オブジェクト指向哲学
～入門編～

第12回　　汎用的なリストと JavaAPI

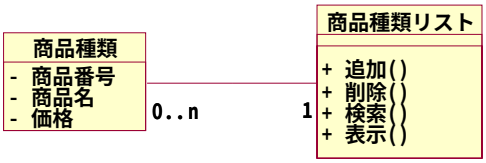
～クラスの再利用について考えよう～

学習目標

- クラスの再利用について議論できる
  - 汎用的なクラスとはどういうものか説明できる
  - 汎用的なクラスを作るために、効果的に継承を利用することを議論できる
  - 継承と委譲による設計の違いをクラスの再利用という側面から説明できる
- 簡単なコレクション JavaAPI を利用したプログラムが書ける
  - クラスのキャストを正しく使ったプログラムが書ける
  - Object 型の存在を説明できる

12.1. 汎用的なリスト

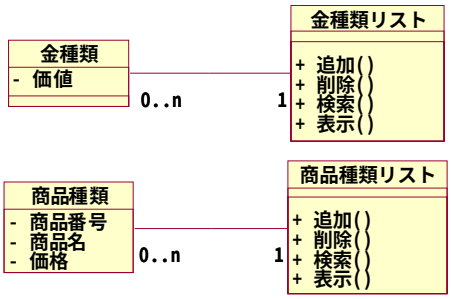
リストによる 1 対他の関連の実装方法について、「商品種類」と「商品種類リスト」という題材を用いて様々な方法を考えてきました。



このような 1 対多の関連を実装するケースは良くあるケースなので、これからのプログラミングを楽にするために、今回は汎用的なリストをつくり、再利用することを考えます。

12.1.1. 今回考える題材

今回考える題材はおなじみの「商品種類」とそのリストの他に、似たようなケースとして「金種類」を管理しなければならないケースを題材とします。これらはほとんど同じ構造をしています。



12.1.2. 再利用を考える

次のページに、これらの題材のソースコードを示します。

よくみると商品種類リストも金種類リストもほとんど同じ重複コードになっています。しかし、少し工夫しないとこれらの重複コードをなくし、再利用できるリストを作ることはできません。問題の分析をしていきましょう。

＜考えよう！＞2 つのプログラムの異なる部分に注目して、何故重複コードになってしまうのか考えよう

例題 12-1：クラスの再利用を考える題材(ItemTypeList.java)

display()メソッドと search()メソッドは削ってある

```
1:    /**
2:    * オブジェクト指向哲学　入門編
3:    * 例題 12-1：クラスの再利用を考える題材
4:    * 商品種類と金種類をリストに追加するプログラム
5:    *
6:    * 商品種類リストクラス
7:    */
8:    public class ItemTypeList {
9:
10:     private int ARRAY_SIZE = 20;         //配列の大きさ
11:     private ItemType[] itemTypeArray;    //商品種類を保存する配列
12:     private int size = 0;                //現在入っている要素数
13:
14:     /**
15:     * コンストラクタ
16:     */
17:     public ItemTypeList() {
18:       itemTypeArray = new ItemType[ARRAY_SIZE];//配列を初期化する
19:     }
20:
21:     /**
22:     * リストに商品種類を追加する
23:     */
24:     public void add(ItemType newItemType){
25:       itemTypeArray[size] = newItemType;
26:       size++;
27:     }
28:
29:     /**
30:     * 指定された商品種類をリストから削除する
31:     */
32:     public void remove(int deleteID){
33:       int i=0;//ループの回数を保存する
34:       for(i=0;i<size;i++){
35:         if(itemTypeArray[i].getId() == deleteID){//見つかった
36:           itemTypeArray[i] = null;//見つかったら、削除する（実は不要）
37:           break;
38:         }
39:       }
40:
41:       //残りの要素をシフトする
42:       for(;i<size-1;i++){
43:         itemTypeArray[i] = itemTypeArray[i+1];
44:       }
45:
46:       size--;
47:     }
48:   }
```

