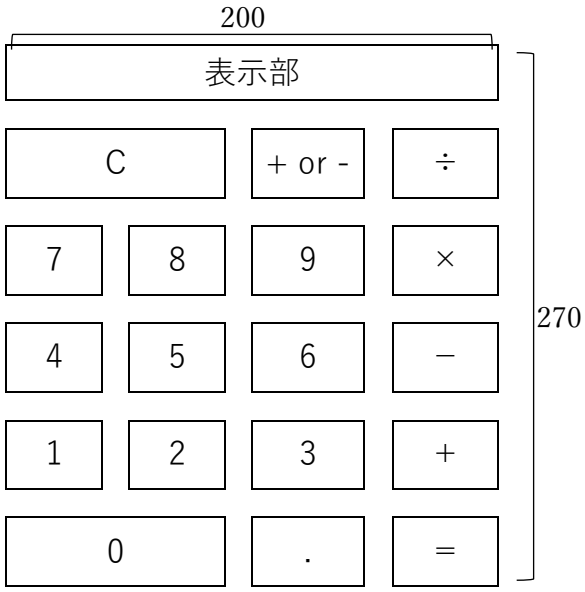
電卓アプリケーションの概要

授業課題で作成した電卓のアプリケーションのレポートになります。

こちらの無断利用による責任は負いません。参考にする分には構いません。

① 仕様

《完成イメージ》



・クリアボタンが押されたら，全てリセットされ，再び計算が可能

・小数点付きの数も扱える

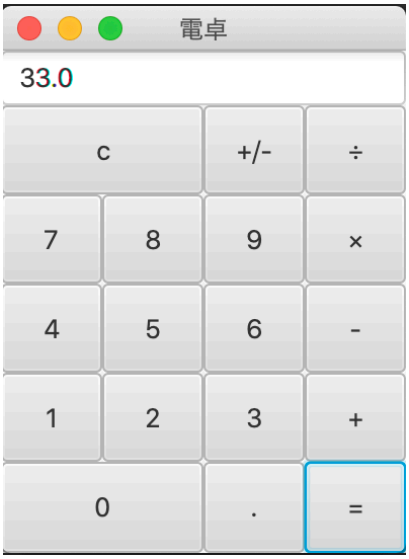
・加減乗除ができる

・不正な入力を行なっても再起動なしに計算できる

・マイナスの数も扱える

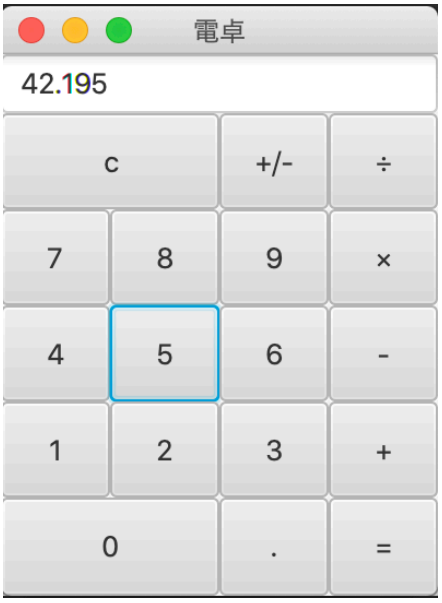
② 実行結果

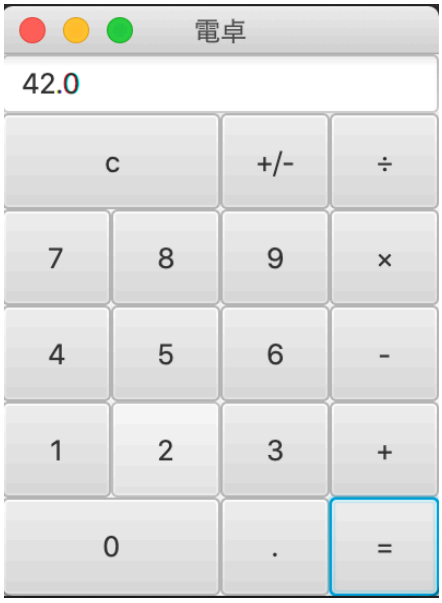
- 5+10+23=33



３つ以上の計算も行えた。

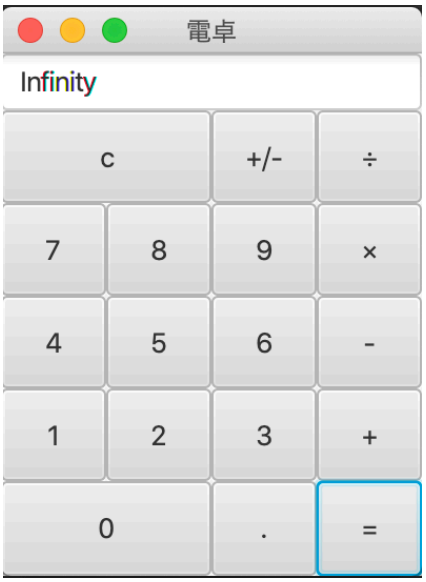
- 42.195-0.195=42.0





小数点付きの数の計算も扱えた。

- 3.14÷0

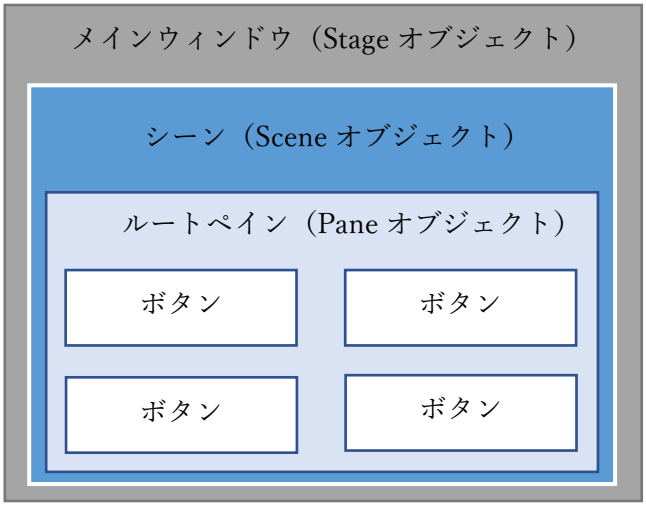


エラーが発生するが,「C」ボタンで再び通常の計算に戻ることができた。

③　プログラムについて

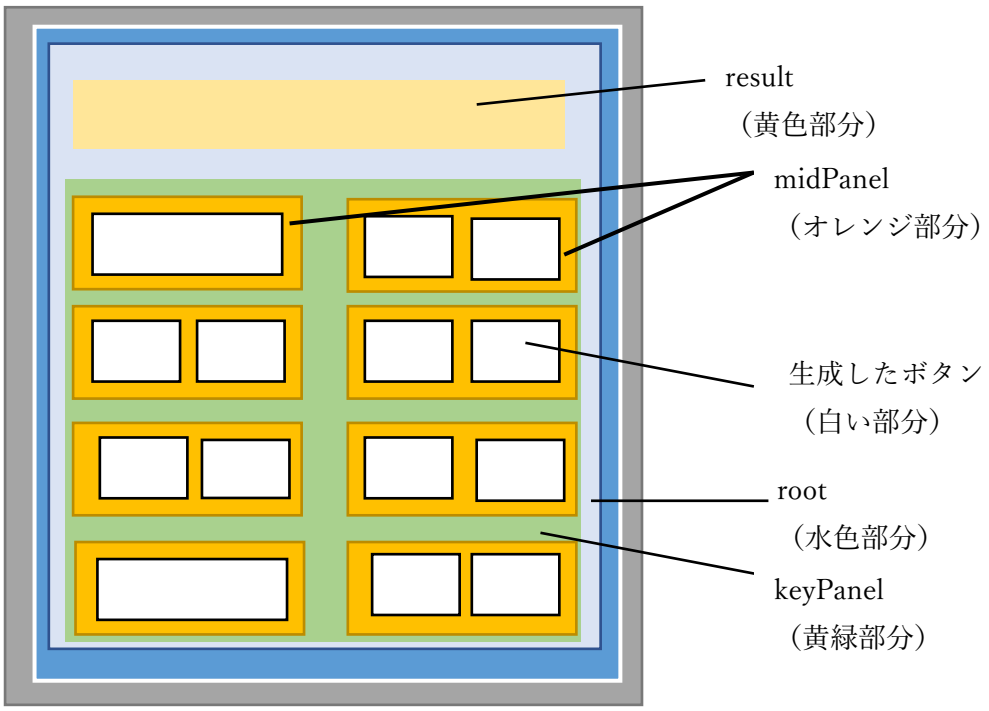
今回のアプリケーションは JavaFX を用いて作成した。

1. クラス名は「Dentaku2020」とし,JavaFX の Application クラスを継承している。
   結果の表示部を TextField を用いて result という名称にした。
   JavaFX の Button クラスを numbutton と calcbutton と clearbutton を生成。
2. start メソッドは表示画面の定義をしている。
   ①メインウィンドウ（Stage オブジェクト）の設定
   stage クラスの横幅を 200px，高さを 270px に設定し，タイトルを「電卓」とした。
   Stage オブジェクトに結果を表示する部分であるテキストフィールドの result を横幅 200px，高さ 20px で一番上に配置する。
   ②各種ボタンの生成
   i=0～10 の数字を numbutton[i]のボタンに表示する。
   s=String.valueOf(i);は，int 型である i を String 型に変換したものを String 型である s に代入している。
