ソフトウェア工学

7. オブジェクト指向とは

2017年12月27日

慶應義塾大学 理工学部 管理工学科
飯島 正 (iijima@ae.keio.ac.jp)

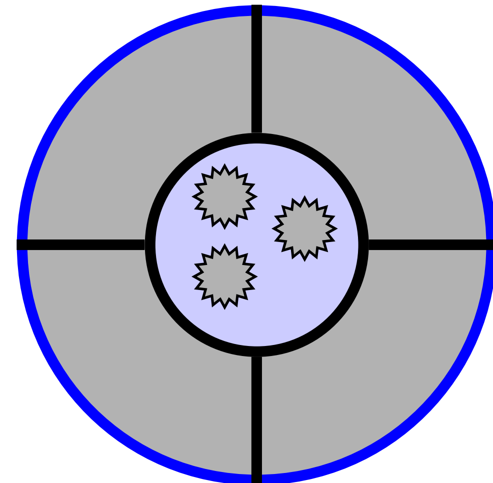
オブジェクト指向とは…

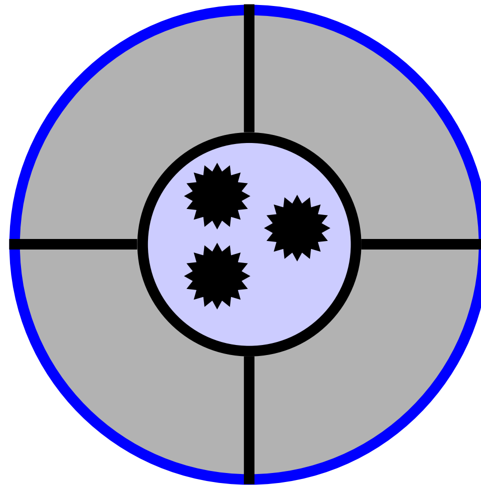
- オブジェクト指向とは…

- プログラムを
  オブジェクトの集まり
  で創る
  考え方(プログラミングパラダイム)
  です．．．

オブジェクト(Object)

- オブジェクト
  ＝その定義が本来のクラス

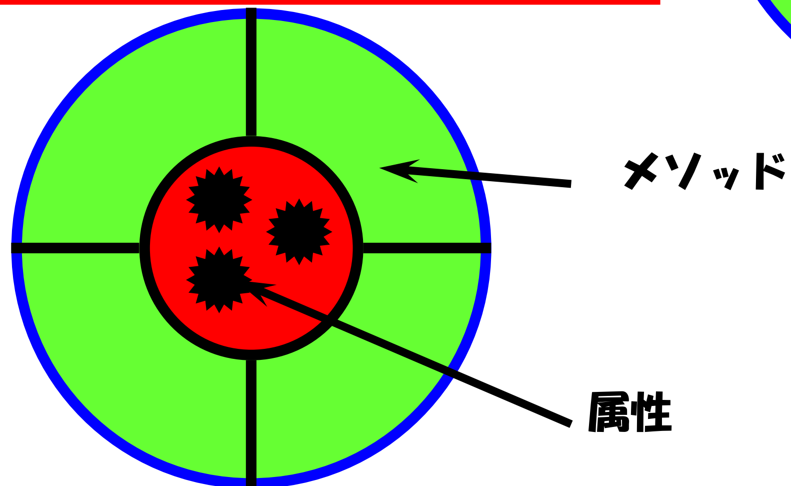




オブジェクト(Object)

- カプセル化
  ＝関連する属性とメソッドを
  パッケージングすること

- スイカのように皮（メソッド）が
  種（属性）を守っている
- メソッドを介して属性にアクセスする

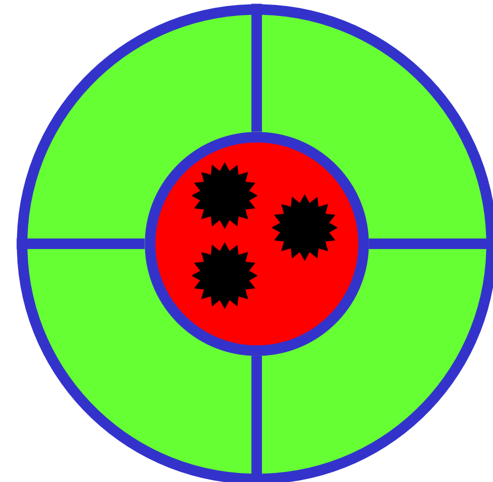


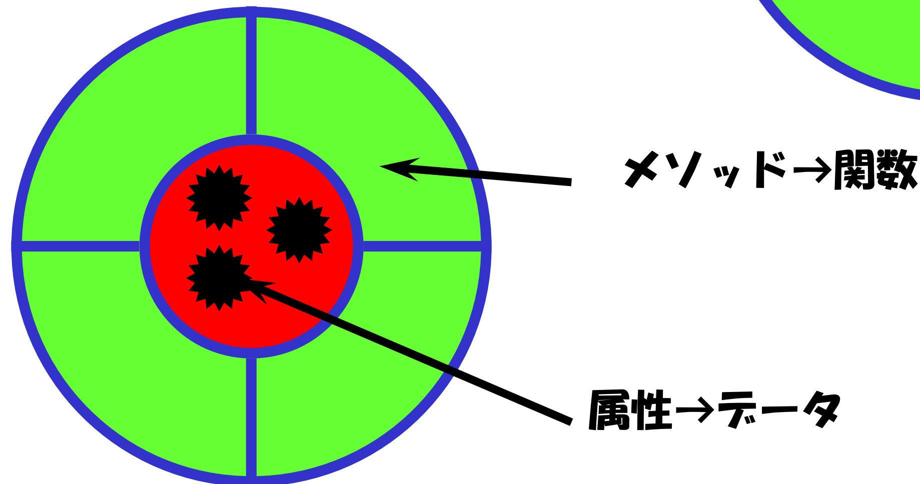
 iijima@ae.keio.ac.jp

オブジェクト(Object)

- カプセル化
  ＝関連する属性とメソッドをパッケージングすること

- スイカのように皮（メソッド）が種（属性）を守っている
- メソッドを介して属性にアクセスする

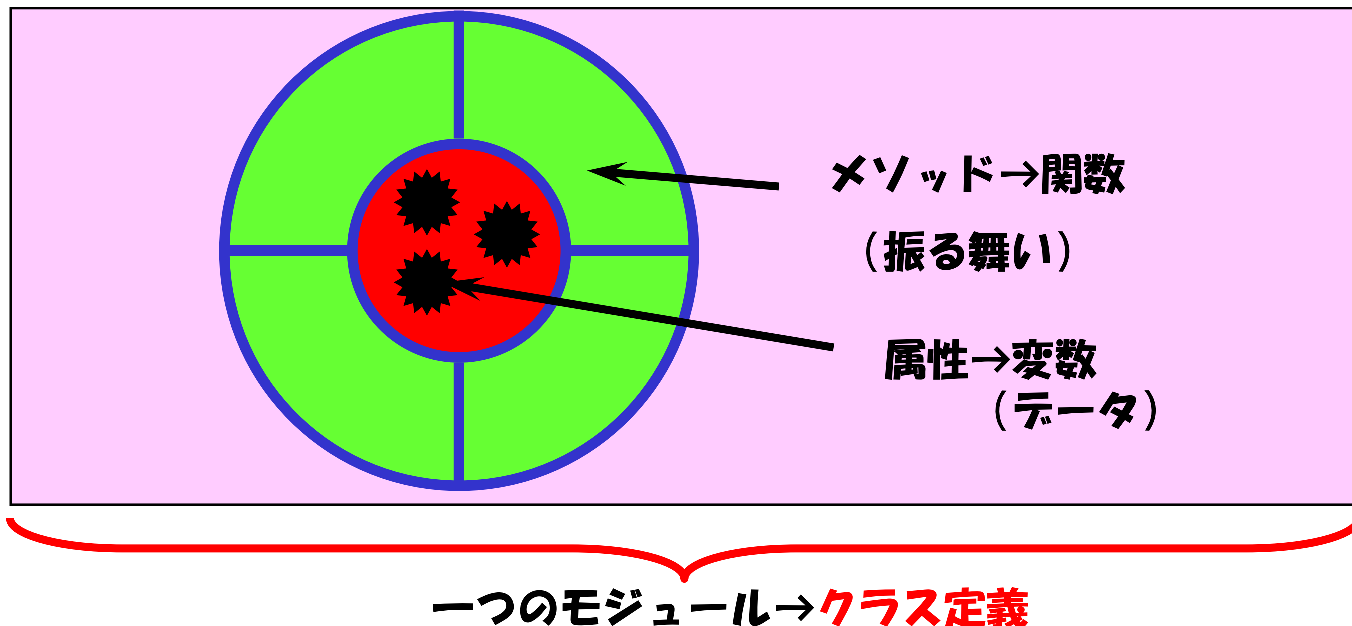




オブジェクト(Object)

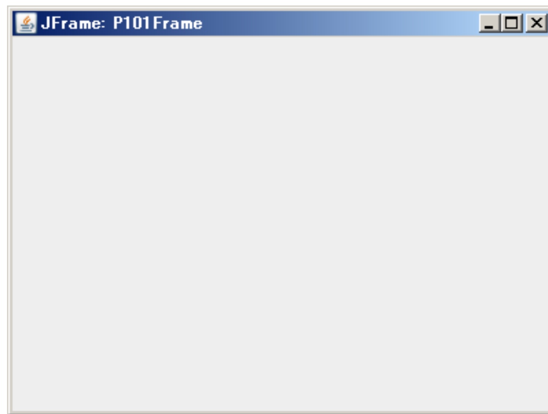
- カプセル化
  ＝関連する属性とメソッドを
  パッケージングすること

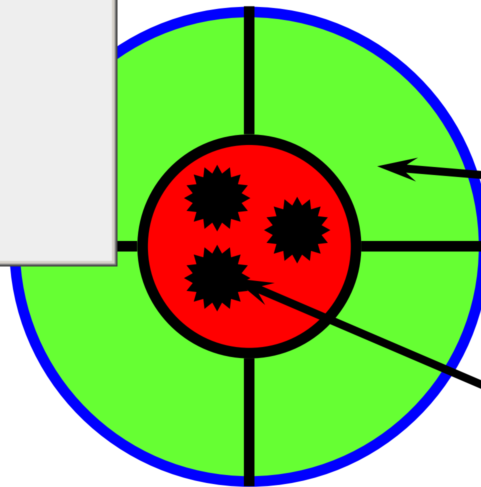
- スイカのように皮（メソッド）が
  種（属性）を守っている
- メソッドを介して属性にアクセスする

