ソフトウェア工学実習
Software Engineering Practice
(第04回)

SEP04-001　第02回の復習

慶應義塾大学・理工学部・管理工学科
飯島 正
iijima@ae.keio.ac.jp

これまでの復習

第03回（イベント処理）

話の流れ（第03回の復習）

第01回にzipで配布したP0104パッケージを確認する

↓

イベント処理のための委譲に基づくリスナモデル

↓

NetBeansを使った場合（CounterFrameを確認する）

iijima@ae.keio.ac.jp

話の流れ（第03回の復習）

第01回にzipで配布したP0104パッケージを確認する

↓

イベント処理のための委譲に基づくリスナモデル

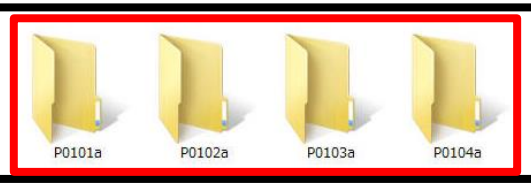
↓

NetBeansを使った場合（CounterFrameを確認する）

iijima@ae.keio.ac.jp

GUI (Graphical User Interface) 最初の一歩

第01回の授業では，zipファイルで例題プログラムを配布しました



例題0101: 単純なフレーム：　　P0101パッケージ

例題0102: ラベル付きフレーム：P0102パッケージ

例題0103: ボタンの表示:　　P0103パッケージ

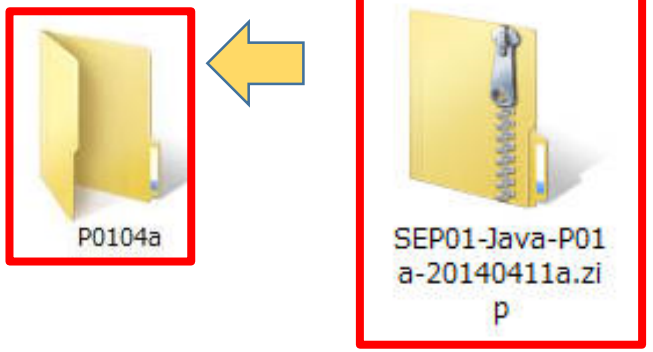
例題0104: ボタンのアクション:　P0104パッケージ

この手順で
実習を
行いました

第01回の授業では，zipファイルで例題プログラムを配布しました

ボタンのアクション:P0104パッケージ(ButtonFrameAクラス)
↑第01回にzipファイルで配布


```
package p0104;
// ================================
import javax.swing.*;
import java.awt.*;
// ========================================
public class ButtonFrameA extends JFrame {
    private JLabel  aLabel; //ラベル
    private LikeButton aButton; //ボタン
    public ButtonFrameA() { //コンストラクタ(インスタンス生成時の初期設定)
        … }
    public static void main( String[] args ) { //メイン・メソッド
        … }
}
// ============================================
```



ボタンを
クリックすると
カウントアップ
します

ボタンのアクション:P0104パッケージ(ButtonFrameAクラス)
↑第01回にzipファイルで配布

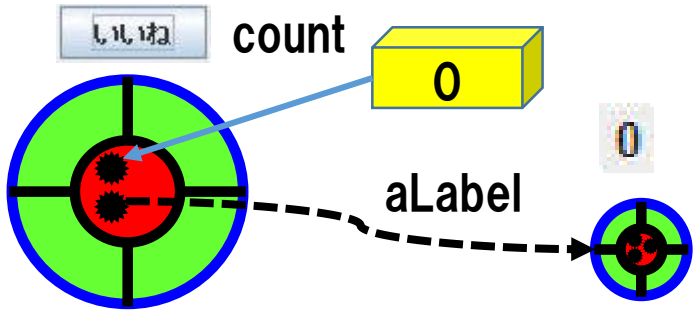

```
package p0104;
// ================================
import javax.swing.*;
import java.awt.*;
// ================================================
public class ButtonFrameA extends JFrame {
   private JLabel  aLabel; //ラベル
   private LikeButton aButton; //ボタン
}
// ================================================
```

属性aLabelとaButtonの値は,
それぞれのオブジェクトへの参照である.

aLabelとaButtonは,
クラス参照型の
変数です

ボタンのアクション:P0104パッケージ(LikeButtonクラス)
↑第01回にzipファイルで配布


```
package p0104;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
// ================================================
class LikeButton extends JButton implements ActionListener {
   // ================================================
   private int count = 0; // カウンタ(整数)
   private JLabel aLabel; // カウントを表示するラベル(への参照)
   // ================================================
   public LikeButton( JLabel aLabel ) { //コンストラクタ
        ...
        addActionListener(this);
         ...
   }
   // ================================================
   public void actionPerformed( ActionEvent e ) { // イベント・ハンドラ
      ... }
}
```

