オブジェクト指向哲学
～入門編～

第6回　データ構造とアルゴリズムの結合

～手続き指向からオブジェクト指向へ(Ⅱ)～

学習目標

- データ構造とアルゴリズムが結合することの利点を説明できる
  - クラスのデータをカプセル化することの利点を説明できる
- データ構造とアルゴリズムを結合させたクラスを使ったプログラムが書ける
- クラス図が読める

6.1.　データ構造とアルゴリズムの結合

6.1.1.　仕様変更に伴う Main プログラムの変更

今回は、Main プログラム（メインクラスの main()に書かれたプログラム）と第 4 回、第 5 回で学んだ 2 種類の商品種類リストクラスの関係を掘り下げます。

今回のプログラムでは、2 種類の商品種類リストクラスを区別するために、次のようにクラスの名前を変更します。

● 第 4 回において配列で実装された ItemTypeList クラス
今回は「ItemTypeArrayList」クラスと改名します。

● 第 5 回において連結リストで実装された ItemTypeList クラス
今回は「ItemTypeLinkedList」クラスと改名します。

①.配列リストによる商品種類追加プログラム

まず、配列リストによる商品種類の追加プログラムを眺めて見ましょう。変更されているのは、クラス名のみです。

例題 6-1：配列を利用した、商品種類の追加(Example6_1.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 6-1：配列を利用した、商品種類の追加
4:      * 商品種類をリストに追加するプログラム
5:      *
6:      * メインクラス
7:      */
8:      public class Example6_1 {
9:
10:       /**
11:       * メイン
12:       * 取り扱う商品種類を追加するプログラム
13:       */
14:       public static void main(String[] args) {
15:
16:         //商品種類配列リストを生成する
17:         ItemTypeArrayList itemTypeList = new ItemTypeArrayList();
18:         //商品種類を保存するための配列を定義する
19:         ItemType[] itemTypeArray = new ItemType[10];
20:
21:         //商品種類を追加する
22:         itemTypeList.add(itemTypeArray,new ItemType(1001,"コーラ",120));
23:         itemTypeList.add(itemTypeArray,new ItemType(1002,"ソーダ",120));
24:         itemTypeList.add(itemTypeArray,new ItemType(1003,"お茶 ",120));
25:         itemTypeList.add(itemTypeArray,new ItemType(1004,"DD レモン",120));
26:         itemTypeList.add(itemTypeArray,new ItemType(1005,"ウーロン茶",120));
27:
28:         //商品種類リストを表示する
29:         itemTypeList.display(itemTypeArray);
30:
31:       }
32:     }
```

②.実装方法変更

ここで仕様変更があり、頻繁に要素の大きさが変わるようになることを考えます。

配列では大きさが限られてしまい不都合なので､メモリを効率的に使うことのできる連結リストに実装を変更します。

メインプログラムは次のように変更されます。

例題 6-2:連結リストを利用した、商品種類の追加(Example6_2.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 6-2：連結リストを利用した、商品種類の追加
4:      * 取り扱う商品種類を追加するプログラム
5:      *
6:      * メインクラス
7:      */
8:      public class Example6_2 {
9:
10:       /**
11:       * メイン
12:       * 取り扱う商品種類を追加するプログラム
13:       */
14:       public static void main(String[] args) {
15:
16:         //商品種類連結リストを生成する
17:         ItemTypeLinkedList itemTypeList = new ItemTypeLinkedList();
18:         //商品種類を保存するための連結始点終点を生成する
19:         LinkTerminal linkTerminal = new LinkTerminal();
20:
21:         //商品種類を追加する
22:         itemTypeList.add(linkTerminal,new ItemType(1001,"コーラ",120));
23:         itemTypeList.add(linkTerminal,new ItemType(1002,"ソーダ",120));
24:         itemTypeList.add(linkTerminal,new ItemType(1003,"お茶",120));
25:         itemTypeList.add(linkTerminal,new ItemType(1004,"DD レモン",120));
26:         itemTypeList.add(linkTerminal,new ItemType(1005,"ウーロン茶",120));
27:
28:         //商品種類リストを表示する
29:         itemTypeList.display(linkTerminal);
30:
31:       }
32:     }
```

③.またまた実装変更！？

次に、また仕様変更が起こり、高速に検索できる配列を利用することになりました。また、例題 6-1 のようなプログラムに戻していきます。このプログラムは 100 要素の商品種類を追加するようになっています。