   numbutton[]，calcbutton[]，clearbutton に表示する文字を定義する。
   ③ボタンのサイズの指定
   仕様にあるように「０」と「c」のボタンは大きく，横幅 100px，高さ 50px とし，そのほかのボタンは横幅と高さを 50px の正方形とした。
   ④ボタンが押された時に呼び出すイベントハンドラを登録する。
   cleabutton.setOnAction(event -> ClearButtonPressed());
   は cleabutton(「c」)が押された時に，ClearButtonPressed()メソッドを呼び出し，動作する。
   ⑤ボタンの配置



JavaFX でのアプリケーションの階層構造は上図のようになっている。

今回の電卓のボタンの配置の階層構造は以下のようになっている。



ミドルサイズのキー（オレンジの部分）を生成し，そこに生成したボタンを GridPane を用いて配置する。

```
GridPane midPanel2 = new GridPane();
GridPane.setConstraints(numbutton[11],　0, 0);
GridPane.setConstraints(calcbutton[0],　1, 0);
```

GridPane.setConstraints で指定されたボタンの配置を定義する。

```
midPanel2.getChildren().addAll(numbutton[11], calcbutton[0]);
```

getChildren().addAll で配置するボタンを midPanel2 に追加する。

次に，midPanel を keyPanel に配置していく。
同様にして，ルートペインの下地にあたる root に結果表示画面 result とボタンが配置されている keyPanel を配置していく。

root ペインを元に Scene を生成し，stage に scene を配置する。

```
stage.show();
```

はステージの show メソッドでアプリケーションウィンドウを表示する。

3. ClearButtonPressed はクリアボタンの定義をしている。
   画面に表示されているテキストを削除し，格納されている値を 0 にする。

4. NumButtonPressed (0〜10)は数字の書かれたボタンを定義している。

   result.appendText(numbutton[0].getText());
   は numbutton[0]に格納されている文字列（ここでは 0 ）を result にコピーし，表示画面に表示する。