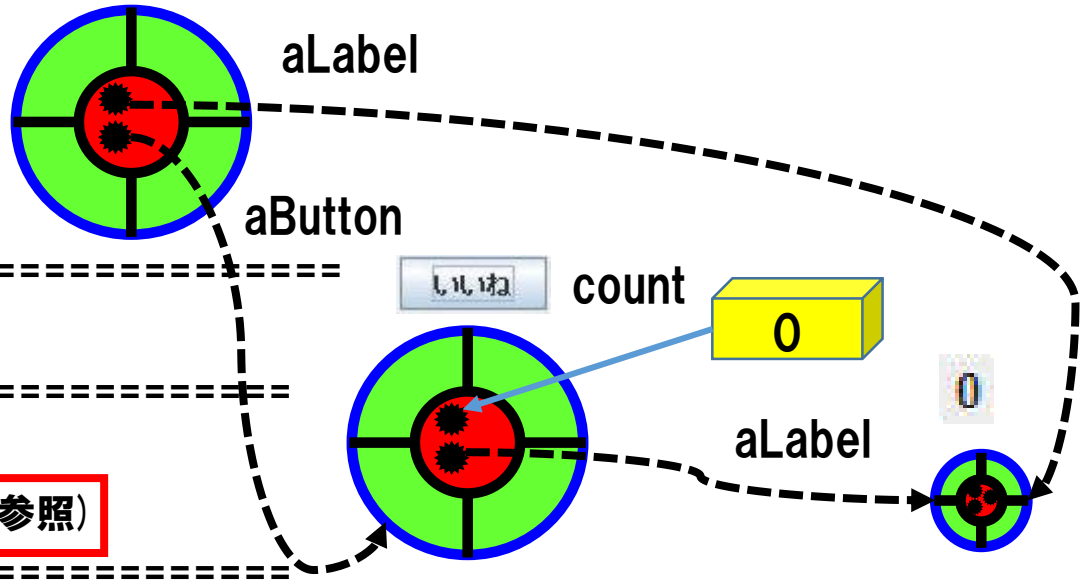


Likeボタンが
内部状態として
Count変数を
持っています.

iijima@ae.keio.ac.jp

ボタンのアクション:P0104パッケージ(LikeButtonクラス)
↑第01回にzipファイルで配布

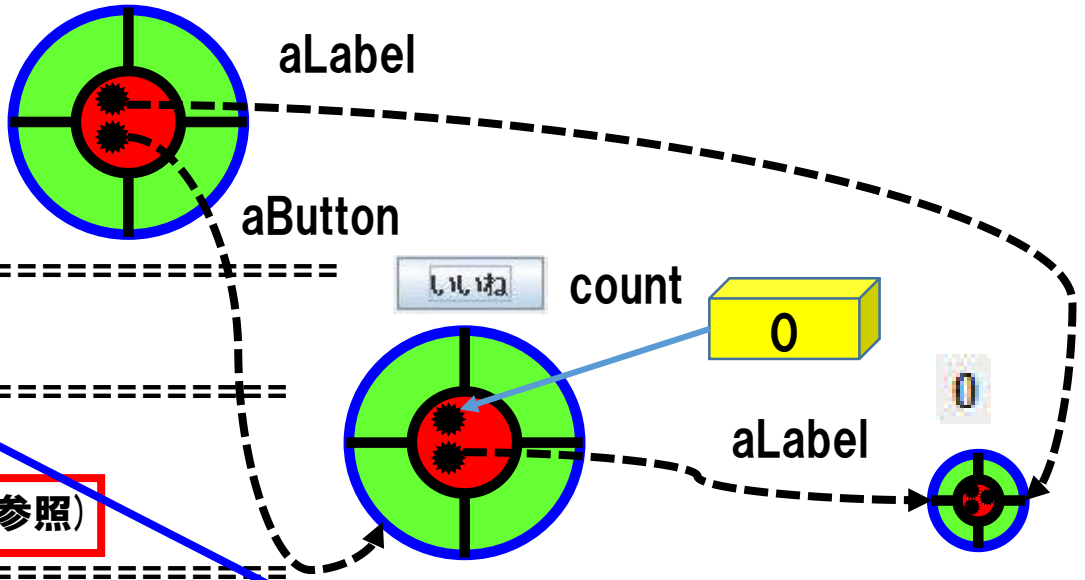

```
package p0104;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
// ================================================
class LikeButton extends JButton implements ActionListener {
  // ================================================
  private int count = 0; // カウンタ(整数)
  private JLabel aLabel; // カウントを表示するラベル(への参照)
  // ================================================
  public LikeButton( JLabel aLabel ) { //コンストラクタ
      ...
      addActionListener(this);
       ...
  }
  // ================================================
  public void actionPerformed( ActionEvent e ) { // イベント・ハンドラ
     ... }
}
```



ラベルオブジェクト
への参照は
ButtonFrameAと
LikeButtonで
共有します

ボタンのアクション:P0104パッケージ(LikeButtonクラス)
↑第01回にzipファイルで配布


```
package p0104;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
// ================================================
class LikeButton extends JButton implements ActionListener {
  // ================================================
  private int count = 0; // カウンタ(整数)
  private JLabel aLabel; // カウントを表示するラベル(への参照)
  // ================================================
  public LikeButton( JLabel aLabel ) { //コンストラクタ
    ...
    addActionListener(this);
    ...
  }
  // ================================================
  public void actionPerformed( ActionEvent e ) { // イベント・ハンドラ
    ... }
}
```



自分自身(this)を,
自分のActitonListerとして
登録しています

ActitonLister
インタフェース
を実装しています
(インタフェース
については,
今後説明します)