しかし、このプログラムは意図した通りに動いてくれませんでした。何故でしょう。

＜考えよう！＞このプログラムのまずい点

例題 6-3：また仕様変更(Example6_3.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 6-3：また仕様変更
4:      * 商品種類をリストに追加するプログラム
5:      *
6:      * メインクラス
7:      */
8:      public class Example6_3 {
9:
10:       /**
11:       * メイン
12:       * 取り扱う商品種類を追加するプログラム
13:       */
14:       public static void main(String[] args) {
15:
16:         //商品種類配列リストを生成する
17:         ItemTypeArrayList itemTypeList = new ItemTypeArrayList();
18:         //商品種類を保存するための配列を定義する
19:         ItemType[] itemTypeArray = new ItemType[100];
20:
21:         //商品種類を追加する
22:         for(int i=0;i<100;i++){
23:           itemTypeList.add(itemTypeArray,new ItemType(i,"コーラ",120));
24:         }
25:
26:         //商品種類リストを表示する
27:         itemTypeList.display(itemTypeArray);
28:
29:       }
30:     }
```

6.1.2. 問題の本質

問題の本質を、2 つの側面から考えて見ましょう。

①.クラスの役割分担

現状のクラスの役割分担から、何故このような問題が起こったのか、考えてみましょう。

②.意味を明確にしたプログラムを書きたい

役割分担の問題と似ていますが、メインプログラムが本来やるべき仕事は、商品種類の追加や表示を順序良く命令することで、できるならば、その本来の仕事の意味を明確にしたプログラムを書きたいのです。

6.1.3.　データ構造とアルゴリズムの結合

問題を解決するために、商品種類リストに商品リストを管理する役割を一手に引き受けてもらうことにします。

プログラムの意味を考えると、配列とそのアルゴリズム、連結リストとそのアルゴリズムは、データ構造とアルゴリズムは表裏一体のきっても切り離せない関係ですね。これらを「意味のカタマリ」として結合できるのは、オブジェクト指向の大きな利点といわれています。

＜考えよう！＞データ構造とアルゴリズムを結合させる利点！

次のページから、データ構造とアルゴリズムを結合させ「配列リスト」、「連結リスト」という「意味のカタマリ」を作った場合のプログラムを示します。これらを読み、次の視点からデータ構造とアルゴリズムを結合させることの利点について議論してみましょう。

①各クラスの役割という視点から

-
-
-

②Main クラスのプログラムの明確化という視点から

-
-
-

③役割の明確化、プログラムの明確化は、バグの発見にどう寄与するでしょうか？

-
-
-

データ構造とアルゴリズムを結合したプログラムを、①配列リスト版、②連結リスト版の順に示します。

①.配列リスト

例題 6-4：配列のデータ構造とアルゴリズムを結合する(Example6_4.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 6-4：配列のデータ構造とアルゴリズムを結合する
4:      * 商品種類をリストに追加するプログラム
5:      *
6:      * メインクラス
7:      */
8:      public class Example6_4 {
9:
10:       /**
11:       * メイン
12:       * 取り扱う商品種類を追加するプログラム
13:       */
14:       public static void main(String[] args) {
15:
16:         //商品種類配列リストを生成する
17:         ItemTypeArrayList itemTypeList = new ItemTypeArrayList();
18:
19:         //商品種類を追加する
20:         itemTypeList.add(new ItemType(1001,"コーラ",120));
21:         itemTypeList.add(new ItemType(1002,"ソーダ",120));
22:         itemTypeList.add(new ItemType(1003,"お茶",120));
23:         itemTypeList.add(new ItemType(1004,"DD レモン",120));
24:         itemTypeList.add(new ItemType(1005,"ウーロン茶",120));
25:
26:         //商品種類リストを表示する
27:         itemTypeList.display();
28:
29:       }
30:     }
```

例題 6-4：配列のデータ構造とアルゴリズムを結合する

(ItemTypeArrayList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 6-4：配列のデータ構造とアルゴリズムを結合する
4:      * 商品種類をリストに追加するプログラム
5:      *
6:      * 商品種類配列リストクラス
7:      */
8:      public class ItemTypeArrayList {
9:
10:       ItemType[] itemTypeArray = new ItemType[10];  //商品種類を保存する
11:
12:       /**
13:       * 商品種類を追加する
14:       */
15:       public void add(ItemType addItemType){
16:         //商品種類が入っていない箱を探す
17:         for(int i=0;i<10;i++){
18:           if(itemTypeArray[i] == null){//入っていない
19:             itemTypeArray[i] = addItemType;//書き込む
20:             break;
21:           }
22:         }
23:       }
24:
25:       /**
26:       * 商品種類リストを表示する
27:       */
28:       public void display(){
29:         for(int i=0;i<10;i++){
30:           if(itemTypeArray[i] != null){//商品種類が入っている
31:
System.out.println(itemTypeArray[i].id+":"+itemTypeArray[i].name+":"+itemTypeArray[i].pr
ice+"は販売中です");
32:           }
33:         }
34:       }
35:
36:     }
```

データ構造

データ操作の
アルゴリズム

②.連結リスト

例題 6-5：連結リストのデータ構造とアルゴリズムを結合する

(Example6_5.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 6-5：連結リストのデータ構造とアルゴリズムを結合する
4:      * 商品種類をリストに追加するプログラム
5:      *
6:      * メインクラス
7:      */
8:      public class Example6_5 {
9:
10:       /**
11:       * メイン
12:       * 取り扱う商品種類を追加するプログラム
13:       */
14:       public static void main(String[] args) {
15:
16:         //商品種類配列リストを生成する
17:         ItemTypeLinkedList itemTypeList = new ItemTypeLinkedList();
18:
19:         //商品種類を追加する
20:         itemTypeList.add(new ItemType(1001,"コーラ",120));
21:         itemTypeList.add(new ItemType(1002,"ソーダ",120));
22:         itemTypeList.add(new ItemType(1003,"お茶",120));
23:         itemTypeList.add(new ItemType(1004,"DD レモン",120));
24:         itemTypeList.add(new ItemType(1005,"ウーロン茶",120));
25:
26:         //商品種類リストを表示する
27:         itemTypeList.display();
28:
29:       }
30:     }
```