5. NumButtonPressed11 は入力された数の正負を決めるボタンを定義している。

```
public void NumButtonPressed11() {
        resultValue1 = (Double.valueOf(result.getText()));
        if(resultValue1 <= 0) {//resultValue が負の時絶対値を返す
                resultValue1 = Math.abs(resultValue1);
                ans = String.valueOf(resultValue1);
                result.clear();
                result.appendText(ans);
        }else {//resultValue が正の時，負の値を返す
                resultValue1 = resultValue1 - resultValue1 * 2;
                ans = String.valueOf(resultValue1);
                result.clear();
                result.appendText(ans);
        }
}
```

   result に格納されている文字列をダブル型に変換し，resultValue1 に格納する。
   resultValue1 が 0 以下の時，絶対値を返し，resultValue1 に代入する。
   resultValue1 の値を文字列に変換し，ans に格納する。
   result をクリアにし，ans を result に表示する。

   同様にして，resultValue1 が正の時は負の値を返し，文字列に変換して result に表示する。

6. CalcButtonPressed(0～3)は演算子ボタンの定義をしている。

```
public void CalcButtonPressed0() {
        currentOp = 0;//0 を currentOp に代入（除算）
        resultValue = (Double.valueOf(result.getText()));//result の値を数値に変換
        result.clear();//テキストフィールドをクリアする
}
```

