



ソフトウェア工学実習

Software Engineering Practice

(第04回)

SEP04-003 アクセス修飾子とカプセル化

こんにちは。
この授業は、
ソフトウェア
工学実習
です

慶應義塾大学・理工学部・管理工学科
飯島 正

iiijima@ae.keio.ac.jp



今回の話題（第04回）

アクセス修飾子とカプセル化

今回は、
アクセス指定
と
カプセル化
について
説明します



話の流れ (第04回)

SEP04

3

第01回にzipで配布したP0104パッケージを確認する



アクセス指定とカプセル化



Javaのアクセス修飾子



NetBeansで作ったCounterFrameを確認する



NetBeansでのGUIコンポーネントのプロパティ指定



実習編: 「メソッドで属性を保護する」意義を確認

話の流れは
...



話の流れ (第04回)

SEP04

4

第01回にzipで配布したP0104パッケージを確認する

アクセス指定とカプセル化

Javaのアクセス修飾子

NetBeansで作ったCounterFrameを確認する

NetBeansでのGUIコンポーネントのプロパティ指定

実習編: 「メソッドで属性を保護する」意義を確認

話の流れは
...



ボタンのアクション:P0104パッケージ (LikeButtonクラス)

SEP04

5

↑ 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
```

```
... }
```

```
// =====
```

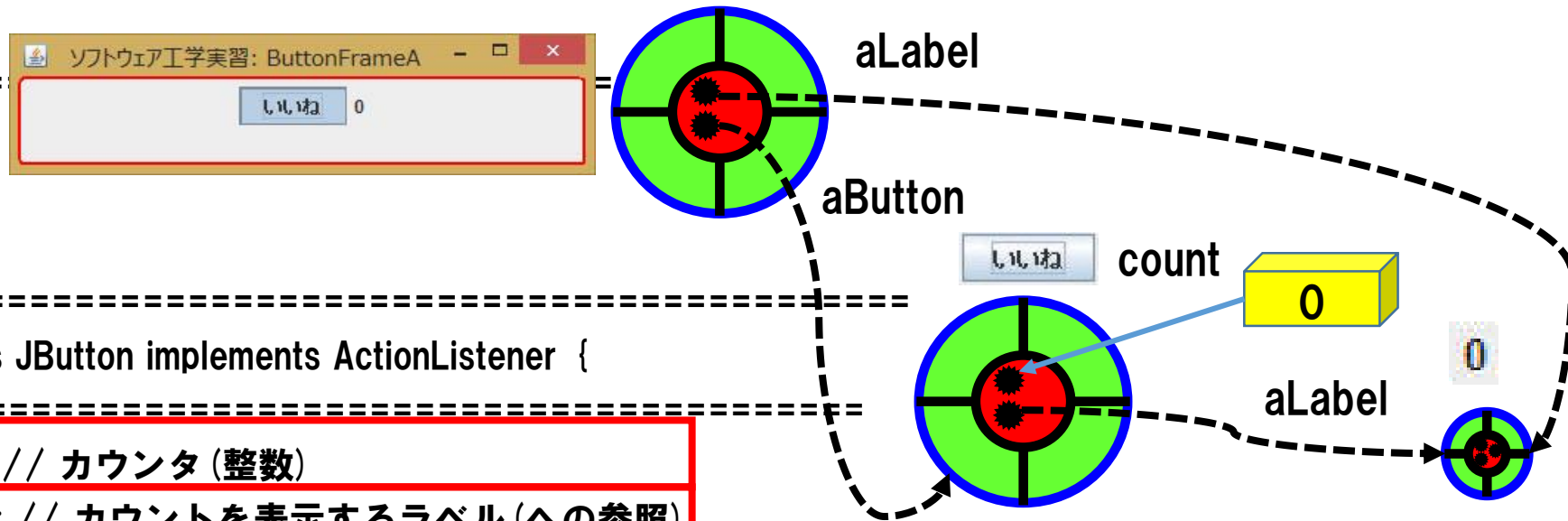
```
public void actionPerformed ( ActionEvent e ) { // イベント・ハンドラ
```

```
... }
```

```
// =====
```

```
}
```

```
// =====
```



属性(フィールド変数)はprivateに、
メソッドはpublicに
設定するのが、
典型的です



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

SEP04

6