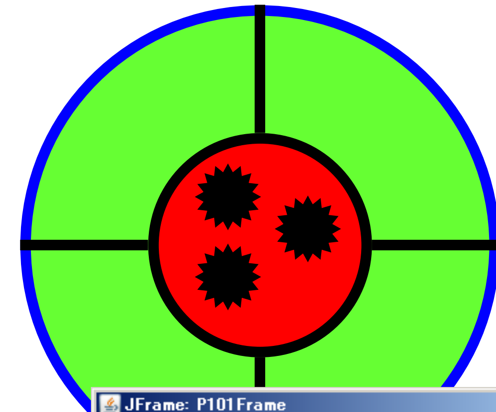


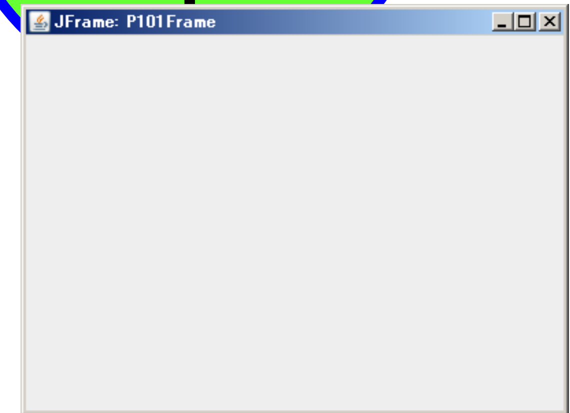
ウィンドウもオブジェクト

ウィンドウは，一つ一つがオブジェクト







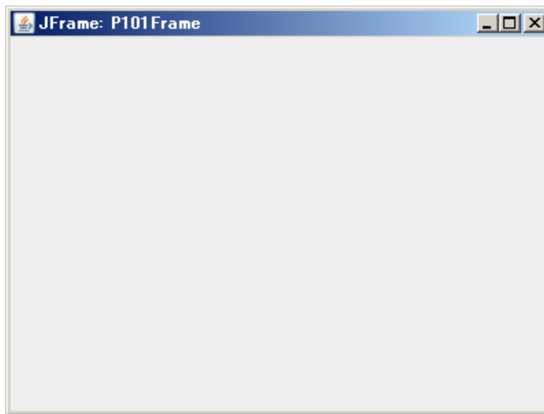


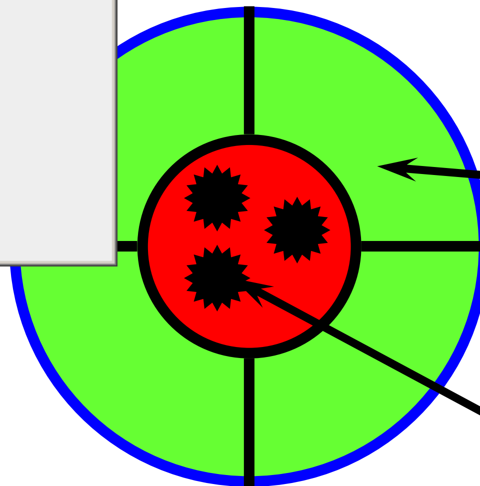
メソッド

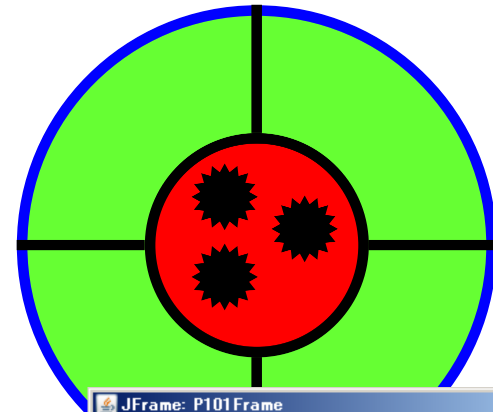
属性

ウィンドウもオブジェクト：属性を持つ

ウィンドウ・オブジェクトは，それぞれ大きさや位置，タイトルバーの文字列といった**固有の属性（データ：内部状態）**を持っている



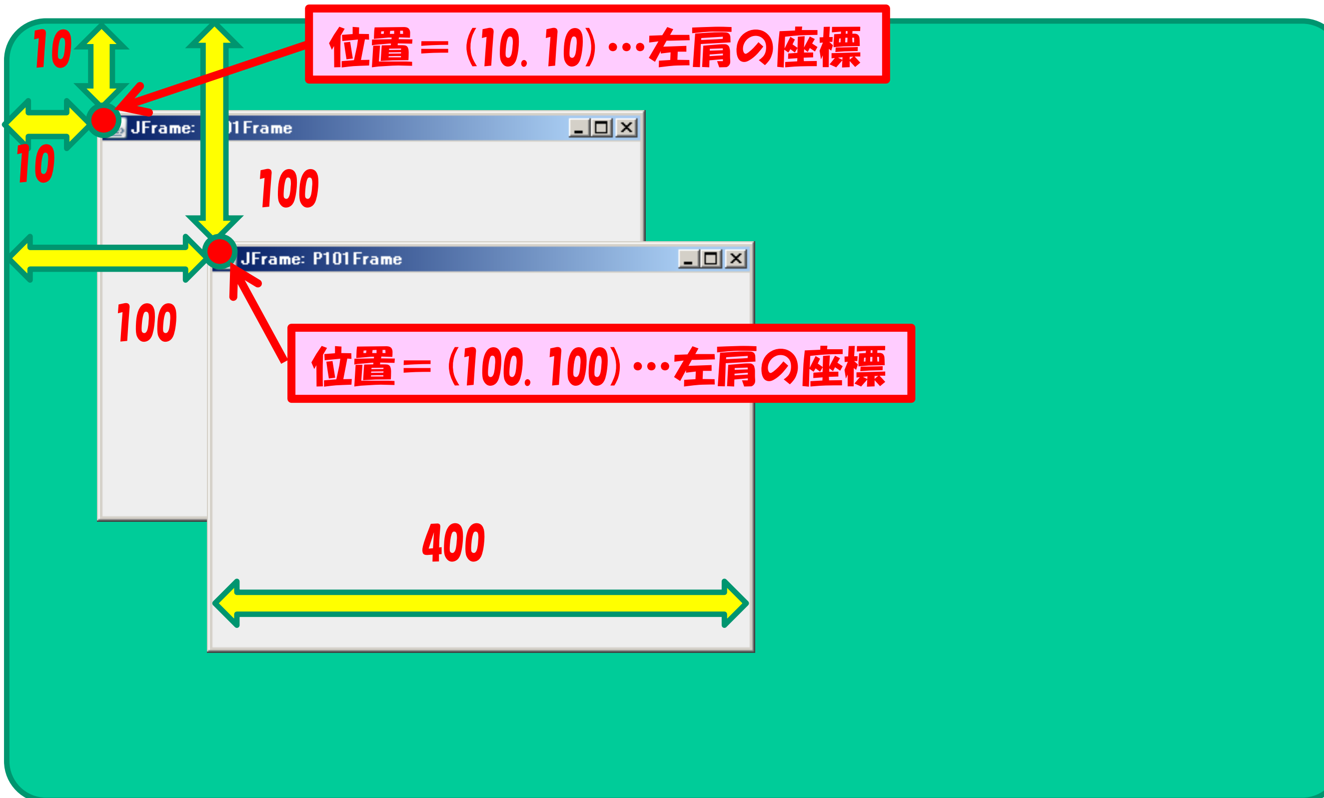




メソッド

属性
- 位置＝(10, 10)…左肩の座標
- 大きさ=(400, 300)…幅と高さ
- タイトルバーの文字列

ウィンドウもオブジェクト：属性を持つ

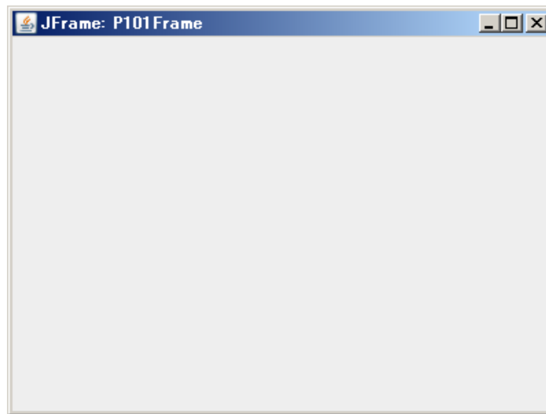


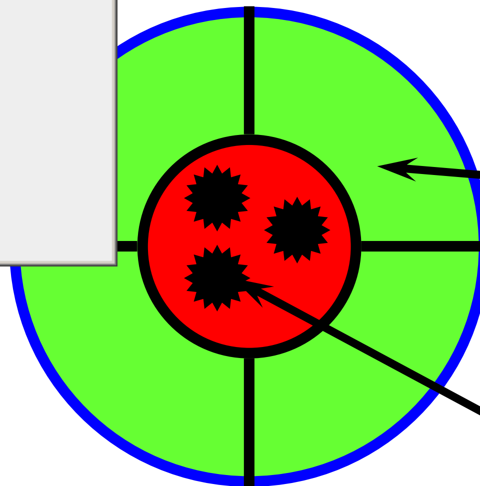
属性
- 位置＝(10. 10)…左肩の座標
- 大きさ=(400, 300)…幅と高さ
- タイトルバーの文字列

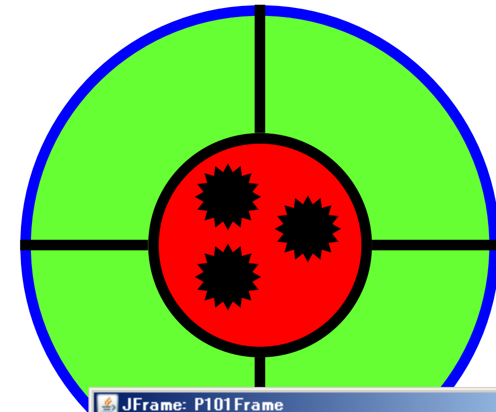
属性
- 位置＝(100. 100)…左肩の座標
- 大きさ=(400, 300)…幅と高さ
- タイトルバーの文字列

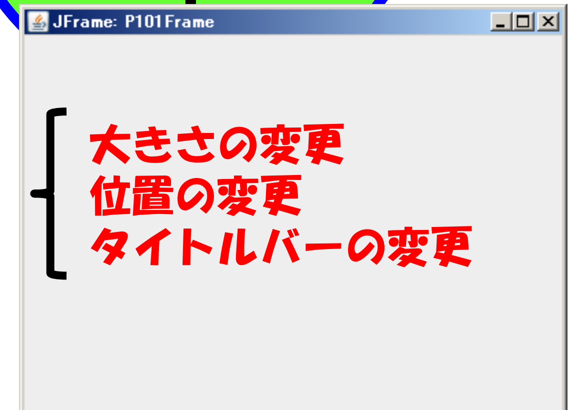
ウィンドウもオブジェクト：メソッドを持つ

ウィンドウ・オブジェクトは，それぞれが持つ属性の値を変更するための**メソッド（関数）**を持っている









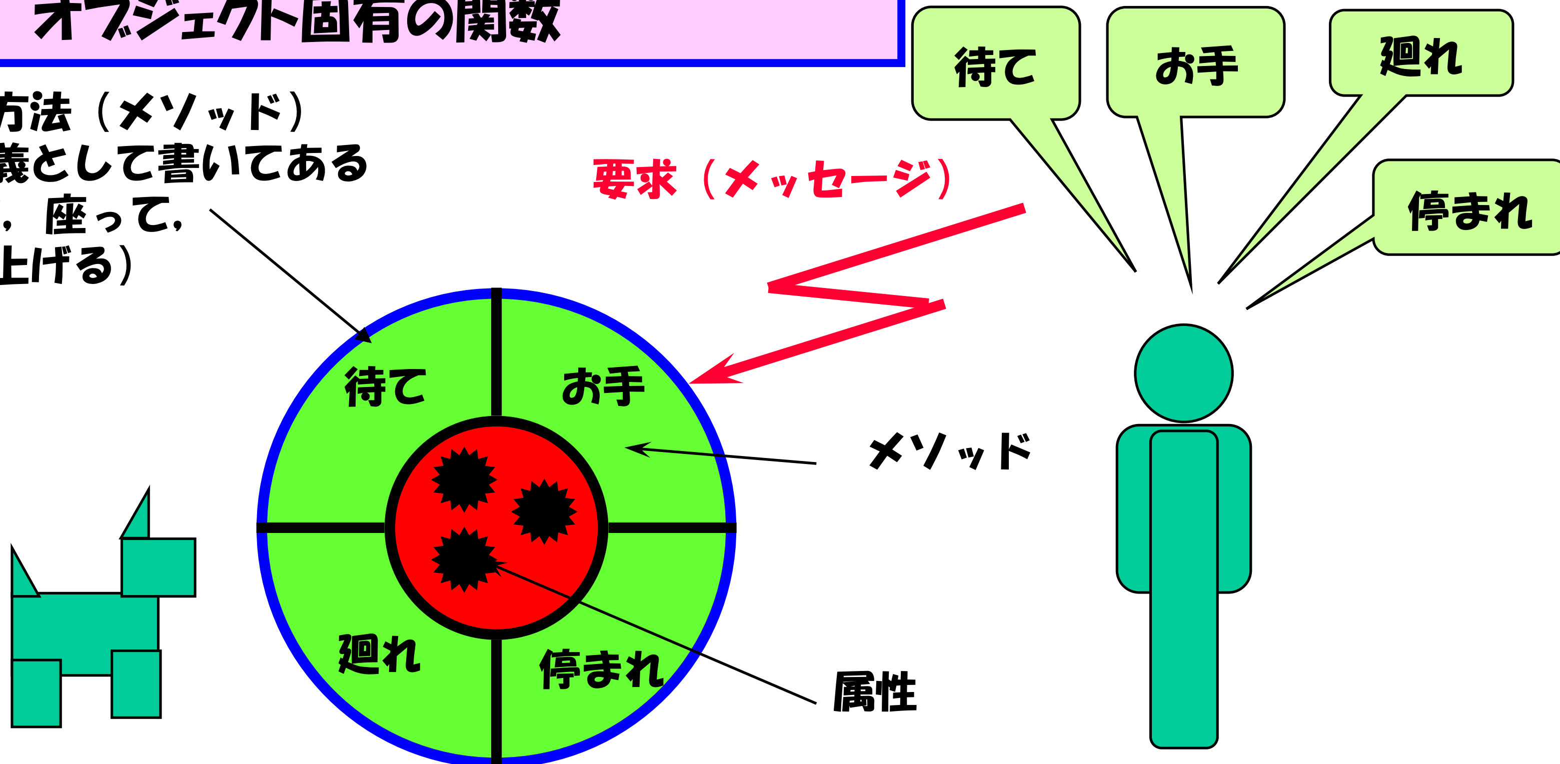
メソッド

- 大きさの変更
- 位置の変更
- タイトルバーの変更

属性

メソッド(振る舞い，メンバ関数)

- メソッド⇒オブジェクトの振る舞い
    - 外部へ作用する機能
    - 内部状態変数を書き換えるデータ操作
  - 外部から起動できる個々のオブジェクト固有の関数



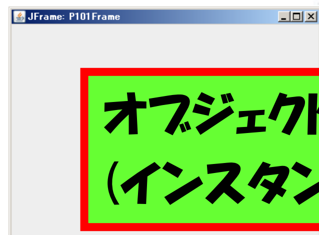
オブジェクトはクラスから作る

クラスに対して,
クラスから生成した
オブジェクトのことを
インスタンス
(実例/実体)
と呼ぶ

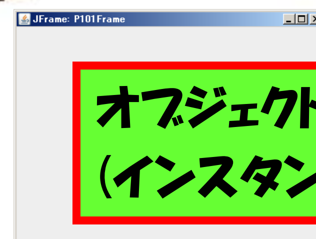
クラス定義

属性定義
- 位置
- 大きさ
- タイトルバーの文字列

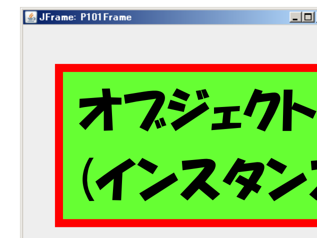
メソッド定義
- 大きさの変更
- 位置の変更
- タイトルバーの変更



オブジェクト(インスタンス)