例題 6-5：連結リストのデータ構造とアルゴリズムを結合する

(ItemTypeLinkedList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 6-5：連結リストのデータ構造とアルゴリズムを結合する
4:      * 商品種類をリストに追加するプログラム
5:      *
6:      * 商品種類連結リストクラス
7:      */
8:      public class ItemTypeLinkedList {
9:
10:       LinkObject first;//始点
11:       LinkObject last;//終点
12:
13:       /**
14:       * 商品種類を追加する
15:       */
16:       public void add(ItemType addItemType){
17:         //追加する連結インスタンスを生成する
18:         LinkObject addLink = new LinkObject();
19:         addLink.data = addItemType;
20:
21:         if(first == null){//連結リストが空のとき
22:           first = addLink;
23:           last = addLink;
24:         }else{//連結リストが空でないとき
25:           last.next = addLink;
26:           last = addLink;
27:         }
28:       }
29:
30:       /**
31:       * 商品種類リストを表示する
32:       */
33:       public void display(){
34:         LinkObject current = first;//今たどっている連結インスタンス
35:         while(current != null){
36:
System.out.println(current.data.id+":"+current.data.name+":"+current.data.price+" は販売
中です");
37:           current = current.next;
38:         }
39:       }
40:
41:     }
```

データ構造

データ操作の
アルゴリズム

6.2. カプセル化

6.2.1. 不正な操作

ItemTypeArrayList クラスにおいて、以下のようなコードは一般的に誤った使い方、不正な扱いとされます。

```
//商品種類リストインスタンスを生成する
ItemTypeArrayList itemTypeArrayList = new ItemTypeArrayList();

//商品種類を追加する
itemTypeArrayList.add(new ItemType(1001,"コーラ"));
itemTypeArrayList.add(new ItemType(1002,"ソーダ"));
itemTypeArrayList.itemTypeArray[0] = null;
```

なぜこれが「誤った使い方」とされるのでしょうか？

-
-
-

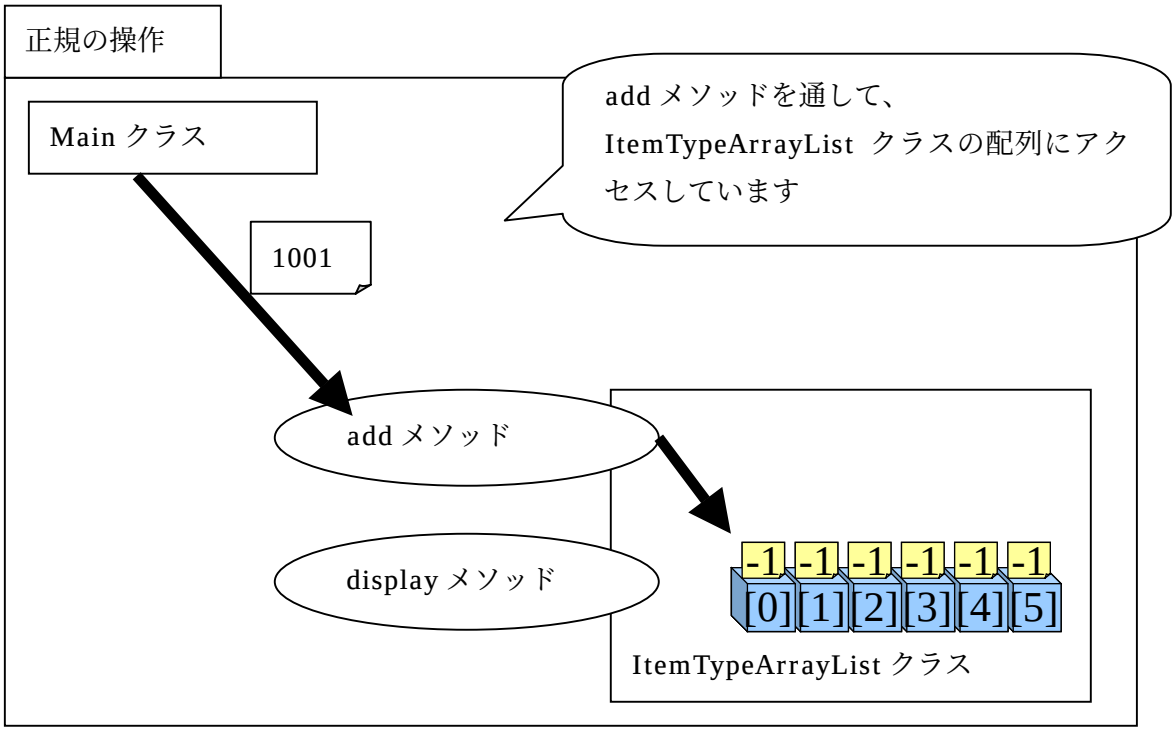
では以下の **ItemTypeLinkedList** クラスの使い方はどうなるでしょう。

