情報処理技法
(Javaプログラミング)2

第10回
操作に対して処理が行われるGUI(2)

人間科学科コミュニケーション専攻

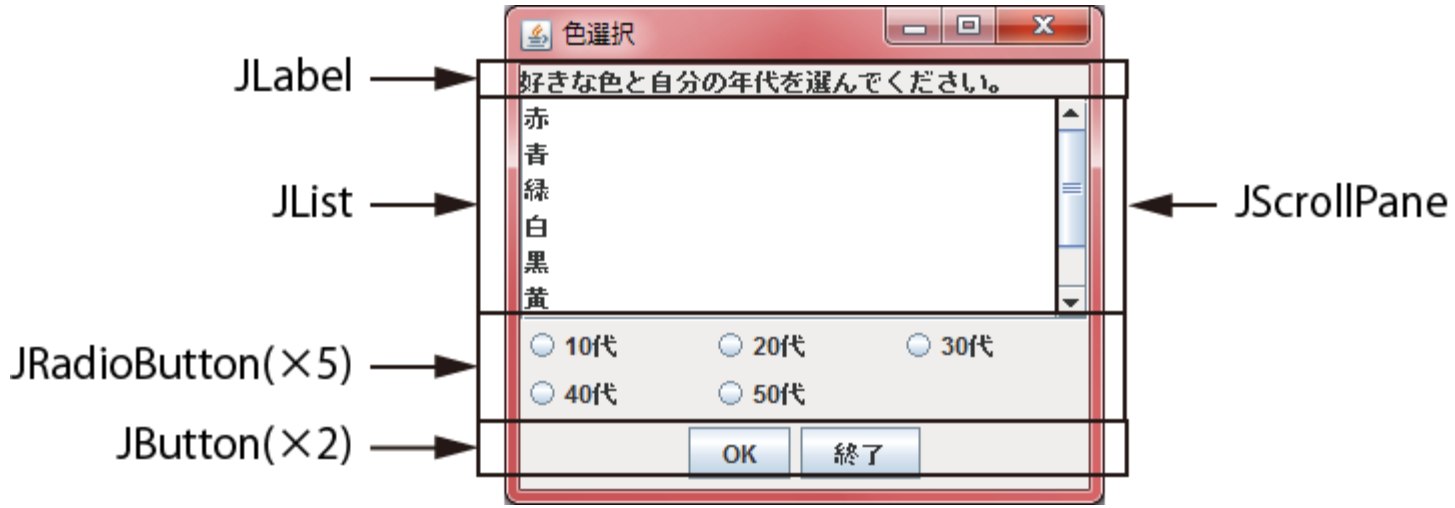
白銀 純子

第10回の内容

- ボタンを押したときのウィンドウ操作
- 入力された情報を次のウィンドウに送るには?
- GUIでのファイル操作

前回の復習問題の解答(1)

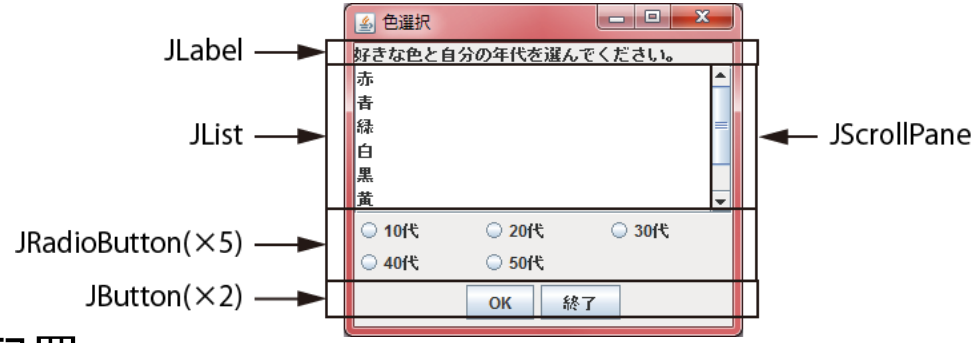
- 下図のウィンドウについて、レイアウトマネージャだけで部品を配置するとき、下記の点について考えて答えなさい。ただし、レイアウトマネージャは、授業で説明したBorderLayout・GridLayout・FlowLayoutのどれかとする。
  - JFrameにどのレイアウトマネージャを設定するか
  - どの部品をJPanelでグループ化し、そのJPanelにはどのレイアウトマネージャを設定するか
    ※部品をBorderLayoutのJFrameまたはJPanelに配置するときには、その部品を東・西・南・北・中央のどの位置に配置するかも答えること



