上級プログラミング2（第3回）

工学部　情報工学科
木村昌臣

今日のテーマ

- GUIプログラミング入門
  - AWT

JavaでGUIを作る方法(API)

- AWT
  - Abstract Window Toolkit
  - GUIをつくるクラス群を提供 (基本!)
  - OSによらない外観
    - 逆にいえばOSネイティブなlook and feelではない
- Swing
  - AWTをもとに
- JavaFX
  - JDK1.8からの新しいクライアントUIライブラリ

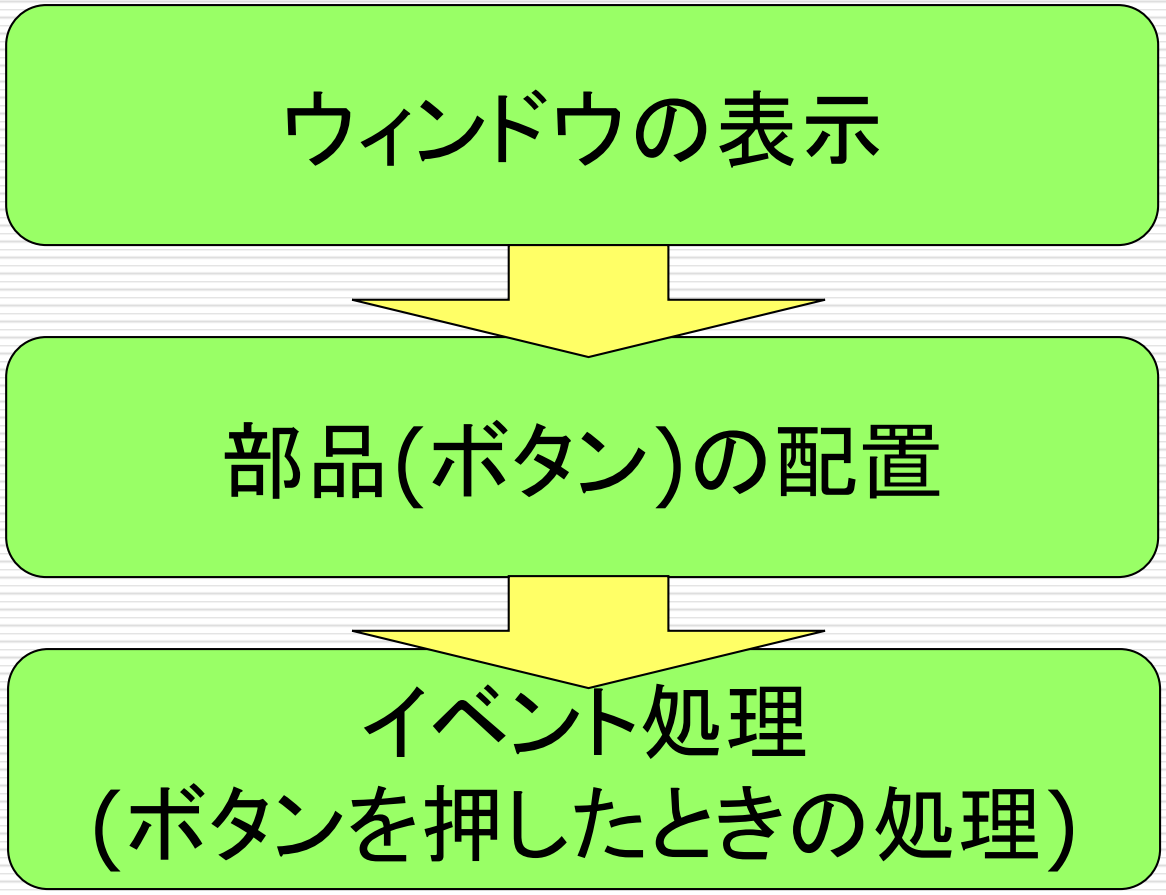
JavaによるGUIプログラミング入門

カウンタを作ろう

作りたいプログラムの仕様

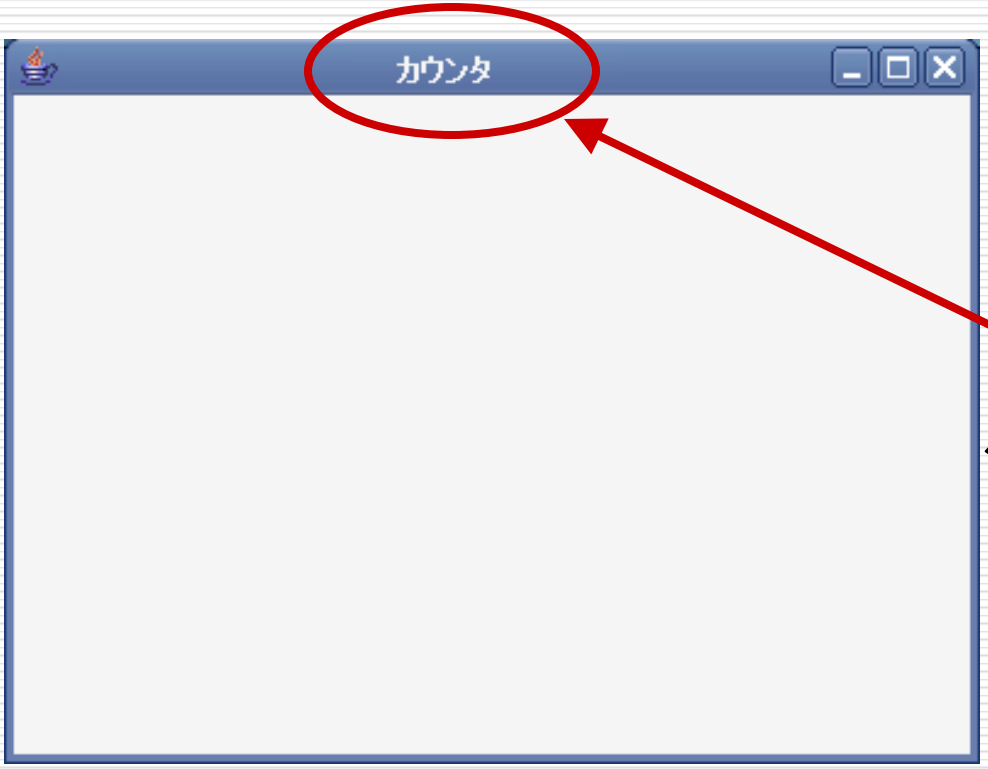
- GUIのウィンドウの上で操作
- テキストボックス（テキストフィールド）に数字を表示
- ボタンを押すとテキストボックスの数字の値が1ずつ増加

