グラフィックス及び演習・第 8 回

2023 年 7 月 31 日

今回の内容

8.1 これまでのまとめ

JFC による GUI の構築 [第 2, 3, 5 回]

- Java には、Java Foundation Classes (JFC) と呼ばれる、グラフィカルユーザーインタフェイス (GUI) を構築するための枠組み (クラス群) が用意されている。
- JFC のクラスライブラリとして、Swing と呼ばれる GUI 部品のセットが提供されている。
- GUI アプリケーションのウィンドウやその内部の GUI 部品に対するユーザーの操作 (キーボード操作、マウス操作など) を、一般にイベントと呼び、イベントの発生に対するプログラム側の処理をイベント処理と呼ぶ。
- JFC では、イベント処理や、再描画の処理をイベントディスパッチスレッドど呼ばれるスレッドで行う。
- Swing に含まれる GUI 部品が描画される際には、そのオブジェクトをターゲットとして、`paintComponent` というメソッドが起動されるので、これを再定義することで、そのクラス独自の描画を行うことができる。
- JFC では、GUI の各部品に対してイベントハンドラ (イベントリスナ) を登録しておくことでイベント処理を行う。
- 各イベントハンドラの資格は `ActionListener` や `MouseListener` などのインタフェースとして定義されており、発生したイベントが `ActionEvent` や `MouseEvent` などのクラスのインスタンスとして表現され、そのオブジェクトを引数として、イベントハンドラの特定のインスタンスメソッド (`actionPerformed` など) が起動される。

インタフェース [第 3 回]

- インタフェースとはオブジェクトの資格のこと。
- インタフェースはインタフェース宣言を行うことで定義される。
- インタフェース宣言では、抽象メソッド (どのような型の引数をもらって、どのような型の戻りを返す、どのような名前のメソッドなのか) の宣言を行う[1]。
- 既存のインタフェースを元にして、新しいインタフェースを宣言することもできる。
- あるクラスのオブジェクトが、あるインタフェースが要求する条件を満たすとき、そのクラスはそのインタフェースを実装していると言う。

[1] 他にも、定数 (final なクラス変数) やメンバクラス、メンバインタフェースの宣言を行うことができる。

- 1 つのクラスが複数のインタフェースを実装することもできる。
- そのクラスが実装しているインタフェース群は、クラス宣言の `extends` 節の後 (なければ クラス名 の後) に、`implements` 節を使って宣言する。

例外 [第 4 回]

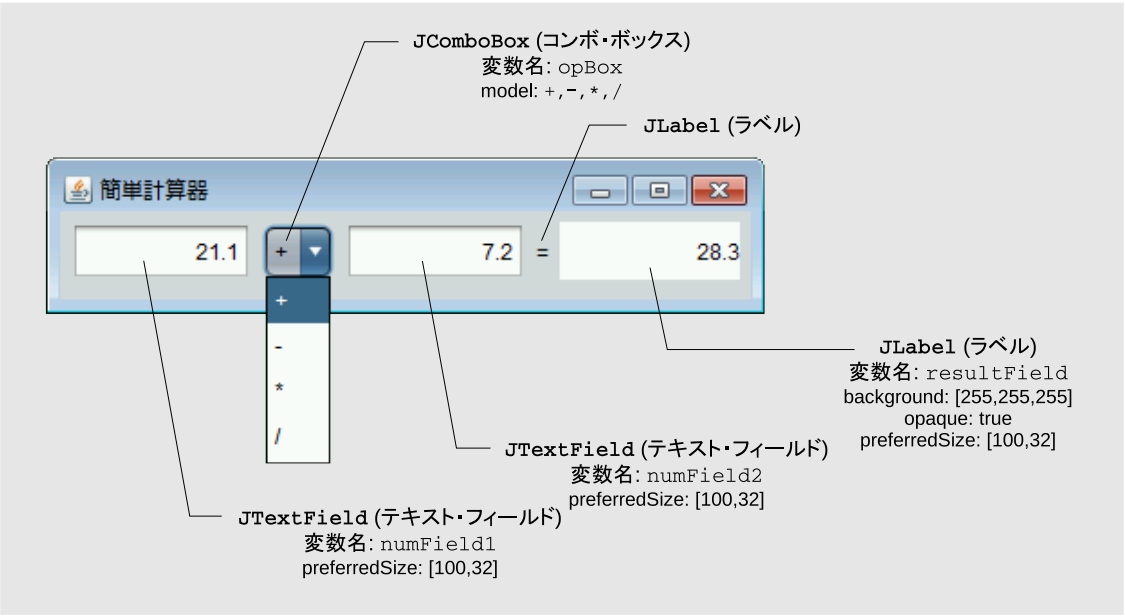
- プログラムの実行中に発生する予期しない出来事を例外と呼ぶ。
- 例外の発生を検知して、それに対する処置を行うことを例外処理と呼ぶ。
- Java では、例外が発生すると、例外を表すクラスのインスタンスが生成され、そのオブジェクトがスローされる。
- Java では `try` 文とその `catch` 節を用いて、スローされた例外をキャッチし、例外処理を行うことができる。