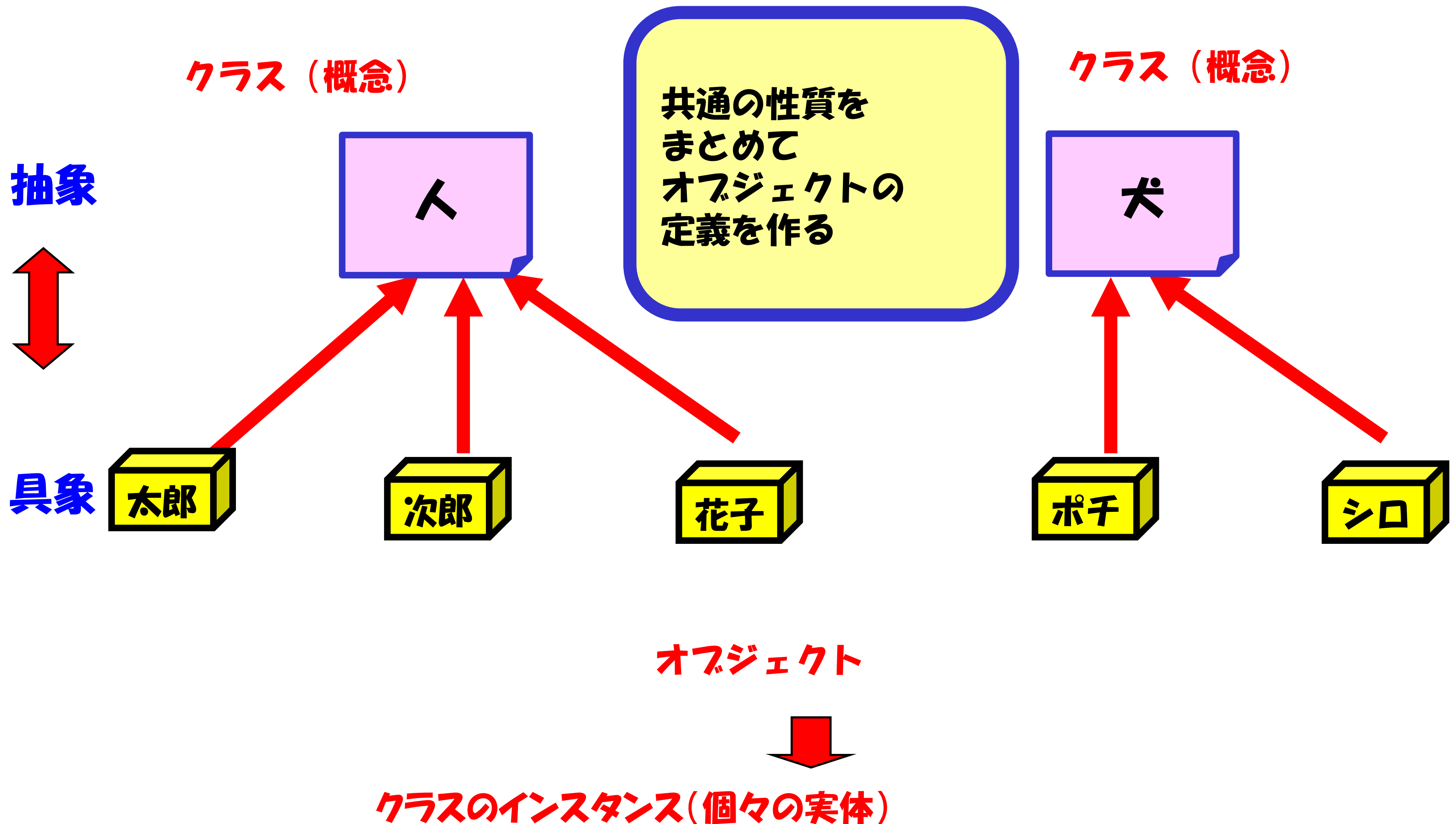


オブジェクト(インスタンス)

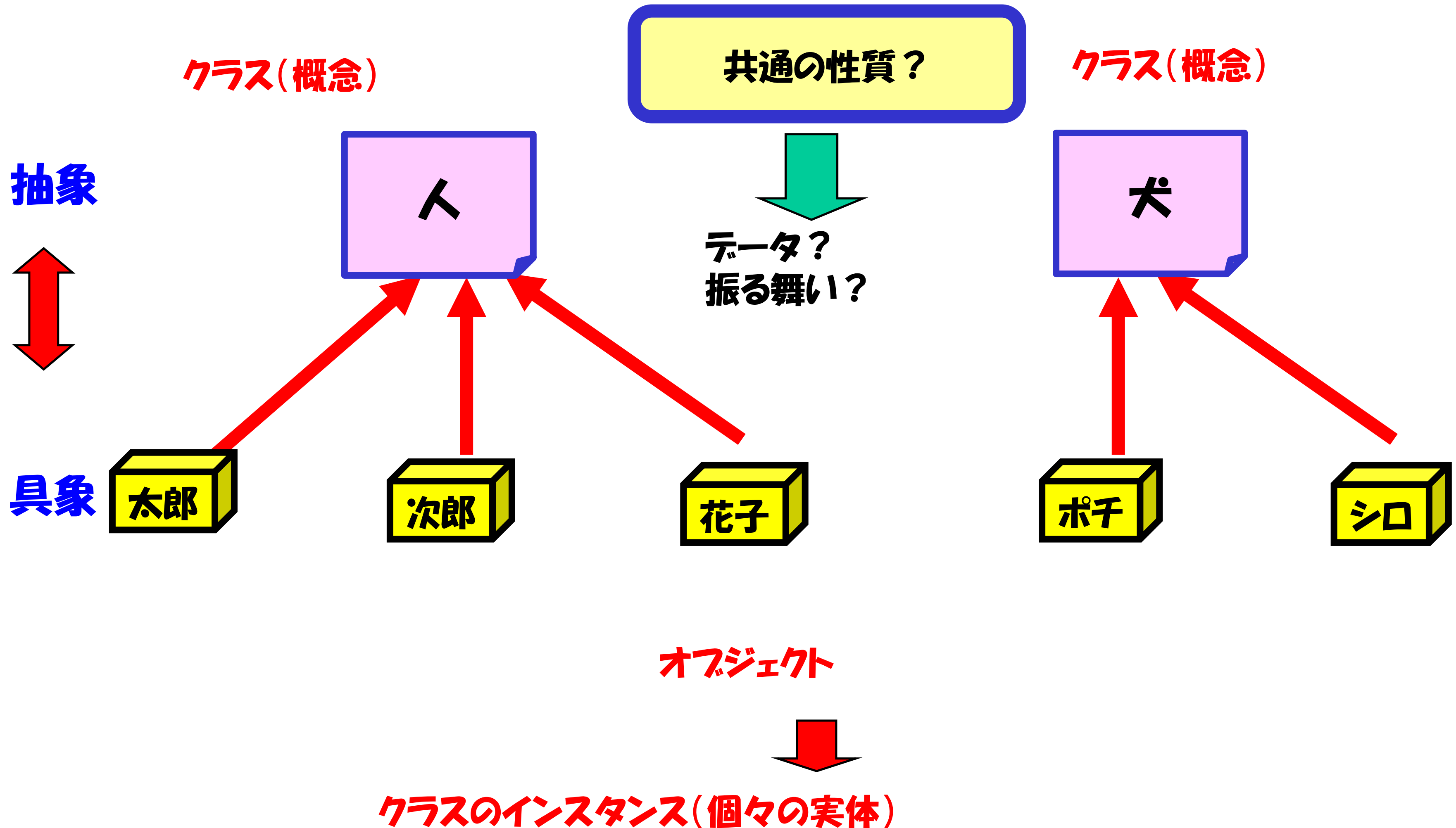


オブジェクト(インスタンス)

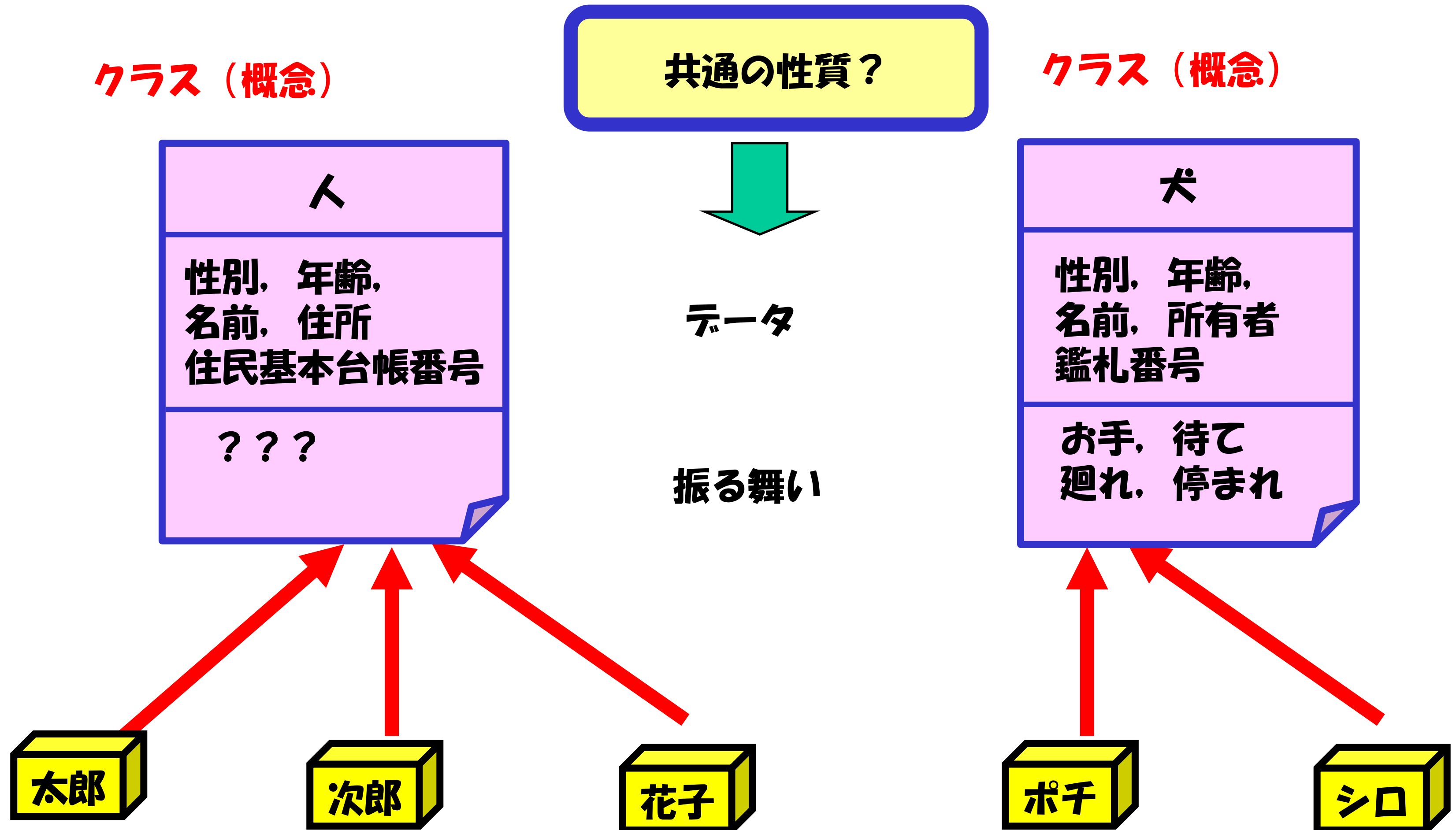
クラスとインスタンス(1/2)



クラスとインスタンス(1/2)



クラスとインスタンス(1/2)



クラスとインスタンス(2/2)

クラス＝テンプレート（定規）→枠組，仕様

宣言とか定義

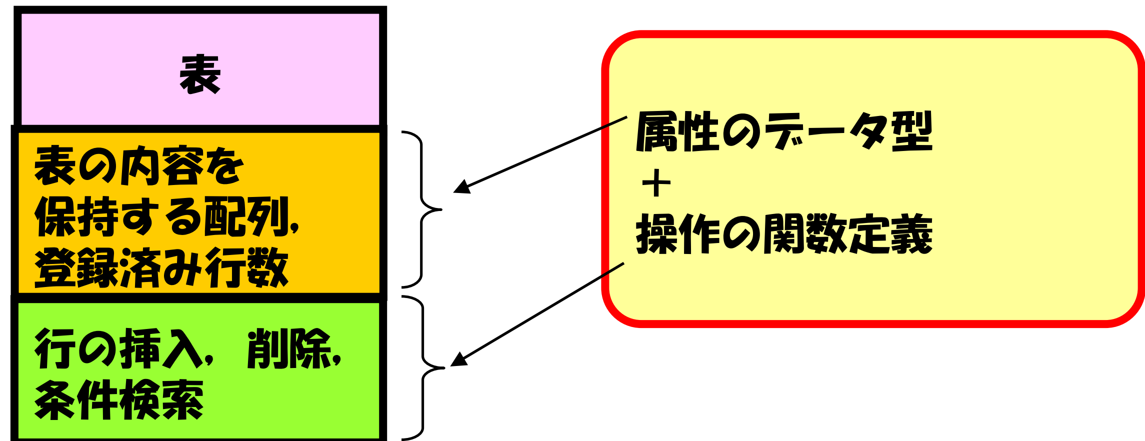
表

属性のデータ型
＋
操作の関数定義

クラスとインスタンス(2/2)

