プログラミングIII

元木クラス
第4週 (5/8, 9)
メソッドのオーバーライド,
オブジェクト指向①

Overload of Method

メソッドのオーバーロード

オーバーロード

- オーバーロード (overload)とは
  同じ名前のメソッドを複数宣言すること
  （ただし、引数は異なる必要がある）
  - 日本語では「多重定義」

- 同じ名前でも大丈夫？
  呼び出し時に指定される引数のタイプによって実行されるメソッドまたはコンストラクタが区別される

メソッドのオーバーロードの例

```
class Example {
    public static void methodA() {
        System.out.println("引数はありません");
    }
    public static void methodA(int i) {
        System.out.println("int型の値" + i + "を受け取りました");
    }
    public static void methodA(double d) {
        System.out.println("double型の値" + d + "を受け取りました");
    }
    public static void methodA(String s) {
        System.out.println("文字列" + s + "を受け取りました");
    }
    public static void main(String[] args) {
        methodA();
        methodA(1);
        methodA(0.1);
        methodA( “Hello” );
    }
}
```

オーバーロードができない場合

- 変数の名前が異なるだけではオーバーロードできない

```
public static void methodA(int i) {略}
Public static void methodA(int j) {略}
```

- 戻り値の型が異なるだけではオーバーロードできない

```
public static void methodA(int i) {略}
Public static int methodA(int i) {略}
```

シグネチャ

- 「メソッド名」「引数の型」「引数の数」の3つの要素をシグネチャと呼ぶ。

- シグネチャが同じメソッドを宣言することはできない。

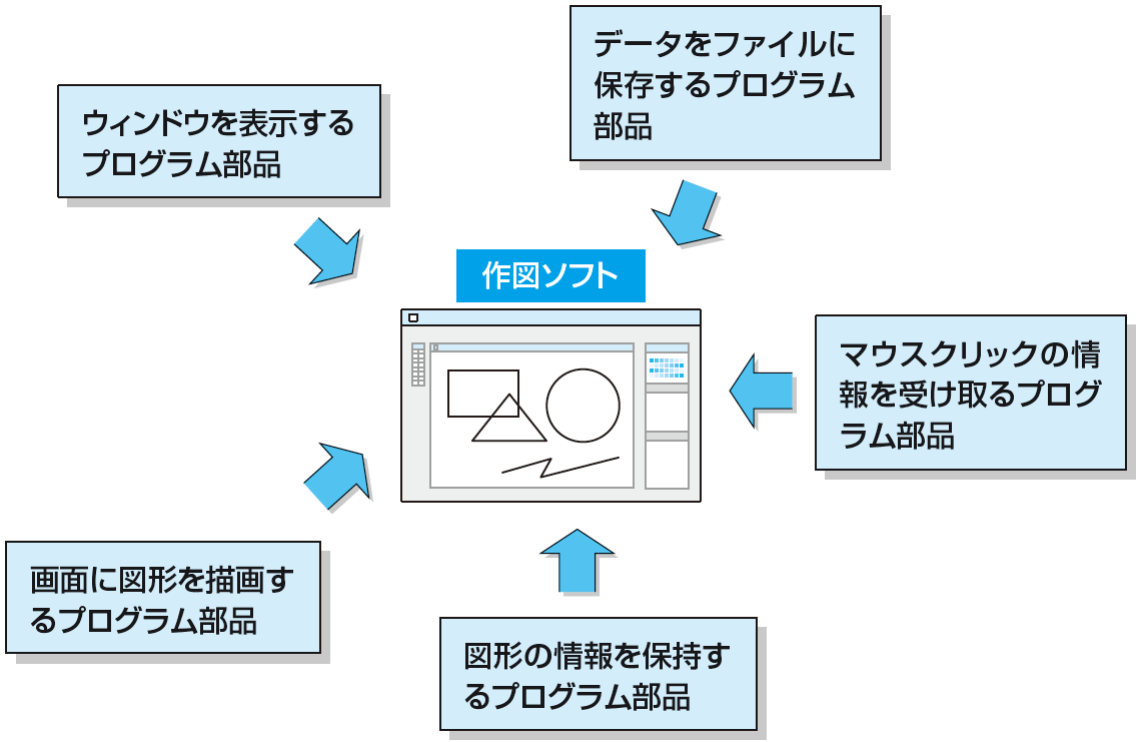
クラスの基本

オブジェクト指向とは

- プログラム部品を組み合わせることでプログラム全体を作成する
- プログラムを自動車に例えると・・
  - 自動車は様々な部品から構成される
    車体・エンジン・タイヤ・ヘッドライト
  - 最終製品は部品の組み合わせ
  - それぞれの部品の内部構造を知らなくても、組み合わせ方（使い方）がわかればよい
  - 部品単位でアップデートできる

