Ｊａｖａ入門

（№.６）

科　　名		氏　　名	
Tutorial group			

2005.2

1 テキストフィールド(TextField)

テキストフィールド(TextField)は､1行だけの編集可能な文字列を表示/入力するためのコンポーネントである。

テキストフィールドは、TextComponent を継承している。

【例題 1】

［ファイル名］

P326password.java

［プログラムの説明］

テキスト・P325～P326 password クラスのプログラムを作成しなさい。ただし、クラス名"password"はクラス名"P326password"とすること。

【考察】

2 ラベル(Label)

ラベル(Label)は、1行だけの編集不可能な文字列を表示するためのコンポーネントである。

表示される文字列は、初期状態ではコンポーネントの"**左揃え**"で表示される。

【例題 2】

［ファイル名］

P328label.java

［プログラムの説明］

テキスト・P328 label クラスのプログラムを作成しなさい。ただし、クラス名"label"はクラス名"P328label"とすること。

【考察】

3 ボタン(Button)

ボタン(Button)は、マウスで Click して様々な処理を実行させる場合に用いるコンポーネントである。

【例題 3】

［ファイル名］

P332button.java

［プログラムの説明］

テキスト・P331～P332 button クラスのプログラムを作成しなさい。ただし、クラス名"button"はクラス名"P332button"とすること。

【考察】

【例題 4】

［ファイル名］

P332buttonA.java, P332buttonB.java, P332buttonC.java

［プログラム１］－ "P332button.java"をコピーして良い。

```
import  java.applet.Applet;
import  java.awt.*;
import  java.awt.event.*;

/*<APPLET
    CODE = P332buttonA.class
    WIDTH = 200
    HEIGHT = 200>
</APPLET>*/

public  class   P332buttonA extends Applet  implements  ActionListener  {
    TextField   text1;
    Button      button1;

    public  void    init(){
        text1 = new TextField(20);
        add(text1);
        button1 = new Button("Click Here!");
        add(button1);
        button1.addActionListener(this);
    }

    public  void    actionPerformed(ActionEvent event){   ←――①
        String  msg = new String("Hello from Java!");
```

```
        text1.setText(msg);
    }
}
```

【考察】

・ actionPerformed()メソッドについて考察しなさい。

［プログラム２］－ "P332buttonA.java"をコピーして良い。

```
import  java.applet.Applet;
import  java.awt.*;
import  java.awt.event.*;

/*<APPLET
    CODE = P332buttonB.class
    WIDTH = 200
    HEIGHT = 200>
</APPLET>*/

public  class   P332buttonB extends Applet  implements  ActionListener  {
    TextField   text1;
    Button      button1, button2;    <-------------- ②

    public  void    init(){
        text1 = new TextField(20);
        add(text1);
        button1 = new Button("Click Here!");
        add(button1);
        button1.addActionListener(this);
        button2 = new Button("Click Here!");
        add(button2);                              <-- ③
        button2.addActionListener(this);
    }

    public  void    actionPerformed(ActionEvent event){   <-- ①
        String  msg = new String("Hello from Java!");
        text1.setText(msg);
    }
}
```

【考察】

・ actionPerformed()メソッドについて考察しなさい。

[プログラム３]－ "P332buttonB.java"をコピーして良い。

```
import  java.applet.Applet;
import  java.awt.*;
import  java.awt.event.*;

/*<APPLET
    CODE = P332buttonC.class
    WIDTH = 200
    HEIGHT = 200>
</APPLET>*/

public  class   P332buttonC extends Applet  implements  ActionListener  {
    TextField   text1;
    Button      button1, button2;

    public  void    init(){
        text1 = new TextField(20);
        add(text1);
        button1 = new Button("Click Here!");
        add(button1);
        button1.addActionListener(this);
        button2 = new Button("Click Here!");
        add(button2);
        button2.addActionListener(this);
    }

    public  void    actionPerformed(ActionEvent event){
        String  msg = new String("Hello from Java!");
        if(event.getSource() == button1){
            text1.setText(msg);
        }
        if(event.getSource() == button2){
            text1.setText("Hello to Java!");
        }
    }
}
```

