第４章．クラスの基本

【学習のねらい】

① クラスの定義とオブジェクトの生成の仕方を学習し、自分でクラスを定義できるようになる。
② クラスの継承機能を学習する。

本章では、クラスの定義やオブジェクトの生成、そして継承機能など、クラスの基本を学習します。Java 言語はオブジェクト指向プログラミング言語であると言われています。そしてオブジェクト指向プログラミングとは、様々なクラス（既存のクラス、および既存のクラスに拡張機能が付加された新たなクラス）を駆使して、プログラムを作成する手法に他なりません。したがって、クラスの基本が理解できればオブジェクト指向プログラミングの本質を理解できるようになります。オブジェクト指向については、プログラミングの技術的な事項に止まらず、その背後にある考え方や概念など学習すべき点は多々あるのですが、ここではそれらには深くは立ち入りません。後の「データ構造とアルゴリズム論」の学習に必要な部分に話を絞って（必要最小限だけ）学習することにします。

４－１　本章の学習を始める前にーちょっとした心の準備

クラスあるいはオブジェクトという言葉には、これまで何度も遭遇してきました。少し振り返っておきましょう。まず前期テキスト（以下テキスト）の p.21 を読み返して下さい。そこには

Java 言語プログラムはクラスの集まり

という記述があります。実は Java 言語でプログラムを作るということはクラスを定義する（作成する）ことに他なりません。これまでみなは、フレームに関する処理を記述したクラス（のみ）を作成してきた事になります（Frame1.java のことです）。さて、さらに、テキストの同じ箇所には、次のようなクラスの定義（の仕方）が与えられていました。

```
class クラス名 {
    クラスの定義部分
}
```

次節で、実際に（独自の）クラスを定義してみます。思ったよりも簡単にクラスを定義できる（作れる）事に気がつくでしょう。

続いてテキスト p.47 のコラムを読んで下さい。そこでは、オブジェクトについて

プロパティとメソッドを持っているモノ

と説明されていました。これにより、テキストフィールドやボタンなどのコンポーネントが全てオブジェクトであることが分かります。それでは、クラスとオブジェクトの関係はどうなっているのでしょうか･･･？それについてはテキスト p.48 のコラム「クラスとオブジェクトについて」を読み返して下さい。そこでの説明から分かるように、クラスは（必要

な）プロパティとメソッドを定義した**設計図**です。そしてその設計図に従って作られた**製品**がオブジェクトである、という関係になっています。

Java 言語では、プロパティにあたるものを**フィールド**と呼びます。ですから、Java 言語におけるクラスの定義は、**フィールドとメソッドの定義**ということになります。したがって、上で述べたクラスの定義の仕方をもう少し具体的に書くと、

＜Java 言語によるクラスの定義の仕方＞

```
class クラス名 {
  フィールドの定義
        ・・・
  メソッドの定義
        ・・・
}
```

という事になります。

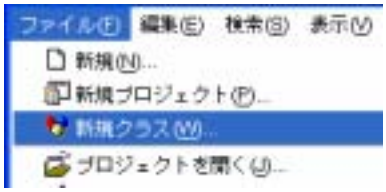
さて、それでは以上を念頭に置いて以下の学習に進みましょう。一般的な話はここまでにして、以下は具体的なプログラミングを通じて実践的に理解して行くことにします。

４－２　クラスの定義とオブジェクトの生成

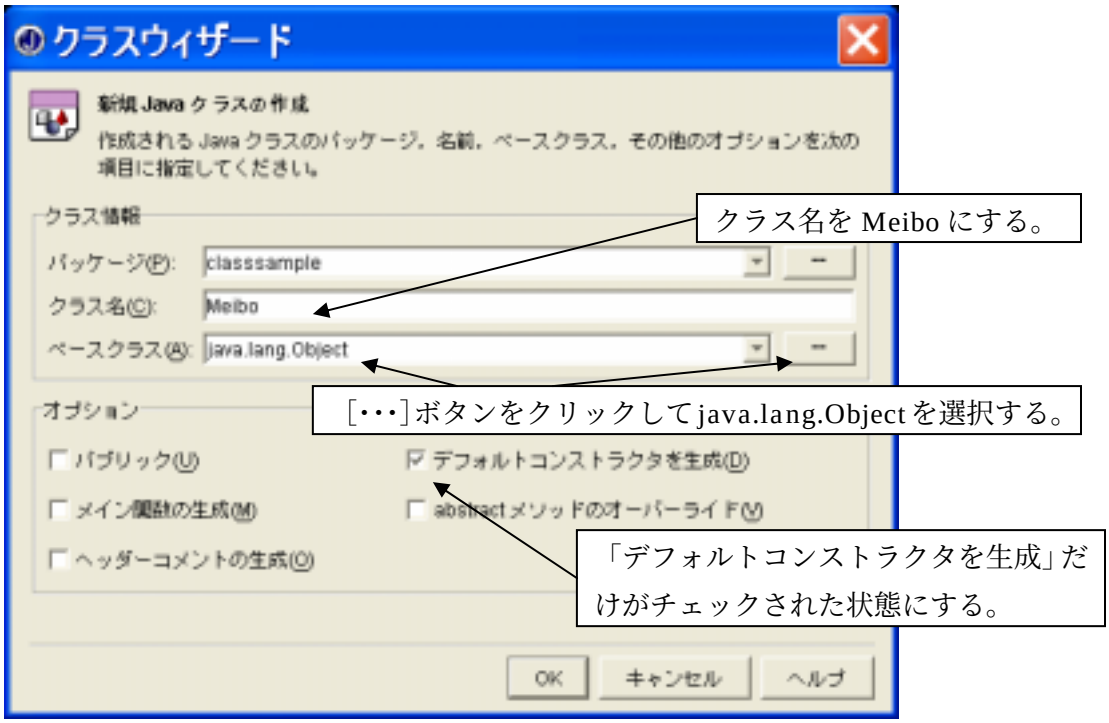
論より証拠、まずは簡単なクラスを定義して（作って）みましょう。簡単な例として、氏名と年齢がフィールド（変数）として定義され、その氏名と年齢を紹介するメッセージを与えるメソッドを有する（名簿）クラスを考えます。

