第4章 イベント処理と GUI 部品

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、＿＿＿＿＿＿＿＿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

```
 1 import javax.swing.*;
 2 import java.awt.*;
 3 import java.awt.event.*;    /* 1 */
 4
 5 class MouseTest extends JPanel
 6                  implements MouseListener /* 2 */ {
 7     private int x = 50, y = 20;
 8
 9     public MouseTest() {
10         setPreferredSize(new Dimension(150, 150));
11         addMouseListener(this);    /* 3 */
12     }
13
14     /* 4 */
15     public void mouseClicked(MouseEvent e) {
16         x = e.getX();
17         y = e.getY();
18         repaint();
19         return;
20     }
21
22     public void mousePressed(MouseEvent e) {}  /* 5 */
23     public void mouseReleased(MouseEvent e) {} /* 5 */
24     public void mouseEntered(MouseEvent e) {}  /* 5 */
25     public void mouseExited(MouseEvent e) {}   /* 5 */
26
27     @Override
28     public void paintComponent(Graphics g) {
29         super.paintComponent(g);
```

```
30          g.drawString("HELLO WORLD!", x, y);
31      }
32  }
33
34
35  void main() {/* 省略 */}
```

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。

なお、Java はクリックの判定が何故か厳しくて、押すときと離すときに少しでも マウスカーソルが移動しているとクリックと判定してくれない。

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＿＿＿＿＿＿＿＿を implement していることを宣言している。（`/* 2 */`）

- コンストラクターで、`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ーを定義している。（`/* 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、いちおう定義している。（`/* 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、`JPanel` クラスで定義済のメソッドで、その中で間接的に `paintComponent` メソッドを呼び出している。

自分で、呼出すコードを書かなくても、`paintComponent` メソッドが利用されている。オブジェクト指向言語に特有の振舞いである。

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、`JPanel` 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。

`paintComponent` メソッドと `mouseClicked` 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（ドット）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。

つまり、Java ではメソッドは基本的に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）が、他の引数と異なり名前がついていない。 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ものである。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、 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。（`extends` とは異なり） `implements` のあとには複数個のインタフェースを書くことができる。この場合、インタフェースをコンマ `,` で区切って書く。（あとで例を示す。）

`MouseListener` インタフェースの定義 （ただし一部簡略化）

```java
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。だから `MouseTest` クラスが `MouseListener` を実装していなければ、コンストラクターの中の `addMouseListener(this)` はエラーになる。

Q 4.3.1 `Foo` という名前の `JPanel` 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ーを定義して、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 `class Foo extends JPanel` に続く 2 ワードを書け。

答: `class Foo extends JPanel` _____

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 円を描画し、円の内部をマウスでクリックすると「あたり」、円の外部をクリックすると「はずれ」という文字列を描画するプログラムを書け。

問 4.3.4 § 3.3 の問 3.3.6 で作成した 2 次元の色のグラデーションを作成する GUI アプリケーションを改良して、マウスをクリックした位置の色で何かの文字列を描画する GUI アプリケーション（カラーピッカー）を作成せよ。

また、カラーピッカーを利用して、16 × 16 などのサイズのドット絵を作成するプログラムを書け。

問 4.3.5 § 3.4 の `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。

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` に変わり、Enter と Exit がないだけで、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

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

class KeyTest extends JPanel
              implements KeyListener {
    private int x = 50, y = 20;

    public KeyTest() {
        setPreferredSize(new Dimension(150, 150));
        setFocusable(true);
        addKeyListener(this);
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(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;
        }
        // System.err.printf("key = %d%n", k);
        repaint();
    }
    public void keyReleased(KeyEvent e) {}
    public void keyPressed(KeyEvent e) {}
}

void main() {/* 省略 */}
```

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ーである。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 JPanel 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ーを定義して、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 class Bar extends JPanel に続く 2 ワードを書け。

答: class Bar extends JPanel _____

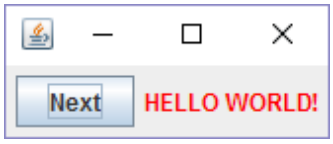
問 4.4.3 KeyTest.java を拡張して、上下に加えて左右にも文字列を動かせるようにせよ。

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。（**ヒント:** keyTyped ではなく、keyPressed または KeyReleased メソッドを使う必要がある。）