説明の流れ



フレーム（枠）ウィンドウの表示

- まずは、こんなのを表示したい



ウィンドウを表示するだけのプログラム

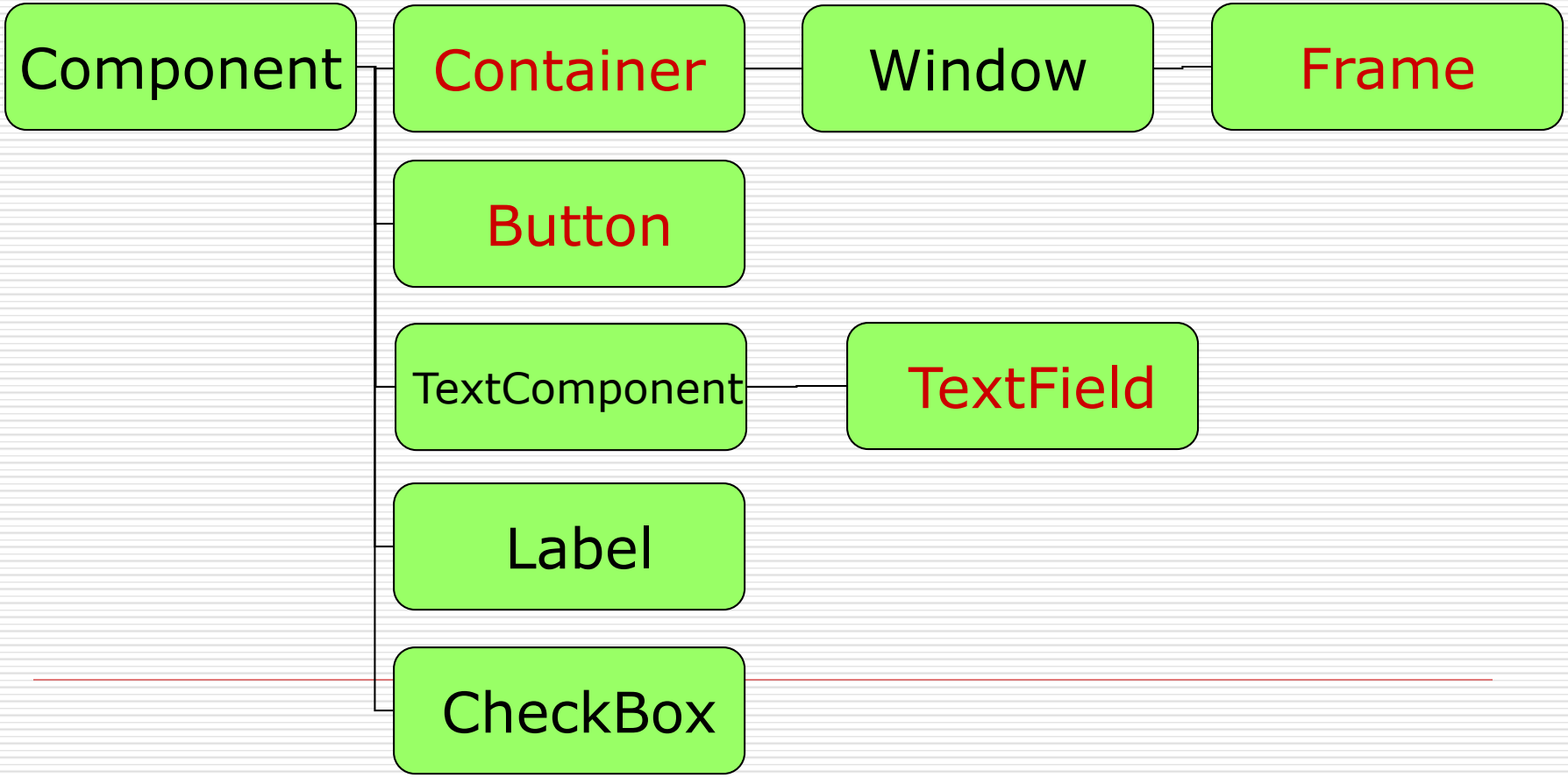
```
import java.awt.*; // Frameはjava.awtパッケージ内

class MyFrame{

        public static void main(String[] args){
                Frame frame=new Frame("カウンタ");
                frame.setSize(400,300);
                frame.setVisible(true);    //  表示させる
        }
}
```

