



発展プログラミング演習II

4. オブジェクト指向 クラスとインスタンス

コンピュータ理工学部
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



オブジェクト

- 前回までのプログラミング
 - `int` 型の変数や配列が主役となっていた.
 - より高度なプログラムには, より高度なデータ構造が必要となる.
 - C言語では構造体, 共用体, 列挙体があった.
 - Java言語では, **オブジェクト**が高度なデータ構造に当たる.

例題4.1 会員カード

- 会員カードを作成することを考える.
 - ある店が会員カードを発行して, お得意様にポイントカードサービスを提供したい.
 - このときの会員カードに含まれるべき性質は?
 - 会員氏名
 - 会員番号
 - 累計ポイント
 - この会員カードの機能は?
 - ポイントを追加する.
 - ポイントを使用する.
- フィールド(Field)
- メソッド(Method)
- メンバ(Member)

例題4.1 会員カードのフィールドを定義する.

- 右のテンプレートを参考にして, 以下のプログラムを作成せよ.
 - 以下の3つのフィールドを作成せよ.
 - int型の会員番号
(memberId)
 - String型の氏名 (name)
 - int型の累計ポイント
(points)

```
public class MemberCard{  
    型名 フィールド名;  
    ...  
}
```

例題4.2 会員カードクラスを使う

- 下のプログラムを作成して, 実行せよ.
 - MemberCard.javaと同じディレクトリに置くこと.

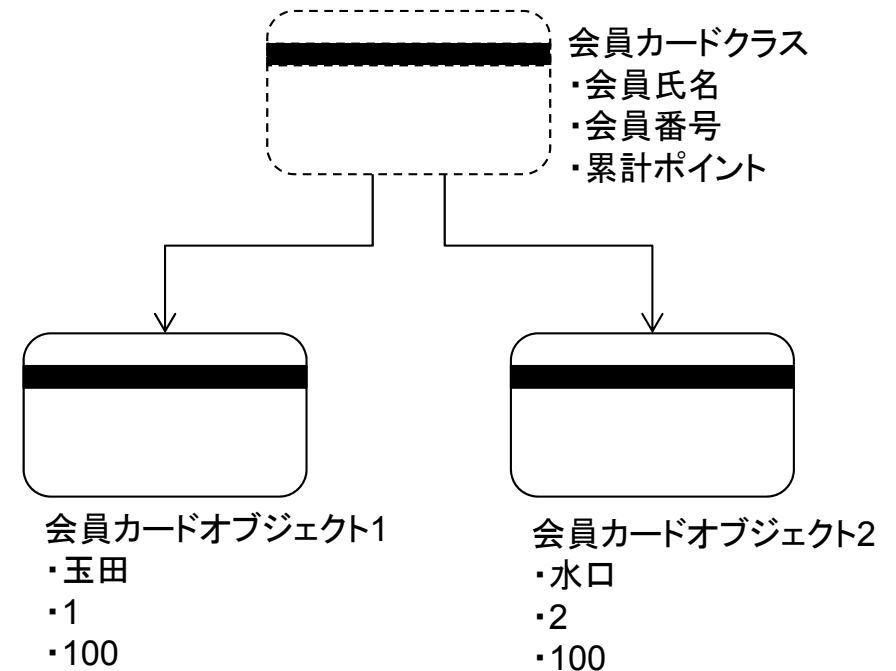
```
public class Example0301{  
    public static void main(String[] args){  
        MemberCard m1;  
        m1 = new MemberCard();  
        m1.memberId = 1;  
        m1.name = "玉田";  
        m1.points = 100;  
  
        System.out.println("会員番号は" + m1.memberId + "番");  
        System.out.println("会員氏名は" + m1.name);  
        System.out.println("累計ポイントは" + m1.points + "点です. ");  
    }  
}
```

教科書p.51 例3.1

会員カードオブジェクトを作る

- クラスからオブジェクトはnewで作成する.

```
MemberCard m1;  
m1 = new MemberCard();  
m1.memberId = 1;  
m1.name = "玉田";  
m1.points = 100;  
MemberCard m2 = new MemberCard();  
m2.memberId = 2;  
m2.name = "水口";  
m2.points = 100;
```

