情報処理技法
(Javaプログラミング)2

第7回
GUIプログラミングの基礎

人間科学科コミュニケーション専攻
白銀 純子

第7回の内容

- GUIプログラミングの基礎～ウィンドウを作ってみよう～

前回の出席課題の解答

- コンビニの商品管理プログラムを考えるとき、下記のクラスを作る場合に、どのような継承関係にすれば良いか、考えて答えなさい。
  - 商品, パン, アイス, ドリンク, コーヒー, ジュース, お茶

解答

- 親クラス: 商品, 子クラス: パン, アイス, ドリンク
- 親クラス: ドリンク, 子クラス: コーヒー, ジュース, お茶

GUIプログラミングの準備

前期の復習～パッケージ～(1)

- パッケージ: Javaに用意されている様々な機能の分類
  - プログラミングのために使うことのできる機能を、その内容ごとに分類したもの
  - 「java.io」, 「java.lang」はパッケージの名前
  - 各機能の名前は、「パッケージ名+名前」がフルネーム
    - 例えば入力のための「BufferedReader」という機能は、フルネームは「java.io.BufferedReader」
    - 「java.io.*」と書くと、「java.io」というパッケージに所属する全ての機能、という意味

前期の復習～パッケージ～(2)

- Javaファイルの中で書く機能については、その機能を使うことを明示しておくことが多い
  - 明示した場合: その機能の名前をパッケージ名を省略可能(「java.io.BufferedReader」であれば、「BufferedReader」とだけ書けばよい)
  - 明示しなかった場合: その機能の名前をフルネームで記述
- 明示するには:「import」というキーワードの後に続けて機能の名前を書く
  - 1つ1つの機能の名前を指定するか、「java.io.*」のように、1つのパッケージ内の機能を全て、と指定する
    - 「*」で「全て」という意味

パッケージの例

- 「java.io」: 標準入出力やファイル入出力などの入出力を扱うクラスが分類されている
- 「java.lang」: Stringなどの、Javaで扱う基本的なクラスが分類されている
- 「java.awt」: GUIやグラフィックを扱うための基本的なクラスが分類されている
- 「javax.swing」: 高機能なGUIを扱うためのクラスが分類されている

パッケージの使い方

- Javaでは、クラスのフルネームは、「パッケージ名.クラス名」
  - プログラム中では、本来は、フルネームでクラスを記述
  - 「import」文で、そのクラスが所属するパッケージを指定しておくと、「パッケージ名.」を省略可能

「import」文

- Javaファイル中で、「このクラスを使う」という宣言
  - 本来は、「String」クラスは、「java.lang.String」と書く
  - 「import」文で宣言しているために、「String」のみでよくなっている
- 例
  - 「import　java.io.BufferedReader;」
    - 「BufferedReader」クラスを利用するという宣言
    - こう書くと、「BurfferedReader」クラス以外の「java.io」のクラスはパッケージ名を省略できない
  - 「import　java.io.*;」
    - 「java.io」以下の全てのクラスを利用する、という宣言(「*」で、「全てのクラス」を意味)

Javaで用意されているクラス

- Javaには、目的ごとに様々なクラスが用意されている
  - ファイルを扱うためのクラス: File
  - 文字列を扱うためのクラス: String

- 各クラスには、クラスの目的を果たすために、様々なメソッドが用意されている
  - 標準入力・ファイル入力で、1行ずつ読み込むメソッド: BufferedReaderクラスの「readLine」
  - 文字列の部分文字列を作るためのメソッド: Stringクラスの「substring」

特殊なメソッド～コンストラクタ～

- クラスには、オブジェクトを作成すると同時に実行される特殊なメソッド(コンストラクタ)を定義できる
- 定義方法:「public クラス名(引数のリスト) {...}」
  - 引数のリストはなくてもよい
  - 「{」から「}」の間に、オブジェクトを作成すると同時に実行される処理内容を記述する
  - オブジェクトを作成するとき、「new クラス名();」とするのは、コンストラクタを呼び出している、ということ
- コンストラクタで処理するもの
  - フィールドの値を設定
  - GUIを扱うクラスの場合、GUIのウィンドウを作成する処理

