Chapter 22.Swing・AWTのコンポーネント

２２－１．コンポーネントクラス

２２－１－１．コンポーネントクラスの階層

SwingのGUIの部品としてアプレット上に配置され、何らかの操作の対象となる構造物は、コンポーネントと呼ばれていますが、本文の中でも今までJButton,JTextArea,あるいはJTextFieldなどを紹介してきました。他にもいろいろなクラスが用意されていて、これらはすべてJComponentというクラスのサブクラスとなっています。

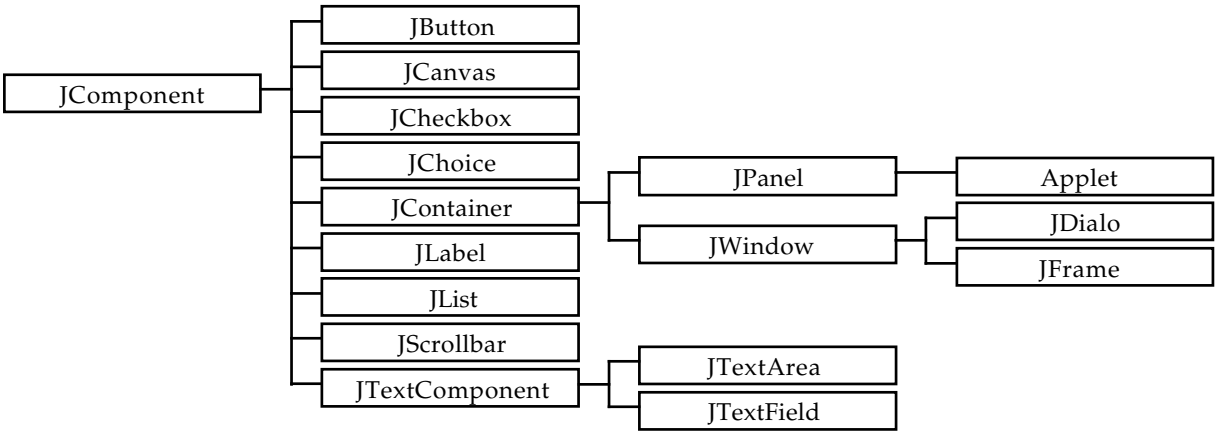


図 22-1 JComponent以下の主要なクラス（一部）

JComponentクラスには、これらのGUIの部品に対しての共通して使われる特性がインスタンスメソッドとして用意されています。当然JComponentクラスのメソッドは、それらのサブクラスのオブジェクトからも呼び出すことができます。上の図にあるように、アプレットもJComponentクラスのサブクラスとなっています。

２２－１－２．JComponentクラスの主要なメソッド

以下のメソッドは、上記のどのクラスでも用いることができます。

▼大きさについて
大きさや位置については、レイアウトマネージャによって、コンポーネントの位置や大きさは自動的に決まってしまいますが、レイアウトマネージャをOFFにすることにより、以下のようなメソッドを使って、明示的に場所や配置を行なうこともできます。以下の記述において、戻り値がない場合は型（**void**）を省略します。

Dimension getSize( )	大きさを返す
setSize(**int** *w*, **int** *h*) ・setSize(Dimension *d*)	大きさを設定する

Dimensionクラスのオブジェクトのインスタンス変数height,widthを用いて、大きさを整数値で得ることができます。コンポーネントは配置されている必要があります。下の例は、ボタンを使っていますがが、他のコンポーネントでも同じようにメソッドを呼び出すことができます。

```
JButton  b = new JButton( "Hurry Up!" );  add( b );
Dimension  bsize = b.getSize( );
System.out.println( "Width: " + bsize.width + "  Height : " + bsize.height );
b.setSize( 100, 40 );
```

▼位置について
位置はコンポーネントを包含する外側のコンポーネント（コンテナと呼ばれています）上の座標となります。

コンポーネントの左上の座標が対象となります。

Point getLocation( )	座標を得る
setLocation(**int** *x*, **int** *y*)・setLocation(Point *p*)	座標を設定する

座標値はPointクラスのオブジェクトのインスタンス変数x, yによって知ることができます。

```
Point blocation = b.getLocation( );
System.out.println( "x axis: " + blocation.x + "  y axis : " + blocation.y );
b.setLocation( 10, 20 );
```

▼位置と座標を同時に
コンポーネントの左上の座標と幅・高さを同時に扱うメソッドです。今までの２つの機能を合わせたものになっています。Rectangleクラスには、それらの情報を示すための、x,y,width,heightというインスタンス変数が定義されています。

Rectangle getBounds( )・setBounds(Rectangle *r*)	座標値・大きさの獲得と設定
setBounds(**int** *x*, **int** *y*, **int** *width*, **int** *height*)	座標値・大きさを４つの値で設定

▼色関係のメソッド
色を指定しない限り、コンポーネントの色は標準の色（グレー）で設定されます。色指定をすることによって、ボタンの色や、テキストフィールドの文字の色などを変えることができます。

Color getBackground( )・setBackground(Color *color*)	背景色の獲得と設定
Color getForeground( )・setForeground(Color *color*)	描画色の獲得とを設定する

