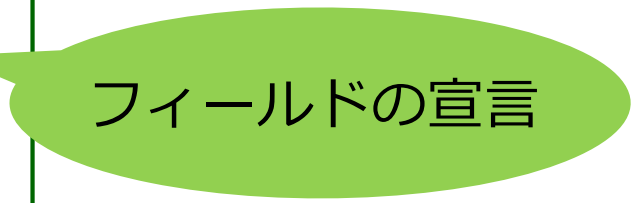


フィールドの記述

- フィールドの宣言
  - 型名の後ろにフィールド(変数)の名前を記述
- 初期値を付けることも可能

```
class Smartphone {
    String name;    // 名前を格納
    int price;      // 価格を格納

    // getName()
    // setPrice()
    // getPrice()
    // getPriceInfo()
    // print()
}
```



メソッドの記述

- メソッドの宣言(定義)
  - 戻り値の型名の後ろにメソッドの名前と引数のリストを記述

```
class Smartphone {
    String name;
    int price;
    String getName() { return name; }    // 名前を取得
    void setPrice(int p) { price = p; }  // 価格を設定
    int getPrice() { return price; }     // 価格を取得
    String getPriceInfo() {              // 価格情報を取得する
        return getPrice() + “-yen”;
    }
    void print() {   // 名前と価格の情報を表示
        System.out.println(name + ": " + getPriceInfo());
    }
}
```

メソッドの宣言

this と super

- 特定のオブジェクトやコンストラクタを指し示す
- this
  - 自クラスのインスタンスを指す
- this()
  - 自クラスのコンストラクタを呼び出す
  - コンストラクタ先頭でのみ記述可能
- super ※次回説明
  - 親クラスのメンバを参照する際に利用
- super() ※次回説明
  - 親クラスのコンストラクタを呼び出す
  - コンストラクタ先頭でのみ記述可能

```
class Clazz {
   int x;
   Clazz(int x) {
     this.x = x;
   }
   Clazz() {
      this(1);
   }
   void m() { … }
   void n() {
     this.m();
   }
}
```

コンストラクタの記述

- コンストラクタの宣言(定義)
  - インスタンスの生成時に呼び出されるメソッド
  - クラスと同じ名前で宣言
    - コンストラクタの宣言を省略すると
      引数なしのコンストラクタが自動的に作成される


```
class Smartphone {
    Smartphone(String name, int p) {
        this.name = name;    // 引数nameの値をフィールドnameに代入
        price = p;           // 引数pの値をフィールドnameに代入
    }
    Smartphone(String name) {
        this(name, 30000);   // 自クラスのコンストラクタを呼び出す
    }
    … (省略)
}
```

import文

- 他のパッケージ内のクラスを利用する場合

  import文を利用

  - import + 利用するクラス名(パッケージ名含む)

    ```
    import java.util.ArrayList;
    ```

  - java.langパッケージ内の型名のimportは省略可能

    System, String, Object 等

  - import文を記述しなくても，コード中に完全修飾名を記述すれば利用可能
    - コードが見にくくなるので基本的に避ける
  - ワイルドカード文字(*)を使えば、特定のパッケージ内のクラスを明示的なimportなしで利用可能

