オブジェクト指向哲学
～入門編～

第10回　　スタックとキュー

～データを収納する便利なデータ構造～

学習目標

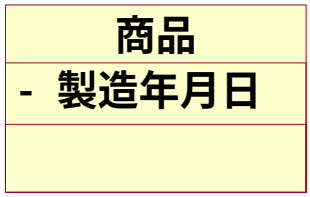
- スタック、キューの概念を説明できる
  - スタック、キューを利用して問題解決ができる
  - スタックを使ったプログラムが書ける
  - キューを使ったプログラムが書ける

10.1. 商品の補充と販売

10.1.1.　商品の補充と販売をする

①.商品クラス

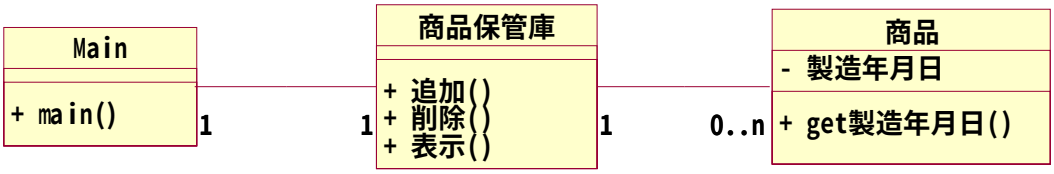
いままで議論してきた「商品種類」は商品の種類のことでした。今日はいよいよ実際の「商品」を扱います。



商品の変数は
「製造年月日」だけとします。

②.商品を収納しておくクラス設計

　商品を収納するには保管庫が必要です。今回実装する商品の補充の販売プログラムの基本構造をクラス図に示します。



③.補充と販売の仕様

- 商品を売ってほしい
- 古いものから順に売ってほしい
- 補充した数だけ売ってほしい

(1)**シナリオ**

1. 12 月 3 日製造のコーラを補充する
2. 12 月 4 日製造のコーラを補充する
3. 12 月 5 日製造のコーラを補充する
4. 12 月 3 日製造のコーラを販売する
5. 12 月 4 日製造のコーラを販売する

10.1.2.　最初に入れたものを最初に取り出す

商品保管庫に必要な機能

- 後から入れたものは、列の最後に追加されてほしい
- 取り出すときは、列の先頭から取り出してほしい

このようなデータ構造のことを**キュー**と呼びます

身の回りのキューを考えてみよう

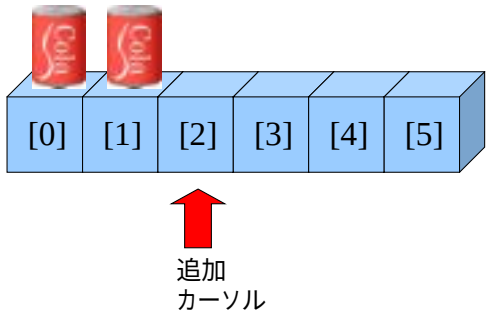
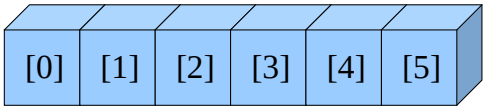
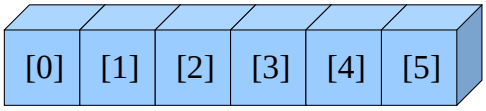
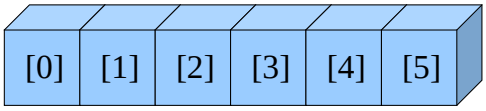
-
-
-
-

10.1.3.　配列を使ってキューを実装する

キューを配列を使って実装するためのアルゴリズムを考えてみましょう

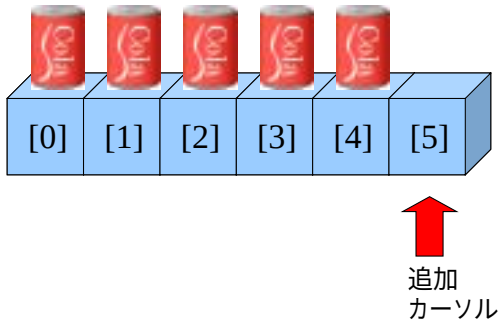
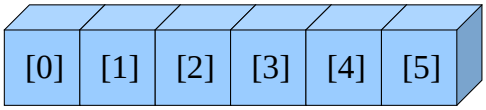
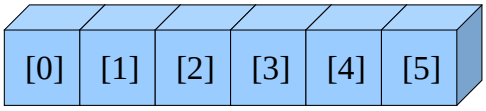
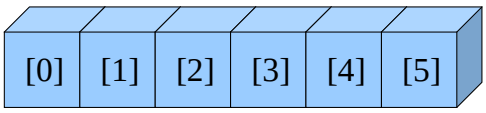
①.簡単なアルゴリズム

(1)追加

<table>
<tr><td> 
</td><td></td></tr>
<tr><td> 
</td><td>追加カーソルがある番地に商品を追加します</td></tr>
<tr><td> 
</td><td>追加カーソルを移動します</td></tr>
<tr><td> 
</td><td>追加カーソルがある番地に商品を追加します</td></tr>
</table>