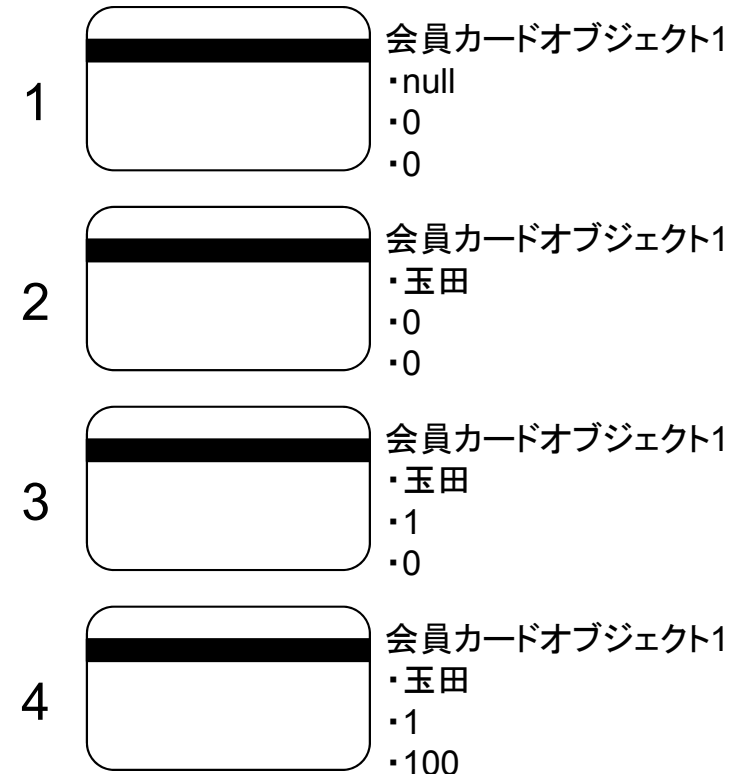


会員カードオブジェクトの フィールドへのアクセス (1/2)

```
1. MemberCard m1 = new  
   MemberCard();  
2. m1.name = "玉田";  
3. m1.memberId = 1;  
4. m1.points = 100;
```

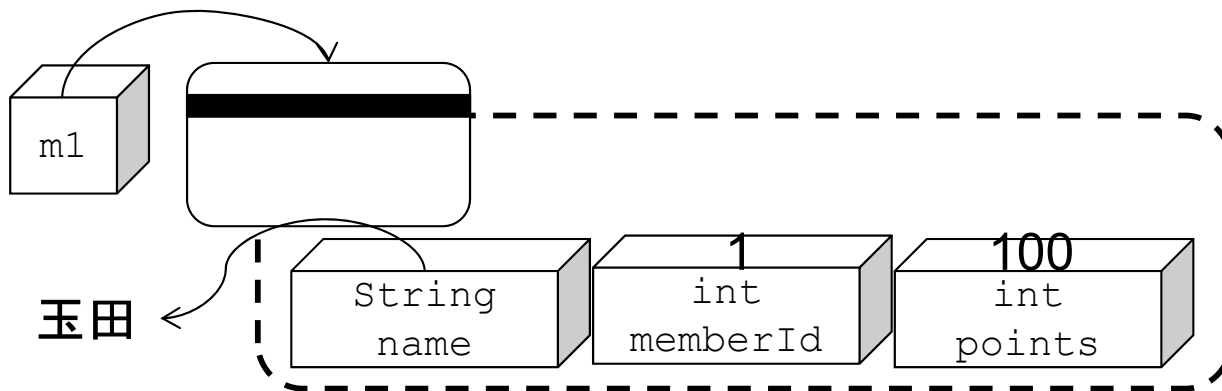
- オブジェクトのフィールド値の読み書きをフィールドへのアクセスと言う。
 - アクセスはオブジェクト名とフィールド名をドット(.)でつなげる。

オブジェクト

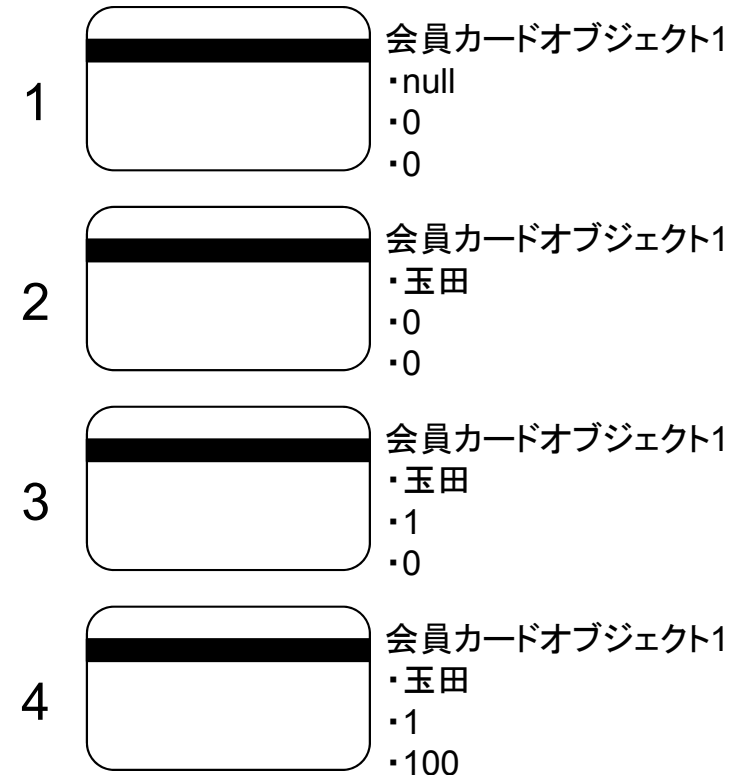


会員カードオブジェクトの メンバへのアクセス (2/2)

```
1. MemberCard m1 = new  
   MemberCard();  
2. m1.name = "玉田";  
3. m1.memberId = 1;  
4. m1.points = 100;
```



オブジェクト



クラスを利用する場合のプログラム手順

- クラスを宣言する.
 - いま考えたいものに対して、共通の使用を設計する.
 - 会員カード(`MemberCard`)には、会員番号(`memberId`), 氏名(`name`), ポイント(`points`)の3種類のデータを持つ.
- オブジェクトを作成する.
 - 宣言したクラスをもとに、個別のモノを作成する.
 - 会員カードを1枚作成し(`new`), そのデータを指定した(`name="玉田", memberId=1, points=100`).
- Javaでは1クラスに対して、1つのクラスファイル(`.class`)が作成される.
 - ソースコードがなくても、クラスファイルがあれば他人が利用できる.