例題 12-1：クラスの再利用を考える題材(MoneyTypeList.java)

display()メソッドと search()メソッドは削ってある

```
1:    /**
2:    * オブジェクト指向哲学　入門編
3:    * 例題 12-1：クラスの再利用を考える題材
4:    * 商品種類と金種類をリストに追加するプログラム
5:    *
6:    * 金種類リストクラス
7:    */
8:    public class MoneyTypeList {
9:
10:     private int ARRAY_SIZE = 20;         //配列の大きさ
11:     private MoneyType[] moneyTypeArray;   //金種類を保存する配列
12:     private int size = 0;                //現在入っている要素数
13:
14:     /**
15:     * コンストラクタ
16:     */
17:     public MoneyTypeList() {
18:       moneyTypeArray = new MoneyType[ARRAY_SIZE];//配列を初期化する
19:     }
20:
21:     /**
22:     * リストに金種類を追加する
23:     */
24:     public void add(MoneyType newMoneyType){
25:       moneyTypeArray[size] = newMoneyType;
26:       size++;
27:     }
28:
29:     /**
30:     * 指定された金種類をリストから削除する
31:     */
32:     public void remove(int deleteID){
33:       int i=0;//ループの回数を保存する
34:       for(i=0;i<size;i++){
35:         if(moneyTypeArray[i].getId() == deleteID){//見つかった
36:           moneyTypeArray[i] = null;//見つかったら、削除する（実は不要）
37:           break;
38:         }
39:       }
40:
41:       //残りの要素をシフトする
42:       for(;i<size-1;i++){
43:         moneyTypeArray[i] = moneyTypeArray[i+1];
44:       }
45:
46:       size--;
47:     }
48:   }
```

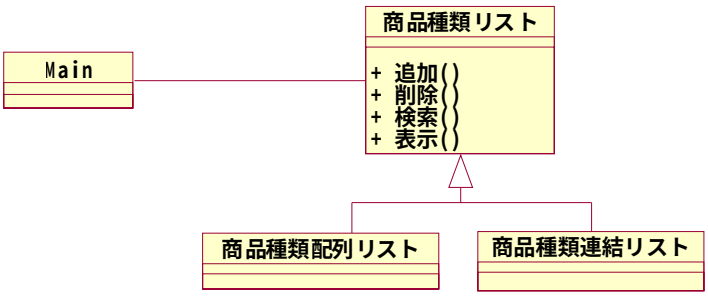
①.問題の分析

- プログラムが異なる点
  - ➢
  - ➢

②.型を同じにするためには

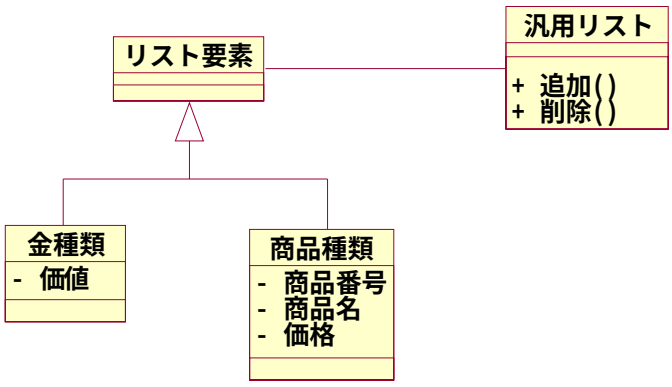
型を同じにするために、「継承」の考え方を利用します。

第 7 回で「継承」によって、「プログラムの意味」に注目したクラスを作ることができることを学習してきました。



同じように、「商品種類」、「金種類」という二つの型（クラス）をリストの立場になって考えてみましょう

「プログラムの意味」をリストの立場から考えた時、2 つには、「リストに格納される要素」という共通の意味があることが分かります。共通の意味があるので、継承を適用することができます。これで汎用的なリストが作れそうです。



