オブジェクト指向哲学
～入門編～

第14回　GUI の構成とイベント･ドリブン

～GUI を使ったプログラム(Ⅰ)～

学習目標

- イベント･ドリブンプログラミングの利点を説明できる
- Swing を利用して簡単な GUI プログラムが書ける
  - 簡単なカスタムウインドウを表示できる
  - イベントハンドラの生成･登録ができる
- Model と View を分離する設計について議論できる

今回は、いよいよ GUI(Graphical User Interface)を利用した自動販売機アプリケーションを構築します。

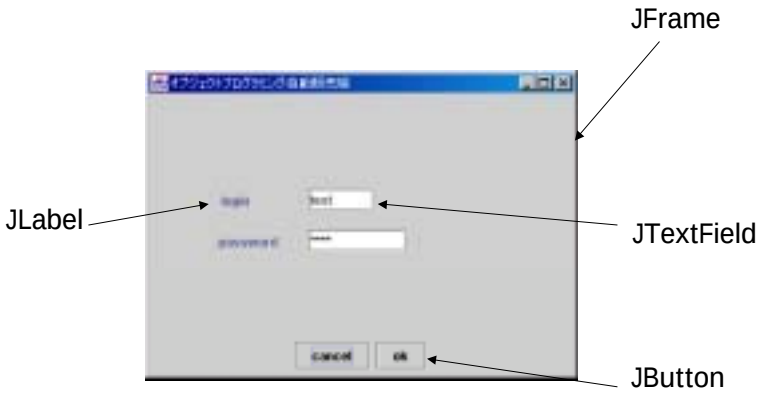
14.1. SwingAPI を使う

GUI を利用したプログラムは Java が提供するクラス･ライブラリを使います。GUI に関するクラス･ライブラリは「Swing」と呼ばれていて、GUI を構成するための数々の再利用可能なクラスが含まれています。

14.1.1. Swing 概要

Swing はオブジェクト指向で作られたクラス･ライブラリで、目に見えるすべてのものがオブジェクトとして構成されています。Swing では、それらのオブジェクトを「コンポーネント」と呼びます。コンポーネントとは「構成要素」という意味です。つまり、これらのオブジェクトは GUI を構成する要素であり､それらが集まって GUI が構成されているというわけです。

これらのコンポーネントを組み合わせることによって､例えば､下図のようなログインウインドウができます。



このログインウインドウは良く使われる基本的なコンポーネントを四種類から構成されています。

- JFrame
  最小化、最大化、終了ボタンが標準装備されているウインドウ･クラスです。
- JLabel
  文字やアイコン貼り付けることができるラベルです。
- JButton
  ユーザのマウスクリックに対応できるボタンです。
- JTextField
  ユーザが入力できるテキストフィールドです

14.1.2.　Swing を使ってみよう

①.ウインドウを出す

ウインドウを出すプログラムから初めてみましょう。ウインドウは JFrame クラスのインスタンスを生成し、表示メソッドを呼び出すことで表示されます。swing クラスライブラリを利用するための import 文を忘れないように。

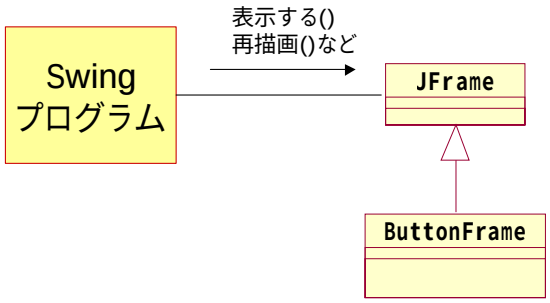
例題 14-1：ウィンドウを表示する(Example14_1.java)

```
1:      import javax.swing.*;//swing クラスライブラリの利用を宣言する
2:      import java.awt.*;//swing クラスライブラリの利用を宣言する
3:
4:      /**
5:      * オブジェクト指向哲学　入門編
6:      * 例題 14-1：ウィンドウを表示する
7:      * ウィンドウを表示するプログラム
8:      *
9:      * メインクラス
10:     */
11:     public class Example14_1 {
12:
13:       /**
14:       * プログラム･メイン
15:       * フレームを起動する
16:       */
17:       public static void main(String args[]){
18:
19:         JFrame frame = new JFrame();//Swing で提供される JFrame インスタンス生成
20:
21:         frame.setTitle("初めてのウインドウ");//タイトル設定
22:         frame.setSize(200,200);//大きさ設定
23:         frame.setLocation(50,50);//位置設定
24:         frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);//ウインドウが閉じたとき
にプログラムが終了するように設定
25:
26:         frame.setVisible(true);//表示する
27:       }
28:     }
```

②.カスタムウインドウを作る

例題 14-1 ではただウインドウを出すだけでした。次は、このウインドウにボタンを一つ乗せてみましょう。ボタンを乗せるためには、JFrame を継承したカスタムウインドウ(ButtonFrame)を作り、その上にボタンを乗せるという方法をとります。

Swing プログラムは JFrame に対して再描画や表示などの目的をプログラミングしてあるために、継承を使ってカスタムウインドウを作ることができるのです。継承の利点を生かしていますね。



例題 14-2：カスタムフレームを作る(Example14_2.java)

```
1:      import javax.swing.*;//swing クラスライブラリの利用を宣言する
2:      import java.awt.*;//swing クラスライブラリの利用を宣言する
3:
4:      /**
5:      * オブジェクト指向哲学　入門編
6:      * 例題 14-2：カスタムフレームを作る
7:      * ボタンを乗せたウィンドウを表示するプログラム
8:      *
9:      * メインクラス
10:     */
11:     public class Example14_2 {
12:
13:       /**
14:       * プログラム･メイン
15:       *
16:       * フレームを起動する
17:       */
18:       public static void main(String args[]){
19:
20:         JFrame frame = new ButtonFrame();//カスタムフレームインスタンス生成
21:
22:         frame.setTitle("初めてのウインドウ");//タイトル設定
```

```
23:         frame.setSize(200,200);//大きさ設定
24:         frame.setLocation(50,50);//位置設定
25:         frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);//ウインドウが閉じたとき
にプログラムが終了するように設定
26:
27:         frame.setVisible(true);//表示する
28:       }
29:     }
```