【基礎課題 4-1】

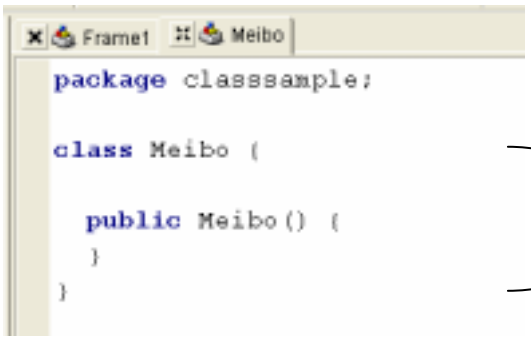
まず新規作成でアプリケーションを作って下さい。後の説明のため、プロジェクトの名前は ClassSample として下さい。それ以外は、フレームタイトルを「名簿クラス」とする他はデフォルトのままで結構です。ここまでは通常のアプリケーション作成と同様です。次にアプリケーションブラウザのファイルメニューから「新規クラス」を選択して下さい。



すると次のようなクラス設計ウィザードが現れます。ここで、クラス名を Meibo としましょう。そしてベースクラスを以下の様に設定します。また、オプション欄は「デフォルトコンストラクタを生成」欄のみがチェックされた状態にします。



設定後［OK］ボタンをクリックすると、次のように、クラス定義の編集画面（Meibo.java の編集画面）が現れます。



クラス Meibo の定義部分

ここで、上のクラスの定義部分を次のように記述して下さい。

```
class Meibo {
  private int Age;  //年齢フィールドの宣言
  private String Name;  //氏名フィールドの宣言
  public Meibo(int Nenrei,String Shimei) {
    Age=Nenrei;
    Name=Shimei;
  }

  public String getMessage() {
    String Message="私は"+Name+"と申します。"+"年齢は"+Age+"歳です。";
    return Message;
  }
}
```

①フィールド変数の定義

②コンストラクタの定義

③メソッド getMessage()の定義

＜プログラムの解説＞

① この Meibo クラスで使用するフィールド変数、Age（年齢）および Name（氏名）を定義（宣言）しています。要するに変数の宣言です。修飾子 **private** については、4-3 節で説明します。

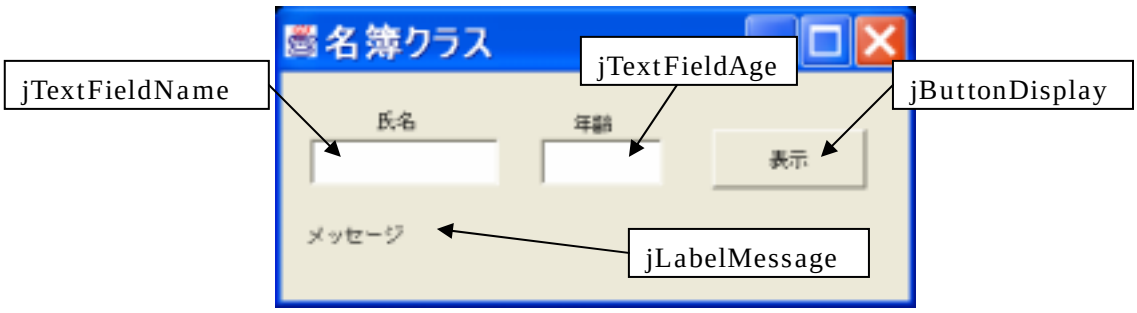
② コンストラクタとは ”建設する人 ” という意味ですが、端的に言うと当該クラスからオブジェクトを生成するときに必ず呼び出される特別なメソッドです。その意味は、以下でオブジェクトの生成を行ったときに明確になります。通常はこのコンストラクタ内でオブジェクトの初期化に必要な処理を行います。ここでは、年齢（Nenrei）と氏名（Shimei）に対応する値を引数として受け取り、それをフィールド変数 Age と Name に代入しています。なお、**public** の意味については、上と同じく 4-3 節で説明します。

