

プログラム設計

<http://bit.ly/design4d>

GUI (3)

前期 第13週

2019/7/17

今回の内容

- ▶ ポップアップメニュー
- ▶ レイアウトマネージャ
- ▶ WindowListenerインターフェース
- ▶ アダプタクラス
- ▶ イベントに関する命名規則
- ▶ 無名クラス

【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**前期第13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 week13
$ cd week13
```

※課題で作るファイル名は各自で決めて構いません。

今回の内容

- ▶ ポップアップメニュー
- ▶ レイアウトマネージャ
- ▶ WindowListenerインターフェース
- ▶ アダプタクラス
- ▶ イベントに関する命名規則
- ▶ 無名クラス

「1_13_MyChoice.java」を実行してみましょう

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

class MyFrame {
    private Frame f1;
    private Choice ch1;
    public MyFrame() {
        f1 = new Frame("フレーム");
        ch1 = new Choice();
        ch1.add("項目1");
        ch1.add("項目2");
        f1.add(ch1, BorderLayout.CENTER);
        f1.setSize(200, 100);
        f1.setVisible(true);
    }
    public static void main(String[] args) {
        MyFrame obj = new MyFrame();
    }
}
```

ポップアップメニューの
インスタンスを作る

ch1に対するaddメソッドで、
メニュー項目を追加する

Choiceの主なメソッド

▶ String getSelectedItem()

… 現在選択されている項目の文字列を取得する

【例】

```
String str = ch1.getSelectedItem();
//ch1で選択されている項目の文字列を取得して
//strに代入する
```

▶ int getSelectedIndex()

… 現在選択されている項目のインデックスを取得する

(何も選択されていない場合は -1 を返す)

【例】

```
int i = ch1.getSelectedIndex();
//ch1で選択されている項目のインデックスを取得して
//iに代入する
```

今回の内容

- ▶ ポップアップメニュー
- ▶ レイアウトマネージャ
- ▶ WindowListenerインターフェース
- ▶ アダプタクラス
- ▶ イベントに関する命名規則
- ▶ 無名クラス

「1_13_MyLayout.java」を実行してみましょう

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

class MyLayout {
    private Frame f1;
    public MyLayout() {
        f1 = new Frame("MyLayout");
        f1.setLayout(new FlowLayout());
        f1.add(new Button("Sunday")));
        f1.add(new Button("Monday")));
        f1.add(new Button("Tuesday")));
        f1.add(new Button("Wednesday")));
        f1.add(new Button("Thursday")));
        f1.add(new Button("Friday")));
        f1.add(new Button("Saturday")));
        f1.setSize(900, 300);
        f1.setVisible(true);
    }
    public static void main(String[] args) {
        MyLayout obj = new MyLayout();
    }
}
```

FrameのメソッドsetLayoutで、
そのフレームのレイアウトを設定する

FlowLayoutは、
部品を左から右に1
つずつ追加する

実行して表示されたフ
レームの大きさを変える
と、中の部品の配置が
自動的に調整される事
を確認しましょう。

赤線部分を変更して実行してみましょう

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

class MyLayout {
    private Frame f1;
    public MyLayout() {
        f1 = new Frame("MyLayout");
        f1.setLayout(new GridLayout(2, 4));
        f1.add(new Button("Sunday"));
        f1.add(new Button("Monday"));
        f1.add(new Button("Tuesday"));
        f1.add(new Button("Wednesday"));
        f1.add(new Button("Thursday"));
        f1.add(new Button("Friday"));
        f1.add(new Button("Saturday"));
        f1.setSize(900, 300);
        f1.setVisible(true);
    }
    public static void main(String[] args) {
        MyLayout obj = new MyLayout();
    }
}
```

GridLayoutは、フレームを格子に分割して部品を左から右, 上から下に1つずつ追加する (2行4列に分割している)