ここで,
イベント処理が
定義されています

ボタンがクリックされた時に
発生するActionEventに対して,
反応するイベントハンドラ

iijima@ae.keio.ac.jp

話の流れ（第03回の復習）

第01回にzipで配布したP0104パッケージを確認する

↓

イベント処理のための委譲に基づくリスナモデル

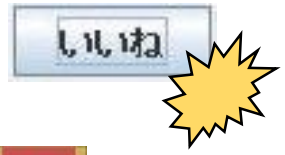
↓

NetBeansを使った場合（CounterFrameを確認する）

iijima@ae.keio.ac.jp

第03回目の講義で分かってほしかったこと: イベント処理
イベント駆動プログラム

- イベント処理
  - ボタンがクリックされると，そのことを意味する
    アクションイベント
    が発生します．
  - そして，それに反応して，**イベントハンドラ**と呼ばれるメソッドが起動されます．
    - しかし，**明示的**にイベントハンドラを呼び出すメッセージを記述することはありません．
    - でも，オブジェクト指向というのは，オブジェクト間の**メッセージ渡し**が基本です．
    - 一体どうなっているのでしょう？
    - 実は，次で説明する「移譲に基づくリスナ・モデル」で**暗黙的**に（裏方で）メッセージが送られています．



イベントハンドラ

ボタンがクリックされたときの実行メカニズム


```
public void actionPerformed ( ActionEvent e ) {
        count++;
        aLabel.setText (Integer.toString ( count ) );
}
```

第03回目の講義で分かってほしかったこと: イベント処理
イベント駆動プログラム

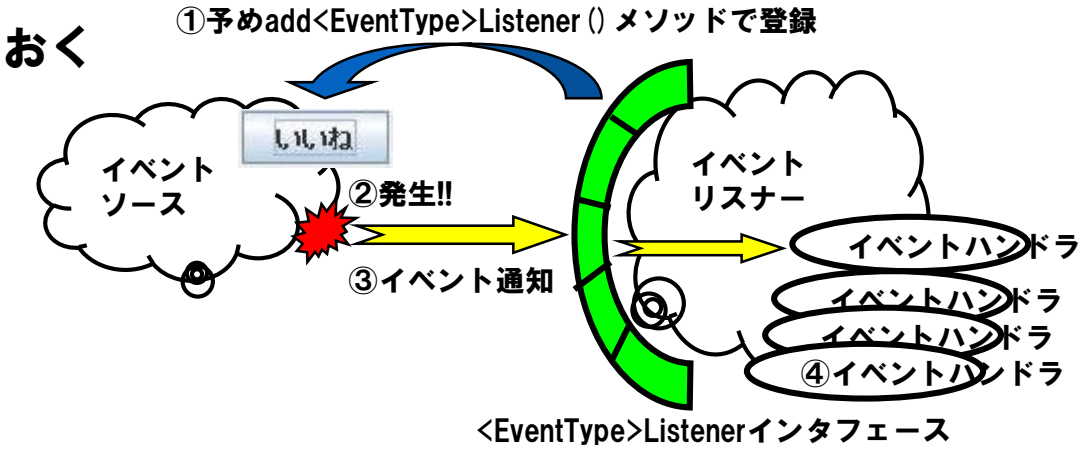
- 手順

①予めコンポーネントにイベントリスナを登録しておく

②そのコンポーネント（イベントソース）で，
イベントが発生する
（例えばボタン上でマウスボタンが押される）

③イベントソースに登録されている
全イベントリスナに，そのイベントが通知される

④リスナで，対応するメソッド（イベントハンドラ）
が起動される

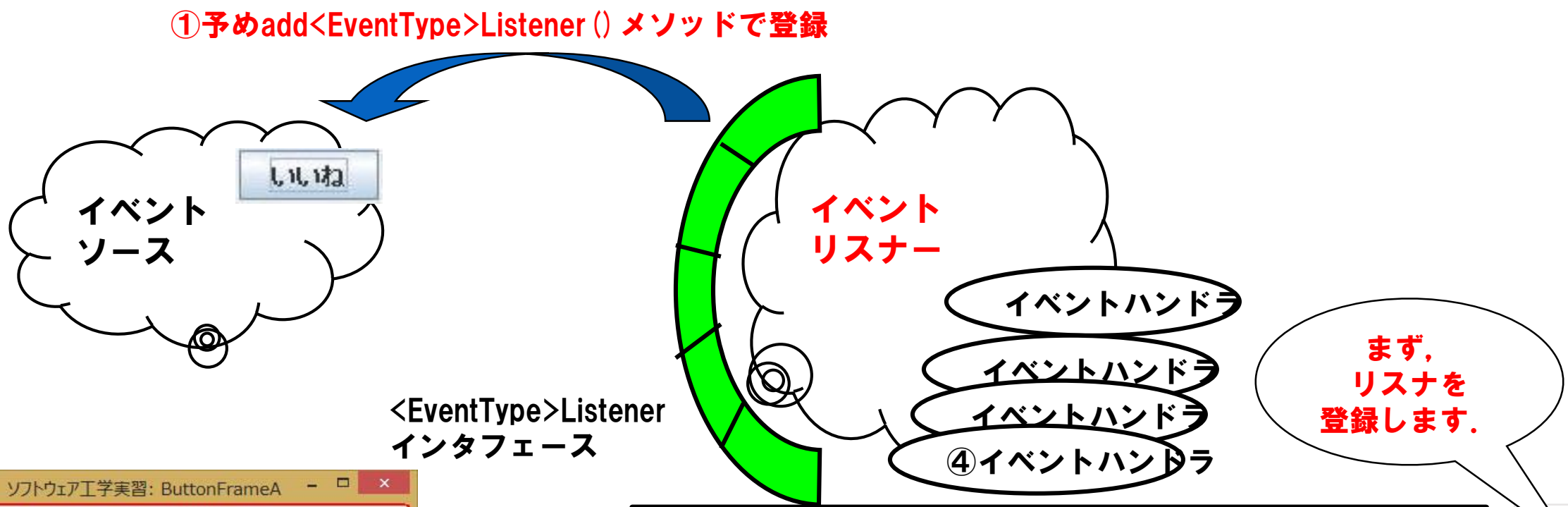


イベントをメッセージで表現する

```
public void actionPerformed ( ActionEvent e ) {
        count++;
        aLabel.setText (Integer.toString ( count ) );
}
```
イベントハンドラ

