Swing 実習マニュアル

第三版

大黒学

Swing 実習マニュアル・第三版
著者——大黒学

2002 年　5 月　2 日（木）　Java 実習マニュアル・第零版発行
2005 年　1 月 28 日（金）　Swing 実習マニュアル・第一版発行
2005 年　3 月　4 日（金）　第二版発行
2006 年　3 月　6 日（月）　第三版発行

目次

第 1 章　Swing の基礎

1.1　Swing とは何か

1.1.1　ユーザーインターフェース

機械などが持っている、人間に何かを報告したり人間からの指示を受け付けたりする部分のことを、「ユーザーインターフェース」(user interface) と言います。

コンピュータのプログラムも、たいていのものは何らかのユーザーインターフェースを持っています。プログラムのユーザーインターフェースとしては、現在、CUI と GUI と呼ばれる 2 種類のものが主流です。

CUI(character user interface) というのは、文字列を媒介とするユーザーインターフェースのことです。CUI の場合、プログラムは文字列を出力することによって人間に何かを報告し、人間はキーボードから文字列を入力することによってプログラムに指示を与えます。

それに対して、GUI(graphical user interface) は、図形を媒介とするユーザーインターフェースです。GUI の場合、プログラムはモニターの画面に図形を表示することによって人間に何かを報告します。そして人間は、「ポインティングデバイス」(pointing device) と呼ばれる装置（たとえばマウスなど）を使ってその図形を操作することによってプログラムに指示を与えます。

1.1.2　GUI のライブラリー

それでは、GUI を持つプログラムを書くためには、いったいどうすればいいのでしょうか。

オペレーティングシステムの多くは、GUI を作るための基本的な機能を持っています。ですから、オペレーティングシステムがアプリケーションプログラムに対して公開している、API(application programming interface) と呼ばれる関数群を使うことによって、GUI を持つプログラムを書くことができます。

API というのはオペレーティングシステムごとに異なりますので、それを生のまま使ってプログラムを書くと、そのプログラムは、ほかのオペレーティングシステムに移植することが困難なものになってしまいます。移植性の高いプログラムを書くためには、生の API ではなくて、オペレーティングシステムから独立したライブラリーを使う必要があります。

GUI を作るためのライブラリーで、オペレーティングシステムから独立しているものの一例としては、John Ousterhout さんという人が開発した、Tk と呼ばれるライブラリーがあります（Tk という名前は、toolkit という単語を縮めたものです）。

1.1.3　AWT と Swing

Java というプログラミング言語では、ライブラリーに相当するもののことを「パッケージ」(package) と呼びます。Java の開発環境はさまざまなパッケージを含んでいるわけですが、その中には、GUI を作るためのパッケージが二つあります。ひとつは AWT、もうひとつは Swing と呼ばれるパッケージです。

AWT(Abstract Window Toolkit) は、Java の歴史の上で、かなり初期のころから提供されているパッケージです。それに対して、Swing は、比較的最近になって提供されるようになったパッケージです。

Swing は、AWT から完全に独立したパッケージではなくて、AWT の機能を利用することによって実現されています。また、Swing を使って GUI を持つプログラムを書く場合には、多くの場合、Swing のクラスだけではなくて、AWT のクラスも使う必要があります。

1.1.4　この文章について

この「Swing 実習マニュアル」という文章は、Swing を使って GUI を持つプログラムを書く方法について解説したものです。前述したように、Swing のクラスだけでは GUI を作ることができませんので、必要に応じて AWT のクラスについても解説しています。

Swing や AWT はあくまで Java のパッケージですから、Java が使えなければ、それらのパッケージを使ってプログラムを書くことはできません。この文章は、読者はすでに Java が使えるようになっていると仮定して書かれています。この文章を読んでいて、Java に関してわからない点があった場合は、Java に関する書籍などを参照してください。

1.1.5 パッケージ名とクラス名

Java のプログラムの中では、パッケージに含まれるクラスは、パッケージ名とクラス名とを組み合わせた名前によって指定されます。Swing と AWT は、それぞれ、

AWT `java.awt`

Swing `javax.swing`

というパッケージ名を持っていますので、たとえば、色をあらわす AWT のクラスは、

```
java.awt.Color
```

という名前によって指定されます。

GUI を持つプログラムを書く場合は、Swing や AWT のクラス名が頻出しますので、プログラムの先頭に `import` 文を書くことによって、Swing や AWT のパッケージ名を省略できるようにするといいでしょう。

Swing に含まれているクラスのうちには、名前の先頭に大文字の `J` が付いているものがたくさんあります（たとえば `JFrame` や `JButton` など）。その `J` は、同じ用途で使われるクラスが AWT にも含まれている場合に、そのクラスと区別するために付けられたものです。

1.2 フレーム

1.2.1 ウィンドウ

画面の上に表示される、独立した長方形の領域のことを、「ウィンドウ」(window) と呼びます。

ウィンドウは、大きく 2 種類に分類することができます。ひとつは、プログラムが動作しているあいだは常に存在しているウィンドウで、もうひとつは、必要に応じて一時的に表示されるウィンドウです。前者は「フレーム」(frame) と呼ばれ、後者は「ダイアログボックス」(dialog box) と呼ばれます。

この節では、GUI を持つプログラムを書くための第一歩として、フレームを表示するプログラムの書き方について説明したいと思います。

1.2.2 コンポーネント

Java では、GUI を作るための部品のことを「コンポーネント」(component) と呼びます。ウィンドウも、コンポーネントの一種です。

コンポーネントは、何らかのオブジェクトによって表示されます。コンポーネントを表示するオブジェクトは、そのコンポーネントの種類に対応するクラスから生成されます。たとえば、フレームを表示するオブジェクトは、フレームのクラスから生成されます。コンポーネントにはさまざまな種類がありますので、Swing や AWT というパッケージの中には、それぞれのコンポーネントに対応するさまざまなクラスが含まれています。

この文章では、コンポーネントの種類をあらわす言葉を、画面の上に表示されるものを指示するために使うだけではなくて、画面の上にコンポーネントを表示するオブジェクトを指示するためにも使います。ですから、たとえば「フレーム」という言葉も、画面の上に表示されるフレームという意味で使われる場合と、画面の上にフレームを表示するオブジェクトという意味で使われる場合とがありますので、注意してください。

1.2.3 フレームの生成と初期設定

フレームというコンポーネントを生成する Swing のクラスは、`JFrame` という名前です。ですから、

```
JFrame frame = new JFrame();
```

という宣言文を書けば、フレームが生成されて、それを指し示す参照が `frame` という変数に設定されます。

多くの場合、コンポーネントを使うためには、オブジェクトを生成するだけではなくて、そのオブジェクトに対して何らかの初期設定をする必要があります。フレームを使う場合は、

- フレームを閉じる操作に対応する動作
- フレームの大きさ
- フレームの可視状態（見えるかどうかという状態）

定数	動作
JFrame.DO_NOTHING_ON_CLOSE	何もしない。
JFrame.HIDE_ON_CLOSE	フレームを隠す（デフォルト）。
JFrame.DISPOSE_ON_CLOSE	フレームを破棄する。
JFrame.EXIT_ON_CLOSE	プログラムを終了させる。

表 1.1: フレームを閉じる操作に対応する動作をあらわす定数

- フレームのタイトル

というような属性に対する初期設定が必要になります。

1.2.4 フレームを閉じる操作に対応する動作

GUI を持つプログラムを終了させたいときは、普通、そのプログラムが表示しているフレームのうちでもっとも基本となっているものを閉じる、という操作をします。ですから、そのようなフレームは、それを閉じる操作に対して、プログラムを終了させるという動作が設定されている必要があります。それに対して、基本となっているもの以外のフレームについては、それを閉じる操作をしてもプログラムが終了してしまわないように設定しておかないといけません。ちなみに、生成された直後フレームには、それを閉じる操作に対応する動作として、そのフレームを隠すという動作が設定されています。

フレームを閉じる操作に対応する動作は、 setDefaultCloseOperation という長い名前のメソッドを呼び出すことによって変更することができます。このメソッドは、動作をあらわす整数を引数として受け取ります。動作をあらわす整数は、表 1.1 に示されている定数を使って指定します。たとえば、

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

という式文を書くことによって、 frame という変数が指し示しているフレームに設定されている、それを閉じる操作に対応する動作を、プログラムを終了させるという動作に変更することができます。

1.2.5 フレームの大きさ

フレームの大きさは、 setSize というメソッドを呼び出すことによって変更することができます。このメソッドは、引数として 2 個の整数を受け取ります。1 個目はフレームの横の長さで、2 個目は縦の長さです（単位はピクセル）。たとえば、

```
frame.setSize(400, 300);
```

という式文を書くことによって、 frame という変数が指し示しているフレームの大きさを、横 400 ピクセル、縦 300 ピクセルに変更することができます。

1.2.6 フレームの可視状態

コンポーネントは、可視か不可視か、つまり見えるか見えないかというどちらかの状態を持っています。ちなみに、フレームは、新しく生成された直後は不可視の状態になっています。

コンポーネントの可視状態は、 setVisible というメソッドを呼び出すことによって変更することができます。このメソッドは、可視状態をあらわす真偽値を引数として受け取ります。 true が可視、 false が不可視をあらわします。たとえば、

```
frame.setVisible(true);
```

という式文を書くことによって、 frame という変数が指し示しているコンポーネントの可視状態を可視に変更することができます。

それでは、Swing を使ったプログラムの最初の例として、ただ単にフレームを表示するだけのプログラムを書いてみましょう。

プログラムの例 PFrame.java

```
import javax.swing.*;

public class PFrame {
```

```
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
        frame.setVisible(true);
    }
}
```

1.2.7 フレームのタイトル

フレームやダイアログボックスは、「タイトルバー」(title bar) と呼ばれる領域を持っていて、その領域に文字列を表示することができます。タイトルバーに表示される文字列は、そのフレームやダイアログボックスの「タイトル」(title) と呼ばれます。

JFrame クラスが持っているコンストラクタとしては、引数を受け取らないもののほかに、1 個の文字列を受け取るものもあります。そのコンストラクタは、受け取った文字列をタイトルとして設定します。

次のプログラムは、「私はフレームです。」という文字列がタイトルとして設定されたフレームを表示します。

プログラムの例 PTitle.java

```
import javax.swing.*;

public class PTitle {
    public static void main(String[] args) {
        JFrame frame = new JFrame("私はフレームです。");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
        frame.setVisible(true);
    }
}
```

フレームやダイアログボックスのタイトルは、プログラムの実行中に変更することも可能です。タイトルを変更したいときは、setTitle というメソッドを呼び出します。このメソッドは、文字列を引数として受け取って、タイトルをその文字列に変更します。たとえば、

```
frame.setTitle("新しいタイトル");
```

という式文を書くことによって、frame という変数が指し示しているフレームのタイトルを「新しいタイトル」に変更することができます。

1.2.8 サブクラスのコンストラクタでの初期設定

生成したコンポーネントを初期設定するためにいくつかのメソッドを呼び出す必要がある場合は、オリジナルなクラスのサブクラスを作って、そのサブクラスのコンストラクタの中でそれらのメソッドを呼び出すようにすると、プログラムの見通しがよくなります。

次のプログラムは、JFrame のサブクラスを作って、そのサブクラスのコンストラクタの中で、フレームを初期設定するためのメソッドを呼び出しています。

プログラムの例 PSubclass.java

```
import javax.swing.*;

public class PSubclass extends JFrame {
    PSubclass(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public static void main(String[] args) {
        PSubclass frame = new PSubclass("サブクラス");
    }
}
```

定数	タイプ
JOptionPane.ERROR_MESSAGE	エラー
JOptionPane.INFORMATION_MESSAGE	通知
JOptionPane.WARNING_MESSAGE	警告
JOptionPane.QUESTION_MESSAGE	質問
JOptionPane.PLAIN_MESSAGE	指定なし

表 1.2: メッセージのタイプを指定する定数

1.3 定型的なダイアログボックス

1.3.1 メッセージのダイアログボックス

JOptionPane というクラスは、メッセージを表示したり単純な質問をしたりする定型的なダイアログボックスを表示するクラスメソッドを持っています。

たとえば、JOptionPane.showMessageDialog というクラスメソッドは、メッセージのダイアログボックスを表示します。

JOptionPane.showMessageDialog は、

(1) 親フレームまたは null
(2) 表示するオブジェクト（文字列、文字列の配列など）
(3) タイトル（文字列）
(4) メッセージのタイプをあらわす整数（定数で指定する）

という 4 個の引数を受け取ります。ただし、このメソッドには、4 個の引数を受け取るもののほかに、名前が同じで引数の個数が異なるいくつかの変種があります。JOptionPane のほかのクラスメソッドも、同じようにいくつかの変種を持っています。

1 個目の引数としてフレームを渡した場合、ダイアログボックスは、そのフレームの中央に表示されます。フレームではなくて null を渡した場合は、画面の中央に表示されます。

2 個目の引数は、表示したいメッセージです。文字列の配列を渡した場合は、それぞれの文字列を縦に並べたものが表示されます。

3 個目の引数は、ダイアログボックスのタイトルバーに表示される文字列です。

4 個目の引数は、メッセージのタイプをあらわす整数です。この整数は、表 1.2 に示される定数を書くことによって指定することができます。ダイアログボックスに表示されるアイコンは、この引数で指定されたメッセージのタイプによって決定されます。

プログラムの例 PMessage.java

```
import javax.swing.*;

public class PMessage extends JFrame {
    PMessage(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public static void main(String[] args) {
        PMessage frame = new PMessage(
            "メッセージのダイアログボックス");
        JOptionPane.showMessageDialog(frame,
            "私はメッセージです。", "メッセージ",
            JOptionPane.INFORMATION_MESSAGE);
    }
}
```

1.3.2 質問のダイアログボックス

定数	選択肢のタイプ
JOptionPane.YES_NO_OPTION	はい、いいえ
JOptionPane.YES_NO_CANCEL_OPTION	はい、いいえ、取消し
JOptionPane.OK_CANCEL_OPTION	了解、取消し

表 1.3: 選択肢のタイプを指定する定数

定数	選択肢
JOptionPane.YES_OPTION	はい
JOptionPane.NO_OPTION	いいえ
JOptionPane.OK_OPTION	了解
JOptionPane.CANCEL_OPTION	取消し

表 1.4: 選択肢を指定する定数

JOptionPane.showConfirmDialog というクラスメソッドを呼び出すことによって、質問のダイアログボックスを表示することができます。このダイアログボックスには、選択肢をあらわす何個かのボタンが表示されていて、そのいずれかをクリックすることによって質問に答えることができます。

ここでは、JOptionPane.showConfirmDialog のいくつかの変種のうちで、4 個の引数を受け取るものを紹介します。それらの引数は、最初の 3 個はメッセージのダイアログボックスと同じですが、4 個目の引数だけが少し違います。

4 個目の引数は、選択肢のタイプをあらわす整数です。この整数は、表 1.3 に示される定数を書くことによって指定することができます。ダイアログボックスに表示されるボタンは、この引数で指定されたタイプによって決定されます。

JOptionPane.showConfirmDialog は、どの選択肢が選択されたかということをあらわす整数を戻り値として返します。この整数についても、表 1.4 に示されるような定数が定義されています。

プログラムの例 PConfirm.java

```
import javax.swing.*;

public class PConfirm extends JFrame {
    PConfirm(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public static void main(String[] args) {
        PConfirm frame = new PConfirm(
            "質問のダイアログボックス");
        int answer = JOptionPane.showConfirmDialog(frame,
            "あなたはプログラミングが好きですか？", "質問",
            JOptionPane.YES_NO_OPTION);
        if (answer == JOptionPane.YES_OPTION)
            JOptionPane.showMessageDialog(frame,
                "それは素晴らしい。", "判定",
                JOptionPane.INFORMATION_MESSAGE);
        else
            JOptionPane.showMessageDialog(frame,
                "それは残念ですね。", "判定",
                JOptionPane.INFORMATION_MESSAGE);
    }
}
```

1.3.3　入力のダイアログボックス

JOptionPane.showInputDialog というクラスメソッドを呼び出すことによって、文字列を入力してもらうためのダイアログボックスを表示することができます。

JOptionPane.showInputDialog に渡す引数は、メッセージのダイアログボックスを表示するクラスメソッドに渡す引数と同じです。

JOptionPane.showInputDialog は、入力された文字列を戻り値として返します。ただし、「取消し」のボタンがクリックされた場合は、戻り値として null を返します。

次のプログラムは、フレームのタイトルを、ダイアログボックスに入力された文字列に変更します。

プログラムの例　PInput.java

```
import javax.swing.*;

public class PInput extends JFrame {
    PInput(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public static void main(String[] args) {
        PInput frame = new PInput(
            "入力のダイアログボックス");
        String title = JOptionPane.showInputDialog(frame,
            "タイトルを入力してください。", "入力",
            JOptionPane.PLAIN_MESSAGE);
        if (title != null)
            frame.setTitle(title);
    }
}
```

第 2 章　グラフィックス

2.1　グラフィックスの基礎

2.1.1　座標系

コンポーネントの上には、図形や文字列などのグラフィックスを描画することができます。そこで、この章では、コンポーネントの上にグラフィックスを描画する方法について説明したいと思います。

グラフィックスを描画するためには、座標系にもとづいて、その位置を指定する必要があります。コンポーネントの上の位置を指定するための座標系は、コンポーネントの左上の隅が原点 $(0, 0)$ になっていて、x 軸は右向き、y 軸は下向きになっています。そして、長さの単位としてはピクセルが使われます。ですから、$(200, 150)$ という座標は、コンポーネントの左上の隅を出発点として、右へ 200 ピクセル、下へ 150 ピクセルだけ移動した位置をあらわしていることになります。

2.1.2　再描画

コンポーネントの上に描画されたグラフィックスは、いつまでも存在しつづけるとは限りません。グラフィックスは、さまざまな原因によって消去されてしまうことがあります。グラフィックスが消去される原因としては、グラフィックスの上に重なっていたウィンドウが移動するとか、ウィンドウの大きさが変更される、というようなものがあります。

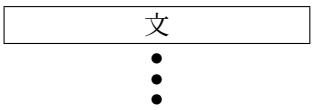
グラフィックスをコンポーネントの上に存在させ続けるためには、それが消去されるたびに同じものを描画し直す必要があります。消去されてしまったグラフィックスを描画し直すことを、グラフィックスを「再描画する」(repaint) と言います。

コンポーネントは、paint というメソッドを持っています。このメソッドは、コンポーネントの上のグラフィックスが消去されたときに自動的に呼び出されます。ですから、このメソッドを

サブクラスでオーバーライドして、その中でグラフィックスを描画すると、そのグラフィックスは、消去されるたびに自動的に再描画されることになります。

paint の定義は、

```
public void paint(Graphics g) {
  super.paint(g);
```




```
}
```

と書きます。このように、paint は、Graphics というクラスのオブジェクトを引数として受け取ります。なお、Graphics は Swing ではなくて AWT のクラスで、java.awt というパッケージの中にあります。

2.1.3 内部が塗りつぶされた楕円

Graphics というクラスのオブジェクトは、グラフィックスを描画するためのさまざまなフィールドとメソッドから構成されています。たとえば、このクラスのオブジェクトは、fillOval という、内部が塗りつぶされた楕円をコンポーネントの上に描画するメソッドを持っています。

fillOval は、楕円に外接する長方形の位置と大きさをあらわす 4 個の整数を引数として受け取ります。それらの引数は、1 個目と 2 個目が長方形の左上の頂点の x 座標と y 座標、3 個目が長方形の横幅、4 個目が長方形の高さです。たとえば、g という変数が Graphics クラスのオブジェクトを指し示しているとするとき、

```
g.fillOval(100, 80, 200, 140);
```

という式文を実行すると、$(100, 80)$ という位置に左上の頂点がある、横幅が 200 ピクセル、高さが 140 ピクセルの長方形に内接する楕円が描画されます。

プログラムの例 PFillOval.java

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

public class PFillOval extends JFrame {
    PFillOval(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.fillOval(100, 80, 200, 140);
    }

    public static void main(String[] args) {
        PFillOval frame = new PFillOval(
            "内部が塗りつぶされた楕円");
    }
}
```

2.1.4 コンポーネントの大きさの取得

コンポーネントの大きさは、getWidth と getHeight というメソッドを呼び出すことによって調べることができます。getWidth はコンポーネントの横の長さを戻り値として返し、getHeight は縦の長さを返します。

次のプログラムは、フレームをどんな大きさに変更しても、常にフレームの中央に円を描画します。

プログラムの例 PSize.java

```
import javax.swing.*;
```

```
import java.awt.*;

public class PSize extends JFrame {
    final static int RADIUS = 50;

    PSize(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.fillOval(getWidth()/2 - RADIUS,
                   getHeight()/2 - RADIUS,
                   RADIUS*2, RADIUS*2);
    }

    public static void main(String[] args) {
        PSize frame = new PSize("円は常にフレームの中央");
    }
}
```

2.1.5　インセット

フレームというコンポーネントは、グラフィックスを描画することのできる領域の外側に、縁取りやタイトルバーなどの、グラフィックスを描画することのできない領域を持っています。フレームの境界線と、描画可能な領域の境界線との間隔は、「インセット」(inset) と呼ばれます。

インセットは、getInsets というメソッドを呼び出すことによって求めることができます。このメソッドは、Insets というクラスのオブジェクトを戻り値として返します。このクラスのオブジェクトは、top、bottom、left、right という 4 個のフィールドを持っていて、それぞれのフィールドには、上、下、左、右のインセットが格納されます。

次のプログラムは、フレームの原点からインセットだけ右下へ移動した位置を左上の頂点とする長方形に内接する円を描画します。

プログラムの例　PInsets.java

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

public class PInsets extends JFrame {
    PInsets(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public void paint(Graphics g) {
        super.paint(g);
        Insets insets = getInsets();
        g.fillOval(insets.left, insets.top, 150, 150);
    }

    public static void main(String[] args) {
        PInsets frame = new PInsets("左上のインセット");
    }
}
```

2.2　色

2.2.1　色を変更するメソッド

Graphics クラスのオブジェクトは、色をあらわすオブジェクトをひとつ保持していて、グラフィックスを描画するときは、そのオブジェクトがあらわしている色を使います。色をあらわすオブジェクトというのは、Color という AWT のクラスから生成されるオブジェクトです。

Graphics クラスのオブジェクトが保持している、色をあらわすオブジェクトを、別の色のオブジェクトに変更すると、それ以降は、変更後の色でグラフィックスが描画されます。色のオブジェクトを変更したいときは、Graphics クラスのオブジェクトが持っている setColor というメソッドを呼び出します。このメソッドは、Color クラスのオブジェクトを引数として受け取って、それをレシーバーに設定します。

2.2.2 Color クラスのフィールドによる色の指定

Color クラスのオブジェクトは、new を使って生成してもかまわないのですが、Color クラスのいくつかのフィールドには、あらかじめ生成されているオブジェクトが設定されています。色をあらわすオブジェクトが設定されているのは、

```
black   cyan      gray    lightGray   orange   red      yellow
blue    darkGray  green   magenta     pink     white
```

という 13 個のフィールドで、それぞれのフィールドの名前は、設定されている色を示しています。ですから、

```
g.setColor(Color.orange);
```

という式文を実行することによって、g が指し示している Graphics クラスのオブジェクトに対して、オレンジ色を設定することができます。

プログラムの例 POrange.java

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

public class POrange extends JFrame {
    POrange(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(Color.orange);
        g.fillOval(100, 80, 200, 150);
    }

    public static void main(String[] args) {
        POrange frame = new POrange("オレンジ色の楕円");
    }
}
```

2.2.3 光の三原色による色の指定

Color クラスのフィールドに設定されているもの以外の色を使いたいときは、Color クラスのオブジェクトを生成する必要があります。

Color クラスはさまざまなコンストラクタを持っていますが、ここでは、3 個の整数を受け取るコンストラクタを紹介しましょう。このコンストラクタに渡すのは、赤、緑、青、という光の三原色の明るさをあらわす整数です。原色の明るさは、0 から 255 までの整数で指定します。たとえば、

```
new Color(0, 0, 255)
```

という式で生成されたオブジェクトがあらわしている色は、赤が 0、緑が 0、青が 255 ですから、青色ということになります。

プログラムの例 PRGB.java

メソッド名	説明
drawLine	直線を描画します。
drawRect	長方形の輪郭を描画します。
fillRect	長方形を塗りつぶします。
drawRoundRect	角が丸い長方形の輪郭を描画します。
fillRoundRect	角が丸い長方形を塗りつぶします。
drawOval	楕円の輪郭を描画します。
fillOval	楕円を塗りつぶします。
drawArc	楕円弧の輪郭を描画します。
fillArc	楕円弧の内部の扇形を塗りつぶします。

表 2.1: 図形を描画する主要なメソッド

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

public class PRGB extends JFrame {
    PRGB(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(new Color(0, 128, 255)); // 空色
        g.fillOval(30, 50, 200, 150);
        g.setColor(new Color(128, 255, 0)); // 黄緑
        g.fillOval(90, 80, 200, 150);
        g.setColor(new Color(192, 0, 255)); // 紫
        g.fillOval(150, 110, 200, 150);
    }

    public static void main(String[] args) {
        PRGB frame = new PRGB("空色と黄緑と紫");
    }
}
```

2.3　図形の描画

2.3.1　図形を描画するメソッド

Graphics クラスのオブジェクトは、図形を描画するメソッドとして、すでに紹介した fillOval だけではなくて、表 2.1 に示されているようなさまざまなメソッドを持っています。この節では、これらのメソッドの使い方について説明していくことにします。

2.3.2　直線

drawLine というメソッドを呼び出すことによって、直線を描画することができます。

drawLine には、4 個の整数を引数として渡します。1 個目と 2 個目は、直線の一方の端の x 座標と y 座標で、3 個目と 4 個目は、もう一方の端の x 座標と y 座標です。たとえば、

```
g.drawLine(100, 200, 300, 150);
```

という式で drawLine を呼び出したとすると、$(100, 200)$ と $(300, 150)$ とをつなぐ直線が描画されます。

プログラムの例　PLine.java

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

public class PLine extends JFrame {
    PLine(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(Color.red);
        g.drawLine(50, 60, 350, 260);
    }

    public static void main(String[] args) {
        PLine frame = new PLine("直線の描画");
    }
}
```

2.3.3 長方形

Graphics クラスのオブジェクトは、長方形を描画するメソッドをいくつも持っています。そのうちのひとつ、 drawRect は、長方形の輪郭だけを描画するメソッドです。

drawRect には、引数として 4 個の整数を渡します。最初の 2 個は、長方形の左上の隅の x 座標と y 座標です。そして 3 個目は長方形の横の長さ、4 個目は縦の長さです。たとえば、

```
g.drawRect(60, 50, 200, 150);
```

という式文で drawRect を呼び出したとすると、左上の頂点の座標が $(60, 50)$ で、横の長さが 200、縦の長さが 150、という長方形の輪郭が描画されます。

fillRect というメソッドは、引数として 4 個の整数を受け取って、それらの引数で指定された長方形の内部を塗りつぶします。4 個の引数の意味は、 drawRect と同じです。

drawRoundRect と fillRoundRect は、角が丸くなった長方形を描画するメソッドです。これらのメソッドは、どちらも、引数として 6 個の整数を受け取ります。最初の 4 個の引数の意味は、drawRect と同じです。5 個目と 6 個目は、長方形の角に取り付ける楕円が内接する長方形の大きさです。5 個目は横の長さで、6 個目は縦の長さです。たとえば、

```
g.drawRoundRect(100, 150, 200, 300, 40, 80);
```

という式文で drawRoundRect を呼び出したとすると、長方形の四隅に、横の長さが 40、縦の長さが 80 の長方形に内接する楕円が取り付けられることになります。

プログラムの例 PRect.java

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

public class PRect extends JFrame {
    PRect(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(Color.green);
        g.drawRect(40, 50, 140, 100);
        g.fillRect(220, 50, 140, 100);
        g.drawRoundRect(40, 170, 140, 100, 50, 30);
        g.fillRoundRect(220, 170, 140, 100, 50, 30);
    }
```

```
    public static void main(String[] args) {
        PRect frame = new PRect("長方形の描画");
    }
}
```

2.3.4　楕円弧

drawArc は、楕円弧の輪郭を描画するメソッドで、fillArc は、楕円弧の内部の扇形を塗りつぶすメソッドです。

drawArc と fillArc は、引数として 6 個の整数を受け取ります。最初の 4 個は、楕円弧が内接する長方形の位置と大きさで、それぞれの引数の意味は drawRect などと同じです。5 個目と 6 個目は、楕円弧を描画する角度の範囲です（角度の単位は度）。5 個目の引数は、右方向を 0 度としたときに楕円弧の描画を開始する角度で、6 個目の引数は、描画を開始する角度を 0 度としたときに描画を終了する角度です。角度は、反時計回りはプラス、時計回りはマイナスで指定します。たとえば、

```
g.drawArc(50, 30, 200, 150, 90, -135);
```

という式で drawArc を呼び出したとすると、左上の頂点が $(50, 30)$、横の長さが 200、縦の長さが 150、という長方形に内接する楕円弧の輪郭が、上方向から右斜め下方向まで時計回りに描画されることになります。

