第15回　　オブジェクトの状態とその遷移

～GUI を使ったプログラム(Ⅱ)～

学習目標

- オブジェクトの状態遷移について議論できる
  - 状態遷移図が読める
  - 簡単な状態遷移図が書ける
- 状態遷移図を基にして正しいプログラムが書ける

15.1. ユーザビリティーの高い自動販売機へ

前回ついに GUI の自動販売機が完成しました。しかし、使いにくい点が一つあります。購入ボタンを押したのに出てこない時があることです。

押したのに出てこないのはバグではありません。前回作ったプログラムが正しくできていれば、在庫や投入金のチェックが行われ、在庫がなければ、当然商品を出すことはできません。

問題は、ボタンを押してから「在庫が足りません」、「投入金が足りません」と怒られることです。そんな使いにくい自動販売機を使いたいと思わないでしょう？

みなさんも、例えば、テキストエディタで、「切り取り」ボタンを押してから、「選択範囲が指定されていません」と怒られたことがあるでしょう？ボタンを押してから怒るのではなく、選択範囲が指定されていなければ、ボタンは押せなくなっていたほうがよいと思いませんか？

①.表示でお知らせ

今回は、ユーザビリティーを高くするために、購入ボタンの表示で買えるか買えないかをお知らせすることにします。買える時はランプを点灯させ、買えないときはランプを消灯します。

買える時

買えない時

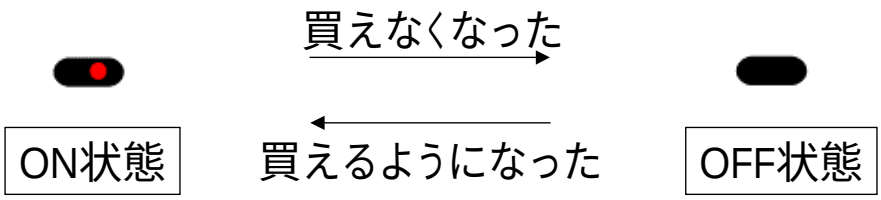
15.2. オブジェクトの状態と遷移

15.2.1.　ボタンの状態を考える

買えるか買えないかをボタンの表示でお知らせするプログラムはどうしたら書くことができるでしょうか。

買える時、買えない時を正しいタイミングで正しく判断し、表示を変えるプログラムを書く必要があります。表示がおかしい場合、ユーザビリティーが高くなるどころか、ユーザは激怒し、使ってくれないソフトウエアになってしまいます。

このようなプログラムを書くときは、「状態」という概念を考えることが重要です。「状態」を正しく設定し、その移り変わり（遷移）を正しくプログラミングすることによって、ユーザビリティーの高いソフトウエアを作ることができます。

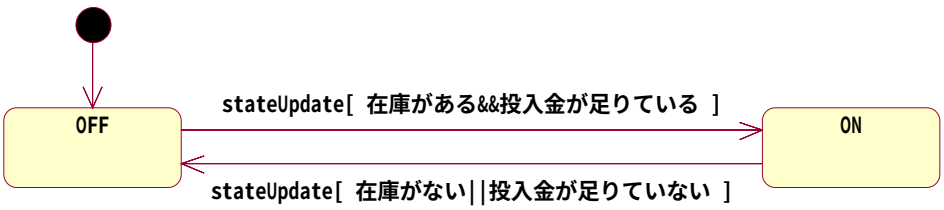


15.2.2.　状態遷移図

状態を伴うプログラムを書くときは、状態遷移図を書くことが重要です。今回は、ON、OFF というシンプルなモデルを扱いますが、複雑なプログラムになればなるほど、状態とその遷移を図式化してから、プログラムを書くことが重要となってきます。