```
//商品種類リストインスタンスを生成する
ItemTypeLinkedList itemTypeLinkedList = new ItemTypeLinkedList();

//商品種類を追加する
itemTypeLinkedList.add(new ItemType(1001,"コーラ"));
itemTypeLinkedList.add(new ItemType(1002,"ソーダ"));
itemTypeLinkedList.add(new ItemType(1003,”お茶”));
itemTypeLinkedList.first = null;

//商品種類を表示する
itemTypeLinkedList.display();
```

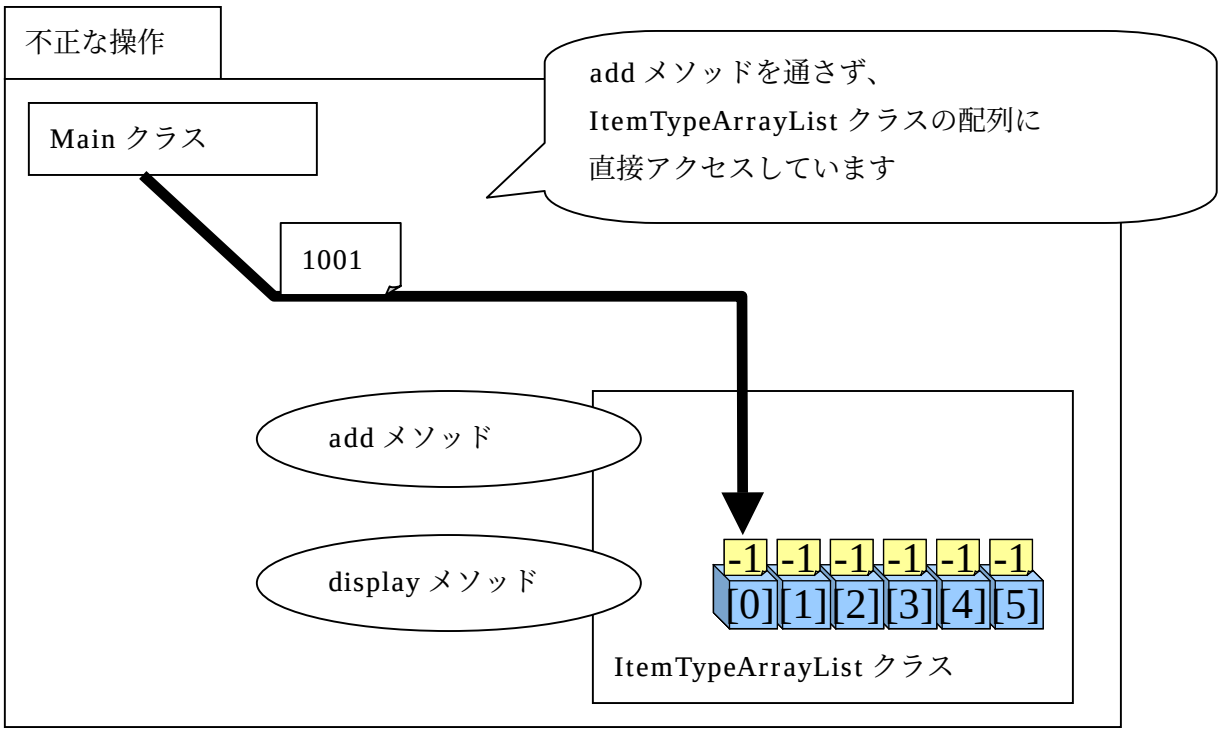
このプログラムを実行したらどんな結果が起こるでしょう？

-
-
-

クラスの属性に対する正規の操作とは、以下のようなイメージです。

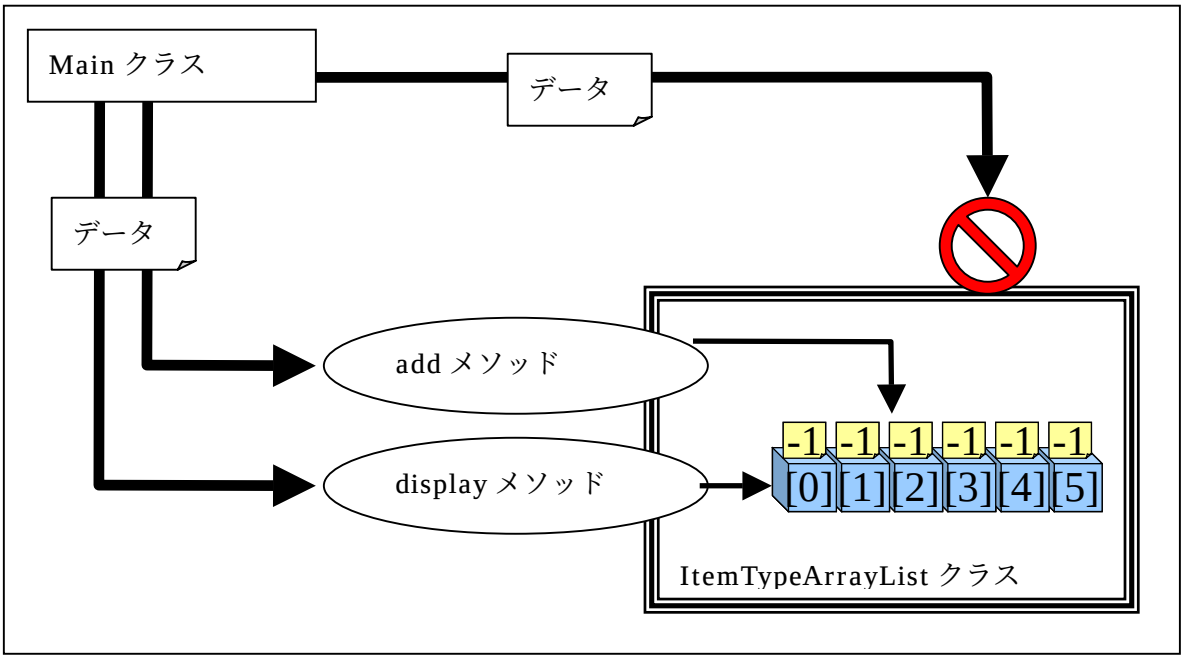


これに対して、不正な操作とは以下のようなイメージです。



6.2.2.　不正な操作を防ぐ

不正な操作を防ぐためには、データのカプセル化をすることが非常に有効です。カプセル化を行うと、以下のようにデータの直接参照、不正な操作を防ぐことができます。



カプセル化によって、どんなメリットがあるでしょうか？

-
-
-

6.2.3.　Java におけるカプセル化

Java においてデータのカプセル化を行うためには、「private」アクセス修飾子を使います。**ItemTypeArrayList** クラスの配列データをカプセル化すると、以下のようになります。

