



ソフトウェア工学実習

Software Engineering Practice

(第05回)

SEP05-001 継承(その1)

継承とは, 抽象クラス, インタフェース

慶應義塾大学・理工学部・管理工学科
飯島 正

ijima@ae.keio.ac.jp

こんにちは.
この授業は,
ソフトウェア
工学実習
です

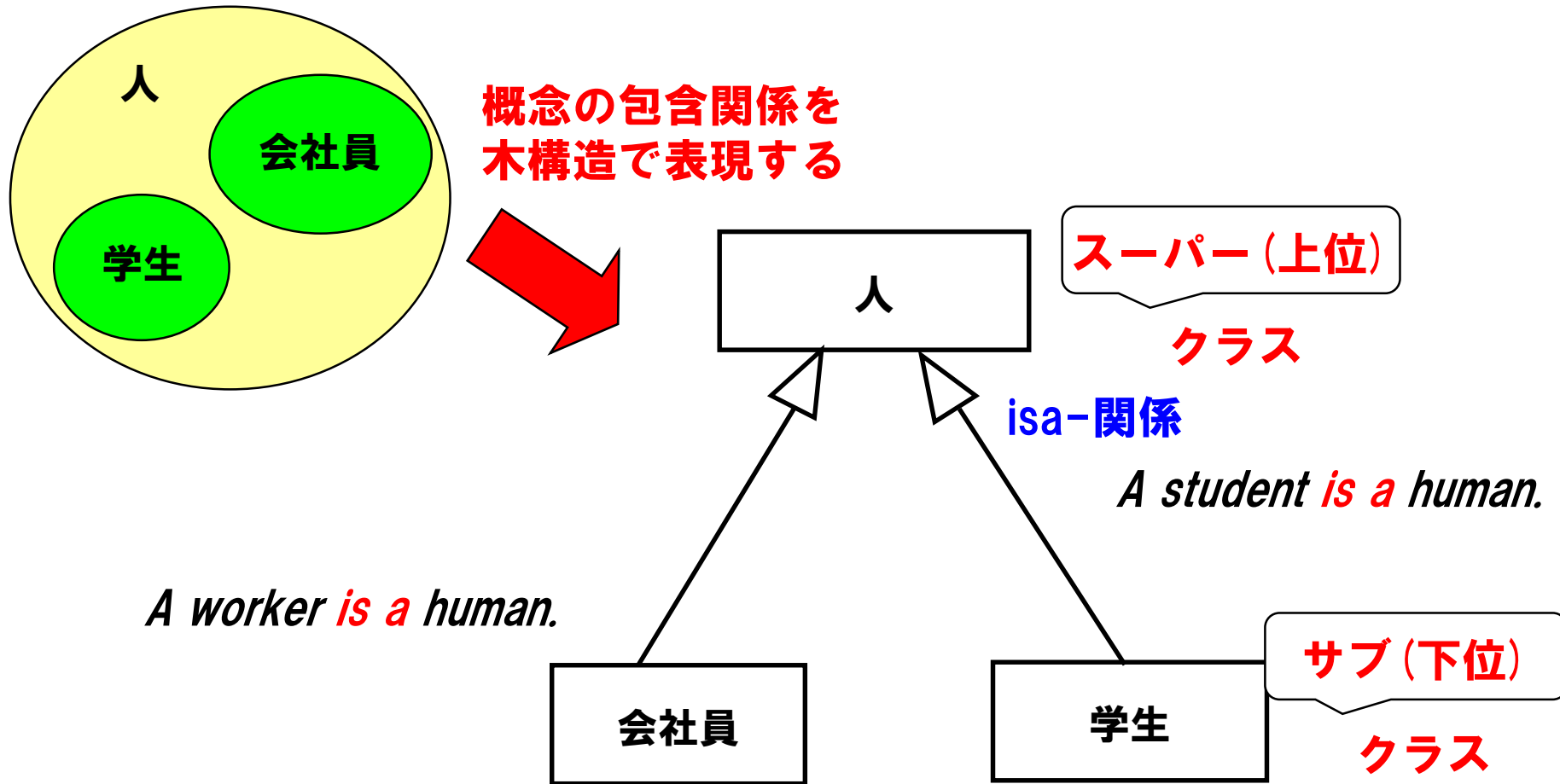


継承とは

今回の話題は、
経書とインタ
フェースです。



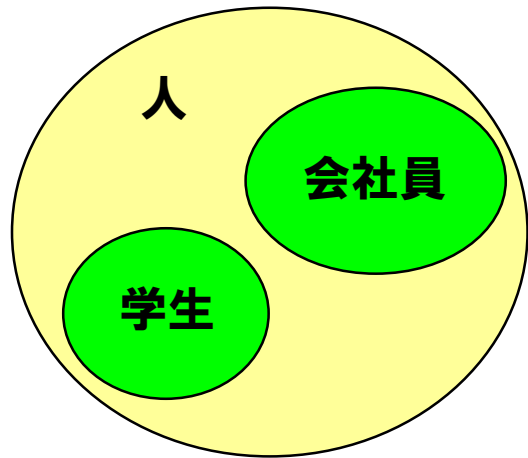
クラス階層（概念階層）



ちょっと抽象的な話ですが、概念の階層構造を考えましょう

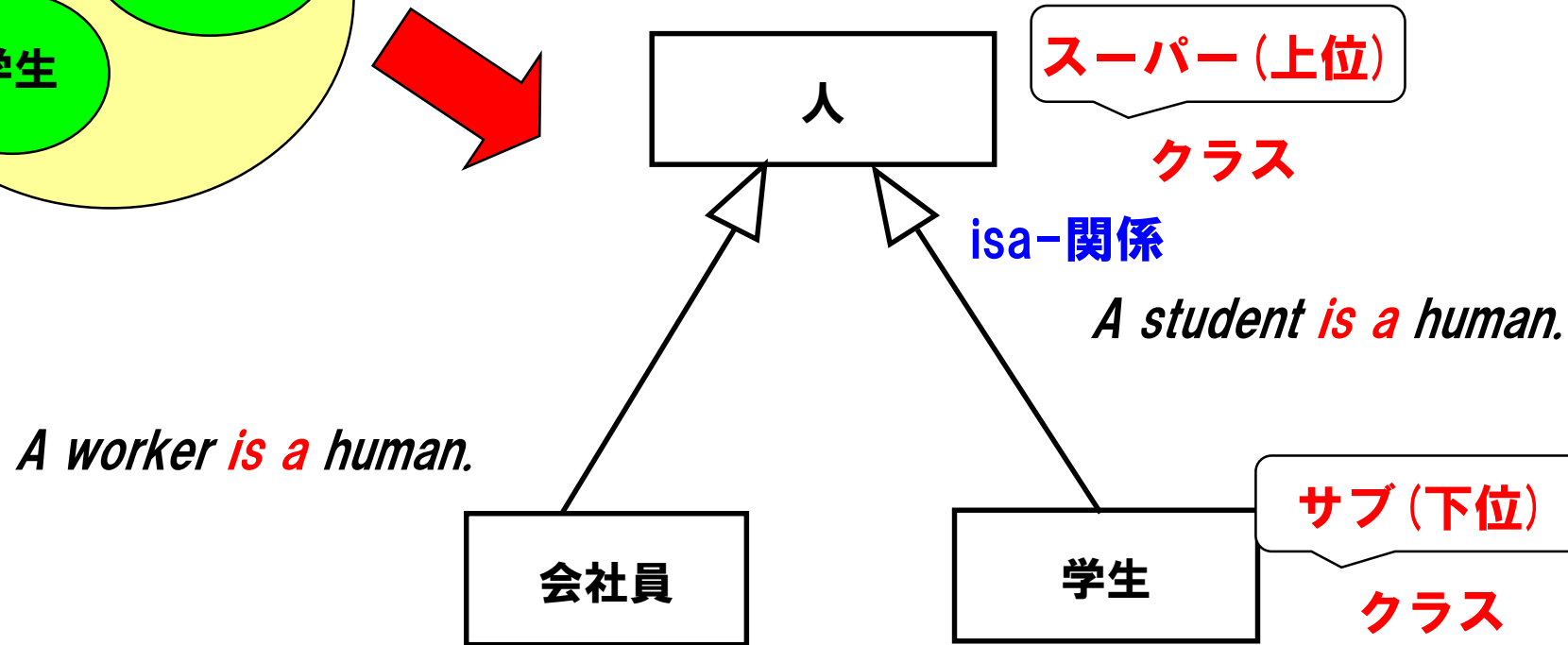


クラス階層（概念階層）



概念の包含関係を
木構造で表現する

※「会社員」でありながら「学生」でもあるという、
ケース（たとえば、社会人博士課程学生などもあります）が
ここでは、そういった共通集合はないものとしましょう



上位概念と下位
概念が
関係づけられま
す。上位クラス
を親クラス、サ
ブクラスを子ク
ラスとして
親子関係と
いうこともあり
ます

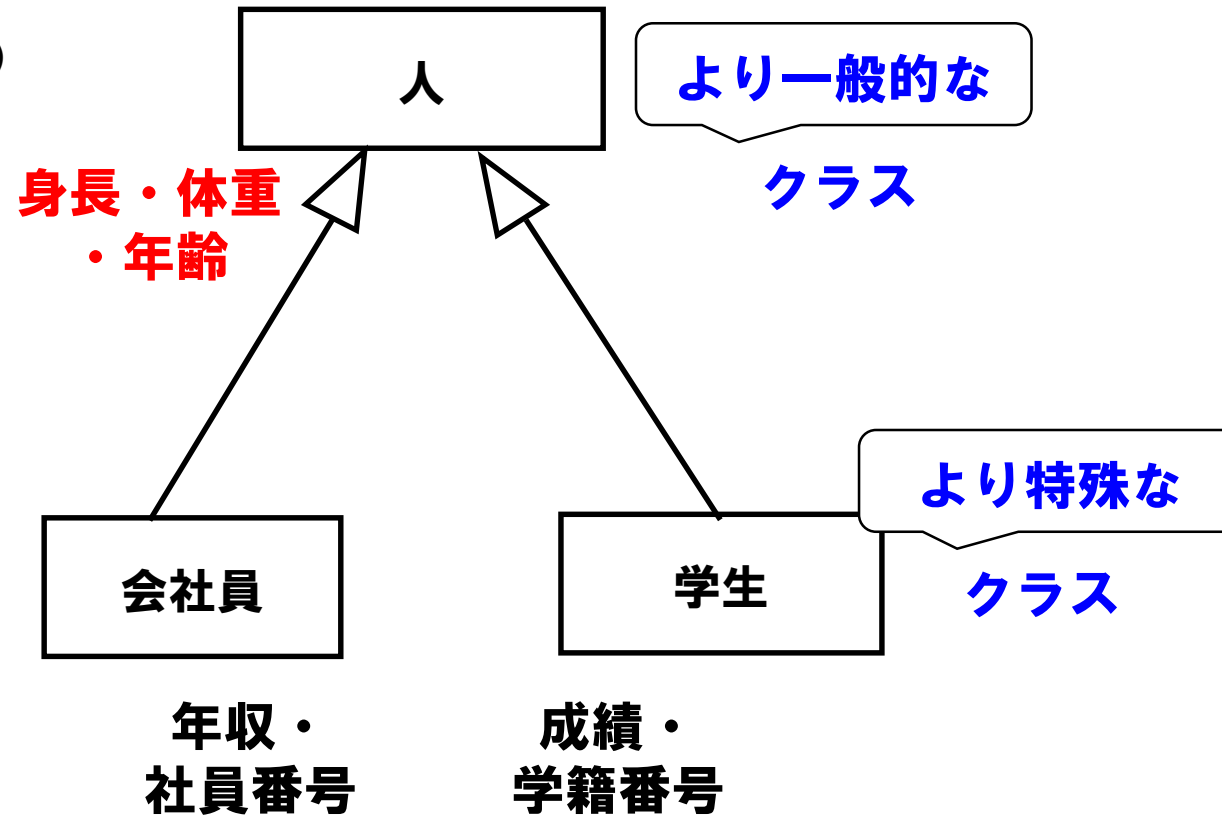
※厳密に言えば、「会社員」や「学生」は、クラスというよりも
ロール（役割）に相当する概念といえるが、わかりやすいので、この例で説明します。



