オブジェクト指向哲学
～入門編～

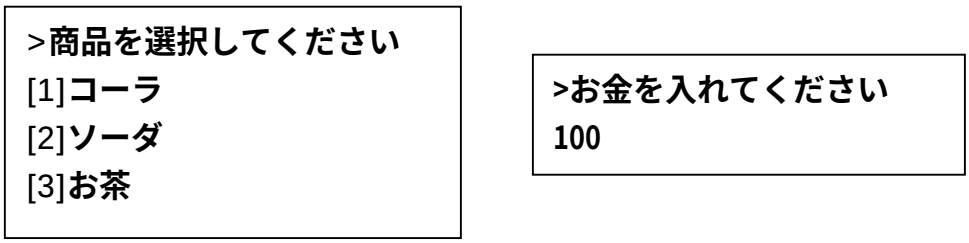
第11回　　アプリケーションの構成

～CUI 自動販売機の完成!～

学習目標

- 簡単なオブジェクト指向アプリケーションが書ける
  - 関連の実装ができる
  - オブジェクトの構造を構成できる
  - シーケンス図が読める

今回は CUI(Character User Interface)による自動販売機アプリケーションを構築します。



11.1. プログラムの仕様を決める

自動販売機といってもみなさんが想像するものは少しづつ異なる可能性があります。アプリケーションを構築する際に、一番重要なのはどのようなアプリケーションを作るのかを決めることです。作るプログラムがどんなサービスを提供するのか決めることを「仕様を決める」といいます。まず、仕様を決めるところから始めましょう。

＜考えよう！＞自動販売機が行う主なサービスを考えてみましょう！

11.1.1.　自動販売機が提供するサービス

今回作る自動販売機が提供するサービスを次のように考えます。

- 管理者に対するサービス
  - 商品種類の管理
  - 商品の在庫管理
- ユーザ（購入する人）に対するサービス
  - 商品の購入

プログラムを書くために、目的を階層構造にしていくことが重要です。（第 1 回を参照）上記の大まかなサービスを基に、さらに細かくプログラムの目的として記述していきましょう。

- 自動販売機を管理する
  - 商品種類を管理する
    - 商品種類を追加する
    - 商品種類を削除する
    - 取り扱っている商品種類を確認する
  - 在庫を管理する
    - 商品を補充する
    - 在庫を確認する

＜考えよう！＞ユーザに対するサービスを目的の階層構造にしてみよう

- 商品を購入する

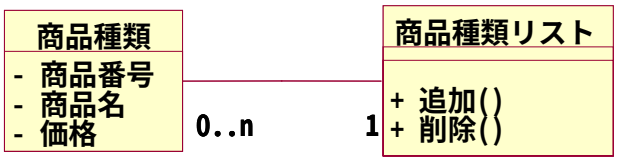
11.2. オブジェクトの構造を考える

11.2.1.　これまで作ったプログラム

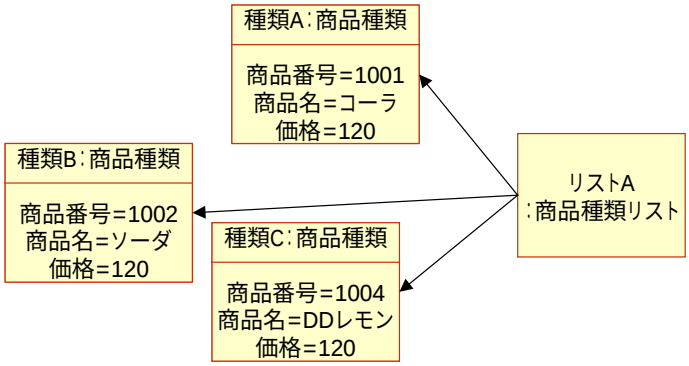
商品種類の管理や、在庫の管理はこれまで作ったプログラムの構造が使えるはずです。これまで作ったプログラムの構造を「クラス/インスタンスの構造」という側面から確認してみましょう。

