プログラム設計 後期中間試験

準備　プログラムを作る前に、以下の操作をしてファイルの準備をしておくこと。

1. 授業の配付資料を全てダウンロードする場合は、以下を実行する（既に実行済みの場合は不要）
   `$  mygitclone-pd`
2. GitHub から自分のリポジトリを clone しておく（既に実行済みの場合は不要）
   `$ mygitclone4d` 自分の GitHub ユーザ名
3. リポジトリのフォルダに移動して、設定用のスクリプトを実行する
   `$ cd ~/prog4d-` ユーザ名
   `$ ./myconf`
4. 今回の定期試験用のフォルダをコピーする
   `$ cp  -r  /usr/local/common/kogai/pd/test2mid  .`（←ここにピリオド）
   `$ cd  test2mid`　（コピーしたフォルダに移動する）
   `$ ls`　（フォルダ内のファイルを確認すると、以下のファイルがコピーされている）
   `AppMain.java  source.java`
5. astah ファイルやスケルトンコードなど、解答のファイルは全て**このフォルダ内に保存すること**
   提出する全ての astah ファイルの図内に、**自分の学科の出席番号と氏名**を「ノート」を使って書くこと
   提出する全てのプログラムファイルの先頭行に、**自分の学科の出席番号と氏名**をコメントとして書くこと

1　astah を使い、次に示すクラスの仕様（1-1）に従って、UML クラス図内に「累乗計算の結果を保持するクラス」を表す図を作成しなさい。

仕様（1-1）クラス Model（$base^0$〜$base^9$ を計算して結果を保持するクラス）

[属性]
- 可視性は private で、int 型の変数 base （累乗計算の底となる値）
- 可視性は private で、int[] 型の変数 value （$base^0$〜$base^9$ を保持する配列）

[操作]
- 可視性は public で、引数なしコンストラクタ
  （base を 2 にして、value は 10 個の要素の配列を参照して、累乗計算するメソッド calc を呼び出す）
- 可視性は public で、属性 base の setter, getter （メソッド setBase, getBase）
- 可視性は public で、属性 value の setter, getter （メソッド setValue, getValue）
  （value の型は int[] であることに注意）
- 可視性は public で、「$base^0$〜$base^9$ を計算して value[0]〜value[9] に格納する」メソッド calc
  `public void calc()`
- 可視性は public で、「base の値を 1 増やして、メソッド calc を呼び出す」メソッド plus
  `public void plus()`
- 可視性は public で、「base の値を 1 減やして、メソッド calc を呼び出す」メソッド minus
  `public void minus()`

2　MVC に基づいた設計によって、「$base^0$〜$base^9$ を結果を表示する」ソフトウェアを開発することを考える。astah を使って、次に示すクラスの仕様（2-1）〜（2-8）に従った Model, View, Controller のクラス図を作成しなさい。
ただし、可視性は属性が非公開（private）、操作が公開（public）とし、関連は多重度を必ず明記すること。

仕様（2-1）Model を表すクラス Model

- 前問で作成したクラス Model

仕様（2-2）View を表すクラス View

属性はなし

[操作]

・コンストラクタ（GUI 画面を作る）

・`public void update()`（GUI 画面の情報を更新する）

・`public void actionPerformed(ActionEvent e)`（GUI のボタンを押した時の処理をする）

仕様（2-3）Controller を表すクラス Controller

属性はなし

[操作]

・`public void plus()`（Model で定義されている追加処理メソッド plus を呼び出す）

・`public void minus()`（Model で定義されている追加処理メソッド minus を呼び出す）

仕様（2-4）インターフェース ActionListener

仕様（2-5）View と Model の相互に誘導可能な関連（関連, 誘導可能性, 多重度に加え、以下の操作も追加する）

[操作]

・View に、Model の setter/getter （setModel, getModel）

・Model に、View の setter/getter （setView, getView）

仕様（2-6）View と Controller の相互に誘導可能な関連（関連, 誘導可能性, 多重度に加え、以下の操作も追加する）

[操作]

