



オブジェクト指向論(Q)

第13回講義資料  
(プログラミング第7回 OOP7)

2023/7/3

來村 徳信

# 前回／今回講義のテーマと流れ

- イベント処理

## 前回 1) Window/Panelに対するマウスイベント

- 前回の目標（レポート課題前半）：図形の内側をクリックして選択する．選択された図形は赤太線で、描画する．
- ステップ 1 (B1)：mouseEventクラス
- ステップ 2 (B2)：クリックによる図形選択
- ステップ 3 (B3)：選択状態に合わせた描画

## 今回 2) UIコンポーネントに対するイベント

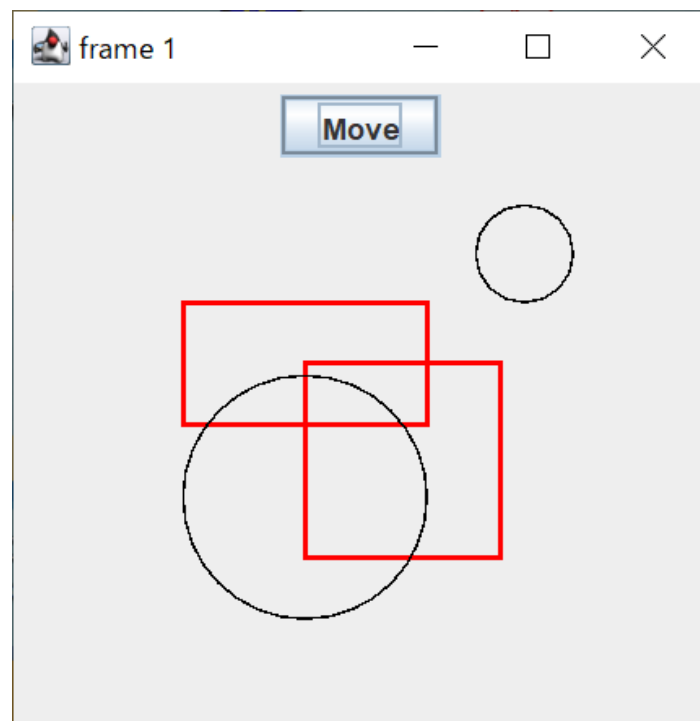
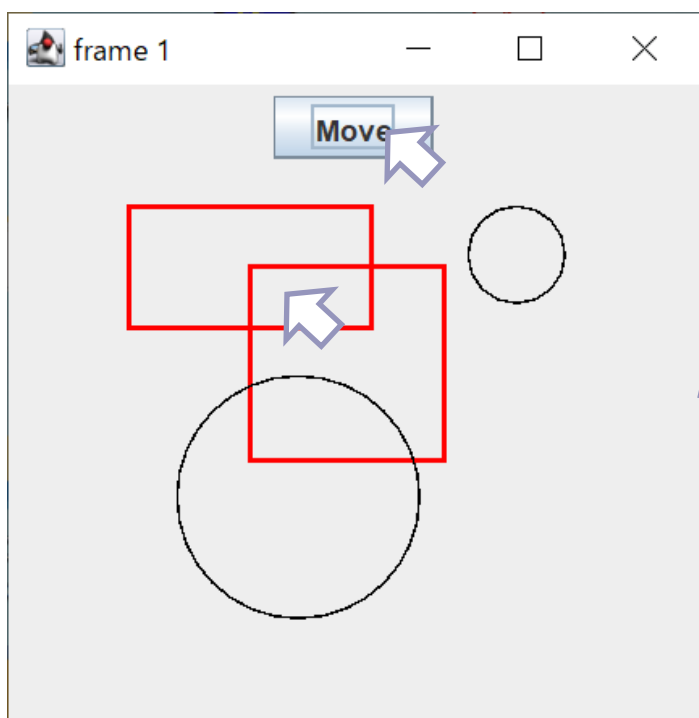
- 今回の目標（レポート課題後半）：ボタンがクリックされたら選択されている図形を移動させる．
- ステップ1：Button へのクリックに対する反応
- ステップ2：クリックされたら選択されている図形を操作する
- 他の実現方法の考察

- 関連の実装と誘導可能性

- フィールドとしての実装．クラスとしての実装

# 後半の目標と動作例

- 「Move」ボタンが押されたら、そのボタンが載っているパネルで「選択状態にある全ての図形」を、右方向に10、下方向に20ずらす。
- 複数の図形が選択状態のときは、全てが移動する。
- 下記の動作例は“Move”ボタンを2回押したときの変化



# Java のイベントモデル (1.1以降)

- 委譲(delegation)イベントモデル

- 「イベント ソース」

- イベントを「発生させる」オブジェクト (例：ウィンドウ)

- 「イベント オブジェクト」


- 1回のイベント (例：クリック1回) に対して1つのイベントオブジェクト (インスタンス) が生成される.
- さまざまな種類がある (クリックやマウスオーバーなど)
- イベントの情報 (クリックなら座標値など) を、インスタンスのフィールド値として持つ.

- 「イベント リスナー」

- イベントが起こったとき「通知される」オブジェクト
- イベントの種類によって特定のメソッドが、イベントオブジェクトのインスタンスを引数として、呼び出される.



# イベントリスナー

- イベントが起こったときに通知されるオブジェクト
  - そのオブジェクトの特定の名前のメソッドが呼ばれる
    - 例：クリックされたら mouseClicked() が呼ばれる。
    - そのときにどのような反応をするかを定義しておく
- 
 ○ 事前に、イベントリスナーのインスタンスを作成し、イベント ソース に「登録」しておく
  - 「私があなたで起こるイベントに反応する」。複数可。
- イベントの種類ごとにイベントリスナーのインタフェースが定義されているので、それを implements し、メソッドの処理内容を実装することで、自分のプログラムとしての反応を定義する。
- 引数はイベントオブジェクトのインスタンス。
- イベントアダプタ(EventAdapter)：イベントリスナーを implements し、空のメソッド（なにもしない）が定義されているもの（後述）

# マウスに関するEventのレベル

- マウスの「イベントオブジェクト」のレベル

- 対応するイベントリスナーのインタフェース

今回

## ActionEvent : 高レベル・イベント

- UIコンポーネントごとに定義された「意味のある」イベントを表す.
- ボタンなら「押されて離された」時のみが重要なので、そのイベントのみを処理できる.
  - ActionListener インタフェースのメソッドは actionPerformed(ActionEvent e) だけ.