クラス間の共通要素（属性，メソッド）

上下関係にあるクラス同士には共通の要素がある。
通常，より一般的なもの（上位クラス）が要素の追加によって
特殊化される（下位クラスが作られる）

Gen-Spec（汎化 - 特化）
階層ともいう
Generalization
-Specialization



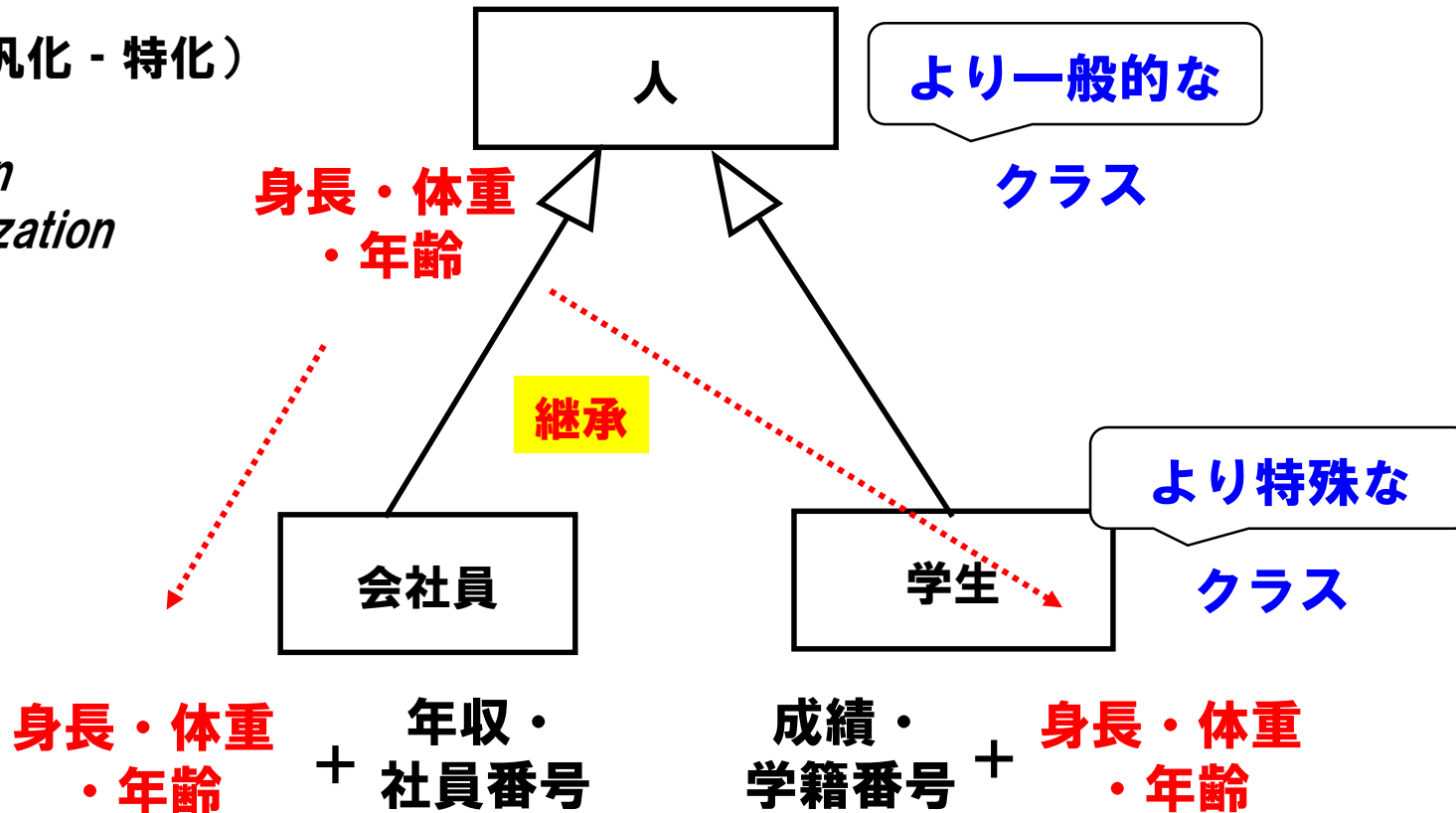
上位概念は
より一般的
で下位概念
はより特殊
です（限定
されていま
す）



クラス間の共通要素（属性，メソッド）

上下関係にあるクラス同士には共通の要素がある。
通常，より一般的なもの（上位クラス）が要素の追加によって
特殊化される（下位クラスが作られる）

Gen-Spec（汎化 - 特化）
階層ともいう
Generalization
-Specialization

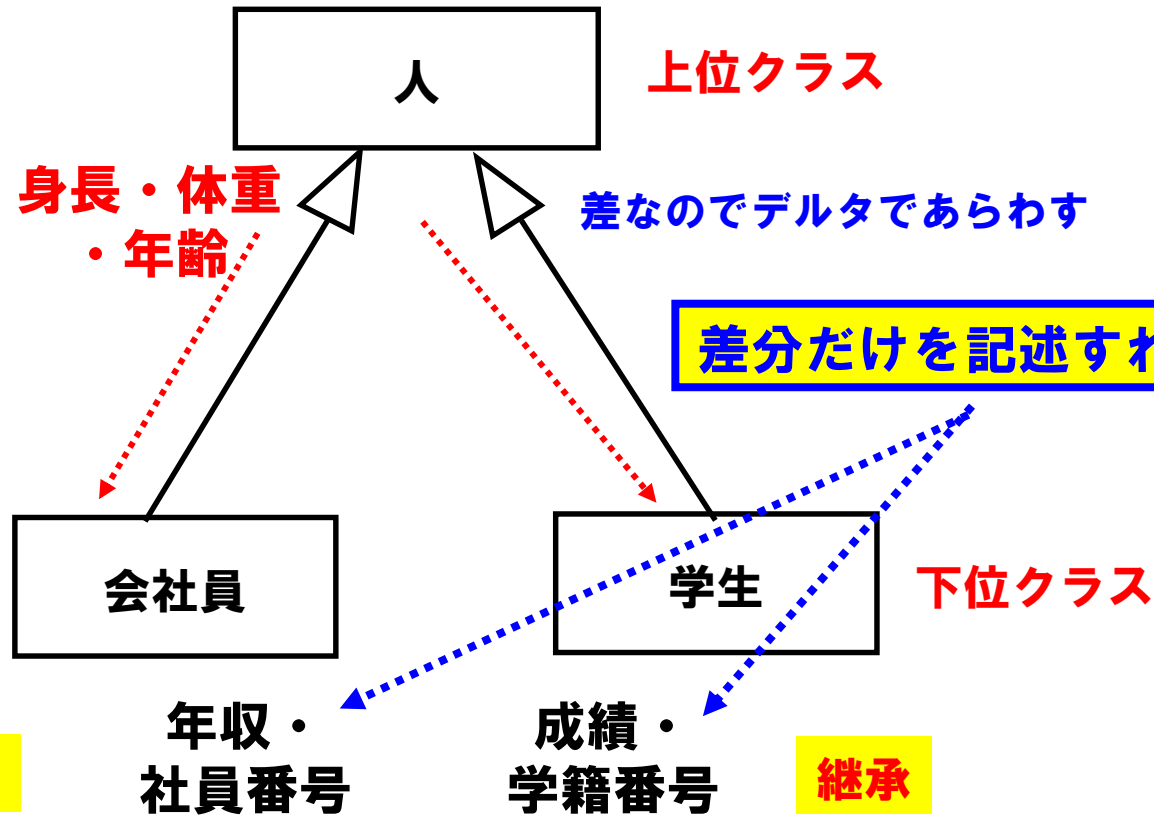


上位概念と下位概念での共通要素は受け継がれます。
親子関係で相続するようなものですね。



性質継承と差分プログラミング

上位クラスから
下位クラスに向かって
属性とメソッドが
継承される



継承されるものは
書かなくてもよい

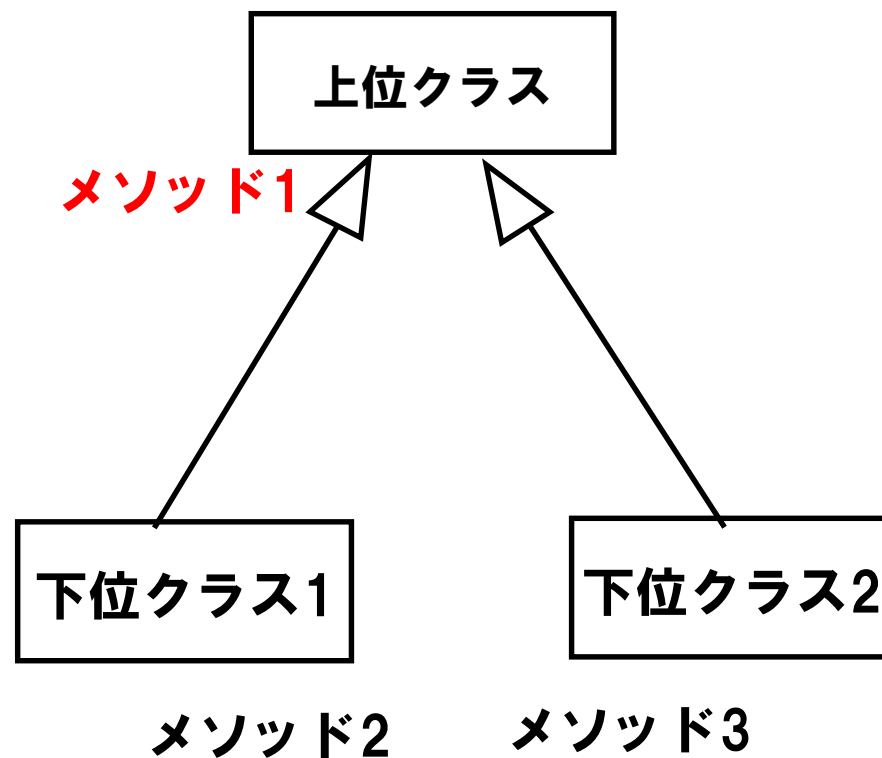
継承される要素は
二重に定義する必要
はありません。
差分だけ定義すれ
ばよさそうです。