例題 14-2：カスタムフレームを作る(ButtonFrame.java)

```
1:      import javax.swing.*;//swing クラスライブラリの利用を宣言する
2:      import java.awt.*;//swing クラスライブラリの利用を宣言する
3:
4:      /**
5:      * オブジェクト指向哲学　入門編
6:      * 例題 14-2：カスタムフレームを作る
7:      * ボタンを乗せたウィンドウを表示するプログラム
8:      *
9:      * ボタンフレームクラス
10:     * フレームクラスを拡張し、ボタンを乗せたフレーム
11:     */
12:     public class ButtonFrame extends JFrame{
13:
14:       /**
15:       * コンストラクタ
16:       */
17:       public ButtonFrame(){
18:
19:         getContentPane().setLayout(null);//ウインドウに載せられるすべてのインスタンス
の位置を自分で設定できるようにする。
20:
21:         //ボタンを設定する
22:         JButton button = new JButton();//ボタンをインスタンス化
23:         button.setText("初めてのボタン");//ボタンのラベル名設定
24:         button.setSize(150,20);//ボタンの大きさ設定
25:         button.setLocation(20,50);//ボタンの位置設定(ウインドウからの相対位置)
26:
27:         //ボタンをウインドウに乗せる
28:         getContentPane().add(button);
29:       }
30:     }
```

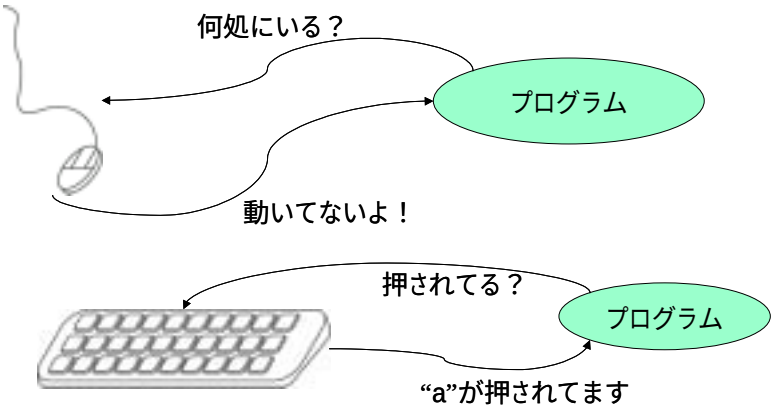
14.2. イベント･ドリブン　プログラミング

14.2.1.　ボタンが押された時の処理を考える

例題 14-2 のプログラムでは、ボタンは乗せられたものの、ボタンを押しても何も起こりません。ボタンを押した時の処理をプログラミングしていないので当然です。今回は、ボタンを押した時の処理を書く方法について、考えていきます。

①.原始的な方法

原始的な方法は、一定時間ごとにハードウエアの状態を調べるという、ポーリングと呼ばれる方法です。



この方法は、Java ではサポートされていないので、仮想言語で考えてみましょう。次のようなメソッドを一定期間ごとに起動すれば、ボタンを押したかどうかが調べられ、処理が書けるようになります。

```
//Java ではない仮想言語
public void ボタンが押されたかどうかを調べて、押されていたら終了する（）{
  マウスの位置を調べる();
  if(マウスの位置が Button の中にある){
     マウスの状態を調べる();
     if(マウスが押されている){
        if(押されているのは、マウスの左ボタンである){
          プログラムを終了する
        }
      }
    }
  }
}
```

＜考えよう！＞ポーリングの問題点

●
●
●

②.イベント･ドリブン　プログラミング

ポーリングの問題点を解決するために、イベント･ドリブンというプログラミングスタイルがとられています。これは処理をイベント毎に記述して、必要に応じて呼び出されるというものです。

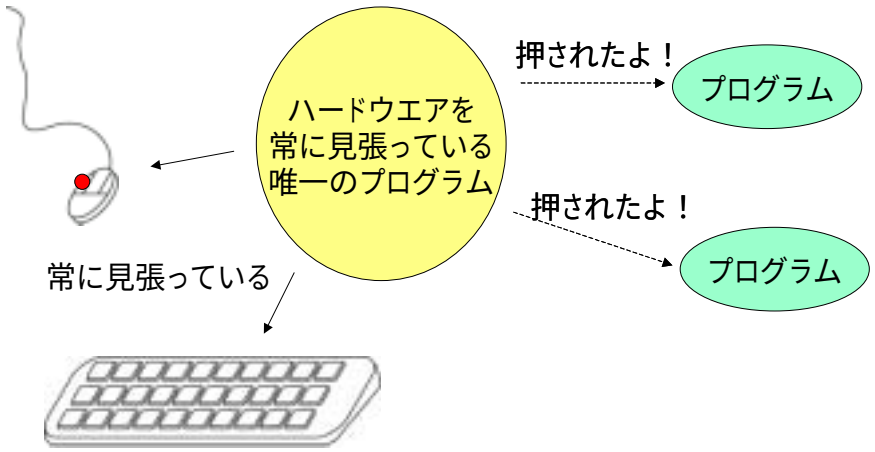
今までのプログラムは、main()から始まって、上から順次実行されるのが基本でした。しかし、イベントドリブン型のプログラミングは、イベント毎にプログラムを記述して、それが必要に応じて呼び出されるというスタイルになります。

例えば、次のようなプログラムを書いておけば、ボタンが押された時にこのメソッドが呼ばれるという仕組みです。

```
public void ボタンが押されたときの処理 () {
  System.exit(0);
}
```

