応用Javaプログラミング
第5回

1. GUIプログラミング2
2. Mediatorデザインパターン

課題解答の提出に関する注意：課題の出された日の翌日以降にサーバーにアップロードすること！

前回作った簡易貯金電卓の見直し

前回作った貯金電卓に下記の条件を追加したい：

① クリアボタンがいつでも押せるが，3つのテキストフィールドに空ではないデータが入っているときのみ，計算ボタンが押せる．

② 計算ボタンを押してから，クリアボタンを押さない限り，3つのテキストフィールドにデータを入力できない．

このような多数のオブジェクトの間の調整を行う必要があるとき，Mediatorデザインパターンを利用すべき．
Mediatorが相談役をつとめ，各オブジェクトが状態の変化をMediatorに随時報告し，Mediatorがオブジェクトからの報告に応じてオブジェクト間の調整を集中管理する．

Mediatorインターフェース

```
package savingCal ;

public  interface MyMediator {
      //指定の部品componentからの報告に基づいて
      //管理下の全部品の状態を管理するためのメソッド
      void colleagueChanged(Jcomponent component) ;
}
```

条件を付けたときの電卓

```
package savingCal ;

//import ...（略）
public class SavingCalMediator extends  SavingCal3  {
    private MyMediator  mediator; //部品の状態を集中管理するMediator

    @Override
    public void initialize( ) {
        super.initialize( )  ;

        //Mediatorを作っておく
        mediator = new MyMediator( ) {
            public void colleagueChanged(JComponent component) {
                //現在の状態を調べてから，設定を変える
                analyzeState(component) ; //自作メソッド
            }
        }

    }
```

次頁へ

条件を付けたときの電卓（続）

```
public void analyzeState(JComponent component) {
    if (component == getButton(0) ) {
        for (int i = 0 ; i < textFields.length ; i++)
            getTextField(i).setEnabled(true);
        getButton(1).setEnabled(false);
    } else if ( component == getButton(1) ) {
        for (int i = 0 ; i < textFields.length ; i++)
            getTextField(i).setEnabled(false);
        getButton(1).setEnabled(false);
    } else {
        int i = 0 ;
        for ( ; i < getTextFields().length ; i++)
            if ( getTextField(i) == component ) break ;
        if ( i == getTextFields().length ) return ;
        i = 0 ;
        for ( ; i < getTextFields().length ; i++)
            if (getTextField(i).getText().isEmpty()) break ;
        if ( i == getTextFields().length ) getButton(1).setEnabled(true);
        else getButton(1).setEnabled(false);
    }
}
```

次頁へ

条件を付けたときの電卓（続）

```
@Override
public void arrangeComponents() {
    super.arrangeComponents();

    //押せるかの初期設定．これはクリアボタンが押されたときの状態と同じ．
    mediator.colleagueChanged( getButton( 0 ) ) ;
}
```

次頁へ

条件を付けたときの電卓（続）

```
//部品にリスナーを追加
@Override
public void addListeners() {
    super.addListeners();

    addTextFieldListener() ; //テキストフィールドに自作リスナーを登録

    //ボタンにもう1つリスナーを登録
    for (int i = 0 ; i < btnNames.length ; i++)
        getButton(i).addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                //Mediatorに状態の変化を伝える
                mediator.colleagueChanged(JComponent)e.getSource());
            }
        }) ;
    }
}
```

次頁へ

条件を付けたときの電卓（続）

```
    //テキストフィールドに「変化認識リスナー」を付けるメソッド
    private void addTextFieldListener() {
        DocumentListener al = new DocumentListener() {
            public void changedUpdate(DocumentEvent e) {
                mediator.colleagueChanged(getTextField(0));
            }
            public void removeUpdate(DocumentEvent e) {
                mediator.colleagueChanged(getTextField(0));
            }
            public void insertUpdate(DocumentEvent e) {
                mediator.colleagueChanged(getTextField(0));
            }
        } ;

        for (int i = 0 ; i < getTextFields().length ; i++)
            getTextField(i).getDocument().addDocumentListener(al) ;
        }
    }
} //SavingCalMediatorクラスの終わり
```

Mediatorデザインパターン

ポイント: メンバーはみんな相談役だけに報告し，メンバーへの指示は相談役だけから来るようにする.

Mediator役（相談役）: **Colleague役**と通信を行って，調整を行うためのインターフェース（API）を定める役. 貯金電卓の例では，MyMediatorインターフェースがこの役をつとめている.

