ソフトウェア工学実習
Software Engineering Practice
(第03回)

SEP03-002 イベント処理

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

前回の例題P0104
から始めましょう

Javaでのクラス定義の書き方（入門編）


```
public class クラス名 {
    変数(属性)宣言;
        …
    メソッド宣言;
        …
}
```

属性
- インスタンス変数
- クラス変数(static変数)

操作
- インスタンス・メソッド
- クラス・メソッド
  (staticメソッド)

Javaでの
クラスの
書き方です

いろいろな用語:

構成要素	一般表現	Smalltalk-80	Java	C++
内部状態	属性	変数	フィールド変数	データメンバ
振る舞い	操作	メソッド	メソッド	メンバ関数

Javaでのインスタンス生成

- クラス定義中のコンストラクタ定義
  - クラス名と同じ名前の特別なメソッド
  - 返却値の型などは指定しない
  - 引数を渡せる，
  - 引数の数やその型が異なれば複数定義できる（オーバーローディング）

コンストラクタの呼び出し
`new クラス名()`
引数が無くても()を付ける

例:
```
new Stack();
new Stack(100);
```

new演算子を付けてコンストラクタを呼ぶとインスタンスを生成して初期化します

配列

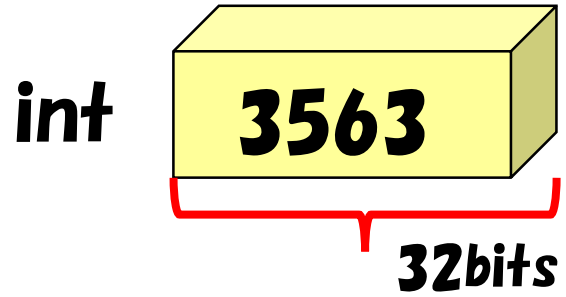
基本データ型配列
```
int a[] = new int[20];
```
クラス参照型配列
```
クラス b[] = new クラス[100];
```

正確に言えばクラス名と同じ名前を持っている特殊なメソッドがコンストラクタである.

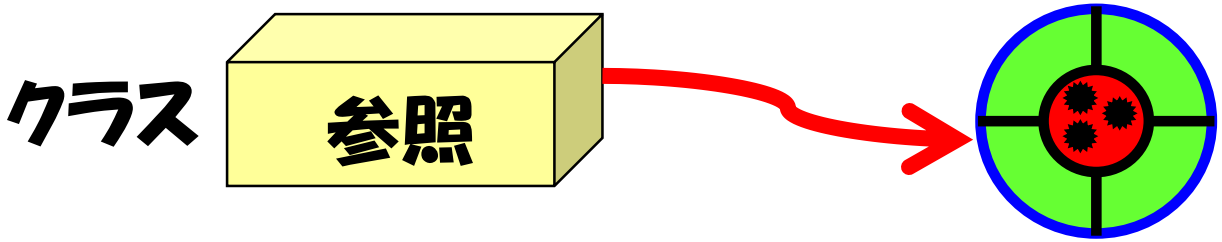
変数には２通りある

- 基本データ型とクラス参照（クラスリファレンス）型

- 基本データ型(primitive data type)変数
  - int, long, float, double, …



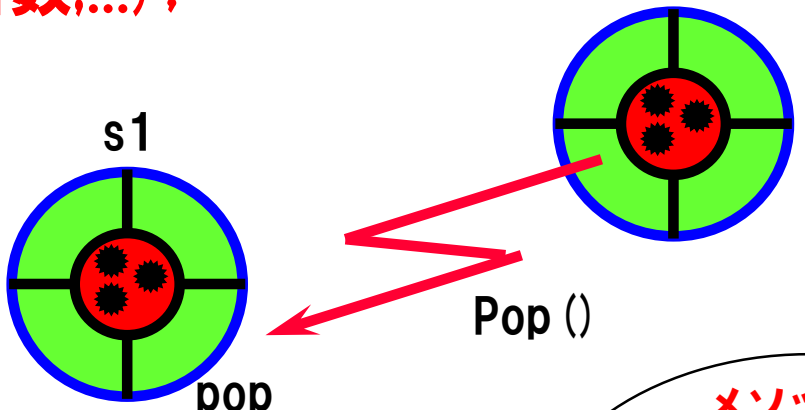
- クラス参照型(class reference type)変数
  - クラス名と同じ名前の型名



Javaでの（インスタンス）メソッド呼出し

- メソッド呼出し

  インスタンス名．メソッド名（実引数,...）;



例えば，s1 . pop ();

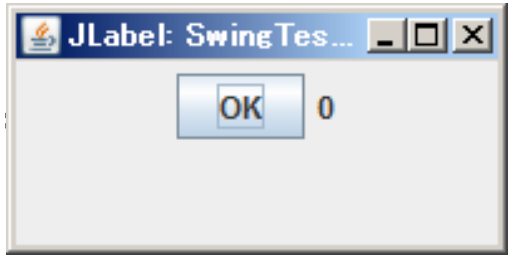
インスタンス名
＝クラス参照型変数の名前
（クラス参照型とは，
クラス名と同じ名前の変数）

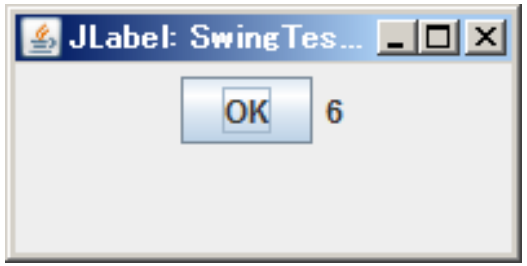
ドット表記法
（C言語の構造体の
メンバアクセスと同じ）

メソッド呼び出し
＝関数呼び出し

メソッドを
呼び出すためには
ドット記法で，
オブジェクトへの
参照に向けて
メッセージを
送ります

ボタンのアクション:P0104パッケージ(クラス)



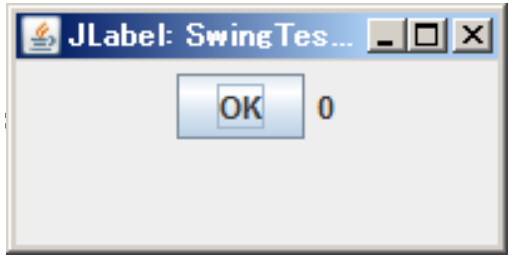


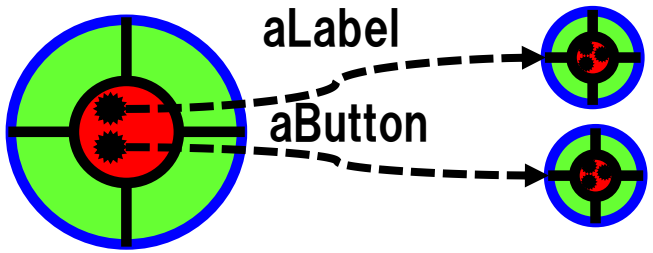

```
package p0104;
// ========================================
import javax.swing.*;
import java.awt.*;
// ========================================
public class ButtonFrameA extends JFrame {
    private JLabel  aLabel; //ラベル
    private LikeButton aButton; //ボタン
    public ButtonFrameA() { //コンストラクタ(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
        … }
}
// ============================================
```

