(8) インタフェース

(8．1)
インタフェースの意味

どのような場合にインタフェースを使うのかを考えます。

(8．2)
Javaでのインタフェース

インタフェースのJavaでの記法を見てみます。

(8．3)
UMLでの記法

UMLでの記法を見てみます。

飛行機オブジェクト

(8．4)
ブラックジャックでのインタフェース

ブラックジャックでのインタフェースの使われ方を確認します。

(8．1．1) 呼ぶ／呼ばれる

■再利用するプログラムをどのように利用するかについて、次のように2つに分けて考えることが出来ます。

(1) 自分が作るプログラムから、他人が作ったプログラムを呼ぶ

(2)他人が作ったプログラムから、自分が作るプログラムを呼ぶ

■上記 (1) の場合は、考えやすいと思います。

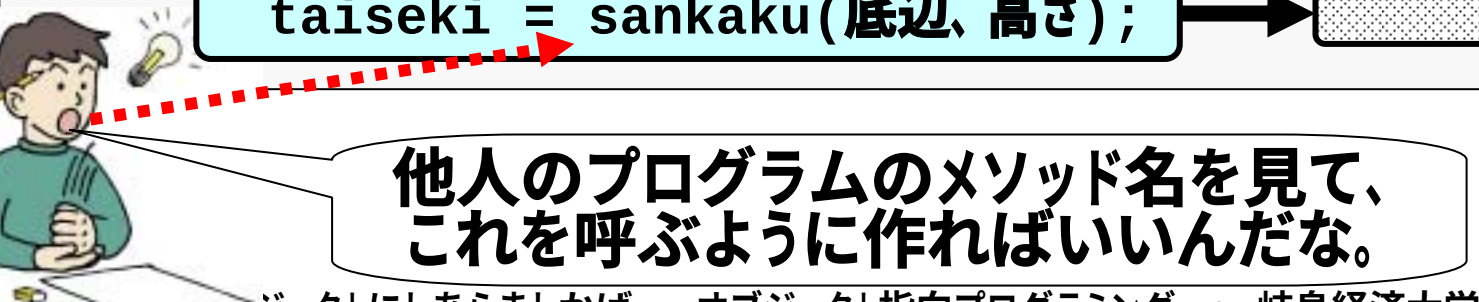
昔のプログラム

```
# 他人のプログラム
int sankaku(int 底辺,int 高さ){
  return(底辺*高さ/2);
}
```

再利用

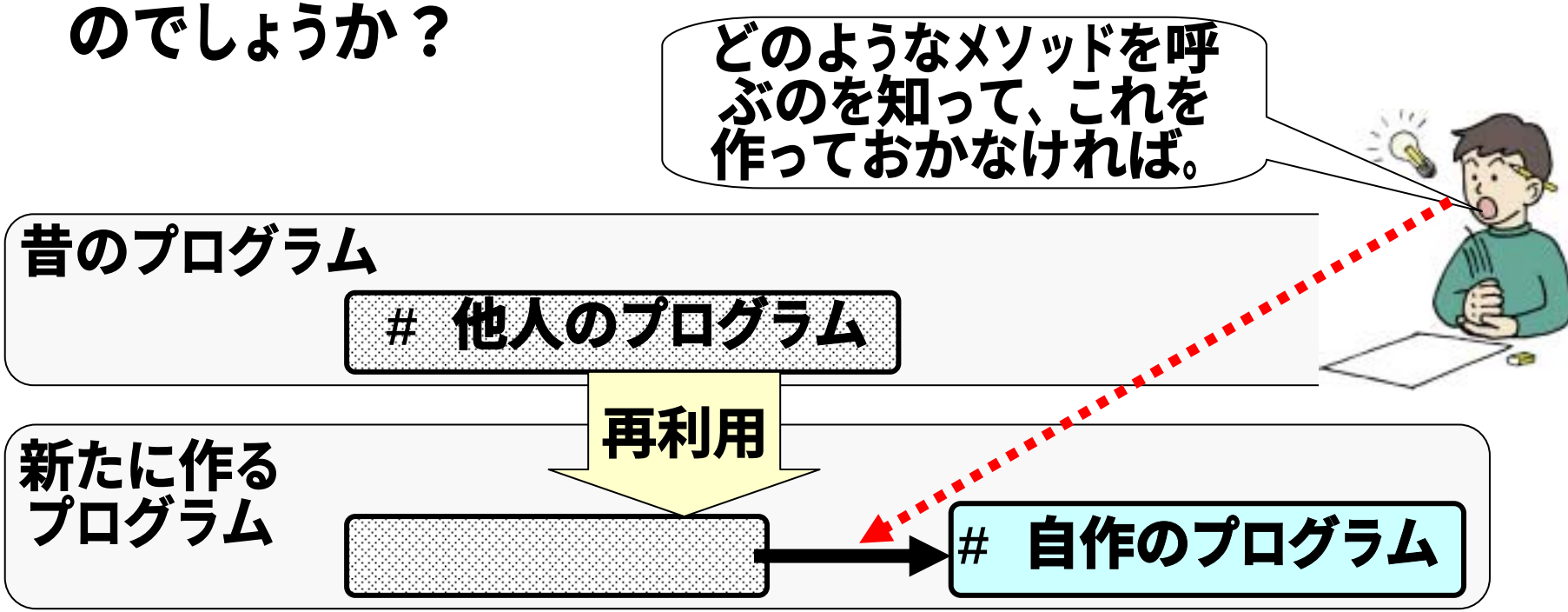
新たに作るプログラム

```
# 自作のプログラム
  taiseki = sankaku(底辺、高さ);
```



(8．1．2) 自作部分が呼ばれる場合

■前スライド中「(2) 他人が作ったプログラムから、自分が作るプログラムを呼ぶ」の場合は、どうすれば良いのでしょうか？



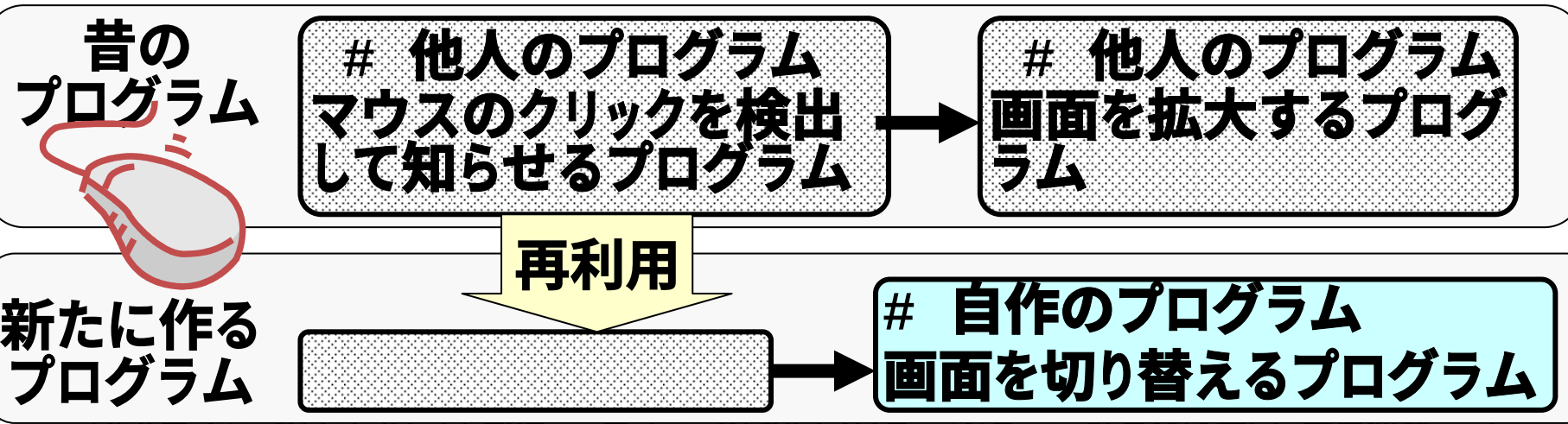
■自作するプログラムでは、「どのようなプログラムが呼ばれるか」を解った上で、これを作っておかなくてはなりません。