12.1.3.　汎用的なリストの実装

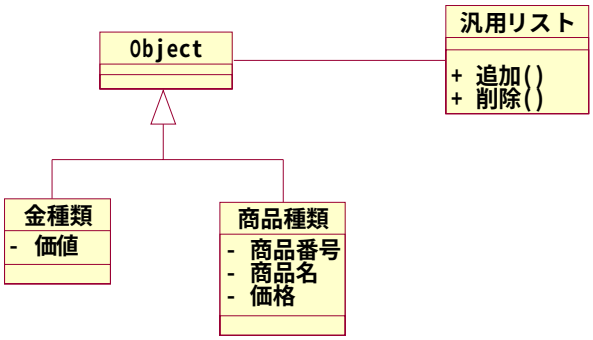
①.Object 型を利用する

Java では、

すべてのクラスは自動的に Object クラスを継承する

という決まりがあります。

本来は、extends と書かなければ継承ができないのですが、何も書かなければ Object クラスを継承するという暗黙の約束があります。つまり、すべてのクラスは Object クラスを継承しているのです。そのため、今回汎用的なリストを作るに当たって、「リスト要素」を Object クラスと考えて実装します。



「List 要素」というクラスを作って実装してもかまわないのですが、Object クラスを使うと、

●すべてのクラスのインスタンスを格納できるリストができる

という利点があるので、こちらを採用することにします。

②.汎用的なリスト

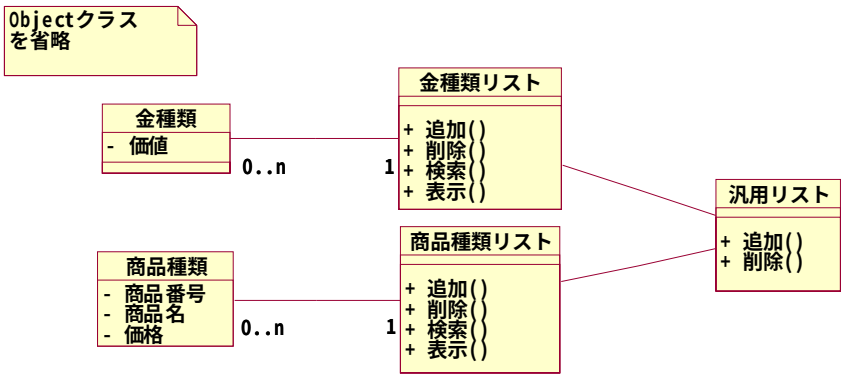
汎用的なリスト(ObjectList クラス)のソースコードです。

例題 12-2：汎用リストを作る(ObjectList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 12-2：汎用リストを作る
4:      * 商品種類と金種類をリストに追加するプログラム
5:      *
6:      * 汎用リストクラス
7:      */
8:      public class ObjectList {
9:
10:       private int ARRAY_SIZE = 20;       //配列の大きさ
11:       private Object[] objectArray;      //Object を保存する配列
12:       private int size;                  //要素数
13:
14:       /**
15:       * コンストラクタ
16:       */
17:       public ObjectList() {
18:         objectArray = new Object[ARRAY_SIZE];//配列を初期化する
19:       }
20:
21:       /**
22:       * リストに Object を追加する
23:       */
24:       public void add(Object newObject){
25:         objectArray[size] = newObject;
26:         size++;
27:       }
28:
29:       /**
30:       * 指定された Object をリストから削除する
31:       */
32:       public void remove(Object object){
33:         int i=0;//ループの回数を保存する
34:         for(i=0;i<size;i++){
35:           if(objectArray[i] == object){//見つかった
36:             objectArray[i] = null;//見つかったら、削除する（実は不要）
37:             break;
38:           }
39:         }
40:
41:         //残りの要素をシフトする
42:         for(;i<size-1;i++){
43:           objectArray[i] = objectArray[i+1];
44:         }
45:
```