前回

## MouseEvent : 低レベル・イベント

- マウスのボタン操作や移動を扱える.
- ➡ インタフェース : MouseListener
  - 中レベル. マウスボタンを押す／離す, クリックなど.
- インタフェース : MouseMotionListener
  - 低レベル. マウスカーソルの移動やドラッグも扱える.

# UIコンポーネントに関するイベント

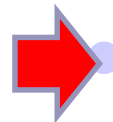
- **ActionEvent**クラス

- <https://docs.oracle.com/javase/jp/8/docs/api/java/awt/event/ActionEvent.html>

- UIコンポーネントごとに決められている特定の動作がされたときに発生する.

- 「意味のある」レベルのイベントだけに反応するため

- イベントソースとその動作例：



- **JButton**クラス

- ボタンが**クリックされた**（押して離された）ときに発生する.

- **JTextField**クラス

- エンターキーが押されたときに発生する。（通常、入力の終わりを意味する）

- ActionEventクラスのメソッドの例：

- **Object getSource()**: イベントが発生したオブジェクト（イベントソース）のインスタンスへの参照を返す.

# ActionEventのリスナークラス

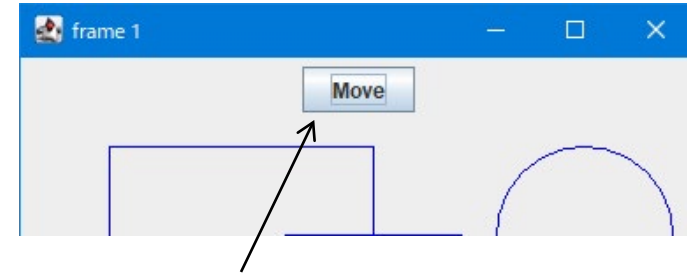
- **ActionListener** インタフェースを実装する
  - <https://docs.oracle.com/javase/jp/8/docs/api/java/awt/event/ActionListener.html>
  - メソッド: **actionPerformed()** のみ
    - 引数はActionEvent インスタンス.
    - ボタンが押されると, そのボタンに登録された ActionListener インスタンスの actionPerformed メソッドが呼び出される.
  - ActionListener インタフェースを **implements** したクラスを定義し, actionPerformed メソッドをオーバーライドすることで, 自分の処理を定義する.
- リスナーのインスタンスを, JButton などのイベントソースに, **addActionListener()** で登録する.
  - 例: JButton のインスタンス **bt** に, ActionListener のインスタンス **al** を追加.

```
bt.addActionListener(al)
```



# ステップ 1 : ボタンに反応させる

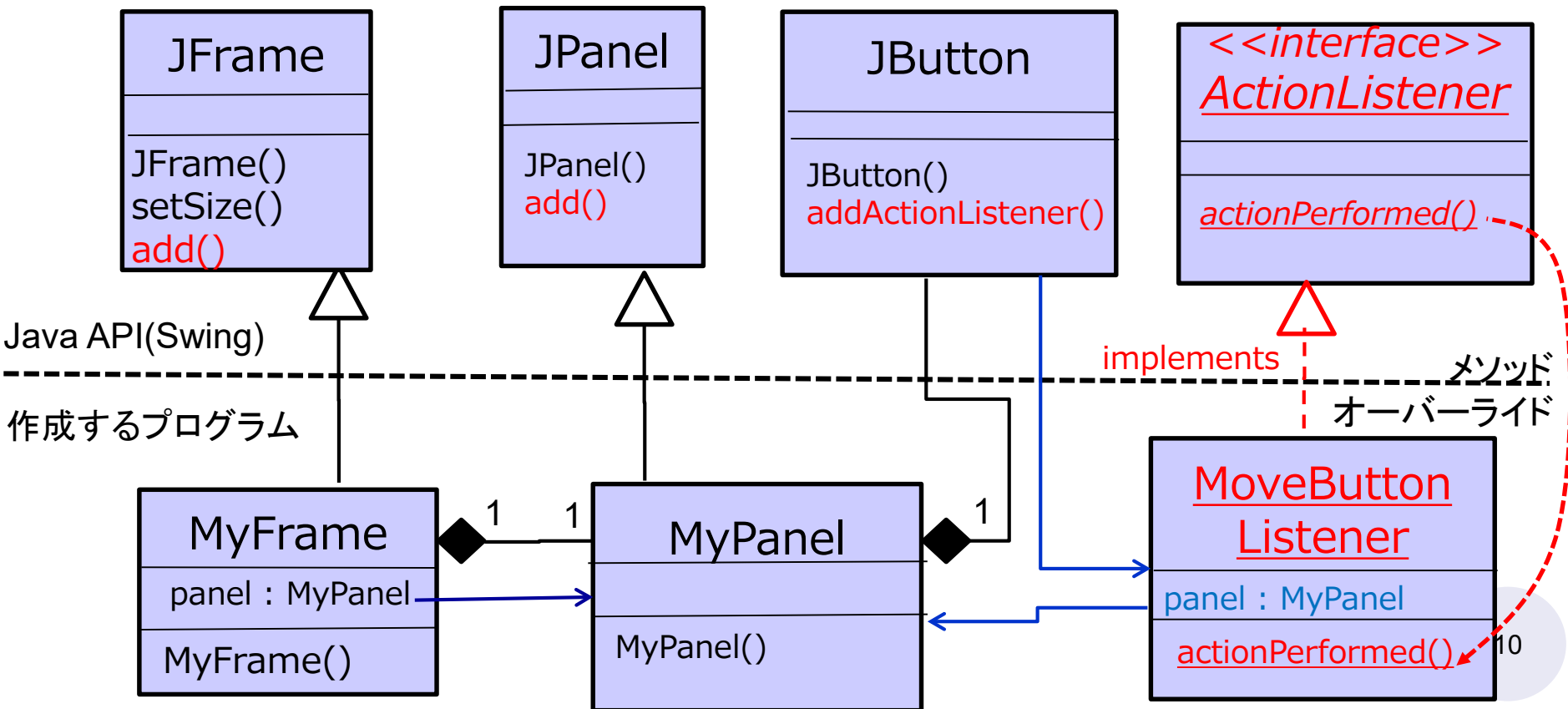
- Windowにボタンを1つ設置する.
  - JButton クラスを使う.
    - コンストラクタ : JButton(String)
  - MyPanel インスタンスに組み込む.
    - 継承されている **add** メソッドを使う.
    - MyPanel インスタンス **mp** に, JButton インスタンス **bt** を組み込む場合 : `mp.add(bt);`
- ボタンが押されると文字列をコンソールに出力する.
  - MoveButtonListenerというクラスを定義する.
  - ActionListener インタフェースを implements する.
    - **actionPerformed()** メソッドをオーバーライドして定義する
  - JButton のインスタンスに「登録」する
    - **addActionListener()** を使う (前のスライド)



Move button is clicked.

