情報実験I
プログラムの仕様とテスト

第4回
5月17日

本日の予定

- テストの実施
- リファクタリングチェック

- 最終課題の説明
- Applet、AWT入門のつづき

テストの実施

- 2人で1組になる。
- 前回と同じく、自分のテストケースで、他方のプログラムをテストする。
- 報告
  - 自分のテストケースの数
  - テスト実施者の学籍番号
  - テスト実施者の氏名
  - テストしてもらったすべてのテストケース数
  - 上記の内、自分のプログラムが合格した数
  - テストしたプログラム（相手のプログラム）は要求を満たしていると思うか？その理由を記せ。
  - テスト項目を定義することでプログラム作成に影響はあったか？

リファクタリングチェック

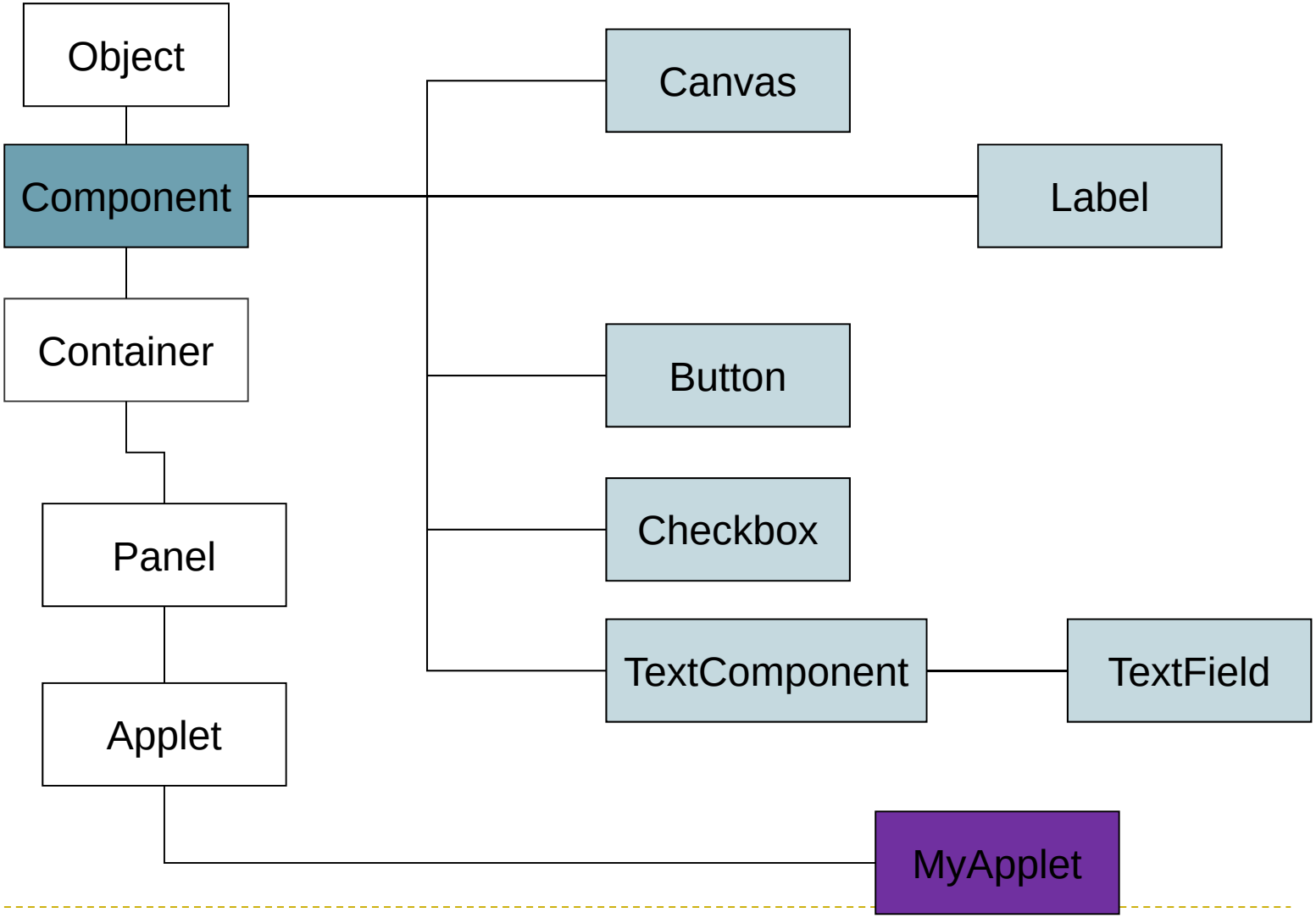
- マジックナンバーはないか？
- わかりにくいフィールド名、メソッド名はないか？
- Board、Rectangleクラスに、入出力のメソッドはあるか？
- Rectangleクラスに必要なメソッドがあるか？
- Commandクラスで、直接Rectangleの属性を操作していないか？

GUI入門(2)

本日の説明

- GUI
  - 入力
    - イベントで取得したい情報とその処理
  - GUIコンポーネントの配置
- 長方形エディタの仕様変更とその実装

Applet上で利用できる様々な部品（Component）



```
>appletviewer class¥Test.html
1:create
2:move
3:expand
4:shrink
5:delete
6:deleteAll
7:intersect
1
create
width = ?
100
height = ?
100
x = ?
20
y = ?
30
c = ?
1:red
2:blue
3:yellow
4:gray
1
1:[w = 100,h = 100,x = 20.0,y = 30.0,color = red]
1
1:create
2:move
3:expand
4:shrink
5:delete
6:deleteAll
7:intersect
```

起動

Version 3

コマンド一覧を表示する：入力の要求

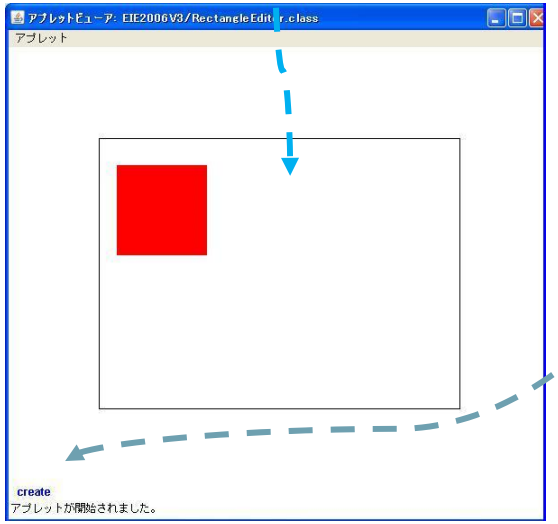
コマンドを選択する：入力

コマンド名の提示：メッセージの出力

コマンドに必要な値を要求する：入力の要求

コマンドに必要な値を入力する：入力

ボードにある長方形データの出力



Version 4　仕様(1)

起動

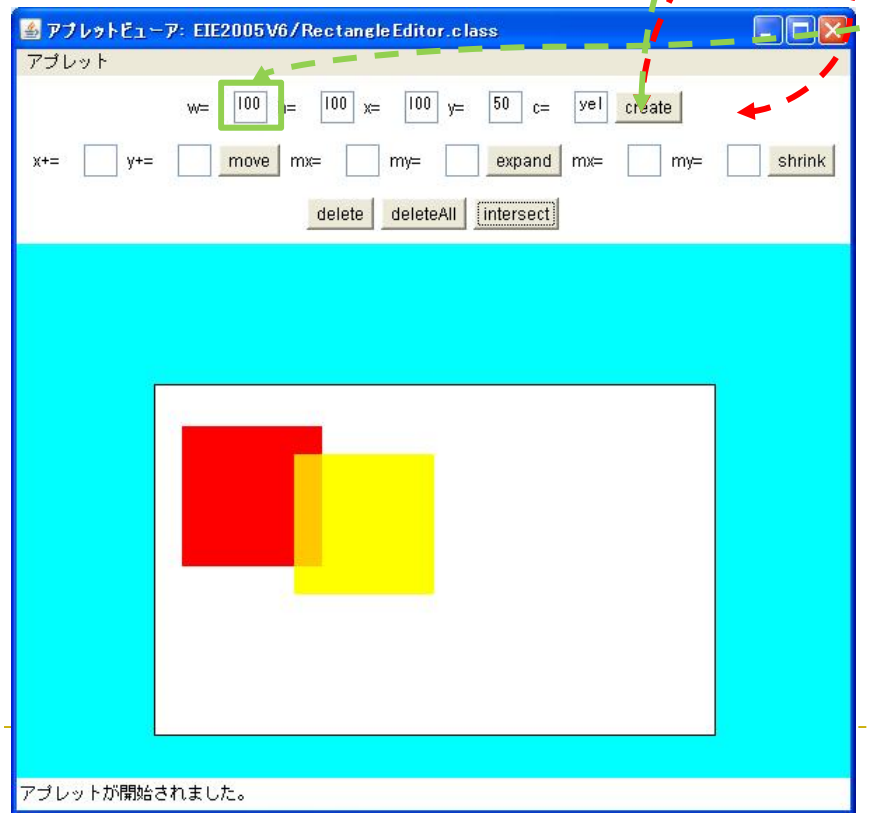
```
>appletviewer class¥Test.html
1:create
2:move
3:expand
4:shrink
5:delete
6:deleteAll
7:intersect
1
create
width = ?
100
height = ?
100
x = ?
20
y = ?
30
c = ?
1:red
2:blue
3:yellow
4:gray
1
1:[w = 100,h = 100,x = 20.0,y = 30.0,color = red]
1
1:create
2:move
3:expand
4:shrink
5:delete
6:deleteAll
7:intersect
```