```
public    ItemType[] itemTypeArray;//商品種類を保存する配列
```

⇩

```
private   ItemType[] itemTypeArray;//商品種類を保存する配列
```

このように ItemTypeArrayList クラスの配列データに private を付けると、外部クラスから配列データにアクセスができなくなります。その証拠に、以下のプログラムをコンパイルするとコンパイルエラーが出るようになります。

```
//商品種類リストインスタンスを生成する
ItemTypeArrayList itemTypeArrayList = new ItemTypeArrayList();

//商品種類を追加する
itemTypeArrayList.add(new ItemType(1001,"コーラ"));
itemTypeArrayList.add(new ItemType(1002,"ソーダ"));
itemTypeArrayList.itemTypeArray[0] = null;
```

→ コンパイル エラー！

①.Java で使えるアクセス修飾子

Java ではアクセスを制御するための「アクセス修飾子」が４種類あります。
本講座で利用するアクセス修飾子は次の２つです。

修飾子	利用例	
public	**public int id;**	全てのクラスから参照可能になります。
private	**private int id;**	そのクラス内でのみ参照可能になります。

本講座では利用しませんが、参考にその他のアクセス修飾子を載せておきます。

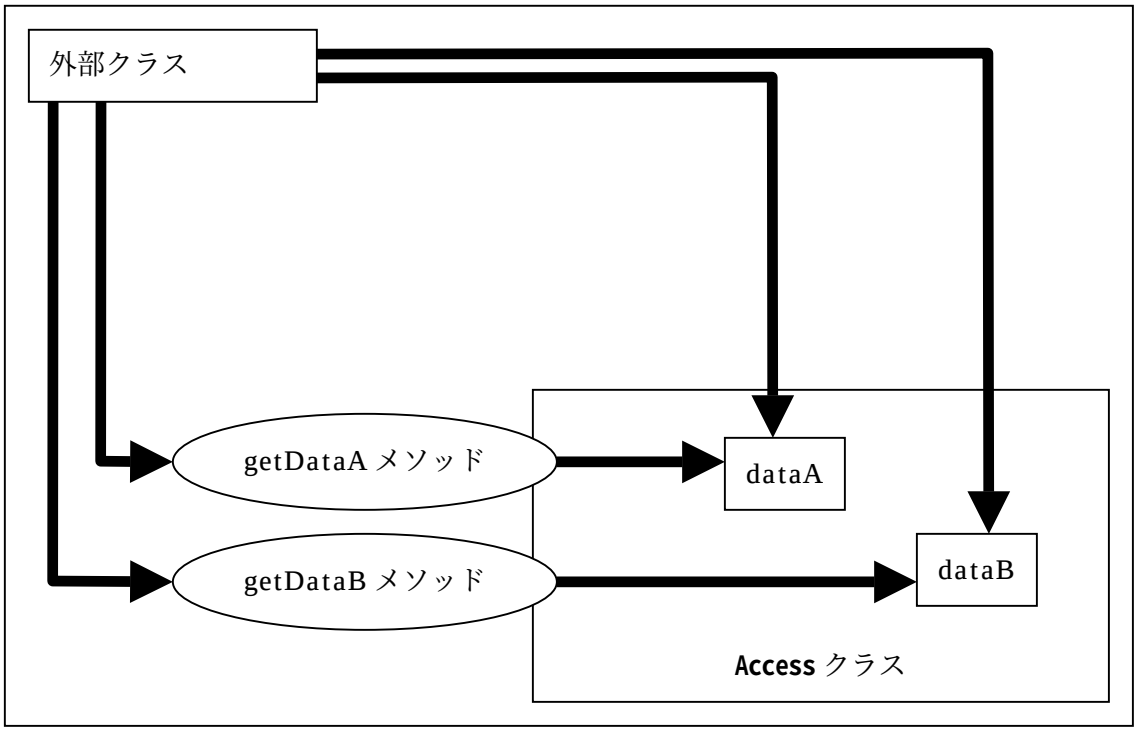
修飾子	利用例	
付けない	**int id;**	パッケージ内でのみ参照可能になります。
protected	**protected int id;**	パッケージ内とそのクラス、サブクラス内でのみ参照可能になります。

※サブクラスに関しては、次回に説明します。
※パッケージに関しては、本講座では取り扱いませんので各自調べてください。

＜考えよう！＞アクセス修飾子を理解できたか、簡単なテスト