```
package p0104;
// =====
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
属性(フィールド変数)はprivate =====
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
// =====
        public void actionPerformed ( ActionEvent e ) { // イベント・ハンドラ
            ...
        }
// =====
}
// =====
```



属性(フィールド変数)はprivateに、
メソッドはpublicに
設定するのが、
典型的です



話の流れ (第04回)

SEP04

7

第01回にzipで配布したP0104パッケージを確認する



アクセス指定とカプセル化



Javaのアクセス修飾子



NetBeansで作ったCounterFrameを確認する



NetBeansでのGUIコンポーネントのプロパティ指定



実習編: 「メソッドで属性を保護する」意義を確認

話の流れは
...



アクセス指定とカプセル化

カプセル化の原則

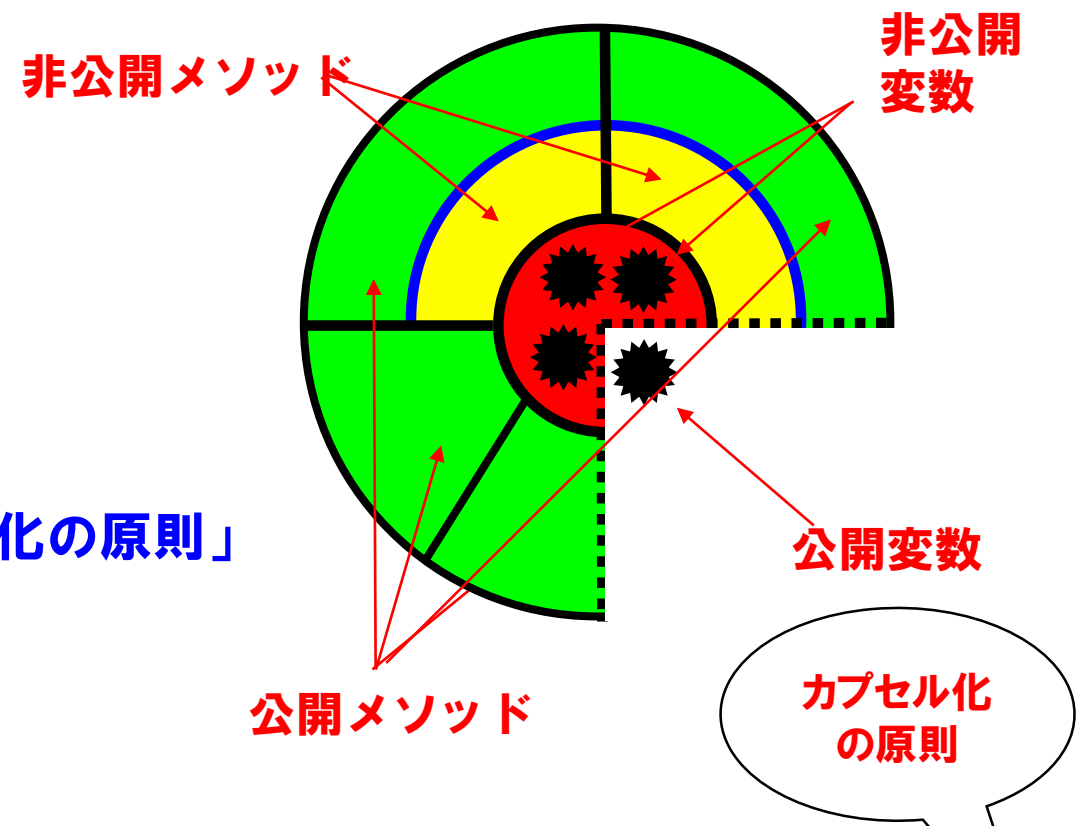
1) 変数は極力、公開せず、
メソッドを介してアクセスするようにする

2) メソッドも必要最小限のもののみ公開する
(外部から直接呼ばれない下請けルーチンのようなメソッドはできるだけ非公開とする)

簡単に言い切ってしまうなら、
属性(変数)を、やたらと公開しないのが「カプセル化の原則」

		変数	メソッド
公開	public	×	○
非公開	private	○	○

※他にも以下のようなアクセス指定がある		
プロテクテッド	protected	※サブクラス化、継承といったメカニズムを学んでからやります
デフォルト	無指定	パッケージ・プライベート ※後で説明します



話の流れ (第04回)

SEP04

9

第01回にzipで配布したP0104パッケージを確認する

アクセス指定とカプセル化

Javaのアクセス修飾子

NetBeansで作ったCounterFrameを確認する

NetBeansでのGUIコンポーネントのプロパティ指定

実習編: 「メソッドで属性を保護する」意義を確認

話の流れは
...



Javaのアクセス修飾子

• アクセス修飾子