```
46:         size--;
47:       }
48:
49:       /**
50:       * リストの要素数を取得する
51:       */
52:       public int size(){
53:         return size;
54:       }
55:
56:       /**
57:       * index 番目の要素を取得する
58:       */
59:       public Object get(int index){
60:         return objectArray[index];
61:       }
62:
63:     }
```

例題に挙げた「商品種類」、「金種類」クラスがこの汎用的なリストを利用する時の、クラス図を示します。



次に、重複コードが除かれた商品種類リストのソースコードを示します。

例題 12-2：汎用リストを作る(ItemTypeList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 12-2：汎用リストを作る
4:      * 商品種類と金種類をリストに追加するプログラム
5:      *
6:      * 商品種類リストクラス
7:      */
8:      public class ItemTypeList {
9:
10:       private ObjectList itemTypeList;   //商品種類を格納するための汎用リスト
11:
12:       /**
13:       * コンストラクタ
14:       */
15:       public ItemTypeList() {
16:         itemTypeList = new ObjectList(); //商品種類を格納するリストを初期化する
17:       }
18:
19:       /**
20:       * リストに商品種類を追加する
21:       */
22:       public void add(ItemType newItemType){
23:         itemTypeList.add(newItemType);
24:       }
25:
26:       /**
27:       * 指定された商品種類をリストから削除する
28:       */
29:       public void remove(int deleteID){
30:         ItemType deleteItemType = search(deleteID);
31:         itemTypeList.remove(deleteItemType);
32:       }
33:
34:       /**
35:       * 指定された商品番号を持つ商品種類を検索する
36:       */
37:       public ItemType search(int searchID){
38:         int len = itemTypeList.size();   //リストの大きさ
39:
40:         for(int i=0;i<len;i++){
41:           ItemType itemType = (ItemType)itemTypeList.get(i);//リストから要素を取り出し
て、商品種類にキャスト
42:           if(itemType.getId() == searchID){
43:             return itemType;//見つかった
44:           }
45:         }
46:
47:         //見つからなかった
48:         return null;
```

```
49:       }
50:
51:       /**
52:       * 商品種類を表示する
53:       */
54:       public void display(){
55:         int len = itemTypeList.size();//リストの大きさ
56:
57:         for(int i=0;i<len;i++){
58:           ItemType itemType = (ItemType)itemTypeList.get(i);//リストから要素を取り出し
て、商品種類にキャスト
59:
System.out.println(itemType.getId()+":"+itemType.getName()+":"+itemType.getPrice()+"   円
");//商品種類を表示
60:         }
61:       }
62:     }
```

12.1.4.　キャスト

①.検索に対応する

商品種類リストや金種類リストでは検索や表示も行われます。しかし、それらのメソッドを汎用化することはできません。

＜考えよう！＞検索や表示の汎用化ができない理由

●
●
●

商品種類リストや金種類リストでの検索や表示に対応するために、汎用リストでは以下のメソッドが追加します。

- public int size()
- public Object get(int index)

例題 12-2(ItemTypeList.java)では、商品種類リストや金種類リストの検索や表示メソッドが、これらの汎用リストが提供するメソッドを使って書き直されています。

②.型変換（キャスト）

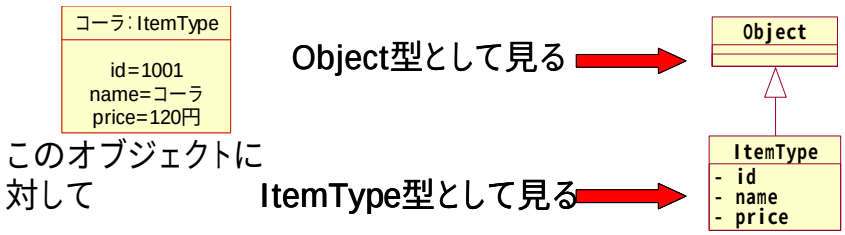
(1)キャスト

例題 12-2(ItemTypeList.java)検索メソッドの 41 行目に注目してください。

```
ItemType itemType = (ItemType)itemTypeList.get(i);
```

