# Java SE 8 / Java FX 8 概説

－　Java の新しい API　－



中村勝則

2016 年 4 月 1 日

# 目 次

# 1　はじめに

Java FX は簡便な方法で GUI や三次元グラフィックスを実現するために開発されたライブラリであり，Swing に代わるものとして位置づけられている．また Java FX は，RIA アプリケーション（Rich Internet Application）の開発を指向しており，異なる計算機環境でも統一的な方法でアプリケーションプログラムの開発を可能にする．

Java のバージョンは本書を執筆している時点（2015 年 8 月）で「8」であり，Java FX のバージョンも「8」である．（以後「Java FX 8」と表記する）このバージョンの Java FX は，Swing と比べても GUI やグラフィックスの表現能力が高く，今後は Java における GUI ライブラリの標準的な存在となる．

Java FX 8 では，**FXML** と呼ばれる拡張された XML 言語によって GUI を記述することができ，CSS や JavaScript を併用することで，GUI 構築における高い表現力を実現している．

# 2　GUI 構築の基本

## 2.1　Application クラス

Java FX には **Application** クラスが提供されており，アプリケーションプログラムはこのクラスの拡張クラスとして実装する．またアプリケーション起動時にそのクラスの main メソッドが起動する．main メソッド内で **launch メソッド**を呼び出すことで，GUI 構築とイベント処理が開始する．

アプリケーションのクラス内では **start メソッド**を記述し，start メソッド内に GUI の構築とイベント処理に関する詳細を記述する．start メソッドは Application クラス内で定義されているメソッドであり，プログラマが start メソッドを記述する場合は，**@Override アノテーション**を start メソッドの記述の直前に付けて**オーバーライド**する必要がある．

## 2.2　Stage クラスと Scene クラス

**Stage クラス**のオブジェクトはウィンドウの基盤となるもので，ウィンドウとその内容は Stage クラスのオブジェクト上に構築する．GUI を構成する部品やグラフィックス（2D,3D の図形オブジェクト），あるいはそれらをまとめる**コンテナ**などは階層的な**シーングラフ**として構築（図 1）される．

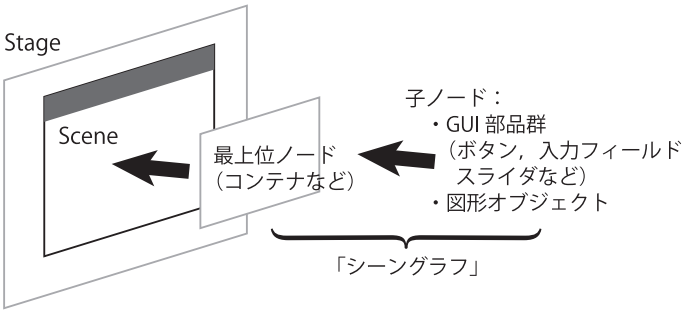



図 1: ウィンドウの成り立ち

GUI のオブジェクト群はいくつかの大きなカテゴリに分類されており，まず基本的かつ重要なものとして，**Containers** と **Controls** のカテゴリのものを取り上げて説明する．

これからサンプルプログラムを例示して，Java FX アプリケーション構築の基本的な流れについて説明する．

**【サンプル】**ボタンをクリックすると標準出力にメッセージを出力するプログラム：FXsample01

まず，Application クラスの拡張クラスとして FXsample01 クラスを定義する．Application クラスは，クラス階層 javafx.application 配下に定義されており，これを使用するには，予め必要なクラスライブラリを**インポート**しておく必要がある．

アプリケーションプログラムの構造は次のようになる．

**アプリケーション** FXsample01 **の定義**

```
import javafx.application.*;
import javafx.stage.*;
import javafx.event.*;
public class FXsample01 extends Application {
        @Override
        public void start(Stage stg) {
                （GUI の構築とイベント処理の記述）
        }
        public static void main(String[] args) {
                launch(args);
        }
}
```

Container のカテゴリに属するオブジェクト（Pane など）は GUI を構築する“ 台紙 ”のようなものであり，この上に Controls のカテゴリに属するオブジェクト（ボタンやテキストフィールドなど）を配置して GUI を構築する．GUI のオブジェクト群は階層的に関連付けられて配置する．すなわち，Container に属する Pane オブジェクトの配下に Controls に属するオブジェクト（例えばボタン：Button オブジェクト）を追加するという形を取る．これがシーングラフである．

Scene クラスのオブジェクトは，シーングラフと背景などの属性を保持するものである．ここで重要なポイントとして，**1 つの Scene オブジェクトには，シーングラフの頂点となる 1 つのオブジェクト（Container など）があり，その配下に階層的に GUI オブジェクトが関連付けられている**ということがある．

例えば，GUI の台紙となる **AnchorPane クラス**のオブジェクトを生成し，それを頂点とする Scene クラスのオブジェクトを生成し，最後にそれを Stage クラスのオブジェクトに登録して GUI が一応完成する．あとは，生成した AnchorPane オブジェクトの配下に Controls カテゴリのオブジェクト（Button オブジェクトなど）を次々と生成して追加する．

ボタンを 1 つ設置して，これをクリックすると標準出力に ”Hello World!” と表示するプログラムを次に示す．

**サンプルプログラム：**FXsample01.java

```
1  import javafx.application.*;
2  import javafx.event.*;
3  import javafx.scene.*;
4  import javafx.scene.control.*;
5  import javafx.scene.layout.*;
6  import javafx.stage.*;
7
8  public class FXsample01 extends Application {
9      @Override
10     public void start(Stage stg) {
11         AnchorPane root = new AnchorPane();
12         Scene scene = new Scene(root, 300, 100);
13         stg.setTitle("Hello World!");
14         stg.setScene(scene);
15
16         Button btn = new Button();
17         btn.setText("Say 'Hello World'");
18         btn.setOnAction(new EventHandler<ActionEvent>() {
19             @Override
20             public void handle(ActionEvent event) {
21                 System.out.println("Hello World!");
22             }
23         });
24         root.getChildren().add(btn);
25         btn.relocate(80,30);
26
27         stg.show();
```

```
28      }
29
30      public static void main(String[] args) {
31          launch(args);
32      }
33  }
```

**解説：**

- 10 行目：　このアプリの最初の Stage オブジェクトを stg として受け取る
- 11 行目：　AnchorPane クラスのオブジェクト root を生成
- 12 行目：　それを元に，300×100 のサイズの Scene クラスのオブジェクト scene を生成
- 14 行目：　scene を stg に登録
  setScene メソッドを使用する．
- 16 行目：　Button クラスのオブジェクト btn を生成
- 18〜23 行目：　ボタンのクリックを受けてメッセージを表示するイベント処理を記述
  setOnAction メソッドを使用して，生成した EventHandler を登録している．
  handle メソッドに処理を記述してオーバーライドする．
- 24 行目：　btn を root の配下に登録
  getChildren メソッドと add メソッドを使用する．
- 25 行目：　btn の配置を設定
  relocate メソッドを使用する．
- 27 行目：　stg の表示を実行
  show メソッドを使用する．

このプログラムをコンパイルして実行すると次のような表示となる．


図 2: FXsample01 を実行したところ

**【まとめ】**

- ウィンドウは Stage オブジェクト上に Scene オブジェクトを登録することで作成される．
- ウィンドウの内容は Scene オブジェクト上のシーングラフとして構築される．
- シーングラフの最上位の Node[1] を Scene オブジェクトに登録する．

## 2.3　ウィンドウサイズの変更を禁止する設定

一般的にはアプリケーションのウィンドウサイズはユーザが変更できるということが前提になっているが，これを禁止することも可能である．ユーザによるウィンドウサイズの変更を禁止するには，当該ウィンドウの Stage オブジェクトに対して setResizable メソッドを次のようにして実行する．

**ユーザによるウィンドウサイズ変更を禁止する設定**

**実行例：**　stg.setResizable(false);　　（Stage stg のリサイズ禁止）

[1]Node は javafx.scene.Node ライブラリとして提供されるクラスであり，シーングラフを構成する要素となるオブジェクトのクラス群を収める上位クラスである．(付録 11.4 参照)

## 2.4 GUI の可視属性の設定

Containers や Controls などの GUI オブジェクトには**可視属性**があり，その設定によって表示／非表示を制御することができる．

**GUI の可視属性の設定**

**実行例：** o1.setVisible(false);　　（o1 オブジェクトを非表示に設定）
引数に true を与えると o1 が表示される．

## 2.5 複数のウィンドウを生成する方法

アプリケーションには最初にシステムから与えられた Stage オブジェクトがあり，GUI を構成したコンテナオブジェクトを持つ Scene オブジェクトは，この最初の Stage オブジェクトに登録する．この状態では 1 つのウィンドウのみが存在する形となる．

複数のウィンドウを持つアプリケーションを作成する場合は，第 2，第 3 のウィンドウを次々と生成することになるが，それら毎にコンテナオブジェクトと Scene オブジェクトを生成する必要があり，更にそれらの Scene オブジェクト毎に別々の Stage オブジェクトを生成する必要がある．

複数の Stage オブジェクトの間には階層的な関係があり，最初にシステムから与えられた Stage オブジェクト（親ステージ）の配下に，後から生成された Stage オブジェクト（子ステージ）を登録することになる．(図 3 参照)

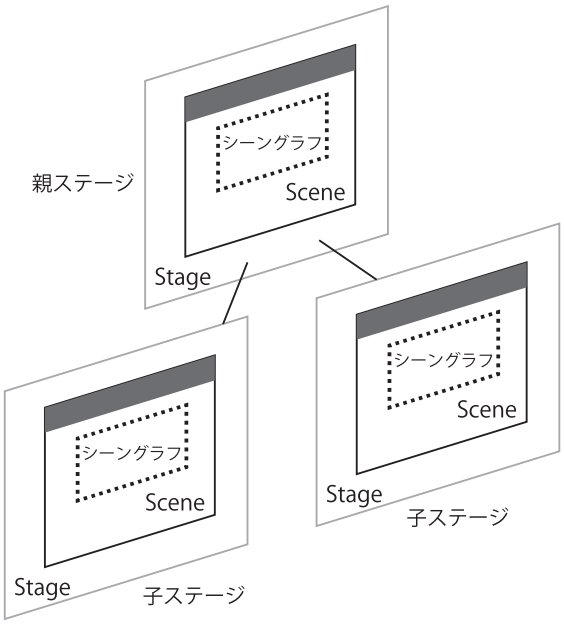



図 3: 複数ウィンドウの構築

2 つ目のウィンドウを生成する方法について，サンプルプログラム FXsample02.java を示しながら説明する．

**サンプルプログラム：**FXsample02.java

```
1 import javafx.application.*;
2 import javafx.event.*;
3 import javafx.scene.*;
4 import javafx.scene.control.*;
5 import javafx.scene.layout.*;
6 import javafx.stage.*;
7
8 public class FXsample02 extends Application {
9     @Override
```

```
10      public void start(Stage stg) {
11
12          // 最初のウィンドウ
13          AnchorPane root = new AnchorPane();
14          Scene scene = new Scene(root, 300, 100);
15          stg.setTitle("Hello World!");
16          stg.setScene(scene);
17
18          Button btn = new Button();
19          btn.setText("Say 'Hello World'");
20          btn.setOnAction(new EventHandler<ActionEvent>() {
21              @Override
22              public void handle(ActionEvent event) {
23                  System.out.println("Hello World!");
24              }
25          });
26          root.getChildren().add(btn);
27          btn.relocate(80,30);
28
29          stg.show();
30
31          // 2番目のウィンドウ
32          Stage stg2 = new Stage();
33          AnchorPane child = new AnchorPane();
34          Scene scene2 = new Scene(child, 300, 100);
35          stg2.setTitle("Second Window");
36          stg2.setScene(scene2);
37
38          stg2.initOwner(stg);
39          stg2.show();
40      }
41
42      public static void main(String[] args) {
43          launch(args);
44      }
45  }
```

**【解説】**

● **32〜36 行目：** 2つ目のウィンドウを生成している．

● **38 行目：** 2つ目のウィンドウのステージを最初のウィンドウのステージに接続している．

このプログラムでは，新たに生成した 2 つ目の Stage オブジェクト stg2 を，既存の Stage オブジェクト stg の配下に登録している．

**Stage オブジェクトの上下関係の設定**

**実行例：** stg2.initOwner(stg);
新たに生成した stg2 を既存の stg の配下に登録する．

このプログラムを実行した様子を図 4 に示す．
実行した最初の時点では，2 つのウィンドウは重なって表示されるが，適宜移動することができる．

## 2.6 アプリケーションの終了に関する処理

アプリケーションを意図的に終了させるには Platform クラスのクラスメソッド exit を実行する．例えば，多くのアプリケーションプログラムでは,「File」メニューから「Quit（終了）」を選択するとアプリケーションの実行が終了するが，これは，アプリケーションプログラム自体を終了させるメソッドを呼び出すという形で実現されている．

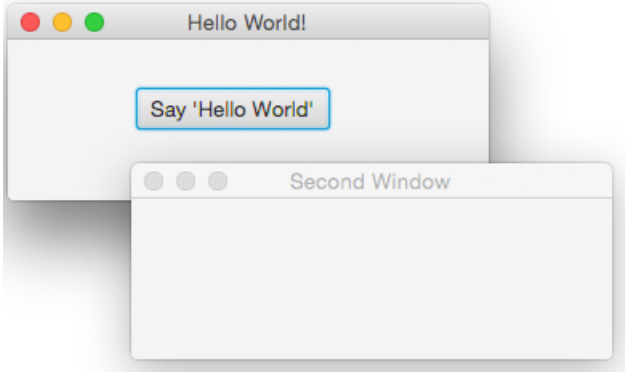



図 4: FXsample02 を実行したところ

**アプリケーションプログラムを終了する方法**

**実行例：** Platform.exit();

### 2.6.1 アプリケーション終了時に呼び出されるメソッド

Platform.exit() の呼び出しなどでアプリケーションプログラムが終了する際，終了の直前に stop メソッドが呼び出される．アプリケーションの終了直前に行う処理は，この stop メソッドをオーバーライドして記述するとよい．

**stop メソッドの記述**

```
@Override
public void stop() throws Exception {
        （ここにアプリケーション終了直前に行う処理を記述する）
}
```

# 3 三次元グラフィックス

ここでは，三次元グラフィックスの作成について説明する．Java FX における三次元グラフィックスは円柱，直方体，球，メッシュなどを基本オブジェクトとして構成する．また三次元グラフィックスの構成の流れも，GUI 構築のそれとほぼ同じである．すなわち，Scene の中で階層的にオブジェクトを追加してゆく**シーングラフ**の形をとっている．

まずはじめに，三次元グラフィックス作成に必要な基礎事項について説明する．

## 3.1 基礎事項

**【三次元座標系】**

Java FX における三次元座標系は図 5 のようなもの（左手座標系）である．これは三次元の幾何学で一般的に採用されている座標系と同じであるが，コンピュータの一般的な二次元グラフィックスの座標系との親和性を実現するため，Z 軸を手前から奥に向けての座標軸にしている．

**【グラフィクス作成のための基本的なクラス】**

Java FX では，Group クラスのオブジェクトが三次元オブジェクトを束ねるものであり，生成した三次元のオブジェクトはこれの配下に登録してゆく．Group クラスは，先の GUI 構築の解説のところで取り上げた Pane に相当するものと考えると良い．このクラスのオブジェクトに対して表 1 にある Box や Cylinder などのオブジェクトを生成して登録してゆく．

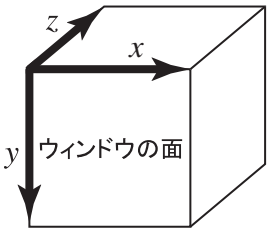



図 5: Java FX における三次元座標系

表 1: 三次元グラフィックス作成のためのクラス

| クラス | 役割／目的 |
|---|---|
| Group | シーングラフの頂点 |
| Box | 直方体 |
| Cylinder | 円柱 |
| Sphere | 球 |
| MeshView | メッシュ |
| PhongMaterial | 三次元オブジェクト表面の材質 |
| AmbientLight | 環境光源 |
| PointLight | 点光源 |
| ParallelCamera | カメラ（直投影型） |

三次元グラフィックスのオブジェクトは，光源からの光を反射して，その反射光をカメラが撮影するという形で可視化する．Java FX では 2 種類の光源を用いる．1 つは**環境光源**，もう 1 つは**点光源**である．

環境光源は方向性の無い光源であり，この光源からの光は，三次元オブジェクトに満遍なく照射される．これに対して点光源は，光源の位置が明確に定められるものであり，点光源からは光が放射状に照射されて三次元オブジェクトを照らす．

**【材質とテクスチャ】**

三次元オブジェクトの表面には**材質**を設定する．材質を設定するオブジェクトとして PhongMaterial クラスがあり，このクラスのオブジェクトには**拡散反射光**の色や**鏡面反射光**の色を設定したり，あるいはビットマップ画像（**テクスチャ**）を貼り付けることができる．

**【カメラ】**

**カメラ**は生成した三次元オブジェクトを撮影し，その映像をウィンドウに表示する．カメラは定められた位置から三次元オブジェクトを撮影する．

## 3.2 直方体，円柱，球

直方体，円柱，球を表示するサンプルプログラム FX3Dsample01.java を示しながら，三次元グラフィックス作成の流れを説明する．

**サンプルプログラム：**FX3Dsample01.java

```
1 import javafx.application.*;
2 import javafx.scene.*;
3 import javafx.scene.layout.*;
4 import javafx.scene.paint.*;
5 import javafx.stage.*;
6 import javafx.scene.shape.*;
7 import javafx.scene.transform.*;
```

```
 8 import javafx.geometry.*;
 9
10 public class FX3Dsample01 extends Application {
11
12     @Override
13     public void start(Stage stg) {
14         //--- Top Node and Scene ---
15         Group root = new Group();
16         Scene scene = new Scene(root, 1024, 768, Color.rgb(0,0,0));
17
18         // Axis for Rotation
19         Point3D aX = new Point3D(100,0,0);
20         Point3D aY = new Point3D(0,100,0);
21         Point3D aZ = new Point3D(0,0,100);
22
23         //--- Solid Model Generation ---
24         // (Box)
25         Box bx1 = new Box(300d,200d,150d);
26         root.getChildren().add(bx1);
27         PhongMaterial mt1 = new PhongMaterial();
28         mt1.setDiffuseColor(Color.rgb(150,0,0));
29         mt1.setSpecularColor(Color.rgb(255,0,0));
30         mt1.setSpecularPower(5d);
31         bx1.setMaterial(mt1);
32         bx1.getTransforms().addAll(
33             new Translate(-300d,0d,0d), // 平行移動
34             new Rotate(30,aX), // X軸回りの回転
35             new Rotate(30,aY), // Y軸回りの回転
36             new Rotate(20,aZ)  // Z軸回りの回転
37         );
38         // (Cylinder)
39         Cylinder cl1 = new Cylinder(80d,300d);
40         root.getChildren().add(cl1);
41         PhongMaterial mt2 = new PhongMaterial();
42         mt2.setDiffuseColor(Color.rgb(0,150,0));
43         mt2.setSpecularColor(Color.rgb(0,255,0));
44         mt2.setSpecularPower(10d);
45         cl1.setMaterial(mt2);
46         cl1.getTransforms().addAll(
47             new Rotate(30,aX),
48             new Rotate(-20,aZ)
49         );
50         // (Sphere)
51         Sphere sp1 = new Sphere(140d);
52         root.getChildren().add(sp1);
53         PhongMaterial mt3 = new PhongMaterial();
54         mt3.setDiffuseColor(Color.rgb(0,0,150));
55         mt3.setSpecularColor(Color.rgb(0,0,255));
56         mt3.setSpecularPower(5d);
57         sp1.setMaterial(mt3);
58         sp1.getTransforms().addAll(
59             new Translate(280d,0d,0d)
60         );
61
62         //--- Light Setting ---
63         AmbientLight aLight = new AmbientLight(Color.rgb(127, 127, 127));
64         root.getChildren().add(aLight);
65
66         PointLight pLight = new PointLight(Color.rgb(255,255,255));
67         pLight.setTranslateX(500d);
68         pLight.setTranslateY(-300d);
69         pLight.setTranslateZ(-200d);
70         root.getChildren().add(pLight);
71
72         //--- Camera Setting ---
73         ParallelCamera cmr = new ParallelCamera();
74         cmr.getTransforms().addAll(
75             new Translate(-512d,-384d,0d)
76         );
```

```
77          scene.setCamera(cmr);
78
79          //--- Window Activation ---
80          stg.setTitle("FX3Dsample01");
81          stg.setScene(scene);
82          stg.show();
83      }
84
85      public static void main(String[] args) {
86          launch(args);
87      }
88  }
```

このプログラムを実行すると図 6 のように表示される．

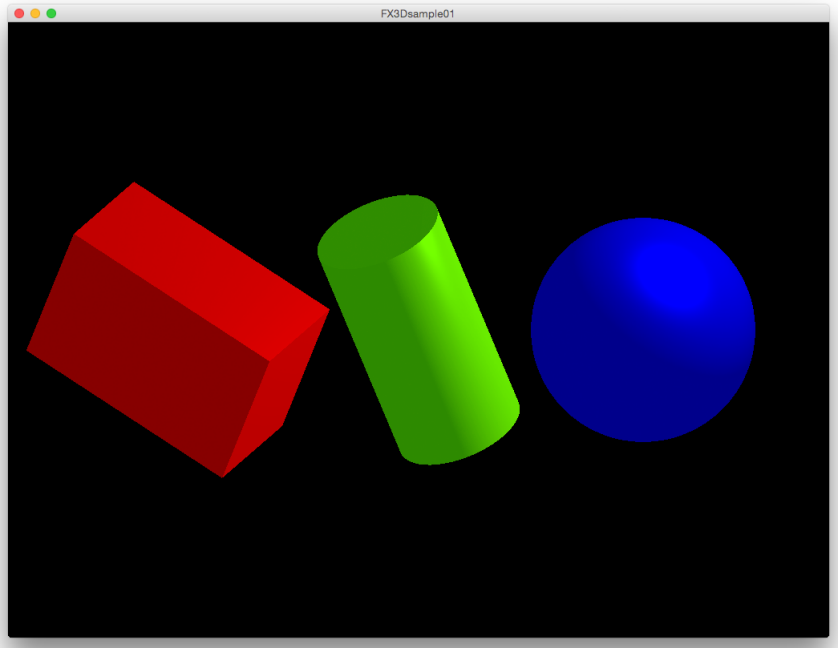



図 6: FX3Dsample01 を実行したところ

次に，このサンプルプログラムについて解説する．

### 3.2.1 シーングラフの準備

15 行目でシーングラフの頂点となる Group クラスのオブジェクト root を生成している．

```
Group root = new Group();
```

これを Scene オブジェクトに登録（16 行目）する．

### 3.2.2 Color クラスによる色の指定

今回のプログラムでは 16 行目のような記述により，Scene オブジェクトの生成時に「塗り色」（fill）を指定して背景色を設定している．

```
Scene scene = new Scene(root, 1024, 768, Color.rgb(0,0,0));
```

Scene のコンストラクタの最後の引数として与えているものが色の指定である．これは Color クラスを利用したものであり，静的メソッド rgb により色の情報を与えている．

**Color クラスによる色の指定**

**書式：** Color.rgb(int R, int G, int B)
赤，緑，青の値を 0〜255 の範囲の整数値で指定する．

今後も色の指定を行う様々な局面でこの方法を用いる．

### 3.2.3 基本的な三次元オブジェクトの扱い

**直方体**は **Box** クラスクラスのオブジェクトとして，次のようにして生成する．

**直方体の生成**

**実行例：** Box bx1 = new Box(W,H,D); （直方体 bx1 が生成される）
double W ： x 方向のサイズ（幅）
double H ： y 方向のサイズ（高さ）
double D ： z 方向のサイズ（奥行き）

**円柱**は **Cylinder** クラスクラスのオブジェクトとして，次のようにして生成する．

**円柱の生成**

**実行例：** Cylinder cl1 = new Cylinder(R,H); （円柱 cl1 が生成される）
double R ： 半径
double H ： y 方向のサイズ（高さ）

**球**は **Sphere** クラスクラスのオブジェクトとして，次のようにして生成する．

**球の生成**

**実行例：** Sphere sp1 = new Sphere(R); （球 sp1 が生成される）
double R ： 半径

**材質**は **PhongMaterial** クラスクラスのオブジェクトとして，次のようにして生成し，各種設定を行う．

**材質の生成と設定**

**実行例：** PhongMaterial mt1 = new PhongMaterial(); （材質 mt1 が生成される）
mt1.setDiffuseColor(Color.rgb(150,0,0)); ： 拡散反射光の設定（暗い赤）
mt1.setSpecularColor(Color.rgb(255,0,0)); ： 鏡面反射光の設定（明るい赤）
mt1.setSpecularPower(5d); ： 鏡面反射の強さ設定（艶の強さ）
bx1.setMaterial(mt1); ： 直方体 bx1 に材質 mt1 を設定

このように，材質オブジェクトを生成して反射光などの設定を行い， setMaterial メソッドによって，三次元オブジェクトの表面に与える．

### 3.2.4 平行移動，回転，拡縮

三次元オブジェクトに対する**平行移動**と**回転**は，次のように **getTransforms** メソッドと，**addAll** メソッドを用いて行うことができる．

**平行移動と回転の設定**

**実行例：** bx1.getTransforms().addAll( ... ); （bx1 を平行移動や回転する）

**解説：** addAll の引数には，変形処理を表す次のようなオブジェクトを生成して与える．

new Translate(-300d,0d,0d) ： 各座標の移動量を指定して平行移動

new Rotate(30,aX) ： 回転（軸ベクトル aX 回りに 30 度）

new Scale(2d,3d,4d) ： 拡縮（X,Y,Z 軸方向にそれぞれ 2 倍,3 倍,4 倍）

この例では回転軸として三次元ベクトル（**Point3D** クラスのオブジェクト） aX が用いられているが，これは次のようにして生成する．

**三次元成分（ベクトル）を保持するオブジェクト**

**実行例：** Point3D aX = new Point3D(100,0,0); （ (100,0,0) を保持する aX）

ベクトル成分を保持する場合に用いる．

シーングラフの各種ノードに対する平行移動と回転，拡縮は表 2 のようなメソッドを用いて設定することもできる．

表 2: Node オブジェクトに対する平行移動，回転，拡縮

| メソッド | 処理 |
|---|---|
| setTranslateX(double Tr) | X 方向に Tr 並行移動 |
| setTranslateY(double Tr) | Y 方向に Tr 並行移動 |
| setTranslateZ(double Tr) | Z 方向に Tr 並行移動 |
| setRotationAxis(Point3D A) | 回転軸をベクトル A に設定 |
| setRotate(double R) | 回転軸回りに R 回転 |
| setScaleX(double M) | X 方向に M 倍 |
| setScaleY(double M) | Y 方向に M 倍 |
| setScaleZ(double M) | Z 方向に M 倍 |

### 3.2.5 光源

環境光源は **AmbientLight** クラスのオブジェクトであり，次のようにして生成する．

**環境光源**

**実行例：** AmbientLight aLight = new AmbientLight(Color.rgb(127, 127, 127));

（RGB 成分 (127,127,127) の環境光源 aLight を生成）

三次元オブジェクトと同じ Group に登録することで光を照らす．

点光源は，それが配置された位置から光を放射する光源であり，**PointLight** クラスのオブジェクトである．次のようにして生成して設定する．

**点光源**

**実行例：** PointLight pLight = new PointLight(Color.rgb(255,255,255));
（RGB 成分 (255,255,255) の点光源 pLight を生成）
三次元オブジェクトと同じ Group に登録することで光を照らす.
pLight.setTranslateX(500d); pLight を X 方向に 500 移動する.
pLight.setTranslateY(-300d); pLight を Y 方向に-300 移動する.
pLight.setTranslateZ(-200d); pLight を Z 方向に-200 移動する.