第03回目の講義で分かってほしかったこと: イベント処理
Delegation-based Listenerモデル（委譲に基づくリスナ・モデル）

- ①予めコンポーネントにイベントリスナを登録しておく



イベントをメッセージで表現する

```
public void actionPerformed ( ActionEvent e ) {
        count++;
        aLabel.setText (Integer.toString ( count ) );
}
```

第03回目の講義で分かってほしかったこと: イベント処理
Delegation-based Listenerモデル（委譲に基づくリスナ・モデル）

- ②そのコンポーネント（イベントソース）で，イベントが発生する（例えばボタン上でマウスボタンが押される）

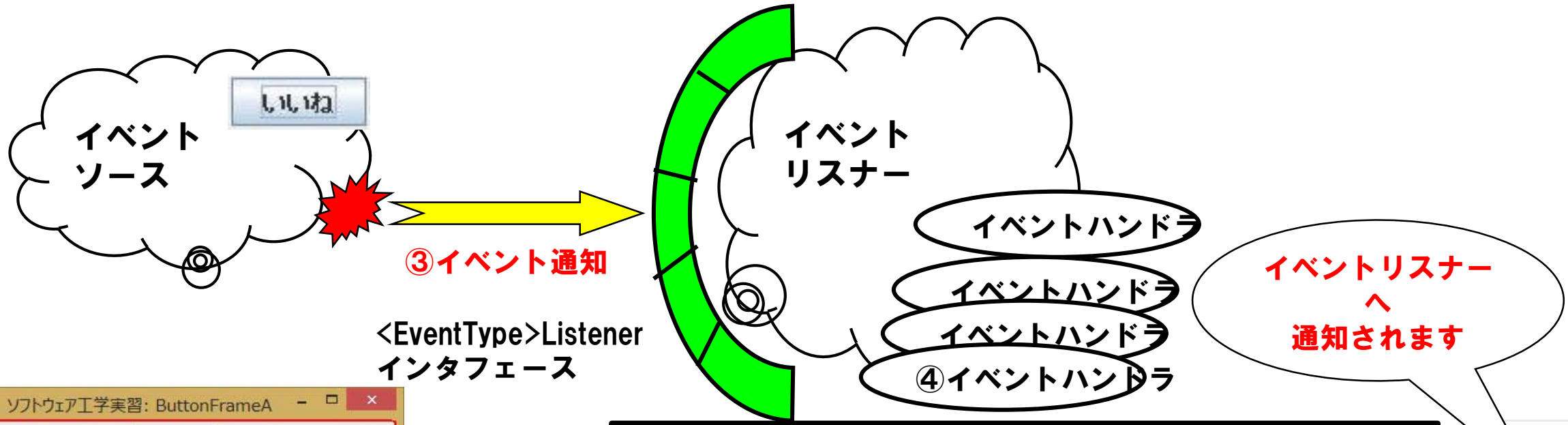
イベントをメッセージで表現する

```
public void actionPerformed ( ActionEvent e ) {
        count++;
        aLabel.setText (Integer.toString ( count ) );
}
```

iijima@ae.keio.ac.jp

第03回目の講義で分かってほしかったこと: イベント処理
Delegation-based Listenerモデル（委譲に基づくリスナ・モデル）

- ③イベントソースに登録されている全イベントリスナに，そのイベントが通知される

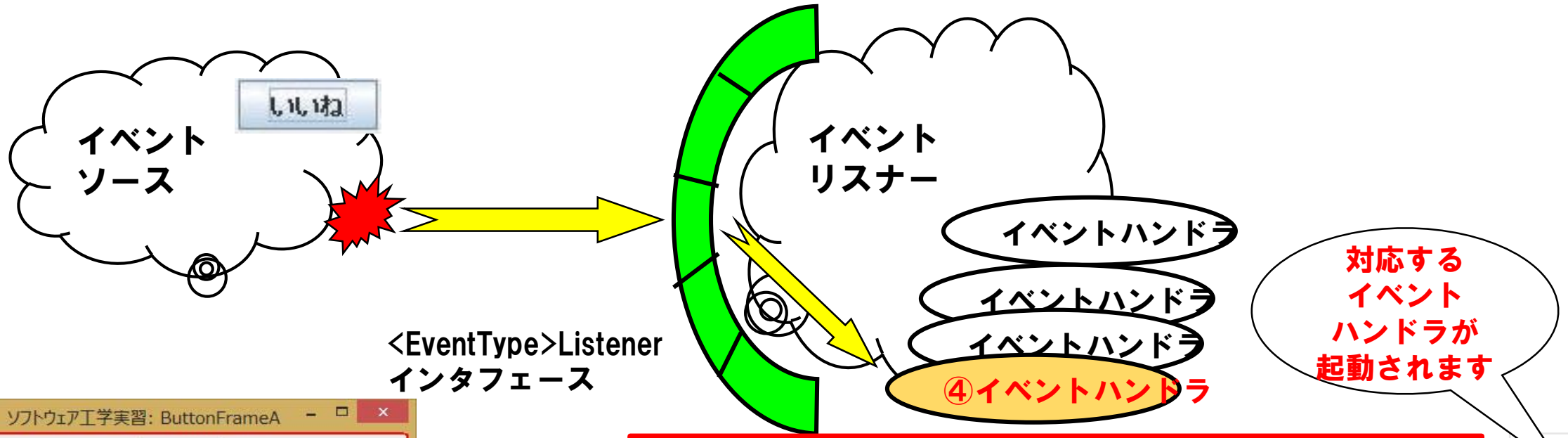


イベントをメッセージで表現する

```
public void actionPerformed ( ActionEvent e ) {
        count++;
        aLabel.setText (Integer.toString ( count ) );
}
```