実行して表示されたフレームの大きさを変えると、中の部品の配置が自動的に調整される事を確認しましょう。

今回の内容

- ▶ ポップアップメニュー
- ▶ レイアウトマネージャ
- ▶ **WindowListener**インターフェース
- ▶ アダプタクラス
- ▶ イベントに関する命名規則
- ▶ 無名クラス

WindowListenerインターフェース

APIドキュメントから、インターフェース

「`java.awt.event.WindowListener`」を検索して調べてみると、次のような**抽象メソッド**が定義されている

<https://docs.oracle.com/javase/jp/11/docs/api/index.html>

- ▶ `windowActivated` … ウィンドウがアクティブになると呼び出される
- ▶ `windowClosed` … ウィンドウが閉じられたら呼び出される
- ▶ `windowClosing` … ウィンドウが閉じるときに呼び出される
- ▶ `windowDeactivated` … ウィンドウがアクティブでなくなると呼び出される
- ▶ `windowDeiconified` … ウィンドウが最小化から通常に戻ると呼び出される
- ▶ `windowIconified` … ウィンドウが最小化されると呼び出される
- ▶ `windowOpened` … ウィンドウが可視になったときに呼び出される

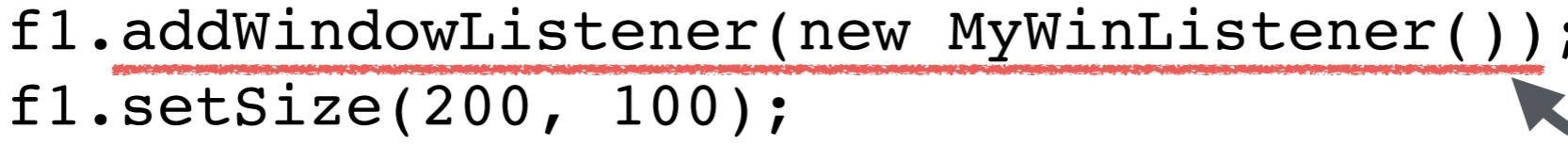
「1_13_MyWindow.java」を実行してみましょう

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

class MyWindow {
    private Frame f1;
    public MyWindow() {
        f1 = new Frame("フレーム");

        f1.addWindowListener(new MyWinListener());
        f1.setSize(200, 100);
        f1.setVisible(true);
    }
    public static void main(String[] args) {
        MyWindow obj = new MyWindow();
    }
}
```

フレームf1にWindowリスナの登録をする



※次のスライドに続く

「1_13_MyWindow.java」を実行してみましょう

インターフェースWindowListenerを実装すると、このインターフェースに含まれるメソッドを全て定義する必要がある

```
class MyWinListener implements WindowListener {  
    public void windowClosing(WindowEvent e) {  
        System.out.println("終了します。");  
        System.exit(0);  
    }  
    public void windowClosed(WindowEvent e) {}  
    public void windowActivated(WindowEvent e) {}  
    public void windowDeactivated(WindowEvent e) {}  
    public void windowDeiconified(WindowEvent e) {}  
    public void windowIconified(WindowEvent e) {}  
    public void windowOpened(WindowEvent e) {}  
}
```

フレームを閉じる時の処理を定義している

その他のメソッドは使わないので、処理を空にして定義している

実行して表示されたフレームを閉じると、アプリケーションが終了される事を確認しましょう。

今回の内容

- ▶ ポップアップメニュー
- ▶ レイアウトマネージャ
- ▶ WindowListenerインターフェース
- ▶ アダプタクラス
- ▶ イベントに関する命名規則
- ▶ 無名クラス

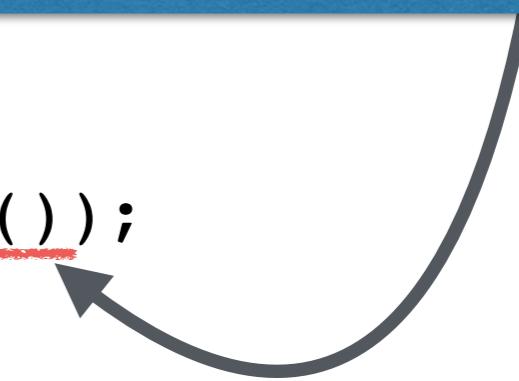
「1_13_MyWindow.java」の赤線部分を変更しましょう

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

class MyWindow {
    private Frame f1;
    public MyWindow() {
        f1 = new Frame("フレーム");

        f1.addWindowListener(new MyWinAdapter());
        f1.setSize(200, 100);
        f1.setVisible(true);
    }
    public static void main(String[] args) {
        MyWindow obj = new MyWindow();
    }
}
```