### 3.2.6 直投影型カメラ

ここでは直投影型のカメラである **ParallelCamera** クラスのオブジェクトの生成と設定について説明する.

**カメラ（直投影型）**

**実行例：** ParallelCamera cmr = new ParallelCamera(); （カメラ cmr を生成）
三次元オブジェクトと同じ Scene に登録することで撮影する.
cmr.getTransforms().addAll( new Translate(-512d,-384d,0d) ); カメラ位置を (-512,-384,0) に移動する.
scene.setCamera(cmr); Scene へのカメラの登録

このように，シーンに対して setCamera メソッドを実行することでカメラが設置される.

### 3.2.7 透視投影型カメラ

先に，直投影型のカメラとして ParallelCamera について説明したが，このカメラでは遠近法による投影ができない．実際の立体的視野では遠くのものは小さく，近くのものは大きく見える，いわゆる遠近法による投影となる．遠近法の投影を実現する**透視投影型カメラ**も使用することができる.

基本的には，ParallelCamera の代わりに PerspectiveCamera を使えば良い.

サンプルプログラムのカメラを PerspectiveCamera に変えて実行した様子を図 7 に示す.

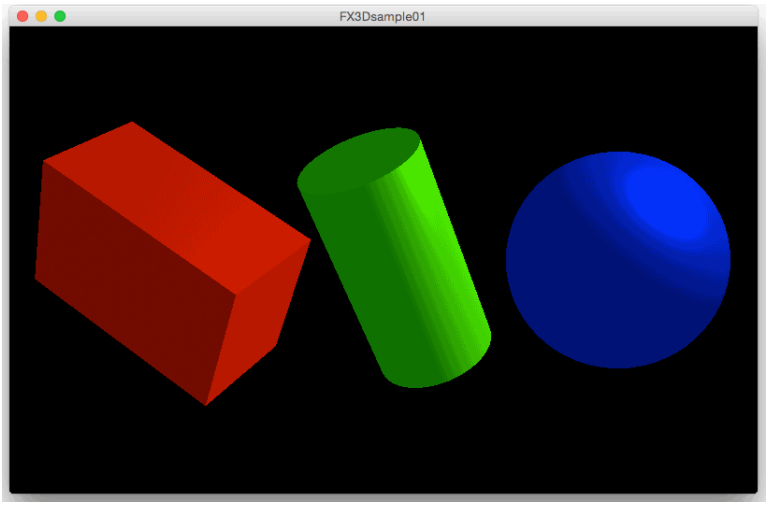



図 7: 実行したところ

## 3.3 メッシュ・グラフィックス

前述の解説では直方体，円柱，球の生成を取り扱ったが，実際の CG 制作では，より複雑なオブジェクトを表示することが求められる．ここでは**メッシュ・グラフィクス**を用いて複雑な形状のオブジェクトを生成する方法について

説明する．

メッシュ・グラフィックスは**三角形の集合体として立体を表現する**手法である．すなわち，**複数の三角形の頂点をつなげて多面体を作り上げる**という方法で三次元のオブジェクトを表現する．

メッシュ・グラフィックスを理解する上で最初に重要なのが「メッシュの裏表」である．三次元オブジェクトの表面には先に説明した材質やテクスチャを設定して，光源の光を照射することで可視化する．このため，材質やテクスチャを設定するための面を表面として決める必要があり，三角形のメッシュを作成した場合も，どちらの面を表とするかを決定しなければならない．

三角形のメッシュを定義するためには，3 つの頂点の座標を与える．このとき頂点の座標を与える順番で，メッシュの表面を決定する．例えばメッシュ生成時に，図 8 のように 1 〜 3 の順番で頂点の座標データを与えると，図中の法線の向きが表面となる．

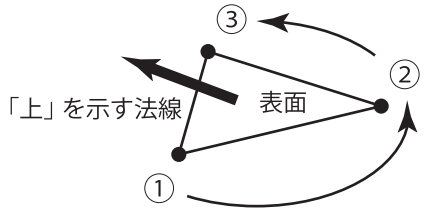



図 8: 三角形メッシュの「表面」

このような順序で頂点を与えて法線を決定するモデルは，図 9 のような「右ねじ」であり，右手を使って表現することもある．

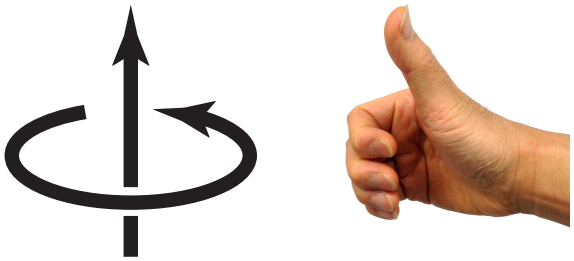


図 9: 右ねじの表現

三角形以外の多角形をポリゴンとして組み合わせて三次元オブジェクトを作成することも多い．三角形以外の多角形を作成する場合も，基本的には三角形のメッシュを組み合わせる．例えば，四角形を作成する場合は，図 10 のように，右ねじモデルで作成した 2 つの三角形メッシュの頂点を合わせる方法を取る．

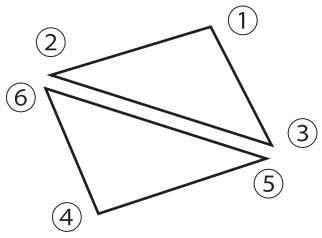



図 10: 三角形メッシュで作る四角形

次に，サンプルプログラムを示しながらメッシュによる三次元オブジェクトの生成について説明する．

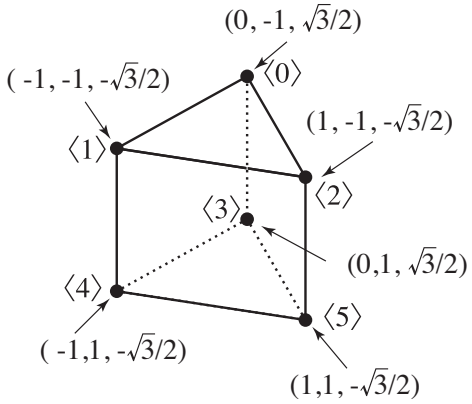



図 11: 生成する三角柱

**【三角柱の生成】**

図 11 のような三角柱の生成について考える.(図 11 の〈1〉～〈5〉は，三角柱の頂点の番号である)
三角形のメッシュの集まりとして三角柱を生成するが，側面の長方形は三角形の組み合わせで実現する．更にできあがった三角柱の表面に，図 12 のような画像をテクスチャとして貼り付けることを考える．

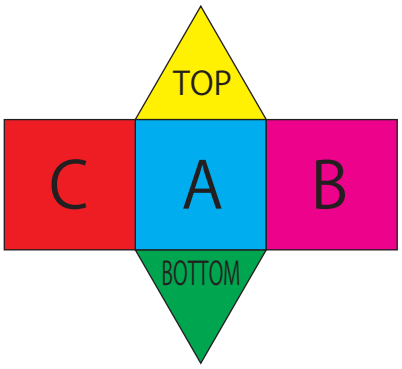



図 12: 使用するテクスチャ

### 3.3.1 メッシュへのテクスチャの貼付け

画像をテクスチャとしてメッシュに貼り付ける場合，構成する三次元オブジェクトの展開図を**正規化された座標系**（ (0,0)～(1,1) の座標範囲 ）にマッピングし，その正規化された座標系に画像を合わせて貼り付けるという方法を取る．例えば図 11 の三角柱の展開図を正規化された座標系にマッピングすると，図 13 のようなものとなる．
図 13 中の [1]～[9] は，図 11 の三角柱の頂点が対応する点である．

**【メッシュ・グラフィックス作成の手順】**

メッシュ・グラフィックス作成に必要な各種の座標データなどを保持するオブジェクトのクラスとして TriangleMesh がある．すなわち，このクラスのオブジェクトに，メッシュを構成する三角形の頂点の座標や，テクスチャのマッピングに必要な正規化された座標のデータなどを設定する．

メッシュでオブジェクトを構成する場合，頂点の座標を設定するだけでは不十分である．すなわち，与えられた座標データを**どの順番で三角形の頂点にするかという設定**が必要となる．更に，メッシュオブジェクトの頂点の座標と，マッピングのための正規化座標系との対応も与える必要があり，それらも TriangleMesh クラスのオブジェクトに設定する．

TriangleMesh クラスのオブジェクトに必要なデータを設定した後，このオブジェクトを元に MeshView クラスのオブジェクトを生成する．この MeshView クラスのオブジェクトが表示すべきグラフィックのオブジェクトとなり，こ

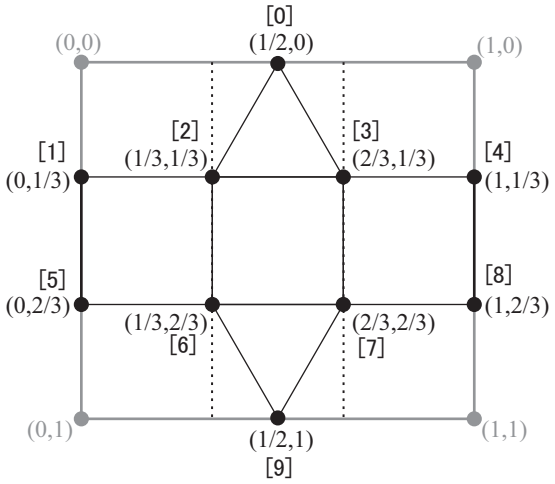



図 13: 展開図の正規化座標系へのマッピング

れを Group オブジェクトに追加することで可視化する.

次にサンプルプログラム FX3Dsample03.java を示し，メッシュ・グラフィックスの作成について説明する.

**サンプルプログラム：** FX3Dsample03.java

```
1  import javafx.application.*;
2  import javafx.scene.*;
3  import javafx.scene.layout.*;
4  import javafx.scene.paint.*;
5  import javafx.scene.shape.*;
6  import javafx.scene.image.*;
7  import javafx.scene.transform.*;
8  import javafx.stage.*;
9  import javafx.geometry.*;
10
11 public class FX3Dsample03 extends Application {
12
13     // sqrt(3)/2
14     public static float R32 = 0.8660f;
15     // 1/3, 2/3
16     public static float T31 = 0.3333f,
17                         T32 = 0.6667f;
18
19     public Image img;
20
21     @Override
22     public void start(Stage Stage) {
23
24         img = new Image("file:texture.gif");
25
26         //--- Top Node and Scene ---
27         Group root = new Group();
28         Scene scene = new Scene(root, 500, 400, Color.BLACK);
29
30         // Axis for Rotation
31         Point3D aX = new Point3D(100,0,0);
32         Point3D aY = new Point3D(0,100,0);
33         Point3D aZ = new Point3D(0,0,100);
34
35         //--- Mesh Model Generation ---
36         TriangleMesh tm1 = new TriangleMesh();
37         tm1.getPoints().addAll(
38              0f,    -1f,     R32,   // <0>
39             -1f,    -1f,    -R32,   // <1>
40              1f,    -1f,    -R32,   // <2>
41              0f,     1f,     R32,   // <3>
42             -1f,     1f,    -R32,   // <4>
```

```
             1f,     1f,    -R32    // <5>
        );
        tm1.getTexCoords().addAll(
            0.5f,   0f,     // [0]
            0f,     T31,    // [1]
            T31,    T31,    // [2]
            T32,    T31,    // [3]
            1f,     T31,    // [4]
            0f,     T32,    // [5]
            T31,    T32,    // [6]
            T32,    T32,    // [7]
            1f,     T32,    // [8]
            0.5f,   1f      // [9]
        );
        tm1.getFaces().addAll(
            0,0, 1,2, 2,3,  // <0>[0], <1>[2], <2>[3]
            5,7, 4,6, 3,9,  // <5>[7], <4>[6], <3>[9]
            2,3, 1,2, 4,6,  // <2>[3], <1>[2], <4>[6]
            2,3, 4,6, 5,7,  // <2>[3], <4>[6], <5>[7]
            1,2, 0,1, 4,6,  // <1>[2], <0>[1], <4>[6]
            0,1, 3,5, 4,6,  // <0>[1], <3>[5], <4>[6]
            3,8, 0,4, 2,3,  // <3>[8], <0>[4], <2>[3]
            5,7, 3,8, 2,3   // <5>[7], <3>[8], <2>[3]
        );
        MeshView mv1 = new MeshView(tm1);
        root.getChildren().add(mv1);
        PhongMaterial mt1 = new PhongMaterial();
        mt1.setDiffuseMap(img);
        mt1.setDiffuseColor(Color.rgb(255,255,255));
        mt1.setSpecularColor(Color.rgb(255,255,255));
        mt1.setSpecularPower(5d);
        mv1.setMaterial(mt1);
        mv1.getTransforms().addAll(
            new Scale(100d,100d,100d),
            new Rotate(20d,aX),
            new Rotate(50d,aY),
            new Rotate(0d,aZ)
        );

        //--- Light Setting ---
        AmbientLight aLight = new AmbientLight(Color.rgb(127, 127, 127));
        root.getChildren().add(aLight);

        PointLight pLight = new PointLight(Color.rgb(255,255,255));
        pLight.setTranslateX(600d);
        pLight.setTranslateY(-600d);
        pLight.setTranslateZ(-400d);
        root.getChildren().add(pLight);

        //--- Camera Setting ---
        ParallelCamera cmr = new ParallelCamera();
        cmr.getTransforms().addAll(
            new Translate(-250d,-200d,-150d)
        );
        scene.setCamera(cmr);

        //--- Window Activation ---
        Stage.setTitle("FX3Dsample01");
        Stage.setScene(scene);
        Stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

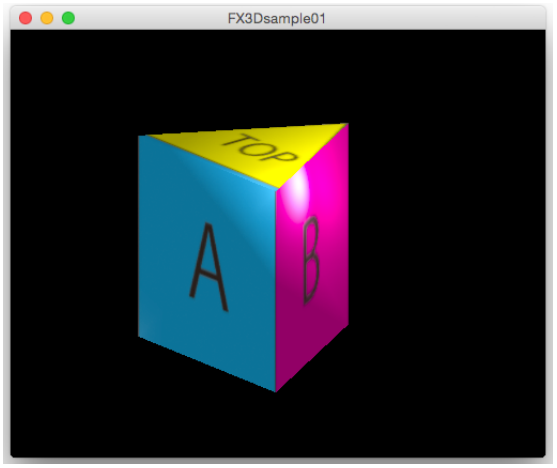



図 14: 実行したところ

**解説：**

座標データの中に $\frac{\sqrt{3}}{2}$，$\frac{1}{3}$，$\frac{2}{3}$ といった数値が多く見られるが，簡便のため，これらの数値を変数 R32，T31，T32 に予め格納（14 行目〜17 行目）している．

メッシュ・グラフィックス作成に必要な各種の座標データを保持するのは **TriangleMesh** クラスのオブジェクトであり，次のようにして生成する．

**TriangleMesh オブジェクトの生成**

**実行例：** TriangleMesh tm1 = new TriangleMesh(); （tm1 としてオブジェクトが生成される）

このオブジェクトに頂点の座標を設定するには次のように getPoints メソッドと addAll メソッドを使用する．

**メッシュ・オブジェクトの頂点座標データの設定**

**実行例：** tm1.getPoints().addAll( ... ); （tm1 に頂点の座標データを設定する）
'...' の部分に座標データを X,Y,Z の順で繰り返し記述する．

更に tm1 オブジェクトには，テクスチャマッピングのための正規化された座標系のデータも設定する．これには getTexCoords メソッドと addAll メソッドを使用する．

**テクスチャマッピングのための正規化座標データの設定**

**実行例：** tm1.getTexCoords().addAll( ... );
（tm1 にテクスチャマッピングのための正規化座標データを設定する）
'...' の部分に座標データを X,Y の順で繰り返し記述する．

そして tm1 オブジェクトに，メッシュオブジェクトの頂点の座標と，正規化された座標系の対応を設定することで，三次元オブジェクトとしての体裁が整う．このためには getFaces メソッドと addAll メソッドを使用する．

**頂点座標とテクスチャマッピング用正規化座標の対応の設定**

**実行例：** tm1.getFaces().addAll( ... );
（メッシュオブジェクトの頂点座標とマッピング用正規化座標を対応させる）
'...' の部分に対応する点の番号を対にして**右ねじの順番**で記述する．

テクスチャとして使用する画像は Image クラスのオブジェクト [2] として扱う．

**画像ファイルの読込み**

**実行例：** Image img = new Image("file:texture.gif");
ファイル texture.gif を読み込んで img オブジェクトに与える．

用意した Image オブジェクトをグラフィックオブジェクトに貼り付けるには，まず材質のオブジェクトに setDiffuseMap メソッドを用いて Image オブジェクトを設定し，それをグラフィックオブジェクトの材質として与える．

**テクスチャの貼り付け**

**実行例：** mt1.setDiffuseMap(img);
mv1.setMaterial(mt1);
材質 mt1 に Image オブジェクト img を設定し，それをグラフィックオブジェクト mv1 に与える．

# 4 FXML を利用した GUI 構築

Java FX のアプリケーションを作成する場合，GUI を **FXML** と呼ばれる拡張された XML で記述して構築することができる．これにより，CSS[3] や JavaScript[4] を併用した表現力の高い GUI を実現することができる．ここでは，GUI を構築するための FXML 文書と Java FX プログラムの連携に関する基本的な部分についてサンプルプログラムを示しながら説明する．

**【FXML コンテンツの作成について】**

実際には，テキストエディタなどを用いてプログラマが直接 FXML 文を記述するのは開発効率の面からも現実的ではなく，**Java FX Scene Builder**（「4.3 Java FX Scene Builder」 参照）などのオーサリングツールを用いて GUI を構築するのが一般的である．従って，本書では FXML の詳細に関する解説はせず，理解に必要な最低限の事柄に関して説明する．

## 4.1 サンプルプログラム

ここでは，FXML で記述した FXMLsample01.fxml を読み込んで GUI を実現する Java FX アプリケーション FXMLsample01.java を示しながら説明する．

**FXML ソース：** FXMLsample01.fxml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import java.lang.*?>
4  <?import java.util.*?>
5  <?import javafx.scene.*?>
6  <?import javafx.scene.control.*?>
7  <?import javafx.scene.layout.*?>
8
9  <AnchorPane id="ap1" prefHeight="200" prefWidth="320"
10   xmlns:fx="http://javafx.com/fxml/1" fx:controller="FXMLsample01">
11     <children>
12         <Button layoutX="126" layoutY="90"
13             text="Click Me!" fx:id="btn1" onAction="#onAction_btn1" />
14         <Label layoutX="126" layoutY="120"
15             minHeight="16" minWidth="69" fx:id="lb1" />
16     </children>
```

[2]「5.4 画像データの扱い」参照のこと．
[3]**Cascading Style Sheets：** HTML コンテンツ（Web コンテンツ）をはじめとする XML コンテンツを表示する際の体裁（スタイル）を設定するための記述．
[4]Web ブラウザをはじめとする XML コンテンツのブラウザを制御するプログラミング言語．

```
17 </AnchorPane>
```

これは FXML で GUI を構築した例である．

**1 行目：** 使用する XML のバージョンと，多国語のエンコーディング（文字コード体系）を指定してる．
ここでは，UTF-8 によるエンコーディングを指定している．

**3〜7 行目：** GUI の実装に必要となる Java のライブラリを指定している．
?import タグで必要なライブラリを指定する．

**9〜17 行目：** GUI のシーングラフの記述．各ノードとなるオブジェクトのクラス名をタグとし，それらタグを入れ子にして記述してノードの階層構造を実現する．この例では，AnchorPane クラスのオブジェクトをシーンの頂点とし，その配下に Button クラスのオブジェクトと Label クラスのオブジェクトが 1 つずつ存在する構成となる．ノードに子ノードを登録するには children タグを用いる．

**【FXML 文書と Java プログラムとの連携：1】**

FXMLsample01.fxml で作られるシーングラフと GUI の概観を図 15 に示す．


図 15: FXML で構築した GUI

ここで記述した GUI は Java のプログラムに読み込んで実装する．このため，FXML で記述した GUI やそれらに対するイベント処理を Java のプログラムとして記述することが必要となる．これを行う Java 側のクラス（Application クラスの拡張クラス）の名前を指定する部分が，サンプル中の 10 行目にある

fx:controller="FXMLsample01"

である．このように，シーングラフの頂点となるタグの “fx:controller” 属性の値として Java 側のアプリケーションのクラス名を与えることで，FXML と Java のシーングラフの頂点が対応する．また，各ノードのタグの “fx:id” 属性の値には，そのノードの Java プログラム側でのインスタンス名を指定する．例えば，サンプル中の 13 行目に見られる

fx:id="btn1"

という記述は，Java 側の Button クラスのインスタンス btn1 との対応を実現する．

ノードに発生したイベントに対応する処理（イベントハンドリング）を実現するには，イベントハンドリングのための起点となるイベントを属性名とし，その値として Java 側のメソッド名を与える．例えば，サンプルの 13 行目に

onAction="#onAction_btn1"

という記述があるが，これは「ボタン btn1 がクリックされた場合にイベントハンドラ onAction_btn1 メソッドが起動する」ことを意味する．

Java FX 8 での重要なイベント（属性名）を「11.5 イベントについて」に挙げる．

この FXML を読み込んで使用する Java のプログラム FXMLsample01.java を次に示す．

**Java ソース：** FXMLsample01.java

```
1 import java.net.*;
2 import java.util.*;
3 import javafx.application.*;
4 import javafx.event.*;
5 import javafx.fxml.*;
6 import javafx.scene.*;
7 import javafx.scene.control.*;
8 import javafx.stage.*;
```

```
 9
10 public class FXMLsample01 extends Application
11   implements Initializable {
12
13     //---------------------------------------
14     // ＦＸＭＬドキュメント内で宣言されたオブジェクト
15     //---------------------------------------
16     @FXML   // ラベル
17     Label lb1;
18     @FXML   // ボタン
19     Button btn1;
20
21     //---------------------------------------
22     // イベントハンドリング
23     //---------------------------------------
24     @FXML // ボタンがクリックされたときの処理
25     void onAction_btn1(ActionEvent event) {
26         lb1.setText("ボタンがクリックされました. ");
27         System.out.println("標準出力へのメッセージ");
28     }
29
30     //---------------------------------------
31     // ＧＵＩ実装後の初期化処理
32     //---------------------------------------
33     @Override
34     public void initialize(URL url, ResourceBundle rb) {
35         // 期化処理
36     }
37
38     //---------------------------------------
39     // ＦＸＭＬドキュメントの読込みとＧＵＩの実装
40     //---------------------------------------
41     @Override
42     public void start(Stage stage) throws Exception {
43
44         //--- ＦＸＭＬの読込み ---
45         Parent root = FXMLLoader.load(
46             getClass().getResource("FXMLDocument.fxml"));
47
48         //--- ＧＵＩの作成と表示 ---
49         Scene scene = new Scene(root);
50         stage.setScene(scene);
51         stage.show();
52     }
53
54     //--- メイン ---
55     public static void main(String[] args) {
56         launch(args);
57     }
58 }
```

**解説：**

このプログラムは Application クラスの拡張クラス FXMLsample01 として定義されているが，FXML を扱うためには Initializable インターフェースの機能を引き継ぐ必要があるのでこれをインプリメント（11 行目）している．

**45-46 行目：** シーングラフの頂点ノードの読み込み．

プログラム中に FXMLLoader クラスのクラスメソッド load を実行する記述があるが，これが FXML を読み込んでいる部分である．これにより FXML 文書 FXMLsample01.fxml に記述されている AnchorPane クラスのオブジェクト ap1 が，Java 側の Parent クラスのインスタンス root として生成されて対応する．

GUI を構築する FXML 文書は，それを使用するアプリケーションのリソースとして含めることが一般的である．（「4.2.2 単独で動作するアプリケーションの生成について」を参照のこと）
アプリケーションのリソースとして用意されている FXML 文書を読み込むには，このサンプルのように getClass メソッドと getResource メソッドを用いる．

**【FXML 文書と Java プログラムとの連携：2】**

プログラム中に GUI のコントロールを宣言している部分として，

```
Label lb1;          （17 行目）
Button btn1;        （19 行目）
```

があるが，FXML で記述したものと対応させるために**@FXML アノテーション**を直前に記述する必要がある．同様に onAction_btn1 メソッドも，FXML と対応させるために同様のアノテーションを付けている．

このプログラムを実行した様子を図 16 に示す．


実行直後　　ボタンをクリックしたところ

図 16: プログラムを実行したところ

実際のアプリケーション開発では，作業の効率を上げるために，以下に説明する **NetBeans IDE** といった統合開発環境や，**Java FX Scene Builder** といった GUI 構築のためのオーサリングツールを使用するのが一般的である．

## 4.2　NetBeans IDE の利用

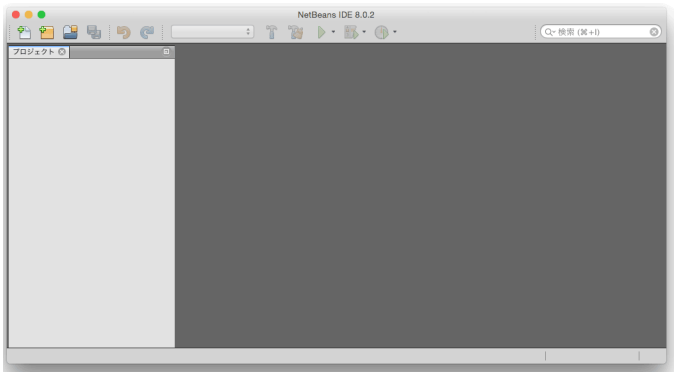
**NetBeans IDE** は同名の団体が配布する統合開発環境であり，フリーソフトウェアとして配布されている．
（ https://netbeans.org/　本書を執筆している時点（2015 年 8 月）での版は 8.0.2 である）
このツールに加えて，後述の Java FX Scene Builder を併用することで，GUI アプリケーションの開発効率を大幅に高めることができる．

ここでは FXML で GUI を実装する JavaFX アプリケーションの開発を例に挙げ，NetBeans IDE の操作の流れについて説明する．

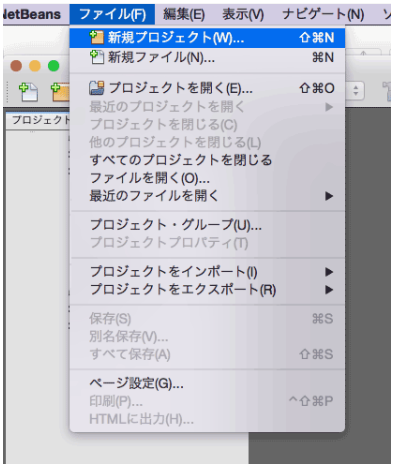
NetBeans IDE を起動すると図 17 の左のようなウィンドウが表示される．開発するアプリケーションは**プロジェクト**と呼ばれる単位で管理され，1 つのプロジェクトの下で必要となるリソース（Java のソースプログラム，FXML ドキュメントをはじめとする素材群）が管理される．
新規にプロジェクトを作成する場合は，図 17 の右のように「ファイル (F)」メニューから「新規プロジェクト (W)」を選択する．すると次に図 18 のようなウィンドウが表示される．ここでは構築するアプリケーションの種類を選択する．