(8．1．3) 自作部分が呼ばれる例

■自作部分が呼ばれるような場合はたくさんあります。

■マウスがクリックされたら画面を切り替えるソフトウェアを作りたいとします。

- マウスがクリックされたら画面を拡大するプログラムが既にあるのであれば、「マウスがクリックを検出して知らせるプログラム」の部分は、再利用できるはずです。
- 新たに作るプログラムの中では、「画面を切り替えるプログラム」を作れば良いのですが、これは再利用した部分から呼ばれることになります。

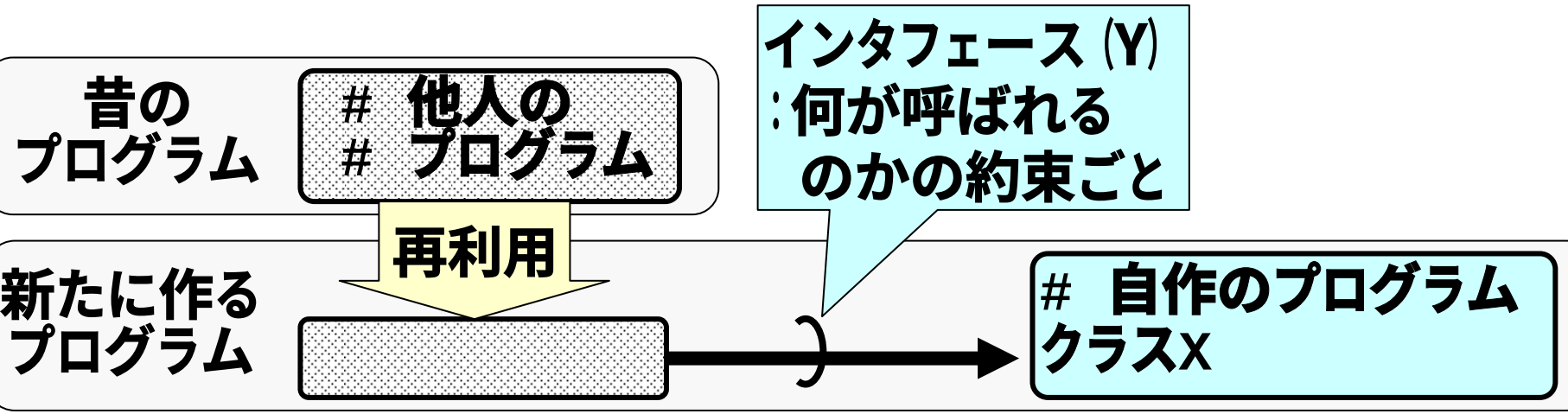


(8．1．4) インターフェース

■「どのようなプログラムが呼ばれるか」の約束事を、“インタフェース”と言います。

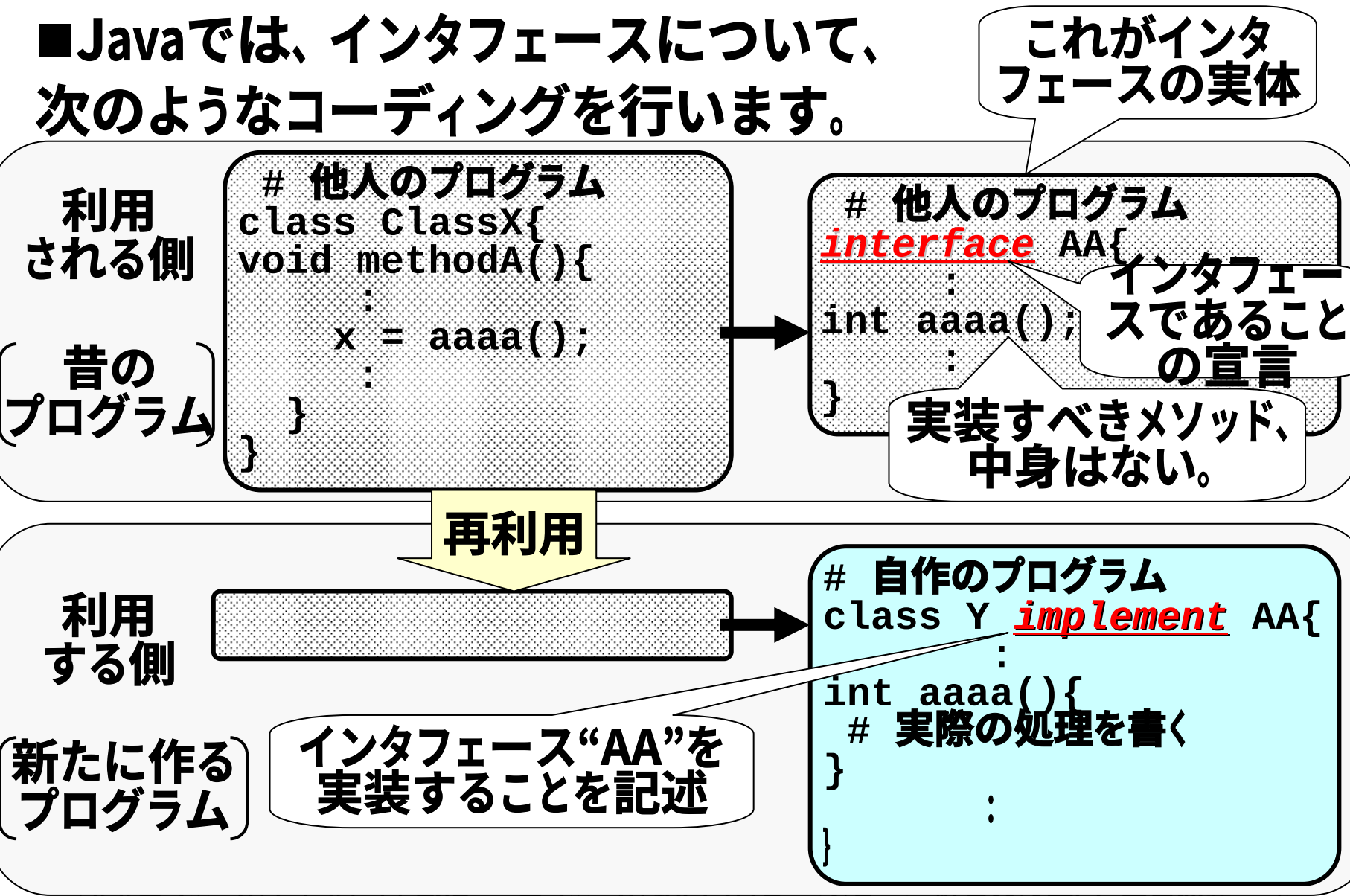
■ある“クラスX”が、“インタフェースY”にそったメソッドをちゃんと備えている場合、次のように言います。

「“クラスX”は、“インタフェースY”を実装(implement)している。」



(8．2．1) Javaでのインタフェース

■Javaでは、インタフェースについて、
次のようなコーディングを行います。

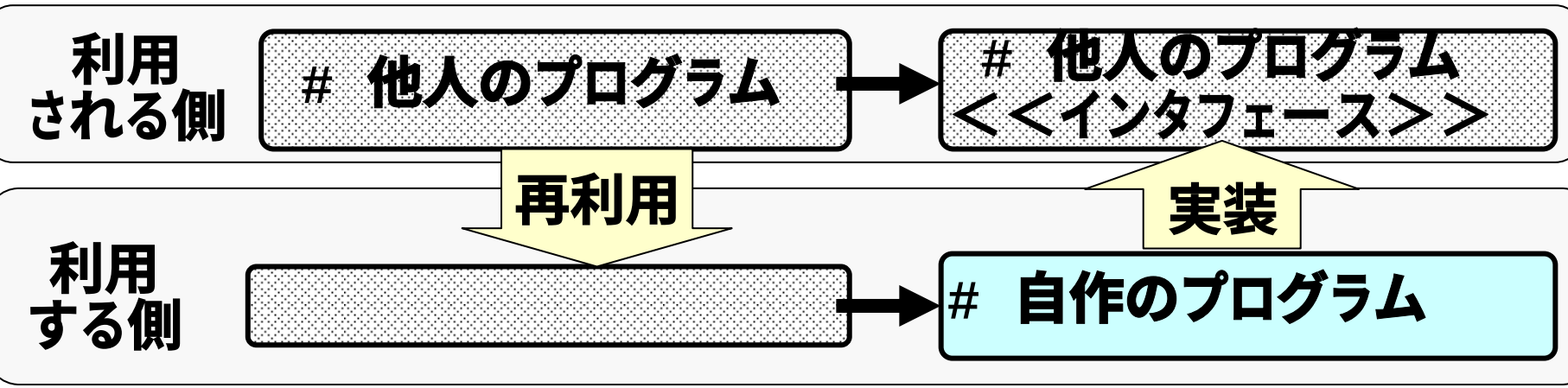


(8．2．2) インタフェースを守らせる

■極論すれば、インタフェースは紙に書いたものでもOKです。しかし、紙に書いただけでは解りにくく、実際にインタフェースを守れているのかを自動的に確認する手段もありません。

