プログラミング２
第12回目資料

中村正樹

インタフェース

- インスタンスフィールドを持たず，メソッドがすべて抽象メソッドな参照型
  - 参照型：配列，クラス
  - 構文：interface *インタフェース名* { *定義* }

```
1: interface Lockable {
2:     boolean lock();
3:     boolean unlock();
4: }
```

Locable.java

インタフェースの実装

- クラスはインタフェースを実装(implements)することができる（拡張(extends)ではなく）
  - 構文：class *クラス名* implements *インタフェース名* { *定義* }

```
1: interface Lockable {
2:     boolean lock();
3:     boolean unlock();
4: }
```

Locable.java

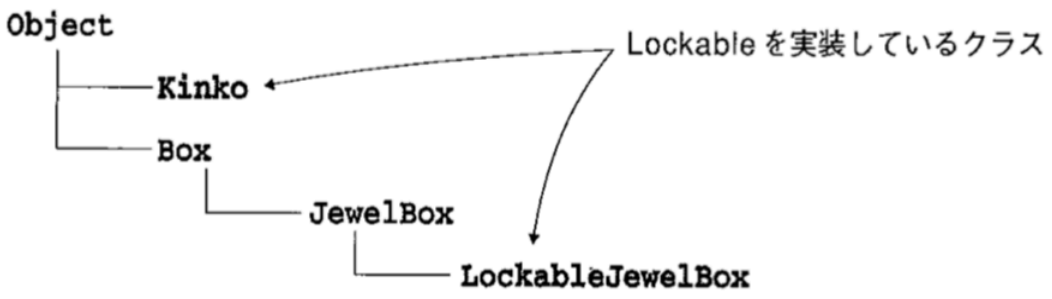
```
1: class Kinko implements Lockable {
2:     @Override
3:     public boolean lock() {
4:         ...
5:     }
6:     @Override
7:     public boolean unlock() {
8:         ...
9:     }
10: }
```


クラス階層との関係

- インタフェースは拡張ではない

```
1: class LockableJewelBox extends JewelBox implements Lockable {
2:     @Override
3:     public boolean lock() {
```

```
1: class Kinko implements Lockable {
2:     @Override
3:     public boolean lock() {
```



インタフェース型の変数

- クラスと同様に，インタフェース型の変数を宣言できる
  - インタフェースを実装したクラスのインスタンスを代入できる

```
class MyNumber implements DebugPrintable {
    int a;
```

```
class MyFileReader extends FileReader implements DebugPrintable {
    String filename = null;
```

```
DebugPrintable obj = new DebugPrintable();          ×誤り
DebugPrintable obj = new MyNumber(123);             ○正しい
DebugPrintable obj = new MyFileReader("input.txt"); ○正しい
DebugPrintable[] objs = new DebugPrintable[2];

objs[0] = new MyNumber(123);
objs[1] = new MyFileReader("input.txt");
```


インタフェースのフィールド

- public static finalのみ（省略可）

```
1: interface DebugPrintable {
2:     int NO_ERROR = 0;
3:     int FILE_ERROR = 1;
4:     int MEMORY_ERROR = 2;
5:     String PREFIX = "ERROR:";
6:     void debugPrint();
7: }
```

```
1: interface DebugPrintable {
2:     public static final int NO_ERROR = 0;
3:     public static final int FILE_ERROR = 1;
4:     public static final int MEMORY_ERROR = 2;
5:     public static final String PREFIX = "ERROR:";
6:     public abstract void debugPrint();
7: }
```

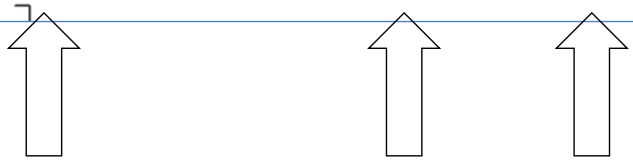
しっかり覚えよう　インタフェースのフィールドとメソッド

インタフェースのフィールドは、public static final である。
インタフェースのメソッドは、public abstract である。

多重継承とインタフェース

- Javaでは複数のクラスを継承できない
- 複数のインタフェースを実装することはできる
  - 擬似的な多重継承

```
class MyReaderWriterClass extends MyMediaClass implements MyReadable, MyWritable {
    @Override
```



スーパークラスは1つ

実装するインタフェースは複数可

クラスとインタフェース比較

	クラス	インタフェース
インスタンス	作れる	作れない
メソッド	いろいろ	必ずpublic abstract
フィールド	いろいろ	必ずpublic static final
スーパークラス	1つだけ	持てない