オブジェクト指向とクラス

- プログラムの部品＝オブジェクト と考える
- オブジェクトがどのようなものか記述したものが「クラス（class）」
- Javaによるプログラミン＝classを定義すること



複雑なプログラムは多くの
プログラム部品から構成される

クラスとインスタンス

- クラス
  - オブジェクトに共通する属性（情報・機能）を抽象化したもの
- インスタンス
  - 具体的な個々のオブジェクト

クラス	インスタンス
自動車	私の愛車、そこにある車
犬	私の飼っているポチ、隣の家のクロ
人	私、私の父、私の母
2次元平面上の点	(x=2,y=5)の点、(x=-1,y=3)の点
円	中心(0,0)半径1の円、中心(3,4)半径5の円

簡単なクラスの宣言とインスタンスの生成

- 例として学籍番号(id)と氏名(name)を持つ学生証を扱うためのクラスは、次のように宣言する

```
class StudentCard {
   int id;
   String name;
}
```

フィールド（StudenCardクラスがもつ情報）

  - クラスの名前は自由に決められる。今回はStudentCardとした
  - idとnameという名前のint型とString型の変数をクラスの中に定義した．このようなクラスをフィールドと呼ぶ
  - idとnameという2つの値をセットにして扱える

C言語での類似の機能：構造体

- Javaのクラス

```
class StudentCard {
  int id;
  String name;
}
```

- C言語の構造体

```
typedef strut {
  int id;
  char* name;
} StudentCard;
```

- フィールドだけであれば，Javaのクラスと C言語の構造体は類似の機能を持つ
  - 様々な種類のデータをまとめて扱える
- 違い：
  - Javaのクラスは，そのクラスに対する処理も記述できる（メソッド）
  - Javaのクラスは，データを持たず，メソッドだけで構成される場合もある

演習

日常を見まわして、クラスの定義をしてみましょう。日本語を使ってかまいません。

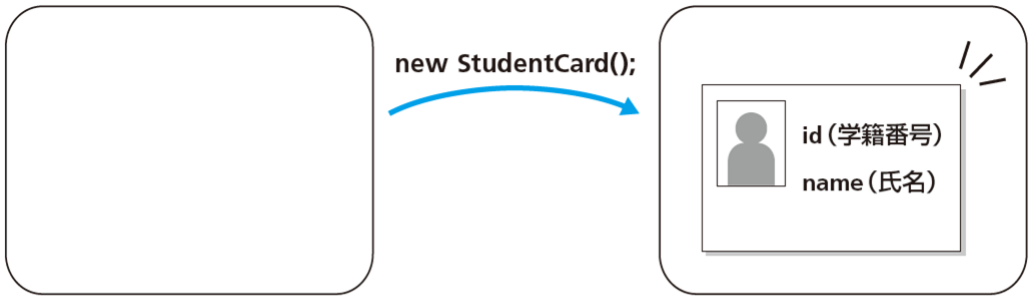
例：

```
class 本 {
  タイトル
  著者
  出版社
}

class 賃貸ルーム {
  広さ
  家賃
  住所
}
```

インスタンスの生成

- インスタンスを生成するにはnewを使用する。

```
new StudentCard();
```



- 生成したインスタンスを変数aに代入できる

```
StudentCard a = new StudentCard();
```

変数の型はクラス名

- インスタンスが持つ変数に値を代入できる

```
StudentCard a = new StudentCard();
a.id = 1234;
a.name = "鈴木太郎";
```

インスタンス変数へのアクセス

- インスタンスが持つ変数（クラスのフィールドに定義された変数）のことを「インスタンス変数」と呼ぶ
  - StudentCardクラスでは変数idとnameがインスタンス変数
- インスタンス変数にアクセスするには
  「インスタンスを代入した変数＋ドット＋インスタンス変数名」