③ メソッドの定義です。この `getMessage()`メソッドは氏名と年齢を紹介するメッセージ文を作成し、それを自身の値（文字列）として返すものです。テキスト 6-8 節「戻り値のあるメソッド」に他なりません。

以上で、Meibo クラスの定義は完了しました。

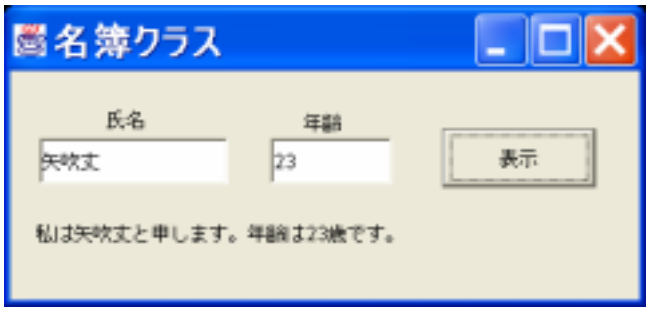
ここで、アプリケーションブラウザの Frame1 タブをクリックして Fram1.java に戻って下さい。

そして次のようなフレームを設計して下さい。



さて、これから作ろうとしているのは、次のようなプログラムです。

氏名欄と年齢欄に適当なデータを入力します。



［表示］ボタンをクリックすると、氏名と年齢を紹介するメッセージが表示されます。

それでは、［表示］ボタンをクリックしたときのイベントハンドラを次のように作成して下さい。

```
void jButtonDisplay_actionPerformed(ActionEvent e) {
  int Age=Integer.parseInt(jTextFieldAge.getText());
  String Name=jTextFieldName.getText();                    ①
  Meibo Meibo1=new Meibo(Age,Name);//名簿オブジェクト Meibo1 の生成 ②
  jLabelMessage.setText(Meibo1.getMessage());      ③
}
```

＜プログラムの解説＞

① 氏名欄と年齢欄に入力した値をそれぞれ、変数 Age および Name に代入します。

② 上で定義した Meibo クラスのオブジェクトを Meibo1 という名前で生成します。この書式は、第 3 章 3-1 節（p.35）で説明した通りです。一般に、あるクラスのオブジェクトを生成する際には

クラス名　オブジェクト名 = **new** クラスのコンストラクタ

Meibo	Meibo1	Meibo(Age,Name)	← 上のプログラムの例

のように記述します。**new** 演算子の横に来るのが**コンストラクタ**です。このように、クラスからオブジェクトを生成するときには当該クラスのコンストラクタ（メソッド）が呼び出され、そこに記述されている初期化処理（今の場合、年齢(Age)と氏名（Name）フィールドの値の設定）が行われます。

クラスの定義のところで、コンストラクタのことを「オブジェクト生成時に最初に呼び出される特別なメソッド」と説明しましたが、これでその意味が理解できたと思います。

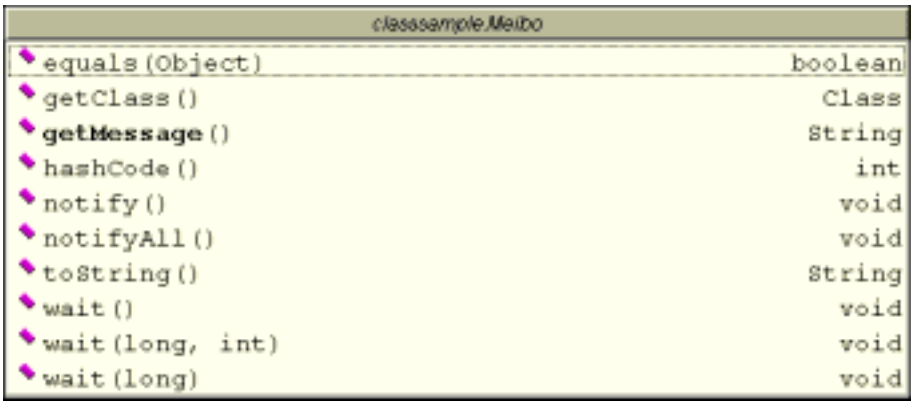
③ Meibo クラスのオブジェクトである Meibo1 には、getMessage()というメソッドを定義しておきました。そこで、「Meibo1.getMessage()」の形で、そのメソッドを呼び出すことができます。このメソッドの値は、氏名と年齢を紹介するメッセージ文でした。このメッセージがラベルに表示されるわけです。