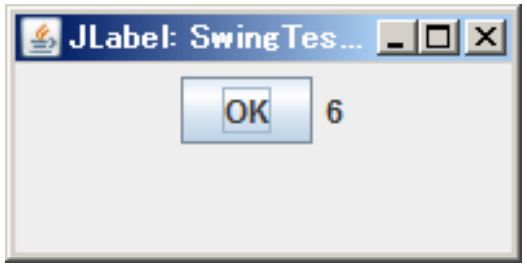
ボタンを
クリックすると
カウントアップする
ようにしましょう

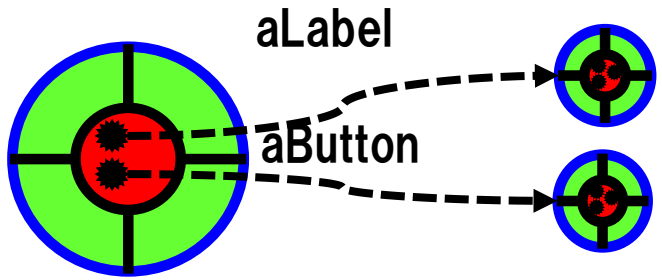
ボタンのアクション:P0104パッケージ(クラス)


```
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
}
// ============================================
```







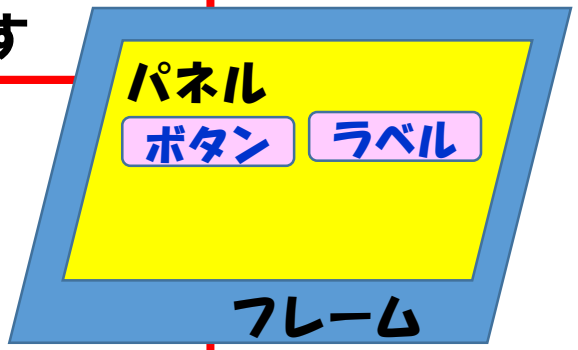


aLabelとaButtonは**属性**だが,
その値は，他のオブジェクトへの**参照**である.

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(コンストラクタ)

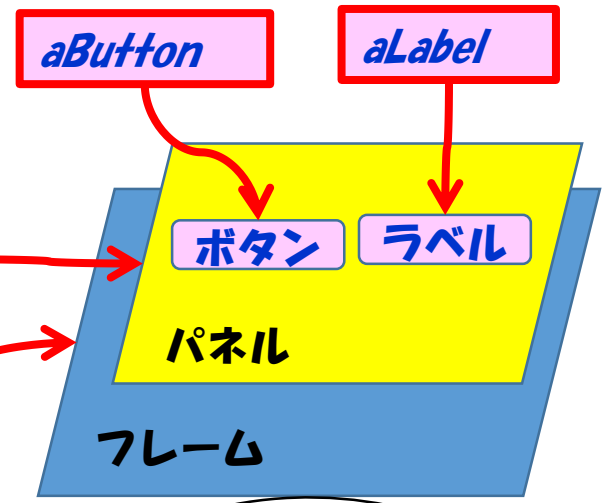

```
public ButtonFrameA() { //コンストラクタ(インスタンスの初期設定)
    //
    aLabel = new JLabel( “0” );       // ラベルを作ります
    aButton = new LikeButton( aLabel ); // ボタンを作ります
    // パネルをつくって，ボタンとラベルを配置します
    JPanel aPanel = new JPanel();
    aPanel.add( aButton );
    aPanel.add( aLabel );
    add( aPanel ); // パネルをウィンドウに配置します
    //終了方法の設定
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    // GUI部品に適したウィンドウサイズに調整
    pack();
    // 位置と大きさの設定
    setBounds( 100, 200, 400, 300 );
}
```



コンストラクタで
オブジェクトの
初期設定を
します

ボタンのアクション:P0104パッケージ(コンストラクタ)


```
public ButtonFrameA () { //コンストラクタ(インスタンスの初期設定)
  //
    this.aLabel = new JLabel( “0” );      // ラベルを作ります
    this.aButton = new LikeButton( aLabel ); // ボタンを作ります
    // パネルをつくって，ボタンとラベルを配置します
    JPanel aPanel = new JPanel();
    aPanel.add( aButton );
    aPanel.add( aLabel );
    this.add( aPanel ); // パネルをウィンドウに配置します
    //終了方法の設定
    this.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    // GUI部品に適したウィンドウサイズに調整
    this.pack();
    // 位置と大きさの設定
    this.setBounds( 100, 200, 400, 300 );
}
```

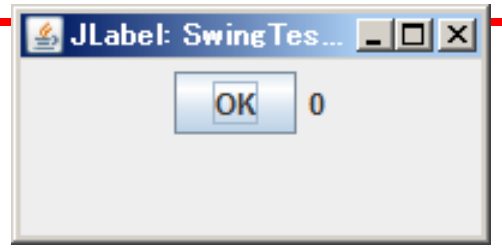


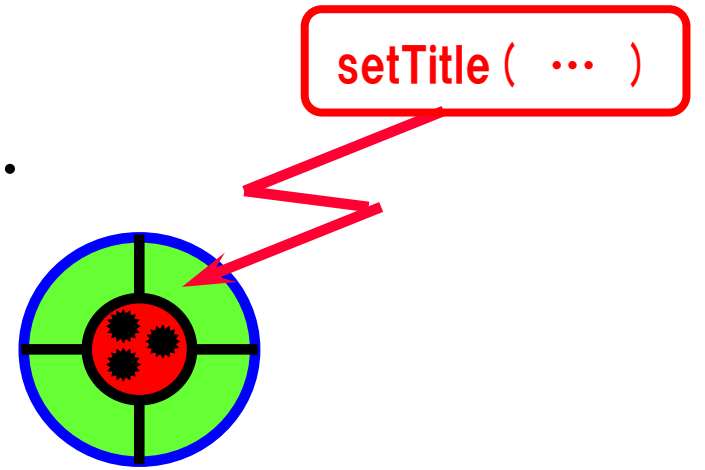
自分自身を
指し示す参照
である
Thisは
省略できます

そのオブジェクト(フレーム)自身を示すthis.は省略可

ボタンのアクション:P0104パッケージ(メイン・メソッド)


```
public static void main ( String [] args ) {
    // GUI部品の取り扱いはmainスレッドではなく
    // awtスレッドで行わねばなりません．そのため，
    // 厳密には下記のプログラムは正しくありません．
    // ですが，スレッドや無名クラスについて
    // 勉強する前に簡単に理解できるように，これを
    // 示しています．通常，問題なく動きます．
    //
    //（1）ボタン付きのウィンドウを一つ作って表示します
    ButtonFrameA frame1 = new ButtonFrameA ();
    frame1.setTitle ( "ソフトウェア工学実習: ButtonFrameA" );
    frame1.setVisible ( true );
}
```