・View に、Controller の setter/getter （setController, getController）

・Controller に、View の setter/getter （setView, getView）

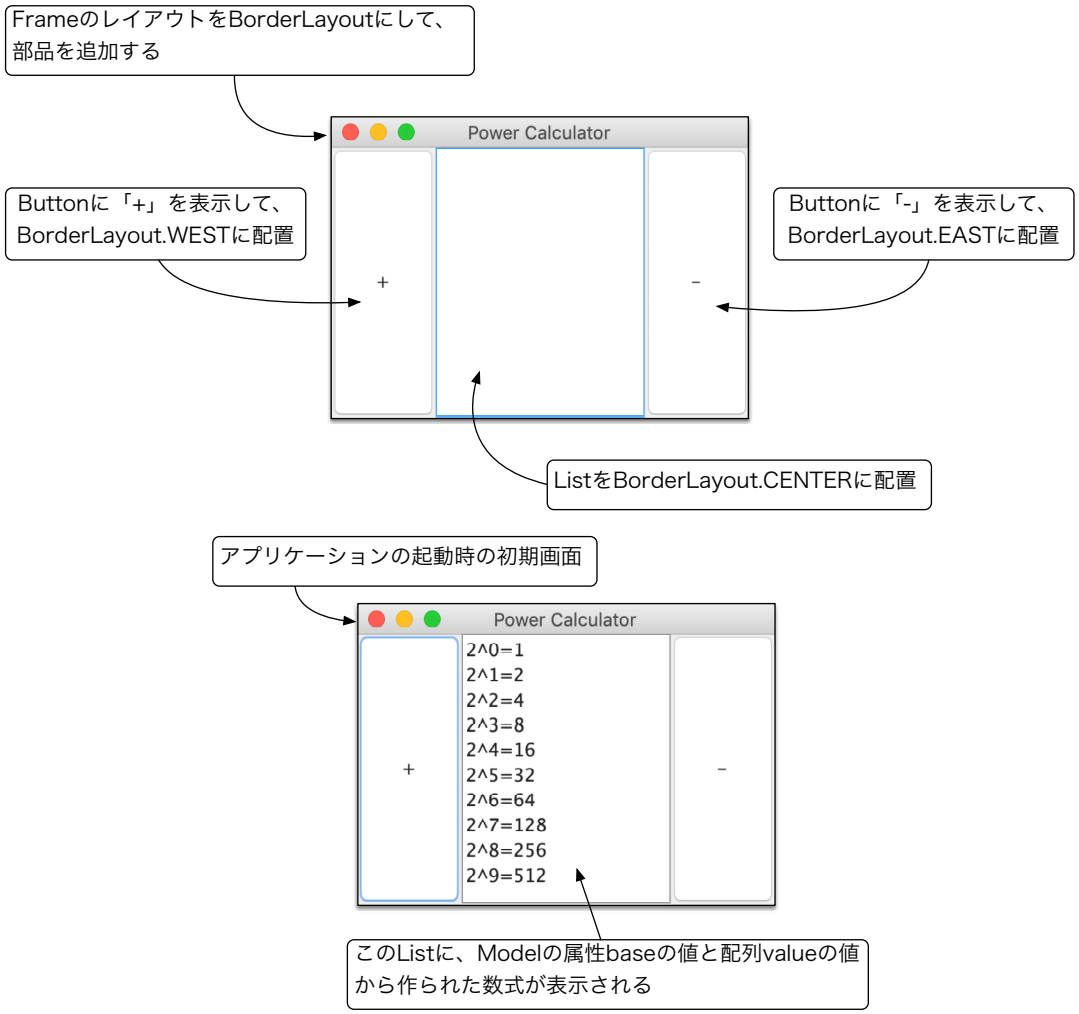
仕様（2-7）Controller から Model へ誘導可能な関連（関連, 誘導可能性, 多重度に加え、以下の操作も追加する）

[操作]

・Controller に、Model の setter/getter （setModel, getModel）

仕様（2-8）View から ActionListener へ実装（実現）を表す関連

3 astah を使って、前問で作成したクラス図から、スケルトンコードを生成し（ActionListener, ActionEvent のコード生成は不要）、次のような初期画面が表示されるように、Java コードを完成させなさい。**実装するメソッドの処理内容については、問 1, 問 2 に示した仕様を参考にして作ること。**（ただし、問 4 で完成させるメソッドについては、ここで実装しなくても初期画面は完成できる。）



