

# プログラム設計

<http://bit.ly/design4d>

## MVCによるクラス設計（1）

後期 第5週

2019/10/28

# まず前回の補足

クラス間の集約関係を配列を使って  
実装する際の**3つのポイント**を説明します。

# クラス型変数の配列を作る

```
public class TodoList {  
    private Todo[] list;  
    private int count;  
    public TodoList() {  
        todo = new Todo[3];  
        count = 0;  
    }  
    //途中省略  
}
```

① 配列を参照する変数を宣言する

② 配列の要素を3つ作成する  
つまり、インスタンスを参照すること  
ができるクラス型の変数が3つ作られる



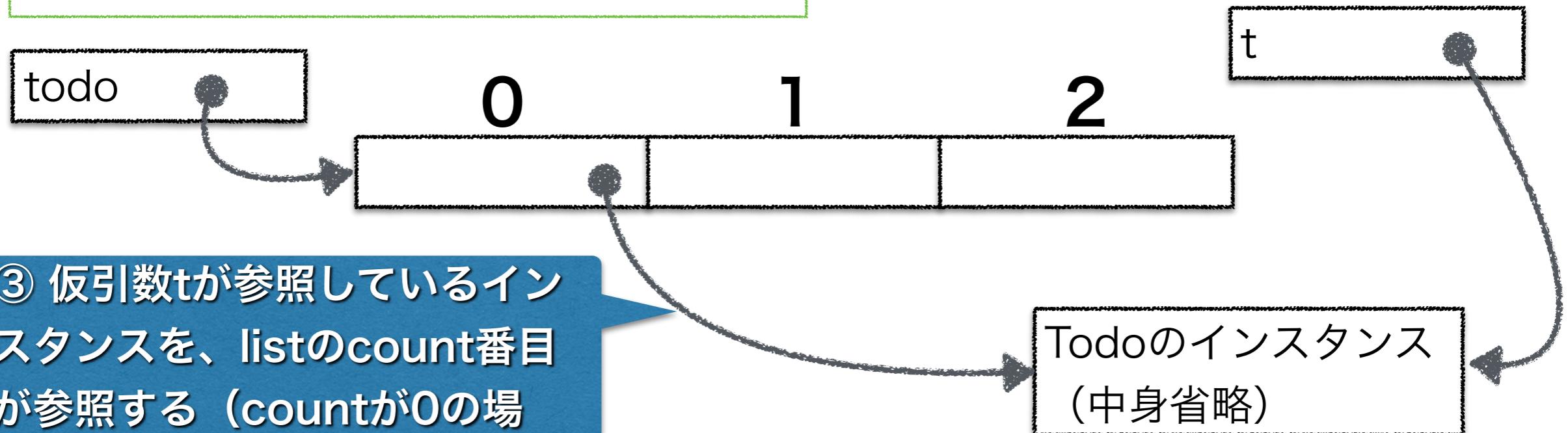
① 配列を参照する変数を宣言する

② 配列の要素を3つ作成して、`todo`がこの配列を参照する

# 各要素からインスタンスを参照する

```
public class TodoList {  
    //省略  
    public void addTodo(Todo t) {  
        if(count>=list.length) {  
            count = 0;  
        }  
        list[count] = t;  
        count++;  
    }  
}
```