Frame1
オブジェクトへ
setTitleメッセージ
を送っています

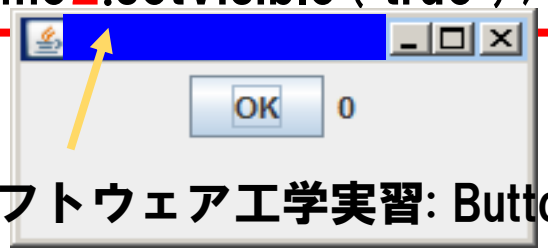
Frame1.setTitle (…)

Frame1オブジェクトへ，
setTitleメッセージを送る

課題：ウィンドウを二つ生成して，タイトルバーの文字列を替えましょう

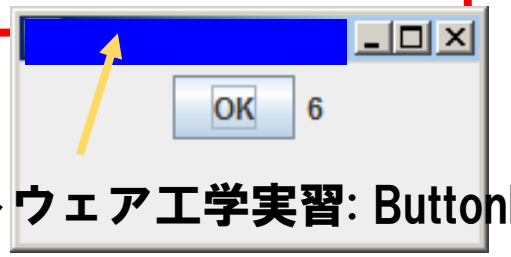

```
public static void main ( String [] args ) {

  // (1) ボタン付きのウィンドウを一つ作って表示します
   ButtonFrameA frame1 = new ButtonFrameA ();
   frame1.setTitle ( "ソフトウェア工学実習: ButtonFrameA-1" );
   frame1.setVisible ( true );

  // (2) ボタン付きのウィンドウをもう一つ作って表示します
   ButtonFrameA frame2 = new ButtonFrameA ();
   frame2.setTitle ( "ソフトウェア工学実習: ButtonFrameA-2" );
   frame2.setVisible ( true );
}
```

２つめの
ウィンドウを
生成し，
それぞれ独立に
カウントしましょう



frame1

ソフトウェア工学実習: ButtonFrameA-1

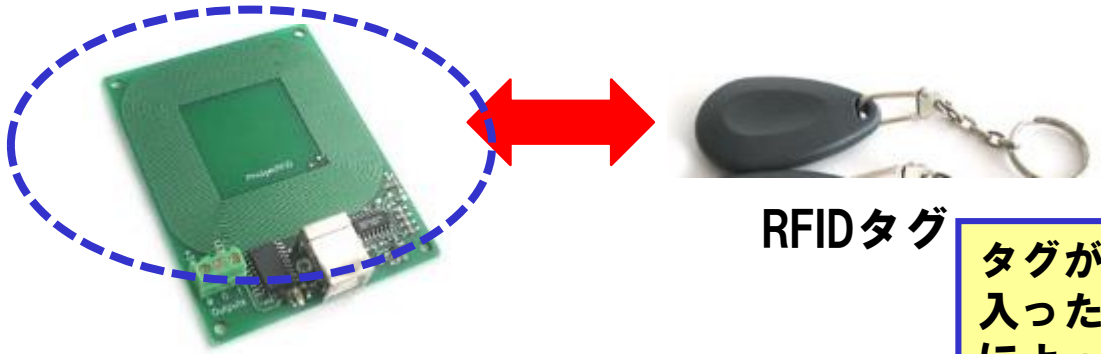


frame2

ソフトウェア工学実習: ButtonFrameA-2

イベント駆動プログラム

- RFID (ICタグのセンシング・イベントのデモ)



RFIDタグ

タグがアンテナの有効範囲に
入ったり，出たりするイベント
によって計算が起動される

カウンター

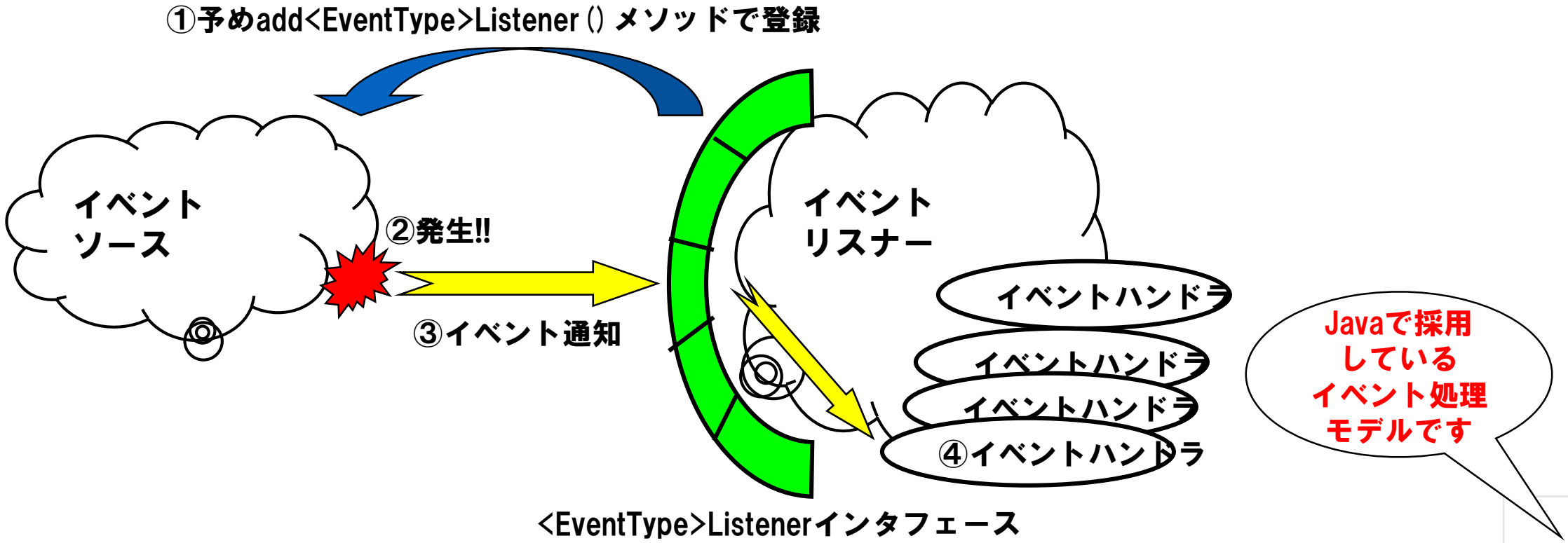
GUI (Graphical User I/F) の場合．
ボタンの上で
マウスがクリックされると
アクションイベントが発生し，
カウントアップする．