このコードは、汎用リストから取得したインスタンスを Object 型から ItemType 型に変換しているものです。

汎用リストの get(index)メソッドは返り値が Object 型です。しかし、欲しい要素は ItemType 型で、そもそも返り値として受け取ったインスタンスは ItemType クラスのインスタンスです。なぜなら、このクラスの add メソッドでは ItemType クラスのインスタンスを汎用リストに追加しているからです。



型変換といっても、実際にインスタンスの型が変わってしまうわけではなく、見え方が変わるだけです。実体は ItemType 型ですが、汎用リストでは、すべて Object 型として扱わなければならないために、汎用リストからの返り値はどうしても Object 型になってしまうのです。

```
(クラス名)
```

のようにクラス名に括弧を付けて代入することにより、型変換（キャスト）が行われます。

もちろん、ItemType のインスタンスではないのに型変換（例えば Money クラスのインスタンスを ItemType 型に変換）することはできません。そのため、キャストをするクラス間が継承関係になっていることが大前提となります。

(2)自動的に行われるキャスト

また、例題 12-2(ItemTypeList.java)追加メソッドの 23 行目に注目してください。

```
itemTypeList.add(newItemType);
```

汎用リストの追加メソッドは、add(Object object) であり、ItemType 型を引数にとることはできないはずですが、このプログラムは正しく動いています。なぜでしょうか。

この場合は、ItemType クラスは Object クラスを継承しているので、ItemType クラスが Object であることは明らかです。そのため自動的にキャストが行われるのです。

もちろん、以下のように書いても大丈夫です。

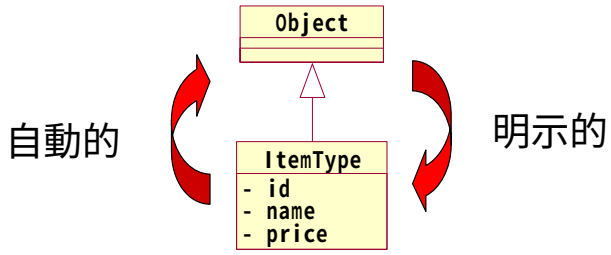
```
itemTypeList.add((Object)newItemType);
```