スーパーインタフェース	複数指定可能
	(implementsを使う)
	class A implements B, C {
	...
	}
		複数指定可能
		(extendsを使う)
		interface X extends Y, Z {
		...
		}

課題12

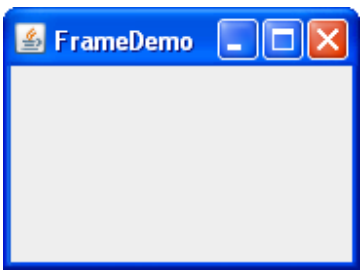
- GUI（Graphical User Interface）版のエディタプログラムをベースに下記の修正をせよ
  - Movableインタフェースを作り，FigureをMovableの実装クラスとせよ

```
interface MovableObject {
    // x, y座標にそれぞれmx, myを加える
    void move(int mx, int my);
}
```

  - Figureのサブクラスを作成し，add Circ x y r を実装せよ

javax.swing.JFrameクラス

- JFrameクラスのインスタンスとして実装されるフレームは，タイトルやウィンドウを閉じるまたはアイコン化するボタンコンポーネント等をサポートする
- 通常GUIを使用するアプリケーションには，少なくとも1つのフレームが含まれる．

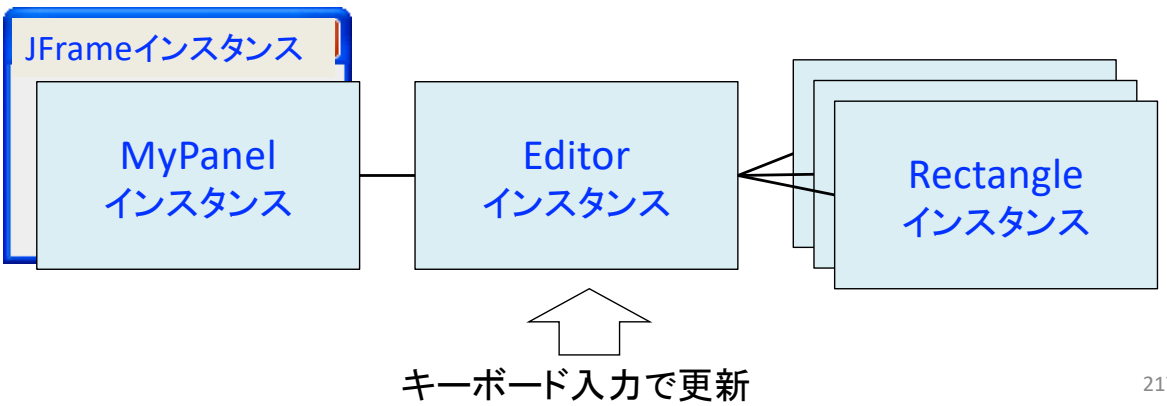


javax.swing. JPanelクラス

- JPanelクラスは，軽量コンポーネント用の汎用コンテナを提供する
  - ボタンやテキストを追加したり，ペイントをカスタマイズしたりできる
  - フレームにパネルを追加して使う

JPanel のサブクラスMyPanel

- JPanelクラスを拡張して，自前のパネルクラスMyPanelを作成する
  - Editorインスタンスをフィールドに持つ



Kadai12クラス（mainメソッド）前半

- JFrameインスタンスmyWindow と MyPanelインスタンスmyPanelを生成し，myWindowにmyPanelを追加する

```
public static void main(String[] args) {
    Editor editor = new Editor();

    // GUI版の追加部分：はじまり
    JFrame myWindow = new JFrame();
    myWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    myWindow.setSize(800,600); // フレームのサイズを指定
    myWindow.setTitle("Nakamura's Figure Editor");
    MyPanel myPanel = new MyPanel(editor);
    myWindow.add(myPanel);      //フレームに自作パネルを追加
    myWindow.setVisible(true);  // 見えるようにする
    // GUI版の追加部分：おわり

    BufferedReader input =  new BufferedReader (new InputStre
```


Kadai12クラス（mainメソッド）後半

- 前回同様，キーボードからコマンド入力のループを回す
  - myWindowのrepaintメソッドを呼び出す
  - ループから抜けたらmyWindowsを閉じる

```
// GUI版の追加部分：おわり

BufferedReader input =  new BufferedReader (new InputStreamReader (System.in));
boolean goOn = true;
while(goOn){
    goOn = editor.execute(input);
    myWindow.repaint(); // コマンドを入力するたびにフレームの中身を再描画させる
}
myWindow.dispose(); // 終わったらフレームを閉じる
```