RFIDのデモ

iijima@ae.keio.ac.jp

【参考】 Delegation-based Listenerモデル（委譲に基づくリスナ・モデル）

- イベントソース：イベントの発生源のオブジェクト
- イベントリスナ：イベントハンドラのメソッドを持ち，イベント発生通知を受けてイベント処理を実行するオブジェクト



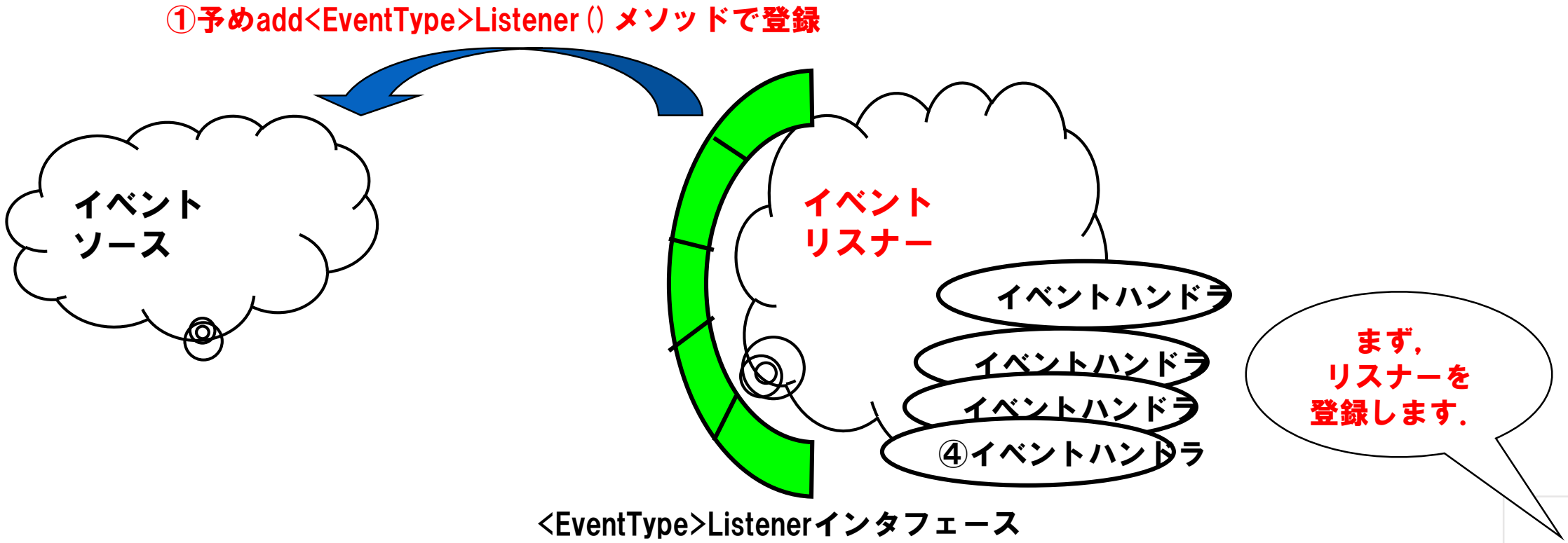
Javaで採用
している
イベント処理
モデルです

イベントをメッセージで表現する

iijima@ae.keio.ac.jp

【参考】 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

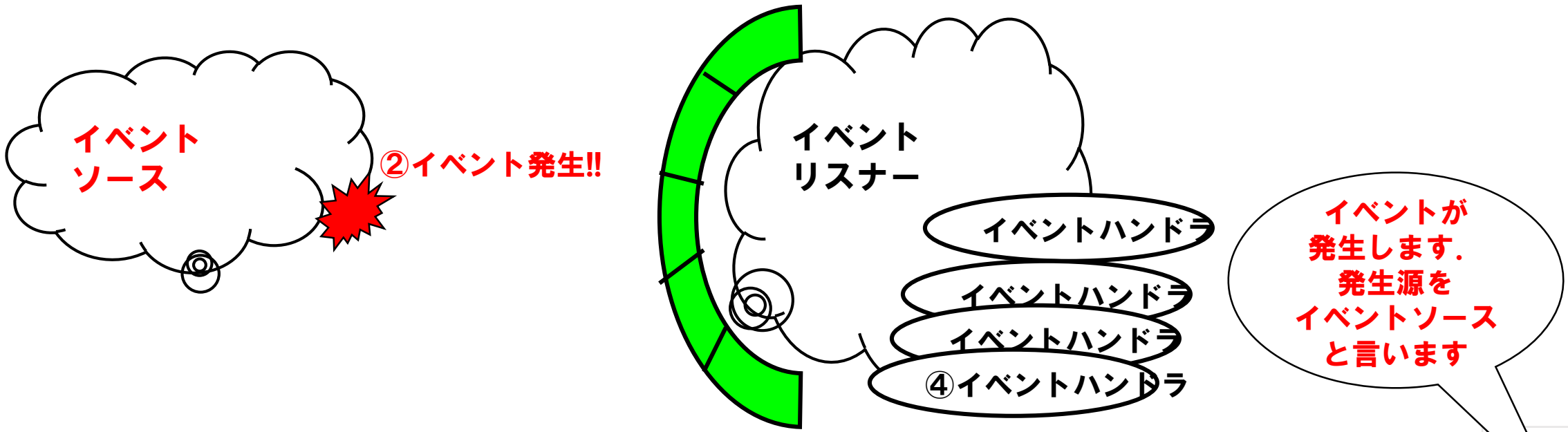


イベントをメッセージで表現する

iijima@ae.keio.ac.jp

【参考】 Delegation-based Listenerモデル（委譲に基づくリスナ・モデル）

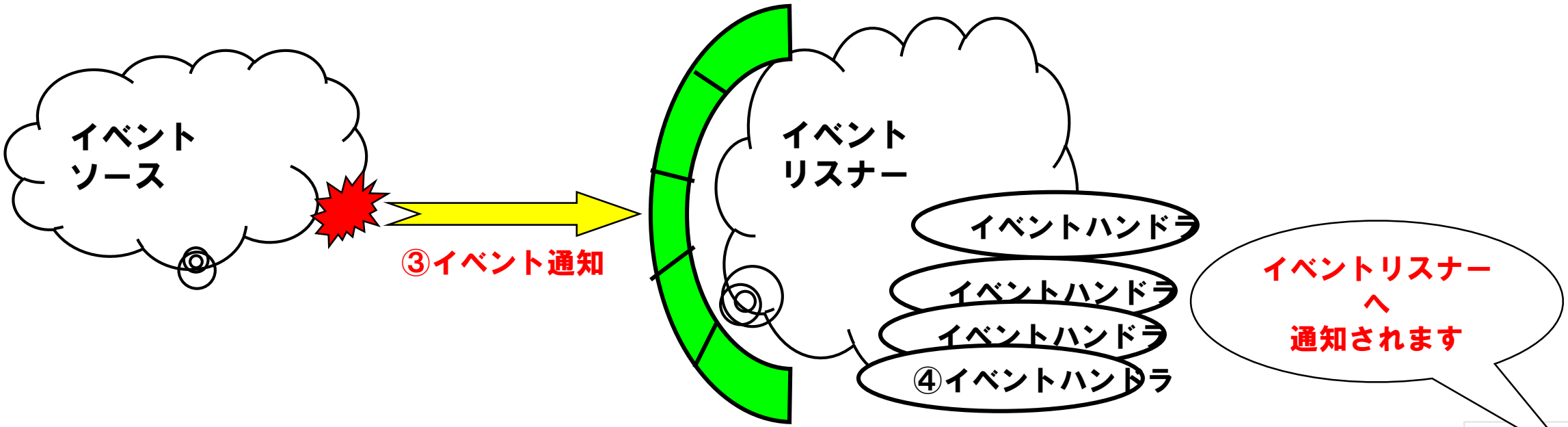
- ②そのコンポーネント（イベントソース）で，イベントが発生する（例えばボタン上でマウスボタンが押される）