コンストラクタの例

クラスの定義

```
public  class  Student {
  String  number, score, circle;
  public  Student(String  n,  String  s,  String  c) {
    number = n;
    score = s;
    circle = c;
  }
}
```

コンストラクタ

オブジェクト作成例

```
Student  azuma;
azuma = new Student("K14X1001", "A", "オーケストラ");
```

```
Student  azuma;
azuma = new Student();
azuma.number = K14X1001;
azuma.score = "A";
azuma.circle = "オーケストラ"
```

同じ意味

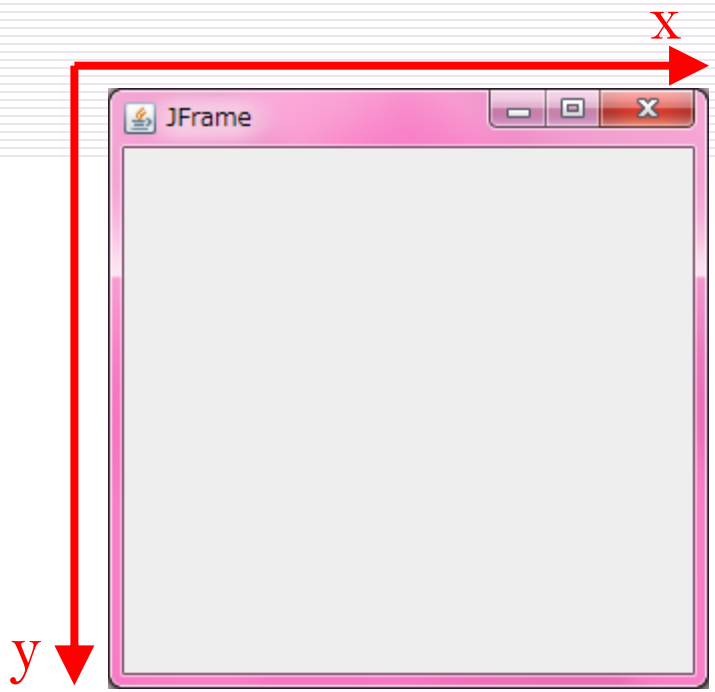
ウィンドウ作成

GUIとは

- GUI: Graphical User Interface
  - 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
    - ボタンや入力フィールドでソフトウェアを操作
    - ソフトウェアからの処理結果の提示
    - etc.
  - Graphical: 視覚的
  - User: 人間の利用者
  - Interface: ものとものとの接点(人間とソフトウェアとの間に限らず)

ウィンドウの作り方の基本

1. ウィンドウ内の構成(ボタンや入力フィールドなどの配置を決める)
2. 「JFrame」というウィンドウの土台を作る
3. 土台の上に、ボタンなどのGUIの部品を配置していく
   - 配置は、座標で指定する(今回は)
   - x座標は右方向
   - y座標は下方向(グラフの座標軸とは逆)



部品の作り方

- 部品は、一種のデータ型(クラス)
  - 部品: ボタンや入力フィールドなどの1つ1つのGUIの要素
  - 1つ1つの部品の変数を宣言する
  - 部品の作成 = 部品のオブジェクト作成
    - *部品名 変数名*=new *部品名*();
  - 作成した部品に、いろいろな情報を設定する(部品を置く位置や大きさ、部品の見た目上の名前など)
  - 作成した部品をJFrameに貼り付ける

※JFrameも部品の1つ

GUIプログラムの基本形(1)

```
import  java.io.*;
import  java.lang.*;
import  javax.swing.*;

public  class  クラス名  extends JFrame {
  部品の変数の宣言

  public  クラス名() {
    getContentPane().setLayout(null);
    setTitle(タイトルバーに表示する名前);
    setSize(横の長さ, 縦の長さ);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
  }
 public  static  void main(String[]  args) {
     new  クラス名();
  }
}
```

JFrame以外の部品を作成したり情報を設定する領域

GUIプログラムの基本形(2)

```
import  java.io.*;
import  java.lang.*;
import  javax.swing.*;

public  class  クラス名   extends JFrame {
  部品の変数の宣言

  public  クラス名() {
    getContentPane().setLayout(null);

    setTitle(タイトルバーに表示する名前);
    setSize(横の長さ, 縦の長さ);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
  }
 public  static  void main(String[]  args) {
     new  クラス名();
  }
}
```