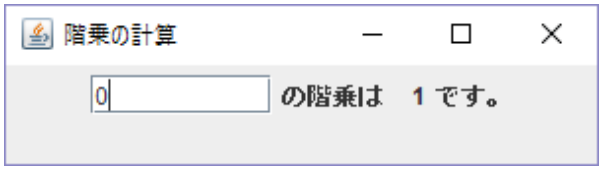
参考:
https://docs.oracle.com/javase/jp/25/docs/api/java.desktop/java/awt/event/KeyEvent.html

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ーで対応するのは、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ーが用意されている。

このイベントハンドラーは 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

```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4
5 class ChangeColor extends JPanel
6                    implements ActionListener {
7     private final Color[] cs = {
8       Color.RED, Color.BLUE, Color.GREEN, Color.ORANGE
9     };
```

```
    private int i = 0;
    private JLabel label;

    public ChangeColor() {
        JButton b = new JButton("Next");
        b.addActionListener(this);          /* 1 */
        label = new JLabel("HELLO WORLD!");
        label.setForeground(cs[i]);/* 前景色の変更 */
        setLayout(new FlowLayout());        /* 2 */
        add(b); add(label);                 /* 3 */
    }

    public void actionPerformed(ActionEvent e) {
        i = (i + 1) % cs.length;
        label.setForeground(cs[i]);
    }
}

void main() {/* 省略 */}
```

新しいボタンを作成するのに、JButton クラス のコンストラクターを用いる。このコンストラクターは、ボタンに表示する文字列を引数に取る。JLabel は単に文字を表示するためだけの GUI 部品である。生成した部品を GUI アプリケーション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。ちなみに JPanel の場合は、実はデフォルトのレイアウトが FlowLayout なので、この行はなくても構わない。

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、他のメソッドはなく、actionPerformed というただ一つのメソッドのみ宣言されている。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、イベントが発生する 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。
（イベントが発生する 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 JPanel を継承するクラスに actionPerformed イベントハンドラーを定義して、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 class Baz extends JPanel に続く 2 ワードを書け。
>
> 答: class Baz extends JPanel _____

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.*;

class Factorial extends JPanel
                implements ActionListener {
  private JTextField input;
  private JLabel output;

  public Factorial() {
    setPreferredSize(new Dimension(300, 50));
    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。"));
  }

  // factorial -- 階乗のこと
  private static int factorial(int n) {
    int r = 1;
    for (; n > 0; n--) {
      r *= n;
    }
    return r;
  }

  public void actionPerformed(ActionEvent e) {
    try {
      int n = Integer.parseInt(input.getText());
      /* ラベルの表示文字列の変更 */
      output.setText("  " + factorial(n));
    } catch (NumberFormatException ex) {
      input.setText("数値!");
    }
  }
}

void main() {/* 省略 */}
```

JTextField のコンストラクターは、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。

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。

_____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"` のような整数を表す文字列が入っているとき、これを 123 という `int` 型に変換した値を返す Java の式を書け。

答: _____

問 4.5.5 第 3 章の 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。

4.6 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。この中でエラーが起こらなければ、そのまま次へ進む。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マッチするか調べ、マッチすればその後のブロック$_2$を実行する。

`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。その場合は発生した例外にマッチする例外型を持つ、最初の `catch` 節が選択される。“変数”に例外の情報を持つオブジェクトが初期値として渡されてブロックが実行される。また、Java 7 以降では、“例外型”を“|”で区切って、複数個列挙できるようになった。

`catch （例外型1 | 例外型2 | … | 例外型n 変数）ブロック` のように `catch` 節を書くと、例外が例外型$_1$, 例外型$_2$, ..., 例外型$_n$ のいずれかにマッチするとき、後ろのブロックが実行される。マッチするものがなければ、現在実行しているメソッドを呼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。

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。

次のプログラムの断片は、「|」を使った catch 節と finally 節の使用例である。

```
    BufferedReader fin = null;
    int i;
    try {
      fin = new BufferedReader(new FileReader(f));
      i =  Integer.parseInt(fin.readLine());
    } catch (FileNotFoundException // ファイルがなければ
          | NullPointerException  // ファイルが空なら
          | NumberFormatException e) { // 数でないなら
      i = 0;  // 0 に
    } finally {
      if (fin != null) {
        fin.close();  // closeを忘れない
      }
    }
```

なお、この例のように try のなかでファイルをオープンして最終的にクローズするという形は頻繁に使われるので、try-with-resources 文という別のかたちも用意されている。 try-with-resources 文では try キーワードのあとの丸括弧の中に、変数宣言を書く。変数宣言が複数の場合は「;」で区切る。各変数宣言の右辺は通常、ファイルなどをオープンする式になっている。このような変数で示されるオブジェクトは try-with-resources 文のあとでクローズされる。