# ステップ 1 のクラス図

- `awt.event.ActionListener` を implements して, `MoveButtonListener` クラスを定義
- `addActionListener` メソッドを用いて, `JButton` インスタンスに, `MoveButtonListener` インスタンスを結びつける.



# MoveButtonListenerの定義

- テンプレートが OOP6-A に含まれる (このファイルだけ)
- actionPerformed メソッドの実装を後で追加する

```
public class MoveButtonListener
    implements ActionListener {
    private MyPanel panel;

    MoveButtonListener(MyPanel p) {
        super();
        this.panel = p;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.print("Move Button is clicked");
        if (this.panel != null) {
            //insert here
        }
    }
}
```

ボタンが押されたときに、  
このメソッドが呼ばれる

文字列をコンソールに表示する

※後で記述する

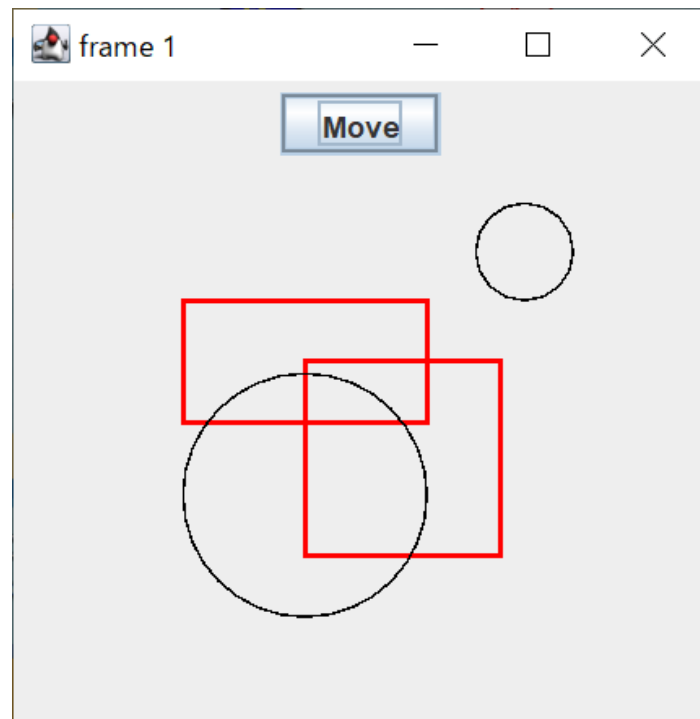
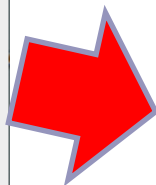
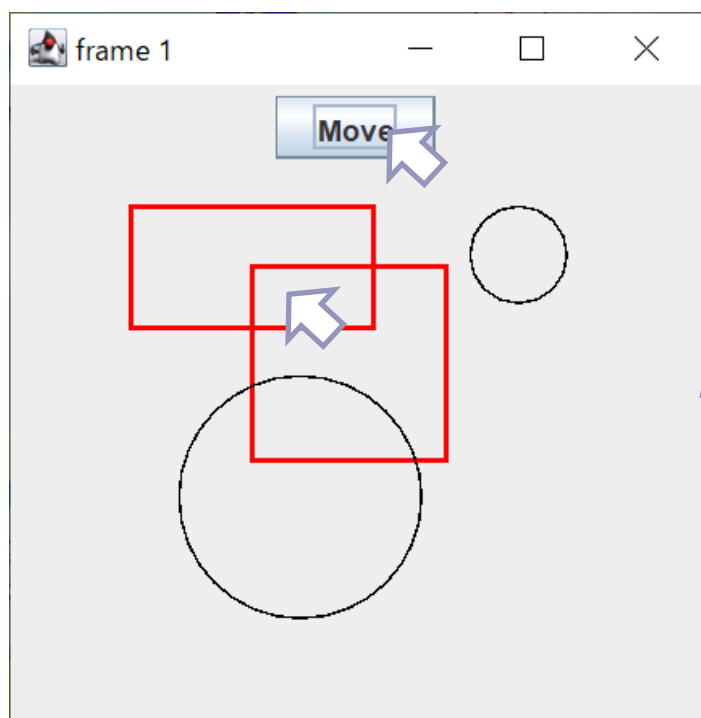
# Buttonの作成とListenerの登録

- JButton のインスタンスを作る
  - MyPanel インスタンス（自分自身）に組み込む.
- MoveButtonListenerのインスタンスを作る.
  - "Move"ボタンのインスタンスに登録する.

```
public MyPanel() { JButton のインスタンスを  
    super(); 作る (引数が表示文字列)  
    ...  
    JButton movebt = new JButton ("Move");  
    (1) (2) ; ← MyPanel インスタンスに組み込む  
  
    ActionListener movebtAl アクションリスナーの  
        = new MoveButtonListener ( (3) インスタンスを作る );  
    (4) .addActionListener ( (5) );  
    ...  
    ← ActionListenerインスタンスを ボタンインスタンスに登録する.
```

# ステップ2：選択図形を移動させる

- 「Move」ボタンが押されたら、そのボタンが載っているパネルで「選択状態にある全ての図形」を、右方向に10、下方向に20ずらす。
- 複数の図形が選択状態のときは、全てが移動する。
- 下記の動作例は“Move”ボタンを2回押したときの変化



# ステップ2の大まかな方針

## ○ MoveButtonListenerクラス

- actionPerformed メソッド内で, MyPanel インスタンスにむけて moveShapes() メソッドを呼ぶ
  - 前回の panelClicked() の呼び出しと同じように.
  - 引数は一回のクリックで移動させる量.