たとえば、アプレットでColor *backcolor* = getBackground( ); と記述するとアプレットの背景色を得ることができます。変数*b*で示されるボタンに対して*b*.setBackground(Color.yellow);と記述しますと、ボタン全体の色を変えることができます。あるいは、Color *forecolor* = getForeground( ); でアプレットの描画色をとれますし、*b*.setForeground(Color.red); と指定しますとボタンの文字色が赤色になります。

▼フォント関係のメソッド
フォントについては第８章で説明しました。ボタンやテキストフィールドの字体を変えることができます。

Font getFont( )	コンポーネントで現在用いられているフォントを返す
void setFont(Font *font*)	そのコンポーネントで用いるフォントを設定する

たとえば、変数*ta*で示されるテキストエリアに対して、*ta*.setFont(**new** Font("SansSerif", Font.BOLD, 18)); で入力される文字のフォントを変えることができます。

▼表示・利用制御のためのメソッド
パラメータの値が**true**の場合は、そのコンポーネントが見えていなければ、表示させたり、利用可能にします。通常は、配置された時点は表示・利用可能な状態になっています。**false**の場合は、そのコンポーネントを見せなくしたり、利用不可にします。

void setVisible(**boolean** *visibleSwitch*)	表示するかどうか設定する
boolean isVisible( )	見えているかどうかを得る
void setEnabled(**boolean** enableSwitch)	利用可能にするかどうかを設定する
boolean isEnabled( )	利用可能かどうか返す

そのコンポーネントの状態を得るメソッドでは、設定値と同様に**true**ならば、見えている・利用可能になっています。そのコンポーネントが利用可能であれば、イベントなどを発生させることができます。処理系によっては、利用可能でない場合は、表示の少し暗めに替えるなどの見た目上の措置も講じられます。

```
b.setVisible( true );          // ボタンbが表示されていなければ表示する
b.setEnable( false );          // ボタンbは見えているが、押せないようにする
```

▼描画領域や包含関係を求めるメソッド
この他に、これまでアプレットなどで使ってきたメソッドが用意されています。drawLineなどの描画などで必要なのが、次のようなメソッドです。

```
Graphics getGraphics( )   そのコンポーネント用の描画領域（グラフィックコンテキスト）を返す
JContainer getParent( )   そのコンポーネントが含まれている外側のコンポーネントを返します。
```

ただし、そのコンポーネントが配置され、画面上にないと**null**が返されます。以下の記述は、アプレットにおいて描画領域を得るものです。

```
Graphics g = this.getGraphics( );          // アプレットの描画領域を得る
g.drawLine( 10, 10 , 100, 100 );
```

▼描画関係
以下の２つのメソッドは、アプレットのときに出てきました。

```
paint( Graphics g )        描画用（オーバーライディングされる）
repaint( )                 再描画する
```

２２－１－３．いろいろなコンポーネントと、そのイベントリスナーの書き方

本文中で採り上げたボタン・テキストフィールド、テキスト領域以外にも、様々なコンポーネントが用意されています。それらのコンポーネントのうち、ユーザの操作（マウスクリック、テキスト入力）などを受け取り、それに対応したイベントを発生させるものがあります。そのようなコンポーネントは、発生したイベントを受け取り、対応するメソッドで処理するイベントリスナーを登録することができます（処理する必要がなければ登録しなくても構いません）。イベントリスナーに対応しているコンポーネントとしては、次のようなものがあります。

構造物	内容	利用するインターフェース
JButton	ボタン	ActionListener
JCheckbox	チェックボックス	ItemListener
JChoice	選択メニュー	ItemListener
JList	スクロールリスト	ActionListenerかItemListener
JScrollbar	スクロールバー	AdjustmentListener
JTextArea	テキスト領域	TextListener
JTextField	テキストフィールド	ActionListenerかTextListener

たとえば、アプレット上でJChoiceを用意して、ユーザから何かを選んだときにイベントを発生させ、処理を行なうには、次のように、インターフェースをクラス定義に加え、対応する処理メソッドを用意させます。これは、第９章でボタンのときに、同様の処理を記述するやり方を既に学びました。また、各インターフェースに対応して、用意しなければならない処理メソッドの一覧を以下に挙げておきます。

インターフェース	実装するイベント処理用のメソッド
ActionListener	**public void** actionPerformed(ActionEvent *e*)
ItemListener	**public void** itemStateChanged(ItemEvent *e*)
TextListener	**public void** textValueChanged(TextEvent *e*)
AdjustmentListener	**public void** adjustmentValueChanged(AdjustmentEvent *e*)

上記の例のようにインターフェースをクラス定義に加え、処理用のメソッドを登録すれば、コンポーネントに対してのユーザの操作を受け取ることができます。処理用のメソッドが登録されていないコンポーネントにユーザが操作を加えても何も起こりません（ただし、ボタンを押したりすることやテキストフィールドに文字

を入力することなど、そのコンポーネントに固有の操作は行なえます）。

２２－２．各コンポーネントの詳細

以下に各コンポーネントについて紹介していきます。ただしJCanvas、ScrollPaneやJContainerなどを用いたスマートコンポーネントについては、別の章で紹介します。

２２－２－１．ボタン（JButton）

以前の章でも紹介しました。レイアウトによって、表示形態は異なりますが、ユーザからのマウス入力を得るものです。

▼コンポーネントを作るために次のようなコンストラクタがあります。

JButton( )・JButton(String *label*)　　ラベルなしの・指定されたラベルのボタンを作る

▼ボタンに表示されているラベル（表示文字列）についてのメソッド

String　getJLabel( )・setJLabel(　String　*label*)　　ラベルを獲得・設定

▼イベントリスナーの追加・削除、およびアクションコマンドの獲得と設定のためのメソッド

addActionJListner(ActionListener *listener*)・removeActionJListner(ActionJListner *listener*)