- クラス名や個々の変数, メソッド宣言の前に指定して, スコープ（有効範囲）を制御する
- public,private,protected, 無指定にも意味がある.
- カプセル化の発想から, 変数は極力, 非公開にして, 公開メソッドを介した, 管理されたアクセスのみを受け付けるように指定するのが一般的である.

アクセスレベル	アクセス修飾子	
公開	public	どのクラス(のオブジェクト)からでもアクセス可能
非公開	private	同一クラス(のオブジェクト)からだけアクセス可能 ※同じクラスなら, 別のオブジェクト同士でも相互にアクセス可能
プロテクトド	protected	同一クラスとそのサブクラスだけからアクセス可能 ※サブクラスからなら, 他のパッケージ内からでもアクセス可能 ※但し, 同一パッケージ内からなら, どのクラスからでもアクセス可能
デフォルト	無指定	同一パッケージ内のクラス(のオブジェクト)からならアクセス可能

アクセス
修飾子



アクセス修飾子とパッケージとの関係

アクセス修飾子	同一パッケージ内			他のパッケージ		アクセス可能なオブジェクト
	同一クラス	サブクラス	他のクラス	サブクラス	他のクラス	
public	○	○	○	○	○	どこからでもアクセス可能
private	○	×	×	×	×	同一クラスからのみアクセス可能
protected	○	○	○	○	×	同一クラスとサブクラス, 同一パッケージ
無指定	○	○	○	×	×	同じパッケージ内ならアクセス可能



クラスに指定できるアクセス修飾子

	クラス	メソッド	フィールド変数
public	○	○	○
private	×	○	○
protected	×	○	○
無指定	○	○	○

クラスに指定
できるアクセス
修飾子は限定
されている



話の流れ (第04回の復習)

SEP04

13

第01回にzipで配布したP0104パッケージを確認する



アクセス指定とカプセル化



Javaのアクセス修飾子



NetBeansで作ったCounterFrameを確認する



NetBeansでのGUIコンポーネントのプロパティ指定

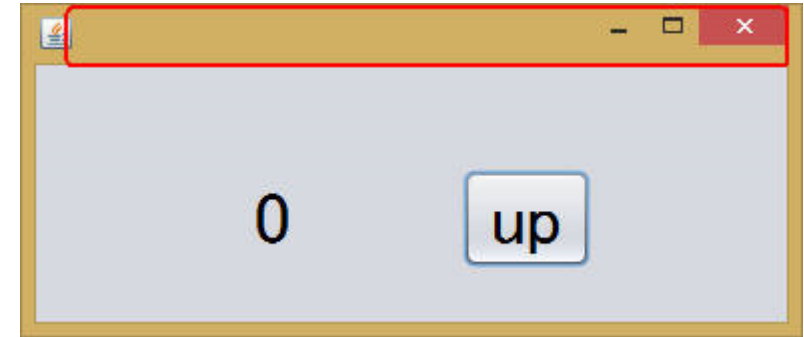
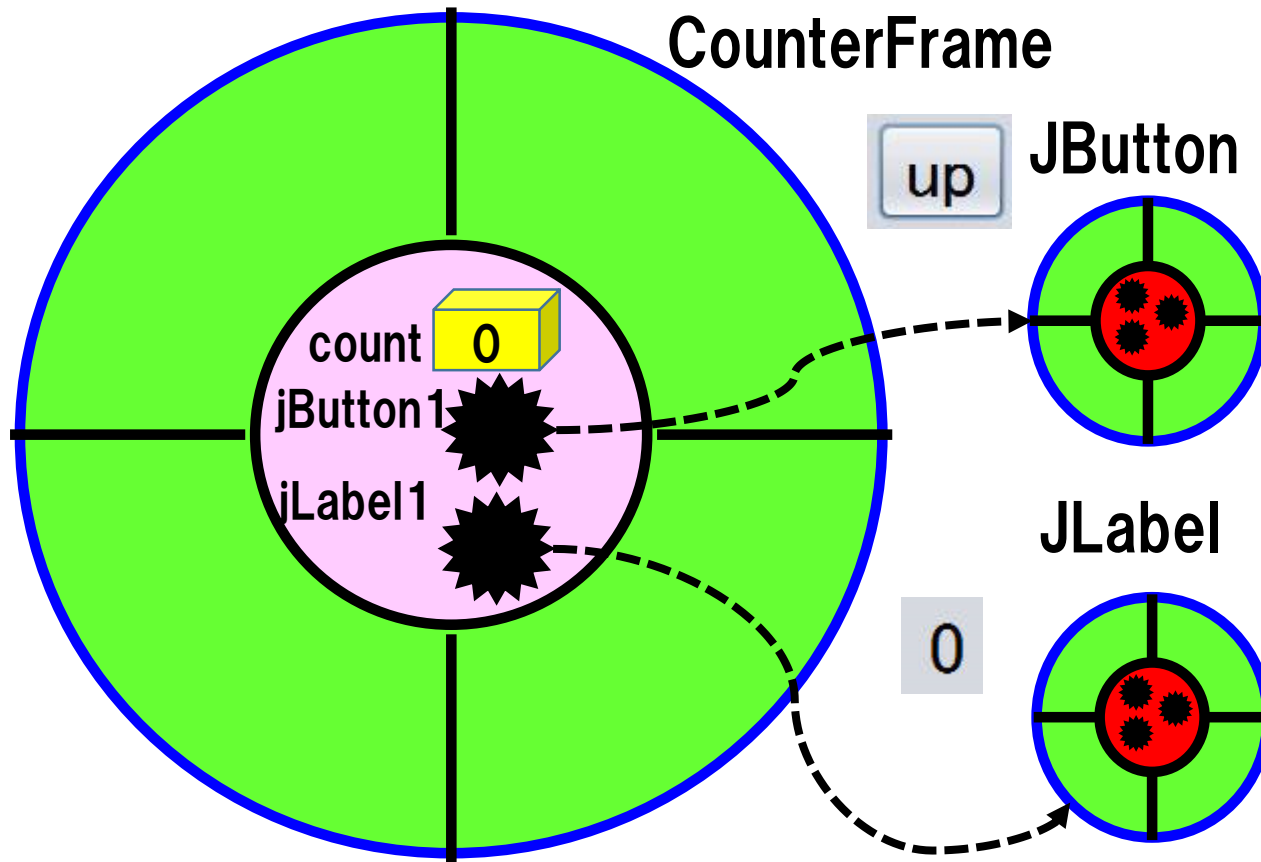


実習編: 「メソッドで属性を保護する」意義を確認

話の流れは
...



前回, NetBeansで作ったCounterFrame