前回の復習問題の解答(2)

解答例1:

- JPanelを3つ用意(panel1, panel2, panel3とする)
- JFrameにBorderLayoutを設定
  - ✓ JLabelをJFrameの北に配置
  - ✓ panel1をJFrameの中央に配置
  - ✓ panel3をJFrameの南に配置



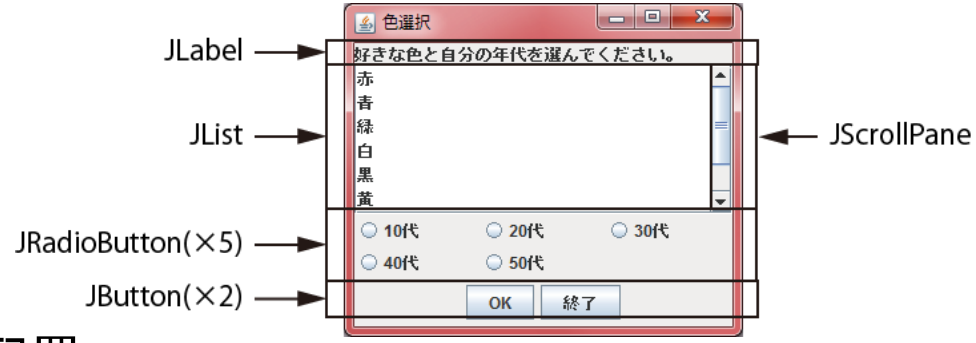
- panel1にBorderLayoutを設定
  - JScrollPaneをpanel1の中央に配置
  - panel2をpanel1の南に配置

- JListをJScrollPaneに貼り付け
- panel2にGridLayout(3, 2)またはFlowLayoutを設定
  - ✓ 5つのJRadioButtonをpanel2に配置
- panel3にFlowLayoutを設定
  - ✓ 2つのJButtonをpanel3に配置

前回の復習問題の解答(3)

解答例2:

- JPanelを3つ用意(panel1, panel2, panel3とする)
- JFrameにBorderLayoutを設定
  - ✓ panel1をJFrameの中央に配置
  - ✓ panel3をJFrameの南に配置



- panel1にBorderLayoutを設定
  - ✓ JLabelをpanel1の北に配置
  - ✓ JScrollPaneをpanel1の中央に配置
  - ✓ panel2をpanel1の南に配置

- JListをJScrollPaneに貼り付け
- panel2にGridLayout(3, 2)またはFlowLayoutを設定
  - ✓ 5つのJRadioButtonをpanel2に配置
- panel3にFlowLayoutを設定
  - ✓ 2つのJButtonをpanel3に配置

