プログラム設計 後期中間試験

全てのプログラムファイルの先頭行に、コメントとして自分の番号と名前を書くこと。

準備 **試験で利用可能なコードが事前に用意されている。端末内で、以下のコマンドを実行してコピーしておくこと。**

```
$ cp /usr/local/common/kogai/201811pd.java . （←ここにピリオド）
```

記名について

- 提出する全ての asta 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
- 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

1 astah を使い、次に示すクラスの仕様（1-1）〜（1-3）に従って、UML クラス図内に「アクティビティ記録（歩数と前日比）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 ActModel（今回と前回の歩数を保持し、前日比を計算するクラス）

[属性]

・可視性は private で、int 型の変数 previous（前回の歩数）
・可視性は private で、int 型の変数 current（今回の歩数）
・可視性は private で、int 型の変数 difference（前回と今回の差分）

[操作]

・可視性は public で、引数なしコンストラクタ（previous を 3000, current を 5000, difference を 2000 にする）
・可視性は public で、属性 previous の setter, getter （メソッド setPrevious, getPrevious）
・可視性は public で、属性 current の setter, getter （メソッド setCurrent, getCurrent）
・可視性は public で、属性 difference の setter, getter （メソッド setDifference, getDifference）
・可視性は public で、「今回の歩数を登録して、前日比を計算する」を表すメソッド add（引数 x は今回の歩数）
`public void add(int x)`

仕様（1-2）クラス ActWeekly（ActModel を集約するクラス）

属性, 操作はなし

仕様（1-3）クラス間の関係

・クラス ActWeekly は、ActModel を集約していて、ActWeekly 側の多重度は 1、ActModel 側の多重度は 0 から 7 までである
・ActWeekly と ActModel 間の集約は、ActWeekly から ActModel へは誘導可能だが、その逆は誘導不可である

2 MVC に基づいた設計によって、「歩数を入力して、前回と今回の歩数と前日比を表示する」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 ActModel

- 前問で作成したクラス ActModel
  （クラス ActWeekly は、以降の問で使用しないが、前問の解答としてクラス図に**そのまま残しておく**）

仕様（2-2）View を表すクラス ActView

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 ActController

属性はなし

[操作]

・`public void add(int x)`（Model で定義されている追加処理メソッド add 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]

・ActView に、ActModel の setter/getter （setActModel, getActModel）

・ActModel に、ActView の setter/getter （setActView, getActView）

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]

・ActView に、ActController の setter/getter （setActController, getActController）

・ActController に、ActView の setter/getter （setActView, getActView）

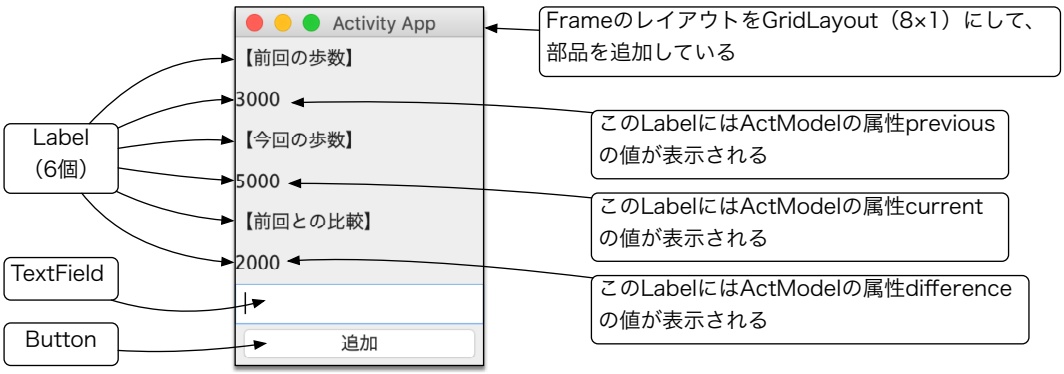
仕様（2-7）Controller から Model へ誘導可能な関連（関連, 誘導可能性, 多重度に加え、以下の操作も追加する）

[操作]

・ActController に、ActModel の setter/getter （setActModel, getActModel）

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, ActWeekly 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 3 つの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。）