```
12 public class CounterFrame extends javax.swing.JFrame {  
13     private int count;
```

```
110 // Variables declaration - do not modify  
111 private javax.swing.JButton jButton1;  
112 private javax.swing.JLabel jLabel1;  
113 // End of variables declaration  
114  
115  
116 }
```

NetBeans
でも...



前回, NetBeansで作ったCounterFrame

- NetBeansが自動生成したコード (initComponents) が畳み込まれている

```
15  /**
16   * Creates new form CounterFrame
17   */
18  public CounterFrame() {
19      initComponents();
20  }
21
22  /**
23   * This method is called from within the constructor to initialize
24   * WARNING: Do NOT modify this code. The content of this method is always
25   * regenerated by the Form Editor.
26   */
27  @SuppressWarnings("unchecked")
28  Generated Code
71
72  private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
73      count++;
74      jLabel1.setText( Integer.toString( count ) );
75  }
76
```

コンストラクタから呼び出されている
initComponentsというメソッドは,
NetBeansがデザインビューの情報から自動生成している



どこで定義されているのか？

コンストラクタの中の
initComponent



前回, NetBeansで作ったCounterFrame

- NetBeansが自動生成したコード (initComponents) が畳み込まれている


15 `/**`
16 `* Creates new form CounterFrame`
17 `*/`
18 `public CounterFrame() {`
19 `initComponents();`
20 `}`
21
22 `/**`
23 `* This method is called when the form is first loaded by the IDE.`
24 `* To initialize the form and its components, the method initComponents`
25 `* is called. This method is always`
26 `* generated by the IDE.`
27 `*/`
28 `@SuppressWarnings("unchecked")`
29 `Generated Code`
71 `// <editor-fold defaultstate="collapsed" desc="Generated Code">`
72 `private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {`
73 `count++;`
74 `jLabel1.setText(Integer.toString(count));`
75 `}`
76

コンストラクタから呼び出されている
initComponentsというメソッドは,
NetBeansがデザインビューの情報から自動生成している

[+] をクリックすると畳み込まれている行
を展開する.

どこで定義されているのか?

畳み込まれ
ている行を
展開します



前回, NetBeansで作ったCounterFrame

- NetBeansが自動生成したコード (initComponents) が畳み込まれている