## ○ MyPanelクラス

- void moveShapes(int dx, int dy) を定義する.
- 仕様: 選択されているすべての図形を (dx,dy) だけ動かす.
- 内部動作: MyPanel インスタンスに登録されている全ての図形インスタンスに向けて, moveSelected(dx,dy) を呼び出す.

## ○ boolean moveSelected(int dx, int dy)

- 仕様: 自分が選択状態であれば, 自分を(dx,dy)だけ動かす.
- void move(int dx, int dy) : 選択状態によらず, (dx,dy) だけ自分を右下に移動させる. OOP5-A でも Rect/Circleクラスで定義済み. これらをそのまま使う.

# MoveButtonListener

- actionPerformed()から, **MyPanel の moveShapes** を呼ぶ.
- テンプレートが OOP6-A に含まれる (このファイルだけ)

```
public class MoveButtonListener implements
        ActionListener {
    private MyPanel panel;
```

```
    MoveButtonListener(MyPanel p) {
        super();
        this.panel = p;
    }
```

イベントソースのパネルを覚えておく

ボタンがクリックされると  
このメソッドが呼ばれる

```
@Override
public void actionPerformed(ActionEvent e) {
    System.out.println("Move button is clicked.");
    if (this.panel != null) {
        //insert here
    }
}
```

this.panel へ向けて  
moveShapes を呼ぶ.

ここで  
使うため

※後述の  
「考察」を参照

# MyPanel : moveShapes()

- void moveShapes(int dx, int dy)
  - 自分に登録されている全ての図形インスタンスに向けて moveSelected(dx,dy) メソッドを呼び出し,  
「もし自分が選択状態であれば, (dx,dy)だけ右下方向へ移動せよ」と伝える.
    - ヒント : draw メソッドの呼び出し方をまねる
    - moveSelected() の戻り値は無視してよい.
  - 処理の終了後に this.repaint() を呼ぶこと
    - 移動した図形を描画し直すために, 再描画が必要.

```
public class MyPanel extends JPanel{  
    public void moveShapes (int dx, int dy) {  
        // insert here  
        this.repaint();  
    }  
}
```



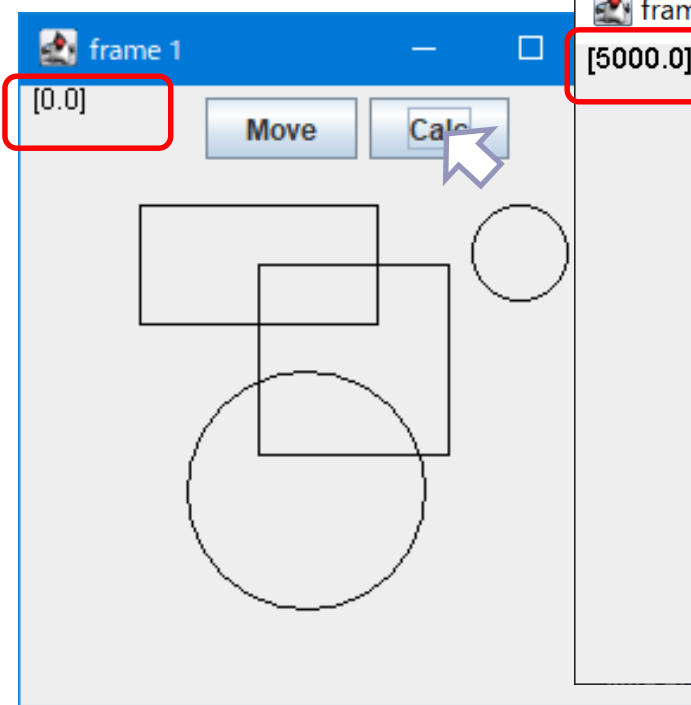
# 選択状態による移動

- boolean **moveSelected**(int dx, int dy)
  - 仕様：自分がもし「選択状態」であれば、自分の座標値を(dx,dy)だけ移動させ、true を返す。  
選択状態でなければ、なんにもせずに、false を返す。
  - 内部動作：自分のフィールド値を調べることで、自分が選択状態かどうかを判断して、選択状態であれば、自分に向けてmove(dx,dy) を呼び出す。
    - move(dx, dy) は選択状態によらず、(dx,dy) だけ自分を右下に移動させるメソッド。OOP5-Aの Rect/Circleクラス で定義済み。これらをそのまま使う。
  - どのクラス／インタフェースで**宣言**されるべきか？
    - 抽象メソッドとして宣言すること。
  - どのクラスで**実装**されるのがよいかを考察して、実装する。

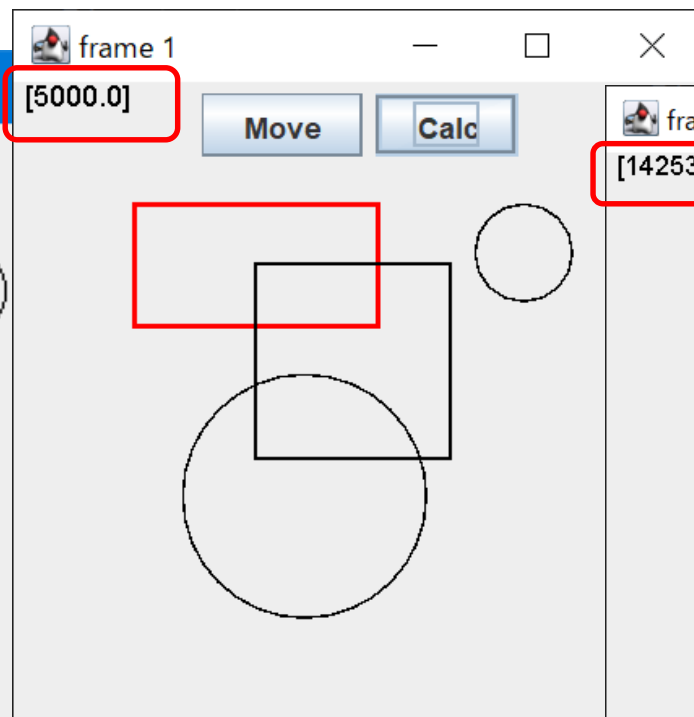
# ステップ3：発展（任意）

- 「Calc」ボタンが押されたら、そのボタンが載っているパネルの、現在選択されている図形インスタンスの面積の和を計算して、Panelに表示する。
  - コンソールに出力ではない。
  - 面積の和は小数点以下2桁目まで（3桁目以下を切り捨てる）