```
    int i;
    try (BufferedReader fin =
           new BufferedReader(new FileReader(f))) {
      i =  Integer.parseInt(fin.readLine());
    } catch (FileNotFoundException // ファイルがなければ
          | NullPointerException  // ファイルが空なら
          | NumberFormatException e) { // 数でないなら
      i = 0;  // 0 に
    }
```

また、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

```
1  void main() {
2    int i;
3    for (i = -3; i <= 3; i++) {
4      try {
5        System.out.printf("10 / %d = %d%n",
6            i, 10 / i);
7      } catch (ArithmeticException e) {
8        System.out.println("エラー: " + e.toString());
9      }
10   }
11   System.out.println("終");
```

```
12 }
```

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
終
```

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。

ファイル TryCatchQuiz.java

```
1  void main() {
2    int i;
3    try {
4      for (i = -3; i <= 3; i++) {
5        System.out.printf("10 / %d = %d%n",
6            i, 10 / i);
7      }
8    } catch (ArithmeticException e) {
9      System.out.println("エラー: " + e.toString());
10   }
11   System.out.println("終");
12 }
```

答:

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

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 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、＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿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（検査例外、checked exception）もある。入出力に関する例外の java.io.IOException などである。

4.7 throw 文

使用頻度は低いが、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。

次の少々わざとらしい例は、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

```
1  void main(String[] args) {
2    int i, m = 1;
3    try {
4      for (i = 0; i < args.length; i++) {
5        m *= foo(args[i]);
6      }
7    } catch (Exception e) {
8      m = 0;
9    }
10   System.out.println("答は " + m + " です。");
11 }
12
13 int foo(String arg) throws Exception {
14   int a = Integer.parseInt(arg);
15   if (a == 0)
16     throw new Exception("zero");
17   return a;
18 }
```

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.*;

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

    public Graph2() {
        setPreferredSize(new Dimension(200, 200));
        input = new JTextField("", 16);
        input.addActionListener(this);
        setLayout(new FlowLayout());
        add(input);
    }

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

        for (i = 0; i < n; i++) {
            g.setColor(cs[i % 2]);
            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();
    }
}

void main() {/* 省略 */}
```

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　　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。

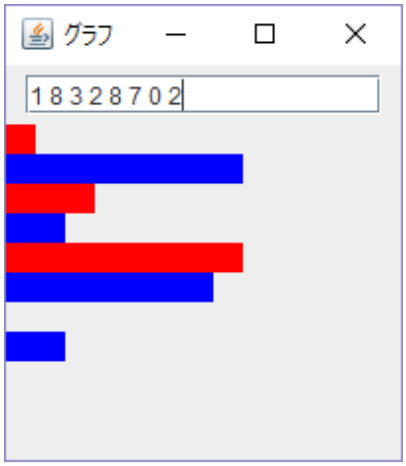
さらに Integer.parseInt メソッドで文字列から整数へ変換している。上の actionPerformed メソッドの中身は、空白で区切られた文字列を配列に変換する典型的な方法である。

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、＿＿＿＿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（https://docs.oracle.com/javase/jp/25/docs/api/java.base/java/util/regex/Pattern.html）を参照すること。

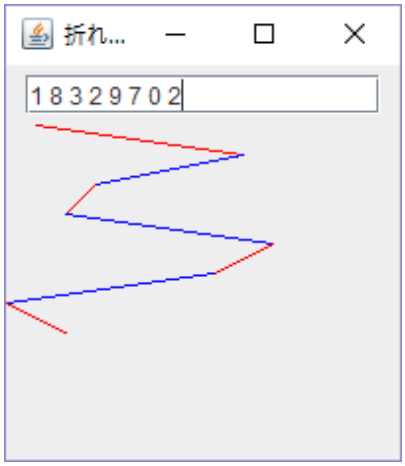
Q 4.8.2 str という String 型の変数に "087-864-2015" という文字列が入っているとき、これを '-' 区切りで分割した、String 型の配列を返す Java の式を書け。

答: ＿＿＿＿＿＿＿＿

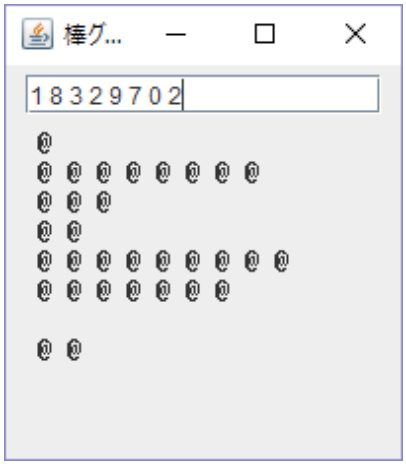
問 4.8.3 テキストフィールドに与えられた数値データから折れ線グラフを生成する GUI アプリケーションを書け。（例外 ArrayIndexOutOfBoundsException が出ないように注意すること。 n 個の点を結ぶ線は n - 1 本であることに注意する。）



棒グラフ



折れ線グラフ



文字によるグラフ

4.9 配列の生成

new オペレーターは配列を生成するときにも使用することができる。`new int[n]` 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、`new int[] {1, 7, 4, 10}` 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。

なお、配列変数の宣言と同時に初期値を指定するときは、new 演算子を使わず、次のように単にブレースの中に初期値を列挙することができる。