以降では「Java FX FXML アプリケーション」を開発するものとして説明する．このウィンドウでそれを選択し，「次」ボタンをクリックすると図 19 のようなウィンドウが表示される．
ここでは「プロジェクト名 (N)」の項目にプロジェクトの名前を入力（サンプル中では “FXMLappl001” を入力）して「終了 (F)」ボタンをクリックする．次に図 20 のようなウィンドウが表示される．
図 20 のウィンドウでは，アプリケーション開発の雛形となるソースプログラムが自動的に作成され，それらがウィンドウ内のタブにあらわれているのがわかる．

ここで自動生成されたソースプログラムは以下のようなものである．




NetBeans IDE を起動した所




新規プロジェクトの作成

図 17: NetBeans IDE

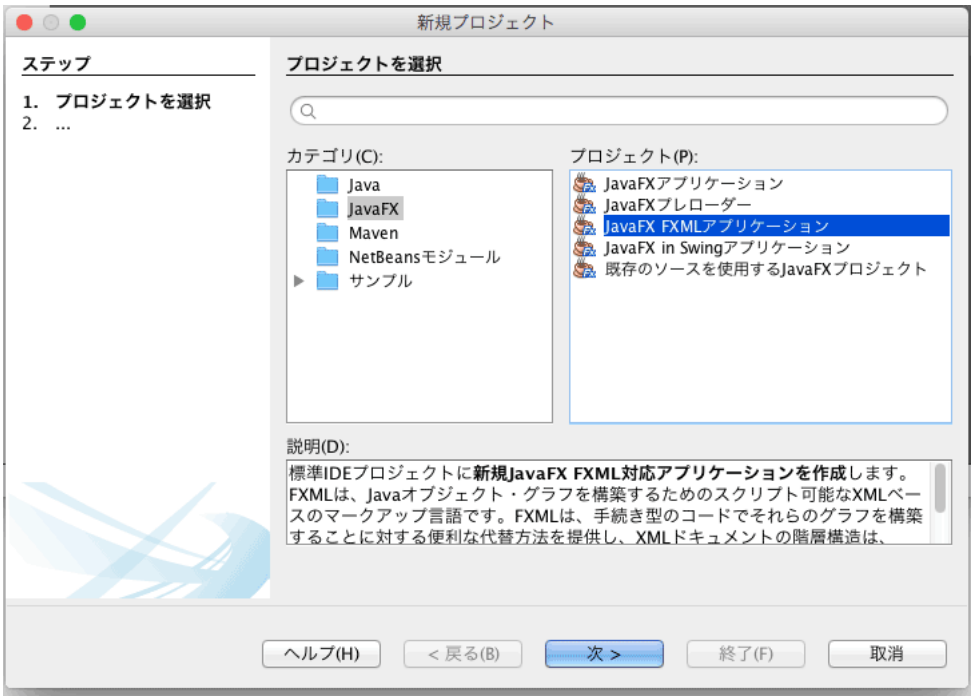



図 18: 「JavaFX」の「JavaFX FXML アプリケーション」を選択

**雛形ソース (1)：main メソッドを含むプログラム**　FXMLAppl001.java

```
1  package fxmlappl001;
2
3  import javafx.application.Application;
4  import javafx.fxml.FXMLLoader;
5  import javafx.scene.Parent;
6  import javafx.scene.Scene;
7  import javafx.stage.Stage;
8
9  public class FXMLAppl001 extends Application {
10
11     @Override
12     public void start(Stage stage) throws Exception {
13         Parent root = FXMLLoader.load(getClass().getResource("FXMLDocument.fxml"));
14
15         Scene scene = new Scene(root);
16
17         stage.setScene(scene);
```

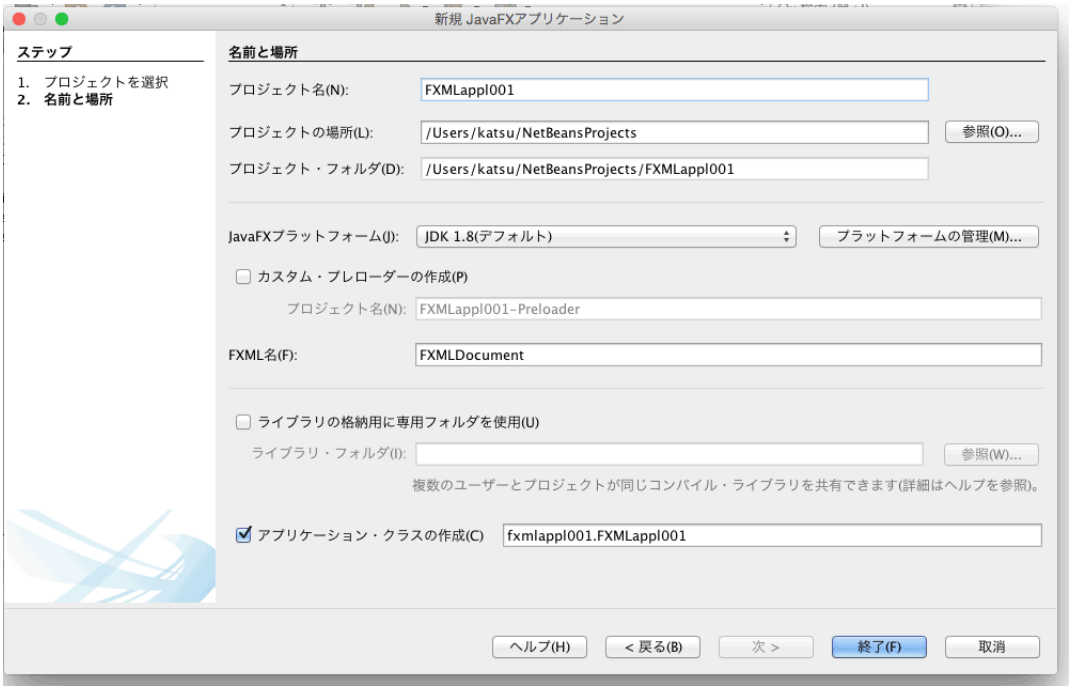



図 19: プロジェクト作成のための各種設定

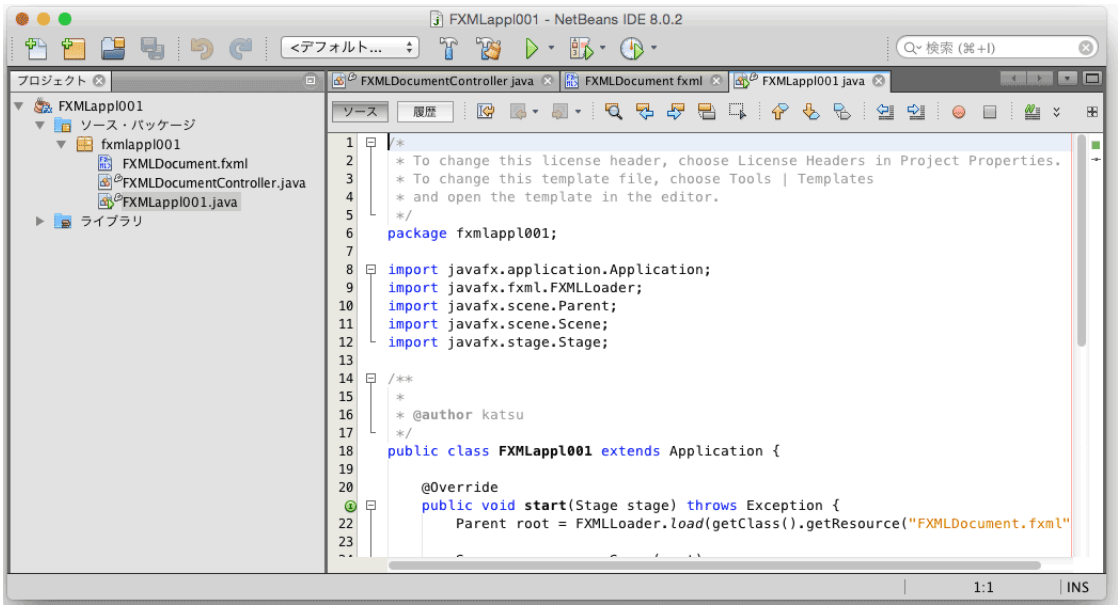



図 20: プロジェクトの雛形が用意される

```
18          stage.show();
19      }
20
21      public static void main(String[] args) {
22          launch(args);
23      }
24
25  }
```

**雛形ソース (2)：FXML で構築した GUI を扱うプログラム**　FXMLDocumentController.java

```
1  package fxmlappl001;
2
3  import java.net.URL;
4  import java.util.ResourceBundle;
5  import javafx.event.ActionEvent;
```

```
6  import javafx.fxml.FXML;
7  import javafx.fxml.Initializable;
8  import javafx.scene.control.Label;
9
10 public class FXMLDocumentController implements Initializable {
11
12     @FXML
13     private Label label;
14
15     @FXML
16     private void handleButtonAction(ActionEvent event) {
17         System.out.println("You clicked me!");
18         label.setText("Hello World!");
19     }
20
21     @Override
22     public void initialize(URL url, ResourceBundle rb) {
23         // TODO
24     }
25 }
```

**雛形ソース (3)：FXML ドキュメント**　FXMLDocument.fxml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import java.lang.*?>
4  <?import java.util.*?>
5  <?import javafx.scene.*?>
6  <?import javafx.scene.control.*?>
7  <?import javafx.scene.layout.*?>
8
9  <AnchorPane id="ap1" prefHeight="180" prefWidth="320"
10   xmlns:fx="http://javafx.com/fxml/1"
11   fx:controller="FXMLsample01">
12     <children>
13         <Button layoutX="80" layoutY="70" minWidth="170" fx:id="btn1"
14             text="Click Me!" onAction="#onAction_btn1" />
15         <Label layoutX="80" layoutY="100" minWidth="170" fx:id="lb1" />
16     </children>
17 </AnchorPane>
```

この段階で，ボタンを 1 つ備えたアプリケーションが既に完成しており，ボタンをクリックするとウィンドウ内のラベルに文字を表示し，標準出力にメッセージを出力するものになっている．

IDE のウィンドウ上部に図 21 のようなボタンがあり，これをクリックするとソースプログラムがコンパイルされてアプリケーションが実行される．

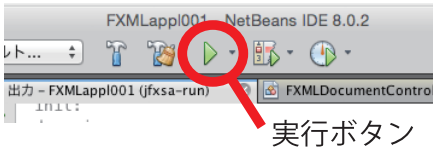



図 21: 実行ボタン

実行ボタンをクリックしてアプリケーションを実行したところを図 22 に示す．

### 4.2.1　生成された雛形を利用したアプリケーション開発

生成された雛形のプログラムを編集して拡張する形でアプリケーションを開発する．先の例では，次のような 3 つの雛形が生成されている．

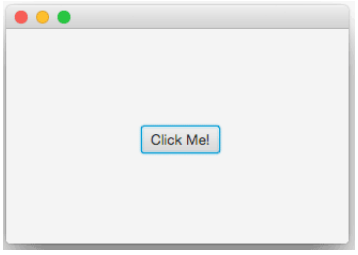



図 22: サンプルアプリの実行

**雛形ソース (1)：main メソッドを含むプログラム**　FXMLAppl001.java
**雛形ソース (2)：FXML で構築した GUI を扱うプログラム**　FXMLDocumentController.java
**雛形ソース (3)：FXML ドキュメント**　FXMLDocument.fxml

NetBeans IDE では，JavaFX FXML アプリケーションの雛形を自動生成する際，Java のソースプログラムを 2 つのファイルに分けている．1 つは，Application クラスの拡張クラスとしてアプリケーションの主たるプログラムとなるファイルで，上記**「雛形ソース (1)」**の FXMLAppl001.java がそれである．2 つ目は，Initializable インターフェースの機能を実装して FXML 文書との連携を実現するプログラムとなるファイルで，上記**「雛形ソース (2)」**の FXMLDocumentController.java がそれである．

GUI の構築とイベントハンドリングの実装は，上記**「雛形ソース (2)」**の Java プログラムと，上記**「雛形ソース (3)」**の FXML 文書を編集することで行う．FXML 文書の内容を直接編集することもできるが，後述する Java FX Scene Builder を用いて編集することで GUI のオーサリングを直観的に行うことができる．

### 4.2.2　単独で動作するアプリケーションの生成について

NetBeans IDE でアプリケーションを開発すると，当該プロジェクトのディレクトリに「dist」という名前のサブディレクトリができる．アプリケーションが正しくビルドされると，dist ディレクトリの中に，“プロジェクト名.jar” という名前を持つファイルができる．これは，JVM（Java 仮想マシン）上で単独のアプリケーションとして起動するものである．

この “プロジェクト名.jar” は，コンパイル済みのプログラム（中間コード）だけでなく，実行時に必要となる各種リソース（アプリケーションが実行時に使用する素材データ）も含んでおり，単独で配布することができる．アプリケーションの実行時にリソースとして使用するデータは，当該プロジェクトのソースプログラムを配置するディレクトリ「src」の中に配置しておく．

FXML ドキュメントを編集して，高度な GUI を構築するに当たって，次に説明する Java FX Scene Builder を利用すると，GUI 構築の作業の効率を大幅に高めることができる．

## 4.3　Java FX Scene Builder

Java FX Scene Builder は Java の開発と管理をしている ORACLE（http://www.oracle.com/）が配布するフリーの GUI オーサリングツールである．本書を執筆している時点（2015 年 7 月）での Java FX Scene Builder の版は 2.0 である．Java FX Scene Builder は ORACLE の Web サイトからダウンロードできる．

Java FX Scene Builder を起動すると図 23 のようなウィンドウが表示される．

このウィンドウの中央部に構築すべき GUI のウィンドウ（コンテナ）が表示されている．また左右には「アコーディオン」と呼ばれる上下に見え隠れするインターフェースがある．ウィンドウ左上にある「Library」には GUI の各種の部品があり，ユーザはここから所望する GUI 部品を選択して中央部のコンテナで GUI を構築する．

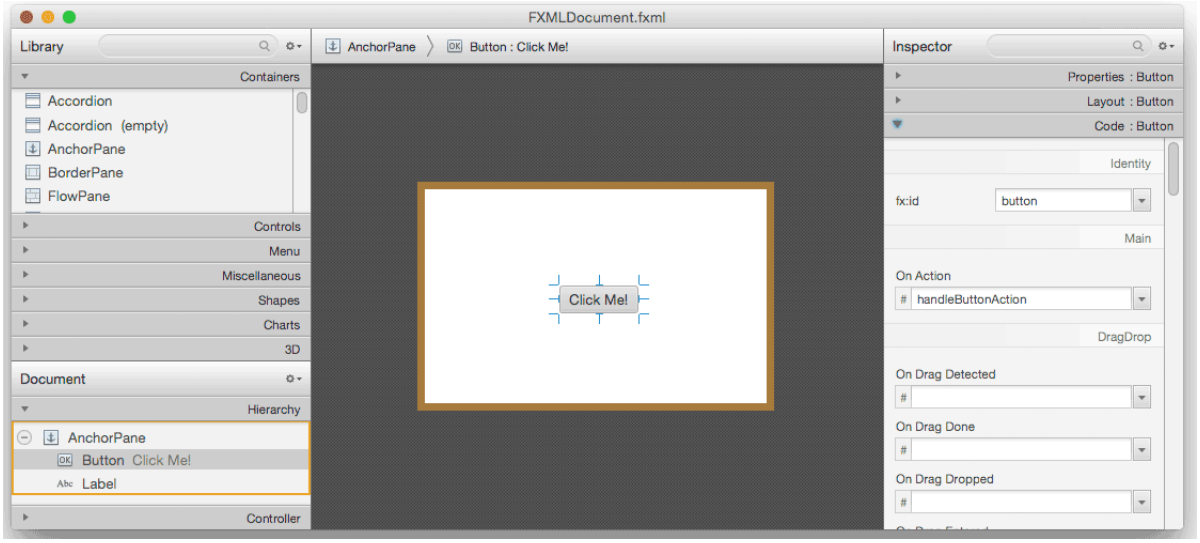



図 23: Java FX Scene Builder（全体）

ウィンドウ左下の「Document」には，構築されている GUI の階層構造が表示されており，この部分を通して GUI 部品の選択が可能である．ウィンドウ右にある「Inspector」には，選択した GUI に関する属性の閲覧と設定ができる．


図 24: Java FX Scene Builder（各部）

Java FX Scene Builder で作成した GUI を保存すると，その GUI の内容を反映した FXML ドキュメントができる．でき上がった FXML ドキュメントの最上位のコンテナのタグには fx:controller="クラス名" を追加して，GUI の制御に対応する Java のクラス名を指定する．

### 4.3.1 インスペクタ

インスペクタについて若干説明する．

**【プロパティ：Properties】**

Properties 内では，選択したオブジェクトの基本的な属性を設定する．CSS に関する設定もここで行う．

**【レイアウト：Layout】**

Layout 内では，サイズや配置などの属性を設定する．

**【コード：Code】**

Code 内では，Java との連携に関する設定を行う．作成する FXML ドキュメント内におけるオブジェクトと，Java 側のインスタンス名を対応（fx:id を設定）させたり，イベント発生時に呼び出すメソッド（イベントハンドラ）の指定（4.3.2 参照）などを行う．

### 4.3.2 イベントハンドラとの関連付け

先に挙げたサンプルのプロジェクト FXMLAppl001 では,「Click Me」と表示されたボタンをクリックするとメッセージが表示されるプログラムとなっている．これは FXML ドキュメントで記述したボタン（fx:id は button）と，FXMLDocumentController.java 中のメソッド handleButtonAction を関連付けることで実現している．

インスペクタの Code 内にある OnAction に入力された handleButtonAction がボタンとイベントハンドラを対応付けている．(図 25 参照）もちろん，fx:id に入力されている button は FXMLDocumentController.java 中で Button のインスタンス button として宣言されている必要があり，このとき@FXML アノテーションを付けることが必要である．

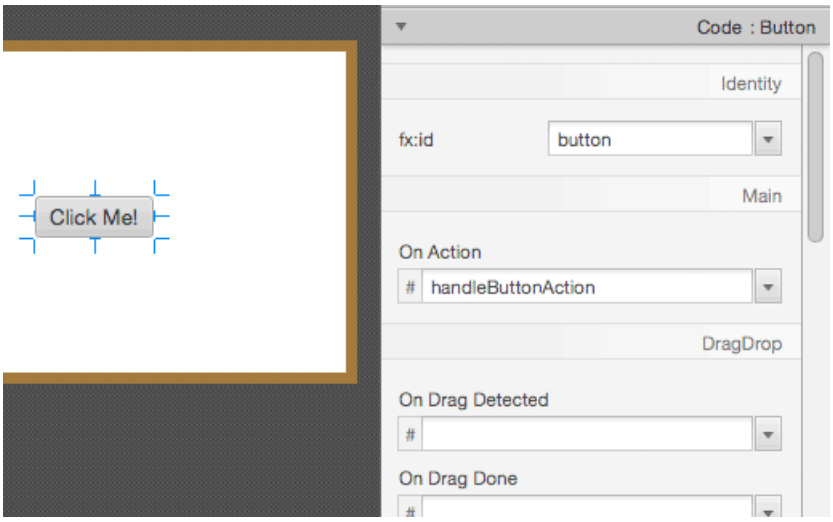



図 25: ボタンとイベントハンドラの関連付け

### 4.3.3 NetBeans IDE との連携

Java FX Scene Builder は NetBeans IDE と連携して使用することができる．NetBeans IDE のウィンドウ内の左側にプロジェクトの階層を表示することができる.(図 26）ここで，FXML 文書をダブルクリックすると Java FX Scene Builder が起動し，その FXML 文書の内容が読み込まれて表示される．Java FX Scene Builder で編集して保存した FXML 文書の内容は，直ちに NetBeans IDE 側にも反映される

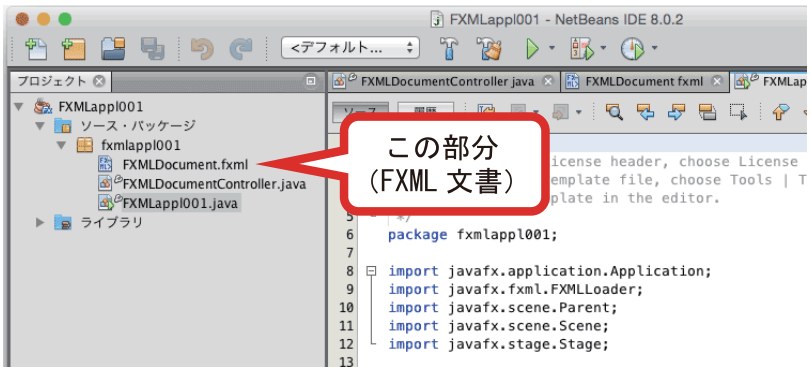



図 26: NetBeans IDE から Java FX Scene Builder を起動する

# 5 図形の描画

## 5.1 Shape を利用した描画

ここでは，シーンのノードとして追加する（描画する）二次元の図形の中でも基本的なものについて説明する．

**Shape クラス**

Shape クラス配下には基本的な図形を描画するための各種のクラスがある．それらクラスのオブジェクトを生成して，各種 Pane クラスのオブジェクトなど Node オブジェクトに登録する（getChildren メソッド，add メソッドなどを使用）ことで図形を表示する．

### 5.1.1 線の描画： Line / Polyline

**■ 線分： Line クラス**

コンストラクタ： Line(X1,Y1,X2,Y2)

開始位置の座標 (X1,Y1)，終了位置の座標 (X2,Y2) として線分を生成する．座標成分の値は double 型で与える．

**■ 折れ線： Polyline クラス**

コンストラクタ： Polyline(Pa)

節点の座標位置を保持する double 型の配列オブジェクト Pa を与えて生成する．折れ線の開始点，節点，終点の座標位置が

(X0,Y0), (X1,Y1), (X2,Y2), … ,(Xe,Ye)

の場合，それら成分を全て並べた double 型の一次元配列をコンストラクタの引数に与える．

折れ線の節点の座標は，Polyline のインスタンスを生成した後で追加することもできる．それを行う場合は getPoints メソッドと add メソッドを使用する．例えば Polyline クラスのインスタンス L があるとき，

```
L.getPoints().add(50d);
L.getPoints().add(100d);
```

とすると，座標データ (50,100) が節点として折れ線 L の末尾に追加される．

線の太さや色，実線/点線/破線といった属性を与えるための各種のメソッドがあり，次にそれらについて説明する．

### 5.1.2 線に属性を与えるメソッド

● setStroke(線の色)

線分や折れ線などのオブジェクトに対して線の色を与える．線の色を表現するには Color クラスを用いる．

**例．** L.setStroke(Color.rgb(255,0,0))

これはオブジェクト L に線の色（赤）を与えている例である．Color クラスのメソッド rgb の引数には RGB 値を順番に整数型の値（0〜255）で与える．

● setStrokeWidth(太さ)

線分や折れ線などのオブジェクトに対して線の太さを与える．線の太さは double 型の値で与える．
（単位：ポイント）

**例．** L.setStrokeWidth(8.0)

これはオブジェクト L の線の太さを 8.0 ポイントにしている例である．

**● 破線の設定**

線に破線を設定するには，線と空白の長さの値の列を与える．

**例．** L.getStrokeDashArray().addAll(20d,5d,5d,5d)

このように getStrokeDashArray メソッドと，addAll メソッドを用いて線と空白の長さを設定する．この例では線 L は，図 27 のような一点鎖線として描かれる．


図 27: 破線の設定の例

● setStrokeLineCap(形状)

線の先端の形状（図 28）を設定する．

**例．**L.setStrokeLineCap(StrokeLineCap.SQUARE)

この例では，線 L の両端が四角い形となる．

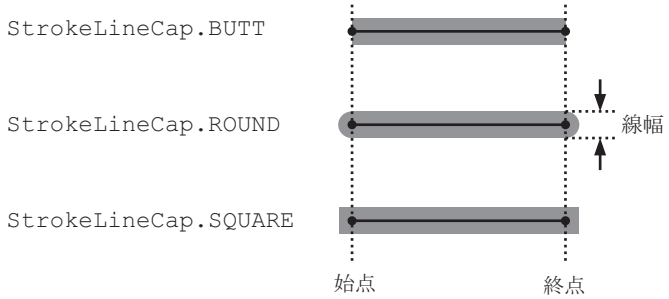



図 28: 線の先端の形状

● setStrokeLineJoin(形状)

折れ線の曲がり角の形状（図 29）を設定する．

**例．**L.setStrokeLineJoin(StrokeLineJoin.ROUND)

この例では，折れ線 L の曲がり角が丸くなる．

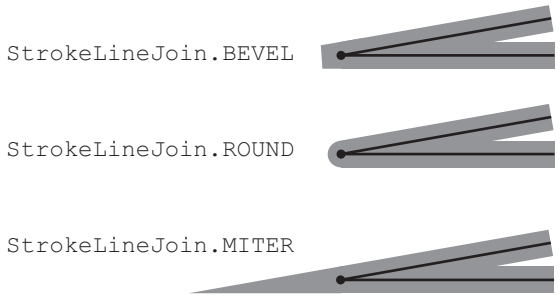



図 29: 線の曲がり角の形状

曲がり角の形状として StrokeLineJoin.MITER を指定すると，図 30 のように突出した曲がり角の長さを調整（切り落とし）することができる．どのくらい突出した場合に切り落とすかの基準は次の setStrokeMiterLimit で設定する．

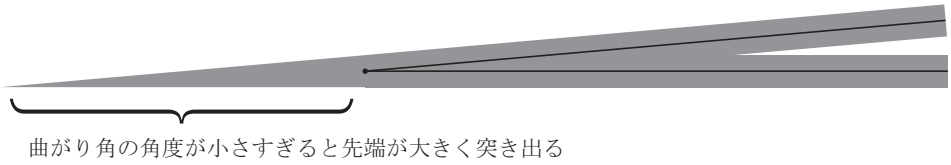



図 30: 曲がり角の突出

● setStrokeMiterLimit(限界の値)

このメソッドを実行すると，図 31 の $M/d$ が指定した「限界の値」（double 型）を超えた場合，描画時に突出部を切り落とす処理を行う．

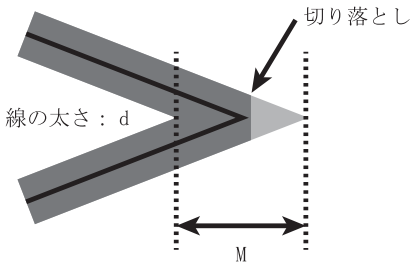



図 31: 突出部を切り落とす基準
$M/d$ を算出して切り落としを判定する

### 5.1.3 図形描画における「枠」と「塗り」

Shape を用いて図形を描画する場合，図形は「枠」と「塗り」から成ると考える.（図 32）

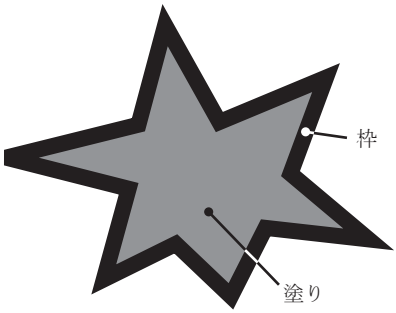



図 32: 図形描画における「枠」と「塗り」

先に説明した線分や折れ線には「枠」の部分のみ存在するが，これから説明する各種の図形には「枠」の部分に加えて「塗り」の部分が存在する．先に説明した線の属性を与えるメソッドは，以降で説明する各種の図形にも使用できる.「塗り」に関しては後の「5.1.5 塗りの属性」で説明する.

### 5.1.4 各種図形の描画： Rectangle / Circle / Ellipse / Arc / Polygon

**■ 四角形： Rectangle クラス**
コンストラクタ： Rectangle(X,Y,W,H)
座標位置 (X,Y) に幅 W, 高さ H の四角形を描画する．引数の型は全て double である.

**■ 円： Circle クラス**
コンストラクタ： Circle(X,Y,R)
中心の座標 (X,Y)，半径 R の円を描画する．引数の型は全て double である.

**■ 楕円： Ellipse クラス**
コンストラクタ： Ellipse(X,Y,Rx, Ry)
中心の座標 (X,Y)，横方向の半径 Rx，縦方向の半径 Ry の楕円を描画する．引数の型は全て double である.

**■ 弧： Arc クラス**
コンストラクタ： Arc(X,Y,Rx, Ry, As, Ar)
弧を描画する．中心の座標 (X,Y)，横方向の半径 Rx，縦方向の半径 Ry の楕円を元に，開始角度 As，描画範囲の角度 Ar で取り出した形で弧で描画する.（角度は 360 進法の度で指定）引数の型は全て double である.

弧には setType メソッドを使用して ArcType （図 34）を指定することができる．例えば Arc クラスのオブジェクト A があるとき，これを開いた弧にするには A.setType(ArcType.OPEN) とする.

**■ 多角形： Polygon クラス**
コンストラクタ： Polygon(Pa)

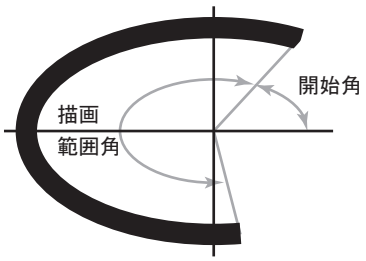



図 33: 弧の描画


図 34: 弧のタイプ

多角形を描画する．コンストラクタの引数には多角形の頂点の座標を格納する配列を与える．頂点の座標データの与え方は折れ線 Polyline の節点の座標データと同じ考え方（5.1.1）で構成すればよい．

### 5.1.5 塗りの属性

図形の塗り色を設定するには，そのオブジェクトに対して setFill メソッドを実行する．例えば図形のオブジェクト P がある場合，