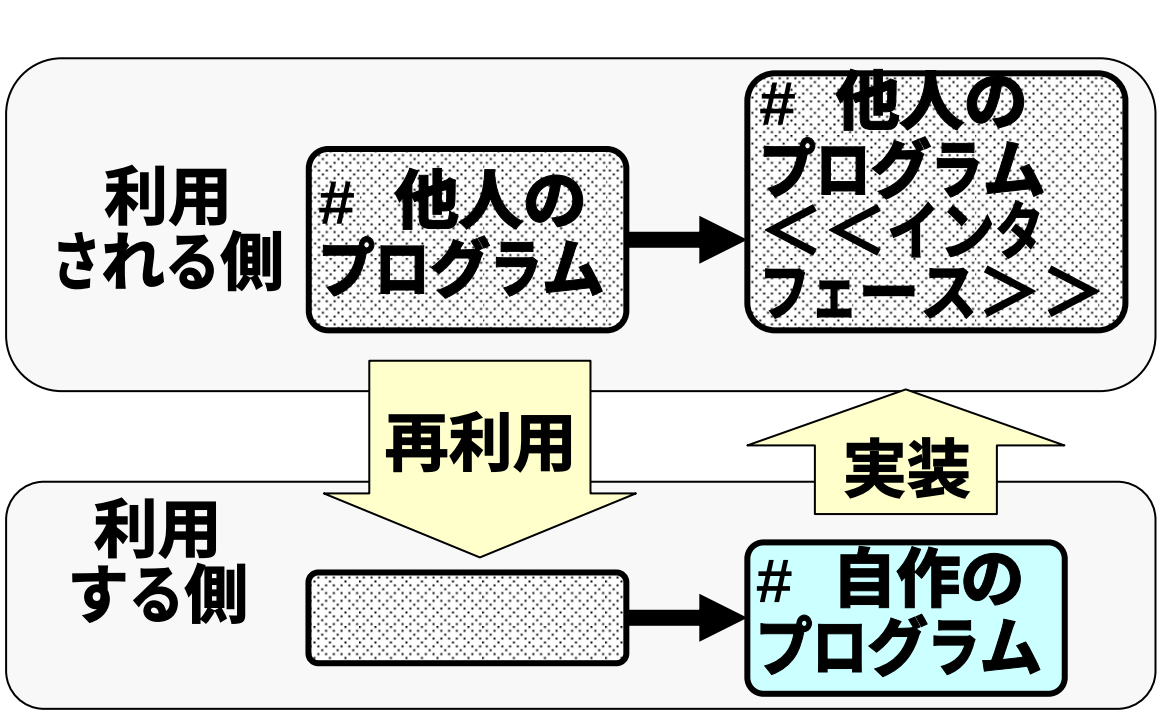
■Javaでは、必ず意図した方法でプログラムが呼ばれるように、利用される側でインタフェースの定義をプログラムとして供給するわけです。

■自作のプログラムの中でインタフェースを実装(implement)すると、ちゃんと約束どおりに出来ていない場合、コンパイルでエラーとなります。

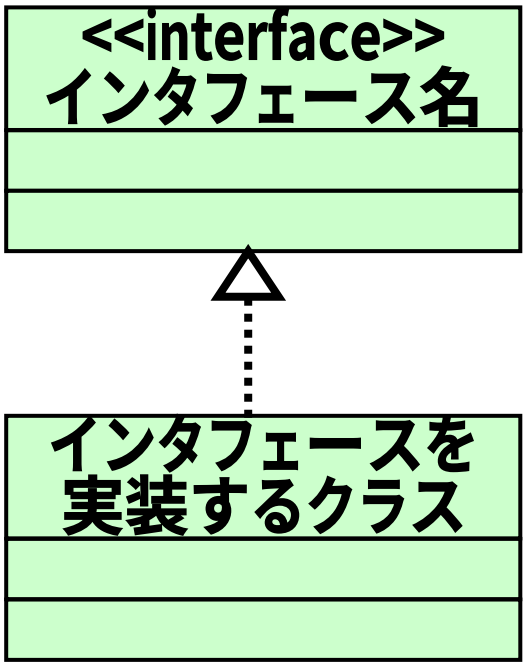


(8．3) UMLでの記述

■インタフェースは、UML上で次のように記します。



＜UML＞



■「<<interface>>」と記すことで、クラスと区別します。

■標準ライブラリで提供されるインタフェース等、良く知られたインタフェースの実装は、クラス図上では省略することが多いようです（継承についても、同じことが言えます）。

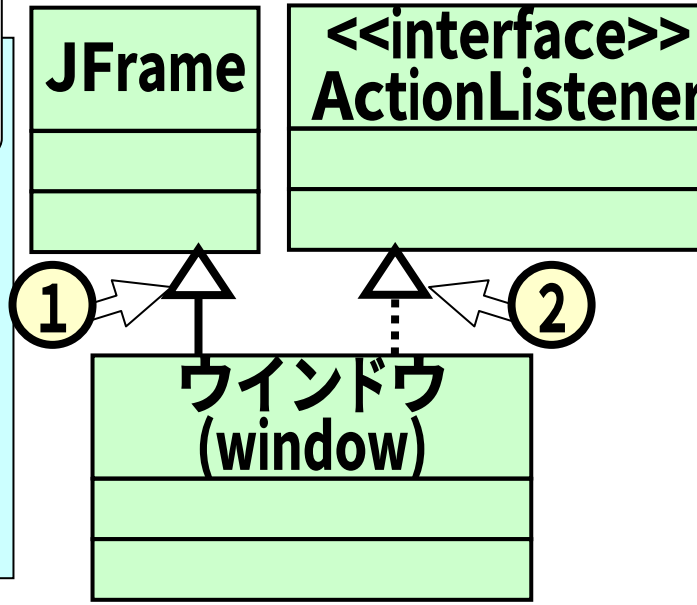
(8．4．1) ブラックジャックでのインタフェース

■ブラックジャックの“ウインドウ･クラス”は、
- クラス“JFrame”を継承しています(①)。
- インタフェース“ActionListener”を実装しています(②)。

```
// ウインドウ･クラス (Windo
package blackjack;
import javax.swing.JFrame;
import java.awt.event.ActionListener;


public class Window extends JFrame
        implements ActionListener {
    :
```