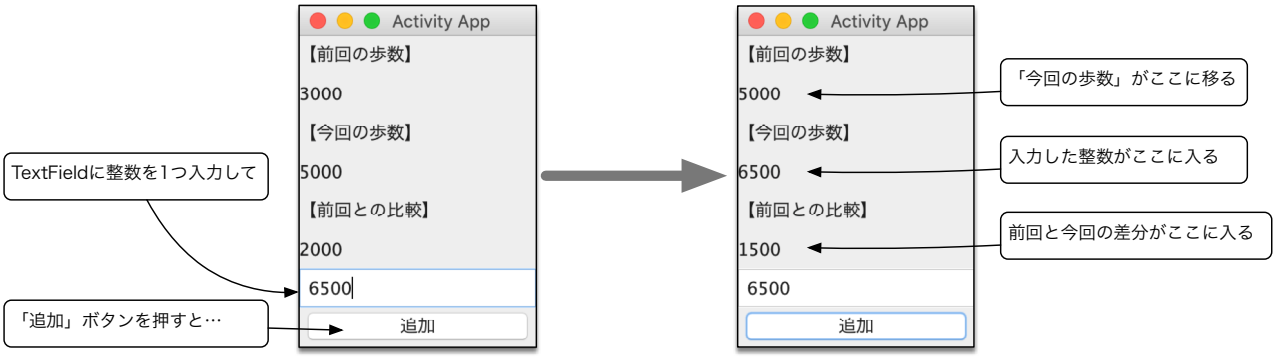
クラス ActView のメソッド update は以下のようになる。（コピーしたファイルから入手可能）

```
public void update() {
    //actModel は ActModel 型、l2, l4, l6 は Label 型のインスタンスを参照しているとする
    int p = actModel.getPrevious();
    int c = actModel.getCurrent();
    int d = actModel.getDifference();
    l2.setText(String.valueOf(p));
    l4.setText(String.valueOf(c));
    l6.setText(String.valueOf(d));
}
```

main の処理は以下のようになる。（コピーしたファイルから入手可能）

```
class ActMain {
    public static void main(String[] args) {
        ActView v = new ActView();              //MVC のインスタンスを作る
        ActModel m = new ActModel();
        ActController c = new ActController();
        v.setActController(c);                  //View と Controller を相互に参照する
        c.setActView(v);
        v.setActModel(m);                       //View と Model を相互に参照する
        m.setActView(v);
        c.setActModel(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、「歩数の登録と前日比の表示」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 ActView のメソッド actionPerformed
  以下のように、ActController のメソッド add を呼び出す（コピーしたファイルから入手可能）

```
public void actionPerformed(ActionEvent e) {
    //tf1 は TextField 型の変数であり、歩数を表す整数が文字列として入力されているとする
    String s1 = tf1.getText();
    int n1 = Integer.parseInt(s1);
    actController.add(n1);
}
```

- クラス ActController のメソッド add
  以下のように、ActModel のメソッド add を呼び出す（コピーしたファイルから入手可能）

```
public void add(int x) {
    actModel.add(x);
}
```

- クラス ActModel のメソッド add
  以下のような計算処理をする
  1. previous に current を代入する
  2. current に x を代入する
  3. difference に、current から previous の差を代入する
  4. 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

* **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など）

* 試験中は、演習室外へのネットワークアクセスは遮断される

答案の提出

- 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
- 提出する全ての asta 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
- 保存したファイルは次のように「report」コマンドで提出する
  （ちゃんと提出できた場合は、「Succeed.」と画面に表示される）