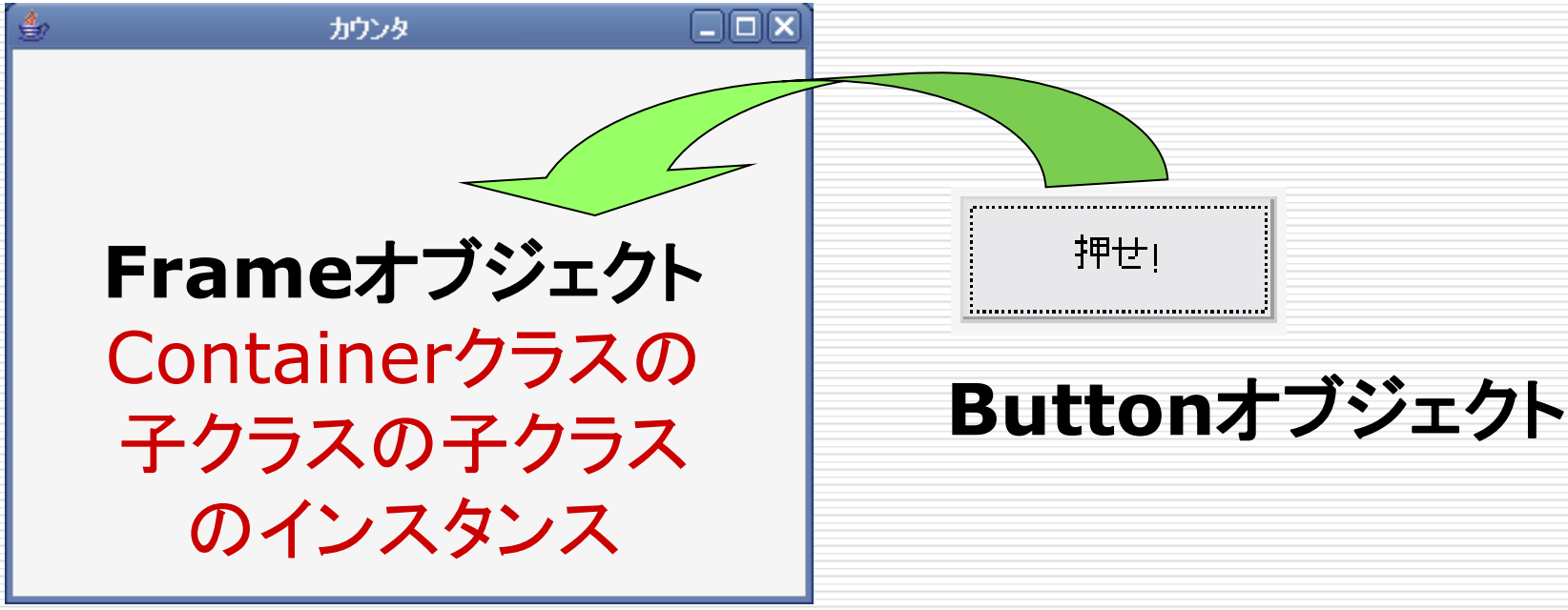
GUIを構成する主要なクラス群(AWT)

java.awtパッケージに含まれるクラスを使う。
Scrollbarなど他にもいくつか存在する。



Containerクラス

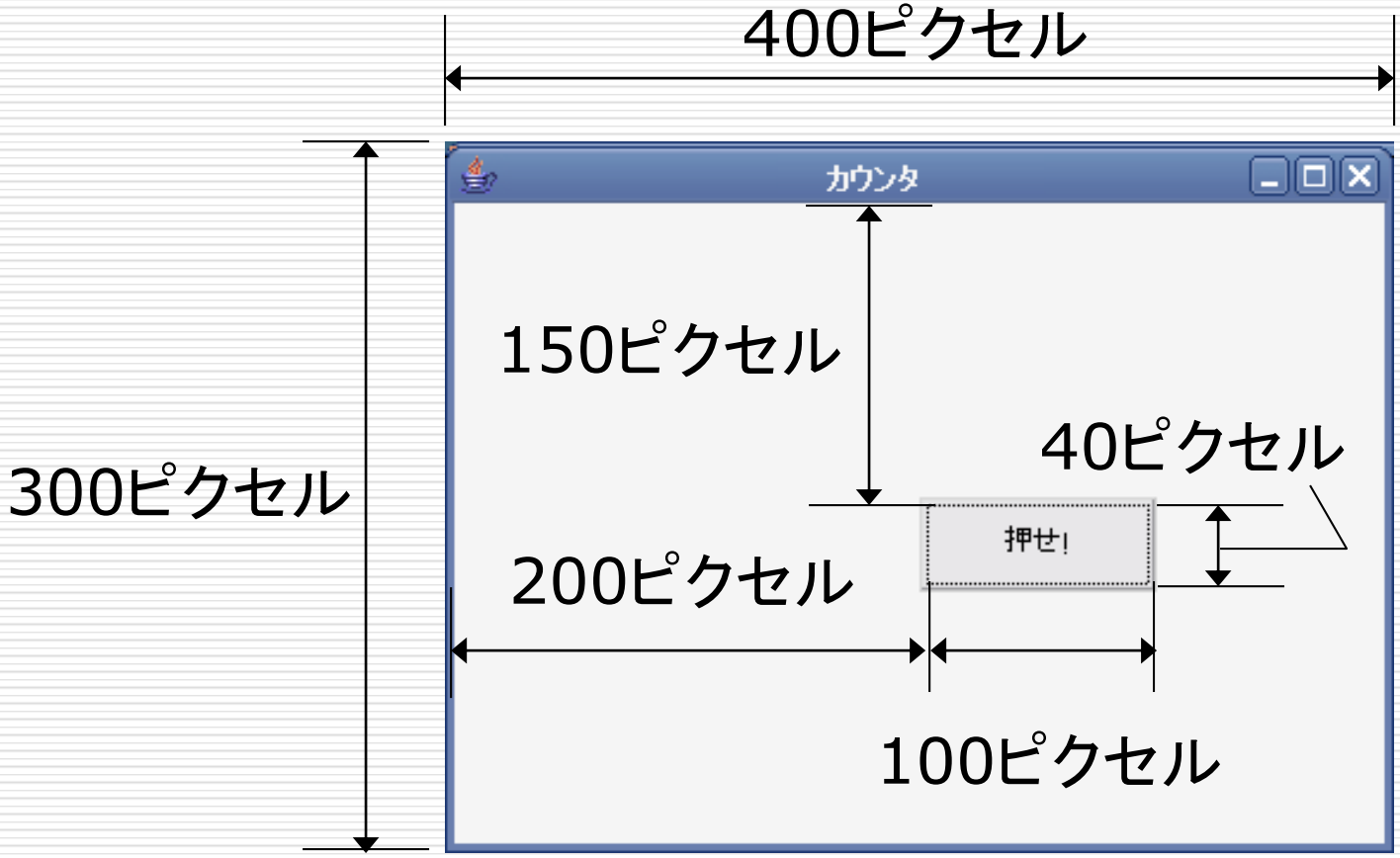
Containerクラス（もしくはその子孫のクラス）は、
他のGUIコンポーネントを中に配置できる