イベントをメッセージで表現する

【参考】 Delegation-based Listenerモデル（委譲に基づくリスナ・モデル）

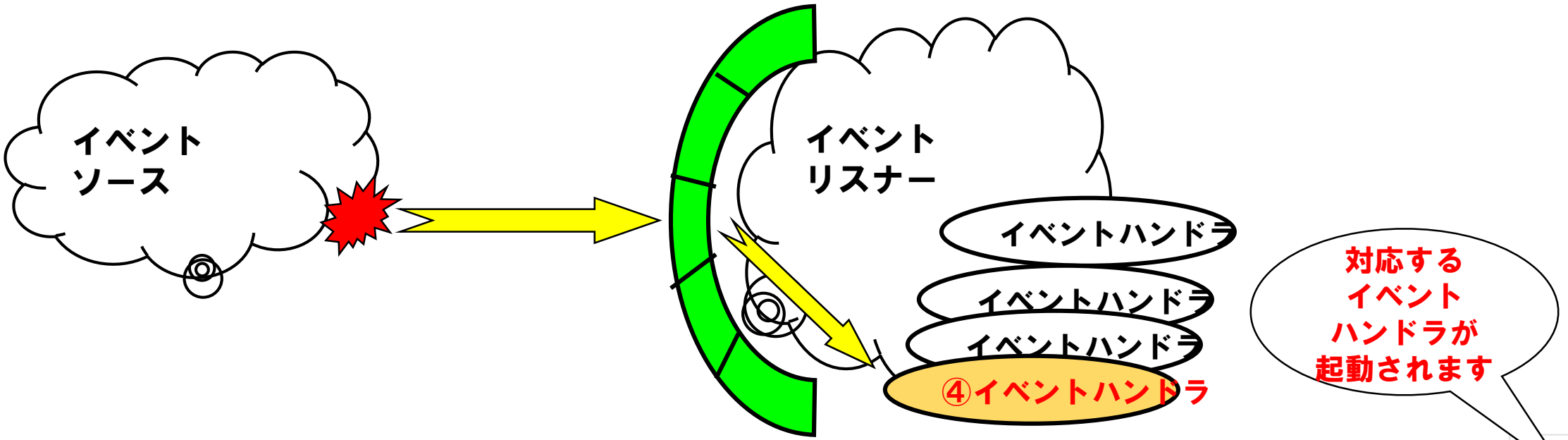
- ③イベントソースに登録されている全イベントリスナに，そのイベントが通知される



イベントをメッセージで表現する

【参考】 Delegation-based Listenerモデル（委議に基づくリスナ・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

