

- グラフィックス

★10 グラフィックス

★10.3 Canvas クラス (p.235)

Canvas クラスを利用する場合、グラフィックスを描く手順は次のようになる。

1. Canvas オブジェクトに対して `getGraphicsContext2D` メソッドを呼び出して `GraphicsContext` クラスのオブジェクトを取得する。
2. 取得したオブジェクトのメソッド (p.236-238 参照) を使って描画する

`GraphicsContext` オブジェクトは、対応する `Canvas` オブジェクトの現在の描画状態などの情報を保持するとともに、図形などを描画するメソッドを備えている。

`Canvas` クラスを使う場合、`Shape` クラスの場合と違って個々の図形はオブジェクトではない。そのため図形に対して個別にイベント処理を設定することはできない (☆1) が、そのかわり描画は高速である。

以下に、`Canvas` クラスを利用するプログラムの例を示す。イベント処理も行うものになっている。

☆1) マウスイベントなどの場合、カーソル位置を取得 → それを対象とする円や矩形の内部であるかどうか判定 → イベント処理、というような手続きを書けば、`Shape` オブジェクトの場合よりも煩雑ではあるが、擬似的に個々の図形に対するイベント処理を実現できる。

```

CanvasSample.java
1  import javafx.application.Application;
2  import javafx.stage.*;           // Stage
3  import javafx.scene.*;          // Scene
4  import javafx.scene.layout.*;   // Pane, HBox
5  import javafx.scene.canvas.*;   // Canvas, GraphicsContext
6  import javafx.scene.paint.*;    // Color
7  import javafx.scene.image.*;   // Image
8  import javafx.event.*;         // MouseEvent
9
10 public class CanvasSample extends Application {
11
12     @Override
13     public void start(Stage pstage){
14
15         // Canvas オブジェクトを直接レイアウトコンテナに入れることもできるが、
16         // この例では Pane に入れて Pane を HBox に入れている
17         Pane pane1 = new Pane(); // Pane コンテナその1
18         Pane pane2 = new Pane(); // Pane コンテナその2
19         Pane pane3 = new Pane(); // Pane コンテナその3
20         Canvas canvas1 = new Canvas(200, 200); // 描画用 Canvas その1
21         Canvas canvas2 = new Canvas(200, 200); // 描画用 Canvas その2
22         Canvas canvas3 = new Canvas(200, 200); // 描画用 Canvas その3
23         pane1.getChildren().add(canvas1); // canvas1 を pane1 に載せる
24         pane2.getChildren().add(canvas2); // canvas2 を pane2 に載せる
25         pane3.getChildren().add(canvas3); // canvas3 を pane3 に載せる
26
27         HBox root = new HBox(pane1, pane2, pane3); // ルートノード
28         Scene scene = new Scene(root);
29         pstage.setTitle("CanvasSample");
30         pstage.setScene(scene);
31         pstage.show();

```

(次頁へつづく)

CanvasSample.java (つづき)

```
32
33     pane2.setStyle("-fx-background-color: #87CEEB;"); // 背景色を設定
34     drawFigures(canvas1); // canvas1 に描画
35     drawFigures2(canvas2); // canvas2 に描画
36     drawImages(canvas3); // canvas3 に描画
37 }
38
39 // 変数 c が表す Canvas 上に図形を描画
40 void drawFigures(Canvas c) {
41     // グラフィックスコンテキストを取得
42     GraphicsContext gc = c.getGraphicsContext2D();
43
44     gc.setStroke(Color.BLUE); // 線の色を設定
45     gc.setLineWidth(3); // 線の太さを設定
46     gc.strokeRect(30, 50, 100, 80); // 矩形
47
48     gc.setFill(Color.YELLOW); // 塗りつぶしの色を設定
49     gc.fillOval(80, 80, 100, 80); // 楕円
50 }
51
52 // 変数 c が表す Canvas 上に図形を描画 & イベント処理
53 void drawFigures2(Canvas c) {
54     GraphicsContext gc = c.getGraphicsContext2D();
55     gc.setFill(Color.BLUE);
56
57     // イベントハンドラを設定 (ドラッグすると●を描く)
58     c.setOnMouseDragged((event) -> {
59         double x = event.getX(), y = event.getY();
60         gc.fillOval(x - 10, y - 10, 20, 20);
61     });
62 }
63
64 // 変数 c が表す Canvas 上に画像を表示 & イベント処理
65 void drawImages(Canvas c) {
66     GraphicsContext gc = c.getGraphicsContext2D();
67     Image img = new Image("blackuni.jpg");
68
69     // イベントハンドラを設定 (クリックすると画像を描く)
70     c.setOnMouseClicked((event) -> {
71         double x = event.getX(), y = event.getY();
72         gc.drawImage(img, x - 20, y - 15, 40, 30);
73     });
74 }
75
76 public static void main(String[] args) {
77     launch(args);
78 }
79 }
80
81
82 }
```