コマンド一覧を表示する：入力の要求

コマンドを選択する：入力

コマンドに必要な値を要求する：入力の要求

コマンドに必要な値を入力する：入力



Version 4　仕様(2)

起動

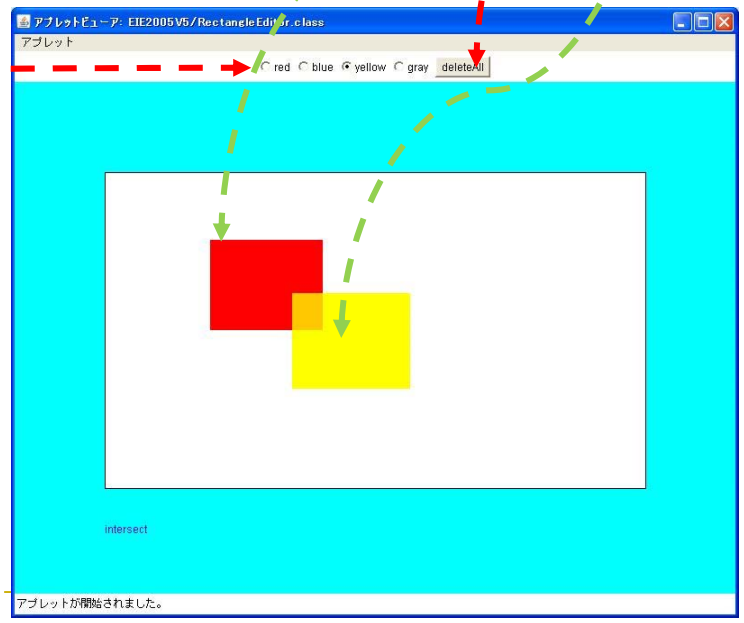
```
>appletviewer class¥Test.html
1:create
2:move
3:expand
4:shrink
5:delete
6:deleteAll
7:intersect
1
create
width = ?
100
height = ?
100
x = ?
20
y = ?
30
c = ?
1:red
2:blue
3:yellow
4:gray
1
1:[w = 100,h = 100,x = 20.0,y = 30.0,color = red]
1
1:create
2:move
3:expand
4:shrink
5:delete
6:deleteAll
7:intersect
```

コマンド一覧を表示する：入力の要求

コマンドを選択する：入力

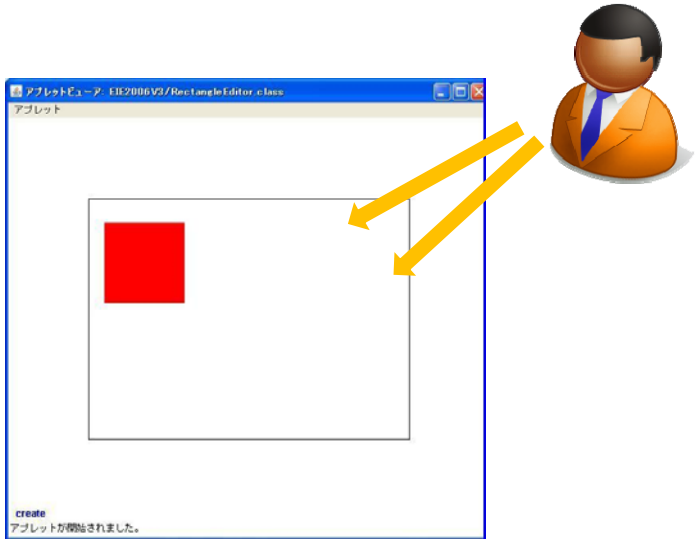
コマンドに必要な値を要求する：入力の要求

コマンドに必要な値を入力する：入力



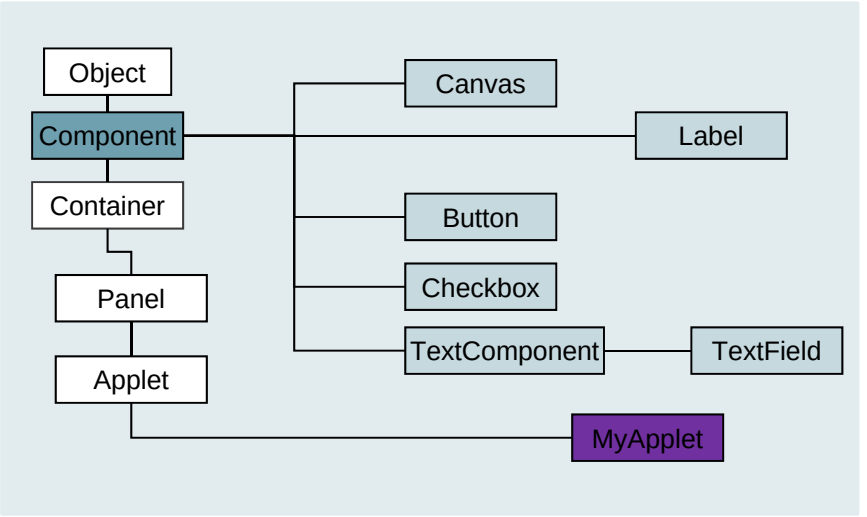
長方形エディタでの入力とは

- 入力の要求
  - コマンド一覧を表示する　⇒
    - コマンドをシステムに指示することが目的
    - 一覧が見えなくとも、コマンドを指定できればよい
  - コマンドに必要な値を要求する　⇒
    - 明示的に要求項目を表示する
    - コマンドに付随しているとして、明示的に要求しない
- 入力
  - コマンドを選択する　⇒
    - アプレットに対するイベント
  - コマンドに必要な値を入力する　⇒
    - アプレットに対するイベント
      - 長方形を指定する
      - 大きさを指定する
      - 位置を指定する



GUIにおける入力

コンポーネント(Component)が
さまざまなイベント(Event)を受け付ける



- コンポーネントが受けられる入力の種類

  ⇒ コンポーネント上でのイベントの種類と取得できるデータ

- イベントの処理
  - イベントはオブジェクトとしてそのイベントが生じたときのデータを保持している(イベントオブジェクトから得られるデータ)
  - コンポーネントはそのイベント発生元となるオブジェクトに付随した処理をもつ
  - イベントオブジェクトのもつ情報を使ってイベント発生時の処理を定義する

イベントの種類と取得できるデータ

- テキスト入力　TextField
  - 文字データ
- ボタン選択　Button
  - 「あるボタンを選択した」というデータ
- チェックボックス　Checkbox
  - 「ある項目が選択されている」というデータ
- マウス操作（押す・放す・クリックする・ドラッグする．．．）MouseEvent
  - 操作の位置データ
  - 操作のボタン
  - 操作の種類
- キー操作　KeyEvent
  - 指定したキーが押されているか否かのデータを取得する

イベントオブジェクトから得られるデータ

- コンポーネント上でのイベントで得たいデータは何か？
  - 何らかの刺激があったということ
  - その刺激が起こった場所
  - その刺激が起こったときの識別可能な状態
  - ActionEvent
    - なんらかの動き
  - ItemEvent
    - 項目が選ばれる
  - MouseEvent
    - マウスが押される・放される．．．　位置データ
  - KeyEvent
    - 押されたキーの種類