```
P.setFill(Color.rgb(0, 255, 0));
```

とすると，P の塗りが緑となる．塗りの無い図形にするには，setFill メソッドの引数に null を指定して実行する．

### 5.1.6 文字の描画： Text

コンストラクタ： Text(X,Y, 文字列)

“文字列”で与えた文字列を (X,Y) の位置に描画する．例えば

```
Text tx1 = new Text(140d,450d,"This is a test text.");
```

とすると，(140,450) の座標位置に “This is a test text.” という文字列が描画される．(座標成分は double 型で与える)

#### ■ 書体，サイズ，スタイルの設定

Text のインスタンスに書体，サイズ，スタイルを設定するには，それら属性を Font クラスのオブジェクトに設定して Text オブジェクトに与える．例えば Text クラスのオブジェクト t があるとき，setFont メソッドを使用して，

```
Font fnt = new Font("Arial Bold Italic",72);
t.setFont(fnt);
```

とすると，t は Arial Bold Italic の書体でサイズが 72 ポイントの文字列となる．

#### 5.1.6.1 システムで利用できるフォントを調べる方法

Font クラスのクラスメソッド getFontNames を用いると，そのシステムで利用できるフォントの一覧を List クラスのオブジェクトとして取得することができる．利用可能なフォントの一覧を表示するプログラム ShowFontsFX.java を次に示す．

**プログラム：** ShowFontsFX.java

```
1 import java.util.*;
2 import javafx.scene.text.Font;
```

```
 3
 4 class ShowFontsFX {
 5     public static void main( String argv[] ) {
 6         int c;
 7         String fnt;
 8
 9         //  利用できるフォントのリストの取得
10         List<String> flist = Font.getFontNames();
11
12         //  利用できるフォントの数
13         int n = flist.size();
14
15         //  フォントの名前を1つずつ表示
16         for ( c = 0; c < n; c++ ) {
17             fnt = flist.get(c);
18             System.out.println(c+": \""+fnt+"\"");
19         }
20     }
21 }
```

**【解説】**

**1〜2 行目：** 必要なライブラリの読み込み.

**10 行目：** 利用できるフォントのリストの取得.

**16〜19 行目：** リストに取得されたフォント名を 1 つずつ表示している.

実行結果の例を次に示す.

《実行結果の例：ShowFontsFX.java》

```
                :
195: "Goudy Old Style"
196: "Goudy Old Style Bold"
197: "Goudy Old Style Italic"
198: "Goudy Stout"
199: "HGP 創英角 UB"
200: "HGP 創英角体"
201: "HGP 創英 EB"
202: "HGP 教科書体"
203: "HGP 明朝 B"
204: "HGP 明朝 E"
205: "HGP 行書体"
                :
```

### 5.1.7 画像の表示： ImageView

コンストラクタ： ImageView(画像オブジェクト)

引数に与えた画像オブジェクトをもつ ImageView を生成する．画像オブジェクトは Image クラス（詳しくは「5.4 画像データの扱い」で解説する）のインスタンスとして生成する．例えば，カレントディレクトリに画像ファイル Earth.jpg がある場合,

Image img = new Image("file:Earth.jpg")

とすると，その画像ファイルの内容を保持する Image オブジェクト img ができる．次に,

ImageView iv = new ImageView(img)

とすると，画像オブジェクト img を持つ ImageView オブジェクト iv が生成される.

### 5.1.8 Shape オブジェクトの位置の設定： relocate

Shape オブジェクトの位置は relocate メソッドで設定できる．例えば，図形オブジェクト S がある場合,

S.relocate(150d,200d)

とすると，S の位置が (150,200) になる．

## 5.2 Canvas を利用した描画

Canvas クラスのオブジェクトは，それ自身が描画対象である．各種の図形や文字列を描画した Canval オブジェクトが 1 つの Node である．Canvas オブジェクトは次に説明する GraphicsContext オブジェクトを持ち，このオブジェクトに対して各種の描画メソッドを実行することで図形の描画を行う．

コンストラクタ： Canvas(横幅, 高さ)
引数に与えたサイズの Canvas を生成する．(引数は double 型)

### 5.2.1 GraphicsContext

GraphicsContext は Canvas オブジェクトに含まれており，描画に先立って，対象の Canvas オブジェクトから GraphicsContext オブジェクトを取得しておく必要がある．例えば，Canvas オブジェクト cv がある場合，

GraphicsContext gc = cv.getGraphicsContext2D()

とすると，cv に対して描画するための GraphicsContext オブジェクト gc が取得できる．以後はこの gc に対して各種の描画メソッドを実行して図形を描く．

### 5.2.2 各種の描画メソッド

ここでは GraphicsContext に対して具体的に描画をするためのメソッドについて説明する．

**■ 線/折れ線**

・strokeLine(X1,Y1,X2,Y2)
座標位置 (X1,Y1) から (X2,Y2) にかけて直線を描く．引数は全て double 型である．

・strokePolyline(Ax,Ay,N)
折れ線の開始点，節点，終点の座標データを与えて描画する．Ax は横方向の座標成分を保持する double 型の 1 次元配列，Ay は縦方向の座標成分を保持する double 型の 1 次元配列である．N には描画対象の配列の要素数を int 型の数値で与える．

描画に先立って線の属性を GraphicsContext に与えておく．

**■ 線と塗りの属性の指定**

GraphicsContext に線の属性を与えるメソッドを次に挙げる．

表 3: 線の属性

| メソッド | 説明 |
|---|---|
| setStroke(Color.rgb(R,G,B)) | 線の色を RGB 値で設定する．R,G,B は全て int 型整数で 0〜255 の値 |
| setLineWidth(W) | W には線の太さ（単位：ポイント）を double 型の値で与える． |
| setLineCap(C) | C には線の両端の形状を StrokeLineCap で指定する．(5.1.2 参照) |
| setLineJoin(J) | J には折れ線の曲がり角の形状を StrokeLineJoin で指定する．(5.1.2 参照) |
| setMiterLimit(L) | L には MiterLimit を double 型の値で指定する．(5.1.2 参照) |

塗りの属性の指定には「5.1.5 塗りの属性」で説明した方法と同様のメソッド（setFill）を使用する．

**■ 四角形/楕円/弧/多角形**

・strokeRect(X,Y,W,H), fillRect(X,Y,W,H)
座標位置 (X,Y) に横幅 W 高さ H の四角形（図 35）を描く．strokeRect は「枠」のみで，fillRect は「塗り」のみ

で描く．

・strokeOval(X,Y,W,H),　fillOval(X,Y,W,H)

座標位置 (X,Y) に横幅 W 高さ H の楕円形（図 35）を描く．strokeOval は「枠」のみで，fillOval は「塗り」のみで描く．

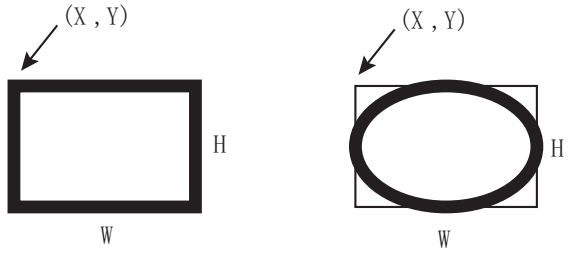



図 35: 四角形，楕円の位置とサイズ

・strokeArc(X,Y,W,H,As,Ar,Atype),　fillArc(X,Y,W,H,As,Ar,Atype)

座標位置 (X,Y) に配置された横幅 W 高さ H の楕円形を元に，開始角度 As，描画範囲の角度 Ar で取り出した形で弧を描く．位置とサイズ，角度に関する引数には double 型の値を指定する．strokeArc は「枠」のみで，fillArc は「塗り」のみで描く．ArcType には弧のタイプを指定する．(5.1.4 参照)

・strokePolygon(Ax,Ay,N),　fillPolygon(Ax,Ay,N)

頂点の座標データを与えて多角形を描画する．Ax は頂点の横方向の座標成分を保持する double 型の 1 次元配列，Ay は頂点の縦方向の座標成分を保持する double 型の 1 次元配列である．N には描画対象の配列の要素数を int 型の数値で与える．strokePolygon は「枠」のみで，fillPolygon は「塗り」のみで多角形を描画する．

**■ 文字列**

・strokeText(S,X,Y),　fillText(S,X,Y)

(X,Y) の座標位置に文字列 S を表示する．strokeText はアウトライン文字で，fillText は通常の（塗りによる）文字で表示される．

文字の描画に先立って，setFont メソッドを用いて，書体やサイズを GraphicsContext に与えておく．

・setFont(F)

F には Font クラスのオブジェクトを与える．(5.1.6 参照)

## 5.3　チャートの描画

Java FX には Chart クラスが提供されており，各種のグラフやチャートを作成するための豊富な機能が利用できる．ここでは Chart クラスの代表的なものを選んで，基本的な使用方法に限って説明する．

### 5.3.1　棒グラフ：BarChart

図 36 のような縦の棒グラフの作成を例に挙げて説明する．

**■ 棒グラフを構成するための基本的な方法**

棒グラフを作成するためのクラスに BarChart がある．棒グラフは二次元のグラフであり縦軸と横軸からなる．BarChart のインスタンスを生成する際もこれを踏まえて，それぞれの軸の型を決定する．図 36 のようにカテゴリ別にデータをプロットする場合は横軸のデータを表現するために CategoryAxis クラスのオブジェクトを，縦軸の数値を表現するために NumberAxis クラス のオブジェクトを与えて BarChart のインスタンスを生成する．

横軸を文字列（String クラス），縦軸を数値（Number クラス）とする場合は，次のようにしてインスタンスを生成する．

```
CategoryAxis xAxis = new CategoryAxis();
```

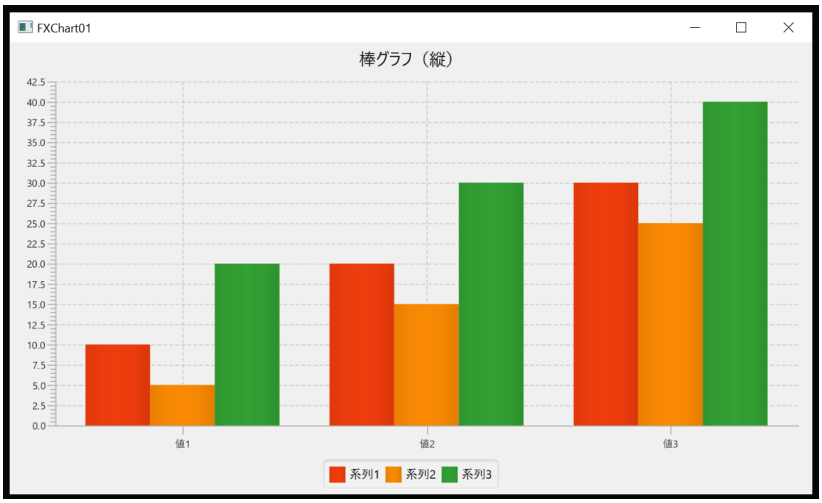



図 36: 棒グラフ（縦）の例

```
NumberAxis yAxis = new NumberAxis();
BarChart<String,Number> bc = new BarChart<>(xAxis,yAxis);
```

これにより，縦棒グラフのインスタンス bc が生成される．

グラフをプロットするためのデータは系列毎に作成して，それを BarChart のオブジェクトに与える．データ系列を実現するためのクラスに XYChart.Series クラスがあり，XYChart.Series クラスのオブジェクトにデータを与え，それを BarChart オブジェクトに与えることでグラフが描かれる．

図 36 のグラフの場合，“系列 1, 系列 2, 系列 3” というデータ系列と，“値 1, 値 2, 値 3” というデータ項目がある．例えばこの “系列 1”のデータを作成して棒グラフオブジェクト bc に与えるには次のように記述する．

```
XYChart.Series<String,Number> series1 = new XYChart.Series<>();
series1.setName("系列 1");      //  系列名の付与
series1.getData().add(new XYChart.Data<String,Number>("値 1", 10d));
series1.getData().add(new XYChart.Data<String,Number>("値 2", 20d));
series1.getData().add(new XYChart.Data<String,Number>("値 3", 30d));
bc.getData().add(series1);      //  系列データをグラフに与える
```

これにより，データ系列オブジェクト series1 が生成され，それに対してデータ項目名と値が設定され，それが bc に与えられる．

この例の中にある XYChart.Data クラスのオブジェクトが 1 つのデータを表している．データ系列オブジェクト series1 に対して，getData メソッド，add メソッドをプロットに必要な回数繰り返し使用して，系列を構成する一連の XYChart.Data オブジェクトを与える．系列名の付与には setName メソッドを使用する．

以下，同様に “系列 2”，“系列 3”も作成して bc に与える．

図 36 のグラフを作成するサンプルプログラム FXChart01.java を次に示す．

**[プログラム：FXChart01.java]**

```
1  import javafx.application.*;
2  import javafx.stage.*;
3  import javafx.scene.*;
4  import javafx.scene.layout.*;
5  import javafx.scene.chart.*;
6
7  public class FXChart01 extends Application {
8
9      @Override
10     public void start( Stage stg ) {
11         //-- ウィンドウの準備 --
12         AnchorPane root = new AnchorPane();
13         Scene sc = new Scene(root,800,450);
```

```
14          stg.setTitle("FXChart01");
15          stg.setScene(sc);
16
17          //-- 棒グラフの準備 --
18          CategoryAxis xAxis = new CategoryAxis();
19          NumberAxis yAxis = new NumberAxis();
20          BarChart<String,Number> bc = new BarChart<>(xAxis,yAxis);
21          bc.setTitle("棒グラフ（縦）");
22          bc.setPrefWidth(800d);
23          bc.setPrefHeight(450d);
24          bc.setCategoryGap(50d);
25          bc.setBarGap(0d);
26          //-- 系列1の作成 --
27          XYChart.Series<String,Number> series1 = new XYChart.Series<>();
28          series1.setName("系列1");
29          series1.getData().add(new XYChart.Data<String,Number>("値1", 10d));
30          series1.getData().add(new XYChart.Data<String,Number>("値2", 20d));
31          series1.getData().add(new XYChart.Data<String,Number>("値3", 30d));
32          bc.getData().add(series1);
33          //-- 系列2の作成 --
34          XYChart.Series<String,Number> series2 = new XYChart.Series<>();
35          series2.setName("系列2");
36          series2.getData().add(new XYChart.Data<String,Number>("値1", 5d));
37          series2.getData().add(new XYChart.Data<String,Number>("値2", 15d));
38          series2.getData().add(new XYChart.Data<String,Number>("値3", 25d));
39          bc.getData().add(series2);
40          //-- 系列3の作成 --
41          XYChart.Series<String,Number> series3 = new XYChart.Series<>();
42          series3.setName("系列3");
43          series3.getData().add(new XYChart.Data<String,Number>("値1", 20d));
44          series3.getData().add(new XYChart.Data<String,Number>("値2", 30d));
45          series3.getData().add(new XYChart.Data<String,Number>("値3", 40d));
46          bc.getData().add(series3);
47
48          //-- 表示 --
49          root.getChildren().add(bc);
50          stg.show();
51      }
52
53      public static void main( String argv[ ] ) {
54          launch(argv);
55      }
56  }
```

**参考：** 縦横の軸を逆にして，横向きの棒グラフ（図 37）を作成するサンプルプログラム FXChart02.java を挙げる．

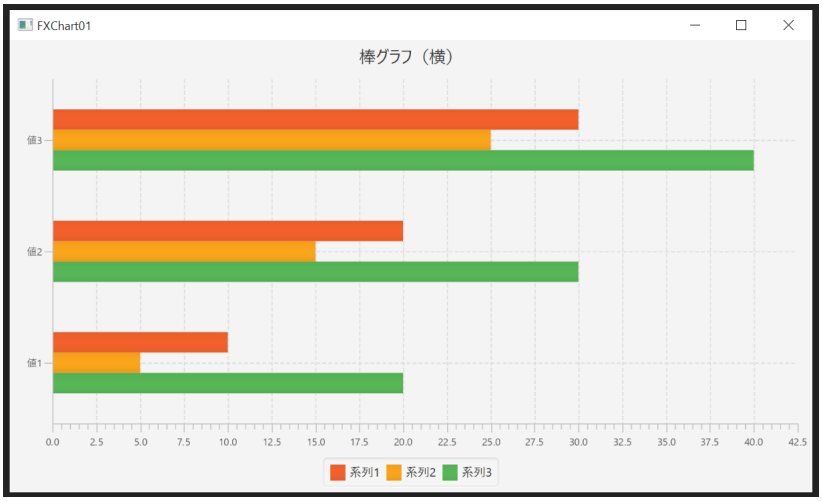



図 37: 棒グラフ（横縦）の例

[プログラム：FXChart02.java]

```
import javafx.application.*;
import javafx.stage.*;
import javafx.scene.*;
import javafx.scene.layout.*;
import javafx.scene.chart.*;

public class FXChart02 extends Application {

    @Override
    public void start( Stage stg ) {
        //-- ウィンドウの準備 --
        AnchorPane root = new AnchorPane();
        Scene sc = new Scene(root,800,450);
        stg.setTitle("FXChart02");
        stg.setScene(sc);

        //-- 棒グラフの準備 --
        NumberAxis xAxis = new NumberAxis();
        CategoryAxis yAxis = new CategoryAxis();
        BarChart<Number,String> bc = new BarChart<>(xAxis,yAxis);
        bc.setTitle("棒グラフ（横）");
        bc.setPrefWidth(800d);
        bc.setPrefHeight(450d);
        bc.setCategoryGap(50d);
        bc.setBarGap(0d);
        //-- 系列1の作成 --
        XYChart.Series<Number,String> series1 = new XYChart.Series<>();
        series1.setName("系列1");
        series1.getData().add(new XYChart.Data<Number,String>(10d,"値1"));
        series1.getData().add(new XYChart.Data<Number,String>(20d,"値2"));
        series1.getData().add(new XYChart.Data<Number,String>(30d,"値3"));
        bc.getData().add(series1);
        //-- 系列2の作成 --
        XYChart.Series<Number,String> series2 = new XYChart.Series<>();
        series2.setName("系列2");
        series2.getData().add(new XYChart.Data<Number,String>(5d,"値1"));
        series2.getData().add(new XYChart.Data<Number,String>(15d,"値2"));
        series2.getData().add(new XYChart.Data<Number,String>(25d,"値3"));
        bc.getData().add(series2);
        //-- 系列3の作成 --
        XYChart.Series<Number,String> series3 = new XYChart.Series<>();
        series3.setName("系列3");
        series3.getData().add(new XYChart.Data<Number,String>(20d,"値1"));
        series3.getData().add(new XYChart.Data<Number,String>(30d,"値2"));
        series3.getData().add(new XYChart.Data<Number,String>(40d,"値3"));
        bc.getData().add(series3);

        //-- 表示 --
        root.getChildren().add(bc);
        stg.show();
    }

    public static void main( String argv[ ] ) {
        launch(argv);
    }
}
```

### 5.3.2 折れ線グラフ：LineChart

図 38 のような折れ線グラフの作成を例に挙げて説明する．

**■ 折れ線グラフを構成するための基本的な方法**

折れ線グラフを作成するためのクラスに LineChart がある．折れ線グラフは二次元のグラフであり縦軸と横軸から

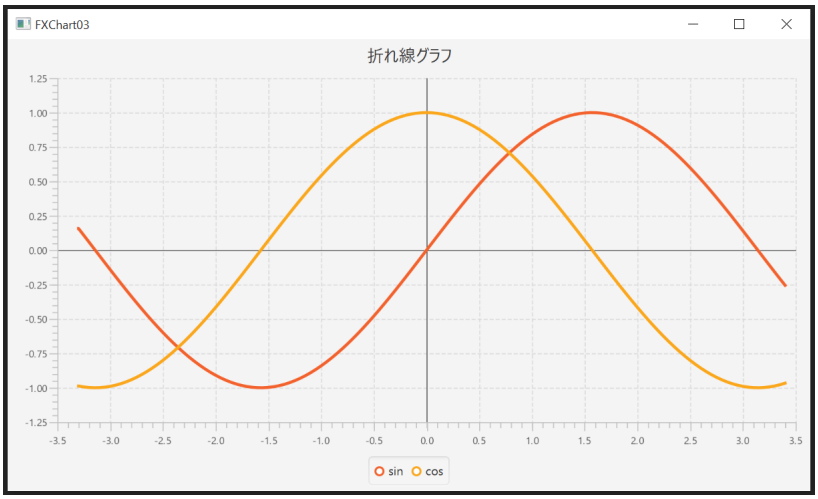



図 38: 折れ線グラフの例

なる．LineChart のインスタンスを生成する際もこれを踏まえて，それぞれの軸の型を決定する．ここでは縦横とも数値の場合を考える．

縦横とも数値なので，折れ線グラフを作成するには，次のように NumberAxis クラスのオブジェクトを両方の軸に与えて LineChart のインスタンスを生成する．

```
NumberAxis xAxis = new NumberAxis();
NumberAxis yAxis = new NumberAxis();
LineChart<Number,Number> lc = new LineChart<>(xAxis,yAxis);
```

これにより，折れ線グラフのインスタンス lc が生成される．

図 38 のグラフでは sin と cos の 2 つの系列をプロットしている．グラフをプロットするためのデータ列は系列毎に作成して，それを LineChart のオブジェクトに与える．データ系列を実現するためのクラスに XYChart.Series クラスがあり，XYChart.Series クラスのオブジェクトにデータを与え，それを LineChart オブジェクトに与えることでグラフが描かれる．基本的な考え方と方法については棒グラフ作成の場合とほぼ同じなので，そちら（5.3.1）を参照すること．

図 38 のグラフを作成するサンプルプログラム FXChart03.java を次に示す．

**[プログラム：FXChart03.java]**

```
1  import javafx.application.*;
2  import javafx.stage.*;
3  import javafx.scene.*;
4  import javafx.scene.layout.*;
5  import javafx.scene.chart.*;
6
7  public class FXChart03 extends Application {
8
9      @Override
10     public void start( Stage stg ) {
11         //-- ウィンドウの準備 --
12         AnchorPane root = new AnchorPane();
13         Scene sc = new Scene(root,800,450);
14         stg.setTitle("FXChart03");
15         stg.setScene(sc);
16
17         //-- 折れ線グラフの準備 --
18         NumberAxis xAxis = new NumberAxis();
19         NumberAxis yAxis = new NumberAxis();
20         LineChart<Number,Number> lc = new LineChart<>(xAxis,yAxis);
21         lc.setTitle("折れ線グラフ");
22         lc.setPrefWidth(800d);
23         lc.setPrefHeight(450d);
```

```
24          lc.setCreateSymbols(false);     // Dot Symbol Nullification
25          //-- 系列の作成 --
26          XYChart.Series<Number,Number> series1 = new XYChart.Series<>();
27          XYChart.Series<Number,Number> series2 = new XYChart.Series<>();
28          series1.setName("sin");
29          series2.setName("cos");
30          double x, y1, y2;
31          for ( x = -3.3; x <= 3.4; x += 0.05 ) {
32              y1 = Math.sin(x);
33              y2 = Math.cos(x);
34              series1.getData().add(new XYChart.Data<Number,Number>(x,y1));
35              series2.getData().add(new XYChart.Data<Number,Number>(x,y2));
36          }
37          lc.getData().add(series1);
38          lc.getData().add(series2);
39
40          //-- 表示 --
41          root.getChildren().add(lc);
42          stg.show();
43      }
44
45      public static void main( String argv[ ] ) {
46          launch(argv);
47      }
48  }
```

### 5.3.3 円グラフ：PieChart

図 39 のような円グラフの作成を例に挙げて説明する．

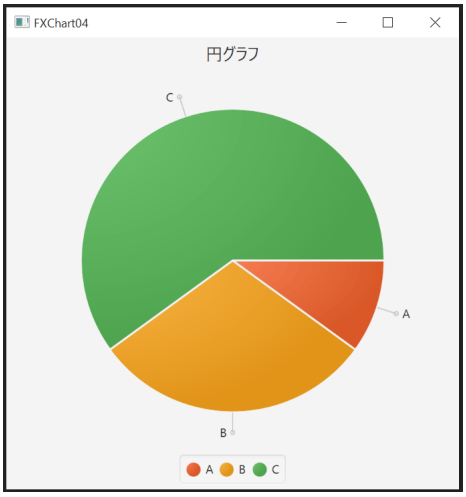



図 39: 円グラフの例

**■ 円グラフを構成するための基本的な方法**

円グラフを作成するためのクラスに PieChart がある．このクラスのインスタンスを生成するのは簡単で，次のようにする．

PieChart pc = new PieChart();

これにより，円グラフのインスタンス pc が生成される．

円グラフのデータ項目は PieChart.Data クラスのオブジェクトで表し，これの「列」を与えることでグラフを描画する．データ列は ObservableList クラスのオブジェクトとして PieChart オブジェクトに与える．データ列の作成例を次に示す．

```
ObservableList<PieChart.Data> pcd = FXCollections.observableArrayList(
    new PieChart.Data("A",10),
    new PieChart.Data("B",30),
    new PieChart.Data("C",60)
);
```

データ列の生成には例にあるように FXCollections クラスのクラスメソッド observableArrayList を使用する．この例ではデータ列のオブジェクトが pcd として生成されている．あとは次の例のように，これを PieChart オブジェクトに与えることでグラフが描画される．

```
pc.getData().addAll(pcd);
```

図 39 のグラフを作成するサンプルプログラム FXChart04.java を次に示す．

**[プログラム：**FXChart04.java]