他人が作ったインタフェースを利用する場合も、クラス同様、インポートします。

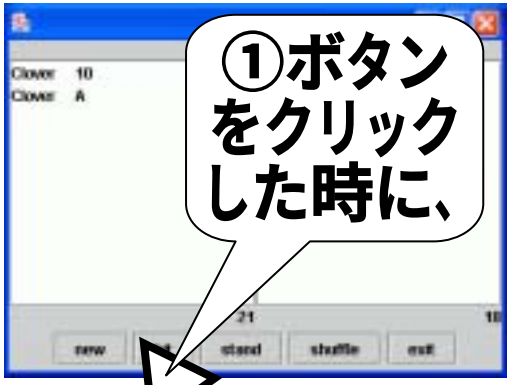


■しかしながら、このような継承･実装をクラス図に記すことはあまりないようです(自作のクラスを継承･実装する場合は、記すべきです)。

(8．4．2) 何をしたいのか？

■ブラックジャックでは、次のような動作を行います。

- ①ボタンをクリックしたときに、
- ②これを検出して、③知らせる。
- ④何か処理をする。



①ボタンをクリックした時に、

③知らせる

②これを検出して、

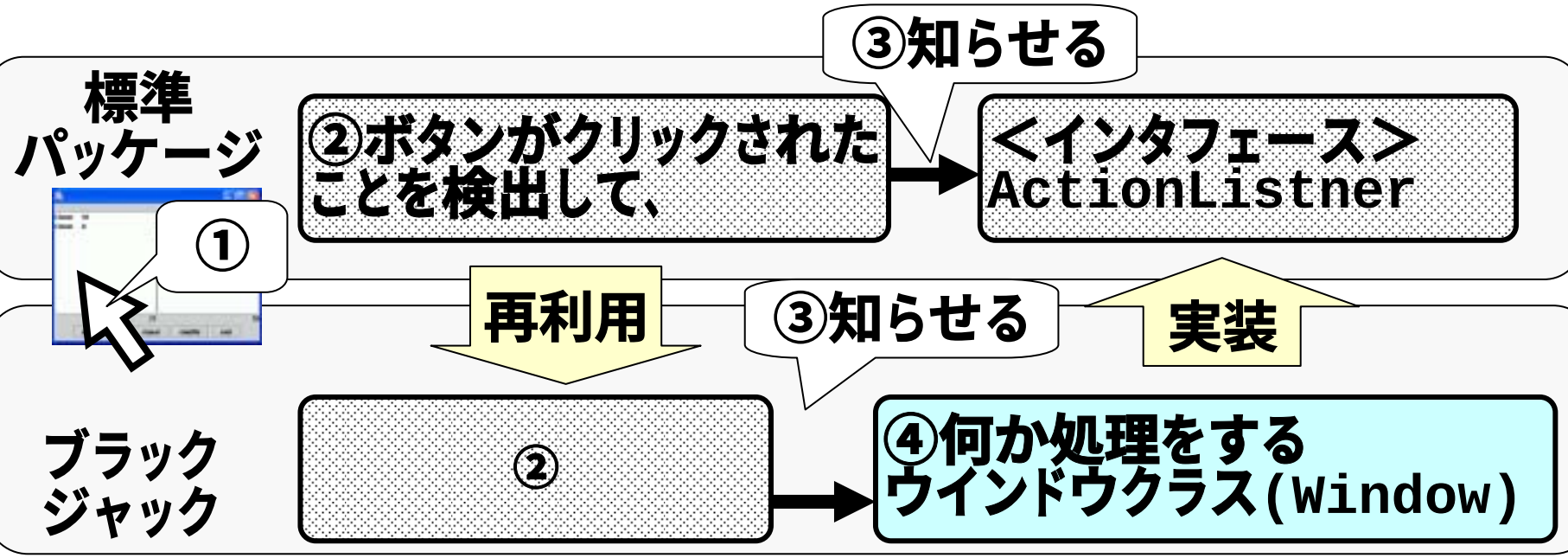
④何か処理をする

■このとき②の動作は、多くのプログラムで共通に使用するものであり、また、物理的なコンピュータを意識せねばならない点で煩雑でもあります。ここの部分は、他人が作ったプログラムを利用したいところです。

(8．4．3) ActionListener

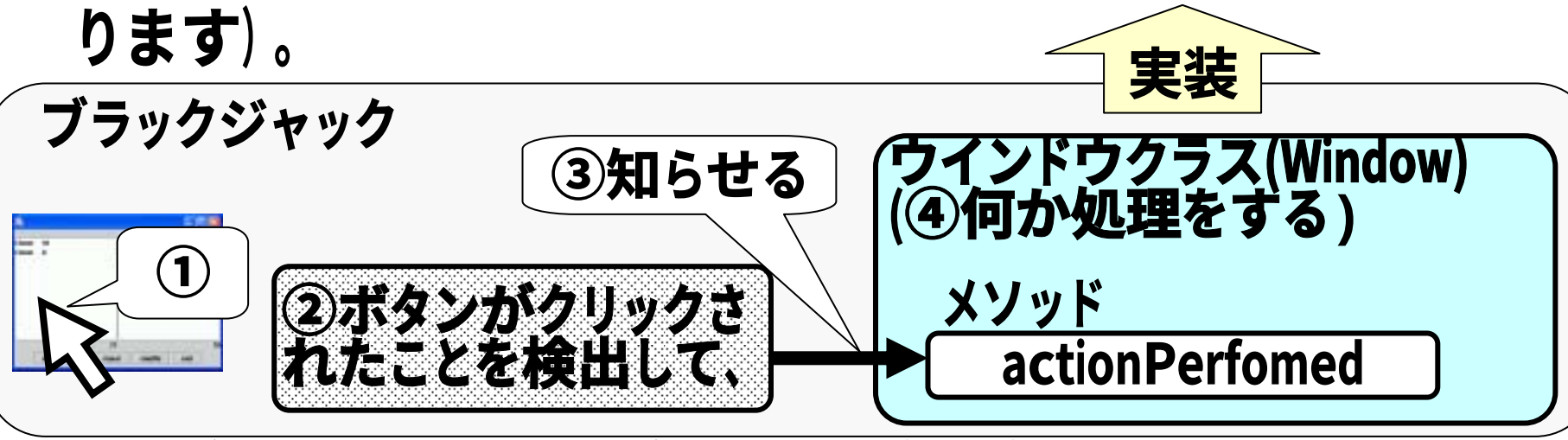
■前スライドの「②クリックを検出して、③知らせる」ところは、ウインドウクラスが、“ActionListener”を実装することにより実現されています。

■ウインドウクラスでは、「④何か処理をする」ところだけを作成すれば良いわけです。



(8．4．4) “actionPerformed”メソッド

- “ActionListener”を実装したウインドウクラスでは、当然「③知らせ」を受けるもの、すなわちメソッドが必要です。
- このメソッドが、“actionPerformed (動作が実行された) ”メソッドです。
- “ActionListener”を実装したクラスでは、必ず“actionPerformed (動作が実行された) ”メソッドを作らなければなりません (作らないとコンパイルエラーとなります) 。