差分プログラミングにおける継承の打ち消し（オーバーライド）

SEP05

8

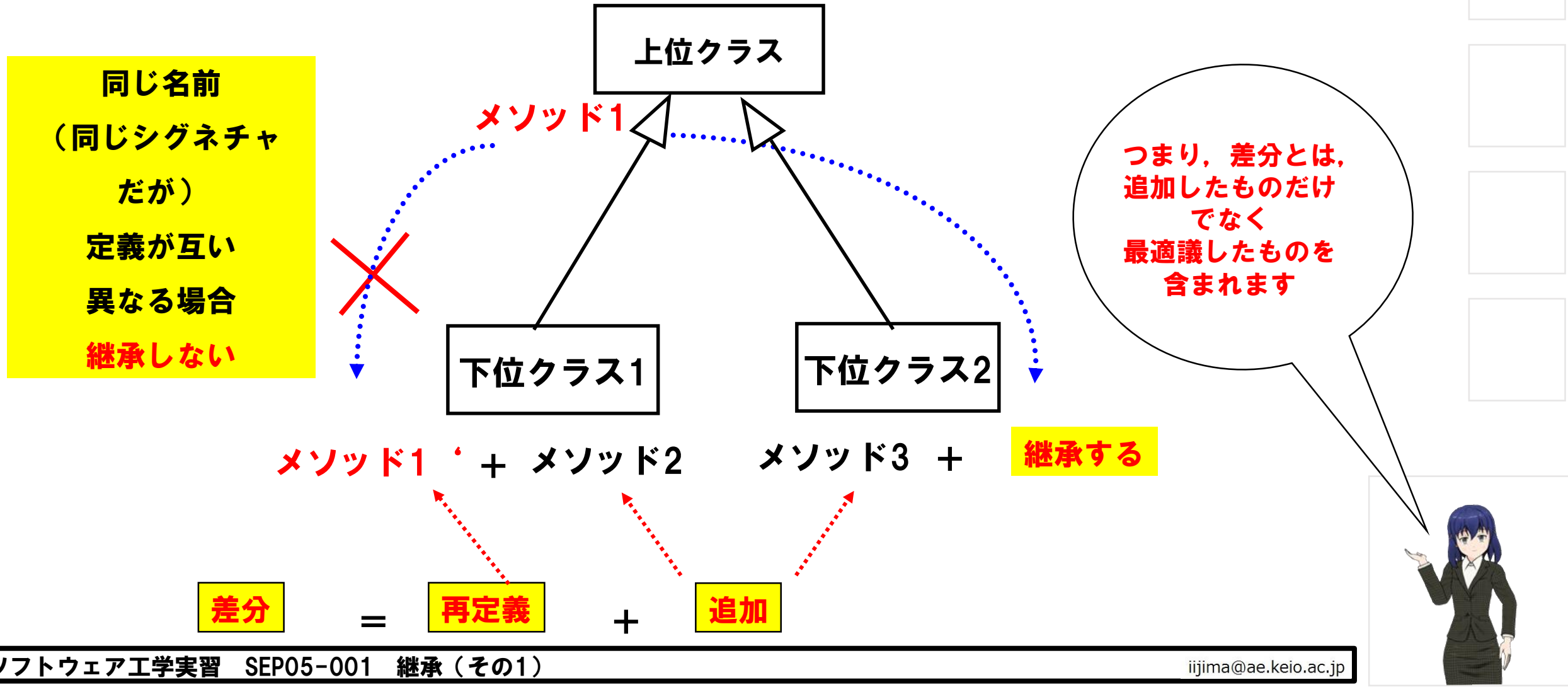
同じ名前
(同じシグネチャ
だが)
定義が互い
異なる場合
継承しない



サブクラスで
メソッドの定義
を上書きするこ
ともできます

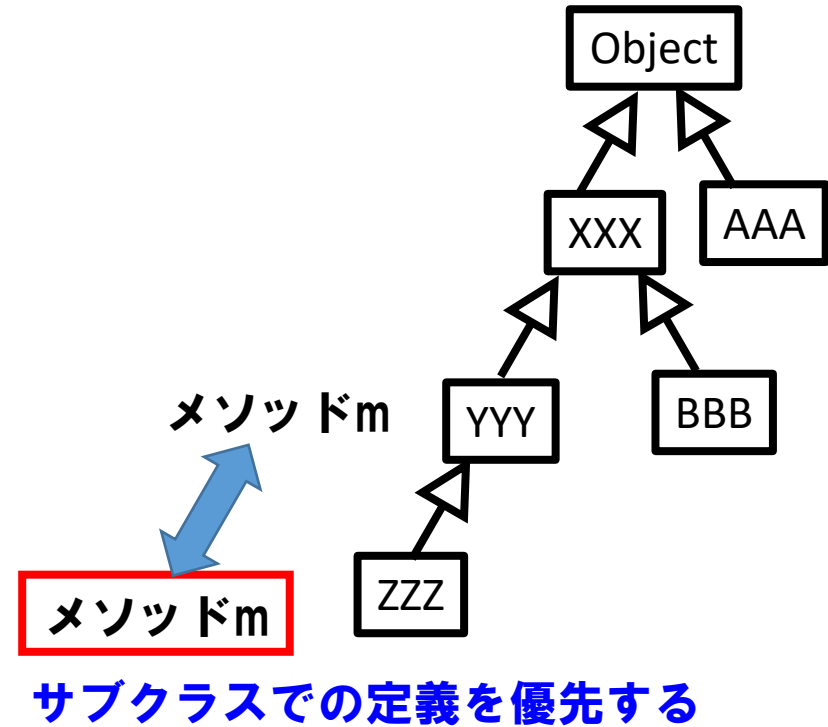


差分プログラミングにおける継承の打ち消し（オーバーライド）



継承の打ち消し（オーバーライド）

- ・ 継承の打ち消し（オーバーライド）
 - ・ 上位クラスで定義されているメソッドと、同じシグネチャ（メソッド名、~~返戻値の型~~、引数の个数・型・順序）を持つメソッドが、下位クラスで定義されていたら、上位クラスのメソッドを継承せず、より下位クラスでの定義の方を優先する。



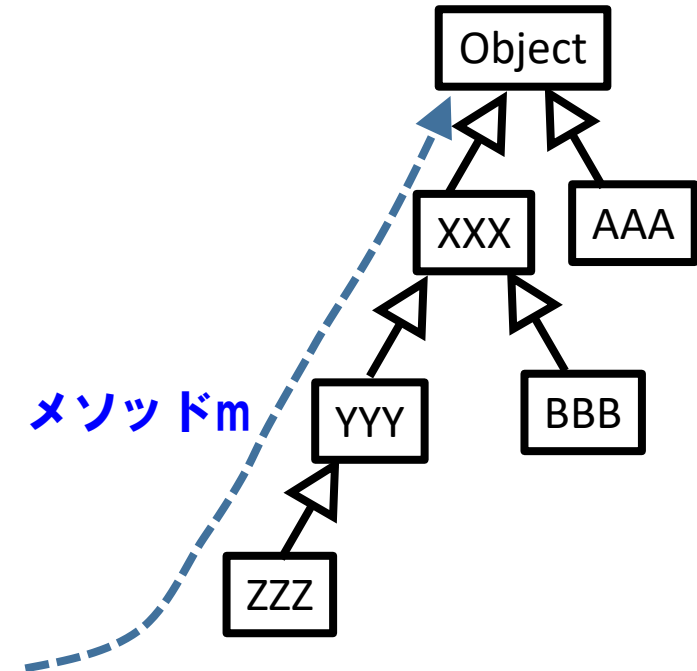
継承階層は何レベルもの
ありえます。より下位の
クラスの定義が優先され
ると捉えることもできま
すね



継承の打ち消し（オーバーライド）

- 継承の打ち消し（オーバーライド）

- 上位クラスで定義されているメソッドと、同じシグネチャ（メソッド名、~~返戻値の型~~、引数の个数・型・順序）を持つメソッドが、下位クラスで定義されていたら、上位クラスのメソッドを継承せず、より下位クラスでの定義の方を優先する。