MyPanel 前半

- MyPanelクラスは，JPanelを拡張する
- フィールドにEditorクラスのインスタンスeditorを持たせる

```
import javax.swing.*;
import java.awt.*;

public class MyPanel extends JPanel{
    private static final long serialVersionUID = 1L;
    private Editor editor; // フィールドにEditorインスタンスを持っている
    public MyPanel(Editor editor) {
        super();
        this.editor = editor;
    }
```

MyPanel 後半

- JPanelのpaintComponentメソッドをOverrideする
  - editorの持つ図形すべてのdrawメソッドを呼び出す
  - myWindowのrepaintが呼ばれると，そのコンポーネントのmyPanelのpaintComponentが呼ばれる

```
    public void paintComponent(Graphics g){ // フレームのrepaintから呼ばれる
        // editorの各図形のdrawを呼び出す
        Figure[] flist = editor.getFlist();
        for(int i = 0; i < flist.length; i++) {
            flist[i].draw(g);
        }
    }
}
```

DrawableObjectインタフェースとFigureクラス

- drawメソッドを持つインタフェース
  - MyPanelクラスのpaintComponentメソッドから呼ばれる

```
import java.awt.*;

interface DrawableObject {
    void draw(Graphics g);
}
```

- FigureクラスをDrawableインタフェースの実装クラスとして宣言する

```
abstract class Figure extends Point implements DrawableObject{
    Figure(int x, int y){..
    public String toString() {..
}
```

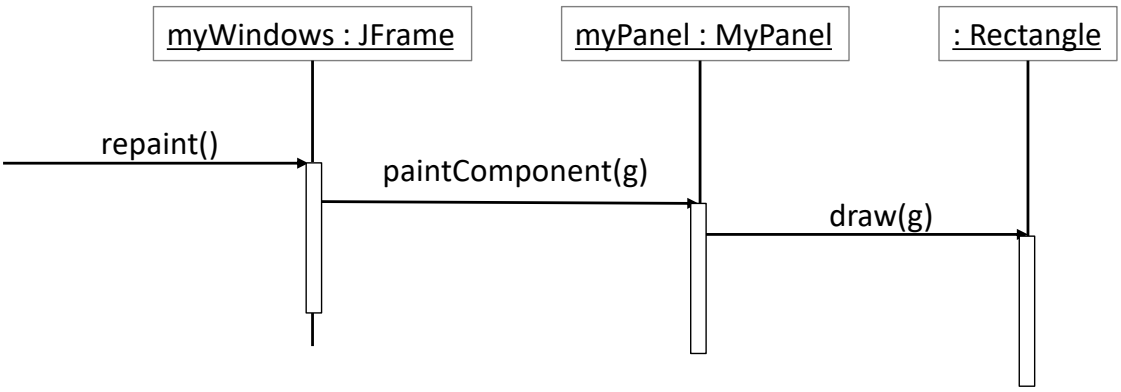

Rectangleクラス

- Figureのdrawメソッドを定義する
  - 色を白に指定する（setColor）
  - 塗りつぶし四角を書く（fillRect）
  - （黒で）四角の枠を書く（drawRect）

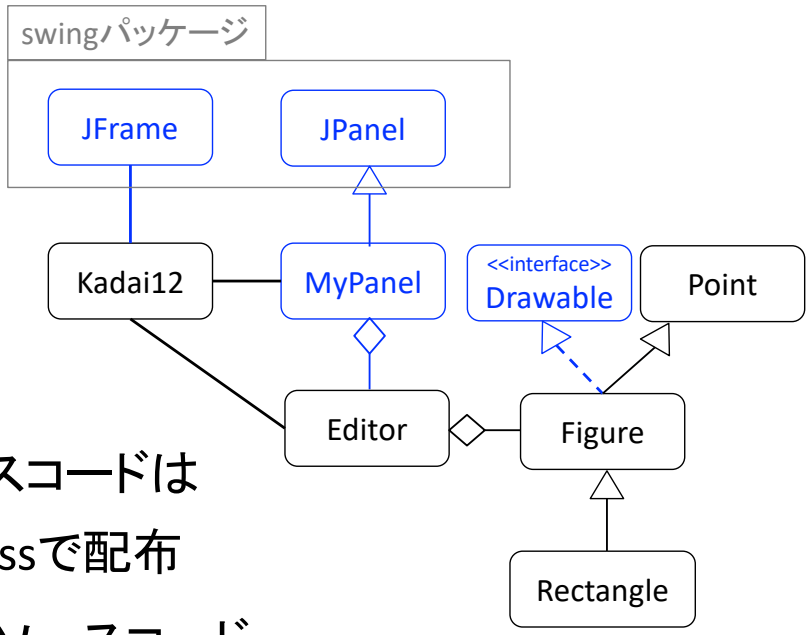
```
import java.awt.*;

class Rectangle extends Figure {
    private int width;
    private int height;
    public Rectangle(int x, int y, int w, int h) {..
    int getW() { return width; }
    int getH() { return height; }
    void setSize(int w, int h) {..
    public String toString() {..
    // パネルのpaintから呼ばれる
    public void draw(Graphics g) {
        // 色を白に指定して，塗りつぶしの四角を描く
        g.setColor(Color.white);
        g.fillRect(getX(), getY(), width, height);
        // 色を黒に指定して，枠だけの四角を描く
        g.setColor(Color.black);
        g.drawRect(getX(), getY(), width, height);
    }
}
```