GUIプログラミングで使うクラスが入っているパッケージの宣言

JFrame以外の部品を作成したり情報を設定する領域

GUIプログラムの基本形(3)

```
import  java.io.*;
import  java.lang.*;
import  javax.swing.*;

public  class  クラス名  extends JFrame {
  部品の変数の宣言

  public  クラス名() {
    getContentPane().setLayout(null);
    setTitle(タイトルバーに表示する名前);
    setSize(横の長さ, 縦の長さ);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
  }
 public  static  void main(String[]  args) {
     new  クラス名();
  }
}
```

ウィンドウの土台は、JFrameというクラスを継承して作ることが一般的

JFrame以外の部品を作成したり情報を設定する領域

GUIプログラムの基本形(4)

```
import  java.io.*;
import  java.la
import  javax.sw

public  class  クラス名  extends JFrame {
```

様々なGUIの処理をしていくために、部品の変数は
クラスのフィールド(どのメソッドにも含まれない)として宣言

部品の変数の宣言

```
  public  クラス名() {
    getContentPane().setLayout(null);
```

JFrame以外の部品を作成したり情報を設定する領域

```
    setTitle(タイトルバーに表示する名前);
    setSize(横の長さ, 縦の長さ);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
  }
 public  static  void main(String[]  args) {
     new  クラス名();
  }
}
```

GUIプログラムの基本形(5)

```
import  java.io.*;
import  java.lang.*;
import  javax.swing.*;

public  class  クラス名   extends JFrame {
  部品の変数の宣言

  public  クラス名() {
    getContentPane().setLayout(null);
    setTitle(タイトルバーに表示する名前);
    setSize(横の長さ, 縦の長さ);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
  }
  public  static  void main(String[]  args) {
      new  クラス名();
  }
}
```

ウィンドウの表示の処理は、
コンストラクタ内に記述

JFrame以外の部品を作成したり情報を設定する領域

GUIプログラムの基本形(6)

```
import  java.io.*;
import  java.lang.*;
import  javax.swing.*;

public  class  クラス名   extend
  部品の変数の宣言

  public  クラス名() {
   getContentPane().setLayout(null);
   setTitle(タイトルバーに表示する名前);
   setSize(横の長さ, 縦の長さ);
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   setVisible(true);
  }
 public  static  void main(String[]  args) {
     new  クラス名();
  }
}
```

JFrameクラスから継承されたメソッド
(ウィンドウ自体に対する様々な設定をしている)

JFrame以外の部品を作成したり情報を設定する領域

GUIプログラムの基本形(7)

```
import  java.io.*;
import  java.lang.*;
import  javax.swing.*;

public  class  クラス名  extends JFrame {
  部品の変数の宣言

  public  クラス名() {
    getContentPane().setLayout(null);
    setTitle(タイトルバーに表示する名前);
    setSize(
    setDefa
    setVis b
  }
 public  static  void main(String[]  args) {
    new  クラス名();
  }
}
```

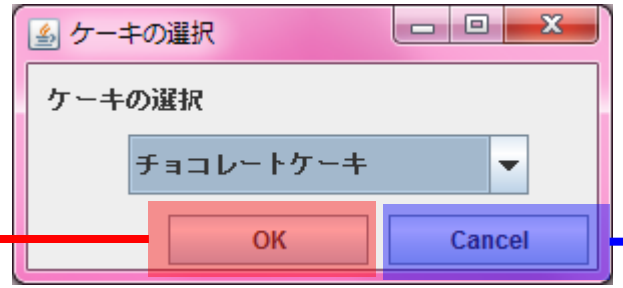
JFrame以外の部品を作成したり情報を設定する領域

コンストラクタ内でウィンドウの表示の処理を書いているので、
mainメソッドではオブジェクトを作成するだけでOK

部品の扱い方

- 部品に対しては、様々な操作をすることができる
  - 見た目の名前をつける
  - 貼り付ける座標や大きさを決める
  - etc.
- 部品に対して様々な操作を行うために、メソッドが用意されている
- ウィンドウ中に1つ部品を作るにあたり、その部品のオブジェクトを1つ作成する
  - 部品はJavaでクラスとしてあらかじめ提供されている
- 「*部品の変数名.メソッド名(引数)*」で部品に対する設定を行う

