第4章　イベント処理と GUI部品

アプレットのようなグラフィカルなユーザーインタフェース（GUI）を持つプログラムは、ユーザーがマウスのボタンを押したとき、キーボードのキーを押したときなどに何らかの反応を示さなければいけないことが多い。この章では、このような何らかの イベントに反応するプログラムの書き方について学習する。

また、GUI 部品のボタンや、ユーザーがデータを入力するためのテキストフィールドをユーザーインタフェースに付け加える方法についても学ぶ。

また、この章では、Java に特有のデータ型・クラスに関する話題（総称クラス・ゴミ集めなど）をいくつか紹介する。また、Java のプログラムで頻繁に利用することになる重要なメソッドなどもここで紹介する。

例外処理の **try**〜**catch** 文は C 言語にはない構文なので、ここで紹介する。

4.1　イベント処理

ユーザーがマウスボタンをクリックした、キーボードのキーを押した、などのイベントに対応するためには、＿＿＿＿＿＿＿＿＿＿＿＿（空欄 4.1.1）と呼ばれるメソッドを定義する必要がある。

例題 4.1.1 マウスボタン

ファイル MouseTest.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;  /* 1 */

public class MouseTest extends JApplet
                       implements MouseListener { /* 2 */
  int x=50, y=20;

  @Override
  public void init() {
    addMouseListener(this); /* 3 */
  }

  public void mouseClicked(MouseEvent e) { /* 4 */
    x = e.getX();  y = e.getY();
    repaint();
    return;
```

```
  }

  public void mousePressed(MouseEvent e) {}  /* 5 */
  public void mouseReleased(MouseEvent e) {} /* 5 */
  public void mouseEntered(MouseEvent e) {}  /* 5 */
  public void mouseExited(MouseEvent e) {}   /* 5 */

  @Override
  public void paint(Graphics g) {
    super.paint(g);
    g.drawString("HELLO␣WORLD!", x, y);
  }
}
```

このプログラムは、文字列を表示し、マウスがクリックされると、その場所に文字列を移動する。

いくつか注目すべき点がある。

- まずイベントを扱うために java.awt.event.*を import している。(/* 1 */)

  イベントを扱うプログラムは大抵この import 文が必要になる。

- さらにこのクラスが、mouseClicked というメソッドを持っていることを示すために、MouseListener という＿＿＿＿＿＿＿＿＿＿(空欄 4.1.2)を implement していることを宣言している。(/* 2 */)

- init メソッドで、addMouseListener(this) を呼んで、マウスのイベントを、this オブジェクトに結び付けている。(/* 3 */)(this は一般にメソッドを実行中のオブジェクト自身を指す。詳しくは後述する。 ここでは実質的には、mouseClicked メソッドを指す。)

- mouseClicked というマウスボタンのクリックに対応するイベントハンドラを定義している。(/* 4 */) mouseClicked メソッドは MouseEvent 型の引数を受け取る。

- MouseListener インタフェースの他のメソッド (mousePressed, mouseReleased, mouseEntered, mouseExited) は何もしないメソッドとして、いちおう定義しておく。(/* 5 */)

このクラスは、文字列を表示する位置をフィールド x, y として保持している。mouseClicked メソッドは、これらのフィールドを、マウスの押された位置にしたがって変更する。マウスの押された位置 (単位はピクセル) を知るには、MouseEvent クラスの getX , getY というメソッドを使う。そのあと repaint を呼び出して再描画を要求する。repaint は、JApplet クラスで定義済のメソッドで、その中で paint メソッドを呼び出している。

フィールドの宣言　フィールド (インスタンス変数) の宣言は、クラス定義の中に、メソッドの定義と同じレベルに (メソッドの定義の外に) 並べて書く。フィールドはそのクラス中のすべてのメソッドから参照することができる。(ある意味で C 言語の大域変数と似ている。)

前述したようにクラスはオブジェクトの雛型である。MouseTest クラスに x, y というフィールドを宣言したということは、MouseTest クラスのインスタンスが作られるときに、JApplet クラスが持っているすべての構成要素の他に、x, y という名前の、int 型の構成要素ができるということである。

paint メソッドの中でこの x, y を参照している。このようにメソッドの中で自分自身のフィールドやメソッド（スーパークラスで定義されているものも含む）を参照するときは、ピリオドを使った記法は必要ない。

フィールドはオブジェクトが存在している間は値を保持している。これに対して、メソッドの中で宣言された変数（例えば、Othello.java の i や j）の寿命はメソッドの呼出しの間だけである。2 度め以降の呼び出しでも以前の値は保持していない。

4.2 this

this は、メソッドを所有しているオブジェクト自身を指す Java のキーワードである。他の言語では、self という言葉が使われることがある。

Java ではメソッドは次のような形で呼び出される。

```
object.method(arg1, arg2, ... )
```

このとき、. の前に書かれている *object* も内部的にはメソッドの引数の一つとして渡されている。（でないと、フィールドや他のメソッドを参照できない。）これをメソッドの中で明示的に取り出すのが、**this** キーワードである。

4.3 インタフェース

インタフェース（interface）は、あるクラスが特定の名前と型のメソッドを持っていることを示すために使われる。そしてそのクラスのインスタンスが、示されたメソッドを必要とする場所で使用できることを示す。インタフェースは C++などにはない、Java 特有のメカニズムである。一言でいえば、メソッドの定義を持たず、＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿[（空欄 4.3.1）]のみを指定したクラスみたいなもののことである。Java は多重継承（複数のスーパークラスを継承すること）を許さない代わりに、このインタフェースという仕組みを提供している。

MouseListener インタフェースの定義（ただし一部簡略化）