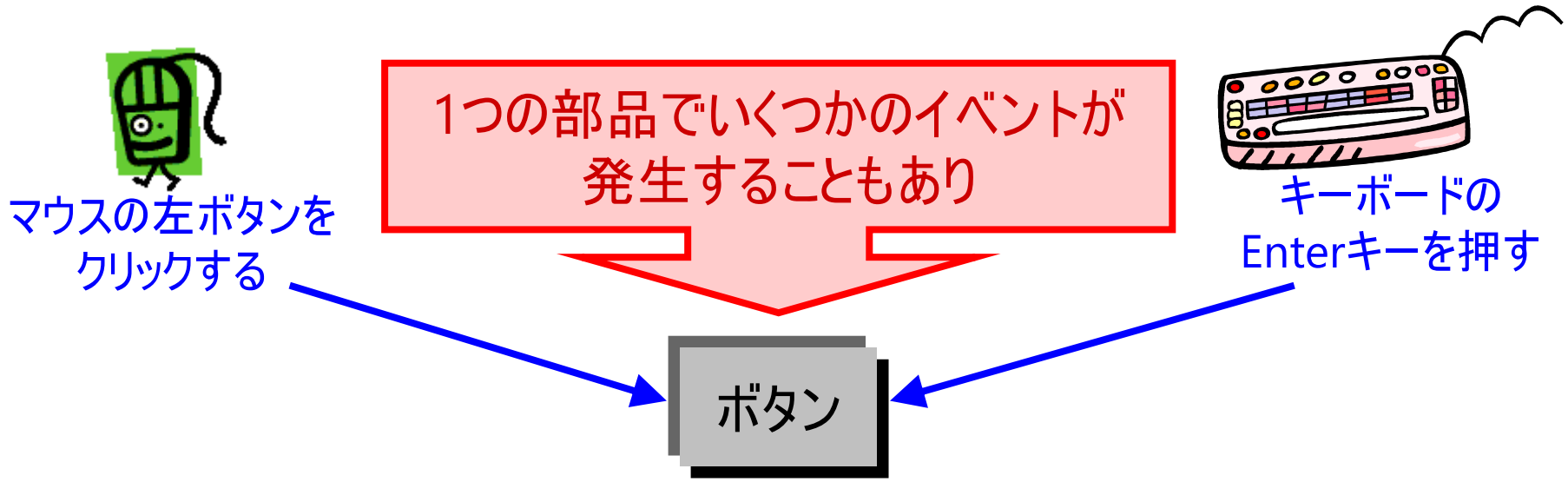
前回の復習

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

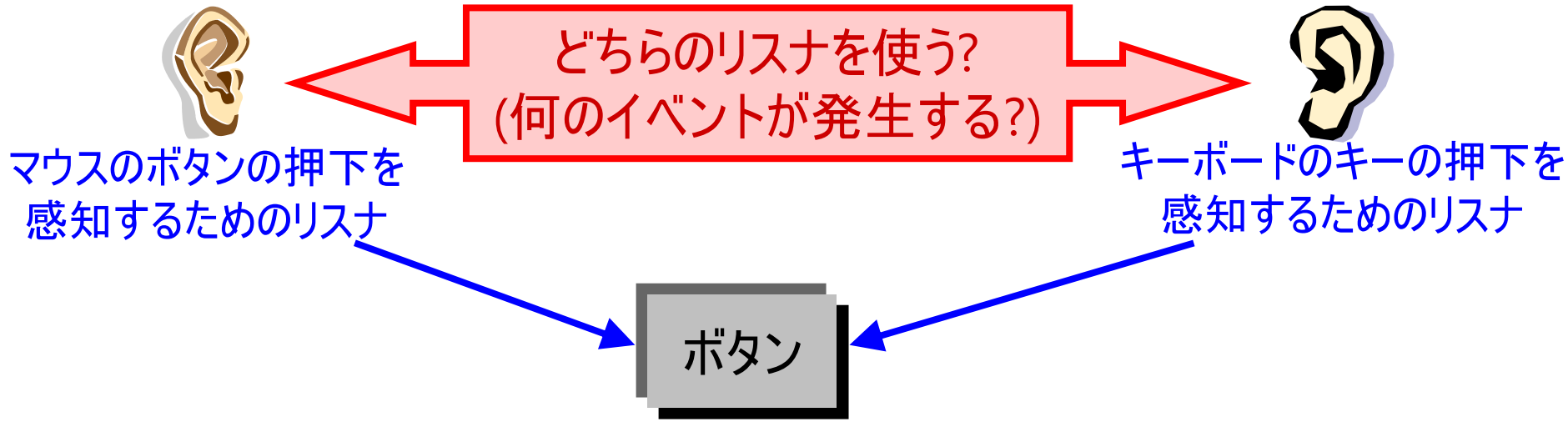
GUIに処理を付加するには?(1)

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
    GUI部品の変数宣言
    public  クラス名() { /* コンス
        . . . . . .
        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 */
                                                    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  impl
    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  clas
    GUI部品(
    public  ク
        ....
        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 c
    GUI部
    publ
        ...
        イ
        ...
    }
    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
        ...
    }
}
```

- 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

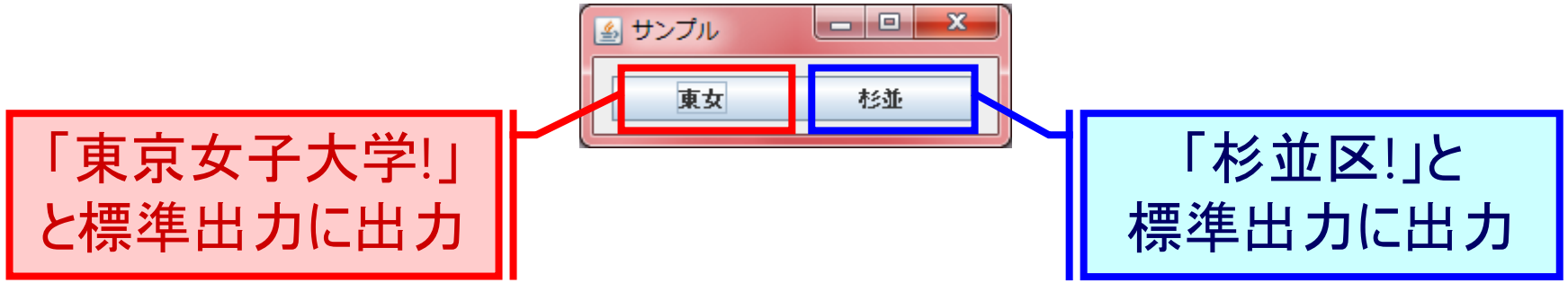
- もっともオーソドックスなリスナ
- ボタンをマウスの左ボタンで押すとき、メニューから選択するときのリスナ
- オーバーライドするメソッドは
  「actionPerformed(ActionEvent　*引数名*)」
  - 戻り値は「void」
  - 引数は「ActionEvent」
    - 主に、「ボタンを押す」という意味のイベント

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();
```