1部品につき1オブジェクト




```
JButton  okBut;
....
okBut = new JButton();
okBut.setText("OK");
...
```

```
JButton  cancelBut;
....
cancelBut = new JButton();
cancelBut.setText("Cancel");
...
```

1つの部品につき、1つ変数を宣言をしてオブジェクトを作成、設定が必要

部品に用意されているメソッド(基本)

JFrame(1)

- getContentPane().setLayout(*部品配置の規則*)
  - JFrameに、部品の配置の規則を設定するメソッド
  - 引数の値として、今回は「null」(何もなし)を設定しておく
    - 後日、様々な配置の規則を説明
- getContentPane().add(*JFrameに貼り付ける部品の変数名*)
  - JFrameに部品を貼り付ける(配置する)ためのメソッド
- setTitle(*タイトルバーに表示する言葉*)
  - ウィンドウのタイトルバーの言葉を設定するメソッド
  - 引数はString型

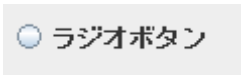
JFrame(2)

- setSize(*JFrameの横の長さ, JFrameの縦の長さ*)
  - ウィンドウの大きさを設定するメソッド
  - 引数はどちらもint型
- setDefaultCloseOperation(*閉じたときの動作*)
  - 「×」ボタンを押してウィンドウを閉じたときの動作を設定するメソッド
  - 引数として、「JFrame.EXIT_ON_CLOSE」(プログラムを終了する)を設定しておく
- setVisible(*ウィンドウを表示するかしないか*)
  - ウィンドウをディスプレイに表示するかしないかを決めるメソッド
  - 引数として、「true」(表示する)を設定しておく

ボタン系とラベル(部品の種類)

- ボタン系・ラベルのクラス名
  - JButton: ボタン

  - JRadioButton: 複数の中から1つだけを選択するボタン



  - JCheckBox: 複数の中のいくつかを選択するボタン

  - JToggleButton: 複数の中から1つだけまたはいくつかを選択するボタン (選択されると引っ込んだ状態になる)

  - JLabel: 1行の文字列を表示するための部品

ボタン系とラベル(利用できるメソッド)

- setText(*見た目の名前*)
  - ディスプレイに表示されるボタンやラベルの名前
  - 引数はString型
- setBounds(*x座標, y座標, 横の長さ, 縦の長さ*)
  - 部品ののJFrame上での位置と大きさを設定するメソッド
  - 引数はすべてint型

ボタンやラベルに絵を貼り付ける方法

- 絵のファイルを用意する
  - Javaファイルと同じフォルダの中に絵のファイルを置く
- 「setIcon(new ImageIcon("絵のファイル名"))」で絵を貼り付ける
  - 「setText」メソッドの代わりに「setIcon」メソッドを使う

```
JLabel label = new JLabel();
label.setIcon(new ImageIcon("絵のファイル名"));
```

JRadioButtonのグループ化

- JRadioButtonは、そのままでは、択一選択にはならない

→ JRadioButtonのグループ化(どのJRadioButtonの間で択一選択をするかの設定)

→ 「ButtonGroup」というクラスを利用

ButtonGroupで、グループの「枠」を作成し、JRadioButtonをその「枠」に登録することでグループ化をする

ButtonGroup

- 1つのグループにButtonGroupのオブジェクト(変数)を1つ
- JRadioButtonをグループ化する方法:
  ButtonGroupの変数名.add(*JRadioButtonの変数名*)

例

```
JRadioButton one, two, three;
ButtonGroup number;
public RadioButtonSample() extends JFrame{
    number = new ButtonGroup();
    one = new JRadioButton();
    one.setText("1");
    number.add(one);
    one.setBounds(5, 5, 30, 25);
    . . . . . .
}
```

入力フィールド

- 入力フィールドのクラス名
  - JTextField: 1行のみの文字列の入力のための部品

  - JTextArea: 複数行の文字列の入力のための部品

- setBounds(*x座標, y座標, 横の長さ, 縦の長さ*)
  - 部品ののJFrame上での位置と大きさを設定するメソッド
  - 引数はすべてint型

