- イベント処理のしくみ
- はじめてのイベント処理—Button のアクションイベント
- CheckBox のアクションイベント

★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）(☆ 1) という．

☆ 1) 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** (☆ 2) 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 (☆ 3) 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．

☆2) イベントハンドラ: event handler.
☆ 3) 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** (☆ 4) 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 (☆ 5) 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）．

☆ 4) 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.

☆ 5) 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  }
```

★ 9.3 アクションイベントの処理

★ 9.3.1 CheckBox のアクションイベント

別のイベント処理を考えてみよう．この例では，チェックボックスのクリック時に発生するイベントの処理を行なう．p.204 の表を見ると，次のことがわかる．

- イベントオブジェクトのクラスは
- 対応するイベントハンドラプロパティは

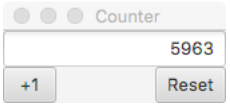
これらの情報に基づいてプログラムを書くと，以下のようなものになる．このプログラムは，**CSS（カスケーディング・スタイルシート）**を使ってスタイルを指定する例にもなっている（詳しくは p.186 参照）．

TrueUni3.java

```
   (import 文は省略)
 9  public class TrueUni3 extends Application {
10      @Override
11      public void start(Stage pstage) {
12          ImageView ivb = new ImageView(new Image("blackuni.jpg"));
13          ImageView ivw = new ImageView(new Image("whiteuni.jpg"));

14          Label label = new Label("", ivw);
15          CheckBox cb = new CheckBox("真のお姿");

16          HBox root = new HBox(label, cb);
17          root.getStyleClass().add("hbox");

18          Scene scene = new Scene(root);
19          scene.getStylesheets().add("trueUni3.css");

20          pstage.setScene(scene);
21          pstage.setTitle("TrueUni3");
22          pstage.sizeToScene();
23          pstage.show();

24          cb.setOnAction((event)-> {
25              if(cb.isSelected()) label.setGraphic(ivb);
26              else                label.setGraphic(ivw);
27          });
28      }
        (main メソッドは省略)
32  }
```

trueUni3.css（TrueUni3.class と同じディレクトリに置いておく）

```
.hbox{
    -fx-spacing: 20;           /* 子ノード間隔 */
    -fx-padding: 15 10 15 15;  /* 余白 */
    -fx-alignment: center;     /* 子ノードの位置合わせ */
}
```



練習問題　以下のプログラムを実行すると，右のように TextField（p.198 参照）と 2 つの Button が上下に並んだウィンドウが表示される．TextField には最初は 0 と表示されている．このプログラムを改造して，次のような動作をさせよう．

- 「+1」ボタンを押すと TextField の数字が 1 ずつ増えていく
- 「Reset」ボタンを押すとその数字が 0 になる

ヒント: 数を数えるための変数が必要だろう．イベント処理時に外から呼ばれるので，ローカル変数ではなくインスタンス変数としよう．

Counter.java

```
   (import 文のうちおなじみの部分は省略)
 6  import javafx.geometry.*;  // Pos
 7
 8
 9  public class Counter extends Application {
10
11      @Override
12      public void start(Stage pstage) {
13          TextField tf = new TextField(String.valueOf(0));
14          tf.setAlignment(Pos.CENTER_RIGHT);

15          Button bp = new Button(" +1 ");
16          Button br = new Button("Reset");
17          BorderPane root = new BorderPane();
18          root.setTop(tf);
19          root.setLeft(bp);
20          root.setRight(br);
21          Scene scene = new Scene(root);
22          pstage.setScene(scene);
23          pstage.setTitle("Counter");
24          pstage.sizeToScene();
25          pstage.show();
26

27

28

29

30

31

32

33
34      }
        (main メソッドは省略)
38  }
```