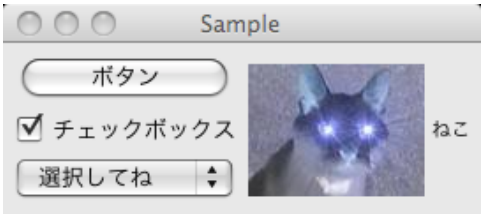
★ 8 GUI クラス　★ 8.1 GUI クラス　★ 8.2 JavaFX アプリケーションの基本
★ 8.3 UI コンポーネントの配置　★ 8.4 JavaFX の UI コンポーネント

★8　GUI クラス

★ 8.1　GUI クラス (p.165)

★ 8.1.1　GUI を備えたプログラムの構成

GUI（Graphical User Interface, グラフィカルユーザインターフェース）とは，右図のように，グラフィックスを用いた表示に対してユーザがマウスなどのポインティングデバイスを用いた直感的な操作（ボタンを押したりメニューを選択したり）を行うことでプログラムとやりとりできるようにした操作環境のことをいう．



教科書に記されているように，GUI を備えたプログラムは次のような動きをする必要がある．

- ボタンやメニューなどの **UI 部品（コンポーネント**と呼ぶ）をウィンドウ上に配置して表示する
- マウスやキーボードを通したユーザからの入力を感知して，それに応じたふるまいをする（**イベント**を処理するという）

今回は，このうち一つ目について学ぶ．二つ目は次回以降の予定．

★ 8.1.2　GUI ライブラリの種類

Java で GUI を備えたプログラムを開発するには，標準で提供されている **AWT パッケージ**，**Swing パッケージ** や **JavaFX パッケージ** を利用するのが一般的である．これらのパッケージには，ボタンやメニューなどのコンポーネントを表示するためのクラスや，マウスクリックやキー入力などのイベントを処理するためのクラスなどが含まれている．この授業では JavaFX を利用する．

- AWT(Abstract Window Toolkit): 基本的なグラフィックスやコンポーネントの描画や配置，イベント処理などを行うためのクラスの集まり
- Swing: AWT を拡張して作られた GUI 用のクラスの集まり．
- JavaFX: 最近新たに開発された．マルチタッチ (☆ 1) をサポートしタブレットやスマートフォンにも対応している，ウェブページのスタイル設定の技術である CSS(☆ 2) を利用して「見た目」と「処理」を分けて記述できる，等により，複雑な GUI を持つアプリの開発がより簡単になっている．

☆ 1) タッチスクリーンを 2 本指，3 本指で操作するなど．

☆ 2) CSS: Cascading Stype Sheets

★ 8.2　JavaFX アプリケーションの基本 (p.170)

★ 8.2.1　Hello.java

JavaFX を用いた例として，`Hello.java` というサンプルプログラム（ソースは最後の頁にあります）を説明する．実際にそのソースを見る前に，まずはその実行結果をよく観察しよう．また，このウィンドウがどのような構造をしているか観察しておこう．

Hello クラスの実行画面

観察
・タイトルバー（Hello と書いてある）やアイコン化／ウィンドウ最大化などのボタンのついたウィンドウが表示されている（注）
・「こんにちは」と表示されている．
・ 二つのボタンがあり，押している間だけ見た目が変わる．

注: ウインドウそのものの画面上での配置，タイトルバーやウインドウ枠の見た目を制御するのは，個々の GUI プログラムの役割ではなく，ウインドウマネージャと呼ばれるソフトウェアの役割である

ウィンドウの構造・コンポーネントの構成
(1) ウィンドウに対応した Stage クラスのオブジェクトが作られる
(2) そこに，コンポーネントを配置するための入れ物となる**コンテナ**（Scene クラスのオブジェクト）をおく
(3) そこに，様々なコンポーネントを配置する．コンポーネントの配置は，**シーングラフ**という階層構造で表される．

Hello.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
 7  public class Hello extends Application {


 8      @Override
 9      public void start(Stage pstage) {


10          Button b1 = new Button("ボタン 1");     //ボタン生成
11          Button b2 = new Button("ボタン２");     //ボタン生成
12          Label label = new Label("こんにちは"); //ラベル生成


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
18          pstage.setScene(scene);         //ステージ (ウィンドウ) にシーンを入れる
19          pstage.setTitle("Hello");       //ウィンドウタイトルの設定
20          pstage.sizeToScene();           //ウィンドウサイズをシーンに合わせる
21          pstage.show();                  //ウィンドウを表示
22      }


23      public static void main(String[] args) {
24          launch(args);
25      }
26  }
```

★ `@Override` と書いてあると，直後に定義するメソッドが何かの再定義であることがコンパイラに伝わる．なくてもよいが，書いておくとメソッド名間違いなどを指摘してもらえてうれしい（p.101）．

★ 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）(☆ 3).

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 (☆ 4)(☆ 5)(☆ 6)．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．

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` (☆ 7)
- 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`

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 (☆ 8)．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 (☆ 9).

☆ 3) 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`

☆ 4) プロパティもスーパークラスから継承される．

☆ 5) 第 7 回講義資料の「カプセル化」の項も参照.

☆ 6) 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．

☆ 7) 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.

☆ 8) 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．

☆ 9) 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 (☆ 10)．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．

☆ 10) 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 (☆ 11)．

☆ 11) 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 (☆ 12)(☆ 13).

```
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.

☆ 12) 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.

☆ 13) 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 (☆14)．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）．

☆ 14) 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．

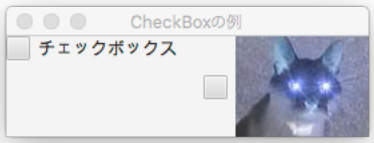
★ 8.4.4 CheckBox (p.195)

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 graphic 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.