除算のボタンが押された時に起動し，
currentOp に除算の時０，乗算の時１，減算の時２，加算の時３を代入する。
resultValue に result に表示されている文字列をダブル型に変換した数値を代入する。
次の値を入力するために，テキストフィールドの文字列をクリアにする。

7. CalcButtonPressed4 は「＝」が押された時に起動する。

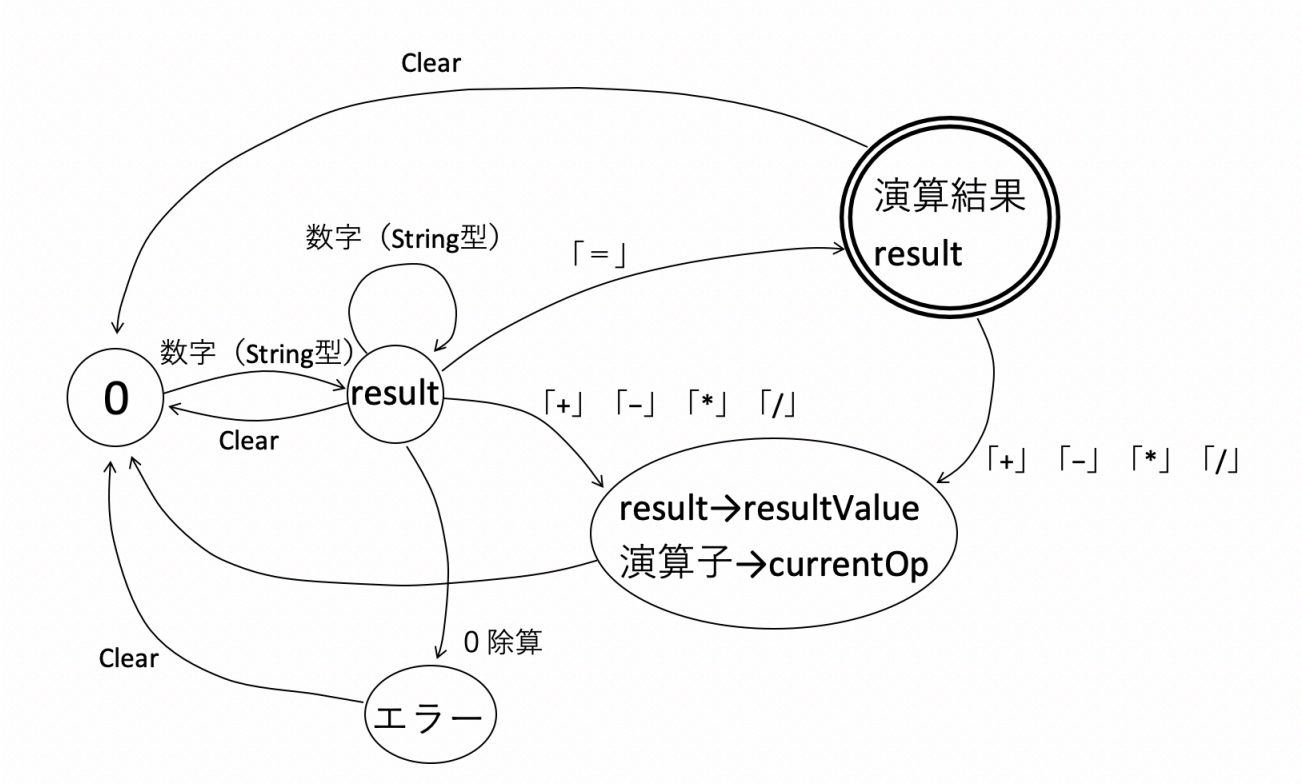
```
//=が押された時に起動する
        public void CalcButtonPressed4() {
                afterresultValue = (Double.valueOf(result.getText()));
                if(currentOp == 0) {
                        resultValue = resultValue / afterresultValue;
                }
                if(currentOp == 1) {
                        resultValue = resultValue * afterresultValue;
                }
                if(currentOp == 2) {
                        resultValue = resultValue - afterresultValue;
                }
                if(currentOp == 3) {
                        resultValue = resultValue + afterresultValue;
                }

                ans = String.valueOf(resultValue);
                result.clear();
                result.appendText(ans);
        }
```

result に格納されている文字列をダブル型に変換し，afterresultValue に格納する。
currentOp に格納された数値に応じて，resultValue と afterresultValue の演算を行い，その結果

