



オブジェクト指向哲学

～入門編～

第4回 クラスとインスタンス

～手続き指向からオブジェクト指向へ()～

学習目標

- クラスとインスタンスの簡単な説明ができる
 - クラスとインスタンスの違いを説明できる
 - クラスを使って、変数をカタマリにしたプログラムが書ける
 - クラスを使って、メソッドをカタマリにしたプログラムが書ける

4.1. 同じ意味のデータはカタマリに

4.1.1. 商品名、価格を扱うプログラム

今までは商品種類といえば商品番号だけでしたが、なんとなく味気ないプログラムでしたね。ここからは商品名、価格も扱っていきたいと思います。今までの方法で、商品名と価格を扱えるようにしたプログラムを例題 4-1 に示します。

例題 4-1: 商品名、価格を扱う(Example4_1.java)

```
1:  /**
2:  * オブジェクト指向哲学 入門編
3:  * 例題 4-1: 商品名、価格を扱う
4:  * 取り扱う商品種類を管理するプログラム
5:  *
6:  * メインクラス
7:  */
8:  public class Example4_1 {
9:
10:     /**
11:     * メイン
12:     * 取り扱う商品種類を管理するプログラム
13:     */
14:     public static void main(String[] args) {
15:
16:         //自動販売機プログラムの開始を知らせる
17:         System.out.println("自動販売機のサービスが開始しました。");
18:
19:         //商品番号を保存するための配列を定義する
20:         int[] idArray = new int[10];
21:         //商品番号を保存するための変数を初期化する
22:         for(int i=0;i<10;i++){
23:             idArray[i] = -1; //何も入っていないことを-1として扱う
24:         }
25:
26:         //商品名を保存するための配列を定義する
27:         String[] nameArray = new String[10];
28:
29:         //商品の価格を保存するための配列を定義する
30:         int[] priceArray = new int[10];
31:         //商品の価格を保存するための変数を初期化する
32:         for(int i=0;i<10;i++){
33:             priceArray[i] = -1; //何も入っていないことを-1として扱う
34:         }
35:
36:         //商品種類を追加する
37:         add (idArray,nameArray,priceArray,1001,"コーラ",120); //コーラ
```

```
38:         add (idArray,nameArray,priceArray,1002,"ソーダ",120);//ソーダ
39:         add (idArray,nameArray,priceArray,1003,"お茶",120);//お茶
40:
41:         //商品種類リストを表示する
42:         display(idArray,nameArray,priceArray);
43:     }
44:
45:     /**
46:     * 商品種類を追加する
47:     */
48:     public static void add(int[] targetIdArray,String[] targetNameArray,int[]
targetPriceArray,int addID,String addName,int addPrice){
49:         //商品番号が入っていない箱を探して新しい商品番号を書き込む
50:         for(int i=0;i<10;i++){
51:             if(targetIdArray[i] == -1){//入っていない
52:                 targetIdArray[i] = addID;//書き込む
53:                 break;
54:             }
55:         }
56:
57:         //商品名が入っていない箱を探して新しい商品名を書き込む
58:         for(int i=0;i<10;i++){
59:             if(targetNameArray[i] == null){//入っていない
60:                 targetNameArray[i] = addName;//書き込む
61:                 break;
62:             }
63:         }
64:
65:         //商品価格が入っていない箱を探して新しい商品価格を書き込む
66:         for(int i=0;i<10;i++){
67:             if(targetPriceArray[i] == -1){//入っていない
68:                 targetPriceArray[i] = addPrice;//書き込む
69:                 break;
70:             }
71:         }
72:     }
73:
74:     /**
75:     * 商品種類リストを表示する
76:     */
77:     public static void display(int[] idArray,String[] nameArray,int[] priceArray){
78:         for(int i=0; i<10; i++){
79:             if(idArray[i] != -1 && nameArray[i] != null && priceArray[i] != -1){//商品種
種類が入っている
80:                 System.out.println(idArray[i]+":"+nameArray[i]+":"+priceArray[i]+" は販売
中です");
81:             }
82:         }
83:     }
84: }
```

4.1.2. 意味が分かりにくいプログラム

商品番号、商品名、価格とそれぞれ配列を用意することによって、ついに「コーラ」や「120円」などと表示するプログラムができます。しかし、このプログラム、何か気持ち悪くありませんか？

< 議論しよう！ > 例題 4-1 の問題点を議論しよう

4.1.3. 意味が明確なプログラムにするには

.オブジェクト

「商品種類」を「追加」する

という意味が明確なプログラムにするには、商品番号と商品名、価格は、一つのカタマリ「商品種類」としてプログラムで扱えるようにします。

このようなカタマリを、オブジェクト指向では、「オブジェクト」と呼びます。オブジェクト指向では、意味が明確なプログラムを書くために、データを意味ごとにカタマリにして、「オブジェクト」として捉えます。