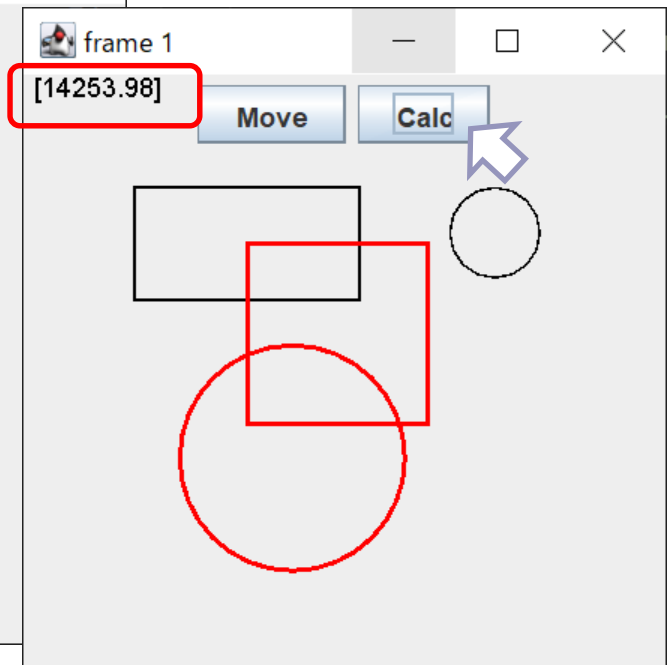
押す前



四角形を選択して押した後



四角形と円を選択して押した後



# 発展：大まかなステップ

- (1) ボタンをもう 1 つ追加する
  - MyPanel クラスで、ボタンを 1 つ追加で作成し、CalcButtonListener クラスのインスタンスと関係づける。
    - MoveButtonListener.java をまねて  
CalcButtonListener.java を追加・定義する。
- (2) 面積を求める。
  - calcSelectedShapeArea() を実装する。
  - 個々の図形インスタンスの選択状態に応じて、面積を足していく。
  - 「選択状態に応じて」はどのように実装すべきか。
    - selected フィールドに関する「注意点」に留意する。
- (3) 面積をパネルに描画する。
  - MyPanel クラスの paintComponent() 内で行う。
  - 和を小数点以下3桁目で切り捨てて表示する。

# 発展：ヒント

- 結果をパネル上に表示するにはどうしたらよい？
  - MyPanelのpaintComponent() メソッド内で描画する必要がある。
    - calcSelectedShapeArea() は合計値を計算するだけで、計算後に this.repaint() を呼んで、再描画をリクエストするだけ。
  - 計算結果をどのように保持して、paintComponent()で利用すればよい。ヒント：その値はMyPanelインスタンスごと。

## ○ 文字列の描画

- double 型変数 *d* の値を、座標(5,10)に黒色で描画するコード

```
import java.awt.Color; ← ファイルの先頭部分に追加する  
g.setColor(Color.BLACK);  
g.drawString("[ " + d + "]", 5, 10);
```

## ○ 小数点3桁目以下を切り捨てる

- `double Math.floor(double d)` を使ってよい。そのまま使うと小数点以下を切り捨てる。小数点以下2桁目まで表示するには？
  - 参考：<https://docs.oracle.com/javase/jp/8/docs/api/java/lang/Math.html#floor-double->

# 考察：イベントリスナーと処理対象

## ● 課題：

- 今回の例では、ボタンをクリックすると、ボタンが載っている MyPanel インスタンスに向けてメソッドを呼び出したい。
  - MoveButtonListener インスタンスの actionPerformed() から、MyPanel インスタンスの moveShapes() を呼びたい。
- イベントソースはあくまでボタン(JButtonインスタンス)
- ボタンが押されたときに呼ばれる actionPerformed メソッドの引数の ActionEvent も、イベントソースである JButton インスタンスの情報だけを持っている。
- 呼び出すべき MyPanel インスタンスが分からない...
- **一般的問題**：イベントリスナーの処理対象をどうやって知るか

## ● 解決策

- (1) コンストラクタとフィールドを使う
- (2) 内部クラス（匿名クラスやラムダ式）を使う。

# 解決策(1)：テンプレートで採用

- MoveButtonListener インスタンス自身に覚えさせておく.
  - インスタンスフィールド `MyPanel panel` を用意する
  - MoveButtonListener インスタンスの生成時に代入する.
    - コンストラクタの引数として `MyPanel` インスタンスをとる
    - コンストラクタ内で、フィールド `panel` に代入する.
    - コンストラクタを呼び出すのは `MyPanel` のメソッドだから、引数として `this` をセットして、呼べばよい.
  - `actionPerformed()` メソッド内で、`panel` フィールドが保持するインスタンスへ `moveShapes`メソッドを呼ぶ
    - MoveButtonListenerインスタンスは、自身が押されたときの、操作対象となる `MyPanel` インスタンスへの参照を持つ.
- 実は前回OOP6でも同じ手法を用いていた.
  - `MyMouseListener` のコンストラクタと `mouseClicked()`からの `panelClicked()` の呼び出し.

# MoveButtonListener

- actionPerformed()から, **MyPanel** の **moveShapes** を呼ぶ.
- テンプレートが OOP7-A に含まれる (このファイルだけ)

```
public class MoveButtonListener implements
    ActionListener {
    private MyPanel panel;
```

```
    MoveButtonListener(MyPanel p) {
        super();
        this.panel = p;
    }
```

イベントソースのパネルを覚えておく

ボタンがクリックされると  
このメソッドが呼ばれる

```
@Override
```

```
public void actionPerformed(ActionEvent e) {
    System.out.println("Move button is clicked.");
    if (this.panel != null) {
        //insert here
    }
}
```

this.panel へ向けて  
moveShapes を呼ぶ.

