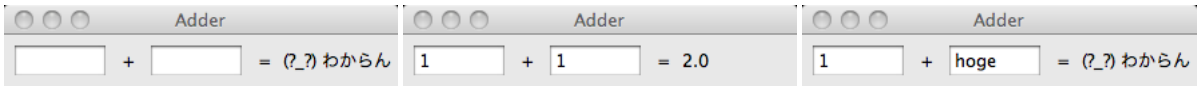
- テキストフィールドのイベント
- マウスイベント
- イベント駆動

★10　イベント処理（承前）

★10.4　もういっぱい—テキストフィールドのイベント



今度は，テキストフィールド（キー入力できる箱）を使ってみよう．Swing の JTextField を用いることにする (☆ 1)．JTextField のようなコンポーネントは，その上で Enter キーが押されたときに ActionEvent を発生させる．

Q1. ということは，リスナーには何を実装すればよいのだろう？

☆1) JTextField の使い方（コンストラクタの書き方など）は Java API 参照．

Adder.java（import 文，main メソッドは省略）

```
 :
 5  public class Adder extends JPanel implements ActionListener{
 6
 7      JTextField aField, bField;  // テキストフィールド
 8      JLabel sumLabel;            // 足し算結果表示用ラベル
 9
10      public Adder(){
11
12          aField = new JTextField(5);
13          bField = new JTextField(5);
14          sumLabel = new JLabel("(?_?) わからん");
15
16          aField.addActionListener(this);
17          bField.addActionListener(this);
18
19          setLayout(new FlowLayout(FlowLayout.LEFT));
20          add(aField); add(new JLabel(" + "));
21          add(bField); add(new JLabel(" = "));
22          add(sumLabel);
23      }
24
25      public void actionPerformed(ActionEvent e){
26
27          double a, b;
28
29          // try-catch 文は「例外処理」のためのしくみ（cf. 教科書第 9 章）. try ブロック内で
30          // 「数値の形式が変だよという例外」が発生したら catch ブロック内が実行される.
31          try{
32              // テキストフィールドに入力された文字列を double に変換
33              a = Double.parseDouble(aField.getText());
34              b = Double.parseDouble(bField.getText());
35              sumLabel.setText( String.valueOf(a + b) );
36          }catch(NumberFormatException error){
37              // 上記の変換がうまくいかなかったとき
38              a = b = 0.0;
39              sumLabel.setText("(?_?) わからん");
40          }
41      }
 :
53  }
```

★10.5 別腹？—マウスイベント (p.171)

次に，マウスカーソルの動きやマウスボタンのクリックに応じたイベント処理の方法を取り上げる．前回の復習をかねて，そのようなイベント処理のプログラムを書くための下調べをしよう．

1. どのようなイベント（どのイベントクラス）を扱うのか，それには何というリスナーインターフェースを用いるのかを調べる (p.162 表 13.1)
2. そのリスナーでは何というイベント処理メソッドを定義する必要があるのかを調べる (p.164 表 13.2)
3. そのイベントの情報を取得するためのメソッドを調べる (p.165 表 13.3)

以下のプログラムは，マウスボタンが押された時と離された時の座標を変数に覚えておき，その 2 点間を結ぶ線分を描くものである．このプログラムの場合，上記の下調べの結果は次のようになるだろう．

1.

2.

3.

DrawByMouse.java の一部（p.171 のものとはちょっとだけ違う）

```
 :
 5  public class DrawByMouse extends JPanel implements MouseListener {
 6      int x1, y1, x2, y2;

 7      public DrawByMouse(){
 8          setBackground(Color.white);
 9          setPreferredSize(new Dimension(250, 250));
10          addMouseListener(this);
11      }

12      protected void paintComponent(Graphics g){
13          super.paintComponent(g); // 背景色でパネルを塗りつぶす
14          g.drawLine(x1,y1,x2,y2); // (x1,y1) と (x2,y2) を線で結ぶ
15      }

16      public void mousePressed(MouseEvent e){ //マウスを押した時の処理
17          x1 = e.getX();
18          y1 = e.getY();
19          System.out.println("Pressed! (" + x1 + ", " + y1 + ")");
20      }

        // マウスボタンを離した時の処理
21      public void mouseReleased(MouseEvent e){
22          x2 = e.getX();
23          y2 = e.getY();
24          repaint();      //ボタンを離したら再描画（paintComponent が呼ばれる）
25          System.out.println("Released! (" + x2 + ", " + y2 + ")");
26      }
```