③ 仮引数tが参照しているインスタンスを、listのcount番目が参照する



# 今回と次回の内容

前回作成したTodo管理の設計に対して、  
さらにクラスを追加してアプリケーション  
開発を進めていきます。

# MVCとは

ソフトウェア（特にGUIやWebブラウザが伴うアプリケーション）を開発する際、クラスを3つのタイプに分けて設計します。

## ▶ Model

ソフトウェアで管理されるデータ持つクラス

## ▶ View

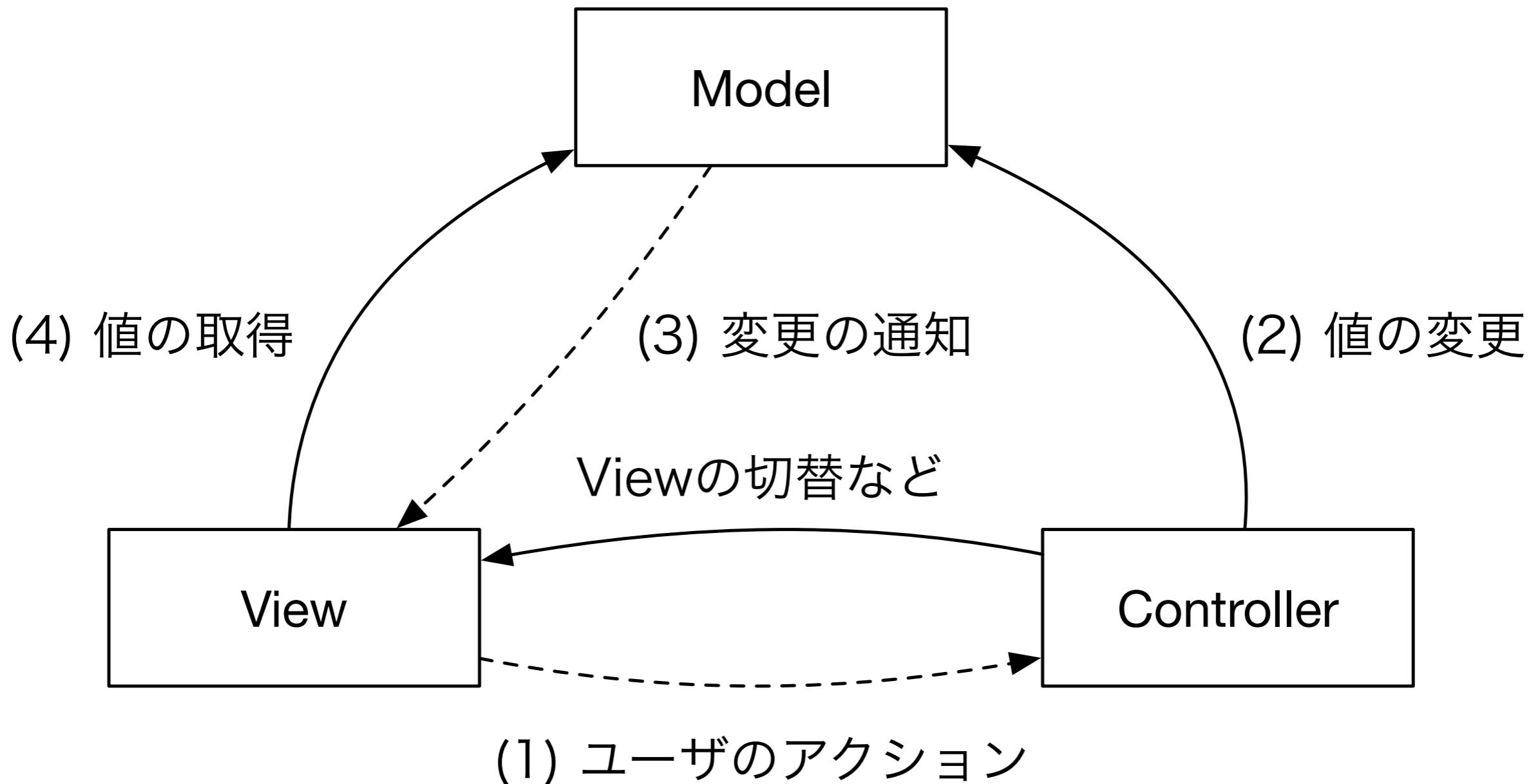
利用者のユーザインターフェースとなるクラス

## ▶ Controller

利用者からの要求に応じてソフトウェアの処理を制御するクラス

# MVCの関係

MVCの一般的な動きの流れは(1)~(4)のようになります。



# 今回は

前回までに作成したTodo管理のクラスを  
利用して、MVCに基づいたアプリケー  
ションのModelとViewの部分の開発を進  
めていきます。

# 【課題の準備】

演習室で作業する前に、以下のコマンドを入れるだけで準備が完了する

```
$ mygitclone4d 「自分のGitHubユーザ名」  
$ cd prog4d-(ユーザ名)  
$ ./myconf
```