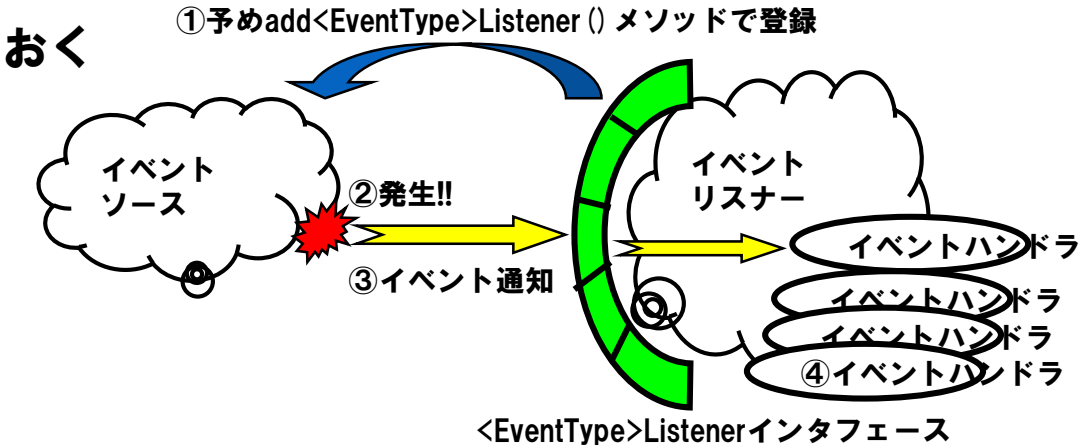


<EventType>Listenerインタフェース

イベントをメッセージで表現する

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

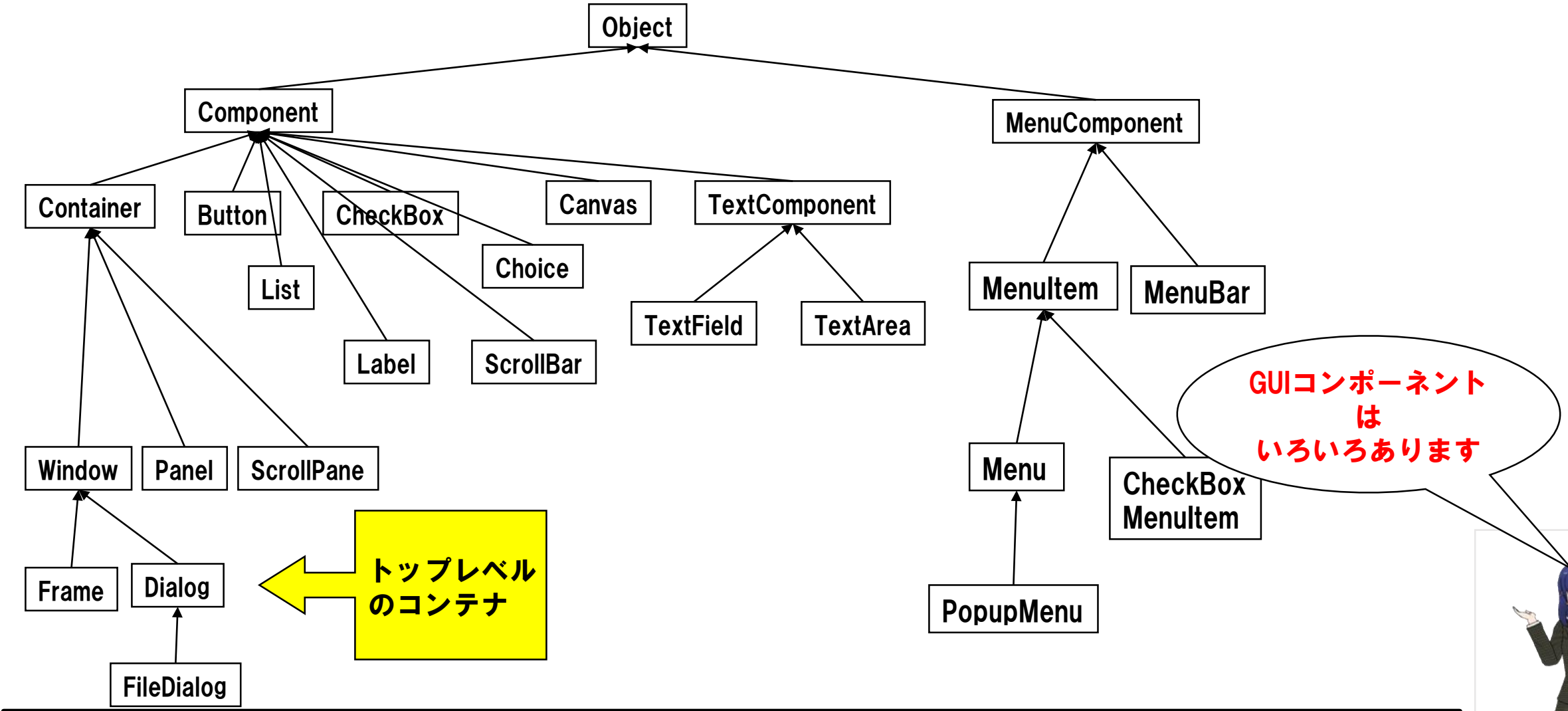


イベントをメッセージで表現する

iijima@ae.keio.ac.jp

GUIコンポーネントの階層

- コンポーネント毎に，発生するイベントが決まっている



iijima@ae.keio.ac.jp

【参考】 GUIコンポーネント毎のイベントとハンドラ(1/3)

Component	ComponentEvent	componentMoved ()
		componentResizes ()
		componentShown ()
		componentHidden ()
	FocusEvent	focusGained ()
		focusLost ()
	KeyEvent	keyPressed ()
		keyReleased ()
		keyTyped ()
	MouseEvent	mouseClicked ()
		mouseEntered ()
		mouseExited ()
		mousePressed ()
		mouseReleased ()
		mouseDragged ()
		mouseMoved ()

iijima@ae.keio.ac.jp

【参考】GUIコンポーネント毎のイベントとハンドラ(2/3)

Button	ActionEvent	actionPerformed ()
MenuItem		actionPerformed ()
List		actionPerformed ()
Choice	ItemEvent	itemStateChanged ()
CheckBox		itemStateChanged ()
CheckBox MenuItem		itemStateChanged ()
Text Component	TextEvent	textValueChanged ()
TextField	ActionEvent	actionPerformed ()

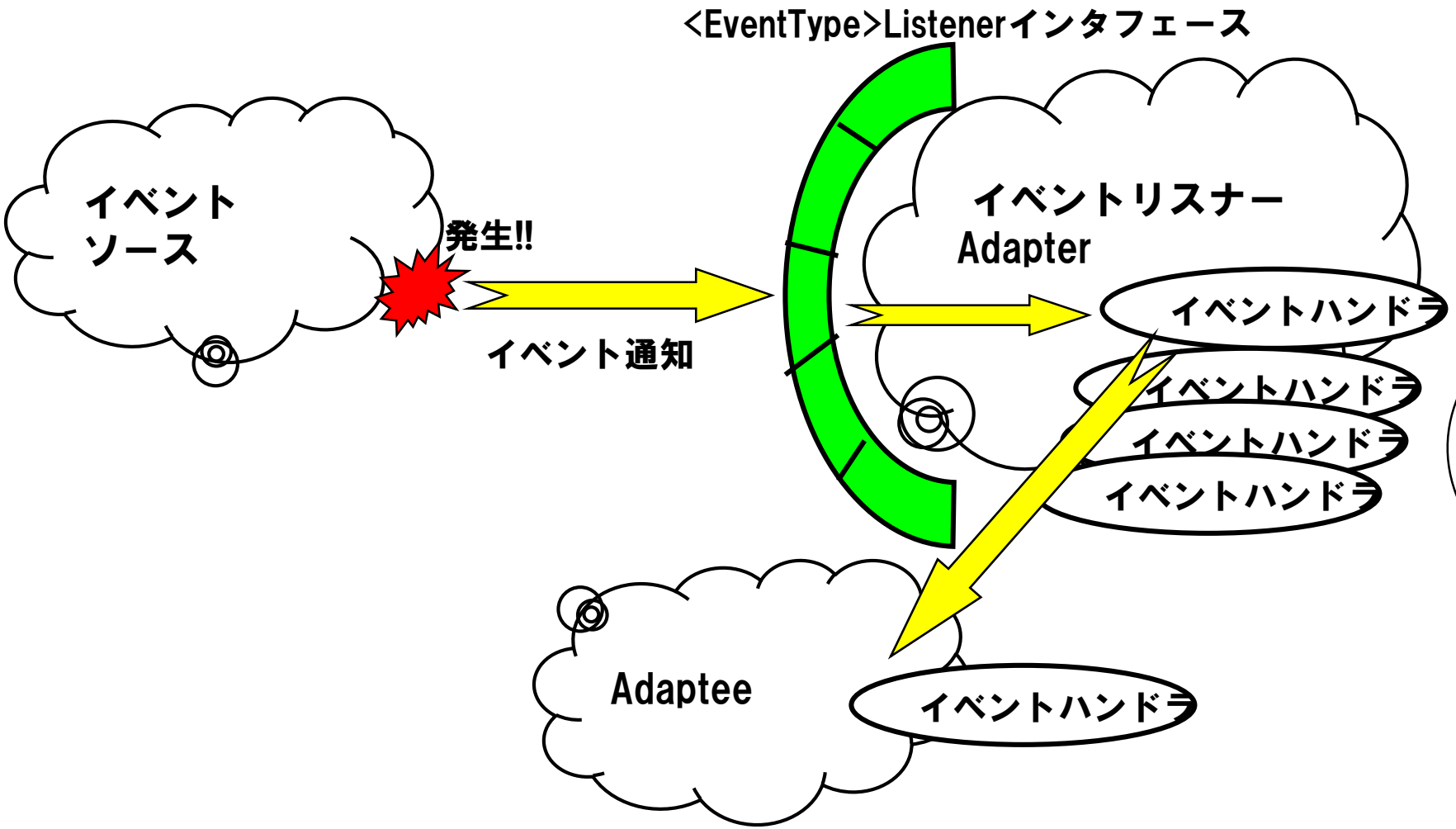
iijima@ae.keio.ac.jp

【参考】 GUIコンポーネント毎のイベントとハンドラ(3/3)