ここで  
使うため

※この  
「考察」を参照

# 解決策(2)：内部クラス

- MyPanel クラスの「**内部(inner)クラス**」として、MoveButtonListenerを定義する。
  - 「匿名クラス」や「ラムダ式」で書かれることも多い。

```
public class MyPanel extends JPanel {  
    private List<Shape> shapeList;  
    public MyPanel() {  
        super();  
        JButton movebt = new JButton("Move");  
        this.add(movebt);  
        movebt.addActionListener(new MoveButtonListener());  
    }  
    private class MoveButtonListener implements  
        ActionListener {  
        public void actionPerformed(ActionEvent e) {  
            moveShapes(20,10); // shapeList にもアクセス可能.  
        }  
    }  
}
```

デフォルトで定義される引数なしのコンストラクタの呼び出し

MyPanel クラスの内側（内部）で定義されている

外側のMyPanel のメソッドやフィールドが「暗黙的に」継承されているから.  
this はつかない（つけられない）.

(略:MyPanelの続き)



# 前回／今回講義のテーマと流れ（再掲）

- イベント処理

- 1) Window/Panelに対するマウスイベント

- 前回の目標（レポート課題前半）：図形の内側をクリックして選択する．選択された図形は青線で、描画する．
- ステップ 1 (B1)：mouseEventクラス
- ステップ 2 (B2)：クリックによる図形選択
- ステップ 3 (B3)：選択状態に合わせた描画

- 2) UIコンポーネントに対するイベント

- 今回の目標（レポート課題後半）：ボタンがクリックされたら選択されている図形を移動させる．
- ステップ1：Button へのクリックに対する反応
- ステップ2：クリックされたら選択されている図形を操作する
- 他の実現方法の考察

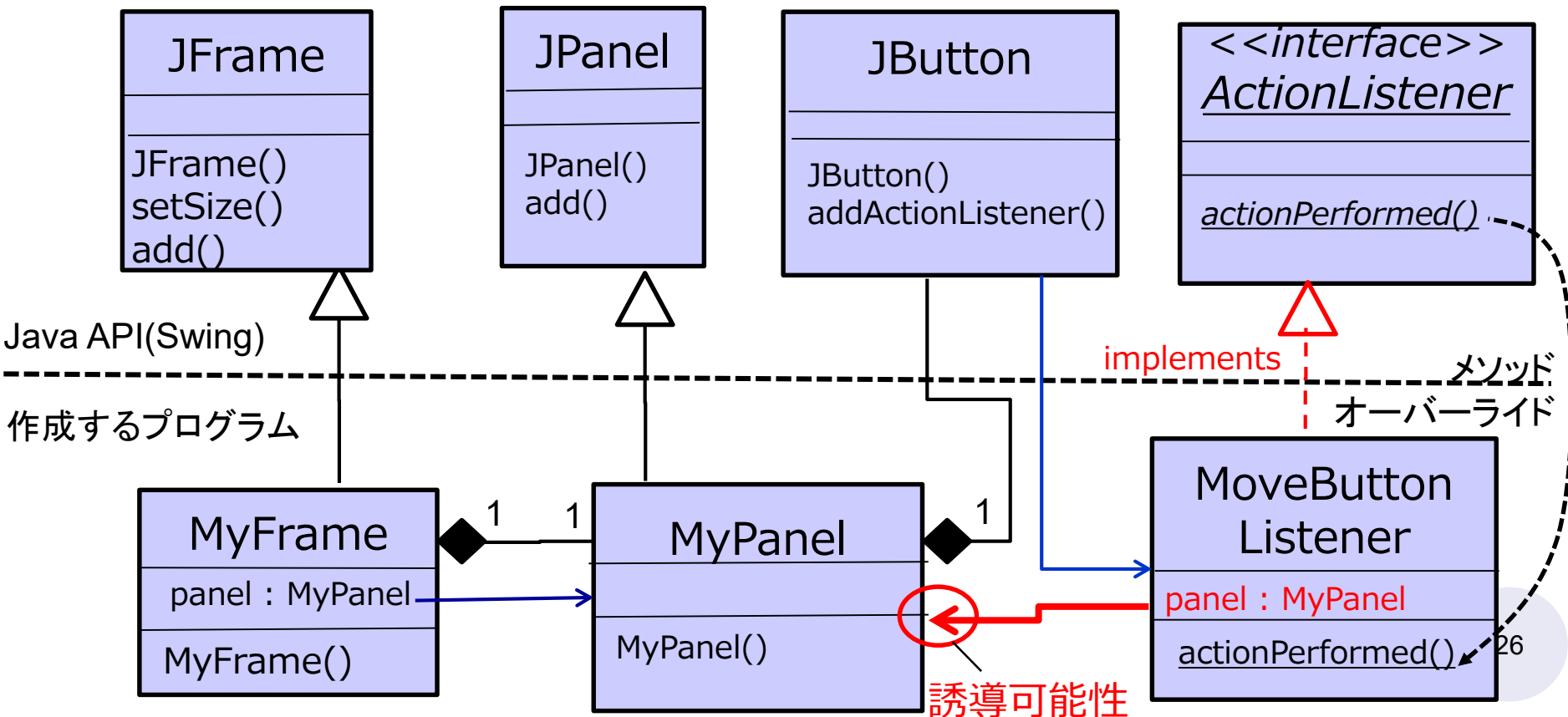


## 関連の実装と誘導可能性

- フィールドとしての実装．クラスとしての実装

# UMLの「関連」の実装の問題

- MoveButtonListener インスタンスからMyPanelインスタンスへの「誘導可能性」をどう実現するかという問題である
  - 解決策(1)は、MoveButtonListener のフィールド panel で、下記の「関連」を実装することで、**その方向の**誘導可能性を実現した、と言える。