第03回目の講義で分かってほしかったこと: イベント処理
Delegation-based Listenerモデル（委譲に基づくリスナ・モデル）

- ④リスナで，対応するメソッド（イベントハンドラ）が起動される



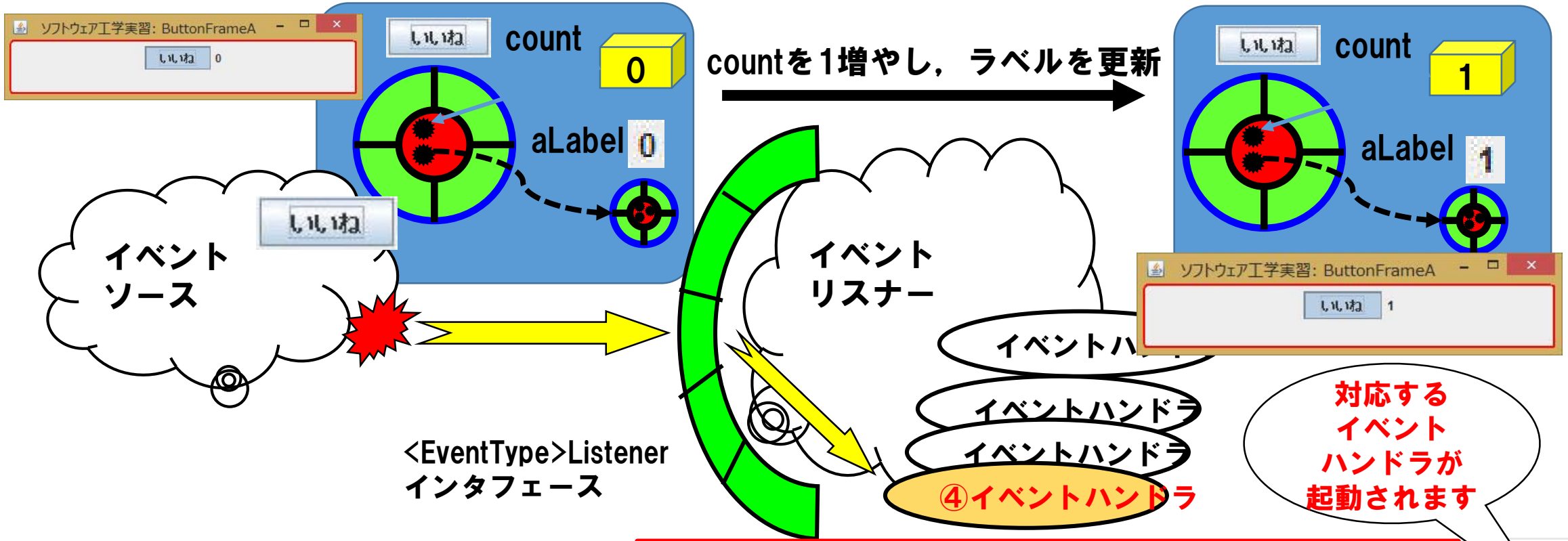
対応する
イベント
ハンドラが
起動されます

イベントをメッセージで表現する

```
public void actionPerformed ( ActionEvent e ) {
        count++;
        aLabel.setText (Integer.toString ( count ) );
}
```

第03回目の講義で分かってほしかったこと: イベント処理
Delegation-based Listenerモデル（委譲に基づくリスナ・モデル）

- ④リスナで，対応するメソッド（イベントハンドラ）が起動される

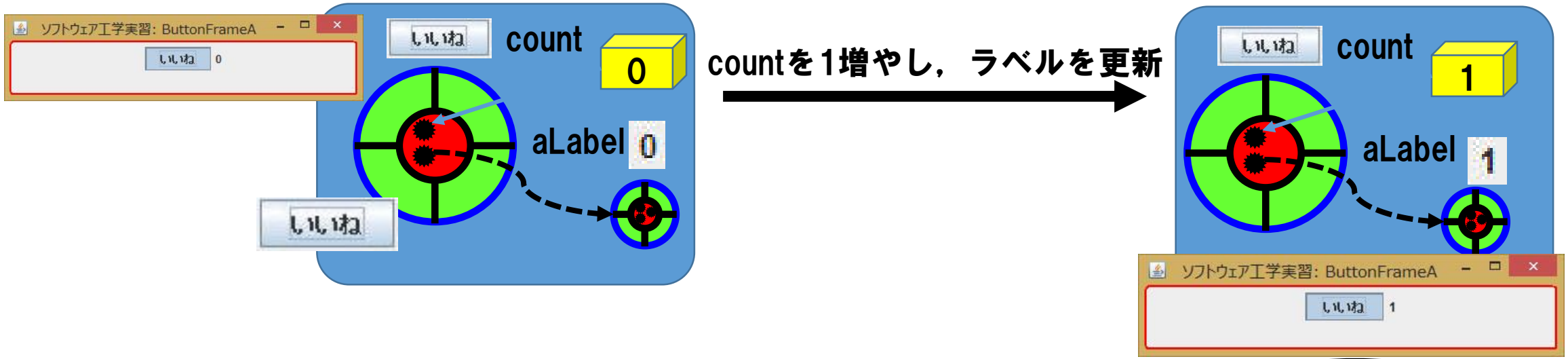



```
public void actionPerformed ( ActionEvent e ) {
        count++;
        aLabel.setText (Integer.toString ( count ) );
}
```