イベント発生元オブジェクトに付随した処理

- パッケージ　java.awt.evt

- ActionListener
  - Buttonが押されたときのイベントを処理する
  - ActionEvent

```
public void actionPerformed(ActionEvent evt) {
…
}
```

- ItemListener
  - Checkboxが選択された時のイベントを処理する
  - ItemEvent

```
public void itemStateChanged(ItemEvent e){
…
}
```

イベント発生元オブジェクトに付随した処理

- MouseListener
  - マウスが操作されたときのイベントを処理する
  - MouseEvent

```
public void mousePressed(MouseEvent evt){… }
public void mouseReleased(MouseEvent evt){…}
public void mouseClicked(MouseEvent evt){…}
public void mouseEntered(MouseEvent evt){…}
public void mouseExited(MouseEvent evt){…}
```

- MouseMotionListener
  - マウスが動作しているときのイベントを処理する
  - MouseEvent

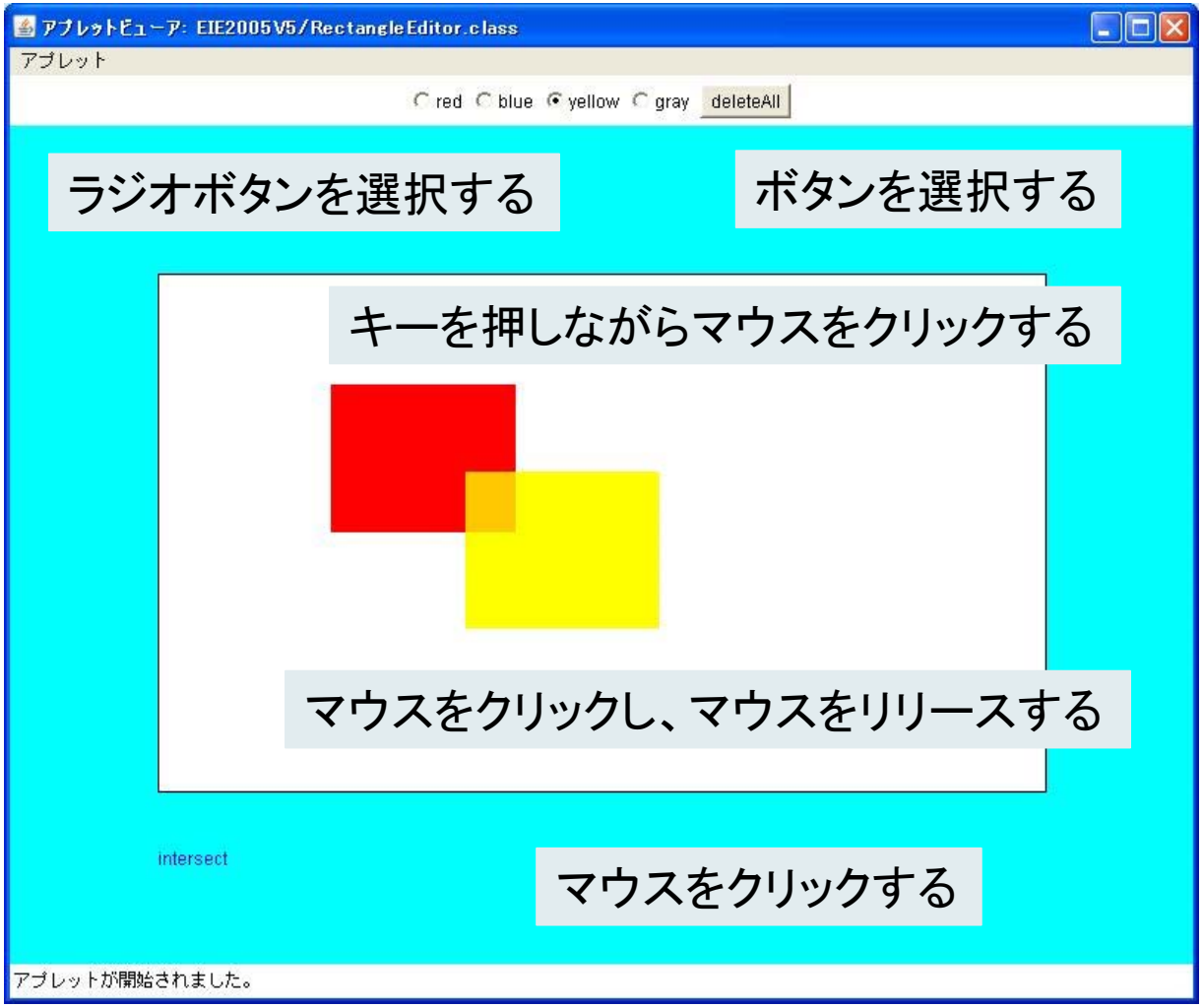
```
public void mouseDragged(MouseEvent evt){…}
public void mouseMoved(MouseEvent evt){…}
```