クラス Model のコンストラクタは以下のような処理で実装する。

```
public Model() {
    //base の値を 2 にする
    //int 型の 10 要素の配列を作り value で参照する
    //Model 自身のメソッド calc を呼び出す
}
```

クラス Model のメソッド calc は以下のようになる。（ファイル「source.java」から入手可能）

```
public void calc() {
    for(int i=0; i<value.length; i++) {
        value[i] = (int)Math.pow(base, i);  //クラス Math のメソッド pow を使って、base の i 乗を計算している
    }
}
```

クラス View のメソッド update は以下のようになる。（ファイル「source.java」から入手可能）

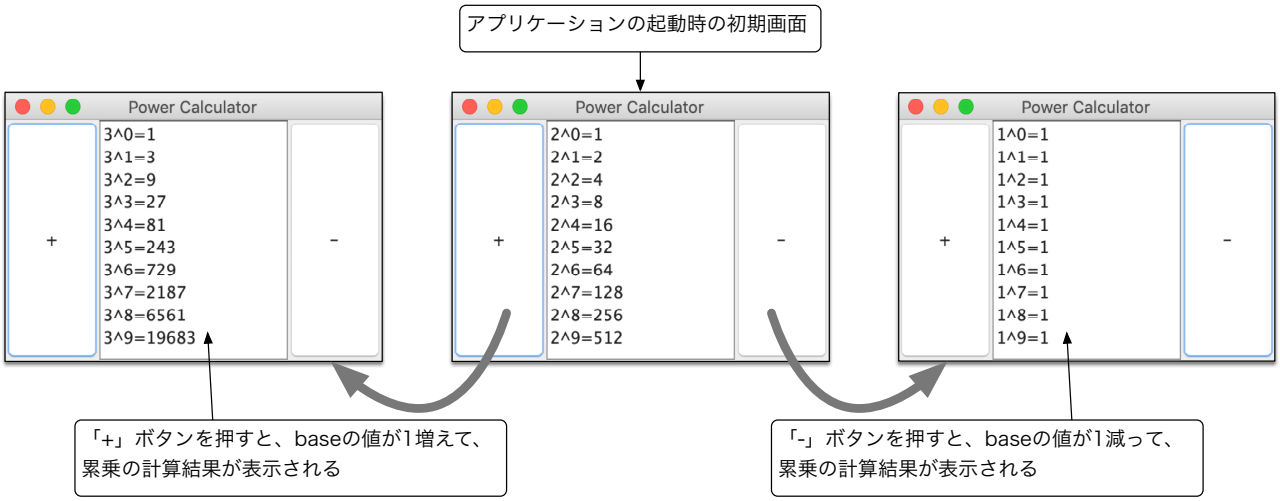
```
public void update() {
    //result は List 型のインスタンスを参照しているとする
    result.removeAll();
    int b = model.getBase();
    int v[] = model.getValue();
    for(int i=0; i<v.length; i++) {
```

```
        String si = String.valueOf(i);
        String sv = String.valueOf(v[i]);
        String sb = String.valueOf(b);
        result.add(sb+"^"+si+"="+sv);
    }
}
```

main の処理は以下のようになる。（ファイル「AppMain.java」に含まれている。）

```
class AppMain {
    public static void main(String[] args) {
        View v = new View();              //MVC のインスタンスを作る
        Model m = new Model();
        Controller c = new Controller();
        v.setController(c);               //View と Controller を相互に参照する
        c.setView(v);
        v.setModel(m);                    //View と Model を相互に参照する
        m.setView(v);
        c.setModel(m);                    //Controller から Model を参照する
        v.update();                       //モデルの初期値を表示する
    }
}
```