    ```
    import java.util.*;
    ```

クラスの宣言と可視性

- 1つのソースファイル内では,
  トップレベルのクラス1つしか公開できない
  - 公開するクラスには通常publicを付ける

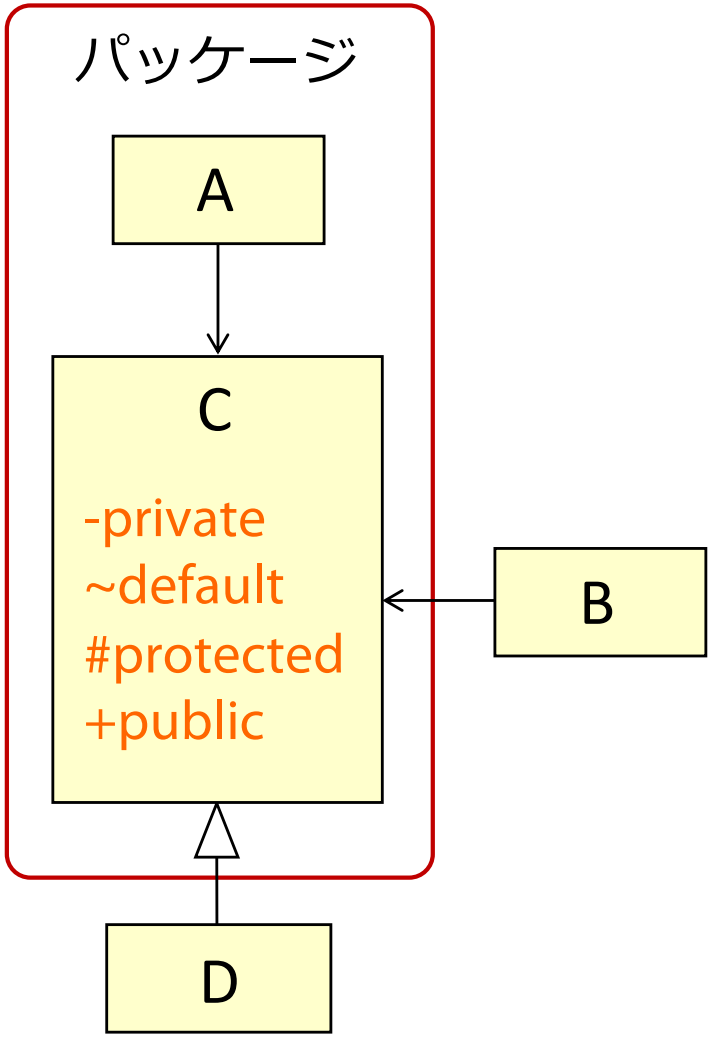
- ファイル名とトップレベルのクラス名は一致
  - publicでないクラスを記述することも可能
    - publicでない場合でも同一パッケージからは利用可能
  - クラス内部に他のクラスを宣言可能 (内部クラス)

```
public class Student {
…
}
class Clazz { … }
```

ファイル名
Student.java

クラスメンバの可視性

- private
  - 外部から参照不可 (C)
- default (何も記述しない)
  - 同一パッケージから参照可(A, C)
- protected
  - 同一パッケージとサブクラス から参照可 (A, C, D)
- public
  - 外部から参照可 (A, B, C, D)



可視性の設定

- 標準的なポリシー
  - フィールドは基本的にすべてprivate
  - getter, setterを用意する
    - getter: フィールドの値を取得するためのメソッド
      (通常は get+フィールドの名前 という名前)
    - setter: フィールドに値を設定するためのメソッド
      (通常は set+フィールドの名前 という名前)
  - コンストラクタで初期値を設定する
  - できるだけsetterを作らない
    - 外部から値を変えられないフィールドの方が管理が楽

クラスの完成(可視性の追加)

```
public class Smartphone {
   private String name;
   int price;

   public Smartphone(String name, int p) {
      this.name = name;
      price = p;
   }
   public Smartphone(String name) {
      this(name, 30000);
   }
   public String getName() {
      return name;
   }
   public void setPrice(int p) {
      price = p;
   }
   public int getPrice() {
      return price;
   }
   private String getPriceInfo() {
      return getPrice() + "-yen";
   }
   public void print() {
      System.out.println(
        name + ": " + getPriceInfo());
   }
}
```

インスタンスの生成

- new演算子を利用
  - コンストラクタの呼出し

```
public class Smartphone {
   private String name;
   int price;

   public Smartphone(String name, int p) {
      this.name = name;
      price = p;
   }
   public Smartphone(String name) {
      this(name, 30000);
   }
   …
}
```

インスタンスへの参照を
格納している変数の名前

```
Smartphone phone1 =
new Smartphone("ABC", 35000);
```

name: “ABC”
price: 35000

```
Smartphone phone2 =
new Smartphone("XYZ");
```

name: “XYZ”
price: 30000

インスタンスへのアクセス

- ドット(.)演算子を利用
  - インスタンスの名前.フィールドの名前あるはメソッドの名前

```
public class Smartphone {
    private String name;
    int price;

    public int getPrice() {
        return price;
    }
    …
}
```

```
Smartphone phone1 =
new Smartphone("ABC", 35000);

System.out.println(phone1.price);

System.out.println(phone1.getPrice());
```

name: “ABC”
price: 35000

静的メンバ

- staticキーワードを付けて宣言

```
static int x = 10;
static int abs(int a) { … }
```

- インスタンスではなく，クラス名を使ってアクセス
  - クラス変数
    - そのクラスのすべてのインタンスで共有される
  - クラスメソッド
    - インスタンスなしで呼び出せる

```
Math.abs(-10);
```

```
public class Hello {
   public static void main(String[] args) {
      System.out.println("Hello Java");
   }
}
```