String　getActionCommand(　)・setActionCommand(　String *command*)

▼ボタンを使った例

信号機のように赤、緑、黄色のボタンを交互に表示するアプレットで、潜在的に３つのボタンがあり、押す度に表示されるボタンを変えるものです。

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;

public class SignalButtons extends Applet implements ActionListener {
        JButton  red, green, yellow;

        public void init( ) {
                green = new JButton( "Green" );
                green.setBackground( Color.green );
                green.addActionListener( this );
                add( green );
                yellow = new JButton( "Yellow" );
                yellow.setBackground( Color.yellow );
                yellow.addActionListener( this );
                add( yellow );
                red = new JButton( "Red" );
                red.setBackground( Color.red );
                red.addActionListener( this );
                add( red );
                green.setVisible( false );
                yellow.setVisible( false );
        }
        public void actionPerformed( ActionEvent e ) {
                if ( red.isVisible( ) ) {  red.setVisible( false ); green.setVisible( true );  }
                else if ( green.isVisible( ) ) {  green.setVisible( false ); yellow.setVisible( true );  }
                else {   yellow.setVisible( false ); red.setVisible( true );  }
        }
}
```

２２－２－２．チェックボックス（JCheckbox）

ボタンの一種ですが、チェックマークを付けるか付けないかというユーザの入力を受けるためのものです。

▼コンポーネントを作るために次のようなコンストラクタがあります。

JCheckbox( )・JCheckbox(String *JLabel*)　空の・指定されたラベルのチェックボックスを生成する
JCheckbox(String *JLabel* , **boolean** *state*) 加えて、状態（onなら**true**、offなら**false**）も指定
JCheckbox(String *JLabel* , **boolean** *state* , JCheckboxGroup *group*)
加えて、指定されたチェックボックスグループに所属させる

▼表示や状態をみるためのメソッドとしては、次のようなものがあります。

String getJLabel( )・setJLabel(String *JLabel*)	チェックボックスのラベル
boolean getState( )・setState(**boolean** *state*)	チェックの状態（on=**true**, off=**false**)
JCheckboxGroup getJCheckboxGroup( )	チェックボックスグループを得る
setJCheckboxGroup(JCheckboxGroup *group*)	チェックボックスグループに属させる

▼イベントリスナーを追加・削除するメソッドとしては、次のようなものがあります。

addItemListener(ItemListener *Listener*)・removeItemListener(ItemListener *Listener*)

▼チェックボックスを使った例

枠を描くボックス、内側を塗りつぶすボックス、色を反転させるボックスを用意して、四角形を描かせるようなアプレットを記述しました。ボックスに操作が加えられたときに起動されるitemStateChangedメソッドでは、ただ単にrepaintを呼び出すだけで、paintメソッドの方で各ボックスの状態をgetStateメソッドでチェックしています。

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;

public class CheckTester extends Applet implements ItemListener {
        JCheckbox  frame = new JCheckbox( "JFrame", true ),
                fill = new JCheckbox( "Fill" ),
                reverse= new JCheckbox( "Reverse" );
        public void init( ) {
                frame.addItemListener( this );
                fill.addItemListener( this );
                reverse.addItemListener( this );
                add( frame ); add( fill ); add( reverse );
        }
        public void paint( Graphics g ) {
                super.paint( g );
                if ( fill.getState( ) ) {  g.setColor( (reverse.getState(  )) ? Color.blue : Color.red );
                        g.fillRect( 20, 30, 100, 100 );  }
                if ( frame.getState( ) ) {  g.setColor( (reverse.getState(   )) ? Color.red : Color.blue );
                        g.drawRect( 20, 30, 100, 100 );    }
        }
        public void itemStateChanged( ItemEvent e ) {  repaint( );  }
}
```

２２－２－３．チェックボックスグループ（JCheckboxGroup）

複数のチェックボックスを用意し、どれか１つのチェックボックスだけを選択させる（通称ラジオボタンと呼ばれている）ときに用いるクラスです。

▼コンポーネントを作るために次のようなコンストラクタがあります。

JCheckboxGroup( )　チェックボックスグループを作る

▼表示や状態をみるためのメソッドとしては、次のようなものがあります。なお、getSelectedJCheckboxでは、選ばれているチェックボックスがない場合は、**null**が替わりに返されます。

JCheckbox　getSelectedJCheckbox( )　　選ばれているチェックボックスを返す
setSelectedJCheckbox(JCheckbox *box*)　　指定されたチェックボックスを選ぶ

▼チェックボックスグループを使った例
以下の断片的な記述は、アプレットでチェックボックスグループを用意して、３つのフォント名を選ぶものです。実際にフォントを変えるには、ItemListenerを用いてitemStateChangedメソッドで変更させるようにします。

```
String    fontname [ ] =  {"Seif", "SansSerif", "MonoSpaced" };          // インスタン変数として
JCheckboxGroup          group= new JCheckboxGroup( );
JCheckbox    fontbox [ ] = new  JCheckbox[  fontname.length ];;
for  ( int i = 0;  i< fontname.length ; i++ ) {                          //   initメソッドの中で
        fontbox[ i ] = new JCheckbox(   fontname[ i ] );
        fontbox[ i ].setJCheckboxGroup(  group );   add( fontbox[ i ] ); }
```