商品番号=1001
商品名="コーラ"
価格=120

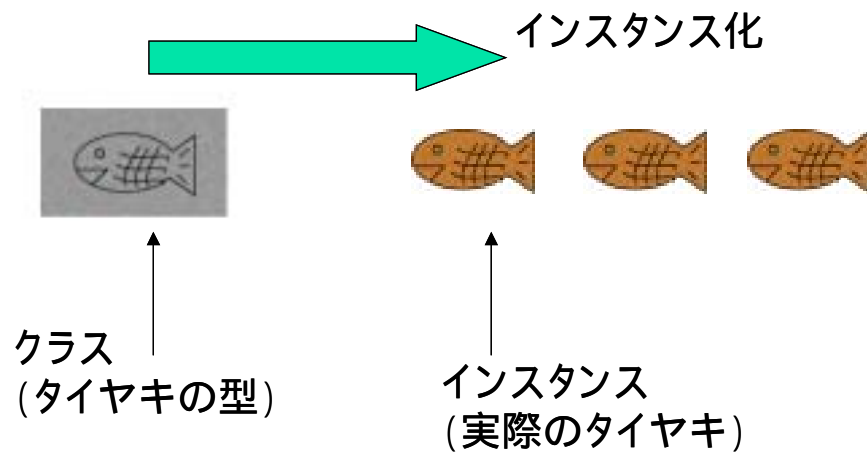
商品番号=1002
商品名="ソーダ"
価格=120

商品番号=1003
商品名="お茶"
価格=120

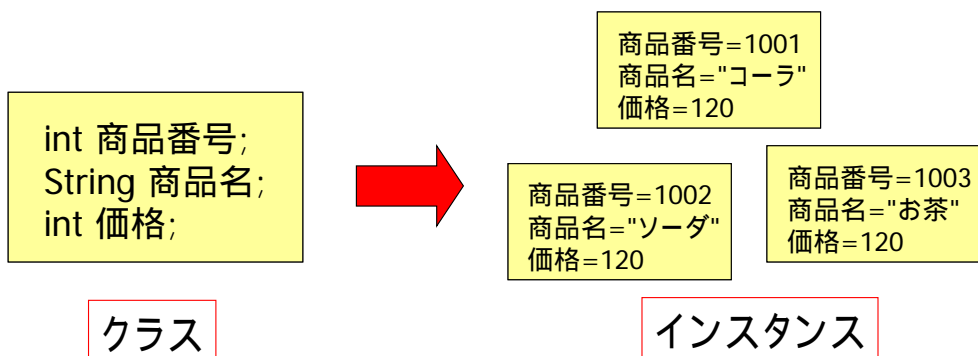
.クラス

オブジェクト指向の世界では、いきなりオブジェクトを作ることができないことになっています。オブジェクトを作る雛型を用意して、その雛型を使ってオブジェクトを作る必要があります。

クラスからできたオブジェクトのことを「インスタンス」と呼び、クラスからインスタンスを生成することを「インスタンス化」といいます。



今回は、以下のようにクラスを定義して、そのクラスからインスタンスを作成します。



4.2. Java でクラスを使ったプログラムを書く

Java においてクラス/インスタンスを使ったプログラムを書く方法を説明します。

4.2.1. クラスを使ったプログラム

.プログラムを分ける

まず、クラス/インスタンスは、プログラムを分けて書く必要があります。「意味」ごとにプログラムを分けるのです。これは非常に分かりやすいですね。

今回は、「商品種類」クラスと「メイン」クラスごとにプログラムを分けることとなります。

```
/**
 * 商品種類クラス
 */
public class ItemType{

    int id;//商品番号
}
```

ItemType.java

```
/**
 * メインクラス
 */
public class Example4_2{

    public static void main( String[] args){

    }
}
```

Example4_2.java

重要なことは、異なるプログラムは異なるファイルにすることです。今回の場合は `ItemType.java` というファイルを用意して、その中で `ItemType` クラスを定義します。ここでも「クラス名とファイル名は同じにする」という決まりを守ってください。

.クラスを定義する

では、早速「商品種類」クラスを定義してみましょう。

例題 4-2:初めてのクラス(ItemType.java)

```
1:  /**
2:   * オブジェクト指向哲学 入門編
3:   * 例題 4-2:初めてのクラス
4:   * 取り扱う商品種類を管理するプログラム
5:   *
6:   * 商品種類クラス
7:   */
8:  public class ItemType {
9:
10:     int id;          //商品番号
11:     String name;    //商品名
12:     int price;      //価格
13:
14: }
```