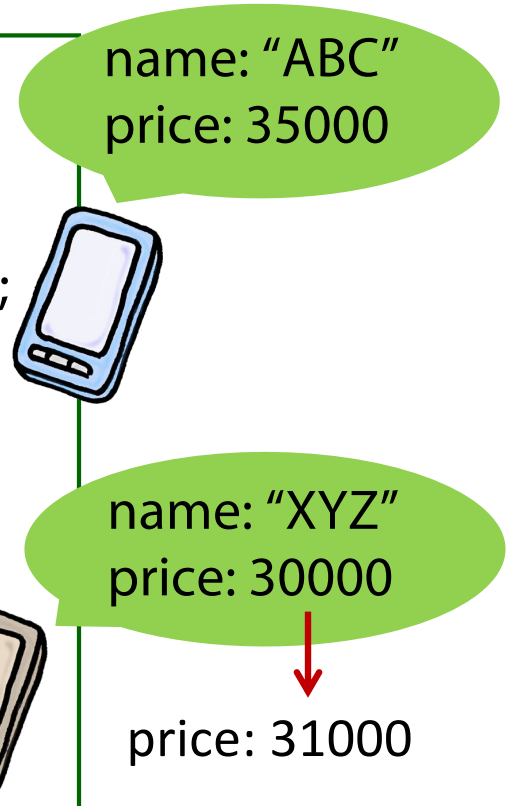
main()はクラスメソッド
Hello.classの開始時点は
必ず1つなので

クラスの利用

```
public class Main1 {

   public static void main(String[] args) {
      Smartphone phone1 = new Smartphone(“ABC”, 35000);
      System.out.println("Price = " + phone1.price);
      System.out.println("Price = " + phone1.getPrice());

      Smartphone phone2 = new Smartphone("XYZ");
      System.out.println("Price = " + phone2.price);
      phone2.setPrice(phone2.price + 1000);
      System.out.println("Price = " + phone2.getPrice());
   }
}
```



実行結果

```
Price = 35000
Price = 35000
Price = 30000
Price = 31000
```

代入

- 基本型
  - 代入により値が複製

```
int a, b;
a = 1;
b = a;
a = 2;
```

a: 1
b: 1

a: 2
b: 1

- 参照型
  - 代入によりインスタンスへの参照値が複製される ＝ 別名(alias)

```
Smartphone phone = new Smartphone("A", 30000);
Smartphone alias = phone;
phone.price = 20000;
```

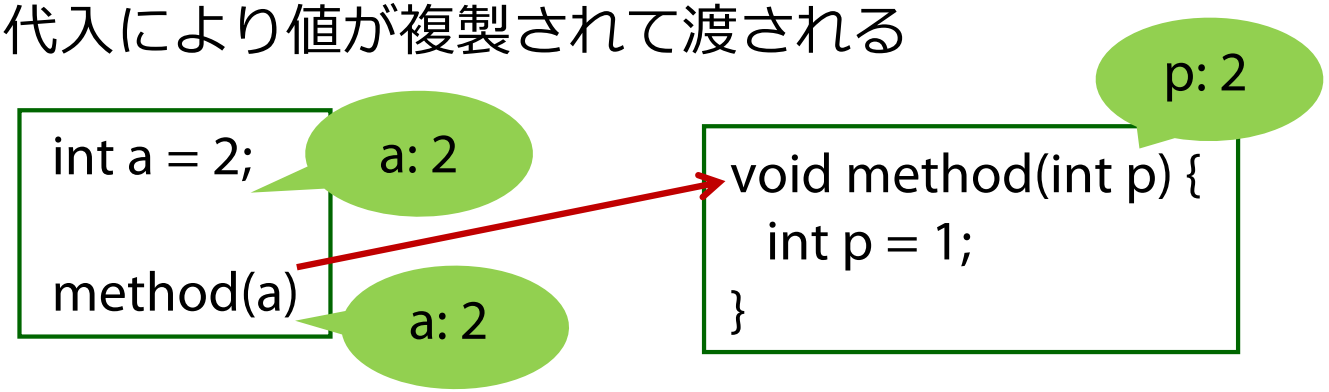
phone.price: 20000
alias.price: 20000

```
Smartphone phone = new Smartphone("A", 30000);
Smartphone alias = phone;
alias.price = 20000;
```