例題4.3 会員カードの メソッドを定義する.

- 右のテンプレートを参考にして, 以下のプログラムを作成せよ.
 - 会員のデータを閲覧するshowメソッド.
 - 例題4.2で出力する内容を表示するメソッド.
 - 戻り値はなし(void).

```
public class MemberCard{  
    戻り値の型名 メソッド名(引数リスト){  
        処理;  
        ...  
        return 式;  
    }  
}
```

MemberCard.javaに追記せよ.

例題4.4 会員カードクラスを使う

- 下のプログラムを作成して, 実行せよ.
 - MemberCard.javaと同じディレクトリに置くこと.

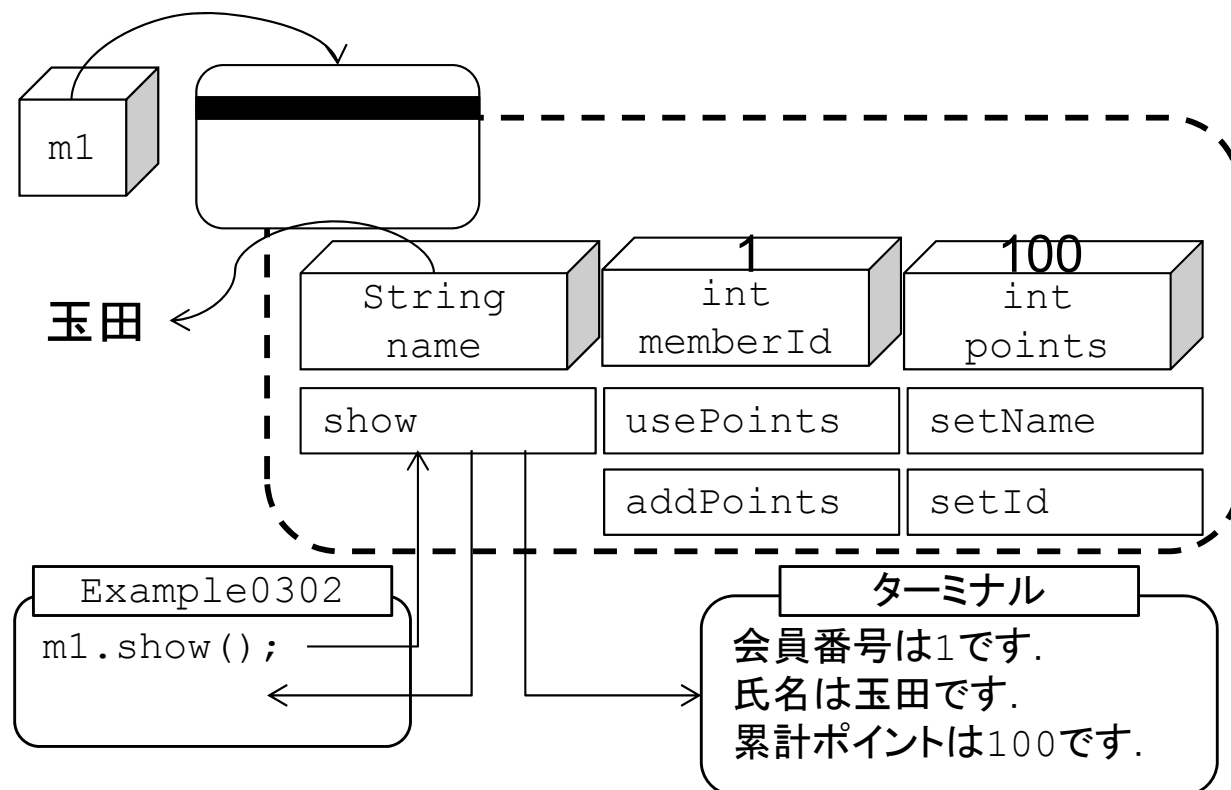
```
public class Example0302{  
    public static void main(String[] args){  
        MemberCard m1;  
        m1 = new MemberCard();  
        m1.memberId = 1;  
        m1.name = "玉田";  
        m1.points = 100;  
  
        m1.show();  
    }  
}
```

例題4.2を変更すること

教科書p.59 例3.2 改

オブジェクトのメソッド呼び出し

- オブジェクトのメソッドを実行することをメソッド呼び出しと言う。
 - 呼び出しはオブジェクト名とメソッド名をドット(.)でつなげる。
 - メソッド名の後ろに括弧を書き、括弧内にメソッドに渡す引数を書く。



引数を持つメソッド

- 右のメソッドを呼び出すとき、括弧内に`id`を渡さなければならない。
 - `setId(1);`
 - 上記の呼び出しの場合、右上のメソッド内の`id`は1として実行される。
 - `memberId`はフィールド。

```
void setId(int id){  
    memberId = id;  
}
```

int型の変数`id`が引数。
メソッド内で`id`を使用している。

例題4.5 会員カードの メソッドを定義する. 引数

- 右のテンプレートを参考にして, 以下のプログラムを作成せよ.
 - 名前を決めるsetNameメソッド.
 - 引数に名前を受け取り, フィールドnameに代入するメソッド.
 - 戻り値はなし(void).
 - 会員番号を決めるsetIdメソッド.
 - 引数に会員番号(id)を受け取り, フィールドmemberIdに代入するメソッド.
 - 戻り値はなし(void).
 - 現在のポイントにポイントを追加するaddPointsメソッド.
 - 引数に追加するポイントを受け取り, フィールドpointsに加算するメソッド.
 - 戻り値はなし(void).
- Example0302も作成したメソッドを使うように適切に変更すること.