ConcreteMediator役（具体的な相談役）: **Mediator役**のインターフェースを実装し，実際に調整を行う役. 貯金電卓の例では，フィールドmediatorを作るときに定めた無名クラスがこの役を担う.

Colleague役（同僚役）: **Mediator役**と通信を行うインターフェース（API）を定める役. 貯金電卓の例では，この役を設けていない.

ConcreteColleague役（具体的な同僚役）: **Colleague役**を実装する役. 貯金電卓の例では，3つのテキストフィールド と 2つのボタン と totalLabelがこの役をつとめている.

上記の枠組みに忠実に沿って書いた貯金電卓のプログラムはMediator.java, Colleague.java, ColleagueTextField.java, ColleagueButton.java, SavingCalMed.java からなる.

必須演習課題

下記の条件を満たす肥満度電卓を作れ：

条件：身長と体重が1以上の実数，性別が1か2のときのみ，
計算ボタンが押せるようにせよ．

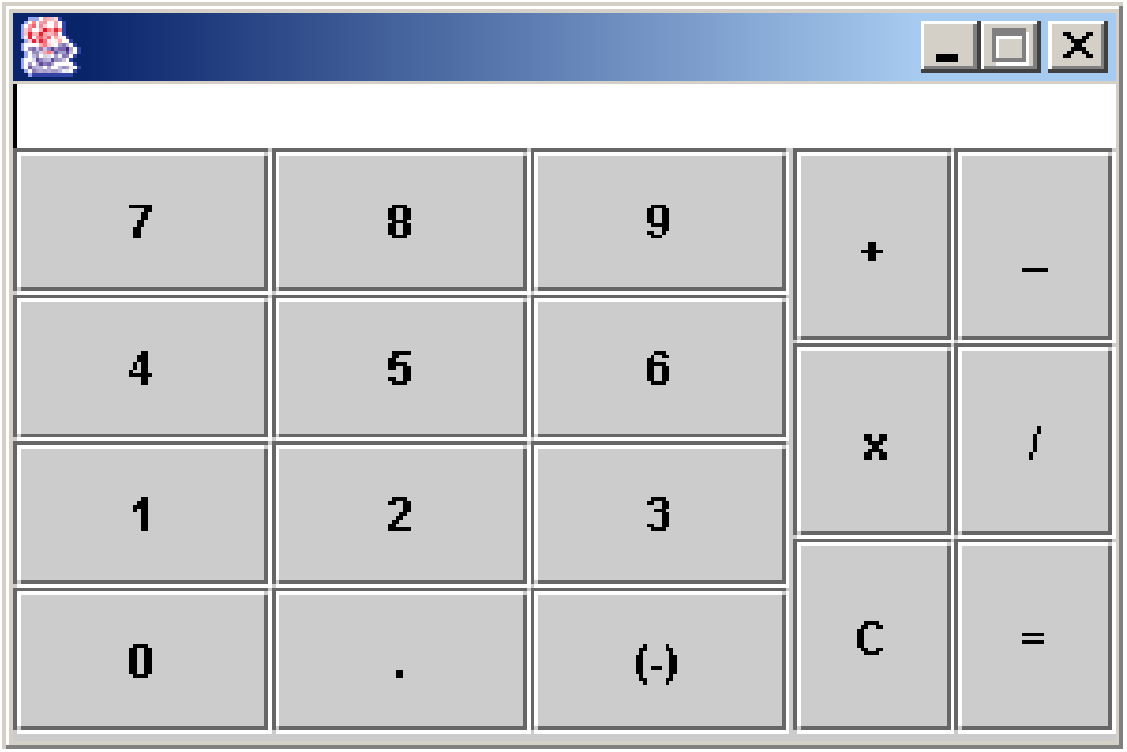
注：できれば，前回の課題で作ったクラスを継承すること．

標準体重の計算法：
男の場合　（身長－80）＊0.7
女の場合　（身長－70）＊0.6

肥満度の計算法：
（体重－標準体重）／体重

練習問題（ちょっと難しい）

AbstractGUIクラスを継承して，次の電卓を作れ：



提出しなくてよい．
提出しても採点しない．

この課題でGUI部品の配置が複雑なので，GridBagLayout を使うかまたは GridLayout を複数組み合わせて使うとよいであろう．