①.商品種類管理プログラム

第 9 回までに作ってきた、商品種類の管理プログラムの構造をクラス図とインスタンス図で示します。



クラス図

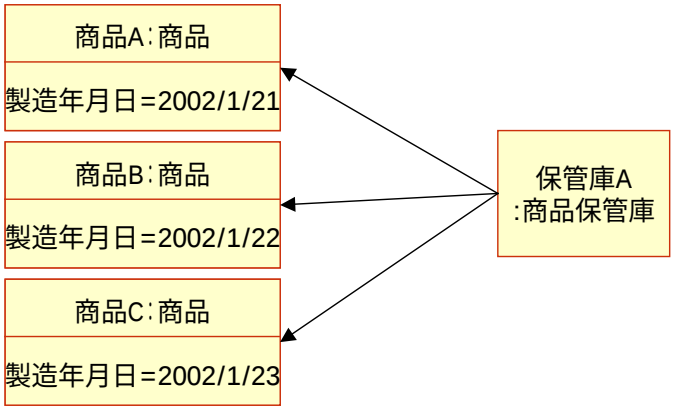


インスタンス図

②.商品種類管理プログラム

第 10 回で作った､商品の管理プログラムの構造をクラス図とインスタンス図で示します。

クラス図



インスタンス図

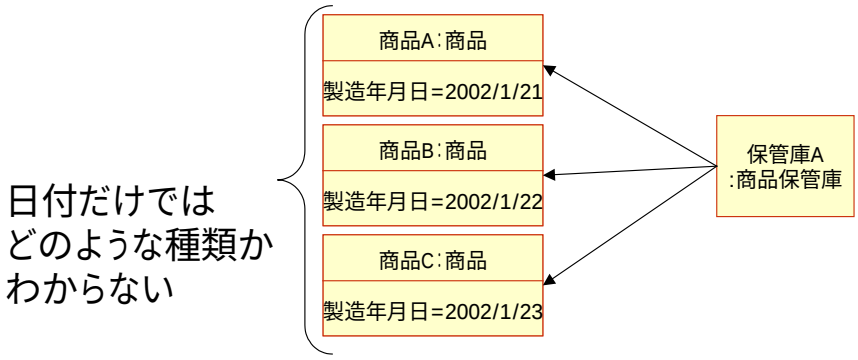
11.2.2.　オブジェクトの構造をつなげるには

2 つのプログラムのオブジェクトの構造をつなげる方法を考えましょう。

①.商品と商品種類の関係

(1)問題点

第 10 回で作ったプログラムの「商品」というクラスは、属性が製造年月日だけだったので、どのような種類かわからないことが問題です。

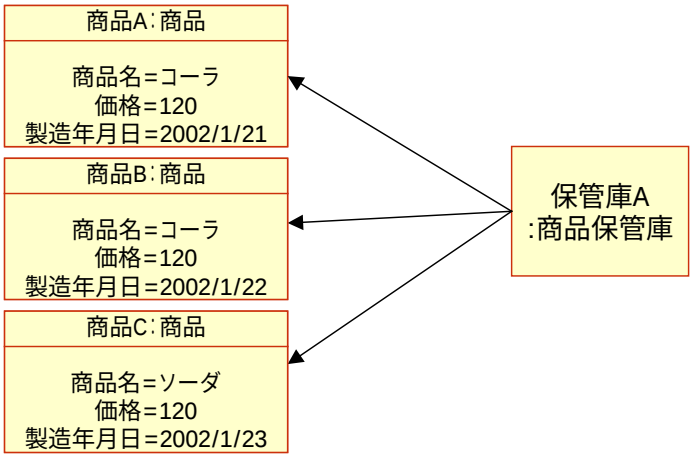


(2)解決方法

この問題の解決方法を考えます。

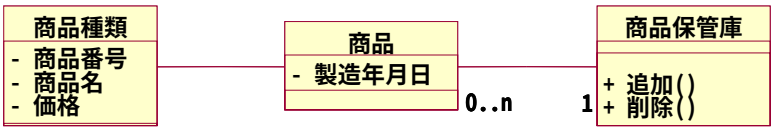
解決案 I：商品に名前を加える

クラス図

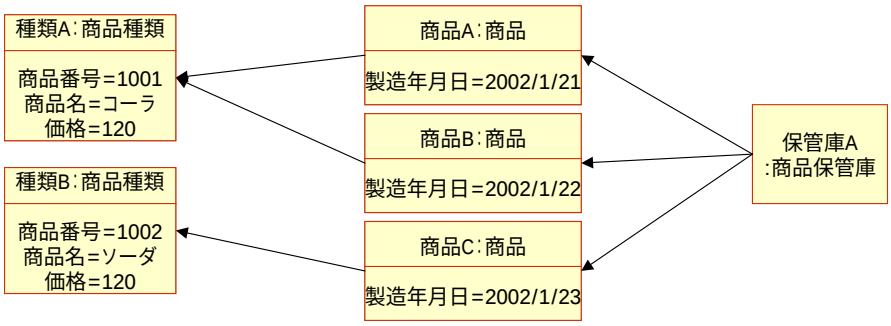


インスタンス図

解決案Ⅱ：商品と商品種類をリンクする



クラス図



インスタンス図

(3)どちらの解決方法が適切か

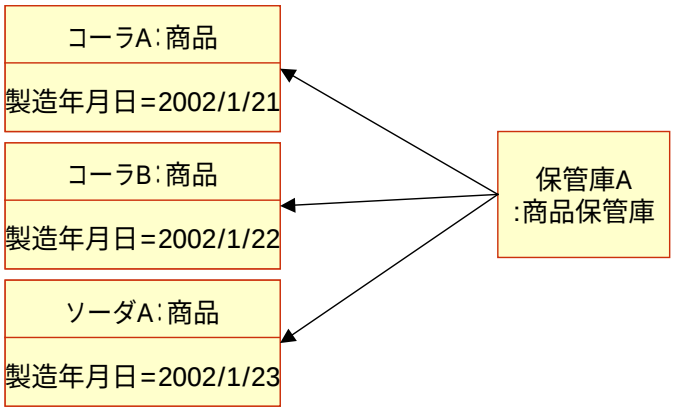
どちらの解決方法が適切なのか、議論してみましょう

- 価格の変更はどちらがしやすいか
- ユーザは「商品」を選ぶのか「商品種類」を選ぶのか
- その種類の商品が売り切れてしまった場合どうなるか

②.商品種類と商品保管庫の関係

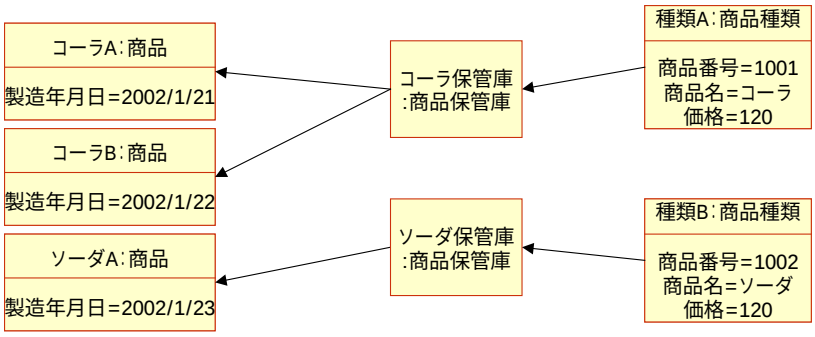
(1)問題点

異なる商品が同じ保管庫に入っていることが問題です。コーラやソーダが一つの保管庫に入っていたら、取り出す時大変です。



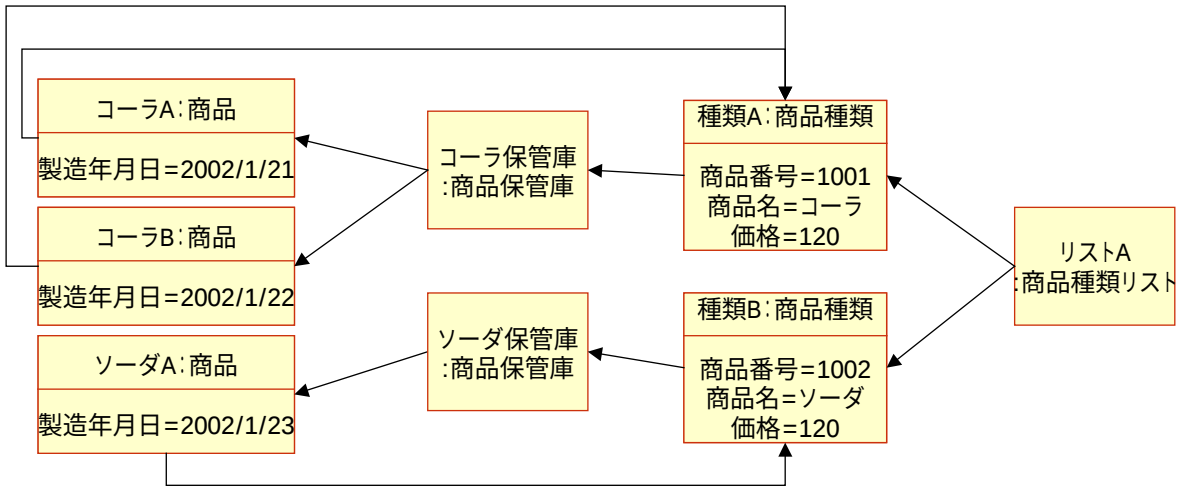
(2)解決方法

解決方法は、種類ごとに保管庫を用意する方法です。つまり、商品種類と保管庫をリンクする方法です。



③.つながった2つのプログラム

つながった 2 つのプログラムのインスタンスの構造をみてみましょう。



＜考えよう＞クラス図を書いてみましょう

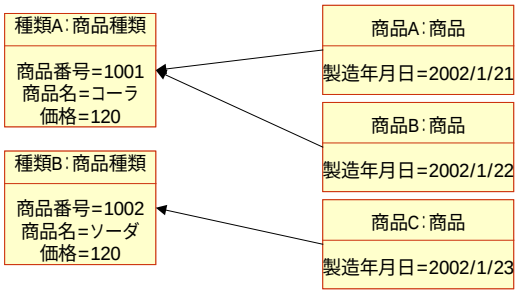
11.2.3.　オブジェクトの構造を構築するための実装

オブジェクトの構造は、プログラム上では参照として実装されます。インスタンス図における矢印がそのまま参照の方向になると考えてください。

ここでは、インスタンス図またはクラス図に従って、プログラム上に参照を配置して、うまく構造を構築できるような仕組みをつくります。

①.商品と商品種類

商品は、商品種類とつなげた（関係をもたせた）ため、参照をもつ必要があります。



インスタンス図

クラス変数に商品種類の参照を宣言し、コンストラクタで参照を設定するようになっています。これは設定メソッドを用意してもかまいません。

また、その商品がどの商品種類なのかを調べることができるように、取得メソッドを追加します。

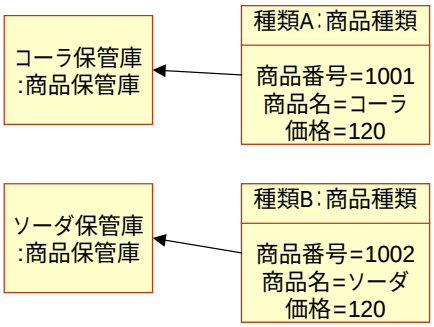
実装例を次ページに示します。

例題 11-1：自動販売機アプリケーションの構築(Item.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 11-1：自動販売機アプリケーションの構築
4:      * 商品種類の管理、商品の管理、商品の販売を行うプログラム
5:      *
6:      * 商品クラス
7:      */
8:      public class Item {
9:
10:       private String date;//製造年月日
11:       private ItemType itemType;//商品に対応する商品種類
12:
13:       /**
14:       * コンストラクタ
15:       */
16:       public Item(String newDate,ItemType newItemType) {
17:         date = newDate;
18:         itemType = newItemType;
19:       }
20:
21:       /**
22:       * 製造年月日を取得する
23:       */
24:       public String getDate() {
25:         return date;
26:       }
27:
28:       /**
29:       * 商品に対応する商品種類を取得する
30:       */
31:       public ItemType getItemType() {
32:         return itemType;
33:       }
34:     }
```

②.商品種類と商品保管庫

商品種類と商品保管庫も①と同様の関係です。



インスタンス図

クラス変数に商品保管庫の参照を宣言しているところ､取得メソッドをまでは①とまったく同じです。異なるのは、コンストラクタで参照を生成するようになっていることです。保管庫を生成して、そのまま参照を持ちつづけることになります。

この方法は①では使えません。何故か考えてみましょう。

例題 11-1:自動販売機アプリケーションの構築(ItemType.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 11-1：自動販売機アプリケーションの構築
4:      * 商品種類の管理、商品の管理、商品の販売を行うプログラム
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:       private ItemStock itemStock;    //この種類の商品だけを格納する商品保管庫
14:
15:       /**
16:       * コンストラクタ
17:       */
18:       public ItemType(int newID,String newName,int newPrice) {
19:         id = newID;
20:         name = newName;
21:         price = newPrice;
22:         itemStock=new ItemStock();
23:       }
24:
```

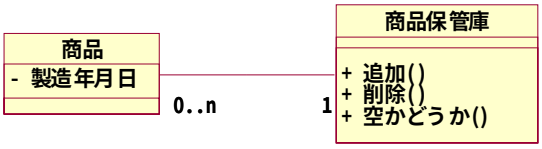
```
25:       /**