クラス＝テンプレート（定規）→枠組，仕様

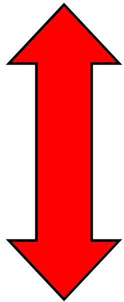
宣言とか定義



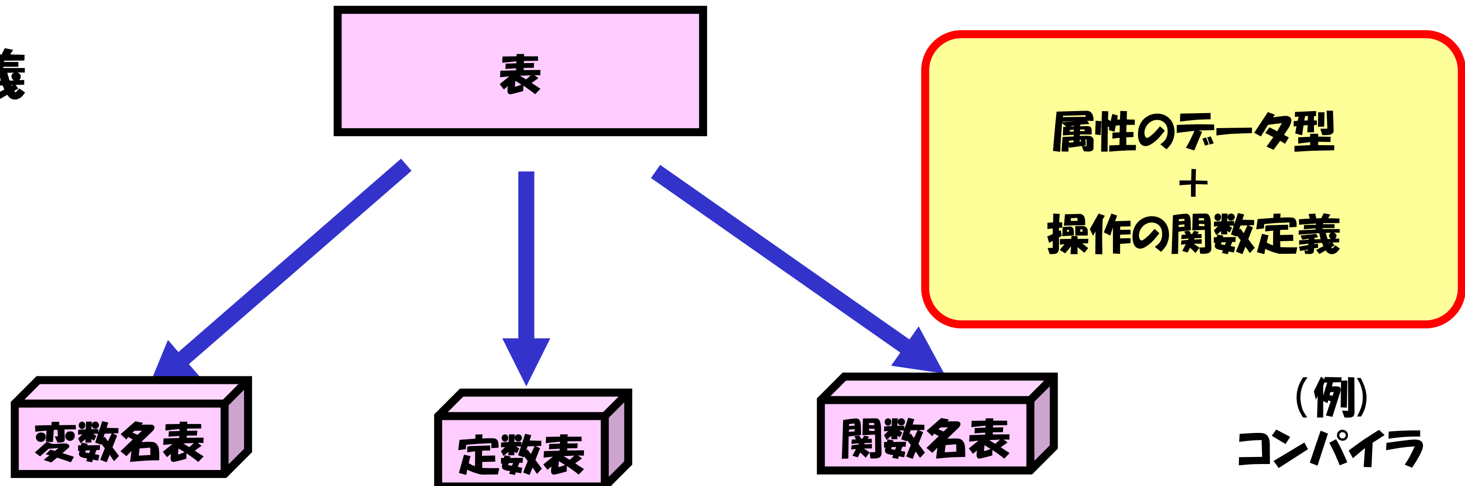
クラスとインスタンス(2/2)

クラス＝テンプレート(定規)→枠組，仕様

宣言とか定義



個々の値



(例)
コンパイラ

インスタンス＝実体→実際のデータが入っている

同じカテゴリに属す(同じ性質を持つ)オブジェクトは，
クラスから生成される(同じクラスに属する)

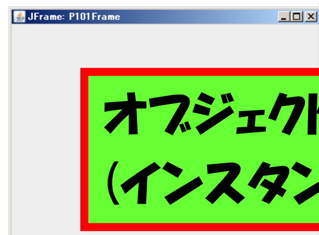
オブジェクトはクラスから作る

クラスに対して，
クラスから生成した
オブジェクトのことを
インスタンス
（実例/実体）
と呼ぶ

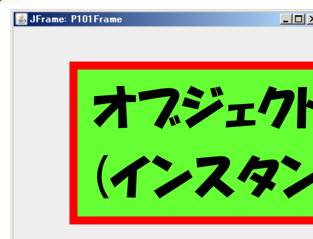
クラス定義

属性定義
- 位置
- 大きさ
- タイトルバーの文字列

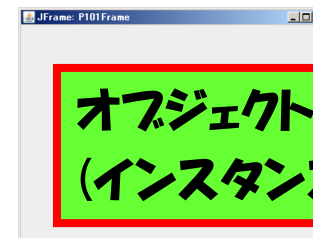
メソッド定義
- 大きさの変更
- 位置の変更
- タイトルバーの変更



オブジェクト
（インスタンス）

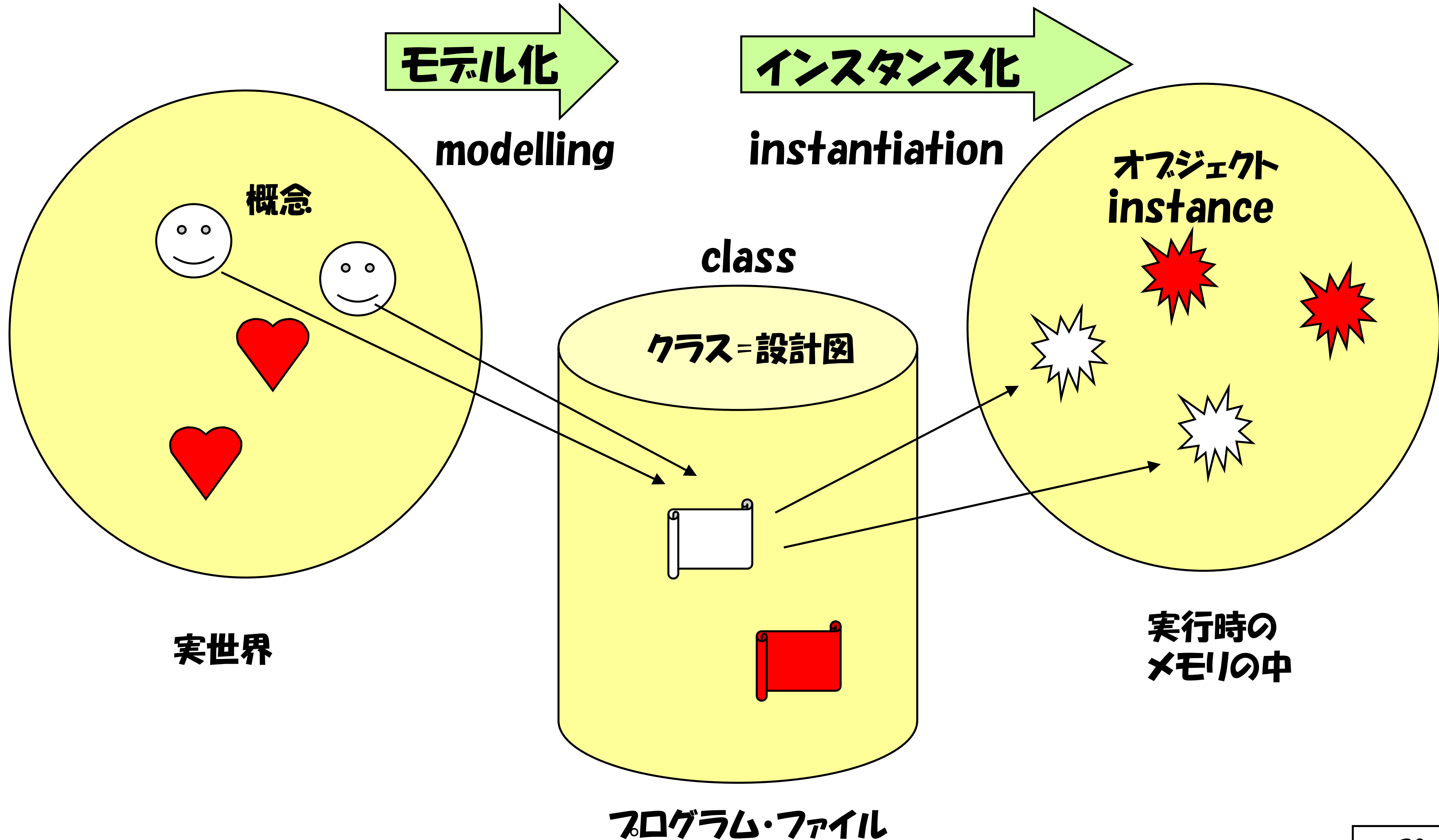


オブジェクト
（インスタンス）



オブジェクト
（インスタンス）

概念→クラス→オブジェクト



例: エレベータのモデル化("what")
～オブジェクトとクラス～

- 個々のエレベータの籠(1号機, 2号機)はオブジェクト
- 共通仕様をモデル化したクラスから生成したインスタンス
- 個々のオブジェクト(インスタンス)は,
  それぞれに固有の状態を持つ

エレベータの共通仕様⇒クラス

属性:番号
属性:階
属性:目標階リスト
属性:移動方向(上, 下, 停止)
属性:ドア(開, 閉)

生成時に決まる属性(属性:番号)

途中で変化する状態(属性:階, 目標階リスト, 移動方向, ドア)

1号機

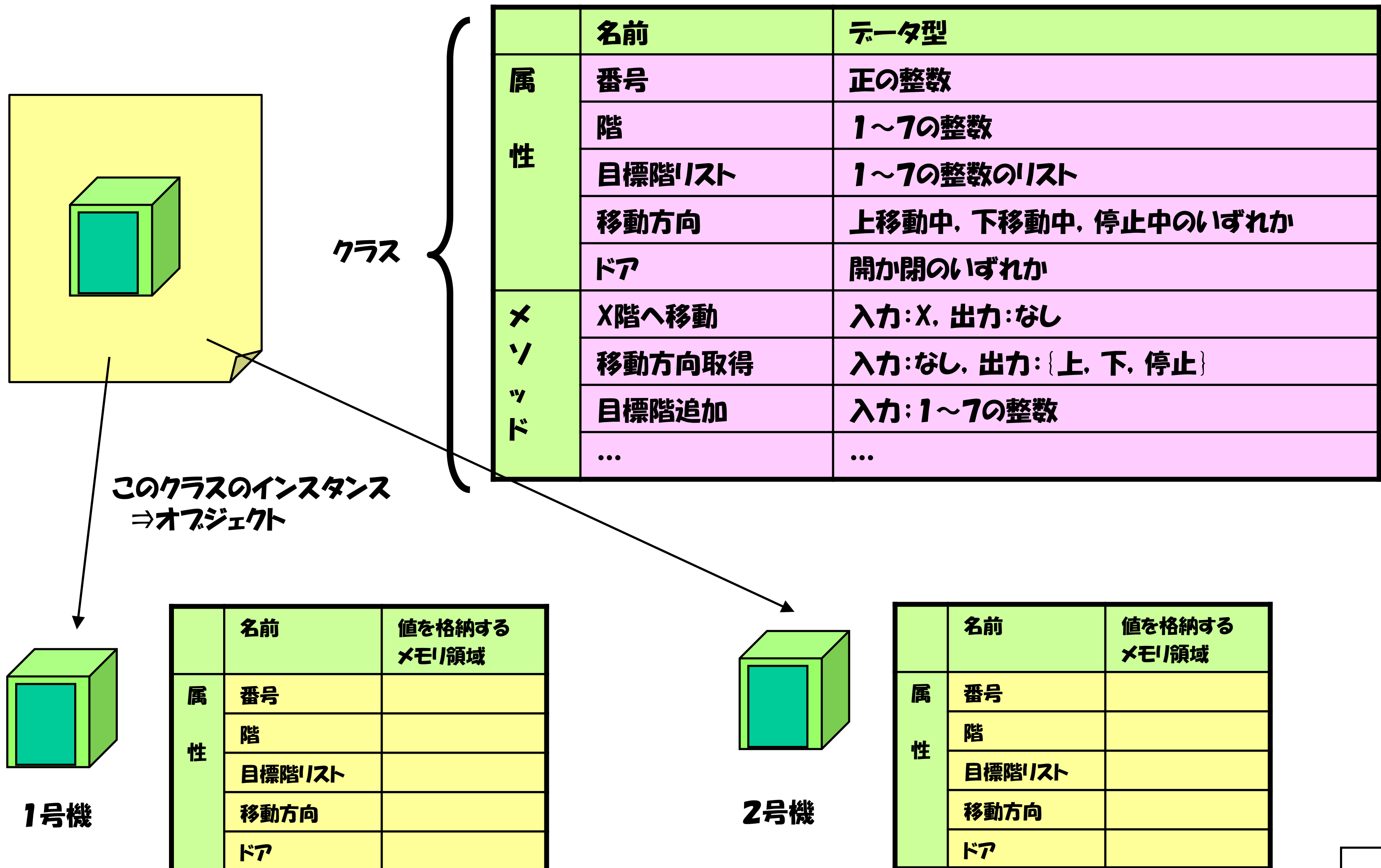
属性:番号=1
属性:階=3
属性:目標階リスト={6,7}
属性:移動方向=上移動中
属性:ドア=閉

2号機

属性:番号=2
属性:階=1
属性:目標階リスト={}
属性:移動方向=停止中
属性:ドア=開

例: エレベータのモデル化(“what”)

～オブジェクトとクラス～



	名前	データ型
属性	番号	正の整数
	階	1～7の整数
	目標階リスト	1～7の整数のリスト
	移動方向	上移動中，下移動中，停止中のいずれか
	ドア	開か閉のいずれか
メソッド	X階へ移動	入力:X，出力:なし
	移動方向取得	入力:なし，出力:{上，下，停止}
	目標階追加	入力:1～7の整数
	…	…