フレームf1のリスナに、アダプタクラス
を継承して作ったクラスを使う



※次のスライドに続く

「1_13_MyWindow.java」に次のクラスを追加しましょう (クラスMyWinListenerは不要になります)

クラスWindowAdapterは、インターフェースWindowListenerを実装し、空のメソッドを定義しただけのクラス（アダプタクラスと呼ぶ）

```
class MyWinAdapter extends WindowAdapter {  
    public void windowClosing(WindowEvent e) {  
        System.out.println("終了します。 (アダプタクラス) ");  
        System.exit(0);  
    }  
}
```

アダプタクラスを使った処理であることが分かるように、出力している

アダプタクラスを継承すると、既に全てのメソッドの処理が空で定義されているため、自分が使いたいメソッドだけをオーバーライドするだけによくなる

実行して表示されたフレームを閉じると、アプリケーションが終了される事を確認しましょう。

今回の内容

- ▶ ポップアップメニュー
- ▶ レイアウトマネージャ
- ▶ WindowListenerインターフェース
- ▶ アダプタクラス
- ▶ イベントに関する命名規則
- ▶ 無名クラス

イベントに関する命名規則

イベントで扱われるクラス, メソッド, インタフェースの名前の付け方には、次のような**命名規則がある**

- ▶ リスナ … ○○Listener
- ▶ リスナ登録メソッド … add○○Listener
- ▶ イベントクラス … ○○Event
- ▶ アダプタクラス … ○○Adapter

【例】

○○が**Window**の場合

→ **WindowListener**, **addWindowListener**, **WindowEvent**, **WindowAdapter**

○○が**Mouse**の場合

→ **MouseListener**, **addMouseListener**, **MouseEvent**, **MouseAdapter**

※ただし、Actionにはアダプタクラスが存在しない

(ActionListenerに含まれているのがactionPerformedだけなので)

今回の内容

- ▶ ポップアップメニュー
- ▶ レイアウトマネージャ
- ▶ WindowListenerインターフェース
- ▶ アダプタクラス
- ▶ イベントに関する命名規則
- ▶ 無名クラス

「1_13_MyWindow.java」の赤線部分を変更しましょう (クラスMyWinListener, MyWinAdapterは不要になります)

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

class MyWindow {
    private Frame f1;
    public MyWindow() {
        f1 = new Frame("フレーム");

