



# 発展プログラミング演習II

## 7. オブジェクト指向 継承

---

玉田 春昭 水口 充



# オブジェクト指向の基礎

---

- クラス
- インターフェース
- メソッド
- フィールド
- 継承
- アクセス制御
- 多態性
- 例外

# 作りたいソフトウェアをどうやって作る？

既存の似たソフトウェア  
を修正して作った方が  
簡単だし、楽.



既存のソフトウェア

作りたいソフトウェア



全くの1から  
作成する？



開発者



# 例題7.1 ゴールドメンバカードの作成

- MemberCard.javaをMoodleからダウンロードする。
- MemberCard.javaと同じディレクトリで右の2つのプログラムを作成する。

```
class GoldMemberCard extends MemberCard{
    private static int numberOfGoldMember = 0;
    private int goldNumber;
    public GoldMemberCard(String name,
                           int initPoints){
        super(name, initPoints);
        numberOfGoldMember++;
        goldNumber = numberOfGoldMember;
    }

    public int getGoldNumber(){
        return goldNumber;
    }
}
```

```
public class Example0701{
    public static void main(String[] args){
        GoldMemberCard g1 =
            new GoldMemberCard("玉田", 1000);
        String name1 = g1.getName();
        System.out.println("会員氏名は"
                           + name1 + "です。");
    }
}
```

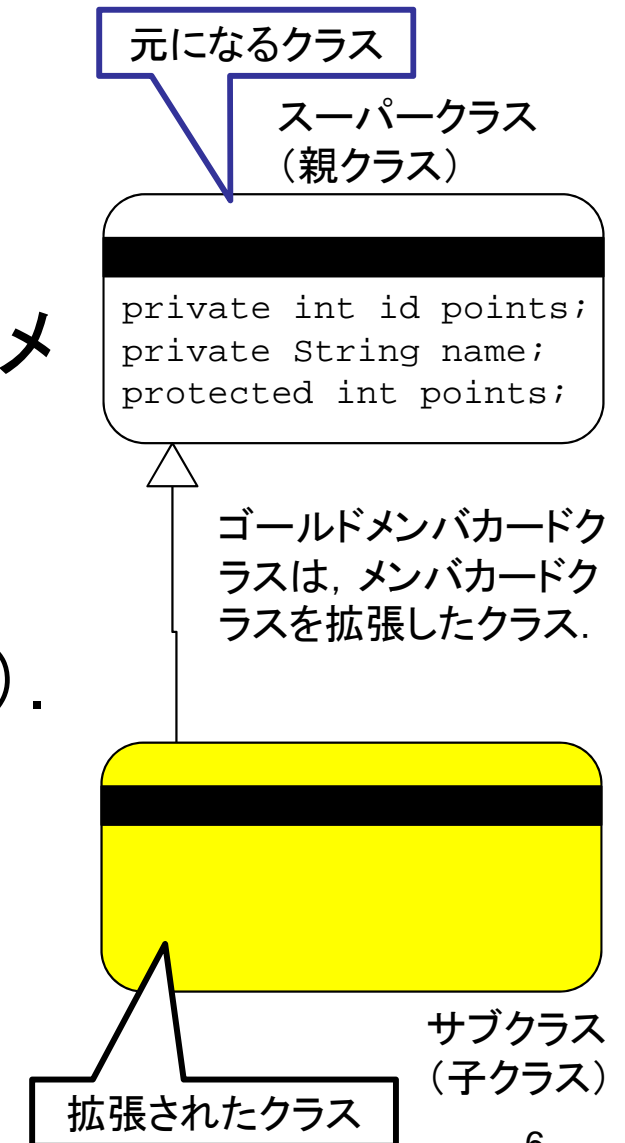


# なぜ拡張する？

- メンバカードのソースコードをコピーして新たなゴールドカードを作成したら良いのでは？
  - もとのメンバカード内にバグが見つければメンバカードとゴールドカードの両方を修正しなければならない。
    - コピペして作ったプログラムがいっぱいあればそれだけ修正コストが増す。
- コピペせずに書き直せば良いのでは？
  - メンバカードとゴールドメンバカードの両方を扱えない。
- その解決策として継承 (Inheritance)がある。
  - 元のクラスからの差分のみを書いて新たなクラスを作成する。

# 継承 (Inheritance)

- ゴールドメンバーカードはメンバーカードを継承している。
  - メンバーカードのメンバ(フィールド+メソッド)を全て持つ.
  - メンバーカードとして扱える.
  - 独自の拡張がある(かもしれない).