このように、クラスを定義するとそのクラスのメソッドは「クラス名．メソッド」の形で呼び出すことができます。フィールドについても同様です。

なお、上のプログラムの最終行

```
jLabelMessage.setText(Meibo1.getMessage());
```

を記述する際、Meibo1.とキー入力した段階で少し待つと、JBuilder の補完機能により、次のように、Meibo クラスに定義されたフィールドとメソッドが表示されます。



メソッド `getMessage()`がちゃんと現れている事が分かるでしょう（それ以外のメソッド等は、ベースクラスとして選択した「`java.lang.Object`」クラスに定義されていたものです)。ところが、同じくクラスの中で定義したはずのフィールド Age と Name は現れません。それは、二つのフィールド変数を宣言する際に **private** 修飾子を指定したからです。このように、**private** を指定するとそのフィールドやメソッドは他のクラスからは参照できず、また当然ながらアクセスできません。なぜこのような事をするのでしょう。その点について、次節で少し説明しましょう。

４－３　修飾子 private と public について

まず、private や public などの修飾子に関する基本事項を以下に整理しておきます。

① private や public とはそのフィールド変数やメソッドの参照（アクセス）可能範囲を指定します。

② private は非公開という意味です。正確に言うと、それが定義されたクラスの外に対しては非公開と言うことです。ですから、private がつけられたフィールドやメソッドは、それを定義したクラス内でのみ参照可能となります。

③ public は公開という意味です。この場合は他のクラスから参照可能になります。

それでは、実際に【基礎課題 4-1】のプログラムを用いて、private が指定されたフィールド変数にアクセスできない、ということを確認しておきましょう。

【基礎課題 4-2】

【基礎課題 4-1】のプログラム（ClassSample）を開いて下さい。そして［表示］ボタンのイベントハンドラに、次のように枠線部を挿入してみて下さい。

```
void jButtonDisplay_actionPerformed(ActionEvent e) {
  int Age=Integer.parseInt(jTextFieldAge.getText());
  String Name=jTextFieldName.getText();
  Meibo Meibo1=new Meibo(Age,Name); //名簿オブジェクト Meibo1 の生成
  Meibo1.Name="学院太郎";
  jLabelMessage.setText(Meibo1.getMessage());
}
```

要するに（プログラムとしての意味はないのですが）、強引に Meibo クラスの Name フィールドに値を代入しようと試みるわけです。そして、どのようなエラーメッセージが出るかを確認して以下に記入して下さい。

＜エラーメッセージ＞

エラーメッセージを確認したら、上の枠線部を削除して下さい。

次に、Name フィールドの修飾子を public に変更してみましょう。Meibo タグをクリックして、Meibo クラスの定義部へ移動して下さい。

そして、フィールド変数 Name の修飾子を波線部のように public に変更してみて下さい。

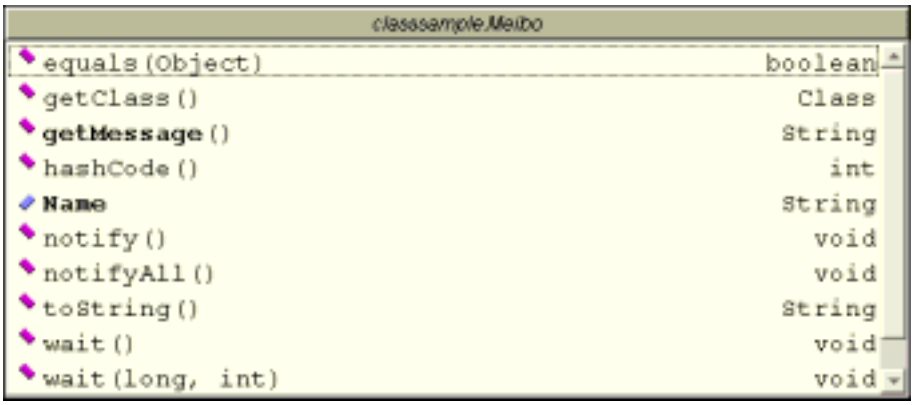
```
class Meibo {
  private int Age;
  public String Name;
  public Meibo(int Nenrei, String Shimei) {
    . . .
```