```
/**
* アクセス修飾子を解説するためのクラス
*/
public class Access{
  public int dataA;
  private int dataB;

  private int getDataA(){
    return dataA;
  }

  public int getDataB(){
    return dataB;
  }
}
```

このようなクラスがあったとします。その場合、メソッドやデータに対するアクセスの矢印が以下のようになりますが、実際はアクセスが禁止されているはずの矢印が２本あります。どの矢印がアクセスできないでしょうか？



6.2.4.　商品種類クラスのカプセル化

クラスの変数は全て private にして、変数にアクセスする場合はメソッドを介して行うようにしました。

例題 6-6：カプセル化(ItemType.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 6-6：カプセル化
4:      * 商品種類をリストに追加するプログラム
5:      *
6:      * 商品種類クラス
7:      */
8:      public class ItemType {
9:
10:     private int id;                 //商品番号
11:     private String name;            //商品名
12:     private 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:       id = newID;
19:       name = newName;
20:       price = newPrice;
21:     }
22:
23:     /**
24:     * 商品 ID を取得する
25:     */
26:     public int getId() {
27:       return id;
28:     }
29:
30:     /**
31:     * 商品名を取得する
32:     */
33:     public String getName() {
34:       return name;
35:     }
36:
37:     /**
38:     * 価格を取得する
39:     */
40:     public int getPrice() {
41:       return price;
42:     }
43:
```

private により、
直接的な参照の禁止

コンストラクタで
id と name と price を設定
したらもうデータ変更がで
きない仕様になっています

public メソッドを介
してデータ参照が可
能

例題 6-6：カプセル化(ItemTypeArrayList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 6-6：カプセル化
4:      * 商品種類をリストに追加するプログラム
5:      *
6:      * 商品種類配列リストクラス
7:      */
8:      public class ItemTypeArrayList {
9:
10:       private ItemType[] itemTypeArray = new ItemType[10];  //商品種類を保存するための
配列
11:
12:       /**
13:       * 商品種類を追加する
14:       */
15:       public void add(ItemType addItemType){
16:         //商品種類が入っていない箱を探す
17:         for(int i=0;i<10;i++){
18:           if(itemTypeArray[i] == null){//入っていない
19:             itemTypeArray[i] = addItemType;//書き込む
20:             break;
21:           }
22:         }
23:       }
24:
25:       /**
26:       * 商品種類リストを表示する
27:       */
28:       public void display(){
29:         for(int i=0;i<10;i++){
30:           if(itemTypeArray[i] != null){//商品種類が入っている
31:
System.out.println(itemTypeArray[i].getId()+":"+itemTypeArray[i].getName()+":"+itemTypeA
rray[i].getPrice()+"は販売中です");
32:           }
33:         }
34:       }
35:
36:     }
```

①.設定と取得を分ける

代入できなくなるのではないか？そんな心配はありません。ItemType クラスにおいて、外部から商品名を変更できる仕様にしたい場合は以下のように書きます。

```
/**
* 名前を設定するメソッド
*/
public void setName(String newName){
  name = newName;
}
```

＜考えよう！＞わざわざカプセル化し、メソッドを分ける理由