8行目が重要です！

- **public**

そのクラスの「可視性」(あとで説明します)を定義します。

- **class**

「クラス」を定義しますよという意味です。省略できません。

- **ItemType**

クラスの名前です。省略できません。任意の名前を付けることができますが、慣習として、頭文字は大文字にします。(第一回の命名規則を参考にしてください)

10行目から12行目まで、カタマリにする変数を定義します。

- 変数定義部分

```
int id;//商品番号
String name;//商品名
int price;//商品価格
```

クラスの中、メソッドの外に書かれるのが、カタマリにすべき変数となります。

.インスタンスを生成する

では、作った `ItemType` クラスを利用して、インスタンスを作ってみましょう。

例題 4-2:初めてのクラス(Example4_1.java)

```
1:  /**
2:  * オブジェクト指向哲学 入門編
3:  * 例題 4-2:初めてのクラス
4:  * 取り扱う商品種類を管理するプログラム
5:  *
6:  * メインクラス
7:  */
8:  public class Example4_2 {
9:
10:     /**
11:     * メイン
12:     * 取り扱う商品種類を管理するプログラム
13:     * 商品種類インスタンスを生成し、表示する
14:     */
15:     public static void main(String[] args) {
16:
17:         //自動販売機プログラムの開始を知らせる
18:         System.out.println("自動販売機のサービスが開始しました。");
19:
20:         //商品種類を追加する
21:         //コーラ商品種類クラスをインスタンス化して追加する
22:         ItemType cola = new ItemType();
23:         cola.id = 1001;
24:         cola.name = "コーラ";
25:         cola.price = 120;
26:
27:         //コーラ商品種類を表示してみる
28:         System.out.println(cola.id+": "+cola.name+": "+cola.price+"は販売中です");
29:
30:     }
31:
32: }
```

- 22 行目が重要です。

クラスは変数の型になります：

```
ItemType cola;
```

クラスは、int、String などと同じように変数の型として考えます。ここでは「ItemType 型の変数 cola を宣言する」という意味になります。

インスタンスを生成します：

```
new ItemType();
```

インスタンスを一つ作るには、new 演算子を使います。

インスタンスを生成して代入する：

```
ItemType cola = new ItemType();
```

ItemType 型の変数を定義から生成、代入を一行で書くと上記のようになります。

- 23 から 25 行目

インスタンスの変数への代入：

```
cola.id = 1001;  
cola.name = "コーラ";  
cola.price = 120;
```

インスタンスの変数への代入が行われています。

インスタンスの変数にアクセスするには以下のように書きます。

```
インスタンス名.変数名
```

4.2.2. クラスの配列を使う

ItemType というクラスをすることによって、商品種類というカタマリでプログラムが書けるようになりました。それでは、商品種類を管理するプログラムを書いていきましょう。