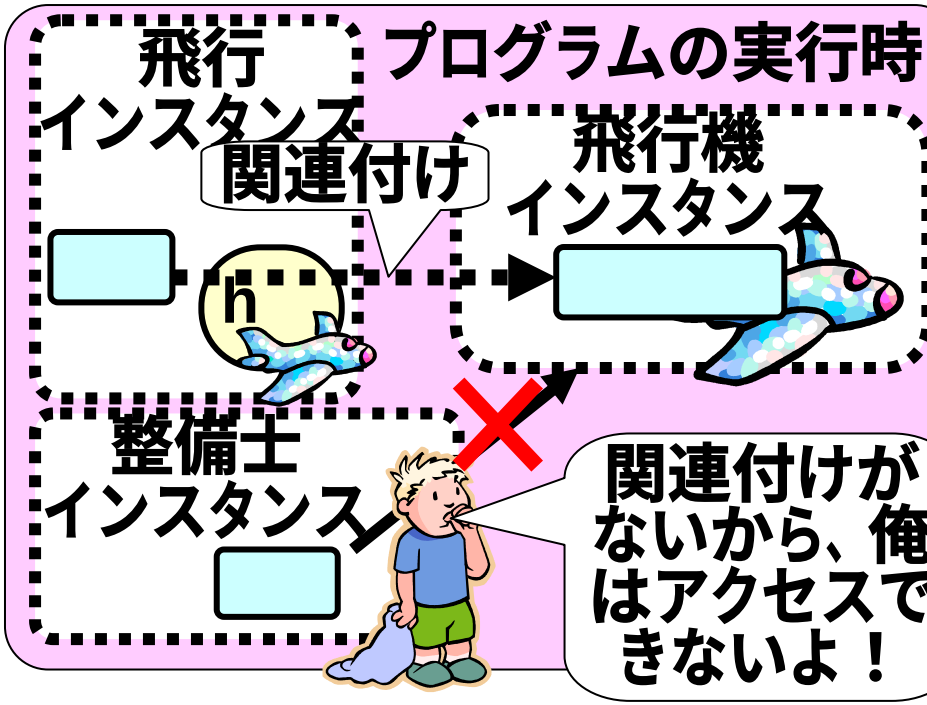


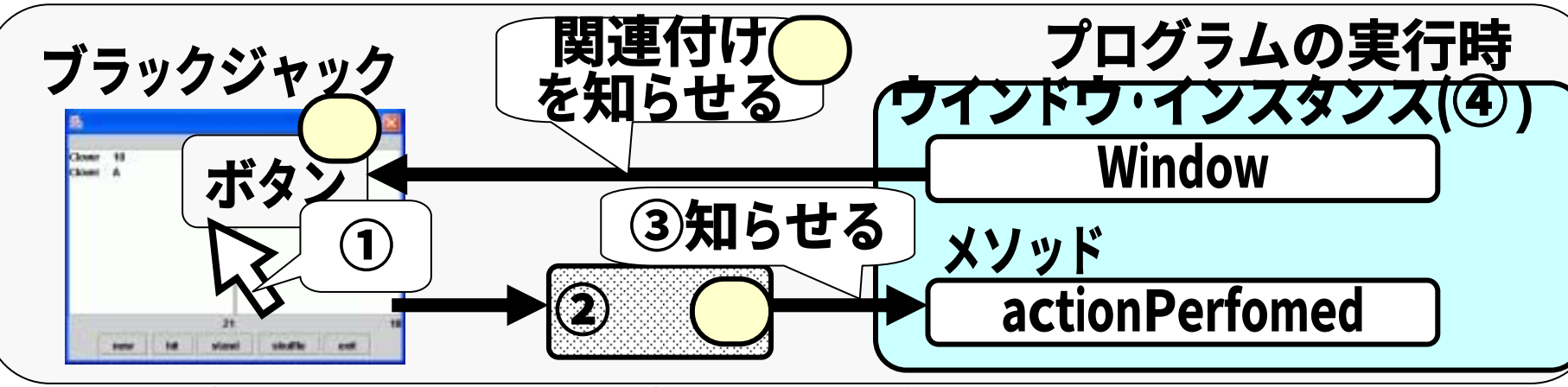
(8．4．5) ボタンとインスタンスとの関連付け

■プログラムの実行時、“ActionListener”クラスは、インスタンスとして動作します。

■スライド(6)章で見たとおり、インスタンスを呼ぶには関連付けが必要でした。

■ブラックジャックでは、ウインドウ･インスタンスは、自分自身への関連付けをボタンに対して設定しています。





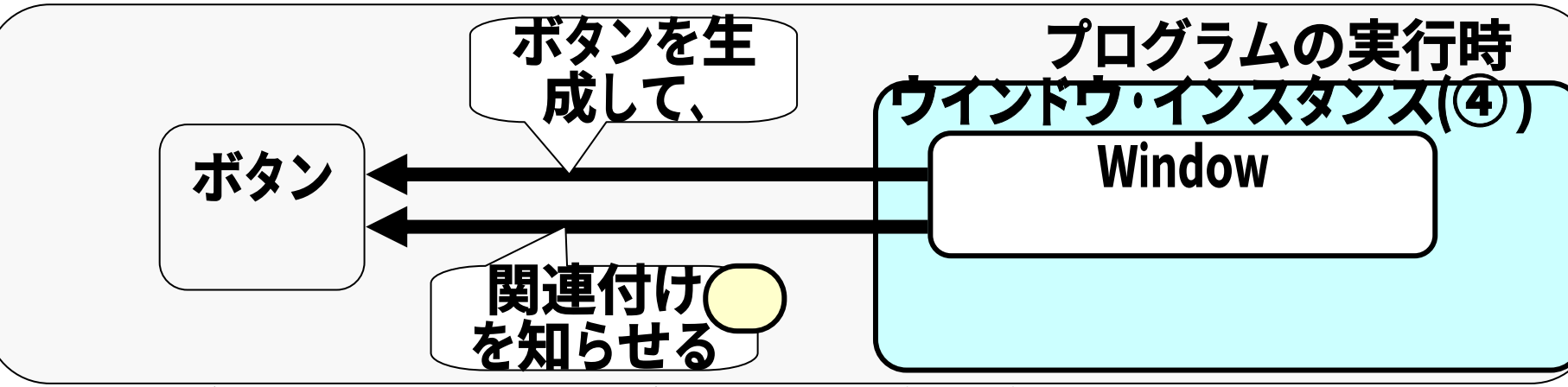
(8．4．6) プログラムの動き　－1－

■ウインドウ･インスタンスでは、最初にボタンを生成して、自分自身への関連付け（”this”）をボタンに知らせます。

```
// ウインドウ･クラス （Window.java）
     :
public class Window extends JFrame
         implements ActionListener {
     :
 public Window() {
  newButton = new JButton("new");
     :
  newButton.addActionListener(this);
     :
}
```

①

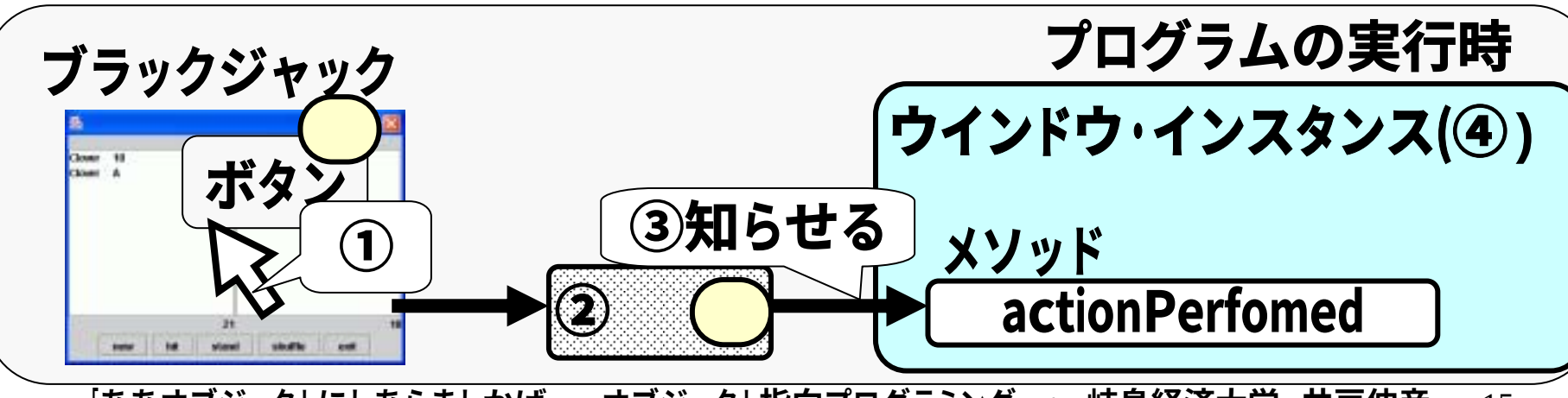
②



(8．4．7) プログラムの動き　－2－

■ボタンがクリックされると、ウインドウ・インスタンスの“actionPerformed”メソッドが呼ばれます。ここでボタンがクリックされたことによる処理を行います。

```
// ウインドウ・クラス (Window.java)
     :
public class Window extends JFrame
        implements ActionListener {
     :
  public void actionPerformed
     (Event ae){
    Object obj = ae.getSource();
    if(obj.equals(newButton)) {
    :
}
```



(8．5．1) 演習：課題9－1

■井戸のネットワークドライブから、“マイドキュメント”の下の“javawork”へ、次のファイルをコピーしてください。

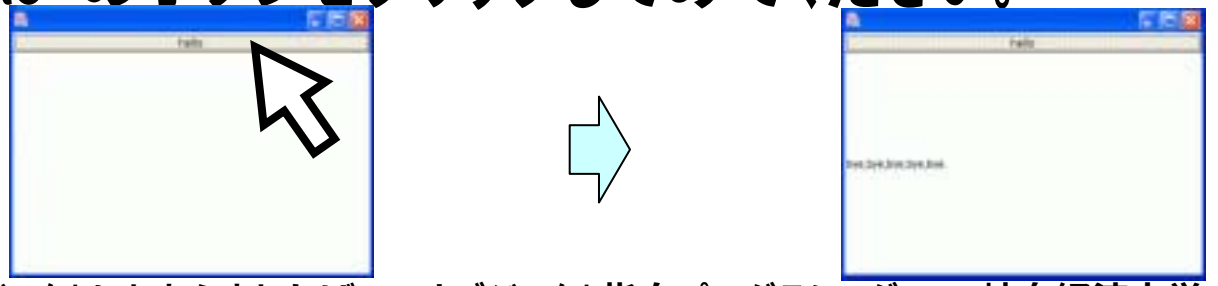
- Window.java

■コンパイルして実行します。

- [start]-[すべてのプログラム]-[アクセサリ]-[コマンドプロンプト]を選択してください。
- 次のようなイメージで動作します。

```
> cd javawork          ← javaworkのフォルダに移動
> javac Window.java    ← コンパイル
> java Window          ← Window.classのプログラムを起動
```

