情報知能工学演習 V

アプレット

担当教官：村尾 元 (murao@al.cs.kobe-u.ac.jp)
TA：稲元 勉 (inamoto@al.cs.kobe-u.ac.jp)

Last Updated: 2002 年 7 月 5 日（金）

目 次

第1章　まずはアプレットを作ってみよう

1.1　アプレットとは？

アプレットとは，一般的に言えば，他のドキュメントに埋め込まれて利用される小さなプログラムのことである．特に Java のアプレットと言う場合には，HTML 文書，つまり WWW のページに埋め込まれて利用される Java プログラムのことを指す（図 1.1）．WWW のページに埋め込まれた Java のアプレットは，あたかもそのページの一部であるかのように表示され，グラフィックやアニメーションの表示，さらにはユーザの要求に応じて表示内容を変えるといったような，対話的な操作環境を提供する．

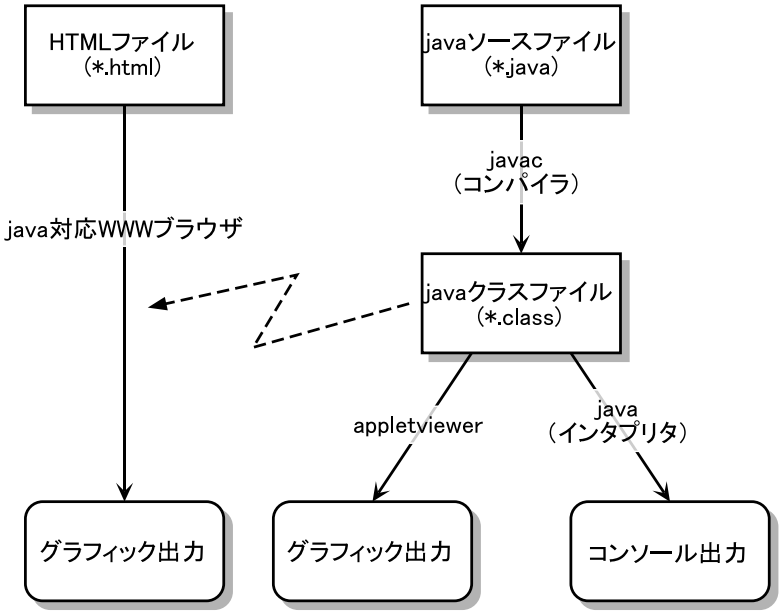


図 1.1: Java プログラムの実行形式

1.2　簡単なアプレットを作ってみよう

アプレットの作成には，プログラムを `Applet` クラスのサブクラスとして作成する必要がある．まず，画面に文字列「`Hello World`」と表示するアプレットの例を以下に示す.

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

public class HelloApplet extends Applet {
    public void init() {
        setBackground(Color.white);
    }

    public void paint(Graphics g) {
        g.drawString("Hello World", 50, 20);
        g.drawRect(0,0,getSize().width-1,getSize().height-1);
    }
}
```

このプログラムのコンパイルは通常の Java のクラスのコンパイルと同様に，以下のように行なう．

```
% javac HelloApplet.java Enter
```

1.3 アプレットを実行する

先に述べたように，アプレットは単独のプログラムではなく，HTML 文書に埋め込まれて実行されるプログラムなので，HTML 文書中に，アプレットを読み込んで実行するというタグを埋め込む必要がある．アプレットの読み込みと実行を指定するタグを以下に示す．

アプレットタグの書式

```
<APPLET CODE="実行するアプレット" WIDTH=300 HEIGHT=200>
</APPLET>
```

CODE の引数には実行するアプレット名を指定し，WIDTH および HEIGHT の引数には，アプレットの大きさを数値で指定する．APPLET タグで囲まれた間には，アプレットに渡すパラメータを記述することができるが，ここでは省略している．

例えば，先のアプレット HelloApplet を埋め込んだ HTML 文書は以下のように記述することができる．

```
<HTML>
<HEAD><TITLE>HelloApplet の実験</TITLE></HEAD>

<BODY BGCOLOR=#FFFFFF>
<H1>HelloApplet の実験</H1>

<APPLET CODE="HelloApplet" WIDTH=300 HEIGHT=200>
</APPLET>

</BODY>
</HTML>
```

これを実行したいアプレットと同一のディレクトリに作成し，Java アプレットに対応した WWW ブラウザを用いて表示させると指定されたアプレットが実行される．手軽にアプレットのテストをしたいのであれば appletviewer というアプレット専用のブラウザを用いることもできる．例えば上の HTML 文書の名前を HelloApplet.html とすると，以下のように実行する．