変更後 Frame1 のソースに戻って、再び、前ページで行った通り次の枠線部を記入して下さい。

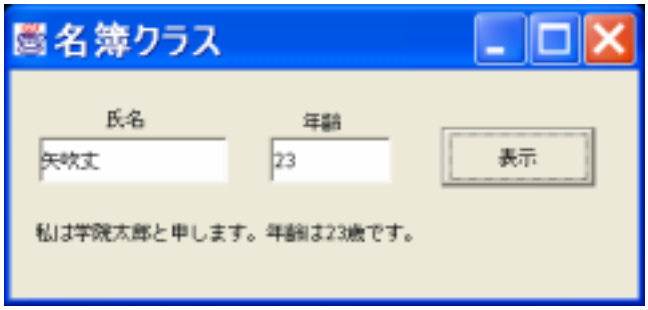
```
Meibo1.Name="学院太郎";
```

再度入力

その際、Meibo1. とキー入力した段階で少し待つと、次のリストが現れ、今度はフィールド変数 Name も表示されている、つまり参照可能であることが確認できるでしょう。したがって、上で生じたエラーは起こらずに、このまま実行できます。このことを確認して下さい。



ただし、今度の場合プログラムを実行すると、例えば次のような実行結果になり、氏名欄に入力した氏名と異なる氏名がメッセージとして表示されることになります。



動作確認後、プログラムを【基礎課題 4-1】の状態に戻して下さい（フィールド Name の修飾子を private に戻し、Frame1.java に挿入した箇所を削除する）。そして次の解説を読んで下さい。

★解説　なぜフィールド変数を private にするのか？ーデータの保護

上の例のようなおかしな結果は、オブジェクトのフィールドに不用意にアクセスしてデータを書き換えてしまったことから起こったと言えます。もっともこれは少し不自然な例ですね。しかし、プログラムがより複雑になり、クラスを作った人間とそれを利用する人間が異なるような場合を想定してみて下さい。上の例で言うと、クラス Meibo.java を作った人と、Frame1.java を作る人が異なる様な場合です。上の例では、クラス Meibo では、コンストラクタのみで氏名と年齢を設定し、それ以後はそれらフィールドの値を変更しないということが(恐らく)想定されています。フィールド変数を **private** にしている限りは、直接アクセスできないのですから、変更される心配はありません。しかし、**public** にしてしまうと、どのプログラムつまりクラスからもフィールド変数が見えてしまい、変更する事も可能になります。そこで、クラス Meibo の制作者の意図を理解していない人が、フィールド変数の値を不用意に変更してしまい、その結果、プログラムの実行結果が意図通りにならない、という事態が生じてしまいます。そこで、このようなミスを未然に防ぐために、「フィールド変数は（公開しなければ著しく不便になるなどの特別な理由がない限りは）**private** にし、不用意なアクセスによるミスを防ぐようにする」という手法がプログラム技術者の間では浸透しています。つまり、データ保護の観点から **private** にしている訳です。

しかし、途中でアクセスできないと困る場合があります。その時には、当該のフィールドの値を得る、あるいは値を設定するためのメソッドを（**public** メソッドとして）定義してそのメソッドを経由して当該フィールドの値にアクセスできるようにします。例えば、jTextField 欄に、ある文字を入力する場合には、`jTextField.setText(″文字列″)`という `setText()`メソッドを用いた事を思い出して下さい。実は text プロパティ、つまり text フィールドは **private** になっているので、直接 `jTextField.text` などのように、値を設定することができません。そこで、**public** で定義された `setText()`メソッドが用意されているわけです。

以上より、一般にクラスを定義する際には、

フィールドは **private** で宣言し、メソッドは **public** で宣言する

ということになります。もちろん、他のクラスから直接アクセスする必要があればフィールドを **public** で宣言し、同様に、クラス内部のみで使用し外部で公開する必要のないメソッドは **private** で宣言した方が良いことになります。

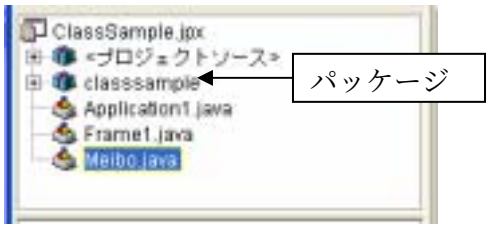
恐らく、上の説明だけではまだピンと来ない人もいる事と思います。それも仕方ありません。恐らく本当にデータ保護の必要性を実感するのは、大きなプログラムを開発する、あるいは複数の人間でプログラムを開発する、という経験をした時でしょう。その時まで上の説明を頭に止めておいてくれれば結構です。

★　補足　修飾子を指定しない場合

実はクラス内でフィールドやメソッドを宣言する際、private あるいは public を指定しなくてもエラーにはなりません。そして省略した場合は、**パッケージ**内のクラスから参照可能、という意味になります。

ここで、パッケージについては、少し説明しておきましょう。実は、JBuilder では、アプリケーションの新規作成を行う場合、自動的に関連するプログラム（クラス）を一つのパッケージ（プロジェクト名と同名）にまとめてくれていたのです。例えば、前節で作ったプログラムの場合、プロジェクトペインは次のようになっています。これは、「classsample」パッケージの中に、「Application1.java」、「Frame1.java」そして「Meibo.java」というプログラム（クラス）が含まれることを意味しています。