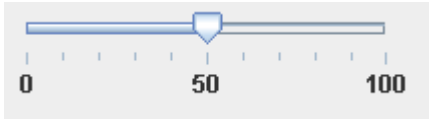
JComboBox

- 複数項目の中から1つを選択するためのリスト
  - 県名の選択などによく使われている

- addItem(*リストの項目名*)
  - リストに項目を追加するメソッド
  - 引数はString型
  - 必要な項目の分だけこのメソッドを書く
- setBounds(*x座標, y座標, 横の長さ, 縦の長さ*)
  - 部品ののJFrame上での位置と大きさを設定するメソッド
  - 引数はすべてint型

JSlider(1)

- 数値を選択するためのバー



- setMaximum(*最大値*)
  - 目盛りの最大値を設定するメソッド
  - 引数はint型
- setMinimum(*最小値*)
  - 目盛りの最小値を設定するメソッド
  - 引数はint型

JSlider(2)

- setMajorTickSpacing(*大目盛りの間隔*)
  - 大きく表示される目盛りの間隔を設定するメソッド
  - 引数はint型
  - setPaintTicks, setPaintLabelsをtrueにしないと、目盛りは表示されない
- setMinorTickSpacing(*小目盛りの間隔*)
  - 小さく表示される目盛りの間隔を設定するメソッド
  - 引数はint型
  - setPaintTicksをtrueにしないと、目盛りは表示されない

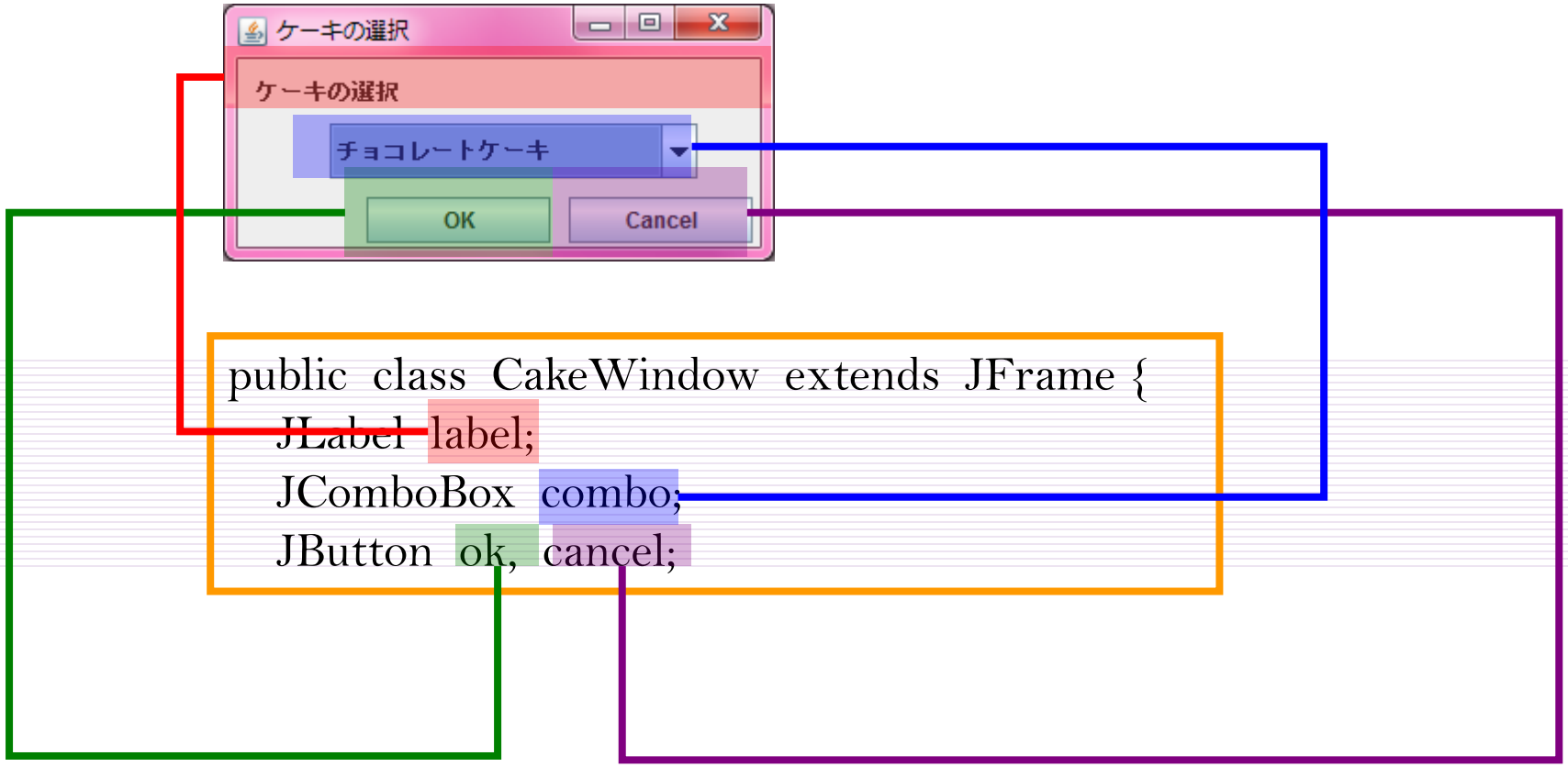
JSlider(3)