```
$ ~kogai/report  pd2mid  「プログラムファイル」
```

- 各問で作成したファイルを以下のように、report コマンドを分けて提出する

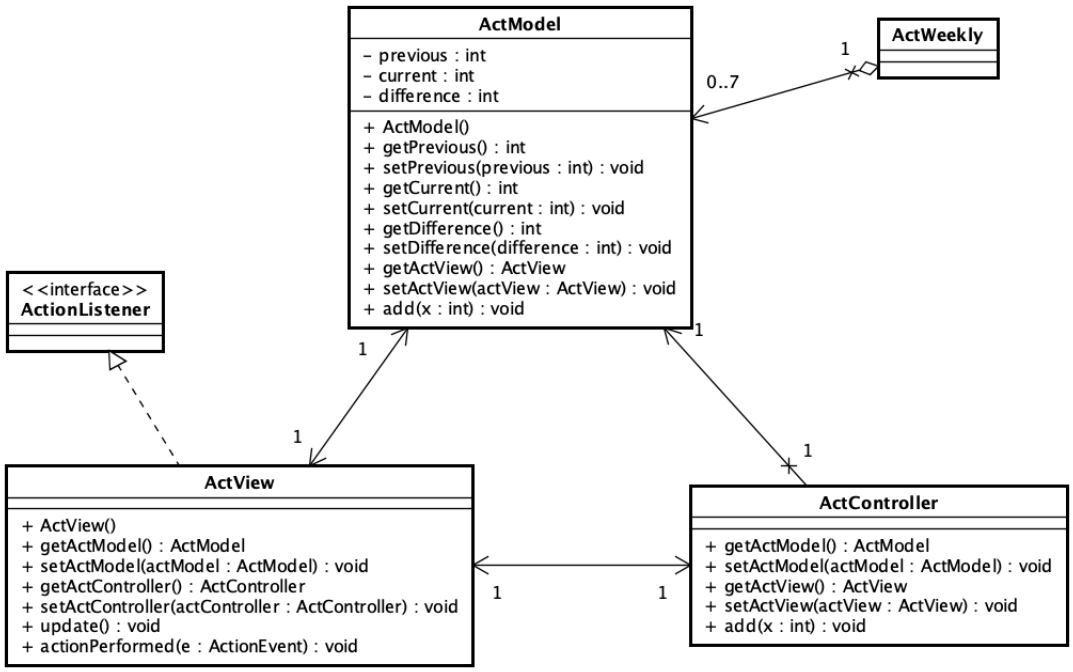
```
$ ~kogai/report  pd2mid  （問 1 と問 2 で作ったクラス図）.asta
$ ~kogai/report  pd2mid  ActModel.java
$ ~kogai/report  pd2mid  ActView.java
$ ~kogai/report  pd2mid  ActController.java
```

- 同じ問題に対して、複数の提出ファイルが存在した場合は、更新日時が新しい方を提出ファイルとする

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 80.9 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（メソッド update, actionPerformed, add, main）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（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 add（5 点）
- Model のメソッド add（10 点）

```
//ActModel.java
public class ActModel {
    private ActView actView;
    private int previous;
    private int current;
    private int difference;
    public ActModel() {
        previous = 3000;
        current = 5000;
        difference = 2000;
    }
    public int getPrevious() {
        return previous;
    }
    public void setPrevious(int previous) {
        this.previous = previous;
    }
    public int getCurrent() {
        return current;
    }
    public void setCurrent(int current) {
        this.current = current;
    }
    public int getDifference() {
        return difference;
    }
    public void setDifference(int difference) {
        this.difference = difference;
    }
    public ActView getActView() {
        return actView;
    }
    public void setActView(ActView actView) {
        this.actView = actView;
    }
    public void add(int x) {
        previous = current;
        current = x;
        difference = current-previous;
        actView.update();
    }
```

```
}

//ActView.java
import java.awt.*;
import java.awt.event.*;

public class ActView  implements ActionListener {
    private ActModel actModel;
    private ActController actController;
    private Frame f1;
    private TextField tf1;
    private Label l1, l2, l3, l4, l5, l6;
    private Button add;
    public ActView() {
        f1 = new Frame("Activity App");
        l1 = new Label("【前回の歩数】");
        l2 = new Label();
        l3 = new Label("【今回の歩数】");
        l4 = new Label();
        l5 = new Label("【前回との比較】");
        l6 = new Label();
        tf1 = new TextField();
        add = new Button("追加");

        f1.setLayout(new GridLayout(8, 1));
        f1.add(l1);
        f1.add(l2);
        f1.add(l3);
        f1.add(l4);
        f1.add(l5);
        f1.add(l6);
        f1.add(tf1);
        f1.add(add);
        f1.pack();
        f1.setVisible(true);

        add.addActionListener(this);
    }
    public ActModel getActModel() {
        return actModel;
    }
    public void setActModel(ActModel actModel) {
        this.actModel = actModel;
    }
    public ActController getActController() {
        return actController;
    }
    public void setActController(ActController
                                 actController) {
        this.actController = actController;
    }
    public void update() {
        int p = actModel.getPrevious();
        int c = actModel.getCurrent();
        int d = actModel.getDifference();
        l2.setText(String.valueOf(p));
        l4.setText(String.valueOf(c));
        l6.setText(String.valueOf(d));
    }
    public void actionPerformed(ActionEvent e) {
        String s1 = tf1.getText();
        int n1 = Integer.parseInt(s1);
        actController.add(n1);
    }
    public static void main(String[] args) {
        new ActView();
    }
}

//ActController.java
public class ActController {
    private ActModel actModel;
    private ActView actView;
    public ActModel getActModel() {
        return actModel;
    }
    public void setActModel(ActModel actModel) {
        this.actModel = actModel;
    }
    public ActView getActView() {
        return actView;
    }
    public void setActView(ActView actView) {
        this.actView = actView;
    }
    public void add(int x) {
        actModel.add(x);
    }
}

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