ボタンを押したときのウィンドウ操作

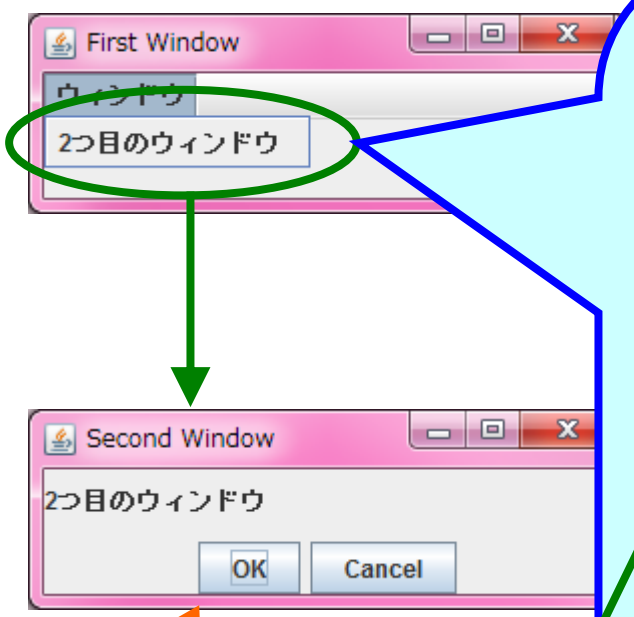
ボタンを押したときのウィンドウ操作

- ウィンドウに関して、行われる主な処理としては…
  - 別のウィンドウを表示
  - 現在表示しているウィンドウを消去

別のウィンドウを表示(1)

1. 2つ目のウィンドウを表示するプログラムを書く
   - 原則として、1つのクラスに1つのウィンドウのプログラム
   - 2つ目以降のクラスには「public static void main(String[] args)」は不要
   - 2つ目以降のクラスには「setDefaultClosingOperation」も不要
     - 2つ目のウィンドウを閉じたときにソフトが終了すると困るから
2. 1つ目のウィンドウのリスナのメソッドの中に2つ目のウィンドウを表示する命令を書く
   - 「new *2つ目のウィンドウのクラス名*();」でウィンドウを表示

別のウィンドウを表示(2)




```
public  FirstWin() {
  .....
  secondWin = new JMenuItem();
  secondWin.addActionListener(this);
  .....
}
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == secondWin) {
    new SecondWin();
  }
}
public static void main(String[] args) {
  new FirstWin();
}
```

```
public SecondWin() {
   .....
   setSize(300, 100);
   setTitle("Second Window");
   setVisible(true);
}
```

「public static void main(String[] args)」と
「setDecaultClosingOperation(...)」は不要

別のウィンドウを表示(3)

- プログラムのコンパイル:
  javac 1つ目のファイル.java 2つ目のファイル ...
  で、関係するファイルをコンパイル可能
- プログラムの実行:
  java *mainを持つクラスのクラス名*
  で実行
  - 「mainを持つクラスのクラス名」が、1つ目のウィンドウ

3つ目以降のウィンドウ

- 1つ目のウィンドウから2つ目のウィンドウを表示する場合と同様
- n番目のウィンドウからn+1番目のウィンドウを表示する場合
  - n+1番目のウィンドウを表示するプログラムを書く
  - n番目のウィンドウ内のリスナのメソッド内にn+1番目のウィンドウを表示する命令を書く
    - 「new *n+1番目のウィンドウのクラス名*();」でウィンドウを表示