●
●
●

(2)**削除**

	
	配列の先頭に入っている商品を削除します
	配列の中身と追加カーソルをずらします
	配列の先頭に入っている商品を削除します

●
●
●

②.配列を使ったキューの実装

配列を使ったキューのプログラムリストを例題 10-1 に示します。

例題 10-1：配列を使ったキュー(Example10_1.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 10-1：配列を使ったキュー
4:      * 商品の補充と販売をするプログラム
5:      *
6:      * メインクラス
7:      */
8:      public class Example10_1 {
9:
10:       /**
11:       * 商品の補充、販売をするプログラム・メイン
12:       */
13:       public static void main(String args[]){
14:         ItemStock stock = new ItemStock();//商品が入るキューをインスタンス化する
15:
16:         //商品を３つ補充する
17:         stock.supply(new Item("11/1"));
18:         stock.supply(new Item("11/2"));
19:         stock.supply(new Item("11/3"));
20:
21:         //商品を販売する
22:         System.out.println("製造年月日"+stock.takeout().getDate()+"の商品を販売しました");
23:
24:         //商品を２つ補充する
25:         stock.supply(new Item("12/1"));
26:         stock.supply(new Item("12/2"));
27:
28:         //商品を販売する
29:         System.out.println("製造年月日"+stock.takeout().getDate()+"の商品を販売しました");
30:
31:         //キューの中身の表示
32:         stock.display();
33:
34:       }
35:
36:     }
```

例題 10-1：配列を使ったキュー(ItemStock.java)

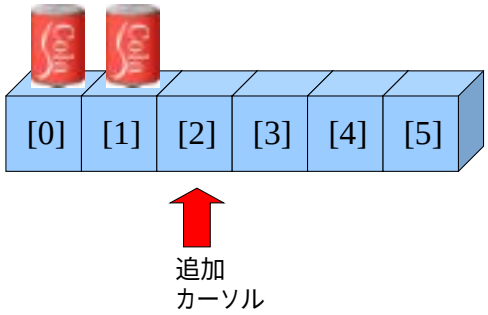
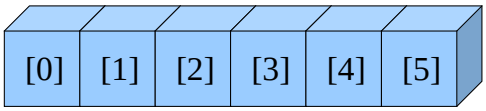
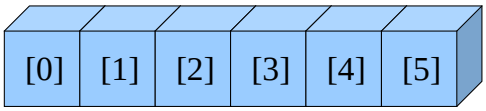
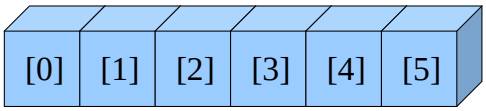
```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 10-1：配列を使ったキュー
4:      * 商品の補充と販売をするプログラム
5:      *
6:      * 商品保管庫クラス
7:      * 補充した順に取り出す構造（キュー）になっている
8:      */
9:      public class ItemStock {
10:
11:       private int ARRAY_SIZE = 6;                        //配列の大きさ
12:       private Item[] itemArray = new Item[ARRAY_SIZE]; //商品を格納する配列
13:       private int size;                                  //要素数を保存する変数
14:
15:       /**
16:       * 商品を補充する
17:       */
18:       public void supply(Item insertItem){
19:         itemArray[size] = insertItem;//最後に挿入する
20:         size++;//要素数を増やす
21:       }
22:
23:       /**
24:       * 商品を取り出す
25:       * （最初に補充されたものを取り出す）
26:       */
27:       public Item takeout(){
28:         Item removeItem = itemArray[0];//取り出す商品を一時保存しておく
29:
30:         //要素を一つずつ左にずらす
31:         for(int i=0;i<size-1;i++){
32:           itemArray[i] = itemArray[i+1];
33:         }
34:
35:         size--;//要素数を減らす
36:
37:         return removeItem;
38:       }
39:
40:       /**
41:       * 商品保管庫の中身を表示する
42:       */
43:       public void display(){
44:         System.out.println("---商品保管庫キューの先頭---");
45:         for(int i=0;i<size;i++){
46:           System.out.println("商品の製造年月日："+itemArray[i].getDate());
47:         }
48:         System.out.println("---商品保管庫キューの最後尾---");
49:       }
50:
```

例題 10-1：配列を使ったキュー(Item.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 10-1：配列を使ったキュー
4:      * 商品の補充と販売をするプログラム
5:      *
6:      * 商品クラス
7:      */
8:      public class Item {
9:
10:       private String date;//製造年月日
11:
12:       /**
13:       * コンストラクタ
14:       */
15:       public Item(String initDate) {
16:         date = initDate;
17:       }
18:
19:       /**
20:       * 製造年月日を取得
21:       */
22:       public String getDate(){
23:         return date;
24:       }
25:
26:     }
```

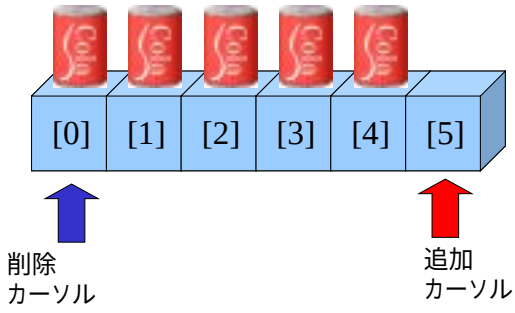
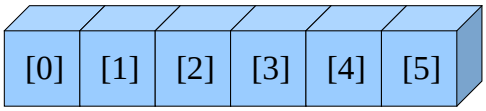
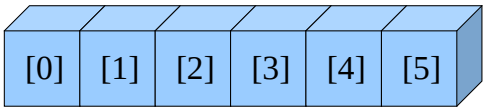
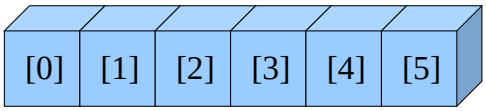
③.円環状にすることにより効率を上げる