を resultValue に代入する。
resultValue の値を文字列の型である String に変換し，ans に格納する。
一度，result をクリアし，result に ans に格納された文字列を表示する。

④ 動作原理

各キーを押した時の動作原理を状態遷移図を以下に示す。



初期値は 0 で，clear ボタンが押されたら，初期値に戻る。
数字ボタンが押されると result に数字が追加される。
演算ボタンが押されたら，入力された数と演算ボタンを記憶し，再び次の数字の入力に移行する。
イコールボタンが押されたら，記憶していた数値と次に入力された数値を記憶された演算子で計算を行い結果を出力する。
0 除算はエラーを出力し，Clear ボタンでリセットされ，再び計算できるようにする。

⑤ 考察

今回は JavaFX の GUI を用いて電卓を作った。授業では swing と JavaCC を用いていたが，私は javaFXのみで極力シンプルに作ることを意識した。基本はswingと同じ機能が用意されているので，作りやすかった。

ボタンを配置するところが難しく，一番時間がかかった。GridPane を 2 度用いて，ルートペインに階層構造を持たせて作るように工夫した。それによって，ボタンのサイズをかえることができた。ボタンの形をいろいろ変えてみたり，色を変えていけるとさらに面白いオリジナルの画面が作れると思った。

似たような部分が多いので，もっと for 文を用いてシンプルにすることができたと思う。それぞれの変数の名称などもっと工夫できればよかった。

テキストフィールドは String 型で，演算を行う時には double 型でないといけないのでそこの変換の部分をどのタイミングに置くかが難しかった。また，前に入力された数値をどのように保存して，演算子を記憶してなどの動作の順序を考えていくのが難しいと感じた。だが，状態遷移図に起こすことで“見える化”ができるので，「このような状態の時，どうなっているのか」というのが理解しやすく，プログラムを作る上で，このような UML を用いる重要性が理解できた。