２２－２－４．選択メニュー（JChoice）

ポップダウン型のメニューで、ユーザに与えられた選択肢のうちから、一つを選ばせるために使います。

▼コンポーネントを作るために次のようなコンストラクタがあります。
JChoice( )　　空の選択メニューをつくる

▼表示や状態をみるためのメソッドとしては、次のようなものがあります。
String getItem(**int** *index*)　　指定された番号の項目を返す
int getItemCount( )　　メニューに項目がいくつあるか返す
int getSelectedIndex( )・String getSelectedItem( )　　現在選択されている項目の番号や名前
add(String *item*) ・ addItem(String *item*)　　メニューに項目を追加する
insert(String *item* , **int** *index*)　　指定された番号に項目を挿入する
remove(**int** *index*) ・remove(String *item*) ・removeAll( )　　指定された項目を削除する
select(**int** *index*)・ select(String *item*)　　指定された項目を選択する

▼イベントリスナーを追加・削除するメソッドとしては、次のようなものがあります。
addItemListener(ItemListener *Listener*)・removeItemListener(ItemListener *Listener*)

▼選択メニューを使った例
５つの色から、色を選ばせる選択メニューを持ったアプレットの例で、描画される四角形の色がそれに応じて変化します。ItemListenerのitemStateChangedメソッドで変更しています。

```
import  java.awt.*;
import  java.awt.event.*;
import  java.applet.*;
import  javax.swing.*;

public class ColorSelector extends Applet implements ItemListener {
        JChoice  colorselect;          int        selected = 0;
        String    names [ ] = {"red", "green", "blue", "orange", "yellow" }
        Color     colors [ ] = { Color.red, Color.green, Color.blue, Color.orange, Color.yellow };

        public void init( ) {
                colorselect = new   JChoice( );
                for ( int  i=0; i< names.length ; i ++ ) {  colorselect.addItem( names[ i ] ); }
                colorselect.addItemListener( this );   colorselect.select( 0 );
                add( colorselect );
        }
        public void paint( Graphics g ) {
                super.paint( g );
                g.setColor( colors[ selected ] );
                g.fillRect( 20, 30, 100, 100 );
```

```
        }
        public void itemStateChanged( ItemEvent e ) {
            selected = colorselect.getSelectedIndex( );
            repaint( );
        }
    }
```

２２－２－５．ラベル（JLabel）

ユーザにテキスト情報を見せるためにあります。ユーザがテキストを操作することはできません。ただし、文字列の変更などができるので、グラフィックコンテキストに直接描画するdrawStringよりはラベルを使った方がレイアウト上扱いやすいときもあります。

▼コンポーネントを作るために次のようなコンストラクタがあります。

JLabel( )・JLabel(String *message*) 空のラベル・messageで指定されたラベルを作る
JLabel(String *message*, **int** *align*) 加えて、位置揃えをalignで指定する

▼ラベル記憶されている文字列や位置揃えのために次のようなメソッドがあります。

String getText( )・setText(String *message*) ラベル文字列の獲得・指定
int getAlignment( )・ setAlignment(**int** *align*) ラベルの位置揃えの獲得・指定

▼位置揃えのための整数値としては、次のようなものが（標準では左揃え）用意されています。

JLabel.LEFT…左揃え JLabel.CENTER…中央揃え JLabel.RIGHT…右揃え

▼ラベルを使った例
次の断面的な記述は、アプレットのinitメソッドで２つのラベルを追加するためのものです。１つは、コンストラクタですべてを指定していますし、もう１つは生成した後から指定しています。

```
add( new JLabel( "Please input name of a fighter", JLabel.CENTER ) );
JLabel level = new JLabel( ); level.setText( 10 + " liters" ); level.setAlignment( JLabel.RIGHT );
add( level );
```

２２－２－６．リスト（JList）

ユーザに選択メニューと同じく与えられた選択肢の中から、選ばせるものですが、複数の項目を選択させることも可能です。使い方は、選択メニュー（JChoice）と似ています。

▼コンポーネントを作るために次のようなコンストラクタがあります。引数の*multiple*が**true**のときは複数選択が可能になりです。**false**は複数選択不可となります。指定しなければ、複数選択不可になっています。

JList( )・JList(**int** *rows*) リストを作る・行数を指定してリストを作る
JList(**int** *rows*, **boolean** *multiple*) 複数選択可能なリストを作る

▼表示や状態をみるためのメソッドとしては、次のようなものがあります。複数の項目が指定されている場合は、配列として返されます。

String getItem(**int** *index*) 指定された番号の項目を返す
String [ ] getItems( ) リスト上のすべての項目を返す
int getItemCount( ) リストに項目がいくつあるか返す
int getRows( ) リストの行数を返す
int getSelectedIndex( )・**int** [ ] getSelectedIndexes( ) 選択されている項目の番号を返す
String getSelectedItem( ) 選択されている項目の名前を返す
add(String *item*)・ addItem(String *item*) リストの最後に項目を追加する
insert(String *item* , **int** *index*) 指定された項目を挿入する
remove(**int** *index*)・remove(String *item*)・removeAll( ) 指定された項目を削除する
select(**int** *index*)・select(String *item*) 指定された項目を選択する
void deselect(**int** *index*) 指定された番号の項目を選択を外す