```
% appletviewer HelloApplet.html Enter
```

演習

上記の HelloApplet.java と HelloApplet.html を入力し，アプレットを実行してみよう．

第2章　アプレットの構造

2.1　アプレットが依存するパッケージ

Applet クラスは java.applet パッケージ内に属するのでこれを import する必要がある．また，多くの場合，アプレットはブラウザの画面上にグラフィックを表示することになる．グラフィック表示のための基本的なクラスを提供するのが java.awt パッケージである．AWT は Abstract Window Toolkit の略で，このパッケージには，グラフィック描画やイベント処理などユーザと対話的に処理するためのクラスが集められている．これらのパッケージを読み込むには，アプレットのファイル (*.java ファイル) の先頭で以下のように記述する．

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

2.2　インスタンスのメソッド

アプレットには init(), start(), stop()，paint()，destroy() の 5 つのインスタンスメソッドが用意されており，サブクラスでこれらをオーバーライドすることにより自分のアプレットを作成する．

init() メソッド

書式 : `public void init() { ... }`

アプレットが読み込まれた時，最初に一回だけ実行される．このメソッドの中では，メンバ変数の初期化やアプレットの大きさなどの変更などを行なう．

start() メソッド

書式 : `public void start() { ... }`

ブラウザの起動や再読込によって，アプレットの実行が開始されるときに呼び出される．

stop() メソッド

書式 : `public void stop() { ... }`

他の Web ページに移動するなどしてアプレットの実行が停止されるときに呼び出される.

paint() メソッド

書式 : `public void paint(Graphics g) { ... }`

アプレットの描画を行なうメソッド. アプレットが最初に表示されたときに実行されるほか, ブラウザを隠していた他のウィンドウが閉じられて, ブラウザの内容の再表示が必要になった場合にも実行される.
`paint()` メソッドには, `Graphics` クラスのインスタンスが引数として渡される. これはアプレットの表示部に対応しており, このインスタンスのメソッドを呼ぶ事により, ブラウザにグラフィックを表示する.
他のメソッド中でアプレットの再表示が必要になった時には `repaint()` メソッドを呼び出すことにより適当なタイミングで `paint()` が実行され再描画が行われる.

destroy() メソッド

書式 : `public void destroy() { ... }`

アプレットの終了時に一度だけ実行される. ここでは, ファイルを閉じるなど, プログラムの終了時に必要な処理を行なう.

その他に使えるメソッドとして `repaint()` メソッドがある. このメソッドを呼び出すと画面を消去してから `paint()` メソッドが呼び出されるので, 強制的に再描画したい場合に呼び出すと良い.

2.3　画面への描画

ブラウザ上のアプレット表示領域にグラフィックや文字列を描画するには, `paint()` メソッドの引数で与えられた `Graphics` クラスのインスタンスの描画メソッドを呼びだす. 以下に代表的な幾つかのメソッドを示す.

drawLine() メソッド

書式 : `drawLine(int x1,int y1,int x2,int y2);`

座標 `(x1,y1)` から `(x2,y2)` への直線を描く.

drawString() メソッド

書式 : `drawString(String str, int x, int y);`

座標 `(x,y)` の位置に `str` で指定される文字列を表示する．文字列の座標の原点は下図に示すように左下である.

drawRect() メソッド

書式 : `drawRect(int x,int y,int width,int height);`

座標 `(x,y)` を左上として，幅 `width`，高さ `height` の長方形を描く.

drawOval() メソッド

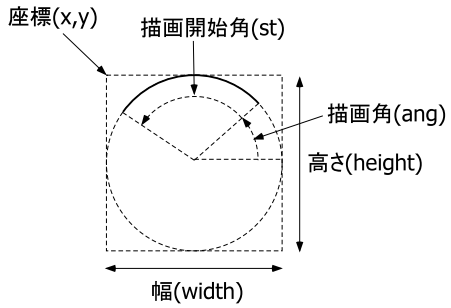
書式 : `drawOval(int x,int y,int width,int height);`

座標 `(x,y)` を左上とする幅 `width`，高さ `height` の長方形に内接する楕円を描く.

drawArc() メソッド

書式 : `drawArc(int x,int y,int width,int height,int st,int ang);`

座標 (x,y) を左上とする幅 width，高さ height の長方形に内接する楕円を，角度 st 度から始め，ang 度ぶん描画する．



setColor() メソッド

書式 : `setColor(Color c);`

描画色を c で指定される色にする．c には，以下の表に示す予め決められた色を指定しても良いし，新たに Color クラスのインスタンスを作成してこれを指定しても良い．

名前	色	名前	色	名前	色	名前	色
black	黒	blue	青	cyan	シアン	gray	灰色
darkGray	濃い灰色	green	緑	lightGray	明るい灰色	magenta	マゼンタ
orange	オレンジ	pink	ピンク	red	赤	yellow	黄色
white	白						

例えば，直線をオレンジ色で描画する場合には以下のように記述する．

```
public void paint(Graphics g) {
    g.setColor(Color.orange);
    g.drawLine(x1,y1,x2,y2);
}
```

さらに drawRect(), drawOval(), drawArc() に対応して，それぞれ fillRect(), fillOval(), fillArc() という塗り潰しをするメソッドも用意されている．

表示領域の大きさは，変更しない限りは HTML 文書の <APPLET>タグで指定された WIDTH および HEIGHT によって決定される．座標系は図 2.1 に示すように，左上を原点 (0,0) として右方向が x 軸の正，下方向が y 軸の正となる．

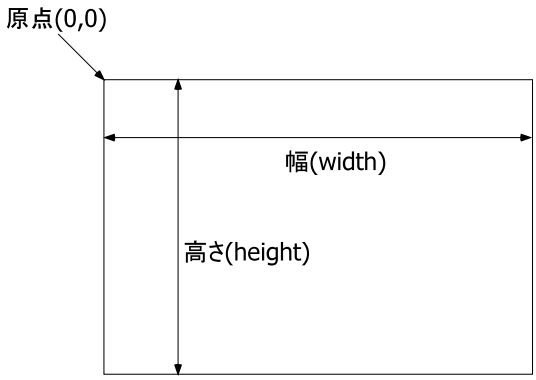


図 2.1: アプレットの描画領域

アプレットの表示領域の大きさはアプレットのメソッド `getSize()` で知ることができる．返り値は `Dimension` クラスのインスタンスで，そのメンバ `width` および `height` により幅および高さを知ることができる．以下に例を示す．

```
public void paint(Graphics g) {
    Dimension d = getSize();
    g.drawRect(0,0,d.width-1,d.height-1);
}
```

その他の描画メソッドについては，

`http://java.sun.com/products/jdk/1.3/ja/docs/ja/api/java/awt/Graphics.html`

などを参照のこと．

演習

- 本文で紹介した `Graphics` クラスのメソッドのうち，少なくとも 4 種類のメソッドを一回づつ用い，何か意味のある絵を表示するアプレットを作成しよう．
- 絵の左上に学番と名前および絵のタイトルを `drawString()` メソッドを用いて表示しよう．
- 作成したアプレットを

  `http://www.csedu.kobe-u.ac.jp/home/t00xxxxx/java/e5.html`

  という URL で参照できるように HTML 文書を作成し，公開してみよう．

第3章　イベントの取り扱い

3.1　イベント駆動型プログラム

これまでに作成してきたような，コンソールで実行される文字ベースのプログラムでは，プログラムが入力を必要とするときにプロンプトを表示して入力待ちの状態になれば良かったが，アプレットのようにグラフィックを利用したプログラムでは，ウィンドウが画面上に表示されている間，常にキーボードやマウスからの入力を受け付ける必要がある．

このようなプログラムでは，ユーザからの入力などを**イベント**と呼び，これに応じて動作するように作成する．これをイベント駆動型プログラム（もしくは” イベントドリブン : Event Driven”）と言う．

イベント駆動型プログラムでは，イベントを監視しながら，なおかつ内部では計算などの作業を行う必要があるため，文字ベースのプログラムに比べてプログラミングそのものが非常に困難となる．

そこで Java のアプレットでは，演習 IV ですでに学んだスレッドを用いて，これをシンプルに解決している．すなわち，専用の特殊なスレッドが常にイベントを監視し，イベントが発生した場合には，これに応じたメソッドを呼び出してくれるというものである．もし内部で別の作業を行う必要があるならば，それに割り当てるスレッドを新たに生成すれば良い．

イベントを監視するスレッドは，アプレットの開始時に自動的に開始されるので，プログラムを作成するときにはこれを意識する必要はない．このスレッドは，イベントが発生すると，これに対応する**リスナー**がアプレットに備わっているかを調べ，備わっていればリスナーを呼び出す．このリスナーについては次の章で説明しよう．

3.2　イベントとリスナーのクラス

コンピュータの GUI（Graphical User Interface）は常に新しくなっており，将来，どのようなイベントの処理がプログラムに要求されるかは分からないため，`Applet` クラスには特定のイベントを監視し，その要求を受け取るという機能は備わっていない[1]．

その代わりにリスナーと呼ばれるインターフェイスを利用する．リスナーはイベントに対応して，これを受け取るメソッドを持っている．表 3.1 にイベントとそれに対応するイベントクラスおよびリスナー，リスナーが有するメソッドを列挙する．

[1] Java1.0 まではアプレットが特定のイベントを受け取るメソッドを持っており，これを実装することでイベント処理を行うことができた．この方法は記述が簡単ではあるが，柔軟性に欠けるため Java1.1 以降では推奨されない．

表 3.1: Java1.1 以降におけるイベントとリスナの対応

イベントクラス	リスナインターフェイス	リスナメソッド
ActionEvent	ActionListener	actionPerformed()
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged()
ComponentEvent	ComponentListener	componentHidden() componentMoved() componentResized() componentShown()
ContainerEvent	ContainerListener	componentAdded() componentRemoved()
FocusEvent	FocusListener	focusGained() focusLost()
ItemEvent	ItemListener	itemStateChanged()
KeyEvent	KeyListener	keyPressed() keyReleased() keyTyped()
MouseEvent	MouseListener	mouseClicked() mouseEntered() mouseExited() mousePressed() mouseReleased()
	MouseMotionListener	mouseDragged() mouseMoved()
TextEvent	TextListener	textValueChanged()
WindowEvent	WindowListener	windowActivated() windowClosed() windowClosing() windowDeactivated() windowDeiconified() windowIconified() windowOpened()

イベントが発生した際には，イベントクラスのインスタンスがリスナメソッドに引数として渡され，これに基づいて必要な情報を得ることができる．例えば，MouseEvent クラスは getX()，getY() というメソッドを有しており，イベントの発生した時点におけるマウスカーソルの位置を調べることができる．

3.3　イベント処理のプログラム例

実際の利用法を見たほうが良いだろう．次に簡単なお絵かきソフトのプログラムを示す．