ボタンなどの部品をフレームに配置

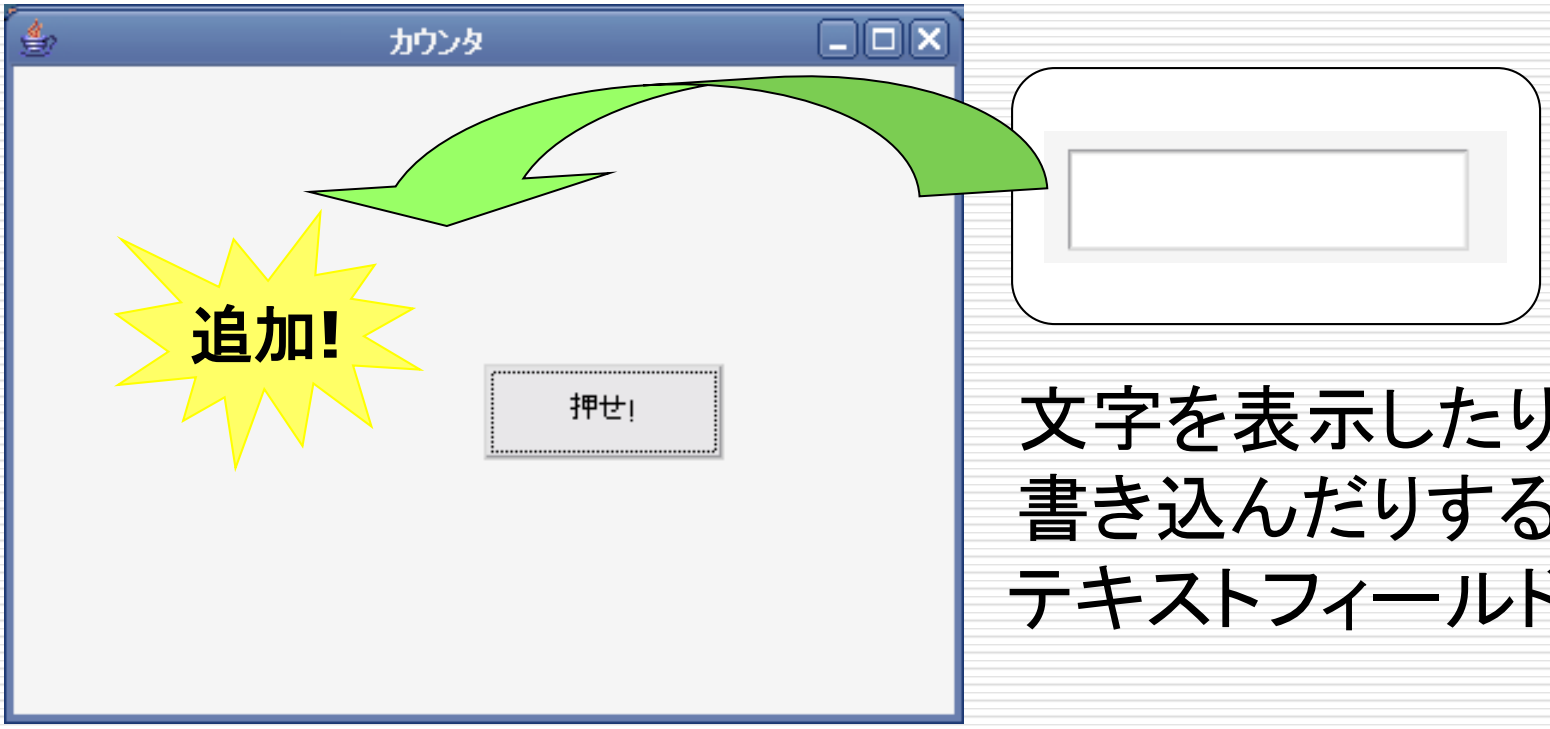
```
import java.awt.*;
class MyContainer{
        public static void main(String[] args){
                Frame frame=new Frame("カウンタ");
                    frame.setSize(400,300);
                    frame.setLayout(null);
                Button button=new Button("押せ!");
                    button.setLocation(200,150);
                    button.setSize(100,40);
                frame.add(button);
                    frame.setVisible(true);
        }
}
```