```
//商品種類クラス
public class ItemType{
  public String name; // 商品名
}
```

VS

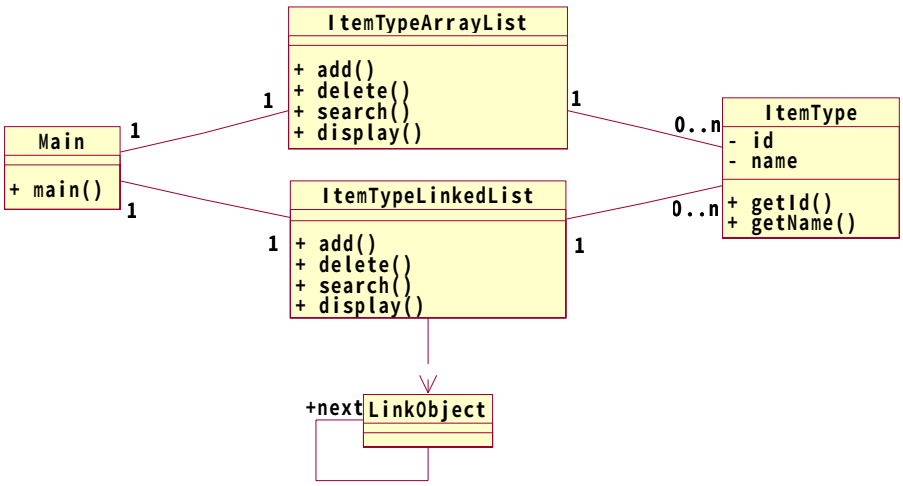
```
//商品種類クラス
public class ItemType{
  private String name; // 商品名