＜プロジェクトペイン＞



この３つのプログラム（クラス）の冒頭部を見て下さい。いずれも、

```
package classsample;
```

と記述されているはずです。これは、当該プログラム（クラス）が「classsample」パッケージに含まれていることを意味しているのです。一般に Java 言語では、関連するクラスを一つのパッケージにまとめておきます。それは、皆が関連するファイルをフォルダにまとめるのと全く同様です。

さて、修飾子の話に戻りましょう。修飾子を省略すると、このパッケージ内のクラスからはアクセス可能になります。しかし、別のパッケージのクラスからは参照できません。一方 public にすると、別のパッケージのクラスからもアクセス可能となります。

【基礎課題 4-3】

修飾子の指定によって、クラスのフィールドやメソッドのアクセス範囲を指定することができます。それらの関係を下表にまとめました。アクセスできる場合は○できない場合は×を記入して表を完成させて下さい。

場所	private	省略	public
同じクラス	○	○	○
パッケージ内の（他の）クラス			
パッケージ外の（他の）クラス			

※　上の表の１行目が全て○であるのは、同じクラス内であれば、private、修飾子省略そして public いずれの場合も（当該フィールドあるいはメソッドに）アクセス可能であることを意味しています。

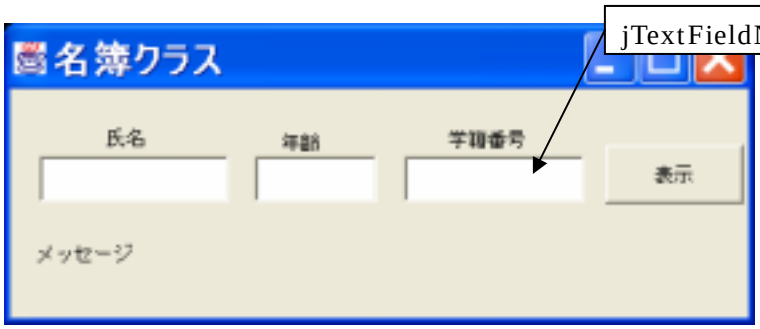
４－４　クラスの継承機能

クラスには、**継承**という大変便利な機能があります。今、ある既存のクラスと大部分は同じであるものの、そこに新たなフィールドやメソッドを加えたクラスを新規に定義したい、という場面を想定しましょう。このような場合（たとえ、カット&ペーストを利用するにしても）、最初から新たなクラスを定義することは効率が悪い様に思えます。同等の部分はそのまま受け継ぎ、変更部分のみを修正するだけで新たなクラスを（効率よく）定義することはできないでしょうか？実はそのような都合の良い処理を実現してくれるのが、**継承**機能なのです。ここでは、【基礎課題 4-1】で作成した Meibo クラスを拡張するという場面を想定して、継承機能を使ったクラスの定義の仕方を学習しましょう。

【基礎課題 4-4】

【基礎課題 4-1】のプログラム（プロジェクト ClassSample）を開いておいて下さい。今、Meibo クラスに学籍番号（No）フィールドを追加して、さらに、その学籍番号もメッセージに加えるように拡張したクラス Gakuseki を定義する事にしましょう。