- “hello”のボタンをクリックしてみてください。



(8．5．2) 演習9：課題9－2

- Window.javaのファイルをメモ帳で開き、“hello”のボタンをクリックすると自分の名前が出てくるように書き換えてください。
- 再度コンパイルして実行してください。
- 出来たファイルの名前を学籍番号に変えて、井戸のネットワークドライブに提出してください。

(9．1) Javaを取り巻く世界

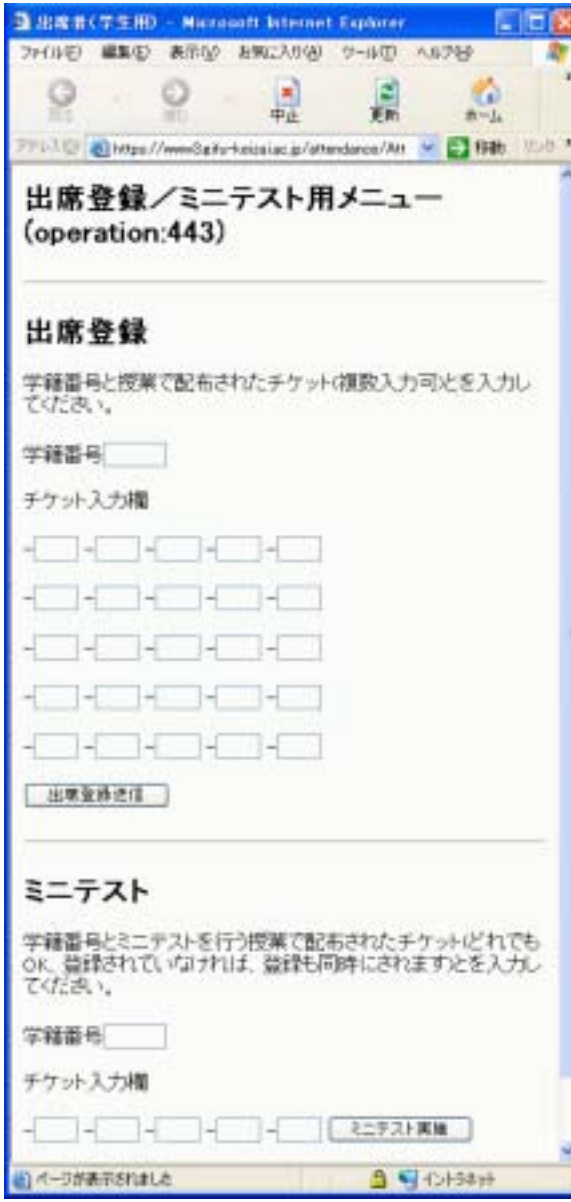
- スライド(8)章までは、主にオブジェクト指向の仕組みを記してきました。
- 仕組みが解れば、どんどんプログラムが書いていけるかと言えば、必ずしもそういう訳ではありません。
- オブジェクト指向プログラミング(具体的にはJava)の場合、「プログラムを再利用する」ことが大きな眼目であり、「人のプログラムを利用出来ること」が出来なければ意味が半減します。
- スライド(7)章で見た標準パッケージ以外にも、利用出来るものは沢山あります。

(9．2) Webアプリケーションを開発する場合

■ご存知のとおり、インターネット上Webサービスは伸張しつづけ、世の中全体を大きく変えることが予想されています。

■現在、講師(井戸)のWebサイトでは、出席確認とミニテストのシステムが利用出来るようになっています。

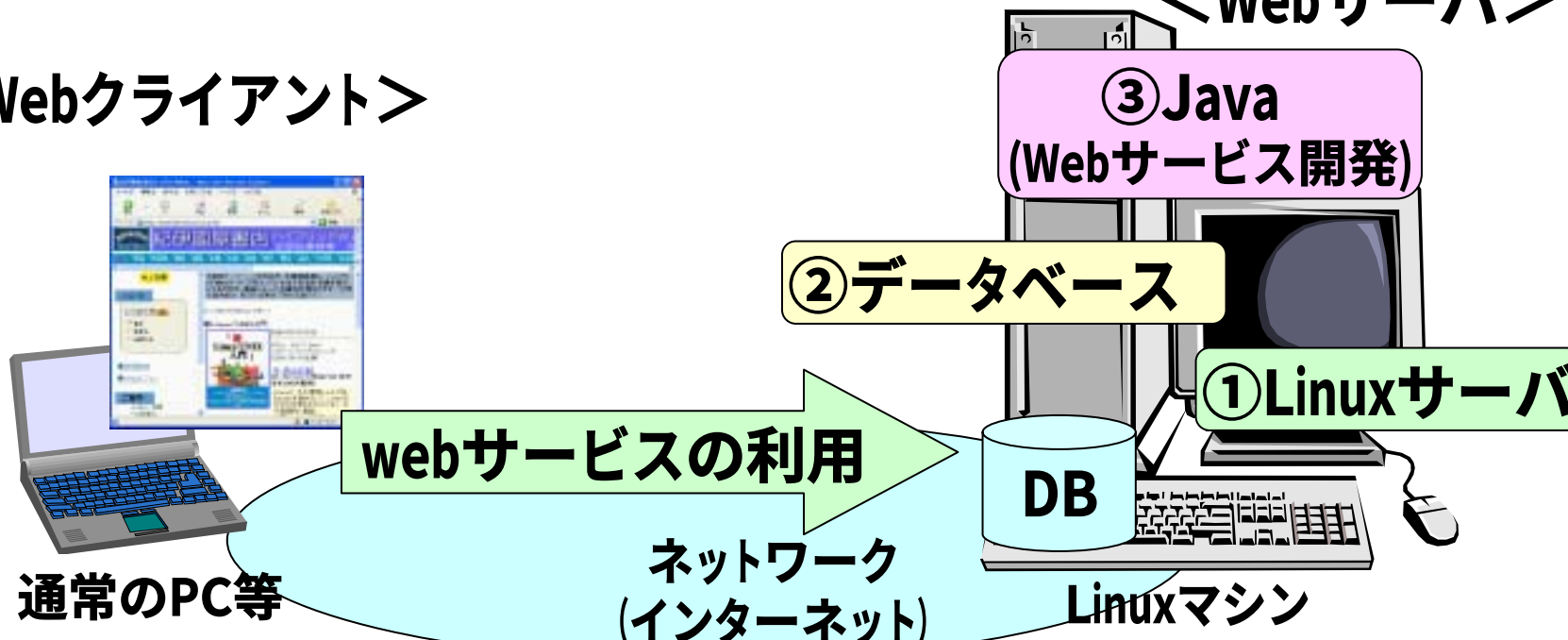
■このシステムは、Javaを使って作成されています。その構築の仕方について順番に見ていきます。



(9．2．1) サービス提供の枠組み

■出席確認のWebサービスは、講師が管理するWebサーバ上で実現され、インターネットに接続されたWebクライアント(通常のPCです)から利用します。

■Webサーバは、大別すると、①Linuxサーバ、②データベース、③Javaの3つの技術を使って構築されています。これは、Webシステム開発方法として、典型的なもののひとつです。