  //名前を得る
  public String getName(){
    return name;
  }

  //名前を設定する
  public void setName(String newName){
    name = newName;
  }
}
```

6.3.　クラスの構造を図解する――クラス図

ここまでで **Main、ItemType、ItemTypeLinkedList、ItemTypeArrayList、LinkObject** など、たくさんのクラスが登場するにしたがってクラスの構成が複雑になってきました。そこでクラスの構成を見やすくするために、図解する技術が必要になってきました。それがクラス図です。

6.3.1.　クラス図とは

クラス図は、クラス構造を分かりやすくするためのものです。以下にここまでで登場したクラスの構造を表したクラス図を示します。

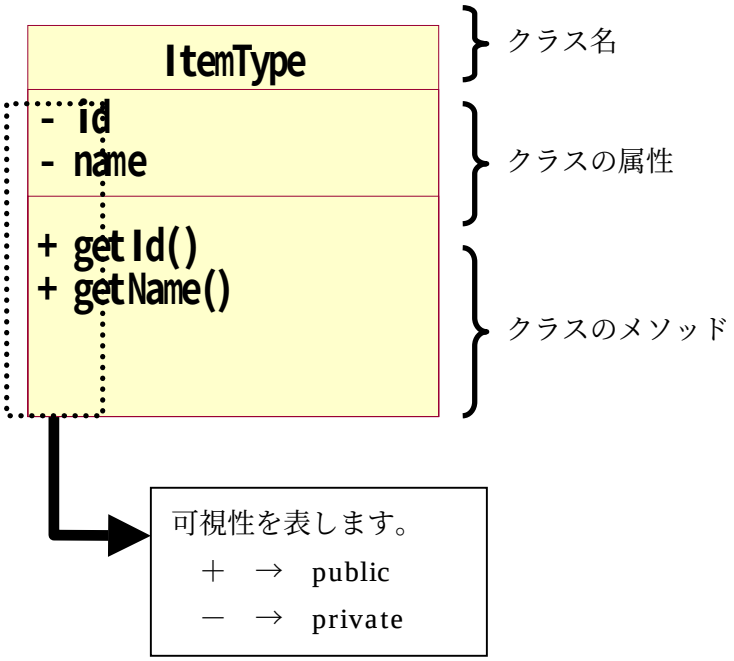


クラス図は、クラスを示す四角形と、クラスとクラスの関係を表す線（矢印や点線になったりする）で構成されます。

6.3.2.　記法

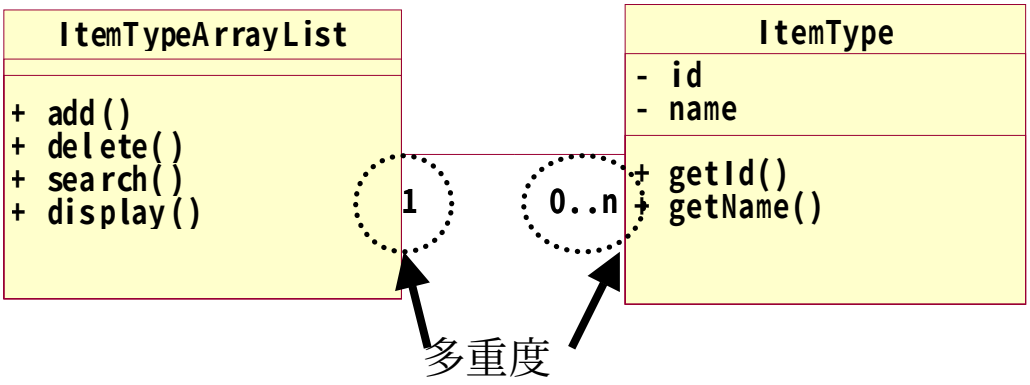
①.クラスを表現する

クラスは四角形で表現します。以下にクラスの例を示します。



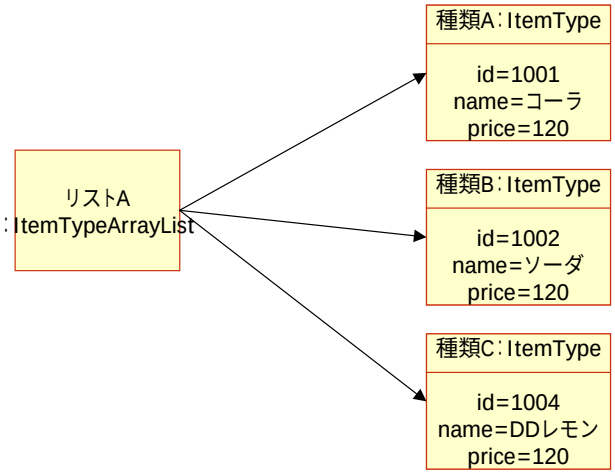
②.クラス間の関係を表現する

あるクラス（のインスタンス）が他のクラス（のインスタンス）の参照を持っている時、そのクラス間には「関係」があるとして線が引かれます。例えば ItemTypeArrayList クラス（のインスタンス）は ItemType クラス（のインスタンス）を配列として持っているので、両者の間には関係があります。



ここで「多重度」を使うと、相手クラス（のインスタンス）の参照を何個持っているかを表現することができます。例えばこの図では「ItemTypeArrayList クラス（のインスタンス）は、ItemType クラス（のインスタンス）の参照を１～ｎ個持っている」「ItemType クラス（のインスタンス）は ItemTypeArrayList クラス（のインスタンス）の参照を１個持っている」という事が表現されています。

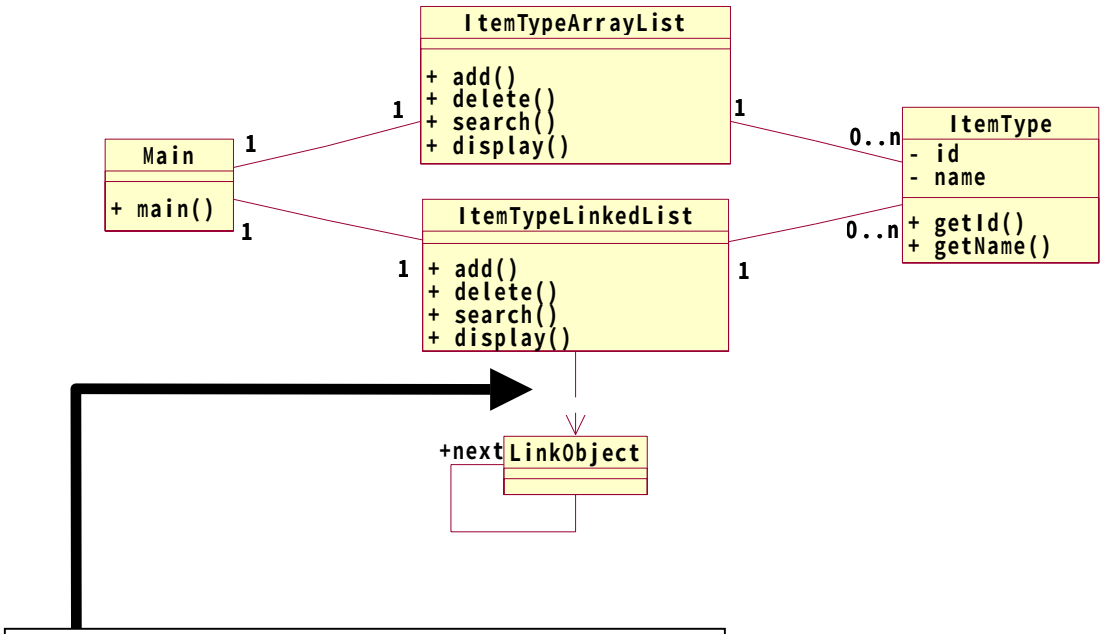
ItemTypeArrayList クラスと ItemType クラスの関係をインスタンス図で書くと、以下のようになります。



この図から、ItemTypeArrayList クラス（のインスタンス）と ItemType クラス（のインスタンス）は「１対ｎの関係」が成り立っていることがわかります。

6.3.3.　今回までのクラス図

以上から、今回まで作ったプログラムのクラスの関係をクラス図で表すと、以下のようになります。



Tips：依存

クラス図で点線の矢印は、依存する関係を表します。

ここでは「ItemTypeLinkedList クラスは、LinkObject クラスに依存します」という意味になります。つまり LinkObject の実装を変えると、その影響が ItemTypeLinkedList クラスにも及ぶことになります。

練習問題

＜記述問題＞

☆記述問題 6-1

　データ構造とアルゴリズムを結合する利点を述べよ。また、データ構造とアルゴリズムを結合してはならない場合を考えよ。

☆記述問題 6-2

　データをカプセル化する利点を簡潔に述べよ。

＜プログラム問題＞

☆プログラム問題 6-1

　プログラム問題 4-2、プログラム問題 5-1 で作ったプログラムのデータ構造とアルゴリズムを結合し、ItemTypeArrayList、ItemTypeLinkedList クラスを完成させよ。

- プログラム仕様

1　コーラ、ソーダ、お茶の順番に登録する 2　ソーダを検索する 3　ソーダを削除する 4　ソーダを検索する 5　商品種類を表示する 上記の仕事を配列と連結リストで 2 回行う

- クラス図