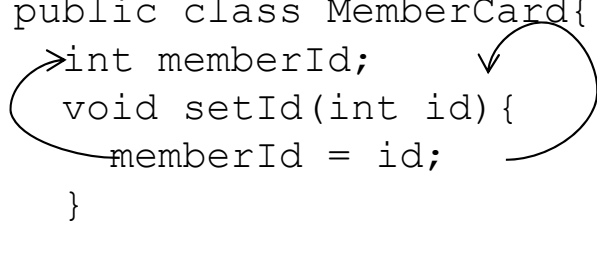
```
public class MemberCard{  
    戻り値の型名 メソッド名(引数リスト) {  
        処理;  
        ...  
        return 式;  
    }  
}
```

MemberCard.javaに追記せよ.

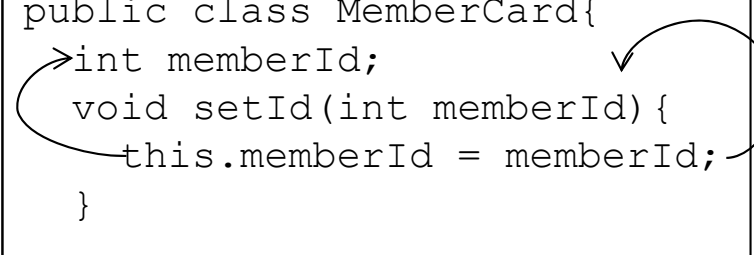
フィールドとローカル変数の区別

- フィールドとローカル変数が同じ名前の場合.

```
public class MemberCard{  
    >int memberId;  
    void setId(int id){  
        memberId = id;  
    }  
    ...  
}
```

A curved arrow originates from the parameter 'id' in the 'setId' method signature and points to the 'memberId' variable in the assignment statement 'memberId = id;'. Another curved arrow originates from the 'memberId' field declaration and points to the same 'memberId' in the assignment statement, illustrating that the local variable 'id' is being used to update the field.

```
public class MemberCard{  
    >int memberId;  
    void setId(int memberId){  
        this.memberId = memberId;  
    }  
    ...  
}
```

A curved arrow originates from the parameter 'memberId' in the 'setId' method signature and points to the 'memberId' variable in the assignment statement 'this.memberId = memberId;'. Another curved arrow originates from the 'memberId' field declaration and points to the 'this.memberId' part of the assignment statement, illustrating that the local variable 'memberId' is being used to update the field.

- 左の例は`id`と`memberId`を区別できるので、問題ない.
- 右の例はフィールドの`memberId`とメソッド引数(ローカル変数)の`memberId`と区別できない.
 - フィールドの場合, 変数名の前に`this.`を付けるとフィールドの参照と扱われる.

例題4.6 会員カードの メソッドを定義する. 複数の引数.

- 右のテンプレートを参考にして, 以下のプログラムを作成せよ.
 - フィールドを初期化するinitメソッドを追加する.
 - 引数はid, name, initialPointsの3つ.
 - 引数はコンマ(,)で区切る.
 - initメソッド内部で, setId, setName, addPointsメソッドを呼び出す.
 - 戻り値はなし(void).
- Example0302も作成したメソッドを使うように適切に変更すること.

```
public class MemberCard{  
    戻り値の型名 メソッド名(引数リスト){  
        処理;  
        ...  
        return 式;  
    }  
}
```

MemberCard.javaに追記せよ.

例題4.7 会員カードの メソッドを定義する. 返り値

- 右のテンプレートを参考にして, 以下のプログラムを作成せよ.
 - ポイントを使用するusePointsメソッド.
 - 引数に使用するポイントを受け取り, そのポイントだけ減算する.
 - ただし, 現在のポイントが受け取ったポイントより低かった場合は, 現在のポイントだけ使うようにする.
 - 使ったポイントを返す.
- Example0302に適当にポイントを使う処理を追加すること.

```
public class MemberCard{  
    返り値の型名 メソッド名(引数リスト) {  
        処理;  
        ...  
        return 式;  
    }  
}
```

MemberCard.javaに追記せよ.