実行結果



frame.setLayout(null)でボタンの自由配置を実現している

今度はテキストフィールドを追加してみよう



文字を表示したり
書き込んだりする
テキストフィールド

テキストフィールドの追加

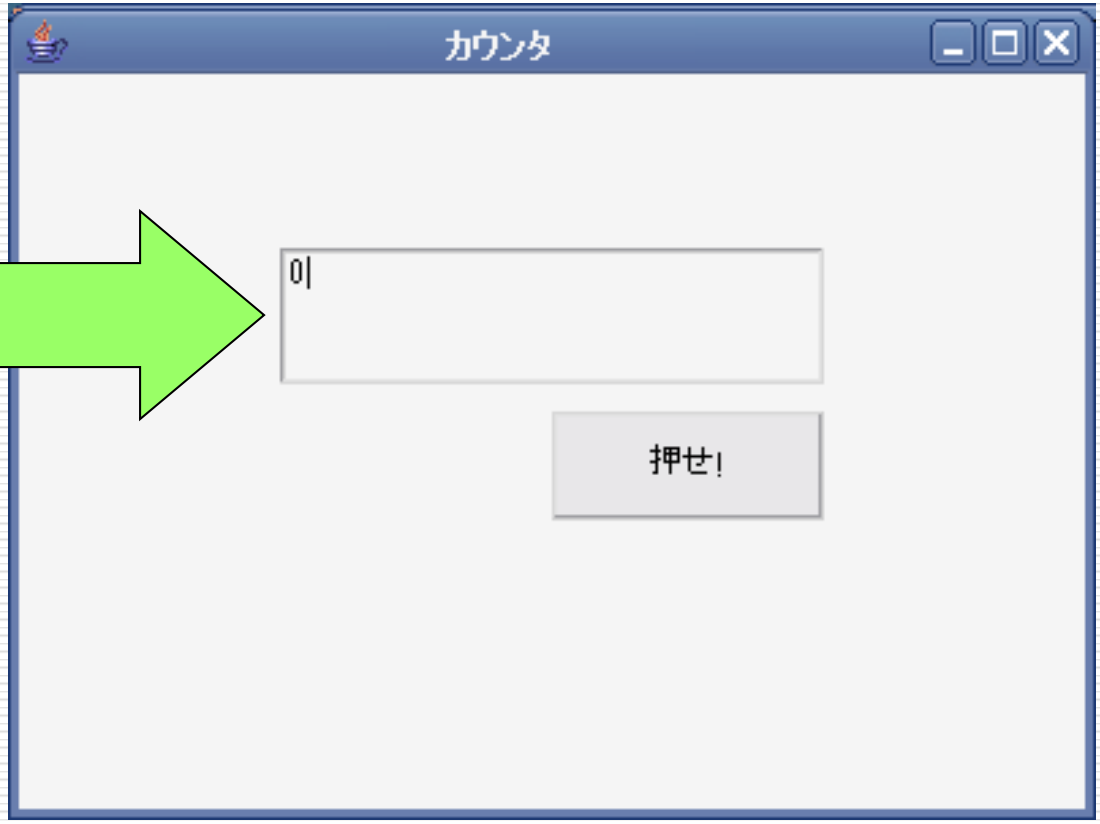
```
import java.awt.*;
class MyContainer2{
        public static void main(String[] args){

                ・・・・FrameやButton部分は略・・・・

                TextField txt=new TextField("0");
                        txt.setLocation(100,90);
                        txt.setSize(200,50);
                frame.add(button);
                frame.add(txt);
                frame.setVisible(true);
        }
}
```

