★8 GUI クラス（承前）

★8.2 JavaFX アプリケーションの基本 (p.170)

★8.2.2 UI コンポーネントクラスのプロパティ

Q1. 教科書 p.173–175「13.2.2 画像の表示」の説明を読んで，Hello.java を改造しよう（実習のページ参照）(☆ 1).

☆ 1) p.174 のプログラムサンプルには以下の誤植がある．(x) `JButton` → (o) `Button`

ウェブ上の JavaFX の API や教科書の関連箇所を見ると，コンストラクタやメソッドなどの従来からある項目とともに，**プロパティ**という見慣れない項目が挙がっている．プロパティは，オブジェクトの状態を表すデータを扱うための仕掛けである (☆ 2)(☆ 3)(☆ 4)．乱暴な言い方をすると，

「private なインスタンス変数 + その値を取得・設定するメソッド + α」

という代物である．

☆ 2) プロパティもスーパークラスから継承される．

☆ 3) 第 7 回講義資料の「カプセル化」の項も参照．

☆ 4) プロパティは Java の標準機能ではないが，より後発のオブジェクト指向言語（C#, Swift, etc.）ではデフォルトで備わっている．

プロパティの値を取得したり設定したりするメソッドには，ルールに従って名前が付けられる．あるクラスに hoge という名前のプロパティが備わっていたとすると，次のようになる．

- 値を取得するメソッド: `getHoge` (☆ 5)
- 値を設定するメソッド: `setHoge`
- boolean の値 (true/false) を取得するメソッド: `isHoge`
- プロパティ自身を取得するメソッド: `hogeProperty`

☆ 5) プロパティ名の先頭を大文字にしたものが使われる．この例では `hoge` → `Hoge`．他のメソッドも同様．

例えば，p.175 の ImageView クラスの API 仕様を見ると，「プロパティ」項に

```
DoubleProperty fitWidth
```

という記載がある．これは，このクラスのオブジェクトには fitWidth というプロパティが存在し，その値が double 型であることを表している．したがって，このプロパティの値を取得するメソッドは

と定義されている (☆ 6)．また，値を設定するメソッドは

のように戻り値なしで double 型の値を受け取るよう定義されている．API にはこのプロパティは「画像表示領域の幅を表す」と説明されており，これらのメソッドを使うことでこのプロパティにアクセスすることができる (☆ 7).

☆ 6) 戻り値型の前に付くアクセス修飾子等は省略して説明している．

☆ 7) 同様に，ImageView クラスの preserveRatio プロパティは boolean 型の値を持つ．この場合，値を取得するメソッドは `boolean isPreserveRatio()` ということになる．

★ 8.3 UI コンポーネントの配置 (p.175)

JavaFX のプログラムでは，UI コンポーネントは Node というクラスのサブクラスのオブジェクトであり，それらをシーングラフに追加することで画面に表示される．また，これらコンポーネントのクラスは Region クラスのサブクラスでもある (☆ 8)．Region クラスの機能を使えば，各コンポーネントの大きさ，背景，境界などを設定できる．

☆ 8) Region クラスは Node のサブクラスである．p.169 の図参照．

しかし，UI コンポーネントの大きさや位置をプログラム作成者が全て個別に指定するのは大変なので，コンポーネントの大まかな配置方針を指定したらあとは自動的に各コンポーネントの大きさや位置を制御してくれる仕組みが存在す．それが**レイアウトコンテナ**である．JavaFX では Pane クラスのサブクラスとして用意されている (☆ 9)．

☆ 9) javafx.scene.layout パッケージ内にある．

Hello.java の該当箇所を抜き出して，その使い方の一例を説明する．

Hello.java の一部

```
 :
13          BorderPane root = new BorderPane();  //レイアウトコンテナ生成

14          root.setTop(label);   //レイアウトコンテナへ配置

15          root.setLeft(b1);
16          root.setRight(b2);

17          Scene scene = new Scene(root);  //シーンに入れる
 :
```

レイアウトマネージャには配置方式の違いによって様々なものがあるが，教科書では，`FlowPane`, `BorderPane`, `VBox`, `HBox`, `GridPane` の 4 つを説明している．これらの詳しい使い方については，教科書 p.179–185 参照．

Q2. `Hello.java` の 13–16 行目を次の 1 行に書き換えたらどうなるか (☆ 10)(☆ 11)．

```
FlowPane root = new FlowPane(label, b1, b2);
```