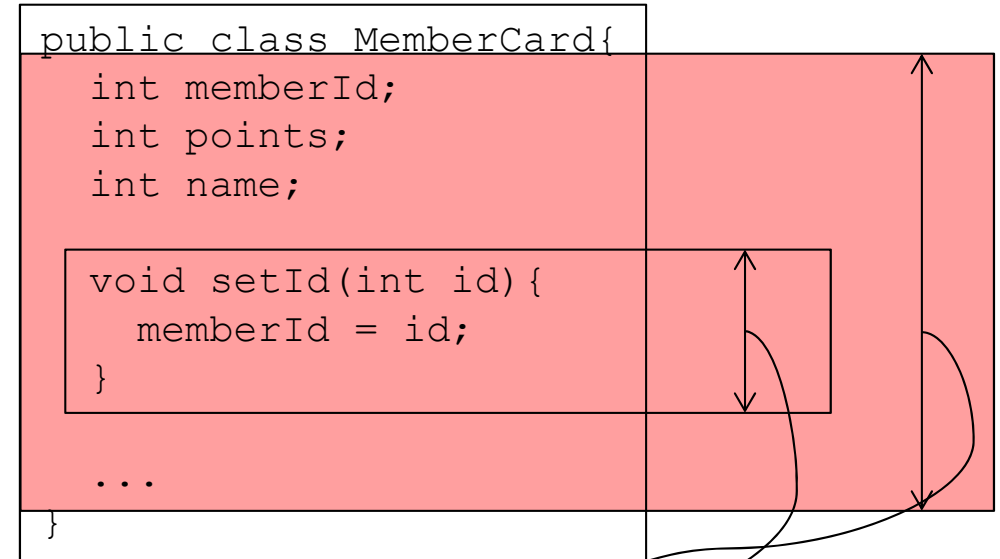


setter/getter

- setter/getterはフィールドの値を設定・取得するためのメソッド.
 - setter: フィールドに値を設定するためのメソッド.
 - `setXxx (...)`
 - getter: フィールドの値を返すだけのメソッド.
 - `getXxx ()`

フィールド (Field)

- メソッド(C言語の関数に相当するもの)をまたがって値が保たれる.
 - `init`メソッドで設定された値が`setId`メソッド内でも有効.
 - ローカル変数(局所変数)はメソッドをまたがって存在できない.
 - スコープが異なる.



`id`の有効範囲

`memberId`, `points`, `name`の有効範囲

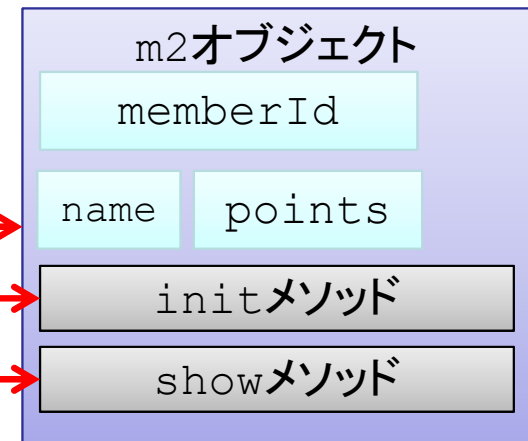
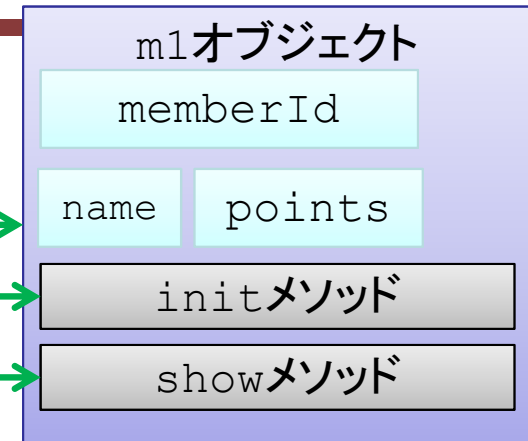


例題4.8 会員を追加する.

- Example0302を以下のように変更せよ.
 - MemberCard型の変数m2を追加する.
 - m2に会員番号2, 初期ポイント200, 各自の名前を設定する.
 - m2のポイントを適当に増減させる.
 - m1とm2の情報を出力する(showメソッドを呼び出す).

フィールドとメソッドの関係

```
public static void main(String[] args){  
    MemberCard m1 = new MemberCard();  
    m1.init(1, "玉田", 100);  
    MemberCard m2 = new MemberCard();  
    m2.init(2, "水口", 200);  
  
    m1.show();  
    m2.show();  
}
```



- m1とm2のフィールドの値は互いに関係しない.
- 別のオブジェクトm3を作成しても, m1, m2の値に影響しない.

クラスとは

- Javaプログラムで使われる部品をオブジェクトと呼ぶ.
- クラスとは部品 (=オブジェクト) の設計図のこと.
 - 同じクラス定義を元にいくつものオブジェクトを作れる.
- MemberCardオブジェクトとは
 - MemberCardクラスの定義を設計図に作成した (new) オブジェクト.
 - MemberCardオブジェクトのグループを MemberCardクラスと呼ぶ.

課題4.1 Pointクラスの作成

- 右のCalcクラスをオブジェクトを使ったプログラムに書き直せ.
 - Pointクラスを作成せよ.
 - scaleメソッドの戻り値をPointオブジェクトに変更せよ.
 - scaleメソッドの引数x, yをPointオブジェクトに置き換えよ.
 - runメソッドの最後の出力をPointクラスのshowメソッド呼び出しに変更せよ. ただし, 出力結果は変更しないこと.

```
public class Calc{
    public void run(){
        int x = 3, y = 2;
        int[] v = scale(x, y, 2);
        x = v[0];
        y = v[1];
        System.out.println(x + " " + y);
    }
    public int[] scale(int x, int y, int s){
        int[] vec = new int[2];
        vec[0] = x * s;
        vec[1] = y * s;
        return vec;
    }
    public static void main(String[] args){
        Calc c = new Calc();
        c.run();
    }
}
```




課題4.1に関する注意事項

- ファイル名, ✕切など
 - クラス名: Point_学生証番号
 - ソースファイルをMoodleの「課題4.1: Pointクラスの作成」に提出.
 - 学生証番号は6桁とし, 最初にgは付けない.
 - ✕切: 2012年10月18日(木) 9:00.
- 評価は自動的に行うため, 出力結果は変えないこと.