通常のPCでは
プログラムへの入力は
●キーを押す
●マウスをクリックする
ことでのみできる

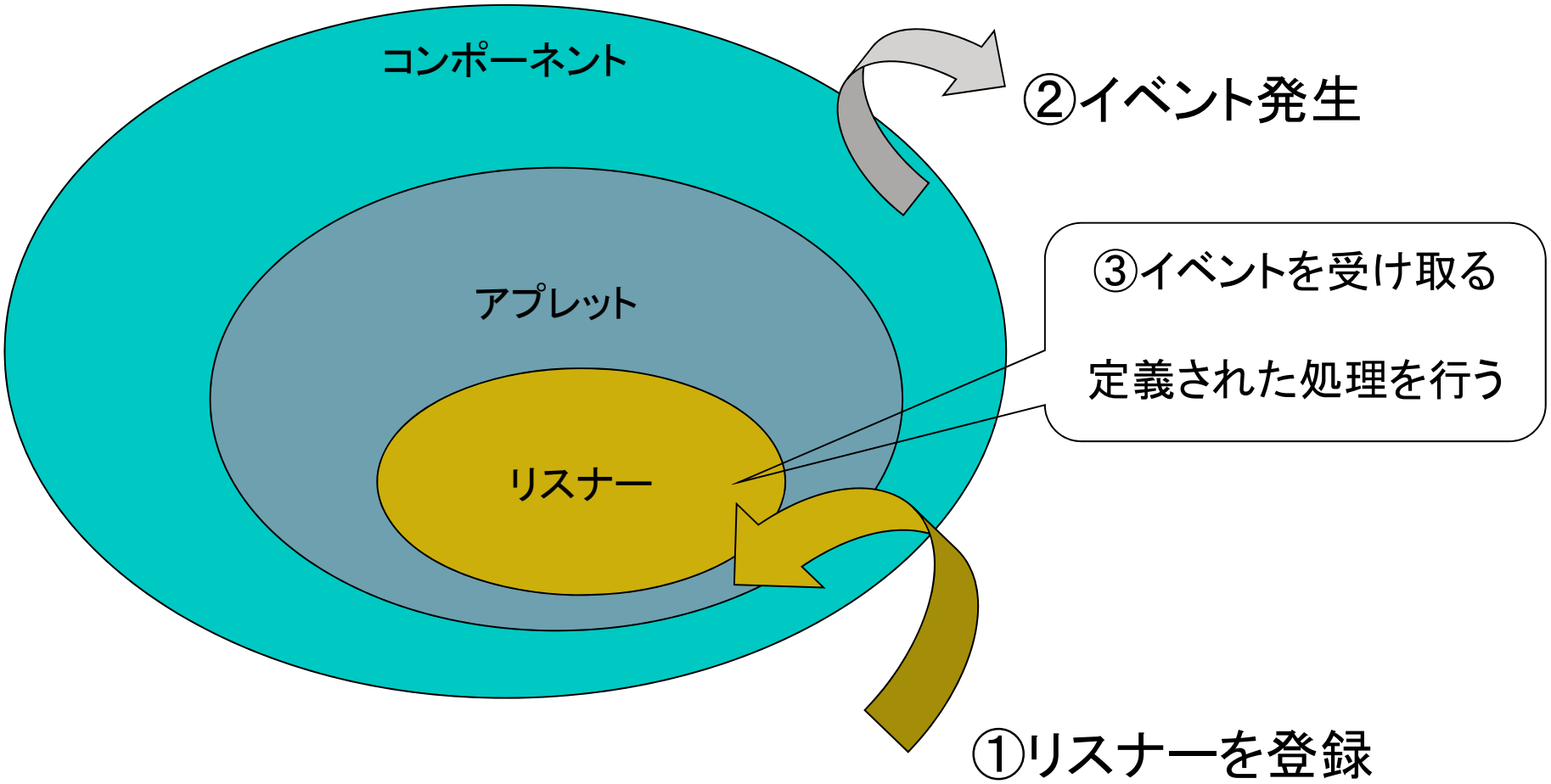
「ボタンを選択した」つもりでも、これも「マウスをクリックした」ことと同じ！？

1つのアプレットに対するこれらのイベントをどのように区別すればよいのか？

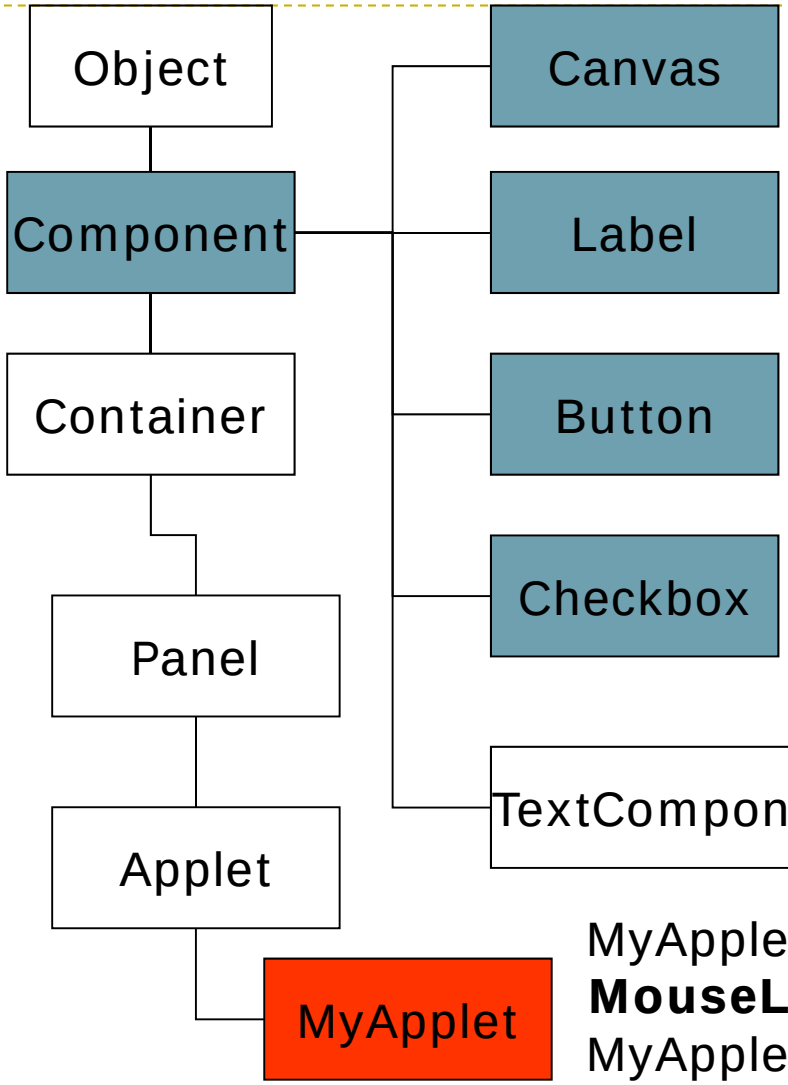
ボタンの選択やラジオボタンの選択をマウスの操作を区別して検知できればよい？



イベントモデル


コンポーネント
アプレット
リスナー
②イベント発生
③イベントを受け取る
定義された処理を行う
①リスナーを登録

イベントの処理



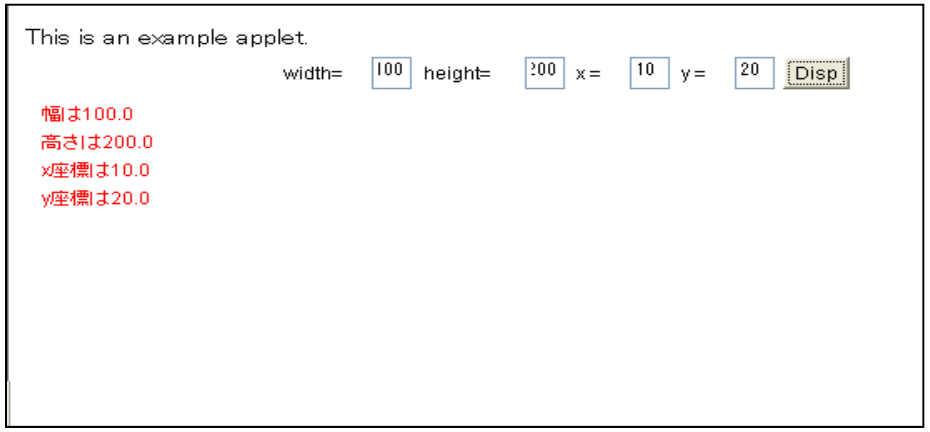
Buttonからイベントを受け取るために
ActionListenerを登録する
Buttonが押されたときの振る舞いを
actionPerformedで定義する

Checkboxの項目イベントを受け取るために
ItemListenerを登録する
Checkboxが選択されたときの振る舞いを
itemStateChangedで定義する

MyAppletでマウスイベントを受け取るために
MouseListenerを登録する ⟶ implements
MyApplet上でマウスイベントが発生したときの
振る舞いをmouse***で定義する

サンプル

- CheckboxExample
  - Checkbox
  - CheckboxGroup
  - 複数のチェックボックスをグループにして単一選択ができるようにする。選択された色で文字列を表示する。



- ActionExample
  - Label
  - TextField
  - Button
  - ラベルに対応する値をテキストフィールドに入力し、Dispボタンを押すことで表示する。

CheckboxExample

パッケージの宣言

APIのインポート

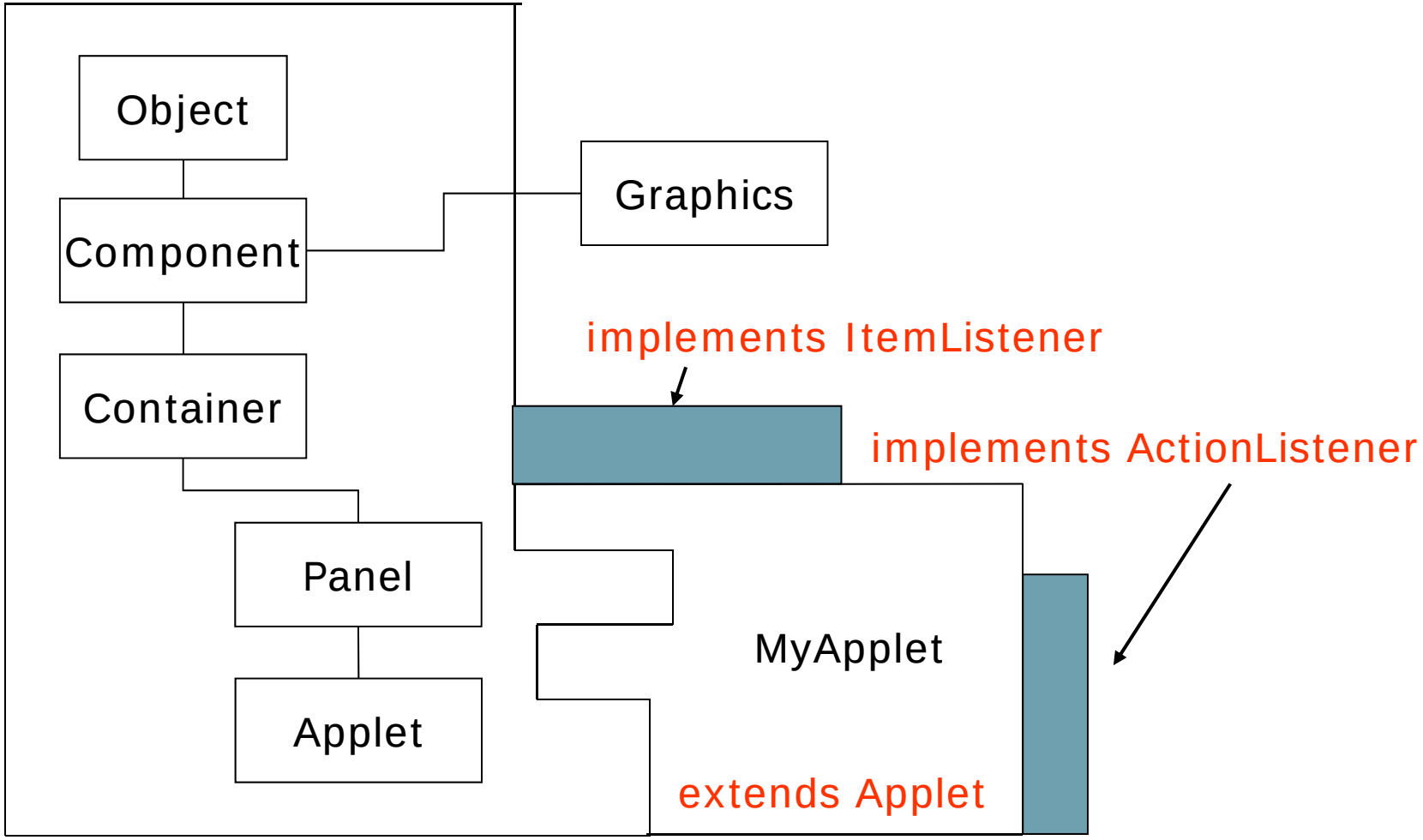
```
package Example;
import java.applet.Applet;
import java.awt.Checkbox;
import java.awt.CheckboxGroup;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
/**
 * ラジオボタンを定義する:Checkboxを使って択一式のラジオボタンをつくる。
 * Checkboxが選択されたことを感知するため、ItemListenerを実装する。
 */
public class CheckboxExample extends Applet implements ItemListener {
        Checkbox chx1, chx2, chx3, chx4;
        boolean b1, b2, b3,b4;
        Color color= Color.red;
```