- setPaintTicks(*目盛りを表示するどうか*)
  - 目盛りを表示するかどうかを設定するメソッド
  - 表示する場合は、引数として「true」を設定しておく
- setPaintLabels(*大目盛りの数値を表示するかどうか*)
  - 大目盛りの数値を表示するかどうかを設定するメソッド
  - 表示する場合は、引数として「true」を設定しておく
- setBounds(*x座標, y座標, 横の長さ, 縦の長さ*)
  - 部品ののJFrame上での位置と大きさを設定するメソッド
  - 引数はすべてint型

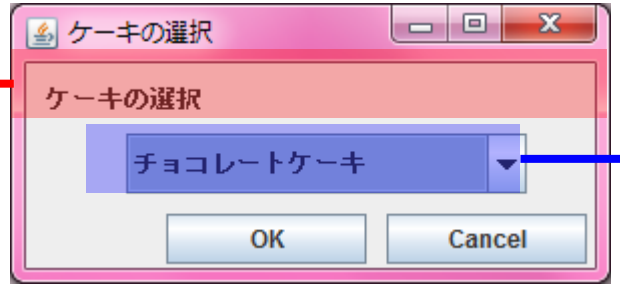
JPanel

- 他の部品を配置するための土台
  - 複数の部品を1つのグループとして扱う場合
  - 画像を表示したり絵を描いたりするキャンバスとしても利用可能
- setBounds(*x座標, y座標, 横の長さ, 縦の長さ*)
  - 部品ののJFrame上での位置と大きさを設定するメソッド
  - 引数はすべてint型

例(変数宣言)



例(部品作成)




```
public  CakeWindow() {
  getContentPane().setLayout(null);

  label = new JLabel();  /* JLabelの作成 */
  label.setText("ケーキの選択");
  label.setBounds(10, 5, 280, 25);
  getContentPane().add(label);

  combo = new JComboBox(); /* JComboBoxの作成 */
  combo.addItem("チョコレートケーキ");
  combo.addItem("チーズケーキ");
  combo.setBounds(50, 35, 200, 30);
  getContentPane().add(combo);
```

例(部品作成)(続き)


```
    ok = new JButton(); /* JButtonの作成 */
    ok.setText("OK");
    ok.setBounds(70, 75, 100, 25);
    getContentPane().add(ok);
    cancel = new JButton(); /* JButtonの作成 */
    cancel.setText("Cancel");
    cancel.setBounds(180, 75, 100, 25);
    getContentPane().add(cancel);

    setTitle("ケーキの選択");
    setSize(300, 140);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
}
```

部品作成・ウィンドウ表示処理の順序

```
public class CakeWindow extends JFrame {
   JLabel label;
   JComboBox combo;
   JButton ok, cancel;

   public CakeWindow() {
      getContentPane().setLayout(null);

      label = new JLabel();  /* JLabelの作成 */
      label.setText("ケーキの選択");
      label.setBounds(10, 5, 280, 25);
      getContentPane().add(label);

      .... 略 ....

      setTitle("ケーキの選択");
      setSize(300, 140);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setVisible(true);
   }
```