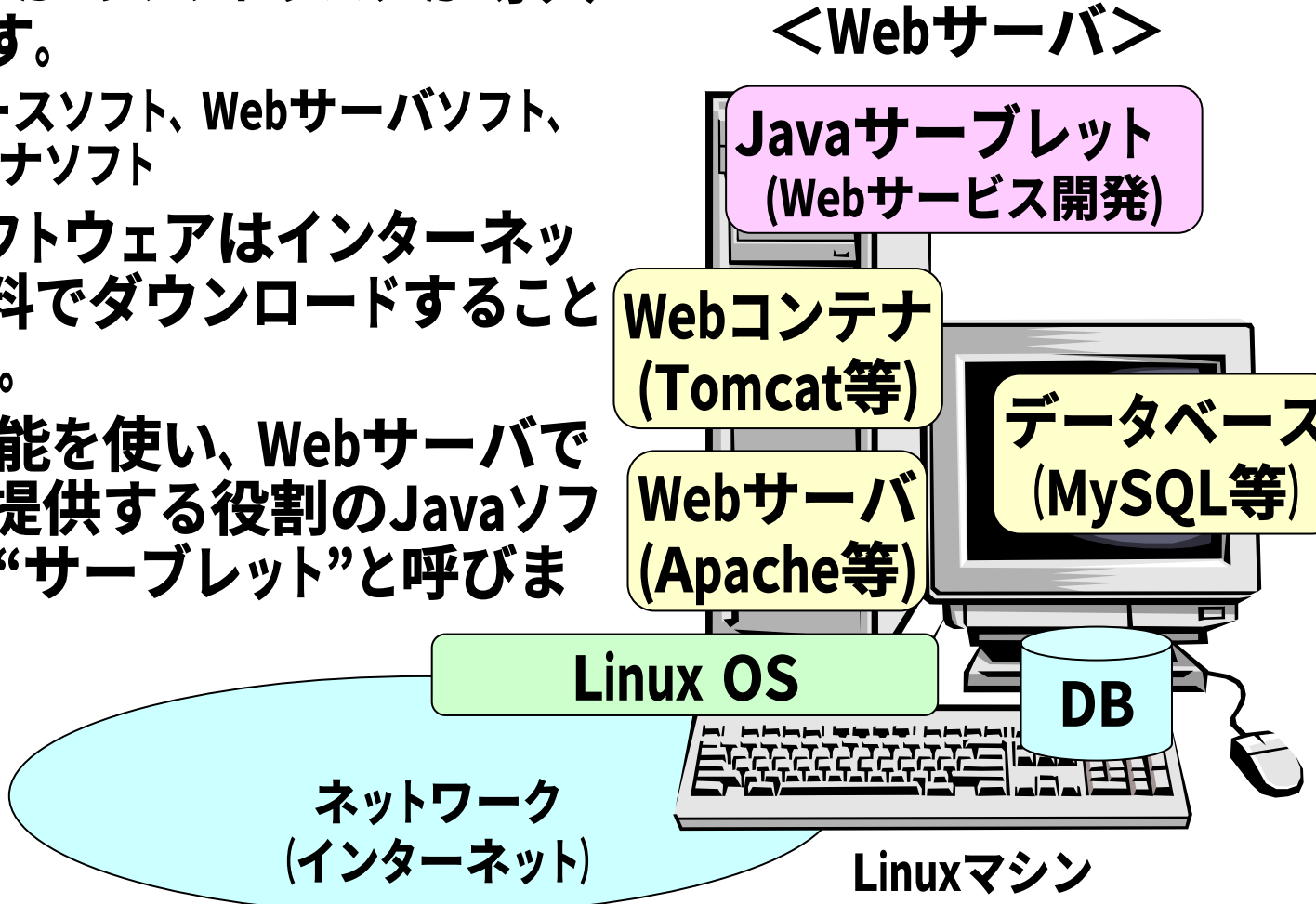
(9．2．2) Webサーバ

■Webサーバには、サービスを提供するために必要な機能を実現するために、いくつかのソフトウェアが導入されています。

- データベースソフト、Webサーバソフト、Webコンテナソフト

これらのソフトウェアはインターネット上から無料でダウンロードすることが出来ます。

■これらの機能を使い、Webサーバでサービスを提供する役割のJavaソフトウェアを、“サーブレット”と呼びます。

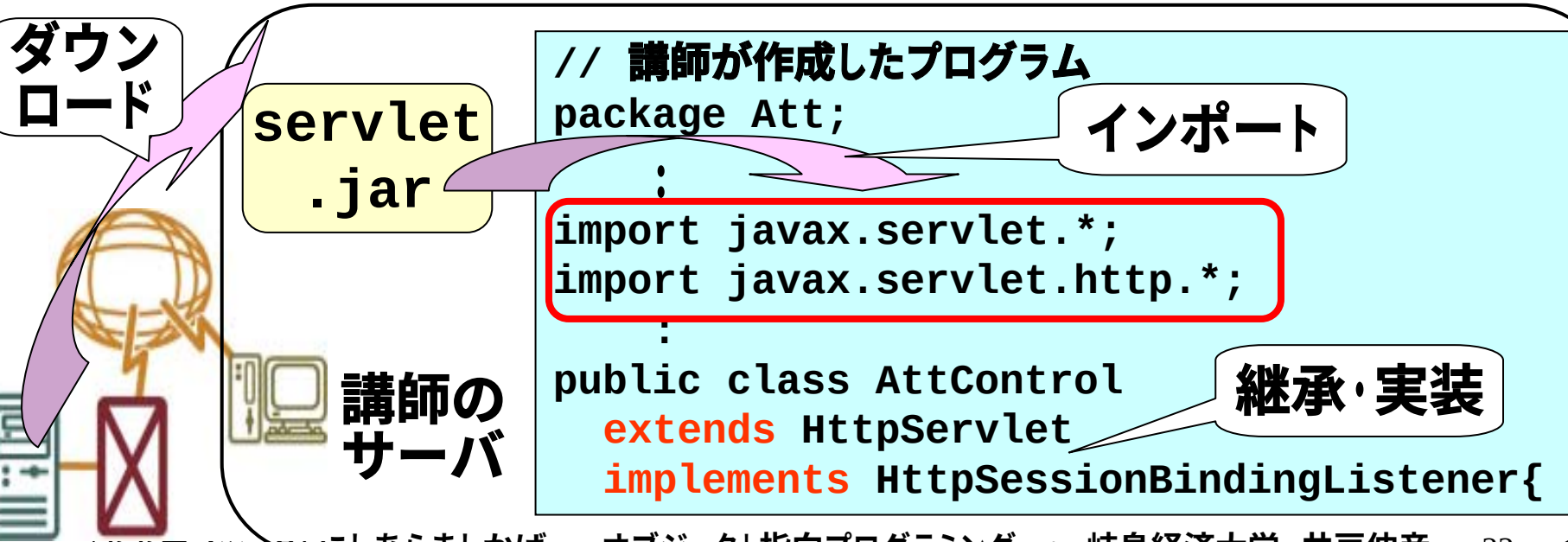


(9．2．3) サーブレット

■サーブレットはJavaで作成していく訳ですが、いちからコードを作成していく訳ではありません。

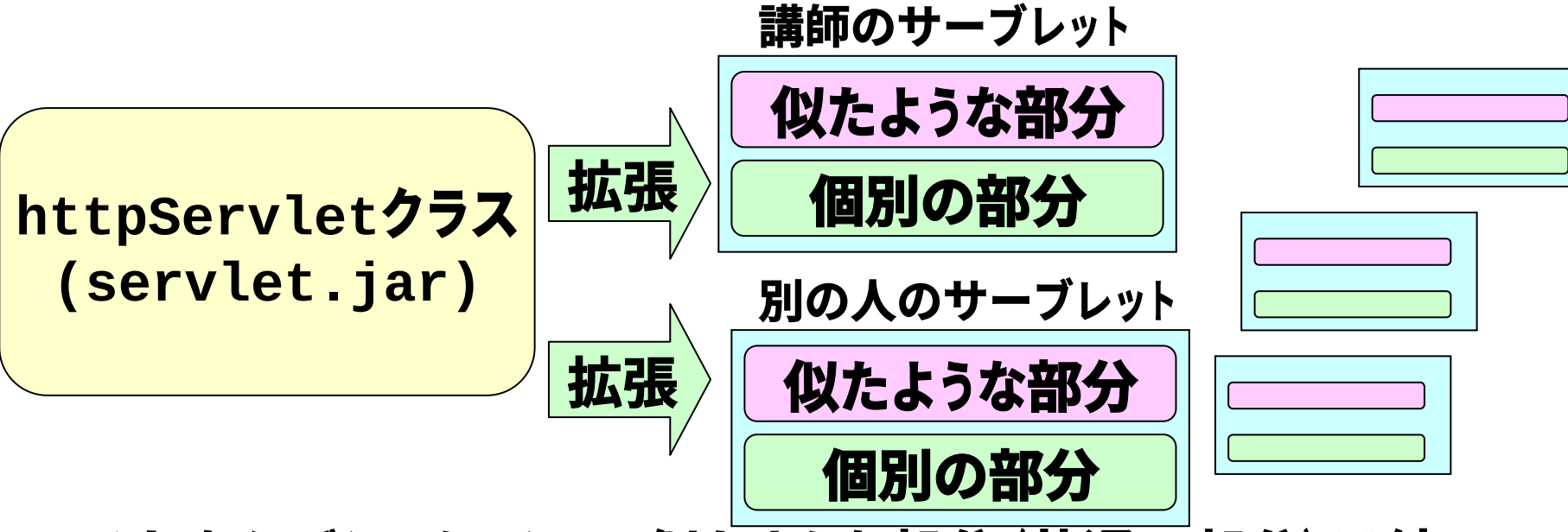
■まず、“servlet.jar”というファイルをネットワーク上から入手します。これには、サーブレットの元となるプログラムが入っています。