追加や検索を行うには、配列が便利です。ここでは、「クラス配列」を使って、プログラムする方法を説明します。

商品種類の配列を使った、商品種類の管理プログラム(追加のみ)を行うプログラムを例題 4-3 として示します。

例題 4-3: 商品種類を使った自動販売機(Example4_3.java)

```
1:    /**
2:    * オブジェクト指向哲学 入門編
3:    * 例題 4-3: 商品種類配列を使った自動販売機
4:    * 取り扱う商品種類を管理するプログラム
5:    *
6:    * メインクラス
7:    */
8:    public class Example4_3 {
9:
10:       /**
11:       * メイン
12:       * 取り扱う商品種類を管理するプログラム
13:       * コーラ、ソーダ、お茶を追加し、リストを表示する
14:       */
15:       public static void main(String[] args) {
16:
17:           //自動販売機プログラムの開始を知らせる
18:           System.out.println("自動販売機のサービスが開始しました。");
19:
20:           //商品種類を保存するための配列を定義する
21:           ItemType[] itemTypeArray = new ItemType[10];
22:
23:           //商品種類を追加する
24:           //商品種類インスタンスをインスタンス化(生成)して追加する
25:           ItemType cola = new ItemType();
26:           cola.id = 1001;
27:           cola.name = "コーラ";
28:           cola.price = 120;
29:           add(itemTypeArray, cola);
30:
31:           //ソーダ商品種類インスタンスをインスタンス化(生成)して追加する
```

```

32:     ItemType soda = new ItemType();
33:     soda.id = 1002;
34:     soda.name = "ソーダ";
35:     soda.price = 120;
36:     add(itemTypeArray,soda);
37:
38:     //お茶商品種類インスタンスをインスタンス化（生成）して追加する
39:     ItemType tea = new ItemType();
40:     tea.id = 1003;
41:     tea.name = "お茶";
42:     tea.price = 120;
43:     add(itemTypeArray,tea);
44:
45:     //商品種類リストを表示する
46:     display(itemTypeArray);
47:
48: }
49:
50: /**
51:  * 商品種類を追加する
52:  */
53: public static void add(ItemType[] targetArray,ItemType addItemType){
54:     //商品種類が入っていない箱を探して格納する
55:     for(int i=0;i<10;i++){
56:         if(targetArray[i] == null){//入っていない
57:             targetArray[i] = addItemType;//書き込む
58:             break;
59:         }
60:     }
61: }
62:
63: /**
64:  * 商品種類リストを表示する
65:  */
66: public static void display(ItemType[] targetArray){
67:     for(int i=0;i<10;i++){
68:         if(targetArray[i] != null){//商品種類が入っている
69:             System.out.println(targetArray[i].id+":"+targetArray[i].name+":"+targetArray[i].price+"
は販売中です");
70:         }
71:     }
72: }
73:
74: }

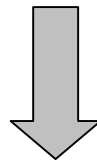
```

.配列の宣言

今までは、商品種別を扱う際には `int` の配列を使って表現していました。それを、`ItemType` クラスに変えます。

int 型配列の宣言

```
//商品種別を保存するための配列を定義する  
int[] itemTypeArray = new int[10];
```



ItemType 型配列の宣言

```
//商品種別を保存するための配列を定義する  
ItemType[] itemTypeArray = new ItemType[10];
```

`ItemType` 型の配列の宣言は、基本的には `int` 型の配列の宣言と同じです。

.配列への代入

配列へのインスタンスの代入は、次のようなプログラムで記述されます。

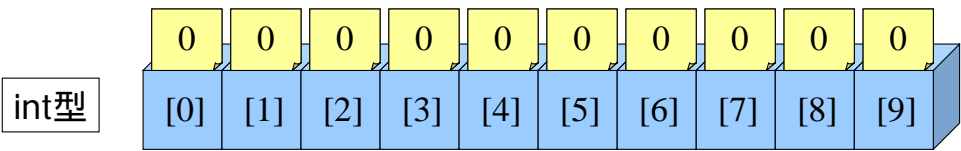
```
//配列を作る  
ItemType[] itemTypeArray = new ItemType[10];  
  
//ItemType インスタンスを作って、配列の0番地に代入する  
itemTypeArray[0] = new ItemType();  
  
//配列の0番地にある ItemType インスタンスの変数にデータを入れる。  
itemTypeArray[0].id = 1000;  
itemTypeArray[0].name = "コーラ";
```

.クラス配列と int 配列の違い 初期化

クラスの配列と int の配列は、初期化する際には大きな差が出ます。以下をご覧ください。

int 型配列の初期化

```
//intの配列を作る
int[] itemInfoArray = new int[10];
```

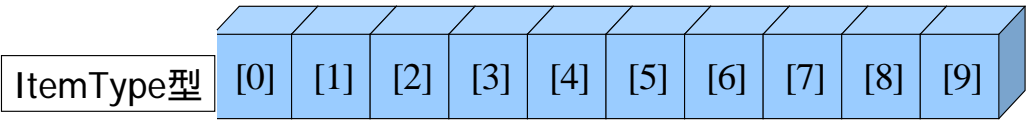


The diagram shows a horizontal array of 10 blue boxes representing array elements, indexed from [0] to [9]. Above each box is a yellow box containing the number 0. To the left of the array is a label 'int型' in a box.

int 型の配列の場合は、配列の各要素に初期値として 0 が入りました。

ItemType 型配列の初期化

```
//ItemType 型の配列を作る
ItemType[] itemTypeArray = new ItemType[10];
```



The diagram shows a horizontal array of 10 blue boxes representing array elements, indexed from [0] to [9]. To the left of the array is a label 'ItemType型' in a box.

