プログラミング言語IIIB(Java) テーマ18

グラフィックス描画処理

Java Swingでウィンドウ上にグラフィックスを描く処理は以下の手順で行う.

1. JPanelクラスを継承した独自の描画(びょうが)専用パネルクラスを作成する.
   ※普段配置用に使っているパネルを, グラフィックスを描く「キャンバス」に転用するということ.
2. 描画パネルを配置する. 配置方法は通常のGUI部品と同じ.
3. 描画パネルのpaintComponentメソッドをオーバーライド(上書き定義)し, そのメソッドの中で, 各種の図形や画像を描く.

以下に描画パネルを使用するコードの例を示す.

```
// グラフィックス描画専用の自作パネルクラス
public class MyPanel extends JPanel // **** JFrameではない!! ****
{
    public MyPanel()
    {
        super.setBackground(Color.black); // 背景色を黒に設定(この時点ではまだ実際には描画されない)
        super.setPreferredSize(new Dimension(640, 240)); // 大きさを640x240に設定
    }
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g); // まずJPanelのデフォルトの描画処理(背景色でクリア)を行い,
        g.setColor(Color.green); // 前景色を設定し,
        g.drawLine(50, 50, 100, 100); // 独自の図形を描画する.
        g.drawOval(32, 16, 256, 128); // 引数は座標や大きさなど(各自で調べましょう).
        g.drawString("文字列をグラフィックスとして表示", 48, 48);
            :
```

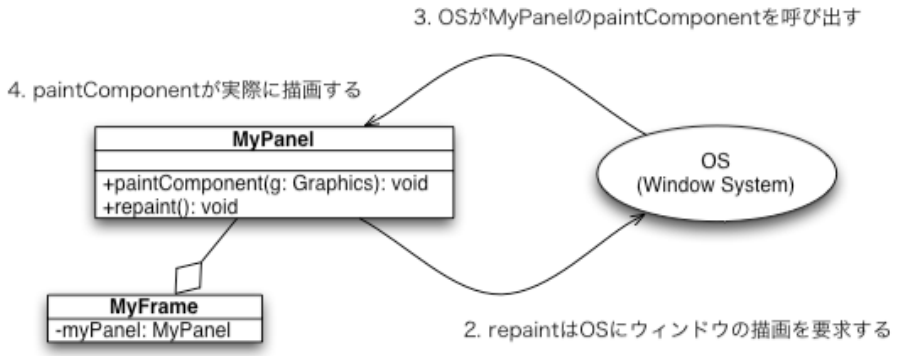
```
// 配置を行う自作フレームは従来通り
public class MyFrame extends JFrame
{
    private MyPanel mp; // MyFrame HAS-A MyPanel の関係を構築

    public MyFrame()
    {
        this.mp = new MyPanel(); // 描画用のパネル
        JPanel panel1 = new JPanel(); // 配置用のパネル
        panel1.add(this.mp);
        super.getContentPane().add(panel1);
    }
    main略
```

注意を要する点は以下の通り.

- グラフィックスを描くpaintComponentメソッドは, ウィンドウが画面に最初に表示された時や, ウィンドウが切り替わって前面に出た時など, ウィンドウをユーザーが見ている「画面に描く」ことが必要になった時点で, OSが自動的に呼び出す. プログラム内で直接呼び出すメソッドではない.
- ユーザーから操作があった時に, 即座に反応してグラフィックスを描画したい場合は, actionPerformedの中などで描画パネルに対してrepaintメソッドを呼び出す. paintComponentではないので注意.

```
// actionPerformedの中など, 意図的に再描画したい時点で
this.mp.repaint();
        ~~~~~~~paintComponentではない
```

repaintメソッドはJPanelクラスであらかじめ定義されており, 呼び出されるとOSにウィンドウの再描画を要求する. 再描画を要求されたOSは, paintComponentメソッドを呼び出す. この様に間接的にpaintComponentが呼び出されることになる. この仕組みをしっかり理解して欲しい.