26:       * 商品 ID を取得する
27:       */
28:       public int getId() {
29:         return id;
30:       }
31:
32:       /**
33:       * 商品名を取得する
34:       */
35:       public String getName() {
36:         return name;
37:       }
38:
39:       /**
40:       * 価格を取得する
41:       */
42:       public int getPrice() {
43:         return price;
44:       }
45:
46:       /**
47:       * 商品保管庫を取得する
48:       */
49:       public ItemStock getItemStock(){
50:         return itemStock;
51:       }
52:     }
```

③.既につながっている部分

(1)商品と商品保管庫

第 10 回においてすでにキューで実装されていますのでそのまま使えます。

また、アプリケーションにおいては空かどうかを調べるメソッドが必要ですので、追加します。



クラス図

例題 11-1：自動販売機アプリケーションの構築(ItemStock.java)

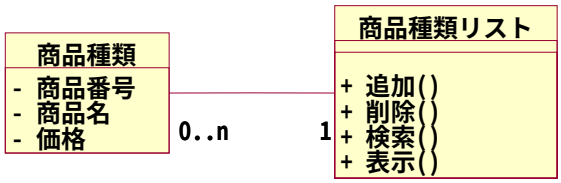
```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 11-1：自動販売機アプリケーションの構築
4:      * 商品種類の管理、商品の管理、商品の販売を行うプログラム
5:      *
6:      * 商品保管庫クラス
7:      * 補充した順に取り出す構造（キュー）になっている
8:      */
9:      public class ItemStock {
10:
11:       private int ARRAY_SIZE = 20;  //配列の大きさ
12:       private int addCursol = 0;    //追加カーソル
13:       private int removeCursol =0;  //削除カーソル
14:       private Item[] itemArray;     //商品を格納する配列
15:
16:       /**
17:       * コンストラクタ
18:       */
19:       public ItemStock() {
20:         itemArray = new Item[ARRAY_SIZE]; //配列を初期化する
21:       }
22:
23:       /**
24:       * 商品を補充する
25:       */
26:       public void supply(Item insertItem){
27:         itemArray[addCursol] = insertItem;//追加カーソルの位置に挿入する
28:         addCursol++; //追加カーソルを右にずらす
29:         if (addCursol >= ARRAY_SIZE){//配列の最後にきたら
30:           addCursol = 0;//ラップアラウンドする
31:         }
32:       }
```

```
33:
34:       /**
35:       * 商品を取り出す
36:       *  （最初に補充されたものを取り出す）
37:       */
38:       public Item takeout(){
39:         if(addCursol == removeCursol){//商品がない
40:           return null;//何も入っていないので null を返す
41:         }
42:         Item removeItem = itemArray[removeCursol];//削除カーソルの商品を取り出す
43:         itemArray[removeCursol]=null;//削除カーソルの商品を削除する
44:         removeCursol++;//削除カーソルを右にずらす
45:         if (removeCursol >= ARRAY_SIZE){//配列の最後にきたら
46:           removeCursol =0;              //ラップアラウンドする
47:         }
48:         return removeItem;//削除した商品を返す
49:       }
50:
51:       /**
52:       * 商品保管庫が空（売り切れ）かどうかを調べる
53:       */
54:       public boolean isEmpty(){
55:         if (addCursol == removeCursol){
56:           //売り切れだった
57:           return true;
58:         }else{
59:           //売り切れではなかった
60:           return false;
61:         }
62:       }
63:
64:       /**
65:       * 商品保管庫の中身を表示する
66:       */
67:       public void display(){
68:         System.out.println("---商品保管庫キューの先頭---");
69:         int displayCursol = removeCursol;//表示のために配列を走査するカーソル
70:         while(displayCursol != addCursol){
71:           System.out.println("商品の製造年月日："+itemArray[displayCursol].getDate());
72:           displayCursol++;//表示カーソルを右にずらす
73:           if(displayCursol >= ARRAY_SIZE){//配列の最後にきたら
74:             displayCursol = 0;//ラップアラウンドする
75:           }
76:         }
77:         System.out.println("---商品保管庫キューの最後尾---");
78:       }
```