# 継承 (インヘリタンス; inheritance)

---

- あるクラスを継承してできたクラス (サブクラス) は以下の特徴を持つ.
  - 親クラスと同じメソッドを持つ.
    - 明示的に書かなくても良い.
  - 同じインターフェースを実装する.
    - 明示的に書かなくても良い.
  - 新たにメソッドを定義すれば, そのメソッドが加わる.
    - 親クラスには含まれないメソッドを持つ.
  - 新たにフィールドを定義すれば, そのフィールドが加わる.
    - 親クラスには含まれないフィールドを持つ.
- 追加したい部分だけ書き足せば良い.
  - 修正が非常に簡単になる.
    - コピー&ペーストで同じコード断片がいろんなところにあると, 修正が難しい.



# オーバーライド

---

オーバーロードと区別しよう.

- メンバカード
  - ポイントの上限は100,000ポイント.
- ゴールドメンバカード
  - ポイントの追加時に5%のボーナスが付く.
  - ポイントの上限なし.
- ポイントの追加はaddPointsメソッドで行う.
  - メンバカードにもゴールドメンバカードにもaddPointsメソッドを追加する.
  - ゴールドメンバカードのaddPointsメソッドにより、親クラスであるメンバカードのaddPointsメソッドは隠される.



# 例題7.2 オーバーライド

- GoldMemberCardにaddPointsメソッドを追加せよ.
  - MemberCardにint型のtotalPointsが定義されている.
  - 継承されたGoldMemberCardにも定義されているものとして扱える.
  - totalPointsはprotectedなので、継承されたクラスからも見える.

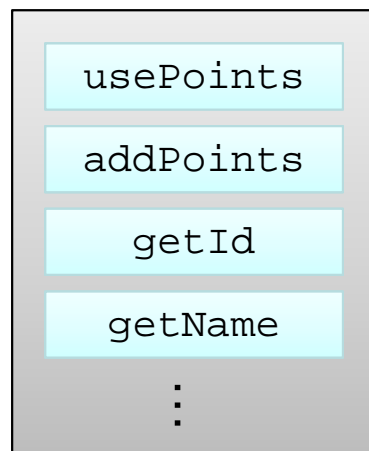
```
public void addPoints(int points){
    // pointsの5%を計算してbonusに代入する.
    // 小数点以下切り捨て(int型にキャストする).
    // pointsとbonusをtotalPointsに足す.
    System.out.println(points +
        "ポイントを追加しました(内ボーナスポイント "
        + bonus + "ポイント)");
}
```

```
public class Example0702{
    public static void main(String[] args){
        GoldMemberCard g1 =
            new GoldMemberCard("玉田", 1000);
        String name1 = g1.getName();
        System.out.println("会員氏名は"
            + name1 + "です.");
        // 適当にポイントを増減させる.
    }
}
```

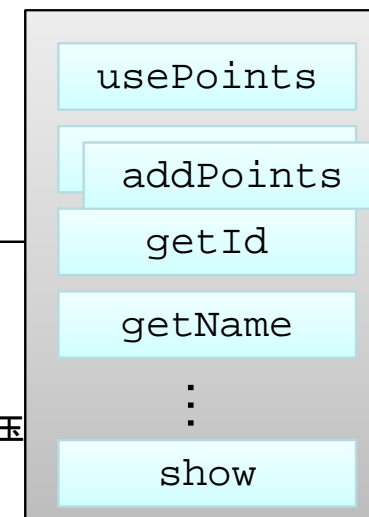
# メソッドのオーバーライド (1/2)

- addPointsメソッドはMemberCardクラスとGoldMemberCardクラスの両方に定義されている。
  - addPointsメソッドはサブクラス (GoldMemberCard) で新たに定義しなおされた。
  - これをメソッドの**オーバーライド(Override)**もしくは**上書き**と呼ぶ。

MemberCardクラス




GoldMemberCardクラス



GoldMemberCardクラス  
がMemberCardクラスを  
継承した。

# メソッドのオーバーライド (2/2)

- GoldMemberCardオブジェクトのaddPointsメソッドを呼ぶと, GoldMemberCardクラスで新しく定義されたaddPointsメソッドが実行される.
  - MemberCardクラスで定義されているaddPointsメソッドは実行されない.
  - どうしてもMemberCardクラスのaddPointsメソッドを実行したい場合は**super**を介して呼び出す.
- `super.addPoints(points);`
  - `super`はスーパークラスのメソッドを呼ぶ.
    - 上書きされる前のメソッドを呼ぶことができる.
    - ただし, サブクラスからしか呼ぶことができない.
  - `this`は同じオブジェクトのメソッドを呼ぶときに用いる.