- もう少し正確には、インスタンス化するときに使ったクラスから見て、継承の木構造中で根（ルート;JavaではObjectクラス）まで遡っていく際に、より手前にあるクラスのメソッドを優先する（単一継承 ⇒ 単一ルートの場合）。

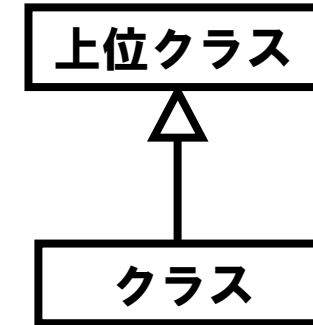
Javaの場合、クラス定義の継承は単一継承、すなわち、親クラスは1つです



【参考】Javaでの継承の指定方法

- 実は、CounterFrameは、JFrameのサブクラス

```
public class クラス名 extends 上位クラス名 {  
    ...  
}
```

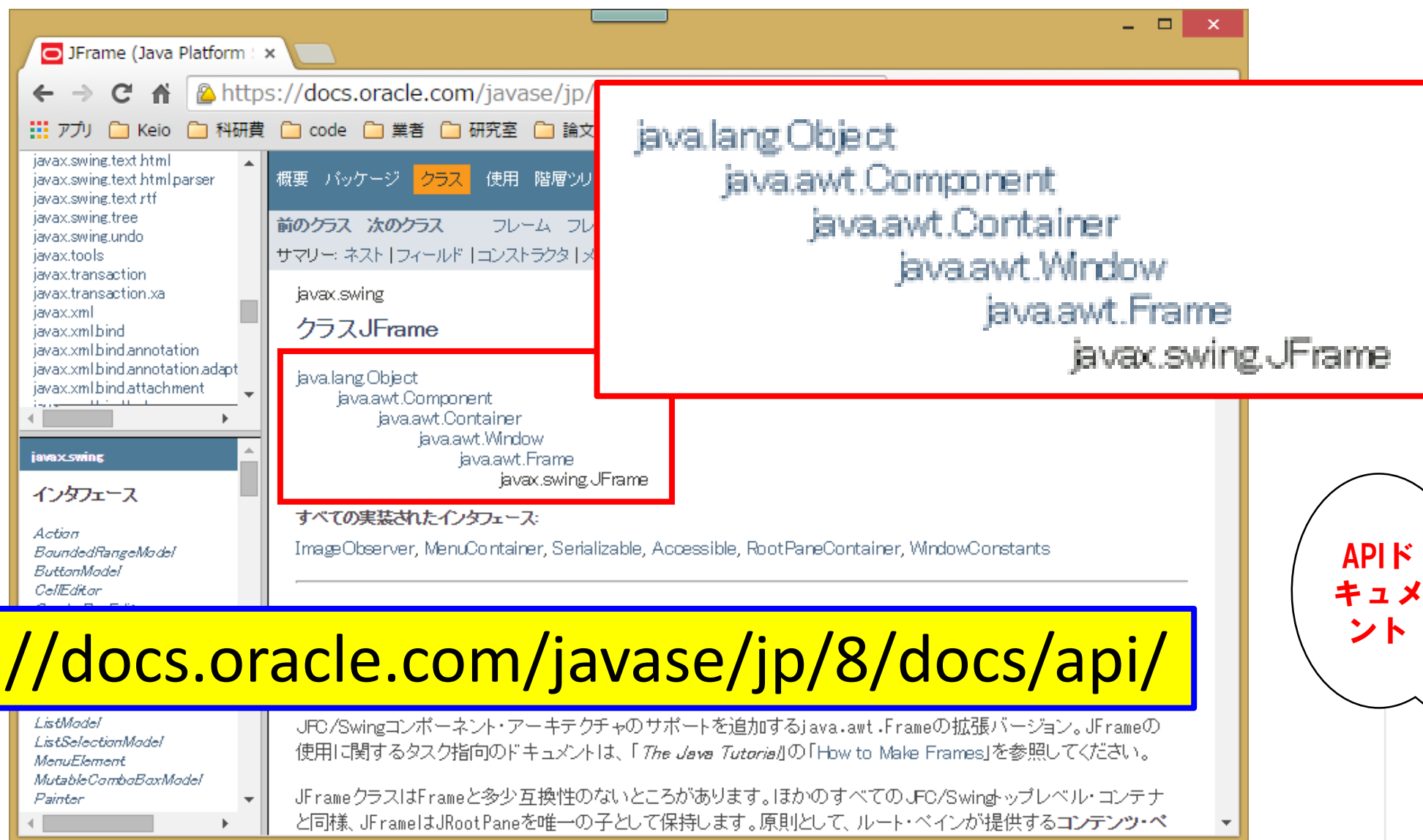


- ◆ 単一継承
 - ◆ 上位クラスは一つだけ指定できる
 - ◆ なぜ、多重継承はよくないのか…
- ◆ オーバーライド（継承の打ち消し）
 - ◆ 上位クラスとシグネチャ（名前、引数/返却値の型と個数）が同じメソッドは、下位クラスで定義されている方を優先する
- ◆ 動的束縛とポリモルフィズム

Javaでは、
extendsという
キーワードを使
います



APIドキュメントで、JFrameを見てみます。



The screenshot shows the Oracle Java API documentation for `JFrame`. A red box highlights the class hierarchy:

- `java.lang.Object`
- `java.awt.Component`
- `java.awt.Container`
- `java.awt.Window`
- `java.awt.Frame`
- `javax.swing.JFrame`

Below the hierarchy, the text reads: "すべての実装されたインタフェース: ImageObserver, MenuContainer, Serializable, Accessible, RootPaneContainer, WindowConstants".

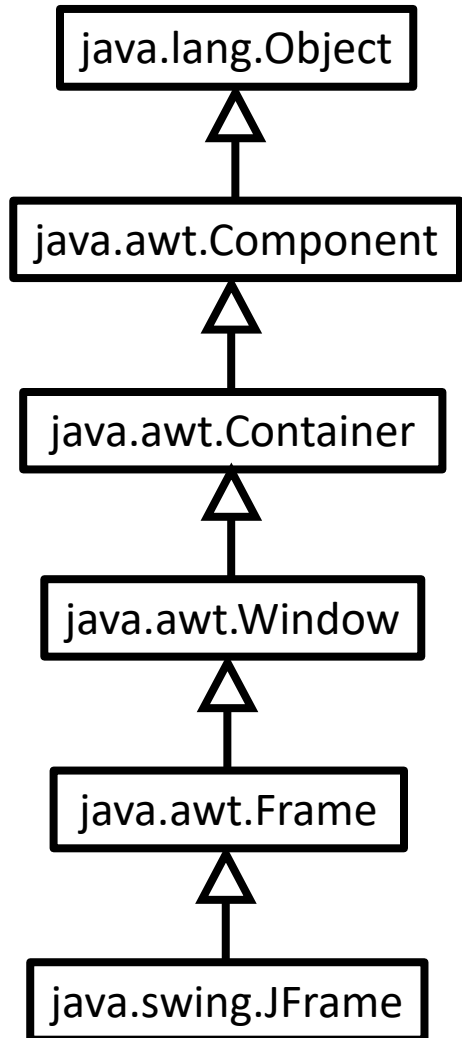
A yellow box at the bottom of the screenshot contains the URL: <http://docs.oracle.com/javase/jp/8/docs/api/>

APIド
キュメ
ント



APIドキュメントで、JFrameを見てみます。

- 最上位クラスをObjectとし、階層構造が構成されています。



```
java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Frame
javax.swing.JFrame
```