(2)商品種類と商品種類リスト

第 9 回までに配列や連結リストを使って実装してきました。

今回使う商品種類リストは第 7 回で実装された配列リストを利用しています。クラス図とソースを示します。



クラス図

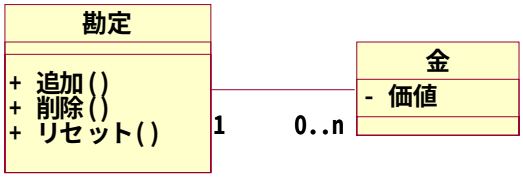
例題 11-1：自動販売機アプリケーションの構築(ItemTypeList.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 11-1：自動販売機アプリケーションの構築
4:      * 商品種類の管理、商品の管理、商品の販売を行うプログラム
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:
49:       /**
50:       * 指定された商品番号を持つ商品種類を検索する
51:       */
52:       public ItemType search(int searchID){
53:         //リニアサーチで商品種類を探す
54:         for(int i=0;i<size;i++){
55:           if(itemTypeArray[i] != null){
56:             if(itemTypeArray[i].getId() == searchID){//見つかった
57:               return itemTypeArray[i];
58:             }
59:           }
60:         }
61:
62:         //見つからなかった
63:         return null;
64:       }
65:
66:       /**
67:       * 商品種類リストを表示する
68:       */
69:       public void display(){
70:         for(int i=0;i<size;i++){
71:           //商品種類を表示
72:
System.out.println(itemTypeArray[i].getId()+":"+itemTypeArray[i].getName()+":"+itemTypeA
rray[i].getPrice()+"円");
73:         }
74:       }
75:     }
```

(3)投入されたお金の管理

第 10 回で使った、スタックを使ってお金を管理する勘定クラスと金クラスがそのまま使えます。



クラス図

ただし、商品の販売が終了した時には、勘定を一旦クリアする必要がありますので、勘定をリセットするメソッドを追加します。

勘定と金クラスのソースを示します。

例題 11-1：自動販売機アプリケーションの構築（Money.java）

```
76:     /**
77:     * オブジェクト指向哲学　入門編
78:     * 例題 11-1：自動販売機アプリケーションの構築
79:     * 商品種類の管理、商品の管理、商品の販売を行うプログラム
80:     *
81:     * 金クラス
82:     */
83:     public class Money {
84:
85:       private int value;  //金の値（単位：円）
86:
87:       /**
88:       * コンストラクタ
89:       */
90:       public Money(int newValue) {
91:         value = newValue;
92:       }
93:
94:       /**
95:       * 値を取得する
96:       */
97:       public int getValue(){
98:         return value;
99:       }
100:    }
```

例題 11-1：自動販売機アプリケーションの構築（Account.java）

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 11-1：自動販売機アプリケーションの構築
4:      * 商品種類の管理、商品の管理、商品の販売を行うプログラム
5:      *
6:      * 勘定クラス
7:      * Undo(やり直し)ができるように、最後に投入された金から取り出す構造（スタック）になっている
8:      */
9:      public class Account {
10:
11:       private int ARRAY_SIZE = 20;     //配列の大きさ
12:       private Money[] moneyArray;      //金を保存する配列
13:       private int cursol = 0;          //追加・削除する際に使うカーソル
14:
15:       /**
16:       * コンストラクタ
17:       */
18:       public Account() {
19:         moneyArray = new Money[ARRAY_SIZE]; //配列を初期化する
20:       }
21:
22:       /**
23:       * 金を投入する
24:       */
25:       public void insert(Money insertMoney){
26:         moneyArray[cursol] = insertMoney;//カーソル位置に金を投入する
27:         cursol++;//カーソルをひとつ進める
28:       }
29:
30:       /**
31:       * 投入した金を undo(やり直し)する
32:       */
33:       public Money undo(){
34:         if(cursol <= 0){//取り出す金がない
35:           return null;//取り出せるものがないので null を返す
36:         }
37:         cursol--;//カーソルをひとつ戻す
38:         Money money = moneyArray[cursol];//カーソル位置にある金を保存しておく
39:         moneyArray[cursol]=null;//削除する
40:
41:         return money;//削除した金を返す
42:       }
43:
44:       /**
45:       * 合計金額を計算する
46:       */
47:       public int getAmount(){
```

```
48:         int amount=0;
49:         for(int i=0; i<cursol; i++){
50:           amount = amount + moneyArray[i].getValue();//合計金額を加算
51:         }
52:         return amount;
53:       }
54:
55:       /**
56:       * リセットする
57:       */
58:       public void reset(){
59:         cursol = 0;//カーソルを初期化
60:         moneyArray = new Money[ARRAY_SIZE];//配列を初期化
61:       }
62:
63:       /**
64:       * 勘定の中身を表示する
65:       */
66:       public void display(){
67:         System.out.println("---勘定スタックの先頭---");
68:         int displayCursol = cursol-1;//表示のために配列を走査するカーソル
69:
70:         while(displayCursol >= 0){
71:           System.out.println("金の価値："+ moneyArray[displayCursol].getValue());
72:           displayCursol--;//表示用カーソルを戻す
73:         }
74:         System.out.println("---勘定スタックの最後尾---");
75:       }
76:
77:     }
```

11.2.4. クラスの関係の実装 まとめ

- 1対1の関係の実装
  - 参照で実装する
  - コンストラクタで設定、または生成する
    - 設定メソッドを用意する方法もある
  - 取得メソッドを実装する
- 1対多の関係の実装
  - 配列や、連結リストを使って実装
    - キュー、スタックなどその他いろいろなデータ構造が使える
  - 追加、削除、検索メソッドを実装する

11.3. オブジェクトの構造を構築、操作する

オブジェクトの構造を構築するための仕組みは完成したので、構造を構築し、操作するプログラムをを書いてみましょう。

11.3.1.　オブジェクトの構造を構築･操作するプログラム

このプログラムの前半は、管理者による管理をなぞらえたシナリオとなっていて、商品種類の追加、商品の補充をするプログラムとなっています。(A,B の部分) この段階で、商品、商品種類、保管庫のオブジェクトの構造が構築されます。