イベントをメッセージで表現する

iijima@ae.keio.ac.jp

第03回目の講義で分かってほしかったこと: イベント処理
Delegation-based Listenerモデル（委譲に基づくリスナ・モデル）



Countが増えて，ラベルが更新されます

```
public void actionPerformed ( ActionEvent e ) {
        count++;
        aLabel.setText (Integer.toString ( count ) );
}
```

iijima@ae.keio.ac.jp

第03回目の講義で分かってほしかったこと: イベント処理
イベント駆動プログラム

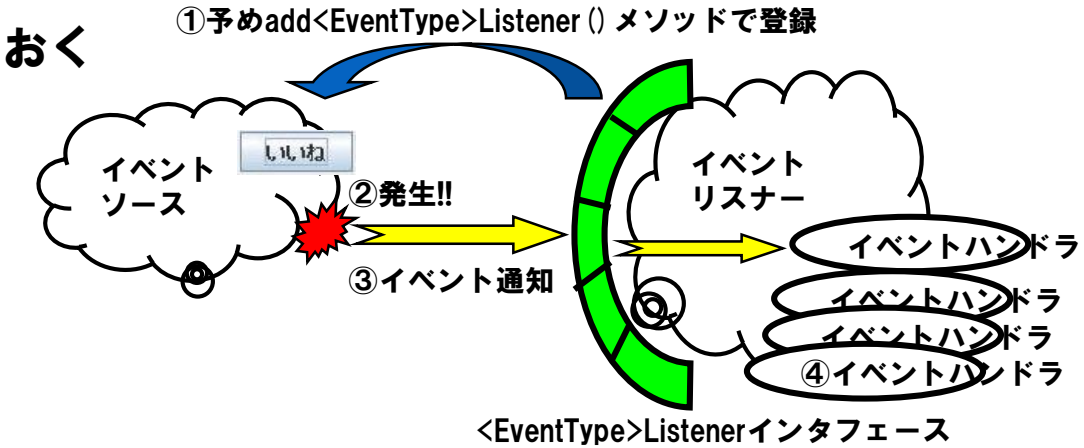
- 手順

①予めコンポーネントにイベントリスナを登録しておく

②そのコンポーネント（イベントソース）で，
イベントが発生する
（例えばボタン上でマウスボタンが押される）

③イベントソースに登録されている
全イベントリスナに，そのイベントが通知される

④リスナで，対応するメソッド（イベントハンドラ）
が起動される



イベントをメッセージで表現する

全体の流れです．

```
public void actionPerformed ( ActionEvent e ) {
        count++;
        aLabel.setText (Integer.toString ( count ) );
}
```
イベントハンドラ

話の流れ（第03回の復習）

第01回にzipで配布したP0104パッケージを確認する

↓

イベント処理のための委譲に基づくリスナモデル

↓

NetBeansを使った場合（CounterFrameを確認する）

iijima@ae.keio.ac.jp

第02/03回目の授業では，NetBeansを利用しました

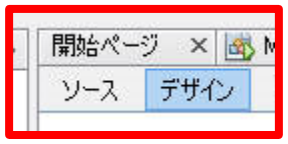
- JFrameフォームを選択し，フォームエディタ（デザインビュー）を使うことで非常に簡単に，イベント駆動プログラムを作ることができました．

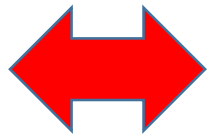
- ですが，
  NetBeansでは，手軽にイベント駆動プログラムが作れてしまいすぎる
  が故に，将来的な拡張を考えると，あまり良くないプログラムを作ってしまう傾向があります
  - →その点については，今回，伏線を引いておいて，次回考えることにしましょう

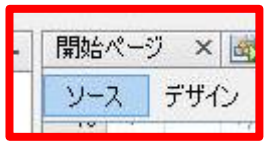
- また，イベントハンドラの名前や，リスナの登録について，zipファイルで配布した例題と異なっています
  - →この点については，今回，種明かしをしましょう

第03回目の授業で分かってほしかったこと:
イベント駆動（NetBeansを使うと…）

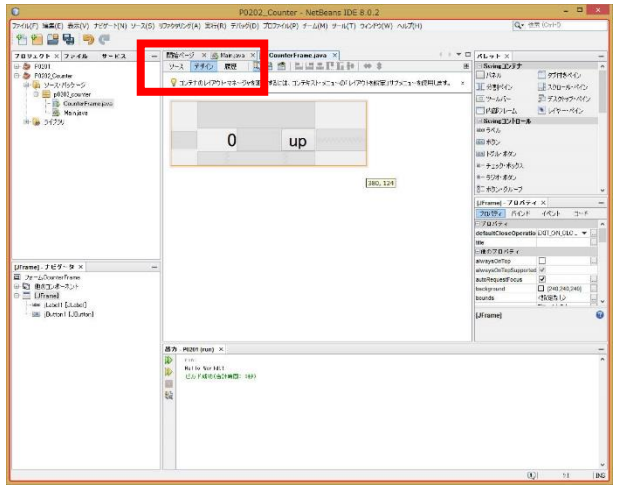
- 部品の配置　→　デザインビュー
- デザインビューで変数名を設定することで，
  ソースコードとデザインとがつながる．
- イベントハンドラ
  デザインビューで，**ボタンをダブルクリック**すると，
  ソースビューで，メソッド（**イベントハンドラ**）が生成される