(1)**追加**

	
	追加カーソルがある番地に商品を追加します
	追加カーソルを移動します
	追加カーソルがある番地に商品を追加します

●
●
●

(2)**削除**

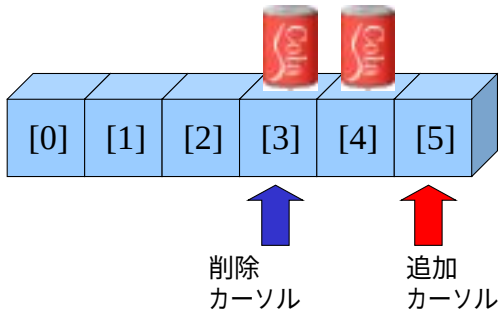
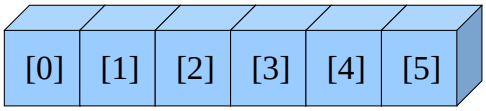
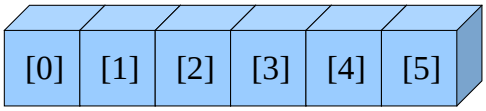
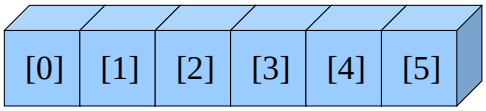
	
	削除カーソルがある番地の商品を削除します
	削除カーソルを一つずらします
	削除カーソルがある番地の商品を削除します

●

●

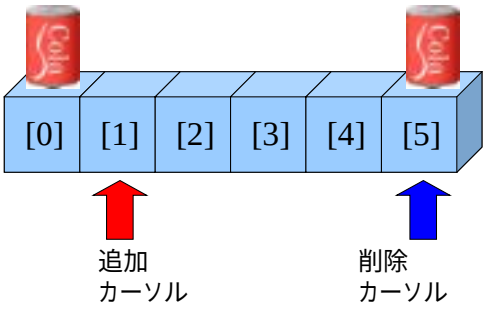
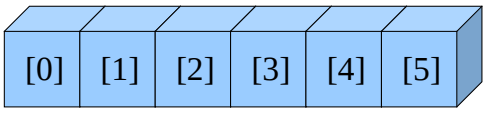
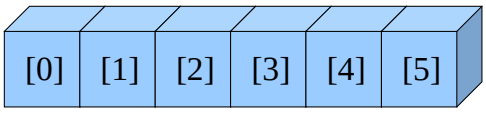
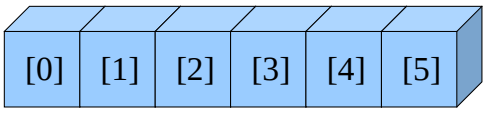
●