ところが、ItemType 型の配列の場合は、初期値として何も入っていません!
このようにインスタスが何も入っていない状態のことを **null** といいます。

入っていない (null) 状態の変数に、代入 / 参照してはいけません。もしやろうとした場合は、エラーが発生します (詳しくは次ページ以降の Tips を参照)。必ず配列の各要素にインスタスを代入した上で、代入 / 参照をするようにしましょう。

Java Tips 例外

以下のような、`null` 状態のインスタンスに参照するようなプログラムを実行した場合はどうなるでしょうか？

```
//これはまずいプログラム
public class FailProgram{
    public static void main(String[] args){
        ItemType[] itemTypeArray = new ItemType[10]; //配列を作る

        //itemTypeArray[0]はnull
        itemTypeArray[0].id = 1001;
        itemTypeArray[0].name = "コーラ";
        itemTypeArray[0].price = 120;
    }
}
```

実行すると以下のような例外を出力して、プログラムが強制的に終了してしまいます。

```
C:\objprog>java FailProgram
Exception in thread "main" java.lang.NullPointerException
    at FailProgram.main(FailProgram.java:7)
```

これは、「FailProgram クラスの main メソッド内にて（ソースファイル FailProgram.java の7行目において）インスタンスが入っていないのに代入 / 参照しようとしたので、例外が発生しました。」という意味のメッセージです。

このプログラムはインスタンスに対して代入することを意図したプログラムなのでした。しかし、実際はインスタンスはなかったため、代入できなかったわけです。「例外」は意図した通りの状態ではない時に発生する一種のエラーです。今回のように `null` 状態のインスタンスに代入 / 参照しようとした時に発生する例外を、`NullPointerException` と呼びます。

その他の例外を紹介します。

- **ArrayIndexOutOfBoundsException**

配列に無い番地を読もうとした時に発生する例外です。例えば上記の `itemTypeArray` 配列の場合は、`itemTypeArray[-1]` や `itemTypeArray[20]` などを参照 / 代入しようとする時、この例外が発生します。

- **ArithmeticException**

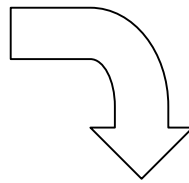
算術計算で問題が発生した時に発生する例外です。例えば整数を「ゼロで割る」と、この例外が発生します。

4.2.3. コンストラクタ

コンストラクタとは、インスタンスを生成する時に実行される特別なメソッドです。一般的にコンストラクタを使って生成したインスタンスの属性値を初期化するのに使います。

コンストラクタを使った結果、以下のようにソースコードがより簡潔で美しくすることができます。

```
//商品種類をインスタンス化する
//コーラ
ItemType cola = new ItemType();
cola.id = 1001;
cola.name = "コーラ";
cola.price = 120;
//ソーダ
ItemType soda = new ItemType();
soda.id = 1002;
soda.name = "ソーダ";
soda.price = 120;
//お茶
ItemType greentea = new ItemType();
greentea.id = 1003;
greentea.name = "お茶";
greentea.price = 120
```



```
//商品種類をインスタンス化する
//コーラ
ItemType cola = new ItemType(1001,"コーラ",120);
//ソーダ
ItemType soda = new ItemType(1002,"ソーダ",120);
//お茶
ItemType greentea = new ItemType(1003,"お茶",120);
```


.コンストラクタの宣言

例題 4-4: コンストラクタ(ItemType.java)

```
1:  /**
2:  * オブジェクト指向哲学 入門編
3:  * 例題 4-4: コンストラクタ
4:  * 取り扱う商品種類を管理するプログラム
5:  *
6:  * 商品種類クラス
7:  */
8:  public class ItemType {
9:
10:     int id;          //商品番号
11:     String name;    //商品名
12:     int price;      //価格
13:
14:     /**
15:     * コンストラクタ
16:     */
17:     public ItemType(int newID,String newName,int newPrice) {
18:         name = newName;
19:         id = newID;
20:         price = newPrice;
21:     }
22: }
```

17 行目

クラス名と同じ名前で、メソッドのようなものを宣言します(同じ名前であればいけません!)。メソッドと違って、void などの返り値宣言がありません。(static もありません)