java.awt.Graphicsクラス

- myWindowのrepaintが呼ばれると，その構成要素のpaintComponen(g)tメソッドが呼ばれる
  - 引数gはmyPanelのグラフフィクスコンテキスト（Graphicsインスタンス）
- paintComponent(g)は，各図形のdraw(g)を呼び出し，gに対して，四角など描画する



ベースプログラムの構成



- 各ソースコードはWebClassで配布
- 詳細はソースコードのコメントを参照

課題12-1

- Movableインタフェースを作り，FigureをMovableの実装クラスとせよ
  - 抽象メソッドmoveをどこかで定義する必要がある
  - moveコマンドも定義する（Editorクラスのexecuteメソッド内）
    - コマンド「move i mx my」を入力すると，i 番目の図形の座標を +mx, +myだけ移動する

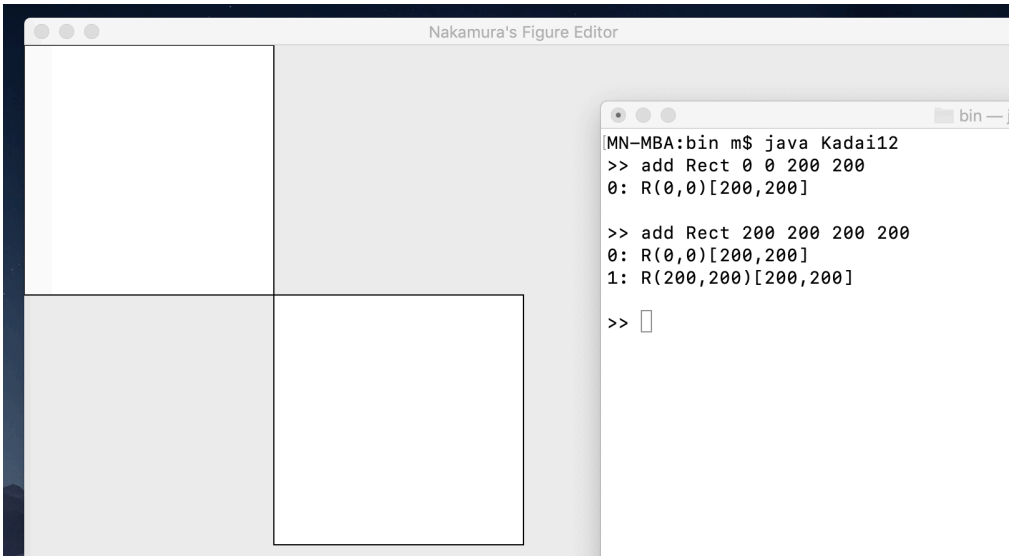
```
interface MovableObject {
    // x, y座標にそれぞれmx, myを加える
    void move(int mx, int my);
}
```

```
abstract class Figure extends Point
                        implements DrawableObject, MovableObject{
```