(3)**追加：配列の最後まできたとき**

	
	追加カーソルがある番地に商品を追加します
	追加カーソルを移動します ＜ラップアラウンド＞
	追加カーソルがある番地に商品を追加します

●
●
●

(4)**削除：配列の最後まできたとき**

	
	削除カーソルがある番地の商品を削除します
	削除カーソルを一つずらします ＜ラップアラウンド＞
	削除カーソルがある番地の商品を削除します

●
●
●

④.配列を使った円環キューの実装

円環キューで実装しなおしたプログラムリストを例題 10-2 に示します。

例題 10-2：円環キューの実装(ItemStock.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 10-2：円環キューの実装
4:      * 商品の補充と販売をするプログラム
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 = 6;//配列で円環キューを実装するときは１サイズ分大きくする。すなわちこれは５つしか入らない。
12:       private Item[] itemArray = new Item[ARRAY_SIZE];//商品を格納する配列
13:
14:       private int addCursor = 0;//追加カーソル
15:       private int removeCursor =0;//削除カーソル
16:
17:       /**
18:       * 商品を補充する
19:       */
20:       public void supply(Item insertItem){
21:         itemArray[addCursor] = insertItem;//追加カーソルの位置に挿入する
22:         addCursor++; //追加カーソルを右にずらす
23:         if (addCursor >= ARRAY_SIZE){//配列の最後にきたら
24:           addCursor = 0;//ラップアラウンドする
25:         }
26:       }
27:
28:       /**
29:       * 商品を取り出す
30:       * （最初に補充されたものを取り出す）
31:       */
32:       public Item takeout(){
33:         //実装を考えてください
34:         return null;//消してください
35:       }
36:
37:       /**
38:       * 商品保管庫の中身を表示する
39:       */
40:       public void display(){
41:         System.out.println("---商品保管庫キューの先頭---");
42:         int displayCursor = removeCursor;//表示のために配列を走査するカーソル
43:         while(displayCursor != addCursor){
44:           System.out.println("商品の製造年月日："+itemArray[displayCursor].getDate());
```

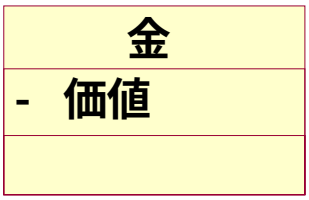
```
45:           displayCursor++;//表示カーソルを右にずらす
46:           if(displayCursor >= ARRAY_SIZE){//配列の最後にきたら
47:             displayCursor = 0;//ラップアラウンドする
48:           }
49:         }
50:         System.out.println("---商品保管庫キューの最後尾---");
51:       }
52:
53:     }
```

10.2. 投入された金を管理する

10.2.1.　投入された金の管理

①.金クラス

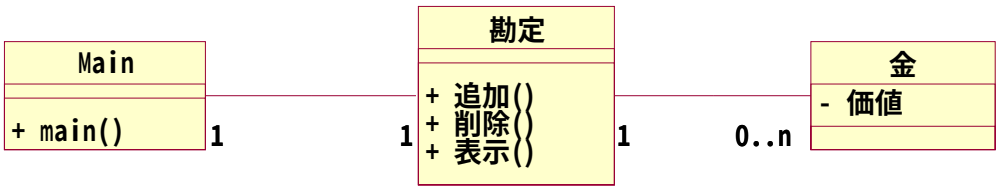
「金」クラス
　お金の一つ一つをオブジェクトと考えます。



金の変数は、
「価値」だけとします。
値は、100、10 などの整数が入ります

②.投入金を収納しておくクラス設計

　投入された金を管理するする勘定が必要です。今回実装する投入金の管理プログラムの基本構造をクラス図に示します。



③.投入された金の管理の仕様

- お金を投入できる
- 投入したお金を Undo できる

※ 実際の販売機でこの例のようなケースはあまりありませんが、一般のソフトウェアでの Undo 機能は重要な機能ですね

※ Undo を実現するには追加と逆順で削除する必要があります。

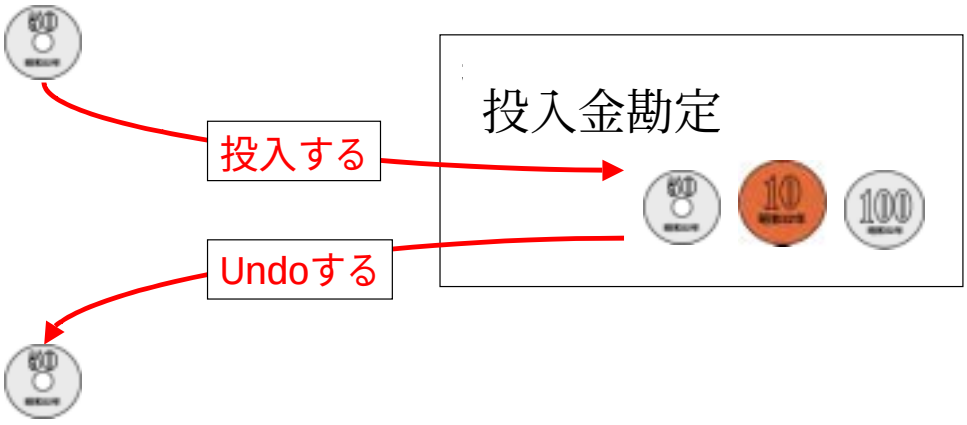
(1)シナリオ

1. 100 円を入れる（投入総額 100 円）
2. 10 円を入れる（投入総額 110 円）
3. 50 円を入れる（投入総額 160 円）
4. （お金を入れすぎたので）Undo すると 50 円が戻る（投入総額 110 円）
5. （お金を入れすぎたので）Undo すると 10 円が戻る（投入金額 100 円）

10.2.2.　最後に入れたものを最初に取り出す

投入金保管庫に必要な機能

- 後から入れたものは、列の先頭に追加されてほしい
- 取り出すときは、列の先頭から取り出してほしい



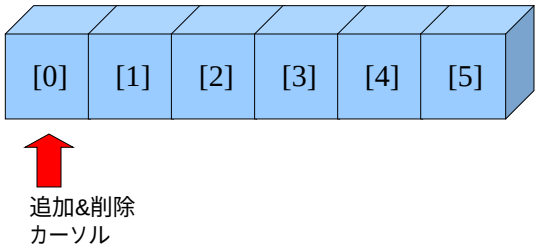
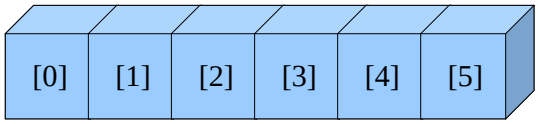
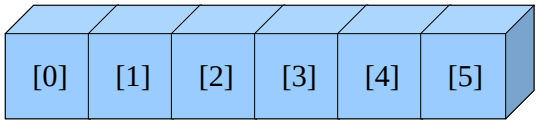
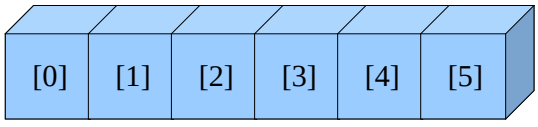
このようなデータ構造のことを**スタック**と呼びます

身の回りのスタックを考えてみよう

-
-
-
-

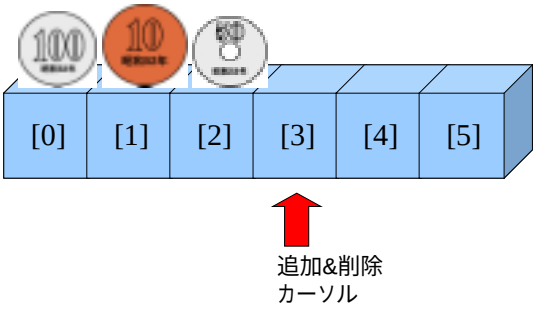
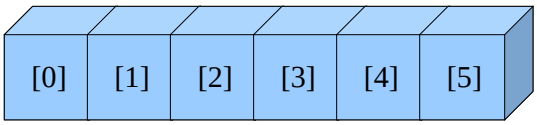
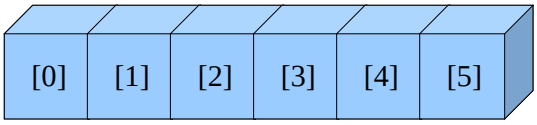
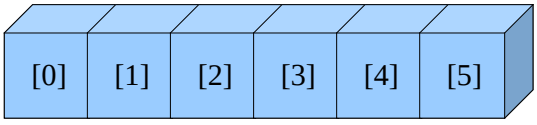
10.2.3.　配列を使ってスタックを実装する

(1)**追加**

	
	カーソルのところに 100 円を追加します
	カーソルを一つずらします
	カーソルのところに 10 円を追加します

●
●
●

(2)**削除**

	
	カーソル隣の金を削除します
	カーソルを一つずらします
	カーソル隣の金を削除します

●
●
●

②.配列を使ったスタックの実装

配列を使ったスタックのプログラムリストを例題 10-3 として示します。

例題 10-3：スタックの実装(Example10_3.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 10-3：スタックの実装
4:      * 金の投入と Undo をするプログラム
5:      *
6:      * メインクラス
7:      */
8:      public class Example10_3 {
9:
10:       /**
11:       * ユーザからの入力を受け付けて、金の投入、Undo を行うプログラム・メイン
12:       */
13:       public static void main(String args[]){
14:         Account account = new Account();//投入金勘定（スタック）をインスタンス化する
15:
16:         //お金を投入する
17:         account.insert( new Money(100) );
18:         account.insert( new Money(50)  );
19:         account.insert( new Money(10)  );
20:
21:         //undo してお金を一つ取り出す。
22:         System.out.println("お金が"+account.undo().getValue()+"円出てきました");
23:
24:         //お金を投入する
25:         account.insert( new Money(500)  );
26:         account.insert( new Money(1000) );
27:
28:         //undo してお金を一つ取り出す。
29:         System.out.println("お金が"+account.undo().getValue()+"円出てきました");
30:
31:         //勘定の中身を表示する
32:         account.display();
33:       }
34:
35:     }
```