```
28 // <editor-fold defaultstate="collapsed" desc="Generated Code">
29 private void initComponents() {
30
31     jLabel1 = new javax.swing.JLabel();
32     jButton1 = new javax.swing.JButton();
33
34     setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
35
36     jLabel1.setFont(new java.awt.Font("MS UI Gothic", 0, 36)); // NOI18N
37     jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
38     jLabel1.setText("0");
39
40     jButton1.setFont(new java.awt.Font("MS UI Gothic", 0, 36)); // NOI18N
41     jButton1.setText("up");
42     jButton1.addActionListener(new java.awt.event.ActionListener() {
43         public void actionPerformed(java.awt.event.ActionEvent evt) {
44             jButton1ActionPerformed(evt);
45         }
46     });
47
48     javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
49     getContentPane().setLayout(layout);
```

このメソッドの中で、
GUIコンポーネントが
生成されて、
配置されている。

文字サイズなども
設定されている
様子が分かる。
→補足

Layoutマネージャも
設定されているが、
これについては
今日は少しだけ

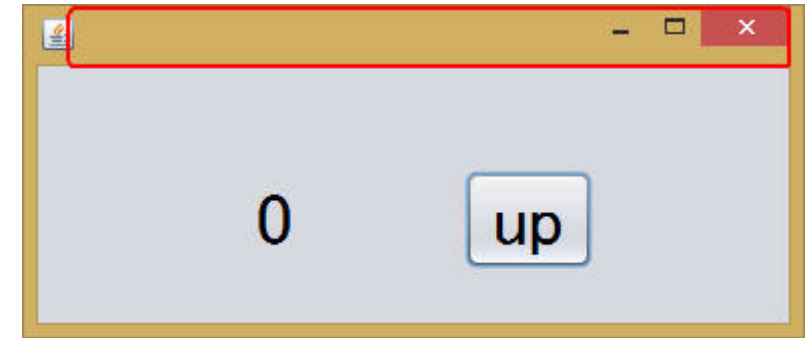
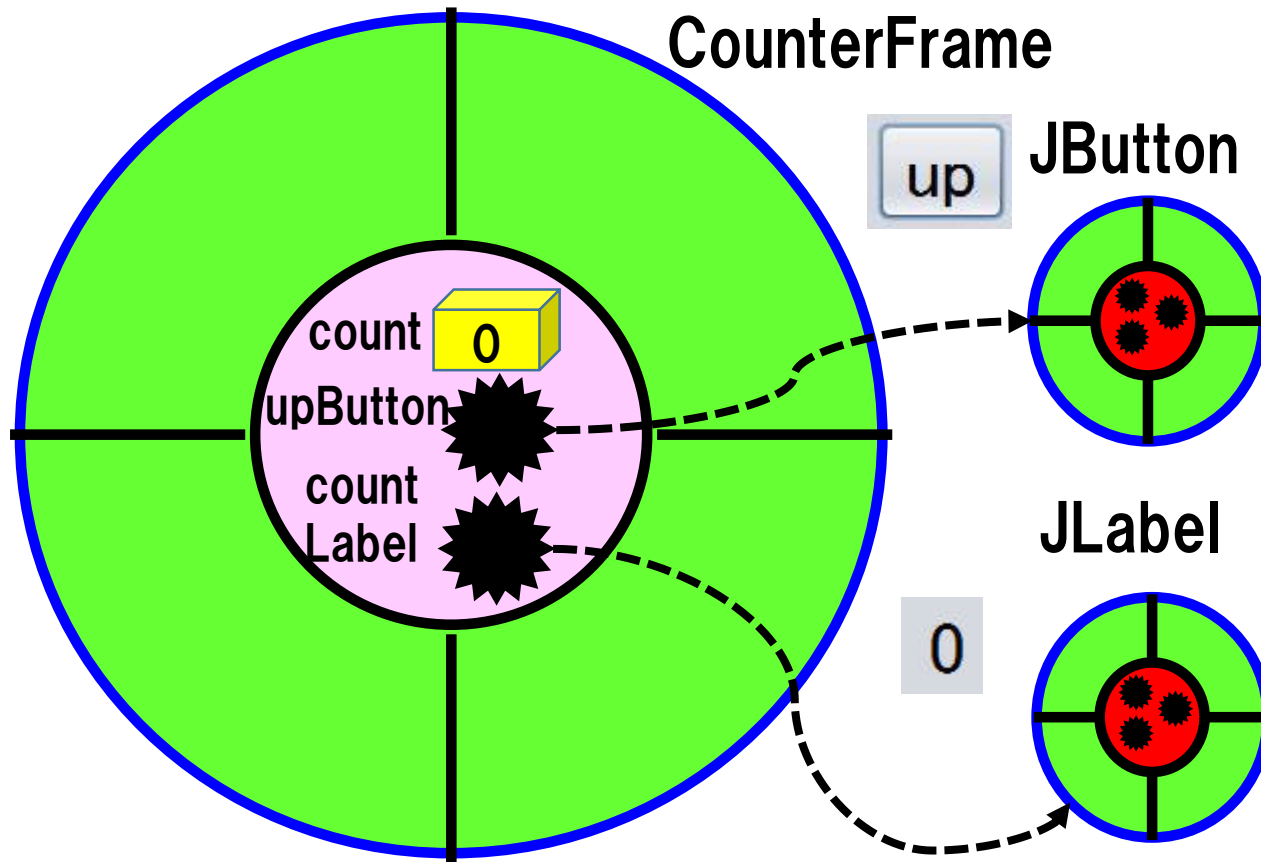
initCom
ponents
の中を
見てみ
ましょ
う

イベントハンドラも
GUI部品ごとに
わかり易い名前を
つけている。



前回, NetBeansで作ったCounterFrame

- 変数名を, 通し番号ではなく, わかりやすいモノに付け替えます



jButton1
↓
upButton

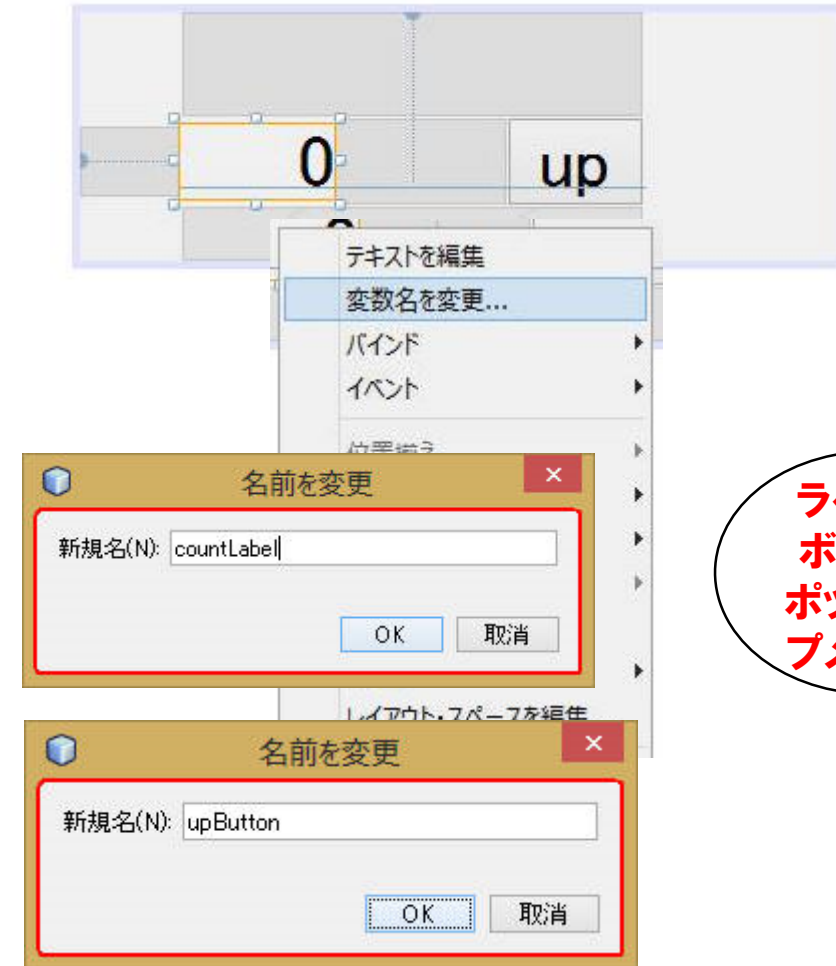
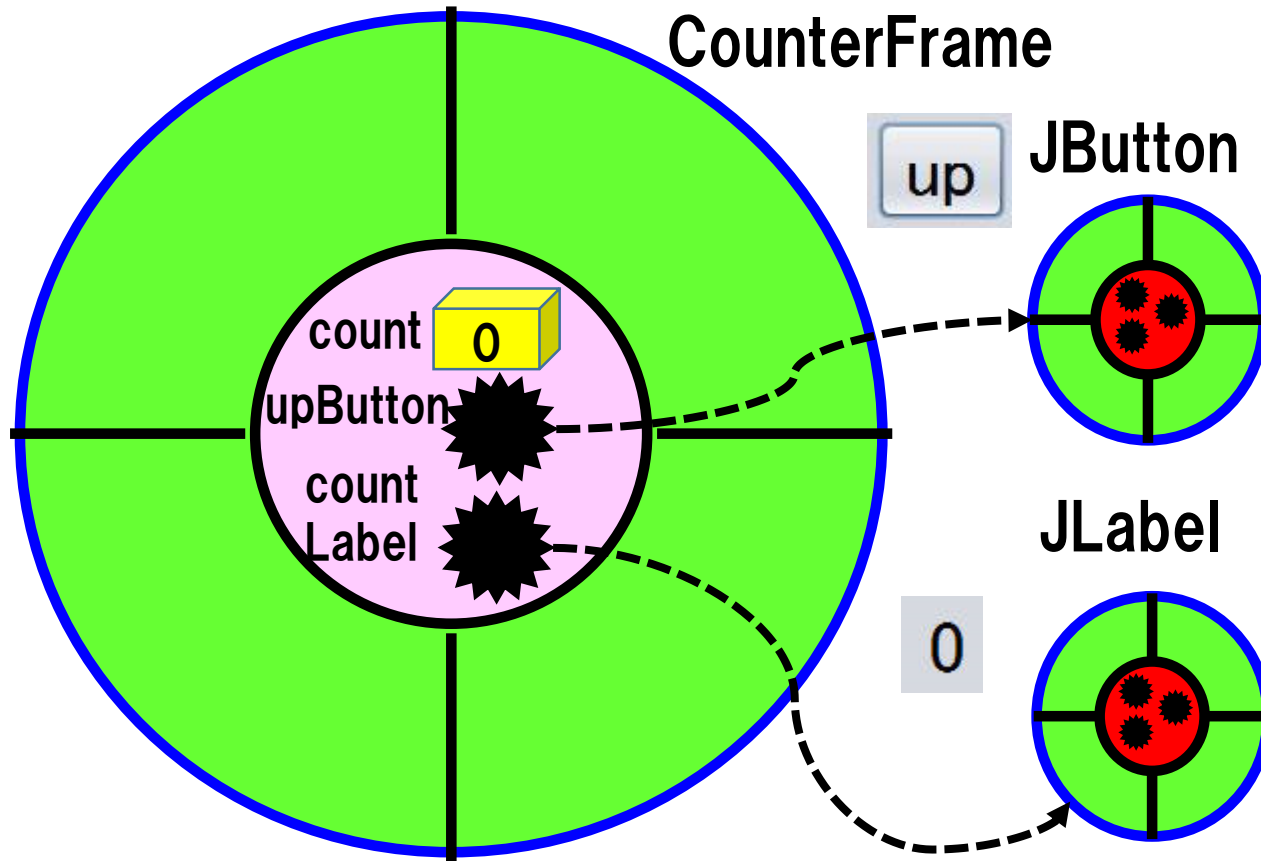
jLabel1
↓
countLabel

意味のある
名前に変更



前回, NetBeansで作ったCounterFrame

- 変数名を, 通し番号ではなく, わかりやすいモノに付け替えます

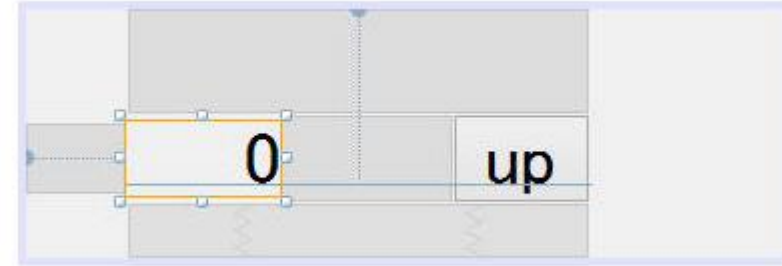
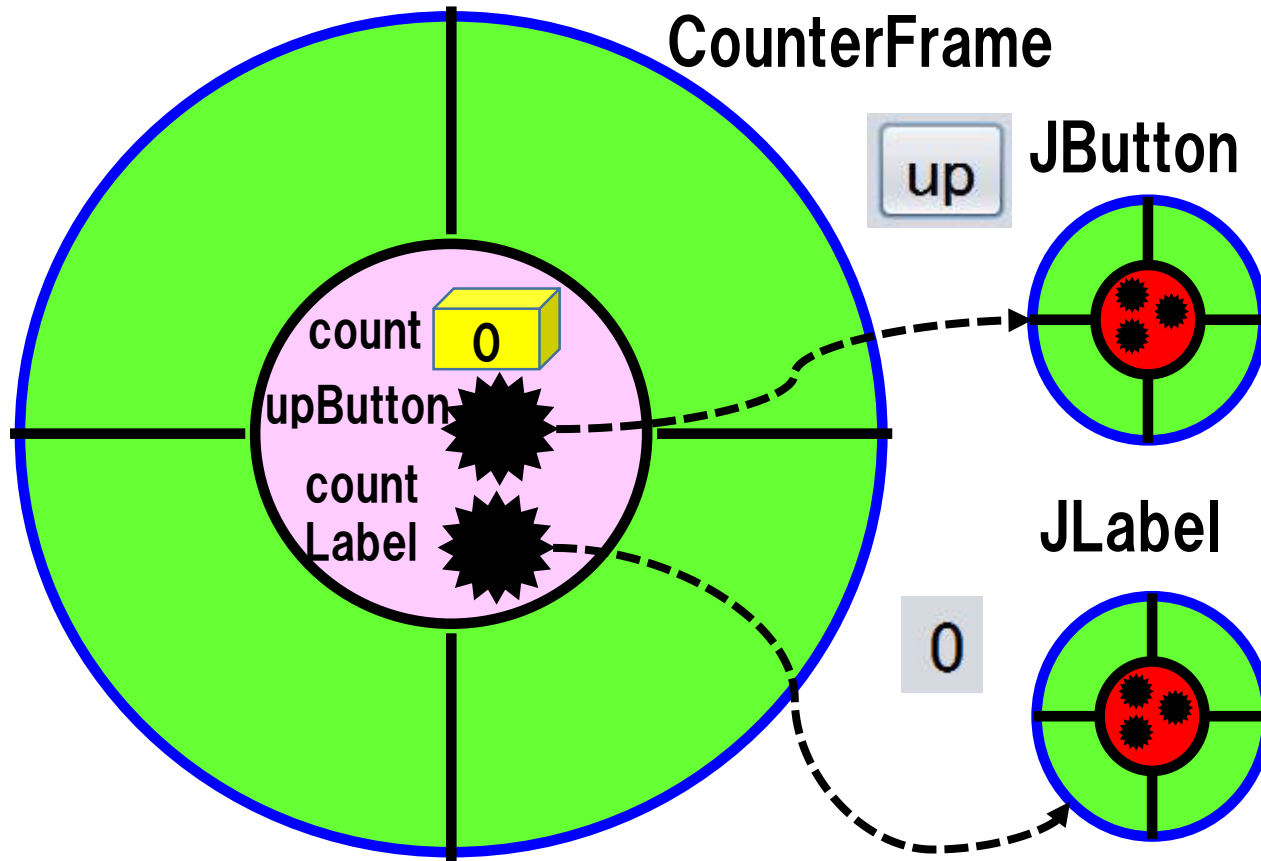


ラベルや
ボタンの
ポップアッ
プメニュー



前回, NetBeansで作ったCounterFrame

- 変数名を, 通し番号ではなく, わかりやすいモノに付け替えます



ソースコードに
反映されました