4 前問で作ったプログラムに対して、「base の値を増減させて累乗計算の結果を表示する」の機能を完成させなさい。完成したプログラムは、以下のような動作となる。



次のメソッドを実装する必要がある。

- クラス View のメソッド actionPerformed
  以下のように、Controller のメソッド plus と minus を呼び出す

```
public void actionPerformed(ActionEvent e) {
    //押したのが「+」ボタンだったら、Controller のメソッド plus を呼び出す
    //押したのが「-」ボタンだったら、Controller のメソッド minus を呼び出す
    //押したボタンを判別するのは「e.getSource()」との比較でできる（前期講義参照）
}
```

- クラス Controller のメソッド plus, minus
  以下のように、それぞれ Model のメソッド plus と minus を呼び出す

```
public void plus() {
    model.plus();
}
public void minus() {
    model.minus();
}
```

- クラス Model のメソッド plus, minus
  以下のように base の増減をして累乗計算をする

```
public void plus() {
  //base の値を 1 増やす
  //自身のメソッド calc を呼び出して累乗の計算をする
  //View のメソッド update を呼び出す
}
public void minus() {
  //base の値を 1 減らす
  //自身のメソッド calc を呼び出して累乗の計算をする
  //View のメソッド update を呼び出す
}
```

問題はここまで（各 25 点）

定期試験の実施について

試験中に使用できるもの

- 筆記用具（メモ用紙は必要な人に配布）
- 演習室のコンピュータ一台（一つの机に一人の配置で、座る場所はどこでもよい）

試験中に参照できるもの

- 自分のホームディレクトリ（ホームフォルダ）以下に保存されているファイル
  （定期試験では紙媒体のものは参照不可）

* **上記以外の情報を参照することは不正行為とする**
  （例：USB で接続された機器に保存されているファイルの参照、Web ブラウザやネットワークを介した情報の参照、自分の PC を使用する、など）
* 試験中（開始 5 分後〜開始 60 分後）は、演習室外へのネットワークアクセスは遮断される
* GitHub への提出のためのコマンドに限ってネットワーク利用が可能（それ以外は不正行為とする）

答案の提出

1. 提出する全ての astah ファイルの図内に、**自分の学科の出席番号と氏名**を「ノート」を使って書く
2. 提出する全てのプログラムファイルの先頭行に、**自分の学科の出席番号と氏名**をコメントとして書く
3. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
$ git commit -m "後期中間提出"
$ git push origin master
```

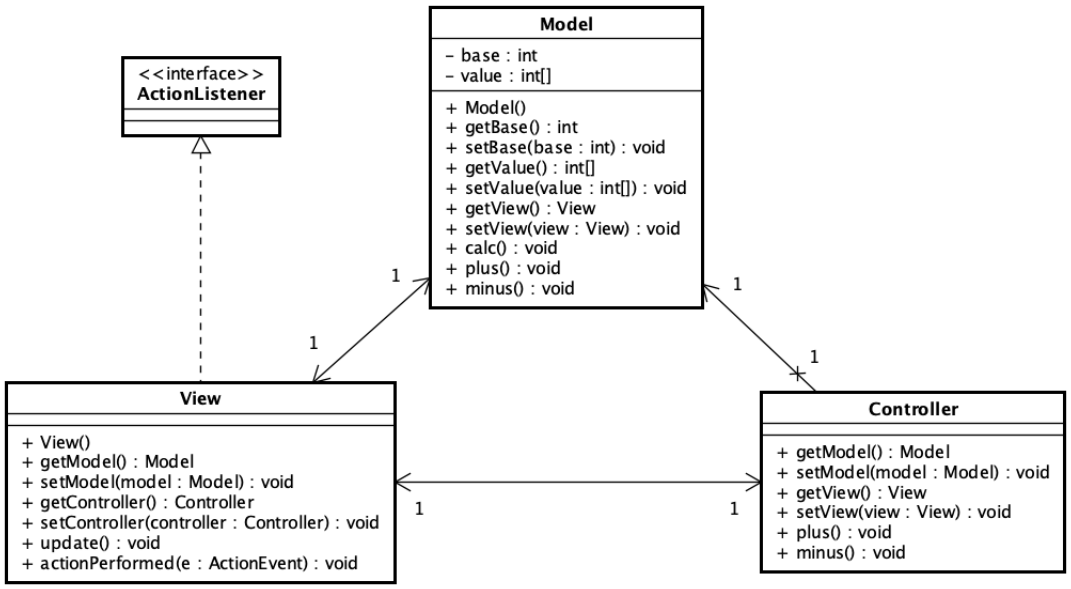
（push が成功すると「done」が含まれたメッセージが表示される）

4. 提出が完了しているかを確認したい人は声をかけて下さい。（その場で教員側の画面で確認します）

プログラム設計 後期中間試験模範解答（試験時間 90 分，平均 87.5 点）

採点について　不十分な箇所につき −2 点、エラーとなる箇所を −4 点を基本とする。1〜4 各 25 点

1 2



3 4　問題に示したコードは減点対象外とする

問 3 配点の内訳（この順で答案に左から点数を記載している）

- スケルトンコードを生成している（5 点）
- 各クラスの setter, getter（5 点）
- Model のコンストラクタ（5 点）
- View のコンストラクタ（5 点）
- View のメソッド update,
  Model のメソッド calc（5 点）

問 4 配点の内訳（この順で答案に左から点数を記載している）

- View のメソッド actionPerformed（10 点）
- Controller のメソッド plus, minus（5 点）
- Model のメソッド plus, minus（10 点）