スレッド [第 7 回]

- プログラムの一筋の実行の流れをスレッドと呼ぶ。
- Java では、複数のスレッドを同時並行的に実行することができる。
- プログラムの特定の部分が、同時に複数のスレッドで実行されないようにすることを排他制御と呼ぶ。
- Java では、`synchronized` 文や、メソッド宣言に対する `synchronized` 修飾子を用いて排他制御を行うことができる。

8.2 演習問題

1. 次の図のような電卓プログラム `Calc.java` を作成しなさい。



ただし、Calc クラスは JFrame のサブクラスとし、タイトル (title) を「簡単計算器」としてください。また、numField1 や opBox、numField2 で ActionEvent が発生した時に、次のようなメソッドを起動して、resultField が更新されるようにしてください。

```
void doCalc() {
    try {
        double x = Double.parseDouble(numField1.getText());
        double y = Double.parseDouble(numField2.getText());
        double r = 0.0;
        String s = (String) opBox.getSelectedItem();
        if ("+".equals(s)) {
            r = x + y;
        } else if ("-".equals(s)) {
            r = x - y;
        } else if ("*".equals(s)) {
            r = x * y;
        } else if ("/".equals(s)) {
            r = x / y;
        }
        resultField.setText("" + r);
    } catch (NumberFormatException e) {
        resultField.setText("Error!");
    }
}
```

ヒント: 次のプログラムの空欄部分を埋めるか、NetBeans IDE などの GUI デザイン機能を使用しましょう。

Calc.java

```
 1 import java.awt.*;
 2 import javax.swing.*;
 3
 4 public class Calc implements Runnable {
 5     String[] ops = { "+", [          ] };
 6     JTextField numField1 = new JTextField();
 7     JComboBox opBox = new JComboBox(new DefaultComboBoxModel(ops));
 8     JTextField numField2 = new [          ]();
 9     JLabel equalLabel = new [          ]("=");
10     JLabel resultField = new [          ]();
11
12     void doCalc() {
13         try {
14             double x = Double.parseDouble(numField1.getText());
15             double y = Double.parseDouble(numField2.getText());
16             double r = 0.0;
17             String s = (String) opBox.getSelectedItem();
18             if ("+".equals(s)) {
19                 r = x + y;
20             } else if ("-".equals(s)) {
21                 r = x - y;
22             } else if ("*".equals(s)) {
23                 r = x * y;
24             } else if ("/".equals(s)) {
25                 r = x / y;
26             }
```

```
27              resultField.setText("" + r);
28          } catch (NumberFormatException e) {
29              resultField.setText("Error!");
30          }
31      }
32
33      public void run() {
34          numField1.setPreferredSize(new Dimension(100, 32));
35          numField2.setPreferredSize(                          );
36          resultField.setPreferredSize(                          );
37          resultField.setBackground(                 );
38          resultField.setOpaque(                 );
39          numField1.addActionListener(                 );
40          numField2.addActionListener(                 );
41          opBox.addActionListener(                 );
42          JFrame f = new JFrame(                 );
43          f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
44          f.setLayout(new FlowLayout());
45          f.add(                 );
46          f.add(                 );
47          f.add(                 );
48          f.add(                 );
49          f.add(                 );
50          f.pack();
51          f.setVisible(true);
52      }
53
54      public static void main(String[] args) {
55          SwingUtilities.invokeLater(new Calc());
56      }
57  }
```

8.3 付録: Java のキーワード類一覧

`abstract` — 抽象クラス、あるいは抽象メソッドの宣言であること示す修飾子 [第 3 回 5 ページ, 第 5 回 13 ページ]

`boolean` — 真理値の型 (プリミティブ型の一つ)

`break` — `break` 文の始まり (`break` ラベル `;` の書式で、ラベル を持つ文を `break` することもできる)

`byte` — 8 bit の符号付き整数の型 (プリミティブ型の一つ)

`catch` — `try` 文の `catch` 節の始まり [第 4 回 8 ページ]

`char` — UTF-16 の文字コードでの 1 符号単位 (16 bit の符号付き整数) の型 (プリミティブ型の一つ)

`class` — クラス宣言の始まり[2]

`const` — 使用できない

`continue` — `continue` 文の始まり (`continue` ラベル `;` の書式で、ラベル を持つ `while` 文、`for` 文、`do` 文を `continue` することもできる)

`default` — `switch` 文内の `default:` ラベル

`do` — `do` 文の始まり

`double` — 64 bit の浮動小数点数の型 (プリミティブ型の一つ)

`else` — `if` 文の `else` 節の始まり

`extends` — クラス宣言の `extends` 節 (そのクラスの直接のスーパークラスを指定する) の始まり

`false` — `boolean` 型で偽を表すリテラル

`final` — そのクラスのサブクラスが宣言できないこと、その変数が (一旦初期化したら) 値を変えられないこと、あるいは、そのメソッドが (サブクラスで) 再定義できないことを示す修飾子