後半は、ユーザが商品を買う時をなぞらえたシナリオとなっており (C の部分)、ここで、金勘定のオブジェクトの構造が構築されます。また、前半で作られたオブジェクトの構造を操作して、商品を購入するプログラムとなっています。

例題 11-1：自動販売機アプリケーションの構築(Example11_1.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 11-1：自動販売機アプリケーションの構築
4:      * 商品種類の管理、商品の管理、商品の販売を行うプログラム
5:      *
6:      * メインクラス
7:      */
8:      public class Example11_1 {
9:
10:       public static void main(String[] args) {
11:         ItemTypeList itemTypeList = new ItemTypeList();   //商品種類リストを生成する
12:         Account account = new Account();                  //投入金勘定を生成する
13:
14:         //--------管理者--------
15:
16:         //商品種類を追加する
17:         itemTypeList.add(new ItemType(1001,"cola",120));      //コーラを追加
18:         itemTypeList.add(new ItemType(1002,"soda",120));      //ソーダを追加
19:
20:         //商品を補充する
21:         //保管庫を取得する
22:         ItemStock colaStock;//コーラの保管庫
23:         ItemType colaItemType;//コーラ商品種類
24:         colaItemType = itemTypeList.search(1001);             //コーラ商品種類を検索
25:         colaStock = colaItemType.getItemStock();              //コーラ保管庫を取得
26:
27:         //商品を補充する
28:         colaStock.supply(new Item("2002/04/28",colaItemType));//4/28 のコーラを補充す
る
29:         colaStock.supply(new Item("2002/04/29",colaItemType));//4/29 のコーラを補充す
る
```

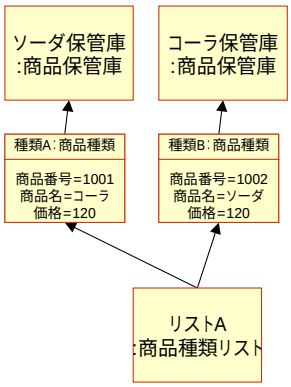
A

B