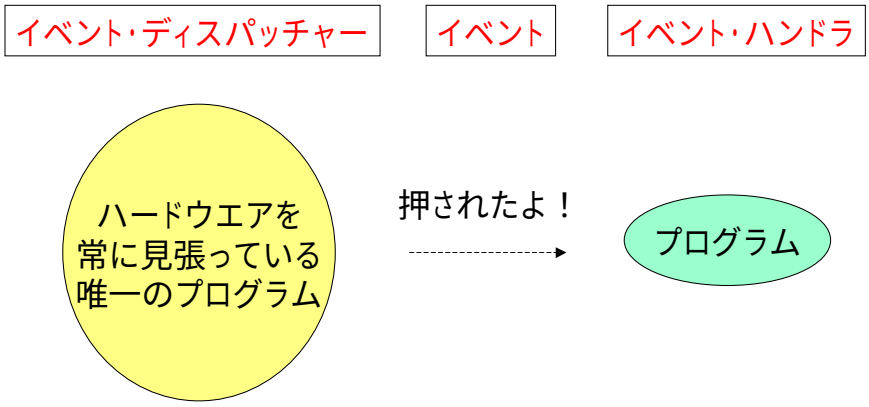
(1)イベント･ドリブンの考え方

イベント･ドリブンの考え方としては、プログラムがハードウエアを一生懸命監視するのは大変なので、唯一のプログラムとして見張り役を用意して、押された時にプログラムに通知してあげるというものです。見張り役プログラムは、ハードウエア割り込み等の特殊な機構を使って見張っているので、効率よく見張ることができます。



(2)イベント･ドリブンの用語

今回は、イベント･ドリブン型のプログラミングを行いますので、それにまつわる用語を覚えておきましょう。



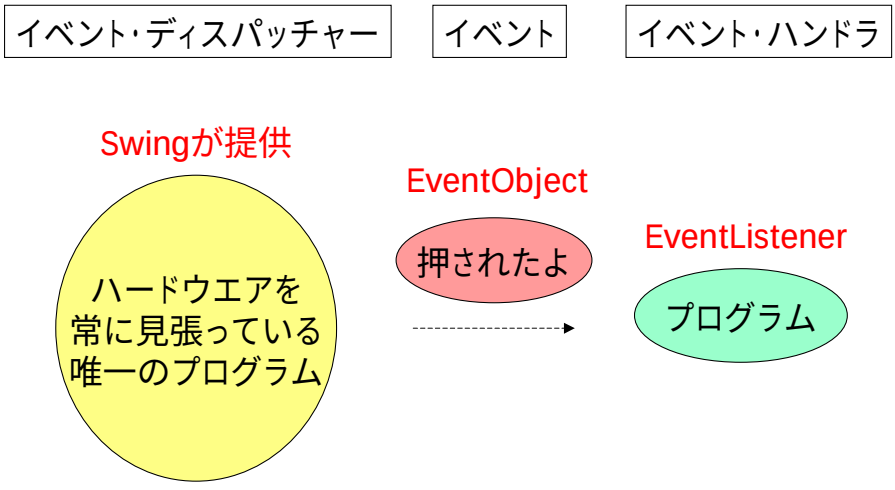
14.2.2.　Java によるイベント･ドリブン型プログラミング

①.Java によるイベント･ドリブン

Java の Swing フレームワークはイベント･ドリブン型プログラミングができる機能を標準で持っています。

Java ではイベントハンドラのことを"イベントリスナ"と呼び、オブジェクトとして表現されます。「EventListener」というインターフェイスがあり、イベントハンドラは必ずこのインターフェイスを実装します。

また、Java ではイベントもオブジェクトとして扱います。「EventObject」というインターフェイスがあり、イベントは必ずこのインターフェイスを実装します。



②.イベントハンドラの作成

例題 14-2 を発展させ、ボタンが押されたら、プログラムを終了するイベントハンドラを作成してみましょう。このハンドラを「ExitEventListener」と名づけます。メインクラスは例題 14-2 と同じです。

例題 14-3：イベントハンドラの作成(ExitButtonListener.java)

```
1:      import java.awt.event.*;//event 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 14-3：イベントハンドラの作成
6:      * ボタンを押したらプログラムを終了するプログラム
7:      *
8:      * ボタンのイベントハンドラクラス
9:      */
10:     public class ExitButtonListener implements ActionListener{
11:
12:       /**
13:       * ボタンが押されたときのイベントハンドラ
14:       */
15:       public void actionPerformed(ActionEvent e){
16:         System.exit(0);//プログラムを終了する
17:       }
18:
19:     }
```

③.イベントハンドラの登録

イベントハンドラの登録は、フレームクラスで行います。ボタンクラスの addActionListener()というメソッドで登録します。

例題 14-3：イベントハンドラの作成(ButtonFrame.java)

```
1:      import javax.swing.*;//swing クラスライブラリの利用を宣言する
2:      import java.awt.*;//swing クラスライブラリの利用を宣言する
3:
4:      /**
5:      * オブジェクト指向哲学　入門編
6:      * 例題 14-3：イベントハンドラの作成
7:      * ボタンを押したらプログラムを終了するプログラム
8:      *
9:      * ボタンフレームクラス
10:     * フレームクラスを拡張し、ボタンを乗せたフレーム
11:     */
12:     public class ButtonFrame extends JFrame{
13:
14:       /**
15:       * コンストラクタ
16:       */
17:       public ButtonFrame(){
18:
19:         getContentPane().setLayout(null);//ウインドウに載せられるすべてのインスタンス
の位置を自分で設定できるようにする
20:
21:         //ボタンを設定する
22:         JButton button = new JButton();//ボタンをインスタンス化
23:         button.setText("初めてのボタン");//ボタンのラベル名設定
24:         button.setSize(150,20);//ボタンの大きさ設定
25:         button.setLocation(20,50);//ボタンの位置設定(ウインドウからの相対位置)
26:
27:         //イベントハンドラを設定する
28:         ExitButtonListener listener = new ExitButtonListener();//イベントハンドラをイ
ンスタンス化
29:         button.addActionListener(listener);//イベントハンドラを受信者として登録する
30:
31:         //ボタンをウインドウに乗せる
32:         getContentPane().add(button);
33:       }
34:     }
```

④.インナークラス

Java ではインナークラスという機能があり、クラスの中にクラスを書くことができます。例題 14-3 の ExitButtonListner を ButtonFrame のインナークラスとして書くと次のようなプログラムになります。

例題 14-4：インナークラス(ButtonFrame.java)

