オブジェクト指向プログラミング(2)

第3回

横山　孝典
E-mail: tyoko@tcu.ac.jp

オブジェクト指向のキーワード

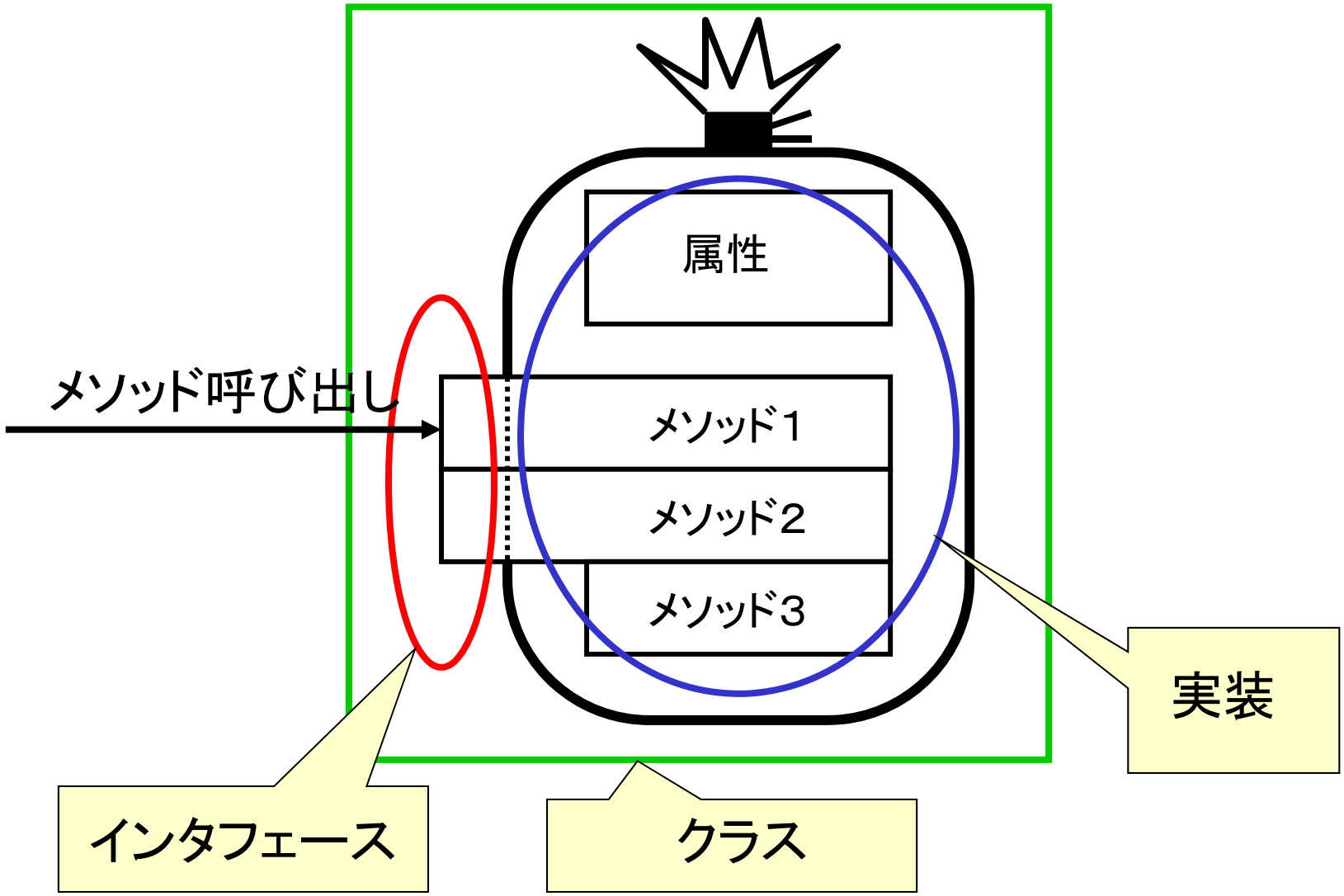
これまでのキーワード

- クラスとインスタンス
- コンストラクタ
- カプセル化（インターフェースと実装）
- 継承（型）
- ポリモーフィズム
- 関連
- 集約とコンポジション
- デザインパターンとフレームワーク