コンストラクタには引数を取ることもできます。ここでは商品番号と商品名、価格の初期データを引数として受け取ろうとしています。

18~20 行目

コンストラクタの中で渡された初期データを代入することで、ItemType クラスの商品番号と商品名、価格を初期化します。

.コンストラクタの呼び出し

ItemType クラスで宣言されたコンストラクタを呼び出すメインクラスを示します。

例題 4-4: コンストラクタ(Example4_4.java)

```
1:  /**
2:   * オブジェクト指向哲学 入門編
3:   * 例題 4-4: コンストラクタ
4:   * 取り扱う商品種類を管理するプログラム
5:   *
6:   * メインクラス
7:   */
8:  public class Example4_4 {
9:
10:     /**
11:     * メイン
12:     * 取り扱う商品種類を管理するプログラム
13:     * コーラ、ソーダ、お茶を追加し、リストを表示する
14:     */
15:     public static void main(String[] args) {
16:
17:         //自動販売機プログラムの開始を知らせる
18:         System.out.println("自動販売機のサービスが開始しました。");
19:
20:         //商品種類を保存するための配列を定義する
21:         ItemType[] itemTypeArray = new ItemType[10];
22:
23:         //商品種類を追加する
24:         add(itemTypeArray, new ItemType(1001, "コーラ", 120));
25:         add(itemTypeArray, new ItemType(1002, "ソーダ", 120));
26:         add(itemTypeArray, new ItemType(1003, "お茶", 120));
27:
28:         //商品種類リストを表示する
29:         display(itemTypeArray);
30:
31:     }
32:
33:     /**
34:     * 商品種類を追加する
35:     */
36:     public static void add(ItemType[] targetArray, ItemType addItemType){
37:         //商品種類が入っていない箱を探す
38:         for(int i=0; i<10; i++){
39:             if(targetArray[i] == null){//入っていない
40:                 targetArray[i] = addItemType;//書き込む
41:                 break;
42:             }
43:         }
44:     }
45: }
```

```
44:     }
45:
46:     /**
47:     * 商品種類リストを表示する
48:     */
49:     public static void display(ItemType[] targetArray){
50:         for(int i=0;i<10;i++){
51:             if(targetArray[i] != null){//商品種類が入っている
52:
53: System.out.println(targetArray[i].id+": "+targetArray[i].name+": "+targetArray[i].price+"
は販売中です");
54:             }
55:         }
56:     }
57: }
```

24 ~ 26 行目

コンストラクタが呼び出されています。

```
add(itemTypeArray,new ItemType(1001,"コーラ",120));
```

このように、`ItemType` インスタンスを生成する時に実引数を渡すと(この場合は `1001`、`コーラ`、`120`)、コンストラクタの仮引数(この場合は `newNo`、`newName`、`newPrice`)にデータが入り、コンストラクタの中の処理が実行されます。その結果、生成されたインスタンスには既に商品番号、商品名、価格のデータが入っているのです。

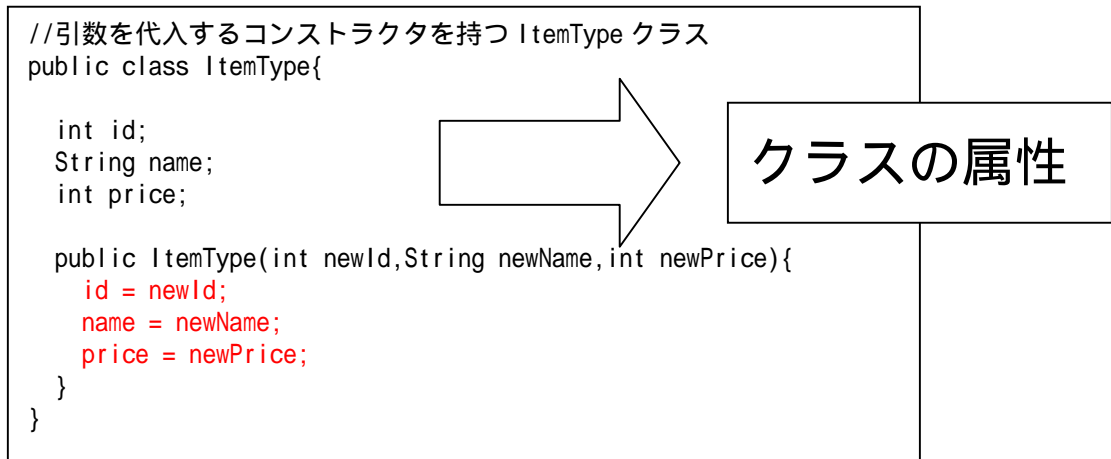
Java Tips 変数のスコープ(2)

前ページの `ItemType.java` のリストを眺めると、コンストラクタの中で `id` と `name` と `price` にアクセスしている事がわかります。しかし以前は「メソッド内で宣言した変数はメソッド内でのみ有効」と教わりましたね。`id` と `name` と `price` はコンストラクタの外で宣言されているにもかかわらず、使うことができます。それは何故でしょう。

```
//引数を代入するコンストラクタを持つ ItemType クラス
public class ItemType{

    int id;
    String name;
    int price;

    public ItemType(int newId,String newName,int newPrice){
        id = newId;
        name = newName;
        price = newPrice;
    }
}
```



クラスの属性

`id` と `name` と `price` はクラスの属性として宣言されています。変数をクラス属性として宣言すると、この二つの変数は `ItemType` クラスのどこでも使うことができます。