```
public interface MouseListener {
  public void mouseClicked(MouseEvent e);
  public void mousePressed(MouseEvent e);
  public void mouseReleased(MouseEvent e);
  public void mouseEntered(MouseEvent e);
  public void mouseExited(MouseEvent e);
}
```

例えば、インタフェース MouseListener の定義は、上のように MouseClicked などのメソッドの引数、戻り値の型を宣言している。(このインタフェースの定義は、もともと用意されているので、自分でする必要はない。) クラスと異なり、このメソッドの具体的な定義はインタフェース内のどこにもない。

あるクラスが、あるインタフェースを実装していることを宣言するためには _____(空欄 4.3.2) というキーワードを用いる。例えば、addMouseListener の引数は MouseListener インタフェースを実装していなければならない。

Q 4.3.1 Foo という名前のアプレットクラス (つまり JApplet を継承するクラス) に mouseClicked などいくつかのイベントハンドラを定義して、マウスイベントに反応するプログラムを作成するとき、import 文と public class Foo extends JApplet に続く 2 ワードを書け。
答: public class Foo extends JApplet _____

Q 4.3.2 mouseClicked メソッドの定義の中で repaint() の呼出しがなければ、MouseTest.java はどのような振舞いをするか?
答: _____

問 4.3.3 Othello.java を改良して、マウスで指示をすると石を置けるようにせよ。つまり、盤面をクリックすると、空→白丸→黒丸→空→…の順に変わるようにせよ。

問 4.3.4 まず正多角形を描画し、マウスボタンを押すと、その場所から多角形の各頂点へ直線を描画するプログラムを書け。

4.4　キーボード イベント

次の例題はマウスではなく、キーボードからのイベントを扱う。メソッド名やクラス名の Mouse が Key に変わるだけで、大部分は MouseTest.java に似ている。

例題 4.4.1 (参考) キーボード
U(p), D(own) の各キーが押されると文字列が移動する。

ファイル KeyTest.java

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

public class KeyTest extends JApplet implements KeyListener {
  int x=50, y=20;

  @Override
  public void init() {
    setFocusable(true);
```

```
    addKeyListener(this);
  }

  @Override
  public void paint(Graphics g) {
    super.paint(g);
    g.drawString("HELLO␣WORLD!", x, y);
  }

  public void keyTyped(KeyEvent e) {
    int k = e.getKeyChar();
    if (k=='u') {
      y-=10;
    } else if (k=='d') {
      y+=10;
    }
    repaint();
  }
  public void keyReleased(KeyEvent e) {}
  public void keyPressed(KeyEvent e) {}
}
```

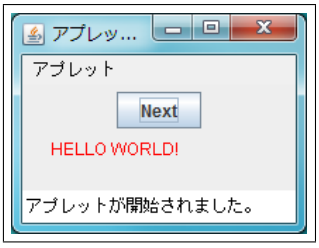
KeyListener インタフェースは keyPressed, keyReleased, keyTyped の 3 つのメソッドからなる。このうちの keyPressed が「キーが押し下げられた」とき、keyReleased が「キーが離された」とき、に対応するイベントハンドラである。実際に押されたキーに対応する文字を知るには KeyEvent クラスの getKeyCode というメソッドを用いる。keyTyped は、keyPressed, keyReleased よりも高レベルなイベントで「文字が入力された」ときに対応するイベントである。このメソッドでは、KeyEvent クラスの getKeyChar メソッドを用いて、入力された文字を知ることができる。(つまり、Shift キーを押しながら, “a” キーを押した場合は 'A' という文字が返る。)

Q 4.4.2 Bar という名前のアプレットクラス（つまり JApplet を継承するクラス）に keyTyped などいくつかのイベントハンドラを定義して、キーイベントに反応するプログラムを作成するとき、import 文と public class Bar extends JApplet に続く 2 ワードを書け。
答: public class Bar extends JApplet _____

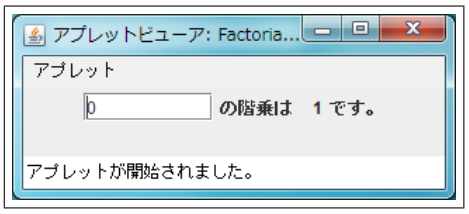
問 4.4.3 KeyTest.java を拡張して、上下・左右・斜めにも文字列を動かせるようにせよ。
　さらにカーソルキー（矢印キー）を利用する方法を調べよ。KeyTest.java を改良して、カーソルキーで文字を動かせるようにせよ。
参考: *(JDKDIR)*/docs/ja/api/java.awt.event.KeyEvent.html

4.5　GUI 部品

大抵の GUI は、ボタン、テキストフィールド、ラベル、チェックボックスなどの GUI 部品から構成されている。



ボタンの例
（ChangeColor.java）



テキストフィールドの例
（Factorial.java）

プログラムは、ボタンが押された、テキストフィールドが書き換えられた、などのイベントにも反応しなければならない。このようなイベントに対して、mouseClicked や keyPressed のような低レベルなイベントハンドラで対応するのは、不可能ではないにしても困難である。そこで GUI 部品に対するイベントには _____（空欄 4.5.1）というイベントハンドラが用意されている。