楕円弧は、まず最初に円弧が作られて、その上下方向と左右方向の比率を変えることによって作られる、と考えることができます。引数で指定する角度の範囲は、比率を変える前の円弧に適用されるものですので、引数で指定した角度と完成した楕円弧の角度は、かならず一致するわけではありません。

プログラムの例　PArc.java

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

public class PArc extends JFrame {
    PArc(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(Color.blue);
        g.drawArc(20, 50, 200, 120, 225, 270);
        g.fillArc(180, 140, 200, 120, -45, -270);
    }

    public static void main(String[] args) {
        PArc frame = new PArc("楕円弧の描画");
    }
}
```

2.4　文字列の描画

2.4.1　文字列を描画するメソッド

Graphics クラスのオブジェクトは、コンポーネントの上にグラフィックスとして文字列を描画する、drawString というメソッドを持っています。

drawString は、3 個の引数を受け取ります。1 個目は描画する文字列で、2 個目と 3 個目は、文字列を描画する位置の x 座標と y 座標です。たとえば、

```
g.drawString("反射衛星砲", 100, 70);
```

という式文で drawString を呼び出したとすると、$(100, 70)$ という位置に「反射衛星砲」という

文字列が描画されます。

プログラムの例 PString.java

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

public class PString extends JFrame {
    PString(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(new Color(0, 128, 0));
        g.drawString("私は文字列です。", 160, 140);
    }

    public static void main(String[] args) {
        PString frame = new PString("文字列の描画");
    }
}
```

2.4.2 フォントの設定

Graphics クラスのオブジェクトは、フォントをあらわすオブジェクトをひとつ保持していて、drawString は、そのオブジェクトがあらわしているフォントを使って文字列を描画します。

Graphics クラスのオブジェクトが保持しているフォントのオブジェクトは、Graphics クラスのオブジェクトが持っている setFont というメソッドを呼び出すことによって変更することができます。setFont は、フォントのオブジェクトを引数として受け取って、それをレシーバーに設定します。

フォントをあらわすオブジェクトは、Font というクラスから生成されます。Font クラスからオブジェクトを生成するときは、普通、3 個の引数を受け取るコンストラクタを使います。

コンストラクタに渡す引数の 1 個目は、「論理フォント名」(logical font name) と呼ばれる名前をあらわす文字列です。論理フォント名というのは、Java のプログラムの中で使われる、環境に依存しないでフォントを指定するための名前のことです。論理フォント名としては、Serif、SansSerif、Monospaced などがあります。

引数の 2 個目は、フォントのスタイルをあらわす整数です。この整数は、Font クラスで定義されている、

Font.PLAIN	普通のスタイル
Font.ITALIC	イタリック体
Font.BOLD	太字

という定数を使って指定します。イタリック体でかつ太字というスタイルは、

```
Font.ITALIC|Font.BOLD
```

という式の値、つまりそれらのスタイルをあらわす整数の論理和によって指定されます。

そして引数の 3 個目は、フォントの大きさをあらわす整数です。フォントの大きさは、単位としてポイントを使って指定します。

ですから、たとえば、

```
new Font("SansSerif", Font.ITALIC, 32)
```

という式を評価することによって、論理フォント名が SansSerif で、スタイルがイタリックで、大きさが 32 ポイント、というフォントをあらわすオブジェクトを生成することができます。

プログラムの例 PFont.java

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

```
public class PFont extends JFrame {
    PFont(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(new Color(0, 128, 128));
        g.setFont(new Font("Serif", Font.PLAIN, 28));
        g.drawString("Serif PLAIN", 30, 70);
        g.setFont(new Font("Serif", Font.ITALIC, 28));
        g.drawString("Serif ITALIC", 30, 110);
        g.setFont(new Font("Serif", Font.BOLD, 28));
        g.drawString("Serif BOLD", 30, 150);
        g.setFont(new Font("Serif",
            Font.ITALIC|Font.BOLD, 28));
        g.drawString("Serif ITALIC|BOLD", 30, 190);
        g.setFont(new Font("SansSerif", Font.PLAIN, 28));
        g.drawString("SansSerif PLAIN", 30, 230);
        g.setFont(new Font("Monospaced", Font.PLAIN, 28));
        g.drawString("Monospaced PLAIN", 30, 270);
    }

    public static void main(String[] args) {
        PFont frame = new PFont("フォントの設定");
    }
}
```

2.5　イメージの描画

2.5.1　イメージのロード

コンポーネントの上には、図形や文字列だけではなくて、イメージを描画することもできます。

Swing は、イメージをあらわすオブジェクトを生成する、ImageIcon というクラスを持っています。コンポーネントの上にイメージを描画するためには、そのための準備として、このクラスのオブジェクトを生成する必要があります。

ImageIcon クラスにはさまざまなコンストラクタが定義されていて、さまざまな方法でオブジェクトを初期化することができます。ファイルからイメージのデータをロードして、そのデータでオブジェクトを初期化したいときは、ファイルのパス名をあらわす文字列を引数として受け取るコンストラクタを使います。たとえば、

```
new ImageIcon("namako.jpg")
```

という式を評価することによって、namako.jpg というパス名を持つファイルからイメージのデータをロードして、そのイメージをあらわすオブジェクトを生成することができます。

2.5.2　イメージを描画するメソッド

Graphics クラスのオブジェクトは、コンポーネントの上にイメージを描画する、drawImage という名前を持ついくつかのメソッドを持っています。ここでは、それらのメソッドのうちで、

```
drawImage(Image, int, int, int, int, ImageObserver)
```

というシグネチャーを持つものを紹介します。

1 個目の引数は、イメージをあらわすオブジェクトです。ただし、drawImage が受け取るのは、ImageIcon クラスのオブジェクトではなくて、Image というクラスのオブジェクトです。ImageIcon クラスのオブジェクトは、自分の内部に Image クラスのオブジェクトを持っていて、getImage というメソッドを呼び出すことによって、それを取り出すことができます。

引数の 2 個目から 5 個目までは、イメージを描画する領域を指定する長方形をあらわす整数です。2 個目が左上の頂点の x 座標、3 個目が y 座標、4 個目が横の長さ、5 個目が縦の長さです。

6 個目の引数は、イメージが描画されるコンポーネントです。これは、 this を評価することによって求めることができます。

ですから、たとえば、 icon という変数が ImageIcon クラスのオブジェクトを指し示しているとするとき、

```
g.drawImage(icon.getImage(), 200, 100, 360, 280, this);
```

という式文を実行することによって、そのオブジェクトがあらわしているイメージを、左上の頂点が (200, 100)、横の長さが 360、縦の長さが 280 という長方形の内部に描画することができます。

プログラムの例　PImage.java

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

public class PImage extends JFrame {
    ImageIcon icon;

    PImage(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
        icon = new ImageIcon("sample.jpg");
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.drawImage(icon.getImage(),
            100, 70, 200, 150, this);
    }

    public static void main(String[] args) {
        PImage frame = new PImage("イメージの描画");
    }
}
```

第 3 章　イベント

3.1　イベントの基礎

3.1.1　イベントとは何か

GUI を持つプログラムの多くは、普段は何もしていない状態になっていて、何らかのきっかけとなる出来事が発生したときだけ、それに対応する動作を実行する、というように作られています。そのような、プログラムを動作させるきっかけになる出来事は、「イベント」(event) と呼ばれます。マウスなどのポインティングデバイスやキーボードなどを人間が操作するという出来事は、イベントの一種です。

そして、イベントが発生したときだけ動作するように作られているプログラムは、「イベント駆動型」(event-driven) である、と言われます。

3.1.2　イベントソースとイベントリスナー

GUI を持つプログラムでは、イベントは、何らかのコンポーネントに関連して発生します。Java では、イベントの発生に関連するコンポーネントのことを、「イベントソース」(event source) と呼びます。

そして、Java では、イベントが発生したときに呼び出されるメソッドを持っているオブジェクトのことを、「イベントリスナー」(event listener) と呼びます。

Java では、イベント駆動型のプログラムは、コンポーネントに対してイベントリスナーを登録する、という形で実現されます。そうしておくと、イベントが発生したとき、そのイベントのイベントソースが、自分に登録されているイベントリスナーの中のメソッドを呼び出すことになります。

コンポーネントにイベントリスナーを登録したいときは、コンポーネントが持っている、そのためのメソッドを呼び出します。イベントリスナーを登録するメソッドは、イベントリスナーを引数として受け取って、それをレシーバーに登録します。

3.1.3 イベントリスナーを生成するクラス

イベントリスナーというのはオブジェクトですから、何らかのクラスから生成されます。イベントリスナーを生成するクラスというのは、「リスナーインターフェース」(listener interface) と呼ばれるインターフェースを実装するクラスのことです。

ですから、原則としては、イベントリスナーを生成するためには、まず、リスナーインターフェースを実装するクラスを定義する必要があります。しかし、その原則のとおりの方法だと、必要ではないメソッドの定義まで書かないといけませんので、かなり煩雑です。そこで、もう少し簡略化された方法も準備されています。それは、「アダプタークラス」(adapter class) と呼ばれるクラスのサブクラスを定義するという方法です（ただし、すべてのリスナーインターフェースに対してアダプタークラスが定義されているわけではありません)。

アダプタークラスのサブクラスとして、イベントに応じて動作するメソッドを持つクラスを定義して、そのクラスからオブジェクトを生成すると、そのオブジェクトがイベントリスナーになります。

イベントリスナーを生成するクラスは、クラス宣言を書くことによって定義してもかまわないのですが、クラスに名前を付ける必要のない場合がほとんどですので、匿名内部クラスを定義する構文、つまり名前のないクラスを定義して、そのクラスからオブジェクトを生成する構文を使うのが普通です。

なお、マウスやキーボードなどの操作による基本的なイベントに関連するインターフェースやクラスは、`java.awt.event` というパッケージの中で定義されています。

3.1.4 イベント処理メソッド

イベントリスナーが持っているメソッドは、「イベント処理メソッド」(event processing method) と呼ばれます。イベント処理メソッドは、対応するイベントが発生したとき、イベントソースによって呼び出されます。ですから、イベント処理メソッドの宣言の中に書かれた動作が、イベントが発生したときに実行されることになります。

イベント処理メソッドの宣言は、リスナーインターフェースを実装する記述、またはアダプタークラスのサブクラスを定義する記述の中に書きます。

イベント処理メソッドは、対応するイベントの種類に応じて名前が決まっています。それらのメソッドは、発生したイベントをあらわすオブジェクトを引数として受け取り、戻り値は返しません。

3.2 マウスのイベント

3.2.1 マウスのクリック

それでは、イベント駆動型のプログラムを実際に Java を使って書いてみることにしましょう。まず最初は、マウスでクリックされたというイベントが発生したときに動作を実行するプログラムです。

マウスのクリックというイベントに反応するプログラムは、

イベントリスナーを登録するメソッド	`addMouseListener`
アダプタークラス	`MouseAdapter`
イベント処理メソッド	`mouseClicked`
イベントのオブジェクトのクラス	`MouseEvent`

という名前のクラスやメソッドを使って書きます。たとえば、フレームを初期化するコンストラクタの中で、

```
addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        JOptionPane.showMessageDialog(null,
            "マウスでクリックされました。",
            "クリックのお知らせ",
            JOptionPane.INFORMATION_MESSAGE);
```

```
        }
    });
```

という式文を実行することによって、マウスでクリックされたときに呼び出される mouseClicked というイベント処理メソッドを持つイベントリスナーをフレームに登録することができます。この例では、フレームがマウスでクリックされると、そのイベントを通知するダイアログボックスが表示されることになります。

プログラムの例 PClick.java

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

public class PClick extends JFrame {
    PClick(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                JOptionPane.showMessageDialog(null,
                    "マウスでクリックされました。",
                    "クリックのお知らせ",
                    JOptionPane.INFORMATION_MESSAGE);
            }
        });
    }

    public static void main(String[] args) {
        PClick frame = new PClick("マウスのクリック");
    }
}
```

3.2.2 イベントによる描画

コンポーネントの上に表示されているグラフィックスを、何らかのイベントが発生したときに更新したい、というときは、イベント処理メソッドの宣言の中で、repaint という、グラフィックスの再描画を要求するメソッドを呼び出します。

次のプログラムは、フレームがマウスでクリックされるたびに、フレームの上に表示されている整数を 1 ずつ大きくします。

プログラムの例 PRepaint.java

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

public class PRepaint extends JFrame {
    int count;

    PRepaint(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                count++;
                repaint();
            }
        });
        count = 0;
    }

    public void paint(Graphics g) {
        super.paint(g);
```

```
        g.setColor(Color.blue);
        g.setFont(new Font("Serif", Font.PLAIN, 128));
        g.drawString(Integer.toString(count), 70, 180);
    }

    public static void main(String[] args) {
        PRepaint frame = new PRepaint("イベントによる描画");
    }
}
```

3.2.3 マウスポインターの位置

イベント処理メソッドは、発生したイベントをあらわしているオブジェクトを引数として受け取ります。マウスによって発生したイベントは、MouseEvent というクラスから生成されるオブジェクトによってあらわされます。このクラスのオブジェクトが持っている getPoint というメソッドは、イベントが発生したときのマウスポインターの位置をあらわしている座標を戻り値として返します。

getPoint が返すのは、Point というクラスから生成されるオブジェクトです。このクラスのオブジェクトは、x と y という int のフィールドを持っていて、それぞれのフィールドには x 座標と y 座標が格納されます。

次のプログラムは、フレームがマウスでクリックされると、そのときのマウスポインターの位置に円を描画します。

プログラムの例 PPoint.java

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

public class PPoint extends JFrame {
    Point clicked;

    PPoint(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                clicked = e.getPoint();
                repaint();
            }
        });
        clicked = new Point(-100, -100);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(new Color(0, 0, 128));
        g.fillOval(clicked.x - 10, clicked.y - 10, 20, 20);
    }

    public static void main(String[] args) {
        PPoint frame = new PPoint("クリックされた位置");
    }
}
```

3.2.4 マウスのボタンの判定

ところで、クリックされたマウスのボタンが左ボタンなのか右ボタンなのかというのは、どのようにして判定すればいいのでしょうか。

Swing は、SwingUtilities という、さまざまなクラスメソッドから構成されるクラスを持っています。イベントを発生させたマウスのボタンは、SwingUtilities の中にある、

isLeftMouseButton　左ボタンならば真。

isMiddleMouseButton 中央ボタンならば真。

isRightMouseButton 右ボタンならば真。

というメソッドを呼び出すことによって判定することができます。これらのメソッドは、イベントのオブジェクトを引数として受け取って、判定の結果を真偽値で返します。たとえば、イベントのオブジェクトを指し示している変数を e とするとき、

```
SwingUtilities.isLeftMouseButton(e)
```

という式を評価すると、その値として、イベントを発生させたボタンが左ボタンならば真、それ以外のボタンならば偽が得られます。

次のプログラムは、フレームが左ボタンでクリックされたときは円を左へ移動させ、右ボタンでクリックされたときは円を右へ移動させます。

プログラムの例 PMouseButton.java

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

public class PMouseButton extends JFrame {
    final static int Y = 140;
    final static int STEP = 20;
    final static int SIZE = 30;
    int x;

    PMouseButton(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                if (SwingUtilities.isLeftMouseButton(e))
                    x -= STEP;
                else if (SwingUtilities.isRightMouseButton(e))
                    x += STEP;
                repaint();
            }
        });
        x = 185;
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(new Color(0, 128, 128));
        g.fillOval(x, Y, SIZE, SIZE);
    }

    public static void main(String[] args) {
        PMouseButton frame = new PMouseButton(
            "マウスのボタンの判定");
    }
}
```

3.2.5 マウスの移動

それでは次に、マウスが移動したというイベントが発生したときに動作を実行するプログラムを書いてみることにしましょう。

マウスの移動というイベントに反応するプログラムは、

イベントリスナーを登録するメソッド	addMouseMotionListener
アダプタークラス	MouseMotionAdapter
イベント処理メソッド	mouseMoved
イベントのオブジェクトのクラス	MouseEvent

という名前のクラスやメソッドを使って書きます。たとえば、フレームを初期化するコンストラクタの中で、

```
addMouseMotionListener(new MouseMotionAdapter() {
    public void mouseMoved(MouseEvent e) {
        position = e.getPoint();
        repaint();
    }
});
```

という式文を実行することによって、マウスが移動したときに呼び出される mouseMoved というイベント処理メソッドを持つイベントリスナーをフレームに登録することができます。この例では、フレームの上でマウスが移動すると、マウスポインターの位置が position に代入されたのち、フレームのグラフィックスが更新されることになります。

次のプログラムは、フレームの上でマウスが移動すると、そのときのマウスポインターの位置をあらわす座標を描画します。

プログラムの例　PMouseMotion.java

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

public class PMouseMotion extends JFrame {
    Point position;

    PMouseMotion(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
        addMouseMotionListener(new MouseMotionAdapter() {
            public void mouseMoved(MouseEvent e) {
                position = e.getPoint();
                repaint();
            }
        });
        position = new Point(0, 0);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(new Color(0, 128, 0));
        g.setFont(new Font("Serif", Font.PLAIN, 58));
        g.drawString(
            "(" + position.x + "," + position.y + ")",
            60, 110);
    }

    public static void main(String[] args) {
        PMouseMotion frame = new PMouseMotion("マウスの移動");
    }
}
```

3.3　キーボードのイベント

3.3.1　キーが押されたというイベント

それでは次に、キーボードのキーが押されたというイベントが発生したときに動作を実行するプログラムを書いてみましょう。

キーが押されたというイベントに反応するプログラムは、

イベントリスナーを登録するメソッド	addKeyListener
アダプタークラス	KeyAdapter
イベント処理メソッド	keyPressed

イベントのオブジェクトのクラス　　KeyEvent

という名前のクラスやメソッドを使って書きます。たとえば、フレームを初期化するコンストラクタの中で、

```
addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent e) {
        JOptionPane.showMessageDialog(null,
            "キーボードのキーが押されました。",
            "キーボードのイベントのお知らせ",
            JOptionPane.INFORMATION_MESSAGE);
    }
});
```

という式文を実行することによって、キーボードのキーが押されたときに呼び出される keyPressed というイベント処理メソッドを持つイベントリスナーをフレームに登録することができます。この例では、キーボードのキーが押されると、そのイベントを通知するダイアログボックスが表示されることになります。

プログラムの例　PKey.java

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

public class PKey extends JFrame {
    PKey(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
        addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                JOptionPane.showMessageDialog(null,
                    "キーボードのキーが押されました。",
                    "キーボードのイベントのお知らせ",
                    JOptionPane.INFORMATION_MESSAGE);
            }
        });
    }

    public static void main(String[] args) {
        PKey frame = new PKey("キーボードのイベント");
    }
}
```

3.3.2 キーコード

ところで、キーボードのイベントを発生させたのがキーボードのどのキーなのかということを判定したいときは、いったいどうすればいいのでしょうか。

キーボードの上にあるそれぞれのキーは、「キーコード」(key code) と呼ばれる整数によって識別されます。キーボードのイベントをあらわすオブジェクトが持っている getKeyCode というメソッドは、イベントを発生させたキーのキーコードを戻り値として返します。ですから、このメソッドを呼び出すことによって、イベントを発生させたのがどのキーなのかということを判定することができます。

キーコードは、KeyEvent クラスの中で定義されている定数によって書きあらわすことができます。それらの定数は、すべて、

VK_ キーの名前

という形になっています。たとえば、数字の 6 は VK_6、英字の M は VK_M、コンマは VK_COMMA、シフトは VK_SHIFT、エンターは VK_ENTER、上向き矢印は VK_UP、ファンクションキーの 1 番 (F1) は VK_F1、という定数であらわされます。

次のプログラムは、フレームの上に描画された円を、左向き矢印のキーが押された場合は左へ移動させ、右向き矢印のキーが押された場合は右へ移動させます。

プログラムの例 PKeyCode.java

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

public class PKeyCode extends JFrame {
    final static int Y = 140;
    final static int STEP = 20;
    final static int SIZE = 30;
    int x;

    PKeyCode(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
        addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                int kc = e.getKeyCode();
                if (kc == KeyEvent.VK_LEFT)
                    x -= STEP;
                else if (kc == KeyEvent.VK_RIGHT)
                    x += STEP;
                repaint();
            }
        });
        x = 185;
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(new Color(128, 0, 255));
        g.fillOval(x, Y, SIZE, SIZE);
    }

    public static void main(String[] args) {
        PKeyCode frame = new PKeyCode("キーコード");
    }
}
```

3.3.3 キーをあらわす文字列

KeyEvent クラスは、getKeyText というクラスメソッドを持っています。このメソッドは、引数としてキーコードを受け取って、それによって指定されたキーをあらわす文字列を戻り値として返します。

次のプログラムは、キーボードのキーが押されると、そのキーをあらわす文字列をフレームの上に描画します。

プログラムの例 PKeyText.java

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

public class PKeyText extends JFrame {
    String pressed;

    PKeyText(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
        addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                pressed = KeyEvent.getKeyText(e.getKeyCode());
                repaint();
```

```
            }
        });
        pressed = "";
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(new Color(0, 0, 128));
        g.setFont(new Font("Serif", Font.PLAIN, 58));
        g.drawString(pressed, 60, 110);
    }

    public static void main(String[] args) {
        PKeyText frame = new PKeyText("キーをあらわす文字列");
    }
}
```

3.4 ウィンドウのイベント

3.4.1 ウィンドウのイベントの基礎

フレームやダイアログボックスなどのウィンドウは、人間が自分に対して何らかの操作を実行したときなどに、それに対応するイベントを発生させます。

ウィンドウが発生させたイベントに反応するプログラムは、

イベントリスナーを登録するメソッド	`addWindowListener`
アダプタークラス	`WindowAdapter`
イベントのオブジェクトのクラス	`WindowEvent`

という名前のクラスやメソッドを使うことによって、書くことができます。

ウィンドウが発生させたイベントを処理するイベント処理メソッドとしては、次のようなものがあります。

`windowOpened`	ウィンドウが最初に可視になった
`windowClosing`	人間によって閉じる操作が実行された
`windowClosed`	ウィンドウが閉じた
`windowActivated`	ウィンドウがアクティブになった
`windowDeactivated`	ウィンドウがアクティブではなくなった
`windowIconified`	ウィンドウが最小化された
`windowDeiconified`	最小化されていたウィンドウが元の状態に復帰した

3.4.2 終了の確認

ウィンドウが発生させるイベントのうちで、もっとも頻繁に利用されるのは、人間によって閉じる操作が実行されたときに発生するイベントでしょう。プログラムの多くは、メインのウィンドウに対して閉じる操作が実行された場合、本当に終了してもいいのかどうかという確認を人間に求めます。そのような処理を実行するためには、閉じる操作が実行されたときに発生したイベントを処理する必要があります。

ウィンドウを閉じる操作が実行されたときの動作は、基本的には、

```
setDefaultCloseOperation
```

というメソッドを使って設定します。閉じる操作によって発生したイベントを処理する場合は、このメソッドによって設定される基本的な動作が、「何もしない」というものになっている必要があります。つまり、

```
setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
```

という設定をしておかないといけないわけです。

プログラムを本当に終了してもいいという確認が得られた場合は、プログラムを本当に終了させないといけないわけですが、その動作は、

```
System.exit(0);
```

という式文で、exitというメソッドを呼び出すことによって実行することができます。

プログラムの例 PConfirmExit.java

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

public class PConfirmExit extends JFrame {
    PConfirmExit(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                confirmExit();
            }
        });
    }

    void confirmExit() {
        int answer = JOptionPane.showConfirmDialog(null,
            "本当に終了してもいいですか？", "終了の確認",
            JOptionPane.YES_NO_OPTION);
        if (answer == JOptionPane.YES_OPTION)
            System.exit(0);
    }

    public static void main(String[] args) {
        PConfirmExit frame = new PConfirmExit(
            "終了の確認をするプログラム");
    }
}
```

3.5 タイマー

3.5.1 人間による操作とは無関係なイベント

イベント駆動型のプログラムは、基本的には、人間が何も操作していないときは何もしないで待機しているわけですが、場合によっては、人間が何も操作していないときにも動作を実行することが必要になることもあります。では、人間が何も操作していないときにも動作を実行するようなイベント駆動型のプログラムは、いったいどのように書けばいいのでしょうか。

イベント駆動型のプログラムは、イベントが発生したときに動作を実行するわけですから、人間が何も操作していないときにも動作を実行させたいのならば、人間による操作とは無関係にイベントを発生させればいい、ということになります。そのような、人間による操作とは無関係なイベントは、「タイマー」(timer) と呼ばれるオブジェクトを使うことによって発生させることができます。

タイマーというのは、一定の時間間隔でイベントを発生させるオブジェクトのことです。タイマーはコンポーネントではありませんが、コンポーネントと同じように、そこにイベントリスナーを登録しておくことができます。タイマーに登録されたイベントリスナーの中にあるイベント処理メソッドは、一定の時間間隔でイベントが発生するたびに呼び出されることになります。

3.5.2 タイマーの生成

タイマーは、Swingの中にあるTimerというクラスから生成されます。Timerクラスのコンストラクタは、2個の引数を受け取ります。1個目はイベントを発生させる時間間隔をあらわす整数（単位はミリ秒）で、2個目はタイマーに登録するイベントリスナーです。

タイマーに登録するイベントリスナーは、ActionListenerというインターフェースを実装するクラスのオブジェクトです。イベント処理メソッドとしては、actionPerformedという名前のものを定義します。このイベント処理メソッドは、ActionEventというクラスのオブジェクトを引数として受け取ります。

ですから、たとえば、

```
    timer = new Timer(1000, new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            count++;
            repaint();
        }
    });
```

という式文を書くことによって、タイマーを生成して、それを timer という変数に代入することができます。この例では、生成されたタイマーは、1000 ミリ秒（1 秒）の間隔でイベントを発生させます。そして、イベントが発生するたびに、count がインクリメントされて、グラフィックスが更新されることになります。

3.5.3 タイマーの起動と停止

タイマーは、生成された直後は停止した状態になっています。タイマーを起動したいときは、そのタイマーが持っている start というメソッドを呼び出します。逆に、動作中のタイマーを停止させたいときは、stop というメソッドを呼び出します。

タイマーが動作しているかそれとも停止しているかということを知りたいときは、isRunning というメソッドを呼び出します。isRunning は、タイマーが動作しているならば真を返し、停止しているならば偽を返します。

次のプログラムは、フレームがクリックされたのち、フレームの上に描画された整数を 1 秒ごとに 1 だけ増加させます。そして、ふたたびフレームがクリックされると、整数の増加を停止させます。

プログラムの例 PTimer.java

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

public class PTimer extends JFrame {
    int count;
    Timer timer;

    PTimer(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setVisible(true);
        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                if (timer.isRunning())
                    timer.stop();
                else {
                    count = 0;
                    timer.start();
                    repaint();
                }
            }
        });
        timer = new Timer(1000, new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                count++;
                repaint();
            }
        });
        count = 0;
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(new Color(0, 128, 128));
        g.setFont(new Font("Serif", Font.PLAIN, 128));
        g.drawString(Integer.toString(count), 70, 180);
    }
```

```
    public static void main(String[] args) {
        PTimer frame = new PTimer("タイマー");
    }
}
```

3.6 ゲーム

3.6.1 キャラクターのクラス

この節では、ゲームのプログラムはいったいどのように書けばいいのか、という問題について考えてみたいと思います。

ゲームのプログラムでは、画面の上の領域、というものが大きな役割を果たします。ゲームのプログラムが取り扱う、画面の上の領域のことを、「キャラクター」(character) と呼ぶことにしましょう。大多数のゲームでは、キャラクターがゲームの進行にともなって移動していき、それらの位置関係によって得点が加算されたり勝敗が決まったりします。

ゲームのプログラムを書く場合は、キャラクターのクラスをどのように定義するか、ということが重要な意味を持ちます。そこで、例として、Characterという名前で、きわめて単純な機能を持つキャラクターのクラスを定義してみることにしましょう。

キャラクターの本質は、画面の上の領域です。領域の形状は、キャラクターごとに違っているのが普通ですが、ここでは、クラスを単純なものにするために、どのキャラクターも領域の形状は長方形だと考えます。

AWT は、長方形の位置と大きさをあらわすRectangleというクラスを持っていますので、このクラスのサブクラスとしてキャラクターのクラスを定義することにしましょう。つまり、

```
class Character extends Rectangle { ··· }
```

というクラス宣言を書くわけです。

Rectangleクラスは、x、y、width、heightという 4 個のフィールドを持っていて、それぞれのフィールドには、長方形の左上の頂点の x 座標と y 座標、そして長方形の横の長さと縦の長さが格納されます。

キャラクターは、位置と大きさのほかにも、さまざまな属性を持つことができます。それらの属性は、キャラクターのオブジェクトの中にあるフィールドによってあらわされます。ここでは、領域の内部にグラフィックスを描画するときに使う色を、キャラクターの属性として加えることにしましょう。そこで、

