- GUI クラス（承前）
- グラフィックスの描き方

★8　GUI クラス （承前）

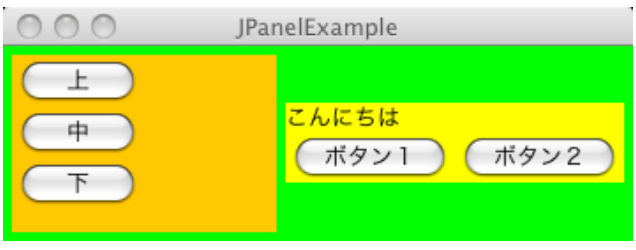
★8.4　前回のつづき

前回の GUI クラスの話のつづき．次のプログラムの実行結果を観察しよう．次のことに注目．

- 前回作成した Hello クラスを「利用」している（同じディレクトリに Hello.class が存在するとしている）
- コンポーネントの背景色やサイズを指定している

JPanelExample2.java の一部

```
 :
 4  public class JPanelExample2 extends JPanel{
 5
 6      JPanel p;

 7      Hello  h;
 8
 9      JPanelExample2(){

10          p = new JPanel();
11          p.setBackground(Color.orange);


12          p.setPreferredSize(new Dimension(150, 100));


13          p.setLayout(new BoxLayout(p, BoxLayout.Y_AXIS));
14          p.add(new JButton("上"));
15          p.add(new JButton("中"));
16          p.add(new JButton("下"));
17

18          h = new Hello();
19          h.setBackground(new Color(255, 255, 0));
20

21          this.setBackground(Color.green);
22          this.add(p);
23          this.add(h);
24      }
 :       (main メソッドは省略)
34  }
```



★9 グラフィックスの描き方

上記のプログラムの場合，グラフィックス描画は次の手順で行われる (☆ 1).

1. トップレベルコンテナ (JFrame) が自分自身を描く
2. トップレベルコンテナの描画域（コンテントペイン）の背景を描き，その上のコンポーネント (JPanelExample2) にそれ自身を描くよう指示する.
3. JPanelExample2 のコンポーネントが自分の背景を描き，その上のコンポーネント（変数 p の指す JPanel と変数 h の指す Hello の 2 つ）にそれら自身を描くよう指示する.
4. 変数 p の指すコンポーネントが背景を描き，その上のコンポーネント（3 つの JButton）にそれら自身を描くよう指示する.
5. それぞれの JButton が自分の背景を描き，ボタンのグラフィックスと文字列を描く.
6. 変数 h の指すコンポーネントが背景を描き，その上のコンポーネント（JLabel と 2 つの JButton）にそれら自身を描くよう指示する.
7. それらが自分の背景と自分自身を描く.

☆ 1) コンポーネントには境界線もあるがその描画の説明は省略している（p.134）. また，コンポーネントの背景は透明にすることもできる. その場合，下のコンポーネントの背景が見える.

これからわかるように，JPanel, JButton, JLabel のいずれのコンポーネントも,「自分の背景と自分自身を描く」→「自分の上にのっているコンポーネントに描画を指示する」という動作をしている. この一連の動作は，これらのスーパークラスである JComponent クラスのメソッドによって，次の手順で行われる.

1. paintComponent メソッドが背景を描く
2. paintBorder メソッドが境界線を描く
3. paintChildren メソッドが自分の上にのってるコンポーネントを描く

Swing に用意されたコンポーネントを利用するだけでなく，自分で任意のグラフィックスを描きたい場合は，適当なコンポーネント（通常は JPanel）の paintComponent メソッドを**再定義** (☆ 2)(☆ 3) し，そこにグラフィックス描画のプログラムを書く. 次のプログラムは，そのようにして画面に線画を描くものである.

☆ 2) スーパークラスのメソッドと同名で同一の引数の型のメソッドをサブクラスで定義すること. **オーバーライド** (override) ともいう. この授業では触れなかったが，教科書第 8 章に詳述されている.

☆ 3) **paintComponent の綴りを間違えると，(1) このクラスには paintComponent は存在しない → (2) スーパークラスである JPanel の paintComponent が呼ばれる → (3) 何のグラフィックスも表示されない，ということになるので要注意.**

CustomGraphics.java

```
 1  import java.awt.*;
 2  import javax.swing.*;

 3  public class CustomGraphics extends JPanel{

 4      public CustomGraphics(){
 5          setBackground(Color.white); // このパネルの背景色を設定
 6          setPreferredSize(new Dimension(250, 250)); // このパネルの好ましいサイズを設定
 7      }

 8      protected void paintComponent(Graphics g){

 9          super.paintComponent(g);      // スーパークラス (JPanel) の paintComponent の実行

10          g.drawLine(50, 50, 200, 200); // 線を描く
11          g.drawLine(200, 50, 50, 200); // 線を描く
12          g.drawRect(50, 50, 150, 150); // 四角を描く
13      }
 :      (main メソッドは省略)
22  }
```