ウィンドウを消す処理

- 「setVisible(false)」でウィンドウが消える
  - setVisible: JFrameクラスで定義されているメソッド
    - 「extends JFrame」でJFrameクラスを継承しているので、「オブジェクトの変数名.setVisible」と
      いう形でなく利用可能

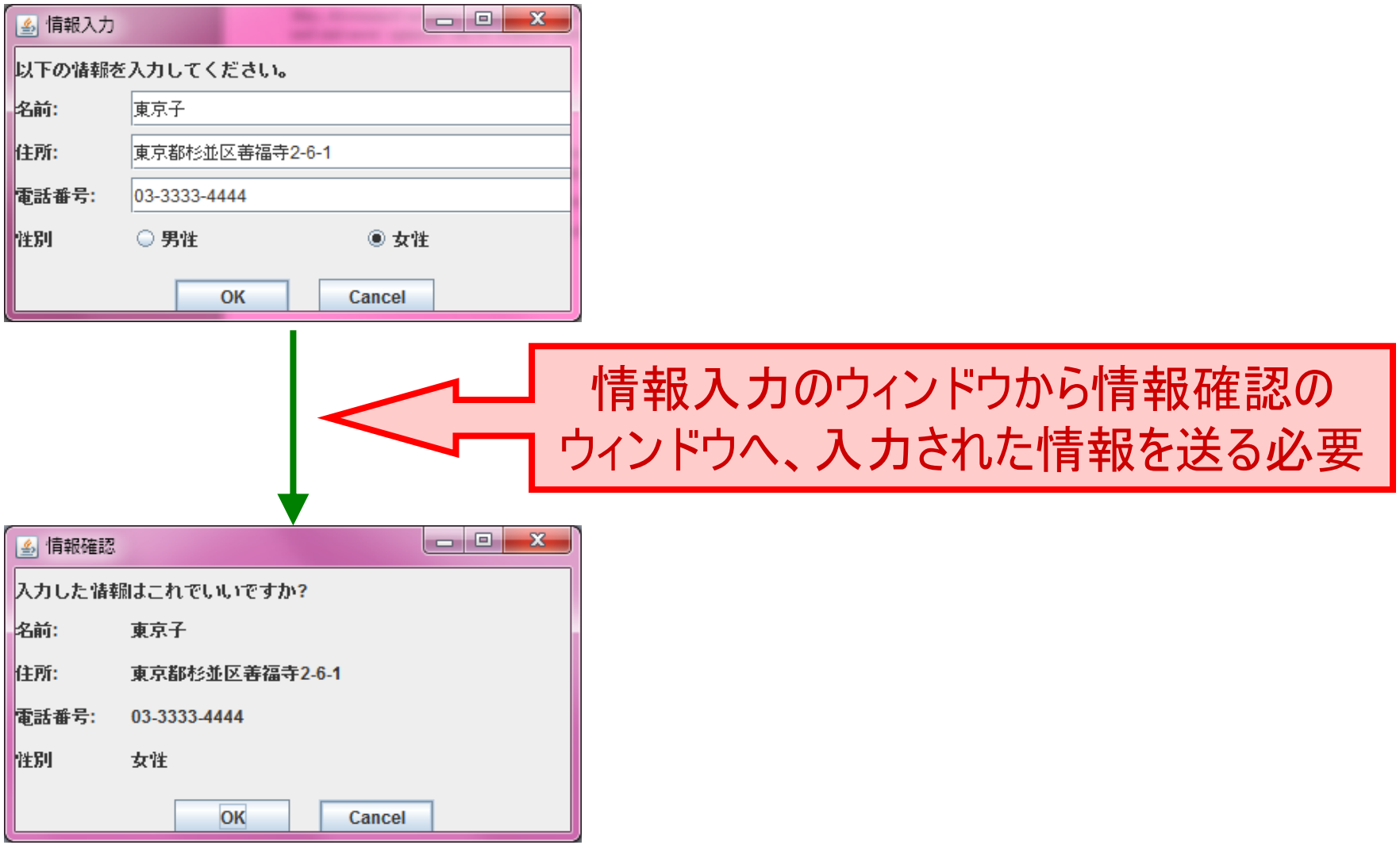
```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == okBut) {
    「OK」ボタンが押されたときの処理
    setVisible(false);
  } else if (e.getSource() == cancelBut) {
    setVisible(false);
  }
}
```

ウィンドウを消す処理

入力された情報を次に送るには?