今回，課題の指定にはなかったが，入力した数値にプラスとマイナスを決める機能も追加した。この機能は，一番最後に作ったのだが，以外とすんなりできた。

今後，追加したい機能は，ルートを与えるような機能や，関数電卓のようにテキストフィールドに数式を表示する機能など追加していきたい。

⑥　プログラム全体

プログラムの全文をここに記載する。

```
package dentaku2020;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class Dentaku2020 extends Application {

	TextField result;//結果の表示
	Button[] numbutton = new Button[12];
	Button[] calcbutton = new Button[5];
	Button clearbutton;
	int i = 0;
	String s;

	public void start (Stage stage) {
		//メインウィンドウの設定
		stage.setWidth(200);
		stage.setHeight(270);
		stage.setTitle("電卓");

		result = new TextField("");//結果の表示画面
		result.setPrefSize(200, 20);

		//数字ボタンの生成
		while(i < 10) {
			s = String.valueOf(i);
			numbutton[i] = new Button(s);
			i++;
		}
		numbutton[10] = new Button(".");
		numbutton[11] = new Button("+/-");
		//計算ボタンの生成
```

```
calcbutton[0] = new Button("÷");
calcbutton[1] = new Button("×");
calcbutton[2] = new Button("-");
calcbutton[3] = new Button("+");
calcbutton[4] = new Button("=");
clearbutton = new Button("c");
//ボタンのサイズ指定
numbutton[0].setPrefSize(100, 50);
for(i = 1; i < 12; i++) {
        numbutton[i].setPrefSize(50, 50);
};
for(i = 0; i < 5; i++) {
        calcbutton[i].setPrefSize(50, 50);
}
clearbutton.setPrefSize(100, 50);

//ボタンのイベントハンドラの登録
clearbutton.setOnAction(event -> ClearButtonPressed());
numbutton[0].setOnAction(event -> NumButtonPressed0());
numbutton[1].setOnAction(event -> NumButtonPressed1());
numbutton[2].setOnAction(event -> NumButtonPressed2());
numbutton[3].setOnAction(event -> NumButtonPressed3());
numbutton[4].setOnAction(event -> NumButtonPressed4());
numbutton[5].setOnAction(event -> NumButtonPressed5());
numbutton[6].setOnAction(event -> NumButtonPressed6());
numbutton[7].setOnAction(event -> NumButtonPressed7());
numbutton[8].setOnAction(event -> NumButtonPressed8());
numbutton[9].setOnAction(event -> NumButtonPressed9());
numbutton[10].setOnAction(event -> NumButtonPressed10());
numbutton[11].setOnAction(event -> NumButtonPressed11());
calcbutton[0].setOnAction(event -> CalcButtonPressed0());
calcbutton[1].setOnAction(event -> CalcButtonPressed1());
calcbutton[2].setOnAction(event -> CalcButtonPressed2());
calcbutton[3].setOnAction(event -> CalcButtonPressed3());
calcbutton[4].setOnAction(event -> CalcButtonPressed4());


//ミドルサイズのキーの生成
```

```
GridPane midPanel1 = new GridPane();
GridPane.setConstraints(clearbutton,    0, 0);
midPanel1.getChildren().addAll(clearbutton);

GridPane midPanel2 = new GridPane();
GridPane.setConstraints(numbutton[11],    0, 0);
GridPane.setConstraints(calcbutton[0],    1, 0);
midPanel2.getChildren().addAll(numbutton[11], calcbutton[0]);

GridPane midPanel3 = new GridPane();
GridPane.setConstraints(numbutton[7],    0, 0);
GridPane.setConstraints(numbutton[8],    1, 0);
midPanel3.getChildren().addAll(numbutton[7], numbutton[8]);

GridPane midPanel4 = new GridPane();
GridPane.setConstraints(numbutton[9],    0, 0);
GridPane.setConstraints(calcbutton[1],    1, 0);
midPanel4.getChildren().addAll(numbutton[9], calcbutton[1]);

GridPane midPanel5 = new GridPane();
GridPane.setConstraints(numbutton[4],    0, 0);
GridPane.setConstraints(numbutton[5],    1, 0);
midPanel5.getChildren().addAll(numbutton[4], numbutton[5]);

GridPane midPanel6 = new GridPane();
GridPane.setConstraints(numbutton[6],    0, 0);
GridPane.setConstraints(calcbutton[2],    1, 0);
midPanel6.getChildren().addAll(numbutton[6], calcbutton[2]);

GridPane midPanel7 = new GridPane();
GridPane.setConstraints(numbutton[1],    0, 0);
GridPane.setConstraints(numbutton[2],    1, 0);
midPanel7.getChildren().addAll(numbutton[1], numbutton[2]);

GridPane midPanel8 = new GridPane();
GridPane.setConstraints(numbutton[3],    0, 0);
GridPane.setConstraints(calcbutton[3],    1, 0);
midPanel8.getChildren().addAll(numbutton[3], calcbutton[3]);
```

```
        GridPane midPanel9 = new GridPane();
        GridPane.setConstraints(numbutton[0],  0, 0);
        midPanel9.getChildren().addAll(numbutton[0]);

        GridPane midPanel10 = new GridPane();
        GridPane.setConstraints(numbutton[10],  0, 0);
        GridPane.setConstraints(calcbutton[4],  1, 0);
        midPanel10.getChildren().addAll(numbutton[10], calcbutton[4]);

        //キーボードの配置の生成
        GridPane keyPanel = new GridPane();
        GridPane.setConstraints(midPanel1, 0, 0);
        GridPane.setConstraints(midPanel2, 1, 0);
        GridPane.setConstraints(midPanel3, 0, 1);
        GridPane.setConstraints(midPanel4, 1, 1);
        GridPane.setConstraints(midPanel5, 0, 2);
