情報処理技法
(Javaプログラミング)2

第11回
操作に対して処理が行われるGUI(1)

人間科学科コミュニケーション専攻

白銀 純子

第11回の内容

- GUIに処理を付加する～ボタンを押すと…?～
- GUIの部品で入力/選択された値を受け取るには?

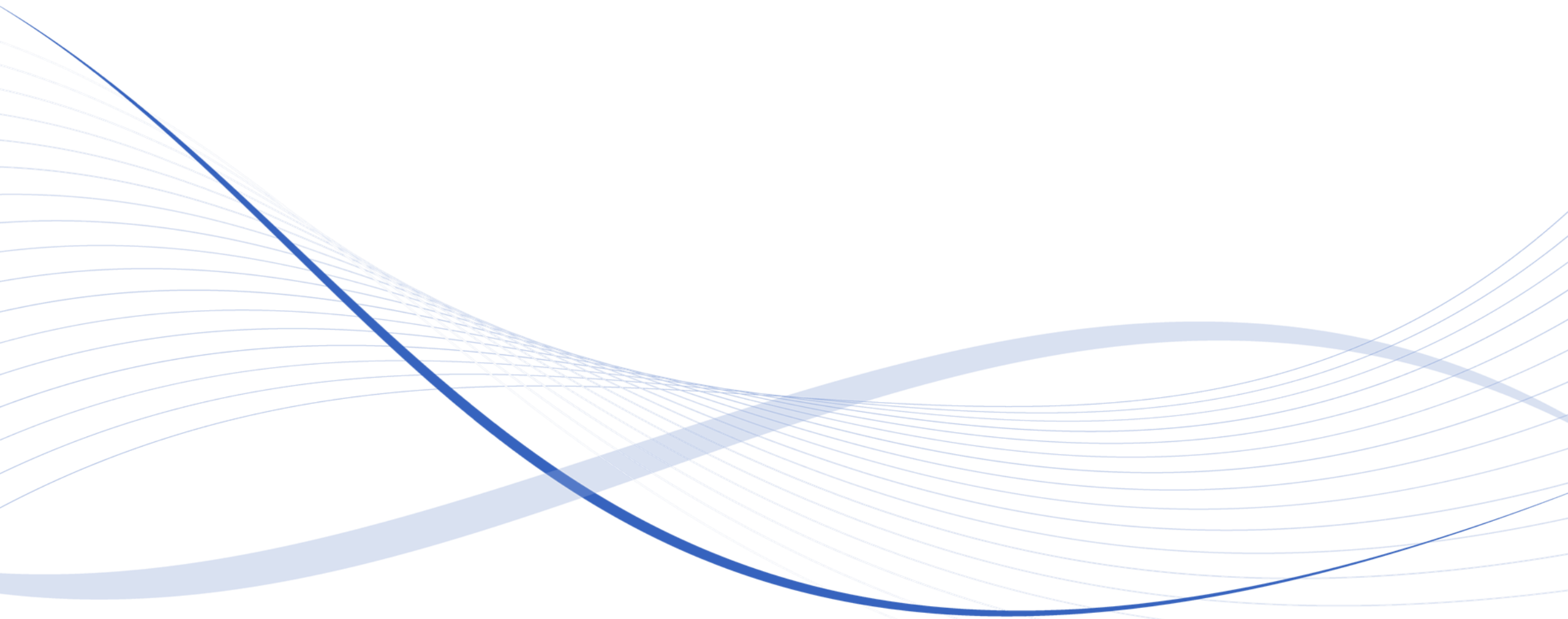
前回の復習問題の解答

- 「GUI」とは何か説明しなさい。

解答例:

人間がソフトウェアとのやりとりの接点を視覚的に表現したもので、例えばボタンや入力フィールドでソフトウェアを操作したり、ソフトウェアからの処理結果をウィンドウの形で表示したりできる。