```
1     int[] arr = {1, 7, 4, 10};
```

この宣言の = の右辺 {1, 7, 4, 10} は文法的には式ではないので、変数の宣言時でなければ使えない。

Q 4.9.1 次の Java プログラム（の一部）の文法的な誤りを指摘せよ。また、どう修正すればよいか？

```
1   …
2   int[] arr;
3
4   if (x > 0) {
5     arr = { 1, 2, 3, 4 };
6   } else {
7     arr = { 5, 6, 7 };
8   }
9
10  for (int i = 0; i < arr.length; i++) {
11    System.out.printf("%d, ", arr[i]);
12  }
```

```
13    …
```

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

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  class AddTime2 extends JPanel
6                  implements ActionListener {
7      private JTextField input; // e.g. 2:45 1:25 3:34
8      private JLabel output;
9
10     public AddTime2() {
11         setPreferredSize(new Dimension(180, 150));
12         input = new JTextField("1:23 4:56", 16);
13         output = new JLabel("00:00");
14         input.addActionListener(this);
15         setLayout(new FlowLayout());
16         add(input);
17         add(new JLabel("の和は"));
18         add(output);
19         add(new JLabel("です。"));
20     }
21
22     // 時間の足し算を関数として定義する。
23     private static int[] addTime(int[] t1, int[] t2) {
24         // 時間を大きさ 2の配列で表す。
25         int[] t3 = { t1[0] + t2[0], t1[1] + t2[1] };
26
27         if (t3[1] >= 60) { // 繰り上がりの処理
28             t3[0]++;
29             t3[1] -= 60;
30         }
31         return t3;  // 新しい配列を返す。
32     }
33
34     public void actionPerformed(ActionEvent e) {
35         String[] args = input.getText().split("\\s+");
36
37         int[] t = {0, 0};
38         for (String s: args) {
39             String[] stime = s.split(":");
40             t = addTime(t, new int[] {
41                     Integer.parseInt(stime[0]),
42                     Integer.parseInt(stime[1])
43                 });
44             // addTimeの呼出し前にその引数に入っていた
45             // 配列は不要となる。あとで GC される。
46         }
47         output.setText(String.format("%02d:%02d",
48                                      t[0], t[1]));
49     }
50 }
51
```

```
52 void main() {/* 省略 */}
```

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 t = addTime(t, …) という呼出しで次々と使われなくなるが、使われなくなったメモリ領域は＿＿＿＿＿＿(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、型パラメーターを持つクラスのことで、Java 5 から導入された。代表的な総称クラスの例として ArrayList, HashMap, LinkedList, ArrayDeque などがあげられる。これらは、いわゆるコンテナー（入れ物となるデータ型）である。型パラメーターは＿＿＿＿＿＿に書かれる。なお、配列も型パラメーターを持つ型だが、総称クラスとはならず、書き方が異なる。

ArrayList は＿＿＿＿＿＿＿＿＿＿＿＿のようなものである。（通常の配列と異なり、各要素の定数時間のアクセスはできない。） ArrayList の型パラメーターは要素の型を表す。（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。

```
// コンストラクターは空の ArrayList 作成
ArrayList<String> arr1 = new ArrayList<String>();
// データ追加
arr1.add("aaa");  arr1.add("bb");  arr1.add("cccc");
// 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。 ArrayList の無引数のコンストラクターは空の ArrayList を生成する。本来は総称クラスのコンストラクターは型パラメーターが必要だが、Java 8 からは文脈から推論できる場合、次のように省略できるようになった.