クラスの宣言
Applet を継承 = CheckboxExampleクラスはAppletの一種
Appletの基本動作(init, start, stop, destroy)の内必要なものを定義する。

ItemListenerを実装
= CheckboxExampleクラスにはチェックボックスが選択されたときの振舞いを定義する。
そのメソッドは void itemStateChanged(ItemEvent e)

基底のクラス(この場合はCheckboxExample)が検知したい入力(イベント)に関するListener
をすべて実装する

フレームワーク


Object
Component
Container
Panel
Applet
Graphics
implements ItemListener
implements ActionListener
MyApplet
extends Applet

```
/**
 * 色を指定するためのラジオボタンの生成:ここでは4つ生成。
 * Checkboxを必要な数だけ作成し、グループ化する。各Checkboxにはラベル、グループ、初期状態を設定する。
 * ここではredが選択されているということ。
 * 各CheckBoxが選択されていることを感知するためにItemListenerを登録する。
 * CheckBoxをアプレットに追加する。
 */
        public void init(){
            CheckboxGroup cbg = new CheckboxGroup();
            chx1 = new Checkbox("red",cbg,true);
            chx1.addItemListener(this);
            add(chx1);

            chx2 = new Checkbox("blue",cbg,false);
            chx2.addItemListener(this);
            add(chx2);

            chx3 = new Checkbox("yellow",cbg,false);
            chx3.addItemListener(this);
            add(chx3);

            chx4 = new Checkbox("gray",cbg,false);
            chx4.addItemListener(this);
            add(chx4);
        }
```

グループ　cbg　を作成

1つめのチェックボックスchx1を作成
ラベルはred　所属するグループはcbg
初期状態はtrue、
すなわち選択されている状態

作成されたチェックボックス　chx1　に
ItemListenerを登録

chx1　を自分自身（このアプレット）
に追加

```
/**
 * ラジオボタンが選択された時の動作
 * 選択されたボタンに応じて文字を表示する色をcolorに設定する。
 */
        public void itemStateChanged(ItemEvent e){
            if(e.getItemSelectable() == chx1) color = Color.red;
            if(e.getItemSelectable() == chx2) color = Color.blue;
            if(e.getItemSelectable() == chx3) color = Color.yellow;
            if(e.getItemSelectable() == chx4) color = Color.gray;

            b1 = chx1.getState();
            b2 = chx2.getState();
            b3 = chx3.getState();
            b4 = chx4.getState();
            repaint();
        }

/**
 * 選択された色で文字を表示する。
 * すべてのボタンの状態を表示する。
*/
        public void paint(Graphics g) {
                g.setColor(color);
                g.drawString("色は" + color, 10, 50);
                g.drawString("今の状態は" + b1+ b2+ b3 + b4, 10, 70);
        }
}
```

色を設定する

現在選択されているチェックボックスオブジェクトを調べる

各チェックボックスの選択状態を調べる

ActionExample

パッケージの宣言

APIのインポート

```
package Example;
import java.applet.Applet;
import java.awt.TextField;
import java.awt.Label;
import java.awt.Button;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
public class ActionExample extends Applet implements ActionListener {
        ...
```

クラスの宣言
Applet　を継承
　＝　ActionExampleクラスはAppletの一種
Appletの基本動作(init, start, stop, destroy)の内必要なものを定義する。

ActionListenerを実装
　＝　ActionExampleクラスには何かアクションが生じたときの振舞いを定義する。
　　そのメソッドは　void actionPerformed(ActionEvent evt)

基底のクラス(この場合はActionExample)が検知したい入力(イベント)に関するListener
をすべて実装する

```
private final int SIZE = 4;
private TextField[] valueFields = new TextField[SIZE];
private Label[] labels = new Label[SIZE];
private double[] values = new double[SIZE];
private Button dispButton;

public void init( ) {
        labels[0] = new Label("width=");
        labels[1] = new Label("height= ");
        labels[2] = new Label("x =");
        labels[3] = new Label("y =");
        for (int i = 0; i < SIZE; i++) {
                this.add(labels[i]);
                valueFields[i] = new TextField("0",5);
                this.add(valueFields[i]);
        }
        this.dispButton = new Button("Disp");
        this.add(dispButton);
        dispButton.addActionListener(this);
}        ...
```

テキストフィールド
文字列を入力後リターンキーを押すことで
イベントが発生する ⇒ 1行分の文字列

addを使って並べた順序
通りに配置される

Component の一種である
ButtonオブジェクトDispButtonをつくる
オブジェクトを自クラスに add する
オブジェクトにListenerを登録する

```
public void actionPerformed(ActionEvent evt) {
        Button button = (Button)evt.getSource( );
        if (button == this.dispButton) {
            for (int i = 0; i < SIZE; i++) {
                    if (valueFields[i].getText( ).equals(""))
                        values[i] = 0;
                    else
                        values[i] = new Double(valueFields[i].getText( )).floatValue( );
            }
            this.repaint();
        }
}
```

いろいろなイベントが発生するので、
その evt の発生元のオブジェクトを取り出す。

Buttonオブジェクトを比較する
同じオブジェクトを参照しているかを検査する

サンプル

- EventExample
  - マウスイベントに応じた処理
    - マウスが押されたとき - greenで塗りつぶされた円を描く
    - マウスがアプレットの領域に入ったとき - 背景色をblackにする
    - マウスがアプレットの領域から出たとき - 背景色をredにする
    - マウスが離されたとき - 背景色をwhiteにする
    - マウスがクリックされたとき - なにもしない