入力された情報を次に送る

- 例えば…入力した情報の確認をするとき



プログラムの書き方(1)

- 情報を受け取るウィンドウのコンストラクタに引数をつける

AddressConfirm.java

```
public AddressConfirm(String name, String address, String tel, String gender) {
  .....
  nameLabel = new JLabel();
  nameLabel.setText("名前: ");
  .....
  nameField = new JLabel();
  nameField.setText(name);
  .....
  addressLabel = new JLabel();
  addressLabel.setText("住所: ");
  .....
  addressField = new JLabel();
  addressField.setText(address);
  .....
}
```

入力された情報を受け取る引数
- name: 名前の受け取り
- address: 住所の受け取り
- tel: 電話番号の受け取り
- gender: 性別の受け取り

コンストラクタの引数(受け取った情報をラベルに表示)

プログラムの書き方(2)

- 情報を入力するウィンドウから、確認するウィンドウを表示するときに、引数付きでウィンドウを作る

AddressInput.java

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == okBut) {
    String name, address, tel, gender;
    name = nameField.getText();
    address = addressField.getText();
    tel = telField.getText();
    if (maleRadio.isSelected() == true) {
      gender = "男性";
    } else {
      gender = "女性";
    }
    new AddressConfirm(name, address, tel, gender);
  }
}
```

入力フィールドから入力された情報を受け取り、変数に代入

JRadioButtonは、どれが押されていれば変数に何を代入するかを記述

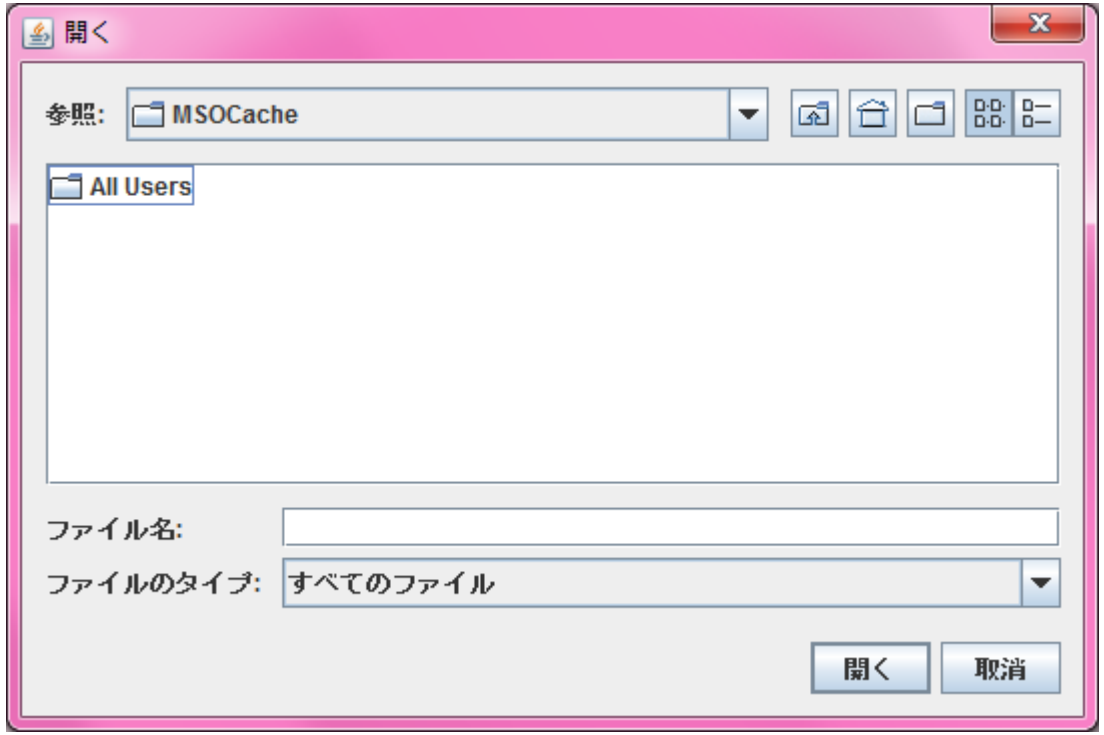
引数を使って確認用ウィンドウに情報を受け渡し

GUIでのファイル操作

GUIでのファイル操作

- ファイル操作専用のGUI部品 – JFileChooser
  - 読み込むファイルを決める
  - 情報を書き出す(保存する)ファイルを決める

JFileChooser



JFileChooser

- これだけでファイル選択のウィンドウを表示する部品
  - JFrameに貼りつける必要はなし
- ウィンドウ表示の方法
  - ファイル読み込み:「showOpenDialog(null)」というメソッドを利用
  - ファイル書き出し:「showSaveDialog(null)」というメソッドを利用
- ウィンドウ表示のメソッドの戻り値が0の場合が、「OK」を押されたとき
- 選択されたファイルの受け取り:「getSelectedFile()」メソッドで受け取り

JFileChooser～読み込み(1)～

- ファイルを読み込む場合

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == okBut) {
    JFileChooser chooser = new JFileChooser();
    int code = chooser.showOpenDialog(null);
    if (code == 0) {
      File readFile = chooser.getSelectedFile();
      try {
        FileReader fr = new FileReader(readFile);
        BufferedReader br = new BufferedReader(fr);
        ……
      }
      catch(IOException ioe) {
      }
    }
  }
}
```

