9.エラーと例外処理

どんなに注意深くプログラムを作成したとしても，プログラムの実行中に何らかの問題が発生することは非常によくあることである。本章では，プログラム実行中のエラーを取り扱う方法として，例外処理（exception　handling）について解説する。

本章では，以下について学習する。

●エラーと例外処理について
●例外とその種類
●例外の送出
●例外の捕捉

9．1　エラー

エラーが発生した場合，プログラムはそのエラーに適切に対処して，そのまま実行を継続できることが望ましい。そうでなくても，ユーザ自身がプログラムに指示を与えたり，終了させたりできるようにするべきである。実際には，まったく対処のしようがないエラーも存在するが，少なくとも実用的なプログラムを設計・制作する際は，エラーを処理するコードを加える労力を惜しむべきではない。

単純なタイプミスや文法上の間違いなどが原因のコンパイルエラーに対して，実行中に発生するエラーは，一般にランタイムエラー（runtime　exception）と呼ばれる。本章で対象となるエラーはランタイムエラーである。このようなエラーに対処できるプログラムを作成するために，Java ではエラーを検出して通知する機構が用意されている。

プログラムを実行する際に発生するエラーには様々なものが考えられる。例えば，数値を入力するべき場所で，あるユーザは自分の名前を入力するかもしれない。何らかの設定ファイルのように，本来は存在するはずのファイルを，ユーザが消去してしまい，それを知らずにプログラムから参照しようとするかもしれない。メモリやディスクが満杯で，それ以上何も処理できない状態になるかもしれない。このように，一口にエラーといっても様々なものが考えられる。さらに，特定の個所で発生する可能性のあるエラーが 1 つであるとは限らない。例えば，「ファイルから値を読み込んで，配列に格納する」というプログラムでは，「ファイルからの値の読み込み」がエラーとなるかもしれないし，「配列に格納する」がエラーになるかもしれない。

ここで重要なのは，エラーが発生した場合，それをプログラムが検知し，どのようなエラーが発生したか分類し，可能なら適切な処理を行うことである。

9．2　エラーの処理の方法

ここでは簡単なエラーの実例を挙げ，それに対応するコードを示すことで典型的なエラーの処理方法について学習する。まず，例となるエラーを発生するコードを以下に示す。

```
MyClass obj ;
obj　＝null ;
obj. doSomething（）;
```

このコードは，MyClass クラスの変数 obj に null を代入して，変数 obj に対して doSomething（）メソッドを呼び出している。実際には，obj が参照するオブジェクトがないのでエラーとなる[1]。このようなエラーを null 参照エラーと呼ぶことにする。もちろん，こんなコードを実際に書くことはないと思うかもしれないが，例えば，メソッドの引数で MyClass のオブジェクトへの参照を受け取り，それに対してメソッドを呼び出すことはよくある。このとき，引数に null が渡されると，結果的に上記のコードと同じ null 参照エラーが発生する。

[1]　後で述べることと関連があるが，Java では NullPointerException が発生する。

このエラーを防ぐ最初の方法は，doSomething（）メソッドを呼び出す前に変数 obj が null かどうかチェックする方法である。この方法は，後で述べる「例外」をサポートしていない言語，例えば C 言語などでは一般的に行われる方法である。もちろん，Java でも同じ方法が使用できる。

```
if（obj＝＝null）{　//　null 参照エラーが起こるかどうかチェック
    //　このままではエラーになるのでなんとかする
    ...
} else {
    obj. doSomethjng（）;
}
```

しかし、doSomething（）メソッドは、null 参照エラー以外のエラーを引き起こす可能性があるかもしれない。この場合，すべてのエラーについて事前にチェックする必要がある。実際には，エラーが起こるより、正常に実行される確率の方が遥かに高いことが多い。ところが，以下のようなコードでは，どこがコードの本筋なのか非常に分かりにくくなってしまう。

```
if（obj＝＝null　）{　//　エラー1（null 参照エラー）のチェック
} else　if（...）{　　//　doSomething（）が引き起こすエラー2 のチェック
  ...
  ...
```

```
} else  if（...）        //  doSomething（）が引き起こすエラーn のチェック
  ...
} else {                 //  これでエラーは起こらないことが確認できた
    obj. doSomething（）;
}
```

Java では、これとは異なるエラー処理機構である例外（exceptlon）が用意されている。例外の処理のイメージを図 9．1 に示す。この方法は，まず，エラーの種類と状態を保持するクラスである例外を定義しておく。9．3 節で詳しく述べるが，ここでは MyException クラスであるとしよう。コード上でエラーが発生しそうな場所を，予約語 try とそれに続くブロックで囲っておく。そして、try ブロックに続けて catch ブロックでエラーを処理するコードを記述する。このエラーを処理するコードは，一般にエラーハンドラ（error handler）と呼ばれる。 catch ブロックは，MyException クラスのオブジェクトを引数とするメソッドのように記述する。try ブロック内のどこかでエラーが発生すると，制御が catch ブロックに移る。このとき，MyException クラスのオブジェクトが catch ブロックに渡される。すなわち，MyException クラスのオブジェクトがボールのように「投げられ」（送出され）て，catch ブロックで「受け取られる」（捕捉される）ことになる。送出については後で述べるためここでは詳しく触れないが，実際のコードでは，予約語 throw を使用することで例外を送出することができる。

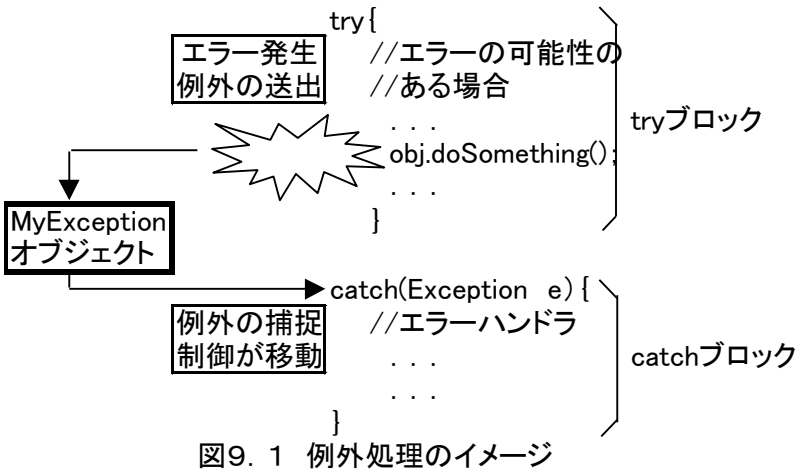

図9．1　例外処理のイメージ

9．3　例外と種類

Java では，何らかのエラーを表すクラスである例外を定義する必要がある。前節で「例外を送出する」するという表現を使用したが，例外は，Java のクラスライブラリに含まれる Throwable クラスから派生する。実は，クラスライブラリで既にいくつかのクラスが定義されており，独自の例外もそれらと同じように定義できる。Java の例外の継承関係を図

9．2 に示す。

Java で例外のスーパークラスとなっているクラスは Throwable クラスである。その名前が示す通り，このクラスのオブジェクト（サブクラスのオブジェクトを含む）だけが「送出」（throw，後述）の対象であり，したがって「捕捉」（catch）の対象でもある。Throwable クラスとそのサブクラスは最低限，引数なしのコンストラクタ，エラーメッセージを表す String クラスのオブジェクトを引数とするコンストラクタの 2 つのコンストラクタを持つ。

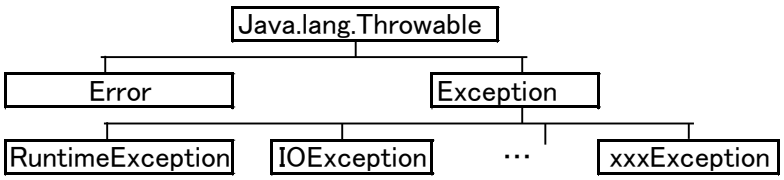

図9．2　Javaの例外の継承関係

Throwable クラスの直接のサブクラスは，Error クラスと Exception クラスである。Error クラスは，通常のアプリケーションでは対処できないような重大なエラー状態を示すクラスである。したがって，一般のコードではこの例外を送出してはならず，捕捉する必要もない。Error に対して Exception クラスは，アプリケーションで捕捉の対処となる例外である。プログラム中で例外を定義する必要がある場合は，Exception クラスを派生して独自の例外クラスを作成する。既に定義されている Exception については，大ききくは RuntimeException とそれ以外の例外に分かれる。

RuntimeException は，主にプログラムのコードそのものの問題によるエラーで，Java 仮想マシンから送出される例外のスーパークラスである。9．2 節で取り上げた null 参照エラーは，明らかに不注意なコードが原因である。この例外が発生すると，Java 仮想マシンは RuntimeException のサブクラス NullPointerException のオブジェクトを送出する。これ以外に，配列の境界を越えて不正なアクセスをした（IndexOutOfBoundsException），不正なキャストを行った（ClassCastException）などが RuntimeException のサブクラスとして定義されている。一般には，RuntimeException とそのサブクラスをプログラムのコードから投げる必要はないはずである。

RuntimeException 以外のクラスは，基本的にプログラムのコードには問題がないが，実行時に何か不都合なことが生じたことを表す例外である。例えば，入出力で発生したエラーや割り込み（IOException），AWT（GUI を構築するためのクラスライブラリ）での例外（AWTException），セキュリティ関係の例外（GeneralSecurityException）などは，RuntimeException とは別に定義されている。