```
Color color;
```

というフィールドを作っておいて、そこにキャラクターの色を設定することにします。

Rectangleクラスが持っているコンストラクタのうちのひとつは、引数として 4 個の整数を受け取って、それらをx、y、width、heightに設定します。このコンストラクタを使うと、Characterクラスのコンストラクタは、

```
Character(int x, int y, int width, int height,
          Color acolor) {
    super(x, y, width, height);
    color = acolor;
}
```

と書くことができます。

3.6.2 領域の内部にあるかどうかの判定

ゲームのプログラムでは、しばしば、キャラクターが何らかの領域の内部にあるかどうか、そして内部にない場合はどの方向へ出ているのかということを判定する必要が生じます。そこで、そのような判定を実行する、insideというメソッドを定義しておくことにしましょう。

insideは、判定の結果を整数であらわして、それを戻り値として返すことにしましょう。そこでまず、その整数を指定するために、

```
final static int INSIDE = 0;
final static int EAST   = 1;
final static int WEST   = 2;
final static int SOUTH  = 3;
```

```
    final static int NORTH  = 4;
```

というように定数を定義しておきます。

そうすると、 inside の定義は、

```
    int inside(Rectangle area) {
        if (x < area.x)
            return WEST;
        else if (x + width > area.x + area.width)
            return EAST;
        else if (y < area.y)
            return NORTH;
        else if (y + height > area.y + area.height)
            return SOUTH;
        else
            return INSIDE;
    }
```

と書くことができます。このメソッドは、引数として長方形を受け取って、レシーバーがその長方形の内部にあるかどうかを判定して、その結果をあらわす整数を戻り値として返します。

3.6.3 当たり判定

ゲームのプログラムでは、しばしば、キャラクターが別のキャラクターと衝突したかどうかを判定する必要が生じます。そのような判定は、「当たり判定」(collision detection) と呼ばれます。

それでは、当たり判定を実行する collision というメソッドを定義してみましょう。このメソッドは、 Character クラスが Rectangle クラスから継承している、 intersection と isEmpty というメソッドを使うことによって、簡単に定義することができます。

intersection は、引数として長方形を受け取って、レシーバーと引数との共通部分をあらわす長方形を戻り値として返すメソッドです。レシーバーと引数とが重なっていない場合は、マイナスの大きさを持つ長方形を返します。

isEmpty は、レシーバーの大きさがマイナスならば真を返して、そうでなければ偽を返すメソッドです。

これらのメソッドを使えば、 collision は、

```
    boolean collision(Character character) {
        return ! intersection(character).isEmpty();
    }
```

と定義することができます。このメソッドは、引数としてキャラクターを受け取って、レシーバーと引数との当たり判定を実行して、それらが衝突しているならば真を返して、そうでなければ偽を返します。

なお、この collision というメソッドによる当たり判定は、かなり手を抜いたものになっています。ゲームのプログラムの中で、当たり判定は、ゲームがプレイヤーに与える感覚を大きく左右する部分ですので、実際にゲームのプログラムを書く場合は、これよりもさらに精妙な当たり判定をすることが必要になります。

3.6.4 ゲームのプログラムの例

それでは、先ほど説明したキャラクターのクラスを使って、ゲームのプログラムをひとつ書いてみましょう。

次の例は、テニスのようなものをプレイすることができるプログラムです（テニスよりもスカッシュに近いかもしれません）。

プログラムの例 PTennis.java

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

class Character extends Rectangle {
    final static int INSIDE = 0;
    final static int EAST   = 1;
    final static int WEST   = 2;
    final static int SOUTH  = 3;
```

```
    final static int NORTH  = 4;

    Color color;

    Character(int x, int y, int width, int height,
              Color acolor) {
        super(x, y, width, height);
        color = acolor;
    }

    int inside(Rectangle area) {
        if (x < area.x)
            return WEST;
        else if (x + width > area.x + area.width)
            return EAST;
        else if (y < area.y)
            return NORTH;
        else if (y + height > area.y + area.height)
            return SOUTH;
        else
            return INSIDE;
    }

    boolean collision(Character character) {
        return ! intersection(character).isEmpty();
    }
}

class Ball extends Character {
    final static int SOUTHEAST = 0;
    final static int SOUTHWEST = 1;
    final static int NORTHWEST = 2;
    final static int NORTHEAST = 3;

    Rectangle court;
    int velocity, direction;

    Ball(Rectangle acourt, int size, int avelocity,
         Color color) {
        super(0, 0, size, size, color);
        court = acourt;
        velocity = avelocity;
        reset();
    }

    void reset() {
        setLocation(court.x, court.y);
        direction = SOUTHEAST;
    }

    void forward(int ax, int ay) {
        setLocation(x + velocity * ax, y + velocity * ay);
    }

    void directionForward() {
        switch (direction) {
        case SOUTHEAST:
            forward(1, 1);
            break;
        case SOUTHWEST:
            forward(-1, 1);
            break;
        case NORTHWEST:
            forward(-1, -1);
            break;
        case NORTHEAST:
            forward(1, -1);
```

```
            break;
        }
    }

    void turn() {
        switch (inside(court)) {
        case WEST:
            if (direction == SOUTHWEST)
                direction = SOUTHEAST;
            else if (direction == NORTHWEST)
                direction = NORTHEAST;
            break;
        case EAST:
            if (direction == SOUTHEAST)
                direction = SOUTHWEST;
            else if (direction == NORTHEAST)
                direction = NORTHWEST;
            break;
        case NORTH:
            if (direction == NORTHEAST)
                direction = SOUTHEAST;
            else if (direction == NORTHWEST)
                direction = SOUTHWEST;
            break;
        }
    }

    void hit() {
        if (direction == SOUTHEAST)
            direction = NORTHEAST;
        else if (direction == SOUTHWEST)
            direction = NORTHWEST;
    }

    boolean south() {
        return direction == SOUTHEAST ||
               direction == SOUTHWEST;
    }

    boolean beyondSouth() {
        return inside(court) == SOUTH;
    }

    void draw(Graphics g) {
        g.setColor(color);
        g.fillOval(x, y, width, height);
    }
}

class Racket extends Character {
    Racket(Rectangle court, int width, int height,
           Color color) {
        super(court.x + (court.width/2 - width/2),
              court.y + (court.height - height*2),
              width, height, color);
    }

    void setX(int ax) {
        x = ax - width/2;
    }

    void draw(Graphics g) {
        g.setColor(color);
        g.fillRect(x, y, width, height);
    }
}
```

```
public class PTennis extends JFrame {
    final static int FRAME_X = 400;
    final static int FRAME_Y = 300;

    int score;
    Rectangle court;
    Ball ball;
    Racket racket;
    Timer timer;

    PTennis(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(FRAME_X, FRAME_Y);
        setVisible(true);
        addMouseMotionListener(new MouseMotionAdapter() {
            public void mouseMoved(MouseEvent e) {
                racket.setX(e.getPoint().x);
                repaint();
            }
        });
        score = 0;
        Insets insets = getInsets();
        court = new Rectangle(insets.left, insets.top,
            FRAME_X - (insets.left + insets.right),
            FRAME_Y - (insets.top + insets.bottom));
        ball = new Ball(court, 20, 10, new Color(128, 0, 0));
        racket = new Racket(court, 60, 15,
                            new Color(0, 0, 128));
        timer = new Timer(10, new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                background();
            }
        });
        timer.start();
    }

    void gameover() {
        JOptionPane.showMessageDialog(this,
            "ゲームオーバーです。",
            "ゲームオーバー",
            JOptionPane.INFORMATION_MESSAGE);
        score = 0;
        ball.reset();
    }

    void background() {
        if (ball.beyondSouth())
            gameover();
        ball.turn();
        if (ball.south() && ball.collision(racket)) {
            ball.hit();
            score++;
        }
        ball.directionForward();
        repaint();
    }

    void drawScore(Graphics g) {
        g.setColor(new Color(0, 128, 0));
        g.setFont(new Font("Serif", Font.PLAIN, 32));
        g.drawString(Integer.toString(score),
                     court.x + 20, court.y + 40);
    }

    public void paint(Graphics g) {
        super.paint(g);
```

```
        drawScore(g);
        ball.draw(g);
        racket.draw(g);
    }

    public static void main(String[] args) {
        PTennis frame = new PTennis("テニスのようなもの");
    }
}
```

第4章　ボタン

4.1　ボタンの基礎

4.1.1　ボタンとは何か

クリックされたときに何らかの動作を実行するコンポーネントは、「ボタン」(button) と呼ばれます。

ボタンには、「プッシュボタン」(push button)、「チェックボックス」(check box)、「ラジオボタン」(radio button) など、さまざまな種類があります。また、メニューやメニューの項目も、ボタンの一種だと考えることができます。

ただし、実際には、「ボタン」という言葉がこのような広い意味で使われるのはまれなことで、多くの場合、この言葉はプッシュボタンという狭い意味で使われます。しかし、「ボタン」という言葉には狭い意味だけではなくてもっと広い意味もあるということは、忘れないようにしていただきたいと思います。

4.1.2　ボタンの生成

Swing では、ボタン（プッシュボタン）は、 JButton というクラスから生成されるコンポーネントです。

JButton クラスが持っているいくつかのコンストラクタのうちのひとつは、引数として文字列を受け取って、その引数を、ボタンの上に表示する文字列として設定します。ですから、

```
JButton button = new JButton("実行");
```

という宣言文を書くことによって、「実行」と表示されたボタンが生成されて、それが button という変数に設定されます。

なお、ボタンなどのコンポーネントの上に表示される文字列は、コンポーネントが持っている setText というメソッドを使うことによって、いつでも変更することが可能です。このメソッドは、引数として文字列を受け取って、それをレシーバーに設定します。

4.1.3　コンテナ

フレームやダイアログボックスは、画面の上に単独で表示することができるという特殊な性質を持っています。それに対して、ボタンのような普通のコンポーネントは、単独では画面の上に表示できません。普通のコンポーネントを画面に表示するためには、すでに画面に表示されている何らかのコンポーネントの上にそれを取り付けるという操作が必要なのです。

自分の上にコンポーネントを取り付けることのできるコンポーネントは、「コンテナ」(container) と呼ばれます。AWT は、コンテナに必要となるさまざまな機能を提供する Container というクラスを持っていて、このクラスを継承するすべてのクラスは、コンテナとして使うことができます。たとえば、 JFrame は、 Container を継承するクラスのひとつです。

コンポーネントをコンテナに取り付けることを、コンポーネントをコンテナに「追加する」(add) と言います。コンテナにコンポーネントを追加したいときは、コンテナが持っている add というメソッドを使います。このメソッドは、コンポーネントを引数として受け取って、それをレシーバーに追加します。たとえば、 container という変数がコンテナを、 component という変数がコンポーネントを指し示しているとするとき、

```
container.add(component);
```

という式文を実行すると、 component が container に追加されることになります。

4.1.4 コンテントペイン

JFrame は、Container を継承するクラスのひとつです。つまり、フレームというのはコンテナの一種ですので、コンポーネントをそれに追加することができます。しかし、フレームというのはコンテナとしては少し特殊で、それにコンポーネントを追加するためには、少し込み入った操作が必要になります。

フレームは、自分の内部に「コンテントペイン」(content pane) と呼ばれるコンテナを持っています。コンテナにコンポーネントを追加したいときは、まずフレームからコンテントペインを取得して、それにコンポーネントを追加しないといけません。

フレームからコンテントペインを取得したいときは、フレームが持っている getContentPane というメソッドを呼び出します。このメソッドは、レシーバーからコンテントペインを取得して、それを戻り値として返します。このメソッドは戻り値の型が Container なのですが、もともとコンテントペインは JPanel というクラスのオブジェクトですので、普通は、キャストを使ってコンテントペインの型を JPanel に戻します。つまり、frame という変数が指し示しているフレームからコンテントペインを取得したいとすれば、

```
(JPanel)frame.getContentPane()
```

という式を書けばいい、ということです。

4.1.5 レイアウトマネージャー

Java では、コンテナの上でのコンポーネントのレイアウト（配置）は、「レイアウトマネージャー」(layout manager) と呼ばれるオブジェクトによって管理されます。レイアウトマネージャーには、コンポーネントをレイアウトする方法が異なるさまざまな種類のものがあります。コンテナには、あらかじめデフォルトのレイアウトマネージャーが設定されていますが、別の種類のレイアウトマネージャーをコンテナに設定することもできます。

レイアウトマネージャーをコンテナに設定したいときは、コンテナが持っている setLayout というメソッドを呼び出します。このメソッドは、引数としてレイアウトマネージャーを受け取って、それをレシーバーに設定します。

各種のレイアウトマネージャーについては次の節で詳しく説明する予定なのですが、ここでも、ひとつだけ紹介しておくことにしましょう。それは、「フローレイアウト」(flow layout) と呼ばれるレイアウトマネージャーです。このレイアウトマネージャーは、コンポーネントを、コンテナに追加された順番のとおりに左から右へ並べていきます。そして、コンテナの右端に到達したのちは、ひとつ下の段へ移って、やはり左から右へと並べていきます。このように、一定の流れに沿ってコンポーネントを並べていきますので、「フローレイアウト」と呼ばれるわけです。

フローレイアウトは、FlowLayout というクラスから生成されるオブジェクトです。container という変数がコンテナを指し示しているとすると、

```
container.setLayout(new FlowLayout());
```

という式文を実行することによって、フローレイアウトを生成して、それをコンテナに設定することができます。

次のプログラムは、ボタンをひとつ生成して、それをフレームに追加します。ただし、そのボタンは、クリックされても動作は何も実行しません。

プログラムの例 PButton.java

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

public class PButton extends JFrame {
    PButton(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.setLayout(new FlowLayout());
        content.add(new JButton("私はボタンです。"));
        setVisible(true);
    }

    public static void main(String[] args) {
        PButton frame = new PButton("何もしないボタン");
```

```
    }
}
```

4.1.6 ボタンの動作

前の章で説明したように、Java では、コンポーネントに対してイベントリスナーを登録することによって、イベントが発生したときのコンポーネントの動作を設定することができます。ボタンについてもまったく同じことですので、ボタンに対してイベントリスナーを登録しておけば、そのボタンは、クリックされたときに何らかの動作を実行することになります。

クリックされたときの動作をボタンに設定したいときは、

イベントリスナーを登録するメソッド	`addActionListener`
イベントリスナーのインターフェース	`ActionListener`
イベント処理メソッド	`actionPerformed`
イベントのオブジェクトのクラス	`ActionEvent`

という名前のインターフェースとクラスとメソッドを使います。これらは、以前、タイマーに登録するイベントリスナーを作ったときに使ったものと同じです。

次のプログラムのフレームに追加されているボタンは、クリックされると、そのことを通知するダイアログボックスを表示します。

プログラムの例 PAction.java

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

public class PAction extends JFrame {
    PAction(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.setLayout(new FlowLayout());
        JButton button = new JButton("メッセージの表示");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null,
                    "ボタンがクリックされました。",
                    "クリックのお知らせ",
                    JOptionPane.INFORMATION_MESSAGE);
            }
        });
        content.add(button);
        setVisible(true);
    }

    public static void main(String[] args) {
        PAction frame = new PAction("ボタンの動作");
    }
}
```

4.2 レイアウト

4.2.1 ふたたびレイアウトマネージャーについて

前の節でも簡単に説明しましたが、Java では、コンテナの上でのコンポーネントのレイアウト（配置）は、「レイアウトマネージャー」(layout manager) と呼ばれるオブジェクトによって管理されます。

レイアウトマネージャーにはさまざまな種類のものがあって、それぞれ、コンポーネントをレイアウトする方法に関して独自の個性を持っています。ですから、複数のコンポーネントから構成されるプログラムを書く場合には、目的に応じた適切なレイアウトマネージャーを選択することが大切です。前の節では、レイアウトマネージャーとしてフローレイアウトというものを紹介

しましたが、この節では、それ以外のレイアウトマネージャーも紹介していきたいと思います。

コンテナには、あらかじめデフォルトのレイアウトマネージャーが設定されています。それとは異なるレイアウトマネージャーを設定したいときは、コンテナが持っている setLayout というメソッドを使います。このメソッドは、引数としてレイアウトマネージャーを受け取って、それをレシーバーに設定します。

4.2.2 ボーダーレイアウト

フレームが持っているコンテントペインというコンテナには、デフォルトのレイアウトマネージャーとして「ボーダーレイアウト」(border layout) と呼ばれるものが設定されています。ちなみに、ボーダーレイアウトは、 BorderLayout というクラスから生成されるオブジェクトです。

ボーダーレイアウトは、コンテナを五つの領域に区切ります。五つの領域というのは、上端、下端、左端、右端、そして中央です。それぞれの領域には、コンポーネントを最大 1 個まで配置することができます。

ボーダーレイアウトを使ってコンポーネントをコンテナに追加するときには、「レイアウト制約」(layout constraints) と呼ばれる、レイアウトを決定する上での制約を指定する必要があります。

コンポーネントをコンテナに追加するときには、コンテナが持っている add というメソッドを使うわけですが、 add には、引数としてコンポーネントだけを受け取るもののほかに、二つ目の引数としてレイアウト制約をあらわすオブジェクトを受け取るものもあります。ボーダーレイアウトを使ってコンポーネントをコンテナに追加するときは、レイアウト制約をあらわすオブジェクトを受け取る add を呼び出します。

ボーダーレイアウトのレイアウト制約というのは、五つの領域のうちのどこに配置するかということです。それらの領域をあらわすオブジェクトは、 BorderLayout クラスの中で定数として定義されています。上端は NORTH、下端は SOUTH、左端は WEST、右端は EAST、中央は CENTER です。たとえば、 container という変数がコンテナを、 component という変数がコンポーネントを指し示しているとするとき、

```
container.add(component, BorderLayout.EAST);
```

という式文を実行することによって、 component を container の右端に配置することができます。

プログラムの例 PBorder.java

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

public class PBorder extends JFrame {
    PBorder(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.add(new JButton("北"), BorderLayout.NORTH);
        content.add(new JButton("南"), BorderLayout.SOUTH);
        content.add(new JButton("東"), BorderLayout.EAST);
        content.add(new JButton("西"), BorderLayout.WEST);
        content.add(new JButton("中央"), BorderLayout.CENTER);
        setVisible(true);
    }

    public static void main(String[] args) {
        PBorder frame = new PBorder("ボーダーレイアウト");
    }
}
```

4.2.3 グリッドレイアウト

次に、「グリッドレイアウト」(grid layout) と呼ばれるレイアウトマネージャーを紹介しましょう。このレイアウトマネージャーは、コンテナを、碁盤の升目のように、縦方向と横方向に整然と並べられた等しい大きさを持つ長方形の領域に分割します。そして、それぞれの領域にひとつのコンポーネントを配置します。

グリッドレイアウトによるコンポーネントの配置は、コンポーネントがコンテナに追加される

順番に依存します。グリッドレイアウトはコンポーネントを、まず一番上の行に配置していきます。そしてその行がすべて埋まったら、次はひとつ下の行に配置していきます。

グリッドレイアウトは、GridLayout というクラスから生成されます。コンストラクタは、普通、2 個の整数を引数として受け取るものを使います。それらの引数は、1 個目が行数で 2 個目が列数です。たとえば、

```
new GridLayout(7, 5)
```

という式でグリッドレイアウトを生成したとすると、それは、コンポーネントを 7 行 5 列に配置することになります。

プログラムの例 PGrid.java

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

public class PGrid extends JFrame {
    PGrid(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.setLayout(new GridLayout(4, 5));
        for (int i = 1; i <= 18; i++) {
            content.add(new JButton(Integer.toString(i)));
        }
        setVisible(true);
    }

    public static void main(String[] args) {
        PGrid frame = new PGrid("グリッドレイアウト");
    }
}
```

4.2.4 パネル

レイアウトマネージャーには、これまでに説明したもののほかにもさまざまな種類があります。しかし、それらの種類のうちに万能のものは存在しません。ですから、どの種類のレイアウトマネージャーを選んでも望みどおりにレイアウトすることができない、ということもしばしばあります。しかし、そのような問題の多くは、複数のレイアウトマネージャーを組み合わせて使うことによって解決することができます。

コンテナというのはコンポーネントを入れることのできるコンポーネントのことです。コンテナもコンポーネントの一種ですから、コンテナにコンテナを追加するということも可能です。複数のレイアウトマネージャーを組み合わせて使うというのは、言い換えれば、異なるレイアウトマネージャーが設定された複数のコンテナをひとつのコンテナに追加するということです。

コンテナに追加するコンテナとしては、普通、「パネル」(panel) と呼ばれるものが使われます。パネルは、コンテナとしての機能以外の余分な機能を持たないシンプルなコンテナです。

パネルは、JPanel というクラスから生成されるオブジェクトです。パネルを生成するときは、普通、引数としてレイアウトマネージャーを受け取るコンストラクタを使います。そのコンストラクタは、引数として受け取ったレイアウトマネージャーを、生成されたパネルに設定します。たとえば、

```
new JPanel(new BorderLayout())
```

という式でパネルを生成したとすると、コンストラクタは、引数として受け取ったボーダーレイアウトをそのパネルに設定します。

次のプログラムは、フレームの左右に二つのパネルを並べて、左のパネルには 1 行 4 列のグリッドレイアウトを設定して、右のパネルには 4 行 1 列のグリッドレイアウトを設定しています。

プログラムの例 PPanel.java

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

public class PPanel extends JFrame {
    PPanel(String title) {
```

```
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.setLayout(new GridLayout(1, 2));
        JPanel left  = new JPanel(new GridLayout(1, 4));
        for (int i = 1; i <= 4; i++) {
            left.add(new JButton(Integer.toString(i)));
        }
        content.add(left);
        JPanel right = new JPanel(new GridLayout(4, 1));
        for (int i = 5; i <= 8; i++) {
            right.add(new JButton(Integer.toString(i)));
        }
        content.add(right);
        setVisible(true);
    }

    public static void main(String[] args) {
        PPanel frame = new PPanel("パネル");
    }
}
```

4.3 トグルボタン

4.3.1 トグルボタンとは何か

「チェックボックス」(check box) と「ラジオボタン」(radio button) は、どちらも、選択されているかまたは選択されていないかという二つの状態のどちらかになっているという共通点を持っていますので、総称して「トグルボタン」(toggle button) と呼ばれます。

4.3.2 トグルボタンの生成

Swing では、チェックボックスは JCheckBox というクラスから生成されるコンポーネントで、ラジオボタンは JRadioButton というクラスから生成されるコンポーネントです。

トグルボタンのコンストラクタのうちで、引数として 1 個の文字列を受け取るものは、その引数を、生成されたトグルボタンの右側に表示する文字列として設定します。たとえば、

```
new JCheckBox("豪華なオプション")
```

という式を評価することによって、「豪華なオプション」という文字列が右側に表示されたチェックボックスを生成することができます。

引数として 1 個の文字列を受け取るトグルボタンのコンストラクタは、生成されたトグルボタンを選択されていない状態に初期化します。トグルボタンを生成するときに、最初から選択されている状態にしたいときは、

```
new JRadioButton("庭付き一戸建て", true)
```

というように、2 個の引数を受け取るコンストラクタを使います。このコンストラクタは、2 個目の引数が真ならば、生成されたトグルボタンを選択された状態に初期化します。

4.3.3 トグルボタンの状態

トグルボタンの状態を調べたいときは、トグルボタンが持っている isSelected というメソッドを使います。このメソッドは、レシーバーが選択されている状態ならば真を返し、そうでなければ偽を返します。

トグルボタンの状態は、普通は人間の操作によって変更されるわけですが、場合によっては、プログラム自身が状態を変更することが必要になるかもしれません。そのような場合は、トグルボタンが持っている setSelected というメソッドを使います。このメソッドは、引数として真偽値を受け取って、それが真ならばレシーバーを選択された状態にして、偽ならば選択を解除します。

プログラムの例 PCheckBox.java

```
import javax.swing.*;
```

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

public class PCheckBox extends JFrame {
    JCheckBox check;

    PCheckBox(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.setLayout(new FlowLayout());
        check = new JCheckBox("顔文字を付ける。");
        content.add(check);
        JButton button = new JButton("謝罪");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String apology = "ごめんなさい。";
                if (check.isSelected())
                    apology += "(^^ゞ";
                JOptionPane.showMessageDialog(null,
                    apology, "謝罪",
                    JOptionPane.INFORMATION_MESSAGE);
            }
        });
        content.add(button);
        setVisible(true);
    }

    public static void main(String[] args) {
        PCheckBox frame = new PCheckBox("チェックボックス");
    }
}
```

4.3.4 ボタングループ

ラジオボタンというのは、いくつかの選択肢の中からひとつだけを選択するために使われるトグルボタンですので、普通、複数のものが集まってひとつのグループを形成します。ひとつのグループを形成するそれぞれのラジオボタンは、選択されているものがひとつだけになるように、状態の変更を互いに連動させる必要があります。

Swing では、そのような択一的な選択を実現するために、「ボタングループ」(button group) と呼ばれるオブジェクトを使います。ボタングループは、複数のボタンを格納することのできる容器になっていて、それに格納されたボタンは、常にひとつだけが選択されている状態になるように管理されます。

ボタングループは、 ButtonGroup というクラスから生成されます。このクラスのコンストラクタは引数を受け取りませんので、

```
new ButtonGroup()
```

という式を評価することによって、ボタングループを生成することができます。

ボタングループにボタンを追加したいときは、ボタングループが持っている add というメソッドを使います。このメソッドは、引数としてボタンを受け取って、それをレシーバーに追加します。たとえば、 group がボタングループを、 button がボタンを指し示しているとするならば、

```
group.add(button);
```

という式文を実行することによって、 button を group に追加することができます。

プログラムの例 PRadioButton.java

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

public class PRadioButton extends JFrame {
    final static int RADIONUM = 3;
```

```
    JRadioButton[] radio;

    PRadioButton(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.setLayout(new FlowLayout());
        ButtonGroup group = new ButtonGroup();
        radio = new JRadioButton[RADIONUM];
        radio[0] = new JRadioButton("朝", true);
        radio[1] = new JRadioButton("昼");
        radio[2] = new JRadioButton("夜");
        for (int i = 0; i < RADIONUM; i++) {
            group.add(radio[i]);
            content.add(radio[i]);
        }
        JButton button = new JButton("挨拶");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int num = 0;
                boolean notfound = true;
                for (int i = 0;
                     notfound && i < RADIONUM; i++) {
                    if (radio[i].isSelected()) {
                        num = i;
                        notfound = false;
                    }
                }
                String salute = "";
                switch (num) {
                case 0:
                    salute = "おはようございます。";
                    break;
                case 1:
                    salute = "こんにちは。";
                    break;
                case 2:
                    salute = "こんばんは。";
                    break;
                }
                JOptionPane.showMessageDialog(null,
                    salute, "挨拶",
                    JOptionPane.INFORMATION_MESSAGE);
            }
        });
        content.add(button);
        setVisible(true);
    }

    public static void main(String[] args) {
        PRadioButton frame = new PRadioButton("ラジオボタン");
    }
}
```

4.4　メニュー

4.4.1　メニューバー

フレームは、「メニューバー」(menu bar) と呼ばれる細長いコンポーネントをタイトルバーの下に表示することができます。メニューバーはコンテナの一種ですが、少し特殊なコンテナで、それに追加することができるコンポーネントはメニューだけに限定されています。

Swing では、メニューバーは JMenuBar というクラスから生成されます。このクラスのコンストラクタは引数を受け取りませんので、

```
new JMenuBar()
```

という式を評価することによって、メニューバーを生成することができます。

フレームにコンポーネントを追加するためには、フレームから取得したコンテントペインにそれを追加しないといけないわけですが、メニューバーは例外です。フレームにメニューバーを追加したいときは、フレームが持っている setJMenuBar というメソッドを使います。このメソッドは、引数としてメニューバーを受け取って、それをレシーバーに追加します。たとえば、frame がフレームを、menuBar がメニューバーを指し示しているとするならば、

```
frame.setJMenuBar(menuBar);
```

という式文を実行することによって、menuBar を frame に追加することができます。

4.4.2 メニュー

Swing では、メニューは、JMenu というクラスから生成されるコンポーネントです。このクラスのコンストラクタのうち、引数として 1 個の文字列を受け取るものは、その引数を、メニューバーの上に表示する文字列として設定します。たとえば、

```
new JMenu("ファイル")
```

という式でメニューを生成して、それをメニューバーに追加したとすると、「ファイル」という文字列がメニューバーの上に表示されることになります。

メニューをメニューバーに追加したいときは、メニューバーが持っている add というメソッドを使います。たとえば、menuBar がメニューバーを、menu がメニューを指し示しているとするならば、

```
menuBar.add(menu);
```

という式文を実行することによって、menu を menuBar に追加することができます。

4.4.3 メニューアイテム

メニューは、「メニューアイテム」(menu item) と呼ばれるコンポーネントから構成されます。Swing は、メニューアイテムを生成するクラスとして、

JMenuItem	普通のメニューアイテム
JCheckBoxMenuItem	チェックボックス付きメニューアイテム
JRadioButtonMenuItem	ラジオボタン付きメニューアイテム

という三つのクラスを提供しています。これらのクラスのコンストラクタのうち、引数として 1 個の文字列を受け取るものは、その引数を、メニューアイテムの上に表示する文字列として設定します。たとえば、