このイベントハンドラは ActionEvent 型の引数を受け取る。ActionEvent 型の引数は、イベントが発生した場所・時間に関する情報などを持っていて、この引数から、どの部品でイベントが発生したかを特定することができる。

例題 4.5.1 ボタンを押すとテキストの色が変わる。

ファイル ChangeColor.java

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

public class ChangeColor extends JApplet
                         implements ActionListener {
  Color[] cs = {Color.RED, Color.BLUE, Color.GREEN, Color.ORANGE};
  int i=0;

  @Override
  public void init() {
    JButton b = new JButton("Next");
    b.addActionListener(this); /* 1 */
    setLayout(new FlowLayout()); /* 2 */
    add(b); /* 3 */
  }

  @Override
  public void paint(Graphics g) {
    super.paint(g);
    g.setColor(cs[i]);
    g.drawString("HELLO␣WORLD!", 20, 50);
```

```
  }

  public void actionPerformed(ActionEvent e) {
    i=(i+1)%cs.length;
    repaint();
  }
}
```

新しいボタンを作成するのに、JButton クラスのコンストラクタを用いる。このコンストラクタは、ボタンに表示する文字列を引数に取る。生成した部品をアプレットの画面に加えるには ____(空欄 4.5.2)というメソッドを用いる。(/* 3 */)

その直前の行(/* 2 */)では、add された部品を配置する方法を指定している。ここでは FlowLayout という単純な配置方法を選択している。

actionPerformed は _____(空欄 4.5.3)インタフェースのメソッドである。このインタフェースには、他のメソッドはない。ボタン b が押されたときに actionPerformed が呼ばれるように、ボタン b の addActionListener メソッドを読んでいることに注意する。(/* 1 */)

この例題では、GUI 部品を 1 つしか使用していないので、 actionPerformed メソッドの引数(e)は調べる必要がない。 actionPerformed が呼び出される度に、フィールド i の値が変更される。(GUI 部品を 2 つ以上使う例はあとで紹介する。)

Q 4.5.2 Baz という名前のアプレットクラス(つまり JApplet を継承するクラス)に actionPerformed イベントハンドラを定義して、ボタンなどのイベントに反応するプログラムを作成するとき、import 文と public class Baz extends JApplet に続く 2 ワードを書け。
答: public class Baz extends JApplet _____

例題 4.5.3 テキストフィールドに数字を入力して、その階乗を計算する。

ファイル Factorial.java

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

public class Factorial extends JApplet
                       implements ActionListener {
  JTextField input;
  JLabel output;

  @Override
  public void init() {
    input = new JTextField("0", 8);
    output = new JLabel("␣␣␣1");
    input.addActionListener(this);
    setLayout(new FlowLayout());
```

```
    add(input);  add(new JLabel("の階乗は"));
    add(output); add(new JLabel("です。"));
  }

  static int factorial(int n) {  // factorial は階乗という意味
    int r = 1;
    for (; n>0; n--) {
      r *= n;
    }
    return r;
  }

  public void actionPerformed(ActionEvent e) {
    try {
      int n = Integer.parseInt(input.getText());
      output.setText("␣␣"+factorial(n));
    } catch (NumberFormatException ex) {
      input.setText("数値!");
    }
  }
}
```

JTextField のコンストラクタは、最初に表示する文字列（String 型）と、表示できる文字数（int 型）の 2 つの引数を取る。 JLabel は単に文字を表示するためだけの GUI 部品である。

ユーザーがテキストフィールドに文字を書き込み、リターンキーを押した時点でイベントが発生する。テキストフィールドの場合もボタンと同じく actionPerformed メソッドで処理する。入力された文字列は actionPerformed の中で input（JTextField クラス）の getText メソッドを使って知ることができる。

このあと output（JLabel クラス）の setText というメソッドを呼び出して、ラベルに表示されている文字列を変更している。

(空欄 4.5.4)は文字列から整数に変換するためのメソッド（java.lang.Integer クラスのクラスメソッド）である。

> java.lang.Integer クラス:
>
> public static int parseInt(String s)
>
> 文字列の引数を符号付き 10 進数の整数型として構文解析した結果生成された整数値が返される。

階乗の計算はクラスメソッド factorial として独立させた。このメソッドは戻り値を持つが、return 文の書き方も C 言語と同じである。

戻り値型 メソッド名（引数の型 引数名, ... ） {
  ...
}

というメソッドの定義の書き方も（戻り値型のまえに public などの修飾子がつくことがあることを除けば）C 言語の関数の書き方と同じである。

Q 4.5.4 str という String 型の変数に"123"のような文字列が入っているとき、これを 123 という int 型に変換する Java の式を書け。
答: _____

4.6 Java の例外処理

try〜catch〜文は

try ブロック$_1$ **catch** (例外型 変数) ブロック$_2$

という形で用いる。

この形は、まずブロック$_1$を実行する。ブロック$_1$の中で例外（エラーと考えて良い）と呼ばれる状況が起こったとき、catch の後ろの例外型がその例外の型と一致するか調べ、一致すればその後のブロック$_2$を実行する。一致するものがなければ、現在実行しているメソッドを呼び出した式を囲んでいる try〜catch 文を探す。それもなければ、さらにメソッド呼出しの履歴をさかのぼって、囲んでいる try〜catch 文を探す。それでもなければプログラムを終了する。