```
113  
114  
115  
116  
117
```

```
// Variables declaration - do not modify  
private javax.swing.JLabel countLabel;  
private javax.swing.JButton upButton;  
// End of variables declaration
```

```
}
```



話の流れ (第04回)

SEP04

21

第01回にzipで配布したP0104パッケージを確認する



アクセス指定とカプセル化



Javaのアクセス修飾子



NetBeansで作ったCounterFrameを確認する



NetBeansでのGUIコンポーネントのプロパティ指定



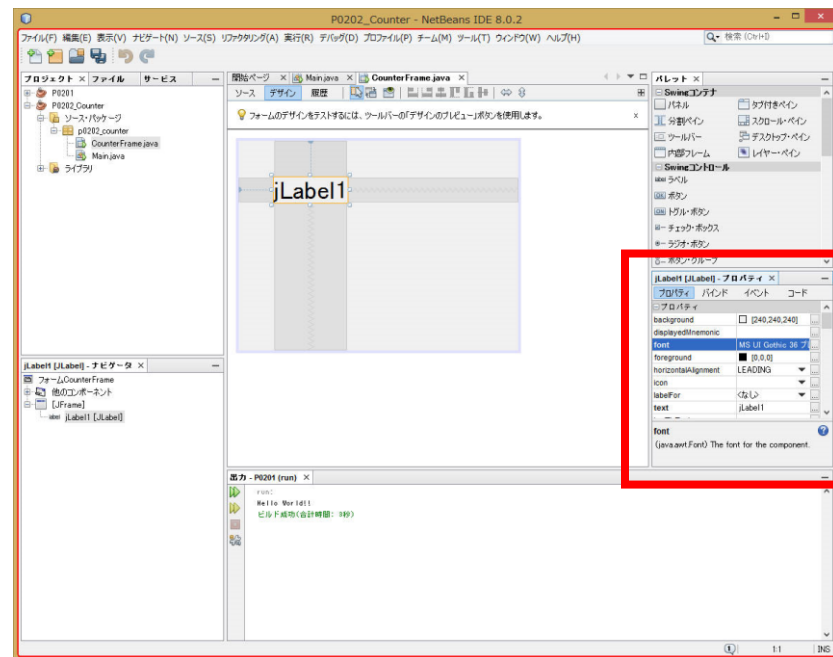
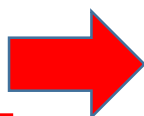
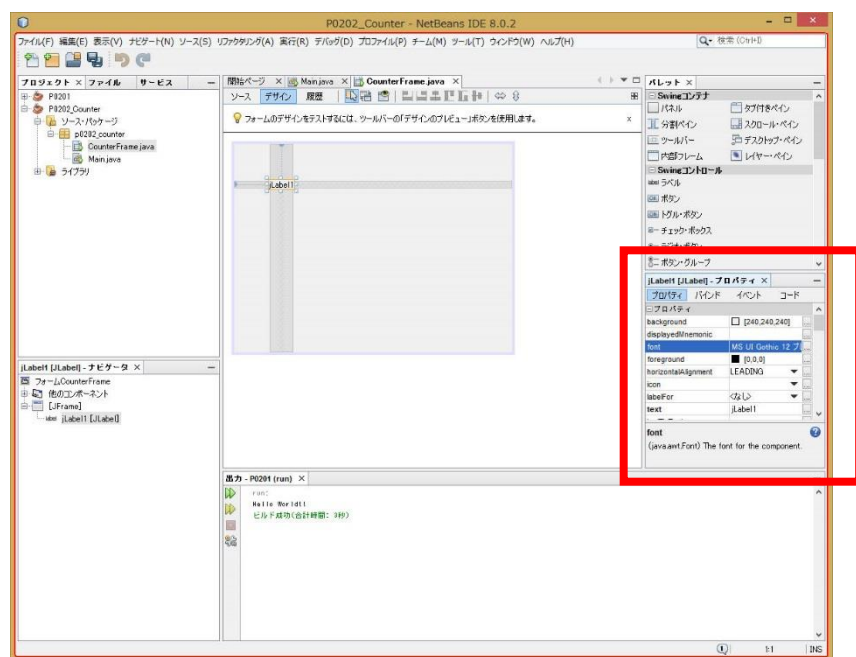
実習編: 「メソッドで属性を保護する」意義を確認

話の流れは
...



GUIコンポーネントのサイズや文字の指定

- Netbeansなら, プロパティエディタで, できてしまいますが...

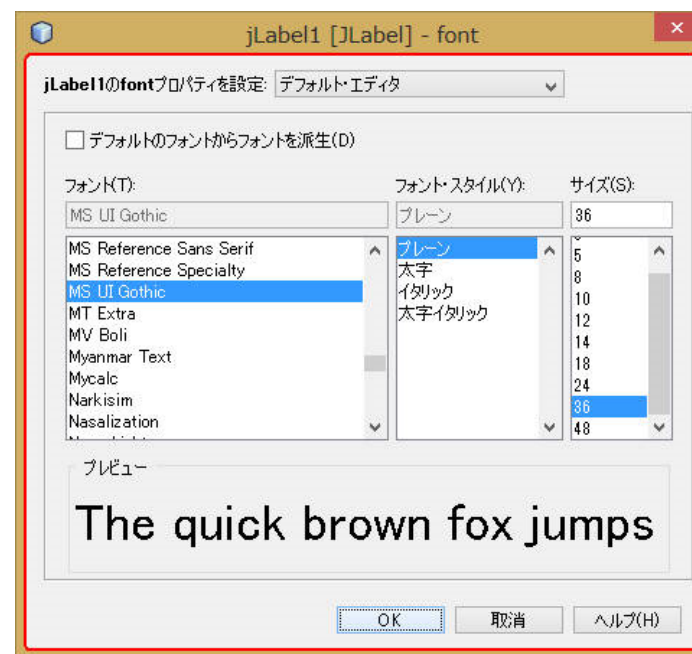
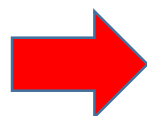
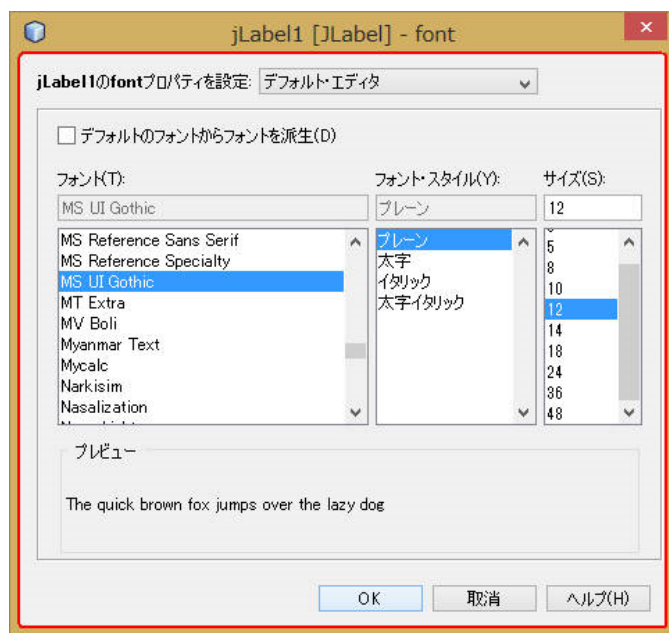


Netbeansなら,
プロパ
ティエ
ディ
タででき
てしま
います



GUIコンポーネントのサイズや文字の指定

- Netbeansなら、プロパティエディタで、できてしまいますが…



簡単ですね



前回, NetBeansで作ったCounterFrame

- NetBeansが自動生成したコード (initComponents) が畳み込まれている

