Ｊａｖａ入門

(№.４)

科　名		氏　名	
Tutorial group			

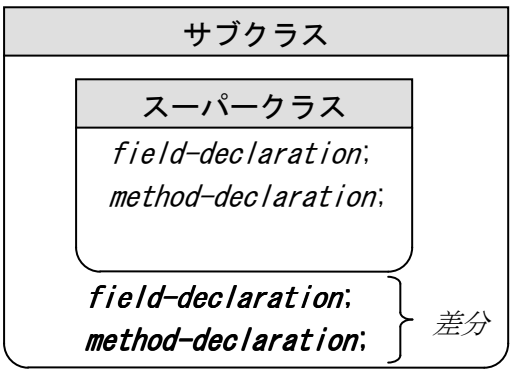
2004.12

1 継承(inheritance)

あるクラスを基にして、新しいクラスを作成することを**派生**(derivation)と言い、このときのデータメンバ/フィールドやメソッドなどを新しいクラスに引き継ぐことを**継承(inheritance)**と言う。

これにより、差分プログラミング(incremental programming)が可能となり、クラス定義は差異部分のコーディングだけで済み、プログラミングの手間が大幅に省力化できる。

継承では、基となったクラスを**スーパークラス**(super class:**基本クラス**)と言い、**派生**によって新たに作成されたクラスを**サブクラス**(sub class:**派生クラス**)と言う。



サブクラスは、**スーパークラス**の非 private メンバにアクセスできる。

サブクラスは、**スーパークラス**に対して独自のメンバを追加できる。

サブクラスは、**スーパークラス**の非 private メンバを**オーバーライド**によって置き換えることができる。

Java では、二つ以上の**スーパークラス**を基にした**多重継承(multiple inheritance)**をサポートしていない。

なお、各最上位階層のクラスは、暗黙的にクラス"java.lang.object"を**継承**している。

(例)
```
class   vehicle {
    statements;
}
```
⇒ 同じ意味である。
```
class   vehicle extends Object  {
    statements;
}
```

【構文①】－ 継承(inheritance)

```
access  class  class-name  extends  super-class  {
    field-declaration;
    method-declaration;
}
```
追加、**オーバーライド**するメンバを記述する。

access … **アクセス指定子**は、クラスをアクセスできる範囲やクラスの性質を指定する(詳細は、「**資料№.３３クラスの宣言/定義**」を参照)。

super-class … このクラスの**スーパークラス**を指定する。

解説：クラス *super-class* を基に、新たにクラス *class-name* を派生する。

Java では、**多重継承**はできない。

【例題 1】

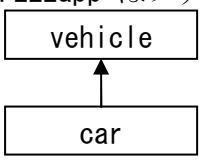
［ファイル名］

P222app.java, P221vehicle.java

［プログラムの説明］

テキスト・P221～P222 vehicle クラス、car クラス及び app クラスのプログラムを作成しなさい。ただし、クラス名"vehicle"はクラス名"P221vehicle"と、クラス名"car"はクラス名"P221car"と、またクラス名"app"はクラス名"P222app"とすること。

なお、クラス名"P221vehicle"とクラス名"P221car"はファイル"P221vehicle.java"に、クラス名"P222app"はファイル"P222app.java"に保存すること。



⇐……**スーパークラス**

⇐……**サブクラス**　　**サブクラス**は、start()メソッドを**継承**する。

派生グラフ

【考察】

・ 前述(例)に倣って、クラス名"P221vehicle"とクラス名"P222app"のクラス宣言に"**extends Object**"を追加コーディングし、動作を確認しなさい。

【例題 2】

［ファイル名］

P223app.java

［プログラムの説明］

テキスト・P223～P224 vehicle クラス、car クラス及び app クラスのプログラムを作成しなさい。ただし、クラス名"vehicle"はクラス名"P223vehicle"と、クラス名"car"はクラス名"P223car"と、またクラス名"app"はクラス名"P223app"とすること。

【考察】

【例題 3】

［ファイル名］

P224app.java

［プログラムの説明］

テキスト・P224～P225 vehicle クラス、car クラス及び app クラスのプログラムを作成しなさ

い。ただし、クラス名"vehicle"はクラス名"P224vehicle"と、クラス名"car"はクラス名"P224car"と、またクラス名"app"はクラス名"P224app"とすること。

【考察】

スーパークラスのコンストラクタ

スーパークラスの**コンストラクタ**は、**サブクラス**にはすべて継承されない。これは、**コンストラクタ**とは、クラス独自のオブジェクト構築の仕組みの一環であるからである。

一方、**サブクラス**のオブジェクトを実引数を指定せずに生成する場合、**スーパークラス**の**デフォルトコンストラクタ**が自動的に呼び出される。なお、**サブクラス**から仮引数付き**コンストラクタ**を呼び出す場合は、super()メソッドを使用する。