例題 10-3：スタックの実装(Account.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 10-3：スタックの実装
4:      * 金の投入と Undo をするプログラム
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）で実
装する
8:      */
9:      public class Account {
10:
11:       private int ARRAY_SIZE = 5;                          //配列の大きさ
12:       private Money[] moneyArray = new Money[ARRAY_SIZE];//金を保存する配列
13:       private int cursor = 0;                              //追加・削除する際に使うカー
ソル
14:
15:       /**
16:       * 金を投入する
17:       */
18:       public void insert(Money insertMoney){
19:         //実装を考えてください
20:       }
21:
22:       /**
23:       * 投入した金を undo(やり直し)する
24:       */
25:       public Money undo(){
26:         //実装を考えてください
27:         return null;//消してください
28:       }
29:
30:
31:       /**
32:       * 勘定の中身を表示する
33:       */
34:       public void display(){
35:         System.out.println("---勘定スタックの先頭---");
36:         int displayCursor = cursor-1;//表示のために配列を走査するカーソル
37:
38:         while(displayCursor >= 0){
39:           System.out.println("金の価値："+ moneyArray[displayCursor].getValue());
40:           displayCursor--;//表示用カーソルを戻す
41:         }
42:         System.out.println("---勘定スタックの最後尾---");
43:       }
44:
45:     }
```

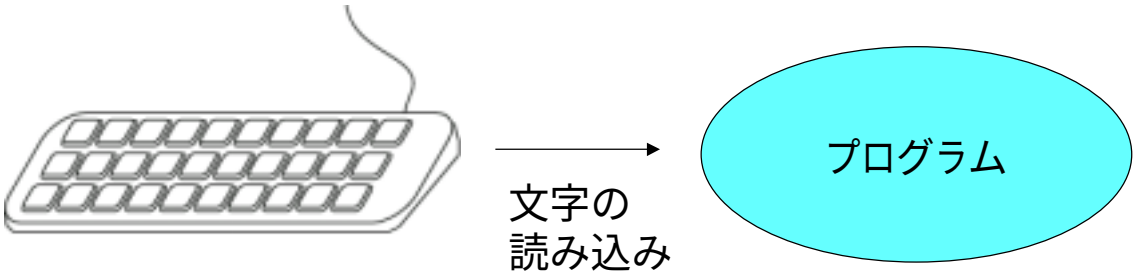
例題 10-3：スタックの実装(Money.java)