```
1  import javafx.application.*;
2  import javafx.stage.*;
3  import javafx.scene.*;
4  import javafx.scene.layout.*;
5  import javafx.scene.chart.*;
6  import javafx.collections.*;
7
8  public class FXChart04 extends Application {
9
10     @Override
11     public void start( Stage stg ) {
12         //-- ウィンドウの準備 --
13         AnchorPane root = new AnchorPane();
14         Scene sc = new Scene(root,450,450);
15         stg.setTitle("FXChart04");
16         stg.setScene(sc);
17
18         //-- 円グラフの準備 --
19         PieChart pc = new PieChart();
20         pc.setTitle("円グラフ");
21         pc.setPrefWidth(450d);
22         pc.setPrefHeight(450d);
23         //-- 系列の作成 --
24         ObservableList<PieChart.Data> pcd =
25           FXCollections.observableArrayList(
26             new PieChart.Data("A",10),
27             new PieChart.Data("B",30),
28             new PieChart.Data("C",60)
29           );
30         pc.getData().addAll(pcd);
31
32         //-- 表示 --
33         root.getChildren().add(pc);
34         stg.show();
35     }
36
37     public static void main( String argv[ ] ) {
38         launch(argv);
39     }
40 }
```

**参考：** 円グラフに与えるデータ列は次のようにして作成することもでき，プログラム中で逐次，項目を追加することもできる．

```
ObservableList<PieChart.Data> pcd = FXCollections.observableArrayList();
pcd.add(new PieChart.Data("A",10));
pcd.add(new PieChart.Data("B",30));
pcd.add(new PieChart.Data("C",60));
```

## 5.4 画像データの扱い

Java FX 8 で画像データ（ビットマップデータ）を扱うためのクラスに Image クラスがあり，読み込んだ画像ファイルの内容を保持したり，ピクセルの値（画素の色）を変更して画像処理を行うために利用できる．ここでは，ピクセル値の取得と設定，入出力の基本的な事柄について説明する．

### 5.4.1 画像データの入力： Image クラス

Image クラスはインスタンス生成時に画像データを読み込む．

コンストラクタ： Image(画像データの URL), Image(画像データのリソース)

画像データの URL は String クラスの値（文字列）で与える．例えば，カレントディレクトリにある画像ファイル “picture.png” を読み込んで Image クラスのオブジェクト img を生成するには．

```
Image img = new Image("file://picture.png");
```

あるいは単純に

```
Image img = new Image("picture.png");
```

とする．また，アプリケーションのリソース[5] “picture.png” から画像データを読み込む場合は getClass メソッドと getResourceAsStreamm メソッドを使用して

```
Image img = new Image(getClass().getResourceAsStream("picture.png"));
```

とする．

**■ 画像サイズの取得**

Image オブジェクトに対して，getWidth，getHeight メソッドを実行することで，横と縦のピクセル数が得られる．戻り値は double 型である．

**ピクセルサイズ取得の例：**

Image オブジェクト img の横と縦のピクセルサイズを int 型の変数 w, h に取得するには次のようにする．

```
int w = (int)img.getWidth();
int h = (int)img.getHeight();
```

### 5.4.2 画素（ピクセル）の操作

**■ WritableImage と PixelWriter**

Image クラスの拡張クラスに WritableImage クラスがあり，このクラスのオブジェクトを使用すると自由に画素を編集できる．

コンストラクタ： WritableImage(W,H)

コンストラクタの引数には画素の縦横のサイズを指定する．W は横幅，H は高さを意味する int 型の値である．

WritableImage クラスのオブジェクトの画素を操作するために PixelWriter クラスがあり，WritableImage のインスタンスから getPixelWriter メソッドを使用して取得する．例えば，WritableImage のインスタンス wimg がある場合，次のようにして PixelWriter オブジェクト pw を取得する．

```
PixelWriter pw = wimg.getPixelWriter();
```

後は得られた PixelWriter オブジェクトに対して画素の操作を行う．

**■ 画素の設定**

PixelWriter オブジェクトを用いて WritableImage の画素を設定するには setColor メソッドを使用する．例えば

[5]「4.2.2 単独で動作するアプリケーションの生成について」参照

PixelWriter オブジェクト pw に対して

pw.setColor(x,y,Color.rgb(255,0,0));

とすると，WritableImage オブジェクト上の (x,y) の位置の画素を赤に設定できる．

**■ 画素の取得**

Image オブジェクト上の画素を取得するには PixelReader クラスを使用する．PixelReader オブジェクトは Image オブジェクトから getPixelReader メソッドを使用して取得する．例えば，Image クラスのインスタンス img がある場合，次のようにして PixelReader オブジェクト pr を取得する．

PixelReader pr = img.getPixelReader();

後は得られた PixelReader オブジェクトに対して getColor メソッドを使用して画素を取得する．例えば PixelReader オブジェクト pr から画素を取得るには

Color c = pr.getColor(x,y);

とする．これにより，Image オブジェクト上の (x,y) の位置の画素が Color オブジェクト c に得られる．

### 5.4.3 画像データの出力

Image クラスのオブジェクトを画像ファイルとして保存するには ImageIO クラスを使用する．このクラスは Swing [6] に含まれる．

ImageIO クラスの write メソッドを使用すると，Swing の画像のためのクラスである BufferedImage のオブジェクトをファイルに出力できる．

**（画像の出力）**

ImageIO.write(画像オブジェクト, 出力ファイルのフォーマット, 出力ファイル)

'画像オブジェクト' は BufferedImag クラスのオブジェクト，'出力ファイルのフォーマット' は保存する画像ファイルのフォーマットで，"png"などと String 型で与える．'出力ファイル' が出力先のファイルで，File クラスのオブジェクトである．

Java FX 8 の Image や WritableImage のオブジェクトをファイルに出力するためには，それらを一旦 BufferedImage クラスのオブジェクトに変換する．この変換処理のために SwingFXUtils クラスのメソッド fromFXImage を使用する．

例えば，WritableImage のオブジェクト wimg を，ファイル "outfile.png" として保存するには次のようにする．

File ofile = new File("outfile.png");
ImageIO.write(SwingFXUtils.fromFXImage(wimg, null), "png", ofile);

**注意：** write メソッドを実行する際には，適切な例外処理の設定をすること．

### 5.4.4 サンプルプログラム

**(1) 3 色の画素を生成して描画する例**

3 色の画素を生成して，図 40 のように表示するプログラム FXimage01.java を示す．このプログラムは表示内容を画像ファイル 'FXimage01.png' と 'FXimage01.gif' として保存する．

**[プログラム：**FXimage01.java]

```
1 import javafx.application.*;
2 import javafx.stage.*;
3 import javafx.scene.*;
```

[6] Java における GUI と二次元グラフィックスの実現には，Java SE 8 が登場するまでは Swing が基本であった．Java SE 8 からも Swing の機能は提供されており，各種の便利な機能が利用できる．


図 40: 画素を直接描いている例

```
4  import javafx.scene.layout.*;
5  import javafx.scene.image.*;
6  import javafx.scene.paint.*;
7  import javafx.embed.swing.*;
8  import javax.imageio.*;
9  import java.io.*;
10
11 public class FXimage01 extends Application {
12
13     @Override
14     public void start( Stage stg ) {
15         //-- ウィンドウの準備 --
16         AnchorPane root = new AnchorPane();
17         Scene sc = new Scene(root,400,120);
18         stg.setTitle("FXimage01");
19         stg.setScene(sc);
20
21         //-- 画像の準備 --
22         WritableImage wimg = new WritableImage(400,120);
23         ImageView iv = new ImageView(wimg);
24         PixelWriter pw = wimg.getPixelWriter();
25         //-- 画像の生成 --
26         int x, y;
27         for ( y = 0; y < 40; y++ ) {
28             for ( x = 0; x < 400; x++ ) {
29                 pw.setColor(x,y,Color.rgb(255,0,0));
30             }
31         }
32         for ( y = 40; y < 80; y++ ) {
33             for ( x = 0; x < 400; x++ ) {
34                 pw.setColor(x,y,Color.rgb(0,255,0));
35             }
36         }
37         for ( y = 80; y < 120; y++ ) {
38             for ( x = 0; x < 400; x++ ) {
39                 pw.setColor(x,y,Color.rgb(0,0,255));
40             }
41         }
42
43         //-- 表示 --
44         root.getChildren().add(iv);
45         iv.relocate(0d,0d);
46         stg.show();
47
48         //-- 画像ファイル出力 --
49         File oFile1 = new File("FXimage01.png");
50         File oFile2 = new File("FXimage01.gif");
51         try {
52             ImageIO.write(SwingFXUtils.fromFXImage(wimg, null), "png", oFile1);
53             ImageIO.write(SwingFXUtils.fromFXImage(wimg, null), "gif", oFile2);
54         } catch (IOException e) {
55             System.out.println("ファイル出力エラー！");
56         }
57
58     }
59
60     public static void main( String argv[ ] ) {
61         launch(argv);
62     }
```

```
63 }
```

### (2) 画像ファイルを読み込んで表示する例

‘Earth.jpg’ として保存されている画像（図 41）を読み込み，それを新規の WritableImage に画素毎に複写して表示（図 40）するプログラム FXimage02.java を示す．このプログラムは表示内容を画像ファイル ‘FXimage02.png’ と ‘FXimage02.gif’ として保存する．

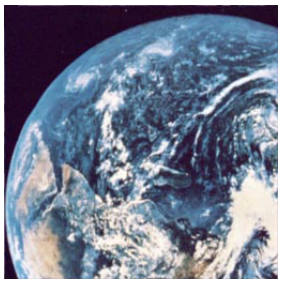


図 41: 読み込む画像：Earth.jpg

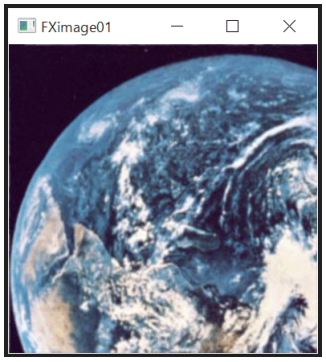



図 42: 読み込んだ画像を複製して表示している例

**[プログラム：FXimage02.java]**

```
1  import javafx.application.*;
2  import javafx.stage.*;
3  import javafx.scene.*;
4  import javafx.scene.layout.*;
5  import javafx.scene.image.*;
6  import javafx.scene.paint.*;
7  import javafx.embed.swing.*;
8  import javax.imageio.*;
9  import java.io.*;
10
11 public class FXimage02 extends Application {
12
13     @Override
14     public void start( Stage stg ) {
15         //-- 画像の読み込み --
16         Image img = new Image("Earth.jpg");
17         PixelReader pr = img.getPixelReader();
18         int w, h;
19         w = (int)img.getWidth();
20         h = (int)img.getHeight();
21
22         //-- ウィンドウの準備 --
23         AnchorPane root = new AnchorPane();
24         Scene sc = new Scene(root,w,h);
25         stg.setTitle("FXimage01");
26         stg.setScene(sc);
27
28         //-- 画像の生成（出力用） --
```

```
29          WritableImage wimg = new WritableImage(w,h);
30          ImageView iv = new ImageView(wimg);
31          PixelWriter pw = wimg.getPixelWriter();
32
33          //-- 画像の複写 --
34          Color c;
35          int x, y;
36          for ( x = 0; x < w; x++ ) {
37              for ( y = 0; y < h; y++ ) {
38                  c = pr.getColor(x,y);
39                  pw.setColor(x,y,c);
40              }
41          }
42
43          //-- 表示 --
44          root.getChildren().add(iv);
45          iv.relocate(0d,0d);
46          stg.show();
47
48          //-- 画像ファイル出力 --
49          File oFile1 = new File("FXimage02.png");
50          File oFile2 = new File("FXimage02.gif");
51          try {
52              ImageIO.write(SwingFXUtils.fromFXImage(wimg, null), "png", oFile1);
53              ImageIO.write(SwingFXUtils.fromFXImage(wimg, null), "gif", oFile2);
54          } catch (IOException e) {
55              System.out.println("ファイル出力エラー！");
56          }
57
58      }
59
60      public static void main( String argv[ ] ) {
61          launch(argv);
62      }
63  }
```

#### 5.4.5 ノードの描画状態のキャプチャ

Pane などの Node の表示状態を WritableImage のオブジェクトに取得（キャプチャ）するための最も基本的な方法を紹介する．

例えば AnchorPane オブジェクト ap の表示状態を WritableImage オブジェクト wimg にキャプチャするには，次のように snapshot メソッドを使用する．

```
ap.snapshot(null, wimg);
```

# 6 時間によるイベント処理（アニメーション）

ここでは，設定されたタイマーに応じて発動するイベント処理について説明する．Java FX では，**アニメーション**（Animation）というクラスでタイミングに関するイベントを取り扱う．

## 6.1 タイミング・イベントについて

GUI におけるイベント処理では，主としてデバイス（キーボードやマウスなど）に対するユーザの操作を処理の起点とする．これとは別に，指定した時間間隔で自動的に発生するイベントも存在し，このようなイベント（**タイミング・イベント**）を起点として，アニメーションのフレーム表示といった，連続した処理を実現することができる．Java SE 8 では，この「連続した処理」を**アニメーション**（Animation）という考え方で扱い，いわゆるアニメーションのタイムラインを実現するための Timeline クラスが提供されている．

## 6.2 タイムラインとキーフレーム

**アニメーションの時間軸を表すクラス** Timeline

**コンストラクタ：** Timeline( new KeyFrame(...) )

アニメーションのキーフレームを実現する KeyFrame クラスのオブジェクト（後述）にはタイミングの時間間隔を設定し，生成した Timeline オブジェクトに繰返し回数などを設定してアニメーションを実行する．

**重要なメソッド：**

・setCycleCount( 繰り返し回数 )　キーフレームの繰り返し回数（タイミングイベントの発生回数）を整数型で指定する．

・play( )　タイムラインの実行（タイミング・イベントの発生）を開始する．

・pause( )　タイムラインの実行を一時停止する．play() で実行を再開することができる．

アニメーションの実行単位（1 コマ）を実現するものを**キーフレーム**といい，KeyFrame クラスを用いて実現する．

KeyFrame **クラス**

**コンストラクタ：** KeyFrame(**時間間隔 , イベントハンドラ**)

時間間隔には Duration クラスの値を与える．時間間隔を与えるための下記のようなクラスメソッドがある．

・Duration.millis( 時間間隔 )　ミリ秒単位で時間間隔を指定する．

・Duration.seconds( 時間間隔 )　秒単位で時間間隔を指定する．

・Duration.minutes( 時間間隔 )　分単位で時間間隔を指定する．

・Duration.hours( 時間間隔 )　60 分単位で時間間隔を指定する．

上記の「時間間隔」は double 型で与える．

KeyFrame クラスのオブジェクトは，タイミング・イベントを受けてイベント処理を実行するためのものである．

## 6.3 サンプルプログラム

### 【1】バウンドするボール

ウィンドウの枠内でバウンドする球体のアニメーションを表示するサンプルプログラムを示しながら解説する．

**ソースプログラム：** AnimSample01.java

```
1  import javafx.util.*;
2  import javafx.application.*;
3  import javafx.event.*;
4  import javafx.scene.*;
5  import javafx.stage.*;
6  import javafx.scene.shape.*;
7  import javafx.animation.*;
8
9  public class AnimSample01 extends Application {
10     double  w = 391d, h = 287d,         // Window Size
11             r = 50d,                    // Radius of Sphere
12             x = 0d, y = 0d,             // Position of Sphere
13             dx = 1d, dy = 1d, dt = 2d;  // Motion Factors
14     Sphere  sp1;    // Bouncing Sphere
15
16     //--------------------------
17     // Bouncing Action
18     //--------------------------
19     void bounce() {
20         x += dx;
21         y += dy;
22         if ( x < 0 ) {
23             x = 0;
24             dx *= -1d;
```

```
25          } else if ( x > w ) {
26              x = w;
27              dx *= -1d;
28          }
29          if ( y < 0 ) {
30              y = 0;
31              dy *= -1d;
32          } else if ( y > h ) {
33              y = h;
34              dy *= -1d;
35          }
36          sp1.relocate(x,y);
37      }
38
39      //--------------------------
40      // GUI & Event Handling
41      //--------------------------
42      @Override
43      public void start(Stage stg) {
44          Group root = new Group();
45          Scene scene = new Scene(root, w+r*2d, h+r*2d);
46          stg.setTitle("Animation Sample 1");
47          stg.setScene(scene);
48
49          sp1 = new Sphere(r);
50          root.getChildren().add(sp1);
51
52          Timeline timer = new Timeline(
53              new KeyFrame(Duration.millis(dt),
54                  new EventHandler<ActionEvent>() {
55                      @Override
56                      public void handle(ActionEvent evt) {
57                          bounce();
58                      }
59                  }));
60          timer.setCycleCount(Timeline.INDEFINITE);
61
62          stg.show();
63          timer.play();
64      }
65
66      //--- Main ---
67      public static void main(String[] args) {
68          launch(args);
69      }
70  }
```

**【解説】**

**52〜59 行目：** タイムラインの生成.

キーフレームを生成する際，handle メソッド内に bounce メソッドを呼び出すように記述している．これにより，タイミング・イベントが発生するたびに bounce を呼び出してアニメーションのフレーム（1 コマ）を実現している．bounce メソッドの定義は 19〜37 行目に

**60 行目：** タイミング・イベントの発生回数の設定.

ここでは Timeline.INDEFINITE を指定しており，これにより際限なくタイミング・イベントを発生することができる.

**63 行目：** アニメーションの開始.（タイミング・イベントの発生開始）

このプログラムを実行すると，ウィンドウ中を球体がバウンドするアニメーションが表示される.（図 43）

**【2】三次元オブジェクトの回転**

「3.2 直方体，円柱，球」で紹介したサンプルプログラムを拡張して作成した，三次元オブジェクトが回転するア

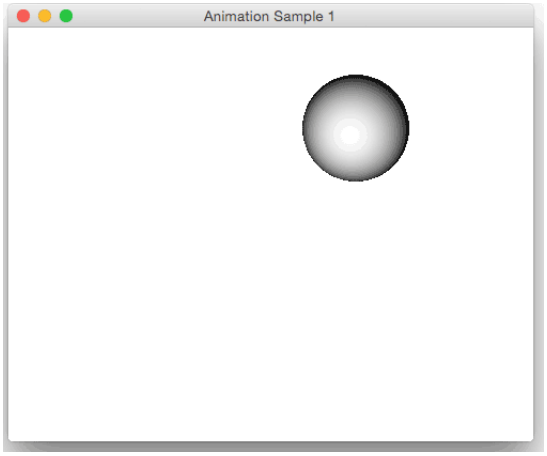



図 43: 実行したところ

ニメーションを実現するサンプルプログラム FX3Dsample04.java を示す.

**ソースプログラム：** FX3Dsample04.java[7]

```
1  import javafx.util.*;
2  import javafx.application.*;
3  import javafx.event.*;
4  import javafx.stage.*;
5  import javafx.scene.*;
6  import javafx.scene.layout.*;
7  import javafx.scene.paint.*;
8  import javafx.scene.shape.*;
9  import javafx.scene.image.*;
10 import javafx.scene.transform.*;
11 import javafx.geometry.*;
12 import javafx.animation.*;
13
14 public class FX3Dsample04 extends Application {
15
16     //--- Elements of the world
17     Group   root,   // for Univers
18             child;  // for Earth
19     Scene scene;
20
21     Point3D aX, aY, aZ; // Rotation Axis
22
23     // Shape3D
24     Box bx1;
25     Cylinder cl1;
26     Sphere sp1;
27
28     // Lights
29     AmbientLight aLight;
30     PointLight pLight;
31
32     PerspectiveCamera cmr;  // Camera
33
34     Image img;  // Texture for Earth
35
36     //--- Rotation Action ---
37     void rot() {
38         bx1.getTransforms().addAll(
39             new Rotate(1,aY)
40         );
41         cl1.getTransforms().addAll(
42             new Rotate(1,aX)
43         );
```

[7]謝辞：このサンプルプログラムを作成するに当たって，インターネットサイト http://www.free-world-maps.com/ の画像 physical-free-world-map-b1.jpg を使用させていただいた.

```
44          child.getTransforms().addAll(
45              new Rotate(1,aY)
46          );
47      }
48
49      @Override
50      public void start(Stage stg) {
51
52          //--- Top Node and Scene ---
53          root = new Group();
54          scene = new Scene(root, 1000, 400, Color.rgb(0,0,0));
55
56          // Axis for Rotation
57          aX = new Point3D(100,0,0);
58          aY = new Point3D(0,100,0);
59          aZ = new Point3D(0,0,100);
60
61          //--- Solid Model Generation ---
62          // (Box)
63          bx1 = new Box(300d,200d,150d);
64          root.getChildren().add(bx1);
65          PhongMaterial mt1 = new PhongMaterial();
66          mt1.setDiffuseColor(Color.rgb(150,0,0));
67          mt1.setSpecularColor(Color.rgb(255,0,0));
68          mt1.setSpecularPower(5d);
69          bx1.setMaterial(mt1);
70          bx1.getTransforms().addAll(
71              new Translate(-300d,0d,0d), // 平行移動
72              new Rotate(30,aX), // X軸回りの回転
73              new Rotate(30,aY), // Y軸回りの回転
74              new Rotate(20,aZ)  // Z軸回りの回転
75          );
76          // (Cylinder)
77          cl1 = new Cylinder(80d,300d);
78          root.getChildren().add(cl1);
79          PhongMaterial mt2 = new PhongMaterial();
80          mt2.setDiffuseColor(Color.rgb(0,150,0));
81          mt2.setSpecularColor(Color.rgb(0,255,0));
82          mt2.setSpecularPower(10d);
83          cl1.setMaterial(mt2);
84          cl1.getTransforms().addAll(
85              new Rotate(30,aX),
86              new Rotate(-20,aZ)
87          );
88          // (Sphere)
89          img = new Image("file:physical-free-world-map-b1.jpg");
90          child = new Group();
91          sp1 = new Sphere(140d);
92          child.getChildren().add(sp1);
93          PhongMaterial mt3 = new PhongMaterial();
94          mt3.setDiffuseMap(img);
95          mt3.setSpecularColor(Color.rgb(127,127,127));
96          mt3.setSpecularPower(5d);
97          sp1.setMaterial(mt3);
98          sp1.getTransforms().addAll(
99              new Translate(80d,0d,0d)
100         );
101         root.getChildren().add(child);
102         child.getTransforms().addAll(
103             new Translate(330d,0d,0d)
104         );
105
106         //--- Light Setting ---
107         aLight = new AmbientLight(Color.rgb(127, 127, 127));
108         root.getChildren().add(aLight);
109
110         pLight = new PointLight(Color.rgb(255,255,255));
111         pLight.setTranslateX(500d);
112         pLight.setTranslateY(-300d);
```

```
113         pLight.setTranslateZ(-200d);
114         root.getChildren().add(pLight);
115 
116         //--- Camera Setting ---
117         cmr = new PerspectiveCamera();
118         cmr.getTransforms().addAll(
119             new Translate(-450d,-200d,-100d)
120         );
121         scene.setCamera(cmr);
122 
123         //--- Window Activation ---
124         stg.setTitle("FX3Dsample01");
125         stg.setScene(scene);
126         stg.show();
127 
128         //--- Animation ---
129         Timeline timer = new Timeline(
130             new KeyFrame(Duration.millis(10),
131                 new EventHandler<ActionEvent>() {
132                     @Override
133                     public void handle(ActionEvent evt) {
134                         rot();
135                     }
136                 }));
137         timer.setCycleCount(Timeline.INDEFINITE);
138         timer.play();
139     }
140 
141     public static void main(String[] args) {
142         launch(args);
143     }
144 }
```

**【解説】**

イベントハンドラの登録の際，handle メソッドの定義内に rot メソッドを実行する記述（134 行目）がある．これによりアニメーションの 1 コマを実現している．rot メソッドの定義は 37〜47 行目にある．

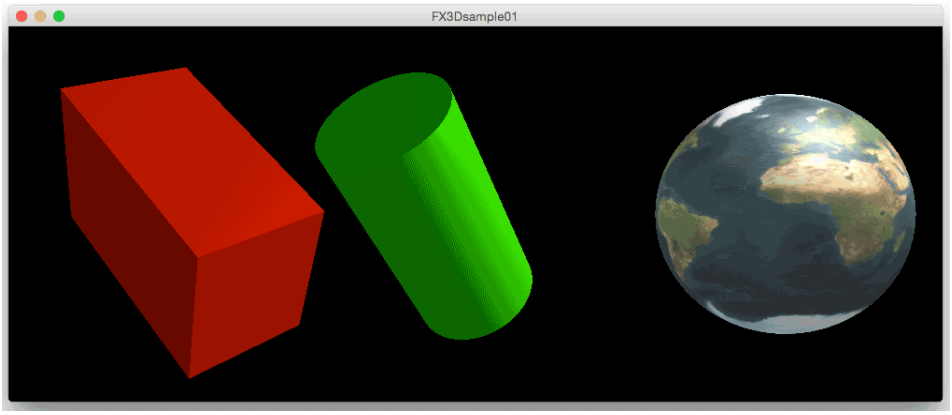



図 44: FX3Dsample04.java を実行したところ（各立体が回転する）

# 7　日付と時刻

ここでは日付と時刻のデータを扱う 2 種類の方法について説明する．1 つは Java SE 7 までにおける方法と，もう 1 つは Java SE 8 から導入された新しい方法である．

## 7.1 旧来の API（Java SE 7）

古い版の Java に対する後方互換性を重視する意味で，Java SE 7 における日付・時刻の取り扱いについて説明する．

### 7.1.1 日付と時刻のためのクラス

日付と時刻の情報を保持するために Date クラスと Calendar クラスがある．前者は連続的な時間を表す単純なクラスであるのに対し，後者は “年，月，日，時，分，秒” といった各種のフィールドを取り扱うのに向いている．

これらのクラスは java.util 配下のライブラリとして提供されており，使用する場合は，java.util.Date と java.util.Calendar をインポートする必要がある．

ここでは，サンプルプログラムを示しながら Date クラスと Calendar クラスについて説明する．

**【サンプルプログラム】**

ここで示すサンプルプログラム DateTest01.java は次のような処理を行うものである．

- 現在の時刻を取得して，書式整形して表示
- 指定した日付と時刻を書式整形して表示
- 時間の加算

**ソースプログラム：** DateTest01.java