```
30:
31:         //--------ユーザ--------
32:
33:         //商品を購入する
34:         //金を投入する
35:         account.insert(new Money(100));                          //100 円投入
36:         account.insert(new Money(50));                           //50 円投入
37:
38:         //商品を選択する
39:         int id = 1001;//コーラ
40:
41:         //商品を受け取る
42:         //商品を取り出す
43:         colaItemType = itemTypeList.search(id);//商品種類を検索
44:         colaStock = colaItemType.getItemStock();//保管庫を取得
45:         Item item = colaStock.takeout();         //商品を一つ取り出す
46:
47:         System.out.println("  製  造  年  月  日  が  "+item.getDate()+"  の
"+item.getItemType().getName()+"を購入しました");
48:
49:         //おつりを受け取る
50:         int amount;//投入総額
51:         amount = account.getAmount();
52:         int change;//おつり
53:         change = amount - item.getItemType().getPrice();
54:         account.reset();//投入金勘定を空にする
55:
56:         System.out.println("おつりは"+change+"円です");
57:       }
58:
59:    }
```

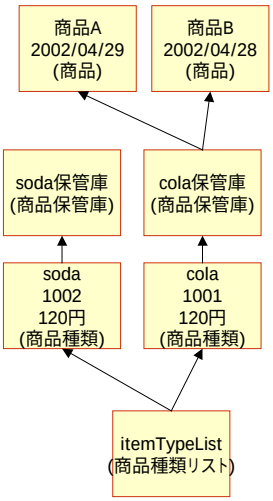
①.商品種類を生成して、リストに追加する(「A」の部分)

この段階で、次のようなインスタンスの構造が構築されます



②.商品を生成して、商品を補充する(「B」の部分)

この段階で、次のようなインスタンスの構造が構築されます

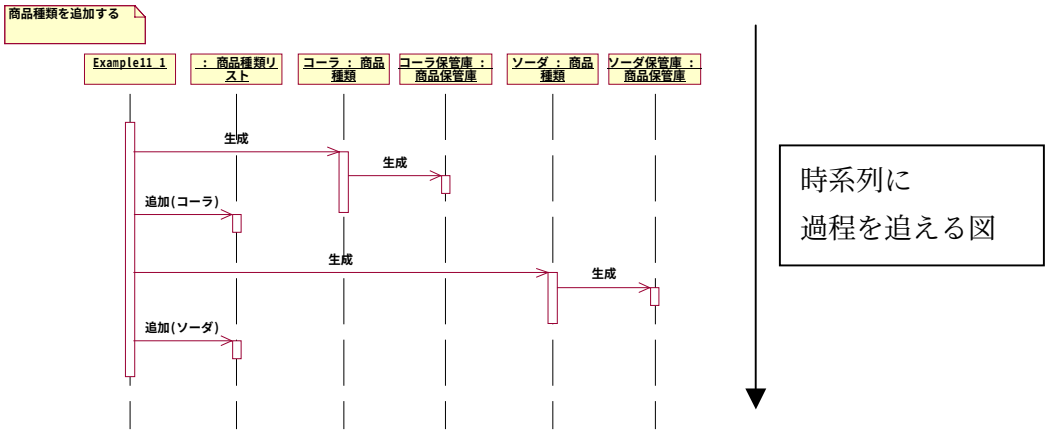


③.金を投入し、商品を購入する(「C」の部分)

インスタンスの構造を書いてみましょう。

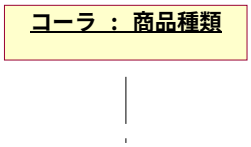
11.3.2.　シーケンス図

完成形だけインスタンスの構造図を見ても、何が起こっているかを理解するのは難しいですね。そのため、インスタンスの構造ができる（変化する）過程とそのやり取りを図式化する方法があります。

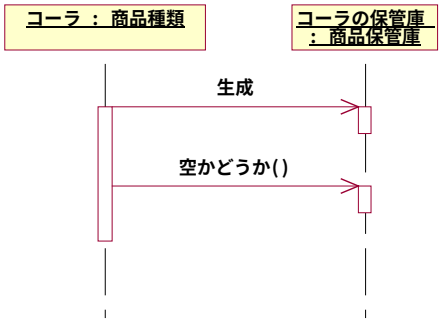


①.記法

上に並んでいるのは、インスタンスたちです。**クラスではありません**ので注意して下さい。四角に「インスタンス名：クラス名」のラベルをつけます

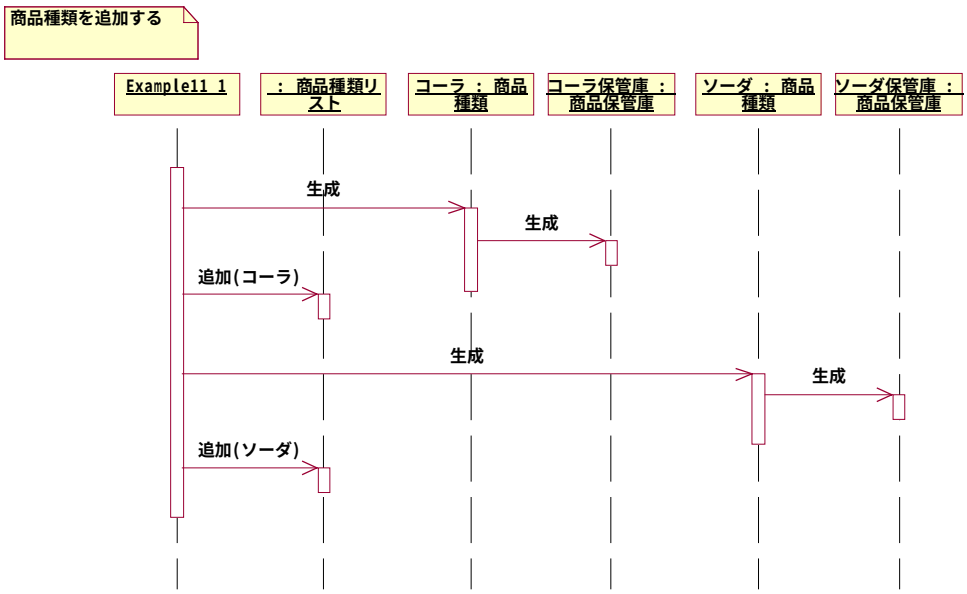


インスタンスから下に延びる線同士を矢印で結び、やり取りを記述します。やり取りはJavaではメソッドでしか行えませんので、メソッドと考えてかまいません。これらのやり取りは上から順番に実行されていきます。

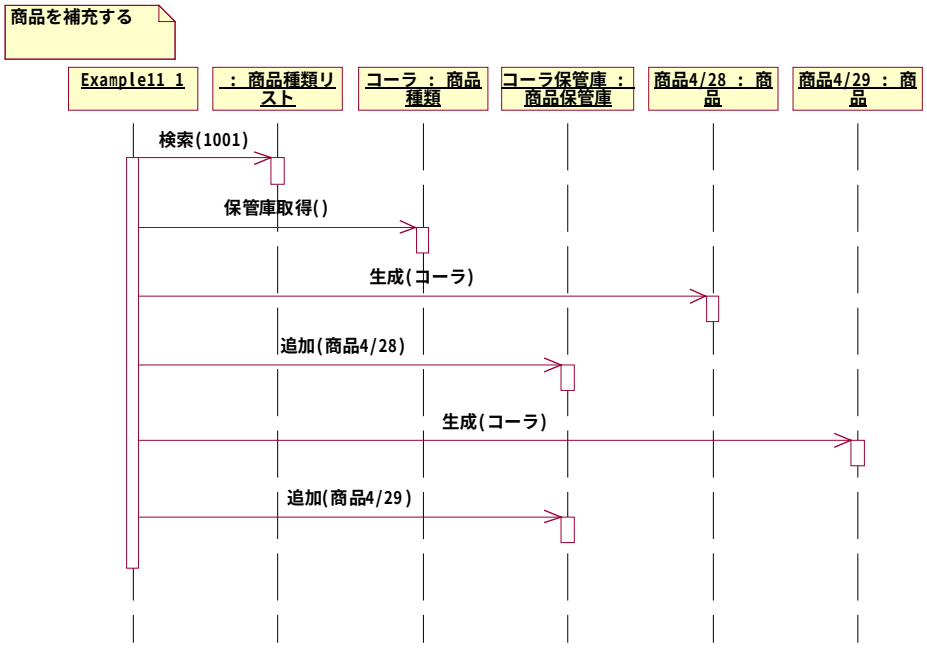


②.例題 11-1 プログラムのシーケンス図

(1)商品種類を生成して、リストに追加する(「A」の部分)



(2)商品を生成して、商品を補充する（「B」の部分）



(3)金を投入し、商品を購入する（「C」の部分）