```
1:      /**
2:      * オブジェクト指向哲学　入門編
3:      * 例題 10-3：スタックの実装
4:      * 金の投入と Undo をするプログラム
5:      *
6:      * 金クラス
7:      */
8:      public class Money {
9:
10:       private int value; //金の値（単位：円）
11:
12:       /**
13:       * コンストラクタ
14:       */
15:       public Money( int initValue ) {
16:         value = initValue;
17:       }
18:
19:       /**
20:       * 金の値を取得する
21:       */
22:       public int getValue(){
23:         return value;
24:       }
25:
26:     }
```

10.3. インタラクティブなプログラム

10.3.1.　キーボード入力

今回のプログラムではよりインタラクティブに、コンソール（標準入力）から文字を読み込むプログラムにします。



コンソールから文字を読み込むのに必要な機能は「java.io パッケージ」の中に用意されています。その機能を使って実装すると、以下のようになります。

```
1:      /**
2:       * ユーザの入力を一行読み込む
3:       */
4:       public static String getInput(){
5:         String buf = null;//読み込んだ文字列を保存する
6:         try{
7:           //読み込むための BufferedReader インスタンスをインスタンス化する
8:           BufferedReader br = new BufferedReader(new InputStreamReader
(System.in));
9:           buf =  br.readLine();//一行読み込む
10:        }catch(Exception ex){
11:          ex.printStackTrace();//入出力エラーが起きた場合エラーを表示する
12:        }
13:        return buf;//読み込んだ文字列を返す
14:      }
```

①.インタラクティブなプログラムの実装

getInput()メソッドを利用する例として、商品の補充と販売を行う例題 10-4 を示します。

例題 10-4：ユーザの入力を受け付ける(Example10_4Item.java)

```
1:      import java.io.*;//入出力する Java のクラスを使うときに宣言しなければならない
2:
3:      /**
4:      * オブジェクト指向哲学　入門編
5:      * 例題 10-4：ユーザの入力を受け付ける
6:      * 商品の補充と販売をするプログラム
7:      *
8:      * 商品保管庫の管理をするメインクラス
9:      */
10:     public class Example10_4Item {
11:
12:       /**
13:       * ユーザからの入力を受け付けて、商品の補充、販売を行なうプログラム
14:       */
15:       public static void main(String args[]){
16:         ItemStock stock = new ItemStock();//商品保管庫（キュー）をインスタンス化する
17:
18:         //メイン・ループ
19:         //終了が呼ばれるまで、メニューを出す
20:         while(true){
21:           //メニューを表示してユーザの入力を受け付けます。
22:           //半角数字以外の文字を入れるとプログラムが落ちるので注意！
23:           System.out.println("補充 1,販売 2,表示 3,終了 4");
24:           String menu = getInput();//ユーザが選んだメニューを保存する
25:
26:           if(menu.equals("1")){
27:             //商品を補充する
28:             System.out.println("商品の製造日を入力してください");
29:             String date = getInput();//ユーザが入力した日付を保存する
30:             stock.supply(new Item(date));//商品を保管庫へ挿入する
31:           }
32:
33:           else if(menu.equals("2")){
34:             //商品を販売する
35:             Item item = stock.takeout();//商品を保管庫から削除する
36:             System.out.println("商品製造日："+item.getDate()+"の商品を販売しました");
37:           }
38:
39:           else if(menu.equals("3")){
40:             //商品一覧を表示する
41:             stock.display();
42:           }
43:
```

```
44:           else if(menu.equals("4")){
45:             //プログラムを終了する
46:             break;//メイン・ループを抜ける
47:           }
48:         }
49:       }
50:
51:       /**
52:       * ユーザの入力を一行読み込む
53:       * 変更不要
54:       */
55:       public static String getInput(){
56:         String buf = null;//読み込んだ文字列を保存する
57:         try{
58:           //読み込むための BufferedReader インスタンスをインスタンス化する
59:           BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
60:           buf =  br.readLine();//一行読み込む
61:         }catch(Exception ex){
62:           ex.printStackTrace();//入出力エラーが起きた場合エラーを表示する
63:         }
64:         return buf;//読み込んだ文字列を返す
65:       }
66:     }
```

例題 10-4：ユーザの入力を受け付ける(Example10_4Money.java)

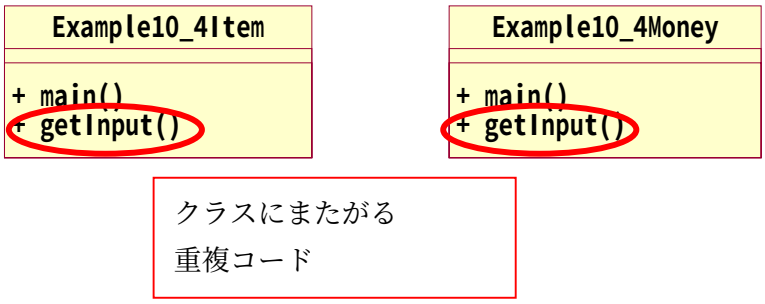
```
1:      import java.io.*;//入出力する Java のクラスを使うときに宣言しなければならない
2:
3:      /**
4:      * オブジェクト指向哲学　入門編
5:      * 例題 10-4：ユーザの入力を受け付ける
6:      * 金の投入と Undo をするプログラム
7:      *
8:      * 勘定の管理をするメインクラス
9:      */
10:     public class Example10_4Money {
11:
12:       /**
13:       * ユーザからの入力を受け付けて、金の投入、Undo を行うプログラム
14:       */
15:       public static void main(String args[]){
16:         Account account = new Account();//投入金勘定（スタック）をインスタンス化する
17:
18:         //メイン・ループ
19:         //終了が呼ばれるまで、メニューを出す
20:         while(true){
21:
22:           //メニューを表示してユーザの入力を受け付けます。
23:           //半角数字以外の文字を入れるとプログラムが落ちるので注意！
```