“**catch** (例外型 変数) ブロック” という形（catch 節）が複数続いても良い。その場合は最初に発生した例外にマッチする例外型を持つ catch 節が選択される。変数に初期値として、例外の情報をもつオブジェクトが渡されてブロックが実行される。

また、最後に “**finally** ブロック” という形（finally 節）がつく場合もある。その場合、finally ブロックは例外が起こったか否か、さらに例外が catch されたか否かにかかわらず、必ず実行される。

例えば、0 による除算を行なうと ArithmeticException という種類の例外が発生する。次のようなプログラムを実行すると、

ファイル TryCatchTest.java

```
public class TryCatchTest {
  public static void main(String[] args) {
    int i;
    for (i=-3; i<=3; i++) {
      try {
        System.out.printf("10/%d␣=␣%d%n", i, 10/i);
      } catch (ArithmeticException e) {
        System.out.println("エラー:␣"+e.toString());
      }
    }
  }
}
```

出力は次のようになる。

```
10/-3 = -3
10/-2 = -5
10/-1 = -10
エラー: java.lang.ArithmeticException: / by zero
```

```
10/1 = 10
10/2 = 5
10/3 = 3
```

i が 0 になった地点で例外が発生し、catch の後のブロックが実行される。その後は、try〜catch 文の次の文の実行を継続する。

Q 4.6.1 次のプログラムの出力を予想せよ。

```
public class TryCatchTest {
  public static void main(String[] args) {
    int i;
    try {
      for (i=-3; i<=3; i++) {
        System.out.printf("10/%d␣=␣%d%n", i, 10/i);
      }
    } catch (ArithmeticException e) {
      System.out.println("エラー:␣"+e.toString());
    }
  }
}
```

答:

_____

_____

_____

_____

_____

_____

_____

ArithmeticException の他に、よく扱う必要のある例外としては次のようなものがある。いずれも java.lang パッケージに属する。java.lang パッケージは標準的なクラスを集めた、import の必要がない（最初から import されている）パッケージである。

NullPointerException	null が通常のオブジェクトとしてアクセスされた
NumberFormatException	Integer.parseInt などで文字列が数値として解釈できない
ArrayIndexOutOfBoundsException	範囲外の添字で配列がアクセスされた

null は、_____（空欄 4.6.1）として使われる定数である。（C 言語の NULL に対応する。）

例外の中には、発生する可能性があるときは try〜catch で囲んで処理する必要があるものもある。入出力に関する例外の java.io.IOException などである。

問 4.6.2 N_gon.java の正多角形の辺の数をテキストフィールドから入力できるようにせよ。

4.7 throw 文

プログラムにより例外を発生させるには throw 文を用いる。

throw 式;

この “式” は例外型（Exception あるいはそのサブクラス）のオブジェクトでなければならない。

次の例は、コマンドライン引数（main メソッドの引数の文字列の配列 args）として渡された数字の積を計算するプログラムである。途中で 0 が出てきた場合は、わざと例外を発生させて、残りのかけ算の処理を行なわないようにしている。（ただしこのプログラム例では、break 文を用いる方が自然である。）

ファイル TryCatchTest2.java

```
public class TryCatchTest2 {
  public static void main(String[] args) {
    int i, m=1;
    try {
      for (i=0; i<args.length; i++) {
        m *= foo(args[i]);
      }
    }  catch (Exception e) {
      m = 0;
    }
    System.out.println("答は␣" + m + "です。");
  }

  public static int foo(String arg) throws Exception {
    int a = Integer.parseInt(arg);
    if (a==0) throw new Exception("zero");
    return a;
  }
}
```

例えば “java TryCatchTest2 1 2 0 3 4 5 6” というコマンドライン引数で実行させると、3 番目の引数の 0 を呼んだ時点で、例外を発生させるため、残りの引数の 3, 4, 5, 6 は無視される。

上の foo メソッドのように本来 try〜catch で処理する必要のある例外を、メソッド内で処理しないメソッドは、throws というキーワードのあとに発生する可能性のある例外をコンマ（,）で区切って列挙しなければならない。

4.8 String クラスの split メソッド

例題 4.8.1 文字列の分割

数値の配列のデータを空白区切りの文字列で渡せるように、Graph.java を拡張する。

ファイル Graph2.java

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

public class Graph2 extends JApplet implements ActionListener {
  int[] is = {};
  JTextField input;
  Color[] cs = {Color.RED, Color.BLUE};
  int scale = 15;

  @Override
  public void init() {
    input = new JTextField("", 16);
    input.addActionListener(this);
    setLayout(new FlowLayout());
    add(input);
  }