GUIに処理を付加～ボタンを押すと…?～

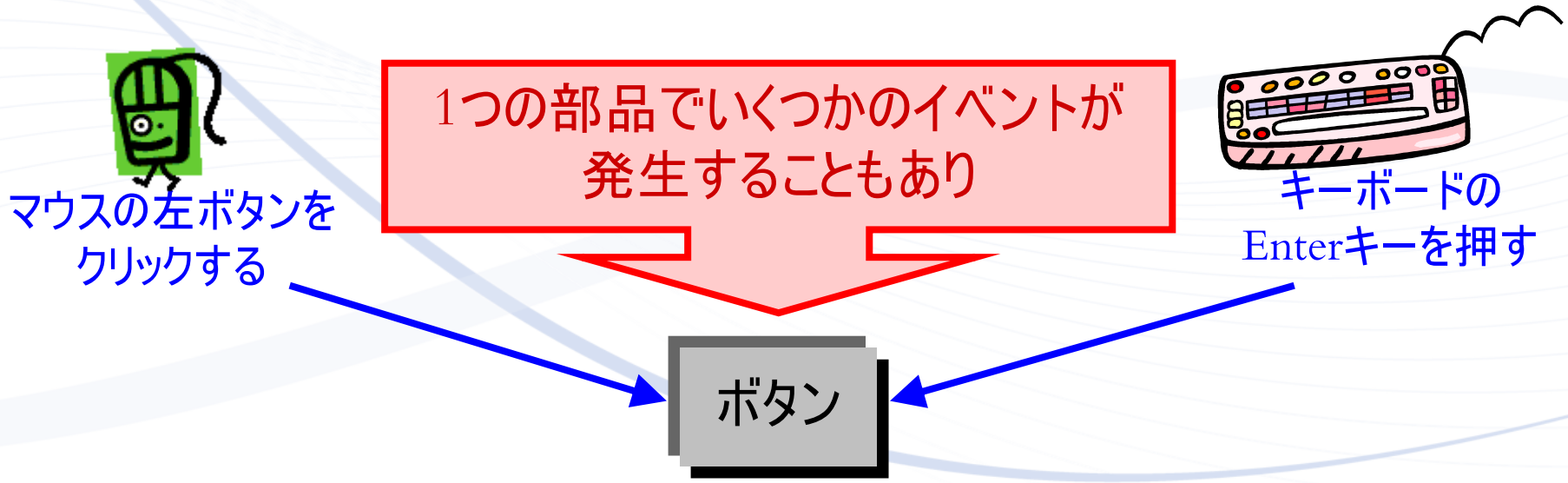


GUIプログラムが動くしくみ

1. 利用者がGUIの部品を操作する
   - 「ボタンを押す」、「キーボードのキーを押す」、「マウスを動かす」などの操作内容のことを「(ユーザ)イベント ((user) event)」と呼ぶ
   - 操作が行われることを、「イベントが発生する」と呼ぶ
2. プログラムが、GUIの部品が操作されたことを知る
   - GUIの部品が操作されたことをプログラムが知るための機能を「リスナ (Listener)」と呼ぶ
3. 操作内容に応じて、決められた処理をする

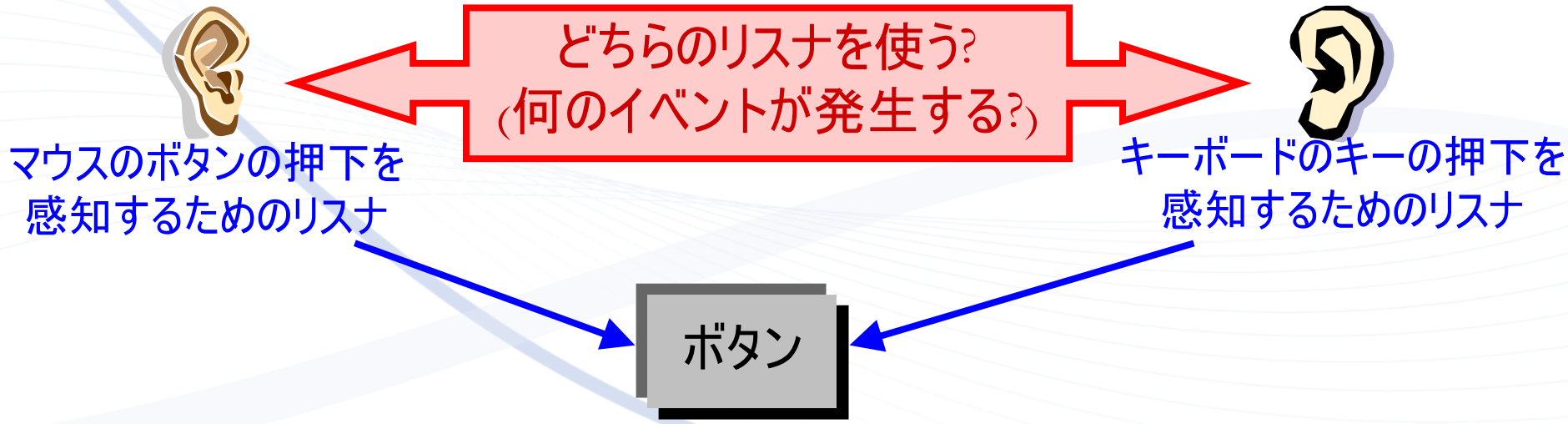
GUIに処理を付加するには?(1)