【考察】

- actionPerformed()メソッドについて考察しなさい。
- getSource()メソッドについて考察しなさい。
- ActionEvent クラスについて考察しなさい。

【例題 5】

［ファイル名］

P333button2.java

［プログラムの説明］

テキスト・P332～P333 button2 クラスのプログラムを作成しなさい。ただし、クラス名"button2"はクラス名"P333button2"とすること。

【考察】

【例題 6】

［ファイル名］－ "P333button.java"をコピーして良い。

P333button2A.java, P333button2B.java, P333button2C.java

［プログラムの説明］

```
import  java.applet.Applet;
import  java.awt.*;
import  java.awt.event.*;

/*<APPLET
    CODE = P333button2A.class
    WIDTH = 200
    HEIGHT = 200>
</APPLET>*/

public  class   P333button2A    extends Applet  implements  ActionListener  {
    TextField       text1;
    Button          button1;

    public  void    init(){
        text1 = new TextField(20);
        add(text1);
        button1 = new Button("Click Here!");
        add(button1);
        button1.addActionListener(this);
    }

    public  void    actionPerformed(ActionEvent event){
        String  msg = new String("Hello from Java!");
        String  command = event.getActionCommand();
        if(command.equals("Click Here!")){
            text1.setText(msg);
        }
    }
```

② 削除

①

③ 変更

}

【考察】

[プログラム２]－ "P333buttonA.java"をコピーして良い。

```
import  java.applet.Applet;
import  java.awt.*;
import  java.awt.event.*;

/*<APPLET
    CODE = P333button2B.class
    WIDTH = 200
    HEIGHT = 200>
</APPLET>*/

public  class   P333button2B    extends Applet  implements  ActionListener  {
    TextField   text1;
    Button      button1, button2;

    public  void    init(){
        text1 = new TextField(20);
        add(text1);
        button1 = new Button("Click Button1");
        add(button1);
        button1.addActionListener(this);
        button2 = new Button("Click Button2");
        add(button2);
        button2.addActionListener(this);
    }

    public  void    actionPerformed(ActionEvent event){
        String  msg = new String("Hello from Java!");
        String  command = event.getActionCommand();
        if(command.equals("Click Button1")){
            text1.setText(msg);
            button1.setLabel("Click There!");
        }
        if(command.equals("Click Button2")){
            text1.setText("Hello to Java!");
        }
    }
}
```

②

①

③

④

【考察】

[プログラム３]

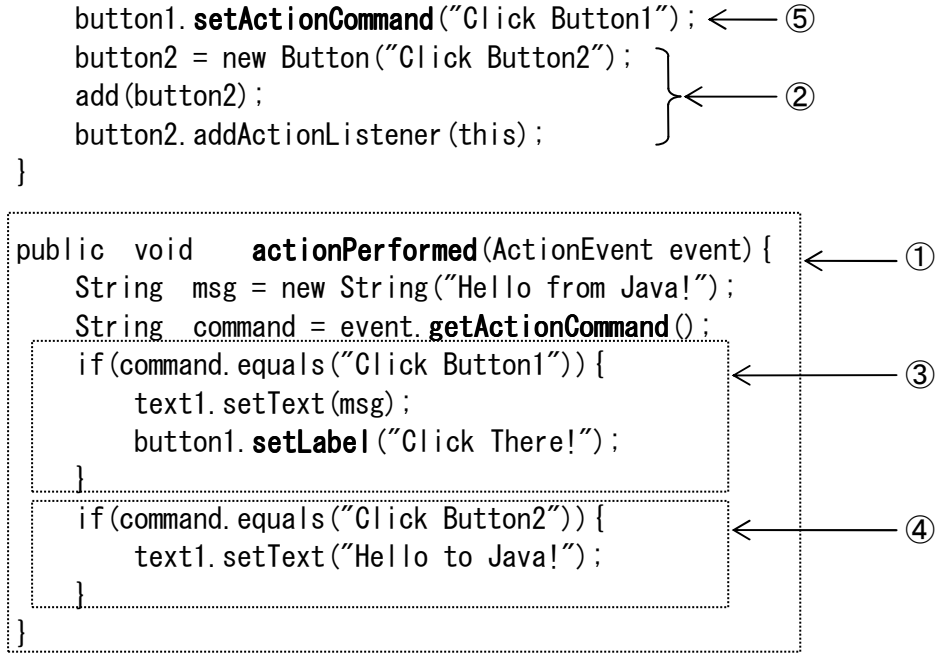
```
import  java.applet.Applet;
import  java.awt.*;
import  java.awt.event.*;

/*<APPLET
    CODE = P333button2C.class
    WIDTH = 200
    HEIGHT = 200>
</APPLET>*/

public  class   P333button2C    extends Applet  implements  ActionListener  {
    TextField   text1;
    Button      button1, button2;

    public  void    init(){
        text1 = new TextField(20);
        add(text1);
        button1 = new Button("Click Button1");
        add(button1);
        button1.addActionListener(this);
        button1.setActionCommand("Click Button1");
        button2 = new Button("Click Button2");
        add(button2);
        button2.addActionListener(this);
    }

    public  void    actionPerformed(ActionEvent event){
        String  msg = new String("Hello from Java!");
        String  command = event.getActionCommand();
        if(command.equals("Click Button1")){
            text1.setText(msg);
            button1.setLabel("Click There!");
        }
        if(command.equals("Click Button2")){
            text1.setText("Hello to Java!");
        }
    }
}
```