1号機

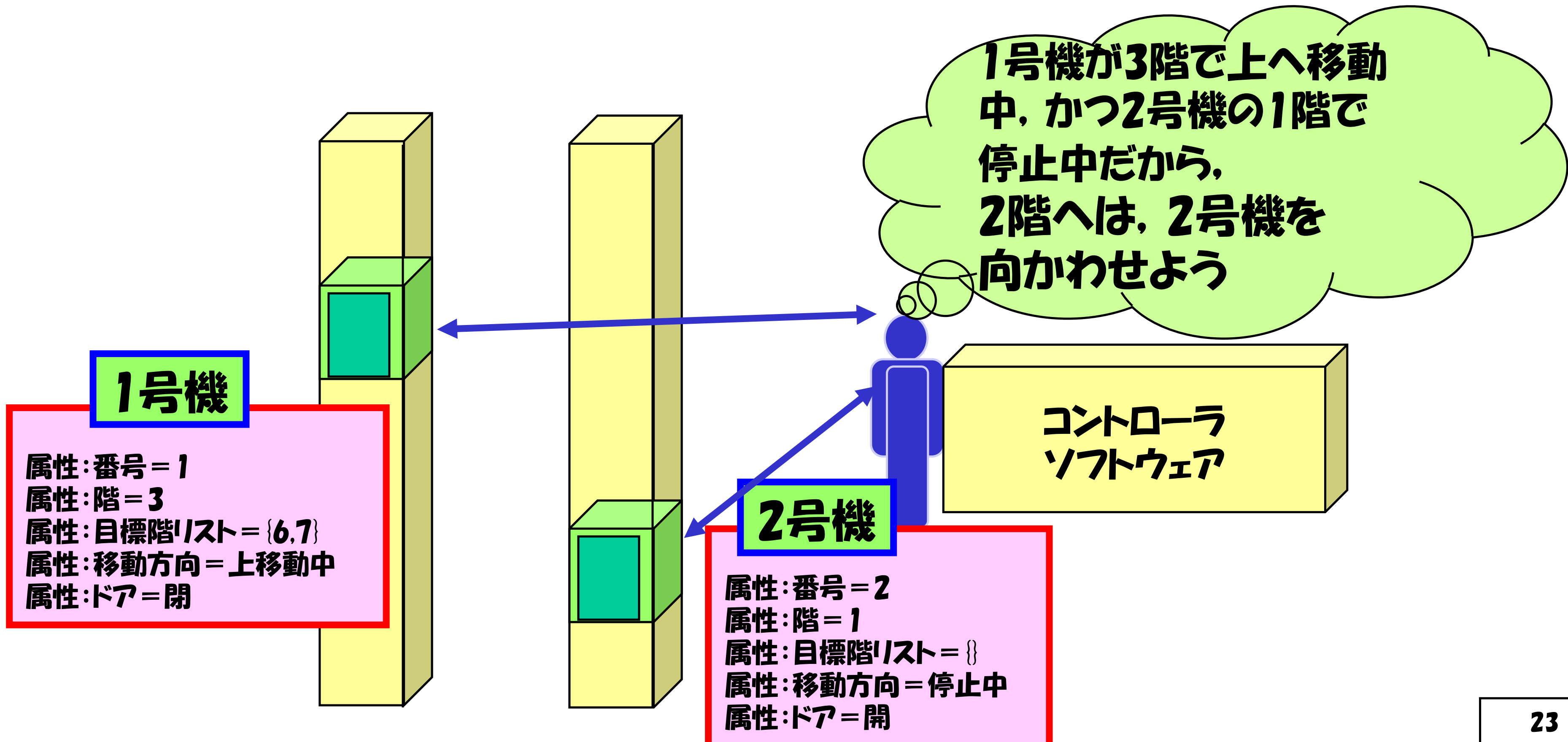
	名前	値を格納するメモリ領域
属性	番号	
	階	
	目標階リスト	
	移動方向	
	ドア	

2号機

	名前	値を格納するメモリ領域
属性	番号	
	階	
	目標階リスト	
	移動方向	
	ドア	

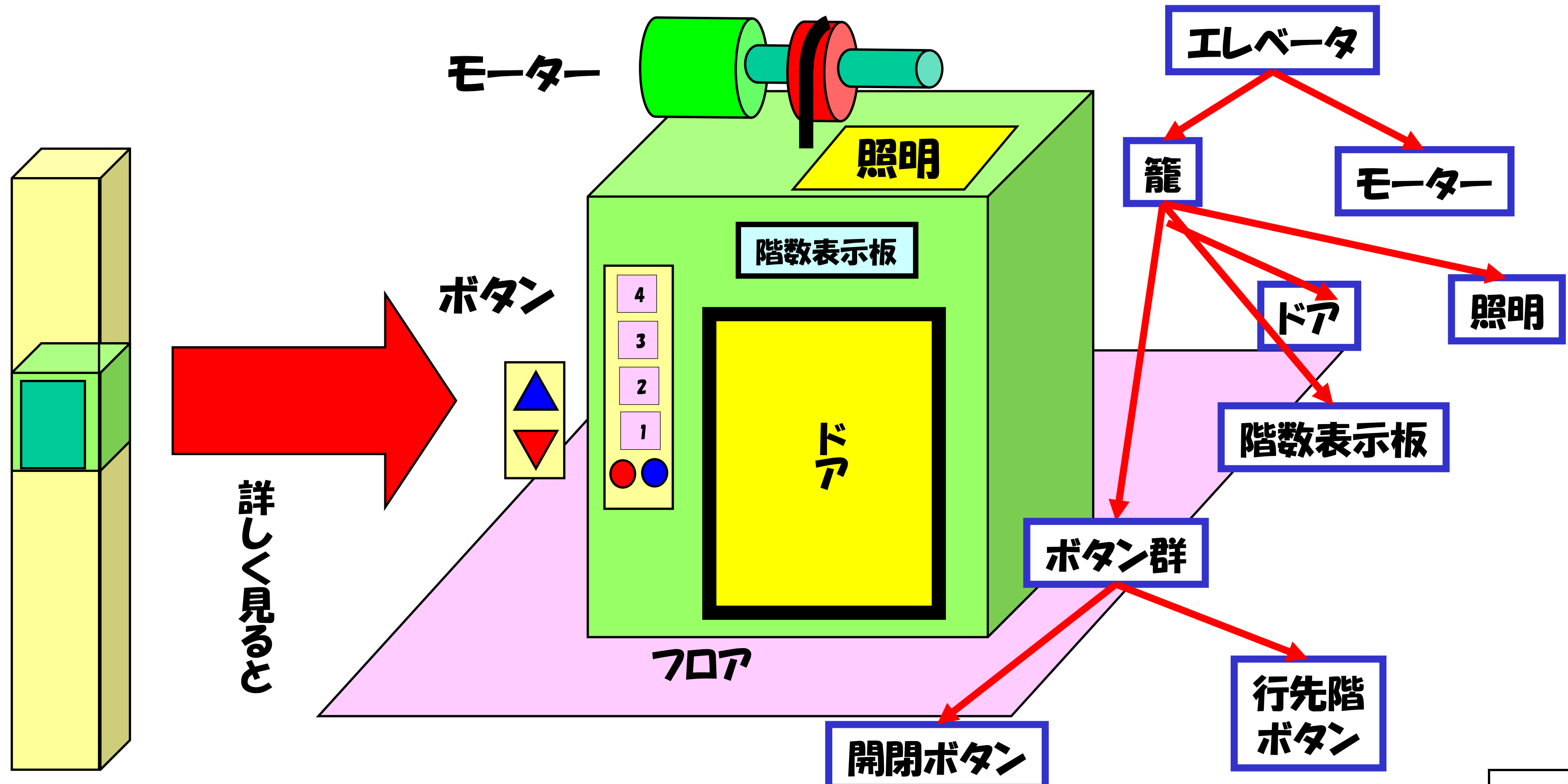
例: エレベータのモデル化("what")
～オブジェクトとクラス～

- コントローラのプログラム
  ⇒シミュレーションが含まれている



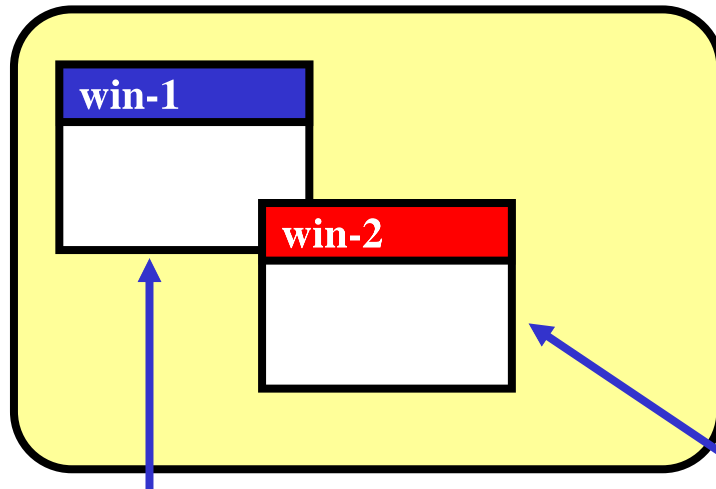
例:エレベータのモデル化⇒分析を続けて
～最初は粗くモデル化して，少しずつ詳細化していく～

- 幾らでも詳細化できてしまうが，必要最小限＋α（将来を見越して…）にとどめる



例:ウィンドウ・システムのモデル化
～オブジェクトとクラス～

- 画面に表示されている個々のウィンドウはオブジェクト
- 共通仕様をモデル化したクラスから生成したインスタンス
- 個々のオブジェクト(インスタンス)は，それぞれに固有の状態を持つ



生成時に決まる属性

途中で変化する状態

ウィンドウの共通仕様⇒クラス

属性:番号
属性:位置(x座標,y座標)
属性:大きさ(縦，横)
属性:タイトル・バーの色
属性:タイトル・バーの文字列
属性:アイコン化状態

属性:番号＝1
属性:位置＝(10,10)
属性:大きさ＝(60,100)
属性:タイトル・バーの色＝青
属性:タイトル・バーの文字列＝"win-1"
属性:アイコン化の状態＝展開

属性:番号＝2
属性:位置＝(100,100)
属性:大きさ＝(60,100)
属性:タイトル・バーの色＝赤
属性:タイトル・バーの文字列＝"win-2"
属性:アイコン化の状態＝展開

クラスとオブジェクト（定義と実体）
～オブジェクトはそれぞれが値を持つ～

定義

生成

実体

クラス

変数宣言：name

変数宣言：age

変数宣言：height

変数宣言：weight

メソッド定義：growAge(y)

メソッド定義:growFat(w)

メソッド定義

オブジェクト

name: “Yamada”

Weight: 60Kg

オブジェクト

name: “Suzuki”

Weight: 40Kg

……

クラスとオブジェクト(定義と実体)
～オブジェクトはそれぞれがメソッドを持つ～

定義

生成

実体

クラス

変数宣言:name

変数宣言:age

変数宣言: height

変数宣言: weight

メソッド定義: growAge(y)

メソッド定義:growFat(w)

メソッド定義

オブジェクト

name: "Yamada"

Weight: 60Kg

growFat(2)

オブジェクト

name: "Suzuki"

Weight: 40Kg

growFat(3)

………

クラスとオブジェクト(定義と実体)
～メソッドは所属オブジェクトのデータを書き換える～

定義

生成

実体

クラス

変数宣言:name

変数宣言:age

変数宣言: height

変数宣言: weight

メソッド定義: growAge(y)

メソッド定義:growFat(w)

メソッド定義

オブジェクト

name: "Yamada"

Weight:
60→62Kg

growFat(2)

オブジェクト

name: "Suzuki"

Weight:
40→43Kg

growFat(3)

....

クラスは，オブジェクトの定義です

- まず
  自前で，成績表クラス
  を作ってみましょう
- 表は，各行をオブジェクトとする配列とします．
- （後々，コレクションフレームワークの
  可変長配列List(ArrayList)で置き換えましょう）

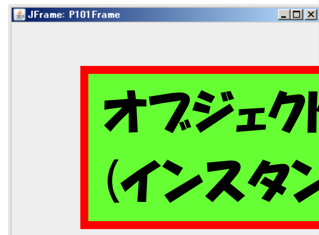
オブジェクトはクラスから作る

クラスに対して,
クラスから生成した
オブジェクトのことを
インスタンス
(実例/実体)
と呼ぶ

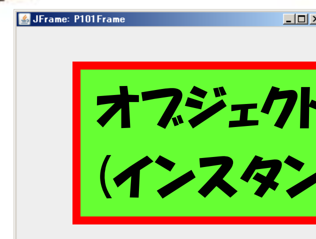
クラス定義

属性定義
- 位置
- 大きさ
- タイトルバーの文字列

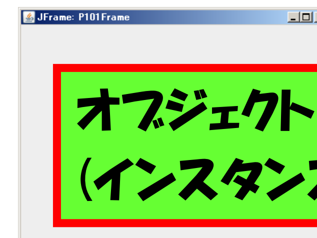
メソッド定義
- 大きさの変更
- 位置の変更
- タイトルバーの変更



オブジェクト
(インスタンス)



オブジェクト
(インスタンス)



オブジェクト
(インスタンス)

成績表クラス

```
import java.io.*;

public class ScoreTable {
    private int maxSize = 0;
    private int size    = 0;
    private Entry[] table = null;

    public ScoreTable( int n ) {
        maxSize = n;
        table = new Entry[maxSize];
        size = 0;
    }
```