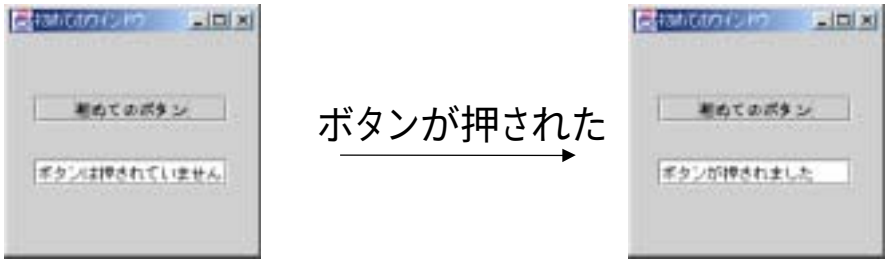
```
1:      import javax.swing.*;//swing クラスライブラリの利用を宣言する
2:      import java.awt.*;//swing クラスライブラリの利用を宣言する
3:      import java.awt.event.*;//event クラスライブラリの利用を宣言する
4:
5:      /**
6:      * オブジェクト指向哲学　入門編
7:      * 例題 14-4：インナークラス
8:      * ボタンを押したらプログラムを終了するプログラム
9:      *
10:     * ボタンフレームクラス
11:     * フレームクラスを拡張し、ボタンを乗せたフレーム
12:     * イベントハンドラとしてインナークラスを使用する
13:     */
14:     public class ButtonFrame extends JFrame{
15:
16:       /**
17:       * コンストラクタ
18:       */
19:       public ButtonFrame(){
20:
21:         getContentPane().setLayout(null);//ウインドウに載せられるすべてのインスタンス
の位置を自分で設定できるようにする
22:         //ボタンを設定する
23:         JButton button = new JButton();//ボタンをインスタンス化
24:         button.setText("初めてのボタン");//ボタンのラベル名設定
25:         button.setSize(150,20);//ボタンの大きさ設定
26:         button.setLocation(20,50);//ボタンの位置設定(ウインドウからの相対位置)
27:
28:         //イベントハンドラを設定する
29:         ExitButtonListener listener = new ExitButtonListener();//イベントハンドラをイ
ンスタンス化
30:         button.addActionListener(listener);//イベントハンドラを受信者として登録する
31:         //ボタンをウインドウに乗せる
32:         getContentPane().add(button);
33:       }
34:
35:       /**
36:       * ボタンを押したときのイベントハンドラ インナークラス
37:       */
38:       class ExitButtonListener implements ActionListener{
39:
40:         /**
41:         * ボタンが押されたときのイベントハンドラ
42:         */
43:         public void actionPerformed(ActionEvent e){
44:           System.exit(0);//プログラムを終了する
45:         }
46:       }
47:     }
```

＜考えよう！＞インナークラスの利点･欠点

●
●
●

⑤.テキストフィールドと連動するプログラム

次に、ボタンが押されたら、テキストフィールドの文字を変更するプログラムを書いてみましょう。次のようなイメージです。



例題 14-5：テキストフィールドと連動する(ButtonFrame.java)

```
1:      import javax.swing.*;//swing クラスライブラリの利用を宣言する
2:      import java.awt.*;//swing クラスライブラリの利用を宣言する
3:      import java.awt.event.*;//event クラスライブラリの利用を宣言する
4:
5:      /**
6:      * オブジェクト指向哲学　入門編
7:      * 例題 14-5：テキストフィールドと連動する
8:      * ボタンを押したらテキストフィールドの内容を変更するプログラム
9:      *
10:     * ボタンフレームクラス
11:     * フレームクラスを拡張し、ボタンとテキストフィールドを乗せたフレーム
12:     * イベントハンドラとしてインナークラスを使用
13:     */
14:     public class ButtonFrame extends JFrame{
15:
16:       private JTextField textField = new JTextField();//テキストフィールド
17:
18:       /**
19:       * コンストラクタ
20:       */
21:       public ButtonFrame(){
```

```
22:
23:         getContentPane().setLayout(null);//ウインドウに載せられるすべてのインスタンス
の位置を自分で設定できるようにする
24:
25:         //ボタンを設定する
26:         JButton button = new JButton();//ボタンをインスタンス化
27:         button.setText("初めてのボタン");//ボタンのラベル名設定
28:         button.setSize(150,20);//ボタンの大きさ設定
29:         button.setLocation(20,50);//ボタンの位置設定(ウインドウからの相対位置)
30:
31:         //イベントハンドラを設定する
32:         TextFieldChangeButtonListener listener = new TextFieldChangeButtonListener();//
イベントハンドラをインスタンス化
33:         button.addActionListener(listener);//イベントハンドラを受信者として登録する
34:
35:         //ボタンをウインドウに乗せる
36:         getContentPane().add(button);
37:
38:         //テキストフィールドを設定する
39:         textField.setText("ボタンは押されていません");
40:         textField.setBounds(20,100,150,20);//位置、大きさ設定(x,y,width,height)
41:
42:         //テキストフィールドをウインドウに載せる
43:         getContentPane().add(textField);
44:       }
45:
46:       /**
47:       * ボタンを押したときのイベントハンドラ インナークラス
48:       */
49:       public class TextFieldChangeButtonListener implements ActionListener{
50:
51:         /**
52:         * ボタンが押されたときのイベントハンドラ
53:         */
54:         public void actionPerformed(ActionEvent e){
55:           textField.setText("ボタンが押されました");
56:         }
57:       }
58:
59:     }
```

14.3. GUI 自動販売機の構成

自動販売機用のコンポーネントを用意してありますので、みなさんは、コンポーネントを追加して、イベント･ハンドラを書くだけとなっています。(勉強家のあなたはソースコードを眺めてみましょう)

14.3.1. GUI 自動販売機概要

GUI 自動販売機は次の 2 つのフレームから構成されています。

- 管理者用ウインドウ(AdminFrame クラス)
  - CUI 版 CUIAdminApp の機能を備えている
- ユーザ用ウインドウ(UserFrame クラス)
  - CUI 版 CUIUserApp の機能を備えている