また，コンストラクタの引数を `b1`, `label`, `b2` の順にしたらどうなるか．

☆ 10) この書き方は，p.179 の API 仕様の FlowPane のコンストラクタのうち一番下のものを使っている．詳しくは p.69「可変長引数」参照．

☆ 11) p.180 にあるように，次のように書いても同じことを実現可能．

```
FlowPane root = new
FlowPane();
root.getChildren().add(label);
root.getChildren().add(b1);
root.getChildren().add(b2);
```

`addAll` を使う手もある．

★ 8.4 JavaFX の UI コンポーネント (p.192)

JavaFX が提供する様々な UI コンポーネントのうちのごく一部を紹介する (☆12)．ここに出てくるクラスの詳しい使い方は p.192–202 参照．それ以外のものについては，JavaFX の API を参照（この科目のウェブページからたどれる）．

☆ 12) 教科書の該当箇所のうち，RadioButton, TextField, メニュー，Chart についてはスキップ．

★ 8.4.1 ラベル: Label (p.192)

文字列やアイコン画像を表示する．ボタンなどと違って入力は受け付けない．

★ 8.4.2 ボタン基本抽象クラス: ButtonBase (p.193)

ボタンやチェックボックスなどの，マウスクリックによるユーザインタフェースを実現するコンポーネントに共通の機能を定義したクラス．以下で説明している Button, CheckBox や，RadioButton などはこのクラスのサブクラスである（後のイベント処理の際にこのクラスの機能を利用する）．

★ 8.4.3 Button (p.194)

文字列とアイコンを貼付けられるボタン．ButtonBase のサブクラス．

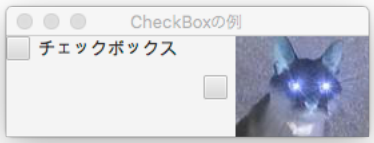
★ 8.4.4 CheckBox (p.195)

選択されている (true)，いない (false)，の 2 状態をもつボタン．ButtonBase のサブクラス．

CheckBoxExample.java の一部（p.196 のものとはちょっと違う）

```
 :
 8  public class CheckBoxExample extends Application {
 9      @Override
10      public void start(Stage pstage) {
11          CheckBox cbox1 = new CheckBox("チェックボックス");
12          CheckBox cbox2 = new CheckBox();
13          ImageView iv = new ImageView(new Image("blackuni.jpg"));
14          cbox2.setGraphic(iv);  // cbox2 の graphics プロパティを設定
15          HBox root = new HBox();
16          root.setSpacing(20);  // root の spacing プロパティを設定
17          root.getChildren().addAll(cbox1, cbox2); // コンテナに配置
18          Scene scene = new Scene(root);
 :
23      }
 :
28  }
 :
```

Q3. 14 行目と 15 行目の間に cbox2 の selected プロパティを true にする行を挿入したい．どのように書けばよいか．また，実行結果はどのように変化するか．



CheckBoxExample

ComboBoxExample

★ 8.4.5 ComboBox (p.197)

複数の項目の中から一つを選択するのに用いる.

ComboBoxExample.java の一部（p.198 のものとはちょっと違う）

```
 :
 7  public class ComboBoxExample extends Application {
 8      @Override
 9      public void start(Stage pstage) {
10          ComboBox<String> cb = new ComboBox<String>();
11          cb.getItems().addAll("寝る", "昼寝する", "ふて寝する", "暴れる");
12          BorderPane root = new BorderPane();
13          root.setCenter(cb);
14          Scene scene = new Scene(root);
 :
19      }
 :
24  }
 :
```

注意: ComboBox の <hoge> という書き方は,「ジェネリック」という機能を使ったものである. hoge の部分で, リストに並べるオブジェクトのクラスを指定する. 上記の例では String クラスのオブジェクトを並べている.「ジェネリック」については, 教科書第 11 章参照.

★9　イベント処理 (p.203)

★9.1　イベント処理のしくみ

ユーザが UI コンポーネントに対して操作（ボタンを押した，メニューの項目を選んだ，キー入力した，etc.）を行なった（これを「**イベント**が発生した」という）ときに，それに反応して行う処理を**イベント処理**（イベントハンドリング）(☆ 13) という．

☆ 13) イベント処理: event handling.