※本体をシャットダウンするまでは、  
上記「mygitclone」と「myconf」の設定は有効です

# 【課題の準備】

以下の流れで、課題のプログラムを作るためのフォルダを準備しましょう。

1. 端末を起動して、以下のコマンドを実行して後期第5週のフォルダを作る

```
$ cd prog4d-(ユーザ名)          ←既に移動しているなら不要)
$ mkdir week205
$ cd week205
```

※課題で作るファイル名は各自で決めて構いません。

# ① Modelを作る

前回までに作成したクラスDate, Todo, TodoListは、Modelに相当するクラスとなります。

未完成の人は、前回の解答ファイル  
「2\_03\_todo\_ans.asta」を使えます。

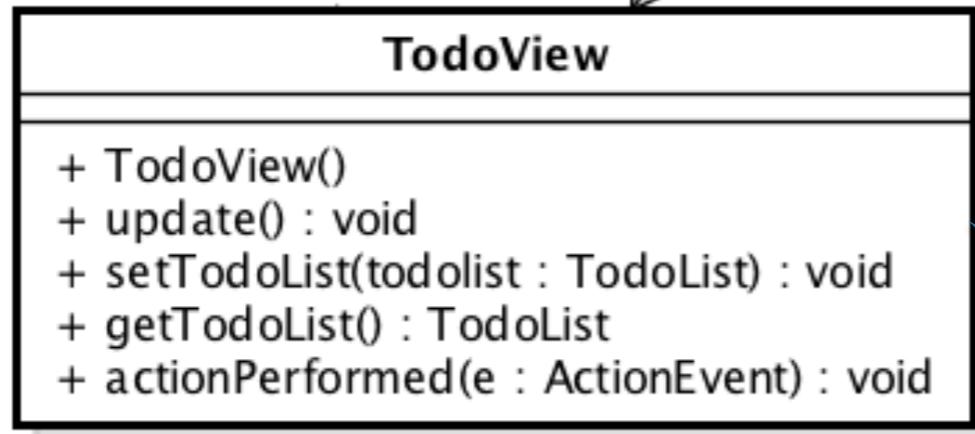
# ② Viewを作る

Viewクラス（TodoView）をクラス図に追加します。