# オーバーライドとオーバーロード

---

- オーバーライド (Override)
  - スーパークラスで定義されているメソッドを上書きして、独自のメソッドを定義すること。
- オーバーロード (Overload)
  - 同じ名前のメソッドをシグネチャを変えて複数個定義すること。
- オーバーロードされたメソッドもオーバーライドできる。
  - ただし、オーバーライドされるのはシグネチャが一致するメソッドだけ。
    - メソッドの名前と引数の型や個数が一致しなければオーバーライドではない。

# 継承に関する規則

- 明示的にスーパークラスを指定しなかった場合, Object クラスがスーパークラスとなる.
  - String クラスと同様に, Java 言語に標準で定義されているクラス.
- クラスはスーパークラスのすべてのメソッド, フィールド, インターフェースを持つ.
  - スーパークラスがさらにスーパークラスを持っていたとしても, 全てのスーパークラスが持つメソッド, フィールドを持つ.
- Java では, 直接のスーパークラスは一つだけ.
  - つまり, extends の後ろに書けるのは1つのクラスだけ.
  - 他の言語では, 複数のスーパークラスを指定できる場合がある.
    - そのような言語でも複数のスーパークラスを持つことは, 推奨されていない.

# サブクラスのコンストラクタ

- コンストラクタ

- new演算子によってオブジェクトが作成されたときに暗黙のうちに呼び出される特別なメソッド.

- サブクラスのコンストラクタの最初の行は**必ず**スーパークラスで定義されたコンストラクタの呼び出し or 他のコンストラクタの呼び出しでなければならない.

```
public Point{  
    super();  
    x = 10;  
}
```

スーパークラスのコンストラクタの呼び出し. ただし, コンストラクタに渡す引数があれば**省略可能**.

コンストラクタのサンプル

# コンストラクタ

- 全てのクラスはコンストラクタが定義されていなければならない。
  - 明示的にコンストラクタを定義しなければ、引数なしのコンストラクタが自動的に定義される。

```
public class Point{  
    public int x, y;  
}
```

実際に上のように書いたとしても、コンパイルすると右のようになる。

```
public class Point{  
    public int x, y;  
    public Point(){  
        super();  
    }  
}
```

extendsがないのに、  
super();の呼び出しをしている。  
何を呼び出している？

Java言語の場合、明示的な  
スーパークラスの指定がな  
ければObjectクラスがスー  
パークラスになる。

# 例題7.3 コンストラクタの オーバーロード

- 右のプログラムを書き, 実行結果を確かめよ.
- コンストラクタもオーバーロード可能.
  - 引数の個数や型が異なるコンストラクタを定義できる.

```
public class Point3D{
    private int x, y, z;

    public Point3D(){
        x = y = z = 0;
    }

    public Point3D(int x0, int y0){
        x = x0;
        y = y0;
        z = 0;
    }

    public static void main(String[] args){
        Point3D p = new Point3D();
        Point3D q = new Point3D(3, 4);
        System.out.println("p.x = " + p.x);
        System.out.println("q.x = " + q.x);
    }
}
```



# 課題7.1 Stackの作成


- Moodleに掲載されているLinkedListクラスを継承してStack\_学生証番号クラス(LIFO)を作成せよ.
  - popメソッドとpushメソッドを作成せよ.
  - popメソッドは一番最後に追加した値(String型)を返す.
  - pushメソッドはString型を受け取り、リストに追加する.
- 右のプログラムが正常に動作するようにせよ.
  - コマンドラインで与えた引数を与えた順とは逆順に表示される.
- 前提として、同じ値が追加されることはないこととする.

```
public class StackTest{
    public void run(String[] args){
        Stack_学生証番号 stack =
            new Stack_学生証番号();
        for(int i = 0; i < args.length; i++){
            stack.push(args[i]);
        }
        while(stack.getSize() != 0){
            String value = stack.pop();
            System.out.println(value);
        }
    }
    public static void main(String[] args){
        StackTest test = new StackTest();
        test.run(args);
    }
}
```