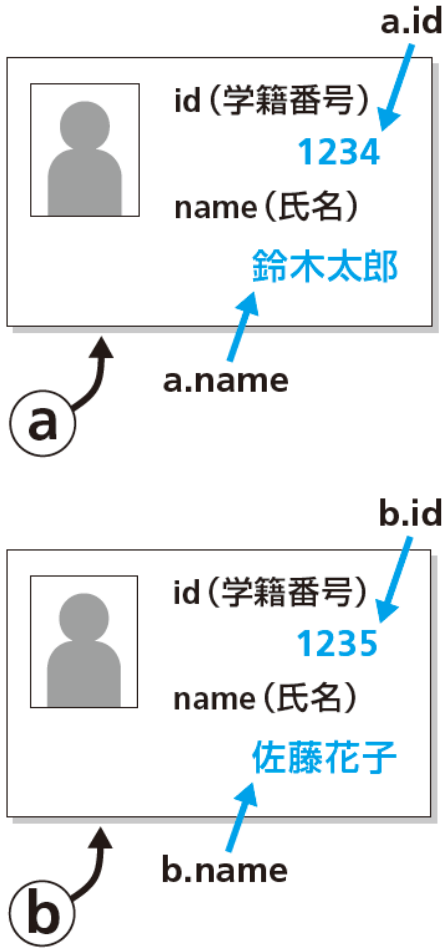
例：

```
StudentCard a = new StudentCard();
a.id = 10;
System.out.println("aのidは" + a.id);
```

  - （ドットを「の」に置き換えて「aのx」と読むとわかりやすい）

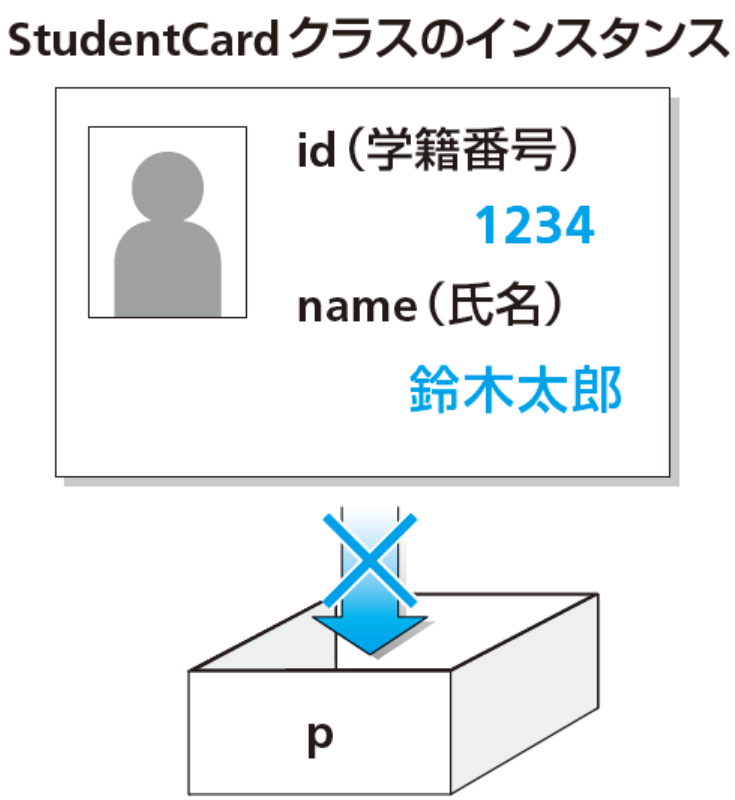
StudentCardクラスの使用例

```
class StudentCard {
  int id; // 学籍番号
  String name; // 氏名
}
public class Example {
  public static void main(String[] args) {
    StudentCard a = new StudentCard();
    a.id = 1234;
    a.name = "鈴木太郎";

    StudentCard b = new SZtudentCard();
    b.id = 1235;
    b.name = "佐藤花子";

    System.out.println("aのidは" + b.id);
    System.out.println("aのnameは" + b.name);
    System.out.println("bのidは" + b.id);
    System.out.println("bのnameは" + b.name);
  }
}
```