(3)自動的キャストと明示的キャスト

これまでのキャストの仕組みをまとめると、

- サブクラス→スーパークラスの変換は自動的に行われる
- スーパークラス→サブクラスの変換は明示的に行わなければならない

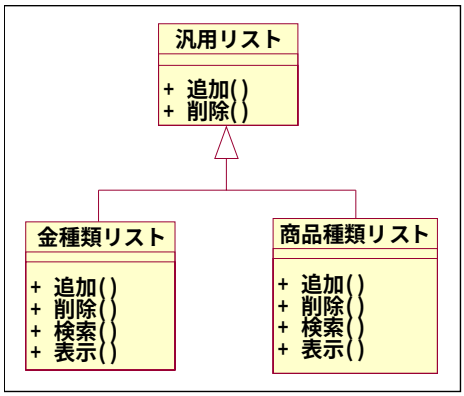
となります。



12.2. 継承 VS 委譲

12.2.1. 継承による設計

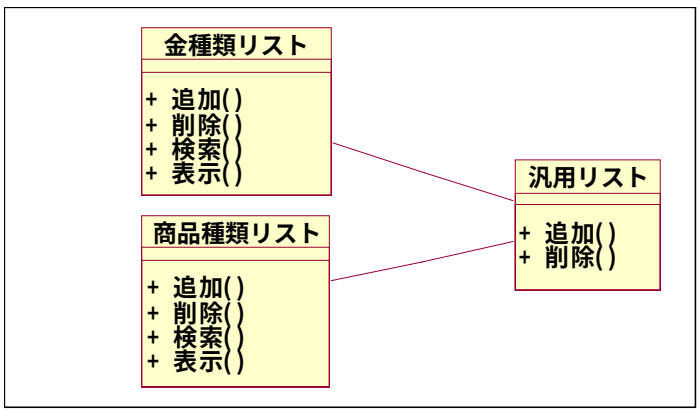
ここまでで汎用的なリストを作ることができました。しかし、なぜ金種類リストや商品種類リストが汎用リストを継承しないのか疑問に思いませんか？