# 課題7.2 Queueの作成

- Moodleに掲載されているLinkedListクラスを継承してQueue\_学生証番号クラス(FIFO)を作成せよ.
  - enqueueメソッドとdequeueメソッドを作成せよ.
  - enqueueメソッドはString型を受け取り, リストに追加する.
  - dequeueメソッドはリストの一番最初の要素を返し, その値をリストから削除する.
- 右のプログラムが正常に動作するようにせよ.
  - コマンドラインで与えた引数が与えた順に表示される.
- 前提として, 同じ値が追加されることはないこととする.

```
public class QueueTest{
    public void run(String[] args){
        Queue_学生証番号 queue =
            new Queue_学生証番号();
        for(int i = 0; i < args.length; i++){
            queue.enqueue(args[i]);
        }
        while(queue.getSize() != 0){
            String value = queue.dequeue();
            System.out.println(value);
        }
    }
    public static void main(String[] args){
        QueueTest test = new QueueTest();
        test.run(args);
    }
}
```



# 課題7.1, 7.2について

---

- 提出物
  - 課題7.1: Stack\_学生証番号.java
  - 課題7.2: Queue\_学生証番号.java
- 提出場所
  - Moodleの「課題7.1: Stack」, 「課題7.2: Queue」
- ✕ 切
  - 2012/11/22(木) 9:00

# 継承と多態性(ポリモルフィズム; Polymorphism)

- スーパークラスとサブクラスの関係はis-a関係.
  - GoldMemberCard is a MemberCard.
    - 逆は成り立たない.
  - スーパークラスの型の変数にサブクラスのオブジェクトの参照を代入できる.
  - しかし, サブクラスで新たに定義されたメソッド, フィールドは呼び出し, 参照不可.
    - getGoldNumberメソッドはGoldMemberCard型の変数に対してしか呼び出せない.

# 例題7.4 ポリモルフィズム

- 以下のプログラムを作成して、以下のことを確認せよ.
  - GoldMemberCardオブジェクトがスーパークラスのMemberCard型の変数に代入できること.
  - 最後の3つのaddPointsメソッドがそれぞれMemberCard, GoldMemberCardのどちらのaddPointsを呼び出しているか.

```
public class Example704{
    public void run(){
        GoldMemberCard card1 = new GoldMemberCard("玉田", 1000);
        MemberCard      card2 = new GoldMemberCard("水口", 1500);
        MemberCard      card3 = new MemberCard("田中", 500);

        card1.addPoints(100);
        card2.addPoints(100);
        card3.addPoints(100);
    }
    // mainメソッドは省略.
}
```

# メソッドディスパッチ (Method Dispatch)

- オーバーライドされているメソッドが呼び出されるとき、変数が表すオブジェクトのクラスに応じて異なるメソッドが実行されること。
- 変数の型に関わらず、表すオブジェクトが持つメソッドが呼び出される。
  - 呼び出し元のオブジェクトも関係ない。

	変数の型	表すオブジェクト	addPointsメソッド
変数card1	GoldMemberCard	GoldMemberCard	GoldMemberCard
変数card2	MemberCard	GoldMemberCard	GoldMemberCard
変数card3	MemberCard	MemberCard	MemberCard

# instanceof演算子

- オブジェクトを表す変数や引数, フィールドが実際にどのクラスのオブジェクトを表しているか調べたいときに用いる演算子.

式 instanceof クラス名

- クラス名の部分はインターフェース名でも可.
- クラス名が表す型の変数に, 演算子の左側の「式」が代入可能ならtrue, そうでないならfalse.
- instanceofの計算結果はすべてboolean型.

# abstract 修飾子

- クラスに付けた場合
  - 抽象メソッドを持つクラスは必ず抽象クラスとなる.
  - クラスのオブジェクトが作成できない **抽象クラス**となる.
    - 一部のメソッドの中身が定義されていないため.
- メソッドに付けた場合
  - メソッドの中身を持たない **抽象メソッド**となる.
    - メソッドの中身はサブクラスで実装されることを期待している.
  - アクセス修飾子は `private` 以外になる.