```
1  import java.util.*;
2  import java.text.*;
3
4  class DateTest01 {
5      public static void main( String argv[] ) {
6          //--- 現在の日付と時刻のデータ ---
7          Date d = new Date();
8
9          //--- 日付の書式データ ---
10         SimpleDateFormat f =
11             new SimpleDateFormat("GG yyyy年MM月dd日 HH:mm:ss.S (E) [z/Z]");
12
13         //--- 現在の日付・時刻の出力 ---
14         System.out.println("現在：    "+f.format(d));
15
16         //--- カレンダーに 1989/01/08 01:02:03.0 を設定 ---
17         Calendar cal = Calendar.getInstance();
18         int y = 1989,   m   = 1,    day = 8,
19             h   = 1,    min = 2,    sec = 3,    ms  = 0;
20         cal.set(Calendar.YEAR,  y);
21         cal.set(Calendar.MONTH, m-1);
22         cal.set(Calendar.DATE,  day);
23         cal.set(Calendar.HOUR_OF_DAY, h);
24         cal.set(Calendar.MINUTE, min);
25         cal.set(Calendar.SECOND, sec);
26         cal.set(Calendar.MILLISECOND, ms);
27         // 日付・時刻の書式で出力
28         Date d2 = cal.getTime();
29         System.out.println("平成開始："+f.format(d2));
30         // 26年後の日付
31         cal.add(Calendar.YEAR,26);
32         d2 = cal.getTime();
33         System.out.println("26年後：  "+f.format(d2));
34     }
35 }
```

**【 解説 】**

**● 7 行目：** 現在の日付と時刻の取得

Date クラスのオブジェクトは新規に作成すると，その時点での日付と時刻の情報を保持している．このオブジェク

トの内容をわかりやすい形に書式整形するために SimpleDateFormat クラスがある．このクラスのオブジェクトには “年，月，日，時，分，秒" といった内容を表現するための書式情報を設定し，それに基いて Date クラスのオブジェクトを文字列オブジェクトの形に変換するのに使用する．

SimpleDateFormat クラスは java.text 配下のライブラリとして提供されており，使用する場合は，java.text.SimpleDateFormat をインポートする必要がある．

時間情報の書式整形

**実行例：**
```
SimpleDateFormat f =
        new SimpleDateFormat("GG yyyy 年 MM 月 dd 日 HH:mm:ss.S (E) [z/Z]");
Date d = new Date();
String dtext = f.format(d);
```
**解説：** format メソッドを用いて，f に設定された書式に従って日付と時刻のデータである d を整形し，でき上がった表現を String クラスの dtext に格納する．

※ format メソッドの戻り値は StringBuffer クラスの値である

整形のための書式を表 4 に示す．

表 4: 日付けのフォーマット

| 書式文字 | 意味 | 書式文字 | 意味 |
|---|---|---|---|
| G | 紀元 | H | 一日における時 (0 〜 23) |
| y | 年 | k | 一日における時 (1 〜 24) |
| M | 月 | K | 午前/午後の時 (0 〜 11) |
| w | 年における週 | h | 午前/午後の時 (1 〜 12) |
| W | 月における週 | m | 分 |
| D | 年における日 | s | 秒 |
| d | 月における日 | S | ミリ秒 |
| F | 月における曜日 | z | 一般的なタイムゾーン（例．PST） |
| E | 曜日 | Z | RFC 822 タイムゾーン（例．-0800） |

● **14 行目：** 書式整形された日付と時刻の表示

● **17 行目：** Calendar クラスのオブジェクトの生成

Calender クラスを用いると，日数や時間の設定，時間経過に関する計算などができる．このクラスのインスタンスを生成するには getInstance メソッド（クラスメソッド）を用いる．インスタンス生成後は，set メソッドを用いて日付や時刻の情報を設定することができる．(18〜26 行目)

日付・時刻の設定

**実行例：**
```
Calendar cal = Calendar.getInstance();
cal.set(Calendar.YEAR, 1989);
```
**解説：** Calendar クラスのインスタンス cal を生成し，それに対して “1989 年" を設定している．

set メソッドの第 1 引数には，日付や時刻を意味するフィールドを指定するが，これは表 5 の通りである．

表 5: Calendar の値のフィールド

| 記述 | 意　味 | 記述 | 意　味 |
|---|---|---|---|
| Calendar.YEAR | 年 | Calendar.MINUTE | 分 |
| Calendar.MONTH | 月 | Calendar.SECOND | 秒 |
| Calendar.DATE | 日 | Calendar.MILLISECOND | ミリ秒 |
| Calendar.HOUR_OF_DAY | 時 | | |

**※注意：**

Calendar クラスの「月」の値（Calendar.MONTH）の範囲は 0〜11 であり,「0」は「1 月」を表す．そのため，サン

プルプログラムの 21 行目では 'm-1' のようにして調整している．

● **28 行目：** Calendar クラスのオブジェクトの値を Date クラスのオブジェクトに変換
Calendar クラスのオブジェクトに対して getTime メソッドを用いて，Date クラスのオブジェクトに変換することができる．

**（参考）**
getTime メソッドとは逆に setTime メソッドを用いると，Date クラスのオブジェクトの値を Calendar クラスのオブジェクトに変換することができる．

● **31 行目：** Calendar クラスのオブジェクトへの時間の加算
Calendar クラスのオブジェクトに対して add メソッドを実行することで時間を加算することができる．add の第 1 引数には加算する時間単位を指定する．これには表 5 に示すものを用いる．add の第 2 引数には加算する値を指定する．今回のサンプルプログラムでは，26 年後の日付と時刻を計算している．

サンプルプログラムを実行した例を次に示す．

《実行結果の例》

```
現在：　　西暦 2015 年 07 月 22 日 11:40:59.472 (水) [JST/+0900]
平成開始：西暦 1989 年 01 月 08 日 01:02:03.0 (日) [JST/+0900]
26 年後：　西暦 2015 年 01 月 08 日 01:02:03.0 (木) [JST/+0900]
```

## 7.2 新しい API（Java SE 8）

Java SE 8 から，ISO 8601 に準拠した形で日付と時刻を扱う新たな API が導入された．新しい API では，和暦や東南アジア諸国の仏暦，中華民国（台湾）の民国暦，イスラム社会のヒジュラ暦などの扱いをサポートしている．また日付や時刻をそれぞれ単独で扱うクラスが新たに加わった．新しい API はスレッドセーフになり，マルチスレッドプログラミングにおいても安全なものになった．

### 7.2.1 基本的なクラス

基本的なクラスとして LocalDateTime，LocalDate，LocalTime の 3 つがある．

基本的なクラス

- LocalDateTime　日付と時刻を保持するクラス
- LocalDate　日付のみを保持するクラス
- LocalTime　時刻のみを保持するクラス

それぞれのクラスのインスタンスを生成するには，クラスメソッドの now を用いる．

**例 1．** LocalDateTime datetime = LocalDateTime.now();
**例 2．** LocalDateTime datetime = LocalDateTime.now(ZoneId.of("Europe/Paris"));

このような方法で生成されたインスタンスには now メソッド実行時点の日付（時刻）が設定されている．例 2 にある "Europe/Paris" はタイムゾーン（表 6 参照）であり，これを与えることで指定した地域における日付・時刻が取得できる．タイムゾーンは ZoneID クラスのクラス・メソッド of を呼び出すことで指定する．
タイムゾーン毎の現在時刻を表示するサンプルプログラムを次に示す．

表 6: タイムゾーンの一覧

| | |
|---|---|
| Africa/Addis_Ababa | Asia/Dhaka |
| Africa/Cairo | Asia/Ho_Chi_Minh |
| Africa/Harare | Asia/Karachi |
| America/Anchorage | Asia/Kolkata |
| America/Argentina/Buenos_Aires | Asia/Shanghai |
| America/Chicago | Asia/Tokyo |
| America/Indiana/Indianapolis | Asia/Yerevan |
| America/Los_Angeles | Australia/Darwin |
| America/Phoenix | Australia/Sydney |
| America/Puerto_Rico | Europe/Paris |
| America/Sao_Paulo | Pacific/Apia |
| America/St_Johns | Pacific/Auckland |
| | Pacific/Guadalcanal |

**ソースプログラム：** DateTest02.java

```
1  import java.time.*;
2
3  class DateTest02 {
4      public static void main( String argv[] ) {
5          //ーー タイムゾーン ーー
6          String[] tz = {
7              "Africa/Addis_Ababa",   "Africa/Cairo",
8              "Africa/Harare",        "America/Anchorage",
9              "America/Argentina/Buenos_Aires",
10             "America/Chicago",
11             "America/Indiana/Indianapolis",
12             "America/Los_Angeles",  "America/Phoenix",
13             "America/Puerto_Rico",  "America/Sao_Paulo",
14             "America/St_Johns",     "Asia/Dhaka",
15             "Asia/Ho_Chi_Minh",     "Asia/Karachi",
16             "Asia/Kolkata",         "Asia/Shanghai",
17             "Asia/Tokyo",           "Asia/Yerevan",
18             "Australia/Darwin",     "Australia/Sydney",
19             "Europe/Paris",         "Pacific/Apia",
20             "Pacific/Auckland",     "Pacific/Guadalcanal"
21         };
22
23         //ーー 現在の日付と時刻のデータ ーー
24         LocalDate[] d = new LocalDate[25];
25         LocalTime[] t = new LocalTime[25];
26         int c;
27         for ( c = 0; c < 25; c++ ) {
28             d[c] = LocalDate.now(ZoneId.of(tz[c]));
29             t[c] = LocalTime.now(ZoneId.of(tz[c]));
30         }
31
32         //ーー 出力 ーー
33         for ( c = 0; c < 25; c++ ) {
34             System.out.println(d[c]+" : "+t[c]+" : "+tz[c]);
35         }
36     }
37 }
```

これを実行すると「実行例：DateTest02.java」のような結果となる．

### 7.2.2 日付・時刻に関する基本的な処理

基本的な処理について，サンプルプログラムを示しながら説明する．

次に示すサンプルプログラム DateTest06.java は，次のような処理を行う例である．

実行例：DateTest02.java

```
2015-07-20 : 13:04:54.609 : Africa/Addis_Ababa
2015-07-20 : 12:04:54.660 : Africa/Cairo
2015-07-20 : 12:04:54.661 : Africa/Harare
2015-07-20 : 02:04:54.670 : America/Anchorage
2015-07-20 : 07:04:54.670 : America/Argentina/Buenos_Aires
2015-07-20 : 05:04:54.672 : America/Chicago
2015-07-20 : 06:04:54.672 : America/Indiana/Indianapolis
2015-07-20 : 03:04:54.673 : America/Los_Angeles
2015-07-20 : 03:04:54.673 : America/Phoenix
2015-07-20 : 06:04:54.673 : America/Puerto_Rico
2015-07-20 : 07:04:54.674 : America/Sao_Paulo
2015-07-20 : 07:34:54.675 : America/St_Johns
2015-07-20 : 16:04:54.675 : Asia/Dhaka
2015-07-20 : 17:04:54.675 : Asia/Ho_Chi_Minh
2015-07-20 : 15:04:54.675 : Asia/Karachi
2015-07-20 : 15:34:54.675 : Asia/Kolkata
2015-07-20 : 18:04:54.675 : Asia/Shanghai
2015-07-20 : 19:04:54.675 : Asia/Tokyo
2015-07-20 : 14:04:54.676 : Asia/Yerevan
2015-07-20 : 19:34:54.676 : Australia/Darwin
2015-07-20 : 20:04:54.676 : Australia/Sydney
2015-07-20 : 12:04:54.676 : Europe/Paris
2015-07-20 : 23:04:54.677 : Pacific/Apia
2015-07-20 : 22:04:54.677 : Pacific/Auckland
2015-07-20 : 21:04:54.677 : Pacific/Guadalcanal
```

- 現在の日付・時刻を取得する
- 日付・時刻の加減の演算を行う.
- 日付・時刻を比較（前後の判定）を行う
- 任意の日付・時刻の値を生成する

**ソースプログラム：** DateTest06.java

```
1  import java.time.*;
2
3  class DateTest06 {
4
5      //--- 曜日テーブル ---
6      static String[] JpWeek = {"日","月","火","水","木","金","土"};
7
8      //--- 日付と時刻の表示＆うるう年の判定 ---
9      static void disp(LocalDateTime d) {
10         int year, month, day, dayY, dayWn, hour, min, sec, mill;
11         DayOfWeek dayW;
12
13         LocalDate date  = d.toLocalDate();  // 日付のみ取り出し
14         LocalTime tm    = d.toLocalTime();  // 時刻のみ取り出し
15
16         year    = d.getYear();
17         month   = d.getMonthValue();
18         day     = d.getDayOfMonth();
19         dayY    = d.getDayOfYear();
20         dayW    = d.getDayOfWeek();
21         dayWn   = dayW.getValue();
22         hour    = d.getHour();
23         min     = d.getMinute();
24         sec     = d.getSecond();
25         mill    = d.getNano() / 1000000;
26
27         System.out.print(year+"/"+month+"/"+day+
28             "(年通算"+dayY+"日目)"+
29             dayW+"("+JpWeek[dayWn%7]+")  ");
30         System.out.println("\t"+hour+":"+min+":"+sec+"."+mill+
31             " うるう年か?("+date.isLeapYear()+")");
```

```
32      }
33
34      //--- メイン ---
35      public static void main( String argv[] ) {
36
37          //ーー 現在の日付と時刻 ーー
38          LocalDateTime d = LocalDateTime.now();
39          System.out.print("d:  ");
40          disp(d);
41
42          //ーー 1年1ヶ月1日1時間1分1秒後の日付 ーー
43          LocalDateTime d2 = d.plusYears(1).plusMonths(1).plusDays(1).
44                               plusHours(1).plusMinutes(1).plusSeconds(1);
45          System.out.print("d2: ");
46          disp(d2);
47
48          //ーー 1年1ヶ月1日1時間1分1秒前の日付 ーー
49          LocalDateTime d1 = d.minusYears(1).minusMonths(1).minusDays(1).
50                               minusHours(1).minusMinutes(1).minusSeconds(1);
51          System.out.print("d1: ");
52          disp(d1);
53
54          //ーー 前後の検査 ーー
55          int c = d.compareTo(d1);
56          if ( c == 0 ) {
57              System.out.println("d == d1");
58          } else if ( c > 0 ) {
59              System.out.println("d > d1");
60          } else {
61              System.out.println("d < d1");
62          }
63
64          //ーー 日付と時刻の生成：一気に設定 ーー
65          LocalDateTime dtn = LocalDateTime.parse("1989-01-08T01:02:03");
66          System.out.print("dtn(1):");
67          disp(dtn);
68
69          //ーー 日付と時刻の生成：2段階に設定 ーー
70          LocalDate dn = LocalDate.parse("1989-01-08");
71          LocalTime tn = LocalTime.parse("01:02:03");
72          dtn = tn.atDate(dn);
73          System.out.print("dtn(2):");
74          disp(dtn);
75          // 逆も
76          dtn = dn.atTime(tn);
77          System.out.print("dtn(3):");
78          disp(dtn);
79      }
80  }
```

**【解説】**

**38〜40 行目：** 現在の日付・時刻を取得して表示する．このサンプルでは，日付・時刻を表示するために disp というクラスメソッドを定義して使用している．

**16〜25 行目：** 表 7 に挙げる各種のメソッドを用いて，年, 月, 日, 時, 分, 秒の それぞれの値を取り出している．

**43〜46 行目：** 表 8 に挙げる各種のメソッドを用いて，1 年 1ヶ月 1 日 1 時間 1 分 1 秒後の 日付・時刻を求め，disp メソッドで表示している．

**49〜52 行目：** 表 8 に挙げる各種のメソッドを用いて，1 年 1ヶ月 1 日 1 時間 1 分 1 秒前の 日付・時刻を求め，disp メソッドで表示している．

**55〜62 行目：** compareTo メソッドを用いて日付・時刻を比較し，その結果を表示している．

**65〜78 行目：** parse メソッド（後述）を用いて日付・時刻の値を生成している．

表 7: 基本的なメソッド

| メソッド | 説明 |
|---|---|
| toLocalDate | LocalDateTime → LocalDate の変換 |
| toLocalTime | LocalDateTime → LocalTime の変換 |
| getYear | 「年」の取り出し |
| getMonthValue | 「月」の取り出し（1〜12） |
| getDayOfMonth | 月の内の「日数」の取り出し |
| getDayOfYear | 年内の「日数」の取り出し |
| getDayOfWeek | 週の内の「曜日」の取り出し<br>戻り値のクラスは DayOfWeek<br>これに対して getValue メソッドを使用することで整数値を取り出すことができる.（日〜土を 0〜6 で表現する） |
| getHour | 「時」の取り出し |
| getMinute | 「分」の取り出し |
| getSecond | 「秒」の取り出し |
| getNano | 「ナノ秒」の取り出し |
| isLeapYear | LocalDate オブジェクトがうるう年かどうかを判定する.<br>（戻り値は boolean） |

**【比較と演算】**

日付けと時刻に関する基本的な演算を表 8 に示す.

表 8: 基本的なメソッド

| メソッド | 説明 | メソッド | 説明 |
|---|---|---|---|
| plusYears(整数値) | 「1 年後」を算出 | minusYears(整数値) | 「1 年前」を算出 |
| plusMonths(整数値) | 「1ヶ月後」を算出 | minusMonths(整数値) | 「1ヶ月前」を算出 |
| plusDays(整数値) | 「1 日後」を算出 | minusDays(整数値) | 「1 日前」を算出 |
| plusHours(整数値) | 「1 時間後」を算出 | minusHours(整数値) | 「1 時間前」を算出 |
| plusMinutes(整数値) | 「1 分後」を算出 | minusMinutes(整数値) | 「1 分前」を算出 |
| plusSeconds(整数値) | 「1 秒後」を算出 | minusSeconds(整数値) | 「1 秒前」を算出 |
| D1.compareTo(D2) | $D1 > D2$ のとき正の値<br>$D1 = D2$ のとき 0<br>$D1 < D2$ のとき負の値 | | |

**【日付けの値を設定する方法】**

次に示すような方法で年月日，時分秒の値の設定ができる.

年月日，時分秒の設定

**実行例：** LocalDateTime dtn = LocalDateTime.parse("1989-01-08T01:02:03");

**実行例：** LocalDate dn = LocalDate.parse("1989-01-08");

**実行例：** LocalTime tn = LocalTime.parse("01:02:03");

**【LocalDate,LocalTime から LocalDateTime を合成する方法】**

LocalDate クラスのオブジェクト dn と LocalTime クラスのオブジェクト tn がある場合，これらのオブジェクトの値を LocalDateTime クラスのオブジェクト dtn に設定する下記のような方法がある.

**例 1．** dtn = tn.atDate(dn);
**例 2．** dtn = dn.atTime(tn);

このように，atDate, atTime メソッドを使用する.

このサンプルプログラムを実行すると「実行例：DateTest06.java」のような表示となる.

《実行例：DateTest06.java》

```
d: 2015/7/22(年通算 203 日目)WEDNESDAY(水) 16:33:27.183 うるう年か?(false)
d2: 2016/8/23(年通算 236 日目)TUESDAY(火) 17:34:28.183 うるう年か?(true)
d1: 2014/6/21(年通算 172 日目)SATURDAY(土) 15:32:26.183 うるう年か?(false)
d > d1
dtn(1):1989/1/8(年通算 8 日目)SUNDAY(日) 1:2:3.0 うるう年か?(false)
dtn(2):1989/1/8(年通算 8 日目)SUNDAY(日) 1:2:3.0 うるう年か?(false)
dtn(3):1989/1/8(年通算 8 日目)SUNDAY(日) 1:2:3.0 うるう年か?(false)
```

### 7.2.3 和暦の扱い

和暦を扱うためのクラスとして JapaneseDate がある．JapaneseDate クラスのインスタンスは，クラスメソッド now で生成することができ，生成時の日付と時刻の値が設定される．

サンプルプログラム DateTest07.java を示しながら和暦の扱いについて説明する．

**ソースプログラム：** DateTest07.java

```
1  import java.time.*;
2  import java.time.chrono.*;
3  import java.time.temporal.*;
4
5  class DateTest07 {
6
7      //--- 年号テーブル ---
8      static String[] JpEra = {"明治","大正","昭和","平成"};
9
10     //--- 和暦出力 ---
11     static void jdisp(JapaneseDate dj) {
12         System.out.print(JpEra[dj.get(ChronoField.ERA)+1]+"(");
13         System.out.print(dj.getEra()+").");
14         System.out.println(dj.get(ChronoField.YEAR_OF_ERA));
15     }
16
17     //--- メイン ---
18     public static void main( String argv[] ) {
19
20         //--- 現在の日付と時刻を和暦で取得 ---
21         JapaneseDate dj = JapaneseDate.now();
22         System.out.println("直接取得：  "+dj);
23
24         //--- LocalDateから和暦に変換 ---
25         LocalDateTime datetime = LocalDateTime.now();
26         LocalDate date = datetime.toLocalDate();
27         dj = JapaneseDate.from(date);
28         System.out.println("和暦に変換："+dj);
29
30         //--- 和暦年号の扱い ---
31         int e;
32         //  (明治6年)
33         dj = JapaneseDate.from(LocalDate.parse("1873-01-25"));
34         jdisp(dj);
35         //  (大正元年)
36         dj = JapaneseDate.from(LocalDate.parse("1912-07-30"));
37         jdisp(dj);
38         //  (昭和元年)
39         dj = JapaneseDate.from(LocalDate.parse("1926-12-25"));
40         jdisp(dj);
41         //  (平成元年)
42         dj = JapaneseDate.from(LocalDate.parse("1989-01-08"));
43         jdisp(dj);
44     }
45 }
```

**【解説】**

**21〜22 行目：** 和暦で日付・時刻を取得して，それを表示している．

**25〜28 行目：** 和暦で日付・時刻を取得して，それを表示している．
ただし，日付・時刻は LocalDateTime クラス（西暦）で取得し，from メソッドを使用して，
それを和暦に変換する形を取っている．（西暦から和暦への変換の例）

LocalData **クラスのオブジェクト** d **を和暦に変換**

**実行例：** JapaneseDate dj = JapaneseDate.from(d);

**注意：** 和暦は明治 6 年（1873 年）以降しか扱えない．

JapaneseDate オブジェクトから各種のフィールド値を取り出すには get メソッドを使用する．

JapaneseData **クラスのオブジェクト** dj **からの値の取得**

**実行例：** int y = dj.get(ChronoField.YEAR_OF_ERA);
解説： 和暦日付 dj から和暦年を取り出す．

get メソッドで値を取り出す場合，ChronoField クラスの必要なメンバを与える（表 9）ことで，必要なフィールドを選択する．

表 9: フィールド値の指定のためのメンバ（一部）

| フィールド名 | 意味 |
|---|---|
| ERA | 年号（戻り値：明治→-1, 大正→ 0, 昭和→ 1, 平成→ 2） |
| YEAR_OF_ERA | 年号内の紀元年 |
| YEAR | 西暦年 |

このサンプルプログラムを実行すると次のような表示になる．

《実行例：DateTest07.java》

```
直接取得：    Japanese Heisei 27-07-24
和暦に変換：Japanese Heisei 27-07-24
明治 (Meiji).6
大正 (Taisho).1
昭和 (Showa).1
平成 (Heisei).1
```

# 8 ラムダ式

Java SE 8 から新たに導入された文法事項に**ラムダ式** [8] がある．ラムダ式は必ずしも Java のプログラミングに不可欠なものではないが，ソースのコーディングを簡潔にすることができる場面がある．例えば，GUI の実装においてイベントハンドラを登録する場合などが 1 つの良い例となる．

## 8.1 イベントハンドラ登録への応用

ボタン（Button）へのイベントハンドラの登録の方法として，handle メソッドをオーバーライドする方法が次の例である．

[8] **ラムダ**： S 式を基本の式とする LISP 系言語の “LAMBDA” に由来する．LISP の LAMBDA は，関数の定義内容そのものを表す実体であり，λ算法（λ計算）と呼ばれる計算モデルを実現したものである．

**イベントハンドラ登録の例 (1)**

```
Button btn = new Button();
btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Hello World!");
    }
});
```

これは，与えられた ActionEvent オブジェクトに対するメソッド handle を新たに生成してボタンに登録する処理であるが，一見して読みにくい．

同様の処理をラムダ式（"->" による表現）を用いて実装すると次のようになる．

**イベントハンドラ登録の例 (2)**

```
Button btn = new Button();
btn.setOnAction( (ActionEvent e) -> {
    System.out.println("Hello World!");
});
```

この例では,「ActionEvent e が与えられた場合に System.out.println("Hello World!") を実行する」という処理を明示的に実体として扱い，それをイベントハンドラとしてボタンオブジェクトに与えていると解釈する．

このように，ラムダ式を用いた方が，意味の解釈においても実際のコーディングにおいても簡潔になることがある．

**ラムダ式**

**処理対象　->　処理内容**

この式は,「処理対象に対して処理内容を実行する」ということを意味する記号的実体である．

先に「2 GUI 構築の基本」で取り上げたサンプルプログラム FXsample01.java をラムダ式を用いて書き換えたものを次に挙げる．

**サンプルプログラム：** FXsample05.java

```
1  import javafx.application.*;
2  import javafx.event.*;
3  import javafx.scene.*;
4  import javafx.scene.control.*;
5  import javafx.scene.layout.*;
6  import javafx.stage.*;
7
8  public class FXsample05 extends Application {
9      @Override
10     public void start(Stage stg) {
11         AnchorPane root = new AnchorPane();
12         Scene scene = new Scene(root, 300, 100);
13         stg.setTitle("Hello World!");
14         stg.setScene(scene);
15
16         Button btn = new Button();
17         btn.setText("Say 'Hello World'");
18         btn.setOnAction( (ActionEvent e) -> {
19             System.out.println("Hello World!");
20         });
```

```
21          root.getChildren().add(btn);
22          btn.relocate(80,30);
23
24          stg.show();
25      }
26
27      public static void main(String[] args) {
28          launch(args);
29      }
30  }
```

### 8.2 関数型インターフェースでの応用

抽象メソッドを 1 つだけ持つインターフェースを**関数型インターフェース**という．関数型インターフェースの抽象メソッドにラムダ式を割り当てて処理を実行することができる．

関数型インターフェースを宣言するには，直前に@FunctionalInterface **アノテーション**を記述する．関数型インターフェースとラムダ式を用いて加算を実行するサンプルプログラム LmbdSample01.java を次に示す．

**サンプルプログラム：** LmbdSample01.java

```
1  //--- 関数型インターフェース（メソッドが1つだけ） ---
2  @FunctionalInterface
3  interface FunIf {
4      public double plus(double a, double b);
5  }
6
7
8  //--- メイン ---
9  public class LmbdSample01 {
10     public static void main( String argv[] ) {
11         double x;
12
13         //--- ラムダ式の割り当て ---
14         FunIf fif = (a,b) -> a+b;
15
16         //--- 実行 ---
17         x = fif.plus( 1.0, 2.0 );
18
19         //--- 出力 ---
20         System.out.println(x);
21     }
22 }
```

プログラム中に

FunIf fif = (a,b) -> a+b;

という記述がある．関数型インターフェースのインスタンス fif を生成して，これに加算処理を表すラムダ式を与えている．

これを実行すると，計算結果の「3.0」が表示される．

## 9 サウンドの再生（Java SE 7）

ここではサウンドを再生する方法（Java SE 7）について説明する．

## 9.1 基礎事項

**サウンドデータの入出力に関するクラス**

(1) AudioInputStream　サウンドデータの入力元となるクラス
(2) SourceDataLine　　サウンドデータの出力先となるクラス

(1) のクラスのオブジェクトから読み込んだサウンドデータを，(2) のクラスのオブジェクトに出力することでサウンドが再生される．読み込みには read メソッドを，出力には write メソッドを使用する．(詳しくは後述する．)

サウンド再生に必要なこれらのクラスを生成するに当たって，いくつかの属性情報を取得する必要がある．

**サウンド入出力に付随するクラス**

(3) AudioFormat　取り扱うサウンドの形式に関する各種の情報を取り扱うためのクラス
(4) DataLine　　システムのサウンド機能に関する各種の情報を取り扱うためのクラス

(3),(4) のクラスのオブジェクトは，出力のためのデータラインとなるオブジェクト (2) を生成するために必要となる．サウンドデータの流れや属性情報などの関連を図 45 に概略的に示す．

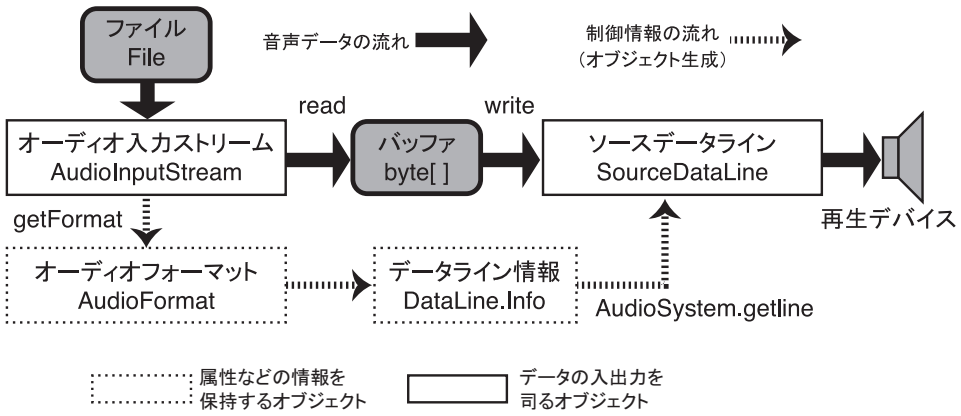



図 45: サウンド再生のシステム構成

図にある通り，サウンドデータは**オーディオ入力ストリーム**から読み込んで**バッファ**に保持し，それを**ソースデータライン**に書き出すことで再生する．

**【サウンドデータの基本的な形式】**

Java では基本的に **WAV 形式**のサウンドデータを扱う．WAV 形式は Microsoft 社と IBM 社が開発したサウンドデータ形式（サウンドフォーマット）であり，Microsoft 社の OS（Windows）において基本的な形式として採用されている．このフォーマットはデータ圧縮に関する仕様を持っているが，圧縮処理されていない PCM サウンドのデータを保存する場合に用いられることが多く，Windows に限らず多くの OS，アプリケーションにおいて，PCM サウンドデータの保存のために採用されている．

**【サウンド再生に必要なクラスライブラリ】**

データの入出力に read, write メソッドを使用するので java.io 以下の必要なライブラリをアプリケーションにインポートする必要がある．またサウンド関連のクラスが javax.sound.sampled 以下にあるので，必要なライブラリをインポートする必要がある．アプリケーションのソースプログラム中で次のように記述しておくと良い．