(次頁へ)

前頁のつづき

(前頁より)

```
        // マウスボタンがクリックされた（ボタンを押して離した）
27      public void mouseClicked(MouseEvent e){   }
        // コンポーネントの領域にカーソルが入った (Entered), 出た (Exited)
28      public void mouseEntered(MouseEvent e){   }
29      public void mouseExited(MouseEvent e){   }
30
 :    (main メソッドはおなじみの構成なので省略)
39  }
```

次の例は，マウスカーソルが動いたときにその座標を取得して処理を行うプログラムである．MouseListener のかわりに MouseMotionListener を用いている．

マウスカーソルに●がついてくる〜

```
 :
 5  public class MouseCursor extends JPanel implements MouseMotionListener {
 6      int x = 0, y = 0;

 7      public MouseCursor(){
 8          setBackground(Color.white); // パネルの背景色
 9          setForeground(Color.red);   // パネルの描画色
10          setPreferredSize(new Dimension(250, 250));
11          addMouseMotionListener(this);
12      }

13      protected void paintComponent(Graphics g){
14          super.paintComponent(g);
15          g.fillOval(x - 10, y - 10, 20, 20);
16      }

17      // マウスカーソルが移動した時
18      public void mouseMoved(MouseEvent e){
19          x = e.getX();
20          y = e.getY();
21          repaint();
22          System.out.println("(" + x + ", " + y + ")");
23      }
24      // マウスドラッグ時（ボタンを押したまま移動)
25      public void mouseDragged(MouseEvent e){   }
26
 :    (またもや main メソッドはおなじみの構成なので省略)
35  }
```

Q2. 14 行目で super.paintComponent() を呼んでいるのをやめるとどうなるか観察しなさい．

さらに次の例は，MouseListener と MouseMotionListener の両方を用いている．このように，複数のリスナーを同時に実装してイベント処理を行うことも可能である（もちろん種類の異なる複数のイベントを一つのプログラムであわせて処理することもできる）．

さらに，クリックすると●の色がぁ〜

●「上記 MouseCursor.java の 5,6 行目を次のように書きかえる」

```
public class MouseCursor extends JPanel implements MouseListener, MouseMotionListener {
    int x = 0, y = 0;  boolean on = false;
```

●「10 行目と 11 行目の間に以下を追加」

```
    addMouseListener(this);
```

●「適当な場所に以下を追加」

```
    public void mouseClicked(MouseEvent e){
        on = !on;  // 真偽反転
        if(on) setForeground(Color.blue);
        else   setForeground(Color.red);
    }
    public void mousePressed(MouseEvent e){  }
    public void mouseReleased(MouseEvent e){  }
    public void mouseEntered(MouseEvent e){   }
    public void mouseExited(MouseEvent e){   }
```

★10.6　イベント駆動

低学年で学んできたような基礎的な C 言語のプログラムや，この授業の前半で学んだような Java のプログラムでは，プログラム作成者がプログラム中で指示した通りの順に処理が進んでいく (☆ 2)．条件分岐によって処理の流れが変わることはあるが，それも作成者の意図したことである．しかし，イベント処理を行うプログラムの場合は事情が異なる．この場合，イベントディスパッチャと呼ばれるものがプログラムの裏で動作しており，発生したイベント毎に適切なイベントリスナーを選択して処理を行わせるようになっている．どのような処理がどのような順で行われるかは，イベントの発生の仕方次第で変化する．このような仕組みに応じたプログラミングを行うことを，**イベント駆動型プログラミング**という．

☆ 2) 実際には，タートルグラフィックスのプログラムでも TurtleFrame クラスなどではイベント処理を行っているので，正確にはこの例にあてはまらない．

イベント駆動型の: event-driven.

宿題？

- 授業中に解説されなかった Q を解いてみよう．
- 次回はこれまでの復習です．