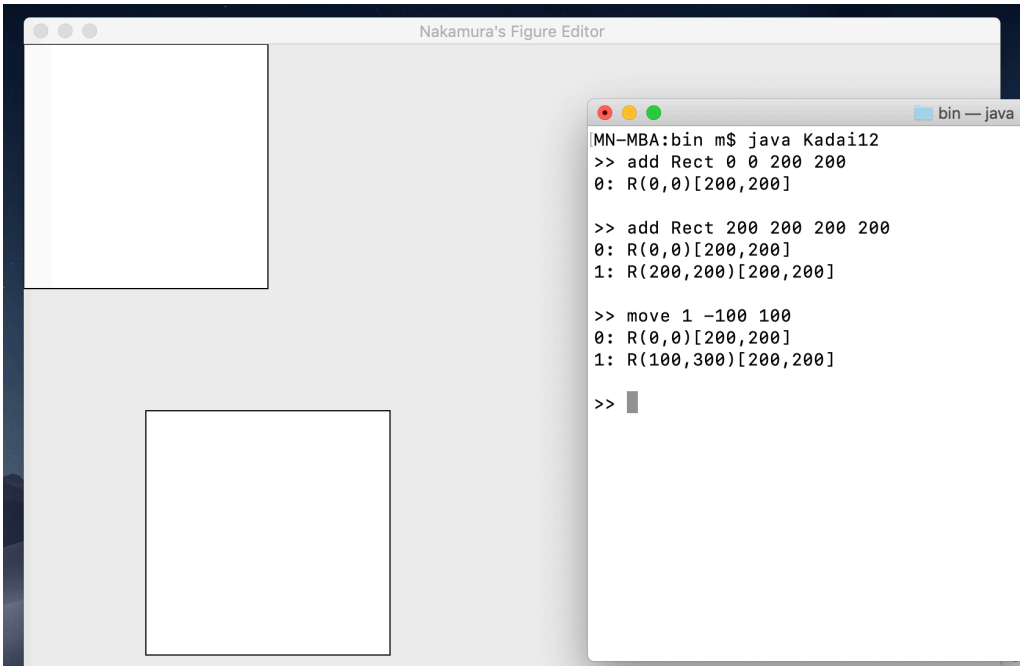
課題12-2

- 四角だけでなく，円も扱えるように改良せよ
  - Circleクラスを作成する（Figureのサブクラス）
  - add Circ *x y r* で中心座標 (*x*, *y*)に半径 *r* の円を描画する
    - EditorクラスのcreateFigureメソッド内
    - Graphicsクラスのメソッドを調べる

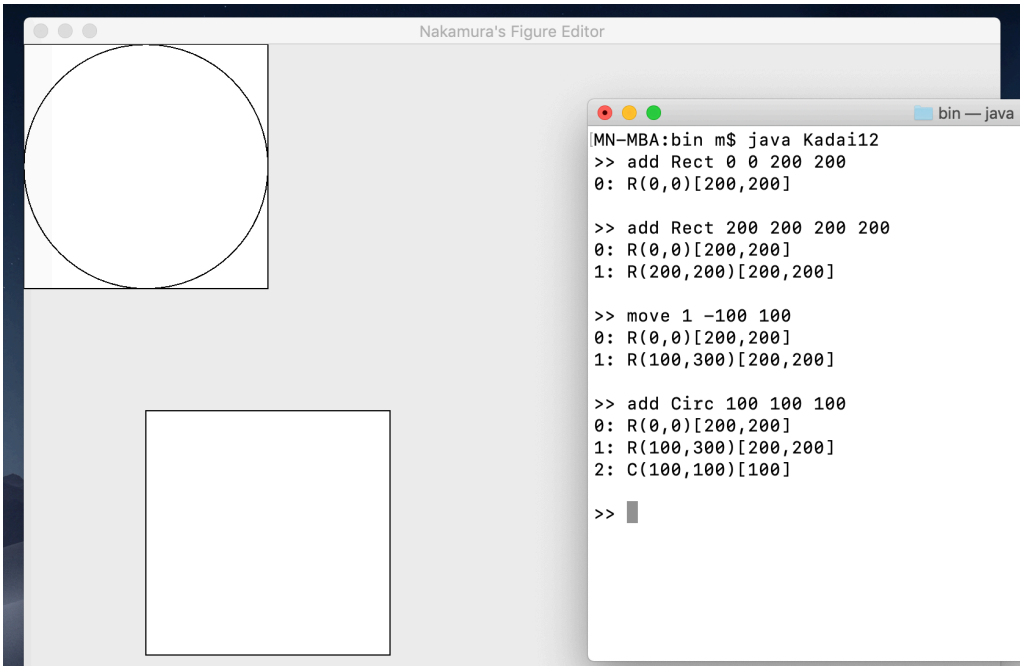
実行例1


Nakamura's Figure Editor
bin —
MN-MBA:bin m$ java Kadai12
>> add Rect 0 0 200 200
0: R(0,0)[200,200]
>> add Rect 200 200 200 200
0: R(0,0)[200,200]
1: R(200,200)[200,200]
>>

実行例2


Nakamura's Figure Editor
bin — java
MN-MBA:bin m$ java Kadai12
>> add Rect 0 0 200 200
0: R(0,0)[200,200]
>> add Rect 200 200 200 200
0: R(0,0)[200,200]
1: R(200,200)[200,200]
>> move 1 -100 100
0: R(0,0)[200,200]
1: R(100,300)[200,200]
>>

実行例3



宿題12

- 第19章 コレクションのList 19-5（277ページ）のプログラムを作成して，テストし，ソースコードと実行結果を提出せよ