変数，属性，プロパティ

メソッド，関数

コンストラクタという特殊なメソッド
（オブジェクト生成時の初期設定に
使われる）

成績表クラス

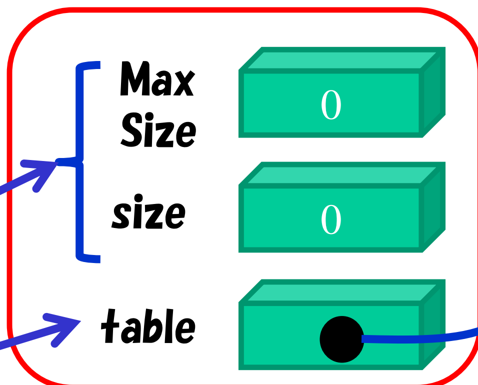
```
import java.io.*;

public class ScoreTable {
    private int maxSize = 0;
    private int size    = 0;
    private Entry[] table = null;
```

基本データ型であるint型

変数，属性，プロパティ

配列は参照型(配列は，newキーワードを使って動的にメモリ空間に確保される)

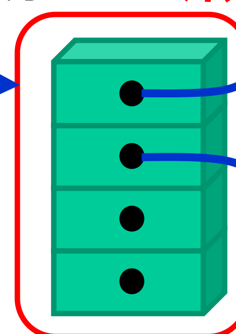
ScoreTableクラスのインスタンス



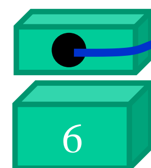
基本データ型

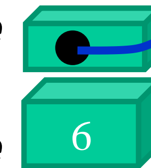
参照型

配列



Entryクラスのインスタンス





"山田"

"田中"

Stringクラスのインスタンス

成績表クラス

メソッド，関数

```
public void add( String name, int math ) {
    if ( size < maxSize ) {
        table[size] = new Entry( name, math );
    }
    size++;
}
```

成績表の操作
（追加）

```
public float averageMath() {
    int sum = 0;
    for ( int i = 0; i < size; i++ ) {
        sum += getMathScore(i);
    }
    return( (float)sum / size );
}
```

成績表の操作
（数学の平均点）

成績表クラス

```
public void printTable() {
    for ( int i = 0; i < size; i++ ) {
        System.out.printf( "%2d %10s %2d¥n", i+1, getName(i),
                                                getMathScore(i) );
    }
}
```

成績表の操作（表示）

```
public String getName( int i ) {
    return( table[i].getName() );
}
```

成績表の操作（名前の取得）

```
public int getMathScore( int i ) {
    return( table[i].getMathScore() );
}
```

成績表の操作（数学の得点の取得）

成績表クラス

```
public static void main( String args[] ) {
    ScoreTable tbl = new ScoreTable( 5 );
    tbl.add( "山田", 3 );
    tbl.add( "田中", 4 );
    tbl.add( "伊藤", 2 );
    tbl.add( "小林", 1 );
    tbl.add( "小泉", 3 );
    tbl.printTable();
    System.out.println( "数学の平均点=" + tbl.averageMath() );
  }
}
```

メインメソッド
（staticメソッド）

成績表クラスのエントリ(レコード)

```
import java.io.*;

public class Entry {
    private String name     = null;
    private int    mathScore = 0;
```

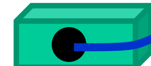
Stringクラス参照型

変数，属性，プロパティ

基本データ型

Entryクラスのインスタンス

name

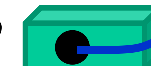


Math score

"山田"

Stringクラスのインスタンス

name



Math score

"田中"

成績表クラスの行(レコード)

```
public Entry( String n, int m ) {
    name      = n;
    mathScore = m;
}
```

コンストラクタ

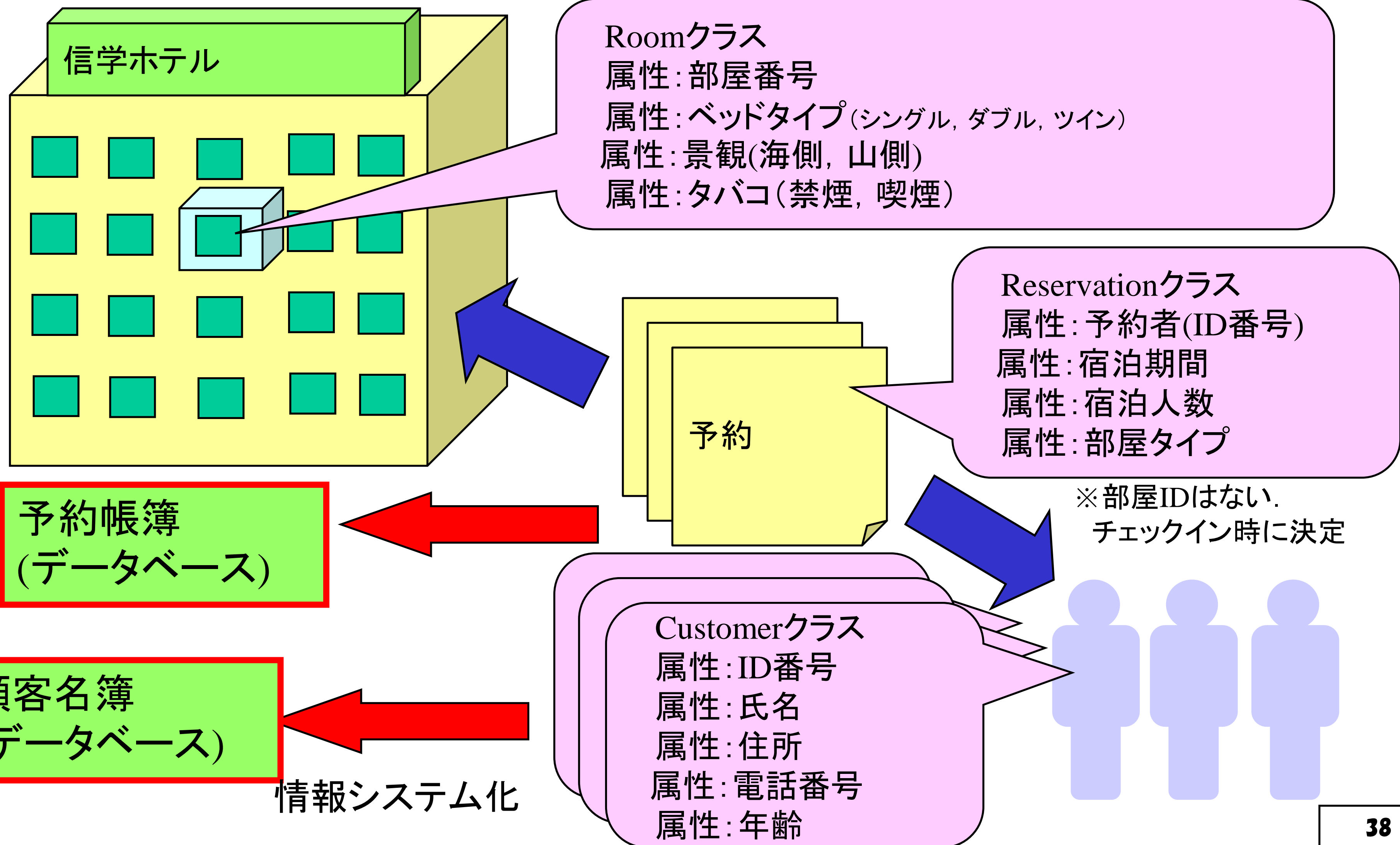
```
public String getName() {
    return( name );
}
```

メソッド:行(レコード)の操作

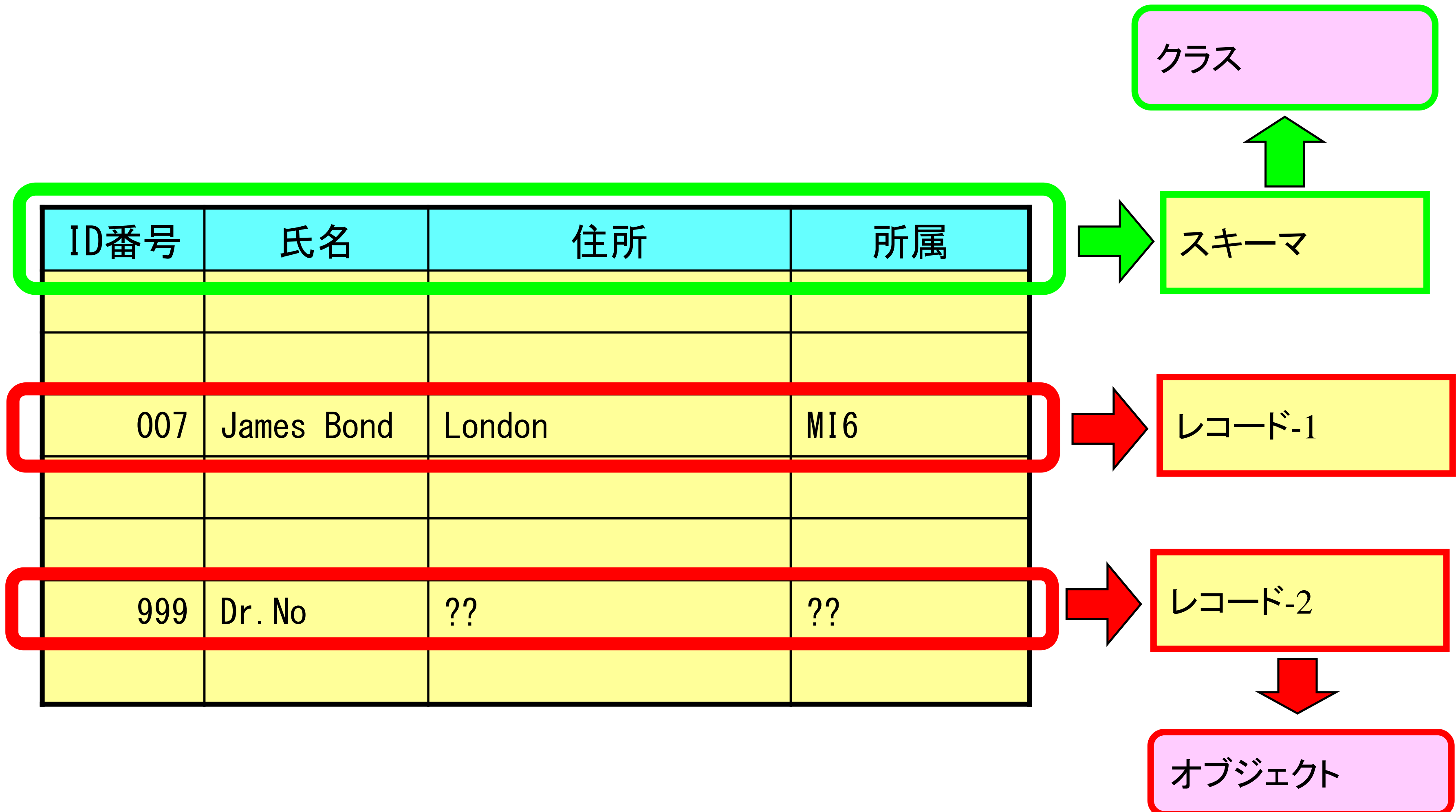
```
public int getMathScore() {
    return( mathScore );
}
}
```