1. GUIの部品のうち、どの部品でイベントが起こったときに処理を行うかを決定
2. その部品でどのようなイベントが起こるのかを決定
   - ボタンを「押す」、入力フィールドで「Enterキーを押す」、などのイベントの種類を決定する
   - 1つの部品で発生するイベントは1つとは限らない



GUIに処理を付加するには?(2)

3. プログラムで、利用するイベントを宣言
4. 目的の部品に、イベントに対応するリスナを登録
   - リスナには多くの種類
   - 部品にどのリスナを登録するかで、どのイベントを受け取ることができるかが決定



イベントに対応するリスナをGUIの部品に登録する

GUIに処理を付加するには?(3)

5. イベントが発生したときの処理内容のプログラムを記述
   - 処理内容は、メソッドの中に書く
   - 処理内容を書くメソッド(メソッドの名前や引数)は、リスナによって決められている

GUIプログラムのカタチ(処理つき)(1)

```
import java.awt.event.*;
import javax.swing.*;

public class クラス名 extends JFrame implements リスナ名 {
    GUI部品の変数宣言
    public クラス名() { /* コンストラクタ */
        . . . . . .
        イベントが発生する部品の変数名.addリスナの名前(this);
        . . . . . .
    }
    public void リスナのメソッド名(イベント名 e) { /* リスナで決められたメソッド */

        イベントが発生したときの処理内容を書く領域

    }
    public static void main(String[] args) {
        new クラス名();
    }
}
```

GUIプログラムのカタチ(処理つき)(2)

```
import java.awt.event.*;
import javax.swing.*;

public class クラス名 extends JF
    GUI部品の変数宣言
    public クラス名() { /* コンスト
        . . . . . .
        イベントが発生する部品の変数名.addリスナの名前(this);
        . . . . . .
    }
    public void リスナのメソッド名(イベント名 e) { /* リスナで決められたメソッド */

        イベントが発生したときの処理内容を書く領域

    }
    public static void main(String[] args) {
        new クラス名();
    }
}
```

- リスナやイベントは、Javaでのクラスの一種
- このJavaファイルで、リスナやイベントを利用する、というパッケージ宣言

GUIプログラムのカタチ(処理つき)(3)

```
import java.awt.event.*;
import javax.swing.*;

public class クラス名 extends JFrame implements リスナ名 {
    GUI部品の変数宣言
    public クラス名() { /* コンストラクタ */
        ...
        ...リスナの名前(this);
        ...
    public void リスナのメソッド名(イベント名 e) { /* リスナで決められたメソッド */
        イベントが発生したときの処理内容を書く領域
    }
    public static void main(String[] args) {
        new クラス名();
    }
}
```

- 発生するイベントに対応するリスナを宣言する
- 「implements」で、一種の継承を意味する
  (リスナは、GUIのクラスに継承させることが多い)

※「implements」は、厳密には継承とは違う(「実装」と呼ぶ)