```
ArrayList<String> arr1 = new ArrayList<>();
```

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: ＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

なお 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ー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。（プリミティブ型は、型の名前が小文字で始まるので区別できる。）このときは Integer, Double などの対応する＿＿＿＿＿＿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 (byte, short, long, float) 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。なお、配列はプリミティブ型ではないので、int[], double[] などは何の問題もなく総称クラスの型パラメーターになることができる。

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、型パラメーターとして 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ー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。

```
// コンストラクターは空の ArrayList 作成
ArrayList<Integer> arr2 = new ArrayList<>();
// 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<> ();
arr1.add(333);          // 型エラー

ArrayList<Integer> arr2 = new ArrayList<> ();
…
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 ArrayList()
空のリストを作成します。
public boolean add(E e)
リストの最後に、指定された要素 (e) を追加します。
public E get(int index)
リスト内の指定された位置 (index) にある要素を返します。

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。（**ヒント:** `double` はプリミティブ型である。）

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、 `paintComponent` メソッドでそれを利用している。なお、配列型は総称クラスの型パラメーターとして問題なく使用することができる。実際、ラッパークラスは要素数 1 の配列のようなものである。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、`ArrayList` を使用している。

ファイル MouseDraw.java

```
 1 import java.awt.*;
 2 import javax.swing.*;
 3 import java.awt.event.*;
 4 import java.util.ArrayList;
 5
 6 class MouseDraw extends JPanel
 7                  implements MouseListener {
 8     private ArrayList<int[]> points;
 9
10     public MouseDraw() {
11         setPreferredSize(new Dimension(640, 640));
12         points = new ArrayList<>();
13         addMouseListener(this);
14     }
15
16     public void mouseClicked(MouseEvent e) {
17         points.add(new int[] { e.getX(), e.getY() });
18         repaint();
19     }
20
21     public void mouseEntered(MouseEvent e) {}
22     public void mouseExited(MouseEvent e) {}
23     public void mousePressed(MouseEvent e) {}
24     public void mouseReleased(MouseEvent e) {}
25
26     @Override
27     public void paintComponent(Graphics g) {
28         super.paintComponent(g);
29
30         int i, n = points.size();
31         for (i = 1; i < n; i++) {
32             int[] p0 = points.get(i - 1);
33             int[] p1 = points.get(i);
34             g.drawLine(p0[0], p0[1], p1[0], p1[1]);
35         }
36     }
37 }
38
```

```
39 void main() {/* 省略 */}
```

例題 4.10.4 色の名前

HashMap は＿＿＿＿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、要素を格納・検索することができる。 HashMap の型パラメーターは 2 つあり、1 つめがキーの型、2 つめが要素の型である。下の例では、HashMap<String, Color>、 つまりキーが String 型で要素が Color 型の連想配列を用いている。要素の格納には 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 HashMap()
空の HashMap を作成します。
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ます。
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します。

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。get メソッドの引数の型として、Object 型が使われている。

ファイル ColorName.java

```
1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.*;
4 import java.util.HashMap;
5
6 class ColorName extends JPanel
7                  implements ActionListener {
8     private HashMap<String, Color> hm;
9     private JTextField input;
10
11    public ColorName() {
12        setPreferredSize(new Dimension(250, 120));
13        // http://www.colordic.org/w/ より抜粋
14        hm = new HashMap<>();
15        hm.put("鴇", new Color(0xf7acbc));
16        hm.put("赤", new Color(0xed1941));
17        hm.put("朱", new Color(0xf26522));
18        hm.put("桃", new Color(0xf58f98));
19        hm.put("緋", new Color(0xaa2116));
20        hm.put("橙", new Color(0xf47920));
21        hm.put("褐", new Color(0x843900));
22        hm.put("黄", new Color(0xffd400));
23        hm.put("鶸", new Color(0xcbc547));
24        hm.put("鶯", new Color(0x87943b));
25        hm.put("緑", new Color(0x45b97c));
26        hm.put("鉄", new Color(0x005344));
27        hm.put("水", new Color(0xafdfe4));
28        hm.put("青", new Color(0x009ad6));
29        hm.put("藍", new Color(0x145b7d));
```

```
        hm.put("紺", new Color(0x003a6c));
        hm.put("菫", new Color(0x6ff0aa));
        hm.put("藤", new Color(0xafb4db));
        hm.put("紫", new Color(0x8552a1));
        hm.put("白", new Color(0xfffffb));
        hm.put("灰", new Color(0x77787b));
        hm.put("黒", new Color(0x130c0e));
        hm.put("紅", new Color(0xd7003a));
        input = new JTextField("紅 白 紺", 8);
        input.addActionListener(this);
        setLayout(new FlowLayout());
        add(input);
    }

    @Override
    public void paintComponent(Graphics g) {
        String text = input.getText();
        super.paintComponent(g);
        g.setFont(new Font("SansSerif",
                           Font.BOLD, 64));
        String[] colors = text.split("\\s+");
        for (int i = 0; i < colors.length; i++) {
            String c = colors[i];
            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();
    }
}

void main() {/* 省略 */}
```