```
import java.io.*;
import javax.sound.sampled.*;
```

サウンドデータのファイルの内容を再生するための方法について，サンプルプログラムを示しながら具体的に説明する．

次に示すプログラム SoundTest01.java は，WAV 形式のサウンドデータ music001.wav の内容を読み込んで再生するプログラムである．

**[プログラム：SoundTest01.java]**

```
1  import java.io.*;
2  import javax.sound.sampled.*;
3
4  public class SoundTest01 {
5      public static void main(String[] args) throws Exception {
6
7          //  オブジェクトの宣言
8          AudioFormat         fmt;
9          DataLine.Info       inf;
10         SourceDataLine      dl;
11         File                af;
12         AudioInputStream    as;
13         byte[ ]             buf;
14         long                sizeDat;
15         int                 sizeBuf;
16
17         //  入力の準備
18         af = new File("music001.wav");  //  音声ファイルを開く
19         as = AudioSystem.getAudioInputStream(af);   //  入力用音声ストリーム作成
20         sizeBuf = as.available( );      //  入出力用バッファサイズ取得
21         buf = new byte[sizeBuf];        //  入出力用バッファ確保
22         fmt = as.getFormat();           //  入力用音声ストリームの情報取得
23
24         //  事前調査
25         sizeDat = fmt.getFrameSize()*as.getFrameLength();
26         System.out.println("総データ長: "+sizeDat+" bytes");
27         System.out.println("バッファ長: "+sizeBuf+" bytes");
28
29         //  出力の準備
30         inf = new DataLine.Info(SourceDataLine.class,fmt);
31         dl = (SourceDataLine)AudioSystem.getLine(inf);  //  出力用データライン作成
32         dl.open(fmt);                                   //  出力用データラインを開く
33         dl.start();     //  出力用データラインの使用開始
34
35         //  データ読込み＆出力
36         as.read(buf,0,sizeBuf);         //  音声データ読込み
37         System.out.println("再生開始");
38         dl.write(buf,0,(int)sizeDat);   //  出力用データラインに出力
39
40         //  終了処理
41         dl.drain();     //  出力用データラインにデータが残留しておれば排出
42         dl.close();     //  出力用データラインを閉じる
43         as.close();     //  入力用音声ストリームを閉じる
44         System.out.println("再生終了");
45     }
46 }
```

**【解説】**

これはサウンドデータファイルを再生するための最も基本的な方法を実装したプログラムである．

**8〜15 行目：** オーディオファイルの再生に必要な各種のオブジェクトや，制御に必要な変数などを宣言している．

**18 行目：** サウンドデータファイル “music001.wav” を File クラスのオブジェクト af に割り当てている．

**19 行目：** AudioSystem クラスのメソッド getAudioInputStream を用いて File クラスのオブジェクト af から AudioInputStream クラスのオブジェクト as を生成している．
以後はこの as がサウンドデータの入力元となる．

**20 - 21 行目：** オブジェクト as に対して available メソッドを実行することで，サウンド入出力に必要なバッファのサイズ（単位：バイト）を取得している．この値を元に 21 行目でバッファ用の記憶域を

確保している．

**22 行目：** オブジェクト as に対して getFormat メソッドを実行することでサウンドの形式に関する情報を取得し，それを AudioFormat クラスのオブジェクト fmt に設定している．

**25 行目：** オブジェクト fmt に対して getFrameSize メソッドを実行することでサウンドの 1 つの**フレームサイズ**（単位：バイト）を取得している．また，オブジェクト as に対して getFrameLength メソッドを実行することで，サウンドデータ全体を再生するのに必要な**フレームの個数**を取得している．すなわち，これら 2 つの数値の積が，再生に必要な有効バイト長となる．(25 行目)

**30 行目：** オブジェクト fmt からシステムのサウンド機能に関する情報を取得して DataLine.Ino のインスタンス inf に設定している．

**31 行目：** オブジェクト inf を元にして AudioSystem クラスのメソッド getLine を実行し，サウンド出力用のデータラインを取得してオブジェクト dl に設定している．
以後はこの dl がサウンドの出力先となる．

ここまでがサウンドデータ再生のための準備である．

**32 - 33 行目：** 出力用データラインをオープン（open メソッド）して，出力可能にして（start メソッド）いる．

**36 行目：** オーディオストリーム as から実際に音声データを読み込み，バッファ buf に格納している．

**read メソッドの説明：** as.read(buf,0,sizeBuf);
as からデータを読込み，buf の先頭（0 バイト目）から sizeBuf の長さだけ取得する．

**38 行目：** 出力データライン dl に buf の内容を出力する．

**write メソッドの説明：** dl.write(buf,0,sizeDat);
出力データライン dl に対して buf の内容を先頭（0 バイト目）から sizeDat の長さだけ出力する．

**41 - 42 行目：** 出力用データラインの終了処理．drain メソッドでデータラインに残留しているサウンドを排出し，close メソッドで閉じている．

**43 行目：** オーディオストリーム as を close メソッドを用いて閉じている．

このサンプルプログラム SoundTest01.java では，ファイルに保存されているサウンドデータを一度に読み取ってバッファに格納し，それを一度に出力データラインに送っている．再生するサウンドデータがあまり大きくない場合（数 MB 程度）は今回の方法でも特に問題はないと思われるが，データサイズが非常に大きい場合（数十 MB あるいはそれ以上）はバッファが確保できないケースも発生しうる．また，再生の開始や停止の処理において大きな遅延時間が生じる場合もあり，入出力用バッファのサイズはあまり大きくしない方がよいこともある．

次に，小さめのバッファで「入出力を小分けにして繰り返す」形のサウンド再生について考える．

ここでは，小さめのサイズの入出力用バッファを用いて「入出力を小分けにして繰り返す」形のサウンド再生について，サンプルプログラム SoundTest02.java を示しながら説明する．

**[プログラム：**SoundTest02.java**]**

```
1  import java.io.*;
2  import javax.sound.sampled.*;
3
4  public class SoundTest02 {
5      public static void main(String[] args) throws Exception {
6
7          //  オブジェクトの宣言
8          AudioFormat         fmt;
9          DataLine.Info       inf;
10         SourceDataLine      dl;
11         File                af;
```

```
12          AudioInputStream    as;
13          byte[]              buf;
14          long                sizeDat;
15          int                 sizeBuf;
16
17          //  入力の準備
18          af = new File("music001.wav");  //  音声ファイルを開く
19          as = AudioSystem.getAudioInputStream(af);   //  入力用音声ストリーム作成
20          sizeBuf = 262144;           //  入出力用バッファサイズ決定： 256KB
21          buf = new byte[sizeBuf];        //  入出力用バッファ確保
22          fmt = as.getFormat();           //  入力用音声ストリームの情報取得
23
24          //  事前調査
25          sizeDat = fmt.getFrameSize()*as.getFrameLength();
26          System.out.println("総データ長: "+sizeDat+" bytes");
27          System.out.println("バッファ長: "+sizeBuf+" bytes");
28
29          //  出力の準備
30          inf = new DataLine.Info(SourceDataLine.class,fmt);
31          dl = (SourceDataLine)AudioSystem.getLine(inf);  //  出力用データライン作成
32          dl.open(fmt);   //  出力用データラインを開く
33          dl.start();     //  出力用データラインの使用開始
34
35          //  データ読込み＆出力
36          dl.start();     //  出力用データラインの使用開始
37          System.out.println("再生開始");
38          int c = 0;      //  読み込んだ回数
39          int lenRead;    //  読み込んだデータ長（1回分）
40          int sizeRead = 0;   //  読み込んだデータ長（トータル）
41          while ( (lenRead = as.read(buf,0,buf.length)) >= 0 ) {  // 音声データ読込み
42              c++;
43              sizeRead += lenRead;
44              System.out.printf("(%02d) %d byte  (total: %d byte)\n",
45                      c,lenRead,sizeRead);
46              dl.write(buf,0,(int)lenRead);   //  出力用データラインに出力
47          }
48
49          //  終了処理
50          dl.drain(); //  出力用データラインにデータが残留しておれば排出
51          dl.close(); //  出力用データラインを閉じる
52          as.close(); //  入力用音声ストリームを閉じる
53          System.out.println("再生終了");
54      }
55  }
```

**【解説】**

処理の大まかな流れは SoundTest01.java とほぼ同じであるが，入出力用のバッファのサイズが異なる．今回のサンプルでは，20 行目で指定しているように 256KB のバッファサイズにしている．このバッファの大きさを超えるサイズのサウンドデータを再生するには，そのデータ全てを読込み終えるまで，入力と出力のサイクルを繰り返すことになる．この繰り返しを行っているのが 41〜47 行目の部分である．

このプログラムを実行したときの表示の例を次に示す．

《実行結果の例》

```
総データ長: 16906472 bytes
バッファ長: 262144 bytes
再生開始
(01) 262144 byte (total: 262144 byte)
(02) 262144 byte (total: 524288 byte)
(03) 262144 byte (total: 786432 byte)
                    :
```

この実行例は，サウンドを再生しながら読込みと出力を行う度にメッセージを表示しているものである．

## 9.2 実用的なサウンド再生

先に紹介したサンプルプログラム SoundTest01.java，SoundTest02.java は専らサウンドデータの読込みと再生を行うものであり，当然ながらこれらのプログラムは，サウンドの再生をしている間は他の処理はできない．

多くの実用的なアプリケーションプログラムの場合，サウンドの再生を伴う処理を行う際は，GUI の操作といった別の処理を平行して行うことができる．ここでは，サウンドデータの読込みと再生を独立したスレッド（Thread）で実行し，サウンド再生中も他の処理を可能にする例について説明する．

**[プログラム：SoundTest03.java]**

```
1  import java.io.*;
2  import javax.sound.sampled.*;
3
4  class Player extends Thread {
5
6      //  オブジェクトの宣言
7      public AudioFormat      fmt;
8      public DataLine.Info    inf;
9      public SourceDataLine   dl;
10     public File             af;
11     public AudioInputStream as;
12     public byte[]           buf;
13     public long             sizeDat;
14     public int              sizeBuf;
15     public boolean          playable = false;
16     public int c = 0;       //  読み込んだ回数
17     public int lenRead;     //  読み込んだデータ長（1回分）
18     public int sizeRead = 0;    //  読み込んだデータ長（トータル）
19
20     //  準備処理
21     public void ready( ) throws IOException,
22                 UnsupportedAudioFileException, LineUnavailableException {
23
24         //  入力の準備
25         af = new File("music001.wav");  //  音声ファイルを開く
26         as = AudioSystem.getAudioInputStream(af);   //  入力用音声ストリーム作成
27         sizeBuf = 262144;           //  入出力用バッファサイズ決定： 256KB
28         buf = new byte[sizeBuf];    //  入出力用バッファ確保
29         fmt = as.getFormat();       //  入力用音声ストリームの情報取得
30
31         //  事前調査
32         sizeDat = fmt.getFrameSize()*as.getFrameLength();
33         System.out.println("総データ長: "+sizeDat+" bytes");
34         System.out.println("バッファ長: "+sizeBuf+" bytes");
35
36         //  出力の準備
37         inf = new DataLine.Info(SourceDataLine.class,fmt);
38         dl = (SourceDataLine)AudioSystem.getLine(inf);  //  出力用データライン作成
39         dl.open(fmt);           //  出力用データラインを開く
40
41         playable = true;
42     }
43
44     //  サウンド再生
45     public void play( ) throws IOException {
46         //  データ読込み＆出力
47         dl.start();     //  出力用データラインの使用開始
48         System.out.println("再生開始");
49         while ( (lenRead = as.read(buf,0,buf.length)) >= 0 && playable ) {
50             c++;
51             sizeRead += lenRead;
52             dl.write(buf,0,(int)lenRead);   //  出力用データラインに出力
53         }
54     }
55
56     //  状態表示
57     public void report( ) {
```

```
58          System.out.printf("%02d回目I/O, %d byte 読込み  (total: %d byte)\n",
59                  c, lenRead, sizeRead);
60      }
61
62      //  終了処理
63      public void terminate( ) throws IOException {
64          //  終了処理
65          dl.drain();     //  出力用データラインにデータが残留しておれば排出
66          dl.close();     //  出力用データラインを閉じる
67          as.close();     //  入力用音声ストリームを閉じる
68          System.out.println("再生終了");
69          playable = false;
70      }
71
72      //  再生実行
73      @Override
74      public void run( ) {
75          if ( playable ) {
76              try {
77                  play( );
78                  terminate( );
79                  playable = false;
80              } catch (IOException e) {
81              }
82          } else {
83              System.out.println("再生の準備ができていません. ");
84          }
85      }
86  }
87
88  class SoundTest03 {
89      public static void main(String[] args) throws Exception {
90          //  サウンド再生
91          Player pl = new Player( );
92          pl.ready( );
93          pl.start( );
94
95          //  コマンドループ
96          int c;
97          while ( true ) {
98              System.out.print("再生中：'9'で終了 > ");
99              c = System.in.read( );
100             if ( c == '9' ) {
101                 pl.playable = false;
102                 break;
103             } else {
104                 pl.report( );
105             }
106         }
107         pl.report( );
108     }
109 }
```

**【解説】**

このプログラムでは Player という名前のクラスを定義して，その中にサウンドデータの読込み，再生，停止といった処理を行うための次のようなメソッドを記述している．

- **ready メソッド：** （21〜42 行目）オーディオ入力ストリーム，サウンド出力用のデータラインの準備．
- **play メソッド：** （45〜54 行目）サウンドデータの読込みと再生を繰り返すループを主とした処理．
- **terminate メソッド：** （63〜70 行目）サウンド再生終了後にオーディオ入力ストリームや出力用データラインを閉じる処理．
- **report メソッド：** （57〜60 行目）サウンド再生中に，処理経過（読み取った回数やデータサイズなど）を表示する処理．　（これは必須の処理ではない）

Player クラスは，Java のスレッドプログラミングのための Thread クラスを拡張したものであり，このクラスのイ

ンスタンスに対して start() メソッドを実行すると，Thread クラスの run() メソッドが起動する．run() メソッドは，拡張クラス側でオーバーライド（@Override）して具体的な処理内容を記述する．この run() メソッドは，main メソッドを実行するスレッドとは別の独自したスレッドとして実行される．

このプログラムの main メソッドの中では，Player クラスのインスタンス pl を生成している．これがオーディオ再生を単独で行う「サウンドプレーヤー」として振る舞う．この pl に対して ready() メソッドを実行すると，オーディオ入力ストリームとサウンド出力用のデータラインを準備する．その後で pl に対して start() メソッドを実行すると，オーバーライドした run() メソッドが起動して独立スレッドとなり，play() メソッドを使用してサウンドを再生する．

96〜107 行目の部分はコマンドループとなっており，サウンドの再生とは別のループである．Enter キーを押すと入出力の状況報告を表示する．また '9' を入力するとサウンド再生を中断してプログラム全体を終了する．

# 10 メディアデータの再生（Java FX 8）

Java FX 8 には音声や動画といったメディアデータを再生するための便利な API が用意されている．Java FX 8 でメディアデータを扱うための基本的なクラスを次に挙げる．

**1) Media**

これは，メディアデータそのものを意味するクラスであり，ファイルやアプリケーションのリソースとして用意されているデータを扱うためのものである．例えば，音楽データファイル "music1.mp3" がある場合を考える．まず，File クラスのオブジェクトを生成して，それに対してこのファイルを割り当てるには，

```
File mf = new File("music1.mp3");
```

とする．次に，この File オブジェクト mf に対して toURI と toString の 2 つのメソッドを用いて，データの URI [9] を取得する．例えば，

```
String uri = mf.toURI().toString();
```

とすると，先のファイルを表す URI が文字列（String）の形で uri に得られる．この URI を用いて次のようにして Media クラスのオブジェクトを生成すると，以後それがメディアデータを意味するオブジェクトとなる．

```
Media md = new Media(uri);
```

**注意）**Media クラスのオブジェクトを生成する際には適切に例外処理をする必要がある．

**2) MediaPlayer**

先に説明した Media クラスのオブジェクトは，単にメディアデータのリソースを保持するのみであり，そのオブジェクト自身にはメディアを再生する機能はない．Media データを実際に再生するためには，それを元にして MediaPlayer クラスのオブジェクトを生成する必要があり，再生や停止，一時停止といった操作は，この MediaPlyaer クラスのオブジェクトに対して行う．MediaPlayer クラスのオブジェクトを生成するには，例えば次のようにする．

```
MediaPlayer mp = new MediaPlayer(md);
```

以後，このオブジェクト mp に対して再生や停止などの操作を行う．

**注意）**MediaPlayer クラスのオブジェクトを生成する際には適切に例外処理をする必要がある．

Media，MediaPlayer は javafx.scene.media 以下のライブラリにある．

**【メディア再生に関する基本的な操作】**

MediaPlayer オブジェクトに対する基本的な操作（代表的なもの）を表 10 に挙げる．

**【Duration クラスについて】**

Duration クラスは，時刻や時間の範囲の値を扱うものである．このクラスのオブジェクトから,「年」「月」「日」「時」

[9] **URI**（Uniform Resource Identifier）： リソースとしてのデータを識別するための抽象的な表現であり，RFC 3986 に規定される形の文字列で表される．URL や URN を含む上位のリソース識別表現である．

表 10: MediaPlayer オブジェクトに対する基本的なメソッド

| メソッド | 機能 |
|---|---|
| play() | メディア再生の開始 |
| pause() | メディア再生の一時停止（停止位置の情報は保持される） |
| stop() | メディア再生の停止（停止位置の情報は失われる） |
| seek( Duration pos ) | 指定したメディアの時刻への再生位置の移動<br>再生位置を表す pos は Duration クラスのオブジェクトとして与える. |
| getCurrentTime() | 再生中のメディアの位置（時刻）の取得<br>結果は Duration クラスの値として得られる. |
| getRate()<br>setRate(double r) | 再生速度の比率（double 型）の取得<br>再生速度の比率を r に設定 |

「分」「秒」「ミリ秒」といったフィールドの値を取り出すことができる．例えば toMillis メソッドを用いて，

```
double pos = mp.getCurrentTime().toMillis();
```

とすると，メディア mp の再生中の時刻をミリ秒の単位で得ることができる．

**【サンプルプログラム】**

音楽データファイル "music1.mp3" を読み取って再生するサンプルプログラム Media00.java を考える．このプログラムは図 46 に示すような GUI を備えたもので,「Start」ボタンをクリックすると再生を開始し,「Pause」ボタンをクリックすると一時停止する．一時停止した場合は再度「Start」ボタンをクリックすることで再生を再開する.「Stop」ボタンをクリックすると再生を終了し，次に「Start」ボタンをクリックすると，メディアの先頭から再生する．

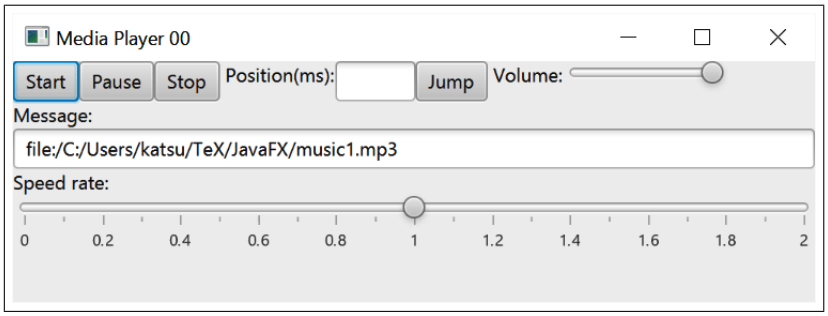



図 46: 簡易型メディアプレーヤ

「Position:」のフィールドにメディア再生位置をミリ秒単位の数値で入力して「Jump」ボタンをクリックすることで，指定した時刻に再生位置を移動することができる．
「Volume:」スライダは音量を,「Speed rate:」スライダは再生速度（比率）を調節するものである．
「Message:」フィールドには，プログラム動作中の各種メッセージ（ファイル名や再生位置など）を表示する．

**[プログラム：Media00.java]**

```
1  import java.io.*;
2
3  import javafx.application.*;
4  import javafx.stage.*;
5  import javafx.event.*;
6  import javafx.scene.*;
7  import javafx.scene.control.*;
8  import javafx.scene.layout.*;
9  import javafx.scene.media.*;
10 import javafx.beans.value.*;
11 import javafx.util.*;
```

```
12
13 public class Media00 extends Application {
14
15     //------ Pulic Variables ------
16     public static MediaPlayer mp;
17     public static double  pos, vol;
18     public static Duration dpos;
19
20     @Override
21     public void start(Stage stg) {
22         //------ GUI Building ------
23         VBox root = new VBox();
24         Scene scene = new Scene(root,500,150);
25         stg.setTitle("Media Player 00");
26         stg.setScene(scene);
27
28         HBox hbox = new HBox();
29         root.getChildren().add(hbox);
30
31         Button btnStart     = new Button("Start");
32         Button btnPause     = new Button("Pause");
33         Button btnStop      = new Button("Stop");
34         Label  lb1          = new Label(" Position(ms):");
35         TextField txtPos    = new TextField();
36         Button btnJmp       = new Button("Jump");
37         Label  lb2          = new Label(" Volume:");
38         Slider sldVol       = new Slider();
39         hbox.getChildren().addAll(
40             btnStart,btnPause,btnStop,lb1,
41             txtPos,btnJmp,lb2,sldVol);
42
43         txtPos.setPrefWidth(50.0);
44         sldVol.setMin(0.0);
45         sldVol.setMax(1.0);
46         sldVol.setValue(1.0);
47         sldVol.setPrefWidth(100.0);
48
49         Label lb3           = new Label("Message:");
50         TextField txtMsg    = new TextField();
51         Label lb4           = new Label("Speed rate:");
52         Slider sldSpd       = new Slider();
53         root.getChildren().addAll(lb3,txtMsg,lb4,sldSpd);
54
55         sldSpd.setShowTickMarks(true);
56         sldSpd.setMajorTickUnit(0.2);
57         sldSpd.setMinorTickCount(1);
58         sldSpd.setShowTickLabels(true);
59         sldSpd.setSnapToTicks(true);
60         sldSpd.setMin(0.0);
61         sldSpd.setMax(2.0);
62         sldSpd.setValue(1.0);
63
64         //------ Media Data ------
65         File mf = new File("music1.mp3");
66         String uri = mf.toURI().toString();
67
68         try {
69             Media md = new Media(uri);
70             try {
71                 txtMsg.setText(uri);
72                 mp = new MediaPlayer(md);
73             } catch (Exception mediaPlayerException) {
74                 txtMsg.setText("MediaPlayer!");
75             }
76         } catch (Exception mediaException) {
77             txtMsg.setText("Media!");
78         }
79
80         //------ Event Handling ------
```