実行結果

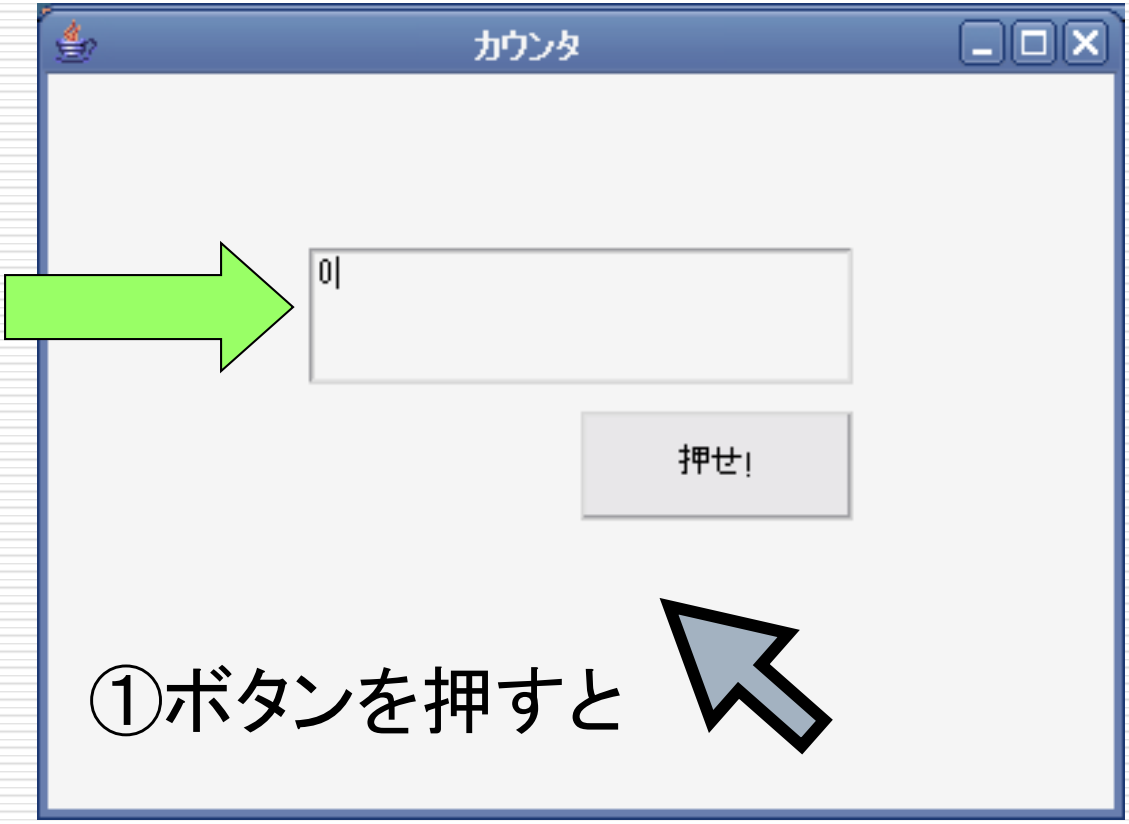
テキストフィールド
がついた



ただし、
まだこの段階では
ボタンを押しても
何も起こらない

やりたいこと

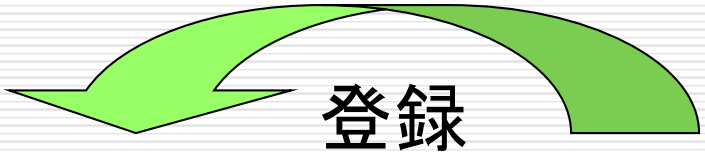
②ここの数が
増える



イベント

- ユーザー側からのアクション（イベント）が発生したら特定の処理を行うプログラムをイベント駆動型と呼ぶ

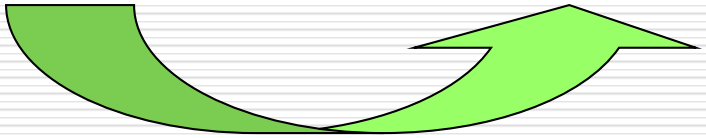
イベント駆動型プログラムの作り方



Button
オブジェクトなど

イベントリスナー

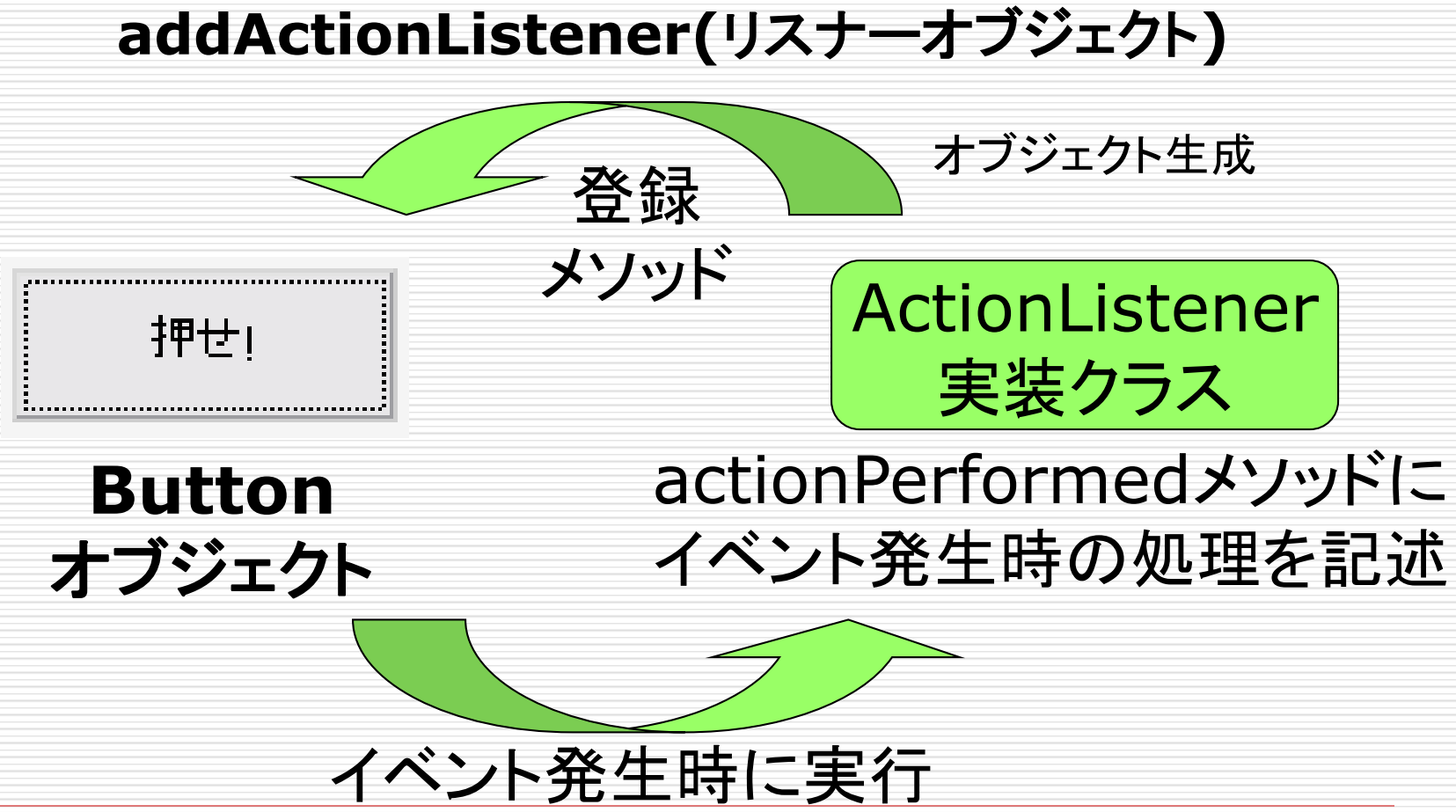
イベントが発生したら
行う処理を定義するクラス



イベントの種類

イベント名	説明	主なイベントソース
Action	ボタンのクリックなど部品に対する操作時に発生	ボタン、メニュー等
Mouse	マウスをクリックしたりドラッグしたりする時に発生	Componentクラスのサブクラス
Key	キーボードのキー入力時に発生	Componentクラスのサブクラス
Window	ウインドウの状態が変わったときに発生	Windowクラスのサブクラス

アクションイベントの場合



イベントリスナー

```
import java.awt.*;
import java.awt.event.*;
class OseActionListener implements ActionListener{
        TextField txt;
        OseActionListener(TextField txt){
                this.txt=txt;
        }
        public void actionPerformed(ActionEvent e){
                String tmp=this.txt.getText();