```
 1: import java.applet.*;
 2: import java.awt.*;
 3: import java.awt.event.*;
 4:
 5: public class Scribble extends Applet
 6:                       implements MouseListener, MouseMotionListener {
 7:     private int lx, ly;
 8:
 9:     public void init() {
10:         this.addMouseListener(this);
11:         this.addMouseMotionListener(this);
12:     }
13:
14:     // MouseLisnter インターフェイスのメソッド
15:     // ユーザがボタンを押したときに呼び出される
16:     public void mousePressed( MouseEvent e ) {
17:         lx = e.getX();
18:         ly = e.getY();
19:     }
20:
21:     // MouseMotionLisnter インターフェイスのメソッド
22:     // ユーザがボタンを押したままマウスをドラッグしたときに呼び出される
23:     public void mouseDragged( MouseEvent e ) {
24:         Graphics g = this.getGraphics();
25:         int x = e.getX();
26:         int y = e.getY();
27:         g.drawLine(lx, ly, x, y);
28:         lx = x;
29:         ly = y;
30:     }
31:
32:     // MouseLisnter インターフェイスのメソッド
33:     // ここでは利用しないもの
34:     public void mouseReleased(MouseEvent e) {}
35:     public void mouseClicked(MouseEvent e) {}
36:     public void mouseEntered(MouseEvent e) {}
37:     public void mouseExited(MouseEvent e) {}
38:
39:     // MouseMotionLisnter インターフェイスのメソッド
40:     // ここでは利用しないもの
41:     public void mouseMoved(MouseEvent e) {}
42: }
```

以下，解説．

6 行目 MouseListener と MouseMotionListener のインターフェイスの実装を宣言している．これにより，この Scribble クラスがこれらのリスナーに対応するイベントの取得を行えることになる．

10,11 行目 イベントのリスナーが自分自身（this）であることを，イベント監視スレッドに登録する．X というイベントに関するリスナーの登録は，以下のような形式で行う．

```
イベントを発生するオブジェクト.addXListner(イベントのリスナー)
```

ここでは Applet クラスそのものがマウスに関するイベントを発生するので,「イベントを発生するオブジェクト」も「イベントのリスナー」もそのアプレット自身（this）となっている．

この後の章で，AWT コンポーネントという，GUI 部品の利用を行うが，これらの部品はそれぞれ対応するイベントを発生することができる．例えば Button クラスはユーザがボタンをクリックしたときに ActionEvent を発生し，Scrollbar クラスはユーザがスクロールバーを操作したときに AdjustmentEvent を発生する．これらについては後ほど紹介する．

16 行目 MouseListener インターフェイスのメソッドである mousePressed() メソッドの実装を行っている．ここで分かるように，リスナーのメソッドには，対応するイベントクラスのインスタンスが引数として渡される（この場合は MouseEvent クラスのインスタンス）．メソッド内では，これを用いてイベントの処理を行うことができる．

34〜41 行目 MouseListener および MouseMotionListener は複数のメソッドを有するが，ここで利用するメソッドは mousePressed() と mouseDragged() のみである．しかし，これらのメソッドはすべて抽象メソッドとして宣言されているので，利用しないメソッドについてもこのように空のメソッドとして実装する必要がある．

これは Java1.1 におけるイベント処理において若干煩雑なところであるが，次に述べる**アダプタ**を利用することで解消することができる．

演習

Scribble プログラムをコンパイル実行し，イベント処理について理解しよう．対応する HTML ファイルが必要になることに注意しよう．

3.4　アダプタの利用

さて，リスナーを実装するたびに利用しない複数のメソッドまで記述しなくてはならいのは手間と言えるだろう．この問題を解決する方法としてリスナーに対応した**アダプタ**というクラスがある．

アダプタは対応するリスナーのメソッドを全て空のメソッドで実装したクラスであり，これを拡張（extends）して，必要なメソッドだけをオーバーライドすることでイベントを処理することができる[2]．

[2]全てのリスナーに対応したアダプタが用意されているわけではないことに注意しよう．概ね，複数のメソッドを持つリスナーにはアダプタが用意されているが，単一のメソッドしか持たないリスナー（例えば ActionListener など）はアダプタが用意されていない．

上のお絵かきプログラムをアダプタを利用して書き換えてみよう．この場合，利用している二つのリスナー（MouseListener および MouseMotionListener）に対応したアダプタを拡張したクラスが必要となるため，クラスは全部で 3 つとなる．ここでは Scribble2 と MyMouseListener，MyMouseMotionListener という名前とした．以下にプログラムを示す．

1. Scribble2 クラス

   これはアプレット本体のクラスである．プログラムは init() メソッドのみからなっており，特に難しいところはないだろう．