【考察】

4 チェックボックス(Checkbox)

チェックボックス(Checkbox)は、ある項目を有効にするか、あるいは無効にするかの二者択一を行なわせる場合に用いるコンポーネントである。

また、**チェックボックス**は"**CheckboxGroup**"**クラス**を利用することで、複数項目の中から一つを選択する**ラジオボタン(radio button)**として使用することができる。

【例題 7】

［ファイル名］

P336checks.java

［プログラムの説明］

テキスト・P336～P337 checks クラスのプログラムを作成しなさい。ただし、クラス名"checks"はクラス名"P336checks"とすること。

【考察】

［プログラムの変更］

"checkbox1.addItemListener(this);"の下に、次の文を挿入しなさい。

checkbox1.**setState**(true); ⇦ 挿入

なお、ファイル名は"P336checksA.java"に、クラス名は"P336checksA"とすること。

【考察】

・ setState()メソッド、getItemSelectable()メソッドについて、考察しなさい。

【例題 8】

［ファイル名］

P338checks2.java

［プログラムの説明］

テキスト・P338～P339 checks2 クラスのプログラムを作成しなさい。ただし、クラス名"checks2"はクラス名"P338checks2"とすること。

【考察】

【例題 9】

［ファイル名］

P340radios.java

［プログラムの説明］

テキスト・P340～P341 radios クラスのプログラムを作成しなさい。ただし、クラス名"radios"はクラス名"P340radios"とすること。

【解説】

Java では、**ラジオボタン**は CheckboxGroup クラスのオブジェクト変数を Checkbox()**コンストラクタ**に指定することで実装する。

【考察】

5 レイアウトマネージャ(layout manager)

Java では、様々な環境下で一つのプログラムが正常に動作することを前提としているため、コンポーネントの配置、形状や大きさなどを、現在の表示の状況に応じて自動的に調整するような仕組みを持っている。この機能を**レイアウトマネージャ**と呼ぶ。

レイアウトマネージャは、あらかじめ数種類用意されており、各コンテナにはデフォルトの**レイアウトマネージャ**クラスのオブジェクト(インスタンス)が実装されている。

(1) **FlowLayout クラス**

矩形領域を提供するクラス(Panel クラスなど)やアプレット(java.applet.Applet)で標準で実装されている**レイアウトマネージャ**である。

FlowLayout クラスは、実装したコンポーネントを左上から順に横一列に整列・配置し、もし一列で収まらなかった場合は自動的に改行して複数列に整列・配置する。

このクラスでは、コンポーネントの実装の際に、配置する場所の指定がいらない。

(2) **GridLayout クラス**

GridLayout クラスは、コンテナ全体を幾つかの領域に分け、その格子状になった領域にコンポーネントを配置する。GridLayout クラスでは、格子の数や間隔などを設定できる。