Window	WindowEvent	windowClosed ()
		windowClosing ()
		windowOpened ()
		windowIconified ()
		windowDeiconified ()
ScrollBar	Adjustment Event	adjustmentValueChanged ()

iijima@ae.keio.ac.jp

【参考】 Adapterの利用

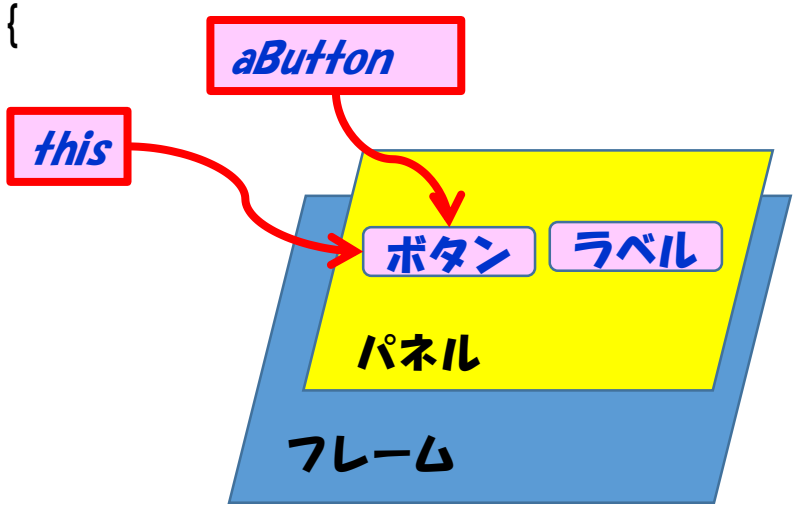


ボタンのアクション:P0104パッケージ (LikeButtonクラス)


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

ボタンのアクション:P0104パッケージ (LikeButtonクラス)

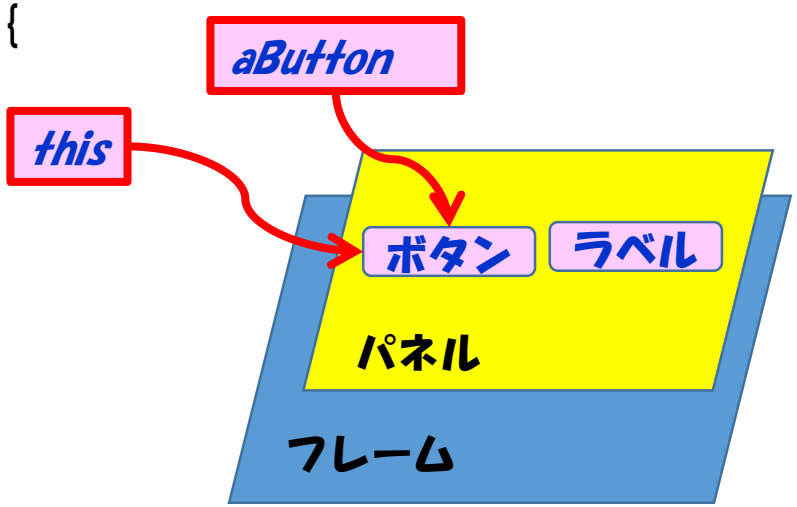
```
class LikeButton extends JButton implements ActionListener {
    private int count = 0; //カウント数を格納する
    private JLabel aLabel  //カウントを表示するラベル

        …


    /**
     * コンストラクタ (インスタンス生成時の初期設定)
     */
       public LikeButton ( JLabel aLabel ) {
              super ( "いいね" ) ;
              addActionListener (this) ;
              this.aLabel = aLabel;
       }
```



this.が付いていると,
属性（フィールド変数）
であることが
わかります.
thisがついていないと,
変数の有効範囲 (Scope)
で近いものになります

ボタンのアクション:P0104パッケージ(LikeButtonクラス)


```
class LikeButton extends JButton implements ActionListener {
    private int count = 0; //カウント数を格納する
    private JLabel aLabel; //カウントを表示するラベル

        …


    /**
     * コンストラクタ(インスタンス生成時の初期設定)
     */
       public LikeButton ( JLabel aLabel ) {
                  super ( "いいね" );
                  addActionListener ( this );
                  this.aLabel = aLabel;
       }
```