```
 1: import java.applet.*;
 2: import java.awt.*;
 3: import java.awt.event.*;
 4:
 5: public class Scribble2 extends Applet {
 6:     int lx, ly;
 7:
 8:     public void init() {
 9:         MouseListener ml = new MyMouseListener(this);
10:         MouseMotionListener mml = new MyMouseMotionListener(this);
11:
12:         this.addMouseListener(ml);
13:         this.addMouseMotionListener(mml);
14:     }
15: }
```

   9,10 行目 二つのリスナーに対応したアダプタのインスタンスを生成する．

   12,13 行目 このインスタンスを，アプレットにおいて発生したイベントの受け取り人として登録する．

2. MyMouseListener クラス

   これは MouseListener に対応するクラスで，MouseAdapter クラスを拡張して作成されている．

```
 1: import java.applet.*;
 2: import java.awt.*;
 3: import java.awt.event.*;
 4:
 5: class MyMouseListener extends MouseAdapter {
 6:     private Scribble2 sc;
 7:
 8:     public MyMouseListener(Scribble2 s) {
 9:         sc = s;
10:     }
11:
12:     public void mousePressed(MouseEvent e) {
13:         sc.lx = e.getX();
14:         sc.ly = e.getY();
15:     }
16: }
```

3. MyMouseMotionListener クラス

これも MyMouseListener クラスと同様で，MouseMotionAdapter クラスを拡張して作成されている．

```
 1: import java.applet.*;
 2: import java.awt.*;
 3: import java.awt.event.*;
 4:
 5: class MyMouseMotionListener extends MouseMotionAdapter {
 6:     private Scribble2 sc;
 7:
 8:     public MyMouseMotionListener(Scribble2 s) {
 9:         sc = s;
10:     }
11:
12:     public void mouseDragged(MouseEvent e) {
13:         Graphics g = sc.getGraphics();
14:         int x = e.getX();
15:         int y = e.getY();
16:         g.drawLine(sc.lx, sc.ly, x, y);
17:         sc.lx = x;
18:         sc.ly = y;
19:     }
20: }
```

上記のアダプタを用いたプログラムは，記述が簡単になり見通しがよくなる一方で，以下のような問題点がある．

- リスナーの初期化時にアプレットのインスタンスを渡す必要がある（`MyMouseListener` クラス：8〜10 行目，`MyMouseMotionListener` クラス：8〜10 行目）こと．
- アプレットの内部変数に，リスナーからアクセスする必要がある（`MyMouseListener` クラス：13,14 行目，`MyMouseMotionListener` クラス：16〜19 行目）こと

これを解決するための方法として Java1.1 から導入された**内部クラス**を利用することができる．

演習

- 上記の `Scribble2` プログラムをコンパイル実行し，イベント処理について理解しよう．実行には対応する HTML ファイルが必要になることに注意しよう．
- 2.3 章の演習で作成したアプレットを以下のように改造しよう．

絵を二枚以上用意し，画面の特定の位置（ユーザに分かるようにする）をクリックすると切り替わるようにする

3.5 内部クラスを用いた方法（いずれ解説予定）

内部クラスについてはまだ演習中で説明をしていないので，ここでは内部クラスを用いたプログラムを示すにとどめ，解説は次回以降に行うことにする．

```
 1: import java.applet.*;
 2: import java.awt.*;
 3: import java.awt.event.*;
 4:
 5: public class Scribble3 extends Applet {
 6:     private int lx, ly;
 7:
 8:     public void init() {
 9:         this.addMouseListener(new MouseAdapter() {
10:             public void mousePressed(MouseEvent e) {
11:                 lx = e.getX();
12:                 ly = e.getY();
13:             }
14:         });
15:
16:         this.addMouseMotionListener(new MouseMotionAdapter() {
17:             public void mouseDragged(MouseEvent e) {
18:                 Graphics g = getGraphics();
19:                 int x = e.getX();
20:                 int y = e.getY();
21:                 g.drawLine(lx, ly, x, y);
22:                 lx = x;
23:                 ly = y;
24:             }
25:         });
26:     }
27: }
```

第4章　AWTコンポーネントの利用

4.1　AWTコンポーネント

GUIコンポーネントは，Javaで利用できるGUI作成用の部品の総称である．その中でも特に，AWTコンポーネントとは，AWTパッケージにおいて定義された一連のGUI部品を指す．これを用いることにより，より直感的なユーザインターフェイスを，容易に実現することができる．AWTの他にも，有名なところではSwingなど，Javaには非常に多くのGUI部品が提供されているが，ここではAWTに含まれるGUI部品の代表的なものを紹介する．より自分の目的にあったGUI部品についてはライブラリのマニュアルなどを参考にして欲しい．

4.2　AWTコンポーネントの利用

Javaで GUI部品を利用するには以下の4つのステップを踏む．

1. **コンポーネントの作成**：GUIコンポーネントは他のJavaのオブジェクトと同様に，コンストラクタを呼び出して作成する．コンストラクタがどのような引数を必要とするかについて，ここに全てを書き下すことは意味がないのでオンラインマニュアルなどを参照して欲しい．

   例えば，”Quit” を表示する `Button` コンポーネントを作成するには次のように行う．

   ```
   Button quit = new Button("Quit");
   ```

   コンポーネントは通常，アプレットの `init()` メソッド内で作成される．

2. **コンテナにコンポーネントを追加**：コンテナはGUI部品を配置するための見えない枠のようなものであり，全てのコンポーネントはコンテナの内部に置く必要がある．Javaにおけるコンテナは全て `java.awt.Container` のサブクラスである．これまでに利用している `Applet` もコンテナである．コンテナはそれ自身がコンポーネントなので，コンテナの中にコンテナを配置（ネスト）することにより，より複雑な利用法が可能である．

   コンテナには，`add()` メソッドを用いてコンポーネントを追加する．例えば，アプレットに `quit()` ボタンを追加するには，次のように記述する．

   ```
   this.add(quit)   /* this はなくても良い */
   ```

   `LayoutManager` を用いてコンポーネントの配置を自動的に決定する場合には，`add()` メソッドに，さらにコンテナ内での位置などを指定する引数が加わる場合もある．

   コンテナへのコンポーネントの追加もアプレットの `init()` メソッド内で行うのが一般的である．

3. **コンテナ内部のコンポーネントの編成あるいは配置**：続いて，コンポーネントの位置と大きさを設定し，GUIの外観を調整する．`setBounds()` メソッドを用いて，各コンポーネントの位置と大きさを，直接指定することもできるが，多くの場合は `LayoutManager` を用いて自動的に配置する．

4. **コンポーネントが発生するイベントの処理**：GUI コンポーネントは，第 3 章で学んだマウス操作やキーボード操作といった低レベルのイベントに応じて，それぞれに定義されたイベントを発生する．例えば，`Button` オブジェクト上でマウスボタンをクリックすると，その `MouseEvent` は，ボタンに `ActionEvent` というイベントを発生させる．

4.3　GUI コンポーネント

4.3.1　AWT に含まれる GUI コンポーネント

AWT に含まれるコンテナを除く GUI 部品の一部を表 4.1 に示す．GUI 部品の完全なリストとその利用法については Java のリファレンスなどを参照して欲しい．

表 4.1: AWT コンポーネント

コンポーネント	説明
`Button`	プッシュ式ボタン
`Canvas`	描画やサブクラスをカスタマイズするのに適した空のコンポーネント
`CheckBox`	選択と選択解除ができるトグルシキボタン
`Choice`	ドロップダウンリストあるいはオプションメニュー
`Label`	一行のテキストを表示
`List`	選択可能な項目のリスト
`Scrollbar`	スクロール用のスライダ
`TextArea`	テキストの表示，編集に使用する複数行のテキスト領域
`TextField`	一行のテキスト入力領域

4.3.2　GUI コンポーネントの利用例

以下に様々な GUI コンポーネントをアプレットに埋め込んだ例を示す．アプレットはそれ自身がコンテナなので，そのまま GUI コンポーネントを追加できることに注意しよう．