                int cnt=Integer.parseInt(tmp);
                this.txt.setText(String.valueOf(cnt+1));
        }
}
```

イベントリスナーの登録

```
import java.awt.*;
class MyCount{
    public static void main(String[] args){
        ・・・・Frame、Button、TextField定義部分は略・・・・

            Button button=new Button("押せ!");
            TextField txt=new TextField("0");
            OseActionListener  lsnr=
                        new OseActionListener(txt)
            button.addActionListener(lsnr);

        ・・・・以降略・・・・
        }
}
```

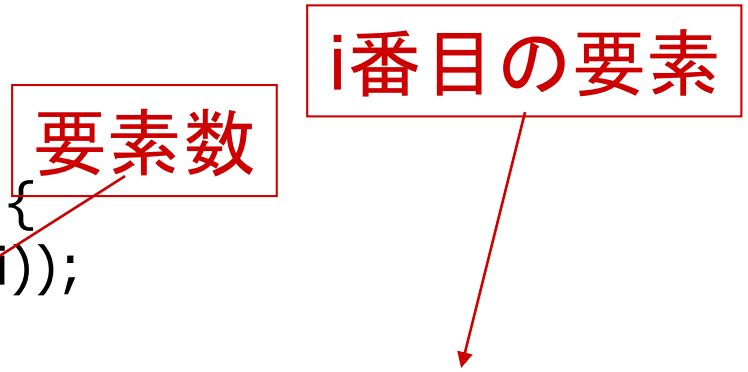
描画

簡易ドローソフトを作る

前提知識

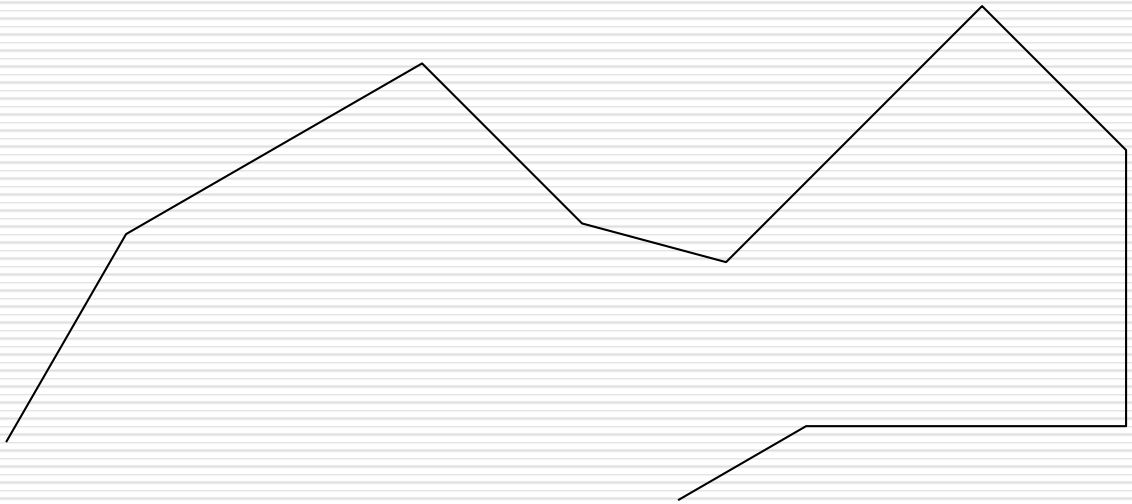
- ArrayListクラス　(java.utilパッケージ)
  - リスト
    - 配列は要素数が固定だが、こちらは可変
  - 使い方（ジェネリクスGenericsを使用している）

```
ArrayList<String> v= new ArrayList<String>();
v.add("あ");
v.add("い");
v.add("う");
for(int i=0; i< v.size(); i++){
    System.out.println(v.get(i));
}
```