GUI部品ですが、
GUI部品を格納
するコンテナ
でもあります



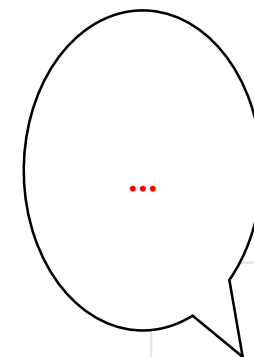
上位クラスから継承されてきたメソッド

SEP05

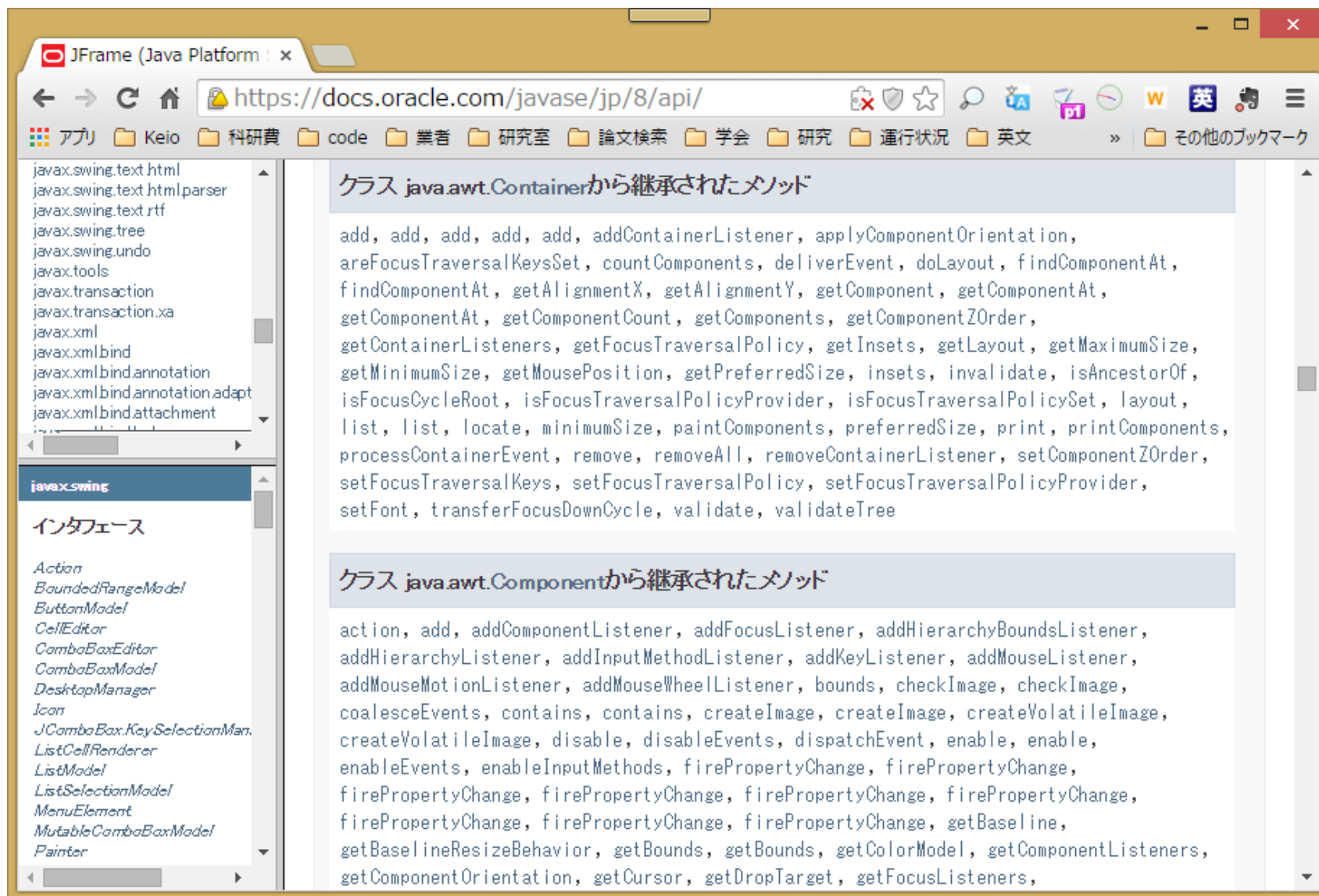
15

The screenshot shows a web browser window with the URL <https://docs.oracle.com/javase/jp/8/api/>. The left sidebar shows a tree of Java packages, with `javax.swing` selected. The main content area displays two sections of inherited methods:

- クラス `java.awt.Frame` から継承されたメソッド**
addNotify, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setBackground, setCursor, setExtendedState, setMaximizedBounds, setMenuBar, setOpacity, setResizable, setShape, setState, setTitle, setUndecorated
- クラス `java.awt.Window` から継承されたメソッド**
addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBackground, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getIconImages, getInputContext, getListeners, getLocale, getModalExclusionType, getMostRecentFocusOwner, getOpacity, getOwnedWindows, getOwner, getOwnerlessWindows, getShape, getToolkit, getType, getWarningString, getWindowFocusListeners, getWindowListeners, getWindows, getWindowStateListeners, hide, isActive, isAlwaysOnTop, isAlwaysOnTopSupported, isAutoRequestFocus, isFocusableWindow, isFocusCycleRoot, isFocused, isLocationByPlatform, isOpaque, isShowing, isValidRoot, pack, paint, postEvent, processEvent, processWindowFocusEvent, processWindowStateEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, reshape, setAlwaysOnTop, setAutoRequestFocus, setBounds, setBounds, setCursor, setFocusableWindowState, setFocusCycleRoot, setIconImages, setLocation, setLocation, setLocationByPlatform, setLocationRelativeTo, setMinimumSize, setModalExclusionType, setSize, setSize, setType, setVisible, show, toBack, toFront



上位クラスから継承されてきたメソッド



上位クラスから継承されてきたメソッド

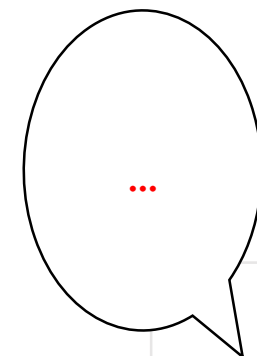
The screenshot shows the Oracle Java API documentation for the `javax.swing.JComponent` class. The left sidebar lists various Java packages and the `javax.swing` package is selected, showing a list of classes including `Action`, `BoundedRangeModel`, `ButtonModel`, `CellEditor`, `ComboBoxEditor`, `ComboBoxModel`, `DesktopManager`, `Icon`, `JComboBox.KeySelectionManager`, `ListCellRenderer`, `ListModel`, `ListSelectionModel`, `MenuItem`, `MutableComboBoxModel`, and `Painter`. The main content area displays the methods inherited from `java.lang.Object` and `javax.swing.MenuContainer`.

クラス `java.lang.Object` から継承されたメソッド

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

インタフェース `javax.swing.MenuContainer` から継承されたメソッド

`getFont, postEvent`



継承の導入

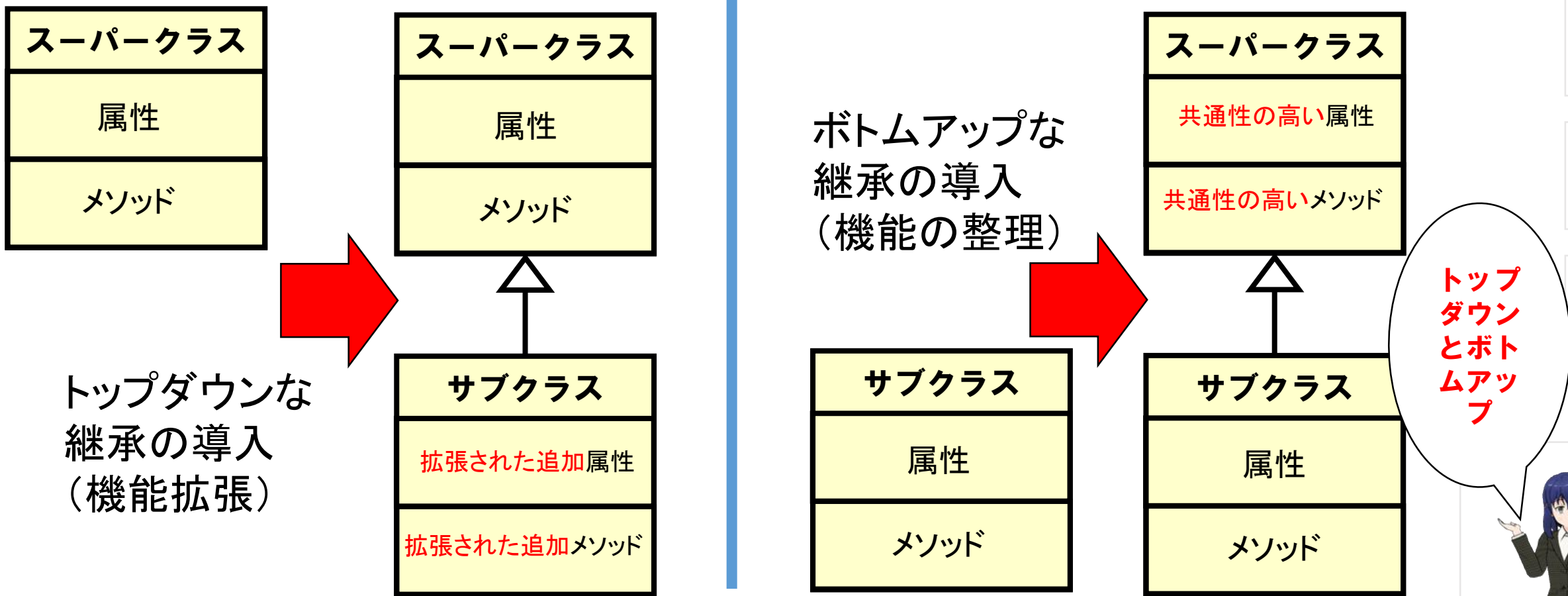
トップダウン，ボトムアップ

では，継承はど
のように導入さ
れるのでしょうか



継承の導入: トップダウンとボトムアップ

- ・ トップダウン：上から下へ / ボトムアップ：下から上へ



ボトムアップな継承関係の導入 (1/4)

SEP05

20

本

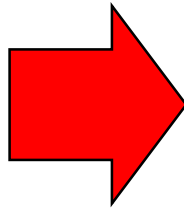
タイトル
著作者
発行日
ページ数

図書館での
蔵書管理用に
「本」クラスが
あります



ボトムアップな継承関係の導入 (2/4)

ある図書館では、当初、書籍しか扱っていなかったが、
各種のAV資料（CDやDVD）も取り扱うようになった
（運用中に発生した仕様の变化）



本
タイトル 著作者 発行日 ページ数

本とAV資料では
取り扱いに若干の
違いがあるが
基本的に同じ

新しいクラスの追加

本	DVD	CD
タイトル 著作者 発行日 ページ数	タイトル 著作者 発行日 収録時間 配給会社	タイトル 著作者 発行日 収録曲目 演奏時間

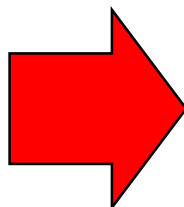
図書館で、
書籍だけでなく
オーディオ/
ビジュアル資
料を扱うよう
になったとしま
す



ボトムアップな継承関係の導入 (2/4)

ある図書館では、当初、書籍しか扱っていなかったが、
各種のAV資料（CDやDVD）も取り扱うようになった
（運用中に発生した仕様の变化）

本
タイトル
著作者
発行日
ページ数



本とAV資料では
取り扱いに若干の
違いがあるが
基本的に同じ

書籍と
オーディ
オ/
ビジュア
ル資料に
も
共通要素
がありま
す



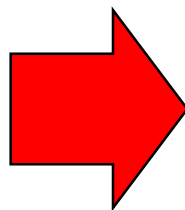
ボトムアップな継承関係の導入 (3/4)