ユーザウインドウ

管理者ウインドウ

管理者ウインドウは、既に実装されているので、みなさんは、ユーザウインドウのプログラミングを行うことになります。

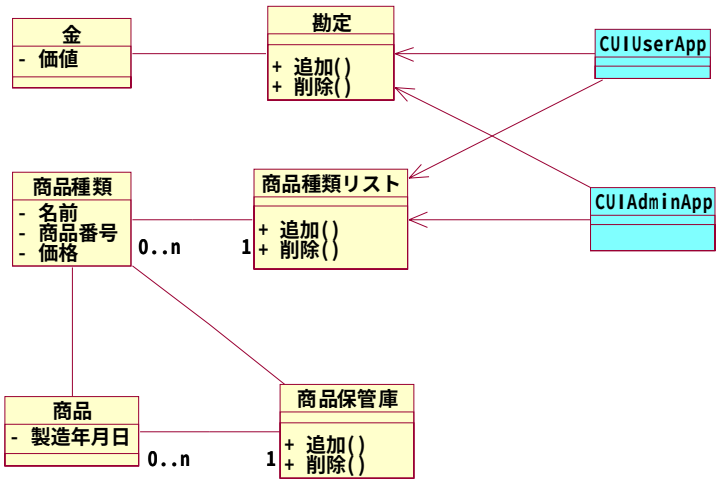
また、双方のウインドウに、「表示更新」ボタンがついています。これは、管理者ウインドウの操作をユーザウインドウに反映したり、その逆を行ったりします。(本当は、自動更新されるプログラムが望ましいのですが、我慢してください…。自動更新する GUI プログラムを書きたい人は、Observer という仕組みを勉強してください。)

14.3.2.　Model と View の分離

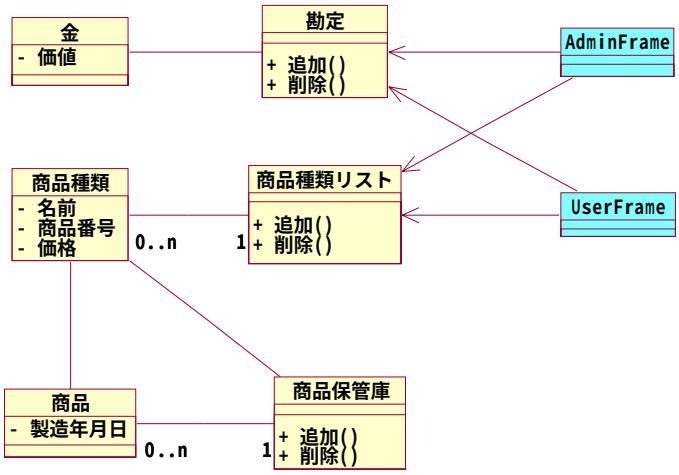
①.CUI 自動販売機の再利用

GUI 自動販売機は、CUI 自動販売機のクラス（インスタンス）構造が再利用されています。再利用されているクラスは「金」「勘定」「商品種類」「商品種類リスト」「商品」「商品保管庫」です。

CUIApp クラスがそのままウインドウになっているので、クラス構造を比較すると、同じ構造になっています。



CUI 自動販売機のクラス図

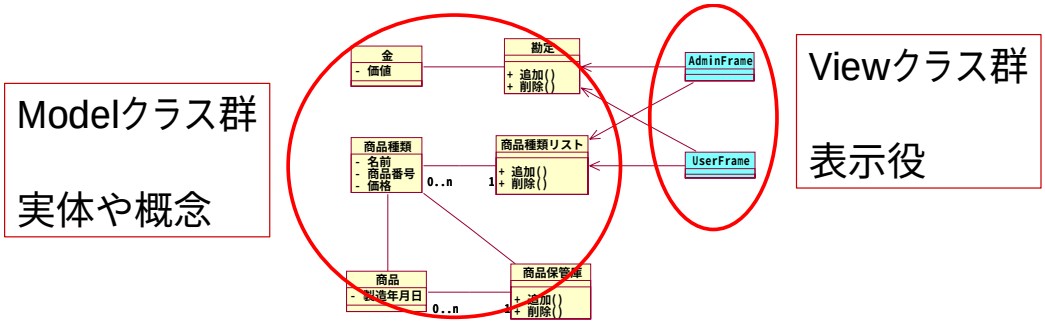


GUI 自動販売機のクラス図

②.Model と View の分離

CUI と GUI で再利用できるものとできないものの違いを考えてみましょう。

- 再利用できるもの
  - 自動販売機における実体や概念の構造に関わる部分
- 再利用できないもの
  - 表示に関わる部分
  - （アプリケーションロジックに関わる部分）



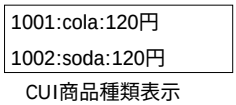
プログラムにおける、実体や概念の構造に関わる部分を一般的に「Model」といいます。そして、表示に関わる部分を「View」といいます。表示に関わるプログラムと実体や概念に関するプログラムを分離することで、様々な表示に対応したプログラムが書けるのです。

今回は、偶然に（意図的に！？）Model と View が分離していたので、Model は再利用して GUI のプログラムが書けるのです。

また､アプリケーションロジックに関する部分も分離するのがプロのオブジェクト指向技術者です。入門編では取り扱いません。まずは、Model と View を分離することを考えましょう。

③.現状の Model の問題点

②で偶然にも Model と View が分離していたと説明しましたが、実は 1 箇所だけ、分離していないところがあります。商品種類リストや商品保管庫にある「display()」メソッドです。このメソッドは、一覧をコンソールに表示するメソッドです。GUI では、当然ウインドウに表示しなければならないので、このメソッドは利用できません。



CUI商品種類表示


GUI商品種類表示

④.display()メソッドの廃止

このままでは、GUI では商品種類リストなどを表示できません。そのために、display()メソッドを廃止して、その代わりにリストにある要素をすべて取得できるメソッドを追加しましょう。これで、Model と View が完全に分離し、CUI にも GUI にも対応できる Model を作ることができます。