参考：メソッド命名のルール

- メソッドの名前は、コンパイラで制限されていない。何でも良いが、推奨されている命名規則がある。
- 推奨されている命名規則（命名ルール）
 - 小文字から始める。× `SetName`. ○ `setName`
 - 単語の頭文字を大文字にして、単語を区切る。× `setname`, ○ `SetName`
 - フィールドの値を設定するメソッドは `set` から始める。
× `nameSet`, ○ `setName`
 - フィールドの値を取得するメソッドは `get` から始める。
× `nameSyutoku`, ○ `getName`
 - ただし、返り値が `boolean` のときは `is` で始める。
 - 動詞から始める。

情報隠ぺい

- MemberCard型のオブジェクトの名前設定方法は2種類ある.
 - `m1.name = "玉田";`
 - `m1.setName("玉田");`
- どっちが良いか.
 - 他のクラスのオブジェクトを直接操作するのは、よくないこととされる.
 - 直接操作すると、そのフィールドを変更したいときの修正が大変になるため。どこで操作されているか確かめるのが大変。
- 情報を隠蔽して、特定の経路でしかアクセスできないようにする.
 - フィールドや一部のメソッドを、特定の領域内でしか使わないようにすること.
 - ブラックボックス化とも言う。
 - オブジェクト指向言語では重要な指針になっている。

例題4.9 情報隠ぺい

- MemberCardのフィールドをすべてprivate宣言せよ.
 - フィールドの型の前にprivateを付ける.
- private宣言すると, そのクラス以外からはアクセスできない.
 - Example0302からprivate宣言したフィールドの参照ができないことを確認すること.

クラスとは (1/2)

- Java言語のプログラムはクラス(class)の定義を集めたもの.

典型的なクラスの定義

```
public class クラス名 {  
    メンバの定義の並び  
}
```

- メンバの定義の並び
 - クラスのメンバの定義を任意順序で並べたもの.
 - メソッド, フィールド, コンストラクタ
 - 順番に決まりはない.

クラスとは (2/2)

- オブジェクトの種類に対する概念.
 - 同じ種類のオブジェクトのグループをクラスと呼ぶ.
 - オブジェクトは必ずどれかのクラスの定義を設計図として作成される.
- 同じクラス定義から作成されたオブジェクトは, 同じ定義のフィールドやメソッドを持つ同種(同型)のオブジェクトとなる.
- クラス名を名前に持つ型のことを参照型(reference type)と呼ぶ.
 - 基本型に対する型.

参照型 (reference type)

- C言語のポインタに相当するもの.
 - 参照型の変数には, オブジェクトへの参照番号が与えられている.
 - しかし, C言語のように, 参照番号を操作する方法は提供されていない.
- ```
MemberCard m1 = new MemberCard();
MemberCard m2 = m1;
```
- 変数m1とm2は同じ参照番号を持ち, 同じMemberCardオブジェクトを指すようになる.
- 配列も参照型.

# メソッド

- オブジェクトが実行可能な処理のこと.

典型的なメソッドの定義

```
public 戻り値の型 メソッド名 (仮引数列) {
 メソッドの動作の定義
}
```

- クラス定義の中にメソッドの定義が含まれていると, そのクラスのオブジェクトはすべてそのメソッドを持つ.
- メソッドの動作の定義**の中では`this`という特別な変数が宣言なしで利用できる.
  - `this`はそのメソッドを現在実行しているオブジェクトを表す.
  - `this`の型はそのメソッドが定義されているクラス.



# フィールド (1/2)

- オブジェクトに固有の変数.

典型的なフィールドの定義

```
public 型 フィールド名;
```

- 局所変数の宣言文の先頭にpublicをつけたような形.
  - publicのほか, protectedやprivateが付けられる
    - 詳細は後日.
- 初期値を指定することができる.
  - `public int x = 10;`
- 初期値を明示的に指定しないとき.
  - 0, nullが初期値として代入される.
  - 局所変数(ローカル変数)は明示に初期化しなければ, 値は未定義となる.
    - 初期値を代入するまで使えない.