SEP05

23

抽象クラスの
導入

ポリモルフィズム
(後述)の導入へ

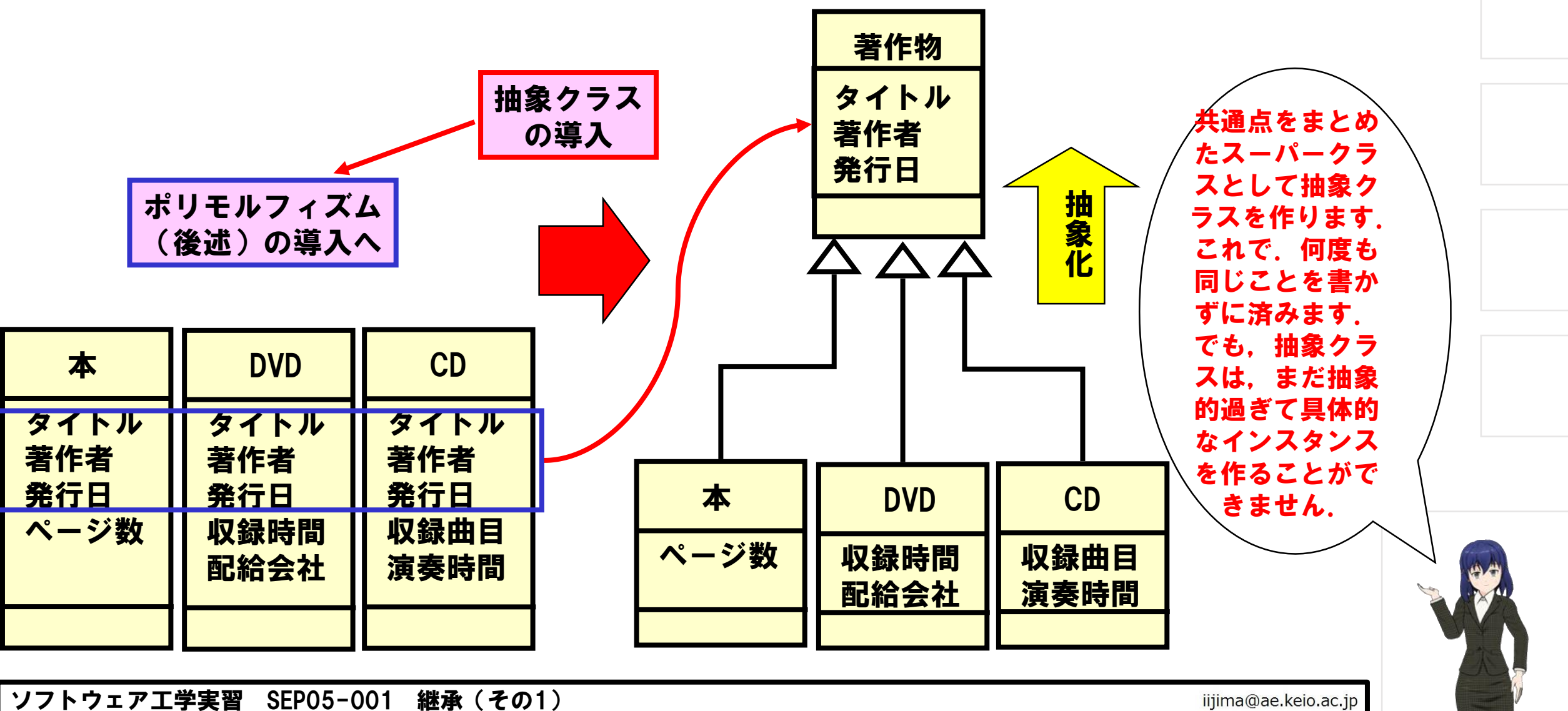


そこで、共通点
をまとめます。
そのために抽象
クラスという概
念を導入します



本	DVD	CD
タイトル	タイトル	タイトル
著作者	著作者	著作者
発行日	発行日	発行日
ページ数	収録時間	収録曲目
	配給会社	演奏時間

ボトムアップな継承関係の導入 (4/4)



抽象クラス

抽象クラスは、
サブクラスの共
通点をまとめて
管理するための
クラスです



Javaでの抽象クラス

・ 抽象クラス

- ・ サブクラスをまとめるため
（概念の整理, ポリモルフィズム (型多様性) の活用)
に, 上位クラスとして導入されるクラス
- ・ このクラスは, 直接インスタンスを作らない (作れない)
- ・ 抽象 (abstract) メソッドを持てる
- ・ 非公開 (private) メンバをもてない
(protectedなら外部から直接アクセスできないが下位クラスに継承される)
- ・ staticメソッドをもてない

構文: クラス修飾子abstractを指定する.

```
abstract class クラス名 {
```

```
    ...
```

```
}
```

抽象クラスのインスタンスを作ろうとするとコンパイラが警告してくれます



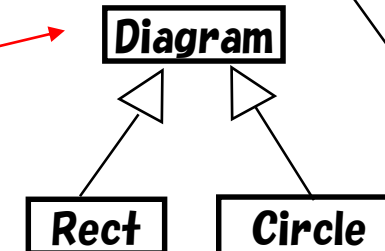
Javaでの抽象クラス

- 抽象メソッド...下位クラスで必ずオーバーライドしなければならない
コンパイラによるチェック

```
public class Diagram {  
    public void move ( int x, int y ) {  
        System.out.println ("Not implemented Yet!");  
    }  
    public void draw (Graphics g) {  
        System.out.println ("Not implemented Yet!");  
    }  
}
```

抽象クラスには、
サブクラスでないと具体的に定義できないメソッドがあります。

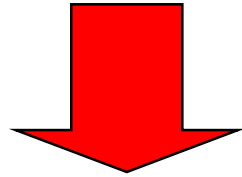
抽象クラス



Javaでの抽象クラス

- 抽象メソッド...下位クラスで必ずオーバーライドしなければならない
コンパイラによるチェック

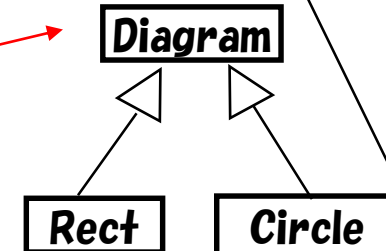
```
public class Diagram {  
    public void move ( int x, int y ) {  
        System.out.println ("Not implemented Yet!");  
    }  
    public void draw (Graphics g) {  
        System.out.println ("Not implemented Yet!");  
    }  
}
```



```
public abstract class Diagram {  
    public abstract void move ( int x, int y ) ;  
    public abstract void draw (Graphics g) ;  
}
```

抽象クラス

抽象メソッドとして
定義することができ
ます。
抽象クラスにしてお
けば、定義し忘れて
も、コンパイラが
警告してくれます



抽象クラスの役割

- すこしずつ、差分定義を付け加えていくことができる
- サブクラスで、確実に定義しなければならないことを明記できる
 - 抽象クラスが残っているとインスタンス生成ができない
 - コンパイラが抽象クラスが残っていることを指摘し、最終的には抽象クラスが残っていない（すべて具体化されたことを）保証してくれる
- ある抽象クラスのサブクラスであれば、必ず持たねばならないメソッドを明示化できる
 - ポリモルフィズム（型多様性）と連動して機能する（以降の回で）

抽象メソッドを定義しておくと、そのクラスのサブクラスが受け付けられるメソッドをチェックできます



インタフェース

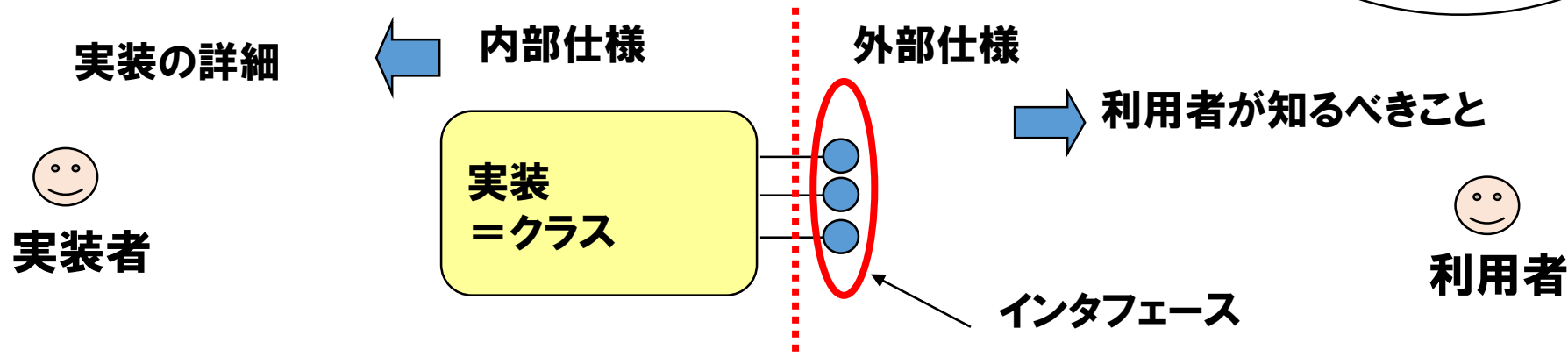
Javaには、抽象
クラスの特
殊なもの
として
インタ
フェース
という
概念が
あり
ます



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
- ・ これによって…
 - ・ 後から、実装をチューンナップできる
 - ・ 情報隠蔽 (information hiding)

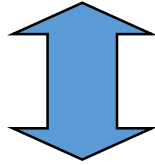
元々、ソフトウェア工学の概念としては、クラスとインタフェースは別のもので、インタフェースは外部仕様、クラス（実装）は内部仕様に相当し、この二つを分離することに意義がありました



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

インタフェース

外部から呼び出せる
メソッドの呼び出し方の情報だけ



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

クラス

属性情報,
メソッドの呼び出し方だけでなく,
その定義本体,
内部的にしか使わない
メソッド情報を含む

インタフェース
で,
外部に公開する
メソッドを明記
します

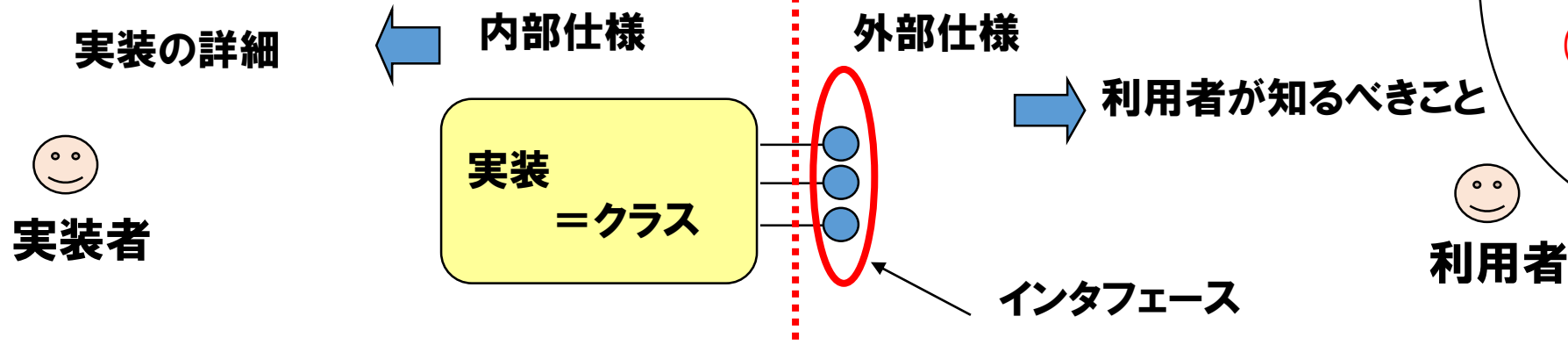


クラス定義におけるインタフェース指定

- ・ 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

```
class クラスXXX implements インタフェースYYY {  
    ...  
}
```