```
81
82          // Button: Start
83          btnStart.setOnAction( (ActionEvent e) -> {
84              pos = mp.getCurrentTime().toMillis();
85              txtMsg.setText(String.valueOf(pos));
86              mp.play();
87          });
88
89          // Button: Pause
90          btnPause.setOnAction( (ActionEvent e) -> {
91              pos = mp.getCurrentTime().toMillis();
92              txtMsg.setText(String.valueOf(pos));
93              mp.pause();
94          });
95
96          // Button: Stop
97          btnStop.setOnAction( (ActionEvent e) -> {
98              pos = mp.getCurrentTime().toMillis();
99              txtMsg.setText(String.valueOf(pos));
100             mp.stop();
101         });
102
103         // Button: Jump
104         btnJmp.setOnAction( (ActionEvent e) -> {
105             pos = Double.parseDouble(txtPos.getText());
106             dpos = new Duration(pos);
107             mp.seek(dpos);
108         });
109
110         // Slider: Volume
111         sldVol.valueProperty().addListener(
112           (ObservableValue<? extends Number> ov,
113            Number old_val, Number new_val) -> {
114              mp.setVolume(new_val.doubleValue());
115              txtMsg.setText(String.valueOf(new_val.doubleValue()));
116         });
117
118         // Slider: Speed
119         sldSpd.valueProperty().addListener(
120           (ObservableValue<? extends Number> ov,
121            Number old_val, Number new_val) -> {
122             mp.setRate(new_val.doubleValue());
123             txtMsg.setText(String.valueOf(new_val.doubleValue()));
124         });
125
126         stg.show();
127     }
128
129     //------ Main ------
130     public static void main(String argv[]) {
131         launch(argv);
132     }
133 }
```

**解説：**

**1〜11 行目：** 必要なライブラリの読込み.

**23〜62 行目：** GUI の構築.

**65〜78 行目：** メディアの準備．処理が正常に終了するかどうかを判定するための例外処理（try〜catch）を行っている.

**83〜124 行目：** イベントハンドラの登録．特に 111〜124 行目にあるように，スライダ sld_Vol,sld_Spd の値が変化した瞬間に，それらの値をメディアの再生に反映させる形になっている.

GUI の値の変化が終了した後ではなく，変化が起こった瞬間を捉えるイベント処理に関しては「11.1.3 値の変化を検知するイベント」を参照のこと.

## 10.1 動画再生に関すること

動画データを再生する場合は，メディアデータを GUI に配置する必要があるが，そのためには MediaView クラスを利用する．このクラスのオブジェクトは MediaPlayer クラスのオブジェクトを保持するものであり，かつ，シーングラフの要素となるので，メディアを GUI の中に配置することが可能となる．

**例．**MediaPlayer クラスのオブジェクト mp を保持する MediaView オブジェクトの生成

MediaView mv = new MediaView(mp);

こうすることで，シーングラフに配置可能な MediaView オブジェクト mv が生成される．(setMediaPlayer メソッドを用いて MediaView オブジェクトに MediaPlayer オブジェクトを与えることもできる)

**【補足】**

Java FX 8 の MediaPlayer を用いたメディア再生は独立したスレッドとして実行されるので，Java SE 7 以前に比べてメディア再生処理の実装が簡便になっている．

# 11　付録

## 11.1　Java FX で利用できる GUI の部品（代表的なもの）

### 11.1.1　コンテナ：Containers

GUI を構築するための基盤ともいえるコンテナの内，代表的なものを図 47 に示す．


(a) アコーディオン　(b) タブペイン　(c) スクロールペイン

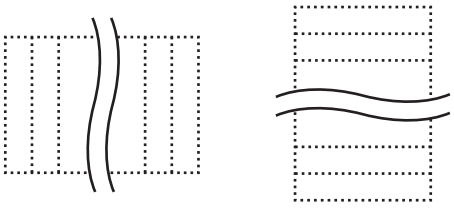
(d) 分割ペイン（横）　(e) 分割ペイン（縦）

図 47: コンテナ

この他にも VBox，HBox なるコンテナがあり，これらを用いることで縦横に領域を分割して GUI を構築することが容易になる．

#### 11.1.1.1　HBox,VBox を用いた GUI の配置

VBox，HBox（図 48）を用いることでウィンドウ内の GUI の配置をデザインする手法がある．



(a) HBox　(b) VBox

図 48: 水平・垂直のコンテナ

これらのコンテナは GUI の部品を縦横に配置するものであり，更に入れ子の構造（階層構造）にすることで高度な GUI を整然と構築する（例：図 49）ことが可能になる．

HBox,VBox を使用して GUI を配置する例をサンプルプログラムを示しながら説明する．

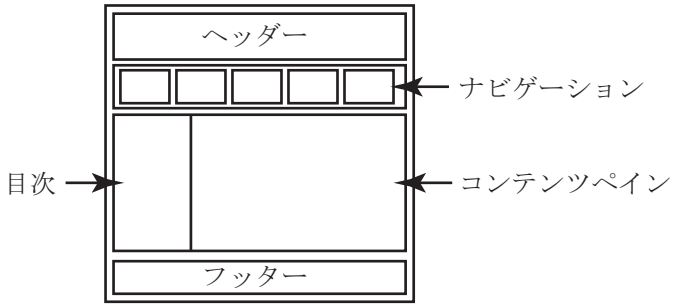



図 49: GUI デザインの例

HBox,VBox ををを使用して図 50 のような GUI を構築することを考える.

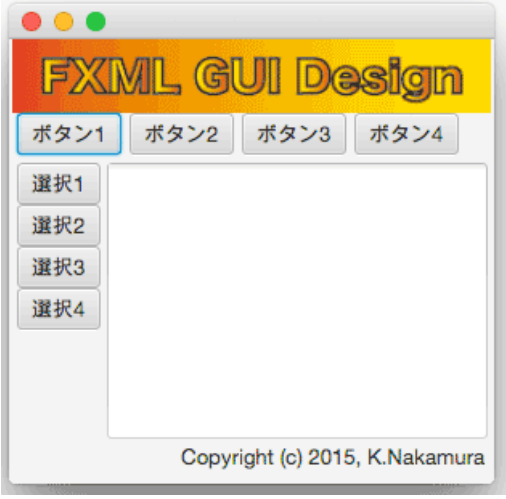



図 50: 作成する GUI

**ソースプログラム：** FXMLDocument.fxml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import java.lang.*?>
4  <?import javafx.geometry.*?>
5  <?import javafx.scene.control.*?>
6  <?import javafx.scene.image.*?>
7  <?import javafx.scene.layout.*?>
8
9  <VBox alignment="TOP_CENTER" maxHeight="-Infinity" maxWidth="-Infinity"
10       minHeight="-Infinity" minWidth="-Infinity"
11       prefHeight="277.0" prefWidth="302.0"
12       xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
13       fx:controller="guidesign01.FXMLDocumentController">
14     <children>
15         <ImageView fitHeight="46.0" fitWidth="418.0"
16                    pickOnBounds="true" preserveRatio="true">
17             <image><Image url="@GUIdesign_header.gif" /></image>
18         </ImageView>
19         <HBox prefHeight="31.0" prefWidth="302.0">
20             <children>
21                 <Button mnemonicParsing="false" text="ボタン1">
22                     <HBox.margin><Insets left="5.0" /></HBox.margin>
23                 </Button>
24                 <Button mnemonicParsing="false" text="ボタン2">
25                     <HBox.margin><Insets left="5.0" /></HBox.margin>
26                 </Button>
27                 <Button mnemonicParsing="false" text="ボタン3">
28                     <HBox.margin><Insets left="5.0" /></HBox.margin></Button>
29                 <Button mnemonicParsing="false" text="ボタン4">
30                     <HBox.margin><Insets left="5.0" /></HBox.margin></Button>
```

```
            </children>
        </HBox>
        <HBox prefHeight="172.0" prefWidth="302.0">
            <children>
                <VBox alignment="TOP_CENTER" prefHeight="172.0" prefWidth="61.0">
                    <children>
                        <Button mnemonicParsing="false" text="選択1" />
                        <Button mnemonicParsing="false" text="選択2" />
                        <Button mnemonicParsing="false" text="選択3" />
                        <Button mnemonicParsing="false" text="選択4" />
                    </children>
                </VBox>
            <TextArea prefHeight="172.0" prefWidth="237.0" />
            </children>
        </HBox>
        <Label alignment="CENTER_RIGHT" prefHeight="22.0" prefWidth="292.0"
               text="Copyright (c) 2015, K.Nakamura" />
    </children>
</VBox>
```

**ソースプログラム：** FXMLDocumentController.java

```
package guidesign01;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Label;

public class FXMLDocumentController implements Initializable {

    @FXML

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }
}
```

**ソースプログラム：** GUIdesign01.java

```
package guidesign01;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class GUIdesign01 extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("FXMLDocument.fxml"));

        Scene scene = new Scene(root);

        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

}
```

### 11.1.2 コントロール：Controlls

特に使用頻度の高いコントロール（GUI 部品）を表 11 にまとめる．

表 11: 使用頻度の高いコントロール

| クラス | 重要なメソッド | 解説 |
|---|---|---|
| Button<br>ボタン1 ボタン2 | | ボタン |
| CheckBox<br>項目1 項目2 | isSelected()<br>setSelected(論理値) | チェックされているかどうか（論理値）を返す．<br>true を引数として与えるとチェックされた状態になる．(false ならその逆) |
| RadioButton<br>選択1 選択2 | isSelected() | チェックされているかどうか（論理値）を返す．<br>setSelected メソッドが使える． |
| ToggleButton<br>スイッチ1 スイッチ2 | isSelected() | 押されているかどうか（論理値）を返す．<br>setSelected メソッドが使える． |
| ComboBox<br>選んでください<br>選択肢1<br>選択肢2<br>選択肢3 | getValue() | 選択されている項目値（登録した型）を返す．<br>選択項目の登録に関しては 11.1.2.1 参照 |
| ListView<br>リスト項目1<br>リスト項目2<br>リスト項目3 | getSelectionModel() | **選択モデル**の取得<br>得られた選択モデルに対して getSelectedItem() を実行すると，選択した項目が得られる．<br>選択項目の登録に関しては 11.1.2.1 参照 |
| Slider<br>0 25 50 75 100 | getValue()<br>setMin(最小値)<br>setMax(最大値)<br>setValue(値) | スライダの値（double）<br>スライダの左端の値の設定（double）<br>スライダの右端の値の設定（double）<br>スライダの値の設定（double） |
| TextField<br>プロンプト文字 | getText()<br>setPromptText(String value)<br>setText(String value) | 入力されたテキスト（String 型）を返す．<br>プロンプト文字列（value）を与える．<br>予めテキスト（value）を与える． |
| PasswordField<br>プロンプト文字 | getText() | 入力されたテキスト（String 型）を返す．<br>入力した文字の表示は隠蔽される．<br>TextField と同様に setText,<br>setPromptText メソッドが使用できる． |
| TextArea<br>ここにテキストを入力 | getText() | 入力されたテキスト（String 型）を返す．<br>TextField と同様に setText,<br>setPromptText メソッドが使用できる． |
| ImageView | setImage(画像オブジェクト)<br><br><br>setScaleX(横比率)<br>setScaleY(縦比率) | ビットマップ画像の表示．<br>‘画像オブジェクト’ には Image 型のオブジェクトを指定する．<br>表示する画像の縮尺を指定する．<br>比率は double 型で指定する． |

11.1.2.1　項目データを与える方法

表 11 にある ComboBox, ListView に項目データを与える方法について例を挙げて説明する．

**項目列の記述の例：FXML の場合**

下の例のように items タグを用いる．

**（ComboBox に項目列を与える例）**

```
1  <ComboBox>
2      <items>
3          <FXCollections fx:factory="observableArrayList">
4              <String fx:value="項目 1" />
5              <String fx:value="項目 2" />
6              <String fx:value="項目 3" />
7          </FXCollections>
8      </items>
9  </ComboBox>
```

items タグの中に更に FXCollections タグを挿入して fx:factory に "observableArrayList" を与え，各データ項目は

＜データ型　fx:value=（値）＞

という形で列挙する．

ListView に項目列を与える場合も同様に items タグを使用する．

**項目列の記述の例：Java のコードで記述する場合**

下の例のように ObservableList クラスのオブジェクトを用いる．

**（ComboBox に項目列を与える例）**

```
1  Combobox cbx;
2  ObservableList<String> lst;
3  lst = FXCollections.observableArrayList(
4      "項目 1",
5      "項目 2",
6      "項目 3"
7  );
8  cbx.setItems(lst);
```

この例では FXCollections クラスのクラスメソッド observableArrayList を用いて ObservableList クラスのオブジェクト lst を生成し，これを setItems メソッドを用いて ComboBox cbx に登録している．

ListView に項目列を与える場合も同様の方法を取る．

### 11.1.3　値の変化を検知するイベント

getValue メソッドで値が得られる GUI のコントロールには，値が変化した際のイベントハンドリングを設定することができる．これは，例えば Slider や ComboBox の値が変化した瞬間に何か処理を行うケースなどに有効な手段となる．

**例．**スライダの値が変化した瞬間に起動するイベントハンドラの設定

ObservableValue クラスを利用すると，値の変化の検出ができる．（このクラスはライブラリ javafx.beans.value 配下

にある）
例えば，Slider sld に値変化を受けるイベントハンドラを登録するには次のようにする．

```
sld.valueProperty().addListener(
    (ObservableValue〈? extends Number〉 ov, Number old_val, Number new_val) -〉{
            （実行する処理の記述）
  });
```

この例にある valueProperty は GUI の値を意味するものであり，これに対してイベントハンドラを設定する．GUI の値が変化した場合は，変化後の値が new_val に設定される．またこの例では，GUI の値は抽象的な数のクラスである Number クラスの値として扱われるので，これを適宜 int,double,long などの型に変換（表 12 参照）して使用する．

表 12: Number クラスから他の数値型への変換

| メソッド | 機能 |
|---|---|
| intValue() | int 型の整数に変換 |
| longValue() | long 型の整数に変換 |
| doubleValue() | double 型の数に変換 |

例えば，上記の new_val の値を double 型に変換するには，new_val.doubleValue() とする．

## 11.2 メニューの構築

プルダウンメニューの構築には MenuBar, Menu, MenuItem の 3 種類のクラスのオブジェクトを使用する．MenuBar はいわゆるメニューバー，Menu はプルダウンするメニュー，MenuItem はメニューの各項目を実現するものである．それらを組み合わせてプルダウンメニューを構築する．（図 51）

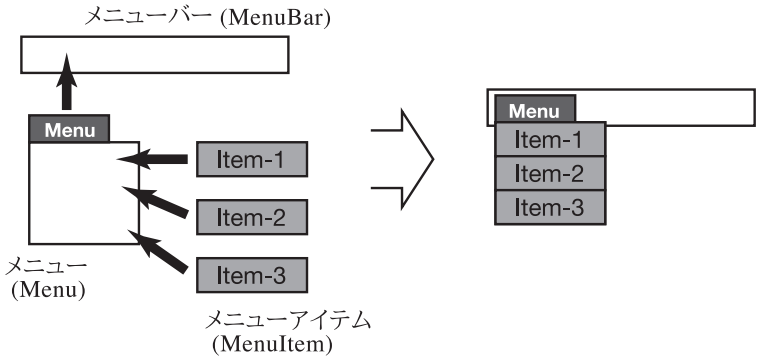



図 51: メニュー構築のイメージ

**メニューの構築**

**メニューバー生成の例：** MenuBar mb = new MenuBar();
（メニューバー mb を新たに生成）
**メニュー生成の例：** Menu mn = new Menu("Menu-1");
（"Menu-1" という見出しを持つメニュー mn を新たに生成）
これをメニューバー mb に登録するには，次のように getMenus, add の 2 つのメソッドを使用する．
mb.getMenus().add(mn);
**メニュー項目生成の例：** MenuItem mi1 = new MenuItem("Item-1");
（"Item-1" という見出しを持つメニュー項目 mi1 を新たに生成）
これをメニュー mn に登録するには，次のように getItems, add の 2 つのメソッドを使用する．
mn.getItems().add(mi1);

メニューを構築するプログラムの例を次に示す

**サンプルプログラム：**FXsample04.java

```
1  import javafx.collections.*;
2  import javafx.application.*;
3  import javafx.event.*;
4  import javafx.scene.*;
5  import javafx.scene.control.*;
6  import javafx.scene.layout.*;
7  import javafx.stage.*;
8
9  public class FXsample04 extends Application {
10     @Override
11     public void start(Stage stg) {
12         VBox root = new VBox();
13         Scene scene = new Scene(root, 300, 100);
14         stg.setTitle("Menu Test");
15         stg.setScene(scene);
16
17         MenuBar mb = new MenuBar();
18         Menu mn = new Menu("Menu-1");
19         mb.getMenus().add(mn);
20
21         MenuItem mi1 = new MenuItem("Item-1");
22         MenuItem mi2 = new MenuItem("Quit");
23
24         mn.getItems().add(mi1);
25         mn.getItems().add(mi2);
26
27         mi1.setOnAction(new EventHandler<ActionEvent>() {
28             @Override
29             public void handle(ActionEvent event) {
30                 System.out.println("mi1 is selected.");
31             }
32         });
33         mi2.setOnAction(new EventHandler<ActionEvent>() {
34             @Override
35             public void handle(ActionEvent event) {
36                 System.out.println("mi2 is selected.(quitting)");
37                 Platform.exit();
38             }
39         });
40         root.getChildren().add(mb);
41
42         stg.show();
43     }
44
45     public static void main(String[] args) {
46         launch(args);
47     }
48 }
```

このプログラムを実行すると図 52 のような GUI ができあがる．

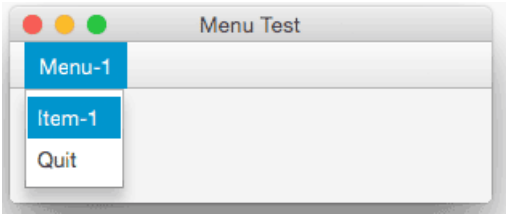



図 52: 実行結果

「Menu-1」から「Quit」を選択すると Platform.exit() が実行され，アプリケーションが終了する．

## 11.3 ファイル選択ダイアログの実装

実用的なアプリケーションプログラムでは，ファイルの読み込みや保存の際に，ファイルを選択するためのダイアログを表示してユーザに操作を促す．Java FX 8 にもファイル選択ダイアログを表示するための API が用意されている．ここでは，読込み用と保存用の 2 種類のファイル選択のダイアログについて説明する．

ファイル選択ダイアログは FileChooser クラスのオブジェクトである．ファイルの読込み用にダイアログを表示する場合は，このクラスのオブジェクトに対して showOpenDialog メソッドを，ファイルの保存用にダイアログを表示する場合は showSaveDialog メソッドを実行する．

**例： 読込み用のファイル選択ダイアログの表示**

```
FileChooser fc = new FileChooser();   //  ダイアログのオブジェクト fc を生成
fc.setTitle("Choose File:");          //  ダイアログにタイトルを設定
File f = fc.showOpenDialog(stg);      //  読込み用ダイアログとしてステージ stg に表示
```

このような方法で，既存のファイルを選択することができ，選択結果の戻り値は File クラスのオブジェクト（選択したファイル）である．また「キャンセル」ボタンを使用した場合など，ファイルが選択されなかった場合は戻り値は null となる．

保存用にファイルを指定するダイアログを表示する場合は，ダイアログオブジェクトに対して showSaveDialog メソッドを実行する．

読込み用，保存用それぞれのダイアログを表示するサンプルプログラム FileDialog1.java を示す．

**ソースプログラム：** FileDialog1.java

```
1  import java.io.*;
2
3  import javafx.application.*;
4  import javafx.event.*;
5  import javafx.stage.*;
6  import javafx.scene.*;
7  import javafx.scene.layout.*;
8  import javafx.scene.control.*;
9
10 public class FileDialog1 extends Application {
11     //  GUIの構築
12     @Override
13     public void start( Stage stg ) {
14         //  ステージとシーン
15         AnchorPane root = new AnchorPane();
16         Scene sc = new Scene(root,280,100);
17         stg.setScene(sc);
18         stg.setTitle("FileDialog1");
19
20         //  ファイルチューザ
21         FileChooser fc = new FileChooser();
22         fc.setTitle("Choose File:");
23
24         //  ファイル選択を表示するためのボタン（読込み用）
25         Button btn1 = new Button("Choose File");
26         btn1.setOnAction( (ActionEvent e)-> {
27             //  ファイルチューザの表示（読込み用）
28             File f = fc.showOpenDialog(stg);
29             //  ファイル選択後の処理
30             if ( f == null ) {
31                 System.out.println("ファイルは選択されませんでした．");
32             } else {
33                 String fn = f.getPath();
34                 System.out.println("ファイル \""+fn+"\" が選択されました．");
```

```
35              }
36          });
37          root.getChildren().add(btn1);
38          btn1.relocate(50,38);
39
40          //  ファイル選択を表示するためのボタン（保存用）
41          Button btn2 = new Button("Save File");
42          btn2.setOnAction( (ActionEvent e)-> {
43              //  ファイルチューザの表示（保存用）
44              File f = fc.showSaveDialog(stg);
45              //  ファイル選択後の処理
46              if ( f == null ) {
47                  System.out.println("ファイルは指定されませんでした. ");
48              } else {
49                  String fn = f.getPath();
50                  System.out.println("ファイル \""+fn+"\" が指定されました. ");
51              }
52          });
53          root.getChildren().add(btn2);
54          btn2.relocate(150,38);
55
56          stg.show();
57      }
58
59      //  終了処理
60      @Override
61      public void stop() throws Exception {
62          System.out.println("終了します. ");
63      }
64
65      //  メイン
66      public static void main( String argv[] ) {
67          launch( argv );
68      }
69  }
```

このプログラムを実行すると，図 53 のようなウィンドウが表示される．

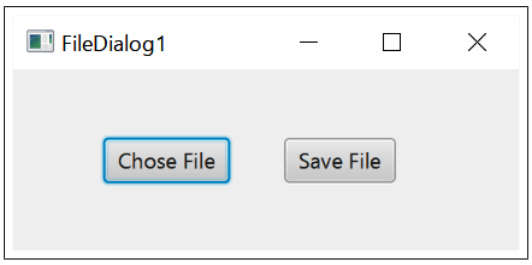



図 53: 実行したところ

ここで「Choose File」ボタンをクリックすると図 54 のようなダイアログが表示され，既存のファイルを選択することができる．また「Save File」ボタンをクリックすると図 55 のようなダイアログが表示され，保存用にディレクトリとファイル名を指定することができる．

## 11.4 Java FX 8 の重要なクラス

## 11.5 イベントについて

### 11.5.1 旧来の GUI で扱うイベント

表 13 参照のこと．

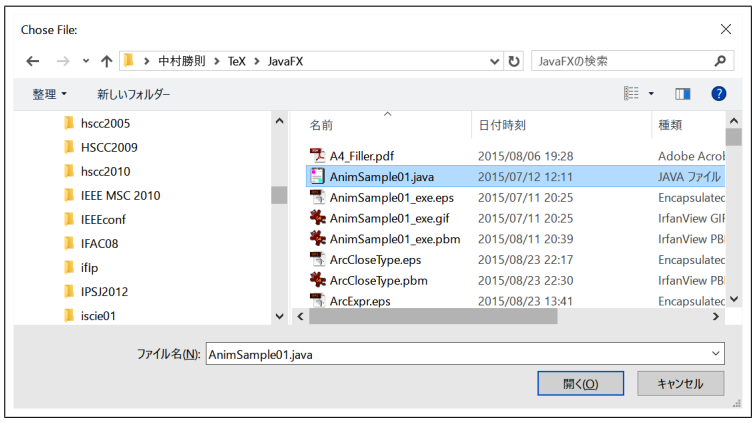



図 54: 読込むファイルを選択するダイアログ

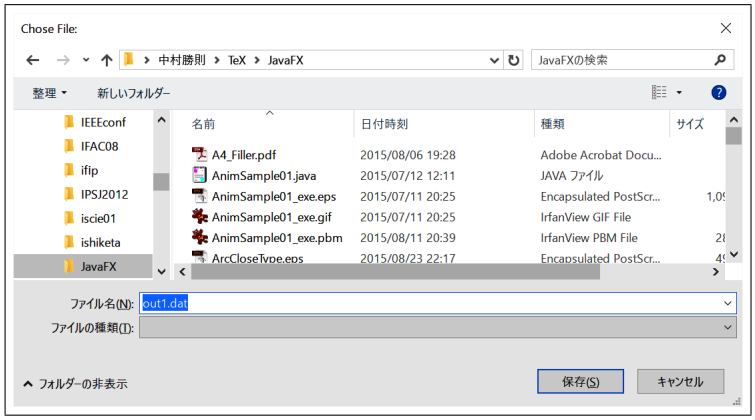



図 55: 保存するファイルを指定するダイアログ

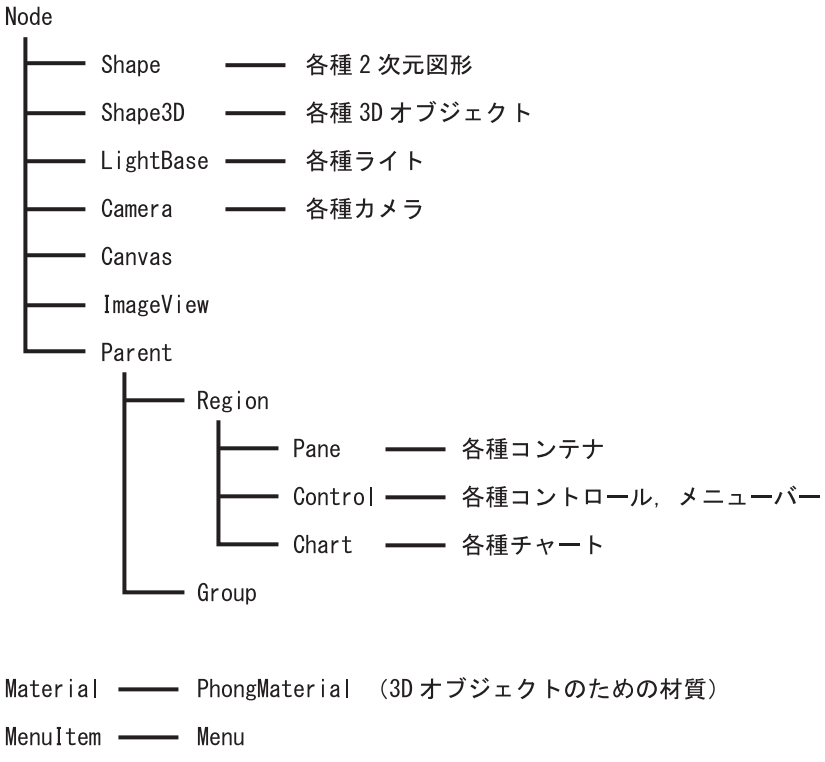



図 56: Java FX 8 の重要なクラス

### 11.5.2 タッチデバイスで扱うイベント

表 14 参照のこと.

表 13: Java FX 8 で扱うイベント（通常の GUI）

| **ウィンドウイベント** [ WindowEvent ] | |
|---|---|
| イベント | 説明 |
| onCloseRequest | 閉じる外部リクエストを受けると発生 |
| onHiding | 非表示になる直前に発生 |
| onHidden | 非表示になった直後に発生 |
| onShowing | 表示される直前に発生 |
| onShown | 表示された直後に発生 |
| **キーボードイベント** [ KeyEvent ] | |
| イベント | 説明 |
| onKeyPressed | キーが押されたことを検出 |
| onKeyReleased | キーが放されたことを検出 |
| onKeyTyped | キータイプ（押して放したこと）を検出 |
| **インプットメソッドイベント** [ InputMethodEvent ] | |
| イベント | 説明 |
| onInputMethodTextChanged | IME での入力変換の検出 |
| **マウスイベント** [ MouseEvent ] | |
| イベント | 説明 |
| onMouseClicked | マウスクリックの検出 |
| onMousePressed | マウスボタンが押されたことを検出 |
| onMouseReleased | マウスバタンが放されたことを検出 |
| onMouseDragged | マウスのドラッグ検出 |
| onMouseEntered | マウスが入ってきたことを検出 |
| onMouseExited | マウスが出て行ったことを検出 |
| onMouseMoved | マウスが動いたことを検出 |
| **マウスドラッグイベント** [ MouseDragEvent ] | |
| イベント | 説明 |
| onMouseDragEntered | マウスがドラッグして入ってきたことを検出 |
| onMouseDragExited | マウスがドラッグして出て行ったことを検出 |
| onMouseDragOver | その上でマウスドラッグが起こっていることを検出 |
| onMouseDragReleased | マウスドラッグが放されたことを検出 |
| onDragDetected | ドラッグ検出 |

表 14: Java FX 8 で扱うイベント（タッチデバイス系）

| **ドラッグイベント** [ DragEvent ] | |
|---|---|
| イベント | 説明 |
| onDragDone | ドラッグの終了（ドラッグされるオブジェクトが検出） |
| onDragDropped | ドラッグ＆ドロップされた（ドロップされるオブジェクトが検出） |
| onDragEntered | ドラッグして入ってきたことを検出 |
| onDragExited | ドラッグして出て行ったことを検出 |
| onDragOver | その上でドラッグが起こっていることを検出 |
| **ローテートイベント** [ RotateEvent ] | |
| イベント | 説明 |
| onRotationStarted | ローテーションアクションの開始を検出 |
| onRotate | ローテーションアクションを検出 |
| onRotationFinished | ローテーションアクションの終了を検出 |
| **スクロールイベント** [ ScrollEvent ] | |
| イベント | 説明 |
| onScrollStarted | スクロールアクションの開始を検出 |
| onScroll | スクロールアクションの検出 |
| onScrollFinished | スクロールアクションの終了を検出 |
| **スワイプイベント** [ SwipeEvent ] | |
| イベント | 説明 |
| onSwipeUp | 上へのスワイプを検出 |
| onSwipeDown | 下へのスワイプを検出 |
| onSwipeLeft | 左へのスワイプを検出 |
| onSwipeRight | 右へのスワイプを検出 |
| **タッチイベント** [ TouchEvent ] | |
| イベント | 説明 |
| onTouchMoved | タッチが動いたことを検出 |
| onTouchStationary | タッチが静止していることを検出 |
| onTouchPressed | タッチの開始を検出 |
| onTouchReleased | タッチの終了を検出 |
| **ズームイベント** [ ZoomEvent ] | |
| イベント | 説明 |
| onZoomStarted | ズームの開始を検出 |
| onZoomFinished | ズームの終了を検出 |
| onZoom | ズームアクションを検出 |

# 索 引