  @Override
  public void paint(Graphics g) {
    super.paint(g);
    int i;
    int n = is.length;

    for (i=0; i<n; i++) {
      g.setColor(cs[i%cs.length]);
      g.fillRect(0, i*scale+30, is[i]*scale, scale);
    }
  }

  public void actionPerformed(ActionEvent e) {
    String[] args = input.getText().split("␣");
    int n = args.length;
    is = new int[n];

    int i;
    for(i=0; i<n; i++) {
      is[i] = Integer.parseInt(args[i]);
    }
    repaint();
  }
}
```

ここでは、空白区切りの文字列を文字列の配列に分割するために String クラスの＿＿＿＿（空欄 4.8.1）メソッドを用いた。

java.lang.String クラス:
public String[] split(String regex)
この文字列を、指定された正規表現 (regex) に一致する位置で分割する。

さらに Integer.parseInt メソッドで文字列から整数へ変換している。上の init メソッドの中身は、空白で区切られた文字列を配列に変換する典型的な方法である。split メソッドのの引数は、区切りに使用する文字列を表す正規表現である。これを","に変更すると、コンマで区切られた文字列を分割することができる。 また、_____(空欄 4.8.2)にすると、空白文字が 2 つ以上連続したり、タブ文字などが混ざったりという場合にも対応できる。Java で使用できる正規表現については、java.util.regex.Pattern クラスのドキュメント（*(JDKDIR)*/docs/ja/api/java/util/regex/Pattern.html）を参照すること。

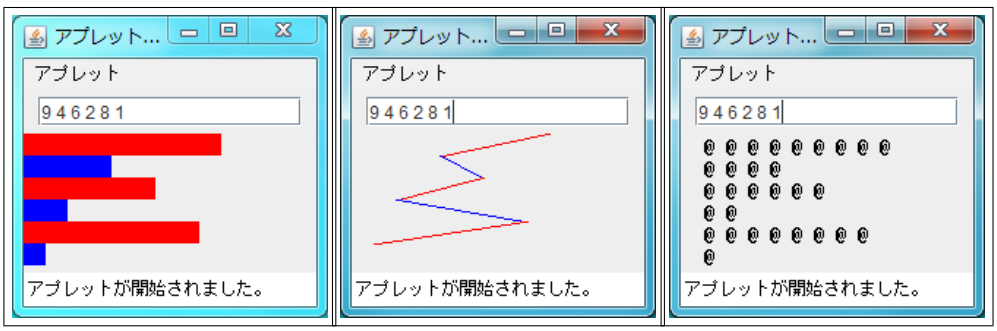
Q 4.8.2 str という String 型の変数に"087-864-2000"という文字列が入っているとき、これを'-' 区切りで、String 型の配列に分割する Java の式を書け。
答: _____

4.9 配列の生成

new オペレータは配列を生成するときにも使用することができる。_____(空欄 4.9.1)は、動的に長さ n の （int 型の）配列を生成する式である。int の代わりに他の型名を使うとその型の配列が生成される。C の配列宣言とは異なり、要素数 n の値がコンパイル時に定まっている必要はない。この形式を使うと配列の各要素は 0（オブジェクト型の場合は null）に初期化される。

一方、_____(空欄 4.9.2)は、要素数を指定するのではなく、初期値を列挙して（int 型の）配列を生成する式である。

問 4.9.1 テキストフィールドに与えられた数値データから折れ線グラフを生成するアプレットを書け。（例外 ArrayIndexOutOfBoundsException が出ないように注意すること。n 個の点を結ぶ線は n-1 本であることに注意する。）



棒グラフ 折れ線グラフ 文字によるグラフ

例題 4.9.2 時間のデータを “9:45 12:35 4:42” というように、空白で区切って渡し、その時間の合計を表示する。

ファイル AddTime2.java

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

public class AddTime2 extends JApplet
                      implements ActionListener {
  JTextField input; // e.g.  2:45 1:25 3:34 2:47 0:24
  JLabel output;

  @Override
  public void init() {
    input = new JTextField("", 16);
    output = new JLabel("00:00");
    input.addActionListener(this);
    setLayout(new FlowLayout());
    add(input);
    add(new JLabel("の和は"));
    add(output);
    add(new JLabel("です。"));
  }

  // 時間の足し算を関数として定義する。
  static int[] addTime(int[] t1, int[] t2) {
    // 時間を大きさ 2 の配列で表す。
    int[] t3 = { t1[0]+t2[0], t1[1]+t2[1] };

    if(t3[1]>=60) { // 繰り上がりの処理
      t3[0]++;
      t3[1]-=60;
    }

    return t3;      // 新しい配列を返す。
  }

  public void actionPerformed(ActionEvent e) {
    String[] args = input.getText().split("\\s+");

    int[] t = { 0, 0 };
    for (String s : args) {
      String[] stime = s.split(":");

      t = addTime(t,  new int[] { Integer.parseInt(stime[0]),
                                  Integer.parseInt(stime[1]) });
      // addTime の呼出し前にその引数に入っていた配列は不要となる。
      // あとで GC される。
```

```
    }
    output.setText(String.format("%02d:%02d", t[0], t[1]));
  }
}
```

addTime はその中で配列を確保して戻り値に用いている。このように new は、C 言語の malloc に近い働きをする。また、このようにして確保された配列は、init の中で addTime を呼ぶときに次々と捨てられるが、これは＿＿＿＿＿＿（空欄 4.9.3）（Garbage Collection, GC）によって自動的に回収される。（C 言語のように free による明示的なメモリの解放は必要ない。） GC のある言語ではこのように次々と新しいデータを生成して、古いデータを捨てるというスタイルが可能になる。

4.10 総称クラスの使用

総称クラス（generic class）は、型パラメータを持つクラスのことで、JDK5.0 から導入された。代表的な総称クラスの例として ArrayList, HashMap, LinkedList などがあげられる。型パラメータは＿＿＿＿＿＿（空欄 4.10.1）に書かれる。