```
 1: import java.awt.*;
 2: import java.applet.*;
 3:
 4: public class GuiExample extends Applet {
 5:     Button okButton;          // クリックボタン
 6:     TextField nameField;      // テキストを入力する textField
 7:     CheckboxGroup radioGroup; // ラジオボタン グループ
 8:     Checkbox radio1, radio2;  // radioGroup 用の選択するラジオボタン
 9:     Checkbox option;          // 独立した選択ボックス
10:
11:     public void init() {
12:         okButton = new Button("A button"); // ボタンの初期化
13:
14:         // テキストとフィールドの長さ
15:         nameField = new TextField("A TextField",100);
16:
17:         // ラジオボタングループを初期化する
18:         radioGroup = new CheckboxGroup();
19:
20:         // 最初のラジオボタン。ラベルテキストを与え、
21:         // どのグループに属するか教え，デフォルト状態を設定する
22:         radio1 = new Checkbox("Radio1", radioGroup, false);
23:         // 選択状態で設定
24:         radio2 = new Checkbox("Radio2", radioGroup, true);
25:
26:         // チェックボックスのラベルと状態
27:         option = new Checkbox("Option",false);
28:
29:         // GUI コンポーネントの位置と大きさを指定する。
30:         okButton.setBounds(20,20,100,30);
31:         nameField.setBounds(20,70,100,40);
32:         radio1.setBounds(20,120,100,30);
33:         radio2.setBounds(140,120,100,30);
34:         option.setBounds(20,170,100,30);
35:
36:         // アプレットに追加
37:         add(okButton);
38:         add(nameField);
39:         add(radio1);
40:         add(radio2);
41:         add(option);
42:     }
43: }
```

演習

4.3.2 章のプログラムを実行してみよう.

4.4　コンテナ

4.4.1　AWT で提供されるコンテナ

AWT で提供されるコンテナを表 4.2 に示す．Frame と Dialog コンテナは独立したウィンドウを表現する．ScrollPane は Java1.1 から追加された AWT コンポーネントで表示領域内に埋め込まれた一つのコンポーネントをスクロールできる.

Panel コンテナは汎用のコンテナであり，それ自体は機能を持っていない．他のコンテナの内部でネストするのに適している．コンテナをネストして利用している例を次節に示す.

表 4.2: AWT で用意されているコンテナ

コンテナ	説明
Container	コンテナ階層のルートクラス
Frame	境界が装飾されているトップレベルのウィンドウ
Window	境界もメニューバーも持たないトップレベルのウィンドウでポップアップメニューなどの作成に利用
ScrollPane	内容をスクロールできるコンテナ
Panel	空のコンテナで，ネストされたレイアウトやサブクラス作成に良い

4.4.2　コンテナの利用例

以下に Panel をネストして利用する例を示す．プログラム内で，add() メソッドを用いて，コンテナにコンポーネント（ここではコンテナ）を追加している点に注意しよう.

```
 1: import java.applet.*;
 2: import java.awt.*;
 3:
 4: public class Containers extends Applet {
 5:     public void init() {
 6:         this.setBackground(Color.white);
 7:         this.setFont(new Font("Dialog", Font.BOLD, 24));
 8:
 9:         Panel p1 = new Panel();
10:         p1.setBackground(new Color(200,200,200));
11:
12:         this.add(p1);
13:         p1.add(new Button("#1"));
14:
15:         Panel p2 = new Panel();
16:         p2.setBackground(new Color(150,150,150));
17:
18:         p1.add(p2);
19:         p2.add(new Button("#2"));
20:
21:         Panel p3 = new Panel();
22:         p3.setBackground(new Color(100,100,100));
23:
24:         p2.add(p3);
25:         p3.add(new Button("#3"));
26:
27:         Panel p4 = new Panel();
28:         p4.setBackground(new Color(150,150,150));
29:
30:         p1.add(p4);
31:         p4.add(new Button("#4"));
32:         p4.add(new Button("#5"));
33:
34:         this.add(new Button("#6"));
35:     }
36: }
```

演習

4.4.2 章のプログラムを実行し，コンテナの利用について理解しよう．

4.5　レイアウトマネージャ

4.5.1　レイアウトマネージャの役割

4.3.2 章の例では，setBounds メソッドを用いて GUI コンポーネントの位置や大きさを指定した．しかし，Java のプログラムは，様々な環境，例えば，携帯電話や PDA など Windows とは画面の解像度や大きさが全く異なるような環境でも同一のプログラムを実行することが可能であり，このような場合には，setBounds メソッドを用いてコンポーネント上の座標で位置や大きさを指定すると，Windows 上では表示されていたボタンが携帯電話の上では表示されなくなってしまうとか，文字の大きさとアンバランスになるなどの問題が発生する．

AWT では，画面の解像度や大きさに依存せずに GUI コンポーネントを配置するための方法としてレイアウトマネージャが用意されている．レイアウトマネージャは,「あるコンポーネントの隣」や「上のほう」といった抽象的な位置関係を指定するだけで，コンテナ内部の GUI コンポーネントを自動的に配置する．

表 4.3 にレイアウトマネージャの一覧を示す．本章の以降では代表的なレイアウトマネージャである FlowLayout，BorderLayout，GridLayout について解説する．この他のレイアウトマネージャについては，インターネット上の情報や書籍などを参考にしてほしい．

表 4.3: AWT レイアウトマネージャ

レイアウトマネージャ	説明
FlowLayout	コンポーネントを追加された順に左から右に配置する．コンポーネントを右に配置する余裕がない場合は，下に折り返されて配置される．
BorderLayout	コンポーネントを中央と上下左右の最大 5 カ所に配置する．
GridLayout	コンポーネントを追加された順に左から右，上から下に格子上に並べる．すべてのコンポーネントの大きさは同一となる．
GridBagLayout	コンポーネントを格子上に並べる．コンポーネントの座標や大きさを格子単位で指定することができる．
CardLayout	各コンポーネントをコンテナを同一の大きさにし，重ねて配置する．同時に表示されるのは指定されたコンポーネントのみであり，これを切り替えることができる．

4.5.2　FlowLayout

FlowLayout レイアウトマネージャは，Panel コンテナのデフォルトのレイアウトマネージャであり，コンポーネントを追加された順に左から右に配置する．コンポーネントを右に配置する余裕がない場合には，下に折り返されて配置される．コンポーネントの大きさは調整されることはなく，右に余白ができたとしても，それはそのままにされる．

FlowLayout のコンストラクタは引数を 3 つ持ち，順に，コンポーネントの配置方法と，左右，上下のコンポーネント間の空白の大きさを指定する．コンポーネントの配置方法としては，左揃え，センタリング，右揃えのいずれかを指定することができる．

以下にプログラム例を示す．このプログラムでは，以下のように FlowLayout が利用されている．

6 行目 アプレットのレイアウトマネージャを FlowLayout に設定する．コンポーネントは左寄せで配置され，コンポーネントの上下左右の間隔は 10 とする．

8 行目 ボタンを 10 個，順に追加する．