UML を用いて購入ボタンの状態遷移図を書いてみましょう。

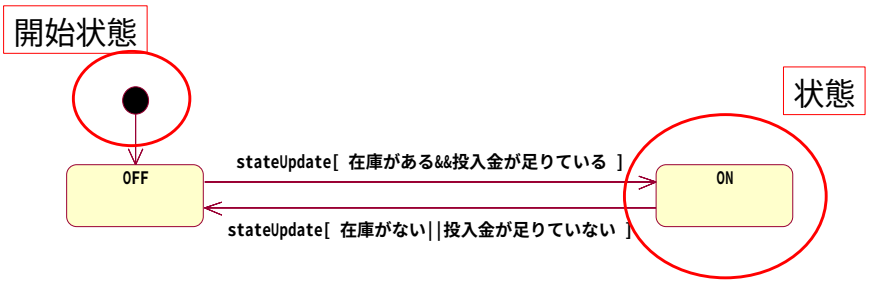
①.購入ボタンの状態遷移図



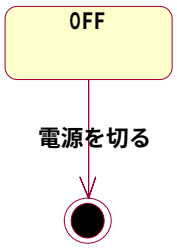
②.状態遷移図の記法

(1)状態

状態には状態名が書かれます。分かりやすい状態名をつけましょう。開始状態は当然ひとつしか在ってはなりません。状態はいくつあってもかまいません。

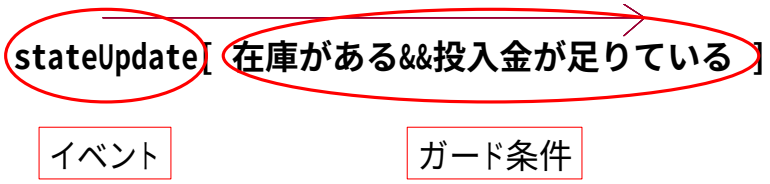


今回は出てきませんが、「終了状態」を記述したい時は、次のような記号を使います。



(2)遷移

状態と状態の間を矢印で結んで遷移を表します。遷移には、イベントとガード条件が書かれます。



● イベント

状態遷移のきっかけとなるイベントです。オブジェクトの状態は何らかのイベントが起こったときだけ遷移します。勝手に遷移することはありません。遷移するタイミングは非常に重要ですので、どのようなイベントがきた時に遷移するのかを明らかにするために書きます。イベント名を書いておきます。Java では、メソッドが呼ばれた時なので、メソ

ッド名を書いておけばよいでしょう。

● ガード条件

状態が遷移する条件のことです。イベントを受け取っても、条件が合わない場合は、状態遷移をしない場合があります。無条件に遷移する場合は書く必要がありません。

まとめると、イベントが起こり、かつガード条件を満たしている時に状態が遷移するということになります。

15.2.3.　状態遷移のプログラミング

状態遷移図ができたら、いよいよ状態遷移のプログラミングです。今回は、購入ボタンの状態遷移プログラムを書きますので、自動販売機用 GUI コンポーネントが入っている gui ディレクトリ（パッケージ）の中の BuyButton というクラスを変更していきます。