なお、コンポーネントの配置の順序は左上から横方向に順番に整列・配置される。

【例題 10】

［ファイル名］

P343flow.java

［プログラムの説明］

テキスト・P343～P344 flow クラスのプログラムを作成しなさい。ただし、クラス名"flow"はクラス名"P343flow"とすること。

【考察】

・ setLayout()メソッドをコメントアウトして、動作を確認しなさい。

【例題 11】

［ファイル名］

P344multiplier.java

［プログラムの説明］

テキスト・P344～P345 multiplier クラスのプログラムを作成しなさい。ただし、クラス名"multiplier"はクラス名"P344multiplier"とすること。

【解説】－ ラッパークラス(wrapper class)

Java では、int 型や char 型などの**基本データ型**(primitive)に対応したクラスが定義されている。これを**ラッパークラス(wrapper class)**と言う。

ラッパークラスは、**基本データ型**の値をオブジェクトとして扱えるようにする。

基本データ型データのオブジェクト化

基本データ型	ラッパークラス	コンストラクタ	値取得
byte	Byte	Byte(byte)	byteValue()
short	Short	Short(short)	shortValue()
int	Integer	Integer(int)	intValue()
long	Long	Long(long)	longValue()
float	Float	Float(float)	floatValue()
double	Double	Double(double)	doubleValue()
char	Character	Character(char)	charValue()
boolean	Boolean	Boolean(boolean)	booleanValue()

ラッパークラスは、次のような場合に使用する。

(1) **基本データ型**データのオブジェクト化

java.util に含まれる多くのクラスは、オブジェクトを対象としているが、**基本データ型**データにこれらのクラスを適用したい場合。⇒ java.util.Vector など

(2) 文字列(String クラス)との相互変換

基本データ型データを文字列に変換したり、文字列を**基本データ型**の値に変換する場合。⇒ toString(**基本データ型**)メソッド, parseXXXX(String 型)メソッド

◇ **基本データ型 → 文字列変換**

ラッパークラスの toString()メソッドあるいは String クラスの valueOf()メソッドで変換する。いずれもクラスメソッドであるから、オブジェクトの生成は不要である。

◇ **文字列 → 基本データ型変換**

int 型データへの変換には、次のクラスメソッドを使用する。

・ Integer.parseInt(String)

・ Integer.parseInt(String, int)

(例)

```
int     int1 = 100;              //  基本データ型(int)変数の定義と初期化
Integer int2 = new Integer(200);//  Integer 型オブジェクトの生成
```

【考察】

[プログラムの変更]

"text1 = new TextField(10);"の直前に、次の文を挿入しなさい。

setLayout(new FlowLayout(FlowLayout.LEFT)); ⇦ 挿入

なお、ファイル名は"P344multiplierA.java"に、クラス名は"P344multiplierA"とすること。

【考察】

【例題 12】

[ファイル名]

P347multiplier2.java

[プログラムの説明]

テキスト・P347～P348 multiplier2 クラスのプログラムを作成しなさい。ただし、クラス名"multiplier2"はクラス名"P347multiplier2"とすること。

【考察】

【例題 13】

[ファイル名]

Samp13.java

[プログラム]

```
import  java.applet.Applet;
import  java.awt.*;
```

```
/*<APPLET
    CODE = Samp13.class
    WIDTH = 200
    HEIGHT = 200>
</APPLET>*/

public  class   Samp13  extends Applet  {
    public  void    init(){
        setLayout(new GridLayout(3, 2));
        Button  b1 = new Button("A");
        Button  b2 = new Button("B");
        Button  b3 = new Button("C");
        Button  b4 = new Button("D");
        Button  b5 = new Button("E");
        Button  b6 = new Button("F");
        add(b1);
        add(b2);
        add(b3);
        add(b4);
        add(b5);
        add(b6);
    }

    public  Insets  getInsets(){            //  getInset()をオーバーライドする
        return new Insets(50, 10, 10, 10);
    }
}
```