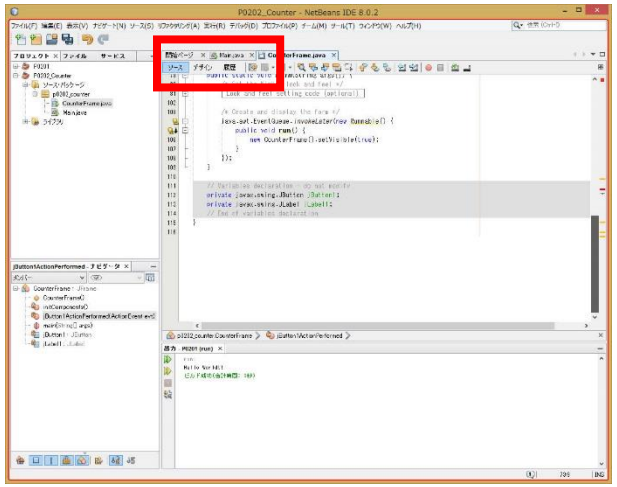






エディタツールバー



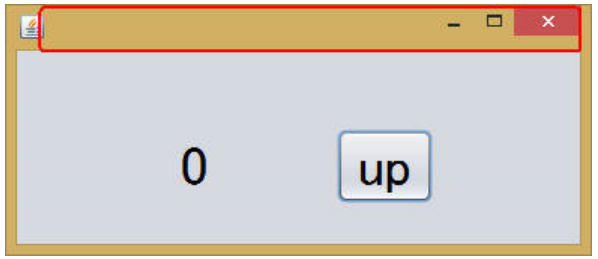


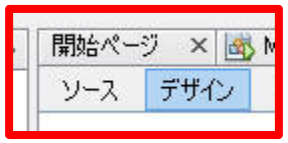
CounterFrame のプログラムをもう一度見てみましょう

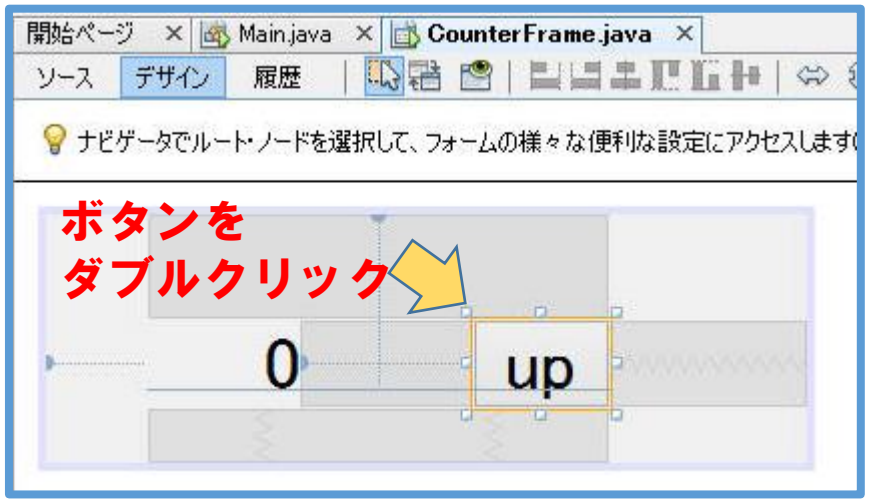
iijima@ae.keio.ac.jp

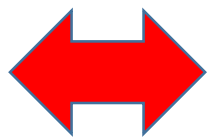
第03回目の授業で分かってほしかったこと:
イベント駆動（NetBeansを使うと…）

- 部品の配置　→　デザインビュー
- デザインビューで変数名を設定することで，
  ソースコードとデザインとがつながる．
- イベントハンドラ
  デザインビューで，**ボタンをダブルクリック**すると，
  ソースビューで，メソッド（**イベントハンドラ**）が生成される