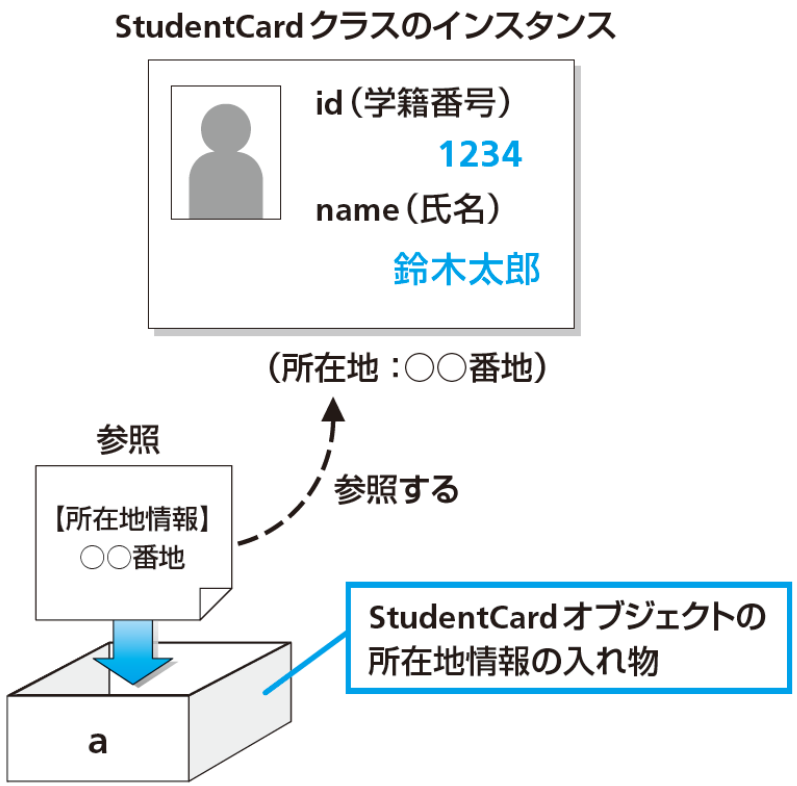
参照型

- Javaで使用できる変数の型
  - 基本型（int, double, boolean など）
  - 参照型（インスタンスへの参照）
- 変数にインスタンスそのものは代入されない



参照の代入

- StudentCard a = new StudentCard();
  としたとき、変数aには、StudentCardクラスのインスタンスの参照が代入される