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.event.*;

public class EventExample extends Applet implements MouseListener{
        // field
        public void init(){ …}
        private void addDisk(int x, int y){
                ...
                repaint();
        }
        public void paint(Graphics g) { ... }

        public void mousePressed(MouseEvent evt){ ... }
        public void mouseEntered(MouseEvent evt){ ... }
        public void mouseClicked(MouseEvent evt){ ... }
        public void mouseReleased(MouseEvent evt){ ... }
        public void mouseExited(MouseEvent evt){ ... }
}
```

その他

- キー操作
  - MouseEventはInputEventの一種
  - InputEventのメソッドでキーの押されている状態をチェック

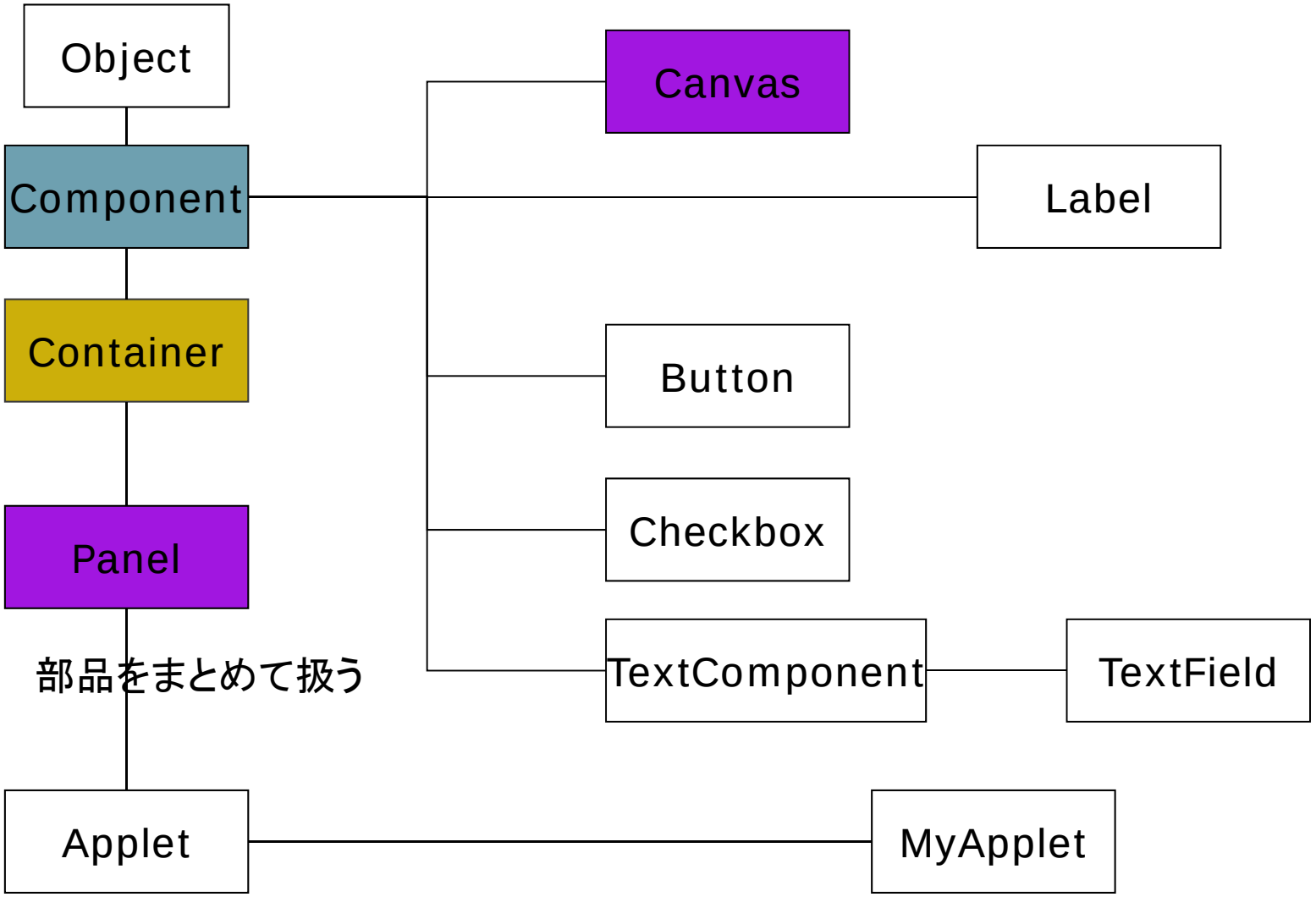
```
public void mousePressed(MouseEvent evt){
    …
    if (evt.isShiftDown() == true){
    …
    }
```

画面を整理する

- 1つの画面上で
  - 様々なイベントに対応する入力
  - 描画
  - 画像の出力

- どのように整理したらよいか？
  - まとめる仕組み
  - レイアウトする仕組み

描画コンポーネントを独立した
矩形の領域として作成する

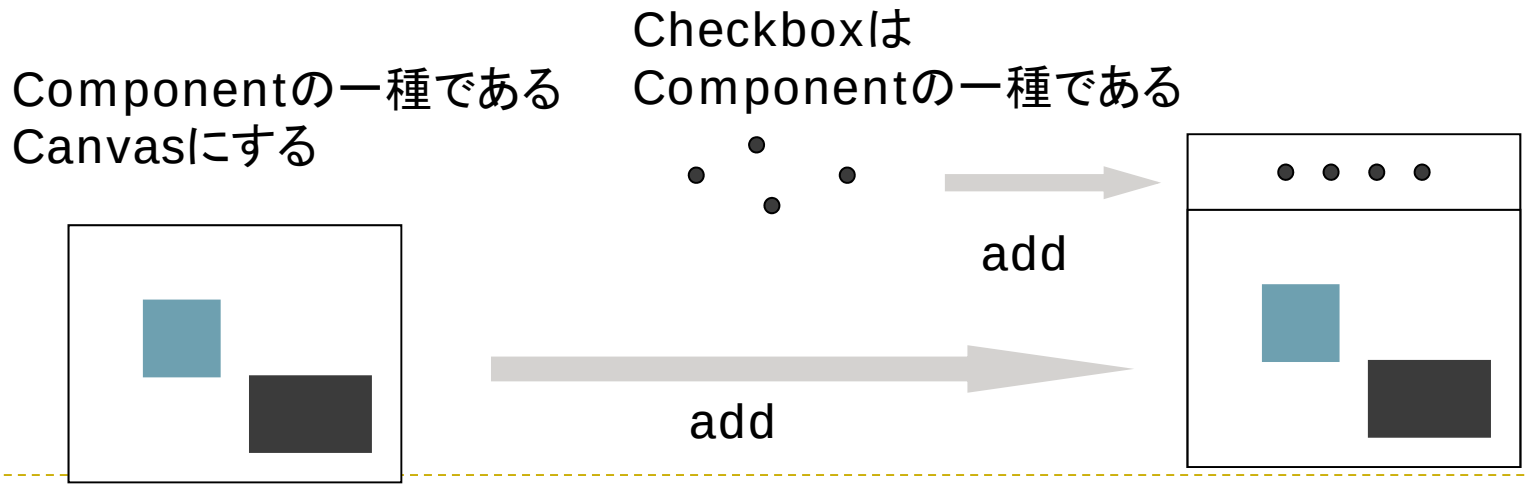


部品をまとめて扱う

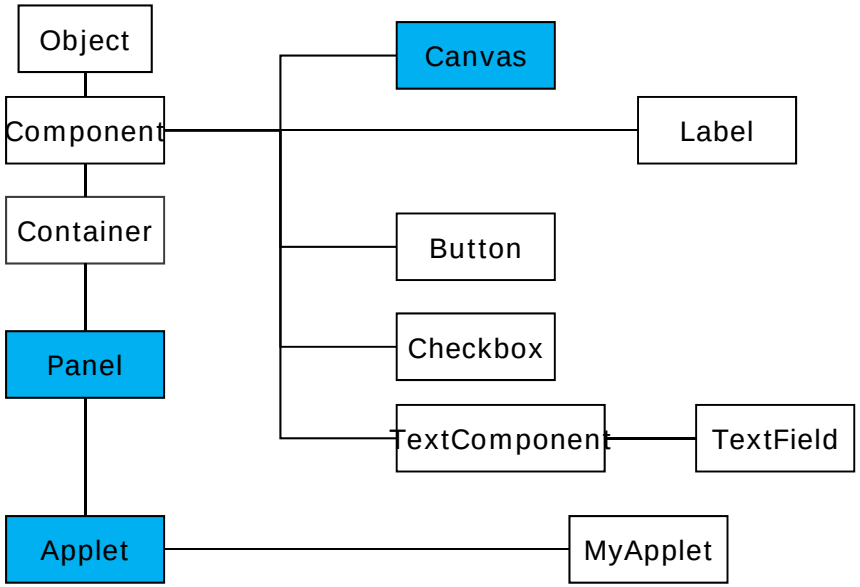
アプレット上に表示するもの

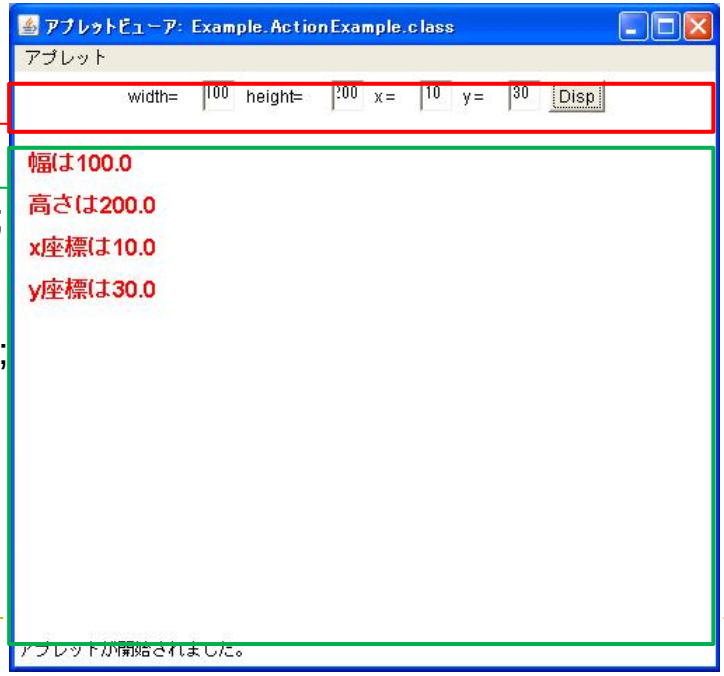
- 一組になった複数のボタン・出力イメージ

- １つの入れ物に複数の部品（Component）を追加する
- add: Container Component　を追加する
- ContainerはComponentの一種であり、Canvas,Button,Checkbox等の集合体