ArrayList は＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿（空欄 4.10.2）である。ArrayList の型パラメータは要素の型を表す。（総称クラスはこのようにコレクション（データの集まり）の型に使われることが多い。）例えば、String 型を要素とする ArrayList は ArrayList<String>となり、次のように使用する。

```
// 空の ArrayList 作成
ArrayList<String> arr1 = new ArrayList<String>();
// データ追加
arr1.add("aaa");  arr1.add("bbb");  arr1.add("ccc");
// データ取出し
String s = arr1.get(1);
```

add メソッドでデータを追加し、get メソッドでデータを取り出すことができる。

Q 4.10.1 Color 型の要素を持つ ArrayList の colors という名前の変数を空の ArrayList に初期化する宣言を書け。
答: ＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

int, double のようなプリミティブ型は総称クラスの型パラメータになることができないという制限があるので注意が必要である。このときは Integer, Double などの対応する＿＿＿＿＿＿＿＿（空欄 4.10.3）と呼ばれるクラスを利用する。Java の主なプリミティブ型とラッパークラスとの対応を以下に挙げる。

プリミティブ型	ラッパークラス
int	Integer
char	Character
double	Double
boolean	Boolean

（ここに挙げている以外のプリミティブ型に対応するラッパークラスは単にプリミティブ型の先頭の文字を大文字にすれば良い。）

ラッパークラスとプリミティブ型の変換はほとんどの場合、自動的に行われる（オートボクシング）ので、int の代りに Integer と書く以外は通常のクラス型をパラメータとするときと変わらない。例えば次のように書くことができる。

```
// 空の ArrayList 作成
ArrayList<Integer> arr2 = new ArrayList<Integer>();
// データ追加
arr2.add(123);  arr2.add(456);  arr2.add(789);
// データ取出し
int i = arr2.get(1);
```

ArrayList<String>に int 型の要素を add したり、ArrayList<Integer>から String 型の要素を get したりするのは、当然型エラー（コンパイル時のエラー）になる。

```
ArrayList<String> arr1 = new ArrayList<String> ();
arr1.add(333);          // 型エラー

ArrayList<Integer> arr2 = new ArrayList<Integer> ();
...
String t = arr2.get(2);  // 型エラー
```

このような型エラーをコンパイル時にちゃんと発見したい、というのが、総称クラスの導入のそもそもの動機である。

API ドキュメントの中では、型パラメーターは E のような仮のクラス名が使われ、

java.util.ArrayList<E>クラス:

```
public boolean add(E e)
```

リストの最後に、指定された要素（e）を追加する。

```
public E get(int index)
```

リスト内の指定された位置（index）にある要素を返す。

のように書かれる。

Q 4.10.2 double 型を保存するための ArrayList の ds という名前の変数を空の ArrayList に初期化する宣言を書け。
答: _____

例題 4.10.3 マウスクリックの位置を保存する

描画データの一時保存に ArrayList を使用する例である。mouseClicked メソッドでクリックされた座標を保存し、paint メソッドでそれを利用している。なお、配列型も総称クラスの型パラメータとして問題なく使用することができる。この例の場合、クリックされる回数が前もってわからないので、配列ではなく ArrayList を使用している。
ファイル MouseDraw.java

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

public class MouseDraw extends JApplet implements MouseListener {
  ArrayList<int[]> points;
  @Override
  public void init() {
    points = new ArrayList<int[]>();
    addMouseListener(this);
  }

  public void mouseClicked(MouseEvent e) {
    points.add(new int[] { e.getX(), e.getY() });
    repaint();
  }

  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {}
  public void mouseReleased(MouseEvent e) {}

  @Override
  public void paint(Graphics g) {
    super.paint(g);

    int i, n = points.size();
    for (i=1; i<n; i++) {
      int[] p0 = points.get(i-1);
      int[] p1 = points.get(i);
      g.drawLine(p0[0], p0[1], p1[0], p1[1]);
    }
  }
}
```

例題 4.10.4 色の名前

HashMap は＿＿＿＿＿＿(空欄 4.10.4)と呼ばれるデータ構造である。通常の配列と異なり、int 型だけではなく、任意の型（String 型など）をキー（添字）として、値を格納・検索することができる。HashMap の型パラメータは 2 つあり、1 つめがキーの型、2 つめが値の型である。下の例では、HashMap<String, Color>、つまりキーが String 型で値が Color 型の連想配列を用いている。値の格納には put メソッド、検索には get メソッドを用いる。

```
java.util.HashMap<K,V>クラス:
  public V put(K key, V value)
```

指定された値（value）と指定されたキー（key）をこのマップに関連付ける

public V get(Object key)

指定されたキー（key）がマップされている値を返す。

Object（java.lang.Object）クラスは Java のすべてのクラスのスーパークラスとなる、クラス階層のルートクラスである。

ファイル ColorName.java

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

public class ColorName extends JApplet implements ActionListener {
  HashMap<String, Color> hm;
  JTextField input;

  @Override
  public void init() {
    // http://www.colordic.org/w/ より抜粋
    hm = new HashMap<String, Color>();
    hm.put("鴇", new Color(0xf7acbc));
    hm.put("赤", new Color(0xed1941));
    hm.put("朱", new Color(0xf26522));
    hm.put("桃", new Color(0xf58f98));
    hm.put("緋", new Color(0xaa2116)); // 以下、割愛
    input = new JTextField("紅白", 8);
    input.addActionListener(this);
    setLayout(new FlowLayout());
    add(input);
  }

  @Override
  public void paint(Graphics g) {
    String text = input.getText();
    super.paint(g);
    g.setFont(new Font("SansSerif", Font.BOLD, 64));
    int i;
    for (i=0; i<text.length(); i++) {
      String c = text.substring(i, i+1);
      Color color = hm.get(c);
      if (color==null) {
        color = Color.BLACK;
      }
      g.setColor(color);
      g.drawString(c, 64*i, 100);
    }
  }
```

```
  public void actionPerformed(ActionEvent e) {
    repaint();
  }
}
```