【解説】－ Insets クラス

Insets クラスは、コンテナの境界の周囲の上下左右の間隔に関する情報をカプセル化する。

【構文】－ Insets クラス

Insets(int *top*, int *left*, int *bottom*, int *right*)

top … コンポーネントの上部の空間の大きさをピクセル単位で指定する。

left … コンポーネントの左側の空間の大きさをピクセル単位で指定する。

bottom … コンポーネントの下部の空間の大きさをピクセル単位で指定する。

right … コンポーネントの右側の空間の大きさをピクセル単位で指定する。

解説：コンテナの Insets を指定する場合、getInsets()メソッドをオーバーライドする。

【考察】

6 パネル(Panel)

パネル(Panel)は、他のコンポーネントを配置するための矩形の領域として表示されるコンテナである。

また、Panel オブジェクト自身もコンポーネントであるから、**パネル**のネストが可能である。

パネルは、通常 FlowLayout クラスの**レイアウトマネージャ**が実装されているが、setLayout()メソッドによって**レイアウトマネージャ**を変更できる。

【例題 14】

［ファイル名］

P350panels.java, P350panelsA.java, P350panelsB.java

［プログラムの説明］

テキスト・P349～P351 panels クラスのプログラムを作成しなさい。ただし、クラス名"panels"はクラス名"P350panels"とすること。

【考察】

［プログラムの変更］

(1) "setLayout(new GridLayout(2, 3));"をコメントアウトして、コンパイル・実行しなさい。ただし、クラス名"panels"はクラス名"P350panelsA"とすること。⇒ P350panelsA.java

(2) "setLayout(new GridLayout(2, 3));"を"setLayout(new GridLayout(2, 3, 20, 20));"に変更して、コンパイル・実行しなさい。ただし、クラス名"panels"はクラス名"P350panelsB"とすること。⇒ P350panelsB.java

【考察】

(3) **BorderLayout クラス**

BorderLayout クラスは、コンテナ全体を上下左右と中央の５か所の領域に分け、それぞれの領域にコンポーネントを配置する。

コンポーネントの配置位置は、BorderLayout クラス内のフィールド「CENTER」、「NORTH」、「SOUTH」、「EAST」及び「WEST」で指定する。

BorderLayout クラスは、ウィンドウ関係のコンテナ(Frame など)に標準で実装されている。

(4) **CardLayout クラス**

CardLayout クラスは、カードを重ねようにしてコンポーネントを重ね合せ、常に重ね合せた一番上のコンポーネントだけがたった一つ表示されるようにする**レイアウトマネージャ**

である。実装された各コンポーネントは、実装した順番に重ね合わされ整理される。
CardLayout クラスの show()、first()、last()、next()及び previous()メソッドによって表示するコンポーネントを移動・表示する。

(5) **GridBagLayout クラス**

GridBagLayout クラスは、GridLayout クラスの機能を強化し、自由なレイアウトを可能にした最も複雑で柔軟な**レイアウトマネージャ**である。
GridBagLayout クラスでは、GridBagConstraints クラスを利用してコンポーネントの位置などの設定を行なった後に、コンテナに実装する。

【例題 15】

［ファイル名］
P353border.java, P353borderA.java

［プログラムの説明］
テキスト・P353～P354 border クラスと textPanel クラスのプログラムを作成しなさい。ただし、クラス名"border"はクラス名"P353border"と、クラス名"textPanel"はクラス名"P353textPanel"とすること。

【考察】

［プログラムの変更］

- BorderLayout()コンストラクタの水平間隔と垂直間隔を設定して、"P353borderA"の動作を確認しなさい。⇒ P353borderA.java

【考察】

【例題 16】

［ファイル名］
P357card.java, P357cardA.java

［プログラムの説明］
テキスト・P356～P358 card クラスと cardPanel クラスのプログラムを作成しなさい。ただし、クラス名"card"はクラス名"P357card"と、クラス名"cardPanel"はクラス名"P357cardPanel"とすること。

【考察】

［プログラムの変更］

・ actionPerformed()メソッドを、次のように変更し、コンパイル・実行しなさい。

⇒ P357cardA.java

```
public void actionPerformed(ActionEvent event){
    cardlayout.next(this);
    repaint();
}
```