        GridPane.setConstraints(midPanel6, 1, 2);
        GridPane.setConstraints(midPanel7, 0, 3);
        GridPane.setConstraints(midPanel8, 1, 3);
        GridPane.setConstraints(midPanel9, 0, 4);
        GridPane.setConstraints(midPanel10, 1, 4);

        //numbutton を keyPanel に追加
        keyPanel.getChildren().addAll(midPanel1, midPanel2, midPanel3, midPanel4,
midPanel5, midPanel6, midPanel7, midPanel8, midPanel9, midPanel10);

        //ルートペインを生成
        GridPane root = new GridPane();
        GridPane.setConstraints(result, 0, 0);
        GridPane.setConstraints(keyPanel, 0, 1);

        //ルートペインに配置
        root.getChildren().addAll(result, keyPanel);

        //stage に scene を配置
        stage.setScene(new Scene(root));
        //表示
```

```
        stage.show();
    }

    double resultValue;//入力された文字列を数値にしたもの
    double afterresultValue;//入力された文字列を数値にしたもの
    double resultValue1;//＋－にする
    int currentOp;//直近に押された演算子を数値として記憶する。除算を 0，乗算を 1，減算を 2，加算を 3 とする。
    String ans;


    //クリアボタンのアクション定義
    public void ClearButtonPressed() {
        result.clear();//表示内容を削除
        resultValue = 0.0;
        afterresultValue = 0.0;
        currentOp = 0;
        result.appendText("");
    }
    //数字ボタンのアクション定義
    public void NumButtonPressed0() {
        result.appendText(numbutton[0].getText());
    }
    public void NumButtonPressed1() {
        result.appendText(numbutton[1].getText());
    }
    public void NumButtonPressed2() {
        result.appendText(numbutton[2].getText());
    }
    public void NumButtonPressed3() {
        result.appendText(numbutton[3].getText());
    }
    public void NumButtonPressed4() {
        result.appendText(numbutton[4].getText());
    }
    public void NumButtonPressed5() {
        result.appendText(numbutton[5].getText());
    }
```

```
public void NumButtonPressed6() {
        result.appendText(numbutton[6].getText());
}
public void NumButtonPressed7() {
        result.appendText(numbutton[7].getText());
}
public void NumButtonPressed8() {
        result.appendText(numbutton[8].getText());
}
public void NumButtonPressed9() {
        result.appendText(numbutton[9].getText());
}
public void NumButtonPressed10() {
        result.appendText(numbutton[10].getText());
}
//「＋/ー」が押された時に起動する
public void NumButtonPressed11() {
        resultValue1 = (Double.valueOf(result.getText()));//result の値を数値に変換
        if(resultValue1 <= 0) {//resultValue が負の時絶対値を返す
                resultValue1 = Math.abs(resultValue1);
                ans = String.valueOf(resultValue1);
                result.clear();
                result.appendText(ans);
        }else {//resultValue が正の時，負の値を返す
                resultValue1 = resultValue1 - resultValue1 * 2;
                ans = String.valueOf(resultValue1);
                result.clear();
                result.appendText(ans);
        }
}

public void CalcButtonPressed0() {
        currentOp = 0;//0 を currentOp に代入（除算）
        resultValue = (Double.valueOf(result.getText()));//result の値を数値に変換
        result.clear();//テキストフィールドをクリアする
}
public void CalcButtonPressed1() {
        currentOp = 1;//1 を currentOp に代入（乗算）
```

```
            resultValue = (Double.valueOf(result.getText()));//result の値を数値に変換
            result.clear();//テキストフィールドをクリアする
        }
        public void CalcButtonPressed2() {
            currentOp = 2;//2 を currentOp に代入（減算）
            resultValue = (Double.valueOf(result.getText()));//result の値を数値に変換
            result.clear();//テキストフィールドをクリアする
        }
        public void CalcButtonPressed3() {
            currentOp = 3;//を currentOp に代入（加算）
            resultValue = (Double.valueOf(result.getText()));//result の値を数値に変換
            result.clear();//テキストフィールドをクリアする
        }
        //=が押された時に起動する
        public void CalcButtonPressed4() {
            afterresultValue = (Double.valueOf(result.getText()));//result の値を数値に
変換
            if(currentOp == 0) {
                resultValue = resultValue / afterresultValue;
            }
            if(currentOp == 1) {
                resultValue = resultValue * afterresultValue;
            }
            if(currentOp == 2) {
                resultValue = resultValue - afterresultValue;
            }
            if(currentOp == 3) {
                resultValue = resultValue + afterresultValue;
            }

            ans = String.valueOf(resultValue);
            result.clear();
            result.appendText(ans);
        }

        public static void main(String[] args) {
            launch();
```

```
            }

    }
```