```
package example;
import java.applet.Applet;
import java.awt.TextField;
import java.awt.Label;
import java.awt.Button;
import java.awt.Color;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Canvas;
import java.awt.Panel;
import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class ActionExample extends Applet{
        InputPanel input;
        DisplayCanvas display;
        public void init( ) {
                display = new DisplayCanvas(this);
                input = new InputPanel(this);
                add(input, BorderLayout.NORTH);
                add(display, BorderLayout.SOUTH);

        }
        public void paint(Graphics g) {
                this.display.display(g);
    }
}
```

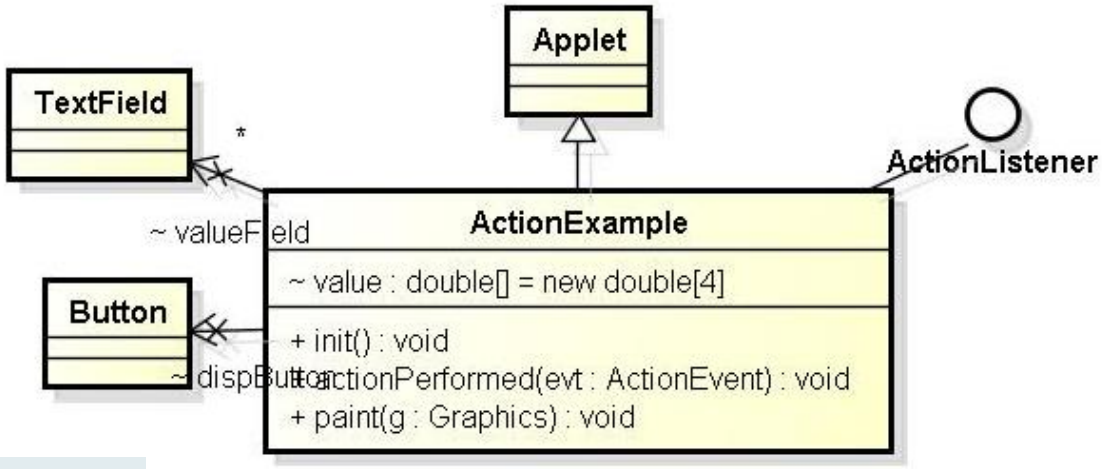




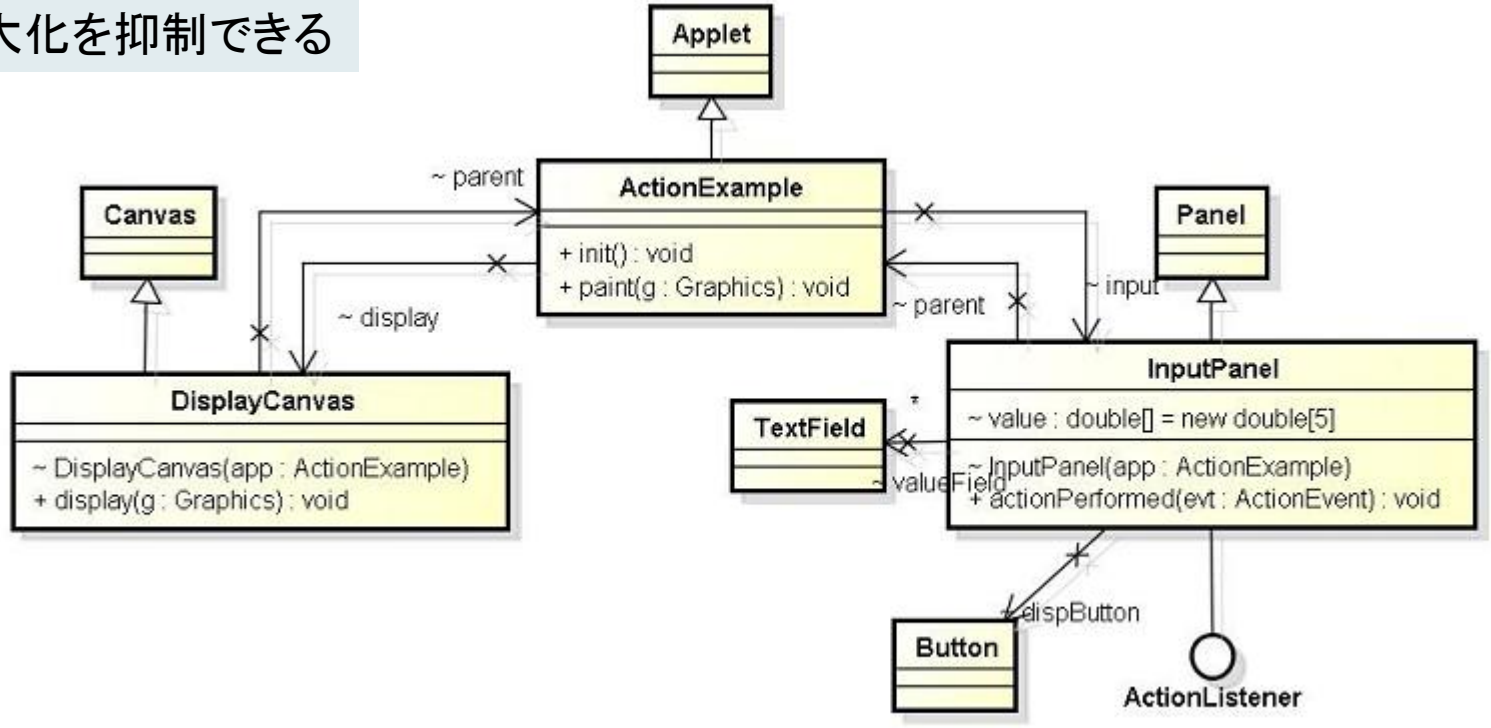
```
class DisplayCanvas extends Canvas {
  ActionExample parent;
  DisplayCanvas(ActionExample2 app){
        this.parent = app;
  }

  public void display(Graphics g) {
        Font font = new Font("TimesRoman",Font.BOLD,12);
        g.setFont(font);
        g.setColor(Color.red);
        FontMetrics fm = g.getFontMetrics(font);
        int h = fm.getHeight();
        g.drawString("幅は" + this.parent.input.value[0], 10, 70);
        g.drawString("高さは" + this.parent.input.value[1], 10, 70+h);
        g.drawString("x座標は" + this.parent.input.value[2], 10, 70 + h*2);
        g.drawString("y座標は" + this.parent.input.value[3], 10, 70 + h*3);
  }
}
```

```
class InputPanel extends Panel implements ActionListener {
        TextField valueField[ ] = new TextField[4];
        double value[] = new double[4];
        Button dispButton;
        ActionExample parent;
        InputPanel(ActionExample app) {
                parent = app;
                Label label[ ] = new Label[4];
                label[0] = new Label("width=");
                label[1] = new Label("height= ");
                label[2] = new Label("x =");
                label[3] = new Label("y =");
                for (int i = 0; i < 4; i++) {
                        this.add(label[i]);
                        this.valueField[i] = new TextField( );
                        this.add(this.valueField[i]);
                }
                this.dispButton = new Button("Disp");
                this.add(this.dispButton);
                this.dispButton.addActionListener(this);
        }
```