```
 1: import java.applet.*;
 2: import java.awt.*;
 3:
 4: public class FlowLayoutExample extends Applet {
 5:     public void init() {
 6:         this.setLayout(new FlowLayout(FlowLayout.LEFT, 10, 10));
 7:         for( int i = 0 ; i < 10 ; i++ ) {
 8:             this.add(new Button("Button #" + i));
 9:         }
10:     }
11: }
```

4.5.3 BorderLayout

BorderLayout はしばしば利用されるレイアウトマネージャであり，これはコンポーネントを中央と，それを取り囲むように上下左右に配置する．コンポーネントを BorderLayout をレイアウトマネージャとして用いたコンテナに追加する場合には，以下のように 2 つ目の引数でどの位置にコンポーネントを追加するか指定する．

```
this.add(button, "North");
```

コンポーネントを追加する位置は図 4.1 のように Center，North，South，East，West という文字列で指定する．

BorderLayout のコンストラクタは引数を 2 つ取り，配置されるコンポーネント間の左右，上下の空白の大きさを指定する．

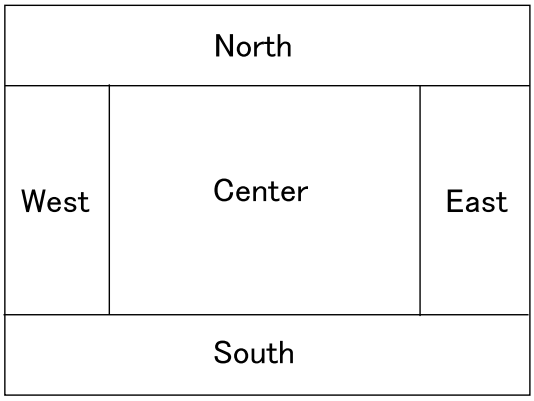


図 4.1: BorderLayout におけるコンポーネントの配置

以下にプログラム例を示す．

```
 1: import java.applet.*;
 2: import java.awt.*;
 3:
 4: public class BorderLayoutExample extends Applet {
 5:     String[] b = {"North", "South", "East", "West", "Center"};
 6:     public void init() {
 7:         this.setLayout(new BorderLayout(10, 10));
 8:         for( int i = 0 ; i < b.length ; i++ ) {
 9:             this.add(new Button(b[i]), b[i]);
10:         }
11:     }
12: }
```

4.5.4 GridLayout

GridLayout は格子上にコンポーネントを配置する．コンポーネントの並びは，追加された順に，左上から右下となる．各格子の大きさは，コンテナの高さと幅を指定された格子の数で割ったものとなり，追加されたコンポーネントは全てこの格子の大きさに揃えられる．

実際のところ，GridLayout は格子状のまま利用されることは少ない．初期化時に，行数もしくは列数のみを指定（他方は 0 とする）して，左右もしくは上下に希望の数だけコンポーネントを配置するために利用されることが多い．似たような目的としては FlowLayout が利用できるが，FlowLayout では，入りきらないコンポーネントは次の行に折り返されるため，GridLayout を用いることのほうが多い．

GridLayout はコンストラクタの引数を 4 つとり，行数，列数と，コンポーネント間の左右・上下の空白の大きさを指定する．

以下にプログラム例を示す．このプログラムでは

6 行目 行数は 0 に，列数は 3 に，左右・上下のコンポーネント間の空白は 10

に指定している．

```
 1: import java.applet.*;
 2: import java.awt.*;
 3:
 4: public class GridLayoutExample extends Applet {
 5:     public void init() {
 6:         this.setLayout(new GridLayout(0, 3, 10, 10));
 7:         for( int i = 0 ; i < 10 ; i++ ) {
 8:             this.add(new Button("Button #" + i));
 9:         }
10:     }
11: }
```

4.5.5 レイアウトマネージャのネスト

ここまで，比較的シンプルなレイアウトマネージャの紹介をしてきた．さらに複雑なレイアウトを作成しようとする場合には，GridBagLayout や，さらには LayoutManager インターフェイスを実装した自前のクラスを作成することで実現できる．しかし，ここまで紹介したシンプルなレイアウトマネージャを階層的に用いることにより，かなりのレイアウトは実現できる．

レイアウトマネージャを階層的に用いるには，基本的なコンテナである Panel クラスのインスタンスを用いる．以下にプログラム例を示す．ここでは BorderLayout の上（North）に，Panel クラスのインスタンスを追加し，この Panel クラスのインスタンスのレイアウトマネージャとして 0 行 3 列の GridLayout を指定することにより，図 4.2 に示すような配置を実現している．