個々の部品作成とウィンドウへの貼り付け

ウィンドウの表示処理

- ウィンドウ上への部品の配置は、コンピュータの処理では、部品の絵を描くようなもの
- ウィンドウを表示したときに、描画処理

↓

ウィンドウを表示後に部品を配置しても、描画処理が行われない

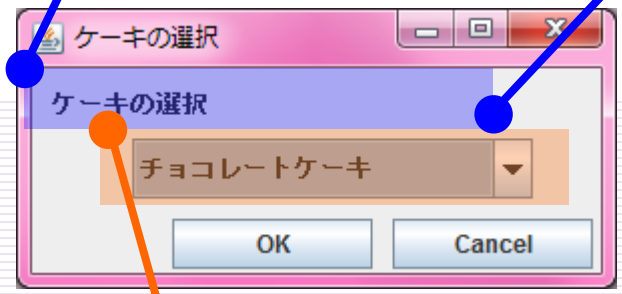
↓

配置した部品が表示されない

「setVisible(true);」の処理は一番最後に!

座標計算(1)

このJLabelの左上隅の座標は
x座標: 10, y座標; 5

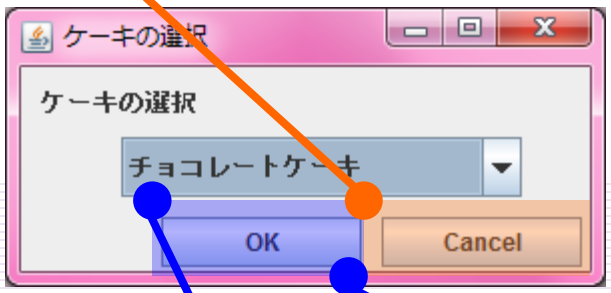


幅: 280, 高さ: 25なので、このJLabelの
右下端の座標はx座標: 290, y座標: 30

上のJLabelが、y座標が30まであるので、このJComboBoxの
左上端のy座標は30以上にしないと、上のJLabelと重なる

座標計算(2)

左のJButtonが、x座標が170まであるので、このJButtonの
左上端のx座標は170以上にしないと、左のJButtonと重なる



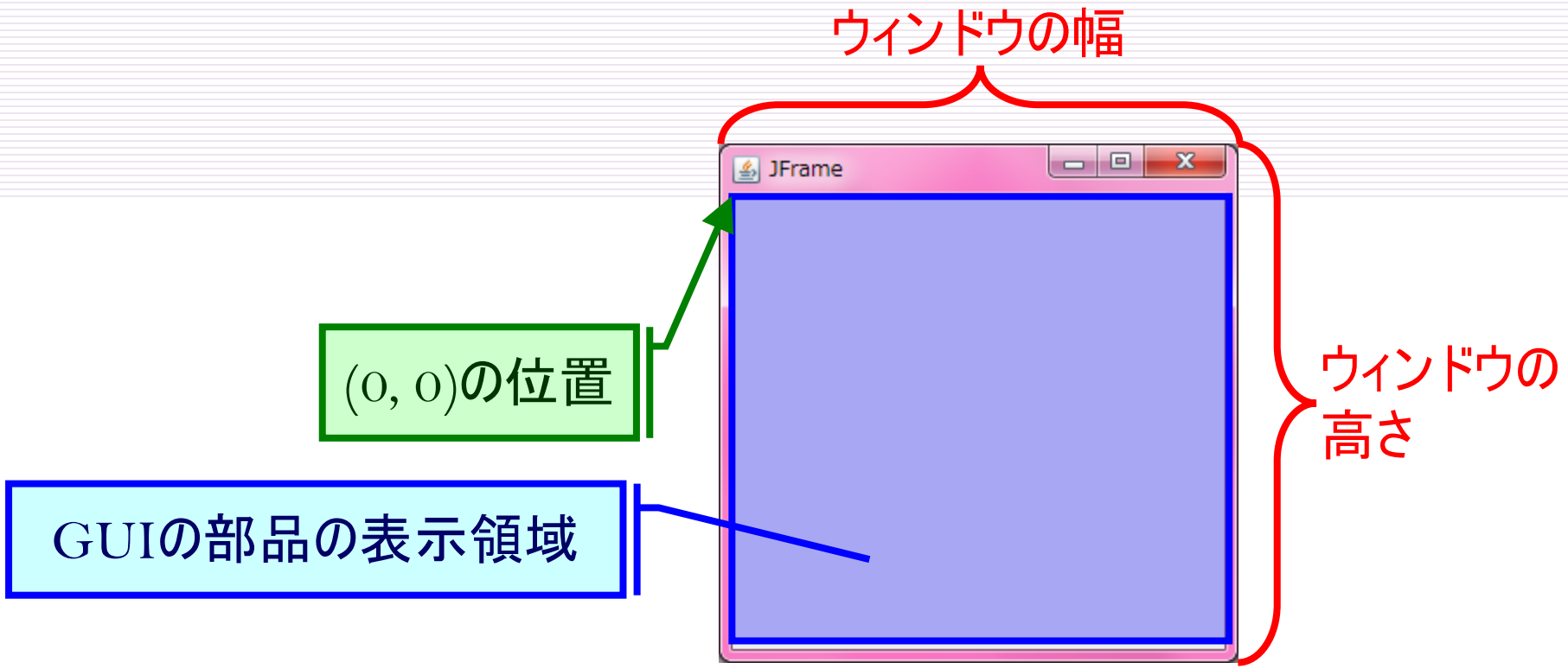
幅: 100, 高さ25なので、このJButtonの
右下端の座標はx座標: 170, y座標: 100