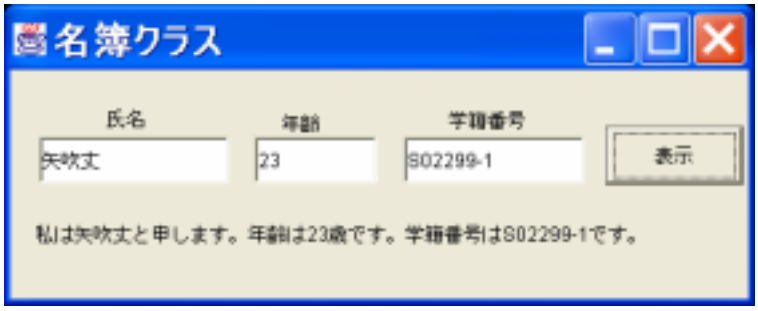
作成するプログラムは次の通りです。



プログラムの起動画面は次の通りです。

【基礎課題 4-1】のプログラムに次のように学籍番号欄を追加します。

プログラム起動後、氏名、年齢そして学籍番号欄に適当なデータを入力します。



入力後、［表示］ボタンをクリックすると、氏名と年齢に加えて学籍番号もメッセージとして表示されます。

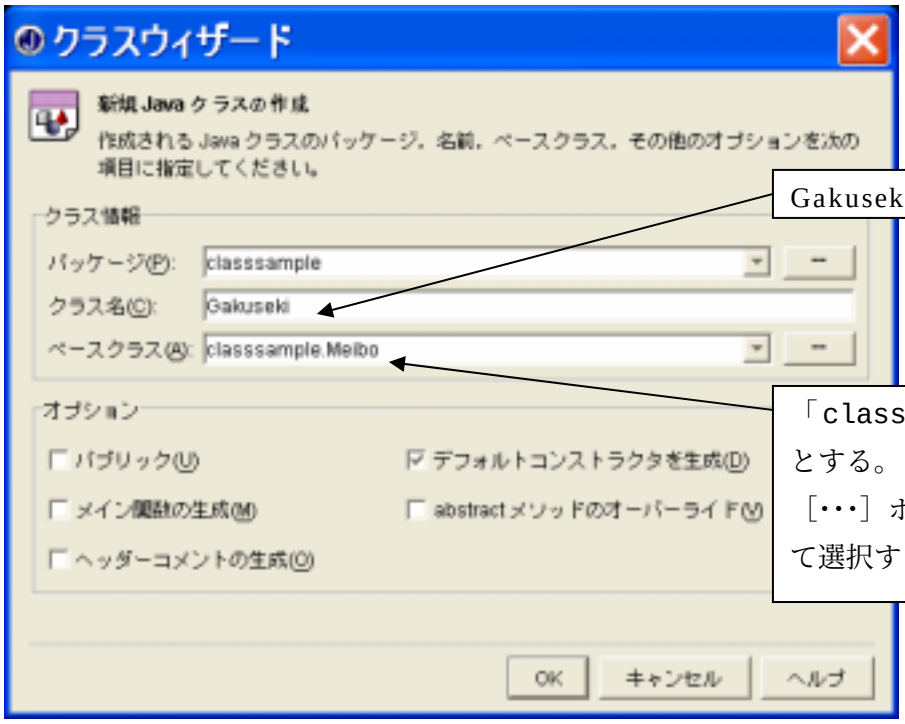
この処理を、Meibo クラスを拡張した Gakuseki クラスを定義し、それを用いることでプログラムしてみることにしましょう。Gakuseki クラスの拡張点は以下の通りです。

① フィールドに No（学籍番号）が加わる。

② メッセージを求めるメソッド `getMessage()`の処理内容に学籍番号表示部分を追加する。

逆に上の 2 点以外は、Meibo クラスのままでよい、つまり継承すれば良いわけです。

それでは、クラス Gakuseki を定義しましょう。要領は、4-2 節で行ったクラスの定義の仕方と同様です。【基礎課題 4-1】のプログラム（ClassSample）が開いた状態で、［ファイル］→［新規クラス］を選択して下さい。すると、（以前同様）クラス設計ウィザードが現れます。ここで、次のようにクラス名とベースクラスを入力します。



ベースクラスとは、継承元となるクラスのことです。今の場合、Meibo クラスを基に拡張するので、Meibo クラスを指定します。その際、パッケージ名を先頭につけて「パッケージ名．クラス名」と指定する必要があります。入力後［OK］ボタンをクリックすると、（やはり以前同様）Gakuseki クラスのファイル定義画面（Gakuseki.java の編集画面）が現れます。

```
package classsample;

class Gakuseki extends Meibo {

  public Gakuseki() {
  }
}
```

クラスの定義部分

この時点で、コンストラクタに関する警告メッセージが現れていますが、気にしなくて結構です。

クラスの定義部分を以下のように記述して下さい。

```
class Gakuseki extends Meibo {                                          ①
  private String No;                                                    ②
  public Gakuseki(int Nenrei,String Shimei,String Bangou) {             ③
    super(Nenrei,Shimei);                                               ④
    No=Bangou;                                                          ⑤
  }

  public String getMessage() {
    String Message=super.getMessage();                                  ⑥
    Message=Message+"学籍番号は"+No+"です。";                              ⑦
    return Message;
  }
}
```

＜プログラムの解説＞

① **extends** は拡張するという意味です。一般に

```
class クラスA extends クラスB {
    ・・・
}
```

の形で、クラス A はクラス B を**継承して**定義される、という意味になります。この命令によって、クラス B の性質は全てクラスAに引き継がれます。

なお、継承元のクラスBのことを**スーパークラス**、継承された結果のクラスAを**サブクラス**と言います。

② 新たに必要になった、（学籍番号に対応する）フィールド変数 No を宣言します。

③ ③～⑤は Gakuseki クラスの**コンストラクタ**です。今の場合、（学籍番号の分だけ）引数が一つ増えました。

④ **super** という変数名は、継承元のクラス（スーパークラス）を指します。つまり、ここでは Meibo クラスを指します。したがって、**super**(Nenrei,Shimei) は Meibo(Nenrei,Shimei)を意味します。つまり Meibo クラスのコンストラクタが呼び出されているわけです。これにより、氏名と年齢のフィールド変数への値の設定は完了します。