JFileChooser～読み込み(2)～

- ファイルを読み込む場合

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == okBut) {
    JFileChooser chooser = new JFileChooser();
    int code = chooser.showOpenDialog(null);
    if (code == 0) {
      File read
      try {
        FileRea
        BufferedReader br = new BufferedReader(fr);
        .....
      }
      catch(IOException ioe) {
      }
    }
  }
}
```

JFileChooserのオブジェクトを作り、
「showOpenDialog(null)」メソッドでウィンドウを表示

JFileChooser～読み込み(3)～

- ファイルを読み込む場合

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == okBut) {
    JFileChooser chooser = new JFileChooser();
    int code = chooser.showOpenDialog(null);
    if (code == 0) {
      File readFile = chooser.getSelectedFile();
      try {
        FileRead
        Buffered
        .....
      }
      catch(IOException ioe) {
      }
    }
  }
}
```

「showOpenDialog(null)」メソッドの戻り値が「0」のときが、「開く」ボタンを押された場合

JFileChooser～読み込み(4)～

- ファイルを読み込む場合

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == okBut) {
    JFileChooser chooser = new JFileChooser();
    int code = chooser.showOpenDialog(null);
    if (code == 0) {
      File readFile = chooser.getSelectedFile();
      try {
        FileReade
        Buffered
        .....
      }
      catch(IOException ioe) {
      }
    }
  }
}
```

JFileChooserで選択されたファイルを受け取り、
Fileクラスの「readFile」変数に代入

「File」クラス: Javaでファイルを扱うために用意されているクラス

JFileChooser～読み込み(5)～

- ファイルを読み込む場合

```
public void
  if (e.getS
    JFileCho
    int code
    if (code
      File readFile = chooser.getSelectedFile();
      try {
        FileReader fr = new FileReader(readFile);
        BufferedReader br = new BufferedReader(fr);
        .....
      }
      catch(IOException ioe) {
      }
    }
  }
}
```

選択されたファイルの内容を読む処理(FileReaderクラスのコンストラクタの引数は、Fileクラスのオブジェクトまたは String型のファイル名)

JFileChooser～書き出し(1)～

- ファイルに書き込む場合

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == okBut) {
    JFileChooser chooser = new JFileChooser();
    int code = chooser.showSaveDialog(null);
    if (code == 0) {
      File writeFile = chooser.getSelectedFile();
      try {
        FileWriter fw = new FileWriter(writeFile);
        PrintWriter pw = new PrintWriter(fw);
        .....
      }
      catch(IOException ioe) {
      }
    }
  }
}
```

JFileChooser～書き出し(2)～

- ファイルに書き込む場合

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == okBut) {
    JFileChooser chooser = new JFileChooser();
    int code = chooser.showSaveDialog(null);
    if (code == 0) {
      File writ
      try {
        FileWrit
        PrintWriter pw = new PrintWriter(fw);
        .....
      }
      catch(IOException ioe) {
      }
    }
  }
}
```

JFileChooserのオブジェクトを作り、
「showSaveDialog(null)」メソッドでウィンドウを表示

JFileChooser～書き出し(3)～

- ファイルに書き込む場合

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == okBut) {
    JFileChooser chooser = new JFileChooser();
    int code = chooser.showSaveDialog(null);
    if (code == 0) {
      File writeFile = chooser.getSelectedFile();
      try {
        FileWrit
        PrintWr
        .....
      }
      catch(IOException ioe) {
      }
    }
  }
}
```