        f1.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.out.println("終了します。 (無名クラス) ");
                System.exit(0);
            }
        });
        f1.setSize(200, 100);
        f1.setVisible(true);
    }
    public static void main(String[] args) {
        MyWindow obj = new MyWindow();
    }
}
```

【無名クラス (anonymous class) 】
new 親クラス名またはインターフェース名()
 フィールドまたはメソッド定義
}

指定した親クラスを自動的に継承して (または指定したインターフェースを自動的に実装して) 、名前を持たないクラスを定義し、そのインスタンスを作る。
単純なイベント処理を定義する際は有効な書き方。

実行して表示されたフレームを閉じると、アプリケーションが終了される事を確認しましょう。

【課題13-1】

課題11-1で作成したGUIアプリケーションに、「フレームを閉じたらアプリケーションを終了する」動作を追加して下さい。



▶ 無名クラスを使ったサンプルプログラムを参考に作成する。

【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

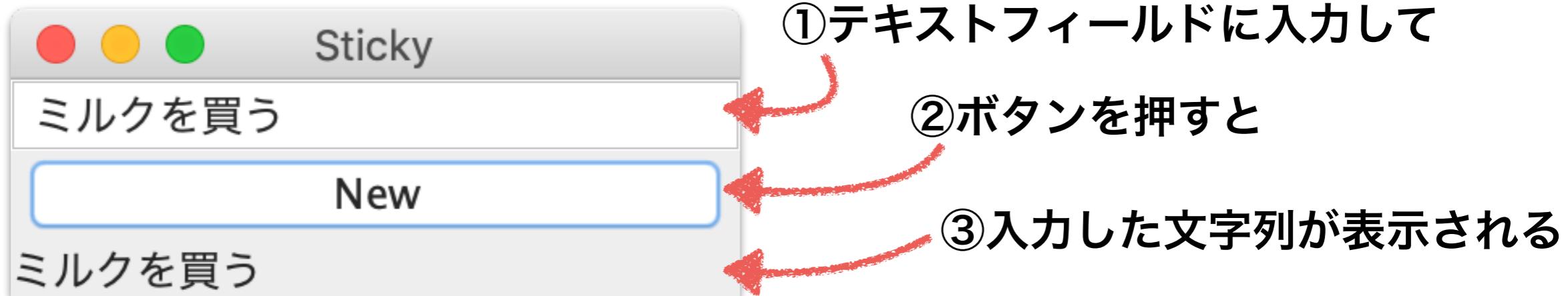
```
$ git add -A  
$ git commit -m "課題13-1提出"  
$ git push origin master
```

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))

【課題13-2】

課題13-1で作成したGUIアプリケーションに「テキストフィールドに文字を入力して、Newボタンを押したら、ラベルにその文字が表示される」動作を追加して下さい。



- ▶ ボタンに対するactionPerformedで以下の処理をする
 - ▶ テキストフィールドのメソッドgetTextで入力されたテキストを取得する
 - ▶ ラベルのメソッドsetTextで取得したテキストをセットする

【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13-2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))

【課題13-3】

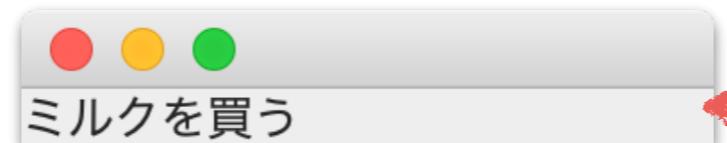
課題13-2で作成したGUIアプリケーションに「テキストフィールドに文字を入力して、Newボタンを押したら、その文字がセットされた**スティッキーが表示される**」動作を追加して下さい。



①テキストフィールドに入力して

②ボタンを押すと

③文字が表示されたスティッキーが作られる



▶ ボタンに対するactionPerformedで以下の処理をする

- ▶ 新しいフレームと新しいラベルを作る
- ▶ フレームとラベルを使って表示するスティッキーの外観を作る
- ▶ テキストフィールドのメソッドgetTextでテキストを取得する
- ▶ ラベルのメソッドsetTextで取得したテキストをセットする

【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13-3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))

【課題13-4】

色付きのスティッキーを作ることができるアプリケーションの外観を作つて下さい。（下図参照）

フレームのレイアウトはGridLayoutで6行1列にする



【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13-4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))

【課題13-5】

課題13-4の「Newボタン」に「色付きスティッキーを作成する」動作を追加して下さい。

色を付ける際は、メニューで選択された項目が何であるかを比較し、それに対応した色に設定するように作る（次のようなif～else if文で、比較処理する）

```
//メニューch1で選択された項目によってラベルstickylabelの文字色を変える  
String fg = ch1.getSelectedItem();  
if(fg.equals("White")) {  
    stickylabel.setForeground(Color.white);  
} else if(fg.equals("Black")) {  
    stickylabel.setForeground(Color.black);  
}  
//以降、色の種類の分だけ if を続ける
```

【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13-5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))