■“servlet.jar”からインポートしたクラス･インタフェースを継承･実装して、プログラムを作成していきます。

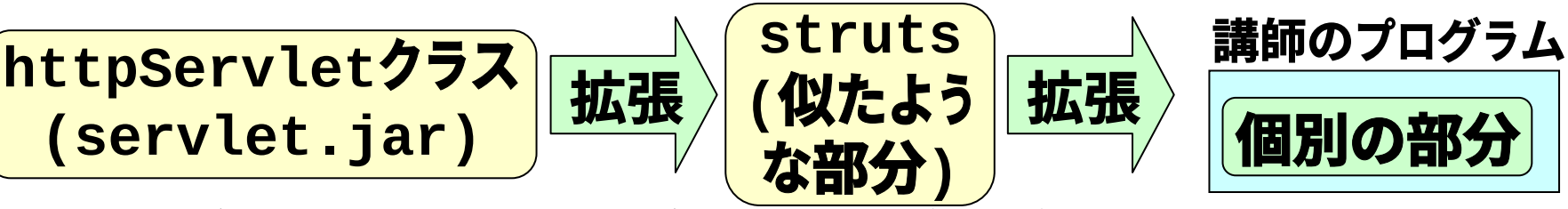


(9．2．4) さらに再利用する

■スライド(9.2.3)のように作成したサーブレットも、多くのWebサービスでは同じようなプログラムを書くことになります。



■それならばということで、似たような部分 (共通の部分) は統一のものを利用しようということで、“struts”という一群のプログラムが提案され、これを利用することが普通になっています。

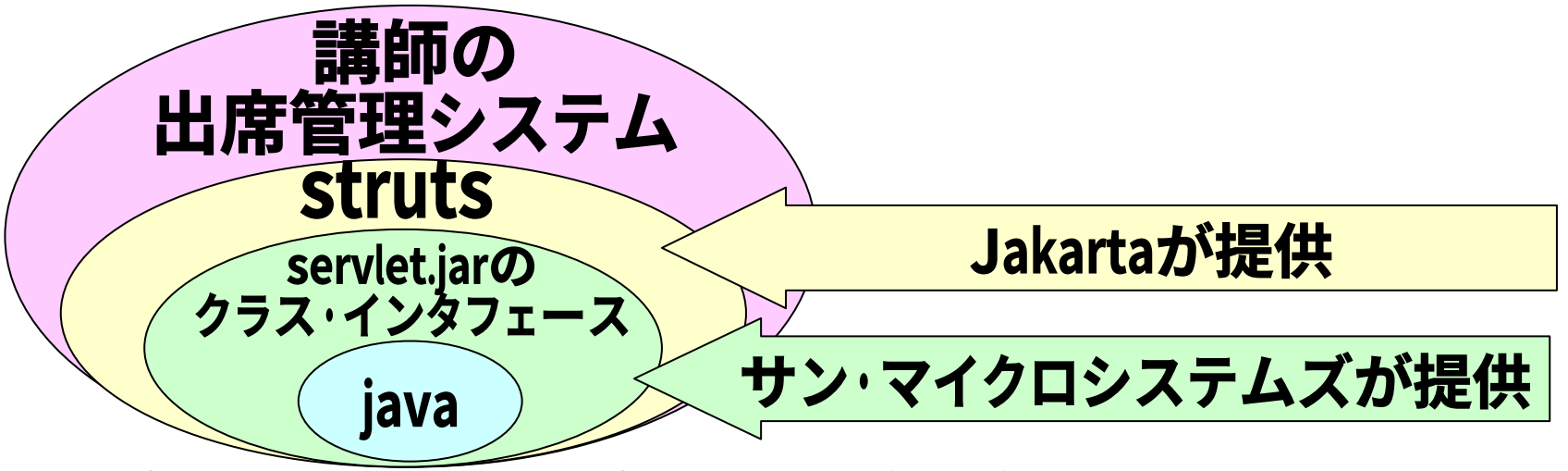


(9．2．5) 小さな整理

■Webアプリケーションを作成する場合、既に開発されている“servlet.jar”の中のクラス･インタフェースを再利用してサーブレットを開発します。

■さらに、既に開発されている“struts”という一群のJavaプログラムを再利用出来ます。

■“servlet.jar”は、Javaの大元締めであるサンマイクロシステムズから提供されていますが、“struts”は、“Jakarta”という非営利団体から提供されています。

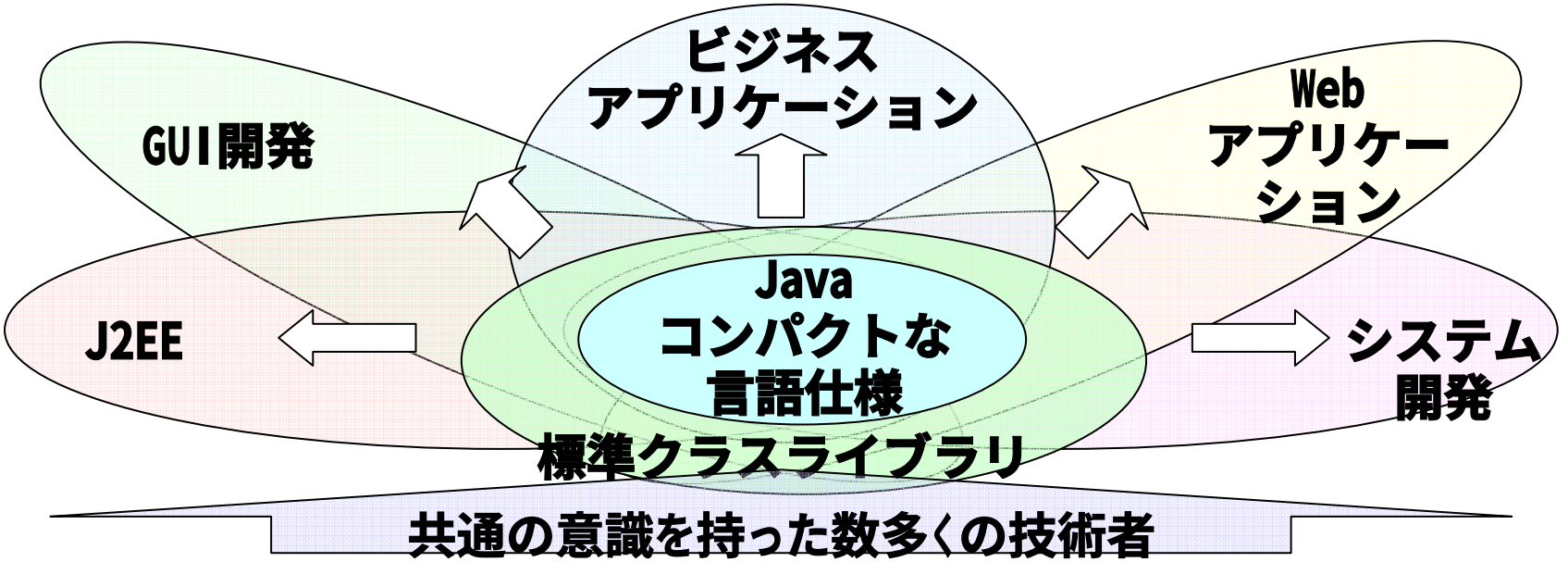


(9．3) Javaを取り巻く世界

■Java自体は比較的コンパクトな仕様を持つプログラミング言語ですが、標準ライブラリを中心に、様々な目的に利用可能なソフトウェアが開発されています。

■それらを利用する仕組みも整備されており、共通の意識を持った数多くの技術者がこの輪の中に加わっています。

■Javaはプログラミング言語であるとともに、ある種の共同体のような側面をもっています。



(9．4) 演習：課題9－1

■Web上で検索を行い、“struts”のファイルをダウンロード出来るサイトを見つけてください。

■“struts”についての説明が書かれたサイトを見つけてください。

■見つけたURLを、井戸までメールで送ってください。

- 宛先：ido@gifu-keizai.ac.jp
- 件名：情報システム設計031201