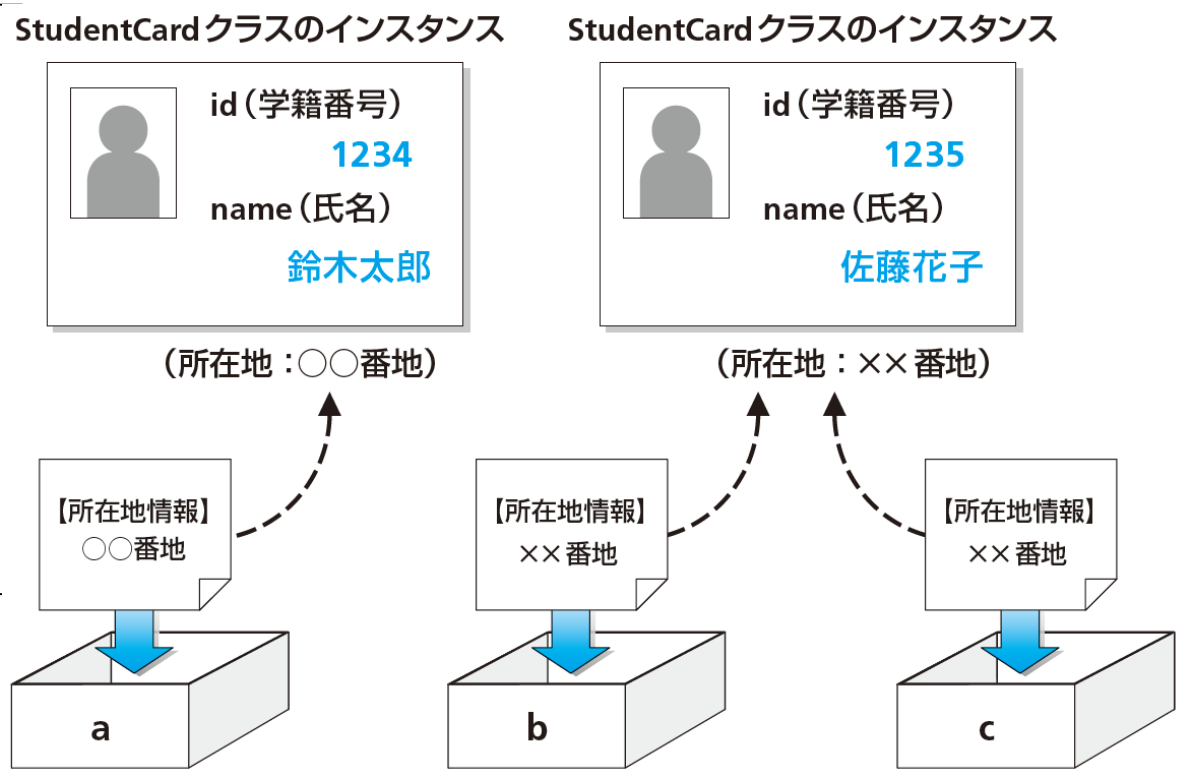
演習

- 右のプログラムコードの意味を考えてみよう

```
class Dog {
    String name;
}

public class Example {
    public static void main(String[] args) {
        Dog dog1 = new Dog();
        dog1.name = "Taro";
        Dog dog2 = new Dog();
        dog2.name = "Pochi";
        Dog dog3 = dog2;
        System.out.println(dog3.name);
        dog3.name = "Jiro";
        System.out.println(dog2.name);
    }
}
```

参照の例

```
StudentCard a = new StudentCard();
StudentCard b = new StudentCard();
StudentCard c = b;
a.id = 1234;
a.name = “鈴木太郎”;
b.id = 1235;
b.name = “佐藤花子”;
```

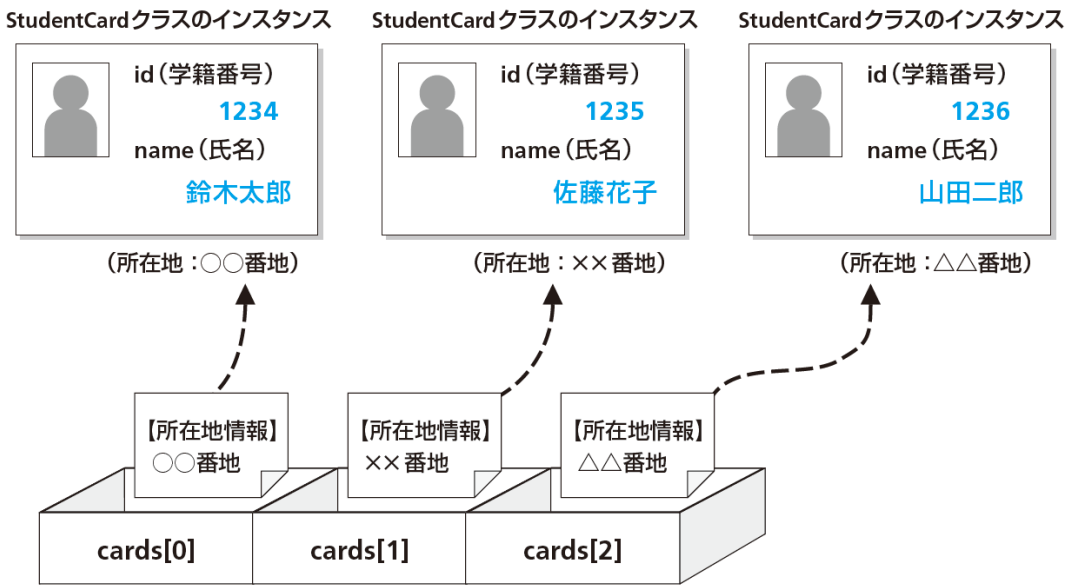


参照の配列

- 基本型の配列と同じように、参照の配列も作成できる。

```
StudentCard[] cards = new StudentCard[3];
cards[0] = new StudentCard();
cards[1] = new StudentCard();
cards[2] = new StudentCard();
cards[0].id = 1234;
cards[0].name = "鈴木太郎";
cards[1].id = 1235;
cards[1].name = "佐藤花子";
cards[2].id = 1236;
cards[2].name = "山田二郎";
```