リストにある要素をすべて取得できるようにするには、リストに次のメソッドを追加します。

int size()	要素数を取得する
ItemType get(int index)	index 番目の商品種類を取得する

このメソッドを使って、例えば次のようなプログラムを書けば、CUI の時に使ったような display()メソッドを実現できます。

```
//商品種類リストを閲覧する
public void showItemTypeList(){
  int len = itemTypeList.size();
  for(int i=0;i<len;i++){
    ItemType itemType = itemTypeList.get(i);
    System.out.println(itemType.getId()+":"
                       +itemType.getName()+":"
                       +itemType.getPrice()+"円");
  }
}
```

このようなプログラムを View に関するクラス書くことによって、View と Model の分離できます。GUI 自動販売機は、新しく作ったメソッドを利用して、ウインドウに商品種類リストを表示できます。

14.3.3.　GUI 自動販売機プログラミング

今回の入門編では、自力で GUI プログラミングをするのは大変なので、コンポーネントが用意され、ほとんど構成されています。

ただし、Model は自分で用意します。Model はいままで作ったものがほとんどそのまま使えるはずです。

そして、ボタンを乗せたり、イベント･ハンドラを実装することによって、自動販売機は完成します。

①.自動販売機用コンポーネント

「オブジェクト指向哲学」用に用意されている自動販売機用コンポーネントを紹介します。

BackgroundPanel 	背景用のコンポーネントです。この上に他の自販機用コンポーネントを乗せることができます。
ItemImagePanel 	商品種類を表示するためのコンポーネントです。
BuyButton 	購入ボタンです。
ProcutOutlet 	商品取り出し口です。 中に商品を表示することができます。
CoinDeposit 	投入金額を表示します。

②.GUI 自動販売機の起動

GUI 自動販売機の起動プログラムです。変更する必要はないです。

「商品種類リスト」と「勘定」は、管理ウインドウとユーザウインドウで共通に利用できる必要があります。（CUI と同様です。）そのため、メインで両ウインドウを生成して、ウインドウに共通の商品種類リストと勘定の参照を渡します。

例題 14-6：初めての GUI 自動販売機(Example14_6.java)

```
1:      import gui.*;
2:
3:      import javax.swing.*;
4:
5:      /**
6:      * オブジェクト指向哲学　入門編
7:      * 例題 14-6：初めての GUI 自動販売機
8:      * GUI 自動販売機アプリケーション
9:      *
10:     * メインクラス
11:     */
12:     public class Example14_6 {
13:
14:       /**
15:       * プログラム･メイン
16:       * 操作するための 2 つのウインドウ（管理ウインドウ,購入ウインドウ）
17:       * を生成、表示する
18:       */
19:       public static void main(String args[]){
20:         ItemTypeList itemTypeList = new ItemTypeList();   //商品種類リストを生成
21:         Account account = new Account();                  //投入金勘定を生成
22:
23:         //オマジナイ
24:         setLookAndFeel();
25:
26:         //管理ウインドウを生成して表示する
27:         AdminFrame adminFrame = new AdminFrame(itemTypeList,account);//生成
28:         adminFrame.setVisible(true);//表示
29:
30:         //購入ウインドウを生成して表示する
31:         UserFrame userFrame = new UserFrame(itemTypeList,account);//生成
32:         userFrame.setVisible(true);//表示
33:       }
34:
35:       /**
36:       * GUI 見え方を変更する(オマジナイ)
37:       */
38:       private static void setLookAndFeel(){
39:         try{
```

```
40:           //Java 風（デフォルト）
41:           //UIManager.setLookAndFeel("javax.swing.plaf.basic.BasicLookAndFeel");
42:
43:           //Windows 風
44:
UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
45:
46:           //Unix 風
47:
//UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
48:
49:         }catch(Exception ex){
50:           System.out.println("サポートされていない LookAndFeel です");
51:           ex.printStackTrace();
52:         }
53:
54:       }
55:     }
```

③.UserFrame の解説

みなさんはこれからユーザウインドウ(UserFrame)のプログラミングをしてもらいます。管理ウインドウ(AdminFrame)はもう既に出来上がっているので、Model さえきちんと実装されていれば、正しく動くはずです。