9．4　例外の送出

例外の送出は，予約語 throw を用いて以下のように行う。

throw　例外のオブジェクト；[2]

ここで，例外のオブジェクトは Throwable から派生したクラスのオブジェクトを指す。例えば，最も一般的な Exception クラスのオブジェクトを送出するには，以下のようにする。

```
throw new Exception（）：
```

または，

```
Exception e＝new Exceptjon（）；
throw e；
```

throw 文は, return 文や break 文のように制御の流れを大きく変える。したがって throw 文が実行されると，その後に続くコードは実行されない。代わりに，例外を捕捉するコードに実行がジャンプする。

例として Exception クラスのオブジェクトの送出を示したが，状況をより明確に表す例外を使用できる。既にクラスライブラリ中で定義されている例外を使用することも考えられる。例えば，引数の値が不正であることを表す IllegalArgumentException や，ファイルやネットワークからある決まった長さのデータを読みこんでいる最中でデータが途切れてしまった場合は，EOFException を使用することが考えられる。さらに，自分で Exception クラスを派生して独自の例外クラスを定義して使用してもよい。

[2]　正確には“throw 式；”であるが，式の型は Throwable として取り扱える型である必要がある。

```
Public class MyException extends Exception{
. . .
}
. . .
   throw new MyException（）；
```

例外を送出する可能性のあるメソッドは，宣言時にそれを明示する必要がある。これは，メソッドの宣言に throws 節を以下のように加える。

メソッド修飾子　戻り値の型メソッド名（引数リスト）throws　例外クラスのリスト

{メソッドのボディ}

例えば，MyExceptionl と MyException2 の 2 つの例外を送出する可能性のある doSomething（）メソッドは，以下のようになる。

```
Public void doSomething（）throws MyExceptionl，MyException2｛
    ...
    if（Errorl）       //  もし例外 1 が発生したら
            throw new MyExceptionl（）；
    ...
    if（Error2）       //  もし例外 2 が発生したら
            throw new MyException2（）；
    ...
}
```

9．5　例外の捕捉

例外を捕捉するには，プログラムのコード上で例外が発生する可能性のある部分を，予約語 try とそれに続くブロックで囲む。メソッドについては，どの例外を発生させるか宣言から分かる。try ブロックに続けて，catch ブロックを記述する。catch ブロックにはエラーハンドラが含まれる。catch ブロックは，引数として例外をとるメソッドに似た形式で記述する。あるメソッドが複数の例外を発生させたり，異なる例外を発生させる複数のメソッドが try ブロックに含まれている場合は，例外に合わせて複数の catch ブロックを記述する。したがって，try－cacth のブロックは，全体として以下のような形式をとる。

```
try｛
        //  例外を起こす可能性のあるコード
        ...
}
catch（MyExcptionl el）｛
        //  MyExceptionl のためのエラーハンドラ
}
catch（MyException2 e2）｛
        // MyException2   e2 のためのエラーハンドラ
}
catch（...）    ...
```

try ブロック中で例外が発生すると，その例外にマッチする catch ブロックを上から順に探していき，最初にマッチした catch ブロックに制御が渡される。この動作は switch 文の動きに似ている。したがって，catch ブロックは，継承関係では末端にあたるクラスからはじめて，後へいくほど一般的な例外になるような順番で記述する。Exceptlon クラスは，そのサブクラス，すなわち，すべての例外とマッチするので注意が必要である。Exception クラスとマッチする catch ブロックを最初に記述すると，他の catch ブロックに制御が渡ることはない。逆に最後に置くと，他の catch ブロックとマッチしないすべての例外を捕捉することができ，switch 文の default タグのような働きをさせることができる。　エラーハンドラに何を記述するかは，どんな例外に対してどのように対処するかによる。可能ならエラーを回復したり，メッセージやスタックのトレース（Exception クラスが保持している）を表示させたり，アプリケーションを終了させたりする。いったん捕捉した例外を再度 throw してもよい。以下のコードは，メッセージを出力した後，例外を再度 throw する例である。

```
...
catch（MyException e）{
    System．out．println（“Caught MyException：”＋e）；
    throw e；
}
...
```

try ブロック中では、例外を投げるだけでなく，return 文が実行されることも考えられる。throw 文にしろ return 文にしろ，いったん実行されると制御が catch ブロックや上位のメソッドに移るため，throw 文，return 文の先に記述してあるコードは実行されない。Java では、メソッドが終了する前に実行するコードを，予約語 finally とそれに続くブロックを使用して記述することができる。このブロックが存在すると，throw 文や return 文にかかわらず，そのメソッドが終了する直前に実効される。
finally ブロックは以下のような形式で記述する。

```
public void funcl（）{
  //　funcl（）メソッドの処理
  ...
  }
  finally｛
      //　funcl（）メソッドを終了する前に実行するコード
  ...
```

```
  }
}
```

finally ブロックには，何らかの後処理，例えば，オープンしたファイルをクローズするなどを記述することができる。

プログラム 9.. 1 に，例外を使用した例を示す。このプログラムは，コマンドラインの引数で名前と年齢を渡すと，“Myname　is…”のように表示するプログラムである。実行例を以下に示す。

```
C：￥JavaDev＞java TestException Ken－ichi　24
Arguments：Ken－ichi，24
Hi　My name　is Ken－ichi，24　years old.
Nice to meet you.
See you　later.
```

上記の例では“Ken－ichi”と“24”’が引数となる。このプログラムでは，引数を必ず 2 つ必要とし，2 番目の引数は（年齢を表すので）数であるものとしている。引数が 2 つあるかは 13 行目の checkArgs（）メソッドでチェックし，2 つない場合は，その中で例外 MyException を throw している（15 行目）。MyException には，エラーメッセージにあたる文字列とエラー番号にあたるリファレンス番号を格納する。main（）メソッドでは MyException を catch しており（34 行目），エラーの際は，エラーメッセージとリファレンス番号を表示する（35 行目）。それ以外のエラーについては 37 行目で catch している。例えば、31 行目で文字列 argv［1］を int 型に変換して引数としているが，数を表さない文字列を変換しようとすると例外が発生し，これは 37 行目で catch される。

正常に実行された場合，MyException を catch した場合，Exception を catch した場合（特に 39 行目で return している点に注意）のいずれの場合でも，finally ブロックが実行され　最後に“See　you　later.”と表示される（42 行目）。

引数の数が 2 つでない場合と数以外の引数を渡した場合の実行例を、以下に示す。

```
C：￥JavaDev＞java TestException Ken－ichi
Exceptionl：MyException：checkArgs（）Method．Ref＃123
See　 you　later.
```

```
C：￥JavaDev＞java TestException Ken－ichi　Okada
Arguments：Ken－ichi，Okada
Exception2：java．lang．NumberFormatException：Okada
```

See you later.

```
1:class MyException extends Exception {
2:   int ref = 0;          /** 参照番号/エラー番号 */
3:
4:   /** コンストラクタ */
5:   MyException(String mesg, int ref) {
6:      super(mesg);
7:      this.ref = ref;
8:   }
9:}
10:
11:public class TestException {
12:   /** コマンドラインの引数をチェックして、表示する */
13:   public static void checkArgs(String argv[]) throws MyException {
14:      if(argv.length != 2)
15:         throw new MyException("checkArgs() Method. ",123);
16:      else
17:         System.out.println("Arguments: " + argv[0] + "," + argv[1]);
18:   }
19:
20:   /** 自己紹介する。"Hi! My name is   ,,, "の表示 */
21:   public static void introduceMyself(String name, int age) {
22:      System.out.println("Hi! My neme is " + name + ", " +
23:                                    age + "years old.");
24:   }
25:
26:   /** 例外をテストするメソッド */
27:   public static void main(String argv[]) {
28:      try {
29:            TestException.checkArgs(argv);                    //引数のチェック
30:            TestException.introduceMyself(argv[0],
31:                                 Integer.parselnt(String argv[1]));    //自己紹介
32:            System.out.println("Nice to meet you. ");    //はじめまして
33:      }
34:      catch(MyException e) {
```

```
35:             System.out.println("Exception1: " + e + " Ref#" + e.ref);
36:         }
37:         catch(Exception e) {
38:             System.out.println("Exception2: " + e);
39:             return;
40:         }
41:         finally {
42:             System.out.println("See you later.");        //「それでは後ほど」
43:         }
44:     }
45:}
```

プログラム９．１　例外

10. AWT（AbstractWindowToolkit）

本章では，コンピュータの画面上にグラフィカルに表示されるインタフエースを伴うアプリケーションを Java 言語で記述する際に利用する，AWT パッケージについて述べる。AWT パッケージにより，アプリケーション開発者はプラットフォームに依存しない GUI を構築することができる。

本章では，以下について学習する。

●AWT とは
●AWT のクラス構成
●イベントの処理
●AWT の様々なクラスの使用方法

10．1　AWT とは

Java 言語において，GUI（Graphical User Interface）を構築するためのクラス群を AWT（Abstract Window Toolkit）という。これは java．awt パッケージに属し，プログラマはパッケージ内の様々なクラスを利用して，簡単に GUI を持つアプリケーションプログラムを組むことができるようになっている。