問 4.10.5 総称クラス `java.util.LinkedList`, `java.util.ArrayDeque` 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 LeftRightButton.java

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

class LeftRightButton extends JPanel
                      implements ActionListener {
    private int x = 20;
    private JButton lBtn, rBtn;

    public LeftRightButton() {
        setPreferredSize(new Dimension(200, 70));
        lBtn = new JButton("Left");
```

```
13          rBtn = new JButton("Right");
14          lBtn.addActionListener(this);
15          rBtn.addActionListener(this);
16          setLayout(new FlowLayout());
17          add(lBtn); add(rBtn);
18      }
19
20      @Override
21      public void paintComponent(Graphics g) {
22          super.paintComponent(g);
23          g.drawString("HELLO WORLD!", x, 55);
24      }
25
26      public void actionPerformed(ActionEvent e) {
27          Object source = e.getSource();
28          if (source == lBtn) {       // lBtnが押された
29              x -= 10;
30          }
31          else if (source == rBtn) { // rBtnが押された
32              x += 10;
33          }
34          repaint();
35      }
36  }
37
38  void main() {/* 省略 */}
```

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 GUI アプリケーションを `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） $(\text{華氏の温度}) = (\text{摂氏の温度}) \times 1.8 + 32$
例えば摂氏 0 度は華氏 32 度、摂氏 100 度は華氏 212 度になる。また、華氏 0 度は摂氏 -17.8 度、華氏 100 度は摂氏 37.8 度となる。華氏は、人間が通常生活できる気温が 0 ～ 100 の範囲に収まるようにしたものである。

（参考） `String` 型を `double` 型（実数の型）に変換するには、
_____ 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、＿＿＿＿＿＿＿＿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 PrintWriter 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 型に変換した値を表す 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 "0.33" という小数第 2 位までの String 型に変換した値を表す Java の式を書け。
答: ＿＿＿＿＿＿＿＿＿＿＿＿

参考:
https://docs.oracle.com/javase/jp/25/docs/api/java.base/java/lang/String.html#format(java.lang.String,java.lang.Object...)

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 メソッドを用いる方法は効率が悪い。次の例のように＿＿＿＿＿＿（インナークラス, inner class） を用いるほうが効率が良い。

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

ファイル LeftRightButton2.java

```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4
5 class LeftRightButton2 extends JPanel {
6     private int x = 20;
7
```

```
    public LeftRightButton2() {
        setPreferredSize(new Dimension(200, 70));
        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);
    }

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

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

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

void main() {/* 省略 */}
```

Java ではクラスの中にクラスを定義することができる（これが内部クラスである）。これは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、`addActionListener` 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。ただ、クラスを定義するので、ソースコードはそれほど短くなっていない。そこで、匿名クラスというものがある。

4.13 匿名クラス

内部クラスに名前をつけずに（＿＿＿＿＿＿, ＿＿＿＿＿＿, anonymous class）そのインスタンスを生成することができる。名前のないクラスのオブジェクトは次のような式で作成する。

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

```
}
```

この形式の前半（このスタイルの部分）は通常のコンストラクターの呼出しと同じカタチで、後半（このスタイルの部分）はクラスの定義の本体（{ と } の間）に同じカタチである。

スーパークラスのコンストラクターは 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 LeftRightButton.java を匿名クラス (anonymous class) を用いて書き換える、

ファイル LeftRightButton3.java

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  class LeftRightButton3 extends JPanel {
6     private int x = 20;
7
8     public LeftRightButton3() {
9        setPreferredSize(new Dimension(200, 70));
10       JButton lBtn = new JButton("Left");
11       JButton rBtn = new JButton("Right");
12       lBtn.addActionListener(new ActionListener() {
13          public void actionPerformed(ActionEvent e) {
14             x -= 10;
15             repaint();
16          }
17       });
18       rBtn.addActionListener(new ActionListener() {
19          public void actionPerformed(ActionEvent e) {
20             x += 10;
21             repaint();
22          }
23       });
24       setLayout(new FlowLayout());
25       add(lBtn); add(rBtn);
26    }
27
28    @Override
29    public void paintComponent(Graphics g) {
30       super.paintComponent(g);
31       g.drawString("HELLO WORLD!", x, 55);
32    }
33 }
34
35 void main() {/* 省略 */}
```