```
new JMenuItem("上書き保存")
```

という式でメニューアイテムを生成したとすると、そのメニューアイテムの上に「上書き保存」という文字列が表示されることになります。

メニューアイテムをメニューに追加したいときは、メニューが持っている add というメソッドを使います。たとえば、menu がメニューを、item がメニューアイテムを指し示しているとするならば、

```
menu.add(item);
```

という式文を実行することによって、item を menu に追加することができます。

メニューアイテムは、普通、選択されたときに何らかの動作を実行します。メニューアイテムはボタンの一種ですので、プッシュボタンの場合とまったく同じ方法でイベントリスナーを作って、やはり同じ方法でそれをメニューアイテムに登録すれば、メニューアイテムが選択されたときに、そのイベントリスナーの中のイベント処理メソッドが呼び出されることになります。

4.4.4 セパレーター

上下のメニューアイテムのあいだに「セパレーター」(separator) と呼ばれる仕切りを挿入したいときは、メニューが持っている addSeparator というメソッドを使います。たとえば、menu がメニューを指し示しているとするならば、

```
menu.addSeparator();
```

という式文を実行することによって、セパレーターを menu に追加することができます。

4.4.5 サブメニュー

メニューには、メニューアイテムだけではなくて、メニューを追加することもできます。つまり、メニューはいくらでも入れ子にすることができるということです。メニューに追加されたメニューは、「サブメニュー」(submenu) と呼ばれます。サブメニューを持つメニューは、サブメニューをあらわす文字列だけを自分の上に表示して、それが選択されたときに、その右側にサブメニューを表示します。

プログラムの例 PMenu.java

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

public class PMenu extends JFrame {
    PMenu(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);
        JMenu food = new JMenu("料理");
        menuBar.add(food);
        JMenuItem curry = new JMenuItem("カレー");
        curry.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showOrder("カレー");
            }
        });
        food.add(curry);
        JMenu teishoku = new JMenu("定食");
        food.add(teishoku);
        JMenuItem tonkatsu = new JMenuItem("とんかつ定食");
        tonkatsu.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showOrder("とんかつ定食");
            }
        });
        teishoku.add(tonkatsu);
        JMenuItem yakiniku = new JMenuItem("焼肉定食");
        yakiniku.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showOrder("焼肉定食");
            }
        });
        teishoku.add(yakiniku);
        JMenu drink = new JMenu("飲み物");
        menuBar.add(drink);
        JMenuItem oolong = new JMenuItem("烏龍茶");
        oolong.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showOrder("烏龍茶");
            }
        });
        drink.add(oolong);
        JMenuItem coffee = new JMenuItem("コーヒー");
        coffee.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showOrder("コーヒー");
            }
        });
        drink.add(coffee);
        drink.addSeparator();
        JMenuItem water = new JMenuItem("水");
        water.addActionListener(new ActionListener() {
```

```
            public void actionPerformed(ActionEvent e) {
                showOrder("水");
            }
        });
        drink.add(water);
        setVisible(true);
    }

    void showOrder(String order) {
        JOptionPane.showMessageDialog(null,
            order + "が注文されました。", "注文の確認",
            JOptionPane.INFORMATION_MESSAGE);
    }

    public static void main(String[] args) {
        PMenu frame = new PMenu("メニュー");
    }
}
```

4.4.6 無効なメニューアイテム

メニューアイテムは、選択できないように無効にしておく、ということができます。無効になっているメニューアイテムは、自分の上の文字列を薄い色で表示します。

メニューアイテムを有効にしたいときや無効にしたいときは、それが持っている `setEnabled` というメソッドを呼び出します。このメソッドは、引数として真偽値を受け取って、それが真ならばレシーバーを有効にして、偽ならば無効にします。

プログラムの例 `PEnabled.java`

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

public class PEnabled extends JFrame {
    JCheckBoxMenuItem toggle;
    JMenuItem item;

    PEnabled(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);
        JMenu menu = new JMenu("メニュー");
        menuBar.add(menu);
        toggle = new JCheckBoxMenuItem(
            "下のメニューアイテムを有効にする");
        toggle.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (toggle.isSelected())
                    item.setEnabled(true);
                else
                    item.setEnabled(false);
            }
        });
        menu.add(toggle);
        item = new JMenuItem(
            "有効または無効なメニューアイテム");
        item.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null,
                    "メニューアイテムがクリックされました。",
                    "イベントの報告",
                    JOptionPane.INFORMATION_MESSAGE);            }
        });
        item.setEnabled(false);
        menu.add(item);
```

```
        setVisible(true);
    }

    public static void main(String[] args) {
        PEnabled frame = new PEnabled(
            "無効なメニューアイテム");
    }
}
```

4.4.7 ニーモニック

メニューやメニューアイテムには、「ニーモニック」(mnemonic) または「ショートカット」(short cut) と呼ばれるキーを設定しておくことができます。そうしておくと、Alt キーを押しながらニーモニックを押すことによって、そのニーモニックが設定されているメニューを選択することができるようになります。そして、メニューが選択されている状態では、ニーモニックのみを押すことによって、それが設定されているサブメニューやメニューアイテムを選択することができます。

メニューやメニューアイテムに対してニーモニックを設定したいときは、ボタンが持っている setMnemonic というメソッドを呼び出します。このメソッドは、引数としてキーコードを受け取って、そのキーコードによって指定されたキーをニーモニックとしてレシーバーに設定します。たとえば、item がメニューアイテムを指し示しているとするとき、

```
item.setMnemonic(KeyEvent.VK_M);
```

という式文を実行すると、そのメニューアイテムに対して、M のキーがニーモニックとして設定されます。

4.4.8 アクセラレーター

メニューアイテムに対しては、ニーモニックとは別に、「アクセラレーター」(accelerator) と呼ばれるキーの組み合わせを設定しておくこともできます。ニーモニックでメニューを操作する場合は、メニューの階層を順番にたどっていく必要があるわけですが、それに対して、アクセラレーターは、それ一発でメニューアイテムを選択することが可能です。

メニューアイテムに対してアクセラレーターを設定したいときは、メニューアイテムが持っている setAccelerator というメソッドを呼び出します。このメソッドは、KeyStroke というクラスのオブジェクトを引数として受け取って、それによって指定されたキーの組み合わせをアクセラレーターとしてレシーバーに設定します。

KeyStroke クラスのオブジェクトは、new を使って生成するのではなくて、そのクラスが持っている getKeyStroke というメソッドを呼び出すことによって取得します。このメソッドは、2 個の引数を受け取ります。1 個目はキーコードで、2 個目は、「修飾子」(modifier) と呼ばれる整数です。

修飾子は、どのキーを押しながらキーを押すのか、ということを指定するための整数です。普通、アクセラレーターとしてはコントロールキーと何らかのキーとを組み合わせたものを使いますので、

```
InputEvent.CTRL_MASK
```

という定数で、コントロールキーを指定します。たとえば、item がメニューアイテムを指し示しているとするとき、

```
item.setAccelerator(
    KeyStroke.getKeyStroke(KeyEvent.VK_A,
                           InputEvent.CTRL_MASK));
```

という式文を実行すると、そのメニューアイテムに対して、コントロールキーと A のキーとの組み合わせがアクセラレーターとして設定されます。

プログラムの例 PMnemonic.java

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

public class PMnemonic extends JFrame {
    PMnemonic(String title) {
        super(title);
```

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);
        JMenu food = new JMenu("料理 (F)");
        food.setMnemonic(KeyEvent.VK_F);
        menuBar.add(food);
        JMenuItem chuudon = new JMenuItem("中華丼 (C)");
        chuudon.setMnemonic(KeyEvent.VK_C);
        chuudon.setAccelerator(
            KeyStroke.getKeyStroke(KeyEvent.VK_C,
                                   InputEvent.CTRL_MASK));
        chuudon.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showOrder("中華丼");
            }
        });
        food.add(chuudon);
        JMenuItem yakisoba = new JMenuItem("焼きそば (Y)");
        yakisoba.setMnemonic(KeyEvent.VK_Y);
        yakisoba.setAccelerator(
            KeyStroke.getKeyStroke(KeyEvent.VK_Y,
                                   InputEvent.CTRL_MASK));
        yakisoba.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showOrder("焼きそば");
            }
        });
        food.add(yakisoba);
        setVisible(true);
    }

    void showOrder(String order) {
        JOptionPane.showMessageDialog(null,
            order + "が注文されました。", "注文の確認",
            JOptionPane.INFORMATION_MESSAGE);
    }

    public static void main(String[] args) {
        PMnemonic frame = new PMnemonic(
            "ニーモニックとアクセラレーター");
    }
}
```

4.5 ポップアップメニュー

4.5.1 ポップアップメニューとは何か

メニューというのは、普通、ウィンドウの上に固定されているメニューバーをクリックしたときに表示されるわけですが、そのような普通のメニューだけではなくて、表示される位置が固定されていないメニューというのも、しばしば有用です。そのような、任意の位置に表示することのできるメニューは、「ポップアップメニュー」(popup menu) と呼ばれます。

Swing では、JPopupMenu というクラスを使うことによって、ポップアップメニューを生成することができます。ポップアップメニューは、普通、引数を受け取らないコンストラクタを使って生成します。

ポップアップメニューはコンテナの一種ですので、add というメソッドを持っています。ですから、それを呼び出すことによって、コンポーネントをポップアップメニューに追加することができます。ただし、ポップアップメニューに追加することのできるコンポーネントは、メニューまたはメニューアイテムに限られます。

4.5.2 ポップアップメニューの表示

ポップアップメニューを表示したいときは、それが持っている show というメソッドを呼び出します。このメソッドは、3 個の引数を受け取ります。

1 個目の引数はポップアップメニューをその上に表示するコンポーネントです。この引数としては、普通、コンポーネントの上で発生したイベントをあらわしているオブジェクトが持っている getComponent というメソッドの戻り値をそのまま渡します。

show に渡す 2 個目と 3 個目は、ポップアップメニューを表示する位置の x 座標と y 座標です。

プログラムの例　PPopupMenu.java

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

public class PPopupMenu extends JFrame {
    JPopupMenu popup;

    PPopupMenu(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        createPopupMenu();
        setVisible(true);
        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                Point p = e.getPoint();
                popup.show(e.getComponent(), p.x, p.y);
            }
        });
    }

    void showOrder(String order) {
        JOptionPane.showMessageDialog(null,
            order + "が注文されました。", "注文の確認",
            JOptionPane.INFORMATION_MESSAGE);
    }

    void createPopupMenu() {
        popup = new JPopupMenu();
        JMenuItem yakisoba = new JMenuItem("焼きそば");
        yakisoba.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showOrder("焼きそば");
            }
        });
        popup.add(yakisoba);
        JMenuItem chuudon = new JMenuItem("中華丼");
        chuudon.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showOrder("中華丼");
            }
        });
        popup.add(chuudon);
        JMenuItem nirareba = new JMenuItem("ニラレバ");
        nirareba.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showOrder("ニラレバ");
            }
        });
        popup.add(nirareba);
    }

    public static void main(String[] args) {
        PPopupMenu frame = new PPopupMenu(
            "ポップアップメニュー");
    }
}
```

第5章 テキスト

5.1 ラベル

5.1.1 テキストとは何か

プログラミングに関連する文脈では、しばしば、「文字列」(string) のことを「テキスト」(text) と呼ぶことがあります。これらの二つの言葉は、ニュアンスは多少違いますが、ほとんど同じ意味だと考えていいでしょう。

Swing のコンポーネントの多くは、テキストを取り扱うことができます。たとえば、前の章で紹介したボタンは、テキストを自分の上や右側に表示することができます。この章では、Swing のコンポーネントのうちで、ボタンよりもさらに密接にテキストに関連するコンポーネントを紹介していきたいと思います。

5.1.2 ラベルの基礎

まず最初に紹介するのは、「ラベル」(label) と呼ばれるコンポーネントです。ラベルは、テキストを表示することを第一の目的とするコンポーネントです。

Swing のラベルは、 JLabel というクラスのオブジェクトです。このクラスのコンストラクタのうち、引数として 1 個の文字列を受け取るものは、受け取った文字列を、ラベルの上に表示されるテキストとして設定します。

プログラムの例 PLabel.java

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

public class PLabel extends JFrame {
    PLabel(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.add(new JLabel("私はラベルです。"),
                 BorderLayout.CENTER);
        setVisible(true);
    }

    public static void main(String[] args) {
        PLabel frame = new PLabel("ラベル");
    }
}
```

5.1.3 テキストの設定

ラベルの上に表示されるテキストの設定は、ラベルを生成するときだけではなくて、それ以降に実行することも可能です。すでに生成されているラベルにテキストを設定したいときは、ラベルが持っている setText というメソッドを使います。このメソッドに引数として文字列を渡すと、その文字列がラベルの上に表示されます。

プログラムの例 PSetText.java

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

public class PSetText extends JFrame {
    JLabel label;

    PSetText(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        label = new JLabel("デフォルトのテキスト");
        content.add(label, BorderLayout.CENTER);
```

```
        JButton button = new JButton("テキストの変更");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String text = JOptionPane.showInputDialog(
                    null, "テキストを入力してください。",
                    "入力", JOptionPane.PLAIN_MESSAGE);
                if (text != null)
                    label.setText(text);
            }
        });
        content.add(button, BorderLayout.SOUTH);
        setVisible(true);
    }

    public static void main(String[] args) {
        PSetText frame = new PSetText("テキストの設定");
    }
}
```

5.2 HTML

5.2.1 コンポーネントの HTML 表示機能

ラベルやボタンのような、自分の上にテキストを表示することのできる Swing のコンポーネントは、HTML でマークアップされた文字列を解釈して、その結果を表示する、という機能を持っています。

HTML でマークアップされた文字列をコンポーネントに表示させたいときは、コンポーネントを生成するときに HTML の文字列をコンストラクタに渡すか、または、 `setText` というメソッドを呼び出して、それに HTML の文字列を渡します。

プログラムの例 `PHTML.java`

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

public class PHTML extends JFrame {
    PHTML(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.add(new JLabel(
            "<html><head></head><body><table border=\"1\">" +
            "<tr><th>名前</th><th>誕生日</th></tr>" +
            "<tr><td>松子</td><td>8 月 14 日</td></tr>" +
            "<tr><td>竹之助</td><td>12 月 26 日</td></tr>" +
            "</table></body></html>"),
            BorderLayout.CENTER);
        setVisible(true);
    }

    public static void main(String[] args) {
        PHTML frame = new PHTML(
            "コンポーネントの HTML 表示機能");
    }
}
```

5.2.2 スタイルシート

HTML の文字列を解釈して表示することのできる Swing のコンポーネントは、CSS で記述されたスタイルシートを解釈して、それを表示に適用する、ということもできます。

HTML の要素の多くは、 `style` という属性を持っています。この属性に、値としてスタイルシートを設定すると、それが要素の表示に適用されます。

また、 `head` 要素の中に `style` という要素を書いて、その内容としてスタイルシートを書くこ

とも可能です。そうすることによって、特定の要素型の要素や、特定のクラスの要素や、特定の識別子を持つ要素に対して、スタイルシートに記述されたスタイルを適用することができます。

プログラムの例 PStylesheet.java

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

public class PStylesheet extends JFrame {
    PStylesheet(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.add(new JLabel(
            "<html><head><style type=\"text/css\">" +
            "p { font-size: 16pt; color: #000099 }" +
            ".emphasis { font-size: 32pt; color: #ff0099; " +
            "    background-color: #ffffcc }" +
            ".border { border-style: solid; " +
            "    border-width: thick }" +
            "</style></head><body>" +
            "<p>普通の段落</p>" +
            "<p class=\"emphasis\">強調された段落</p>" +
            "<p class=\"border\">枠線で囲まれた段落</p>" +
            "</body></html>"),
            BorderLayout.CENTER);
        setVisible(true);
    }

    public static void main(String[] args) {
        PStylesheet frame = new PStylesheet("スタイルシート");
    }
}
```

5.3 テキストフィールド

5.3.1 テキストフィールドの基礎

キーボードから 1 行のテキストを読み込むことができるコンポーネントは、「テキストフィールド」(text field) と呼ばれます。Swing では、テキストフィールドは JTextField というクラスから生成されるオブジェクトです。

入力されたテキストをテキストフィールドから取り出したいときは、テキストフィールドが持っている getText というメソッドを使います。このメソッドは、レシーバーから取り出したテキストを戻り値として返します。また、ラベルと同じように、 setText というメソッドを使うことによって、テキストフィールドにテキストを設定することもできます。

キーボードからテキストを読み込むコンポーネントは、ひとつのコンピュータの画面の上にいくつでも表示することができます。しかし、コンピュータに接続されているキーボードは普通はひとつだけですから、ひとつの時点では、キーボードからの入力を受け付けることのできるコンポーネントはひとつだけです。

コンポーネントがキーボードからの入力を受け付けることのできる状態にあることを、そのコンポーネントに「フォーカス」(focus) があると言います。コンポーネントにフォーカスを設定したいときは、そのコンポーネントが持っている requestFocusInWindow というメソッドを呼び出します。

プログラムの例 PTextField.java

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

public class PTextField extends JFrame {
    JTextField field;
    JLabel label;
```

```
    PTextField(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        field = new JTextField();
        content.add(field, BorderLayout.NORTH);
        label = new JLabel("デフォルトのテキスト");
        content.add(label, BorderLayout.CENTER);
        JButton button = new JButton("テキストの転送");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                label.setText(field.getText());
                field.requestFocusInWindow();
            }
        });
        content.add(button, BorderLayout.SOUTH);
        field.requestFocusInWindow();
        setVisible(true);
    }

    public static void main(String[] args) {
        PTextField frame = new PTextField(
            "テキストフィールド");
    }
}
```

5.3.2 エンターキーによるイベント

テキストフィールドには、エンターキーが押されたときに何らかの動作を実行するイベントリスナーを登録しておくことができます。そのようなイベントリスナーの作り方は、ボタンに登録するイベントリスナーの作り方と同じです。つまり、ActionListenerというインターフェースを実装するクラスを作って、その中でactionPerformedというメソッドを定義すればいいわけです。

プログラムの例 PEnterKey.java

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

public class PEnterKey extends JFrame {
    JTextField field;
    JLabel label;

    PEnterKey(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        field = new JTextField();
        field.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                label.setText(field.getText());
                field.requestFocusInWindow();
            }
        });
        content.add(field, BorderLayout.NORTH);
        label = new JLabel("デフォルトのテキスト");
        content.add(label, BorderLayout.CENTER);
        field.requestFocusInWindow();
        setVisible(true);
    }

    public static void main(String[] args) {
        PEnterKey frame = new PEnterKey(
            "エンターキーによるイベント");
```

```
    }
}
```

5.4 テキストエリア

5.4.1 テキストエリアの基礎

キーボードから複数行のテキストを読み込むことができるコンポーネントは、「テキストエリア」(text area) と呼ばれます。Swing では、テキストエリアは JTextArea というクラスから生成されるオブジェクトです。

テキストエリアでも、getText、setText、requestFocusInWindow などのメソッドを、テキストフィールドと同じように使うことができます。

デフォルトの状態のテキストエリアは、自分の横幅よりも長い行を、折り返さないで一部分だけ表示する設定になっています。行を折り返す設定にしたいときは、テキストエリアが持っている setLineWrap というメソッドを呼び出します。このメソッドは、引数として真偽値を受け取って、それが真ならば行を折り返す設定にして、偽ならば折り返さない設定にします。

プログラムの例　PTextArea.java

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

public class PTextArea extends JFrame {
    JTextArea area;

    PTextArea(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        area = new JTextArea();
        area.setLineWrap(true);
        content.add(area, BorderLayout.CENTER);
        JButton button = new JButton(
          "ダイアログボックスで表示");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null,
                    area.getText(), "入力されたテキスト",
                    JOptionPane.INFORMATION_MESSAGE);
                area.requestFocusInWindow();
            }
        });
        content.add(button, BorderLayout.SOUTH);
        area.requestFocusInWindow();
        setVisible(true);
    }

    public static void main(String[] args) {
        PTextArea frame = new PTextArea("テキストエリア");
    }
}
```

5.4.2 スクロールペイン

コンポーネントは、画面上でのその大きさよりも大きなテキストや画像などを表示することができます。そのような場合は、表示されている部分を移動させることによって、見えていなかった部分を見えるようにすることができます。

コンポーネントによって表示されている部分を移動させることを、そのコンポーネントを「スクロールさせる」(scroll) と言います。コンポーネントのスクロールには、普通、「スクロールバー」(scroll bar) と呼ばれるコンポーネントが使われます。

Swing では、コンポーネントにスクロールバーを取り付けたいときは、「スクロールペイン」

(scroll pane) と呼ばれるものを使います。スクロールペインというのは、スクロールバーそのものではなくて、スクロールバーが取り付けられたコンテナです。スクロールペインにコンポーネントを追加すると、そのコンポーネントは、スクロールペインが持っているスクロールバーによってスクロールさせることができるようになります（ただし、スクロールの必要がない場合、スクロールバーは表示されません)。

スクロールペインは、JScrollPane というクラスから生成されます。コンストラクタとしては、普通、引数として 1 個のコンポーネントを受け取るものを使います。そのコンストラクタは、引数として受け取ったコンポーネントをスクロールペインに追加します。

たとえば、comp という変数がコンポーネントを指し示しているとするとき、

```
new JScrollPane(comp)
```

という式を評価することによって、comp をスクロールさせるためのスクロールペインを生成することができます。

プログラムの例 PScrollPane.java

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

public class PScrollPane extends JFrame {
    PScrollPane(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JTextArea area = new JTextArea();
        JScrollPane scroll = new JScrollPane(area);
        content.add(scroll, BorderLayout.CENTER);
        area.requestFocusInWindow();
        setVisible(true);
    }

    public static void main(String[] args) {
        PScrollPane frame = new PScrollPane(
            "スクロールペイン");
    }
}
```

5.4.3 クリップボード

テキストを編集するときに使われる、テキストを一時的に格納するためのメモリー上の領域は、「クリップボード」(clip board) と呼ばれます。

テキストエリアは、クリップボードを操作する、cut、copy、paste というメソッドを持っています。cut と copy は、テキストエリアの中で選択されているテキストをクリップボードに入れるメソッドです。テキストをクリップボードに入れたのち、cut はそのテキストをテキストエリアから削除しますが、copy はそのまま残します。そして、paste は、クリップボードに格納されているテキストをキャレットの位置に挿入するメソッドです。

次のプログラムは、テキストの切り取り、コピー、貼り付けの機能を持つ、単純なテキストエディターです。

プログラムの例 PEditor.java

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

public class PEditor extends JFrame {
    JTextArea area;
    String path;

    PEditor(String title, String apath) {
        super(title);
        path = apath;
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        area = new JTextArea();
        area.setLineWrap(true);
        JScrollPane scroll = new JScrollPane(area);
        content.add(scroll, BorderLayout.CENTER);
        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);
        JMenu mfile = new JMenu("ファイル");
        menuBar.add(mfile);
        JMenuItem isave = new JMenuItem("保存");
        isave.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                save();
            }
        });
        mfile.add(isave);
        JMenu medit = new JMenu("編集");
        menuBar.add(medit);
        JMenuItem icut = new JMenuItem("切り取り");
        icut.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                area.cut();
            }
        });
        medit.add(icut);
        JMenuItem icopy = new JMenuItem("コピー");
        icopy.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                area.copy();
            }
        });
        medit.add(icopy);
        JMenuItem ipaste = new JMenuItem("貼り付け");
        ipaste.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                area.paste();
            }
        });
        medit.add(ipaste);
        load();
        area.requestFocusInWindow();
        setVisible(true);
    }

    void showError(String message) {
        JOptionPane.showMessageDialog(null, message, "エラー",
            JOptionPane.ERROR_MESSAGE);
    }

    void load() {
        try {
            BufferedReader reader = new BufferedReader(
                new FileReader(path));
            String buffer = "";
            String line;
            while ((line = reader.readLine()) != null)
                buffer += line + '\n';
            reader.close();
            area.setText(buffer);
        } catch (IOException e) {
            showError(e.getMessage());
        }
    }

    void save() {
        try {
```

```
            BufferedWriter writer = new BufferedWriter(
                new FileWriter(path));
            String buffer = area.getText();
            for (int i = 0; i < buffer.length(); i++)
                if (buffer.charAt(i) == '\n')
                    writer.newLine();
                else
                    writer.write((int)buffer.charAt(i));
            writer.close();
        } catch (IOException e) {
            showError(e.getMessage());
        }
    }

    public static void main(String[] args) {
        if (args.length >= 1) {
            PEditor frame = new PEditor(
                "テキストエディター", args[0]);
        } else {
            System.out.println("使い方: java PEditor パス名");
        }
    }
}
```

5.5 エディターペイン

5.5.1 エディターペインの基礎

Swing は、キーボードから複数行のテキストを読み込むことができるコンポーネントとして、テキストエリアのほかにもうひとつ、「エディターペイン」(editor pane) と呼ばれるものを持っています。エディターペインの機能は、基本的にはテキストエリアと同じですが、さらに、書式情報を持つテキストを扱うことができるという機能も備えています。

エディターペインは、JEditorPane というクラスから作られるコンポーネントです。引数を受け取らないコンストラクタを使うことによって、デフォルトの設定を持つエディターペインを生成することができます。

5.5.2 メディアタイプの設定

RFC2045 は、「メディアタイプ」(media type) と呼ばれる文字列を作るための構文を定義しています。メディアタイプというのは、データの種類をあらわす文字列のことです。

エディターペインは、setContentType というメソッドを持っています。これは、メディアタイプを引数として受け取って、それをレシーバーに設定するメソッドです。このメソッドを使ってエディターペインに設定することのできるメディアタイプとしては、次のようなものがあります。

text/plain	プレーンテキスト
text/html	HTML(hypertext markup language) の文書
text/rtf	RTF(rich text format) の文書

エディターペインが持っている getText は、入力されているテキストを、設定されているメディアタイプのデータにして返します。

プログラムの例 PEditorPane.java

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

public class PEditorPane extends JFrame {
    JEditorPane editor;

    PEditorPane(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        editor = new JEditorPane();
        editor.setContentType("text/html");
        content.add(new JScrollPane(editor),
                    BorderLayout.CENTER);
        JButton button = new JButton(
            "ダイアログボックスで表示");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null,
                    editor.getText(), "入力されたテキスト",
                    JOptionPane.INFORMATION_MESSAGE);
                editor.requestFocusInWindow();
            }
        });
        content.add(button, BorderLayout.SOUTH);
        editor.requestFocusInWindow();
        setVisible(true);
    }

    public static void main(String[] args) {
        PEditorPane frame = new PEditorPane(
            "エディターペイン");
    }
}
```

5.5.3 URL によるテキストの設定

エディターペインは、setPage というメソッドを持っています。このメソッドは、引数として URL を受け取って、それによって指定されたリソースをダウンロードして、それを編集の対象としてレシーバーに設定します。

setPage は、無効な URL が指定された場合、 IOException という例外を発生させます。

5.5.4 ハイパーリンクのイベント

エディターペインは、setEditable というメソッドを持っています。引数として偽を渡してこのメソッドを呼び出すと、レシーバーは、編集ができない状態に設定されます。

エディターペインを編集ができない状態に設定すると、その代わりとして、ハイパーリンクのイベントを発生させる機能が有効になります。ハイパーリンクのイベントというのは、エディターペインの上に表示されているハイパーリンクのアンカーをマウスで操作したときに発生するイベントのことです。

ハイパーリンクのイベントを取り扱いたいときは、 javax.swing.event というパッケージの中で定義されている、

イベントリスナーを登録するメソッド	addHyperlinkListener
イベントリスナーのインターフェース	HyperlinkListener
イベント処理メソッド	hyperlinkUpdate
イベントのオブジェクトのクラス	HyperlinkEvent

という名前のインターフェースとクラスとメソッドを使います。

ハイパーリンクのイベントは、アンカーとマウスポインターとが重なったとき、アンカーからマウスポインターが離れたとき、そしてアンカーがクリックされたときに発生します。発生したイベントがそれらのうちのどれなのかということを調べたいときは、イベントのオブジェクトが持っている getEventType というメソッドを使います。このメソッドが戻り値として返す、

```
HyperlinkEvent.EventType
```

というクラスのオブジェクトは、発生したイベントの種類をあらわしていて、次の定数の値と比較することによって、その種類を判定することができます。

HyperlinkEvent.EventType.ENTERED	アンカーとマウスポインターとが重なった
HyperlinkEvent.EventType.EXITED	アンカーからマウスポインターが離れた

HyperlinkEvent.EventType.ACTIVATED アンカーがクリックされた

ハイパーリンクのイベントのオブジェクトは、getURLというメソッドを持っています。このメソッドは、マウスによって操作されたアンカーのリンク先のURLを戻り値として返します。

getURLが返す戻り値は、URLというクラスのオブジェクトです。このオブジェクトがあらわしているURLを文字列に変換したいときは、toStringというメソッドを使います。

プログラムの例 PBrowser.java

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

public class PBrowser extends JFrame {
    JTextField urlfield;
    JEditorPane webpage;
    JLabel status;