12.2.2. 委譲による設計

例題 12-2 で扱った設計は以下のように金種類リストや商品種類リストが汎用リストを利用して実装されている設計です。

これらは、追加や削除の仕事を汎用リストに任せているので「委譲」による設計と呼ばれています。



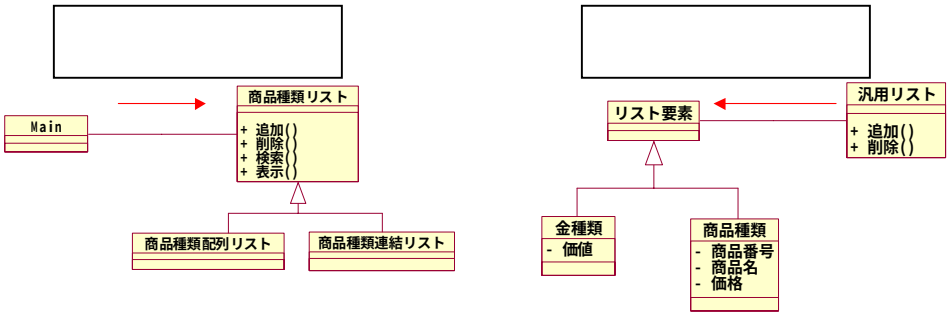
12.2.3.　継承 VS 委譲

どちらの設計も間違いではありません。問題はどちらが優れているかです。

＜議論しよう！＞どちらの設計が優れているか

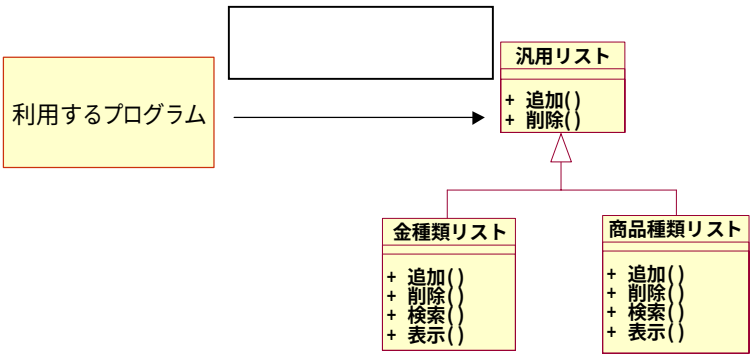
①.継承の利点が生かされる時

継承の利点が生かされる時は、「プログラムの意味」に注目したプログラムが書けることです。



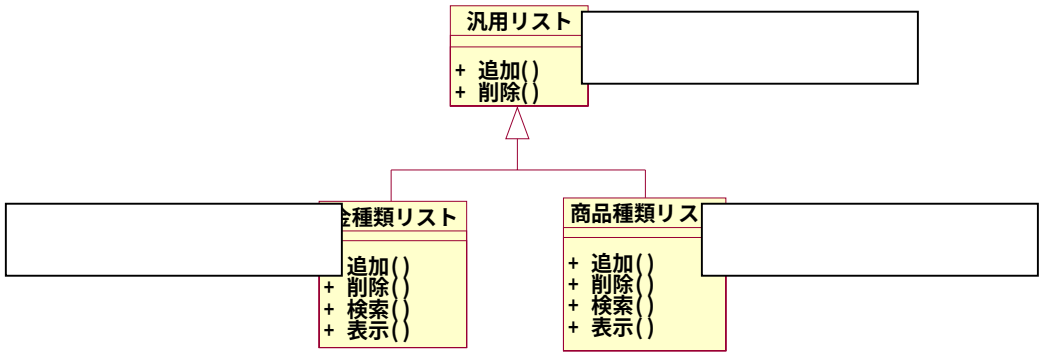
②.継承の利点は生かされるか

金種類リストか、商品種類リストを意識せず、汎用的な List に対してプログラムを書く場合が果たしてあるでしょうか？考えてみましょう。



③.プログラムの意味に注目する

プログラムの意味に注目してみましょう。



④.まとめ

-
-

委譲か継承かは重要な問題です。

継承の利点をよく吟味して適用しましょう。その際に、プログラムの意味に着目するのが非常に重要です。

また、継承の欠点も抑えておきましょう。

- 多重継承できない
- クラスの継承階層を後から変更することは難しい（何故か考えてみましょう）

12.3. JavaAPI を利用する

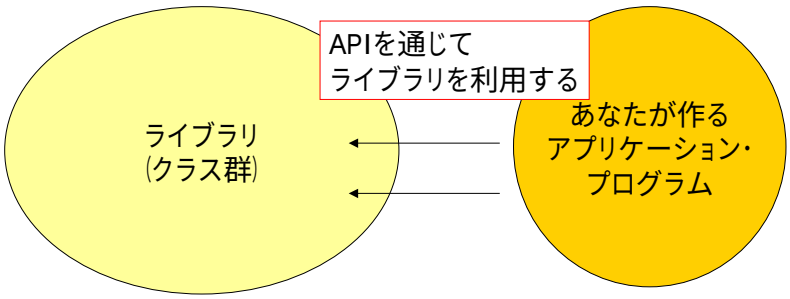
汎用リストのようなクラスはとてもよく使います。プログラムを構成するたびに作るのは面倒ですし、再利用できる汎用リストを作ることが可能でした。自分だけでなく、みんなで使いたいものです。

そのようなことはみんなが考えています。実は Java では誰でも使える汎用リストが提供されています。

リスト以外にも、ファイル読み込みや、GUI を構成するための再利用可能な汎用的なクラスが用意されています。このような、Java が提供してくれる汎用的なクラス群のことを Java クラス・ライブラリといいます。

12.3.1.　JavaAPI

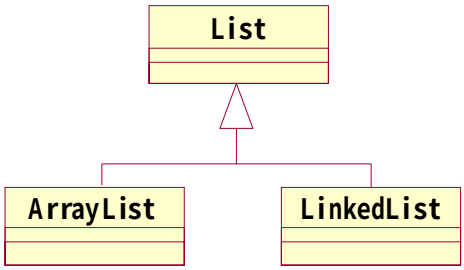
Java クラス・ライブラリは JavaAPI を通して利用します。API とは Application Programming Interface の略で、クラスライブラリを利用するためのインターフェイスです。インターフェイスといってもそんなに難しく考える必要はなく、「クラスの使い方」「メソッドの使い方」と考えればよいでしょう。



12.3.2.　リスト周りの API

今回は、汎用リスト周りの API を使ったプログラミングをしてみましょう。といっても、今まで皆さんが作ってきた汎用リストは、JavaAPI そっくりに作ってきたので、使い方はほとんど一緒です。

①.Java で用意されているリスト



- List(インターフェイス)
- ArrayList
  - 配列で実装されたリスト
- LinkedList
  - 連結リストで実装されたリスト

②.List API

List インターフェイスの主なメソッドは次のようなものです。ArrayList も LinkedList も List インターフェイスを継承していますから、これらのメソッドが実装されています。

void add(Object object)	リストに Object を追加する
void remove(Object object)	リストから Object を削除する
int size()	リストの要素数を得る
Object get(int index)	index 番目の Object を得る