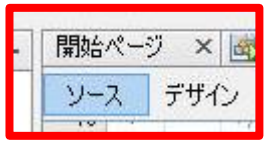




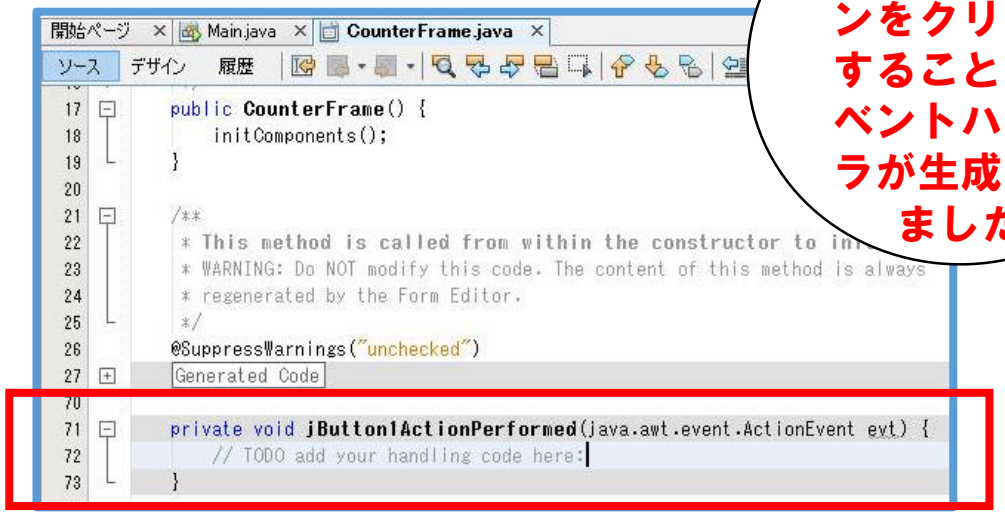




イベント
ハンドラ
の生成



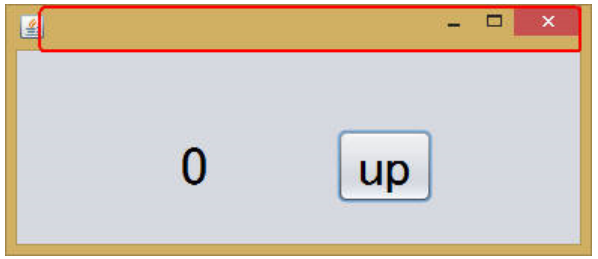
エディタツールバー

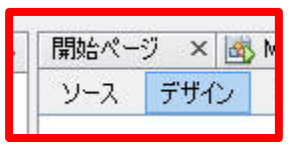


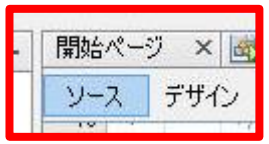
フォームエディタでボタンをクリックすることでイベントハンドラが生成されました

第03回目の授業で分かってほしかったこと: イベント駆動（NetBeansを使うと…）

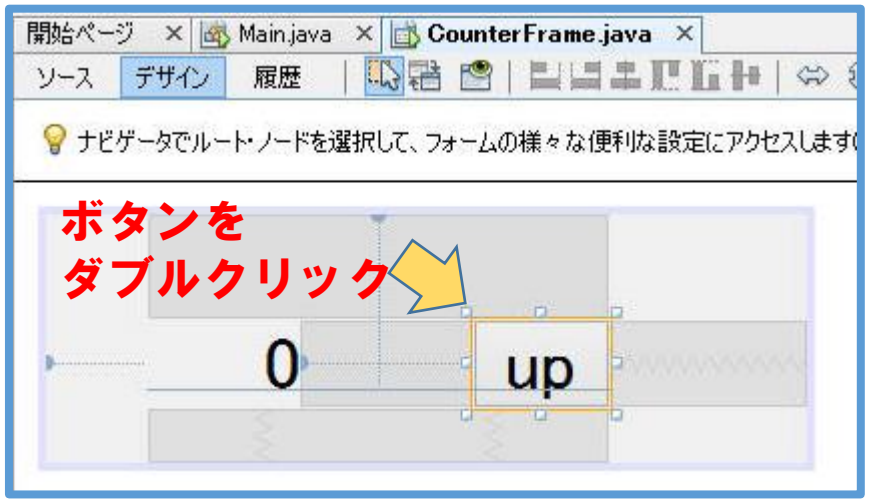
- でも，NetBeansを使った時には…
  - jButton1というボタンのクリック時のイベントハンドラの名前は，
    jButton1ActionPerformed (ActionEvent e)
    となっています．
  - アクションリスナーをボタンに登録した覚えもありません．
  - 簡単に使える点は有り難いのですが，何故これでよいのでしょう→今回，種明かしします



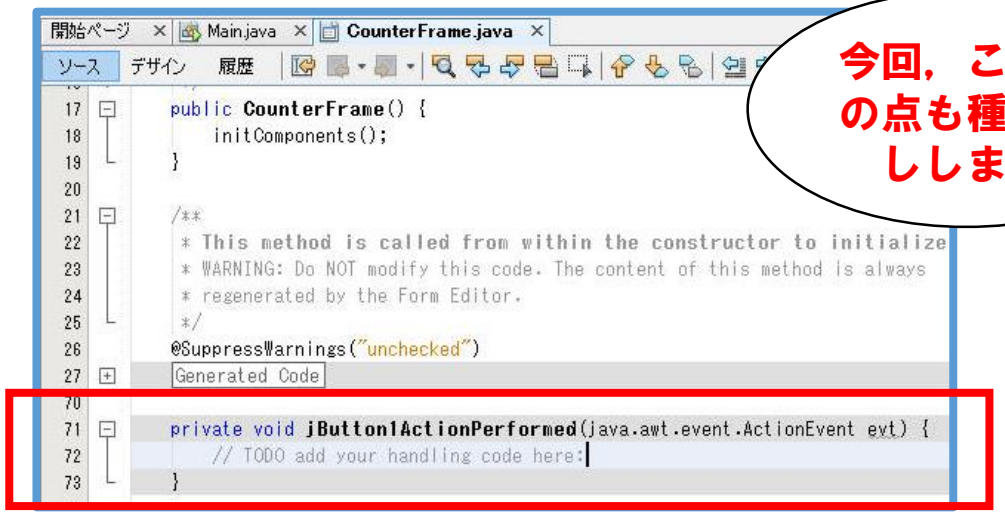




エディタツールバー



イベント
ハンドラ
の生成



iijima@ae.keio.ac.jp