# 例題7.5 抽象クラス

- 継承されることを前提として作成されたクラス。
  - － オブジェクトを作成できない。
- 以下のCardクラスを作成せよ。
  - － Cardクラス内にmainメソッドを作成し、Cardクラスのオブジェクトが作成できないことを確認せよ。

```
public abstract class Card{
    private static int numberOfMembers;
    private int id;
    private String name;
    protected int points;

    public Card(String name, int initPoints){
        numberOfMembers++;
        id = numberOfMembers;
        this.name = name;
        this.points = initPoints;
    }
    // フィールドのgetterを用意する.

    public abstract void show();
}
```

# 抽象クラス (Abstractクラス)

- abstractをメソッドに付けると**抽象メソッド** (abstract method)となり、メソッドの本体は定義できない。
- 抽象メソッドを1つでも持つクラスは**抽象クラス** (abstract class)となる。
- 今まで書いてきたクラスは**具象クラス** (Concrete class)である。

```
public abstract class Card{
    private static int numberOfMembers;
    private int id;
    private String name;
    protected int points;

    public Card(String name, int initPoints){
        numberOfMembers++;
        id = numberOfMembers;
        this.name = name;
        this.points = initPoints;
    }
    // フィールドのgetterを用意する.

    public abstract void show();
}
```



# 例題7.6 抽象クラスを継承する.

---

- Cardクラスを継承してPointCardクラスを作成する.
  - showメソッドをオーバーライドし, id, name, pointsを出力せよ.
  - 出力例
    - id: 1, name: 玉田, points: 100
- 注: Cardクラスで, id, nameフィールドはprivate宣言されている.
  - どのようにアクセスするか考えること.

# アクセス権

- `private`宣言されているフィールド, メソッドはそのクラス自身からしかアクセスできない.
- `protected`宣言されているフィールド, メソッドはサブクラスからもアクセス可能.
- `public`宣言されているフィールド, メソッドはどこからでもアクセス可能

	public	protected	なし	private
クラス自身	○	○	○	○
同一パッケージ	○	○	○	×
サブクラス	○	○	×	×
その他	○	×	×	×

## 課題7.3 図形 (1/3)

- 抽象クラス Shape を作成せよ。
  - 以下のものを定義せよ。
    - 図形の形を表すフィールド: String型のtypeフィールドとそのgetter.
    - 面積を返す抽象メソッド: getArea. 引数はなし, 戻り値はdouble型.
    - コンストラクタはtypeを受け取る.
- Shapeクラスを継承する3つの具象クラスを作成せよ。
  - Rectangle(四角形)クラス, Triangle(三角形)クラス, Trapezoid(台形)クラス.
  - それぞれ, コンストラクタで面積を求めるために必要な情報を受け取る. それぞれ, double型とする.
    - Rectangle: width, height.
    - Traiangle: baseWidth, height.
    - Trapezoid: lowerBaseWidth, upperBaseWidth, height.
  - getAreaメソッドを適切に実装する.

## 課題7.3 図形 (2/3)

- 右のクラスに  
buildRectangle,  
buildTriangle,  
buildTrapezoidメソッドを追加せよ。
  - それぞれ,  
Rectangle, Triangle,  
Trapezoidオブジェクトを作成するメソッド.
  - 底辺や横幅, 高さは  
0以上, 10以下の長さの乱数とする.

```
public class ShapesDemo{
    public void run(){
        Shape[] shapes = new Shape[10];

        for(int i = 0; i < shapes.length; i++){
            int rand = (int)(Math.random() * 3);

            if(i == 0)
                shape[i] = buildRectangle();
            else if(i == 1)
                shape[i] = buildTriangle();
            else if(i == 2)
                shape[i] = buildTrapezoid();
        }
        for(int i = 0; i < shapes.length; i++){
            System.out.println(
                i + ": " + shapes[i].getType() +
                ": " + shape[i].getArea()
            );
        }
    }
}
```

## 課題7.3 図形 (3/3)

- 提出物
  - 全てのソースコードをzip圧縮したもの.
  - 学生証番号のフォルダを作成し, そこにすべてのソースコードを入れること.
    - 学生証番号のフォルダごとzip圧縮すること.
- ✕ 切
  - 2012/11/26(月) 16:45
- 余裕があれば, 別の具象図形を作成し, 追加せよ.
  - 例: 円(Ellipse), 六角形(Hexagon), . . .

## 課題7.3 ヒント

- 三角形のクラスのひな形を以下に示す.
  - 空欄をどのようにすれば良いかを考えよ.
  - 以下の例を基に, Shapeクラス, Rectangleクラス, Trapezoidクラスを実装せよ.

```
public class Triangle extends {
    private double baseWidth;
    private double height;
    public Triangle(double baseWidth, double height){
        super("Triangle");
        this.baseWidth = baseWidth;
        this.height = height;
    }
    public double getArea(){
        double area;
        area =  底辺 × 高さ ÷ 2
        return area;
    }
}
```