▼イベントリスナーの追加・削除のメソッドとしては、次のようなものがあります。ActionListenerとItemListenerのどちらでも用いることができますが、ボタンなど他のコンポーネントがある場合は、アクションコマンドを設定することができませんので、ItemListenerを使った方が良いでしょう。

addItemListener(ItemListener *Listener*) ・ removeItemListener(ItemListener *Listener*)
addActionJListner(ActionListener *Listener*)・removeActionJListner(ActionJListner *Listener*)

▼リストを使った例
次のアプレットはただ単に、ダブルクリックして選択された項目を文字端末に表示するものです。

```
import  java.applet.*;
import  java.awt.*;
import  java.awt.event.*;
import  javax.swing.*;

public  class  ListTester  extends  Applet implements ActionListener {
        String  items [ ] = { "White", "Black", "Green", "Red", "Blue" };
        JList  JList = new  JList( );
        public void init( ) {
                for ( int i = 0 ; i < items.length ; i++ ) {  JList.add( items[ i ] ); }
                JList.addActionListener( this ); add( JList );
        }
        public void actionPerformed(  ActionEvent  e ) {
                System.out.println( JList.getSelectedItem( ) );
        }
}
```

★スクロールバー（JScrollbar）
スクロールバーを表示し、タブの位置によってユーザに数量を決めさせることができます。スクロールバーを配置する場合は、後述するレイアウトではボーダーレイアウトにしておく必要があります。

▼コンポーネントを作るために次のようなコンストラクタがあります。

JScrollbar( )	スクロールバーを作る
JScrollbar(**int** *orientation*)	方向を指定してスクロールバーを作る
JScrollbar(**int** *orientation*, **int** *value*, **int** *visible*, **int** *minimum*, **int** *maximum*)	方向、値、タブの幅、最小値、最大値を指定して作る

▼表示や状態をみるためのメソッドとしては、次のようなものがあります。方向は、縦方向であればJScrollbar.VERTICAL、横方向であればJScrollbar.HORIZONTALという定数が用意されています。ブロック移動量（BlockIncrement）は、タブと矢印の間を空間を押したときの移動量になります。見た目のブロック移動量（VisibleAmount）は、表示されるタブの見た目の移動量（ドット数）になります。

int getBlockIncrement( )・ **int** getMaximum( )・ **int** getMinimum( )
int getOrientation( )・ **int** getValue( )・**int** getVisibleAmount( ) 　　各設定値の獲得
setBlockIncrement(**int** *block*)・setMaximum(**int** *maximum*)
setMinimum(**int** *minimum*)・setOrientation(**int** *orientation*)
setValue(**int** *newValue*)・setVisibleAmount(**int** *newAmount*) 　　各値を設定する
setValues(**int** *value*, **int** *visible*, **int** *min*, **int** *max*) 　　４つの値を設定する

▼イベントリスナーの追加・削除のメソッドとしては、次のようなものがあります。

addAdjustmentListener(AdjustmentListener *Listener*)
removeAdjustmentListener(AdjustmentListener *Listener*)

▼スクロールバーを使った例
ラベルを１つ用意して、0〜100までの間で変わるスクロールバーのタブの位置に併せて、現在値を表示させるアプレットです。

```
import  java.awt.*;
```

```
import  java.awt.event.*;
import  javax.swing.*;

public class JScrollbarTester extends java.applet.Applet implements AdjustmentListener {
        JScrollbar bar= new JScrollbar( JScrollbar.HORIZONTAL, 0, 10, 0, 100 );
        JLabel value = new JLabel( "" + bar.getValue( ) );
        public void init( ) {
                setLayout( new BorderLayout( ) );
                bar.addAdjustmentListener( this );
                add( "South", bar );
                 add( "Center", value );
        }
        public void adjustmentValueChanged( AdjustmentEvent e ) {
                value.setText( "" + bar.getValue( ) );
                repaint( );
        }
}
```

２２－２－７．テキスト領域（JTextArea）

ユーザにテキストを編集させるために使います。テキストフィールドと共通のものは、こちらで扱います。使用例は第１９章を参照してください。

▼コンポーネントを作るために次のようなコンストラクタがあります。スクロールバーの指定は、JTextArea.JScrollbarS_NONE（なし）以外に、JScrollbarS_HORIZONTAL_ONLY（水平方向のみ）、JScrollbarS_VERTICAL_ONLY（垂直方向のみ）、JScrollbarS_BOTH（両方向）と用意されています。

JTextArea( )・JTextArea(String *text*)	空・文字列設定されたものを作る
JTextArea(**int** *rows*, **int** *columns*)	行数・桁数を指定して
JTextArea(String *text*, **int** *rows*, **int** *columns*, **int** *JScrollbars*)	すべてのパラメータを指定して

▼表示や状態をみるためのメソッドとしては、次のようなものがあり、以下のものはJTextFieldと共通です。

String getText( )・setText(String *text*)	テキストの獲得と設定
int getCaretPosition( )・setCaretPosition(**int** *position*)	カーソルの位置（0～）
String getSelectedText( )	選択された範囲のテキストを獲得
int getSelectedStart( )・**int** getSelectedEnd( )	選択された範囲の先頭と最後（0～）
boolean isEditable( )・setEditable(**boolean** *sw*)	編集可能か（**true**なら編集可能）
select(**int** *start*, **int** *end*)・selectAll( )	指定した範囲を選択
Dimension getMinimumSize( )・getPreferredSize( )	最小・適性矩形サイズを得る