AWT が抽象的（abstract）なものであるゆえんは，AWT を利用してアプリケーションを作成すると，MS－Windows や UNIX といったプラットフオームに依存しないプログラムで GUI を実現できることである。GUI 自体は，プラットフォームに特有なツールキットによって，実行時に画面に表示されることになる。

10．2　AWT のクラス構成

AWT には，GUI を構築するための数多くの強力なクラスが含まれている。例えば，ウィンドウを作成するためのクラスやボタンを作成するためのクラス，メニューを作成するためのクラス，そして描画を行うためのクラスなどである。図 10．1 に，java．awt パッケージに含まれる主なクラスとその継承関係を示す[1]。

以下では，この中で基本的なクラスを，サンプルプログラムを示しながら説明する。

[1]　詳細は、JDK に含まれるドキュメントを参照のこと

10．3　Component クラス

一般に Java において，実際に画面上に表示されるオブジェクトのことを「コンポーネント」という。例えば，ボタンオブジェクトやラベルオブジェクト等がそれに含まれる。Component クラスは，java．awt パッケージの中に含まれる主な「コンポーネント」の抽象スーパークラスであり，単独で使われることはほとんどない。代わりに，ボタンオブジ

ェクトやラベルオブジェクトといったクラスは，Component クラスを継承して定義されている。したがって Component クラスには，サブクラス化されたコンポーネントが共通に利用できるメソッドが数多く定義されている。

表 10．1 に，よく使われると思われる代表的なメソッドのいくつかについて示す。詳細は JDK 付属のドキュメントを参照してほしい。

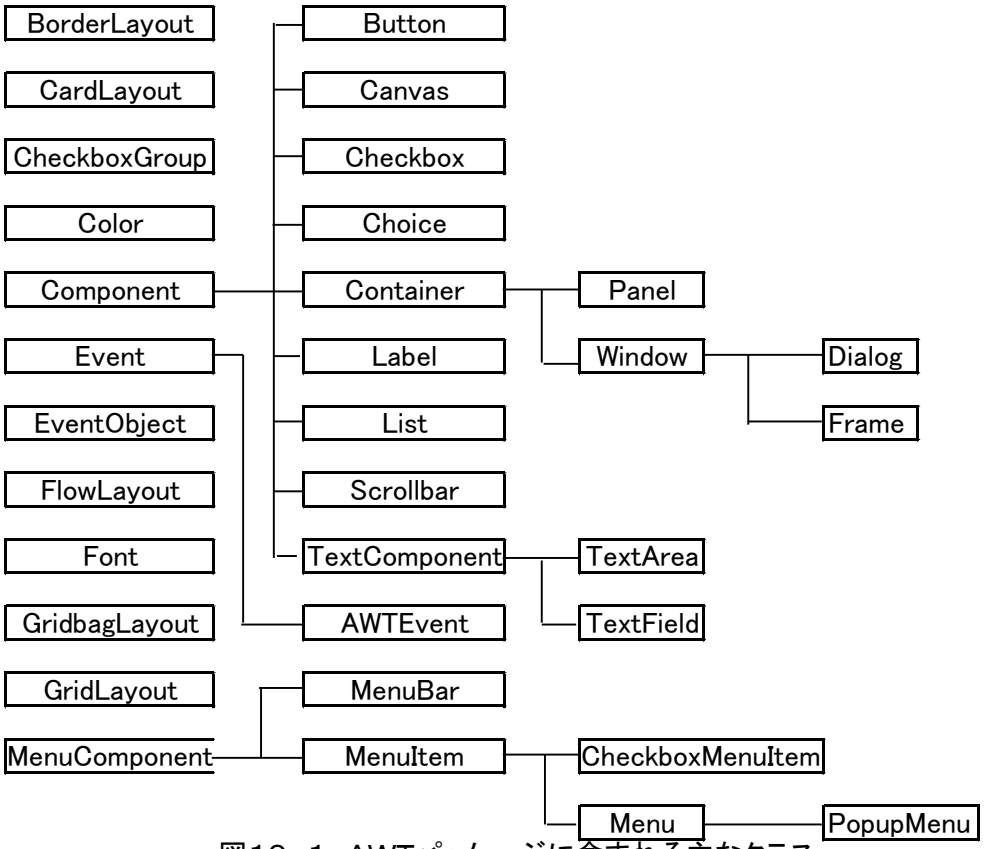

図10．1　AWTパッケージに含まれる主なクラス

表10．1　Componentクラスで定義されているメソッド

返り値	メソッド名	効用
Color	getBackground()	背景色を返す
Font	getFont()	フォントを返す
FontMetrics	getFontMetrics(Font *font*)	フォントメトリックスを返す
Color	getForeground()	前面色を返す
Graphics	getGraphics	グラフィックスを返す
int	getHeight()	高さを返す
String	getName()	名前を返す
Container	getParent()	スーパークラスを返す
Dimension	getSize()	サイズを返す
int	getWidth()	幅を返す
boolean	isEnabled()	使用可能かどうかを判定する
boolean	isShowing()	表示/非表示を判定する
void	Paint()	コンポーネントを描画する
boolean	prepareImage(Image *img*,ImageObserver *o*	表示するイメージを用意する
protected void	processEvent(AWTEvent *e*)	発生するイベントを処理する
void	repaint()	再描画する
void	repaint(int *x*,int *y*,int *width*,int *height*)	指定した矩形領域を再描画する
void	setBackground(Color *c*)	背景色を指定する
void	setEnabled(boolean *b*)	使用可能/不可能にする
void	setFont(Font *f*)	フォントを指定する
void	setForeground(Color *c*)	前面色を指定する
void	setLocation(int *x*,int *y*)	位置を指定する
void	setName(String *name*)	名前を指定する
void	setSize(Dimension *d*)	大きさを指定する
void	setSize(int *x*,int *y*)	大きさを指定する
void	setVisible(boolean *b*)	可視化/不可視化を指定する
String	toString()	文字列表現を返す
void	update(Graphics *g*)	コンポーネントを更新する

10．4　テキストを表示するプログラム

それではAWT入門として，新たにウィンドウを作成し，その上でテキストを表示するアプリケーションプログラムを作成してみる。

```
1:import java.awt.*;      //AWT をインポート
2:
3:public class SampleWindow {
4:
5:   public static void main(String args[]) {
6:      //Frame クラスのオブジェクトを作成する。
7:      Frame f = new Frame();
8:      //Label クラスのオブジェクトを作成する。
9:      Label I = new Label("Hello World!!");
10:      f.add(I);                    //ラベルコンポーネントを追加する。
```

```
11:     f.pack();                  //フレームのサイズを決定する。
12:     f.setVisible(true);  //フレームを可視化する。
13:  }
14:}
```

プログラム１０．１　ウインドウとテキストの表示（SampleWindow.java）

プログラム 10．1 では，まず最初に，標準的なウィンドウを作成するためのクラスである Frame クラスのオブジェクトを作成している。Frame クラスは，タイトルバーとボーダー領域が標準で付いているトップレベルのウィンドウである。

次に，テキストをコンボーネント上で表示することができる Label クラスのオブジェクトを作成する。 Label クラスのコンストラクタには，表示させたいテキストを String 型で指定できる。そして，作成した Label クラスのオブジェクトを Frame クラスのオブジェクトに追加する。ラベルをフレームに貼り付けるというイメージである。これには，追加される側のコンボーネント（ここでは Frame クラス）の add（）メソッドにおいて，引数に追加する側のコンボーネントを指定してやればよい。add（）メソッドは，Container クラスから Frame クラスに継承されている。

ここで，Frame クラスのように他のコンボーネントが追加される側のコンポーネントをコンテナコンポーネントと呼び，Label クラスのように追加する側のコンポーネントを，追加される側のコンポーネントのサブコンポーネントと呼ぶ。

図 10．2 に，コンテナコンポーネントとサブコンポーネントの関係の概念図を示す。コンテナコンポーネントにサブコンポーネントが貼りつ付けられていくということをイメージしてほしい。

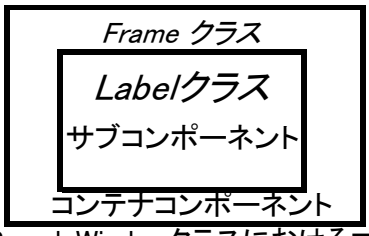

図１０．２　SampleWindowクラスにおけるコンポーネントイメージ図

ここまでの処理が終了すると，後は実際に画面に表示するだけである。表示するためには，ウィンドウの大きさが決定されなければならない。pack（）メソッドは，サブコンボーネント（追加されたコンポーネント）の大きさやレイアウトに従ってウィンドウ（Frame コンボーネント）の大きさを変更するメソッドである。この操作を行わないと，せっかく追加したラベルが表示されないことになる。pack()メソッドは, Window クラスから Frame クラスに継承されている。大きさが決定されると，setVislble（）　メソッドでウィンドウ

を可視化する。

このプログラムを実行すると，“Hello World !!”と書かれた図 10．3 のようなウィンドウが画面上に表示されるはずである。簡単なプログラムであるため，閉じるボタンを押してもまだこのアプリケーションを終了することはできない。したがって，現投階では，コンソールで Ctrl キーを押しながらアルファベットの C のキーを押すことにより終了する。