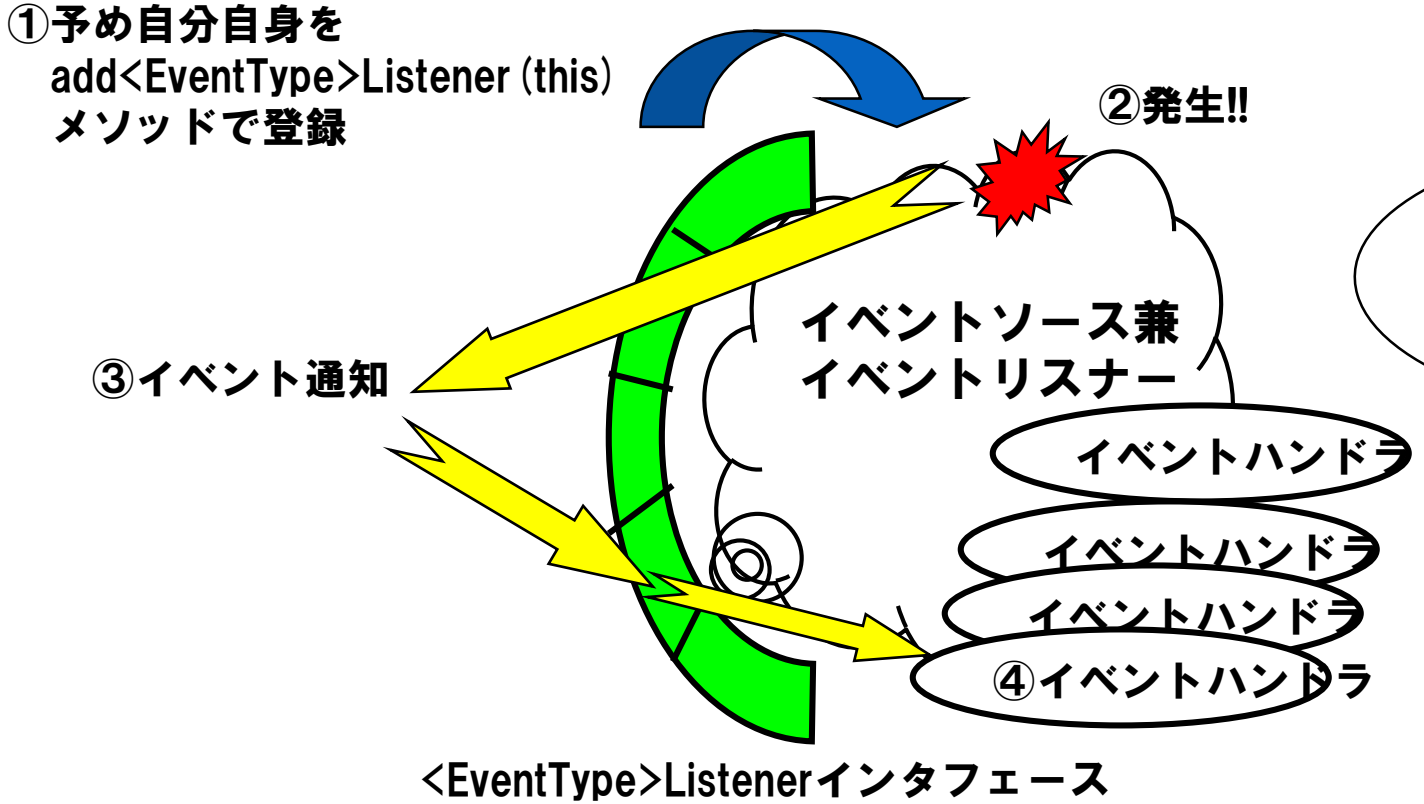


自分自身(this)が
イベントリスナーであり
イベントハンドラを
持っていることを意味する

自分自身を
自分のイベントリスナー
として登録しています.

【参考】 Delegation-based Listenerモデル（委議に基づくリスナ・モデル）

- イベントソース自体がイベントリスナーとして登録されるケース



ボタンのアクション:P0104パッケージ(LikeButtonクラス)


```
class LikeButton extends JButton implements ActionListener {
    private int count = 0; //カウント数を格納する
    private JLabel aLabel; //カウントを表示するラベル
        ...

    // イベント・ハンドラ(ActionEvent)
    public void actionPerformed( ActionEvent e ) {
            // カウントアップし，その値をラベルに表示する
        count++;
        aLabel.setText( Integer.toString( count ) );
    }
```

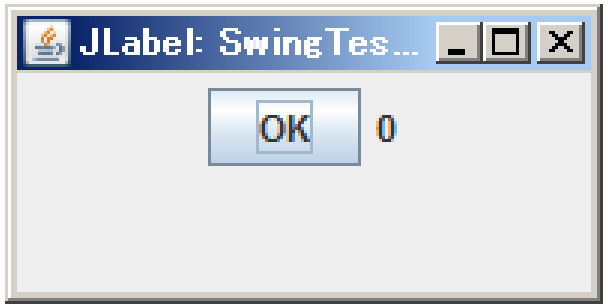
ボタンが押された時のアクション
＝カウントアップして，結果を文字列に変換して表示

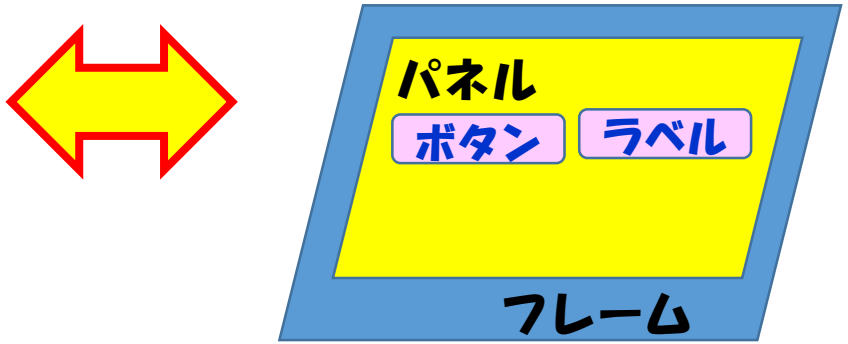
ボタンクリックでは，
Actionイベントが発生し，
ActionPerformedというハンドラが起動される

iijima@ae.keio.ac.jp

ボタンのアクション: P0104パッケージ

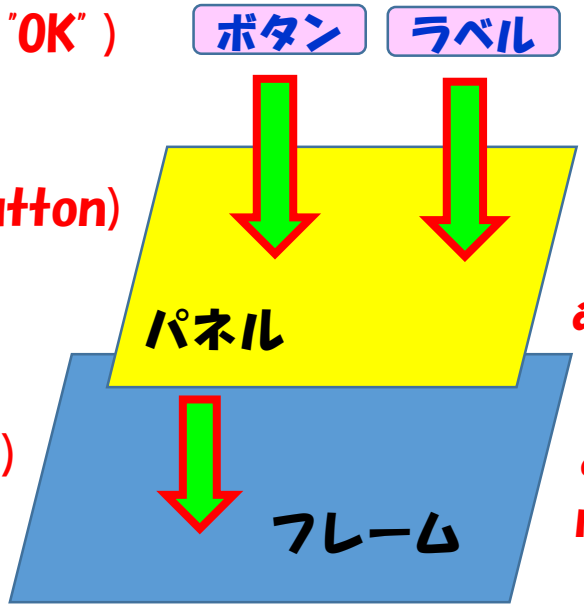
- ボタンやラベルを直接フレームに追加せず，パネルを介しています．
  これはなぜでしょう？　→　**試してみましょう!!**





aButton = new JButton("OK")
オブジェクト生成

aLabel =new JLabel("0")
オブジェクト生成



aPanel.add(aButton)

aPanel.add(aLabel)

aPanel =new JPanel()
オブジェクト生成

add(aPanel)

aFrame =
new P104ButtonFrame()
オブジェクト生成

ボタンやラベルは，
直接フレームに
追加せず，
パネルを挟んでいます．
なぜでしょう？

NetBeansにおけるGUI部品のイベント処理

NetBeansにおけるGUI部品のイベント処理

- デザインビューで，Drag&DropされたGUI部品（Swingコントロール）が，ソースビューに反映される
  - 以下の三つを自動生成し，整合性を保つとともに，詳細を隠ぺいしてくれる
  - （1）GUI部品の参照変数
    - ソースビューのソースコードの最後の部分に追加される
      - 例えば，JLabel countLabel，JButton countButton
    - 二つのビューの不整合が起こらないように，ソースビューでは編集禁止
  - （2）コンストラクタから呼び出されるinitComponents（）メソッド
    - 通常は，fold状態（畳み込まれている）
    - デザインビューに対応するように，部品を配置する
      - インスタンスの生成，プロパティで設定されたフォント設定など
      - フレームのレイアウト，GUI部品の配置
      - GUIの参照変数名を含む名前で，イベントハンドラを呼び出せるように
  - （3）イベントハンドラの管理
    - GUI部品の名前を含むメソッド名でイベントハンドラを呼び出すことができる
      - 例えば，countButtonActionPerformedメソッド

デザインビューから自動生成されるコードに隠されている