```
 1: import java.applet.*;
 2: import java.awt.*;
 3:
 4: public class LayoutNestExample extends Applet {
 5:     public void init() {
 6:         BorderLayout bl = new BorderLayout(); //コンポーネント間の空白は標準
 7:         GridLayout gl = new GridLayout(0,3); // 0 行 3 列，空白は標準
 8:         Panel panel = new Panel();
 9:
10:         this.setLayout(bl); // アプレットのレイアウトは BorderLayout
11:         panel.setLayout(gl); // panel のレイアウトは GridLayout
12:
13:         this.add(new Button("South"), "South");
14:         this.add(new Button("West"),  "West");
15:         this.add(new Button("East"),  "East");
16:         this.add(new Button("Center"),"Center");
17:         this.add(panel, "North"); // Panel のインスタンスを上に追加
18:
19:         panel.add(new Button("North Left"));
20:         panel.add(new Button("North Center"));
21:         panel.add(new Button("North Right"));
22:     }
23: }
```

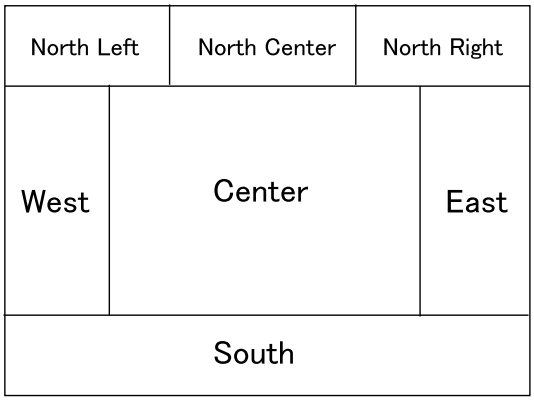


図 4.2: レイアウトマネージャのネスト

4.6　GUI コンポーネントにおけるイベントの処理

4.6.1　概要

本章の最後に，GUI コンポーネントにおけるイベント処理を紹介する．GUI コンポーネントにおけるイベント処理は，基本的には第 3 章で紹介したイベントの取り扱いと同じである．以下にその手順を述べる．

1. GUI コンポーネントが発生するイベント（`XXXEvent`）を調べる．
2. GUI コンポーネントが発生するイベントに対応したリスナー（`XXXListener`）を調べる．必要に応じてアダプタも調べる[a]．アダプタ名は通常，リスナークラスの「`Listener`」の部分を「`Adapter`」に変えたもの（この例では `XXXAdapter`）となる．
3. リスナーまたはアダプタを拡張し，必要なメソッドを実装する．
4. 拡張したリスナーまたはアダプタを GUI コンポーネントの対応するイベントに登録（`addXXXListener()`）する．

[a]全てのリスナーに対応したアダプタが用意されているわけではないことに注意しよう．概ね，複数のメソッドを持つリスナーにはアダプタが用意されているが，単一のメソッドしか持たないリスナー（例えば ActionListener など）はアダプタが用意されていない．

という手順である．

表 4.4 に GUI コンポーネントが発生するイベントの一覧を示す．これに基づいて，`Button` クラスをクリックした場合のイベント処理を考えてみよう．

1. 表 4.4 より，Button クラスのインスタンスをクリックした際に発生するイベントは ActionEvent を発生することが分かる．
2. 表 3.1 より，ActionEvent に対応するリスナーは ActionListener であることが分かる．ActionListener はメソッドを一つしか持たず，対応するアダプタはない．
3. ActionListener のメソッド actionPerformed を実装したクラスを作成する．
4. Button クラスに addActionListener メソッドを用いてリスナーのインスタンスを登録する．

表 4.4: AWT コンポーネントが発生するイベント

コンポーネント	発生するイベント	意味
Button	ActionEvent	ボタンがクリックされた．
Checkbox	ItemEvent	項目が選択/選択解除された．
CheckboxMenuItem	ItemEvent	項目が選択/選択解除された．
Choice	ItemEvent	項目が選択/選択解除された．
Component	ComponentEvent	コンポーネントの移動，大きさ変更，非表示/表示状態の変更．
	FocusEvent	コンポーネントがフォーカスされた/フォーカスを失った．
	KeyEvent	ユーザがキーを押した/離した．
	MouseEvent	マウスに関するイベント．
Container	ContainerEvent	コンテナにコンポーネントが追加/削除された．
List	ActionEvent	ユーザが項目をダブルクリック．
	ItemEvent	項目が選択/選択解除された．
MenuItem	ActionEvent	ユーザがメニュー項目を選択．
Scrollbar	AdjustmentEvent	ユーザがスクロールバーを移動．
TextComponent	TextEvent	ユーザがテキストを変更．
TextField	ActionEvent	ユーザがテキスト編集を完了．
Window	WindowEvent	ウィンドウがオープン/クローズなどされた．

4.6.2 プログラム例

以下に簡単なプログラム例を示す．このプログラムでは，GridLayout を用いてアプレットを上下の二つの部分に分け，上に Button を，下に Canvas を配置している．Button を押すごとに Canvas 上の異なる位置に文字列を表示させるために，ActionListener を実装した ButtonListener クラスを内部に作成し，これを Button クラスに登録している．以下簡単なプログラムの解説：

6〜22 行目 Canvas クラスを拡張して，クラス内クラス（内部クラス）StringCanvas を定義している．内部クラスは，定義されたクラス（ここでは ButtonActionEx）内でのみ有効なクラスである．このクラスはボタンの押下状況（buttonPushed() メソッドで通知される）に応じて，描画を行う paint() メソッドの実装が主である．

24〜36 行目 ActionListner インターフェイスを実装した ButtonListener クラスを定義している．これは

Button が押されたときのイベントを処理するためのクラスで，対応するメソッド actionPerformed() メソッドを実装している．

32〜35 行目 actionPerformed() メソッドでは，ボタンが押されたことを描画用の StringCanvas クラスのインスタンスに通知（buttonPushed()）し，再描画（repaint()）している．repaint() メソッドは，コンポーネントを消去し，paint() メソッドを呼び出すことで再描画を行う．

41 行目 Button クラスのインスタンスに，リスナーを登録する．リスナーは 24〜36 行目で定義された ButtonListener クラスのインスタンスであり，コンストラクタの引数として描画領域である StringCanvas のインスタンスを渡している．

43〜45 行目 アプレットの描画領域を GridLayout によって上下に 2 分割し，上に Button クラスのインスタンス，下に StringCanvas クラスのインスタンスを追加している．