図１０．３　SampleWindow プログラム

10．5　終了可能なウィンドウのプログラム

前節で示したプログラムを発展させて，閉じるボタンを押すとアプリケーションが終了し，ウィンドウが閉じられるようなプログラムを作成してみる。

```
1:import java.awt.*;      //AWT をインポート
2:
3:public class SampleWindow2 extends Frame {
4:   //コンストラクタ
5:   public SampleWindow2(String _title) {
6:      //親クラスのコンストラクタを呼び出す。
7:      super(_title);
8:
9:      //Window イベントが有効であるように設定
10:      enableEvents(AWTEvent.WINDOW_EVENT_MASK);
11:
12:      Label I = new Label("Hello World!!", Label.CENTER);
13:      add(I);
14:      setSize(200,100);
15:      setVisible(true);
16:   }
17:
18:   public void processEvent(AWTEvent e) {
19:      if(e.getID() == Event.WINDOW_DESTROY) {
20:      System.exit(0);
21:      }
```

```
22:  }
23:
24:  public static void main(String args[]) {
25:      SampleWindow2 f = new SampleWindow2("SampleWindow2");
26:  }
27:}
```

プログラム１０．２　ウインドウを閉じるプログラム（SampleWindow2.java）

プログラム 10．2 では，前節で示したプログラムを発展させ，Frame クラスを継承した SampleWindow2 クラスを宣言し，main（）メソッド中の最初で SampleWindow2 クラスのオブジェクトを作成している。Frame クラスのコンストラクタには，タイトルバーに表示されるタイトル名を String 型で指定することができる。そのため，SamleWindow2 クラスのコンストラクタの引数において，タイトル名を String 型の文字列として指定し，コンストラクタ中の super（）でスーパークラスである Frame クラスのコンストラクタを呼び出す際に，その文字列を渡している。

また，Label クラスのオブジェクトもコンストラクタ中で作成しており，その際にラベル中のテキストの位置も定義している。これには，Label. LEF'T、Label. CENTER、Label. RIGHT の 3 種類が指定できるようになっている。さらに本プログラムでは，ウィンドウの大きさを決定するために，pack（）メソッドを用いる代わりに setSize（）を用いて明示的にサイズを設定している。

図 10．4 に，本プログラムを実行したときに表示されるウィンドウを示す。　ウィンドウを閉じるボタンを押すとウィンドウが閉じられるようにするためには，アプリケーションがイベントを処理できるように設定する必要がある。そのためには，まず発生したイベント（ここでは Event．WINDOW__DESTROY）を enableEvents（）メソッドでオペレーティングシステムから受信可能にし，さらに，その受信したイベントを処理するためのコードをオーバーライドした processEvent（）内で定義している。

図１０．４　SampleWindow2 プログラム

このアプリケーションでは，“HelloWorld！！”というテキストがラベルの中央に表示されており，閉じるボタンを押すことによって終了できるようになっている。

10．6　ボタンを伴うアプリケーションのプログラム

GUI コンポーネントとしてよく目にするのが，ボタンオブジェクトである。ボタンを押すことにより何らかの処理がトリガされ、実行される。例えば，ボタンを押すことにより，何らかの情報が処理され，結果が表示されるといった具合にである。本節では，前節のプログラムを拡張して，ラベルの代わりにボタンを貼り付けたプログラムを作成してみる。

```
1:import java.awt.*;      //AWT をインポート
2:
3:public class SampleButton extends Frame {
4:
5:   public SampleButton(String _title) {
6:      super(_title);
7:      enableEvents(AWTEvent.WINDOW_EVENT_MASK);
8:
9:      //ボタンクラスのオブジェクトを生成
10:      Button b = new Button("Push Me!!");
11:
12:      add(b);
13:      setSize(200,100);
14:      setVisible(true);
15:   }
16:
17:   public void processEvent(AWTEvent e) {
18:      if(e.getID() == Event.WINDOW_DESTROY) {
19:      System.exit(0);
20:      }
21:   }
22:
23:   public static void main(String args[]) {
24:      SampleButton sb = new SampleButton("SampleButton");
25:   }
26:}
```

プログラム１０．３　Button クラスの使用例（SampleButton.java）

プログラム 10．3 では，Button クラスのオブジェクトを作成し，Frame クラスを継承した SampleButton クラスに加えている。ボタンが押されたことに対する処理は，まだ何も記述されていない。

図 10．5 に，このプログラムを実行したときに生成される GUI を示す。図を見ても明らかなように，ボタンオブジェクトが表示されている。ただし，ボタンを押してもこのプログラムでは何も起こらない。

図１０．５　SampleButton プログラム

10．7　イベント

本節では，AWT におけるイベント処理のプログラムの仕方について述べる。

10．7．1　イベントとは

10. 6 節のプログラムでは, 画面上に表示されたボタンをマウス等でクリックしたとしても，見た目上ボタンが押されるだけで，その他の反応がまったくない。しかし，ボタンオブジェクトを GUI のコンポーネントとして利用した場合，ボタンが押されたことに対する処理を記述するのが普通である。この「ボタンが押される」という行為をイベントといい，それに対する処理をイベント処理という。

より一般的な言葉を使うならば，イベントとは各コンポーネントで生じる何らかの状態の変化で，そのイベントの発生（状態の変化）を契機に他のコンポーネントが何らかの状態の変化で、そのイベントの発生（状態の変化）を契機に他のコンポーネントが何らかの処理を施すことができるようになっている。

10．7．2　イベントオブジェクト

AWT　に所属するコンポーネントによって発生させられるイベントは, すべて java. awt. Event クラスのサブクラスであり，そのイベントの種類によってグループ化されている。例えば,「ボタンが押された」というイベントは ActionEvent クラスに属し,「マウスがクリックされた」というイベントは MouseEvent クラスに属する。そして，そのオブジェクトは，イベントが発生することにより生成されることになる。イベントを発生させるコンポーネントはイベントソースと呼ばれ　逆にイベントを通知され　処理を行うコンポーネントはイベントリスナと呼ばれる。

すなわち，イベントが発生させられて処理されるまでの流れは，以下のようになる。

1：イベントソースがイベント発生を検知し，イベントオブジェクトのオブジェクトを生成する。

2：イベントソースがイベントリスナにそのイベントオブジェクトを渡す。

3 : イベントオブジェクトを受け取ったイベントリスナが，そのイベントがどのイベントなのかを識別し，それに対する処理を行う。

10．7．3　イベントプログラム

プログラマが AWT を使って GUI を作るときに何らかのイベントに対する処理を記述する場合は，プログラム内で以下の手続きを踏んでおかなければならない。

① イベントリスナであるコンポーネントは，イベントを受け取る準備としてそのためのインタフエースを実装しておく。

② さらにイベントリスナは，ある特定のイベントを受け取った際の処理として，　そのイベントを識別し，それに対応する処理をそのインタフエースの実装の中で記述しておく。

③ イベントソースであるコンポーネントは，自身が発生させたイベントに対するイベオブジェクトを渡すべきイベントリスナを登録しておく。

10．8　イベントに対応したボタンを伴うアプリケーションのプログラム

それでは例として, プログラム 10. 3 で示したボタンを表示するプログラムを拡張して, ボタンが押されたことに対するイベント処理を施したアプリケーションプログラムを書いてみる。

```
1:import java.awt.*;
2:import java.awt.event.*;
3:
4:public class SampleButton2 extends Frame implements ActionListener {
5:
6:   public SampleButton2(String _title) {
7:      super(_title);
8:      enableEvents(AWTEvent.WINDOW_EVENT_MASK);
9:
10:      //ボタンクラスのオブジェクトを生成
11:      Button b = new Button("Push Me!!");
12:
13:      //ボタンのイベントを通知する先を登録
14:      b.addActionListener(this);
15:
16:      add(b);
```

```
17:     setSize(200,100);
18:     setVisible(true);
19:   }
20:
21:   //ActionListener で定義されているインタフェースの実装
22:   public void actionPerformed(ActionEvent e) {
23:     System.out.println("thank you!!");
24:   }
25:
26:   public void processEvent(AWTEvent e) {
27:     if(e.getID() == Event.WINDOW_DESTROY) {
28:     System.exit(0);
29:     }
30:   }
31:
32:   public static void main(String args[]) {
33:     SampleButton2 sb = new SampleButton2("SampleButton");
34:   }
35:}
```

プログラム１０．４　イベント処理（SampleButton2.java）

プログラム 10．4 では，イベントを取り扱うために，まず最初に java．awt．Event パッケージをインポートしている。

このプログラムにおけるイベントリスナは SamleButton クラスである。ボタンが押されたというイベントを受け付け，それに対する処理を実行するために，SampleButton クラスは ActionListener インタフエースを実装している。この場合，インブリメントしなければならないメソッドは actionPerformed（）であり，ボタンが押された際に実行される処理はこの中に記述される。このプログラムでは，イベント処理として，“thank you！！”という文字列を標準出力に書き出している。

図 10．6 にその様子を示す。

図１０．６　SampleButton2 プログラムを実行した場合

またこの場合，生成された Button クラスのオブジェクトがイベントソースとなる。ボタンが押されたというイベントをイベントリスナである SamleButton クラスに通知するために，addActionListener（）メソッドを用いて SampleButton クラスのオブジェクトをイベントリスナとして登録している。これにより，オブジェクト間のイベントの通知が可能になるのである。このプログラムでは，イベントリスナとして自分自身をコンストラクタ中で登録しなければならないため，addActionListener（）メソッドの引数で自分自身への参照である this を指定している。