メソッド:行(レコード)の操作

例: ホテルの予約システム

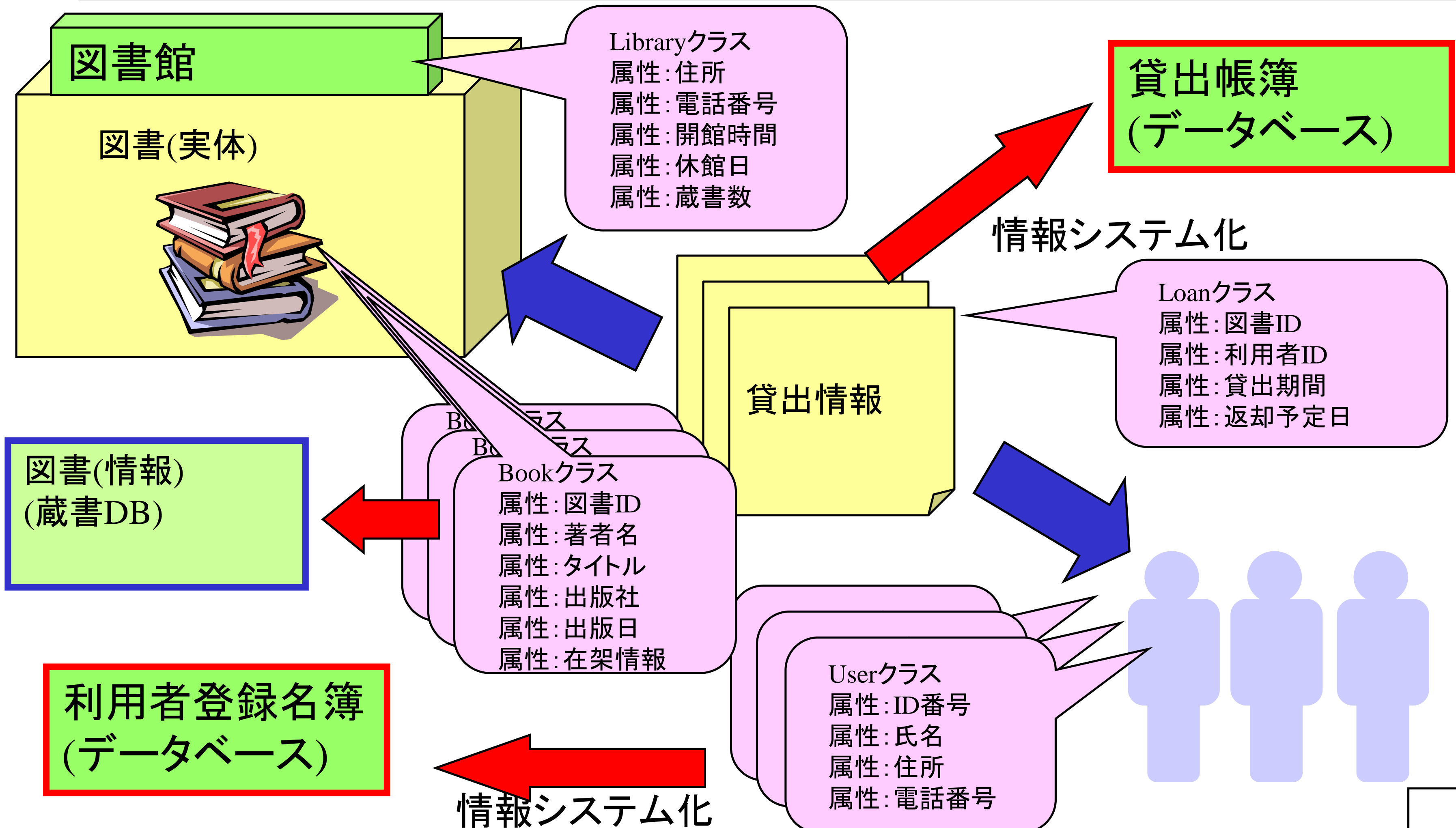


表のスキーマとレコード



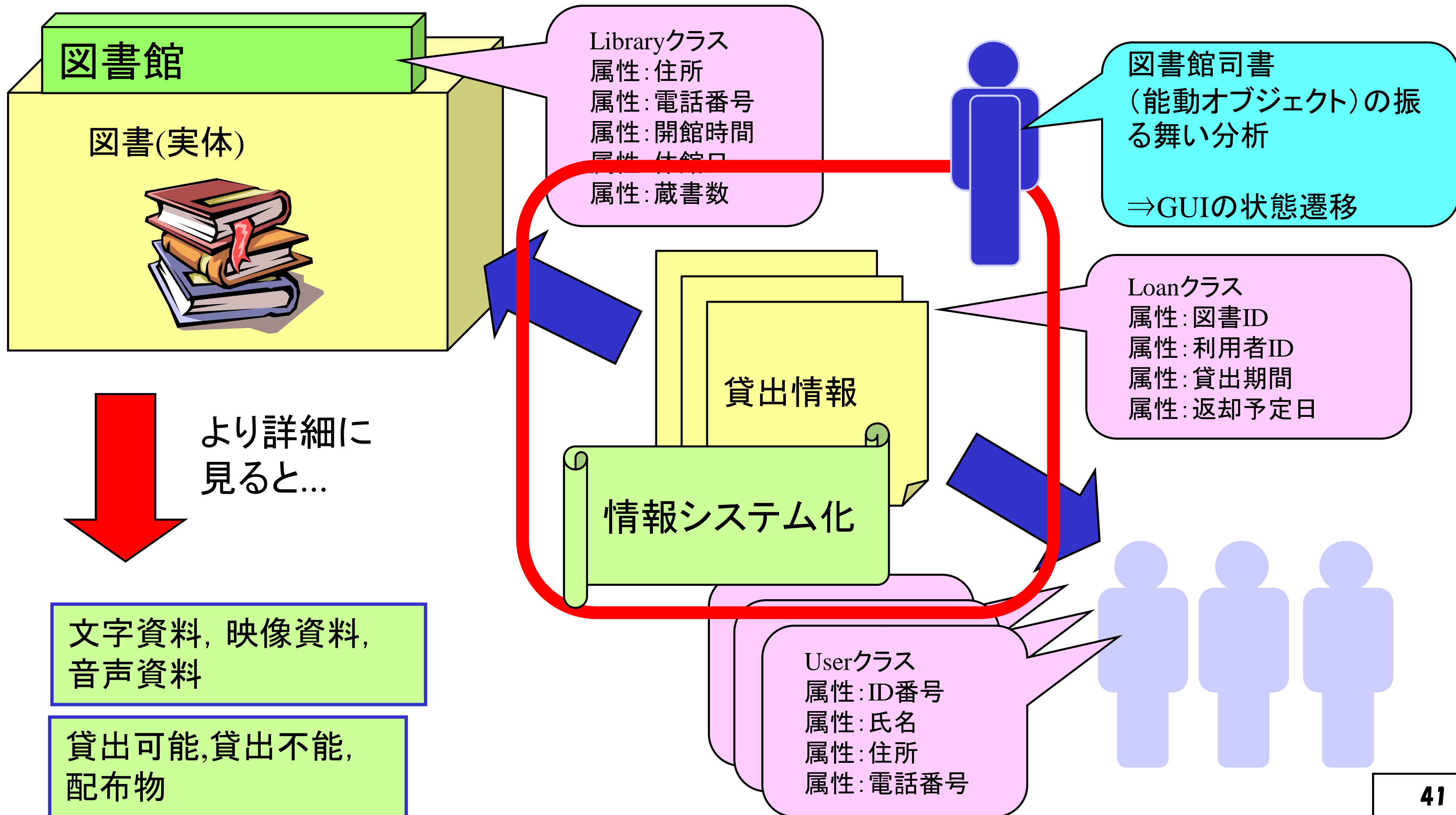
例：図書館の貸し出しシステム

- ドメイン(問題領域)分析によって関連要素を抽出する



例：図書館の貸し出しシステム

- 図書館の司書の振る舞い⇒貸出システムのGUIの状態遷移のモデル化



クラスは，オブジェクトの定義です

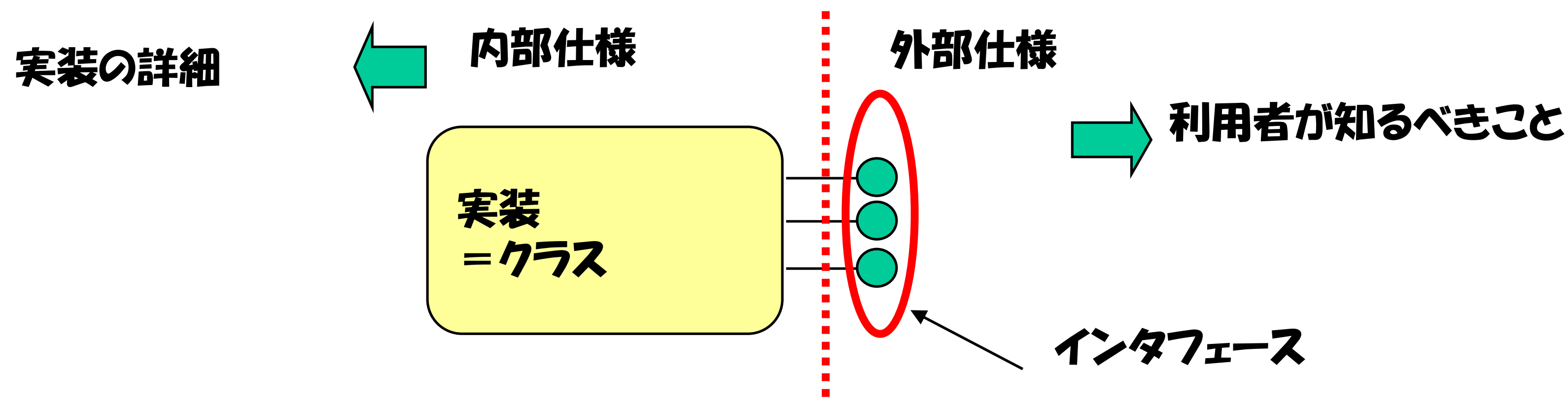
- 今度は，
  新たにオブジェクトを定義する
  のではなく，
  既定義のオブジェクトをうまく使ってみる
  ということをしましょう．

- 具体的には，
  コレクションフレームワークの一種
  List(ArrayList)とMap(HashMap)
  を使って，配列と置き換えます．

参考：インタフェース

インタフェースの概念

- インタフェースと実装の分離
  - インタフェースとクラス
    - 外部仕様と内部仕様
  - 多重インタフェース
- 分散オブジェクトの影響
  - IDL(Interface Definition Language)

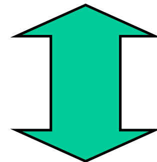


【参考】Javaでのインタフェースとクラス

```
interface StackIF {
        Object pop();
        void push( Object x );
}
```

インタフェース

外部から呼び出せるメソッド情報だけ



```
class Stack implements StackIF{
        Vector   v;
        Object pop() {…};
        void push( Object x ) {…};
}
```

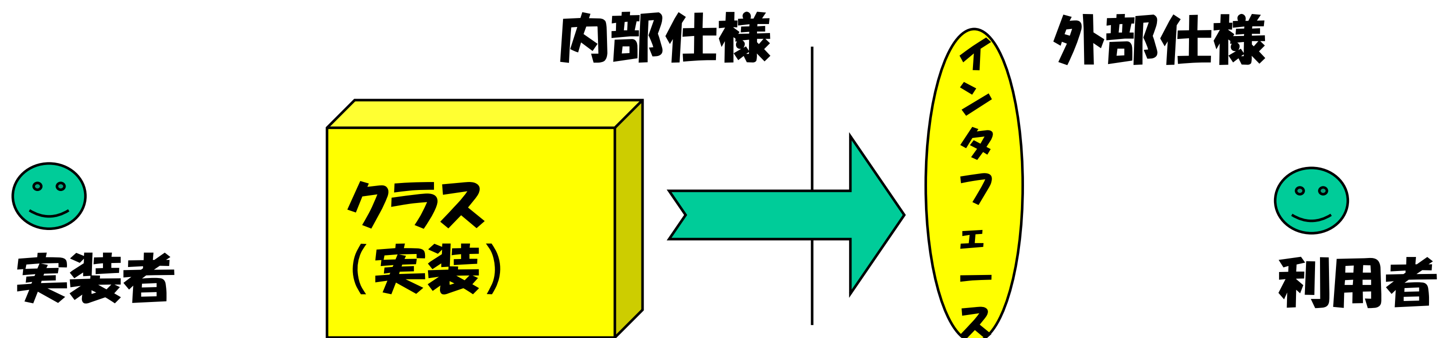
クラス

属性情報,
メソッドの定義,
内部的にしか使わないメソッド情報を含む

【参考】Javaでのインタフェース指定

```
class XXX implements YYY {

                    …

}
```

- インタフェースYYYは，クラスXXXの外部仕様
- クラス(コード)XXXは，インタフェースYYYの実装



【参考】Javaでのインタフェースの利用

- クラスと同じように参照**型**として利用できる.
  - 受け付けられるメソッドが何なのかを規定している

【参考】Javaでの抽象クラスとインタフェースの使い分け

- クラスは単一継承，インタフェースは多重継承
  - インタフェースを複数指定しても，実装を継承しないので問題ない
- インタフェースは，複数指定できる（多重インタフェース）
  - 一つのオブジェクトへの複数の観点を与える
- 抽象クラスには，抽象メソッドだけでなく，具象メソッド(?!)も書ける
  - インタフェースには実装を書くことはできない

コレクションフレームワーク(1)：
List：可変長の配列(のようなもの)

List : リスト

追加	void	add(int, Object)	引数intで指定された位置に，要素を追加する．
	boolean	add(Object)	リストの最後に要素を追加する．
	boolean	addAll(Collection)	リストの最後に，引数Collectionで指定された要素すべてを追加する
	boolean	addAll(int, Collection)	引数intで指定された位置に，引数Collectionで指定された要素すべてを追加する．
	void	clear( )	リストからすべての要素を削除する．
	Object	get(int)	リスト内のインデックス番号で指定された位置にある要素を返す．
	Object	remove(int)	リスト内のインデックス番号で指定された位置にある要素を削除する． 返り値として，削除された要素を返す．
	Object	set(int, Object)	リスト内のインデックス番号で指定された位置にある要素を引数Objectで指定された要素に置き換える．返り値として，置き換えられた古い要素を返す．
	int	size( )	リスト内にある要素の数を返す．

成績表：最低点を取った人(複数名)をリストで保管

- 最低点を取る人は1名とは限りません.
- 最悪，全員が同点なら，全員が最低点（最高点）を取ったことになります.
- 以前，最低点を取った人が複数いる場合，全員の学籍番号を表示するプログラムを作りました.

成績表：最低点を取った人(複数名)をリストで保管

```
public class K003Min3 {
    public static void main( String args[] ) {
        int tokuten[] = {5,7,5,3,8,3,9,3};
        int minId[]   = new int[tokuten.length]; //最低点取得者の学籍番号 (全員同点の場合，全員が最低点)
        int min       = tokuten[0]; // 最低点の候補
        int numMin    = 1; // 最低点取得者数
        minId[0] = 1;
        System.out.println( "学籍番号[1] ⇒" + tokuten[0] +"点" );
        for ( int i = 1; i < tokuten.length; i++ ) {
            System.out.println( "学籍番号[" + (i+1) + "] ⇒" + tokuten[i] +"点" );
            if ( tokuten[i] < min ) {
                min      = tokuten[i];
                minId[0] = i+1;
                numMin   = 1;
            } else if ( tokuten[i] == min ) {
                numMin++;
                minId[numMin-1] = i + 1;
            }
        }
        System.out.println( "最低点は" + min );
        for ( int i = 0; i < numMin; i++ ) {
            System.out.println( "最低点をとったのは" + minId[i] );
        }
    }
}
```

学籍番号1(添字0)の人が最初の最低点候補)

最低点の更新，最低点取得者リストのリセット

最低点取得者リストへの追加

学籍番号2(添字1)の人以降でより低い点を取った人がいないか，同じ最低点を取っている人はいるか，をチェックする．

課題：
成績表：最低点を取った人(複数名)をリストで保管

- 最低点を取った人の配列を
  リストで表現しましょう.