ちなみに変数が見える範囲、有効範囲のことをスコープと言います。クラス属性のスコープは、クラス全体に及びます。一方でメソッドの中で宣言された変数のスコープは、そのメソッドの中だけに限定されます。

4.3. クラスを利用してプログラムを分割する

4.3.1. 意味ごとにプログラムを分割する

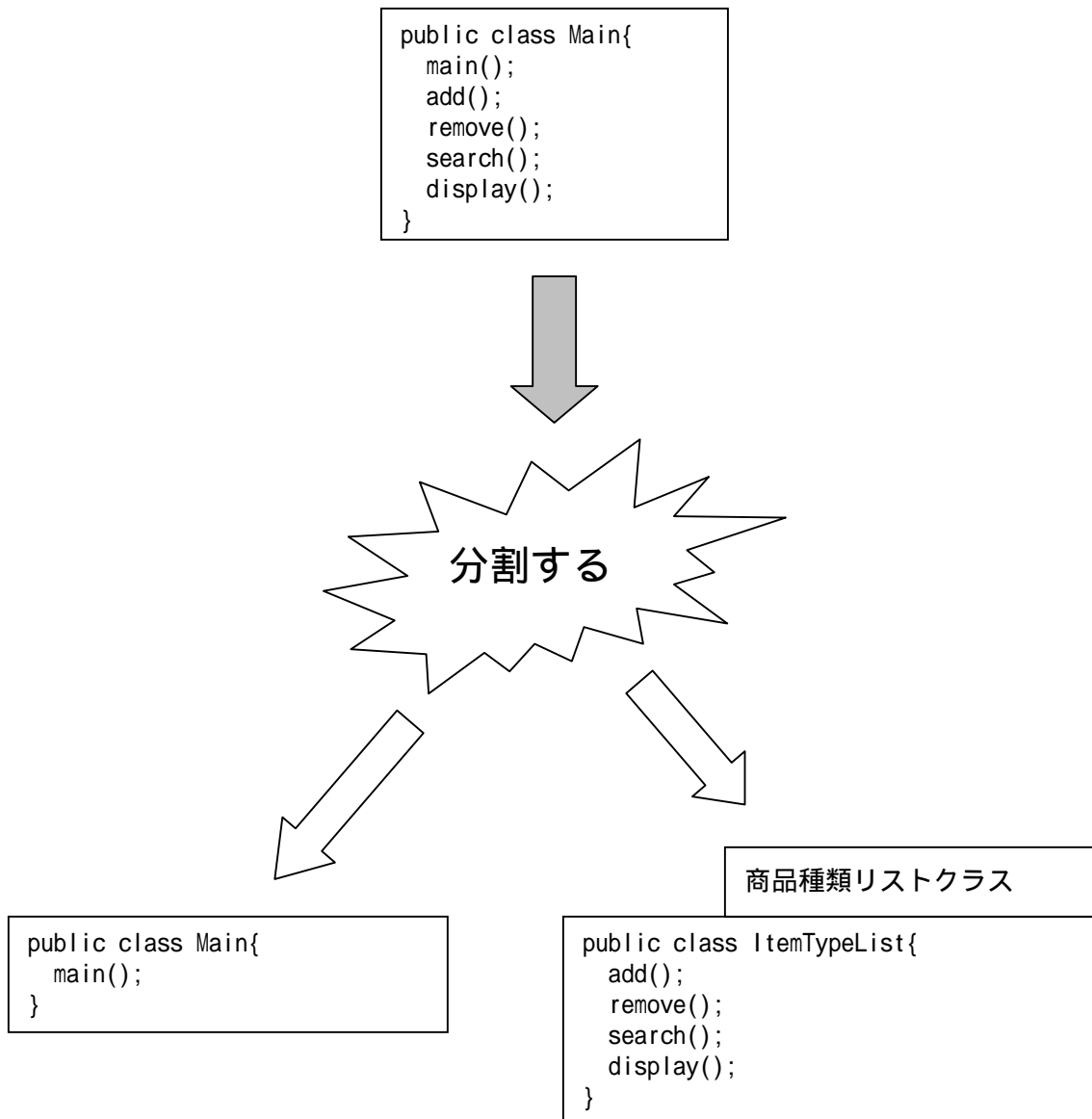
前回は、プログラムの意味を考えて、意味が同じ仕事をカタマリとしてメソッド化する作業を行いました。今回は、さらにプログラムの意味を考えて、メソッドを整理し、より分かりやすいプログラムにすることを考えましょう。

```
public class Example{  
    main();  
    add();  
    remove();  
    search();  
    display();  
}
```