具体的には，イベントが発生することにより，Button クラスがイベントオブジェクトである　ActionEvent クラスのオブジェクトを生成し，それを引数として actionPerformed（）を呼び出すことになる。この ActionEvent クラスには，そのイベントに関する情報が記述されている。actionPerformed（）には，“thank you！！”という文字列を出力するプログラムが記述されている。結果として，ボタンを押すことにより“thank　you！！”という文字列が標準出力に表示されることになる。

10．9　様々なレイアウト機能

AWT は，様々なレイアウト機能を利用することができ，レイアウトマネージャによって管理される。レイアウトマネージャとは，コンテナコンポーネント（Container クラスをスーパークラスに持つ Frame クラス等のクラス）上で配置されるサブコンポーネントを，見栄えの良い形で配置・管理する。表 10．2 に，標準で用意されているレイアウトマネージャを示す。

表10．2　標準で装備されている主なレイアウトマネージャクラス

名前	効用
BorderLayout	東，西、南、北、中央を指定して、コンポーネントを配置する
CardLayout	コンポーネントをカードとして取り扱い、1度に1枚の カードを表示するように配置する
FlowLayout	左から右方向にコンポーネントを配置する
GridBagLayout	大きさの異なるコンポーネントを、縦横に柔軟に配置する
GridLayout	行数と列数を指定して、コンポーネントを配置する

10．10　レイアウト機能を利用したアプリケーションのプログラム

通常，GUI アプリケーションを作成する場合，AWT で定義されている数多くのコンポーネントを組み合わせて利用することになる。1 つの Frame コンボーネントに複数個のサブコンポーネントが配置されることも当然必要となる。それでは，どうやってこれら複数個のサブコンポーネントの配置を適切に制御するのであろうか。

Java は，レイアウトマネージャによってこれを実現している。レイアウトマネージャは，Container クラスを継承した Frame クラスや Panel クラスなどのコンテナコンポーネントに適用される。Frame クラスにデフォルトで設定されているレイアウトマネージャは BorderLayout クラスである。プログラム 10．5 では，BorderLayout の機能を用いて，1 つの Frame コンポーネント上に 5 つの Button コンポーネントを配置している。

```
1:import java.awt.*;
2:import java.awt.event.*;
3:
4:public class SampleBorderLayout extends Frame implements ActionListener {
5:   Button[] b;
6:
7:   //コンストラクタ
8:   public SampleBorderLayout(String _title) {
9:      super(_title);
10:      enableEvents(AWTEvent.WINDOW_EVENT_MASK);
11:
12:      b = new Button[5];
13:
14:      add(b[0] = new Button("WEST"),BorderLayout.WEST);
15:      add(b[1] = new Button("CENTERT"),BorderLayout.CENTER);
16:      add(b[2] = new Button("EAST"),BorderLayout.EAST);
17:      add(b[3] = new Button("NORTH"),BorderLayout.NORTH);
18:      add(b[4] = new Button("SOUTH"),BorderLayout.SOUTH);
19:
```

```
20:     for(int i = 0; i < 5; i++) {
21:       b[i].addActionListener(this);
22:       b[i].setActionCommand(b[i].getLabel());
23:     }
24:
25:     pack();
26:     setVisible(true);
27:   }
28:
29:   // ActionListener で定義されているインタフェースの実装
30:   public void actionPerformed(ActionEvent e) {
31:     System.out.println(e.getActionCommand());
32:   }
33:
34:   public void processEvent(AWTEvent e) {
35:
36:     if(e.getID() == Event.WINDOW_DESTROY) {
37:       System.exit(0);
38:     }
39:   }
40:
41:   public static void main(String args[]) {
42:     SampleBorderLayout f = new SampleBorderLayout("SampleBorderLayout");
43:   }
44:}
```

プログラム１０．５　５つのボタンをレイアウトする例（SampleBorderLayout.java）

BorderLayout では，“East”，“West”，“South”，“North”，“Center”という５つの方角を指定することにより，サブコンポーネントの配置を決定することができる。BorderLayout では，オーバーライドされた add（Component comp，int　index）メソッドによって，追加するコンポーネント（comp）と追加する位置（index）を指定する[3]。図 10．7 に，このプログラムを実行したときに表示される GUI を示す。５つのボタンがそれぞれの方角に従って配置されているのが分かる。

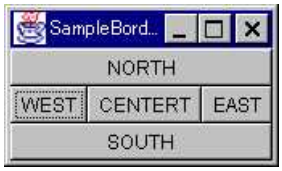

図１０．７　SampleBorderLayout プログラム

[3]　add(String *name*、Component *comp*)によっても追加できるが、推奨されていない。

プログラム 10．5 で，ボタンを押した際のイベント処理は，actionPerformed（）のメソッドの中で記述されている。処理としては、発生したイベントのコマンド名を標準出力（コマンドプロンプト）に出力しており、この処理には ActionEvent クラスの getActionCommand（）メソッドを用いている。各イベントは固有のコマンド名を持つことができ，任意の Button オブジェクトとそのイベントのコマンド名は，Button クラスの setActionCommand（）によって設定することができる。本プログラムでは，コマンド名としてボタンのラベル名を用いている。したがって，GUI の各ボタンを押すと，ラベル名が標準出力に表示されるはずである。
図 10．8 に，本プログラムを実行した場合にコマンドプロンプト上に文字列が表示される様子を示す。

図１０．８　SampleBorderLayout プログラムを実行した場合

10.11　より複雑なコンポーネントの配置を実現するプログラム

10.10 節では, Frame クラスのデフォルトのレイアウトマネージャ機能を用いて, Button クラスのオブジェクトを配置した。しかし，実際にアプリケーションを作成するならば，より柔軟に複数のコンポーネントを配置できなければならない。本節では, Panel クラスと，

10．10 節で用いた BorderLayout クラス以外のレイアウトマネージャを用いて，Button コンポーネントの配置を行うプログラムを示す。

```
1:import java.awt.*;
2:
3:public class SampleComplicatedLayout extends Frame {
4:
5:   public SampleComplicatedLayout(String _title) {
6:      super(_title);
7:      enableEvents(AWTEvent.WINDOW_EVENT_MASK);
8:   }
9:
10:   public void processEvent(AWTEvent e) {
11:      if(e.getID() == Event.WINDOW_DESTROY) {
12:         System.exit(0);
13:      }
14:   }
15:
16:   public static void main(String args[]) {
17:      SampleComplicatedLayout f
18:               = new SampleComplicatedLayout("SampleComplicatedLayout");
19:
20:      Panel panel1 = new Panel(new GridLayout(3, 3));
21:      Panel panel2 = new Panel(new FlowLayout());
22:      Panel panel3 = new Panel(new BorderLayout());
23:
24:      for(int i = 1; i <= 6; i++) {
25:         panel1.add(new Button("CENTER" + i));
26:      }
27:
28:      for(int i = 1; i <= 3; i++) {
29:         panel2.add(new Button("NORTH" + i));
30:      }
31:
32:      panel3.add(new Button("SOUTH1"), BorderLayout.WEST);
33:      panel3.add(new Button("SOUTH2"), BorderLayout.CENTER);
```

```
34:      panel3.add(new Button("SOUTH3"), BorderLayout.EAST);
35:      panel3.add(new Button("SOUTH4"), BorderLayout.NORTH);
36:      panel3.add(new Button("SOUTH5"), BorderLayout.SOUTH);
37:
38:      f.add(new Button("WEST"), BorderLayout.WEST);
39:      f.add(panel1, BorderLayout.CENTER);
40:      f.add(new Button("EAST"), BorderLayout.EAST);
41:      f.add(panel2, BorderLayout.NORTH);
42:      f.add(panel3, BorderLayout.SOUTH);
43:
44:      f.pack();
45:      f.setVisible(true);
46:  }
47:}
```

プログラム１０．６　Panel クラスの使用例（SampleComplicatedLayout.java）

プログラム 10．6 では，Panel クラスを新たに導入している。Panel コンボーネントは，ある複数のコンボーネントを自らのサブコンポーネントとして 1 つにグループ化し，それらに 10．9 節で述べたレイアウト機能を適用することができる，格納用のコンテナコンポーネントである。Panel クラスのオブジェクトを作成するときには，コンストラクタの引数として LayoutManager のオブジェクトを指定することができる。
本プログラムでは，panell，pane12，pane13 のそれぞれに，GridLayout，FlowLayout，BorderLayout を適用している。GridLayout では，オブジェクトを生成する際に行数と列数を指定する必要がある。