問 4.10.5 総称クラス LinkedList の使用法を調べ、プログラムを作成せよ。

4.11 複数の GUI 部品を使用したプログラム例

例題 4.11.1 ボタン 2 つを使ってテキストを左右に移動する。

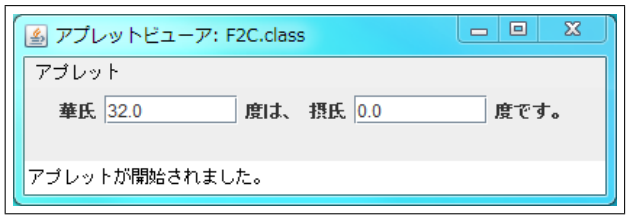
ファイル UpDownButton.java

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

public class UpDownButton extends JApplet implements ActionListener {
  int x=20;
  JButton lBtn, rBtn;

  @Override
  public void init() {
    lBtn = new JButton("Left");
    rBtn = new JButton("Right");
    lBtn.addActionListener(this);
    rBtn.addActionListener(this);
    setLayout(new FlowLayout());
    add(lBtn); add(rBtn);
  }

  @Override
  public void paint(Graphics g) {
    super.paint(g);
    g.drawString("HELLO␣WORLD!", x, 55);
  }

  public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    if (source == lBtn) {         // lBtnが押された
      x-=10;
    } else if (source == rBtn) {  // rBtnが押された
      x+=10;
    }
    repaint();
  }
}
```

このプログラムでは GUI 部品（ボタン）を 2 つ使用しているので、どのボタンが押されたかを actionPerformed メソッド中で調べる必要がある。そのために ActionEvent クラスの getSource というメソッドを用いて、比較演算子（==）で比べることによって、イベントの起こったボタンを特定している。

問 4.11.2 摂氏の温度をテキストフィールドに入力して、これを華氏の温度に変換するアプレットを Factorial.java（**例題**4.5.3）にならって書け。（ただしボタンは使わない。）

さらに、2 つのテキストフィールドを用いて、摂氏と華氏の変換を双方向に行なえる（片方のテキストフィールドの値を変えると、もう片方のテキストフィールドの値が変わる）ようにせよ。



（参考）（華氏の温度）=（摂氏の温度）× 1.8 + 32
例えば摂氏 0 度は華氏 32 度、摂氏 100 度は華氏 212 度になる。
（参考） String 型を double 型（実数の型）に変換するには、＿＿＿＿＿＿＿＿＿＿（空欄 4.11.1）という java.lang.Double クラスのクラスメソッドを使う。

java.lang.Double クラス:
```
public static double parseDouble(String s)
```
指定された String が表す値に初期化された新しい double 値を返す。

Integer.parseInt と使い方が似ているが、double 型を返す。

また逆に、double 型を String 型に変換するときに、書式を指定したい（例えば 小数点以下を 3 桁以内に抑えたい）ときは、＿＿＿＿＿＿＿＿＿＿（空欄 4.11.2）というクラスメソッドを使う。

java.lang.String クラス:
```
public static String format(String format,
                            Object... args)
```
指定された書式の文字列と引数を使って、書式を整えた文字列を返す。

引数の意味は PrintWrite クラスの printf メソッド（System.out.printf など）と同じだが、標準出力に出力するのではなく戻り値として文字列を返す。

Q 4.11.3 str という String 型の変数に"3.14"という文字列が入っているとき、これを 3.14 という double 型に変換する Java の式を書け。
答: ＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

Q 4.11.4 x という double 型の変数に 1.0/3 という式の結果が入っているとき、これを "0.333"という小数第 3 位までの String 型に変換する Java の式を書け。
答: _____

4.12 内部クラス

一方、GUI 部品が多くなってきたときは、**if**〜**else** 文が何重も入れ子になってしまう getSource メソッドではなく、次の例のように内部クラス（inner class）を用いる方が効率が良い。

例題 4.12.1 UpDownButton.java を内部クラスを用いて書き換える

ファイル UpDownButton2.java

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

public class UpDownButton2 extends JApplet {
  int x=20;

  public class LeftListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      x-=10;
      repaint();
    }
  }

  public class RightListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      x+=10;
      repaint();
    }
  }

  @Override
  public void init() {
    JButton lBtn = new JButton("Left");
    JButton rBtn = new JButton("Right");
    lBtn.addActionListener(new LeftListener());
    rBtn.addActionListener(new RightListener());
    setLayout(new FlowLayout());
    add(lBtn); add(rBtn);
  }

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