【考察】

【例題 17】

［ファイル名］

P363gridbag.java, P363gridbagA.java

［プログラムの説明］

テキスト・P363～P365 gridbag クラスのプログラムを作成しなさい。ただし、クラス名"gridbag"はクラス名"P363gridbag"とすること。

【考察】

［プログラムの変更］

・ "P363gridbag.java"に、テキスト･P365 getInsets()メソッドを追加コーディングし、コンパイル・実行しなさい。⇒ P363gridbagA.java
ただし、クラス名"gridbag"はクラス名"P363gridbagA"とすること。

【考察】

【例題 18】

［ファイル名］

P366multiplier3.java

［プログラムの説明］

テキスト・P366～P367 multiplier3 クラスのプログラムを作成しなさい。ただし、クラス名"gmultiplier3"はクラス名"P366multiplier3"とすること。

【考察】

7 Swing パッケージ

Java の GUI に、AWT コンポーネントの他に Swing コンポーネント群がある。

Swing は、Java 2 Standard Edition や Enterprise Edition から提供されている JFC(Java Foundation Classes)/Swing のことであり、大部分は AWT の Container クラスをベースにした JComponent クラスを元にしている。

Swing を使用したすべてプログラムは、最上位レベルの Swing コンテナを必ず一つ持たなければならない。最上位レベルの Swing コンテナには、JFrame、JDialog 及び JApplet などが有り、Swing コンポーネントが描画やイベント処理の処理に必要な機能を提供する。

Swing の主な GUI クラス

クラス	機　　能
JFrame	他のコンポーネントを納めて表示するウィンドウ
JDialog	ダイアログボックスの作成・表示
JButton	ボタンの作成・表示
JMenu	メニューの作成・表示
JCheckBox	チェックボックスの作成・表示
JRadioButton	ラジオボタンの作成・表示
JComboBox	コンボボックスの作成・表示
JTextField	1 行の文字列の入力フィールドの作成・表示
JTextArea	複数行の文字列の入力領域の作成・表示
JLabel	ラベルの作成・表示
JList	リストの作成・表示
JPanel	パネルの作成・表示
JTable	表の作成・表示
JTree	ツリーリストの作成・表示

これらの Swing コンポーネントを使用する場合、"javax.swing"パッケージをインポートする必要がある(概ね、AWT コンポーネントの各クラス名に"J"が付く)。

なお、イベント処理については、Swing コンポーネントが AWT イベント機能を使用するため、"java.awt"パッケージと"java.awt.event"パッケージをインポートする必要がある。

【例題 19】

[ファイル名]

Samp19.java, Samp19A.java, Samp19B.java

[プログラム１]

```
//  JFrame クラス(Swing)を扱うプログラム
import  java.lang.*;
import  javax.swing.*;
import  java.awt.*;

public  class   Samp19  extends JFrame  {
    Container   container = null;       //  Container クラスの変数の宣言と初期化

    public  Samp19(){                   //  コンストラクタ
        super("Try Java swing");        //  スーパークラスのコンストラクタ呼び出し
        container = getContentPane();   //  JFrame クラスのコンテンツペインの取得
        container.setLayout(new GridLayout(2, 2, 20, 20));

        JPanel  panel1 = new JPanel();  //  JPanel オブジェクトの生成
        JPanel  panel2 = new JPanel();
        JPanel  panel3 = new JPanel();
        JPanel  panel4 = new JPanel();
                                        //  JLabel オブジェクトの生成
        JLabel  label1 = new JLabel("映画", JLabel.CENTER);
        JLabel  label2 = new JLabel("音楽", JLabel.CENTER);
        JLabel  label3 = new JLabel("野球", JLabel.CENTER);
        JLabel  label4 = new JLabel("サッカー", JLabel.CENTER);

        panel1.add(label1);             //  JPanel に JLabel オブジェクトを追加
        panel2.add(label2);
        panel3.add(label3);
        panel4.add(label4);

        container.add(panel1);          //  JFrame に JPanel オブジェクトを追加
        container.add(panel2);
        container.add(panel3);
        container.add(panel4);
                                        //  JPanel オブジェクトの背景色の設定
        panel1.setBackground(Color.pink);
        panel2.setBackground(Color.yellow);
        panel3.setBackground(Color.red);
        panel4.setBackground(Color.blue);

        setDefaultCloseOperation(EXIT_ON_CLOSE);//  クローズ処理の設定
        setSize(300, 200);              //  フレームサイズの設定
        setVisible(true);               //  フレームを表示
    }

    public  static  void    main(String args[]){
        Samp19  frame = new Samp19();   //  オブジェクトの生成
    }
}
```