例題 15-1：購入ボタンの状態遷移(BuyButton.java)

```
1:      package gui;
2:
3:      import ItemType;
4:      import UserFrame;
5:      import Account;
6:
7:      import javax.swing.plaf.ComponentUI;
8:      import javax.swing.*;
9:      import java.awt.*;
10:     import java.awt.event.*;
11:
12:     /**
13:     * オブジェクト指向哲学入門 教材
14:     *
15:     * 購入ボタン　クラス
16:     */
17:     public class BuyButton extends JComponent{
18:
19:       //背景画像の名前
20:       private static final String onImgName = "buybuttonon.gif";
21:       private static final String offImgName = "buybuttonoff.gif";
22:       private static final String soldoutImgName = "buybuttonsoldout.gif";
23:
24:       private static final int width = 43;//このコンポーネントの大きさ横
25:       private static final int height = 20;//このコンポーネントの大きさ縦
26:
27:       //状態定数
```

```
28:     public static final int ON = 0;
29:     public static final int OFF = 1;                                    A
30:     public static final int SOLDOUT = 2;
31:
32:     private int state = OFF;      //現在の状態
33:     private ItemType itemType;  //対応する商品種類
34:     private Account account;    //投入金勘定                            B
35:     private UserFrame userFrame;//ユーザフレーム
36:
37:     /**
38:     * コンストラクタ
39:     */
40:     public BuyButton(UserFrame newUserFrame,ItemType newItemType,Account newAccount){
41:
42:       itemType = newItemType;
43:       account = newAccount;
44:       userFrame = newUserFrame;
45:
46:       //パネルの設定
47:       this.setUI(new BuyButtonUI());
48:       this.setPreferredSize(new Dimension(width,height));
49:
50:       //マウスが押されたときの　イベント･ハンドラの登録
51:       this.addMouseListener(new BuyButton_MouseListener());
52:
53:       stateUpdate();
54:     }
55:
56:     /**
57:     * 状態を反映して、表示を更新する
58:     */                                                                  C
59:     public void stateUpdate(){
60:
61:       //状態が OFF の時
62:       if(state == OFF){
63:         //ガード条件
64:         if(!itemType.getItemStock().isEmpty() &&
65:         account.getAmount() >= itemType.getPrice()){//在庫がある&&投入金が足りている
66:           //遷移する
67:           state = ON;
68:           repaint();
69:         }
70:       }
71:
72:       //状態が ON の時
73:       else if(state == ON){
74:         //ガード条件
75:         if(itemType.getItemStock().isEmpty() ||
76:         account.getAmount() < itemType.getPrice()){//在庫がない||投入金が足りていな
い
77:           //遷移する
78:           state = OFF;
79:           repaint();
80:         }
```

```
81:         }
82:       }
83:
84:       /**
85:       * このボタンが押された時のイベント･ハンドラ
86:       */
87:       class BuyButton_MouseListener extends java.awt.event.MouseAdapter{
88:         public void mouseClicked(MouseEvent e) {
89:           if(state == ON){
90:             userFrame.buyButton_pressed(BuyButton.this,itemType);//ユーザフレームのハ
ンドラを呼ぶ
91:           }
92:         }
93:       }
94:
95:       // --------------BuyButton クラス本体ここまで----------------
96:
97:       /**
98:       * 描画アルゴリズムを持つ内部クラス
99:       */
100:      class BuyButtonUI extends ComponentUI {
101:
102:        //背景画像
103:        private Image onImg;
104:        private Image offImg;
105:        private Image soldoutImg;
106:
107:        /**
108:        * コンストラクタ
109:        */
110:        public BuyButtonUI() {
111:          //画像を読み込む
112:          onImg = ImageProvider.getInstance().getImage(onImgName);
113:          offImg = ImageProvider.getInstance().getImage(offImgName);
114:          soldoutImg = ImageProvider.getInstance().getImage(soldoutImgName);
115:        }
116:
117:        /**
118:        * 画面に描画する
119:        */
120:        public void paint(Graphics g,JComponent c){
121:          if(state == ON){         //状態が ON なら ON の画像を描画
122:            g.drawImage(onImg,0,0,c);
123:          }else if(state == OFF){ //状態が OFF なら OFF の画像を描画
124:            g.drawImage(offImg,0,0,c);
125:          }else if(state == SOLDOUT){//状態が SOLDOUT なら SOLDOUT の画像を描画
126:            g.drawImage(soldoutImg,0,0,c);
127:          }
128:        }
129:      }
130:
131:   }
```

状態遷移のプログラミングには様々な方法がありますが､今回は非常にシンプルな方法で実装します。

(1)A：状態を定義する

定数を定義して、ON 状態と OFF 状態を定義します。0 が ON で 1 が OFF だと覚えておいてもよいのですが、後で分からなくなりますし、他人が読んだ時わからないので、定義しておきます。

※変数に「final」修飾子をつけると変更できなくなります。つまり、定数になります。

(2)B：状態を格納する変数

これは、単純に整数型の変数を用意して、現在の状態を保存しておきます。

(3)C：遷移するプログラム

状態遷移図に従って、遷移のプログラムを書いてあります。今回の状態遷移図では、stateUpdate というイベントがきた時にだけ遷移することになっているため、stateUpdate()メソッドの中だけに遷移のプログラムが書かれます。

15.2.4. 状態遷移 まとめ

＜考えよう！＞状態遷移図を書くことの利点を挙げてみよう

●
●
●

練習問題

＜記述問題＞

☆記述問題 15-1

状態が「ON」と「OFF」だけでは、ユーザは「金を十分に入れたのに OFF のまま」と怒ってしまう。購入ボタンに「売り切れ」状態を追加して、状態遷移図を書いてみよ。

☆記述問題 15-2

状態遷移図を書くことの利点を自分なりに説明せよ。

＜プログラム問題＞

☆プログラム問題 15-1

記述問題 15-1 で書いた状態遷移図を参考に、「例題 15-1：購入ボタンの状態遷移」を改造して「売り切れ」状態を表示するプログラムにせよ。

☆プログラム問題 15-2

プログラム問題 15-1 をさらに発展させて、「投入金の Undo ボタン」と「取り出し口から商品取り出しボタン」を付けて、GUI 自動販売機を完成させよ。