4.14 final 修飾子と実質的に final

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メソッドの局所変数は、代入によって値を変更してはいけないことになっている。（なお、フィールドの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

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  class FinalExample extends JPanel {
6     private static final Color[] colors = {
7         Color.RED, Color.GREEN, Color.BLUE};
8     private int c = 0;
9
10    public FinalExample() {
11       setPreferredSize(new Dimension(200, 70));
12       JButton button  = new JButton("Push");
13       button.setForeground(colors[c]);
14       button.addActionListener(new ActionListener() {
15          public void actionPerformed(ActionEvent e) {
16             c = (c + 1) % colors.length;
17             button.setForeground(colors[c]);
18          }
19       });
20       setLayout(new FlowLayout());
21       add(button);
22    }
23 }
24
25 void main() {/* 省略 */}
```

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。(これに対して c や colors はフィールドである。) そのため、button = null; のようにあとで button に別の値を代入しようとするとエラーになる。 なお、代入によって変更されないことを明示的にするため、変数の宣言に final という修飾子を付けることがある。例えば、上の例の場合、次のように宣言する。

```
1  final JButton button  = new JButton("Push");
```

Java 7 以前では、内部クラスから参照されるメソッドの局所変数は final と宣言されている必要があったが、 Java 8 から "実質的に" final であれば（つまり、final をつけてもエラーにならないならば）良いことになった。

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.

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  class Factorial _______________ {
6    public Factorial() {
7      setPreferredSize(new Dimension(300, 50));
8      JTextField input = new JTextField("0", 8);
9      JLabel output = new JLabel("   1");
10     input.addActionListener(_____________________{
11       public void actionPerformed(ActionEvent e) {
12         // actionPerformed の中身は同じなので省略
13       }
14     });
15     setLayout(new FlowLayout());
16     add(input);  add(new JLabel("の階乗は"));
17     add(output); add(new JLabel("です。"));
18   }
19
20   // factorial, main の定義は同じなので省略
21 }
```

問 4.14.3 MouseTest.java, KeyTest.java を匿名クラスを用いて書き換えよ。なお、 MouseAdapter クラス、 KeyAdapter クラスという、それぞれ MouseListener, KeyListener インタフェースのメソッドに対してすべて空のメソッドを定義している（java.awt.event パッケージの）クラスが用意されているので、これらを利用せよ。

4.15 匿名クラスの使用例

匿名クラスは、もちろんイベントリスナー以外にも使うことができる。以下の例は、Graph2.java とほぼ同じ機能のプログラムだが、JPanel コンポーネント（28–43 行目の innerPanel）を使って、テキストフィールドとパネルをレイアウトしている。この JPanel を匿名クラスとして paintComponent メソッドを定義している。

「インスタンス初期化子」とコメントしているブロックはインスタンス初期化子 (instance initializer) と呼ばれる部分である。クラスの内部に、フィールドやメソッドの宣言と並べてブロックを記述すると、このなかの文がインスタンス生成時に実行される。通常は、このような処理はコンストラクターの中に書くが、匿名クラスはコンストラクターを持つことができないので、インスタンス生成時に必要な処理はインスタンス初期化子の中に書く必要がある。

ファイル Graph2InnerPanel.java

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

```
3  import java.awt.event.*;
4
5  class Graph2InnerPanel extends JPanel {
6    private int[] is = {};
7    private final Color[] cs = {Color.RED, Color.BLUE};
8    private final int scale = 15;
9
10   public Graph2InnerPanel() {
11     setPreferredSize(new Dimension(200, 230));
12     JTextField input = new JTextField("", 16);
13     input.addActionListener(new ActionListener() {
14       public void actionPerformed(ActionEvent e) {
15         String[] args = input.getText().split(" ");
16         int n = args.length;
17         is = new int[n];
18
19         int i;
20         for (i = 0; i < n; i++) {
21           is[i] = Integer.parseInt(args[i]);
22         }
23         repaint();
24       }
25     });
26     setLayout(new FlowLayout());
27     add(input);
28     JPanel innerPanel = new JPanel() {
29       { // インスタンス初期化子
30          setPreferredSize(new Dimension(200, 200));
31       }
32       @Override
33       public void paintComponent(Graphics g) {
34         super.paintComponent(g);
35         int i;
36         int n = is.length;
37         for (i = 0; i < n; i++) {
38           g.setColor(cs[i % 2]);
39           g.fillRect(0, i * scale,
40                      is[i] * scale, scale);
41         }
42       }
43     };
44
45     add(innerPanel);
46   }
47 }
48
49 void main() {/* 省略 */}
```