    PBrowser(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JPanel north = new JPanel(new BorderLayout());
        urlfield = new JTextField();
        north.add(urlfield, BorderLayout.CENTER);
        JButton move = new JButton("移動");
        move.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                movePage(urlfield.getText());
            }
        });
        north.add(move, BorderLayout.EAST);
        content.add(north, BorderLayout.NORTH);
        createWebPage();
        content.add(new JScrollPane(webpage),
                    BorderLayout.CENTER);
        status = new JLabel("");
        content.add(status, BorderLayout.SOUTH);
        setVisible(true);
    }

    void hyperlink(HyperlinkEvent e) {
        HyperlinkEvent.EventType type = e.getEventType();
        String url = e.getURL().toString();
        if (type == HyperlinkEvent.EventType.ACTIVATED)
            movePage(url);
        else if (type == HyperlinkEvent.EventType.ENTERED)
            status.setText(url);
        else if (type == HyperlinkEvent.EventType.EXITED)
            status.setText("");
    }

    void createWebPage() {
        webpage = new JEditorPane();
        webpage.setEditable(false);
        webpage.setContentType("text/html");
        webpage.addHyperlinkListener(new HyperlinkListener() {
            public void hyperlinkUpdate(HyperlinkEvent e) {
                hyperlink(e);
            }
        });
    }

    void movePage(String url) {
```

```
        if (url.length() > 0) {
            try {
                webpage.setPage(url);
                urlfield.setText(url);
            } catch (IOException e) {
                JOptionPane.showMessageDialog(this,
                    "表示できません。", "エラー",
                    JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    public static void main(String[] args) {
        PBrowser frame = new PBrowser("ブラウザー");
    }
}
```

第6章　リスト

6.1　基本的なリスト

6.1.1　リストの基礎

いくつかの選択肢の中から1個以上のものを人間に選択させたいときには、「リスト」(list) と呼ばれるコンポーネントが使われます。リストは、「項目」(item) と呼ばれるオブジェクトをいくつか表示して、マウスやキーボードの操作によって、それらのうちのいくつかを選択することができるようにします。

Swing では、リストは、JList というクラスから生成されます。このクラスのコンストラクタのひとつは、オブジェクトの配列を引数として受け取って、その配列のそれぞれの要素を、リストの項目として設定します。

リストは、スクロールバーによってスクロールさせることが可能です。リストをスクロールできるようにする方法は、テキストエリアをスクロールできるようにする方法と同じです。つまり、スクロールペインを生成して、それにリストを追加すればいいわけです。

プログラムの例　PList.java

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

public class PList extends JFrame {
    PList(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        String[] items = new String[40];
        for (int i = 0; i < 40; i++)
            items[i] = "item" + i;
        JList list = new JList(items);
        JScrollPane scroll = new JScrollPane(list);
        content.add(scroll, BorderLayout.CENTER);
        setVisible(true);
    }

    public static void main(String[] args) {
        PList frame = new PList("リスト");
    }
}
```

6.1.2　選択結果の取得

リストを構成する項目のうちで人間によって選択されたものだけをリストから取り出したいときは、リストが持っている getSelectedValues というメソッドを使います。このメソッドは、

選択された項目から構成される配列を戻り値として返します。

プログラムの例 PGetSelected.java

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

public class PGetSelected extends JFrame {
    JList list;

    PGetSelected(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        String[] items = new String[40];
        for (int i = 0; i < 40; i++)
            items[i] = "item" + i;
        list = new JList(items);
        JScrollPane scroll = new JScrollPane(list);
        content.add(scroll, BorderLayout.CENTER);
        JButton button = new JButton("選択結果の取得");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null,
                    list.getSelectedValues(), "選択結果",
                    JOptionPane.INFORMATION_MESSAGE);
            }
        });
        content.add(button, BorderLayout.SOUTH);
        setVisible(true);
    }

    public static void main(String[] args) {
        PGetSelected frame = new PGetSelected(
            "選択結果の取得");
    }
}
```

6.2 リストモデル

6.2.1 リストモデルの基礎

リストの項目は、プログラムの実行中に追加したり削除したりすることができます。この節では、それができるプログラムを書く方法について説明したいと思います。

リストの項目は、「リストモデル」(list model) と呼ばれるオブジェクトによって管理されます。オブジェクトの配列でリストの項目を設定した場合、それらの項目は、追加や削除のできないリストモデルによって管理されることになります。項目の追加や削除のできるリストを作るためには、それができるリストモデルを生成して、それをリストに設定する必要があります。

項目の追加や削除のできるリストを作りたいときは、まず、DefaultListModel というクラス、または、独自の方法で項目を管理するために定義されたクラスからリストモデルを生成します。そして、そのリストモデルを JList クラスのコンストラクタに引数として渡すと、そのリストモデルがリストに設定されます。

DefaultListModel クラスのリストモデルは、addElement というメソッドを持っています。このメソッドは、1 個のオブジェクトを引数として受け取って、それをリストの末尾に項目として追加します。

プログラムの例 PListModel.java

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

public class PListModel extends JFrame {
```

```
    JList list;

    PListModel(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        DefaultListModel model = new DefaultListModel();
        for (int i = 0; i < 40; i++)
            model.addElement("item" + i);
        list = new JList(model);
        JScrollPane scroll = new JScrollPane(list);
        content.add(scroll, BorderLayout.CENTER);
        JButton button = new JButton("選択結果の取得");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null,
                    list.getSelectedValues(), "選択結果",
                    JOptionPane.INFORMATION_MESSAGE);
            }
        });
        content.add(button, BorderLayout.SOUTH);
        setVisible(true);
    }

    public static void main(String[] args) {
        PListModel frame = new PListModel("リストモデル");
    }
}
```

6.2.2 項目の位置

リストの特定の位置に項目を追加したり、特定の項目を削除したりするためには、その位置を指定する必要があります。

リストの中の項目の位置は、「インデックス」(index) と呼ばれる整数によって指定されます。それぞれの項目には、それが並んでいる順番のとおりに、0 から始まるインデックスが与えられています。

選択されている項目のインデックスは、リストが持っている、

`getSelectedIndex` または `getSelectedIndices`

というメソッドを使うことによって取得することができます。index が単数形になっているほうは、選択されている項目のうちで最初に選択されたもののインデックスを返すメソッドで、複数形のほうは、選択されている項目のインデックスを昇順に並べてできる配列を返すメソッドです。

リストの中のどの項目も選択されていない場合、単数形のほうは戻り値として -1 を返し、複数形のほうは空配列を返します。

6.2.3 項目の追加と削除

`DefaultListModel` クラスのリストモデルは、リストに対して項目を追加したり削除したりするメソッドを持っています。

リストに項目を追加したいときは、`insertElementAt` というメソッドを使います。このメソッドは 2 個の引数を受け取ります。1 個目はリストに追加する項目で、2 個目は追加する位置を指定するインデックスです。新しい項目は、インデックスで指定された項目の直前に挿入されます。

リストから項目を削除したいときは、`removeElementAt` というメソッドを使います。このメソッドは、引数としてインデックスを受け取って、それによって指定された位置にある項目を削除します。

項目を追加したり削除したりすると、その位置よりもうしろの項目は、インデックスが変化します。ですから、2 個以上の項目を一度に追加したり削除したりする場合は、インデックスの大きなものから順番に処理するなどの工夫が必要です。

プログラムの例 `PAddRemove.java`

```
import javax.swing.*;
```

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

public class PAddRemove extends JFrame {
    DefaultListModel model;
    JList list;

    PAddRemove(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        model = new DefaultListModel();
        for (int i = 0; i < 40; i++)
            model.addElement("item" + i);
        list = new JList(model);
        JScrollPane scroll = new JScrollPane(list);
        content.add(scroll, BorderLayout.CENTER);
        JPanel south = new JPanel(new GridLayout(1, 2));
        JButton addButton = new JButton("項目の追加");
        addButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String item = JOptionPane.showInputDialog(
                    null, "追加する項目を入力してください。",
                    "入力", JOptionPane.PLAIN_MESSAGE);
                if (item != null) {
                    int index = list.getSelectedIndex();
                    if (index == -1)
                        model.addElement(item);
                    else
                        model.insertElementAt(item, index);
                }
            }
        });
        south.add(addButton);
        JButton removeButton = new JButton("項目の削除");
        removeButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int[] indices = list.getSelectedIndices();
                for (int i = indices.length - 1;
                     i >= 0; i--) {
                    model.removeElementAt(indices[i]);
                }
            }
        });
        south.add(removeButton);
        content.add(south, BorderLayout.SOUTH);
        setVisible(true);
    }

    public static void main(String[] args) {
        PAddRemove frame = new PAddRemove("項目の追加と削除");
    }
}
```

6.3 コンボボックス

6.3.1 コンボボックスとは何か

いくつかの選択肢を表示して、人間にどれかを選択を選択させるためのコンポーネントとしては、リストのほかに、「コンボボックス」(combo box) と呼ばれるものもあります。

コンボボックスは、選択肢となる項目が常にその上に表示されているのではなくて、その右端にあるボタンを人間がクリックしたときだけそれをドロップダウンする、という点がリストとは違っています。ですから、コンボボックスには、占有する画面上の領域がリストに比べて少なくてすむという利点があります。しかし、リストならば複数の項目を選択することができるのに対

して、コンボボックスではひとつの項目しか選択できない、という弱点もあります。

Swing では、コンボボックスは、 JComboBox というクラスから生成されるコンポーネントです。コンボボックスを生成するときは、普通、その選択肢となるオブジェクトの配列をコンストラクタに渡します。

6.3.2 項目の選択に関連するメソッド

コンボボックスの項目の選択に関連するメソッドとしては、次のようなものがあります。

getSelectedItem	選択されている項目を戻り値として返します。
getSelectedIndex	選択されている項目のインデックスを戻り値として返します。
setSelectedIndex	0 またはプラスの整数を引数として受け取って、その整数をインデックスとする項目を、選択された状態します。このメソッドを使うことによって、コンボボックスが最初に表示されたときに選択されている項目を設定することができます。
setMaximumRowCount	1 個のプラスの整数を引数として受け取って、その整数を、選択肢をドロップダウンしたときに画面上に表示される最大の項目数として設定します。

6.3.3 編集可能なコンボボックス

コンボボックスは、ただ単に選択肢の中からひとつを選択するという機能だけではなくて、選択した項目を編集したり、あるいはまったく新しい項目を入力して、それを選択の結果にしたりする、ということもできるようになっています。ただし、デフォルトでは、コンボボックスは編集不可能な状態になっています。

コンボボックスが持っている setEditable というメソッドは、編集が可能かどうかという状態を変更します。このメソッドは、引数として真偽値を受け取って、それが真ならばレシーバーを編集可能にして、偽ならば編集不可能にします。

プログラムの例 PComboBox.java

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

public class PComboBox extends JFrame {
    JComboBox combo;

    PComboBox(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        String[] items = new String[40];
        for (int i = 0; i < 40; i++)
            items[i] = "item" + i;
        combo = new JComboBox(items);
        combo.setSelectedIndex(7);
        combo.setMaximumRowCount(10);
        combo.setEditable(true);
        content.add(combo, BorderLayout.NORTH);
        JButton button = new JButton("選択結果の取得");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null,
                    combo.getSelectedItem(), "選択結果",
                    JOptionPane.INFORMATION_MESSAGE);
            }
        });
        content.add(button, BorderLayout.SOUTH);
        setVisible(true);
    }

    public static void main(String[] args) {
        PComboBox frame = new PComboBox("コンボボックス");
```

```
    }
}
```

6.3.4 コンボボックスモデル

項目を追加したり削除したりすることのできるコンボボックスを作りたいときは、リストの場合と同じように、項目を管理するオブジェクトを作って、それをコンボボックスに設定します。コンボボックスの項目を管理するオブジェクトは、「コンボボックスモデル」(combo box model) と呼ばれます。

Swing には、コンボボックスモデルを生成する DefaultComboBoxModel というクラスがあります。このクラスが持っているコンストラクタのひとつは、オブジェクトの配列を引数として受け取って、それらのオブジェクトをコンボボックスの項目として設定します。

コンボボックスに対して項目を追加したり削除したりしたいときは、コンボボックスモデルが持っている、

```
addElement  insertElementAt  removeElementAt
```

というメソッドを使います。これらのメソッドの使い方は、リストモデルが持っている同じ名前のメソッドの使い方と同じです。

プログラムの例 PComboBoxModel.java

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

public class PComboBoxModel extends JFrame {
    DefaultComboBoxModel model;
    JComboBox combo;

    PComboBoxModel(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        String[] items = new String[40];
        for (int i = 0; i < 40; i++)
            items[i] = "item" + i;
        model = new DefaultComboBoxModel(items);
        combo = new JComboBox(model);
        combo.setSelectedIndex(7);
        combo.setMaximumRowCount(10);
        content.add(combo, BorderLayout.NORTH);
        JPanel south = new JPanel(new GridLayout(1, 2));
        JButton addButton = new JButton("項目の追加");
        addButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String item = JOptionPane.showInputDialog(
                    null, "追加する項目を入力してください。",
                    "入力", JOptionPane.PLAIN_MESSAGE);
                if (item != null) {
                    int index = combo.getSelectedIndex();
                    if (index == -1)
                        model.addElement(item);
                    else
                        model.insertElementAt(item, index);
                }
            }
        });
        south.add(addButton);
        JButton removeButton = new JButton("項目の削除");
        removeButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int index = combo.getSelectedIndex();
                if (index != -1)
                    model.removeElementAt(index);
            }
```

```
        });
        south.add(removeButton);
        content.add(south, BorderLayout.SOUTH);
        setVisible(true);
    }

    public static void main(String[] args) {
        PComboBoxModel frame = new PComboBoxModel(
            "コンボボックスに対する項目の追加と削除");
    }
}
```

6.4 スピナー

6.4.1 スピナーの基礎

いくつかの選択肢の中からひとつを選択してもらうためのコンポーネントとしては、コンボボックスのほかに、「スピナー」(spinner) と呼ばれるものもあります。スピナーは、選択された項目が表示される領域と、上下に並んだ二つのボタンから構成されます。選択された項目は、ボタンをクリックすることによって、その前の項目や次の項目に変更することができます。また、キーボードを使って項目を編集することも可能です。

Swing では、スピナーは、 JSpinner というクラスから生成されます。

6.4.2 スピナーモデル

スピナーによって選択される項目は、「スピナーモデル」(spinner model) と呼ばれるオブジェクトによって管理されます。

SpinnerNumberModel というクラスは、数値の範囲の中からひとつの数値を選択してもらう場合に使われるスピナーモデルを生成します。このクラスは、引数を受け取らないコンストラクタと、4 個の数値を受け取るコンストラクタを持っています。

引数を受け取らないコンストラクタは、初期値（最初に選択されている数値）を 0、下限と上限を無制限、1 回のクリックでの増減値を 1 に設定します。

4 個の数値を受け取るコンストラクタは、1 個目を初期値、2 個目を下限、3 個目を上限、4 個目を増減値として設定します。

プログラムの例 PSpinner.java

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

public class PSpinner extends JFrame {
    JSpinner spinner;

    PSpinner(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        spinner = new JSpinner(
            new SpinnerNumberModel(500, 200, 800, 10));
        content.add(spinner, BorderLayout.NORTH);
        JButton button = new JButton("選択結果の取得");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null,
                    spinner.getValue(), "選択結果",
                    JOptionPane.INFORMATION_MESSAGE);
            }
        });
        content.add(button, BorderLayout.SOUTH);
        setVisible(true);
    }
```

```
    public static void main(String[] args) {
        PSpinner frame = new PSpinner("スピナー");
    }
}
```

第 7 章　ダイアログボックス

7.1　オリジナルなダイアログボックス

7.1.1　ダイアログボックスのクラス

第 1 章で説明したように、Swing では、JOptionPane というクラスのクラスメソッドを呼び出すことによって、ダイアログボックスを表示することができます。しかし、その方法で表示することができるのは、メッセージを表示したり単純な質問をしたりするための定型的なダイアログボックスだけです。そこで、この節では、オリジナルなダイアログボックスを表示するためにはどうすればいいか、ということについて説明したいと思います。

フレームとダイアログボックスは、どちらもウィンドウというコンポーネントの一種です。ですから、ダイアログボックスの作り方は、フレームの作り方とほとんど同じです。

ダイアログボックスというのは、基本的には、JDialog というクラスから生成されるコンポーネントです。このクラスはさまざまなコンストラクタを持っていますが、ここでは、フレームと文字列と真偽値という三つの引数を受け取るものを紹介しましょう。

このコンストラクタが受け取る引数の 1 個目は、ダイアログボックスの親となるフレーム、つまりダイアログボックスを表示する主体となるフレームです。2 個目の引数は、タイトルバーに表示する文字列です。そして 3 個目の引数は、ダイアログボックスをモーダルなものにするか非モーダルなものにするかということを指定する真偽値です。

「モーダルな」(modal) ダイアログボックスというのは、それが表示されているあいだ、それを表示したフレーム（つまり親）に対する操作が不可能になるようなダイアログボックスのことです。それに対して、「非モーダルな」(non-mordal) ダイアログボックスというのは、それが表示されているあいだも、それを表示したフレームに対する操作が可能であるようなダイアログボックスのことです。コンストラクタに渡す 3 個目の引数は、真がモーダルという意味で、偽が非モーダルという意味です。

7.1.2　ダイアログボックスの初期設定

オリジナルなダイアログボックスを扱うためには、それを生成したのちにさまざまなメソッドを呼び出して初期設定をする必要があります。ですから、オリジナルなダイアログボックスを扱うプログラムは、フレームの場合と同じように、サブクラスを定義して、サブクラスのコンストラクタの中で初期設定を実行する、という書き方をするといいでしょう。

ダイアログボックスの大きさは、フレームの場合と同じように、setSize というメソッドを呼び出すことによって設定することができます。

ダイアログボックスを表示する位置は、ダイアログボックスが持っている setLocation というメソッドを呼び出すことによって設定することができます。このメソッドは、x 座標と y 座標を引数として受け取って、それによって指定された位置へダイアログボックスを移動させます。

ダイアログボックスを閉じる操作に対応する動作としては、デフォルトでは「隠す」という動作（つまり可視状態を不可視にするという動作）が設定されています。ダイアログボックスの場合はたいていこの動作のままでいいのですが、フレームの場合と同じように、

```
setDefaultCloseOperation
```

というメソッドを呼び出すことによって変更することもできます。

ダイアログボックスの可視状態も、フレームの場合と同じように、setVisible というメソッドを呼び出すことによって変更することができます。

ダイアログボックスのデフォルトの可視状態は、フレームと同じで、不可視になっています。フレームの場合は、コンストラクタの中で可視状態を可視に変更するのが普通ですが、ダイアログボックスの場合は、普段は不可視の状態にしておいて、必要なときだけ可視にする、という使い方をするのが普通です。

プログラムの例　PDialog.java

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

class EmptyDialog extends JDialog {
    EmptyDialog(JFrame frame, String title, boolean modal) {
        super(frame, title, modal);
        setSize(320, 160);
        setLocation(200, 100);
    }
}

public class PDialog extends JFrame {
    EmptyDialog dialog;

    PDialog(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        dialog = new EmptyDialog(this,
            "私はダイアログボックスです。", true);
        JPanel content = (JPanel)getContentPane();
        JButton button = new JButton(
            "ダイアログボックスの表示");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                dialog.setVisible(true);
            }
        });
        content.add(button, BorderLayout.SOUTH);
        setVisible(true);
    }

    public static void main(String[] args) {
        PDialog frame = new PDialog("ダイアログボックス");
    }
}
```

7.1.3 ダイアログボックスへのコンポーネントの追加

ダイアログボックスにコンポーネントを追加する方法は、フレームにコンポーネントを追加する方法とまったく同じです。まず、getContentPane というメソッドを呼び出してダイアログボックスからコンテントペインを取得して、そののち、add というメソッドを呼び出してコンポーネントをコンテントペインに追加します。

ちなみに、ダイアログボックスのコンテントペインに設定されているデフォルトのレイアウトマネージャーは、フレームのコンテントペインと同じで、BorderLayout です。

プログラムの例 PTwoButtons.java

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

class TwoButtonsDialog extends JDialog {
    String left, right, selected;

    TwoButtonsDialog(JFrame frame, String title,
                     String aleft, String aright) {
        super(frame, title, true);
        selected = left = aleft;
        right = aright;
        setSize(320, 160);
        setLocation(200, 100);
        JPanel content = (JPanel)getContentPane();
        content.setLayout(new GridLayout(1,2));
        JButton leftButton = new JButton(left);
        leftButton.addActionListener(new ActionListener() {
```

```
            public void actionPerformed(ActionEvent e) {
                selected = left;
                setVisible(false);
            }
        });
        content.add(leftButton);
        JButton rightButton = new JButton(right);
        rightButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                selected = right;
                setVisible(false);
            }
        });
        content.add(rightButton);
    }

    String showDialog() {
        setVisible(true);
        return selected;
    }
}

public class PTwoButtons extends JFrame {
    TwoButtonsDialog dialog;
    JLabel label;

    PTwoButtons(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        dialog = new TwoButtonsDialog(this,
            "あなたはどっちが好きですか。",
            "鉄人 28 号", "バビル 2 世");
        JPanel content = (JPanel)getContentPane();
        label = new JLabel();
        content.add(label, BorderLayout.CENTER);
        JButton button = new JButton("質問の表示");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                label.setText(
                    "あなたが選んだのは「" +
                    dialog.showDialog() + "」です。");
            }
        });
        content.add(button, BorderLayout.SOUTH);
        setVisible(true);
    }

    public static void main(String[] args) {
        PTwoButtons frame = new PTwoButtons(
            "二つのボタンを持つダイアログボックス");
    }
}
```

7.1.4 非モーダルなダイアログボックス

非モーダルなダイアログボックス、つまり、それが表示されているあいだも、それを表示したフレームに対する操作が禁止されないダイアログボックスを作りたいときは、三つの引数を受け取るコンストラクタに、3 個目の引数として false を渡します。

プログラムの例 PNonModal.java

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

class LabelDialog extends JDialog {
```

```
    JLabel label;

    LabelDialog(JFrame frame, String title, boolean modal) {
        super(frame, title, modal);
        setSize(320, 160);
        setLocation(200, 100);
        JPanel content = (JPanel)getContentPane();
        label = new JLabel();
        content.add(label, BorderLayout.CENTER);
    }

    void setText(String s) {
        label.setText(s);
        setVisible(true);
    }
}

public class PNonModal extends JFrame {
    LabelDialog dialog;
    JTextField field;

    PNonModal(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        dialog = new LabelDialog(this,
            "入力された文字列", false);
        JPanel content = (JPanel)getContentPane();
        field = new JTextField();
        content.add(field, BorderLayout.NORTH);
        JButton button = new JButton(
            "入力された文字列の表示");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                dialog.setText(field.getText());
                field.requestFocusInWindow();
            }
        });
        content.add(button, BorderLayout.SOUTH);
        field.requestFocusInWindow();
        setVisible(true);
    }

    public static void main(String[] args) {
        PNonModal frame = new PNonModal(
            "非モーダルなダイアログボックス");
    }
}
```

7.2 ファイル選択ダイアログボックス

7.2.1 JFileChooser クラス

GUI を持つアプリケーションは、データをファイルから読み込んだり、データをファイルに保存したりするとき、普通、その対象となるファイルを選択するためのダイアログボックスを表示します。

Swing の中にある JFileChooser というクラスは、ファイルを選択するダイアログボックスを生成します。

JFileChooser はいくつかのコンストラクタを持っていますが、ここでは、引数をまったく受け取らないコンストラクタを使って説明することにします。

ファイル選択ダイアログボックスを表示したいときは、まず、

```
JFileChuuser chooser = new JFileChooser();
```

というような宣言文で、JFileChooser クラスのオブジェクトを生成して、それを変数に設定し

ます。そして次に、そのオブジェクトが持っている、

showOpenDialog	データを読み込むファイルを選択する場合
showSaveDialog	データを保存するファイルを選択する場合

という二つのメソッドのうちのどちらかを呼び出すことによって、ダイアログボックスを表示します。

ファイル選択ダイアログボックスを表示する二つのメソッドは、どちらも、ダイアログボックスの親になるコンポーネント（フレームなど。nullでも可）を引数として受け取ります。そして、ダイアログボックスが閉じられた理由をあらわす、次のような戻り値（型はint）を返します。

JFileChooser.APPROVE_OPTION	承認のボタンが押された。
JFileChooser.CANCEL_OPTION	キャンセルされた。
JFileCHooser.ERROR_OPTION	エラーが発生した。

ファイルを選択した結果は、getSelectedFileというメソッドを呼び出すことによって取得することができます。このメソッドは、選択されたファイルをあらわす、Fileというクラスのオブジェクトを戻り値として返します。

Fileクラスのオブジェクトは、getAbsolutePathというメソッドを持っています。このメソッドは、レシーバーがあらわしているファイルの絶対パス名を戻り値として返します。

プログラムの例 PFileChooser.java

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

public class PFileChooser extends JFrame {
    JLabel label;

    PFileChooser(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        label = new JLabel();
        content.add(label, BorderLayout.CENTER);
        JButton button = new JButton("ファイルの選択");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showFileChooser();
            }
        });
        content.add(button, BorderLayout.SOUTH);
        setVisible(true);
    }

    void showFileChooser() {
        JFileChooser chooser = new JFileChooser();
        int retval = chooser.showOpenDialog(this);
        if (retval == JFileChooser.APPROVE_OPTION)
            label.setText(
                chooser.getSelectedFile().getAbsolutePath());
    }

    public static void main(String[] args) {
        PFileChooser frame = new PFileChooser(
            "ファイル選択ダイアログボックス");
    }
}
```

7.2.2 ファイルフィルター

ファイル選択ダイアログボックスを使ってファイルを選択するとき、特定の種類のファイルだけがその中に表示されるようにしたい、ということがしばしばあります。そのようなときは、

「ファイルフィルター」(file filter) と呼ばれるものを使います。ファイルフィルターというのは、ファイルを通過させるか脱落させるかということを決定するオブジェクトのことです。ファイルフィルターをファイル選択ダイアログボックスに設定しておくと、ファイルフィルターを通過したファイルだけがそこに表示されることになります。

ファイルフィルターを作りたいときは、

```
javax.swing.filechooser.FileFilter
```

という抽象クラスを実装するサブクラスを定義します。そうすると、そのサブクラスから生成されたオブジェクトがファイルフィルターになります。

ファイルフィルターの抽象クラスは、

```
public boolean accept(File file)
public String getDescription()
```

という二つの抽象メソッドを持っていて、これらをサブクラスで実装することができます。

ファイルを通過させるか脱落させるかという判断は、 accept というメソッドによって決定されます。このメソッドは、判定の対象となるファイルを引数として受け取って、真偽値を返します。戻り値が真というのはファイルを通過させるという意味で、偽というのは脱落させるという意味になります。

なお、 accept を実装するとき、引数がディレクトリだった場合は真を返すようにしないと、ディレクトリが表示されなくなってしまいますので、注意してください。

getDescription は、ファイルフィルターを説明する文字列を戻り値として返すメソッドです。このメソッドの戻り値は、ファイル選択ダイアログボックスの中にある、ファイルフィルターを選択するコンボボックスの選択肢の上に表示されます。

ファイル選択ダイアログボックスにファイルフィルターを設定したいときは、ファイル選択ダイアログボックスが持っている addChoosableFileFilter というメソッドを使います。ファイルフィルターをこのメソッドに引数として渡すと、そのファイルフィルターがファイル選択ダイアログボックスに設定されます。

プログラムの例 PFileFilter.java

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

class Filter extends javax.swing.filechooser.FileFilter {
    String message, extension;

    Filter(String msg, String ext) {
        message = msg;
        extension = ext;
    }

    public boolean accept(File file) {
        String name = file.getName().toLowerCase();
        return file.isDirectory() || name.endsWith(extension);
    }

    public String getDescription() {
        return message + " (*" + extension + ")";
    }
}

public class PFileFilter extends JFrame {
    JLabel label;

    PFileFilter(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        label = new JLabel();
        content.add(label, BorderLayout.CENTER);
```