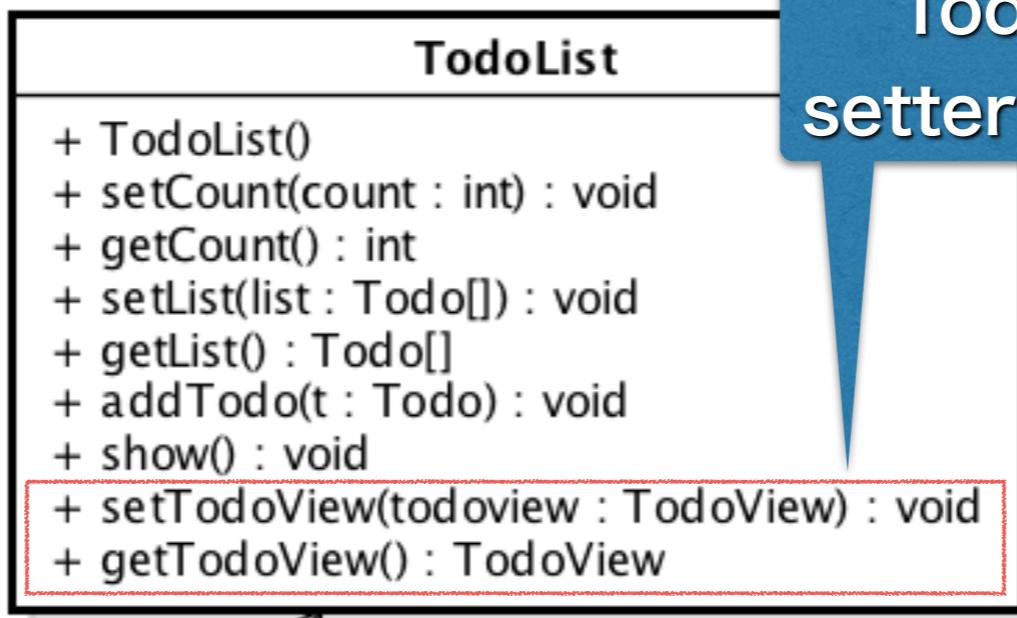
インターフェイスを追加



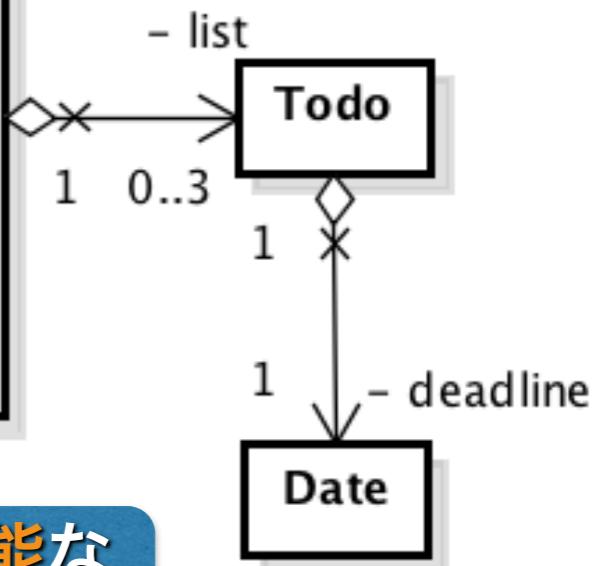
「実現」を追加



TodoViewへの  
setter/getterを追加



相互に誘導可能な  
「関連」を追加



TodoViewクラスを追加

### ③ スケルトンコードを生成する

スケルトンコードの作成で、前回作成したJavaコードを上書きしてしまわないよう注意してください。（前回完成させてい  
る人は、TodoViewのみのスケルトンコー  
ドを生成した方が良いです。）

## ④ クラス間の関係のsetter/getterを実装する

クラスTodoListとTodoViewが**相互に参照できるよう**にするために、それぞれクラスのsetter/getterを実装します。

### ▶ クラスTodoViewに実装

setTodoList(), getList()