UserFrame のプログラミングを始める前に、ほとんど構成されている UserFrame を眺めてみましょう。

例題 14-6：初めての GUI 自動販売機(UserFrame.java)

```
1:      import gui.*;
2:
3:      import javax.swing.*;
4:      import java.awt.*;
5:      import java.awt.event.*;
6:
7:      /**
8:      * オブジェクト指向哲学　入門編
9:      * 例題 14-6：初めての GUI 自動販売機
10:     * GUI 自動販売機アプリケーション
11:     *
12:     * 自動販売機ユーザアプリケーションクラス
13:     * GUI 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フレームクラス
14:     */
15:     public class UserFrame extends JFrame{
16:
17:       private ItemTypeList itemTypeList;  //商品種類リスト
```

```
18:       private Account account;            //投入金勘定
19:
20:       //乗せられるコンポーネント
21:       public BackgroundPanel background = new BackgroundPanel();
22:       public CoinDeposit coinDeposit;
23:       public JButton buttonUpdate = new JButton();
24:       public JPanel panelItems = new JPanel();
25:       public GridLayout gridLayout = new GridLayout();
26:       public ProductOutlet productOutlet = new ProductOutlet();
27:       public JButton buttonTakeOut = new JButton();
28:
29:       /**
30:       * コンストラクタ
31:       */
32:       public UserFrame(ItemTypeList newItemTypeList,Account newAccount){
33:
34:         itemTypeList = newItemTypeList;   //商品種類リストを設定
35:         account = newAccount;             //投入金勘定を設定
36:
37:         //ウインドウの設定
38:         this.setTitle("自動販売機購入画面");
39:         this.setSize(576,578);
40:         GUIUtil.アイコンを設定(this);
41:         GUIUtil.ウインドウを画面の中央に設置(this);
42:         this.setDefaultCloseOperation(EXIT_ON_CLOSE);
43:
44:         //背景画像パネル
45:         this.getContentPane().add(background, BorderLayout.CENTER);
46:
47:         //表示更新ボタン
48:         buttonUpdate.setText("表示更新");
49:         buttonUpdate.setBounds(new Rectangle(384, 19, 92, 27));
50:         buttonUpdate.addActionListener(new ButtonUpdate_ActionListener());
51:         background.add(buttonUpdate, null);
52:
53:         //商品群ディスプレイパネル
54:         panelItems.setOpaque(false);
55:         panelItems.setBounds(new Rectangle(30, 61, 313, 299));
56:         panelItems.setLayout(gridLayout);
57:         gridLayout.setColumns(4);
58:         gridLayout.setRows(0);
59:         background.add(panelItems, null);
60:
61:         //商品取出し口
62:         productOutlet.setBounds(new Rectangle(77, 379, 381, 165));
63:         background.add(productOutlet, null);
64:
65:         //投入金額表示ウインドウ
66:         coinDeposit = new CoinDeposit(newAccount);
67:         coinDeposit.setBounds(new Rectangle(372, 152, 160, 55));
68:         background.add(coinDeposit, null);
69:       }
70:
71:       // -------------状態更新するメソッド群-------------
```

```
72:
73:       /**
74:       * 状態を反映して、表示を更新する
75:       */
76:       public void stateUpdate(){
77:         coinDeposit.stateUpdate();
78:         allItemTypePanelStateUpdate();
79:         itemTypeListStateUpdate();
80:       }
81:
82:       /**
83:       * すべての商品種類パネルに（ボタンを含む）状態を更新通知をする
84:       */
85:       private void allItemTypePanelStateUpdate(){
86:         Component[] itemTypes = panelItems.getComponents();
87:         int len = itemTypes.length;
88:         for(int i=0;i<len;i++){
89:           ((ItemTypeShowPanel)itemTypes[i]).stateUpdate();
90:         }
91:       }
92:
93:       /**
94:       * 商品情報リストパネルを状態に応じて更新する
95:       */
96:       private void itemTypeListStateUpdate(){
97:         //毎回更新すると資源の無駄遣いのため、更新されているかどうか調べる
98:         if(!isItemTypeListStateChange()){
99:           return;//更新されていなかったら表示も更新しない
100:        }
101:
102:        //更新する
103:        //一旦すべての商品種類表示コンポーネントを削除する
104:        panelItems.removeAll();
105:
106:        //商品種類を表示するコンポーネントを生成する
107:        int len = itemTypeList.size();
108:        for(int i=0;i<len;i++){
109:          ItemType itemType = itemTypeList.get(i);
110:          ItemTypeShowPanel itsp = new ItemTypeShowPanel(this,itemType,account);
111:          panelItems.add(itsp);
112:        }
113:        panelItems.validate();
114:        panelItems.repaint();
115:      }
116:
117:      /**
118:      * 商品情報リストの状態が変化したかどうか調べる
119:      */
120:      private boolean isItemTypeListStateChange(){
121:        int len = panelItems.getComponentCount();//現在表示されている商品種類
122:        int itemTypeCount = itemTypeList.size(); //商品種類リストにある商品種類
123:        if(len != itemTypeCount){//数が違ったら更新されている
124:          return true;
125:        }
```

```
126:        //数が同じなら、すべての要素が同じかどうか調べる
127:        for(int i=0;i<len;i++){
128:          ItemTypeShowPanel                    itemTypeShowPanel                    =
(ItemTypeShowPanel)panelItems.getComponent(i);
129:          if(!itemTypeShowPanel.getItemType().equals(itemTypeList.get(i))){//異なる種
類があった
130:            return true;//更新されている
131:          }
132:        }
133:
134:        //すべて同じなら更新されていない
135:        return false;
136:      }
137:
138:      // --------------------イベントハンドラ--------------------
139:
140:      /**
141:      * 状態更新ボタン･イベントハンドラ
142:      */
143:      class ButtonUpdate_ActionListener implements java.awt.event.ActionListener{
144:        public void actionPerformed(ActionEvent e) {
145:          stateUpdate();
146:        }
147:      }
148:
149:      /**
150:      * 購入ボタン･イベントハンドラ
151:      *
152:      * 押されたボタンと、そのボタンが対象とする商品の種類が引数
153:      */
154:      public void buyButton_pressed(BuyButton buyButton,ItemType itemType){
155:      }
156:
```

(1)コンポーネントの定義

19～25 行目にかけて、各種のコンポーネントが定義され、生成されているものもあります。この生成は、コンストラクタが呼ばれるまえに行われます。

(2)コンストラクタ

30～67 行目のコンストラクタでは、各種コンポーネントの位置や大きさ、イベントハンドラの登録などが行われています。

(3)更新メソッド群

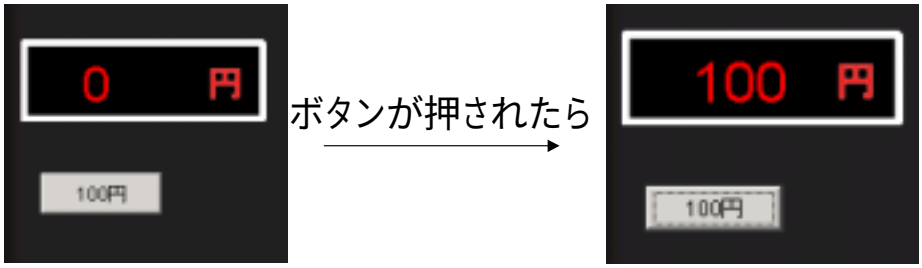
69～134 行目では、表示を更新するためのメソッドがならんでいます。Model の状態が変更された（例えば金が投入された時）には、表示を再描画する必要があります。イベントハンドラを書くときには、Model を更新したら必ず stateUpdate()メソッドを呼び出してください。その他の更新メソッドを利用して、すべてのコンポーネントの表示を再描画します。

(4)**イベント･ハンドラ**

136 行目以降は、イベントハンドラです。まだ実装されていませんので、購入ボタンが押された時の処理などを書きます。

④.100 円投入ボタンを追加する

それでは、GUI 自動販売機の完成に向けて、金を投入できるようにしましょう。



UserFrame に 100 円投入ボタンを追加したソースを示します。

例題 14-7：100 円投入ボタンの追加(UserFrame.java)