「showSaveDialog(null)」メソッドの戻り値が
「0」のときが「開く」ボタンを押された場合

JFileChooser～書き出し(4)～

- ファイルに書き込む場合

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == okBut) {
    JFileChooser chooser = new JFileChooser();
    int code = chooser.showSaveDialog(null);
    if (code == 0) {
      File writeFile = chooser.getSelectedFile();
      try {
        FileWrite
        PrintWrit
        .....
      }
      catch(IOException ioe) {
      }
    }
  }
}
```

JFileChooserで選択されたファイルを受け取り、
Fileクラスの「writeFile」変数に代入

「File」クラス: Javaでファイルを扱うために用意されているクラス

JFileChooser～書き出し(5)～

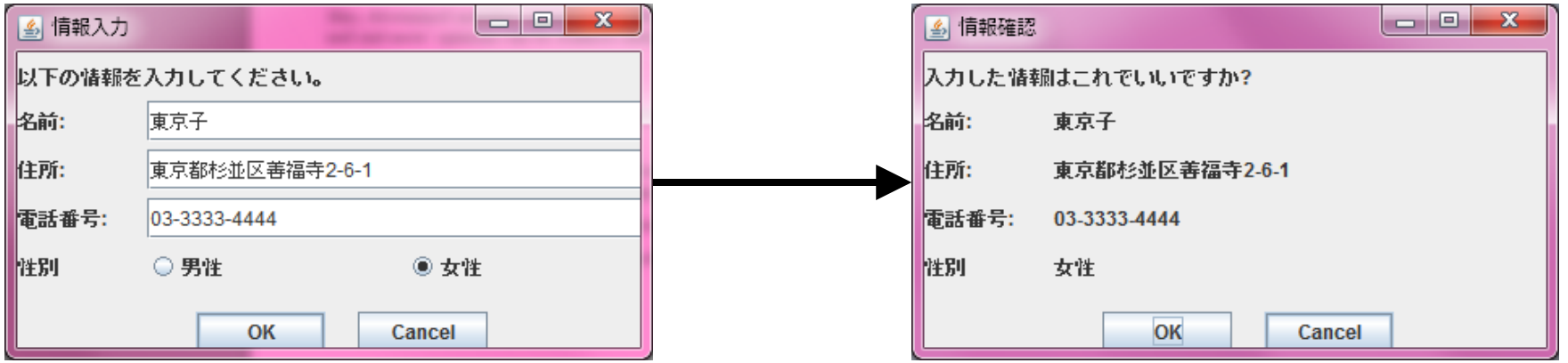
- ファイルに書き込む場合

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == okBut) {
    JFileCh
    int co
    if (code == 0) {
      File writeFile = chooser.getSelectedFile();
      try {
        FileWriter fw = new FileWriter(writeFile);
        PrintWriter pw = new PrintWriter(fw);
        .....
      }
      catch(IOException ioe) {
      }
    }
  }
}
```

選択されたファイルの内容を読む処理(FileReaderクラスのコンストラクタの引数は、Fileクラスのオブジェクトまたは String型のファイル名)

やってみよう!(1)

- 「情報入力」ウィンドウに入力した情報を「情報確認」ウィンドウに表示するプログラム



- 「情報入力」ウィンドウに情報を入力し、「OK」を押したら、ファイル選択ウィンドウからファイルを選択し、そのファイルに入力された情報を書き込むプログラム
  - 「東京子, 東京都杉並区..., 03-3333-4444, 女性」という形で書き込み

やってみよう!(2)

- 下記の処理をするプログラム
  1. ファイルを1つ選択
     - ファイルの内容は下図のようなもの
  2. 1. で選択したファイルの内容を読み込み、1行1行の内容をウィンドウに表示

ファイルの内容の例

```
k14x1001, 東京子, 東京都出身
k14y2030, 善福寺花子, 千葉県出身
k14z3050, 吉祥寺祥子, 埼玉県出身
```

- 学生番号と氏名、出身地を「,」で区切って1人1行で表したもの
- ファイルの行数は最大100行

※どのようなウィンドウの構成にすれば良いかも自分で考えること