※OSやJDKの環境によっては, PCへの負荷を軽減するために, OSの判断でJavaプログラムの画面更新頻度(フレームレート)が遅かったり, 停止することがある. その様に感じられた場合は, 以下のコードを試してみて欲しい.

```
this.mp.repaint();
Toolkit.getDefaultToolkit().sync(); // これを追加してみる
```

Graphicsクラス

描画パネルに各種の図形を描くには, paintComponentメソッドの引数 g (Graphicsクラスのインスタンス)を用いる. このgは, 描画パネルに図形を描くためのペン, と理解すると分かり易い. OSから専用のペンを渡され, それを使って図形を描くイメージである.

```
                                    ↓ここに注目
public void paintComponent(Graphics g)
{
    super.paintComponent(g); // まずJPanelのデフォルトの描画処理(背景色でクリア)を行い,
    g.setColor(Color.green); //  「ペンの色」を設定し,
    g.drawLine(50, 50, 100, 100); // その「ペンを使って」独自の図形を描画する
```

以下に, Graphicsクラスの主な描画メソッドを示す. これら以外にも様々な機能が用意されているので, API仕様やウェブ, 書籍等を調査して積極的に利用に挑戦してみて欲しい.

- drawLine ... 直線を引く
- drawRect ... 長方形を描く
- drawOval ... 楕円を描く
- drawPolygon ... 多角形を描く
- drawString ... 文字列を描く
- drawImage ... 画像を描く(実際には画像データをウィンドウに転写する)

JSliderクラス

ノブ(つまみ)を数直線上で動かして数値を選択入力するための部品である. ノブの位置が変化すると, イベントリスナーにChangeEventが送られる. ChangeEventを処理するリスナーの名前はChangeListener, 呼び出されるメソッドの名前はstateChangedである(大文字小文字に注意). →テーマ17「イベント処理」参照

```
画面イメージ
+------------###-------------+
             ↑ノブ(マウスやキーボードで動かせる)
```

ノブの位置(数値)をゲットするには, スライダーに対してgetValueメソッドを呼び出す. それ以外の, コンストラクタやメソッドの詳細は, 各自で調査してみて欲しい.

課題

以下の課題のレポートは, レポートファイルreport18.txtを作成してアップロードにより提出すること.

1. 円や四角形等のグラフィックスを描くプログラムを作成せよ. ウィンドウが表示された時に, 適当な位置に, 固定された図形が描画されるだけで良い. (ソースをレポート)
    - 上記の最初にある例をコピペしてソースを二つ作る.
    - mainは従来通り, フレームの方に入れる.
2. JTextFieldとJSliderを連携させるプログラムを作成せよ. ここで連携とは, スライダーのノブを動かすと随時それに応じた数値がテキストフィールドに表示されることである. (ソースをレポート)
    - 画面イメージは以下の通り.

```
+------------###-------------+  [50]
             ↑ノブを動かすと    ↑この欄の数値が変わる
```

- まずはスライダーとテキストフィールドを生成して配置する所から作り始める.
- スライダーのイベントを処理するためにimport文の追加が必要.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*; // 今回はこれも入れる
```

- スライダーのノブが動いた時 → ノブの位置(数値)をゲットして → 文字列に変換して → テキストフィールドにセットする.
- スライダーが動いた時に反応させるにはチェンジ3点セットが必要 → テーマ17参照
  - implements ChangeListener
  - this.slider1.addChangeListener(this);
  - public void stateChanged(ChangeEvent e) { }
- 数値(int)を文字列(String)に変換するには "" + 数値 とする(空文字列の後ろに連結する).

3. 課題1と課題2のプログラムを統合し, スライダーを動かすと図形が移動または拡大縮小するプログラムを作成せよ. 移動と拡大縮小はどちらか一つで良い. (ソース, 移動・拡大縮小のどちらかをやったか, をレポート)
   - フレーム側から描画パネル側へ, スライダーの値などを渡したい場合は, セッターを作成して使うと良い.