今回のキーワード（継承（型）に関する追加事項）

- インタフェース
- 多重継承

インタフェース



Javaでは、インタフェースだけを定義することもできる

インタフェースの例

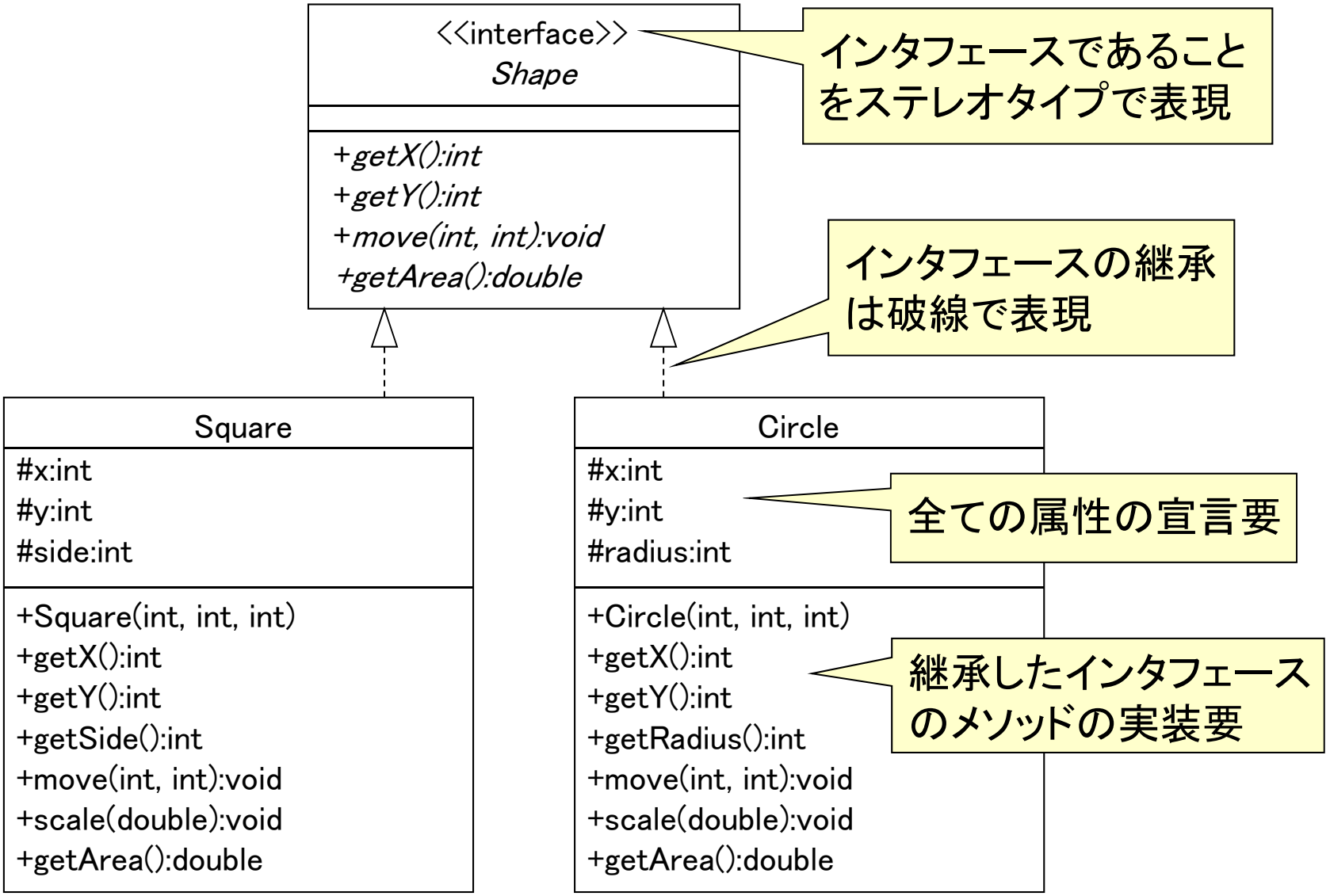
図形一般が持つべきメソッドを（クラスでなく）インタフェースとして宣言

インタフェースの宣言

```
// Shape.java
public interface Shape {   // 図形（インターフェース）
  public int getX();    // X座標の読み出し
  public int getY();    // Y座標の読み出し
  public void move(int dx, int dy);  // 移動
  public double getArea();     // 面積計算
}
```

インタフェースにおけるメソッドは
（abstract をつけなくても）全て抽象メソッド

インタフェースの継承


<<interface>>
Shape
+getX():int
+getY():int
+move(int, int):void
+getArea():double
インタフェースであること
をステレオタイプで表現
インタフェースの継承
は破線で表現
Square
#x:int
#y:int
#side:int
+Square(int, int, int)
+getX():int
+getY():int
+getSide():int
+move(int, int):void
+scale(double):void
+getArea():double
Circle
#x:int
#y:int
#radius:int
全ての属性の宣言要
+Circle(int, int, int)
+getX():int
+getY():int
+getRadius():int
+move(int, int):void
+scale(double):void
+getArea():double
継承したインタフェース
のメソッドの実装要

インタフェースの継承の記述

クラスがインターフェースを継承する場合

```
public class クラス名 implements インタフェース名 {
  . . .
}
```

継承を表す

継承するインタフェース名

インタフェースがインタフェースを継承する場合

```
public interface インタフェース名
                          extends スーパーインタフェース名 {
  . . .
}
```

継承を表す

継承するインターフェース名

インタフェースの継承の例（1）


```
// Square.java
public class Square implements Shape {  // 正方形（図形を継承）
  int x = 0;      // X座標
  int y = 0;      // Y座標
  int side = 0;   // 辺の長さ

  public Square(int xx, int yy, int s) {  // コンストラクタ
    x = xx;         // X座標の初期化
    y = yy;         // Y座標の初期化
    side = s;       // 辺の長さの初期化
  }

  public int getX() {  // X座標の読み出し
    return x;
  }
  public int getY() {  // Y座標の読み出し
    return y;
  }
  public int getSide() {  // 辺の長さの読み出し
    return side;
  }
  public void move(int dx, int dy) {  // 移動
    x = x + dx;
    y = y + dy;
  }
  public void scale(double ratio) {  // 拡大
    side = side * ratio;
  }
  public double getArea() {  // 面積計算
    return (side * side);    // 面積＝辺＊辺
  }
}
```

インタフェースの継承の例(2)

```
// Circle.java
public class Circle implements Shape {   // 円(図形を継承)
  int x = 0;          // X座標
  int y = 0;          // X座標
  int radius = 0;     // 半径

  public Circle(int x, int y, int r) {   // コンストラクタ
    x = xx;           // X座標の初期化
    y = yy;           // Y座標の初期化
    radius = r;       // 半径の初期化
  }

  public int getX() {   // X座標の読み出し
    return x;
  }
  public int getY() {   // Y座標の読み出し
    return y;
  }
  public int getRadius() {   // 半径の読み出し
    return radius;
  }
  public void move(int dx, int dy) {   // 移動
    x = x + dx;
    y = y + dy;
  }
  public void scale(double ratio) {   // 拡大
    radius = radius * ratio;
  }
  public double getArea() {             // 面積計算
    return (radius * radius * 3.14); // 面積＝半径＊半径＊円周率
  }
}
```

インタフェースの継承の例(3)

インタフェースを継承してインタフェースを定義する例

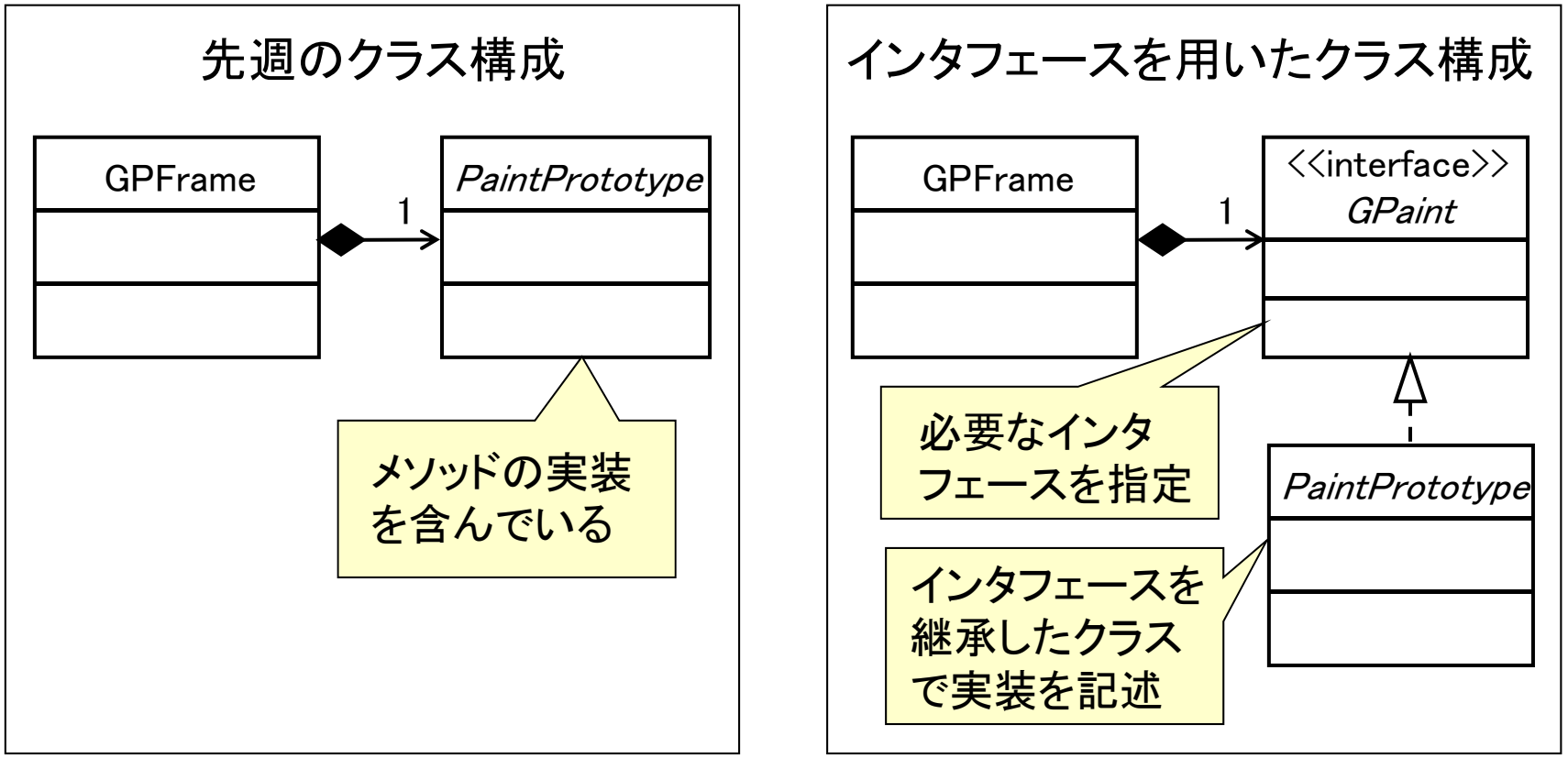
```
// Shape.java
public interface Shape {  // 図形(インターフェース)
  public int getX();   // X座標の読み出し
  public int getY();   // Y座標の読み出し
  public void move(int dx, int dy);  // 移動
  public double getArea();    // 面積計算
}
```

```
// Square.java
public interface Square extends Shape {  // 図形を継承
   public int getSide();           // 辺の長さの読み出し
   public void scale(double ratio);  // 拡大
}
```

インタフェースの例（２－１）

あるクラスと関連のあるクラスが必ず実装すべきメソッドを指定する

先週の課題の例：“GPFrame”のGUIを用いるクラスが実装すべきメソッドを指定するインタフェース“GPaint”を導入



インタフェースの例（２－２）

```
// GPaint.java
import java.awt.Graphics;   // クラスGraphicsを利用する
import java.awt.Color;      // クラスColorを利用する

// 図形作成ツール用のインタフェース
public interface GPaint {
  // 初期化メソッド
  // インスタンス生成後に呼び出される
  public void init(Graphics gra);
  // 「図形」の「直線」が選択された時に呼び出されるメソッド
  public void line(Graphics gra);

        . . . . .

  // マウスキーが離された時に呼び出されるメソッド
  // 離された時点の座標が引数で渡される
  public void mouseReleased(int xx, int yy, Graphics gra);
  // 全ての図形を再表示するメソッド
  // ウィンドウが別のウィンドウに隠されて、再び現れた時などに呼び出される
  public void redrawAll(Graphics gra);
}
```

“GPFrame”から
呼び出される
全てのメソッドを
宣言する

インタフェースの例（２－３）

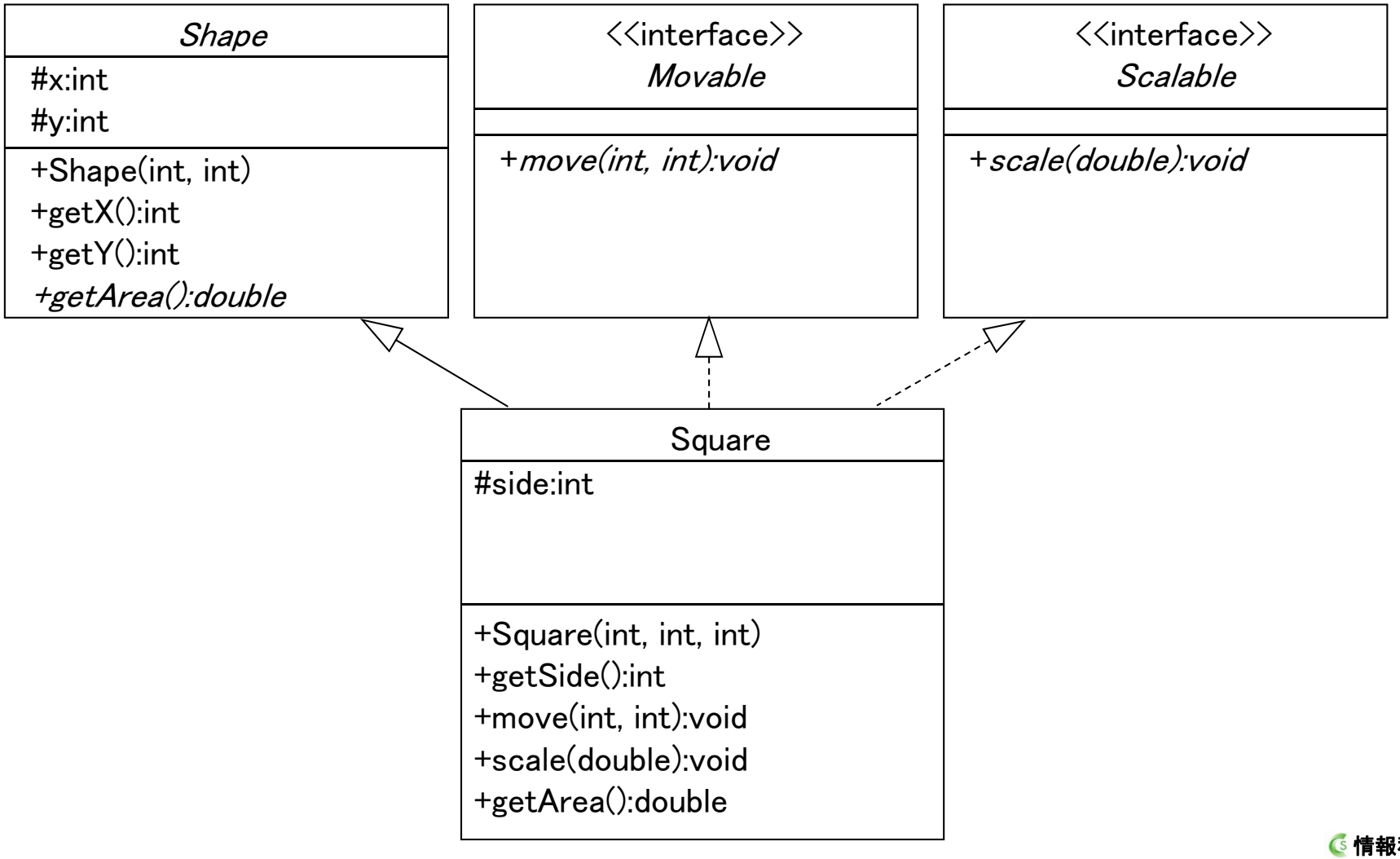
```
// PaintPrototype.java
import java.awt.Graphics;  // クラスGraphicsを利用する
import java.awt.Color;     // クラスColorを利用する

public abstract class PaintPrototype implements GPaint {
  int shapeId = 0;  // 図形の種類を表す変数
   . . .

  //  初期化メソッド
  //  インスタンス生成後に呼び出される
  //  必要であれば、継承したクラスでオーバライドする
  public void init(Graphics gra) {
  }
   . . .
  //  全ての図形を再描画するメソッド
  //  ウィンドウが別のウィンドウに隠されて、再び現れた時などに呼び出される
  //  必要であれば、継承したクラスでオーバライドする
  public void redrawAll(Graphics gra) {
    for (int i=0; i < numOfShape; i++) {
      shape[i].draw(gra);
    }
  }
}
```

インタフェース“GPaint”を継承する

インタフェース“GPaint”で宣言した全てのメソッドを実装する

多重継承

クラスとインタフェースの両者を継承することも可能
クラスは1つしか継承できないが、インタフェースは複数継承可能



多重継承の記述

継承を表す　　継承するクラス名

```
public class クラス名 extends スーパークラス名
        implements インタフェース名1, ･･･, インタフェース名n {
  . . .
}
```

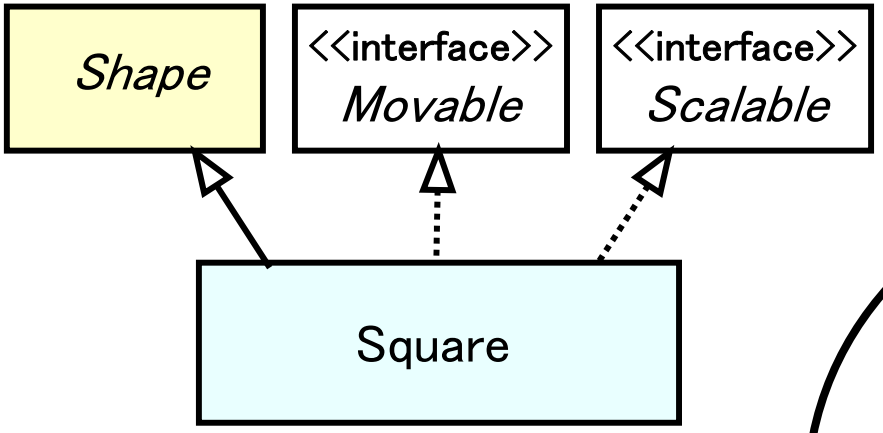
継承を表す　　継承するインタフェース名（複数可）

多重継承の記述例

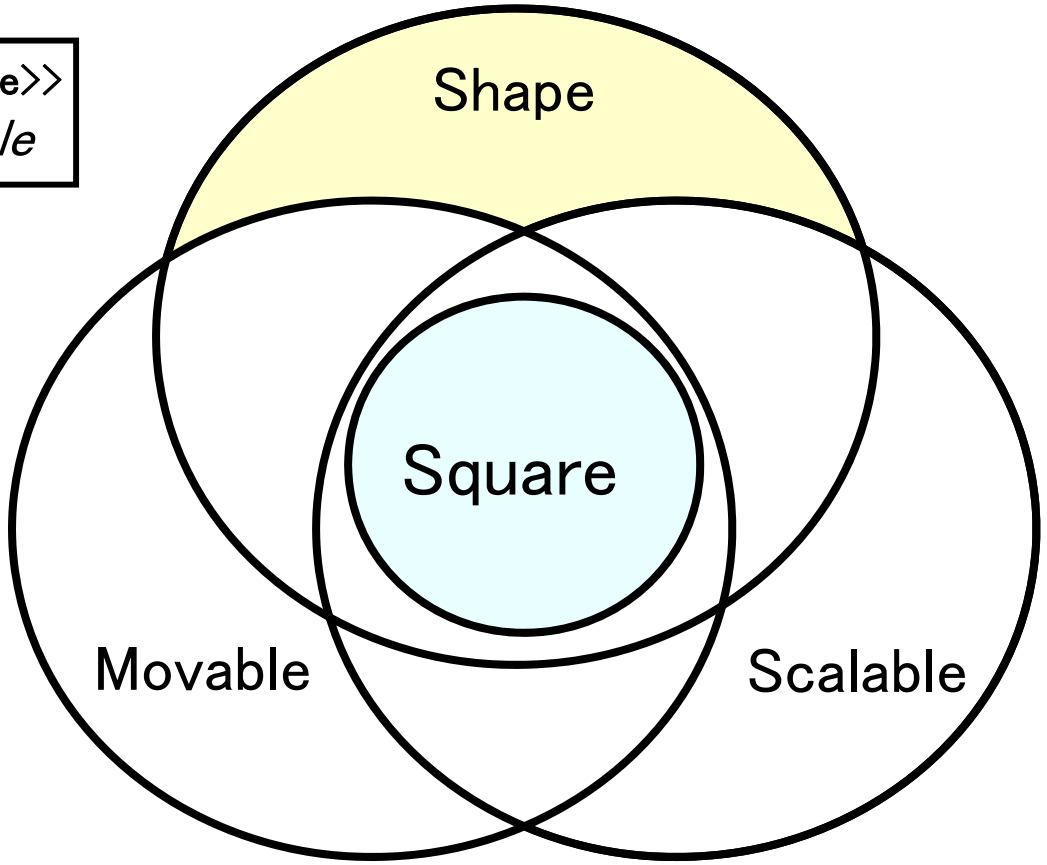
```
// Square.java
public class Square extends Shape
                    implements Movable, Scalable {
  . . .
}
```

インタフェースと型

クラス、インタフェースの継承関係



型の包含関係



インタフェースもクラスと同様の型の規則に従う

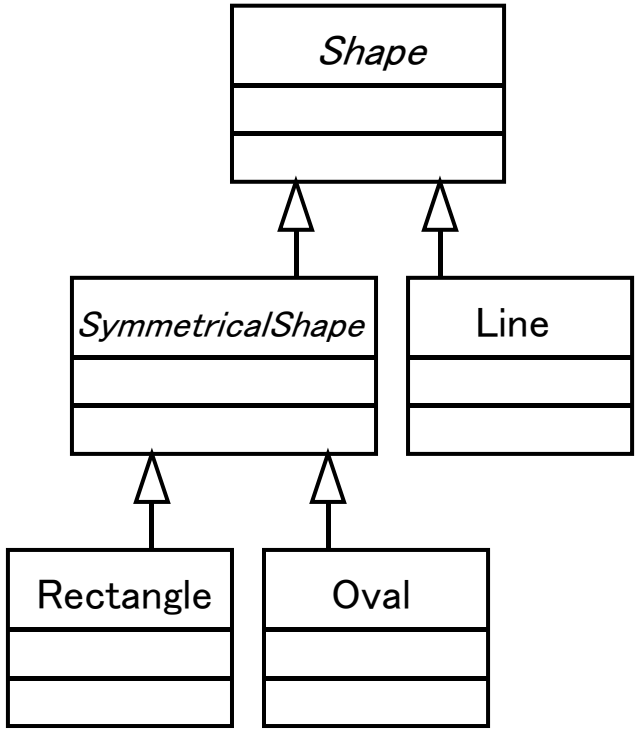
```
Square sq =
       new Square();
Shape sh = sq;
Movable mo = sq;
Scalable sc = sq;
```

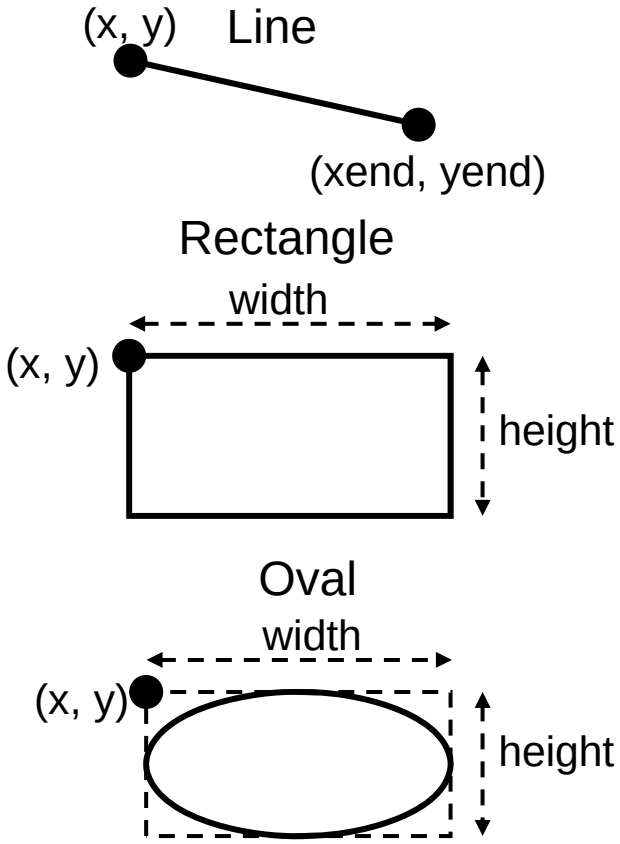
“Square”は“Shape” 、“Movable”、“Scalable”のいずれにも包含される

演習問題(1)

3回かけて、図形エディタを作成する。
今回はその準備として、図形のクラスに編集機能を追加する。
(図形エディタそのものの作成は次回着手)

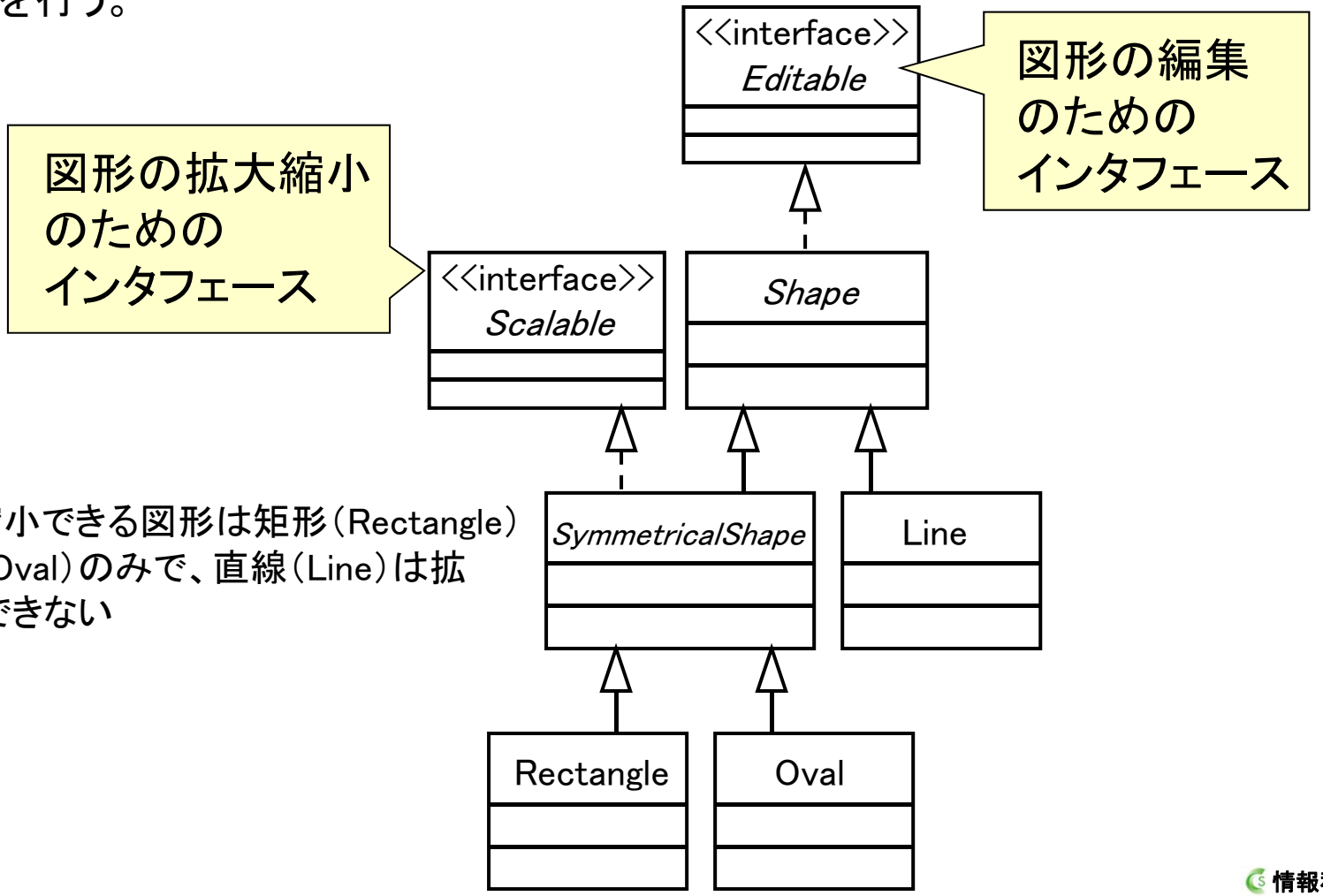
前回使用した、画面表示(のみ)が可能なクラスShape(図形)、Line(直線(線分))、SymmetricalShape(左右対称図形)、Rectangle(矩形)、Oval(円)のソースコードを修正する。





演習問題(2)

図形のクラスに編集機能を追加するため、下図のように、インタフェースEditable、Scalableを継承する。
クラスShape、Line、SymmetricalShape、Rectangle、Ovalに、それらインタフェースで定義されたメソッドを追加する。必要により、属性(フィールド)の追加や、既存メソッドの修正を行う。



注)拡大・縮小できる図形は矩形(Rectangle)と楕円(Oval)のみで、直線(Line)は拡大縮小できない

演習問題(3)

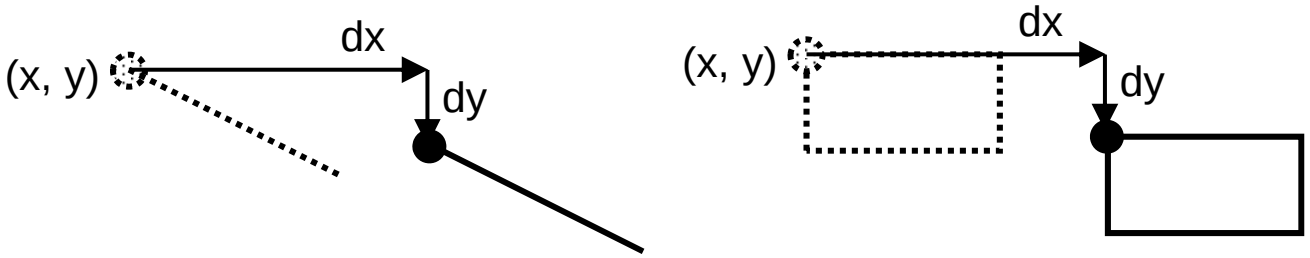
インタフェース“Editable”

```
// Editable.java

// 図形の編集のためのインタフェース
public interface Editable {

  // 図形を移動するメソッド(引数は移動量)
  public void move(int dx, int dy);

  // 図形を移動するメソッド(引数は移動先座標)
  public void moveTo(int xd, int yd);

  // 図形をハイライト表示に設定するメソッド
  public void highlight();

  // ハイライト表示の設定を解除するメソッド
  public void resetHighlight();

  // 図形が指定座標を含むかどうかを示すメソッド
  public boolean contains(int xs, int ys);

}
```

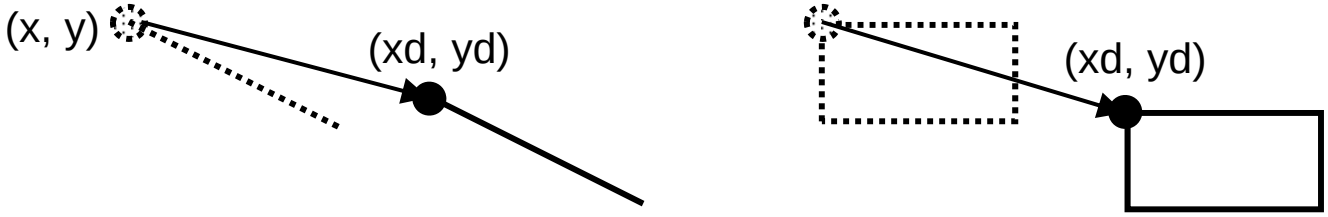
演習問題(4)

インタフェース“Editable”の説明(1)

- メソッド“move(int dx, int dy)”
  - 図形の座標を、(dx, dy)だけ移動する。



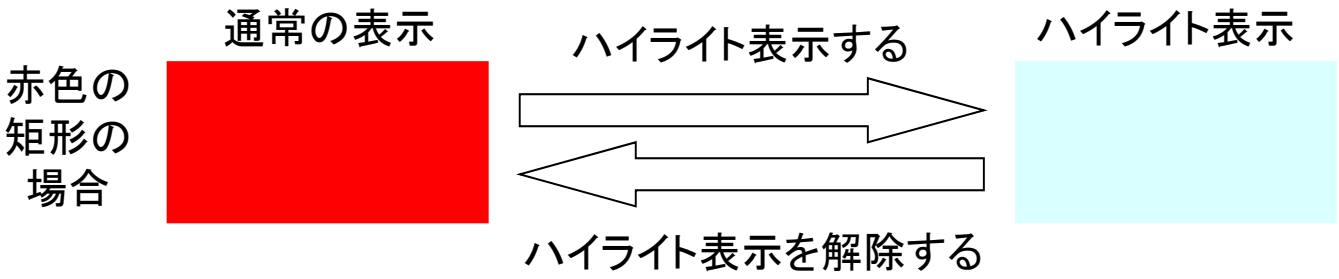
- メソッド“moveTo(int xd, int yd)”
  - 図形の座標を、(xd, yd)に移動する。



演習問題（5）

インタフェース“Editable”の説明（2）

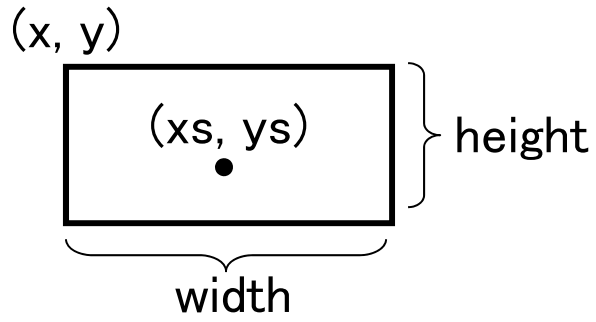
- メソッド“highlight()”、“resetHighlight()”
  - このメソッドは選択した図形を明示するために使用する
    （注：図形を選択する処理自体は、このメソッドには含まない）
  - ハイライト表示では、図形の表示色をシアン（Color.cyan）にする。
  - ハイライト表示するかどうか（ハイライト属性）を記憶するフィールドを設ける。
  - 表示（draw）時に、上記フィールドの値に応じて表示色を変える。
    - 図形本来の色（属性colorの値）を変更するのではなく、ハイライト属性の値によって、一時的に表示を変えるのがよい



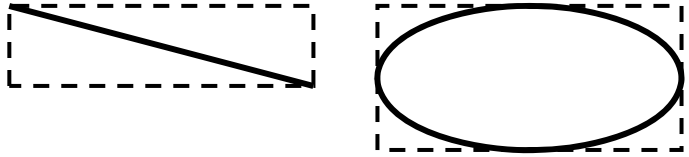
演習問題(6)

インタフェース“Editable”の説明(3)

- メソッド “contains(int xs, int ys)”
  - このメソッドはマウスで図形を選択するときに用いる
    (マウスで指定した座標が図形内に含まれるかどうかで判断する)
    (注:図形を選択する処理自体は、このメソッドには含まない)
    含まれる時は true、含まれない時は false を返す
  - 指定座標 (xs, ys) を含むかどうかは、例えば、以下のようにする。



x <= xs <= x + width
かつ
y <= ys <= y + height
なら、
含まれているとする



簡単のため、線分や楕円の場合は、
外接矩形の範囲内なら含まれている
としてよい

演習問題(7)

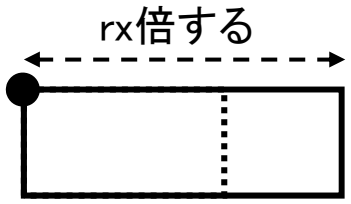
インタフェース“Scalable”

```
// Scalable.java

// 拡大縮小のためのインタフェース
public  interface Scalable {

  // 図形をX方向に拡大・縮小するメソッド
  public void scaleX(double xratio);

  // 図形をY方向に拡大・縮小するメソッド
  public void scaleY(double yratio);

  // 図形を指定座標まで拡大・縮小するメソッド
  public void scaleTo(int xs, int ys);

}
```

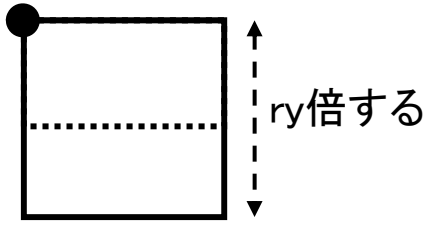
演習問題(8)

インタフェース“Scalable”の説明

- メソッド“scaleX(double rx)”
  - 図形のX方向にrx倍拡大縮小する。

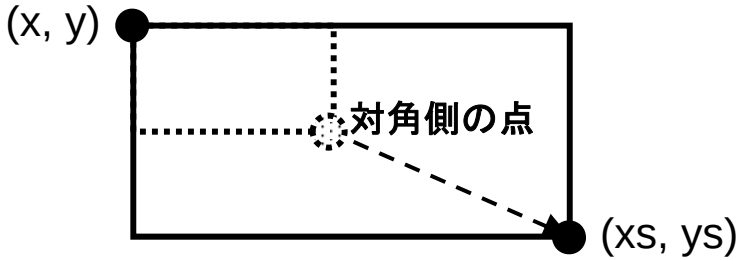


- メソッド“scaleY(double ry)”
  - Y方向にry倍拡大縮小する。

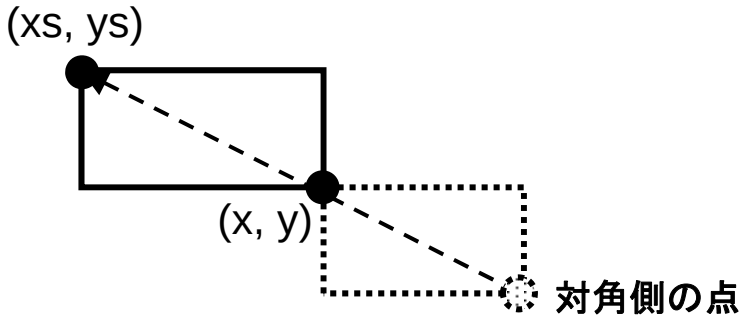


- メソッド“ScaleTo(int xs, int ys)”
  - 図形を座標(xs, ys)まで拡大縮小する。
    (図形の座標の対角側の点が(xs, ys)となるように拡大縮小する)

xs > x, ys > y のとき



xs < x, ys < y のとき



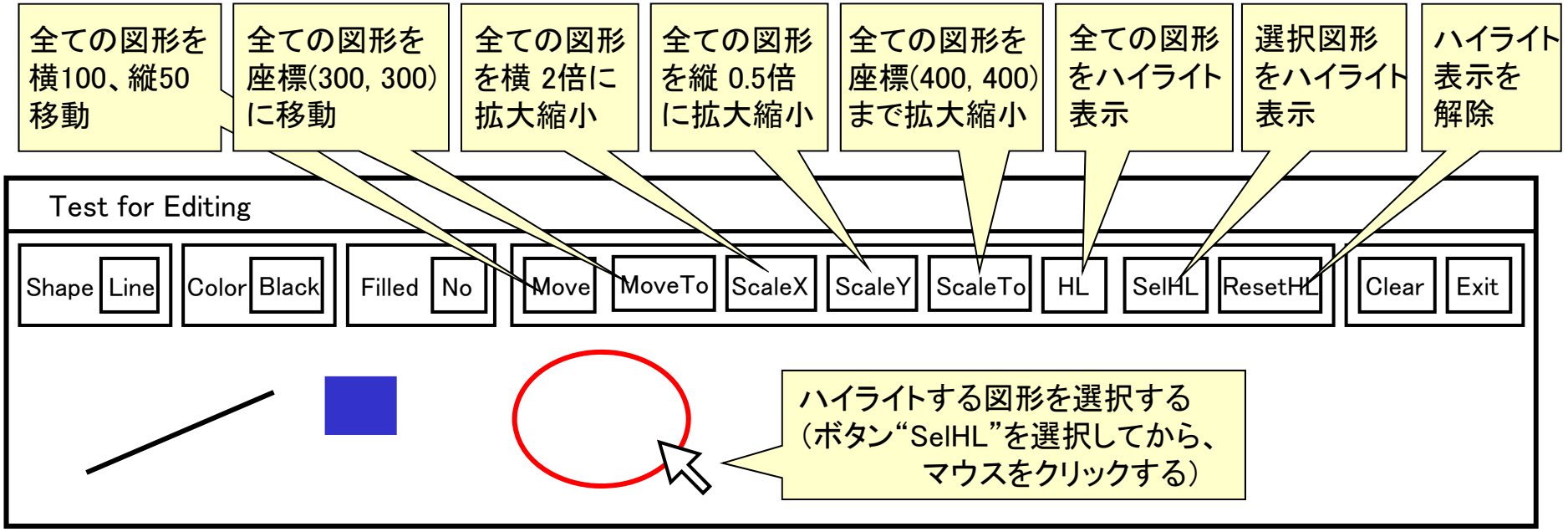
演習問題(9)

動作確認のためのクラス“GTFrame.java”と “EditTest.java”を与える。
また、インタフェース“GPaint.java”と、それを継承するように修正した“PaintPrototype.java”、“PaintTool.java”も与える。
プログラムの起動法は下記。

```
$java GTFrame
```

ペイントツールと同じように図形を描くことができる。
編集テスト用ボタンにより、描いた図形(全部)に対して、移動、拡大縮小、ハイライト表示、ハイライト表示解除のテストが行える。

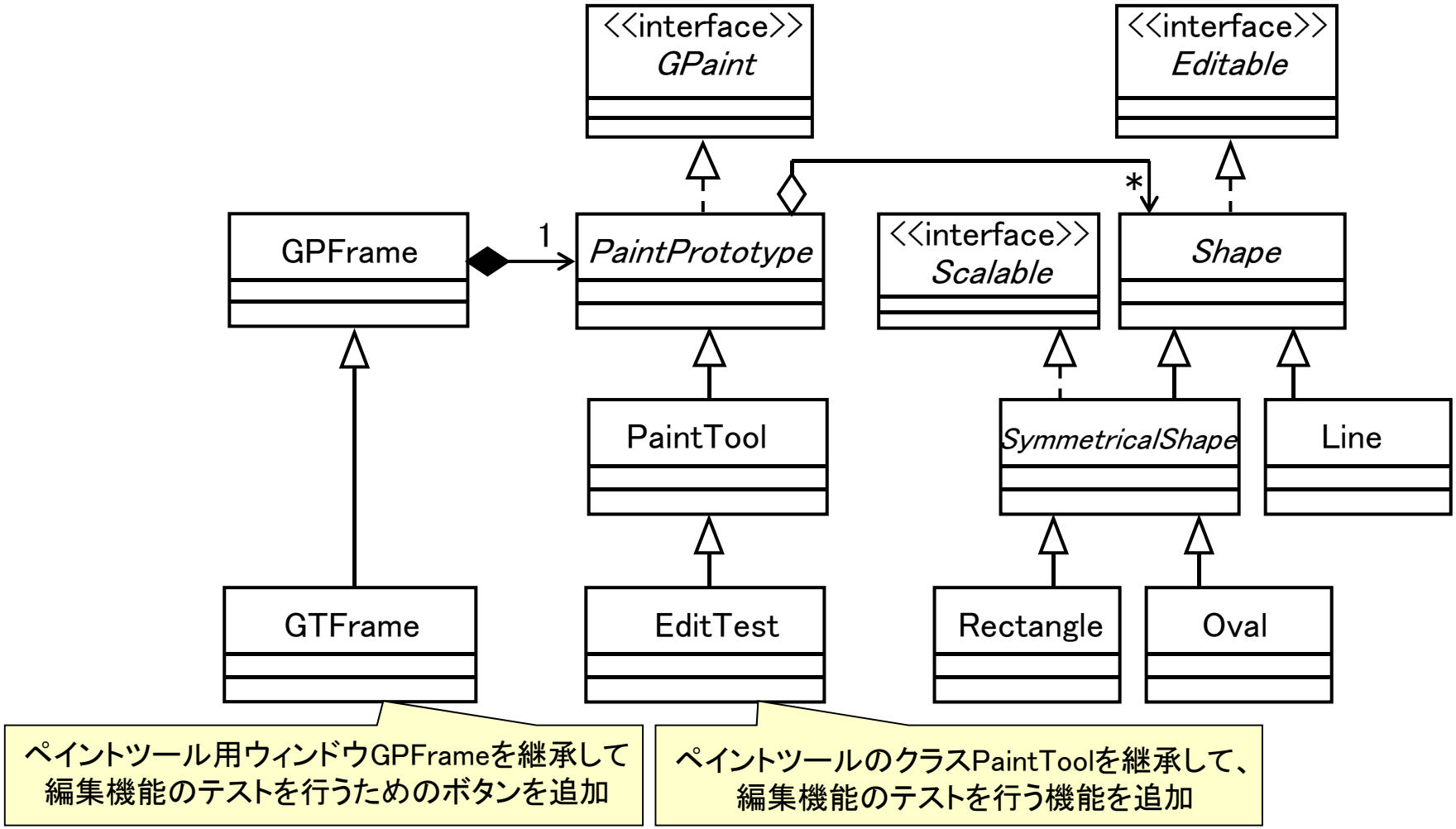
編集機能のテスト用のボタン



“EditTest.java”を修正し、移動量や拡大率を変更してテストすることも有効。

演習問題(10)

編集機能のテスト(動作確認)を行うプログラムのクラス図



作成した“Shape.java”、“Line.java”、“SymmetricalShape.java”、“Rectangle.java”、“Oval.java” を以下により圧縮し、ひとつのファイルとして提出してください。

```
zip src.zip Shape.java Line.java SymmetricalShape.java Rectangle.java Oval.java
```