# フィールド (2/2)

- クラス定義にフィールド定義が含まれていれば、そのクラスのオブジェクトはすべてその名前のフィールドを持つ。
  - フィールドはオブジェクトごとに別々。
    - オブジェクトが異なれば同じフィールドに異なる値を代入できる。
- フィールドは式の中では変数のように扱える。
  - 「オブジェクト式.フィールド名」と書くことで変数と同様に使える。

# コンストラクタとnew演算子

- 初期値を代入するメソッドを用意した場合、オブジェクト作成後、**忘れずに**呼び出さなければならない。
  - 忘れないという保証は全くない。
- newと一緒に初期値を代入するメソッドを呼び出したい場合、コンストラクタ(Constructor)を使う。
  - 全てのクラスにはコンストラクタが存在する。

## コンストラクタの定義

```
public クラス名 (仮引数列) {
 コンストラクタの動作の定義
}
```

- メソッド定義との違い
  - 戻り値の型が省略されている。
  - **コンストラクタの動作**の定義はメソッドの場合とまったく同じ。
  - コンストラクタが引数を取るとき、new演算子に実引数を渡さなければならない。

```
new クラス名 (実引数列)
```



# 例題4.10

---

- MemberCardのinitメソッドをコンストラクタを使って書き直せ.
- Example0302内でのMemberCardのinitメソッド呼び出しをコンストラクタ呼び出しに書き直せ.

# thisの省略

- メソッドやコンストラクタの中では「`this.`」を省略できる.
- フィールドと局所変数を区別したいとき, フィールドに「`this.`」を付ける.
  - フィールドと局所変数(含 仮引数)に同じ名前を付けることが可能.
  - フィールドと同じ名前の局所変数があったとき, そのメソッド(orコンストラクタ)内では`this`を付けなければ局所変数が参照される.



# 今までのプログラム

---

- コンストラクタを作っていなくても、オブジェクトが作成できていた.
- Javaコンパイラが自動的に引数なしのコンストラクタを作っていた.
  - デフォルトコンストラクタと呼ばれるコンストラクタ.
- そのため, `new クラス名 ()` の呼び出しが実行できていた.
  - コンストラクタを自分で定義するとデフォルトコンストラクタは作成されない.



# 例題4.11

---

- 例題4.10でコンストラクタを定義した.
- その場合, デフォルトコンストラクタが作成されないことを確認する.
  - `MemberCard`オブジェクト作成時に, 引数なしのコンストラクタを呼び出して, コンパイルに失敗することを確認する.



# 会員番号の管理

---

- 会員番号は重複しない値を使う.
- MemberCard側で適当に付けてほしい.
  - ユーザプログラム側で会員番号の管理はしたくない.
  - 重複してしまうかもしれない.
- MemberCardオブジェクトは作成されるたびにフィールドが初期化されるので, 管理できない.
- staticフィールドで管理すれば良い.





# staticフィールド

---

- 同じクラスに属す全てのオブジェクトで共有されるフィールド。
  - 通常のフィールドは各オブジェクトごとに異なる値を代入できる.
  - 同じクラス定義であれば, staticフィールドの値はどのオブジェクトでも常に同じ.
- 「クラス名.フィールド名」で参照できる.
  - 通常のフィールドは「オブジェクト式.フィールド名」で参照する.

# staticメソッド

- 「クラス名.メソッド名(実引数列)」で呼び出すメソッド.
- staticメソッドの中では`this`は使えない.
  - `this`を介したフィールドの利用はできない.
    - `this`を省略した形でも利用できない.
  - staticメソッド内で利用できるフィールドはstaticなフィールドだけ.
- 典型的な例は`main`メソッド.
  - 一番最初に呼び出されると決まっているほかは, 他のstaticメソッドとなんら変わるところはない.

# staticイニシャライザ

- Staticフィールドに初期値を与えるには、フィールドの定義と一緒に一行で書く。
  - 初期値を与えなければ0もしくはnullが初期値となる。
- もっと複雑な処理が必要なときにはstaticイニシャライザを利用する。
  - Staticフィールドは同じクラスに属するすべてのオブジェクトで共有されるため、コンストラクタ内で初期化できない。

```
public MemberCard{
 private static int numberOfMember;
 static{
 // staticイニシャライザで初期値を代入しなくても構わないが、
 // どのように使うかのサンプルとして示している。
 numberOfMember = 0;
 }
 ...
}
```

## 例題4.12

- MemberCardクラスにint型のstaticフィールド, numberOfMembersを定義する.
  - 初期値は0とする.
- MemberCardのコンストラクタの引数からidを削除する.
- コンストラクタ内部の処理に以下を追加する.
  - idにnumberOfMembersを代入する.
  - numberOfMembersに1を追加する.

## 課題4.2: 点数計算

- 右のプログラムが動くように、プログラムを作成せよ。
  - `getAverage`, `getMax`, `getMin`は以下のメソッドで与えられた点数の平均, 最大値, 最小値を返すメソッド.
  - `setMathScore`, `setJapaneseScore`, `setEnglishScore`.
- なお, `setXxxxScore`に渡す数値を変更した場合でも正当な結果を返すようにせよ.

```
public class Evaluator{
 public Evaluator(){
 Student s1 =
 new Student("学生証番号", "玉田");
 s1.setMathScore(60);
 s1.setJapaneseScore(80);
 s1.setEnglishScore(50);

 System.out.println(
 "平均: " + s1.getAverage() +
 ", 最大: " + s1.getMax() +
 ", 最小: " + s1.getMin());
 }

 public static void main(String[] args){
 Evaluator evaluator =
 new Evaluator();
 }
}
```



# 課題に関する注意事項

---

- ファイル名, ✕切など
  - クラス名: Student.java
  - ソースファイルをMoodleの「課題4.2: 点数計算」に提出.
  - ✕切: 2012年10月22日(月) 16:45
- 余裕があれば, isValidIdメソッドを追加すること.
  - 与えられた学生証番号が正当な番号かどうかを判定するメソッド.
  - 6桁の学生証番号の全ての数字を足し合わせると, 下一桁が0になる.