```
28 // <editor-fold defaultstate="collapsed" desc="Generated Code">
29 private void initComponents() {
30
31     jLabel1 = new javax.swing.JLabel();
32     jButton1 = new javax.swing.JButton();
33
34     setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
35
36     jLabel1.setFont(new java.awt.Font("MS UI Gothic", 0, 36)); // NOI18N
37     jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
38     jLabel1.setText("0");
39
40     jButton1.setFont(new java.awt.Font("MS UI Gothic", 0, 36)); // NOI18N
41     jButton1.setText("up");
42     jButton1.addActionListener(new java.awt.event.ActionListener() {
43         public void actionPerformed(java.awt.event.ActionEvent evt) {
44             jButton1ActionPerformed(evt);
45         }
46     });
47
48     javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
49     getContentPane().setLayout(layout);
```

initComponents
の中を
見てみ
ましょ
う



各コンポーネントの大きさ

→レイアウトマネージャーをnullに設定しない限り、
最終的にレイアウトマネージャーが管理する。

例)

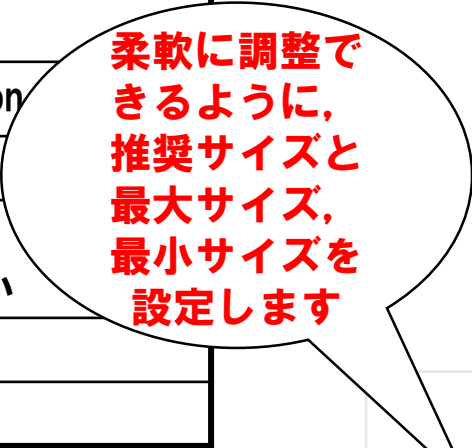
- BorderLayout →配置する位置によって元々のコンポーネントの大きさを尊重する部分と尊重せずに変更してしまう部分とができる
- GridLayoutでは元々のコンポーネントのサイズは無視する
- FlowLayoutのように元々のコンポーネントのサイズをそのまま尊重するレイアウトマネージャーもある

コンポーネント
の大きさは、
レイアウト
マネージャ
によって
影響を受けます



コンポーネントの大きさ指定

推奨 サイズ	void	setPreferredSize (Dimension preferredSize)	コンポーネントの適切なサイズを設定する。
			preferredSize が null の場合、UI で適切なサイズを要求する。
			preferredSize - 新しい推奨サイズ、または null
最大 サイズ	void	setMaximumSize (Dimension maximumSize)	コンポーネントの最大サイズを定数値に設定する。
			getMaximumSize の以降の呼び出しで、常にこの値を返す。 その計算のためにコンポーネントの UI が要求されることはない。
			最大サイズを null に設定すると、デフォルトの動作に戻る
			maximumSize - 要求される最大許可サイズを保持する Dimension
最小 サイズ	void	setMinimumSize (Dimension minimumSize)	コンポーネントの最小サイズを定数値に設定する。
			getMinimumSize の以降の呼び出しで、常にこの値を返す。 その計算のためにコンポーネントの UI が要求されることはない
			最小サイズを null に設定すると、デフォルトの動作に戻る。
			minimumSize - このコンポーネントの新しい最小サイズ



※ディメンジョンの指定にはインスタンスを生成する⇒ new Dimension (200,100)

文字フォント，スタイル，サイズ指定

SEP04

27

フォント (font)**文字のデザインに関するデータ**

フォント名	"Dialog"
	"DialogInput"
	"Monospaced"
	"Serif"
	"SansSerif"
	"Symbol"

字体指定	標準	Font.PLAIN	
	太字	Font.BOLD	
	斜体	Font.ITALIC	
	太字かつ斜体	Font.BOLD Font.ITALIC	

文字の大きさ**ポイント数で指定**

Fontインスタンス の生成の例	new Font("SansSerif", Font.BOLD, 24)
	SansSerif のボールド体の24ポイント
GUI部品へのフォント指定	GUI部品のインスタンス.setFont (フォントのインスタンス)

文字フォントもオブジェクトです。
フォント名や字体、
サイズを指定して
生成します



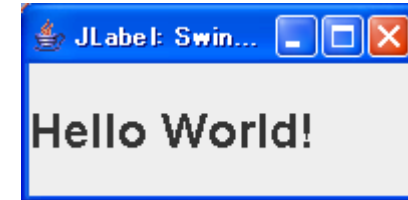
ラベルの文字指定 (1 / 2)

SEP04

28

```
package swingtest;
import javax.swing.*;
import java.awt.*;

public class SwingTest {
    public static void main (String [] args) {
        SwingTestFrame aFrame = new SwingTestFrame ();
        aFrame.setVisible (true);
    }
}
```



簡単な例で、
フォントの設定
について
説明します

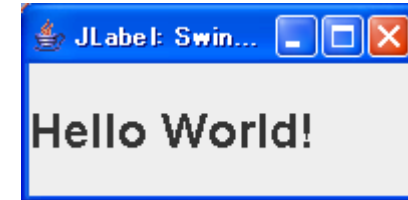


ラベルの文字指定 (2/2)

SEP04

29

```
class SwingTestFrame extends JFrame {  
    private JLabel    aLabel;  
    private Container aContentPane;  
  
    SwingTestFrame () {  
        super ( "JLabel: SwingTestFrame" );  
        aLabel = new JLabel ( "Hello World!" );  
        Font font = new Font ( "SansSerif", Font.BOLD, 24 );  
        aLabel.setFont ( font );  
        aContentPane = getContentPane ();  
        aContentPane.add ( aLabel );  
        setDefaultCloseOperation ( JFrame.EXIT_ON_CLOSE );  
        setBounds ( 10, 10, 400, 300 );  
        pack ();  
    }  
}
```



ラベルに
フォントを
セットします



話の流れ (第04回)

SEP04

30

第01回にzipで配布したP0104パッケージを確認する



アクセス指定とカプセル化



Javaのアクセス修飾子



NetBeansで作ったCounterFrameを確認する



NetBeansでのGUIコンポーネントのプロパティ指定



実習編: 「メソッドで属性を保護する」意義を確認

実習編は
独立した
プレゼンです