⑤ ④で氏名と年齢の設定は終わったので、新たに加わった学籍番号の値をフィールド変数 No に代入します。

⑥ ④で説明した通り、**super**.getMessage()は Meibo.getMessage()を意味します。したがって、この文で、氏名と年齢を表示するメッセージが得られます。

⑦ 新たに加わった学籍番号のメッセージを追加（連結）します。

以上で、Name（氏名）と Age（年齢）に加えて No（学籍番号）をフィールド変数として有し、それら３つを紹介するメッセージを与える getMessage()メソッドを有する、新たなクラス Gakuseki が定義できました。氏名と年齢に関する処理は、Meibo クラスから継承されているので、明示的に記述していません（記述する必要がありません）。新たに加わった部分のみを記述すれば良いのです。これが継承機能のメリットです。

さて、Farem1.java に戻って下さい。ここで、学籍番号欄のテキストフィールド（jTextFieldNo）を追加の上、［表示］ボタンのイベントハンドラを以下の通り修正して下さい。枠線部を追加し、波線部の通りに修正すれば良いのです。

```
    void jButtonDisplay_actionPerformed(ActionEvent e) {
      int Age=Integer.parseInt(jTextFieldAge.getText());
      String Name=jTextFieldName.getText();
      String No=jTextFieldNo.getText();
      Gakuseki Gakuseki1=new Gakuseki(Age,Name,No);
      jLabelMessage.setText(Gakuseki1.getMessage());
    }
```

学籍番号用の変数を追加し、クラスを Meibo から Gakuseki に変更しただけですので、説明の必要はないでしょう。作成後は実行して動作を確認して下さい。

以上のように、基本的なクラスから継承によってある機能を付加して新たなクラスを定義する、という形で、単純なものからより複雑なものへクラスを発展させて行くことが可能です。このように既存のクラスの再利用を効率よく行うための機能が継承機能なのです。

★　補足　Object クラスについて

ここで、鋭い人は、【基礎課題 4-1】で Meibo クラスを作成した際に、ベースクラスとして指定した Object クラスは実は Meibo クラスのスーパークラスではないか、と気づいたと思います。実は、

```
class Meibo {

}
```

を厳密に記述すると

```
class Meibo extends Object {

}
```

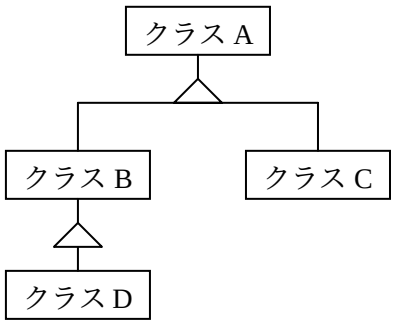
ということになります。つまり、Meibo クラスは一から作ったクラスのように見えましたが実は Object クラスを継承してできたクラスだったのです。このように Java 言語では全てのクラスのルーツとなる Object クラスを用意しておき、そこからの継承によって様々なクラスを定義（作成）しています。その意味で（ユーザが定義する）全てのクラスは継承によって作られることになります。

【基礎課題 4-5】

クラスの継承について以下の問１、問２に答えて下さい。

今、「クラス A を継承してクラス B とクラス C を作り、さらにクラス B を継承してクラス D を定義した」という関係が成り立っている場合、これらクラスの継承関係をＯＭＴ(Object Modeling Technique)という記法を用いて次のような図で表せます。この図をクラスの継承図と呼びます。

＜クラスの継承図＞



問１　以下の①～④は上の継承関係を説明した文です。下線部に「スーパークラス」あるいは「サブクラス」のいずれかを記入して文を完成させて下さい。

① クラス A はクラス B の＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿である。

② クラス C はクラス A の＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿である。

③ クラス B はクラス D の＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿である。

④ クラス B はクラス A の＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿である。

問２　継承について説明している以下の文で正しいものに○、正しくないものに×をつけて下さい。

① サブクラスはスーパークラスの性質（機能）を受け継いでいる。
② スーパークラスはサブクラスの性質（機能）を全て持っている。
③ 一般に、サブクラスとは、スーパークラスを複製したものである。
④ 一般にサブクラスになるほど、性質（機能）が付加されて行く。

【応用課題 4-A】

継承の練習をしてみましょう。【基礎課題 4-4】では、Meibo クラスを継承し、それに学籍番号に関する処理を付加した Gakuseki クラスを定義しました。そこで、さらに適当な項目をメッセージに加えるようにしましょう（男女別や学年など何でも結構です）。この拡張を、Gakuseki クラスからの継承によって新たなクラスを定義しそれを用いることで行って下さい（Gakuseki クラスに当該フィールド変数を付加し、その内容もメッセージに加えるよう `getMessage()`メソッドを修正します）。

プログラムの作成は、これまで用いてきた ClassSample プロジェクトを用い、それにクラス定義を加える形で行って下さい。