```
//Model.java
class Model {
    private View view;
    private int base;
    private int[] value;
    public Model() {
        base = 2;
        value = new int[10];
        this.calc();
    }
    public int getBase() {
        return base;
    }
    public void setBase(int base) {
        this.base = base;
    }
    public int[] getValue() {
        return value;
    }
    public void setValue(int[] value) {
        this.value = value;
    }
    public View getView() {
        return view;
    }
    public void setView(View view) {
        this.view = view;
    }
    public void calc() {
        for(int i=0; i<value.length; i++) {
            value[i] = (int)Math.pow(base, i);
        }
    }
    public void plus() {
        base++;
        this.calc();
        view.update();
    }
    public void minus() {
        base--;
        this.calc();
        view.update();
    }
}
```

```
//View.java
import java.awt.*;
import java.awt.event.*;

class View  implements ActionListener {
    private Model model;
    private Controller controller;
    private Frame fr;
    private Button pl, mi;
    private List result;
    public View() {
        fr = new Frame("Power Calculator");
        pl = new Button("+");
        mi = new Button("-");
        result = new List();

        fr.setLayout(new BorderLayout());
        fr.add(pl, BorderLayout.WEST);
        fr.add(result, BorderLayout.CENTER);
        fr.add(mi, BorderLayout.EAST);

        pl.addActionListener(this);
        mi.addActionListener(this);

        fr.setSize(300, 300);
        fr.setVisible(true);
    }
    public Model getModel() {
        return model;
    }
    public void setModel(Model model) {
        this.model = model;
    }
    public Controller getController() {
        return controller;
    }
    public void setController(Controller controller) {
        this.controller = controller;
    }
    public void update() {
        result.removeAll();
        int b = model.getBase();
        int v[] = model.getValue();
        for(int i=0; i<v.length; i++) {
            String si = String.valueOf(i);
            String sv = String.valueOf(v[i]);
            String sb = String.valueOf(b);
            result.add(sb+"^"+si+"="+sv);
        }
    }
    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==pl) {
            controller.plus();
        }
        if(e.getSource()==mi) {
            controller.minus();
        }
    }
    public static void main(String[] args) {
        new View();
    }
}

//Controller.java
class Controller {
    private Model model;
    private View view;
    public Model getModel() {
        return model;
    }
    public void setModel(Model model) {
        this.model = model;
    }
    public View getView() {
        return view;
    }
    public void setView(View view) {
        this.view = view;
    }
    public void plus() {
        model.plus();
    }
    public void minus() {
        model.minus();
    }
}


//AppMain.java
class AppMain {
    public static void main(String[] args) {
        //MVC のインスタンスを作る
        View v = new View();
        Model m = new Model();
        Controller c = new Controller();
        //View と Controller を相互に参照する
        v.setController(c);
        c.setView(v);
        //View と Model を相互に参照する
        v.setModel(m);
        m.setView(v);
        //Controller から Model を参照する
        c.setModel(m);
        //モデルの初期値を表示する
        v.update();
    }
}
```