8 行目から分かるように，paintComponent メソッドは，Graphics クラスの変数を引数にとる．Graphics クラスは，様々なグラフィックス（図形，文字，画像など）を描画する際に必要となる情報（これを**グラフィックスコンテキスト** (☆ 4) という）をまとめて保持するためのクラスである．

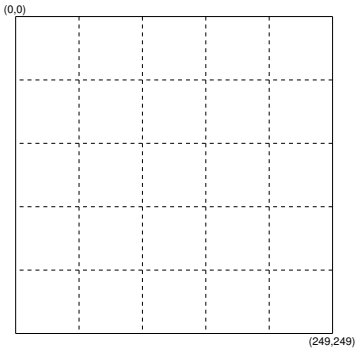
☆ 4) グラフィックスコンテキスト: graphics context. context は「文脈」の意.

図形の描画は，paintComponent メソッドに引数として渡された Graphics クラスのインスタンス（上記の例では変数 g が指すもの）のインスタンスメソッドを呼び出すことで行う．p.139–141 に，基本的な図形の描画のためのメソッドがまとめられている．

グラフィックスの座標系はコンポーネントごとに用意される．コンポーネントの左上隅が原点 $(0, 0)$ で，右に向かうほど x 座標の値が大きくなり，下に向かうほど y 座標の値が大きくなる．上記の例では，パネルのサイズを横 250 ピクセル，縦 250 ピクセルに指定しているので，このパネルの右下隅の座標は $(249, 249)$ となる．コンポーネントの大きさや位置は，p.138 上に説明されているメソッドを使って知ることができる．

Q1. 上記の `CustomGraphics.java` の 10 行目から 12 行目を次のものに置き換えるとどんな描画がされるか考えなさい．

```
g.setColor(new Color(255, 255, 0));
g.fillOval(50, 50, 150, 150);
g.setColor(Color.gray);
g.fillRect(125, 125, 100, 100);
g.setColor(Color.black);
g.drawLine(150, 150, 200, 175);
g.drawLine(150, 150, 175, 200);
```



上記の問題の例では，パネルの背景を描いた後に，大きな黄色い楕円を描く，次に灰色の長方形を描く…という順序で描画の指示が出されている．それでは，画面をよく見ているとこのように図形が順番に描画されていく様子が見られるかというと，実はそうではない．ユーザの目には，最後まで描画された結果がいきなり画面に現れてみえる．これは，Swing のコンポーネントは**ダブルバッファリング**と呼ばれる方法で描画を行うためである．この方法では，画面に表示される描画領域の他にもう一つ，画面には表示されない裏の描画領域（オフスクリーンバッファ）を確保しておき，描画はひとまずこちらに行う．そして，後でこの裏の描画領域の内容を表の描画領域に一気にコピーする．このようにすることで，アニメーション等のように次々に描画内容を変更していく場合でも，画面がちらつくのを防ぐことができる．

プログラムの実行結果を眺めていると，コンポーネントの描画は一度行ったらおしまいのように思える．つまり，paintComponent メソッドはプログラムの実行中に一度しか呼ばれないように思える．ところが実際には，このメソッドは次のようなことが起こる度に何回も呼び出されている (☆ 5)

1. 最初に表示された
2. コンポーネントの大きさが変化した
3. 別のウィンドウが重なって見えなくなった後で，再び見えるようになった (☆ 6)
4. ウィンドウが最小化（アイコン化）されて見えなくなった後で，再び表示された

したがって，一度だけ実行すればよい処理を paintComponent の中に記述したりしないよう注意する必要がある．例えば，画像ファイルを読み込んでその内容を描画するようなプログラムで，画像ファイルの読み込みを paintComponent の中で行うようにしていると，ウィンドウが最小化されたり別のウィンドウに隠された後で再表示される度にファイルの読み込みを行ってしまうことになる．また，描画の際に時間のかかる処理が必要なプログラムの場合は，上述のダブルバッファリングのような処理を自分でプログラミングするといった工夫が必要となることもある (☆ 7)．一方，描画内容を変更したときは，明示的に再描画の指示を出してやる必要がある．このような場合には，repaint メソッドを利用する (☆ 8)．

☆ 5) ためしに paintComponent の中で `println("ほげ〜♪");` してみるとよくわかる．

☆ 6) p.138 の脚注に記されているように，Java SE 6 以降では，この場合には paintComponent は呼ばれなくなった．

☆ 7) p.144「12.5 節 オフスクリーンイメージ」参照．

☆ 8) p.138 参照．

宿題？

- 授業中に解説されなかった Q を解いてみよう．
- 次回は 13 章です（12 章の残りはスキップ）．あらかじめ読んでおくこと．例題のプログラムを作成し実行してみておくこと．