イベントが発生すると，そのイベントの情報（どんな操作がどの位置で行なわれたかなど）を伝えるためのオブジェクトが自動的に作られる．このオブジェクトは**イベントオブジェクト**と呼ばれ，Event クラスのサブクラスのオブジェクトである．p.204 の表 14.1 に示されているように，イベントオブジェクトのクラス（イベントクラス）には様々なものがある．

イベント処理を行うには，イベントに応答させるノード（シーングラフを構成するノード，つまり各種 UI コンポーネントやレイアウトコンテナ）に，**イベントハンドラ** (☆ 14) というオブジェクトをプロパティとして定義する必要がある．イベントハンドラのオブジェクト (☆ 15) には handle という名前のメソッドがあり，イベントが発生すると自動的に呼び出される．この handle メソッドを定義することでイベント処理を設定することができる．

☆ 14) イベントハンドラ: event handler.
☆ 15) 詳しくいうと，EventHandler インタフェイスを実装したオブジェクト．インタフェイスについては教科書第 9 章参照.

まとめると，イベント処理のプログラムを書く手順は次のようになる．

1. どんな種類のイベントを扱うのか，それには何というイベントクラスを使うのか，を調べる
2. どのイベントハンドラ用プロパティを定義すればよいかを調べる
3. イベントハンドラを定義する

★9.2　はじめてのイベント処理—Button のアクションイベント

別頁に示した HelloEvent.java は，次のようなイベント処理を実現している．

この例では，ボタンが押された時に発生するイベントの処理を行なっている．p.204 の表を見ると，次のことがわかる．

- 発生するイベントオブジェクトは ActionEvent クラスのものである
- それに対応するイベントハンドラプロパティは onAction である

これに基づいて，HelloEvent.java の 23 行目から 28 行目でイベントハンドラを定義している．この例では，`b1`, `b2` という 2 つの Button オブジェクトで発生したイベントを処理するので，それぞれの onAction プロパティに EventHandler オブジェクトを設定する（**イベントハンドラを設定する・登録する**という）．`b1` のイベントハンドラの登録は，次のように書かれている．

```
b1.setOnAction((event)-> { label.setText("こんにちは"); });
```

見慣れない書き方だが，setOnAction メソッドの引数には，**ラムダ式** (☆ 16) というものを用いて，クラス定義を省いてメソッドの引数とメソッドの中身だけを記述している．この場合のラムダ式は「イベントオブジェクトを event という名前の引数 (☆ 17) として受け取り { } の中を実行するメソッド」を表しており，これで EventHandler の handle メソッドを書いたことになっている．

イベントハンドラを登録するには，上記のやり方の他にも，addEventHandler という汎用のイベントハンドラ登録メソッドを使う方法などいろいろな書き方ができるのだが，説明は省略する（p.206–208 参照）．

☆ 16) ラムダ式についてはこの授業ではスキップした．教科書第 10 章.

☆ 17) この場合の引数の型は ActionEvent なので `(event)` は `(ActionEvent event)` と書くべきなのだが，型推論という機能のおかげで省略できる．詳しくは p.207.

HelloEvent.java

```
 1  import javafx.application.Application;
 2  import javafx.stage.*;           //Stage
 3  import javafx.scene.*;           //Scene
 4  import javafx.scene.control.*;   //Button, Label
 5  import javafx.scene.layout.*;    //BorderPane
 6  import javafx.event.*;           //ActionEvent
 7
 8  public class HelloEvent extends Application {
 9      @Override
10      public void start(Stage pstage) {
11          Button b1 = new Button("ごあいさつ"); // ごあいさつボタン
12          Button b2 = new Button("消去");       // 消去ボタン
13          Label label = new Label(" ");        // 最初は何も表示しない
14          BorderPane root = new BorderPane();
15          root.setTop(label);
16          root.setLeft(b1);
17          root.setRight(b2);
18          Scene scene = new Scene(root);
19          pstage.setScene(scene);
20          pstage.setTitle("HelloEvent");
21          pstage.sizeToScene();
22          pstage.show();


23          b1.setOnAction((event)-> {      // b1 に対し，ラムダ式でイベント処理の内容を設定
24              label.setText("こんにちは");  // label に文字列を設定
25          });



26          b2.setOnAction((event)-> {      // b2 に対し，ラムダ式でイベント処理の内容を設定
27              label.setText(" ");         // label の文字列を空白に設定
28          });
29      }
30      public static void main(String[] args) {
31          launch(args);
32      }
33  }
```