```
        JButton button = new JButton("ファイルの選択");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showFileChooser();
            }
        });
        content.add(button, BorderLayout.SOUTH);
        setVisible(true);
    }

    void showFileChooser() {
        JFileChooser chooser = new JFileChooser();
        chooser.addChoosableFileFilter(
            new Filter("プレーンテキスト", ".txt"));
        chooser.addChoosableFileFilter(
            new Filter("Ruby のプログラム", ".rb"));
        chooser.addChoosableFileFilter(
            new Filter("Java のプログラム", ".java"));
        int retval = chooser.showOpenDialog(this);
        if (retval == JFileChooser.APPROVE_OPTION)
            label.setText(
                chooser.getSelectedFile().getAbsolutePath());
    }

    public static void main(String[] args) {
        PFileFilter frame = new PFileFilter(
            "ファイルフィルター");
    }
}
```

7.3 色選択ダイアログボックス

7.3.1 色選択ダイアログボックスを表示する三つの方法

Swing が持っている JColorChooser というクラスは、色を選択するためのダイアログボックスを表示するという機能を持っています。

JColorChooser クラスを使って色選択ダイアログボックスを表示する方法としては、次の三つのものがあります。

(1) JColorChooser クラスのクラスメソッドを使う方法。

(2) JColorChooser クラスのオブジェクトと JColorChooser クラスのクラスメソッドとを使う方法。

(3) JDialog クラスのサブクラスのオブジェクトと JColorChooser クラスのオブジェクトとを使う方法。

これらの方法は、上に書かれているものほどプログラムを簡単に書くことができます。しかし、目的に応じた動作をさせることのできる自由度は、下に書かれているものほど大きくなります。どの方法を使えばいいかというのは、その点を考慮して決定する必要があります。

7.3.2 色選択ダイアログボックスを表示するクラスメソッド

まず、第一の方法、すなわち JColorChooser クラスのクラスメソッドを使って色選択ダイアログボックスを表示する方法について説明しましょう。

JColorChooser クラスは、showDialog というクラスメソッドを持っています。これは、色選択ダイアログボックスを表示するメソッドです。

showDialog は、3 個の引数を受け取ります。1 個目は親となるコンポーネント、2 個目はタイトルバーに表示する文字列、そして 3 個目は、最初に選択されている状態にする色です。

showDialog によって表示される色選択ダイアログボックスには、「了解」「取消し」「リセット」というボタンがあります。「了解」を押すと、そのときに選択されていた色が showDialog の戻り値になります。それに対して、「取消し」が押された場合、 showDialog は戻り値として null を返します。そして「リセット」は、色選択ダイアログボックスを表示された直後の状態に戻し

ます。

プログラムの例 PColorChooser1.java

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

public class PColorChooser1 extends JFrame {
    final static int RECTWIDTH = 280;
    final static int RECTHEIGHT = 180;
    Color color;

    PColorChooser1(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JButton button = new JButton("色の選択");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showColorChooser();
                repaint();
            }
        });
        content.add(button, BorderLayout.SOUTH);
        color = Color.white;
        setVisible(true);
    }

    void showColorChooser() {
        Color retval = JColorChooser.showDialog(this,
            "色の選択", color);
        if (retval != null)
            color = retval;
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(color);
        g.fillRect(getWidth()/2 - RECTWIDTH/2,
                   getHeight()/2 - RECTHEIGHT/2,
                   RECTWIDTH, RECTHEIGHT);
    }

    public static void main(String[] args) {
        PColorChooser1 frame = new PColorChooser1(
            "色選択ダイアログボックス・その壱");
    }
}
```

7.3.3 色選択ダイアログボックスを生成するクラスメソッド

次に、第二の方法、すなわち JColorChooser クラスのオブジェクトと JColorChooser クラスのクラスメソッドとを使って色選択ダイアログボックスを表示する方法について説明しましょう。

JColorChooser クラスからオブジェクトを生成すると、それは、色を選択するという機能を持っているコンポーネントになります。

JColorChooser クラスのコンストラクタとしては、普通、引数として 1 個の色（Color クラスのオブジェクト）を受け取るものを使います。このコンストラクタは、受け取った引数を、最初に選択されている色として設定します。

選択された色を JColorChooser クラスのオブジェクトから取り出したいときは、それが持っている getColor というメソッドを呼び出します。このメソッドは、選択された色をレシーバーから取り出して、それを戻り値（型は Color）として返します。

JColorChooser クラスのオブジェクトは、コンポーネントなのですが、フレームでもダイアログボックスでもありませんので、それを単独で画面に表示するということはできません。それ

を画面に表示するためには、フレームかダイアログボックスの上にそれを取り付ける必要があります。

JColorChooserクラスが持っているcreateDialogというクラスメソッドは、色を選択するコンポーネントを取り付けたダイアログボックスを生成して、それを戻り値として返します。このメソッドの戻り値は、型がJDialogですので、それを取り扱う方法は、JDialogクラス（またはそれを継承するクラス）から生成したダイアログボックスを取り扱う方法と同じです。

createDialogは、6個の引数を受け取ります。1個目は親となるコンポーネント、2個目はタイトルバーに表示する文字列、3個目はモーダルにするか非モーダルにするかを指定する真偽値（真はモーダル）、4個目はJColorChooserクラスのオブジェクト、5個目は「了解」が押されたときに実行されるイベントリスナー、そして6個目は「取消し」が押されたときに実行されるイベントリスナーです。

プログラムの例 PColorChooser2.java

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

public class PColorChooser2 extends JFrame {
    final static int RECTWIDTH = 280;
    final static int RECTHEIGHT = 180;
    Color color;
    JColorChooser chooser;
    JDialog dialog;

    PColorChooser2(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JButton button = new JButton("色の選択");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                dialog.setVisible(true);
            }
        });
        content.add(button, BorderLayout.SOUTH);
        color = Color.white;
        chooser = new JColorChooser(Color.white);
        dialog = JColorChooser.createDialog(this, "色の選択",
            true, chooser,
            new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    color = chooser.getColor();
                    repaint();
                }
            },
            new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                }
            });
        setVisible(true);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(color);
        g.fillRect(getWidth()/2 - RECTWIDTH/2,
                   getHeight()/2 - RECTHEIGHT/2,
                   RECTWIDTH, RECTHEIGHT);
    }

    public static void main(String[] args) {
        PColorChooser2 frame = new PColorChooser2(
            "色選択ダイアログボックス・その弐");
```

```
    }
}
```

7.3.4 色を選択する独自のダイアログボックス

第三の方法、すなわち JDialog クラスのサブクラスのオブジェクトと JColorChooser クラスのオブジェクトとを使って色選択ダイアログボックスを表示する方法については、これまでの知識を応用するだけですので、改めて説明しなくても大丈夫でしょう。要するに、色を選択するコンポーネントを取り付けたダイアログボックスのクラスを独自に定義すればいいわけです。

プログラムの例 PColorChooser3.java

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

class ColorDialog extends JDialog {
    JColorChooser chooser;

    ColorDialog(JFrame frame, String title, Color color) {
        super(frame, title, true);
        setSize(500, 500);
        setLocation(200, 100);
        JPanel content = (JPanel)getContentPane();
        chooser = new JColorChooser(color);
        content.add(chooser, BorderLayout.CENTER);
        JButton leftButton = new JButton("了解");
        leftButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                setVisible(false);
            }
        });
        content.add(leftButton, BorderLayout.SOUTH);
    }

    Color showDialog() {
        setVisible(true);
        return chooser.getColor();
    }
}

public class PColorChooser3 extends JFrame {
    final static int RECTWIDTH = 280;
    final static int RECTHEIGHT = 180;
    Color color;
    ColorDialog dialog;

    PColorChooser3(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        color = Color.white;
        dialog = new ColorDialog(this, "色の選択", color);
        JPanel content = (JPanel)getContentPane();
        JButton button = new JButton("色の選択");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                color = dialog.showDialog();
                repaint();
            }
        });
        content.add(button, BorderLayout.SOUTH);
        setVisible(true);
    }

    public void paint(Graphics g) {
        super.paint(g);
```

```
        g.setColor(color);
        g.fillRect(getWidth()/2 - RECTWIDTH/2,
                   getHeight()/2 - RECTHEIGHT/2,
                   RECTWIDTH, RECTHEIGHT);
    }

    public static void main(String[] args) {
        PColorChooser3 frame = new PColorChooser3(
            "色選択ダイアログボックス・その参");
    }
}
```

第 8 章　コンテナ

8.1　スプリットペイン

8.1.1　この章について

以前にも説明したことですが、自分の内部にコンポーネントを追加することができるという機能を持っているコンポーネントは、「コンテナ」(container) と呼ばれます。この実習マニュアルですでに紹介した、コンテントペイン、スクロールペイン、パネルというコンポーネントは、それぞれコンテナの一種です。

コンテナには、これまでに紹介したもののほかにも、まだまだたくさんの種類があります。そこで、この章では、この実習マニュアルにまだ登場していないコンテナをいくつか紹介していきたいと思います。

8.1.2　スプリットペインの生成

Swing には、「スプリットペイン」(split pane) と呼ばれるコンテナがあります。これは、領域を二つの部分に分割して、それらのあいだの境界線を、マウスでドラッグすることによって自由に移動させることができる、という機能を持っているコンテナです。

スプリットペインは、`JSplitPane` というクラスから生成されます。スプリットペインを初期化するためのコンストラクタとして、ここでは 3 個の引数を受け取るものを紹介しましょう。

1 個目の引数は、分割の方向をあらわす整数です。この整数は、`JSplitPane` クラスの中で、

`HORIZONTAL_SPLIT`	水平方向に分割する。
`VERTICAL_SPLIT`	垂直方向に分割する。

と定義されている定数を使うことによって指定することができます。

2 個目と 3 個目の引数は、スプリットペインの二つの領域のそれぞれに追加するコンポーネントです。分割が水平方向の場合は 2 個目が左に表示され、垂直方向の場合は 2 個目が上に表示されます。

8.1.3　ワンタッチエクスパンダー

スプリットペインは、「ワンタッチエクスパンダー」(one touch expander) と呼ばれる機能を持っています。これは、スプリットペインの境界線をマウスでドラッグするのではなくて、境界線の上に表示された小さな三角形をクリックすることによって、境界線を末端まで移動させることができる、という機能です。ただし、ワンタッチエクスパンダーは、デフォルトでは作動しない状態になっています。

ワンタッチエクスパンダーが作動する状態なのかそうでない状態なのかということは、スプリットペインが持っている `setOneTouchExpandable` というメソッドを呼び出すことによって変更することができます。このメソッドは、1 個の真偽値を引数として受け取って、それが真ならばワンタッチエクスパンダーを作動する状態に設定して、偽ならば作動しない状態に設定します。

8.1.4　境界線の位置の設定

スプリットペインの境界線の位置は、それが持っている `setDividerLocation` というメソッドを呼び出すことによって設定することができます。このメソッドは、引数として 1 個の整数を受け取って、それによって指定された位置に境界線を置きます。位置は、分割が水平方向の場合は左

端からの距離で指定され、垂直方向の場合は上端からの距離で指定されます（単位はピクセル）。

プログラムの例 PSplitPane.java

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

public class PSplitPane extends JFrame {
    PSplitPane(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JSplitPane split = new JSplitPane(
            JSplitPane.HORIZONTAL_SPLIT,
            new JLabel("左"), new JLabel("右"));
        split.setOneTouchExpandable(true);
        split.setDividerLocation(160);
        content.add(split, BorderLayout.CENTER);
        setVisible(true);
    }

    public static void main(String[] args) {
        PSplitPane frame = new PSplitPane(
            "スプリットペイン");
    }
}
```

8.2 タブペイン

8.2.1 タブペインの生成

「タブペイン」(tabbed pane) と呼ばれるコンテナは、自分に追加されたコンポーネントを、同時にすべて表示するのではなくて、それらのうちのひとつだけを表示します。タブペインの上に表示されるコンポーネントは、「タブ」(tab) と呼ばれる小さな領域をクリックすることによって自由に選択することができます。

タブペインは、JTabbedPane というクラスから生成されるコンテナです。タブペインを初期化するコンストラクタは三つあって、そのうちのひとつ、引数を受け取らないコンストラクタは、タブペインに対してデフォルトの設定をします。

8.2.2 コンポーネントの追加

タブペインというコンテナに対してコンポーネントを追加するメソッドとしては、ほかのコンテナの場合と同じように add を使ってもいいのですが、タブペインの場合は addTab というメソッドを使うのが普通です。

1 個目の引数として文字列、2 個目の引数としてコンポーネントを渡して addTab を呼び出すと、2 個目の引数がタブペインに追加されます。そして、1 個目の引数は、そのコンポーネントを選択するためのタブの上にタイトルとして表示されます。

8.2.3 表示されるコンポーネントの設定

タブペインの上に表示されるコンポーネントは、タブのクリックによって選択することができるわけですが、プログラムの側でそれを選択したい場合は、setSelectedIndex というメソッドを使います。

タブペインに追加されたコンポーネントには、追加の順番にしたがって、0 から始まるインデックスが与えられます。setSelectedIndex に引数として整数を渡すと、その整数をインデックスとするコンポーネントが選択されることになります。

プログラムの例 PTabbedPane.java

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

public class PTabbedPane extends JFrame {
    PTabbedPane(String title) {
```

```
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JTabbedPane tabbed = new JTabbedPane();
        for (int i = 0; i < 20; i++)
            tabbed.addTab("tab " + i,
                new JLabel("label " + i));
        tabbed.setSelectedIndex(7);
        content.add(tabbed, BorderLayout.CENTER);
        setVisible(true);
    }

    public static void main(String[] args) {
        PTabbedPane frame = new PTabbedPane("タブペイン");
    }
}
```

8.2.4 タブの位置

タブは、上端、下端、左端、右端のいずれかに表示することができます。タブペインを生成するときにタブが表示される位置を設定したいときは、引数として 1 個の整数を受け取るコンストラクタを使います。このコンストラクタは、タブの位置を意味する整数を受け取って、それにしたがってタブの位置を設定します。

タブが表示される位置を示す整数は、

`JTabbedPane.TOP`	上端（デフォルト）。
`JTabbedPane.BOTTOM`	下端。
`JTabbedPane.LEFT`	左端。
`JTabbedPane.RIGHT`	右端。

という定数を使って記述することができます。たとえば、

```
JTabbedPane tabbed = new JTabbedPane(JTabbedPane.BOTTOM);
```

という宣言文でタブペインを生成したとすると、そのタブペインは、タブを下端に表示する設定になります。

8.2.5 タブのレイアウトポリシー

タブペインは、すべてのタブを一列に並べるためのスペースがない場合、自分に設定されている「レイアウトポリシー」(layout policy) と呼ばれるものにしたがって、それをどのように表示するかということを決定します。タブのレイアウトポリシーには、それらをいくつかの段に分けて表示するというものと、それらをスクロールさせることができるようにするという二つのものがあります。

タブペインを生成するときにタブのレイアウトポリシーを設定したいときは、2 個の引数を受け取るコンストラクタを使います。1 個目の引数はタブの位置を意味する整数で、2 個目の引数はレイアウトポリシーを意味する整数です。

レイアウトポリシーを示す整数は、

`JTabbedPane.WRAP_TAB_LAYOUT`	複数の段に分ける（デフォルト）。
`JTabbedPane.SCROLL_TAB_LAYOUT`	スクロールできるようにする。

という定数を使って記述することができます。たとえば、

```
JTabbedPane tabbed = new JTabbedPane(
    JTabbedPane.TOP, JTabbedPane.SCROLL_TAB_LAYOUT);
```

という宣言文でタブペインを生成したとすると、そのタブペインには、すべてのタブを一列に並べるスペースがない場合にはそれをスクロールさせることができるようにするというレイアウトポリシーが設定されることになります。

プログラムの例 PTabbedPane2.java

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

```
public class PTabbedPane2 extends JFrame {
    PTabbedPane2(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JTabbedPane tabbed = new JTabbedPane(
            JTabbedPane.LEFT, JTabbedPane.SCROLL_TAB_LAYOUT);
        for (int i = 0; i < 40; i++)
            tabbed.addTab("tab " + i,
                new JLabel("label " + i));
        tabbed.setSelectedIndex(7);
        content.add(tabbed, BorderLayout.CENTER);
        setVisible(true);
    }

    public static void main(String[] args) {
        PTabbedPane2 frame = new PTabbedPane2(
            "タブの位置とレイアウトポリシー");
    }
}
```

8.3 デスクトップペイン

8.3.1 仮想デスクトップ

ウィンドウがその上に表示される領域は、「デスクトップ」(desktop) と呼ばれます。GUI ではしばしば、デスクトップそっくりのものをウィンドウの内部に作ったものが使われます。そのような、ウィンドウの内部に作られたデスクトップは、「仮想デスクトップ」(virtual desktop) または MDI(multiple document interface) と呼ばれます。

Swing では、仮想デスクトップは「デスクトップペイン」(desktop pane) と呼ばれるコンテナを使うことによって実現することができます。

デスクトップペインは、`JDesktopPane` というクラスから生成されるコンテナです。このコンテナをフレームのコンテントペインに追加すると、そのフレームの内部に仮想デスクトップが表示されます。

`JDesktopPane` クラスのコンストラクタとしては、引数を受け取らないものがひとつあるだけです。ですから、デスクトップペインを生成したいときは、常に、

```
new JDesktopPane()
```

という式を書くことになります。

8.3.2 内部フレーム

デスクトップペインの上に表示されるフレームは、「内部フレーム」(internal frame) と呼ばれます。内部フレームを生成して、`add` を使ってそれをデスクトップペインに追加すると、その内部フレームはデスクトップペインの上に表示されます。

内部フレームは、`JFrame` ではなくて、`JInternalFrame` というクラスから生成されます。

`JInternalFrame` クラスは、いくつかのコンストラクタを持っていますが、ここでは、5 個の引数を受け取るものを紹介しましょう。このコンストラクタが受け取る引数は、

1 個目　文字列　タイトルバーに表示する文字列。
2 個目　真偽値　大きさの変更ができるかどうか。
3 個目　真偽値　閉じることができるかどうか。
4 個目　真偽値　最大化できるかどうか。
5 個目　真偽値　アイコン化できるかどうか。

という意味を持っています。たとえば、

```
new JInternalFrame("題名", true, false, true, false)
```

という式を評価したとすると、タイトルバーに「題名」と表示されていて、大きさの変更と最大化はできるけれども閉じることとアイコン化はできない内部フレームが生成されます。

8.3.3 内部フレームの初期設定

なお、内部フレームは、普通のフレームと同じように、

`setSize`	大きさを変更する。
`setLocation`	位置を変更する。
`setVisible`	可視状態を変更する。

などのメソッドを持っていますので、それらを使うことによって、初期設定を実行することができます。

内部フレームの内部にコンポーネントを表示する方法も、普通のフレームの場合と同じです。まず内部フレームからコンテントペインを取得して、表示したいコンポーネントをそのコンテントペインに追加します。

8.3.4 内部フレームをアクティブにするメソッド

デスクトップペインの上の内部フレームは、デスクトップの上のウィンドウと同じように、ひとつの時点ではひとつだけがアクティブになっています。特定の内部フレームをアクティブにしたいときは、それをマウスでクリックすればいいわけですが、プログラムの側でアクティブな内部フレームを切り替えることも可能です。

内部フレームが持っている `setSelected` というメソッドは、引数として 1 個の真偽値を受け取って、それが真ならばレシーバーをアクティブにして、偽ならば非アクティブにします。なお、このメソッドは、動作を実行することができない場合に、捕獲する必要のある例外を発生させる、という点に注意してください。

プログラムの例 `PDesktopPane.java`

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

class TextAreaFrame extends JInternalFrame {
    static int count = 0, x = 0, y = 0;

    TextAreaFrame() {
        super("frame " + count, true, true, true, true);
        setSize(200, 150);
        setLocation(x, y);
        JPanel content = (JPanel)getContentPane();
        content.add(new JScrollPane(new JTextArea()));
        setVisible(true);
        count++;
        x += 40;
        y += 30;
    }
}

public class PDesktopPane extends JFrame {
    JDesktopPane desktop;

    PDesktopPane(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        desktop = new JDesktopPane();
        content.add(desktop, BorderLayout.CENTER);
        JButton button = new JButton("内部フレームの生成");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                createTextAreaFrame();
            }
```

```
        });
        content.add(button, BorderLayout.SOUTH);
        setVisible(true);
    }

    void createTextAreaFrame() {
        TextAreaFrame frame = new TextAreaFrame();
        desktop.add(frame);
        try {
            frame.setSelected(true);
        } catch(Exception exp) {}
    }

    public static void main(String[] args) {
        PDesktopPane frame = new PDesktopPane(
            "デスクトップペイン");
    }
}
```

8.4 ツールバー

8.4.1 ツールバーとは何か

Swing には、「ツールバー」(tool bar) と呼ばれるコンテナがあります。これは、基本的には自分に格納されているコンポーネントを横一列または縦一列に並べて表示するコンテナです。

しかし、ツールバーの機能はそれだけではありません。ツールバーは、マウスでドラッグすることによって、ウィンドウの上下左右のいずれかの位置へ移動させることができるのです。さらに、ウィンドウの中央またはウィンドウの外へツールバーをドラッグすると、そのツールバーはウィンドウから切り離されて、もとのウィンドウとは別の独立したウィンドウの中に表示されます。

8.4.2 ツールバーの生成

ツールバーは、JToolBar というクラスから生成されます。このクラスには 4 個のコンストラクタがあります。そのうちのひとつは、1 個目の引数として文字列、2 個目の引数として整数を受け取ります。

引数の 1 個目は、ツールバーに与える名前です。この名前は、ツールバーをウィンドウから切り離したときにそれを表示するウィンドウのタイトルバーに表示されます。

引数の 2 個目は、コンポーネントを並べる方向をあらわす整数です。この整数は、

JToolBar.HORIZONTAL	水平方向
JToolBar.VERTICAL	垂直方向

という定数を書くことによって記述することができます。

JToolBar クラスのコンストラクタの残りの三つは、コンポーネントを並べる方向だけを受け取るものと、名前だけを受け取るものと、引数を何も受け取らないものです。名前を受け取らないコンストラクタは、ツールバーに名前を与えません。そして、コンポーネントを並べる方向を受け取らないコンストラクタは、コンポーネントを水平に並べます。

8.4.3 セパレーター

ツールバーには、コンポーネントだけではなく、「セパレーター」(separator) と呼ばれる空間を追加することもできます。セパレーターをコンポーネントとコンポーネントとのあいだに挿入することによって、コンポーネントがいくつかのグループに分かれているということを視覚的に表現することができます。

ツールバーが持っている addSeparator というメソッドは、レシーバーにセパレーターを追加します。

プログラムの例 PToolBar.java

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

```
public class PToolBar extends JFrame {
    PToolBar(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JToolBar bar = new JToolBar("私はツールバーです。",
            JToolBar.VERTICAL);
        for (int i = 0; i < 6; i++) {
            if (i == 3)
                bar.addSeparator();
            bar.add(new JButton("ボタン" + i));
        }
        content.add(bar, BorderLayout.WEST);
        setVisible(true);
    }

    public static void main(String[] args) {
        PToolBar frame = new PToolBar("ツールバー");
    }
}
```

第 9 章　テーブル

9.1　テーブルの基礎

9.1.1　テーブルとは何か

Swing では、「テーブル」(table) と呼ばれるコンポーネントを扱うことができるようになっています。テーブルというのは、いくつかのオブジェクトを縦と横の二つの方向に並べて表示するコンポーネントのことです。テーブルを構成するそれぞれのオブジェクトは、編集することも可能です。

テーブルによる表示や編集の対象となるオブジェクトは、「セル」(cell) と呼ばれる容器に格納されます。また、ひとつのオブジェクトが表示される画面上の領域も、「セル」と呼ばれます。

セルの中に格納されているオブジェクトは、そのセルの「値」(value) と呼ばれます。

画面の上に表示されているセルをダブルクリックすると、そのセルは、値を編集することができる状態になります。

いくつかのセルを横の方向に一列に並べたものは、「行」(row) と呼ばれます。テーブルは、いくつかの行を縦に並べることによって構成されます。また、テーブルの縦の方向に並んでいる一列のセルは、「列」(column) と呼ばれます。

画面上の列の横幅は、マウスの操作によって広くしたり狭くしたりすることができます。また、列の名前が表示されている部分をマウスでドラッグすることによって、列が表示される順番を入れ替えることも可能です。

9.1.2　テーブルの生成

テーブルは、JTable というクラスから生成されます。JTable クラスはさまざまなコンストラクタを持っていますが、ここでは、引数として二つのベクトルを受け取るものを紹介しましょう。

このコンストラクタが受け取る引数の 1 個目は、テーブルによる表示や編集の対象となるオブジェクトから構成される、ベクトルを要素とするベクトルです。その要素になっているそれぞれのベクトルは、テーブルのそれぞれの行に格納されます。そして、要素の要素になっているそれぞれのオブジェクトは、それぞれのセルに格納されます。

引数の 2 個目は、テーブルを構成するそれぞれの列に名前として与える文字列から構成されるベクトルです。

プログラムの例　PTable.java

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

public class PTable extends JFrame {
```

```
    final static int NUMROWS = 60;
    final static int NUMCOLUMNS = 4;

    PTable(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.add(new JScrollPane(createTable()),
            BorderLayout.CENTER);
        setVisible(true);
    }

    JTable createTable() {
        Vector columnnames = new Vector();
        for (int i = 0; i < NUMCOLUMNS; i++)
            columnnames.add(i + "列目");
        Vector rows = new Vector();
        for (int i = 0; i < NUMROWS; i++) {
            Vector columns = new Vector();
            for (int j = 0; j < NUMCOLUMNS; j++)
                columns.add(i + "行目の" + j + "列目");
            rows.add(columns);
        }
        return new JTable(rows, columnnames);
    }

    public static void main(String[] args) {
        PTable frame = new PTable("テーブル");
    }
}
```

9.2 セルの値の取得

9.2.1 選択された行と列のインデックスの取得

テーブルの中の行と列の位置は、リストの項目と同じように、「インデックス」(index) と呼ばれる整数によって指定されます。それぞれの行と列には、0 から始まるインデックスが与えられています。

テーブルは、マウスのクリックによって、その中のいくつかのセルを選択することができます。選択されているセルの行または列のインデックスを調べたいときは、

getSelectedRow　getSelectedColumn

getSelectedRows　getSelectedColumns

というメソッドを使います。単数形になっているものは、最初に選択されたセルの行または列のインデックスを返すメソッドで、複数形になっているものは、選択されたすべてのセルの行または列のインデックスから構成される配列を返すメソッドです。

セルがひとつも選択されていない場合の戻り値は、単数形のメソッドは -1 で、複数形のメソッドは空配列です。

9.2.2 セルの値を取得するメソッド

セルの値を取得したいときは、テーブルが持っている `getValueAt` というメソッドを呼び出します。このメソッドは、引数として 2 個の整数を受け取ります。そして、1 個目の引数を行のインデックス、2 個目の引数を列のインデックスとするセルから値を取り出して、それを戻り値として返します。

ちなみに、`setValueAt` というメソッドを呼び出すことによって、セルに値を設定することも可能です。このメソッドは、3 個の引数を受け取ります。1 個目はセルに設定するオブジェクト、2 個目は行のインデックス、3 個目は列のインデックスです。

プログラムの例　`PGetValue.java`

```
import javax.swing.*;
```

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class PGetValue extends JFrame {
    final static int NUMROWS = 60;
    final static int NUMCOLUMNS = 4;
    JTable table;

    PGetValue(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        createTable();
        content.add(new JScrollPane(table),
            BorderLayout.CENTER);
        JButton button = new JButton("セルの値の取得");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showSelectedValue();
            }
        });
        content.add(button, BorderLayout.SOUTH);
        setVisible(true);
    }

    void createTable() {
        Vector columnnames = new Vector();
        for (int i = 0; i < NUMCOLUMNS; i++)
            columnnames.add(i + "列目");
        Vector rows = new Vector();
        for (int i = 0; i < NUMROWS; i++) {
            Vector columns = new Vector();
            for (int j = 0; j < NUMCOLUMNS; j++)
                columns.add(i + "行目の" + j + "列目");
            rows.add(columns);
        }
        table = new JTable(rows, columnnames);
    }

    void showSelectedValue() {
        int row = table.getSelectedRow();
        int column = table.getSelectedColumn();
        if (column >= 0)
            JOptionPane.showMessageDialog(this,
                table.getValueAt(row, column),
                "セルの値", JOptionPane.INFORMATION_MESSAGE);
        else
            JOptionPane.showMessageDialog(this,
                "セルが選択されていません。",
                "エラー", JOptionPane.ERROR_MESSAGE);
    }

    public static void main(String[] args) {
        PGetValue frame = new PGetValue("セルの値の取得");
    }
}
```

9.3　テーブルモデル

9.3.1　テーブルモデルとは何か

テーブルに格納されているオブジェクトは、テーブルに設定されている「テーブルモデル」(table model) と呼ばれるオブジェクトによって管理されます。

テーブルモデルは、TableModel というインターフェースを実装したクラスから生成されるオブジェクトです。

なお、テーブルモデルに関するインターフェースやクラスは、javax.swing.table というパッケージの中で定義されています。

9.3.2 テーブルモデルの取得

テーブルには、デフォルトで、DefaultTableModel というクラスのテーブルモデルが設定されています。このテーブルモデルは、テーブルに対して、行を挿入したり削除したりするという操作を実行するメソッドを持っています。

テーブルに対する行の挿入や削除を実行するためには、まず、テーブルからテーブルモデルを取得する必要があります。テーブルモデルは、テーブルが持っている getModel というメソッドを呼び出すことによって取得することができます。このメソッドは、レシーバーに設定されているテーブルモデルを戻り値として返します。

なお、getModel の戻り値は、その型が本来のものではなくなっていますので、そのままでは必要なメソッドを呼び出すことができない場合もあります。そのような場合は、