```
24:           System.out.println("お金投入 1,Undo2,表示 3,終了 4");
25:           String menu = getInput();//ユーザが選んだメニューを保存する
26:
27:           if(menu.equals("1")){
28:             //お金を投入する
29:             System.out.println("お金を入力してください");
30:             String valueStr = getInput();//入力結果を coin に保存する
31:             int value = Integer.parseInt(valueStr);
32:             account.insert(new Money(value));//投入金勘定に挿入する
33:           }
34:
35:           else if(menu.equals("2")){
36:             //投入金の Undo(一つ取り消し)をする
37:             Money money = account.undo();//投入金勘定から金を削除する
38:             System.out.println("入れたお金："+money.getValue()+"を取り消しました");
39:           }
40:
41:           else if(menu.equals("3")){
42:             //投入金一覧を表示する
43:             account.display();
44:           }
45:
46:           else if(menu.equals("4")){
47:             //プログラムを終了する
48:             break;//メイン・ループを抜ける
49:           }
50:         }
51:       }
52:
53:       /**
54:       * ユーザの入力を一行読み込む
55:       * 変更不要
56:       */
57:       public static String getInput(){
58:         String buf = null;//読み込んだ文字列を保存する
59:         try{
60:           //読み込むための BufferedReader インスタンスをインスタンス化する
61:           BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
62:           buf =  br.readLine();//一行読み込む
63:         }catch(Exception ex){
64:           ex.printStackTrace();//入出力エラーが起きた場合エラーを表示する
65:         }
66:         return buf;//読み込んだ文字列を返す
67:       }
68:
69:     }
```

10.3.2.　キーボード入力のメソッドはどこに配置する？

①.メインクラスに書く

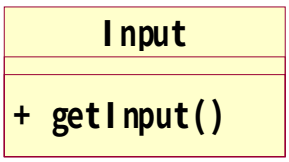
Main メソッドに書くのは明らかに問題です。キーボード入力のメソッドは便利なので、様々な状況で使いたいでしょう。その際にいちいち Main クラスにこのメソッドを追加していってはきりがありません。



2 つのクラスの間にまたがる重複コードを避けるためにはどうしたらいいでしょう。

②.入力を受け付けるクラスを作る

重複コードがクラス間にまたがるということは、クラスの役割分担がうまくいっていない証拠ではないでしょうか。そこで、入力を受けるメソッドを持つ新しいクラスを定義します。これで重複コードはなくなりますし、「入力を受け付ける」というのは、他のプログラムとは明らかに意味が違いますから、プログラムを分割するのは、分かりやすいプログラムにもなります。



意味を考えて分割した Input クラスを例題 10-5 として示します。

例題 10-5：Input クラス導入(Input.java)

```
1:      import java.io.*;//入出力する Java のクラスを使うことを宣言する
2:
3:      /**
4:      * オブジェクト指向哲学　入門編
5:      * 例題 10-5：Input クラス導入
6:      * 金の投入と undo、商品の補充と販売をするプログラム
7:      *
8:      * 入力クラス
9:      */
10:     public class Input {
11:
12:       /**
13:       * ユーザの入力を一行読み込む
14:       * 変更不要
15:       */
16:       public String getInput(){
17:         String buf = null;//読み込んだ文字列を保存する
18:         try{
19:           //読み込むための BufferedReader インスタンスをインスタンス化する
20:           BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
21:           buf =  br.readLine();//一行読み込む
22:         }catch(Exception ex){
23:           ex.printStackTrace();//入出力エラーが起きた場合エラーを表示する
24:         }
25:         return buf;//読み込んだ文字列を返す
26:       }
27:     }
```

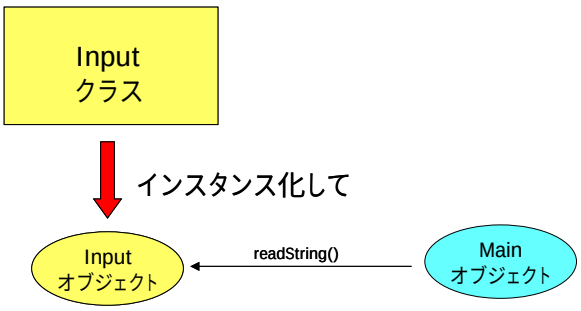
しかし今度は別の問題が生じてしまいます。ある Main クラスの中でこの readString メソッドを使いたい時には、以下のような書き方をする必要があります。