上記の 5 つのメソッドをプログラムの意味を考えて分類するとしたら、どのような分類になるでしょうか？

.メソッドを意味ごとに分類し、クラスを作る

「商品種類リスト」クラスを作り、商品管理のメソッドを移すことでプログラムを分類しましょう。



分類したプログラムを示します。どうですか？意味がより明らかなプログラムになりましたか？

例題 4-5: 商品種類リストクラスの導入(Example4_5.java)

```
1:  /**
2:  * オブジェクト指向哲学 入門編
3:  * 例題 4-5: 商品種類リストクラスの導入
4:  * 取り扱う商品種類を管理するプログラム
5:  *
6:  * メインクラス
7:  */
8:  public class Example4_5 {
9:
10:     /**
11:     * メイン
12:     * 取り扱う商品種類を管理するプログラム
13:     * コーラ、ソーダ、お茶を追加し、リストを表示する
14:     */
15:     public static void main(String[] args) {
16:
17:         //自動販売機プログラムの開始を知らせる
18:         System.out.println("自動販売機のサービスが開始しました。");
19:
20:         //商品種類配列リストを生成する
21:         ItemTypeList itemTypeList = new ItemTypeList();
22:
23:         //商品種類を保存するための配列を定義する
24:         ItemType[] itemTypeArray = new ItemType[10];
25:
26:         //商品種類を追加する
27:         itemTypeList.add(itemTypeArray,new ItemType(1001,"コーラ",120));
28:         itemTypeList.add(itemTypeArray,new ItemType(1002,"ソーダ",120));
29:         itemTypeList.add(itemTypeArray,new ItemType(1003,"お茶",120));
30:
31:         //商品種類リストを表示する
32:         itemTypeList.display(itemTypeArray);
33:
34:     }
35: }
```

他のクラスのメソッドを呼び出すには、インスタンスを生成した上で、以下のような書き方をします。

```
インスタンス名.メソッド名(引数)
```

例題 4-5: 商品種類リストクラスの導入(ItemTypeList.java)

```
1:  /**
2:  * オブジェクト指向哲学 入門編
3:  * 例題 4-5 : 商品種類リストクラスの導入
4:  * 取り扱う商品種類を管理するプログラム
5:  *
6:  * 商品種類リストクラス
7:  */
8:  public class ItemTypeList {
9:
10:     /**
11:     * 商品種類を追加する
12:     */
13:     public void add(ItemType[] targetArray, ItemType addItemType){
14:         //商品種類が入っていない箱を探す
15:         for(int i=0;i<10;i++){
16:             if(targetArray[i] == null){//入っていない
17:                 targetArray[i] = addItemType;//書き込む
18:                 break;
19:             }
20:         }
21:     }
22:
23:     /**
24:     * 商品種類リストを表示する
25:     */
26:     public void display(ItemType[] targetArray){
27:         for(int i=0;i<10;i++){
28:             if(targetArray[i] != null){//商品種類が入っている
29:                 System.out.println(targetArray[i].id+": "+targetArray[i].name+": "+targetArray[i].price+"
は販売中です");
30:             }
31:         }
32:     }
33:
34: }
```

この ItemTypeList クラスのメソッドは、Main クラスのときとは違って「static」が書いてありません。ItemTypeList クラスはインスタンス化されるため、static が必要ないのです。というよりも、インスタンス化されるクラスに static を書いてはいけません。理由は第 10 回で説明します。

練習問題

< 記述問題 >

記述問題 4-1

クラス/インスタンスを使ったプログラムを書くことの利点を説明せよ。

記述問題 4-2

クラスとインスタンスの違いを列挙せよ。

< プログラム問題 >

プログラム問題 4-1

プログラム問題 3-1 で作ったプログラムを「商品種類(ItemType)」クラスを使って、商品名、価格が扱えるようにせよ。

- ファイル、クラス構成

```
public class Exercise4_1{
    main()
    add()
    remove()
    search()
    display()
}
```

Exercise4_1.java

```
public class ItemType{
    int id
    String name
    int price
}
```

ItemType.java

プログラム問題 4-2

プログラム問題 4-1 で作ったプログラムを「商品種類リスト(ItemTypeList)」クラスを作ることによって、メソッドを意味の観点から分類せよ。

- ファイル、クラス構成

```
public class Exercise4_2{
    main()
}
```

Exercise4_2.java

```
public class ItemType{
    int id
    String name
    int price
}
```

ItemType.java

```
public class ItemTypeList{
    add();
    remove();
    search();
    display();
}
```

ItemTypeList.java