```
(DefaultTableModel)table.getModel()
```

というように、キャストを使って本来の型に戻す必要があります。

9.3.3 行の挿入と削除

行の挿入は、デフォルトのテーブルモデルが持っている insertRow というメソッドによって実行されます。このメソッドは、二つの引数を受け取ります。引数の 1 個目は、挿入する位置を指定する整数です。新しい行は、この引数をインデックスとする行の上に挿入されます。引数の 2 個目は、挿入する行に格納されるオブジェクトから構成されるベクトルです。

行の削除は、デフォルトのテーブルモデルが持っている removeRow というメソッドによって実行されます。このメソッドは、1 個の整数を引数として受け取って、その整数をインデックスとする行を削除します。

9.3.4 列の個数の取得

insertRow に対して 2 個目の引数として渡すベクトルは、その大きさが、テーブルの列の個数と一致している必要があります。

テーブルの列の個数は、テーブルモデルが持っている getColumnCount というメソッドを使うことによって調べることができます。このメソッドは、レシーバーが管理しているテーブルの列の個数を戻り値として返します。

プログラムの例　PTableModel.java

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

public class PTableModel extends JFrame {
    final static int NUMROWS = 60;
    final static int NUMCOLUMNS = 4;
    JTable table;
    DefaultTableModel model;

    PTableModel(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        createTable();
        content.add(new JScrollPane(table),
            BorderLayout.CENTER);
        JPanel south = new JPanel(new GridLayout(1, 2));
        JButton insertButton = new JButton("行の挿入");
        insertButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
```

```
                insertRowToTable();
            }
        });
        south.add(insertButton);
        JButton removeButton = new JButton("行の削除");
        removeButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                removeRowFromTable();
            }
        });
        south.add(removeButton);
        content.add(south, BorderLayout.SOUTH);
        setVisible(true);
    }

    void createTable() {
        Vector columnnames = new Vector();
        for (int i = 0; i < NUMCOLUMNS; i++)
            columnnames.add(i + "列目");
        Vector rows = new Vector();
        for (int i = 0; i < NUMROWS; i++) {
            Vector columns = new Vector();
            for (int j = 0; j < NUMCOLUMNS; j++)
                columns.add(i + "行目の" + j + "列目");
            rows.add(columns);
        }
        table = new JTable(rows, columnnames);
        model = (DefaultTableModel)table.getModel();
    }

    void insertRowToTable() {
        int selected = table.getSelectedRow();
        if (selected >= 0) {
            Vector row = new Vector();
            for (int i = 0; i < model.getColumnCount(); i++)
                row.add("新しい行の" + i + "列目");
            model.insertRow(selected, row);
        } else
            JOptionPane.showMessageDialog(this,
                "行が選択されていません。",
                "エラー", JOptionPane.ERROR_MESSAGE);
    }

    void removeRowFromTable() {
        int selected = table.getSelectedRow();
        if (selected >= 0)
            model.removeRow(selected);
        else
            JOptionPane.showMessageDialog(this,
                "行が選択されていません。",
                "エラー", JOptionPane.ERROR_MESSAGE);
    }

    public static void main(String[] args) {
        PTableModel frame = new PTableModel("テーブルモデル");
    }
}
```

9.4　テーブルモデルのイベント

9.4.1　テーブルに対する変更のイベント

テーブルモデルは、自分が管理しているオブジェクトに対して何らかの変更が加えられたときにイベントを発生させます。ですから、そのイベントに対応するイベントリスナーをテーブルモデルに登録しておけば、テーブルに格納されているオブジェクトが変更されたときに何らかの動

作を実行することができるようになります。
テーブルモデルが発生させるイベントを取り扱いたいときは、javax.swing.event というパッケージの中で定義されている、

イベントリスナーを登録するメソッド	addTableModelListener
イベントリスナーのインターフェース	TableModelListener
イベント処理メソッド	tableChanged
イベントのオブジェクトのクラス	TableModelEvent

という名前のインターフェースとクラスとメソッドを使います。

9.4.2 変更された行と列のインデックス

tableChanged というイベント処理メソッドが引数として受け取るイベントのオブジェクトは、

getFirstRow	変更された最初の行のインデックスを返します。
getLastRow	変更された最後の行のインデックスを返します。
getColumn	変更された列のインデックスを返します。
getType	イベントのタイプをあらわす整数を返します。

というメソッドを持っています。ですから、これらのメソッドを使うことによって、変更されたのがどのオブジェクトなのかということと、どのようなタイプの変更が加えられたのかということを判定することができます。
なお、getType の戻り値は、TableModelEvent クラスの中で定義されている、

```
UPDATE   INSERT   DELETE
```

という定数を使って記述することができます。

プログラムの例 PTableModelEvent.java

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

public class PTableModelEvent extends JFrame {
    final static int NUMROWS = 60;
    final static int NUMCOLUMNS = 4;
    JTable table;
    JLabel label;

    PTableModelEvent(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        createTable();
        table.getModel().addTableModelListener(
            new TableModelListener() {
                public void tableChanged(TableModelEvent e) {
                    showChangedCell(e);
                }
            });
        content.add(new JScrollPane(table),
            BorderLayout.CENTER);
        label = new JLabel("まだ初期状態のままです。");
        content.add(label, BorderLayout.SOUTH);
        setVisible(true);
    }

    void createTable() {
        Vector columnnames = new Vector();
        for (int i = 0; i < NUMCOLUMNS; i++)
            columnnames.add(i + "列目");
```

```
        Vector rows = new Vector();
        for (int i = 0; i < NUMROWS; i++) {
            Vector columns = new Vector();
            for (int j = 0; j < NUMCOLUMNS; j++)
                columns.add(i + "行目の" + j + "列目");
            rows.add(columns);
        }
        table = new JTable(rows, columnnames);
    }

    void showChangedCell(TableModelEvent e) {
        if (e.getType() == TableModelEvent.UPDATE) {
            int row = e.getFirstRow();
            int column = e.getColumn();
            label.setText(row + "行目の" + column + "列目が" +
                table.getValueAt(row, column) +
                "に変更されました。");
        }
    }

    public static void main(String[] args) {
        PTableModelEvent frame = new PTableModelEvent(
            "テーブルモデルのイベント");
    }
}
```

9.5　セルレンダラー

9.5.1　列モデル

テーブルを構成するそれぞれの列は、「列モデル」(column model) と呼ばれるオブジェクトによって管理されています。列モデルは、TableColumnModel というインターフェースを実装したクラスから生成されるオブジェクトです。デフォルトでは、DefaultTableColumnModel というクラスの列モデルがテーブルに設定されています。

テーブルが持っている getColumnModel というメソッドは、レシーバーに設定されている列モデルを戻り値として返します。

なお、列モデルに関するインターフェースやクラスは、javax.swing.table というパッケージの中で定義されています。

9.5.2　列の属性をあらわすオブジェクト

列モデルには、テーブルを構成するそれぞれの列の属性をあらわしているオブジェクトが設定されています。列の属性をあらわすオブジェクトは、TableColumn というクラスのオブジェクトです。

列モデルが持っている getColumn というメソッドを呼び出すことによって、特定の列の属性をあらわしているオブジェクトを取得することができます。このメソッドは、引数として 1 個の整数を受け取って、その整数をインデックスとする列の属性をあらわしているオブジェクトを戻り値として返します。

9.5.3　セルレンダラーとは何か

列の属性をあらわすオブジェクトには、「セルレンダラー」(cell renderer) と呼ばれるコンポーネントを設定することができます。セルレンダラーというのは、セルを表示するために使われるコンポーネントのことです。

デフォルトでは、列の属性をあらわすオブジェクトに、セルレンダラーは何も設定されていません。この状態のとき、セルは、デフォルトの規則にしたがって表示されます。

セルレンダラーは、TableCellRenderer というインターフェースを実装したクラスを定義することによって、自由に作り出すことができます。

列の属性をあらわすオブジェクトに対してセルレンダラーを設定したいときは、そのオブジェクトが持っている setCellRenderer というメソッドを呼び出します。このメソッドは、引数としてセルレンダラーを受け取って、それをレシーバーに設定します。

9.5.4 標準的なセルレンダラー

javax.swing.table というパッケージの中には、DefaultTableCellRenderer という、標準的なセルレンダラーを生成するクラスがあります。このクラスのコンストラクタは、引数をまったく受け取らないものがひとつあるだけです。

DefaultTableCellRenderer クラスは、JLabel クラスのサブクラスですので、ラベルが持っているすべてのメソッドを継承しています。

9.5.5 水平方向の配置

DefaultTableCellRenderer クラスが JLabel クラスから継承しているメソッドのひとつに、表示する文字列の水平方向の配置を設定する setHorizontalAlignment というメソッドがあります。このメソッドは、引数として 1 個の整数を受け取って、その整数によって指定される水平方向の配置をレシーバーに設定します。引数として渡す整数は、

```
JLabel.LEFT   JLabel.CENTER   JLabel.RIGHT
```

という定数を使って記述することができます。

9.5.6 背景色と前景色

DefaultTableCellRenderer クラスのセルレンダラーは、背景色と前景色を設定する、

```
setBackground   setForeground
```

というメソッドも持っています。これらのメソッドは、引数として Color クラスのオブジェクトを受け取って、それがあらわしている色をレシーバーに設定します。

プログラムの例 PTableCellRenderer.java

```
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.util.*;

public class PTableCellRenderer extends JFrame {
    final static int NUMROWS = 60;
    final static int NUMCOLUMNS = 4;

    PTableCellRenderer(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JTable table = createTable();
        content.add(new JScrollPane(table),
            BorderLayout.CENTER);
        setVisible(true);
    }

    void setRenderer(JTable table) {
        DefaultTableCellRenderer renderer0 =
            new DefaultTableCellRenderer();
        renderer0.setHorizontalAlignment(JLabel.RIGHT);
        table.getColumnModel().getColumn(0)
            .setCellRenderer(renderer0);
        DefaultTableCellRenderer renderer1 =
            new DefaultTableCellRenderer();
        renderer1.setHorizontalAlignment(JLabel.CENTER);
        table.getColumnModel().getColumn(1)
            .setCellRenderer(renderer1);
        DefaultTableCellRenderer renderer2 =
            new DefaultTableCellRenderer();
        renderer2.setBackground(new Color(0, 0, 128));
        renderer2.setForeground(new Color(255, 255, 128));
        table.getColumnModel().getColumn(2)
            .setCellRenderer(renderer2);
    }
```

```
    JTable createTable() {
        Vector columnnames = new Vector();
        for (int i = 0; i < NUMCOLUMNS; i++)
            columnnames.add(i + "列目");
        Vector rows = new Vector();
        for (int i = 0; i < NUMROWS; i++) {
            Vector columns = new Vector();
            for (int j = 0; j < NUMCOLUMNS; j++)
                columns.add(i + ":" + j);
            rows.add(columns);
        }
        JTable table = new JTable(rows, columnnames);
        setRenderer(table);
        return table;
    }

    public static void main(String[] args) {
        PTableCellRenderer frame = new PTableCellRenderer(
            "セルレンダラー");
    }
}
```

9.5.7　独自のセルレンダラー

セルに格納されているオブジェクトを独自の方法で表示するためには、そのためのセルレンダラーを生成するクラスを定義する必要があります。

セルレンダラーを生成するクラスを定義するための正攻法は、TableCellRenderer というインターフェースを自分で実装することです。しかし、ここでは、もう少しお手軽な方法を紹介したいと思います。

それは、DefaultTableCellRenderer クラスのサブクラスを定義するという方法です。

DefaultTableCellRenderer クラスのオブジェクトは、setValue というメソッドを持っています。このメソッドは、セルに格納されているオブジェクト（型は Object）を引数として受け取って、それを文字列としてレシーバーに設定する、という動作をします。オブジェクトを独自の方法で表示するセルレンダラーを作りたいときは、このメソッドをサブクラスでオーバーライドします。

それでは、オブジェクトを独自の方法で表示するセルレンダラーの例として、文字列があらわしている整数を色の記述とみなして、その色を表示するセルレンダラーを作ってみましょう。

色を整数であらわしている文字列は、Color クラスが持っている decode というクラスメソッドを使うことによって、Color クラスのオブジェクトに変換することができます。このメソッドは、文字列を引数として受け取って、その文字列を Color クラスのオブジェクトに変換した結果を戻り値として返します。なお、このメソッドは、引数が整数をあらわしていない文字列だった場合、NumberFormatException という例外を発生させます。

プログラムの例　PColorRenderer.java

```
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.util.*;

class ColorRenderer extends DefaultTableCellRenderer {
    public void setValue(Object value) {
        Color color;
        try {
            color = Color.decode((String)value);
        } catch (NumberFormatException e) {
            color = Color.white;
        }
        setBackground(color);
    }
}

public class PColorRenderer extends JFrame {
```

```
    final static int NUMROWS = 60;
    final static int NUMCOLUMNS = 4;

    PColorRenderer(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JTable table = createTable();
        content.add(new JScrollPane(table),
            BorderLayout.CENTER);
        setVisible(true);
    }

    JTable createTable() {
        Vector columnnames = new Vector();
        for (int i = 0; i < NUMCOLUMNS; i++)
            columnnames.add(i + "列目");
        Vector rows = new Vector();
        for (int i = 0; i < NUMROWS; i++) {
            Vector columns = new Vector();
            for (int j = 0; j < NUMCOLUMNS; j++)
                if (j == 2)
                    columns.add("0x00ff00");
                else
                    columns.add(i + "行目の" + j + "列目");
            rows.add(columns);
        }
        JTable table = new JTable(rows, columnnames);
        table.getColumnModel().getColumn(2)
            .setCellRenderer(new ColorRenderer());
        return table;
    }

    public static void main(String[] args) {
        PColorRenderer frame = new PColorRenderer(
            "色を表示するセルレンダラー");
    }
}
```

9.6 セルエディター

9.6.1 セルエディターとは何か

列の属性をあらわすオブジェクトには、「セルエディター」(cell renderer) と呼ばれるコンポーネントを設定することができます。セルエディターというのは、セルを編集するために使われるコンポーネントのことです。

セルエディターは、TableCellEditor というインターフェースを実装したクラスを定義することによって、自由に作り出すことができます。

列の属性をあらわすオブジェクトに対してセルエディターを設定したいときは、そのオブジェクトが持っている setCellEditor というメソッドを呼び出します。このメソッドは、引数としてセルエディターを受け取って、それをレシーバーに設定します。

9.6.2 標準的なセルエディター

Swing には、DefaultCellEditor という、標準的なセルエディターを生成するクラスがあります。このクラスは、3 個のコンストラクタを持っています。それぞれのコンストラクタは、それぞれに異なる種類のコンポーネントを引数として受け取って、それを、セルを編集するためのコンポーネントとして設定します。コンストラクタに引数として渡すことのできるコンポーネントは、チェックボックス、コンボボックス、テキストフィールドの 3 種類です。

プログラムの例　PTableCellEditor.java

```
import javax.swing.*;
import javax.swing.table.*;
```

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

public class PTableCellEditor extends JFrame {
    final static int NUMROWS = 60;
    final static int NUMCOLUMNS = 4;

    PTableCellEditor(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JTable table = createTable();
        content.add(new JScrollPane(table),
            BorderLayout.CENTER);
        setVisible(true);
    }

    void setEditor(JTable table) {
        String[] items = {
            "Monday", "Tuesday", "Wednesday", "Thursday",
            "Friday", "Saturday", "Sunday" };
        table.getColumnModel().getColumn(2).setCellEditor(
            new DefaultCellEditor(new JComboBox(items)));
    }

    JTable createTable() {
        Vector columnnames = new Vector();
        for (int i = 0; i < NUMCOLUMNS; i++)
            columnnames.add(i + "列目");
        Vector rows = new Vector();
        for (int i = 0; i < NUMROWS; i++) {
            Vector columns = new Vector();
            for (int j = 0; j < NUMCOLUMNS; j++)
                if (j == 2)
                    columns.add("Monday");
                else
                    columns.add(i + "行目の" + j + "列目");
            rows.add(columns);
        }
        JTable table = new JTable(rows, columnnames);
        setEditor(table);
        return table;
    }

    public static void main(String[] args) {
        PTableCellEditor frame = new PTableCellEditor(
            "セルエディター");
    }
}
```

9.6.3　独自のセルエディター

セルに格納されているオブジェクトを独自の方法で編集するためには、そのためのセルエディターを生成するクラスを定義する必要があります。

セルエディターを生成するクラスを定義するためには、TableCellEditor というインターフェースを実装する必要があります。

Swing は、AbstractCellEditor という抽象クラスを持っています。この抽象クラスは、セルエディターのインターフェースを単純に実装したものです。セルエディターを生成するクラスは、この抽象クラスをスーパークラスにすることによって、比較的簡単に定義することができます。

AbstractCellEditor のサブクラスを定義することによってセルエディターを生成するクラスを作る場合、実装する必要のあるメソッドは、

```
getCellEditorValue   getTableCellEditorComponent
```

という二つだけです。

getCellEditorValue は、エディターによって編集された結果のオブジェクトを戻り値として返すメソッドで、

```
public Object getCellEditorValue() { ... }
```

という形の宣言を書くことによって定義します。

getTableCellEditorComponent は、セルを編集するために使われるコンポーネントを戻り値として返すメソッドで、

```
public Component getTableCellEditorComponent(
        JTable table, Object value, boolean isSelected,
        int row, int column) { ... }
```

という形の宣言を書くことによって定義します。このメソッドは、2 番目の引数として、編集の対象となったセルの値を受け取ります。

セルエディターを生成するクラスを定義するときに忘れてはいけないのは、編集が終了したときに、AbstractCellEditor クラスから継承した、fireEditingStopped というメソッドを呼び出さないといけない、ということです。これは、セルの編集が終了したというイベントを発生させるメソッドです。

プログラムの例 PColorEditor.java

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

class ColorRenderer extends DefaultTableCellRenderer {
    public void setValue(Object value) {
        setBackground((Color)value);
    }
}

class ColorEditor extends AbstractCellEditor
                  implements TableCellEditor {
    Color color;
    JButton button;

    ColorEditor() {
        button = new JButton("色の選択");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showColorChooser();
                fireEditingStopped();
            }
        });
    }

    void showColorChooser() {
        Color retval = JColorChooser.showDialog(null,
            "色の選択", color);
        if (retval != null)
            color = retval;
    }

    public Object getCellEditorValue() {
        return color;
    }

    public Component getTableCellEditorComponent(
            JTable table, Object value, boolean isSelected,
            int row, int column) {
        color = (Color)value;
        return button;
    }
```

```
}

public class PColorEditor extends JFrame {
    final static int NUMROWS = 60;
    final static int NUMCOLUMNS = 4;

    PColorEditor(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JTable table = createTable();
        content.add(new JScrollPane(table),
            BorderLayout.CENTER);
        setVisible(true);
    }

    JTable createTable() {
        Vector columnnames = new Vector();
        for (int i = 0; i < NUMCOLUMNS; i++)
            columnnames.add(i + "列目");
        Vector rows = new Vector();
        for (int i = 0; i < NUMROWS; i++) {
            Vector columns = new Vector();
            for (int j = 0; j < NUMCOLUMNS; j++)
                if (j == 2)
                    columns.add(new Color(0, 255, 0));
                else
                    columns.add(i + "行目の" + j + "列目");
            rows.add(columns);
        }
        JTable table = new JTable(rows, columnnames);
        TableColumn column =
            table.getColumnModel().getColumn(2);
        column.setCellRenderer(new ColorRenderer());
        column.setCellEditor(new ColorEditor());
        return table;
    }

    public static void main(String[] args) {
        PColorEditor frame = new PColorEditor(
            "色を選択するセルエディター");
    }
}
```

第 10 章　落穂拾い

10.1　アプレット

10.1.1　アプレットの基礎

ウェブのブラウザーによって実行される Java のバイトコードは、「アプレット」(applet) と呼ばれます。

Swing を使ってアプレットを作る場合、そのプログラムは、`JApplet` というクラスをスーパークラスとするサブクラスの宣言、という形になります。そのクラスのオブジェクトは、ブラウザーによって生成されますので、それを生成する記述をプログラムの中に書く必要はありません。また、`main` という名前のメソッドを宣言する必要もありません。ですから、

```
public class PEmpty extends javax.swing.JApplet { }
```

という 1 行のプログラムを書くことによって、何もしないアプレットを作ることができます。

10.1.2 アプレットの再描画

アプレットを実行するブラウザーは、アプレットが持っているメソッドを必要に応じて呼び出します。たとえば、アプレットの上のグラフィックスが消去されたときは、paint というメソッドを呼び出します。ですから、アプレットの場合も普通のコンポーネントと同じように、paint をオーバーライドすることによってグラフィックスの再描画を実行することができます。

プログラムの例 PApplet.java

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

public class PApplet extends JApplet {
    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(new Color(0, 128, 255));
        g.fillOval(50, 25, 300, 250);
    }
}
```

10.1.3 アプレットを取り付ける HTML の要素

ウェブページにアプレットを取り付けるためには、そのための要素を HTML 文書の中に書く必要があります。HTML 4.01 では、アプレットを取り付けるための要素として object というものを使うことが推奨されていますが、この要素はブラウザーによる対応が充分ではありませんので、普通はその代わりとして、applet という非推奨の要素が使われます。

applet 要素は、開始タグと終了タグを使って作ります。開始タグの中には、最低限、次の三つの属性に属性値を設定する属性指定を書く必要があります。

code	アプレットのファイル名
width	アプレットを表示する領域の横の長さ（単位はピクセル）
height	アプレットを表示する領域の縦の長さ（単位はピクセル）

たとえば、Namako.clsss というファイルに格納されているアプレットを、幅 400 ピクセル、高さ 500 ピクセルの大きさで表示したいとするならば、

```
<applet code="Namako.class" width="400" height="500">
</applet>
```

という applet 要素を書きます。

HTML 文書の例 applet.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html lang="ja">
<head>
<title>アプレット</title>
</head>
<body>
<applet code="PApplet.class" width="400" height="300">
</applet>
</body>
</html>
```

10.1.4 アプレットビューワー

Java の SDK には、「アプレットビューワー」(applet viewer) と呼ばれる、アプレットを動作させる機能だけを持つブラウザーが付属しています。アプレットビューワーを起動したいときは、

appletviewer HTML 文書の URL &

というコマンドをシェルに入力します。

10.1.5 ブラウザーによって呼び出されるメソッド

ブラウザーは、アプレットが持っているメソッドをさまざまなタイミングで呼び出します。そのような、ブラウザーによって呼び出されるメソッドとしては、paint のほかに次のようなもの

があります。

init	アプレットがロードされたとき
start	アプレットの実行が開始または再開されるとき
stop	アプレットの実行が中断されるとき
destroy	アプレットの実行が終了するとき

コンポーネントの追加やイベントリスナーの登録などの初期設定を実行したいときは、init というメソッドをオーバーライドします。

プログラムの例　PInitApplet.java

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

public class PInitApplet extends JApplet {
    final static Color COLOR = new Color(255, 128, 0);
    final static int RADIUS = 20;
    Vector points;

    public void init() {
        points = new Vector();
        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                points.add(e.getPoint());
                repaint();
            }
        });
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(COLOR);
        for (int i = 0; i < points.size(); i++) {
            Point p = (Point)points.get(i);
            g.fillOval(p.x - RADIUS, p.y - RADIUS,
                       RADIUS * 2, RADIUS * 2);
        }
    }
}
```

HTML 文書の例　init.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html lang="ja">
<head>
<title>アプレットの初期化</title>
</head>
<body>
<applet code="PInitApplet.class" width="400" height="300">
</applet>
</body>
</html>
```

10.1.6　HTML 文書からアプレットへの引数の受け渡し

HTML 文書は、任意の文字列を引数としてアプレットに渡すことができます。

引数をアプレットに渡したいときは、applet 要素の子供として param という要素を書きます。param は、単独タグによって作られる要素で、name という属性に引数の名前を設定して、value という属性に引数そのものを設定します。たとえば、applet 要素の子供として、

```
<param name="hayasa" value="630km/h">
```

という param 要素を書くことによって、630km/h という文字列に hayasa という名前を付けたも

のが、アプレットに引数として渡されます。

引数を受け取ったアプレットがその引数を使うためには、そのためのメソッドを呼び出す必要があります。

アプレットが持っている getParameter というメソッドは、1 個の文字列を引数として受け取って、HTML 文書から受け取った引数のうちで、その文字列を名前として持つものを戻り値として返します。たとえば、先ほど例として書いた param 要素で引数を受け取ったとするときに、

```
getParameter("hayasa")
```

という式で getParameter を呼び出したとすると、その戻り値として 630km/h という文字列が得られます。

プログラムの例　PGetParameter.java

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

public class PGetParameter extends JApplet {
    Color color;
    int radius;
    Vector points;

    public void init() {
        try {
            color = Color.decode(getParameter("color"));
        } catch (NumberFormatException e) {
            color = Color.black;
        }
        try {
            radius = Integer.parseInt(getParameter("radius"));
        } catch (NumberFormatException e) {
            radius = 100;
        }
        points = new Vector();
        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                points.add(e.getPoint());
                repaint();
            }
        });
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(color);
        for (int i = 0; i < points.size(); i++) {
            Point p = (Point)points.get(i);
            g.fillOval(p.x - radius, p.y - radius,
                       radius * 2, radius * 2);
        }
    }
}
```

HTML 文書の例　param.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html lang="ja">
<head>
<title>引数を受け取るアプレット</title>
</head>
<body>
<applet code="PGetParameter.class" width="400" height="300">
    <param name="color"  value="0x99ff00">
    <param name="radius" value="40">
</applet>
```

```
</body>
</html>
```

10.1.7　アプレットへのコンポーネントの追加

アプレットの上には、さまざまなコンポーネントを表示することができます。アプレットの上にコンポーネントを表示する方法は、フレームの場合とまったく同じです。つまり、コンテントペインをアプレットから取得して、それにコンポーネントを追加すればいいわけです。アプレットからコンテントペインを取得する方法や、コンテントペインにコンポーネントを追加する方法も、フレームの場合と同じです。

プログラムの例　PAppletComp.java

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

public class PAppletComp extends JApplet {
    public void init() {
        JPanel content = (JPanel)getContentPane();
        JButton button = new JButton("ボタン");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null,
                    "ボタンがクリックされました。",
                    "お知らせ",
                    JOptionPane.INFORMATION_MESSAGE);
            }
        });
        content.add(button, BorderLayout.SOUTH);
    }
}
```

HTML 文書の例　comp.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html lang="ja">
<head>
<title>アプレットへのコンポーネントの追加</title>
</head>
<body>
<applet code="PAppletComp.class" width="400" height="300">
</applet>
</body>
</html>
```

10.1.8　アプレットへのメニューの追加

フレームと同じように、アプレットの上端にはメニューバーを表示することができます。メニューバーを表示したいときは、フレームの場合と同じように、setJMenuBar というメソッドを呼び出して、引数としてメニューバーをそれに渡します。

プログラムの例　PAppletMenu.java

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

public class PAppletMenu extends JApplet {
    void showOrder(String order) {
        JOptionPane.showMessageDialog(null,
            order + "が注文されました。", "注文の確認",
            JOptionPane.INFORMATION_MESSAGE);
    }

    public void init() {
        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);
```

```
        JMenu food = new JMenu("料理");
        menuBar.add(food);
        JMenuItem okonomi = new JMenuItem("お好み焼き");
        okonomi.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showOrder("お好み焼き");
            }
        });
        food.add(okonomi);
        JMenuItem yakisoba = new JMenuItem("焼きそば");
        yakisoba.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showOrder("焼きそば");
            }
        });
        food.add(yakisoba);
    }
}
```

HTML 文書の例　menu.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html lang="ja">
<head>
<title>アプレットへのメニューの追加</title>
</head>
<body>
<applet code="PAppletMenu.class" width="400" height="300">
</applet>
</body>
</html>
```

10.1.9　ステータスバーによる文字列の表示

ブラウザーのウィンドウの中にある、1 行の文字列を表示することのできる細長い領域は、「ステータスバー」(status bar) と呼ばれます。アプレットは、ステータスバーに文字列を表示するという機能を持っています。

ステータスバーに文字列を表示したいときは、アプレットが持っている showStatus というメソッドを呼び出します。このメソッドに 1 個の文字列を引数として渡すと、その文字列がステータスバーに表示されます。

プログラムの例　PAppletStatus.java

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

public class PAppletStatus extends JApplet {
    Point position;

    public void init() {
        addMouseMotionListener(new MouseMotionAdapter() {
            public void mouseMoved(MouseEvent e) {
                position = e.getPoint();
                showStatus("pointer position: (" +
                    position.x + ", " + position.y + ")");
            }
        });
        position = new Point(0, 0);
    }
}
```

HTML 文書の例　status.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
```

```
<html lang="ja">
<head>
<title>ステータスバーによる文字列の表示</title>
</head>
<body>
<applet code="PAppletStatus.class" width="400" height="300">
</applet>
</body>
</html>
```

10.1.10　アプレット間通信

アプレットは、同じウェブページに取り付けられている別のアプレットを取得するという機能を持っています。この機能を利用することによって、アプレットからアプレットへメッセージを送信するということができます。

アプレットが別のアプレットを取得するためには、それに先立って、「アプレットコンテキスト」(applet context) と呼ばれるものを取得する必要があります。アプレットコンテキストというのは、アプレットが動作している環境をあらわすオブジェクトのことです。

アプレットコンテキストは、アプレットが持っている getAppletContext というメソッドを呼び出すことによって取得することができます。

アプレットコンテキストは、getApplet というメソッドを持っています。このメソッドは、アプレットの名前を引数として受け取って、その名前を持つアプレットを探します。そして、アプレットが見付かった場合はそれを戻り値として返します。見付からなかった場合は、戻り値として null を返します。

getApplet を使ってアプレットを取得するためには、そのアプレットに名前が付いている必要があります。アプレットの名前というのは、 applet 要素が持っている name という属性に設定された文字列のことです。たとえば、 applet 要素の開始タグの中に、

```
name="mimizu"
```

という属性指定を書くことによって、 mimizu という名前をアプレットに与えることができます。

アプレットからアプレットへメッセージを送ることができるようにしたいときは、受信側のアプレットのクラスで、メッセージを引数として受け取るメソッドを定義して、送信側でそのメソッドを呼び出します。ただし、getApplet の戻り値は Applet というクラス[1]のオブジェクトですので、それを継承したクラスで定義されているメソッドを呼び出すためには、それを本来のクラスにキャストする必要があります。

プログラムの例　PAppletReceiver.java

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

public class PAppletReceiver extends JApplet {
    JLabel label;

    public void init() {
        JPanel content = (JPanel)getContentPane();
        label = new JLabel();
        content.add(label, BorderLayout.CENTER);
    }

    public void receiveMessage(String s) {
        label.setText(s);
    }
}
```

プログラムの例　PAppletSender.java

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

public class PAppletSender extends JApplet {
```

[1] Applet というのは、 JApplet のスーパークラスです。

```
    JTextField namefield, messagefield;

    void sendMessage() {
        String name = namefield.getText();
        PAppletReceiver receiver =
            (PAppletReceiver)getAppletContext()
                .getApplet(name);
        if (receiver != null)
            receiver.receiveMessage(messagefield.getText());
        else
            JOptionPane.showMessageDialog(null,
                name + "は見付かりませんでした。", "エラー",
                JOptionPane.INFORMATION_MESSAGE);
    }

    public void init() {
        JPanel content = (JPanel)getContentPane();
        JPanel north = new JPanel(new GridLayout(2, 2));
        north.add(new JLabel("アプレット名: "));
        namefield = new JTextField();
        north.add(namefield);
        north.add(new JLabel("メッセージ: "));
        messagefield = new JTextField();
        north.add(messagefield);
        content.add(north, BorderLayout.NORTH);
        JButton sendbutton = new JButton("送信");
        sendbutton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                sendMessage();
            }
        });
        content.add(sendbutton, BorderLayout.SOUTH);
    }
}
```

HTML 文書の例　send.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html lang="ja">
<head>
<title>アプレット間通信</title>
</head>
<body>
<applet code="PAppletSender.class" width="400" height="300">
</applet>
<applet code="PAppletReceiver.class" width="400" height="300"
        name="receiver1">
</applet>
<applet code="PAppletReceiver.class" width="400" height="300"
        name="receiver2">
</applet>
</body>
</html>
```

10.2 ツリー

10.2.1 ツリーとは何か

ディレクトリやドメインのような階層的な構造は、植物の木に似ているので、「木構造」(tree) と呼ばれます。

何らかの木構造を持つものを操作するプログラムを書く場合は、その木構造を視覚的に反映したコンポーネントを表示すると、そのプログラムを使う人間に対して親切なものになります。そのような、木構造を視覚的に反映したコンポーネントは、「ツリー」(tree) と呼ばれます。

10.2.2 ノードとは何か

木構造を構成するそれぞれの要素は、その木構造の「ノード」(node) と呼ばれます。そして、異なる階層にある二つのノードのあいだに直接的な関係があるとき、上の階層にあるノードは下の階層にあるノードの「親」(parent) と呼ばれ、下の階層にあるノードは上の階層にあるノードの「子供」(child) と呼ばれます。また、木構造のもっとも上の階層にあるノードは、「ルートノード」(root node) と呼ばれます。

ひとつのノードには、任意のクラスのオブジェクトをひとつだけ設定することができます。ノードに設定されているオブジェクトは、そのノードの「ユーザーオブジェクト」(user object) と呼ばれます。

10.2.3 ノードの生成と子供の追加

Swing では、子供の追加や削除のできるノードは、`javax.swing.tree` というパッケージの中で定義されている `MutableTreeNode` というインターフェースを実装するクラスから生成されるオブジェクトです。このインターフェースを実装するクラスとしては、同じパッケージの中で定義されている `DefaultMutableTreeNode` という汎用的なクラスを使うのが普通ですが、目的に応じて自分で定義することも可能です。

`DefaultMutableTreeNode` クラスのコンストラクタとしては、普通、引数として 1 個のオブジェクトを受け取るものが使われます。このコンストラクタは、受け取った引数をユーザーオブジェクトとしてノードに設定します。

生成された直後のノードは、子供をひとつも持っていません。ノードに子供を追加したいときは、ノードが持っている `add` というメソッドを使います。このメソッドは、引数として 1 個のノードを受け取って、それをレシーバーに子供として追加します。

10.2.4 ツリーの生成

Swing では、ツリーは、`JTree` というクラスから生成されるコンポーネントです。

`JTree` クラスは、さまざまなコンストラクタを持っていて、それらのコンストラクタは、それぞれに異なった方法でツリーを初期化します。たとえば、引数として 1 個のノードを受け取るコンストラクタは、そのノードをルートノードとしてツリーに設定します。

10.2.5 ツリーの編集を可能にするかどうかの設定

ツリーは、デフォルトでは、それを構成するそれぞれのノードを編集することができないように設定されています。ツリーの編集を可能にするかどうかという設定を変更したいときは、それが持っている `setEditable` というメソッドを呼び出します。このメソッドは、引数として 1 個の真偽値を受け取って、それが真ならばレシーバーを編集できるように設定して、偽ならば編集できないように設定します。

編集できるように設定されているツリーのノードを実際に編集したいときは、まずそのノードを選択して、そののち、そのノードのユーザーオブジェクトの表示をクリックします。

プログラムの例 `PTree.java`

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

public class PTree extends JFrame {
    PTree(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JTree tree = new JTree(createTree(5, 4, "ノード"));
        tree.setEditable(true);
        content.add(
            new JScrollPane(tree), BorderLayout.CENTER);
        setVisible(true);
    }

    DefaultMutableTreeNode createTree(
            int level, int num, String user) {
```

```
        DefaultMutableTreeNode node =
            new DefaultMutableTreeNode(user);
        if (level > 1)
            for (int i = 1; i <= num; i++)
                node.add(createTree(
                    level - 1, num, user + "." + i));
        return node;
    }

    public static void main(String[] args) {
        PTree frame = new PTree("ツリー");
    }
}
```

10.2.6 ツリーのイベントリスナー

ツリーは、人間によって自分が操作されたときにイベントを発生させますので、それに対応するイベントリスナーをツリーに登録しておけば、ツリーが操作されたときに何らかの動作を実行することができるようになります。

ツリーが発生させるイベントを取り扱うインターフェースやクラスは、javax.swing.event というパッケージの中で定義されています。たとえば、ノードが選択されたときに何らかの動作を実行するプログラムを書きたいときは、そのパッケージの中で定義されている、

イベントリスナーを登録するメソッド	addTreeSelectionListener
イベントリスナーのインターフェース	TreeSelectionListener
イベント処理メソッド	valueChanged
イベントのオブジェクトのクラス	TreeSelectionEvent

という名前のインターフェースとクラスとメソッドを使います。

10.2.7 パス

ところで、ノードが選択されるというイベントが発生したとき、選択されたノードは、どうすれば求めることができるのでしょうか。

ルートノードから出発して、ツリーの枝に沿ってノードをたどっていくことによってできるノードの配列を持っているオブジェクトは、「パス」(path) と呼ばれます。パスは、TreePath というクラスから生成されます。

ノードが選択されたときに呼び出される valueChanged というメソッドは、そのイベントをあらわしているオブジェクトを引数として受け取ります。そのオブジェクトが持っている getPath というメソッドは、選択されたノードで終わるパスを戻り値として返します。ですから、選択されたノードは、getPath の戻り値から末尾のノードを取り出すことによって求めることができます。

パスは、getLastPathComponent というメソッドを持っています。このメソッドを使うことによって、パスが持っているノードのうちで、その末尾にあるものを取り出すことができます。

なお、ノードに設定されている文字列は、ノードが持っている toString というメソッドを呼び出すことによって求めることができます。

プログラムの例 PTreeSelection.java

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

public class PTreeSelection extends JFrame {
    JLabel label;

    PTreeSelection(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        JTree tree = new JTree(createTree(5, 4, "ノード"));
        tree.setEditable(true);
```

```
        tree.addTreeSelectionListener(
                new TreeSelectionListener() {
            public void valueChanged(TreeSelectionEvent e) {
                label.setText(
                    e.getPath().getLastPathComponent()
                        .toString());
            }
        });
        content.add(
            new JScrollPane(tree), BorderLayout.CENTER);
        label = new JLabel("未選択");
        content.add(label, BorderLayout.SOUTH);
        setVisible(true);
    }

    DefaultMutableTreeNode createTree(
            int level, int num, String user) {
        DefaultMutableTreeNode node =
            new DefaultMutableTreeNode(user);
        if (level > 1)
            for (int i = 1; i <= num; i++)
                node.add(createTree(
                    level - 1, num, user + "." + i));
        return node;
    }

    public static void main(String[] args) {
        PTreeSelection frame = new PTreeSelection(
            "ツリーの選択");
    }
}
```

10.3　プログレスバー

10.3.1　プログレスバーとは何か

かなりの時間を必要とする処理を実行するとき、プログラムはしばしば、その進捗状況を人間に知らせるために、進捗状況に応じて表示が変化する細長いコンポーネントを表示します。そのようなコンポーネントは、「プログレスバー」(progress bar) と呼ばれます。

Swing では、JProgressBar というクラスを使うことによって、プログレスバーを生成することができます。

10.3.2　最小値と最大値

JProgressBar クラスはいくつかのコンストラクタを持っていますが、ここでは、2 個の整数を受け取るものを紹介しましょう。このコンストラクタが受け取る 2 個の整数は、プログレスバーの表示を制御するための最小値と最大値です（1 個目が最小値で 2 個目が最大値)。たとえば、

```
new JProgressBar(200, 400)
```

という式で生成されたプログレスバーには、最小値として 200、最大値として 400 が設定されます。

プログレスバーに設定される最小値というのは、処理がまだまったく進んでいないという状況をあらわす整数のことで、最大値というのは、処理が終了したという状況をあらわす整数のことです。最小値と最大値のあいだの整数は、処理の途中だということをあらわします。

プログレスバーに設定されている最小値と最大値は、それが持っている、

`getMinimum`　最小値を戻り値として返す。

`getMaximum`　最大値を戻り値として返す。

というメソッドを使うことによって調べることができます。

10.3.3 進捗状況の表示

処理の進捗状況は、最小値と最大値とのあいだにある 1 個の整数によってあらわされます。進捗状況をあらわす整数をプログレスバーに設定すると、プログレスバーは、それがあらわしている進捗状況を自分の上に表示します。ですから、少しずつ大きな整数をプログレスバーに設定していくことによって、処理が進んでいるということを示すことができます。

進捗状況をあらわす整数は、プログレスバーが持っている setValue というメソッドを呼び出すことによって設定することができます。このメソッドは、引数として 1 個の整数を受け取って、それを現在の進捗状況としてレシーバーに設定します。

プログレスバーが持っている getValue というメソッドは、レシーバーに設定されている進捗状況をあらわす整数を戻り値として返します。

10.3.4 文字列の表示

プログレスバーは、自分の上に 1 個の文字列を表示することができます。ただし、プログレスバーは、デフォルトでは文字列を表示しないように設定されていますので、文字列を表示するためには、あらかじめその設定を変更しておく必要があります。

プログレスバーが持っている setStringPainted というメソッドは、引数として 1 個の真偽値を受け取って、それが真ならばレシーバーに対して文字列を表示するように設定して、偽ならば表示しないように設定します。

プログレスバーに表示させる文字列は、 setString というメソッドを使って設定します。このメソッドは、引数として 1 個の文字列を受け取って、それをプログレスバーの上に表示される文字列として設定します。

プログラムの例 PProgressBar.java

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

public class PProgressBar extends JFrame {
    Timer timer;
    JProgressBar progress;
    JButton start;
    int value;

    PProgressBar(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        progress = new JProgressBar(0, 100);
        progress.setStringPainted(true);
        content.add(progress, BorderLayout.NORTH);
        start = new JButton("スタート");
        start.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                start.setEnabled(false);
                progress.setValue(0);
                timer.start();
            }
        });
        content.add(start, BorderLayout.SOUTH);
        setVisible(true);
        timer = new Timer(100, new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                progressStep();
            }
        });
    }

    void progressStep() {
        int value = progress.getValue() + 1;
        if (value <= progress.getMaximum()) {
            progress.setValue(value);
```

```
            progress.setString(value + "%");
        } else {
            timer.stop();
            start.setEnabled(true);
        }
    }

    public static void main(String[] args) {
        PProgressBar frame = new PProgressBar(
            "プログレスバー");
    }
}
```

10.4 スライダー

10.4.1 スライダーとは何か

人間に整数を入力してもらうために使うことのできるコンポーネントとしては、すでに紹介したテキストフィールドやスピナーなどがありますが、それらよりもさらに直感的に整数を入力することのできるコンポーネントとして、「スライダー」(slider) と呼ばれるものがあります。

スライダーというのは、マウスで移動させることのできるノブを持つコンポーネントです。人間は、ノブの位置を変えることによって、その位置に対応している整数を入力することができます。

Swing では、JSlider というクラスを使うことによって、スライダーを生成することができます。

10.4.2 最大値と最小値と初期値

JSlider は、コンストラクタをいくつも持っています。ここでは、それらのうちで、3 個の整数を受け取るものを紹介したいと思います。引数として渡す 3 個の整数というのは、1 個目が最小値、2 個目が最大値、3 個目が初期値です。たとえば、

```
new JSlider(200, 500, 400)
```

という式でスライダーを生成したとすると、そのスライダーには、最小値として 200、最大値として 500、初期値として 400 が設定されます。

スライダーのノブの位置は、最小値と最大値とのあいだの整数に対応しています。スライダーが持っている getValue というメソッドは、現在のノブの位置に対応する整数を戻り値として返します。

10.4.3 スライダーのイベント

スライダーは、人間がスライダーのノブを移動させたときにイベントを発生させます。ですから、そのイベントに対応するイベントリスナーをスライダーに登録しておくことによって、人間がスライダーのノブを移動させたときに何らかの動作を実行することができます。

スライダーが発生させるイベントを取り扱いたいときは、javax.swing.event というパッケージの中で定義されている、

イベントリスナーを登録するメソッド	addChangeListener
イベントリスナーのインターフェース	ChangeListener
イベント処理メソッド	stateChanged
イベントのオブジェクトのクラス	ChangeEvent

という名前のインターフェースとクラスとメソッドを使います。

10.4.4 目盛りとラベル

スライダーは、目盛りを表示したり、それぞれの目盛りのそばに、それがあらわしている整数のラベルを表示したりすることもできます。

スライダーの目盛りやラベルは、デフォルトでは表示しない状態になっています。目盛りを表示させたいときは、それが持っている setPaintTicks というメソッドを呼び出して、引数として真偽値の真を渡す必要があります。

スライダーに目盛りを表示させるためには、さらに、目盛りの間隔を設定する必要があります。目盛りには、大目盛りと小目盛りの 2 種類があって、それぞれ、

大目盛り　`setMajorTickSpacing`

小目盛り　`setMinorTickSpacing`

というメソッドを呼び出すことによって設定することができます。これらのメソッドには、目盛りと目盛りとの間隔をあらわす整数を引数として渡します。たとえば、スライダーの最小値が 0 で最大値が 100 だとするときに、間隔として 20 を設定したとすると、0、20、40、60、80、100、という位置に目盛りが表示されます。

目盛りを表示させたとしても、デフォルトでは、目盛りがあらわしている整数のラベルは表示されません。それを表示させるためには、`setPaintLabels` というメソッドを呼び出して、引数として真偽値の真を渡す必要があります。

プログラムの例　`PSlider.java`

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

public class PSlider extends JFrame {
    final static int MIN  = 100;
    final static int MAX  = 200;
    final static int INIT = 120;
    JSlider slider;
    JLabel label;

    PSlider(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        createSlider();
        content.add(slider, BorderLayout.CENTER);
        label = new JLabel("value: " + INIT);
        content.add(label, BorderLayout.SOUTH);
        setVisible(true);
    }

    void createSlider() {
        slider = new JSlider(MIN, MAX, INIT);
        slider.addChangeListener(new ChangeListener() {
            public void stateChanged(ChangeEvent e) {
                label.setText("value: " + slider.getValue());
            }
        });
        slider.setPaintTicks(true);
        slider.setMajorTickSpacing(10);
        slider.setMinorTickSpacing(5);
        slider.setPaintLabels(true);
    }

    public static void main(String[] args) {
        PSlider frame = new PSlider("スライダー");
    }
}
```

10.5 ルックアンドフィール

10.5.1 ルックアンドフィールとは何か

GUI が視覚や操作性を通じて人間に与える印象のことを、その GUI の「ルックアンドフィール」(look and feel) と言います。Windows や Mac OS など、各種のオペレーティングシステムの GUI は、それぞれ固有のルックアンドフィールを持っています。

10.5.2 Swing とルックアンドフィール

Swing を使って作られた GUI は、デフォルトでは Metal と呼ばれるルックアンドフィールで表示されるのですが、Swing は、それ以外のルックアンドフィールで GUI を表示する機能も持っています。さらに、プログラムの実行中にルックアンドフィールを変更することも可能です。

Swing の GUI は、Metal と Motif という二つのルックアンドフィールを、Java のバイトコードが動作するどのオペレーティングシステムの上でも使うことができます。それに加えて、Windows の上では Windows に固有のルックアンドフィールを、Mac OS の上では Mac OS に固有のルックアンドフィールを使って GUI を表示することが可能です。

10.5.3 ルックアンドフィールを設定するメソッド

プログラムに対してルックアンドフィールを設定したいときは、

```
UIManager.setLookAndFeel
```

というクラスメソッドを呼び出します。この名前を持つメソッドは二つあって、どちらも、1 個の引数を受け取って、その引数によって指定されたルックアンドフィールをプログラムに対して設定します。ルックアンドフィールを指定するために渡す引数は、ルックアンドフィールを実装しているクラスの名前か、またはそのクラスのオブジェクトです。

前述の `Metal` と `Motif` は、それぞれ、

```
javax.swing.plaf.metal.MetalLookAndFeel
com.sun.java.swing.plaf.motif.MotifLookAndFeel
```

という名前のクラスによって実装されています。また、Windows に固有のルックアンドフィールは、

```
com.sun.java.swing.plaf.windows.WindowsLookAndFeel
```

という名前のクラスによって実装されていて、Mac OS に固有のルックアンドフィールは、

```
apple.laf.AquaLookAndFeel
```

という名前のクラスによって実装されています。ただし、Windows や Mac OS に固有のルックアンドフィールは、それぞれのオペレーティングシステムの上でしか使うことができません。

なお、引数としてクラス名を受け取る `setLookAndFeel` は、サポートされていないルックアンドフィールが指定された場合や、クラスが見付からなかった場合などは、それぞれの場合に応じた例外を発生させます。

10.5.4 ルックアンドフィールを表示に反映させるメソッド

ルックアンドフィールは、それをプログラムに設定した段階では、まだ GUI の表示には反映されません。設定されているルックアンドフィールを GUI の表示に反映させるためには、そのためのメソッドを呼び出す必要があります。

すべてのコンポーネントは、`updateUI` というメソッドを持っています。このメソッドは、プログラムに設定されているルックアンドフィールをレシーバーの表示に反映させる、という動作をします。しかし、GUI を構成しているひとつひとつのコンポーネントに対してこのメソッドを呼び出すというのはとても面倒ですので、Swing では、その作業を簡単にするために、

```
SwingUtilities.updateComponentTreeUI
```

というクラスメソッドを準備しています。このクラスメソッドは、引数としてコンテナを受け取って、そのコンテナの内部にあるすべてのコンポーネントについて、それが持っている `updateUI` を呼び出す、という動作をします。ですから、フレームをこのクラスメソッドに引数として渡すことによって、そのフレームの上に表示されているすべてのコンポーネントに対して、設定されているルックアンドフィールを反映させることができます。

プログラムの例 `PLookAndFeel.java`

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

public class PLookAndFeel extends JFrame {
    final static String[] lookandfeel = {
        "javax.swing.plaf.metal.MetalLookAndFeel",
```

```
        "com.sun.java.swing.plaf.motif.MotifLookAndFeel",
        "com.sun.java.swing.plaf.windows.WindowsLookAndFeel",
        "apple.laf.AquaLookAndFeel"
    };
    JComboBox combo;

    PLookAndFeel(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.setLayout(new FlowLayout());
        combo = new JComboBox(lookandfeel);
        content.add(combo);
        JButton button =
            new JButton("ルックアンドフィールの変更");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                changeLookAndFeel(
                    (String)combo.getSelectedItem());
            }
        });
        content.add(button);
        content.add(new JCheckBox("チェックボックス"));
        content.add(new JRadioButton("ラジオボタン"));
        setVisible(true);
    }

    void changeLookAndFeel(String cname) {
        try {
            UIManager.setLookAndFeel(cname);
            SwingUtilities.updateComponentTreeUI(this);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this,
                e.getClass().getName() + "\n" +
                e.getMessage(),
                "エラー", JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void main(String[] args) {
        PLookAndFeel frame = new PLookAndFeel(
            "ルックアンドフィール");
    }
}
```

10.6 ツールチップ

10.6.1 ツールチップの基礎

コンポーネントの上でマウスポインターをしばらく停止させておいたときに表示される、内容として 1 個の文字列を持つ長方形は、「ツールチップ」(tool tip) と呼ばれます。

Swing では、JComponent というクラスを継承しているクラスから作られたコンポーネントはすべて、ツールチップを表示する機能を持っています。

ツールチップに表示する文字列は、コンポーネントが持っている setToolTipText というメソッドを呼び出すことによって設定することができます。このメソッドに引数として文字列を渡すと、その文字列が、ツールチップの上に表示される文字列として設定されます。文字列ではなくて null を設定すると、ツールチップは表示しないという設定になります。デフォルトでは、文字列の代わりに null が設定されていますので、ツールチップは表示されません。

プログラムの例　PToolTip.java

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

```
public class PToolTip extends JFrame {
    PToolTip(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.setLayout(new FlowLayout());
        JButton button = new JButton("何もしない");
        button.setToolTipText(
            "このボタンは、クリックしても何もしません。");
        content.add(button);
        setVisible(true);
    }

    public static void main(String[] args) {
        PToolTip frame = new PToolTip("ツールチップ");
    }
}
```

10.6.2　ツールチップの上に表示する文字列

コンポーネントは、ツールチップを表示する直前に、getToolTipText というメソッドを呼び出します。そして、このメソッドが戻り値として返した文字列をツールチップの上に表示します。ですから、コンポーネントのクラスのサブクラスを定義して、このメソッドをオーバーライドすることによって、ツールチップの上に表示する文字列を、それを表示する直前に決定することができるようになります。

getToolTipText は、MouseEvent クラスのオブジェクト、つまりマウスのイベントをあらわすオブジェクトを引数として受け取ります。このオブジェクトは、現在のマウスポインターの位置を持っていますので、それを利用することによって、ツールチップの上に表示する文字列を、マウスの位置に応じて変化させることができます。

10.6.3　ツールチップを表示する位置

コンポーネントがツールチップを表示する位置は、getToolTipLocation というメソッドによって決定されます。これは、Point クラスのオブジェクトを戻り値として返すメソッドです。コンポーネントは、ツールチップを表示する直前にこのメソッドを呼び出して、その戻り値として得られた位置にツールチップを表示します。ですから、コンポーネントのクラスのサブクラスを定義して、このメソッドをオーバーライドすることによって、ツールチップを表示する位置を自由に変更することができます。

getToolTipLocation も、getToolTipText と同じように、マウスのイベントをあらわすオブジェクトを引数として受け取ります。ですから、この引数を使うことによって、ツールチップを表示する位置を、現在のマウスポインターの位置に対して相対的に指定することができます。

プログラムの例　PGetToolTip.java

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

class DoNothingButton extends JButton {
    DoNothingButton() {
        super("何もしない");
        setToolTipText("");
    }

    public String getToolTipText(MouseEvent e) {
        Point p = e.getPoint();
        return "(" + p.x + ", " + p.y + ")";
    }

    public Point getToolTipLocation(MouseEvent e) {
        Point p = e.getPoint();
        return new Point(p.x + 20, p.y + 10);
    }
```

```
}

public class PGetToolTip extends JFrame {
    PGetToolTip(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        DoNothingButton button = new DoNothingButton();
        content.add(button, BorderLayout.CENTER);
        setVisible(true);
    }

    public static void main(String[] args) {
        PGetToolTip frame = new PGetToolTip(
            "ツールチップの文字列と表示位置");
    }
}
```

10.7 ボーダー

10.7.1 ボーダーの基礎

コンポーネントとその周囲との境界線に沿って描画される直線は、「ボーダー」(border) と呼ばれます。

Swing では、JComponent というクラスを継承するクラスから作られたコンポーネントはすべて、自分の周囲にボーダーを描画する機能を持っています。

コンポーネントにボーダーを描画させたいときは、ボーダーを描画するオブジェクトをコンポーネントに設定します。ボーダーを描画するオブジェクトというのは、Border というインターフェースを実装するクラスのオブジェクトのことです。

ボーダーを描画するオブジェクトをコンポーネントに設定したいときは、そのコンポーネントが持っている setBorder というメソッドを呼び出して、そのオブジェクトを引数として渡します。

10.7.2 ボーダーを描画するオブジェクトの生成

BorderFactory というクラスは、ボーダーを描画するオブジェクトを生成して戻り値として返す、次のようなクラスメソッドを持っています。

createLineBorder	指定された色で、直線のボーダーを描画します。
createMatteBorder	指定された幅と、指定された色またはアイコンで、飾り縁のボーダーを描画します。
createTitledBorder	タイトルの付いたボーダーを描画します。
createCompoundBorder	二つのボーダーを組み合わせた複合ボーダーを描画します。
createEmptyBorder	指定された幅で、見えないボーダーを作ります。余白を作りたいときに使うと便利です。

プログラムの例　PBorderFactory.java

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

public class PBorderFactory extends JFrame {
    JLabel line, matte, titled, compound, empty;

    PBorderFactory(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        JPanel content = (JPanel)getContentPane();
        content.setLayout(new GridLayout(5, 1));
        createLabels();
        content.add(line);
```

```
        content.add(matte);
        content.add(titled);
        content.add(compound);
        content.add(empty);
        setVisible(true);
    }

    void createLabels() {
        line = new JLabel("createLineBorder");
        line.setBorder(BorderFactory.createLineBorder(
            Color.red));
        matte = new JLabel("createMatteBorder");
        matte.setBorder(BorderFactory.createMatteBorder(
            4, 8, 16, 32, Color.green));
        titled = new JLabel("createTitledBorder");
        titled.setBorder(BorderFactory.createTitledBorder(
            BorderFactory.createLineBorder(Color.magenta),
            "タイトル"));
        compound = new JLabel("createCompoundBorder");
        compound.setBorder(BorderFactory.createCompoundBorder(
            BorderFactory.createMatteBorder(
                6, 6, 6, 6, Color.blue),
            BorderFactory.createMatteBorder(
                6, 6, 6, 6, Color.yellow)));
        empty = new JLabel("createEmptyBorder");
        empty.setBorder(BorderFactory.createCompoundBorder(
            BorderFactory.createEmptyBorder(4, 8, 16, 32),
            BorderFactory.createLineBorder(Color.cyan)));
    }

    public static void main(String[] args) {
        PBorderFactory frame = new PBorderFactory("ボーダー");
    }
}
```

参考文献

[Campione,2001] Mary Campione, Kathy Walrath, and Alison Huml, *The Java Tutorial: A Short Course on the Basics, Third Edition*, Addison-Wesley, The Java Series, 2001, ISBN 0-201-70393-9. 邦訳（安藤慶一）、『Java チュートリアル・第 3 版』、ピアソンエデュケーション、The Java Series、2001、ISBN 4-89471-404-3。

[赤坂,2004] 赤坂玲音、『Java アプリケーション作成講座——Swing プログラミング徹底攻略——』、毎日コミュニケーションズ、2004、ISBN 4-8399-1386-2。

[大村,2002] 大村忠史、『Java GUI プログラミング——さらにパワーアップした Swing——』、Vol. I、カットシステム、2002、ISBN 4-87783-051-0。

[柏原,2002] 柏原正三、『Java GUI コンポーネント完全制覇』、技術評論社、2002、ISBN 4-7741-1551-7。

[丸の内,2001] 丸の内とら、『10 日でおぼえる Java 入門教室 Java 2 SDK 対応』、翔泳社、2001、ISBN 4-7981-0001-3。

索引