# メソッドのオーバーロード

- 同じ名前のメソッドを作ることができる.
  - ただし, 仮引数の個数や, 型が異なる場合だけ.
  - 返り値の型は同じでなければならない.
- 仮引数の数や型が異なる同名のメソッドを複数定義することをメソッドのオーバーロード(Overload)と呼ぶ.
  - 仮引数の数や型のことをシグネチャ(Signature)という.
- 典型例
  - `System.out.println(式);`

## 例題4.13 コンストラクタの オーバーロード

- MemberCardに以下のコンストラクタを追加する.
  - 氏名のみを指定するコンストラクタ.
    - デフォルトの累積ポイント(50)を与える.
  - 既存のコンストラクタ
    - 氏名と累積ポイントを指定している.



# コンストラクタから 別のコンストラクタを呼ぶ

- 重複しているコードがある。
  - プログラムコードの重複は良くない.
- 下のコンストラクタから上のコンストラクタを呼べれば良い。
  - 他のコンストラクタを呼び出すには, `this()`を使う.
  - 使えるのはコンストラクタ内部からのみ.

```
public class MemberCard{
 private static int numberOfMembers = 0;

 private int memberId;
 private String name;
 private int points;

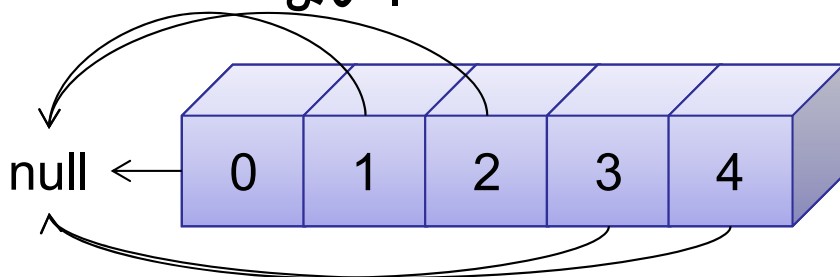
 public MemberCard(String name, int points){
 setName(name);
 addPoints(points);
 setId(numberOfMembers);
 numberOfMembers++;
 }
 public MemberCard(String name){
 setName(name);
 addPoints(50);
 setId(numberOfMembers);
 numberOfMembers++;
 }
 ...
}
```

# 例題4.14 他のコンストラクタの 呼び出し

- 例題4.13で作成したコンストラクタから、元のコンストラクタを呼び出す.
- `this()` は通常のメソッド呼び出しのように、引数も使える.
  - ただし、対応するコンストラクタが必要.

# クラスの配列


- クラスも配列にできる.
  - `MemberCard[] cards = new MemberCard[5];`
  - この命令だと, 入れる場所を用意しただけ.
  - それぞれの要素は何も指していない(`null`)になっている.
    - それぞれの要素ごとに`new`でオブジェクトを作成しなければならない.



```
MemberCard[] cards = new MemberCard[5];
for(int i = 0; i < cards.length; i++){
 cards[i] = new MemberCard(...);
}
```

# 課題4.3, 4.4

- 課題4.3
  - Rockクラスを作成する.
    - Rockクラスはdouble型の重さ(weight)をフィールドに持つ.
    - コンストラクタで重さを設定できる.
    - 重さを返すメソッドを持つ.
- 課題4.4
  - RockManagerを作成する.
    - 10個のRockオブジェクトを作成し、配列に格納する.
    - Rockオブジェクトを作成するときに重さを0.5kg～10kgまで乱数により生成する.
      - Math.random() で 0～1 までの数値が乱数で得られる.
    - Rockオブジェクトをすべて作成したあとで、それぞれの重さと合計の重さを出力するプログラム.



# 課題4.3, 4.4について


---

- ファイル名
  - 課題4.3: Rock.java
  - 課題4.4: RockManager.java
- ✕ 切
  - 2012年10月25日(木) 9:00
- 提出方法
  - ソースファイルをMoodleの「課題4.3: Rock」, 「課題4.4: RockManager」に提出する.

# 課題4.5 リンクリスト

- 右のプログラムが動くように, ListNode クラスを作成せよ.
  - ListNode はリンクリスト構造で作成すること.
  - どんな情報を隠して, どんな情報を公開するのもかも考えること.

```
public class LinkedListDemo{
 public LinkedListDemo(String[] args){
 ListNode root = buildList(args);
 showList(root);
 }
 public ListNode buildList(String[] args){
 ListNode root = new ListNode(args[0]);
 for(int i = 1; i < args.length; i++){
 ListNode next = new ListNode(args[i]);
 next.setNext(root.getNext());
 root.setNext(next);
 }
 return root;
 }
 public void showList(ListNode root){
 ListNode node = root;
 while(node != null){
 System.out.println(node.getValue());
 node = node.getNext();
 }
 }
 public static void main(String[] args){
 LinkedListDemo demo =
 new LinkedListDemo(args);
 }
}
```



# 課題4.5について

---

- ファイル名
  - 課題4.5: ListNode.java
- ✕ 切
  - 2012年10月29日(月) 16:45
- 提出方法
  - ソースファイルをMoodleの「課題4.5: ListNode」に提出する.