`finally` — `try` 文の `finally` 節の始まり [第 4 回 9 ページ]

`float` — 32 bit の浮動小数点数の型 (プリミティブ型の一つ)

`for` — `for` 文の始まり

`goto` — 使用できない

`if` — `if` 文の始まり

`implements` — クラス宣言の `implements` 節 (そのクラスが実装するインタフェース群を指定する) の始まり [第 3 回 6 ページ]

`import` — インポート宣言 (特定のパッケージに属するクラスやインタフェース、あるいは、それらのクラス変数やクラスメソッドなどを単純名で参照する設定) の始まり

---

[2] 型名 `.class` という書式の式で、型名 に対応する `java.lang.Class` クラスのインスタンスを表すのにも使われます。

`instanceof` — 式 `instanceof` 参照型名 という書式の式を書いて、式 を評価した結果が 参照型名 で指定した型のオブジェクトであるかどうかを `boolean` 型で表す演算子 (式 が null 参照の場合は `false` となる)

`int` — 32 bit の符号付き整数の型 (プリミティブ型の一つ)

`interface` — インタフェース宣言の始まり [第 3 回 5 ページ]

`long` — 64 bit の符号付き整数の型 (プリミティブ型の一つ)

`native` — そのメソッドが Java 以外の言語で実装されていることを示す修飾子

`new` — クラスのインスタンスや配列オブジェクト生成する式を始める

`null` — null 参照を表すリテラル

`package` — パッケージ宣言 (このソースファイルで宣言されるクラスやインタフェースが属するパッケージの指定) の始まり

`private` — メンバクラス、メンバインタフェース、インスタンス変数、インスタンスメソッド、コンストラクタ、クラス変数、クラスメソッドの宣言が、それを囲んでいるクラス宣言内だけでアクセスできることを示すアクセス修飾子

`protected` — メンバクラス、メンバインタフェース、インスタンス変数、インスタンスメソッド、抽象メソッド、コンストラクタ、クラス変数、クラスメソッドの宣言が、同じパッケージ内と、サブクラスの宣言内からアクセスできることを示すアクセス修飾子

`public` — クラス、インタフェース、インスタンス変数、インスタンスメソッド、抽象メソッド、コンストラクタ、クラス変数、クラスメソッドの宣言が、プログラムのどこからでもアクセスできることを示すアクセス修飾子

`return` — `return` 文の始まり

`short` — 16 bit の符号付き整数の型 (プリミティブ型の一つ)

`static` — その宣言が、クラス変数、クラスメソッド、クラス初期化子、あるいは静的メンバクラスの宣言であることを示す修飾子[3]、あるいは、`import` 宣言の中で、これら (クラス初期化子を除く) を単純名で参照することを示す

`strictfp` — そのクラス宣言、インタフェース宣言、メソッド宣言では、`IEEE 754` 規格に厳密に従った方法で浮動小数点数の計算を行わなければならないことを示す修飾子

`super` — コンストラクタ本体の先頭で、直接のスーパークラスのコンストラクタを起動する、あるいは、コンストラクタやインスタンスメソッド内などで、スーパークラスの変数やメソッドにアクセスするために `this` の代りに使用する

`switch` — `switch` 文の始まり

`synchronized` — `synchronized` 文 (特定のオブジェクトをロックした上で実行する文) の始まり、あるいは、ターゲットとなるオブジェクトのロックを獲得した上でメソッド本体を実行することを示す修飾子 [第 7 回 9 ページ]

[3] メンバインタフェースの宣言にも付加できますが意味は変りません。

`this` — コンストラクタ本体の先頭で、同じクラスの他のコンストラクタを起動するあるいは、コンストラクタやインスタンスメソッド本体[4]で、初期化の対象となっているオブジェクトやメソッド起動の対象 (ターゲット) となっているオブジェクトを表す式

`throw` — `throw` 文 (例外をスローする文) の始まり [第 4 回 7 ページ]

`throws` — コンストラクタやメソッド宣言の `throws` 節の始まり [第 4 回 8 ページ]

`transient` — そのインスタンス変数やクラス変数が記憶するのは一時的な値のみで、そのインスタンスやクラスの状態の表現に関わっていないこと (状態を保存する際に記録する必要がないこと) を示す修飾子

`true` — `boolean` 型で真を表すリテラル

`try` — `try` 文の始まり [第 4 回 8 ページ]

`void` — メソッド宣言の戻り値の型の位置に書いて、そのメソッドに戻り値がないことを示す

`volatile` — 他のスレッドによる代入も含めて、常に最新の値を参照しなければならないインスタンス変数またはクラス変数であることを示す修飾子 [第 7 回 12 ページ]

`while` — `while` 文の始まり、あるいは `do` 文の `while` 節の始まり

グラフィックス及び演習・第 8 回・終わり

[4] あるいは、インスタンス初期化子やインスタンス変数の初期化子内。