```
 1: import java.applet.*;
 2: import java.awt.*;
 3: import java.awt.event.*;
 4:
 5: public class ButtonActionEx extends Applet {
 6:     class StringCanvas extends Canvas {
 7:         boolean buttonPushed = false;
 8:
 9:         public void buttonPushed() {
10:             buttonPushed = true;
11:         }
12:
13:         public void paint(Graphics g) {
14:             if( buttonPushed ) {
15:                 Dimension d = getSize();
16:                 int x = (int)(Math.random() * d.width);
17:                 int y = (int)(Math.random() * d.height);
18:                 g.drawString("Hello World!", x, y);
19:                 buttonPushed = false;
20:             }
21:         }
22:     }
23:
24:     class ButtonListener implements ActionListener {
25:         StringCanvas scanvas;
26:
27:         ButtonListener(StringCanvas sc) {
28:             super();
29:             scanvas = sc;
30:         }
31:
32:         public void actionPerformed(ActionEvent e) {
33:             scanvas.buttonPushed();
34:             scanvas.repaint();
35:         }
36:     }
37:
38:     public void init() {
39:         Button button = new Button("Draw String");
40:         StringCanvas scanvas = new StringCanvas();
41:         button.addActionListener(new ButtonListener(scanvas));
42:
43:         this.setLayout(new GridLayout(2,0));
44:         this.add(button);
45:         this.add(scanvas);
46:     }
47: }
```

演習

3.4 章のアプレットにボタンを追加し，これを押すことで画像を切り替えるアプレットを作成してみよう．

その際，レイアウトマネージャを用いてみよう．

第5章　Webブラウザを用いないGUIプログラム

5.1　アプレットの利点と欠点

これまでアプレットを用いたGUIプログラムについて説明して来た．アプレットは対応したWebブラウザがあれば，ネットワークを介してプログラムを実行できるという利点がある．しかし，Webブラウザはアプレットを動作させるだけのプログラムではない（appletviewerは別だが）ため，一般に巨大なプログラムであり，コンピュータのリソース（メモリや速度といった資源）をアプレット以外の部分にも大量に消費してしまう．そこで，Javaでは，独立したウィンドウを作成してその上でGUIを扱うという手法が用意されている．このように作成されたプログラムは，Webブラウザに依存せずに独立したアプリケーションとして実行可能である．

5.2　Frameクラス

独立したウィンドウに対応したクラスFrameクラスである．

Frameクラスはトップレベルのウィンドウであり，これまでに紹介してきたその他のAWTコンポーネントと動作が異なる．まず，Frameを画面に表示させるにはshow()というメソッドを呼び出さなければならない．また，その前にウィンドウの大きさを指定する必要がある．

ウィンドウの大きさを指定するメソッドはsetSize()とpack()である．setSize()メソッドは明示的にウィンドウの大きさを指定する方法であり，pack()は内部に含まれるGUIコンポーネントの大きさを調べ，それにあわせてウィンドウの大きさを決定する．

Javaのプログラムが環境非依存であることを考えると，ウィンドウサイズの指定にはpack()メソッドが推奨されるが，内部に，大きさが決まらないコンポーネント（例えばCanvasクラスなど）が含まれる場合には，setSize()メソッドを用いる必要がある．

5.3　Frameを用いたプログラム例

以下にFrameクラスを用いたプログラム例を示す．このプログラムそのものはHelloFrameというクラスであり，コンストラクタは28〜35行目である．その他はアプリケーションとして実行するためのmain()メソッド（17〜26行目），文字を描くためのCanvasクラスのサブクラス（5〜9行目），ボタンのイベントを処理するためのActionListenerを実装したクラス（11〜15行目）である．

その他については概ねアプレットと同じなのでmain()メソッドについて少し詳しく解説しておこう．

18行目 Frameクラスのサブクラス，すなわちウィンドウに対応するクラスであるHelloFrameクラスのインスタンスを生成している．

19〜23行目 ここで生成されたウィンドウは，ネイティブのウィンドウマネージャに管理されるため，普通のアプリケーションと同様にウィンドウには「閉じる」,「最大化」,「最小化」などに対応したボタ

ンが付く．しかし，素の Frame クラスは，これらのイベントを処理しないので，この行では「閉じる」を押された場合のイベント処理を登録している．これは決まった書き方なので覚えておくと良い．

24 行目 ここでウィンドウのサイズを指定している．このウィンドウはサイズが決まらないコンポーネント（Canvas クラス）を含むので明示的にウィンドウサイズを指定している．例えばここを frame.pack() に変更するとどうなるか試してみると良いだろう．

25 行目 ここで初めてウィンドウを表示している．

```
 1: import java.awt.*;
 2: import java.awt.event.*;
 3:
 4: public class HelloFrame extends Frame {
 5:     public class HelloCanvas extends Canvas {
 6:         public void paint(Graphics g) {
 7:             g.drawString("Hello World", 10, 20);
 8:         }
 9:     }
10:
11:     public class ButtonActionListener implements ActionListener {
12:         public void actionPerformed(ActionEvent event) {
13:             System.exit(0);
14:         }
15:     }
16:
17:     public static void main(String[] args) {
18:         Frame frame = new HelloFrame();
19:         frame.addWindowListener(new WindowAdapter() {
20:             public void windowClosing(WindowEvent e) {
21:                 System.exit(0);
22:             }
23:         });
24:         frame.setSize(160, 160);
25:         frame.show();
26:     }
27:
28:     public HelloFrame() {
29:         HelloCanvas canvas = new HelloCanvas();
30:         Button button = new Button("exit");
31:         button.addActionListener(new ButtonActionListener());
32:
33:         add(canvas, "Center");
34:         add(button, "South");
35:     }
36:
37: }
```

このプログラムは普通のアプリケーションとして実行可能なので，コンパイル，実行するには以下のように入力すれば良い．

```
% javac HelloFrame.java
% java HelloFrame
```

5.4 アプレットにもなる GUI アプリケーション

Frame クラスを用いることにより，アプレットにもなり，なおかつ独立したアプリケーションとしても実行可能なプログラムを作成することができる．以下にプログラム例を示す．見て分かると思うが，これは 4.6.2 章のプログラムに main() メソッドを追加したものとなっている．これはアプリケーションとして実行可能であり，なおかつアプレットとしても実行可能である．

```
 1: import java.applet.*;
 2: import java.awt.*;
 3: import java.awt.event.*;
 4:
 5: public class HelloFrameApplet extends Applet {
 6:     class StringCanvas extends Canvas {
 7:         boolean buttonPushed = false;
 8:
 9:         public void buttonPushed() {
10:             buttonPushed = true;
11:         }
12:
13:         public void paint(Graphics g) {
14:             if( buttonPushed ) {
15:                 Dimension d = getSize();
16:                 int x = (int)(Math.random() * d.width);
17:                 int y = (int)(Math.random() * d.height);
18:                 g.drawString("Hello World!", x, y);
19:
20:                 buttonPushed = false;
21:             }
22:         }
23:     }
24:
25:     class ButtonListener implements ActionListener {
26:         StringCanvas scanvas;
27:
28:         ButtonListener(StringCanvas sc) {
29:             super();
30:             scanvas = sc;
31:         }
32:
33:         public void actionPerformed(ActionEvent e) {
34:             scanvas.buttonPushed();
35:             scanvas.repaint();
36:         }
37:     }
38:
39:     public void init() {
40:         StringCanvas scanvas = new StringCanvas();
41:         Button button = new Button("Draw String");
42:         button.addActionListener(new ButtonListener(scanvas));
43:
44:         this.setLayout(new BorderLayout());
45:         this.add(button, "South");
46:         this.add(scanvas, "Center");
47:     }
48:
49:     public static void main(String[] args) {
50:         Frame f = new Frame();
51:         Applet a = new HelloFrameApplet();
52:
53:         a.init();
54:         f.add(a, "Center");
55:
56:         f.addWindowListener(new WindowAdapter() {
57:             public void windowClosing(WindowEvent e) {
58:                 System.exit(0);
59:             }
60:         });
61:         f.setSize(400,400);
62:         f.show();
63:     }
64: }
```

演習

`Frame` クラスを用いて，4.6.2 章で作成したアプレットを独立したアプリケーションとして実行可能にしてみよう.