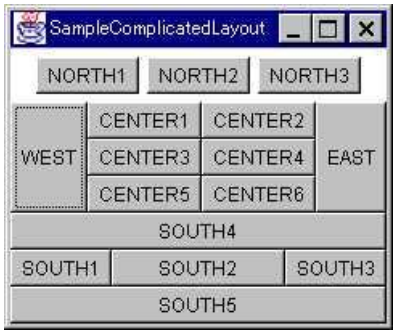

図１０．９　SampleComplicated プログラム

10．12　テキストを入力するアプリケーションのプログラム

GUI を通して，ユーザに何らかの情報を入力してもらうアプリケーションをよく目にす

る。AWT では，テキストを入力させるためのクラスとして TextField と TextArea を用意している。TextField クラスも TextArea クラスも TextComponent クラスをスーパークラスに持ち，多くのメソッドを共有している。違いは，TextField が単一行の入力用なのに対し，TextArea は複数行の入力ができるところである。本節では，TextField にのみ着目してプログラムを作成してみる。

```
1:import java.awt.*;
2:import java.awt.event.*;
3:
4:public class SampleTextField extends Frame implements ActionListener {
5:   Button b;
6:   TextField tf1, tf2;
7:
8:   //コンストラクタ
9:   public SampleTextField(String _title) {
10:      super(_title);
11:      enableEvents(AWTEvent.WINDOW_EVENT_MASK);
12:
13:      //ボタンクラスのオブジェクトを生成
14:      b = new Button("LEFT to RIGHT");
15:      b.addActionListener(this);
16:
17:      //テキストフィールドのオブジェクトを生成
18:      tf1 = new TextField(10);
19:      tf2 = new TextField(10);
20:      tf2.setEditable(false);
21:
22:      add(b, BorderLayout.SOUTH);
23:      add(tf1, BorderLayout.WEST);
24:      add(tf2, BorderLayout.EAST);
25:      pack();
26:      setVisible(true);
27:   }
28:
29:   // ActionListener で定義されているインタフェースの実装
30:   public void actionPerformed(ActionEvent e) {
```

```
31:      tf2.setText(tf1.getText());
32:      tf1.setText("");
33:   }
34:
35:   public void processEvent(AWTEvent e) {
36:      if(e.getID() == Event.WINDOW_DESTROY) {
37:         System.exit(0);
38:      }
39:   }
40:
41:   public static void main(String args[]) {
42:      SampleTextField sb = new SampleTextField("SampleTextField");
43:   }
44:}
```

プログラム１０．７　TextField クラスのの使用例（SampleTextField.java）

プログラム 10．7 は、TextField クラスのオブジェクトを 2 つ，Button クラスのオブジェクトを 1 つ生成し、ボタンを押すことにより，左側のテキストフイールドコンポーネントに入力されているテキストを右側のテキストフィールドコンポーネントに複写する。TextField クラスのオブジェクトの作成には，コンストラクタの引数にカラム数（文字数または幅）を指定できる。このプログラムでは 10 としている。TextField クラスのコンストラクタの引数には何も指定しなくてもいいし，あらかじめ表示しておきたいテキストを String 型で指定することもできる。

さらに、複写される右側のテキストフィールドには，TextField のメソッドの 1 つである setEditable（）で false を指定し，ユーザがキーボードから直接入力できないように設定している。また、このプログラムでは用いていないが，TextArea クラスのオブジェクトを生成するときは，コンストラクタの引数としてカラム数の前に行数も指定し、TextArea コンボーネントの大きさを決定してやる必要がある。もちろん，まったく何も指定しなくても生成することは可能である。

ボタンが押されることによるイベントの処理は，10．6 節でも述べたように actionPerformed（）メソッド内に記述される。本プログラムでは，左側のテキストフィールドから getText（）メソッドで取り出した String 型のテキストを，右側のテキストフィールドに getText（）メソッドで書き込み，さらにその後，左側のテキストフィールドに空文字列を setText（）メソッドで書き込んでいる。図 10．10 に，本プログラムを実行したときに表示される GUI を示す。

図１０．１０　SampleTextField プログラム

10．13　項目チェック機能を備えたアプリケーションのプログラム

　何らかの事柄に対して，ユーザに選択を促すようなアプリケーションもよく見る。この場合，選択肢としてあらかじめいくつかの候補を表示し，アプリケーションを利用するユーザにそれらの中のいくつかを選択してもらうことになる。Java では，このような機能を Checkbox クラスで提供している。プログラム 10．8 にサンプルプログラムを示す。

```
1:import java.awt.*;
2:
3:public class SampleCheckbox extends Frame {
4:
5:   //コンストラクタ
6:   public SampleCheckbox(String _title) {
7:      super(_title);
8:      enableEvents(AWTEvent.WINDOW_EVENT_MASK);
9:
10:      Panel p1 = new Panel();
11:      Panel p2 = new Panel();
12:
13:      add(p1,BorderLayout.WEST);
14:      add(p2,BorderLayout.EAST);
15:
16:      p1.setLayout(new GridLayout(2, 1));
17:      p2.setLayout(new GridLayout(4, 1));
18:
19:      CheckboxGroup cbg = new CheckboxGroup();
20:      p1.add(new Checkbox("MALE", cbg, true));
21:      p1.add(new Checkbox("FEMALE", cbg, false));
22:
23:      p2.add(new Checkbox("fixed phone"));
24:      p2.add(new Checkbox("mobile phone"));
25:      p2.add(new Checkbox("facsimile"));
26:      p2.add(new Checkbox("E-mail address"));
```

```
27:
28:     pack();
29:     setVisible(true);
30:  }
31:
32:  public void processEvent(AWTEvent e) {
33:     if(e.getID() == Event.WINDOW_DESTROY) {
34:        System.exit(0);
35:     }
36:  }
37:
38:  public static void main(String args[]) {
39:     SampleCheckbox sb = new SampleCheckbox("SampleChexkbox");
40:  }
41:}
```

プログラム１０．８　Checkbox クラスの使用例（SampleCheckbox.java）

Checkbox クラスを単独でインスタンス化すると，他の Checkbox オブジェクトと関連付けられないオブジェクトとして生成される。この場合，その項目を選択したとしても、他の Checkbox クラスのオブジェクトに状態の変化は生じない。また，ボックス自体は，MS－Windows の場合は正方形で表示される。

一方，CheckboxGroup クラスをインスタンス化し，作成した複数の Checkbox クラスのオブジェクトをその CheckboxGroup クラスのオブジェクトに関連付けると，複数個の Checkbox オブジェクトをグループ化でき，ユーザはそれらグループの中で唯一のものしか選択できなくなる。この場合，ボックス自体は，MS－Windows では円形で表示されることになる。 CheckboxGroup クラスに Checkbox クラスを関連付けるためには、Checkbox クラスのオブジェクトを生成する際, コンストラクタに CheckboxGroup クラスのオブジェクトを指定する必要がある。

プログラム 10．8 では、“MALE”，“FEMALE”とラベル化された Checkbox クラスのオブジェクトは、1 つのグループとして CheckboxGroup クラスのオブジェクトに関連付けられており，それら 2 つは排他的にしか選択できないようになっている。一方、“fix phone”，“mobile　phone”，“facsimile”，“E－mail　address”とラベル化された Checkbox クラスのオブジェクトは CheckboxGroup クラスに関連付けされていないため，これらは同時に 2 つ以上選択することができるようになっている。

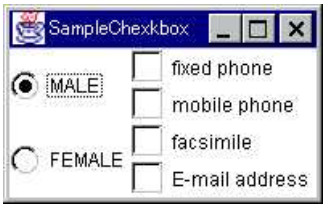

図１０．１１　SampleCheckbox クラスのプログラム

10．14　ポップアップメニューを持つアプリケーションのプログラム

前節と同様に，いくつかの選択肢の中から 1 つのものを選択させる方法として，ポップアップメニューを用いることもできる。これには Choice クラスが利用される。本節では，複数の色の中から 1 つを選択するアプリケーションのプログラム例を示す。

```
1:import java.awt.*;
2:
3:public class SampleChoice extends Frame {
4:
5:   //コンストラクタ
6:   public SampleChoice(String _title) {
7:      super(_title);
8:      enableEvents(AWTEvent.WINDOW_EVENT_MASK);
9:
10:      //Choice クラスのオブジェクトの生成
11:      Choice ch = new Choice();
12:
13:      //選択肢の追加
14:      ch.addItem("BLACK");
15:      ch.addItem("BLUE");
16:      ch.addItem("YELLOW");
17:      ch.addItem("RED");
18:      ch.addItem("WHITE");
19:
20:      add(ch, BorderLayout.CENTER);
21:
22:      setSize(200, 100);
23:      setVisible(true);
24:   }
25:
```

```
26:   public void processEvent(AWTEvent e) {
27:      if(e.getID() == Event.WINDOW_DESTROY) {
28:         System.exit(0);
29:      }
30   }
31:
32:   public static void main(String args[]) {
33:      SampleChoice sb = new SampleChoice("SampleChoce");
34:   }
35:}
```

プログラム１０．９　Choice クラスの使用例（SampleChoice.java）

Choice クラスのオブジェクトを作成する際は，コンストラクタの引数には何も指定できない。オブジェクトを生成した後に，addItem（）メソッドによって，選択肢を加えていく。図 10．12 に，本プログラムにより表示される GUI を示す。

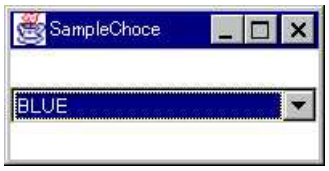

図１０．１２　SampleChoice プログラム

Choice オブジェクトでは、現在選択されている項目がボックスの中に表示される。現在選択されている項目の名前（テキスト）は，getSelectedItem（）メソッドにより String 型として取得することができる。