### ▶ クラスTodoListに実装

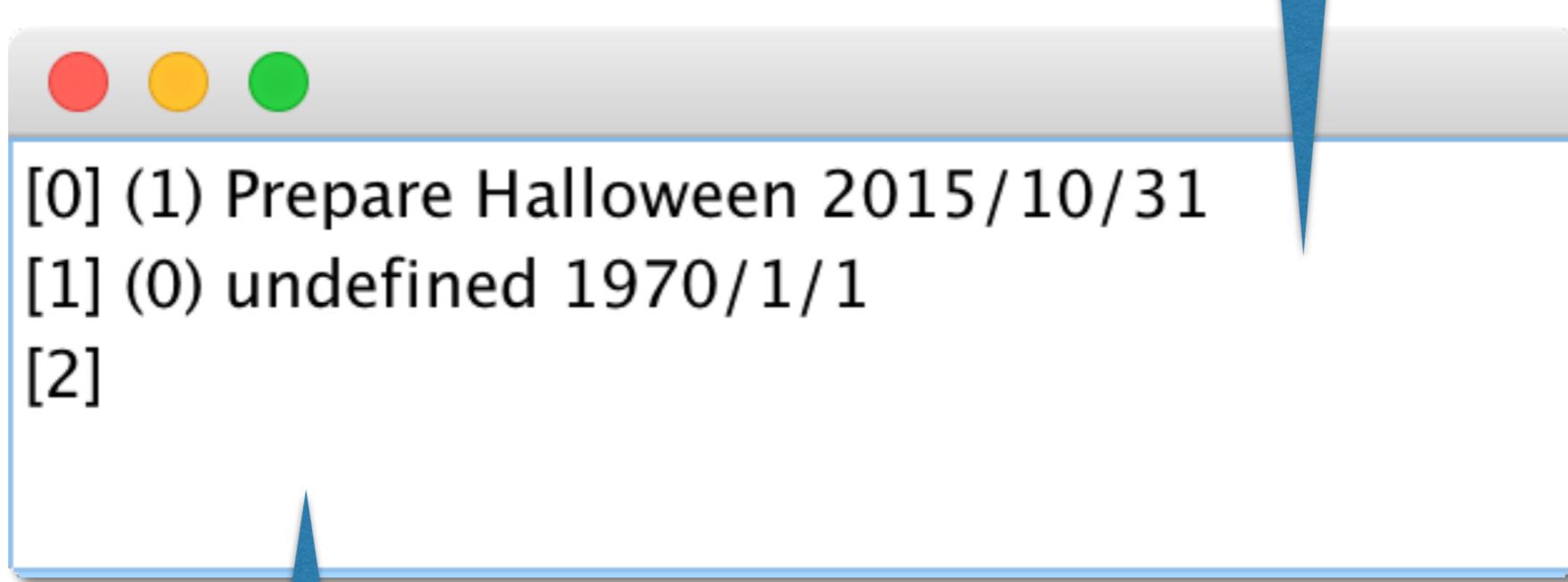
setTodoView(), getTodoView()

(前回完成させている人は、スケルトンコードを生成せずに、手動で追加・実装した方が良いです。)

## ⑤ Viewのユーザインターフェースを作る

クラスTodoViewのコンストラクタに、次のようなGUI画面を作る処理を追加します。

Frameを作り、レイアウトをBorderLayoutにして、CENTERにListを追加している



AWTのクラスListを使って、リスト形式でTodoを表示する  
(リストの中に表示する処理は、Viewのメソッドupdateで実装する)

# ⑥ Viewのメソッドを実装する

MVCモデルの「(4) 値の取得」に相当するメソッド  
updateを実装します。

(2\_05\_update.javaから入手可能)

```
public void update() {  
    l1.removeAll();  
    Todo[] l = todolist.getList();  
    for(int i=0; i<l.length; i++) {  
        String str = "";  
        str += "[" + String.valueOf(i) + "] ";  
        if(l[i] != null) {  
            str += "(" + l[i].getPriority() + ") ";  
            str += l[i].getName() + " ";  
            Date d = l[i].getDate();  
            str += d.getYear() + "/" + d.getMonth() + "/" + d.getDay();  
        }  
        l1.add(str);  
    }  
}
```

リストの内容を全て消す (l1は、AWTのList型の変数)

取得した配列の分だけ追加処理を繰り返す

strにTodoの情報の文字列を連結していく

リストにTodo情報を連結した文字列を追加する

# ⑦ Viewへの通知を追加する

MVCモデルの「(3) 変更の通知」に相当する処理を実装します。(クラスTodoListのメソッドaddTodoに以下を追加)

```
class TodoList {  
    // (途中省略)  
    public void addTodo(Todo t) {  
        // (途中省略)  
        todoview.update();  
    }  
}
```

メソッド内の一  
番最後にupdateの呼び出しを追加

# ⑧ プログラムを実行する

mainを持ったファイル2\_05\_TodoMain2.javaも含め、これまで作った全てのファイルをコンパイルして、ViewとModelのみでの動作を確認してみます。

(現時点ではControllerが実装されていませんので、mainの中に直接「Todo追加」処理を入れます。)

# 【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A  
$ git commit -m "課題5-1提出"  
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog4d-2019/prog4d-\(ユーザ名\)](https://github.com/nit-ibaraki-prog4d-2019/prog4d-(ユーザ名))

# 次回の内容

今回設計したクラス図に、Controllerを追加してアプリケーションを完成させます。