Collections Framework インタフェースと実装の関係

		実装				
		ハッシュテーブル	サイズ変更可能な配列	バランスツリー	リンクリスト	Hash Table + Linked List
I/F	Set	HashSet		TreeSet		LinkedHashSet
	List		ArrayList		Linked List	
	Map	HashMap		Tree Map		LinkedHashMap

インタフェース

Collection	オブジェクトの集まり
Set	重複要素のないオブジェクトの集まり(集合)
List	順序付けられたオブジェクトの集まり(リスト)．重複を許可する． リスト内のどこに各要素が挿入されるかを制御することができ、 利用者は位置(添え字)を指定して各要素にアクセスできる．
Map	キーを値に対応付ける写像． キーは重複して登録することはできない． 各キーはそれぞれ一つの値に写像する．
SortedSet	要素が自然順序付けに従って昇順にソートされた集合を表す． 挿入される要素は，Comparable インタフェースを実装するか， 指定された Comparator によって受け付けられる必要があある．
SortedMap	キーが自然順序付けに従って昇順にソートされたマップを表す． 挿入されるキーは，Comparable インタフェースを実装するか， 指定された Comparator によって受け付けられる必要がある．

実装

Set	HashSet	HashMap のインスタンスに基づくSet インタフェースを実装する. 繰り返し順序について保証しない.
	TreeSet	TreeMap のインスタンスに基づくSet インタフェースを実装する. 要素の昇順でソートされる.
	LinkedHashSet	繰り返し順序を持つSet インタフェースのハッシュテーブルとリンクリストの実装である. 要素がセットに挿入された順序を保持する. 要素をセットに再挿入する場合、挿入順は影響を受けない.
List	ArrayList	Listインタフェースのサイズ変更可能な配列の実装. 配列のサイズを操作するメソッドを提供する.
	LinkedList	List インタフェースのリンクリストの実装. リストの先端および終端にある要素を取得、削除したり、先端および終端に要素を挿入したりするメソッド を提供する. 同期化されないことを除いてVectorとほぼ同等.
Map	HashMap	Map インタフェースのハッシュテーブルに基づく実装. マップの順序について保証しない. 同期化されないこととnullを許容することを除いてHashtableとほぼ同等.
	TreeMap	SortedMap インタフェースの実装に基づく Red-Black ツリー. マップがキーの昇順でソートされる.
	LinkedHashMap	繰り返し順序を持つMap インタフェースのハッシュテーブルとリンクリストの実装. キーがマップに挿入された順序を保持する. キーをマップに再挿入する場合、挿入順は影響を受けない.

Set: 集合

Set	重複要素のないオブジェクトの集まりを管理するクラス

HashSet	Setの機能だけ必要な場合
TreeSet	要素を昇順にソートする必要がある場合
LinkedHashSet	要素の挿入順を保持する必要がある場合

Set : 集合

	boolean	add(Object)	引数で指定された要素が，セットに存在しない場合追加される.
	boolean	addAll(Collection)	引数で指定されたコレクションの要素すべてが，セットに存在しない場合追加される.
	void	clear( )	セットからすべての要素を削除する.
	boolean	contains(Object)	引数で指定された要素がセットに存在する場合，trueを返す
	boolean	containsAll(Collection)	引数で指定されたコレクションの要素すべてが、セットに存在する場合，trueを返す
	boolean	remove(Object)	引数で指定された要素が、セットに存在する場合その要素を削除する
	boolean	removeAll(Collection)	引数で指定されたコレクションの要素の内、セットに含まれる要素を削除する
	Object	retainAll(Collection)	引数で指定されたコレクションの要素の内、セットに含まれる要素を、セット内に保持する
	Object[]	toArray( )	セット内のすべての要素が格納されている配列を返す.
	Object[]	toArray(Object[])	セット内のすべての要素が格納されている配列を返す.配列の実行時の型は引数で指定された型になる.

List: リスト

List	順序付けられたオブジェクトの集まりを管理するクラス.
	重複を許可します。リスト内のどこに各要素が挿入されるかを制御することができ, ユーザーは位置(インデックス)を指定して各要素にアクセスすることができる.

ArrayList	List インタフェースのサイズ変更可能な配列の実装. 配列のサイズを操作するメソッドを提供する.
	インデックスによるランダムアクセスの性能が優れているが, 要素の挿入、削除には向いていない.
LinkedList	List インタフェースのリンクリストの実装. リストの先端および終端にある要素を取得、削除したり、先端および終端に要素を挿入したりするメソッド を提供する. 同期化されないことを除いてVectorとほぼ同等である.
	要素の挿入、削除の性能が優れていますが、インデックスによるランダムアクセスには向いていない.

	インデックスアクセス	Iteratorアクセス	追加	挿入	削除
ArrayList	○	○	○	遅	遅
LinkedList	遅	○	○	○	○

List：リスト

追加	void	add(int, Object)	引数intで指定された位置に，要素を追加する．
	boolean	add(Object)	リストの最後に要素を追加する．
	boolean	addAll(Collection)	リストの最後に，引数Collectionで指定された要素すべてを追加する
	boolean	addAll(int, Collection)	引数intで指定された位置に，引数Collectionで指定された要素すべてを追加する．
	void	clear( )	リストからすべての要素を削除する．
	Object	get(int)	リスト内のインデックス番号で指定された位置にある要素を返す．
	Object	remove(int)	リスト内のインデックス番号で指定された位置にある要素を削除する． 返り値として，削除された要素を返す．
	Object	set(int, Object)	リスト内のインデックス番号で指定された位置にある要素を引数Objectで指定された要素に置き換える．返り値として，置き換えられた古い要素を返す．
	int	size( )	リスト内にある要素の数を返す．

Iterator

次要素の有無	boolean	hasNext( )	繰り返し処理において次の要素がある場合に，trueを返す．
次要素	Object	next( )	繰り返し処理において，次の要素を返す．
削除	void	remove( )	繰り返し処理において，呼び出された最後の要素を削除する

```
for (Iterator i = c.iterator(); i.hasNext()) {
        ... 繰り返される処理 ...
}
```

```
Iterator i = c.iterator();
while (i.hasNext() {
        ... 繰り返される処理 ....
}
```

```
for (Iterator i = hs1.iterator(); i.hasNext();) {
        i.next();
        i.remove();
}
```

ListIterator

次要素の有無	boolean	hasNext( )	繰り返し処理において、 次の要素がある場合にtrueを返す。
次要素	Object	next( )	繰り返し処理において、次の要素を返す。
削除	void	remove( )	繰り返し処理において、 呼び出された最後の要素を削除する。

前要素の有無	boolean	hasPrevious( )	繰り返し処理において、 前の要素がある場合にtrueを返す。
前要素	Object	previous( )	繰り返し処理において、前の要素を返す。
追加		add	
変更		set	
		nextIndex	
		previousIndex	

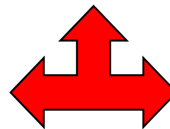
繰り返しと拡張for文

Listの
繰り返し子
Iterator
による
アクセス

```
List names = new ArrayList();
names.add("a");
names.add("b");
names.add("c");

for (Iterator it = names.iterator(); it.hasNext(); ) {
    String name = (String)it.next();
    System.out.println( name );
}
```

Listの添え字
によるアクセス

配列の添え字によるアクセス



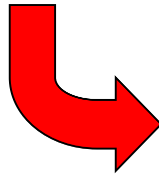
```
String[] names = { "a", "b", "c" };
for ( int i = 0; i < names.length; i++ ) {
    System.out.println( names[i] );
}
```

```
List names = new ArrayList();
names.add("a");
names.add("b");
names.add("c");

for ( int i = 0; i < names.length; i++ ) {
    System.out.println( names.get[i] );
}
```

繰り返しと拡張for文

Listの
繰り返し子
Iterator
による
アクセス

```
List names = new ArrayList();
names.add("a");
names.add("b");
names.add("c");

for (Iterator it = names.iterator(); it.hasNext(); ) {
    String name = (String)it.next();
    System.out.println( name );
}
```



Listの
拡張for文
(JDK-1.5以降)

```
List<String> names = new ArrayList<String>();
names.add("a");
names.add("b");
names.add("c");

for (String name: names) {
    System.out.println( name );
}
```

コレクションフレームワーク(2)：
Map: 写像(ハッシュ表のようなもの)

Map: 写像

Map	キーを値にマッピングする場合に使用するクラス

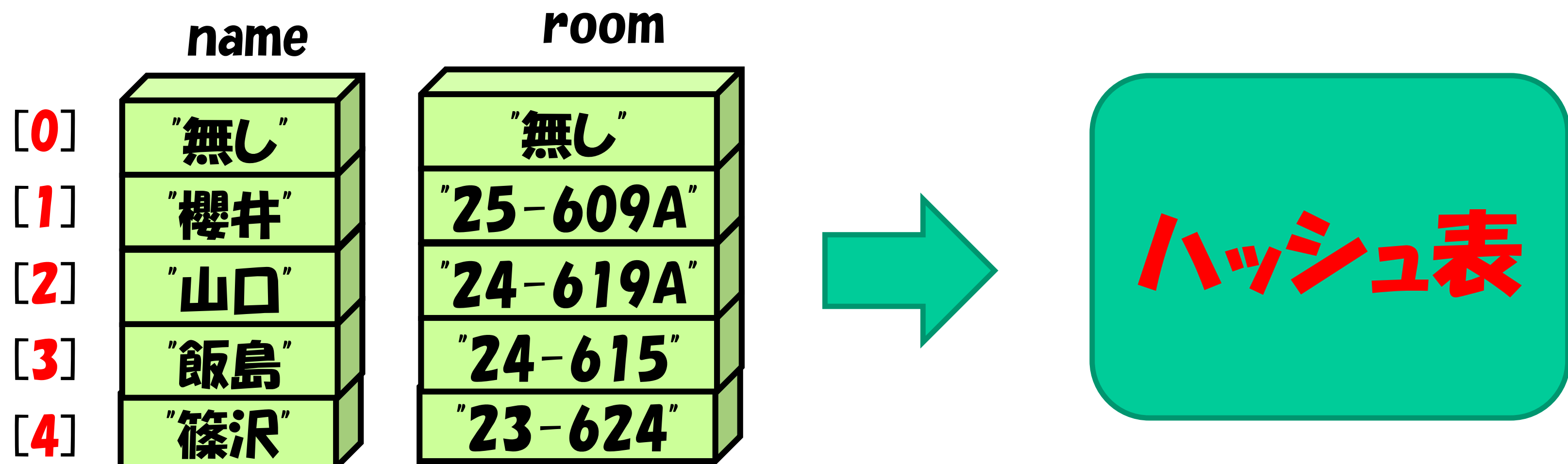
HashMap	Mapの機能だけ必要な場合
TreeMap	キーを昇順にソートする必要がある場合
LinkedHashMap	キーの挿入順を保持する必要がある場合

Map : 写像

	void	clear( )	Mapからすべての要素を削除する.
	boolean	containsKey(Object)	引数で指定されたキーがMapに存在する場合, trueを返す.
	boolean	containsValue(Object)	引数で指定された値がMapに存在する場合, trueを返す.
	Object	get(Object)	引数に指定されたキーに紐付けられた値を返す.
	Object	put(Object, Object)	指定された引数(キー, 値)の対を, Mapに挿入する.
	void	putAll(Map)	引数に指定されたMapの要素すべてを、Mapに挿入する.
	Object	remove(Object)	引数に指定されたキーがMapに存在する場合、そのキーと対応付けられた値を削除する
	Set	entrySet( )	Mapに格納されている要素を持つ, コレクションビューを返す.
	Set	keySet( )	Mapに格納されているキーを持つ, セットビューを返す.
	Collection	values( )	Mapに格納されている値を持つ, コレクションビューを返す.

表の線形探索 →ハッシュ表

- 名前と部屋番号の対応表



Map

キーからの値の検索	get(キー)
登録(既にそのキーが使われていたら,対応する値を返す. さもなくば, nullを返す)	put(キー, 値)
削除	remove(キー)
キーが登録済みかどうか?	containsKey(キー)
大きさを返す	size()
空かどうか?	isEmpty()
空にする	clear()

文字列カウンタ

```
import  java.util.*;

public class MapTest {
    public static void main( String args[] ) {
            CounterMap aMap =
                    new CounterMap();
            aMap.countUp("飯島");
            aMap.countUp("山口");
            aMap.countUp("飯島");
            aMap.countUp("櫻井");
            aMap.countUp("山口");
            aMap.countUp("飯島");
            aMap.print();
    }
}
```

```
class CounterMap extends
                        TreeMap<String,Integer> {
    void countUp( String w ) {
      if ( containsKey( w ) ) {
            put( w, get( w ) + 1 );
      } else {
            put( w, 1 );
      }
    }
    void print() {
        for (String key : keySet() ) {
            System.out.println(key + " => " + get(key));
        }
    }
}
```

次回は. . .

- 次回は, テストもやりますが. . .
- テスト前に, 少し, だけ講義・実習をします.
- 具体的には,
  GUI(グラフィカル ユーザ インタフェース)
  を作ってみましょう.