```
public void actionPerformed(ActionEvent evt) {
   Button button = (Button)evt.getSource( );
   if (button == this.dispButton) {
     for (int i = 0; i < 4; i++) {
           if (this.valueField[i].getText( ).equals(""))
              this.value[i] = 0;
           else
              this.value[i] = new Double(this.valueField[i].getText()).doubleValue( );
          }
      this.parent.repaint();
      }
   }
}
```

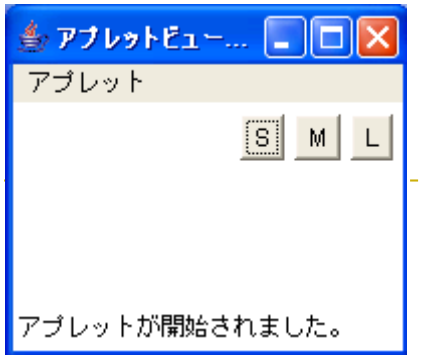


PanelやCanvasを使うと
クラスの肥大化を抑制できる



レイアウト

- FlowLayout
  - Componentを左から右に等間隔に配置する
- BorderLayout
  - Componentを上下左右、中央の位置に配置する
- GrigLayout
  - Componentを同じ大きさで格子状に配置する



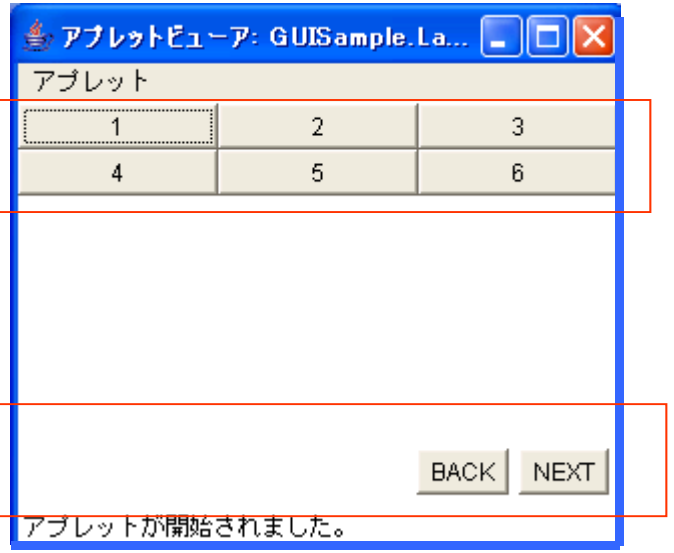
パネルを使用してレイアウトを変える

```
package GUISample;
import java.applet.Applet;
import java.awt.*;                    // Button, Panel, BorderLayout, GridLayout,
FlowLayout

public class PanelSample extends Applet{
     public void init( ) {
                    setLayout(new BorderLayout( ));
                    Panel panel1 = new Panel( );
                              panel1.setLayout(new GridLayout(2, 3));
                              panel1.add(new Button("1"));
                              panel1.add(new Button("2"));
                              panel1.add(new Button("3"));
                              panel1.add(new Button("4"));
                              panel1.add(new Button("5"));
                              panel1.add(new Button("6"));
                    Panel panel2 = new Panel( );
                              panel2.setLayout(new
FlowLayout(FlowLayout.RIGHT));
                              panel2.add(new Button("BACK"));
                              panel2.add(new Button("NEXT"));
                    add("North", panel1);
                    add("South", panel2);
     }
}
```

全体のレイアウトをBorderLayoutで決める
2つのPanelを配置する

CENTER
EAST
NORTH
SOUTH
WEST

6つのボタンを1つのPanelとして扱う
このPanelのレイアウトはGridLayout
ここでは2×3のグリッドの指定がある



2つのボタンを1つのPanelとして扱う
このPanelのレイアウトはFlowLayout
ここでは右寄せの指定がある

CENTER
LEFT
RIGHT

長方形エディタ

標準入力からの入力は何か？

- コマンド（create, move, expand,…）
- コマンドの引数
  - 長方形の幅・高さ・x座標・y座標・色
  - 長方形の選択
  - 移動距離
  - 拡大・縮小の倍率

⇒これらのデータをAppletからの入力に切り替える

入力の方法

- 標準入力からの文字データ

→ Appletに対する様々なイベント

  - テキスト入力
    - 他の動作に必要な文字データをApplet上で入力する
  - ボタン選択
    - ボタンを選択するとそのボタンに対応した動作を行う
    - ボタンを選択することでデータを入力する
  - マウス操作（押す・放す・クリックする・ドラッグする．．．）
    - 操作の位置・操作のボタン・操作の種類によってさまざまなデータを生成する
  - キー操作
    - 指定したキーが押されているか否かのデータを取得する

長方形エディタのGUI化

- 標準出力にボードと長方形を表示する

⇒ 四角形の描画・画面の書き換え

入力仕様（1）

- 標準入力からコマンドを選択する

⇒ ボタンでコマンドを選択する

- 標準入力からコマンドパラメータ（長方形の位置と大きさ・長方形の選択・色の設定）を入力する

⇒ テキストフィールド・マウスクリック

  - テキスト入力によるパラメータの設定
  - マウスイベントによる長方形の特定（ボード内のどの長方形であるかを判定する）

長方形エディタのGUI化（つづき）

入力仕様（2）

- 標準入力から長方形の位置と大きさを指定する

⇒ マウスプレス・マウスリリース・

＋キー（シフト・コントロール等々）の押された状態

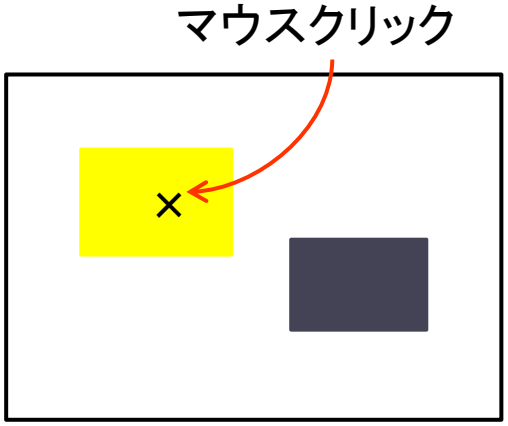
- マウスイベントによるパラメータの設定
  - 位置・大きさ・倍率の計算（コマンドに必要なパラメータの値をマウスイベントから得られたデータから計算する）
- 標準入力から色を指定する

⇒ ラジオボタンの選択されている状態

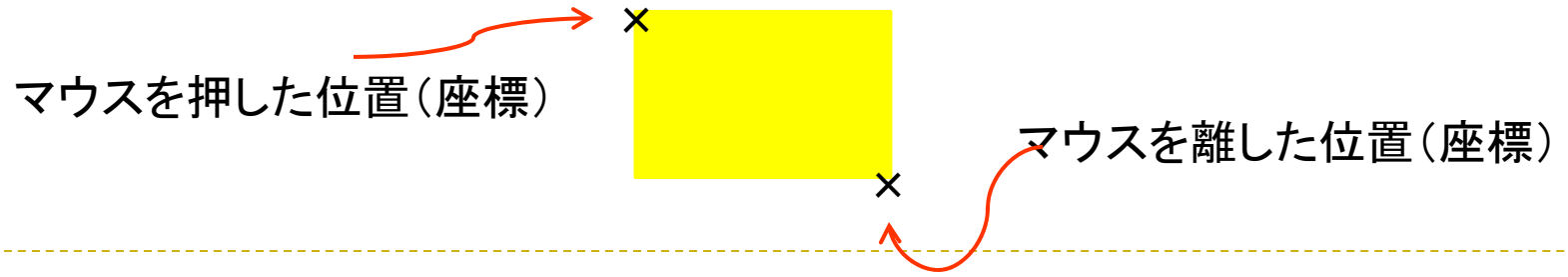
値の受け渡しの変更

- 長方形の選択
  - マウスクリックされた位置（座標）
  - ボード内の長方形の占める範囲

  から計算し、長方形を特定する



- 仕様変更（2）の場合の長方形の作成
  - マウスを押した座標とマウスを離した座標から長方形の幅と高さを計算



長方形エディタで使用するComponentのクラス

- Label
  - テキストフィールドのタイトルを作成する
- TextField
  - コマンドのパラメータを入力する
- Button
  - コマンド
- Checkbox
  - 色を選択する
- Canvas
  - 長方形を描画する領域を定義する