GUIプログラムのカタチ(処理つき)(4)

```
import java.awt.event.*;
import javax.swing.*;

public class クラス名 extends JFrame implements リスナ名 {
    GUI部品の変数宣言
    public クラス名() { /* コンストラクタ */
        ...
        イベントが発生したときの処理内容を書く領域
        ...
    }
    public
    }
    public static void main(String[] args) {
        new クラス名();
    }
}
```

GUI部品の変数は、フィールドとして宣言

- 特に、JTextFieldやJRadioButtonなど、利用者から入力をされる部品はフィールドとして宣言する必要
- イベントが発生したときの処理で、利用者からの入力を処理するときに、変数宣言の位置が重要

GUIプログラムのカタチ(処理つき)(5)

```
import java.awt.event.*;
import javax.swing.*;

public class クラス名 extends JFrame implemen
    GUI部品の変数宣言
    public クラス名() { /* コンストラクタ */
        . . . . . .
        イベントが発生する部品の変数名.addリスナの名前(this);
        . . . . . .
    }
    public void リスナのメソッド名(イベント名 e) { /* リスナで決められたメソッド */

        イベントが発生したときの処理内容を書く領域

    }
    public static void main(String[] args) {
        new クラス名();
    }
}
```

イベントが発生する部品に、リスナを登録

GUIプログラムのカタチ(処理つき)(6)

```
import java.awt.event.*;
import javax.swing.*;

public class
    GUI部品(
    public
        . . . . .
        イベント
        . . . . . .
    }
    public void リスナのメソッド名(イベント名 e) { /* リスナで決められたメソッド */
        イベントが発生したときの処理内容を書く領域
    }
    public static void main(String[] args) {
        new クラス名();
    }
}
```

- イベントが発生したときの処理
- 「implements」で継承(実装)したリスナのメソッドを
  オーバーライドして、処理を記述
  - ✓ メソッド名や引数・戻り値は、リスナで決められている

変数宣言の場所の注意

```
import java.awt.event.*;
import javax.swing.*;

public cl
    GUI部
    public
        . . .
        イベ
        . . .
    }
    public void リスナのメソッド名(イベント名 e) { /* リスナで決められたメソッド */

        イベントが発生したときの処理内容を書く領域

    }
    public st
        .
    }
}
```

- イベントが発生したときの処理では、GUI部品の変数を利用する処理も多い
  - ✓ JTextFieldに入力された文字列を受け取る，など
- コンストラクタ内で変数を宣言すると、リスナのメソッドとはブロックが違う
  = 「変数の宣言がされていない」とコンパイルエラー

リスナのメソッド内で利用するGUI部品の変数は、
必ずフィールド変数として宣言すること

ActionListener

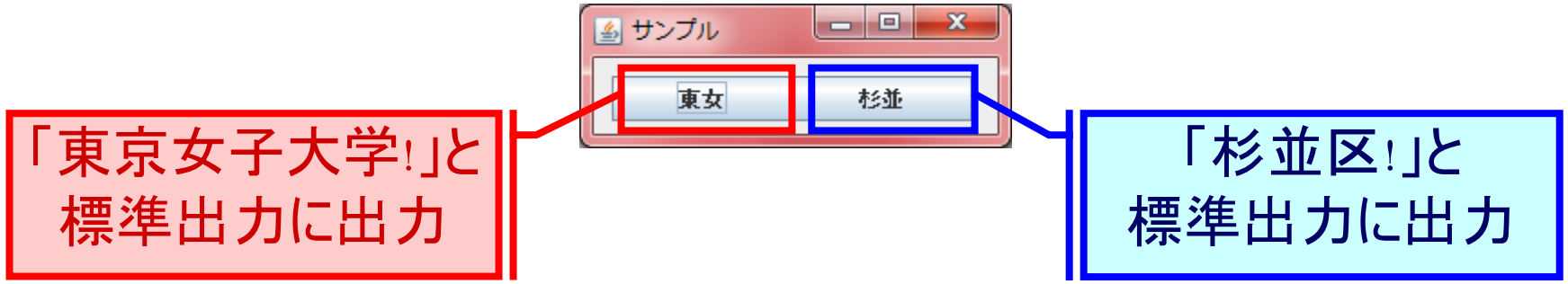
- もっともオーソドックスなリスナ
- ボタンをマウスの左ボタンで押すとき、メニューから選択するときのリスナ
- オーバーライドするメソッドは
  「actionPerformed(ActionEvent *引数名*)」
  - 戻り値は「void」
  - 引数は「ActionEvent」
    - 主に、「ボタンを押す」という意味のイベント

イベントが起こる部品が複数のとき?