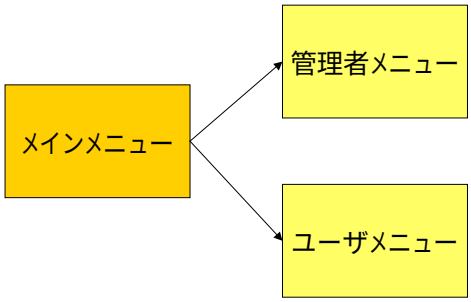
シーケンス図を書いてみましょう！

11.4. CUI アプリケーションの構築

例題 12-1 のプログラムは、コンソールにメニューを表示して、ユーザに入力を促す対話型のアプリケーションとして構成されています。

①.メニューの構成

メニューの構成は次のようになっています。



(1)メインメニュー

メインメニューでは、管理者メニューとユーザメニューのどちらかを選択できます。

メインメニュー:(1,自動販売機の管理をする　2,自動販売機を利用する　q,自動販売機の終了)

(2)管理者メニュー

管理者メニューでは、管理者が管理する目的を選択できます。

管理メニュー:(1,商品種類リストを閲覧する　2,商品種類を追加する　3,商品種類を削除する　4,在庫を閲覧する　5,商品の補充をする　q,メインメニューに戻る)

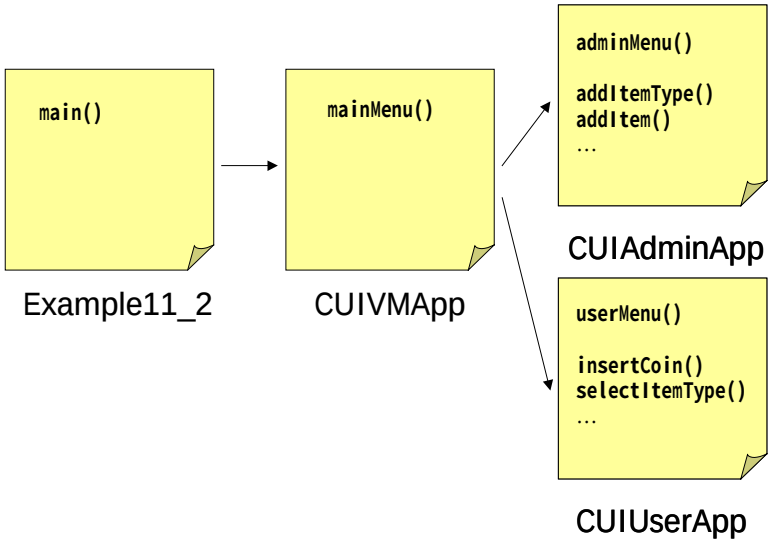
(3)ユーザメニュー

ユーザメニューでは、ユーザが商品を購入する際の目的を選択できます。

ユーザメニュー:(1,商品を選ぶ　2,商品を買う　3,お金を投入する　4,最後に入れた金の取り消し　q,メインメニューに戻る)

②.プログラムの構成

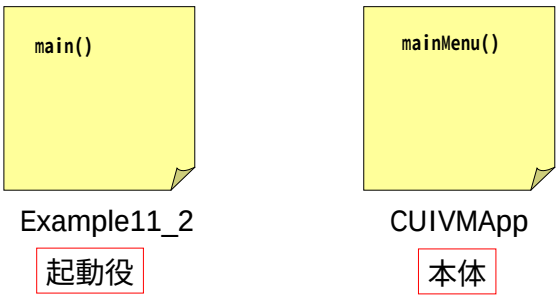
プログラムは、次のような構成になっています。



メニューごとに、プログラムが分かれています。ユーザメニューと、管理者メニューに、実際にオブジェクトの構造を変化させるプログラムがかかれています。

起動役を分割する

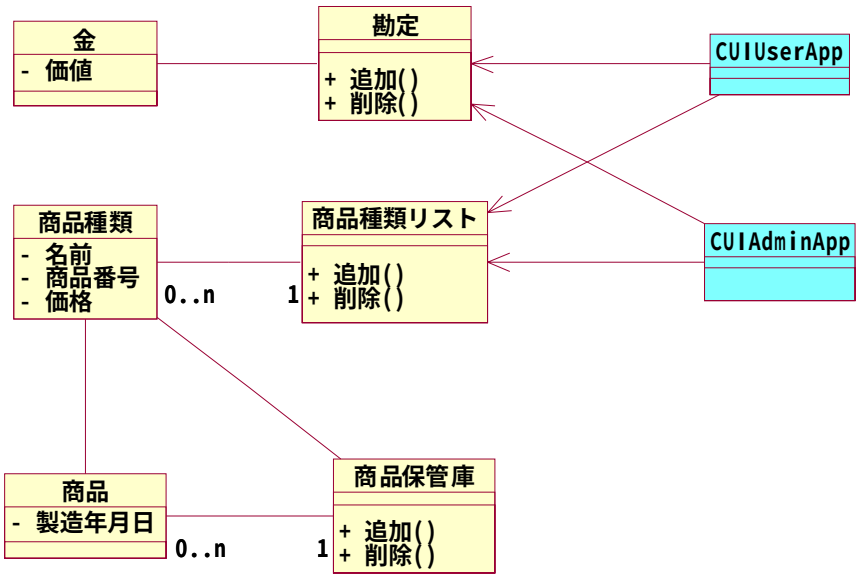
プログラムの起動役と本体を分離することによって、より分かりやすいプログラムになっています。



何故、プログラムの起動役と本体を分離すると分かりやすくなるのでしょうか？それは、起動役と本体はプログラムの意味がまったく異なるからです。起動役と本体がカタマリになっていた場合を考えてみましょう。他人はどこからプログラムが始まるのか探すのが大変になりますね。

③.クラス図

クラス図を下に示します。



自動販売機のアプリケーションプログラムは、勘定と商品種類リストだけ知っています。（参照を持っています）2 つのクラス（インスタンスは一つずつ）から、関連をたどることによって、すべてのインスタンスにたどり着け、構造を変化させることが可能になります。

④.例題 11-2 主要プログラムのソース

例題 11-2：CUI 自動販売機の完成(Example11_2.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 11-2：CUI 自動販売機の完成
4:      * CUI 自動販売機アプリケーション
5:      *
6:      * メインクラス
7:      */
8:      public class Example11_2 {
9:
10:       /**
11:       * プログラム･メイン
12:       *
13:       * 自動販売機アプリケーションを起動する
14:       */
15:       public static void main(String[] args) {
16:         ItemTypeList itemTypeList = new ItemTypeList();   //商品種類リストを生成
17:         Account account = new Account();                  //投入金勘定を生成
18:
19:         CUIVMApp application = new CUIVMApp(itemTypeList,account);//自動販売機アプリケ
ーションを生成する
20:         application.mainMenu();//自動販売機アプリケーションを開始する
21:       }
22:
23:     }
```