10．15　項目をリストするアプリケーションのプログラム

前節で述べた Choice クラスと同様，List クラスも項目の選択に利用できる。Choice クラスとの違いは，List クラスがスクロール機能を伴って項目を表示し，さらにユーザは 1 つ，または複数の項目を一度に選択できるところである。プログラム 10．10 では，前節と同様のものを List クラスで実現している。

```
1:import java.awt.*;
2:
3:public class SampleList extends Frame {
4:
5:   //コンストラクタ
```

```
6:   public SampleList(String _title) {
7:      super(_title);
8:      enableEvents(AWTEvent.WINDOW_EVENT_MASK);
9:
10:     //List クラスのオブジェクトの生成
11:     List I = new List();
12:     I.addItem("BLACK");
13:     I.addItem("BLUE");
14:     I.addItem("YELLOW");
15:     I.addItem("RED");
16:     I.addItem("WHITE");
17:
18:     add(I, BorderLayout.CENTER);
19:
20:     setSize(200, 100);
21:     setVisible(true);
22:  }
23:
24:  public void processEvent(AWTEvent e) {
25:     if(e.getID() == Event.WINDOW_DESTROY) {
26:        System.exit(0);
27:     }
28:  }
29:
30:  public static void main(String args[]) {
31:     SampleList sb = new SampleList("SampleList");
32:  }
33:}
```

プログラム１０．１０　List クラスの使用例（SampleList.java）

また，図 10．13 に，実行した際の GUI を示す。

図１０．１３　SampleList プログラム

10．16　メニューバーを伴うアプリケーションのプログラム

MS－Windows では，私たちが目にするほとんどのアプリケーションにメニューバーが付いている。本節では，アプリケーションへのメニューバーの付け方を学ぶ。AWT において、メニューバーは MenuBar クラス，Menu クラス，そして MenuItem クラスの 3 種類のオブジェクトで作成することができる。プログラム 10．11 に，メニューバー中にメニュー項目として Files，Edit，Help の 3 つを持つアプリケーションを作成するプログラムを示す。

```
1:import java.awt.*;
2:import java.awt.event.*;
3:
4:public class SampleMenuBar extends Frame implements ActionListener {
5:   MenuBar mb;
6:   Menu m1, m2, m3;
7:   MenuItem mi;
8:
9:   //コンストラクタ
10:   public SampleMenuBar(String _title) {
11:      super(_title);
12:      enableEvents(AWTEvent.WINDOW_EVENT_MASK);
13:
14:      //MenuBar クラスのオブジェクトの生成
15:      mb = new MenuBar();
16:
17:      //MenuBar に MenuItem を追加
18:      mb.add(m1 = new Menu("Files"));
19:      mb.add(m2 = new Menu("Edit"));
20:      mb.add(m3 = new Menu("Help"));
21:
22:      //それぞれに Menu に MenuItem を追加
```

```
23:     m1.add(new MenuItem("File Open"));
24:     m1.add(new MenuItem("File Close"));
25:     m1.add(new MenuItem("-"));
26:     m1.add(mi = new MenuItem("Exit"));
27:     mi.addActionListener(this);
28:
29:     m2.add(new MenuItem("Cut"));
30:     m2.add(new MenuItem("Copy"));
31:     m2.add(new MenuItem("Paste"));
32:
33:     m3.add(new MenuItem("Options"));
34:     m3.add(new MenuItem("Version"));
35:
36:     //Frame と MenuBar との関連付け
37:     setMenuBar(mb);
38:
39:     setSize(200, 150);
40:     setVisible(true);
41:   }
42:
43:   //ActionListener で定義されているインタフェースの実装
44:   public void actionPerformed(ActionEvent e) {
45:     if(e.getActionCommand().equals("exit")) {
46:       System.exit(0);
47:     }
48:   }
49:
50:   public void processEvent(AWTEvent e) {
51:     if(e.getID() == Event.WINDOW_DESTROY) {
52:       System.exit(0);
53:     }
54:   }
55:
56:   public static void main(String args[]) {
57:     SampleMenuBar sb = new SampleMenuBar("SampleMenuBar");
58:   }
```

59:}

プログラム１０．１１　Menu クラスと MenuBar クラス（SampleMenuBar.java）

MenuBar クラスは，それ自体ほとんど機能を提供しないが，Menu クラスや MenuItem と Frame クラスを関連付ける重要な役割を持っている。MenuBar クラスと Frame クラスを関連付けるためには，Frame クラスの add（）メソッドの代わりに SetMenuBar（）メソッドを用いなければならない。また，MenuBar クラスと Menu を関連付けるためには，MenuBar クラスの add（）メソッドを用いなければならない。

Menu クラスは，MenuBar から引き出されるプルダウンメニューを提供するクラスである。Menu クラスのオブジェクトは，メニューバー中に表示されるメニューの種類の数だけ作成されなければならない。各メニューの名前は，コンストラクタで指定することができる。本プログラムでは，Files，Edit，Help の 3 つを指定して，それぞれを Frame クラスの add（）メソッドで Frame クラスに追加している。

MenuItem クラスは，それぞれのメニュー中における項目のオブジェクトとなる。項目名はコンストラクタで指定することができ，Menu クラスの add（）で各メニューに追加できる。ただし，項目名として“－”を用いると，その前後の項目間にセパレータが挿入されるようになっている。

Menu クラスには，MenuItem クラスのほかに Menu クラスのオブジェクトもサブコンボーネントとして追加することができる。この場合，メニューの中からサブメニューが表れることになるが，本プログラムでは用いていない。

また，本プログラムでは，例として File メニュー中の Exit 項目（プログラム中の MenuItem 型の変数 mi）に関してのみイベントの処理を記述している。一般に，メニュー中の項目が選択されると ActionEvent 型のイベントオブジェクトを生成し，actionPerformed（）メソッドが呼ばれる。したがって，この中にそのイベントに対する処理を書くことになる。この場合は，アプリケーションを終了する処理を記述している。図 10．14 に，本プログラムを実行したときに表示される GUI を示す。

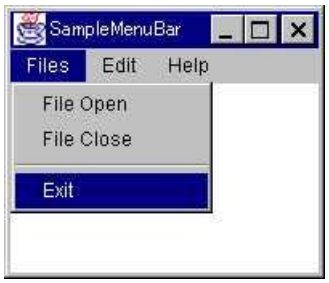

図１０．１４　SampleMenuBar プログラム

11. アプレットプログラミング

これまでJavaで作成したプログラムは,アプリケーションプログラムである。本章では,Java で作成することのできるもうひとつのプログラムであるアプレットの作成について述べる。アプレットはアプリケーションとは異なり，インターネットブラウザなどのプログラムの中で実行される。

本草では，以下について学習する。

●アプレットとは
●アプレットの作成
●アプレットと HTML

11．1　アプレットとは

アプレットとは，インターネットを介して，ユーザのコンピュータ内で実行されているインターネットブラウザやアプレットビユーワなどの，別のアプリケーションにダウンロードされて実行されるプログラムである。プログラミング上，アプレットがアプリケーションと大きく異なるのは，アプレットがプログラム中に main（）メソッドを持たないことである。これは，アプリケーションが起動されると main（）メソッドから自動的に実行されるのに対して，アプレットが実行されるためには別の何らかの制御プログラムを必要とすることを意味している。

11．2　アプレットの簡単なプログラム

11．2．1 アプレットのプログラム例

アプレットのプログラミングには AWT を利用することができる。AWT により，様々なオブジェクトをアプレット上にグラフィカルに表示できる。本節では，簡単なプログラム例を示す。

```
1:import java.applet.*;
2:import java.awt.*;
3:
4:public class SampleApplet extends Applet {
5:   Label I;
6:
7:   public void init() {
8:      setLayout(new BorderLayout());
9:      I = new Label("Hello World!!");
```

```
10:      add(I, BorderLayout.CENTER);
11:  }
12:}
```
プログラム１１．１　簡単なアプレットの例（SampleApplet.java）

アプレットを作成するためには，必ず Applet クラスを継承して作る必要があり，Applet クラスを使用するために java．applet パッケージをインポートしている。また，このプログラムでは java．awt パッケージもインポートしており，AWT を使用して，アプレット上でラベルコンポーネントを表示できるようになっている。上述のように，アプレットのプログラムには main () メソッドが含まれておらず, このプログラムではすべての処理が init （）メソッド中に記述されている。

11．2．2　アプレットと埋め込んだ HTML ファイルの例

アプレットのコードは，HTML ファイルの中でアプレットタグを用いて指定されそのHTML ファイルと一緒に，ウェブブラウザといったユーザ端末内アプリケーションにダウンロードされる。以下に，上述した SampleApplet クラスを埋め込んだ HTML ファイルの例を示す。

```
＜HTML＞
    ＜HEADER＞
    ＜TITLE＞SampleApplet＜／TITLE＞
    ＜／HEADER＞

    ＜BODY＞
      ＜CENTER＞
        ＜APPLET CODE＝“SampleApplet．class”WIDTH＝200　HEIGHT＝100＞
        ＜PARAM NAME＝“ZONE”　VALUE＝“GMT”＞
        ＜／APPLET＞
      ＜／CENTER＞
    ＜／BODY＞
＜／HTML＞
```

ここでは，SampleApplet．java ファイルをコンパイルした SampleApplet．class ファイルが，この HTML ファイルと同じフォルダ中に存在することが前提となっている。このファイルは SampleApplet．html と名前を付けて保存する。