```
Input input = new Input();//ユーザからの入力を得るためのインスタンス
String menu = input.readString()//コンソールからユーザの入力を得る
```

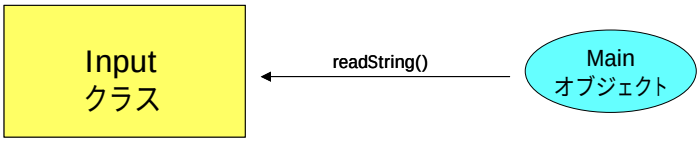
つまりメソッドを使うためにいちいち新しいインスタンスを作らなければならないのです。インスタンスを作るということは当然メモリを余計に消費することになってしまい、メモリの無駄遣いをしてしまいます。

③.クラスメソッドを利用する

これまでのプログラムでは、以下のようにインスタンスを生成して、そのインスタンスに対してメソッドを使ってきました。



しかしクラスメソッドを使うと、インスタンスを作らずに直接クラスに対してメソッドを呼び出すことが可能です。



(1)クラスメソッドの定義と使い方

クラスメソッドを定義するには、メソッド宣言の際に「static 修飾子」を付けます。
クラスメソッドを呼び出せるようにした Input クラスを例題 10-6 に示します。

このようなクラスメソッドを呼び出すには

```
クラス名．クラスメソッド名( ):
```

と書くことで呼び出すことができます。以下にその例を示します.

```
String menu = Input.getInput()//コンソールからユーザの入力を得る
```

(2)クラスメソッドの利用例

このクラスメソッドを利用して、投入された金の管理と Undo を行うプログラムのリスト(例題 10-6)を示します。

例題 10-6：クラスメソッド(Input.java)

```
1:      import java.io.*;//入出力する Java のクラスを使うときに宣言しなければならない
2:
3:      /**
4:      * オブジェクト指向哲学　入門編
5:      * 例題 10-6：クラスメソッド
6:      * 金の投入と undo をするプログラム
7:      *
8:      * 入力クラス
9:      * コンソールでのユーザからの入力を受け取る
10:     */
11:     public class Input {
12:
13:       /**
14:       * ユーザの入力を一行読み込む
15:       * 変更不要
16:       */
17:       public static String getInput(){
18:         String buf = null;//読み込んだ文字列を保存する
19:         try{
20:           //読み込むための BufferedReader インスタンスをインスタンス化する
21:           BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
22:           buf =  br.readLine();//一行読み込む
23:         }catch(Exception ex){
24:           ex.printStackTrace();//入出力エラーが起きた場合エラーを表示する
25:         }
26:         return buf;//読み込んだ文字列を返す
27:       }
28:     }
```

例題 10-6：クラスメソッド(Example10_6Money.java)

```
29:     /**
30:     * オブジェクト指向哲学　入門編
31:     * 例題 10-6：クラスメソッド
32:     * 金の投入と undo をするプログラム
33:     *
34:     * 勘定の管理をするメインクラス
35:     */
36:     public class Example10_6Money {
37:
38:       /**
39:       * ユーザからの入力を受け付けて、金の投入、Undo を行うプログラム・メイン
40:       */
41:       public static void main(String args[]){
42:         Account account = new Account();//投入金勘定（スタック）をインスタンス化する
43:
```

```
44:         //メイン・ループ
45:         //終了が呼ばれるまで、メニューを出す
46:         while(true){
47:
48:           //メニューを表示してユーザの入力を受け付けます。
49:           //半角数字以外の文字を入れるとプログラムが落ちるので注意！
50:           System.out.println("金投入 1,Undo2,表示 3,終了 4");
51:           String menu = Input.getInput();//ユーザが選んだメニューを保存する
52:
53:           if(menu.equals("1")){
54:             //金を投入する
55:             System.out.println("お金を入力してください");
56:             String valueStr = Input.getInput();//入力結果を coin に保存する
57:             int value = Integer.parseInt(valueStr);
58:             account.insert(new Money(value));//投入金勘定に挿入する
59:           }
60:
61:           else if(menu.equals("2")){
62:             //投入金の Undo(一つ取り消し)をする
63:             Money money = account.undo();//投入金勘定から金を削除する
64:             System.out.println("入れた金："+money.getValue()+"を取り消しました");
65:           }
66:
67:           else if(menu.equals("3")){
68:             //投入金一覧を表示する
69:             account.display();
70:           }
71:
72:           else if(menu.equals("4")){
73:             //プログラムを終了する
74:             break;//メイン・ループを抜ける
75:           }
76:         }
77:       }
78:
79:     }
```

練習問題

＜記述問題＞

☆記述問題 10-1

身の回りにあるキュー、スタックをそれぞれ 5 つ挙げよ。

＜プログラム問題＞

☆プログラム問題 10-1

例題 10-6 を参考に、商品の補充と販売をするプログラムを作成せよ。ただし、商品は古いもの（先に入れたもの）から販売されるものする。メニューを表示し、コンソールからユーザの入力を受け付けるインタラクティブなプログラムせよ。

☆プログラム問題 10-2

例題 10-6 を参考に、お金の投入・Undo ができるプログラムを作成せよ。Undo は無限にできるものとする。メニューを表示し、コンソールからユーザの入力を受け付けるインタラクティブなプログラムにせよ。