【解説】

通常、**コンテナ**となり得る**コンポーネント**には、その他の**コンポーネント**を組み込むことがで

きるが、JFrame クラス、JApplet クラス、JWindow クラス及び JDialog クラスについては直接**コンポーネント**を組み込むことができない。
すなわち、これらのクラスの**コンポーネント**の追加やレイアウトなどの処理は、これらのクラスの内部にある JRootPane という**コンテナ**を介して行う。具体的には、JRootPane 中の contentPane に対して行うことになる。
したがって、getContentPane()メソッドによって**コンテンツペイン**(java.awt.Container)を取得する必要がある。

【考察】

［プログラム２］

```
//  JFrame クラス(Swing)を扱うプログラム
import  java.lang.*;
import  javax.swing.*;
import  java.awt.*;

public  class   Samp19A extends JFrame  {
    Container   container = null;       //  Container クラスの変数の宣言と初期化

    public  Samp19A(){                  //  コンストラクタ
        super("Try Java swing <MotifLookAndFeel>");
        container = getContentPane();   //  Jframe クラスのコンテンツペインの取得
        container.setLayout(new GridLayout(2, 2, 20, 20));

        JPanel panel1 = new JPanel();   //  JPanel オブジェクトの生成
        JPanel panel2 = new JPanel();
        JPanel panel3 = new JPanel();
        JPanel panel4 = new JPanel();
                                        //  JLabel オブジェクトの生成
        JLabel label1 = new JLabel("映画", JLabel.CENTER);
        JLabel label2 = new JLabel("音楽", JLabel.CENTER);
        JLabel label3 = new JLabel("野球", JLabel.CENTER);
        JLabel label4 = new JLabel("サッカー", JLabel.CENTER);

        panel1.add(label1);             //  JPanel に JLabel オブジェクトを追加
        panel2.add(label2);
        panel3.add(label3);
        panel4.add(label4);

        container.add(panel1);          //  JFrame に JPanel オブジェクトを追加
        container.add(panel2);
        container.add(panel3);
        container.add(panel4);
                                        //  JPanel オブジェクトの背景色の設定
```

```
        panel1.setBackground(Color.pink);
        panel2.setBackground(Color.yellow);
        panel3.setBackground(Color.red);
        panel4.setBackground(Color.blue);

        setDefaultCloseOperation(EXIT_ON_CLOSE);//  クローズ処理の設定
        setSize(300, 200);              //  フレームサイズの設定
        setVisible(true);               //  フレームを表示
        SwingUtilities.updateComponentTreeUI(this);
    }   //  指定したコンポーネントとそれに add されたすべてのコンポーネントの表示更新

    public  static  void    main(String args[]){
        try{                            //  Look&Feel の設定(選択)
//          UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
//          UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
        }catch(Exception ex){
            ex.printStackTrace();
        }

        Samp19A frame = new Samp19A();
    }
}
```

【解説】

Swing には、Look&Feel に関する管理を行なう**"UIManager"**が用意されている。また、各**コンポーネント**にはUI クラスが用意されており、これが**コンポーネント**の表示を処理している。

Swing **コンポーネント**は、**"Model(構造)、View(表示)、Control(制御)"**の三つに分けて管理されているが、この View(表示)を行なっている部分が UI クラスである。

特定のデザインの元に全ての**コンポーネント**の表示を規定する UI クラスが作成され、一つのパッケージとしてまとめられたものが Look&Feel の元となる。