4.16 ラムダ式

Java 8 からは、`ActionListener` のようにメソッドを一つしか持たないインタフェースを実装する匿名クラスのオブジェクトに対して、＿＿＿＿(lambda expression, λ expression) というメソッド名も省略する書き方ができるようになった。（`MouseListener` や `KeyListener` はメソッドを 2 つ以上持つので、ラムダ式では書けない。）

例題 4.16.1 例えば、`LeftRightButton.java` をラムダ式を用いて書き換えると次のようになる。

ファイル LeftRightButton4.java

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

class LeftRightButton4 extends JPanel {
    private int x = 20;

    public LeftRightButton4() {
        setPreferredSize(new Dimension(200, 70));
        JButton lBtn = new JButton("Left");
        JButton rBtn = new JButton("Right");
        lBtn.addActionListener(e -> {
            x -= 10;
            repaint();
        });
        rBtn.addActionListener(e -> {
            x += 10;
            repaint();
        });
        setLayout(new FlowLayout());
        add(lBtn); add(rBtn);
    }

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

void main() {/* 省略 */}
```

ラムダ式は匿名クラスの（一つしかない）メソッド定義部分だけを抜き出し、さらにメソッド名も省略したものである。引数のリストと関数本体を -> でつなぐかたちになる。

- (型$_1$ 変数$_1$, 型$_2$ 変数$_2$, …, 型$_n$ 変数$_n$) -> { 文のならび }

通常は、引数の型は省略できる。

- (変数$_1$, 変数$_2$, …, 変数$_n$) -> { 文のならび }

さらに、引数が一つの場合は、丸括弧も省略できる。

- 変数 -> { 文のならび }

またブレースの中の文のならびが、式文が一つだけ、または return 文一つだけのときは、ブレース（{ と }）と return、セミコロン (;) も省略することができる。この場合、それぞれ次のような形になる。

- (型$_1$ 変数$_1$, 型$_2$ 変数$_2$, …, 型$_n$ 変数$_n$) -> 式
- (変数$_1$, 変数$_2$, …, 変数$_n$) -> 式
- 変数 -> 式

Q 4.16.2 例 4.14 の FinalExample.java をラムダ式を用いて書き換えよ。

```
button.addActionListener(


);
```

問 4.16.3 § 4.11 の問 4.11.2 の摂氏と華氏の変換 GUI アプリケーションをラムダ式を使って書け。

4.17 javax.swing.Timer クラス

目に見える部品ではないがタイマー（Timer クラス）もボタンなどと同じように ActionEvent を発生させることができる。一定間隔でイベントを発生させることができるため、アニメーションなどに利用できる。

Timer クラスのコンストラクターには、イベントを発生させる間隔（ミリ秒単位）と ActionListener を渡す。（JButton などと同様に、あとから addActionListener メソッドでリスナーを追加することもできる。）

```
public Timer(int delay, ActionListener listener)
```

このクラスのオブジェクトの start メソッドを呼び出すことでタイマーを開始する。また stop メソッドを呼び出すことでタイマーを停止することができる。次の例は、物体を時間の経過に従って円状に動かすプログラムである。

ファイル Guruguru.java

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

class Guruguru extends JPanel {
  private int r = 50;
  private int x = 110, y = 70 ;
  private double theta = 0; // 角度

  public Guruguru() {
    setPreferredSize(new Dimension(200, 180));
    Timer timer = new Timer(30, e -> {
      theta += 0.02;
      x =  60 + (int)(r * Math.cos(theta));
      y = 100 - (int)(r * Math.sin(theta));
      repaint();
    });
    timer.start();
    JButton startBtn = new JButton("start");
    startBtn.addActionListener(e -> timer.start());
    JButton stopBtn = new JButton("stop");
    stopBtn.addActionListener(e -> timer.stop());
    setLayout(new FlowLayout());
    add(startBtn); add(stopBtn);
  }

  @Override
  public void paintComponent(Graphics g) {
    // スーパークラスの paintComponent を呼び出す
```

```
29      super.paintComponent(g);
30      // 全体を背景色で塗りつぶす。
31      g.drawString("Hello, World!", x, y);
32    }
33  }
34
35  void main() {/* 省略 */}
```

4.18 章末問題

問 4.18.1 JTextArea, 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 4.18.2 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、イベントハンドラー、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, getContentPaneメソッド, 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, ArrayDequeクラス、内部クラス、匿名クラス、ラムダ式、 Timerクラス