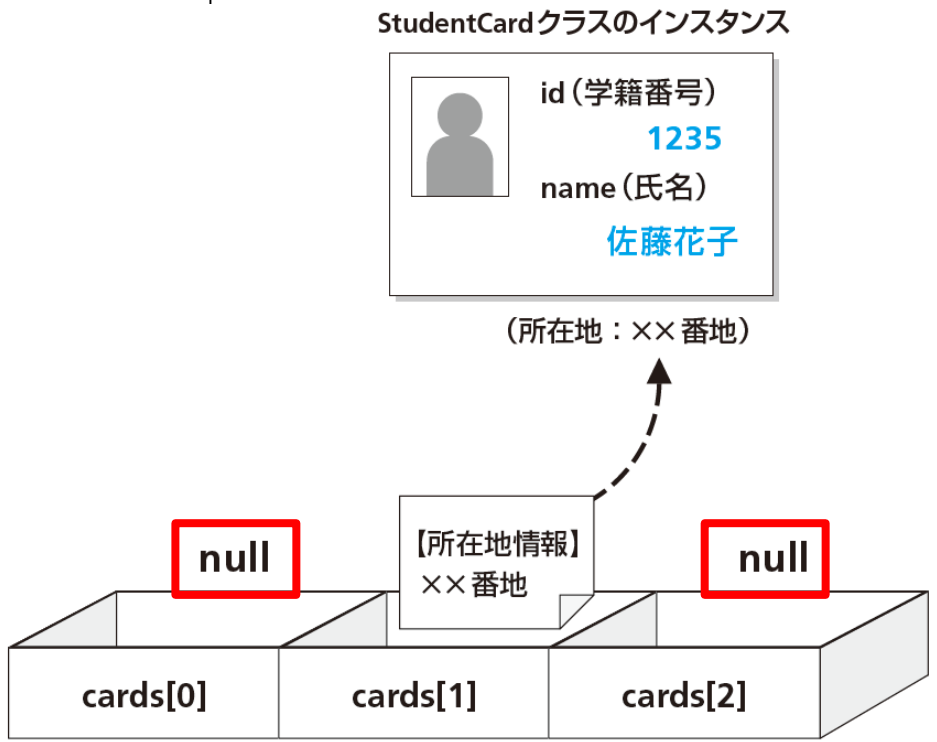
配列を生成



何も参照しないことを表すnull

- 参照型の変数に、何も参照が入っていない状態をnullという

```
StudentCard[] cards = new StudentCard[3];
cards[1] = new StudentCard();
cards[1].id = 1235;
Cards[1].name = ”佐藤花子”
```



nullは参照型の値

- nullは、参照型の変数に代入できる

```
StudentCard a;
a = null;
```

- nullは参照型の変数の値と比較できる

```
StudentCard a = new StudentCard();
if(a == null) {
  System.out.println( “aはnull”);
} else {
  System.out.println( “aはnullでない”);
}
```

参照とメソッド

- メソッドには引数としてインスタンスの参照を受け渡しできる

```
static void clearCardInfo(StudentCard card){
  card.id = 0;
  card.name = “未定”;
}
```

- メソッドの戻り値にすることもできる

```
static StudentCard compareCard(StudentCard card0, StudentCard card1){
  if (card0.id < card1.id) {
    return card0;
  } else {
    return card1;
  }
}
```

クラスの定義とファイル

- 複数のクラスの宣言を1つのファイルの中に記述する以外に、ファイルを分けて記述することもできる。

StudentCard.java

```
public class StudentCard {
  (中略)
}
```

Example.java

```
public class Example {
  public void main (String args) {
    //StudentCardクラスを使った処理を行う
    (中略)
  }
}
```

ワン・モア・ステップ(インスタンス変数の初期値)

- インスタンス変数は、インスタンスが生成されるときに自動的に初期化される

```
Class DataSet {
   int i;              0で初期化される
   double d;           0.0で初期化される
   boolean b;          falseで初期化される
   String s;           nullで初期化される
   Dataset data;       nullで初期化される
}
```

今回の復習と次回に向けた予習範囲

復習と予習

復習

- 教科書の以下の範囲を見直し，サンプルコードを試す
  - 5-1 オブジェクト指向 pp.131-134
  - 5-2 クラスとインスタンス pp.135-139
  - 5-3 参照 pp.140-153

予習

- 教科書の以下の範囲に目を通す
  - 6-1 コンストラクタ pp.163-168
  - 6-2 インスタンスメソッド pp.169-173
  - 6-3 クラス変数とクラスメソッド pp.174-181
  - 7-1 継承とは pp.191-194
  - 7-2 フィールドとメソッドの継承 pp.195-201

おまけ

オブジェクト指向プログラミングに関する読み物

- リーベルG. 高慢と偏見. Press Enter■, @IT, http://el.jibun.atmarkit.co.jp/pressenter/2010/11/1-828a.html, 2010.
  - 「Press Enter■」は，アイティメディア(株)の運営するIT系ニュースサイト「ITmedia」の一部である「@IT」に掲載されている，現役プログラマによる小説形式のブログ
  - 「高慢と偏見」はその第１作
  - 現在，８作目の長編「魔女の刻」の連載が佳境に入っている
  - 連載作品の中には，電子書籍として出版された作品もある．さらに，書籍として出版された作品も１つある
  - システムエンジニアやプログラマの職場の一端が感じられるという意味でもおすすめのブログ