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インタフェース（GUI）を持つプログラムは、ユーザ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 _____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インタフェースに付け加える方法についても学ぶ。

4.1　イベント処理

ユーザ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、_____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); /* 6 */
    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.*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 _____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。)
- _____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 */)
- このクラスの paint メソッドは、その中で super.paint(g) を呼び出している。 super.〜はスーパークラスで定義されているメソッドを呼び出すための書き方である。このプログラムの場合、背景を再描画している。(/* 6 */)

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 メソッドを呼び出す、

4.2 this

this は、メソッドを実行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(*args1, args2, ...*)

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）というのは、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、_____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 _____ 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。

問 4.3.1 *repaint()* がなければ、*MouseTest.java* はどのような振舞いをするか?

問 4.3.2 *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。

問 4.3.3 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。

問 4.3.4 マウスを押した場所を順に結んで、折れ線を描画するプログラムを書け。
アプレットを最小化したり、他のウインドウで隠したりしても、再描画のときに折れ線もちゃんと再描画されるようにせよ。また、総称クラスの *java.util.ArrayList* を使用して、頂点がいくつに増えても対応できるようにすること。

プログラムを簡単にするために、最初の点を *(100,100)* などに固定しても良い。

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

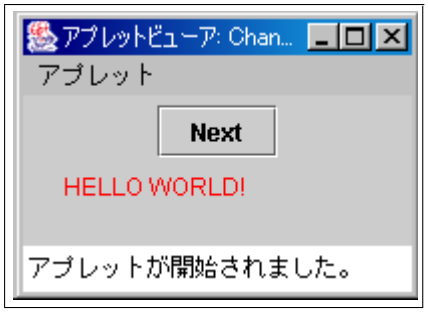
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

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

  @Override
  public void init() {
    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) {}
}
```

KeyListner 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。)

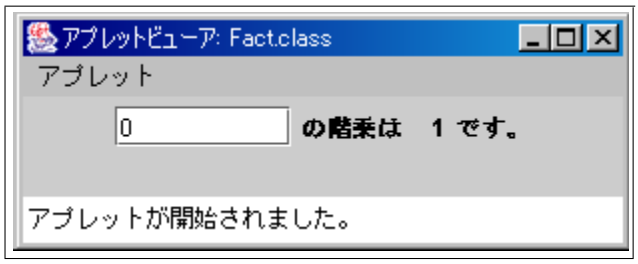
問 4.4.2 *KeyTest.java* を拡張して、上下・左右・斜めにも文字列を動かせるようにせよ。
さらにカーソルキーを利用する方法を調べよ。*KeyTest.java* を改良して、カーソルキーで文字を動かせるようにせよ。
参考: (JDKDIR)/docs/ja/api/java.awt.event.KeyEvent.html

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 _____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 _____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 ____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 ＿＿＿＿＿＿＿＿＿＿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。）

例題 4.5.2 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) {
    int n = Integer.parseInt(input.getText());
    output.setText("  "+factorial(n));
  }
}
```

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。

4.6 複数の GUI 部品を使用したプログラム例

例題 4.6.1 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 left, right;

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

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

  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == left) {          // Leftが押された
      x-=10;
    } else if (e.getSource() == right) { // Right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 4.6.2 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* 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）$= \dfrac{(\text{摂氏の温度}) \times 9}{5} + 32$

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、＿＿＿＿＿＿＿＿＿＿＿＿というクラスメソッドを使う。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、＿＿＿＿＿＿＿＿というクラスメソッドを使う。

4.7　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（innner class）を用いる方が効率が良い。

例題 4.7.1 *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 left  = new JButton("Left");
    JButton right = new JButton("Right");
    left.addActionListener(new LeftListener());
    right.addActionListener(new RightListener());
    setLayout(new FlowLayout());
    add(left); add(right);
  }

  @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。（＿＿＿＿＿＿＿＿）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メンバ（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 を用いるよりも効率が良い。

4.8 匿名クラス

内部クラスに名前をつけずに（＿＿＿＿＿＿, ＿＿＿＿＿＿）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 スーパークラス名 (引数) {
  メソッド・フィールドの定義
}
```

スーパークラス名のところは 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ない。（その場合、スーパークラスは java.lang.Object となる。）

例題 4.8.1 *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 left  = new JButton("Left");
    JButton right = new JButton("Right");
    left.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        x-=10;
        repaint();
      }
    });
    right.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        x+=10;
        repaint();
      }
    });
    setLayout(new FlowLayout());
    add(left); add(right);
  }

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

4.9 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、フィールドを参照する場合にはこのような制限は存在しない。）

例題 4.9.1 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。

4.10 JTextAreaクラス

例題 4.10.1 *JTextArea* — 約数の表示
整数を入力してもらって、その約数をすべて表示する。

ファイル Divisor.java

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

public class Divisor extends JApplet implements ActionListener {
  JTextField input;
  JTextArea output;

  @Override
  public void init() {
    input = new JTextField("0", 8);
    output = new JTextArea(10, 20);
    input.addActionListener(this);
    setLayout(new FlowLayout());
    add(input); add(output);
  }

  public void actionPerformed(ActionEvent e) {
    int i, n = Integer.parseInt(input.getText());

    for(i=1; i<=n; i++) {
      if (n%i==0) {
        output.append(i+"は "+n+"の約数です。¥n");
      }
    }
    output.append("以上¥n¥n");
  }
}
```

このアプレットのように多くのメッセージを表示する場合には _____という部品が便利である。

JTextAreaのコンストラクタの引数は、テキストエリアの _____と _____である。テキストエリアに文字列を表示するには、_____というメソッドを用いることができる。

問 4.10.2 *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 4.10.3 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, 内部クラス、匿名クラス、JTextArea クラス

メモ: