オブジェクト指向プログラミング

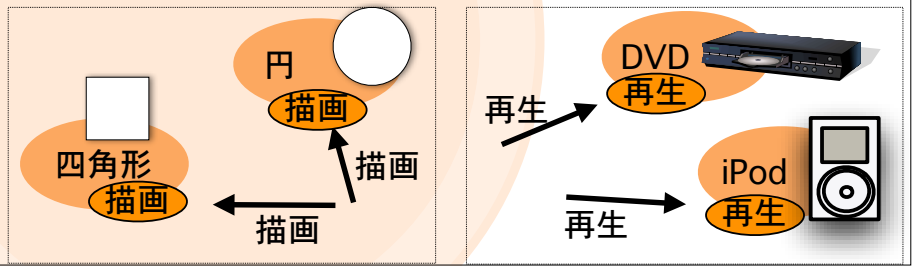
第11回：ポリモーフィズム（インタフェース，抽象クラス）

知識科学教育研究センター
金井 秀明

多態性（多相性）（ポリモーフィズム）

多態性#1

- オブジェクトへのメッセージ（指令）を統一すること．
- 同一メッセージであっても，オブジェクトに応じた振る舞いをするようにすること．



多態性#2

- 同じ名前のメソッドに対して，それぞれ異なる振る舞いを持たせることができる．
- **呼び出す側のソースを全く変更することなく**，プログラムの動作を切り替えることができる．

多態性の実現

- オーバーロード（多重定義）
  - 同一クラス内で同一名のメソッドを複数定義できる.
- オーバーライド
  - スーパークラスで定義されたメソッドと「同じメソッド名・引数の数・型」をもつメソッドをサブクラスで定義できる.

オーバロード

- 「同じクラスに，同じ名前で引数の方が異なるメソッドを定義すること」をメソッドのオーバロード（多重定義）という

```
int add(int a, int b) {...};
double add(double a, double b){...};
```

メソッドのシグネチャ

- Javaでは，オーバロードされたメソッドから，必要なメソッドを見分ける方法として，メソッドのシグネチャを利用する.
  - メソッドのシグネチャとは，メソッドの「名前」，「引数の並び（型，個数）」のこと

```
int add(int a, int b) {...};
double add(double a, double b){...};
```

例

```
class Circle {
  double x; // 中心のx座標値
  double y; // 中心のy座標値
  double radius; //  半径

  void set(double tx, double ty){
    x = tx;
    y = ty;
  }

  void set(double tx, double ty, double r) {
    x = tx;
    y = ty;
    radius = r;
  }
}

//
Circle c1 = new Circle();
c1.set(2.0, 3.0);
C1.set(3.0, 1.0, 3.0);
```

抽象クラス

抽象クラス#1

- 概念としてのみ存在するクラス
- abstractでクラスを修飾：抽象クラス
  - オブジェクトを作成できないクラス
- abstractでメソッドを修飾
  - 処理内容が定義されていないメソッドを持っている．（抽象メソッド）
- abstractで変数を修飾できない

抽象クラス

- 継承されることを前提に定義されたクラス
  - 通常のクラス同様に，フィールドやメソッドを定義てきる．
  - 直接インスタンスが作れない（newできない）．抽象クラスを継承し，サブクラスでインスタンスを作成する．
  - 通常メソッド定義の他に，抽象メソッドを定義できる．

```
abstract class クラス名 {
    クラスの内容
}
```

抽象メソッド＃１

- サブクラスでオーバライドされることを前提としたメソッド

```
abstract 戻り値の型 メソッド名(引数);
```

  - 通常のメソッドとの違いは，「abstractキーワードがつくこと」，「メソッド本体がない」こと．

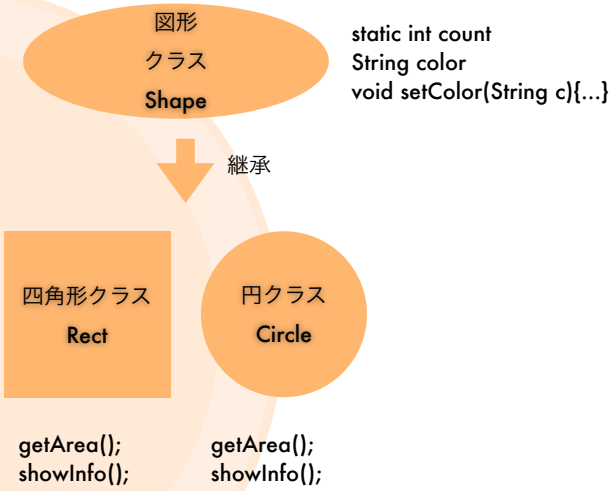
抽象メソッド＃2

- メソッド本体は，抽象クラスを継承したサブクラス内でオーバライドして実装する．
  - メソッドの形式（名前と引数）だけ決め，メソッドの処理内容はサブクラスで定義する．
  - サブクラスで抽象メソッドを実装しない場合，コンパイラエラーがでる．→「オーバライド忘れ」の防止になる．

例：図形クラス#1

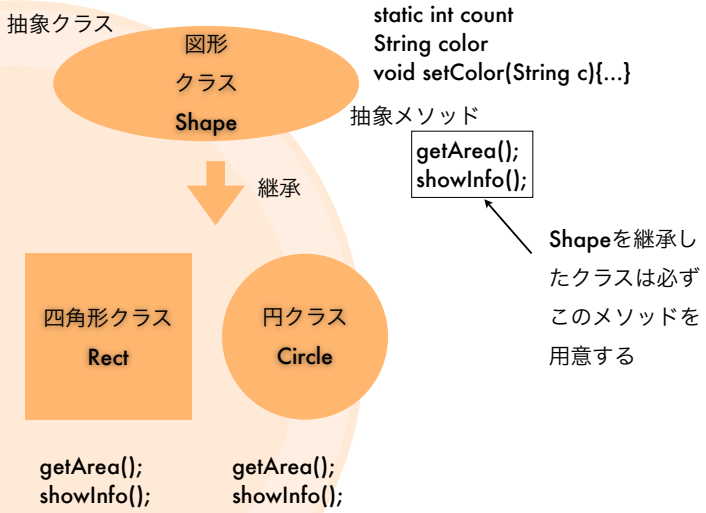
- 図形を管理する「図形クラス」を考える．
  - 図形クラスのサブクラスのインスタンスをいくつ生成したかを保存する変数 count
  - 図形クラスのサブクラス（各種の図形）は，図形の面積を計算するメソッド getArea()
  - 面積を画面に表示するメソッド showInfo()

例： 図形クラス#2



Shape1.java

例： 図形クラス#3



AbShape1.java

例

```
abstract class AbShape {
  static int count = 0;
  String color;

  void setColor(String c){color=c;}

  abstract double getArea();
  abstract void showInfo();
}
```

例

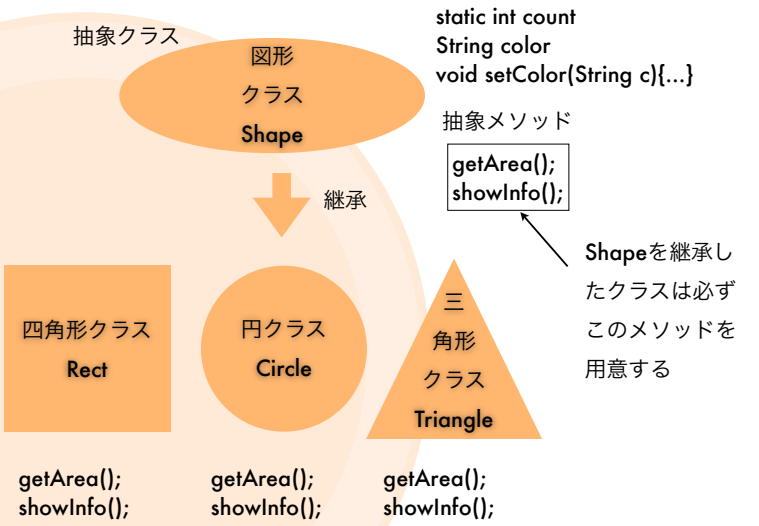
```
class AbRect extends AbShape {
  double width, height;

  AbRect(double w, double h, String c) {
    this.width=w; this.height=h;
    count++;
  }
  public double getArea() {
    return width*height;
  }
  ...
}
```

抽象クラスAbShapeで定義された
抽象メソッドgetArea()の実装

練習1

- 抽象クラスAbShapeを継承するクラス「三角形」を定義せよ.
  - 三角形クラスはフィールドとして，底辺，高さを持つ.

練習1のイメージ



abstract_ren1.java

例:

```
class rei_abstract {
  public static  void main(String  args[]) {
    Shape shapes[] = new Shape[4];
    shapes[0] = new Circle(1.0);
    shapes[1] = new Circle(2.0);
    shapes[2] = new Rect(3.0, 1.0);
    shapes[3] = new Rect(5.0, 20.0);
    for(int i=0; i < shapes.length; i++) {
      shapes[i].getArea();
    }
  }
}
```

例:つづき

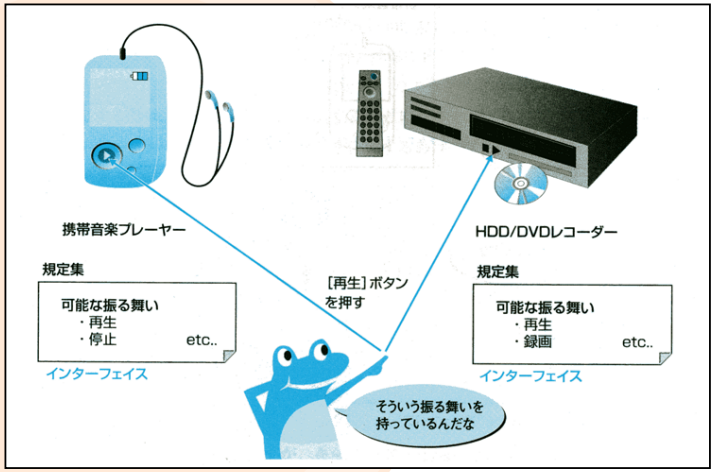
```
class Circle extends Shape {
  private double radius;
  public Circle(double radius) {
    this.radius = radius;
  }
  public void getArea(){
    System.out.println("半径"+radius+"の円の面積
    は"+radius*radius*3.14+"です。");
  }
}

class Rect extends Shape {
  private double width;  private double height;
  public Rect(double width, double height) {
    this.width = width; this.height = height;
  }
  public void getArea(){
    System.out.println("幅"+width+" 高さ"+height+"の長方形の面積
    は"+width*height+"です。");
  }
}
```

インタフェース

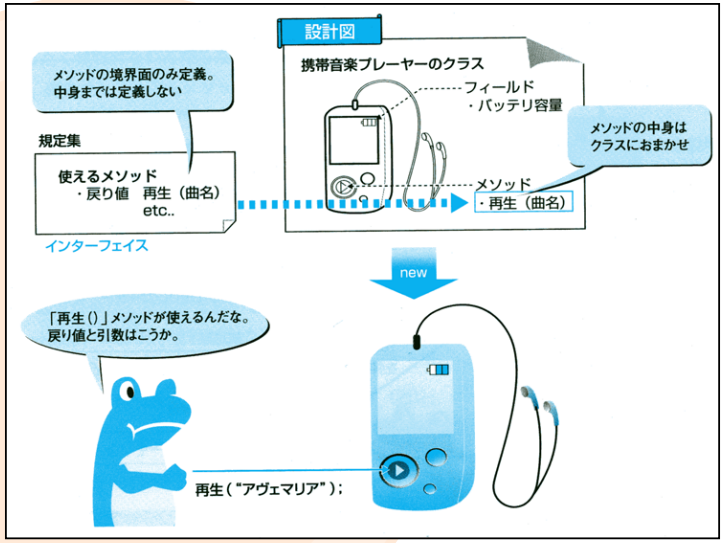
インタフェース？

インタフェース？

- オブジェクトの持つ振る舞いを定めた規定集



インタフェース？



インタフェース#1

- このクラスはどのような機能（メソッド）を持っているかを」を定義する仕組み
- つまり，”規定集”のようなもの．

```
interface インタフェース名 {
    フィールド定義(定数のみ)
    メソッドの定義(シグネチャのみ)
}
```

「abstract」キーワードを付けていない抽象メソッド（ただし，付けても問題はない）

インタフェース＃２

- インタフェースを使うには，インタフェースを「実装（implements）」するクラスを定義する必要がある．
- そのようなクラスは以下のように定義する．

```
class クラス名 implements インタフェース名 {
    メソッドの実装
    …
}
```

通常のクラス定義に「implements インタフェース名」を追加することで，インタフェースを実装したクラスを定義する．

例1

- 規定
  - 図形Shapeインタフェースには，図形の面積を求めるgetArea抽象メソッドが定義されている．

```
interface Shape {
   public double getArea();
}
```

  - 長方形Rectクラス，円CircleクラスはそれぞれgetAreaを実装する．

Rectクラス

```
public double getArea() {
   return width*height;
}
```

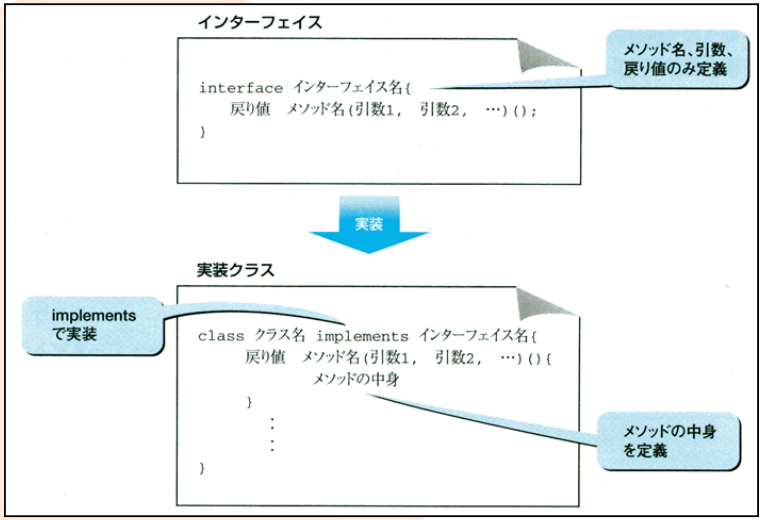
Circleクラス

```
public double getArea() {
   return radius*radius*3.14;
}
```

練習1

- 例１のShapeインタフェースに，図形の外周の長さを求める抽象メソッドgetCircumferenceを定義せよ．
- Rectクラス，Circleクラスにその実装を追加せよ．

インタフェースと実装



インタフェース#3

- インタフェースを実装したクラスは，以下の性質をもつ．
  - メソッドの実装の強制
  - インタフェースの代入互換性（インタフェース型変数）
  - 同時に複数のインタフェースを実装可能（多重継承）

メソッドの実装の強制

- インタフェースを実装したクラスは，必ずインタフェースで定義されているメソッドをオーバライドして，実装しなければならない．

```
interface Shape{
   public double getArea();
  ...
```

```
class Rect implements Shape {
   public double getArea(){
     return width*height;
   }
```

getArea()を実装しないとコンパイラーエラーになる．

インタフェース型変数

- インタフェースを実装したクラスのインスタンスは，インタフェース型の変数に代入できる．
- その変数のメソッドを呼び出すことで，インスタンスのメソッドを呼び出せる．

```
Shape s1 = new Rect(10.0, 5.0);
Shape s2 = new Circle(3.0);
```

```
s1.getAre();
s2.getAre();
```

```
Shape s = new Shape[2];
s[0]=new Rect(10.0, 5.0);
s[1]=new Circle(3.0);
```

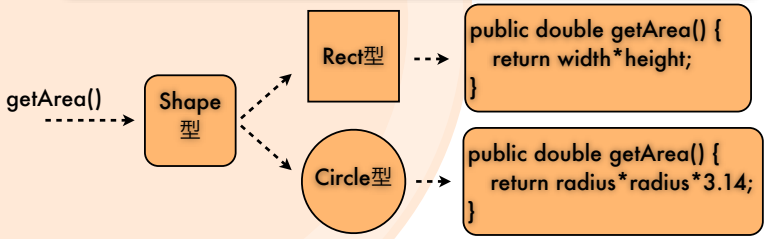
```
for(int i=0;i<s.length;i++) {
 s[i].getArea();
 }
```

多態性#1

- 同じ操作で，異なる動作をさせることができる（ポリモーフィズム：多態性）を実現する．
  - プログラム実行時に，動的に呼びさされるメソッドが切り替わる．（=動的ディスパッチ）
  - インスタンスの型に応じて，呼ばれるメソッドが切り替わる．

多態性#2

```
Class IFSmaple3 {
   public static void main(String[] args) {
      showArea(new Rect(10.0,5.0));
      showArea(new Circle(3.0));
}
   static void showArea(Shape shape) {
      System.out.println(shape.getArea());
   }
```



多態性#3-1

- 実は，if文でも同じことはできる．

多態性

```
Shape shape = new Rect(10.0,5.0);
int area = shape.getArea();
```

if文

```
Shape shape = new Rect(10.0,5.0);
int area;
if(shape instanceof Rect) {
  Rect rect = (Rect) shape;
  area = rect.getArea();
} else if (shape instanceof Circle) {
  Circle circle = (Circle) shape;
  area = circle.getArea();
}...
```

ある変数が何型やどのクラスから生成されたかがわかる．

多態性#3-2

- ただし，新たにクラスを追加した場合には，if文で実装した場合には，else文を追加しなければならない．

instanceof 演算子

- ある変数がどの型やクラスを元に生成されたのかが知りたいときに使う．

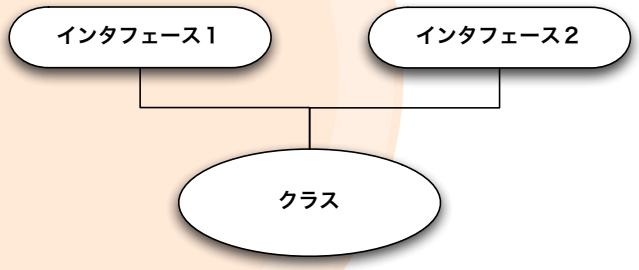
```
<変数> instanceof <型／クラス>
```

多重継承#1

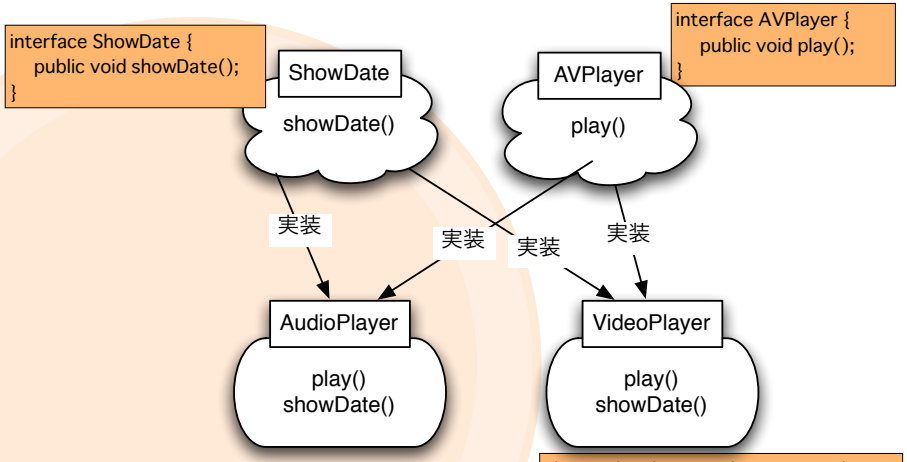
- 多重継承：２つ以上のクラスを継承すること．
- Javaの継承「extends」では多重継承は認めていない．
- ただし，interfaceを使うことで，擬似的に多重継承を行うことができる．

多重継承#2

```
class <クラス名> implements <インタフェース1>,
<インタフェース2>, …{
        …;
}
```



例3



練習2

- 例１のShapeインタフェースを実装するクラスとして，台形Triangleクラスを追加せよ．
  - 三角形の面積=下辺*高さ/2
- 「多態性#2」のようにTriangleクラスを含めて多態性を実行し確認せよ．

練習3

- 練習２で追加したTriangleクラスを含めて，「多態性#3-1」のようにif文で多態性を実現せよ．

インタフェースの利点

- 継承を使わないですむ
  - 継承が多い＝＞クラス間の依存性があがる
- メソッドのオーバライドのし忘れを防止できる.
- 多重継承

クラスとインタフェース

	クラス	インタフェース
オブジェクト	できる	できない
メソッド	いろいろ	必ずabstract
フィールド	いろいろ	必ずpublic static final
スーパクラス	1つだけ	持てない
スーパインタフェース	複数可能 (implements)	複数可能 (extends)