- 「*メソッドの引数名*.getSource()」というメソッドで、どの部品でイベントが発生したかを知ることができる
- このメソッドを使ってイベントが発生した部品を受け取り、if文で処理内容を分岐させる

```
JButton  okBut, cancelBut;
. . . . . .
public  void  actionPerformed(ActionEvent  e) {
    if (okBut == e.getSource()) { /*「okBut」が押された場合 */
        okButが押されたときの処理を書く
    }  else  if (cancelBut == e.getSource()) {
        /*「cancelBut」が押された場合 */
        cancelButが押されたときの処理を書く
    }
}
```

例




```
public class Sample extends JFrame implements ActionListener {
  JButton twcu, suginami;

  public Sample() {    /* コンストラクタ */
    getContentPane().setLayout(null);

    twcu = new JButton("東女");
    twcu.setBounds(10, 10, 100, 25);
    twcu.addActionListener(this);
    getContentPane().add(twcu);
```

例(続き1)

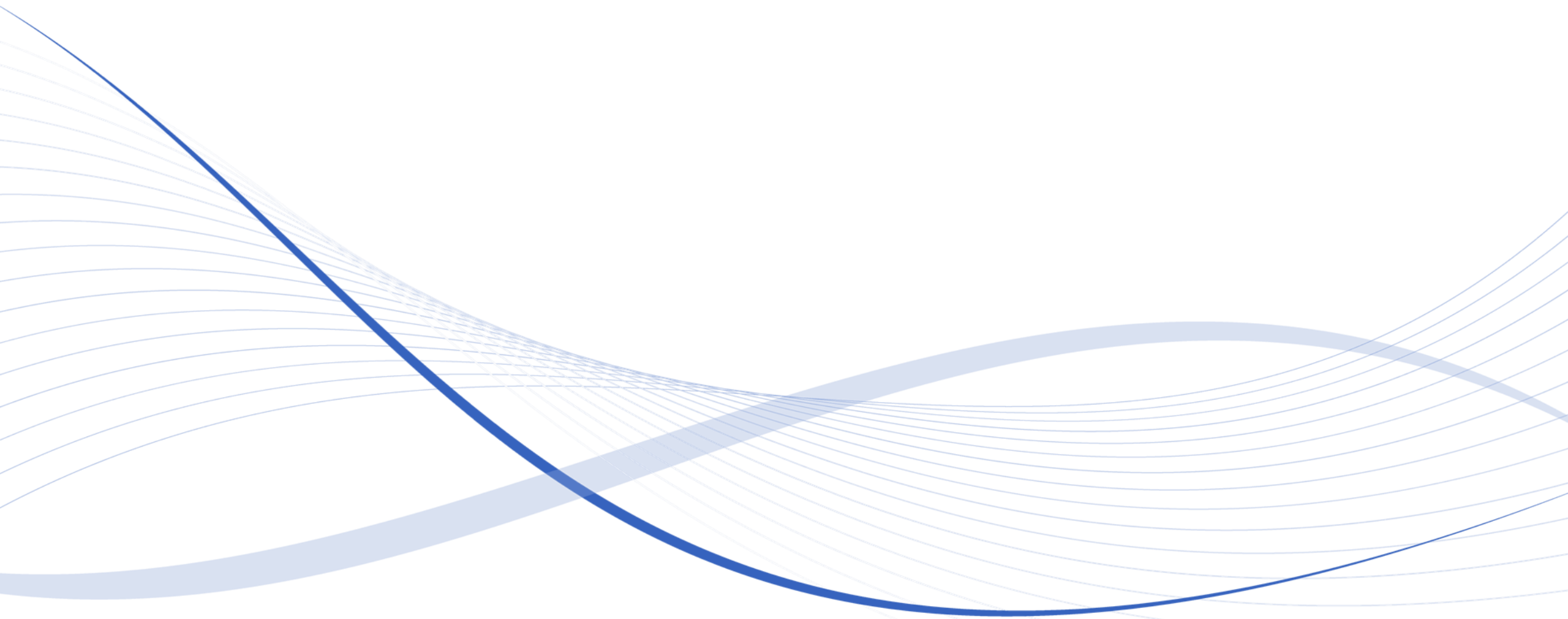
```
    suginami = new JButton("杉並");
    suginami.setBounds(110, 10, 100, 25);
    suginami.addActionListener(this);
    getContentPane().add(suginami);

    setTitle("サンプル");
    setSize(220, 70);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
  }  /* コンストラクタ終わり */
```