```
public class MyFrame extends JFrame implements ChangeListener
{
    public void stateChanged(ChangeEvent e) // スライダーが動いたとき
    {
        if (e.getSource() == this.slider1) {
            int val = this.slider1.値をゲット();
            this.mp.setSize(val); // 描画パネルに値をセットしてから
            this.mp.repaint();  // 描画パネルを再描画
```

```
public class MyPanel extends JPanel
{
    private int size; // 図形の大きさを保存するためのフィールドを用意しておく

    public void setSize(int size) // 引数で値を受け取って
    {
        this.size = size; // フィールドに保存しておく
    }
    public void paintComponent(Graphics g) // this.mp.repaint()で間接的にここが呼ばれる
    {
        g.drawOval( this.sizeを使って大きさを指定 );
        // drawOvalメソッドの引数については各自で調べましょう.
    }
}
```

4. JButton, JSlider, JTextField等の様々な入力部品を用いて, ユーザーの操作に応じて何らかの変化が起こるグラフィックスを描くプログラムを作成せよ. (ソース, 使用方法をレポート)
   - 例えば, ベクトルの加算結果がビジュアル(図)で表示されたり, 数値(パラメーター)がレーダーチャートや棒グラフ, 折れ線グラフで表示されたり, チェックボックスで図形やイラストが切り替わる・変化する等が考えられる. →チェックボックスを使った連携の方法については下記「テクニック」を参照
   - どうしてもアイデアが浮かばない場合は, 課題3にもう一つスライダーを追加し, 図形の移動と拡大縮小が両方できるプログラムでも良い.
   - 注意: ペイントアプリの様な, マウスのドラッグ操作でお絵描きするプログラムではないので勘違いしないこと. →それは次回テーマ19にて

レポート

- 内容: 課題中に指示されている通り. 必要な項目を全て記載しているか, 十分に確認すること.

テクニック

- チェックボックス等の状態に応じて描画内容を切り替えるには, 描画パネルに状態を表すフィールドを追加し, 描画時の判断に利用すると良い. フィールドとゲッター・セッターを思い出して活用して欲しい.

```
【フレーム側】
// ----- actionPerformed の中で ----
if (this.cb1.isSelected()) {    // チェックボックスが選択されているならば
    this.mp.setDrawFlag(true);  // trueをセット
} else {                        // そうでなければ
    this.mp.setDrawFlag(false); // falseをセット
}

// 上記の様にifで分岐させずに, isSelected()の値を直接そのままmpにセットしても良い.

this.mp.repaint();              // セットした後に再描画
```

```
【描画パネル側】
```

```
// ---- フィールドで ----
private boolean drawFlag; // 画像描画フラグ

(drawFlagのセッターも自分で作成して追加) ←もう作れるはず
          ~~~~~~~~~~~~~~~~~~~~

// ---- paintComponent の中で ----
if (this.drawFlag == true) {    // フラグがtrueならば
    図形を描画
}
```

- 図形が複雑または多数になり, paintComponentメソッドが長くなりすぎた場合は, メソッドを分割し, paintComponentの中からgを引数として呼び出せば良い.
- 描画する線の太さを変えたり, 線の端を丸めたりするには, 以下の様にGraphics2Dクラスを用いる. 詳しくはAPI仕様やウェブ, 書籍などを調査してみて欲しい.

```
// 例: 線の太さを8.0にする
Graphics2D g2 = (Graphics2D)g;
g2.setStroke(new BasicStroke(8.0F));
g.drawLine(～);
```

```
// 例: 線の端を丸くする(デフォルトでは角)
Graphics2D g2 = (Graphics2D)g;
g2.setStroke(new BasicStroke(1.0F, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
g.drawLine(～);
```

- アンチエイリアス(ピクセルを滑らかにする機能)も使用可能であるため, 美しさを追求したい人は調査して利用してみると良い.