11．2．3　アプレットの表示

アプレットが表示できるかどうかは，11．2．2 項で示した HTML ファイルをブラウザやアプレットビューワで指定すればよい。指定すると，“HelloWorld！！”と書かれたアプレットが表示される。

ここでは，JDK に付属されているアプリケーションであるアプレットビューワで表示してみる。コンソールにおいて，前節までに作った SampleApplet．class と SampleAppp1et．html ファイルが存在するフォルダにし，コマンドラインで以下の一行を入力する。

＞appletviewer SapmleApplet．html

すると，アプレットビューワアプリケーションが立ち上がり，アプレットが読み込まれ，図 11．1 に示すようにアプレットが表示される。

図１１．１　アプレットビューワで見たアプレットの例

11．3　アプレットと HTML

11．3．1　アプレットタグ

アプレットは，HTML ファイルの中でアプレットタグで指定される。アプレットタグは＜APPLET＞で始まり，＜／APPLET＞で終了する。アプレットタグの一般的な構文を以下に示す。

＜APPLET 属性名＝属性値［　属性名＝属性値.....］＞
［＜PARAM　NAME＝パラメータ名　VALUE＝バラメータ値＞］
［＜PARAM　NAME＝パラメータ名　VALUE＝パラメータ値＞］
...
＜／APPLET＞

（1）属性値の指定

アプレットタグ（＜APPLET＞）中では，属性を指定することができる。その HTML で

書かれたページにどのアプレットを表示するのか，どのくらいの大きさで表示するのかといったことを指定する。属性として指定できる主なものを表 11．1 に示す。

表11．1　アプレットタグに指定できる属性

属性名	属性値として指定すること	必要性
CODE	アプレットとしてダウンロードするクラスのファイル	必須
WIDTH	アプレットを表示する際の幅のサイズ	必須
HEIGHT	アプレットを表示する際の高さのサイズ	必須
CODEBASE	CODEで指定したクラスのファイルが存在するフォルダまでのURL	オプション
ALT	指定されたアプレットを実行できない場合に、代わりに表示されるテキスト	オプション
NAME	アプレットのインスタンスに付ける名前	オプション
ALIGN	アプレットが表示される位置	オプション

（2）パラメータ値の指定

プログラマは，アプレットタグである＜APPLET＞と＜／APPLET＞の間に，アプレット自身に渡すパラメータを指定することができる。パラメータには，特に規定された種類のものはなく，プログラマが自由に「パラメータ名」と「パラメータ値」の組み合わせやその数を決定することができ，それらに対する解釈や処理も，プログラマがアプレットクラス中に自由に記述することができるようになっている。例えば，11．2．2 項で示した HTML ファイルでは，バラメータ名として“ZONE（タイムゾーン)”を，パラメータ値として“GMT”を指定している。

11．3．2　アプレットの制限

一般にアプレットは，ネットワーク上を WWW サーバからダウンロードされてユーザ端末内のブラウザ等で実行されるモジュールである。ユーザ端末からしてみれば，どこの誰が作ったモジュールであるかは定かではなく，もし悪意のあるプログラマが作ったものであれば何をされるか分かったものではない。そういったセキュリティの意味で，アプレットのブラウザ内での振る舞いは非常に制限されている。

例えば，標準のセキュリティの設定では，ダウンロードされたアプレットはダウンロード先の端末のファイルを読み書きすることは禁じられている。また，アプレットが通信することが許されているサーバも，そのアプレット自身がダウンロードされてきた元の WWW サーバに限られている。アプレットは非常に便利なものであるが，プログラマやユーザは，アプレットを作成したり利用したりする際には，セキュリテイといった側面に気をつけなければならない。

11．4　Applet クラス

11．4．1　Applet クラスの階層

Applet クラスの継承関係は，図 11．2 に示すようになる。これにより，Applet クラスが

どういったメソッドを継承し，何ができるのかといったことが分かる。

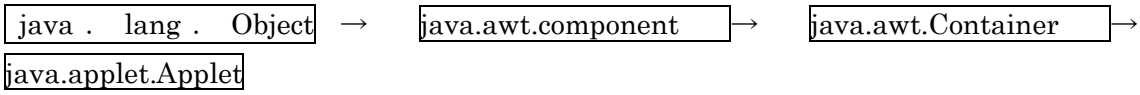

図 11．2　Applet クラスの継承関係

１１．４．２　Applet クラスのメソッド

表１１．２　Applet クラスの主なメソッド

返り値	メソッド名	効用
void	destroy()	アプレットが開放されるときに呼ばれる
AppletContext	getAppletContext()	アプレットのコンテキストを取得する
String	getAppletInfo()	アプレットに関する情報を取得する
URL	getCodeBase()	アプレットの基底URLを取得する
URL	getDocumentBase()	アプレットに埋め込まれているHTML ファイルへのURLを取得する
String	getParameter(String *name*	引数で指定されるパラメータ名の パラメータ値を取得する
void	init()	アプレットがロードされたときによばれる
boolean	isActive()	アプレットがアクティブかどうか を判定する
void	start()	アプレットが実行されるときに呼ばれる
void	stop()	アプレットの実行が停止されたときに 呼ばれる

表 11．2 に，Applet クラスで定義されているメソッドの中で，主なメソッドを示す。詳細は JDK 付属のドキュメントを参照してほしい。

11．4．3　アプレットのライフサイクル（開始と終了）

前述のように，アプレットのプログラムは main（）メソッドを持たない。代わりに，11．4．2 項で述べたように，アプレットのプログラムでは，そのライフサイクルに対応して 4 つのメソッドが呼ばれる。それらは，順番に init（），start（），stop（），そして destroy（）である。

具体的には，ブラウザやアプレットビユーワに，始めてアプレットをそれが埋め込まれている HTML ファイルと共にロードすると，init（）メソッドと start（）の両方が順番に呼ばれるが，その HTML ファイルを再ロードしても，アプレット自身は既にロードされてメモリ内に存在しているので start（）メソッドしか呼ばれないことになる。stop（）メソッドと destroy（）メソッドの関係も同様であり，destroy（）メソッドはアプレット自身がメモリから消去されるときにのみ呼ばれることになる。

11．4．4　パラメータ値の取得

アプレット自身は，HTML ファイルと共にダウンロードされた後起動される際に，

getParameter（）メソッドの引数でパラメータ名を指定することにより，対応するパラメータ値を String 型で取得することができる。例えば，上述のプログラム例では，以下のようにである。

String paramValue；
ParamValue＝getParameter（“ZONE”）；

これにより，String 型の paramValue という変数には“GMT”という文字列が格納されることになる。このようにして取得したパラメータ値をどう使うかは，プログラマ次第である。

11．5　アプレットプログラム例

本節では，アプレットのプログラム例を示す。

```
1:import java.applet.*;
2:import java.awt.*;
3:import java.awt.event.*;
4:
5:public class SampleApplet2 extends Applet implements ActionListener {
6:   Button[] b;
7:   TextField tf;
8:
9:   public void init() {
10:      System.out.println("Applet is loaded.");
11:      setLayout(new BorderLayout());
12:  }
13:
14:  public void start() {
15:      System.out.println("Applet is started.");
16:
17:      b = new Button[4];
18:      add((tf = new TextField()),BorderLayout.CENTER);
19:      add((b[0]   = new Button("WEST")),BorderLayout.WEST);
20:      add((b[1]   = new Button("EAST")),BorderLayout.EAST);
21:      add((b[2]   = new Button("NORTH")),BorderLayout.NORTH);
22:      add((b[3]   = new Button("SOUTH")),BorderLayout.SOUTH);
```

```
23:
24:     for(int i = 0; i <b.length; i++) {
25:         b[i].addActionListener(this);
26:         b[i].setActionCommand(b[i].getLabel() );
27:     }
28:   }
29:
30:   public void stop() {
31:     System.out.println("Applet is stopped.");
32:   }
33:
34:   public void destroy() {
35:     System.out.println("Applet is destroyed.");
36:   }
37:
38:   public void actionPerformed(ActionEvent e) {
39:     tf.setText(e.getActionCommand());
40:   }
41:}
```

プログラム１１．２　アップレットの例

プログラム 11．2 では，AWT を利用してボタンコンポーネントとテキストフィールドコンポーネントをアプレット上に配置している。init（）メソッド中では，アプレット自身のレイアウトマネージャを指定し，start（）メソッド中でそれぞれのコンポーネントを作成し，アプレットに追加している。

ボタンが押されたときのイベントの処理としては，押されたボタンのコマンド名をテキストフィールドに表示するようにしている。このために，Applet クラスを継承した SampleApplet2 クラスは ActionListener インタフエースを実装しており，イベントの処理に対するコードは actionPerformed（）メソッド中に記述してある。図 11．3 に，このプログラム（アプレット）を表示させた場合の様子を示す。

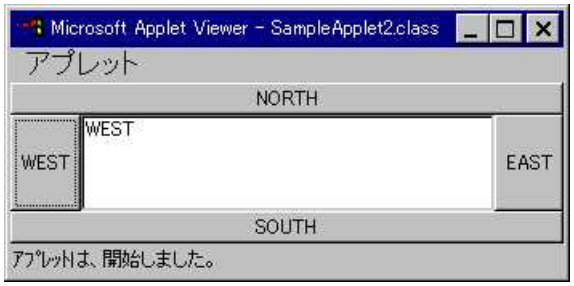

図１１．３　SampleApplet2 アプレット