▼テキストエリア固有のメソッド

append(String *str*)・insert(String *str*, **int** *pos*)	テキストの追加・挿入
replaceRange(String *str*, **int** *start*, **int** *end*)	テキストの置換
int getColumns( )・**int** getRows( )	桁数・行数を得る
setColoumns(**int** columns)・setRows(**int** rows)	桁数・行数を設定

▼イベントリスナーの追加・削除のメソッドとしては、次のようなものがあります。JTextFieldと共通です。１文字変更される度にリスナーが呼び出されます。

addTextListener(TextListener *Listener*)・removeTextListener(TextListener *Listener*)

２２－２－８．テキストフィールド（JTextField）

テキスト領域と同じくユーザにテキストを編集させるために使いますが、より簡略化され、一行だけ入力させるために用います。以下ではJTextAreaと共通なものは省いています。例は第１９章を参照してください。

▼コンポーネントを作るために次のようなコンストラクタがあります。

JTextField( )・JTextField(String *text*)・JTextField(**int** *columns*)	空・テキスト指定・桁数指定
JTextField(String *text* , **int** *columns*)	テキストと桁数を指定

▼表示や状態をみるための固有のメソッドとしては、次のようなものがあります。エコーキャラクタは、パスワードなどユーザの入力を隠したいときに使われます。

int getColumns( )・setColumns(**int** columns)	桁数の獲得・設定
char getEchoChar( )・setEchoChar(char *echo*)	エコーキャラクタの獲得・設定
boolean echoCharIsSet( )	エコーキャラクタが設定されているか

▼テキストフィールド固有のイベント関係のメソッドとしては、次のようなものがあります。リターンあるいはEnterキーを押したときにActionEventを発生させることができるからです。

addActionJListner(ActionListener *Listener*)・removeActionJListner(ActionJListner *Listener*)

２２－３．レイアウト

２２－３－１．基本的なレイアウト

アプレットやパネルなどのレイアウトを指定することができます。レイアウトはレイアウトマネージャーが行ないます。次のようなレイアウトが用意されています。レイアウトなしを指定しますと、各コンポーネントを座標で指定することができ、幅や高さも変えることができます。また、アプレットの標準では、フローレイアウトの中揃えが設定されています。レイアウトの設定によって、ボタンなどの表示がかなり変わりますので、注意してください。

```
▼レイアウトなし        setLayout( null );
▼フローレイアウト      setLayout( new FlowLayout( ) );                    // 中揃え
                       setLayout( new FlowLayout( FlowLayout.LEFT ) );    // 左揃え
                       setLayout( new FlowLayout( FlowLayout.RIGHT ) ); // 右揃え
▼ボーダーレイアウト    setLayout( new BorderLayout( ) );
```

ボーダーレイアウトでは、コンポーネントを配置するときに、以下のように上下左右（東西南北で記述します）、および中央のどの位置に置くか、指定することができます。

```
add( "North", new JButton( "OK" ) ); add( "South", new JButton( "Back" ) );     // 上下に配置
add( "West", new JButton( "Left" ) ); add( "East", new JButton( "Right" ) );    // 左右に配置
add( "Center", new JButton( "Look" ) );                                         // 中央に配置
```

▼グリッドレイアウト
グリッドレイアウトでは、コンポーネントが付け加えられるたびに、上の行の左側から順次配置されていきます。addメソッドで付け加えられる順番に気をつけてください。

```
setLayout( new GridLayout( 2, 3 ) );                    //2行3列
setLayout( new GridLayout( 2, 3, 10, 15 ) );            //水平間隔10ドット、垂直間隔15ドット
```

２２－３－２．グリッドバッグレイアウト

アプレットのレイアウトをグリッドバッグレイアウトで指定することができる。これは、HTMLのTableタグと使い方は似ています。グリッドバッグレイアウトには、補助クラスとしてグリッドバッグ制約クラスが用いられ、一緒に使います。

```
GridBagLayout gb = new GridBagLayout( );
GridBagConstraints gc = new GridBagConstraints( );
```

ボタンなどを配置する場合は、このGridBagConstraintsクラスのオブジェクトをパラメータに取り、GridBagLayoutクラスのオブジェクトに用意されていますsetConstraintsメソッドを用いて、指定します。

```
JButton button = new JButton( "OK" ); gb.setConstraints( button, gc );
```

▼グリッドバッグ制約メソッドには、次のようなインスタンス変数（メンバー）が用意されています。

int	anchor;	配置の仕方
int	fill;	領域一杯に引き延ばすかどうかの指定
int	gridwidth, gridheight;	どれくらいのセルにまたがるか
int	gridx, gridy;	対象となる部品の行と列の指定
int	ipadx, ipady;	左右、上下に余裕を持たせるときの幅、高さ

・anchorが取り得る値としては、CENTER, EAST, NORTHEAST, NORTH, NORTHWEST, WEST, SOUTHWEST, SOUTH, SOUTHEASTがあります。GridBagConstraints.CENTER（中央に揃える）という感じで指定します。指定しなければCENTERになります。
・fillの値が取り得る値としては、BOTH, HORIZONTAL, VERTICAL, NONEがあります。指定しなければ、NONEになります。
・gridx, gridyは、0から始まることに注意しましょう。