例題 11-2：CUI 自動販売機の完成(CUIVMApp.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 11-2：CUI 自動販売機の完成
4:      * CUI 自動販売機アプリケーション
5:      *
6:      * 自動販売機アプリケーションクラス
7:      * CUI により、自動販売機を利用するユーザ、管理する管理者が目的とする処理を行うクラ
ス
8:      */
9:      public class CUIVMApp {
10:
11:       private ItemTypeList itemTypeList;    //商品種類リスト
```

```
12:       private Account account;              //投入金勘定
13:
14:       private CUIUserApp userApp;           //ユーザアプリケーション
15:       private CUIAdminApp adminApp;         //管理者アプリケーション
16:
17:       /**
18:       * コンストラクタ
19:       */
20:       public CUIVMApp(ItemTypeList newItemTypeList,Account newAccount) {
21:         itemTypeList = newItemTypeList; //引数で与えられた商品種類リストを初期化する
22:         account = newAccount;           //引数で与えられた投入金勘定を初期化する
23:
24:         userApp = new CUIUserApp(itemTypeList,account);     //ユーザアプリケーションを
初期化する
25:         adminApp = new CUIAdminApp(itemTypeList,account);   //管理者アプリケーションを
初期化する
26:       }
27:
28:       /**
29:       * メインメニュー
30:       */
31:       public void mainMenu(){
32:
33:         //無限ループ
34:         while(true){
35:           try{
36:
37:             //ユーザが自動販売機を利用する目的を選ぶ
38:             System.out.println("メインメニュー:(1,自動販売機の管理をする 2,自動販売機
を利用する q,自動販売機の終了)");
39:             String input = Input.getInput();
40:
41:             //選ばれた目的を実行する
42:             if (input.equals("1")){       //自動販売機の管理をする
43:               adminApp.adminMenu();
44:             }else if(input.equals("2")){  //自動販売機を利用する
45:               userApp.userMenu();
46:             }else if(input.equals("q")){  //自動販売機の終了
47:               System.exit(0);
48:             }
49:
50:           }catch(Exception ex){
51:             System.out.println("例外が発生しました");
52:           }
53:         }
54:       }
55:
56:     }
```

例題 11-2：CUI 自動販売機の完成(CUIAdminApp.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 11-2：CUI 自動販売機の完成
4:      * CUI 自動販売機アプリケーション
5:      *
6:      * 管理者アプリケーションクラス
7:      * CUI により、自動販売機を管理する管理者が目的とする処理を行うクラス
8:      */
9:      public class CUIAdminApp {
10:
11:       private ItemTypeList itemTypeList;     //商品種類リスト
12:       private Account account;               //投入金勘定
13:
14:       /**
15:       * コンストラクタ
16:       */
17:       public CUIAdminApp(ItemTypeList newItemTypeList,Account newAccount) {
18:         itemTypeList = newItemTypeList; //引数で与えられた商品種類リストを初期化する
19:         account = newAccount;           //引数で与えられた投入金勘定を初期化する
20:       }
21:
22:       /**
23:       * 管理メニュー
24:       * 自動販売機の管理をする
25:       */
26:       public void adminMenu(){
27:
28:         //無限ループ
29:         while(true){
30:
31:           //ユーザが商品を管理する目的を選ぶ
32:           System.out.println("管理メニュー:(1,商品種類リストを閲覧する 2,商品種類を追
加する 3,商品種類を削除する 4,在庫を閲覧する 5,商品の補充をする q,メインメニューに戻
る)");
33:           String input = Input.getInput();
34:
35:           //選ばれた目的を実行する
36:           if (input.equals("1")){       //商品種類リストを閲覧する
37:             showItemTypeList();
38:           }else if(input.equals("2")){  //商品種類を追加する
39:             addItemType();
40:           }else if(input.equals("3")){  //商品種類を削除する
41:             removeItemType();
42:           }else if(input.equals("4")){  //在庫を閲覧する
43:             showItemStock();
44:           }else if(input.equals("5")){  //商品の補充をする
```

```
45:             supplyItem();
46:           }else if(input.equals("q")){  //メインメニューに戻る
47:             break;//無限ループを抜ける
48:           }
49:         }
50:       }
51:
52:       /**
53:       * 商品種類リストを閲覧する
54:       */
55:       public void showItemTypeList(){
56:         itemTypeList.display();
57:       }
58:
59:       /**
60:       * 商品種類を追加する
61:       */
62:       public void addItemType(){
63:
64:         //追加する商品種類を決定する
65:         System.out.println("商品名を入力してください");
66:         String inputName = Input.getInput();
67:         System.out.println("価格を入力してください");
68:         int inputPrice = Integer.parseInt(Input.getInput());
69:         System.out.println("商品 ID を入力してください");
70:         int inputId = Integer.parseInt(Input.getInput());
71:
72:         ItemType itemType = new ItemType(inputId,inputName,inputPrice);//商品種類を生
成する
73:
74:         //同じ ID の商品種類が既に在るかどうか確認する
75:         if(itemTypeList.search(itemType.getId()) != null){//あった
76:           System.out.println("既に同じ ID の商品が存在します");
77:           return;
78:         }
79:
80:         //リストに加える
81:         itemTypeList.add(itemType);
82:
83:         System.out.println("追加しました");
84:       }
85:
86:       /**
87:       * 商品種類を削除する
88:       */
89:       public void removeItemType(){
90:
91:         //削除する商品種類を決定する
92:         showItemTypeList();//選べるように ID を表示する
93:         System.out.println("削除する商品 ID を上記リストから選び入力してください");
94:         int inputId = Integer.parseInt(Input.getInput());
95:
96:         //削除する商品種類が存在するか確認する
97:         if(itemTypeList.search(inputId) == null){
```

```
98:           System.out.println("そのような商品種類はありません");
99:           return;
100:        }
101:
102:        //リストから削除する
103:        itemTypeList.remove(inputId);
104:
105:        System.out.println("削除しました");
106:      }
107:
108:      /**
109:      * 在庫を閲覧する
110:      */
111:      public void showItemStock(){
112:
113:        //在庫を閲覧する対象となる商品の種類を決定する
114:        showItemTypeList();//選べるように ID を表示する
115:        System.out.println("在庫状況を確認する商品 ID を上記リストから選び入力してくだ
さい");
116:        int inputId = Integer.parseInt(Input.getInput());
117:        ItemType itemType = itemTypeList.search(inputId);//リストから検索する
118:
119:        //在庫を表示する
120:        itemType.getItemStock().display();
121:      }
122:
123:      /**
124:      * 商品の補充をする
125:      */
126:      public void supplyItem(){
127:
128:        //商品を補充する対象となる商品の種類を決定する
129:        showItemTypeList();//選べるように ID を表示する
130:        System.out.println("補充する商品 ID を上記リストから選び入力してください");
131:        int inputId = Integer.parseInt(Input.getInput());
132:        ItemType itemType = itemTypeList.search(inputId);//リストから検索する
133:
134:        //補充する商品を決定する
135:        System.out.println("日付を入力してください");
136:        String date = Input.getInput();
137:        Item item = new Item(date,itemType);//生成する
138:
139:        //商品を在庫に補充する
140:        itemType.getItemStock().supply(item);
141:
142:        System.out.println("補充しました");
143:      }
144:
145:    }
```

例題 11-2：CUI 自動販売機の完成(CUIUserApp.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 11-2：CUI 自動販売機の完成
4:      * CUI 自動販売機アプリケーション
5:      *
6:      * ユーザアプリケーションクラス
7:      * CUI により、自動販売機を利用するユーザが目的とする処理を行うクラス
8:      */
9:      public class CUIUserApp {
10:
11:       private ItemTypeList itemTypeList;     //商品種類リスト
12:       private Account account;               //投入金勘定
13:
14:       /**
15:       * コンストラクタ
16:       */
17:       public CUIUserApp(ItemTypeList newItemTypeList,Account newAccount) {
18:         itemTypeList = newItemTypeList; //引数で与えられた商品種類リストを初期化する
19:         account = newAccount;           //引数で与えられた投入金勘定を初期化する
20:       }
21:
22:       /**
23:       * ユーザメニュー
24:       * 自動販売機を利用する
25:       */
26:       public void userMenu(){
27:
28:         //無限ループ
29:         while(true){
30:
31:           //ユーザが利用する目的を選ぶ
32:           System.out.println("ユーザメニュー:(1,商品を選ぶ　2,商品を買う　3,お金を投
入する 4,最後に入れた金の取り消し q,メインメニューに戻る)");
33:           String input = Input.getInput();
34:
35:           if (input.equals("1")){       //商品を選ぶ
36:             selectItem();
37:           }else if(input.equals("2")){  //商品を買う
38:             buyItem();
39:           }else if(input.equals("3")){  //お金を投入する
40:             insertCoin();
41:           }else if(input.equals("4")){  //最後に入れた金の取り消し
42:             undoCoin();
43:           }else if(input.equals("q")){
44:             break;//無限ループを抜ける
45:           }
46:         }
47:       }
48:
```

```
49:       /**
50:       * 商品を選ぶ
51:       */
52:       public void selectItem(){
53:       }
54:
55:       /**
56:       * 商品を買う
57:       */
58:       public void buyItem(){
59:       }
60:
61:       /**
62:       * お金を投入する
63:       */
64:       public void insertCoin(){
65:       }
66:
67:       /**
68:       * 最後に入れた金を取り消す
69:       */
70:       public void undoCoin(){
71:       }
72:     }
```

練習問題

＜プログラム問題＞

☆プログラム問題 11-1

例題 11-2 を参考に、CUIUserApp.java の実装されていないメソッドを実装して、CUI 自動販売機を完成させよ。

☆プログラム問題 11-2

プログラム問題 11-1 のままの CUI アプリケーションでは、Id で検索するために大変扱いづらい。これを名前で検索できるように変更せよ。