簡易ドローソフトの仕様

- マウスをクリックした点を直線で結んでいく

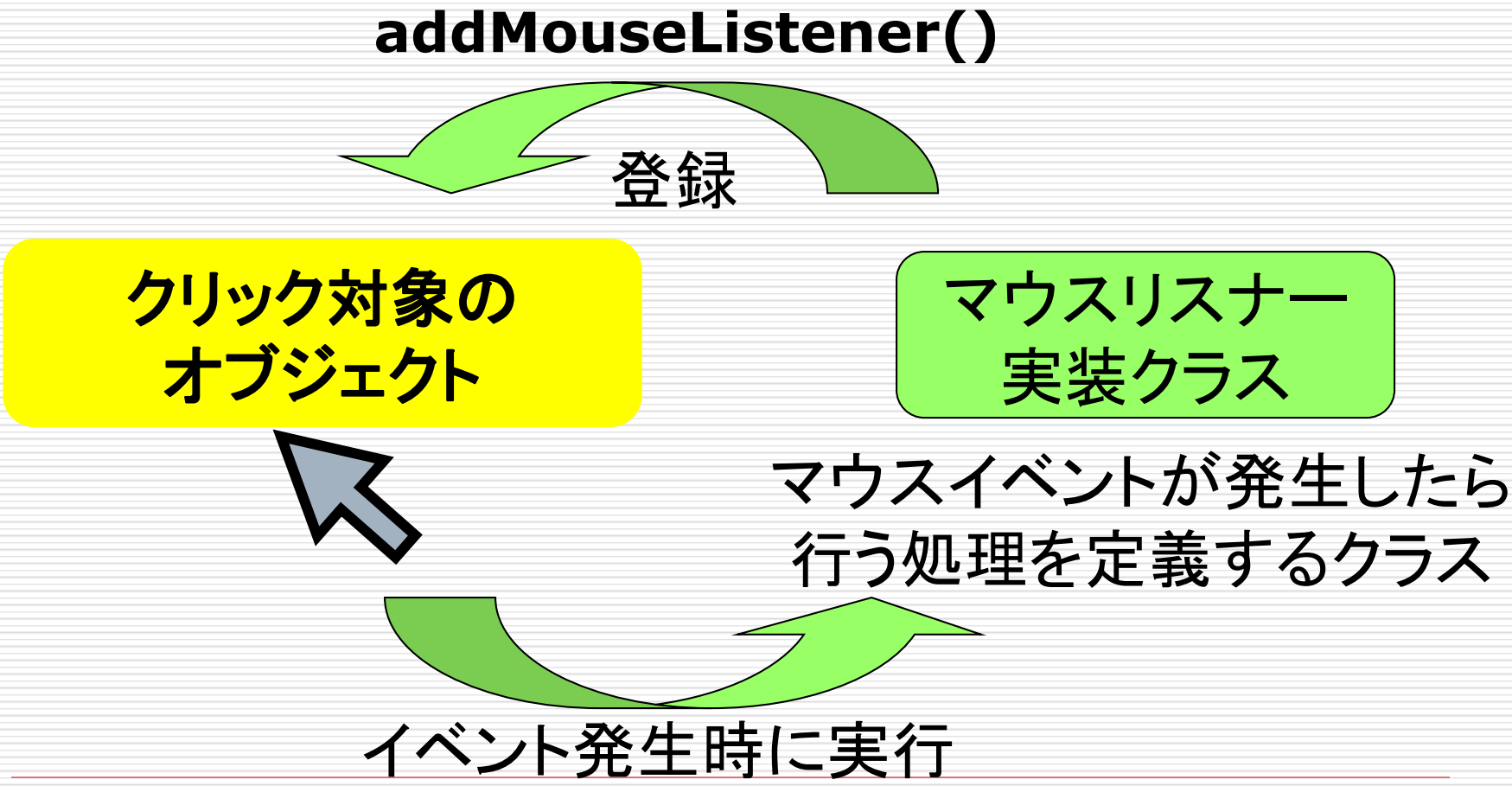


線の描き方

- 描画する処理を書くクラスは、Canvasクラスを継承
- 線を描画する処理をpaintメソッド内で実行
  - paintメソッド内でGraphicsオブジェクトを操作することにより描画

```
public void paint(Graphics g){
        g.drawLine(10,20,110,120);
}
```

マウスからのイベントの取得



※クリック対象オブジェクトとマウスリスナー実装クラスは同一でもよい

ソースプログラム

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;

class Painter{

        public static void main(String[] args){

                Frame frame=new Frame("お絵かきソフト");
                frame.setSize(400,300);
                DrawTool dt=new DrawTool();
                frame.add(dt);
                frame.setVisible(true);
        }
   }
```

//つづく

```
class DrawTool extends Canvas implements MouseListener{
        ArrayList<Point> points =new ArrayList<Point>();
        DrawTool(){
                addMouseListener(this);  //リスナーを登録
        }

        public void paint(Graphics g){
                g.setColor(Color.BLACK);
                if(points.size()>1){
                   for(int i=1; i< points.size(); i++){
                        Point a = points.get(i-1);
                        Point b = points.get(i);
                        g.drawLine(a.x, a.y, b.x, b.y);
                   }
                }
        }
        public void mouseClicked(MouseEvent e){
                Point p = new Point(e.getX(),e.getY());
                points.add(p);
                repaint(); //再描画
        }
```

//つづく

```
        public void mousePressed(MouseEvent e){
        }
        public void mouseReleased(MouseEvent e){
        }
        public void mouseEntered(MouseEvent e){
        }
        public void mouseExited(MouseEvent e){
        }
}
class Point{
        public int x;
        public int y;
        Point(int x, int y){
                this.x=x;
                this.y=y;
        }
}
```