長くなるので更新メソッドは省略してあります。

```
1:      import gui.*;
2:
3:      import javax.swing.*;
4:      import java.awt.*;
5:      import java.awt.event.*;
6:
7:      /**
8:      * オブジェクト指向哲学　入門編
9:      * 例題 14-7：100 投入ボタンの追加
10:     * GUI 自動販売機アプリケーション
11:     *
12:     * 自動販売機ユーザアプリケーションクラス
13:     * GUI 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フレームクラス
14:     */
15:     public class UserFrame extends JFrame{
16:
17:       private ItemTypeList itemTypeList;  //商品種類リスト
18:       private Account account;            //投入金勘定
19:
```

```
20:       //乗せられるコンポーネント
21:       public BackgroundPanel background = new BackgroundPanel();
22:       public CoinDeposit coinDeposit;
23:       public JButton buttonUpdate = new JButton();
24:       public JPanel panelItems = new JPanel();
25:       public GridLayout gridLayout = new GridLayout();
26:       public ProductOutlet productOutlet = new ProductOutlet();
27:       public JButton buttonTakeOut = new JButton();
28:       public JButton button100 = new JButton();
29:
30:       /**
31:       * コンストラクタ
32:       */
33:       public UserFrame(ItemTypeList newItemTypeList,Account newAccount){
34:         itemTypeList = newItemTypeList;   //商品種類リストを設定
35:         account = newAccount;             //投入金勘定を設定
36:         //ウインドウの設定
37:         this.setTitle("自動販売機購入画面");
38:         this.setSize(576,578);
39:         GUIUtil.アイコンを設定(this);
40:         GUIUtil.ウインドウを画面の中央に設置(this);
41:         this.setDefaultCloseOperation(EXIT_ON_CLOSE);
42:
43:         //背景画像パネル
44:         this.getContentPane().add(background, BorderLayout.CENTER);
45:
46:         //表示更新ボタン
47:         buttonUpdate.setText("表示更新");
48:         buttonUpdate.setBounds(new Rectangle(384, 19, 92, 27));
49:         buttonUpdate.addActionListener(new ButtonUpdate_ActionListener());
50:         background.add(buttonUpdate, null);
51:
52:         //商品群ディスプレイパネル
53:         panelItems.setOpaque(false);
54:         panelItems.setBounds(new Rectangle(30, 61, 313, 299));
55:         panelItems.setLayout(gridLayout);
56:         gridLayout.setColumns(4);
57:         gridLayout.setRows(0);
58:         background.add(panelItems, null);
59:
60:         //商品取出し口
61:         productOutlet.setBounds(new Rectangle(77, 379, 381, 165));
62:         background.add(productOutlet, null);
63:
64:         //投入金額表示ウインドウ
65:         coinDeposit = new CoinDeposit(newAccount);
66:         coinDeposit.setBounds(new Rectangle(372, 152, 160, 55));
67:         background.add(coinDeposit, null);
68:
69:         //100 円投入ボタン
70:         button100.setText("100 円");
71:         button100.setBounds(new Rectangle(385, 240, 79, 27));
72:         button100.addActionListener(new Button100_ActionListener());
73:         background.add(button100, null);
```

A

B

```
74:       }
75:
76:       // !!更新メソッドは省略
77:
78:       // --------------------イベントハンドラ--------------------
79:
80:       /**
81:       * 状態更新ボタン･イベントハンドラ
82:       */
83:       class ButtonUpdate_ActionListener implements java.awt.event.ActionListener{
84:         public void actionPerformed(ActionEvent e) {
85:           stateUpdate();
86:         }
87:       }
88:
89:       /**
90:       * 購入ボタン･イベントハンドラ
91:       *
92:       * 押されたボタンと、そのボタンが対象とする商品の種類が引数
93:       */
94:       public void buyButton_pressed(BuyButton buyButton,ItemType itemType){
95:       }
96:
97:       /**
98:       * 100 円投入ボタン･イベントハンドラ                                    C
99:       */
100:      class Button100_ActionListener implements java.awt.event.ActionListener{
101:        public void actionPerformed(ActionEvent e) {
102:          account.insert(new Money(100));//100 円投入する
103:          stateUpdate();//状態を更新する
104:        }
105:      }
106:
107:   }
```

追加されているのは図中の A、B、C の 3 箇所です。

(1)A

ボタンの宣言と生成を行っています。

(2)B

ボタンの位置設定、イベントハンドラの登録、ウインドウへの追加を行っています。

(3)C

追加したボタンのイベントハンドラです。投入金勘定に 100 円を追加しています。stateUpdate()メソッドを呼ばないと自動更新されません。(その場合、更新ボタンを押した時だけ、更新されます。)

練習問題

＜記述問題＞

☆記述問題 14-1

イベント･ドリブン型プログラミングの利点を自分の言葉で説明せよ。

☆記述問題 14-2

Model、View とは何か。分離する利点とともに自分の言葉で説明せよ。

＜プログラム問題＞

☆プログラム問題 14-1

「例題 14-5：テキストフィールドと連動する」を改造し、ボタンが押されるたびに「x 回ボタンが押されました」とテキストフィールドに表示できるようにせよ。

☆プログラム問題 14-2

「例題 14-7：100 円投入ボタンの追加」の GUI プログラム全ソースが exercise14_2 フォルダに入っている。しかし、このプログラムは View 部分のみが完全なソースになっており、Model 部分のメソッドがすべて空になっている。前回までに作った Model と結合し、display()メソッドを変更することによって、自分で Model を作り、配られた View と結合せよ。

☆プログラム問題 14-3

「問題 14-2」を改造し、10 円投入ボタン、50 円投入ボタンを追加せよ。

ヒント：UserFrame クラスだけ書き換えるだけでよい。

☆プログラム問題 14-4

「問題 14-3」を改造し、購入ボタンが押された時に商品が取り出し口に出てくるようにせよ。

ヒント：

- UserFrame クラスにある、buyButton_pressed()メソッドがイベント･ハンドラになっているので、中身を実装する。
- 次ページに、コメントを示すので参考にしてよい。