```
    g.drawString("HELLO␣WORLD!", x, 55);
  }
}
```

Javaではクラスの中にクラスを定義することができる。(＿＿＿＿＿＿(空欄 4.12.1)) これも関数の中に関数を定義できない C との大きな違いである。上の例は、この内部クラス（LeftListener と RightListener）を用いて、actionPerformed メソッドを与えている。内部クラスの中では、その外側のクラスのフィールド（上の例の場合 x）やメソッドなど（上の例の場合 repaint メソッド）を参照することができる。

このように内部クラスを用いると、addActionListner のときに、コンポーネントとメソッドを関連づけることができるので、コンポーネントの数が多いときは getSource メソッドを用いるよりも効率が良い。

4.13　匿名クラス

内部クラスに名前をつけずに(＿＿＿＿＿＿(空欄 4.13.1), ＿＿＿＿＿＿(空欄 4.13.2)) 定義することができる。名前のないクラスのオブジェクトは次のようにして作成する。

```
new スーパークラスのコンストラクタ (引数) {
  メソッド・フィールドの定義
}
```

スーパークラスのコンストラクタは ActionListener のようなインタフェース名でも良い。その場合は下の例のように引数はとらず、() のみを書く。また、その場合のスーパークラスは java.lang.Object となる。

例題 4.13.1 UpDownButton.java を匿名クラス（anonymous class）を用いて書き換える、

ファイル UpDownButton3.java

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

public class UpDownButton3 extends JApplet {
  int x=20;

  @Override
  public void init() {
    JButton lBtn = new JButton("Left");
    JButton rBtn = new JButton("Right");
    lBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
```

```
        x-=10;
        repaint();
      }
    });
    rBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        x+=10;
        repaint();
      }
    });
    setLayout(new FlowLayout());
    add(lBtn); add(rBtn);
  }

  @Override
  public void paint(Graphics g) {
    super.paint(g);
    g.drawString("HELLO␣WORLD!", x, 55);
  }
}
```

4.14 final 修飾子

内部クラス（匿名クラスを含む）はメソッドの中で定義することも可能で、その場合はメソッドの局所変数を参照することもできるが、少し制限がある。

実装上の都合で、内部クラスを生成するとき、参照されているメソッドの局所変数についてはコピーを作る必要がある。このとき局所変数の値が代入によって変更されてしまうと、内部クラス内の変数のコピーは値が変わらず、意味的に変なことになってしまう。

このため、内部クラスから参照される局所変数は、代入によって値を変更してはいけないこと、これを保証するため final という修飾子をつけなくてはいけないことになっている。（なお、フィールドを読み出す場合にはこのような制限は存在しない。）

例題 4.14.1 匿名クラスからメソッドの局所変数を参照する

ファイル FinalExample.java

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

public class FinalExample extends JApplet {
  static final Color[] colors = {Color.RED, Color.GREEN, Color.BLUE};
  int c = 0;

  @Override
```

```
  public void init() {
    final JButton button  = new JButton("Push");

    button.setForeground(colors[c]);
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        c = (c+1) % colors.length;
        button.setForeground(colors[c]);
      }
    });
    setLayout(new FlowLayout());
    add(button);
  }
}
```

この例では button という局所変数が匿名クラスから参照されているため final と宣言されている。

なお、内部クラスから参照されるという理由以外にも final と宣言することがある。代入によって値が変わることがないことが保証されるので、意味が追いやすくなるし、効率上有利になることもある。

上の例では colors はクラスフィールドなので、final と宣言しなくても、内部クラスから参照することはできる。しかし、代入しないことがわかっているので final と宣言している。

Q 4.14.2 Factorial.java（**例題** 4.5.3）を匿名クラスを用いて書き換える。空欄を埋めて、プログラムを完成させよ.

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

public class Factorial                    {
  @Override
  public void init() {
    ______ JTextField input = new JTextField("0", 8);
    ______ JLabel output = new JLabel("␣␣␣1");
    input.addActionListener(                          ) {
      public void actionPerformed(ActionEvent e) {
        // actionPerformed の中身は同じなので省略
      }
    });
    setLayout(new FlowLayout());
    add(input);  add(new JLabel("の階乗は"));
    add(output); add(new JLabel("です。"));
  }

  // factorial の定義は同じなので省略
}
```

問 4.14.3 MouseTest.java, KeyTest.java を匿名クラスを用いて書き換えよ。

問 4.14.4 JTextArea, JPanel, JCheckBox, JComboBox, JList, JTable, JTree など、他の GUI 部品の使用法を調べよ。またこれらのクラスの部品を使ってプログラムを作れ。

問 4.14.5 これまで紹介したプログラムは、FlowLayout を用いていて、GUI 部品がどのように配置されるかについては無関心だった。部品を自分の好みの位置に配置する方法（〜Layout という名前のクラス）を調べよ。

キーワード イベント、イベントハンドラ、keyTyped メソッド, mouseClicked メソッド, actionPerformed メソッド, インタフェース（interface）, MouseListener インタフェース, KeyListener インタフェース, ActionListener インタフェース, this, MouseEvent クラス, KeyEvent クラス, ActionEvent クラス, add メソッド, JButton クラス, JLabel クラス, JTextField クラス, Integer.parseInt メソッド, split メソッド, 総称クラス, ArrayList クラス, HashMap クラス, LinkedList クラス内部クラス、匿名クラス、