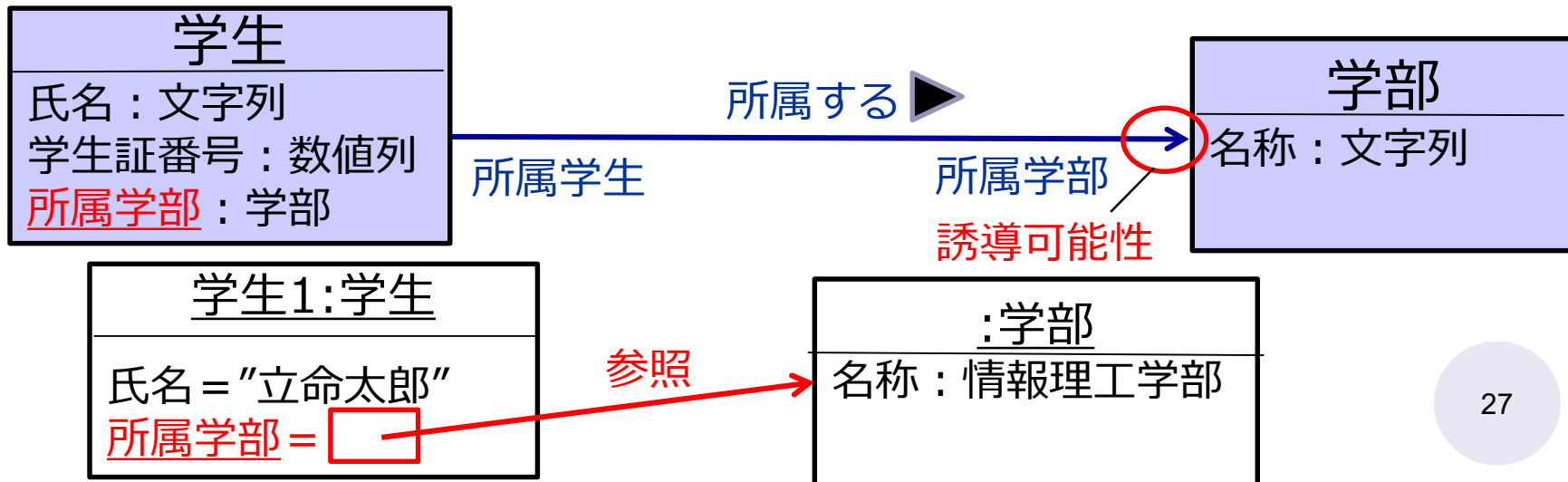


# 関連の実装（プログラム表現）

- 「**関連**」はインスタンス「**フィールド**」として「実装」される。
  - 例：学生と学部「所属する」関係。



- 実装例：「学生」クラスのインスタンスは、「学部」クラスのインスタンスへの**参照**が格納される、「所属学部」という**インスタンスフィールド**を持つ。このとき、学生インスタンスから学部インスタンスへ辿ることができ、メッセージを送る（メソッドを呼ぶ）ことができる。  
「**誘導可能性**」と呼び、クラス図の関連に矢印をつけて表す。



# 関連の実装例：Javaプログラム例(2)

## ● 図書館の(学生)利用者クラス (改)

学生利用者
氏名：文字列
ID番号：数字列
現在借出数：整数
所属学部：学部
is借出OK() 借出()



Student
name : String
id : int
brNum : int
college: College
isBrOK() checkOut()

```
public class Student {  
    private String name;  
    private int id;  
    private int brNum = 0;  
    private College college;  
  
    public Student(String name,  
        College college) {  
        this.name = name;  
        this.college = college;  
    }  
}
```

## ● 動作イメージ

### ○ インスタンス生成とフィールド代入

```
College c1 = new College("ISE")  
Student s1 = new Student("Taro", c1);
```

:Student
name = "Taro"
college = <span style="border: 1px solid red; display: inline-block; width: 50px; height: 20px; vertical-align: middle;"></span>

参照

:College
name = "ISE"

```
public class College {  
    private String name;  
  
    public College(String name) {  
        this.name = name;  
    }  
}
```

# UMLの関連のJavaでの実装

- (a) フィールドとして実装

- インスタンスAの持つインスタンスフィールドの値として、インスタンスBへの参照を保持する。
- (1)単数の場合(0..1)
  - 例1：前ページの「所属学部」
- (2)(3)複数の場合：上限の有無・順序の有無

- (b) クラスとして実装

- UMLでいう「関連を表すクラス」に対応する、クラスをJavaで定義して、そのインスタンスが、A/Bへの参照を保持する。2つのフィールドを使うことになる。

- 参照と誘導可能性

- 片方向的：どちら側のインスタンスが参照を保持するのか
- 両方向的：冗長。便利な場合も。整合性の維持。

# 関連を表す参照の実装の仕方

## (a) フィールドで実装

- 単数である場合(0..1)

- (1) クラス型の「フィールド」で実装できる.

- 例 : MyFrameのMyPanel型のpanelフィールド

- 複数の場合

- (2) 上限数が決まっている場合(0..10など) :

- クラス型を要素とする「配列」で実装できる

- (3) 上限が決まっていない場合(0..\*) :

- (クラスを限定した) 「List」などを使う

- 順序がある場合 : Listインタフェースの実装クラス

- 例 : MyPanelのShape型のshapeList (ArrayList クラス)