今回作った汎用リストとまったく同じメソッドだということが分かるでしょう。

③.LinkedList API

連結リストクラスにだけ存在する特別なメソッドを紹介します。なぜなら、これらのメソッドを使えば、キューやスタックを容易に作ることができるからです。

void addFirst(Object object)	連結リストの最初に Object を追加する
void addLast(Object object)	連結リストの最後に Object を追加する
void removeLast(Object object)	連結リストの最後から Object を削除する
void removeFirst(Object object)	連結リストの最初から Object を削除する

12.3.3.　JavaAPI を利用したプログラム

JavaAPI さえ知っていれば、プログラミング自体はそう難しくありません。始めに import 文を書くだけで、それらのクラスが使えるようになります。

JavaAPI を使って書き直した商品種類リストのソースを次に示します。

例題 12-3：JavaAPI を利用する(ItemTypeList.java)

```
1:    import java.util.*;//java.util クラスライブラリの利用を宣言する
2:
3:    /**
4:    * オブジェクト指向哲学　入門編
5:    * 例題 12-3：JavaAPI を利用する
6:    * 商品種類と金種類をリストに追加するプログラム
7:    *
8:    * 商品種類リストクラス
9:    */
10:   public class ItemTypeList {
11:
12:     private ArrayList itemTypeList;    //商品種類を格納するための汎用リスト
13:
14:     /**
15:     * コンストラクタ
16:     */
17:     public ItemTypeList() {
18:       itemTypeList = new ArrayList(); //商品種類を格納するリストを初期化する
19:     }
20:
21:     /**
22:     * リストに商品種類を追加する
23:     */
24:     public void add(ItemType newItemType){
25:       itemTypeList.add(newItemType);
26:     }
27:
28:     /**
29:     * 指定された商品種類をリストから削除する
30:     */
31:     public void remove(int deleteID){
32:       ItemType deleteItemType = search(deleteID);
33:       itemTypeList.remove(deleteItemType);
34:     }
35:
36:     /**
37:     * 指定された商品番号を持つ商品種類を検索する
38:     */
39:     public ItemType search(int searchID){
40:       int len = itemTypeList.size();    //リストの大きさ
```

```
41:
42:         for(int i=0;i<len;i++){
43:           ItemType itemType = (ItemType)itemTypeList.get(i);//リストから要素を取り出し
て、商品種類にキャスト
44:           if(itemType.getId() == searchID){
45:             return itemType;//見つかった
46:           }
47:         }
48:
49:         //見つからなかった
50:         return null;
51:       }
52:
53:       /**
54:       * 商品種類を表示する
55:       */
56:       public void display(){
57:         int len = itemTypeList.size();//リストの大きさ
58:
59:         for(int i=0;i<len;i++){
60:           ItemType itemType = (ItemType)itemTypeList.get(i);//リストから要素を取り出し
て、商品種類にキャスト
61:
System.out.println(itemType.getId()+":"+itemType.getName()+":"+itemType.getPrice()+"   円
");//商品種類を表示
62:         }
63:       }
64:     }
```

12.3.4. JavaAPI ドキュメント

JavaAPI を調べるためのドキュメントが http://java.sun.com/j2se/1.3/ja/docs/ja/api/index.html にあります。忘れてしまったメソッドや、未知のクラスを使いたい時は、調べて見ましょう。

練習問題

＜記述問題＞

☆記述問題 12-1

継承と委譲について、それぞれの利点と欠点を自分の言葉で説明せよ。

＜プログラム問題＞

☆プログラム問題 12-1

プログラム問題 11-1 で完成した CUI アプリケーションを、JavaAPI を利用して書き直せ。