クラスXXXは
インタフェースYYYを実装
する
(implements)
と読みます



Javaでのインタフェースの定義

```
interface YYY extends superYYY,superYYY2 {  
    メソッドシグネチャ  
    ...  
}
```

◆ インタフェースは多重継承できる

なぜ、インタフェースの多重継承は、
クラス多重継承と違って問題がないのか？
⇒(実装が競合しないから)

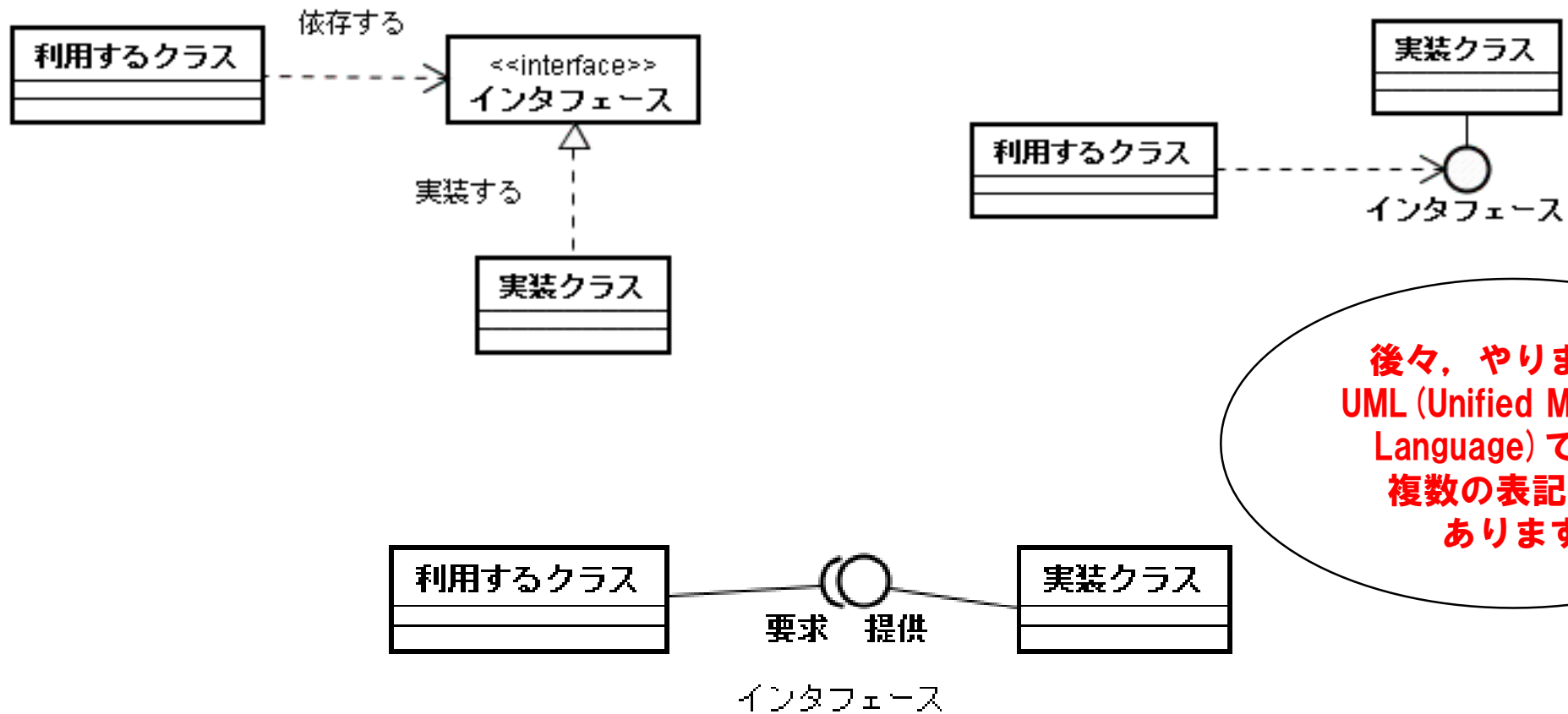
インタフェースにも
継承が使えます。
クラスと異なり、
多重継承ができます。



【参考】インタフェースと実装のUML表記

SEP05

35



後々、やります、
UML (Unified Modeling
Language) では、
複数の表記法が
あります



Javaでのインタフェースの利用

- ・ クラスと同じように変数の、**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
- ・ Javaでは**特殊な抽象クラス**として規定されている
 - ・ すべてのメソッドが**抽象**メソッドに相当する。
 - ・ メソッド定義本体を伴うメソッド宣言が含まれない
 - ・ すべてのメソッドが**public**
 - ・ 定数化 (finalize) されていない属性が存在しない。
 - ・ 値を変更できる属性を持たない
- ・ しかし、（**抽象クラスを含む**）クラスとインタフェースの**本来の役割は全く異なる**
 - ・ 実装（内部仕様）とインタフェース（外部仕様）の相違
 - ・ **あるクラスが、特定のメソッド群を実装していることを保証する**
- ・ **多重インタフェース**
 - ・ あるクラスが複数のインタフェースを実装できる
 - ・ オブジェクトの多面性を表現することもできる

インタフェース
は変数の型とし
て使えます。
オブジェクトが
受け付けられる
メッセージの型
チェックに使え
ます



- **インタフェースの方が都合のいい場合**

- クラスが実装しているメソッド群を保証する
- クラスは単一継承のみ，インタフェースは**多重継承**を許す
 - インタフェースを複数指定しても，実装を継承しない（競合する定義本体がない）ので，問題がない

- **多重インタフェース**

- 一つのクラス定義において，インタフェースは複数指定できる
- 一つのオブジェクトへの複数の観点を与える

- **抽象クラスの方が都合が良い場合**

- 抽象クラスには，抽象メソッドだけでなく，具体的な定義本体をもつメソッド定義も記述できる
 - インタフェースには実装を書くことはできない
 - 抽象クラスであれば，サブクラスで，差分定義を付け加えていくことができる
 - これによって，**定義の共通部分をまとめていく**ことができる

インタフェースは多重継承が可能です。
抽象クラスには、実装メソッドも定義することもできます



これまでの授業の範囲で、
インタフェースが使われていたところ

今後は、
Observer/Observerableパターンや
コレクションフレームワークでも使います

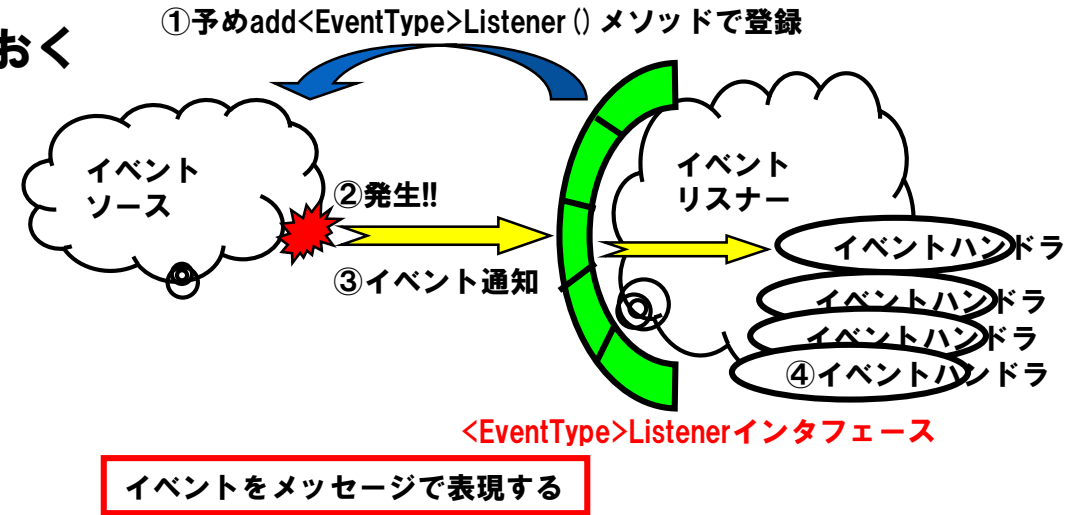
これまでの授業
を振り返って、
インタフェース
の概念を再確認
しましょう



イベント駆動プログラムにおけるリスナ

・ 手順

- ① 予めコンポーネントにイベントリスナを登録しておく
- ② そのコンポーネント（イベントソース）で、イベントが発生する
（例えばボタン上でマウスボタンが押される）
- ③ イベントソースに登録されている全イベントリスナに、そのイベントが通知される
- ④ リスナで、対応するメソッド（イベントハンドラ）が起動される