- 順序がない場合 : Set や Map を使う

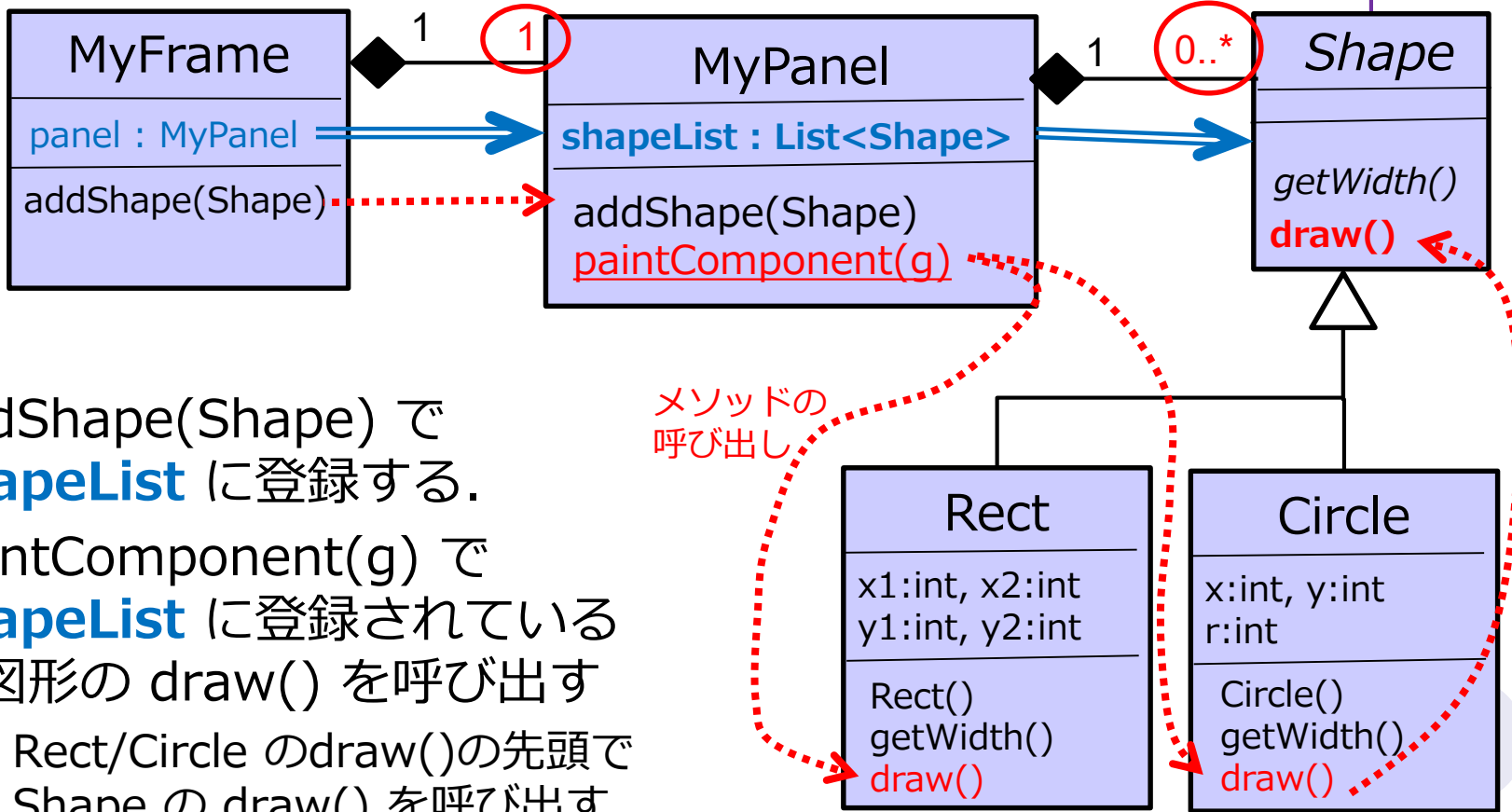
※コレクションフレームワークを参照のこと.

<https://docs.oracle.com/javase/jp/8/docs/technotes/guides/collections/overview.html>

# (a-1)と(a-3)の例

OOP5(改)

- (a-1) MyFrame の panel フィールド
  - MyPanel型. MyFrameのコンストラクタ内で生成・代入
- (a-3) MyPanel の shapeListフィールド
  - Shape型を要素とするArrayList **shapeList** で, Rect/Circleクラスのインスタンスを複数, 保持する.



- addShape(Shape) で **shapeList** に登録する.
- paintComponent(g) で **shapeList** に登録されている全図形の draw() を呼び出す
  - Rect/Circle のdraw()の先頭で Shape の draw() を呼び出す.

# コレクションフレームワーク

- オブジェクトの集まりを扱うクラス群(java.util.)
  - <https://docs.oracle.com/javase/jp/8/docs/technotes/guides/collections/overview.html>

例：

- List インタフェース
  - 順序がある. インデックスを指定してアクセスできる.
- Set インタフェース
  - 順序がない. 要素は重複しない.
- Map インタフェース
  - キーと値の組. 順序がない.
- それぞれいくつかの実装クラスがある.
  - 使えるメソッドは同じ.
  - 異なる特徴をもつ. 例：ある操作に対する速度が異なる.



# 参考：Java の配列

- 特殊なオブジェクトと捉えられる。
- 宣言（どちらの書き方でもよいが、上が現在の標準的）
  - 要素の型[] 配列名. 例：int[] *intArray*; Rect[] *rectArray*;
  - 要素の型 配列名[]. 例：int *intArray*[]; Rect *rectArray*[];
- 生成：実行時に**固定長**
  - new 要素の型[要素数] 例：int[] *intArray* = new int[10];
  - 配列の領域が（動的に）確保される。
    - 要素の値は要素型のデフォルト値(0やnull)で初期化される。
- アクセス
  - 配列名[添え字] でアクセスできる。添え字は0から。
    - 例： *intArray*[2] = 10; *rectArray*[0] = new Rect(...);
  - 配列の長さは、配列名.length で得られる。読み取り専用のフィールド。例： *intArray*.length

## (b) 「関連を表すクラス」として実装

- Javaとしては普通のクラス
  - 参照の持ち方も同じ.
  - 関連に属性を持たせたいときによく使われる.
  - 例 :
    - 「学部所属」 関連クラス : 「入学年度」 フィールド

```
public class Enrollment {  
    private Student student;  
    private College college;  
    private Date enrolledYear;  
    ...  
}
```

```
public class Student {  
    private String name;  
    ..  
}
```

```
public class College {  
    private String name;  
    ..  
}
```

# 前回／今回講義のまとめ

- イベント処理

- 1) Window/Panelに対するマウスイベント

- 前回の目標（レポート課題前半）：図形の内側をクリックして選択する．選択された図形は赤太線で、描画する．
    - ステップ 1 (B1)：mouseEventクラス
    - ステップ 2 (B2)：クリックによる図形選択
    - ステップ 3 (B3)：選択状態に合わせた描画

- 2) UIコンポーネントに対するイベント

- 今回の目標（レポート課題後半）：ボタンがクリックされたら選択されている図形を移動させる．
    - ステップ1：Button へのクリックに対する反応
    - ステップ2：クリックされたら選択されている図形を操作する
    - 他の実現方法の考察

- 関連の実装と誘導可能性

- フィールドとしての実装．クラスとしての実装