▼グリッドバッグレイアウトを用いた例
たとえば、アプレット上につぎのようなレイアウトをしてみます。縦２行、横３列で、こんな感じです。initメソッドの中のレイアウトをする部分だけを記述します。

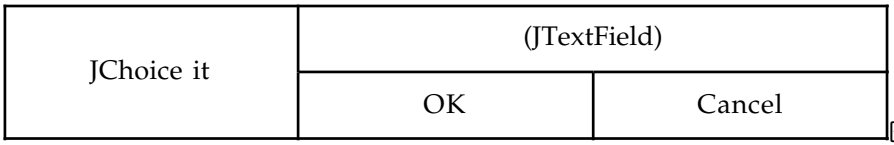
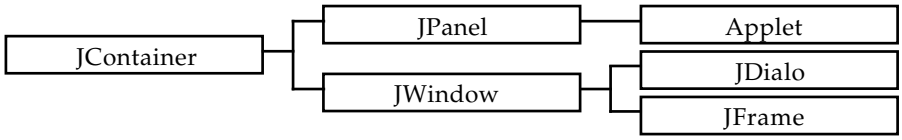


図22-2　例題レイアウト

```
public void init( ) {
        GridBagLayout gb = new GridBagLayout( );
        GridBagConstraints  gc = new GridBagConstraints( );
        gc.fill = GridBagConstraints.BOTH;                          //縦横目一杯に引っ張り
        gc.ipadx = 10;      gc.ipady = 10;                          //縦横に余裕を10ドット
        gc.gridx = 0; gc.gridy = 0;  gc.gridwidth = 1; gc.gridheight = 2;   // JChoice Itボタンの配置
        JButton JChoice = new JButton( "JChoice It" );              //横1列縦２行にまたがる
        gb.setConstraints( JChoice, gc ); add( JChoice );
        gc.gridx = 1; gc.gridy = 0; gc.gridwidth = 2; gc.gridheight = 1;    // JTextFieldの配置
        JTextField field = new JTextField( 20 );                    //横2列縦1行にまたがる
        gb.setConstraints( field, gc ); add( field );
        gc.gridx = 1; gc.gridy = 1; gc.gridwidth = 1; gc.gridheight = 1;    // OKボタンの配置
        JButton ok = new JButton( "OK" ); gb.setConstraints( ok, gc ); add( ok );
        gc.gridx = 2; gc.gridy = 1;                                 // Cancelボタンの配置
        JButton cancel = new JButton( "Cancel" ); gb.setConstraints( cancel, gc ); add( cancel );
}
```

２２－３．ウィンドウのためのクラス

JComponentクラスの下に、他のコンポーネントを包含することができる以下のJContainerクラス群が用意されています。これらを使ってウィンドウプログラミングを行なうことができます。Appletクラスも、この一員です。



図A-11　JContainerクラス以下の主要なクラス

２２－３－１．JPanelクラス

★アプレットなどでサブパネルを利用する

アプレットは、１つのパネルになっていますが、見栄えをよくするためにサブパネルを生成することがありま

す。パネルは、アプレットの一部の領域を分割し、それ自体は通常のアプレットと同じように使うことができます。パネルの中では、レイアウトなどを変えることができるので便利です。以下にそのような記述の例を示しました。ただし、描画領域（グラフィックスコンテキスト）に直接描画するような場合は、スマートコンポーネントにする必要があります。

```
JPanel    mypanel = new JPanel( );                          // パネルを生成
mypanel.setLayout( new GridLayout( 3, 4, 10, 15 ) );        // パネルの中だけグリッドレイアウトに
mypanel.add( new JButton( "WOW" ) );                        // パネルの中にボタンを配置
add( mypanel );                                             // パネルをアプレット上に配置
```

▼JPanelクラスを使った例
アプレットのinitメソッドで配置を行なう部分だけを記述してみます。アプレットの中は、グリッドレイアウトにして、パネルを３枚配置します。パネルの中は、フローレイアウトで、いろいろなコンポーネントを配置していきます。

```
public void init( ) {
        JPanel    myJPanel [ ] = new JPanel[ 3 ];
        setLayout( new GridLayout( myJPanel.length, 1 ) );
        for ( int i=0; i<myJPanel.length ; i++) {
                myJPanel[ i ] = new JPanel( );  add( myJPanel[ i ] ); }
        myJPanel[ 0 ].add( new JCheckbox( "Green" ) );
        myJPanel[ 0 ].add( new JCheckbox( "Red" ) );
        myJPanel[ 0 ].add( new JCheckbox( "Blue" ) );
        myJPanel[ 1 ].add( new JTextField( 60 ) );
        myJPanel[ 2 ].add( new JButton( "OK" ) );
        myJPanel[ 2 ].add( new JButton( "Cancel" ) );
}
```

★カードレイアウト

さまざまなコンポーネントから構成されるアプレット上などで次々と画面を切り替えたい場合は、パネルクラスを併用して、それぞれをカードとして表示します。たとえば、次のように記述しますと２つのパネルを生成することができます。

```
CardLayout  card = new CardLayout( )
setLayout( card );
JPanel    p1 = new JPanel( );  add( "summer", p1 );
JPanel    p2 = new JPanel( );  add( "winter", p2 );
```