このJButtonの左上隅の座標は
x座標: 70, y座標; 75

部品の座標は、上下左右に隣接する部品の四隅の座標を計算した上で
決定すること(でないと、隣接する部品に重なってしまうので注意)

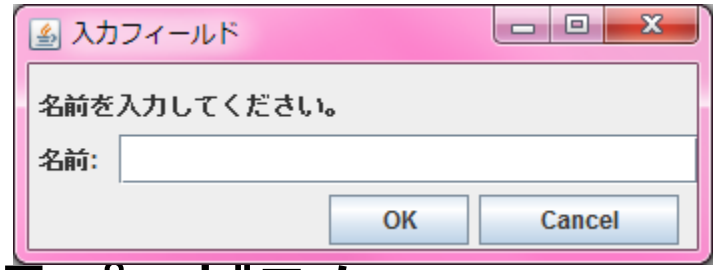
ウィンドウの幅と高さの考え方

- GUIの部品の配置は一領域は、ウィンドウの枠の中
- setSizeメソッドで指定するウィンドウの幅と高さは、ウィンドウの枠の含めた大きさにする必要
  - GUIの部品の表示に必要な幅と高さ+αの大きさが必要

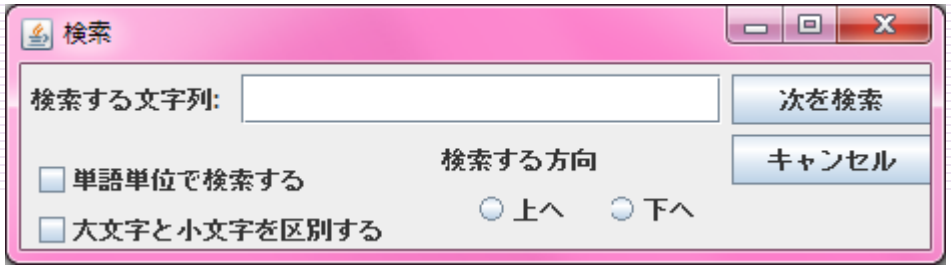


やってみよう!

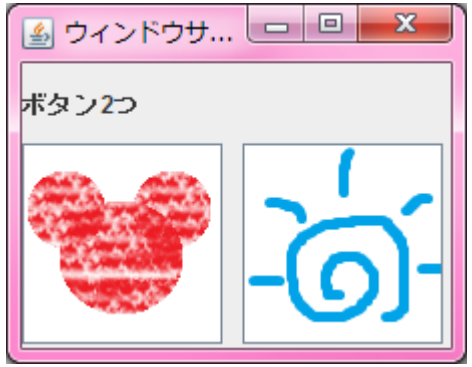
- 下記のウィンドウを作るプログラム



- 下記のウィンドウを作るプログラム(JRadioButtonは択一選択に)



- 下記のウィンドウを作るプログラム
  - 表示する絵は何でも良い



※3つの問題とも、部品の座標や大きさは適当で良い