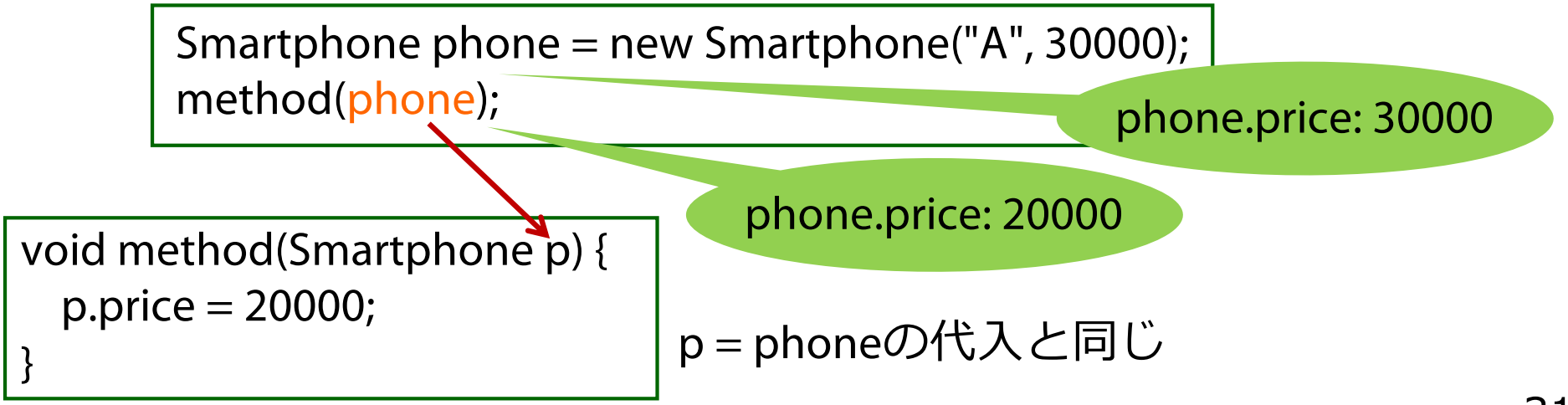
phone.price: 20000
alias.price: 20000

引数渡し

- 基本型の引数渡し
  - 代入により値が複製されて渡される



- 参照型の引数渡し
  - 引数にインスタンスの参照値の複製が渡される(参照渡しではない)



練習問題(Exec1)

```
class Device {
    private int price;
    Device(int price) {
        this.price = price;
    }
    Device() {
        this(10);
    }
    void setPrice(int p) {
        price = p;
    }
    int getPrice() {
        return price;
    }
    String getInfo() {
        return "$" + price;
    }
}
```

```
public class Exec1 {
    public static void main(String[] args) {
        Device device1 = new Device(25);
        System.out.println(device1.getPrice());  // (1)

        Device device2 = new Device();
        System.out.println(device2.getInfo());   // (2)
        device2.setPrice(15);
        System.out.println(device2.getInfo());   // (3)

        Device device3 = device1;
        System.out.println(device3.getInfo());   // (4)
        device3.setPrice(40);
        System.out.println(device1.getInfo());   // (5)
        System.out.println(device3.getInfo());   // (6)

        device3 = device2;
        System.out.println(device3.getInfo());   // (7)
    }
}
```

```
25
$10
$15
$25
$40
$40
$15
```

まとめ

- クラスを必ず記述する
- フィールドとメソッドはクラス宣言の中に記述する
- パッケージを使うと，同じ名前のクラスを区別できる
- thisは自インスタンスを指す
- インスタンスを生成する際には，new演算子を使う
- インスタンスにアクセスする際には，ドット演算子を使う
- クラス変数は，そのクラスのすべてのインタンスで共有される
- クラスメソッドはインスタンスなしで呼び出せる
- 参照型の代入や引数渡しでは，インスタンスの参照値が複製される

次回の講義の最初に小テストを行います