このパネルを切り替えるためには、showメソッドを利用します。

```
card.show( this, "summer" );          // summerパネルの方を見せる
card.show( this, "winter" );          // winterパネルの方を見せる
```

▼カードレイアウトを使った例
カードレイアウトを使ったサンプルのアプレットを記述してみました。それぞれのパネルには、ボタンだけしかありませんが、ボタンが押される度にパネルが切り替わっていきます。

```
import  java.awt.*;
import  java.awt.event.*;
import  java.applet.* ;
import  javax.swing.*;

public class SceneViwer extends Applet implements ActionListener {
        String    course [ ] = { "enterance", "room", "dining", "study", "bedroom", "exit" };
        JPanel    scene [ ] = new JPanel[ course.length ];
        CardLayout        scenario = new CardLayout( );
        int  cur = 0;

        public void init( ) {
                setLayout( scenario );
```

```
                for ( int  i=0; i < course.length ; i++ ) {
                        JButton  b = new  JButton( course[ i ] );
                        b.addActionListener( this );
                        scene[ i ] = new  JPanel(  );  scene[ i ].add( JButton  );
                        add(  course[ i ],  scene[ i ]  );
                }
        }
        public void paint( Graphics g ) {
                super.paint( g );
                 scenario.show( this, course[ cur ] );
        }
        public void actionPerformed( ActionEvent   ae ) {
                cur = (cur+1) % course.length;
                repaint( );
        }
}
```

２２－３－２．JWindowクラスのサブクラスとメニュー

ウィンドウ（JFrame）は、第９章で取り上げました。スマートコンポーネントのものは、次の付録で説明します。なお、ウィンドウやダイアログは、標準設定ではボーダーレイアウトになっているので注意してください。ここでは、ダイアログとユーザインターフェースとして良く使われるメニューについて取り上げます。

★ダイアログ（JDialog）
ダイアログは、ウィンドウ（JFrame）とほとんど同じですが、モーダルダイアログ（ModalJDialog）にすることができるのが大きな違いです。モーダル指定をすると、入力が終了するまで（setVisibleでダイアログを隠すまで）、別のウィンドウに入力が移るのを禁止します。次のアプレットは、アプレット上のボタンが押されたら、ダイアログ上にボタンとテキストフィールドを出し、入力されたテキストを、アプレット上のテキストフィールドに入れるものです。ダイアログのコンストラクタの１番目のパラメータとして、ダミーのウィンドウを作っています。また、３番目の値が**true**になっているのがモーダルダイアログの指定です。

```
import   java.applet.*;
import   java.awt.*;
import   java.awt.event.*;
import   javax.swing.*;

public  class  JDialogTester  extends  Applet implements ActionListener {
        JDialog  dialog = new JDialog( new JFrame( ),  "Enter Text", true);
        JButton  change = new JButton( "Change" ),  ok = new JButton( "OK" );
        JTextField  present = new JTextField( "None", 20 ), input = new JTextField( 30 );

        public void init( ) {
                change.setActionCommand( "Change" );
                change.addActionListener( this );
                ok.addActionListener( this );
                input.addActionListener( this );
                present.setEditable( false );
                add( present );
                add( change );
                dialog.setLayout( new FlowLayout( ) );
                dialog.add( input ); dialog.add( ok );
        }
        public void actionPerformed( ActionEvent e ) {
                if ( e.getActionCommand( ).equals( "Change" ) ) { dialog.show( ); }
                else { JDialog.setVisible( false ); present.setText( input.getText( ) ); repaint( ); }
        }
}
```

★メニュー（Menu）とメニューバー（MenuBar）

ウィンドウ（JFrame）に対しては、メニューバーを（setMenuBarメソッドで）設定することができます。メ

ニューバーには、複数のメニューを設定することができます。１つのメニューには、いくつかのメニュー項目（MenuItem）を登録しておきます。メニューの対処はActionListener（actionPerformedメソッド）で行ないます。メニュー項目には、アクションコマンドを登録しておけますので、それを見て、どのメニュー項目が選ばれたのか判断できます。メニューのメソッドは、JChoiceクラスとだいたい同じになっています。次のアプレットは、メニューバー表示用のサブウィンドウを出して、メニューを選ばせます。選ばれたメニュー項目の文字列を、アプレット上のテキストフィールドに反映させます。

```
import  java.applet.*;
import  java.awt.*;
import  java.awt.event.*;
import  javax.swing.*;

public  class  MenuBarTester  extends  Applet implements ActionListener {
        String  command [ ] = { "Start", "Stop", "Continue", "Reset" };
        MenuBar    mb = new   MenuBar( );
        Menu    menu = new   Menu( "File" );
        JTextField tf = new JTextField( 30 );
        JFrame f = new JFrame( "MenuBarTester" );

        public void init( ) {
                for ( int i = 0; i < command.length; i++ ) {
                        MenuItem  mitem = new MenuItem( command[ i ] );
                        mitem.addActionListener( this );
                        mitem.setActionCommand( command[ i ] );
                        menu.add( mitem );
                }
                mb.add( menu );
                f.setMenuBar( mb );
                f.setSize( 100, 50 );
                f.show( );
                add( tf );
        }
        public void actionPerformed( ActionEvent    e ) {
                tf.setText( e.getActionCommand( ) );
                repaint( );
        }
}
```

２２－３－４．Swing版のJAppletを使う