"UIManager"は、Java の Look&Feel を設定・変更する機能を持っている。

【構文】－ **setLookAndFeel()メソッド**

UIManager.setLookAndFeel(*look&feel*);

look&feel … LookAndFeel クラス名を指定する。

解説：現在の Java で設定されている Look&Feel を、*look&feel* で指定した Look&Feel に変更する。なお、このメソッドは、例外を発生させることがあるため、try{}内に記述する。

Java2 の標準 Look&Feel パッケージ

javax.swing.plaf.metal.MetalLookAndFeel	Java の標準 Look&Feel パッケージ。Metal と呼ばれる。
com.sun.java.swing.plaf.motif.MotifLookAndFeel	UNIX の MOTIFLook&Feel パッケージ。MOTIF と呼ばれる。
com.sun.java.swing.plaf.windows.WindowsLookAndFeel	Windows の Look&Feel パッケージ。

【構文】－ updateComponentTreeUI()メソッド

SwingUtilities.updateComponentTreeUI(*instance*);

instance … 更新するコンポーネントを指定する。

解説：*instance* で指定したコンポーネントと、それに追加(add)されている全てのコンポーネントの UI 表示を更新する。

【考察】

[プログラム３]

```
//  JFrame クラス(Swing)を扱うプログラム
import  java.lang.*;
import  javax.swing.*;
import  java.awt.*;
import  java.awt.event.*;

public  class   Samp19B extends JFrame  implements  ActionListener  {
    Container   container = null;       //  Container クラスの変数の宣言と初期化

    public  Samp19B(){                  //  コンストラクタ
        super("Try Java swing");        //  スーパークラスのコンストラクタ呼び出し
        container = getContentPane();
        container.setLayout(new GridLayout(3, 2, 20, 20));

        JPanel panel1 = new JPanel();   //  JPanel オブジェクトの生成
        JPanel panel2 = new JPanel();
        JPanel panel3 = new JPanel();
        JPanel panel4 = new JPanel();
                                        //  JLabel オブジェクトの生成
        JLabel label1 = new JLabel("映画", JLabel.CENTER);
        JLabel label2 = new JLabel("音楽", JLabel.CENTER);
        JLabel label3 = new JLabel("野球", JLabel.CENTER);
        JLabel label4 = new JLabel("サッカー", JLabel.CENTER);

        panel1.add(label1);             //  JPanel に JLabel オブジェクトを追加
        panel2.add(label2);
        panel3.add(label3);
        panel4.add(label4);

        container.add(panel1);          //  JFrame に JPanel オブジェクトを追加
        container.add(panel2);
        container.add(panel3);
        container.add(panel4);
                                        //  JPanel オブジェクトの背景色の設定
```

```
        panel1.setBackground(Color.pink);
        panel2.setBackground(Color.yellow);
        panel3.setBackground(Color.red);
        panel4.setBackground(Color.blue);
                                        //  JButtonオブジェクトの生成
        JButton button1 = new JButton("Click Here!");
        container.add(button1);         //  JFrameにJButtonオブジェクトの追加
        button1.addActionListener(this);//  JButtonにActionLisenerの追加

        setDefaultCloseOperation(EXIT_ON_CLOSE);//  クローズ処理の設定
        setSize(300, 200);              //  フレームサイズの設定
        setVisible(true);               //  フレームを表示
    }

    public  static  void    main(String args[]){
        Samp19B frame = new Samp19B();
    }

    public  void    actionPerformed(ActionEvent e){
        try{                            //  Look&Feelの設定(選択)
//          UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
//          UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
        }catch(Exception ex){
            ex.printStackTrace();
        }
        SwingUtilities.updateComponentTreeUI(this);
    }   //  指定したコンポーネントとそれにaddされたすべてのコンポーネントの表示更新
}
```

【考察】

- 上記のプログラムは、アプリケーションを"MetalLookAndFeel"で起動し、JButton コンポーネントをclickするとアプリケーションを"MotifLookAndFeel"で表示する。
- Swing コンポーネント群を使用した場合のイベント処理は、基本的には AWT のイベント処理(delegate event model)と同様である。