例(続き2)


```
public void actionPerformed(ActionEvent e) {
  /* ボタンが押されたときの処理内容 */
  if (e.getSource() == twcu) {  /*「東女」が押されたときの処理 */
    System.out.println("東京女子大学!");
  } else {  /*「杉並」が押されたときの処理 */
    System.out.println("杉並区!");
  }
 }
 public static void main(String[] args) {
  new Sample();
 }
}
```

入力/選択された値を受け取るには?



値の受け取り方

- GUIの部品には、それぞれ値を受け取るためのメソッドが用意されている

  → 「*GUIの部品の変数名(オブジェクト名).メソッド名*」で受け取ることができる

- リスナのメソッドの中で、値を受け取り、その値を処理するプログラムを書く

例:

```
JTextField  field;
. . . . . .
public  void  actionPerformed(ActionEvent  e) {
    . . . . . .
    String  text;
    text  = field.getText();
}
```

ボタン系(1)

- JRadioButton, JCheckBox, JToggleButtonで利用可能
- isSelected()
  - 戻り値はboolean型
  - 「true」であれば、選択されている状態
  - 「false」であれば、選択されていない状態

if文を使って、xxxボタンが選択されている(true)であれば…をし、
xxxボタンが選択されていなければ(falseであれば)～をする、などのように処理をする

ボタン系(2)

ボタン系の「isSelected()」メソッドの使い方例

例:

```
JRadioButton  maleRadio, femaleRadio;
. . . . . .
public  void  actionPerformed(ActionEvent  e) {
    . . . . . .
    String  gender;
    if (maleRadio.isSelected() == true) {
      //「男性」のJRadioButtonが選択されている場合
      gender = "男性";
    } else if (femaleRadio.isSelected() == true) {
      //「女性」のJRadioButtonが選択されている場合
      gender = "女性";
    } else {
      // 性別のJRadioButtonが選択されていない場合
      gender = "未選択";
    }
}
```

入力フィールド

- 主にJTextField, JTextAreaで利用可能
- getText()
  - 戻り値はString型

JComboBox

- getSelectedItem()
  - 戻り値は、「Object」という型
  - String型でキャストする

例(JComboBoxの変数名は「comboBox」):

```
String  item;
item  =  (String)  comboBox.getSelectedItem();
```

JSlider

- getValue()
  - 戻り値はint型

JList

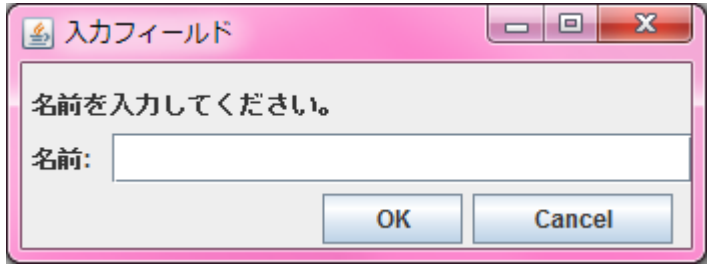
- getSelectedValue()
  - 戻り値は、「Object」という型
  - String型でキャストする

例(JListの変数名は「list」):

```
String  item;
item  =  (String)  list.getSelectedValue();
```

やってみよう!

- 下のウィンドウで、「OK」ボタンを押すと、「名前」の入力欄に入力した文字列を標準出力に出力するプログラム



- 下のウィンドウで、「OK」ボタンを押すと、JComboBoxで選択された文字列を標準出力に出力するプログラム