イベントリスナーはイベントハンドラを実装していることを保証しなければならない



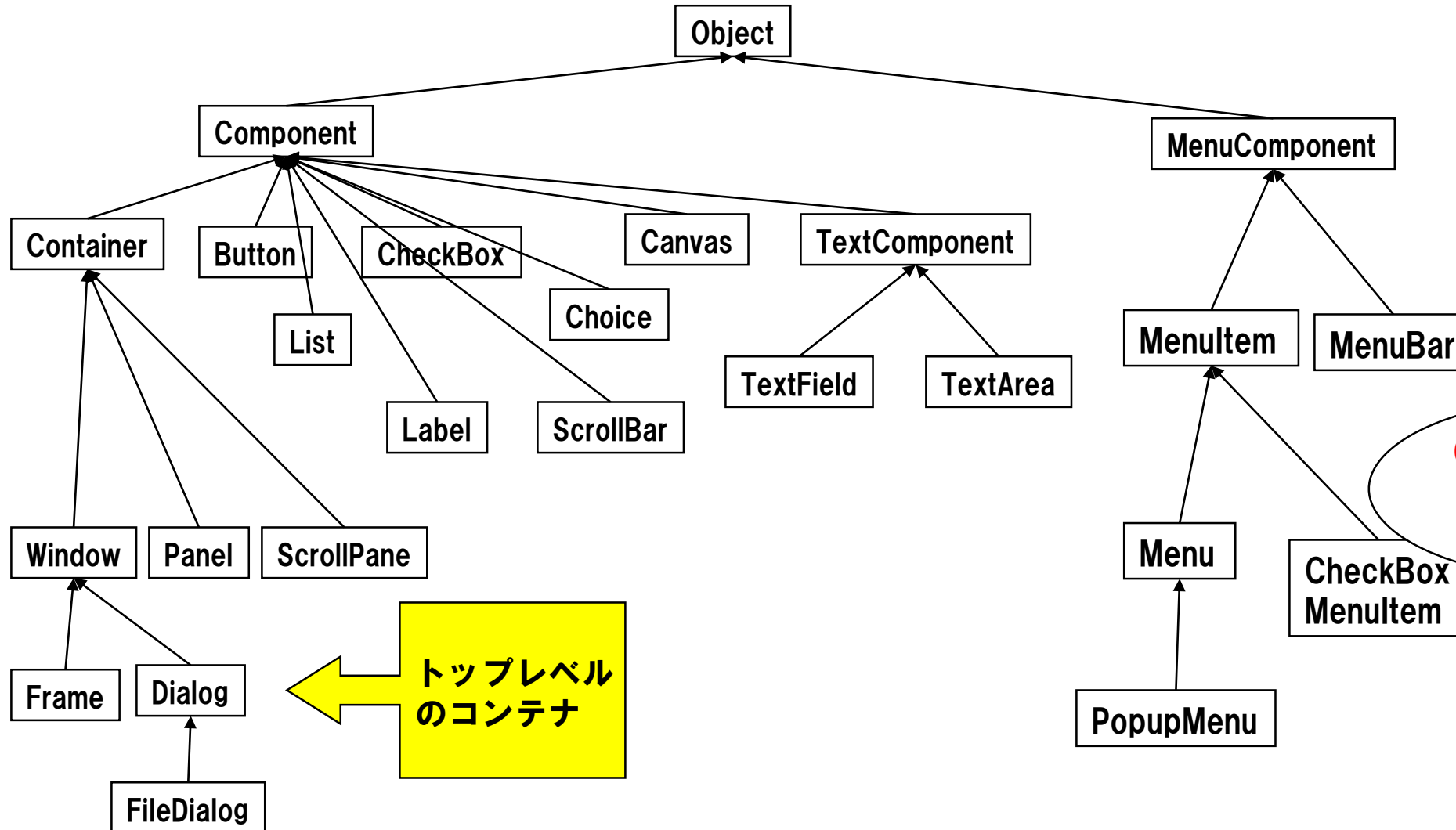
イベントリスナインタフェースの実装宣言が必要

全体の流れです。



GUIコンポーネントの階層

- ・コンポーネント毎に、発生するイベントとハンドラが決まっている



GUIコンポーネント
は
いろいろあります



GUIコンポーネント毎のイベントとハンドラ (1/3)

SEP05

41

- ・コンポーネント毎に、発生するイベントとハンドラが決まっている

Component	ComponentEvent	componentMoved ()
		componentResizes ()
		componentShown ()
		componentHidden ()
	FocusEvent	focusGained ()
		focusLost ()
	KeyEvent	keyPressed ()
		keyReleased ()
		keyTyped ()
	MouseEvent	mouseClicked ()
		mouseEntered ()
		mouseExited ()
		mousePressed ()
		mouseReleased ()
		mouseDragged ()
		mouseMoved ()

コンポーネント毎に、
発生するイベントと
ハンドラが
決まっています。



GUIコンポーネント毎のイベントとハンドラ (2/3)

SEP05

42

- ・コンポーネント毎に，発生するイベントとハンドラが決まっている

Button	ActionEvent	actionPerformed ()
MenuItem		actionPerformed ()
List		actionPerformed ()
Choice	ItemEvent	itemStateChanged ()
CheckBox		itemStateChanged ()
CheckBox MenuItem		itemStateChanged ()
Text Component	TextEvent	textValueChanged ()
TextField	ActionEvent	actionPerformed ()

よくつかう，
GUI部品
です



GUIコンポーネント毎のイベントとハンドラ (3/3)

SEP05

43

- ・コンポーネント毎に，発生するイベントとハンドラが決まっている

Window	WindowEvent	windowClosed ()
		windowClosing ()
		windowOpened ()
		windowIconified ()
		windowDeiconified ()
ScrollBar	Adjustment Event	adjustmentValueChanged ()

Windowや
スクロールバー
のイベントと
ハンドラです



ボタンのアクション:P0104パッケージ (LikeButtonクラス)

SEP05

44

```
package p0104;

// =====

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

// =====

class LikeButton extends JButton implements ActionListener {
    private int count = 0; // カウント数を格納する
    private JLabel aLabel; // カウントを表示するラベル
    public LikeButton ( JLabel aLabel ) { // コンストラクタ
        ... }
    public void actionPerformed ( ActionEvent e ) { // イベント・ハンドラ
        ... }
}
```

前回の例題を
見てみましょう

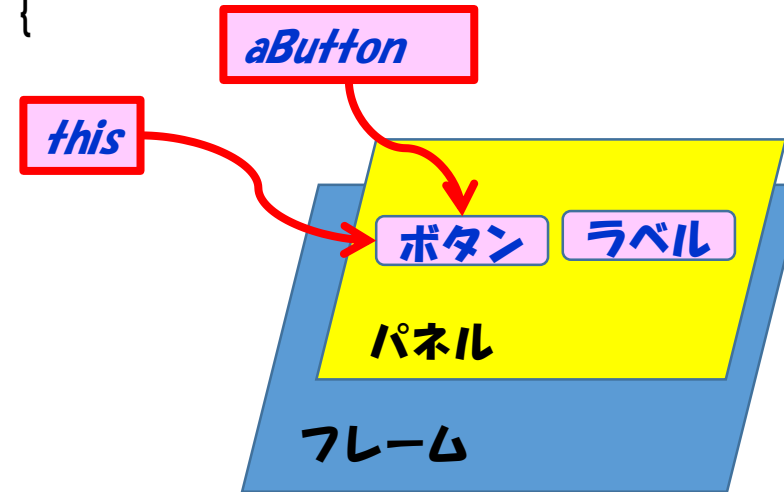


ボタンのアクション:P0104パッケージ (LikeButtonクラス)

SEP05

45

```
class LikeButton extends JButton implements ActionListener {  
    private int count = 0; // カウント数を格納する  
    private JLabel aLabel // カウントを表示するラベル  
  
    ...  
  
    /**  
     * コンストラクタ (インスタンス生成時の初期設定)  
     */  
    public LikeButton ( JLabel aLabel ) {  
        super ( "いいね" );  
        addActionListener (this);  
        this.aLabel = aLabel;  
    }  
}
```



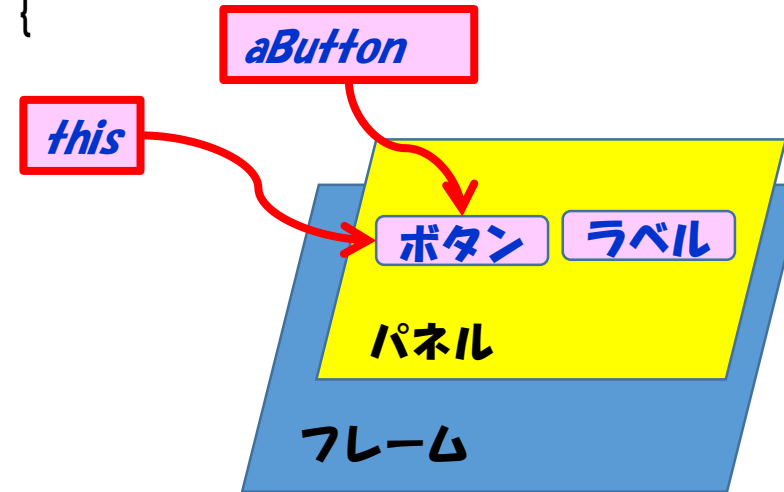
this.が付いていると、
属性（フィールド変数）
であることが
わかります。
thisがついていないと、
変数の有効範囲（Scope）
で近いものになります

ボタンのアクション:P0104パッケージ (LikeButtonクラス)

SEP05

46

```
class LikeButton extends JButton implements ActionListener {  
    private int count = 0; // カウント数を格納する  
    private JLabel aLabel; // カウントを表示するラベル  
  
    ...  
  
    /**  
     * コンストラクタ (インスタンス生成時の初期設定)  
     */  
    public LikeButton ( JLabel aLabel ) {  
        super ( "いいね" );  
        addActionListener ( this );  
        this.aLabel = aLabel;  
    }  
}
```



自分自身を
自分のイベントリスナー
として登録しています。

自分自身 (this) が
イベントリスナーであり
イベントハンドラを
持っていることを意味する



ボタンのアクション:P0104パッケージ (LikeButtonクラス)

SEP05

47

```
class LikeButton extends JButton implements ActionListener {  
    private int count ← 0; // カウント数を格納する  
    private JLabel aLabel ← // カウントを表示するラベル  
    ...  
  
    // イベント・ハンドラ (ActionEvent)  
    public void actionPerformed ( ActionEvent e ) {  
        // カウントアップし、その値をラベルに表示する  
        count++;  
        aLabel.setText ( Integer.toString ( count ) );  
    }  
}
```

ボタンが押された時のアクション
= カウントアップして、結果を文字列に変換して表示

ボタンクリック
では、
Actionイベントが
発生し、
ActionPerformed
というハンドラが
起動される