【構文】 － super()メソッド

super(実引数のリスト);

解説：**サブクラス**の**コンストラクタ**の先頭に super()メソッドの記述があると、Java コンパイラは、**スーパークラス**の仮引数のリストの記述が一致する**コンストラクタ**を呼び出す。

【例題 4】

[ファイル名]

P226app.java

[プログラムの説明]

テキスト・P226 a クラス、b クラス及び app クラスのプログラムを作成しなさい。ただし、クラス名"a"はクラス名"P226a"と、クラス名"b"はクラス名"P226b"と、またクラス名"app"はクラス名"P226app"とすること。

【解説】

```
class  b   extends a   {
}
```
} クラス本体が空の状態

↑の記述は、次のように解釈してコンパイルされる。

```
class  b   extends a   {
    b(){
        クラス a のデフォルトコンストラクタを呼び出す記述。
        super();      ← 記述されているものとして、処理される。
    }
}
```

【考察】

【例題 5】

［ファイル名］

P226app1.java

［プログラムの説明］

テキスト・P226～P227 a クラス、b クラス及び app クラスのプログラムを作成しなさい。ただし、クラス名"a"はクラス名"P226a1"と、クラス名"b"はクラス名"P226b1"と、またクラス名"app"はクラス名"P226app1"とすること。

【考察】

・ **コンストラクタ**の呼び出し順序を考察しなさい。

【例題 6】

［ファイル名］

P227app.java

［プログラムの説明］

テキスト・P227～P228 a クラス、b クラス及び app クラスのプログラムを作成しなさい。ただし、クラス名"a"はクラス名"P227a"と、クラス名"b"はクラス名"P227b"と、またクラス名"app"はクラス名"P227app"とすること。

【考察】

・ **スーパークラス**の**デフォルトコンストラクタ**は、**サブクラス**から暗黙的に呼び出される。

【例題 7】

［ファイル名］

P229app.java

［プログラムの説明］

テキスト・P229 a クラス、b クラス及び app クラスのプログラムを作成しなさい。ただし、ク

ラス名"a"はクラス名"P229a"と、クラス名"b"はクラス名"P229b"と、またクラス名"app"はクラス名"P229app"とすること。

【考察】

・ **スーパークラス**の**コンストラクタ**を明示的に呼び出す場合は、super()メソッドを使用する。

【例題 8】

［ファイル名］

Samp8.java

［プログラムの説明］

TaxPayer クラスから NewTaxPayer クラスを作成し、給与所得者の所得税額を計算するプログラムである。ただし、このプログラムでは、スーパークラスである TaxPayer クラスで定義されているコンストラクタやメソッドを使用する必要がある。

プログラム中の空欄を埋めて、プログラムを完成させ実行しなさい。

［プログラム］

```
import  java.text.*;

//  給与所得者テーブル(スーパークラス)
class   TaxPayer    {
    int     eCode;                  //  管理番号
    String  eName;                  //  氏名
    int     eEarnings;              //  給与収入額
    int     eTaxable;               //  課税所得
    int     eTaxAmount;             //  所得税額

    //  給与所得者情報の設定(引数付きコンストラクタ)
    TaxPayer(_____①_____){
        eCode = code;
        eName = name;
        eEarnings = earnings;
        eTaxable = taxable;
    }

    //  所得税額の計算
    void     setTaxAmt(int taxable){
        if(taxable >= 18000000){
            eTaxAmount = (int)(taxable * 0.37 - 2490000);
        }else if(taxable >= 9000000){
            eTaxAmount = (int)(taxable * 0.30 - 1230000);
        }else if(taxable >= 3300000){
            eTaxAmount = (int)(taxable * 0.20 - 330000);
        }else{
```

```
            eTaxAmount = (int)(taxable * 0.10);
        }
    }

    //  給与収入額、課税所得額の表示(メソッド)
    void    printTaxPayer(){
         DecimalFormat   fmt = new DecimalFormat();
        fmt.applyPattern("#,##0");
        System.out.println("★★★" + eName + "★★★");
        System.out.println("      給与収入額は" + fmt.format(eEarnings) + "円");
        System.out.println("      課税所得額は" + fmt.format(eTaxable) + "円");
     }
}

//  TaxPayer クラスを継承した NewTaxPayer クラス(サブクラス)
class   NewTaxPayer extends   ②    {
    NewTaxPayer(int code, String name, int earnings, int taxable){
        ③ (code, name, earnings, taxable);   //  給与所得者の設定
        super.printTaxPayer();                //  給与収入額と課税所得額の表示
        this.setTaxAmt(taxable);              //  所得税額の計算と設定
    }

    //  所得税額の表示
    void    printTaxPayer(){
        DecimalFormat   fmt = new DecimalFormat();
        fmt.applyPattern("#,##0");
        System.out.println("      所得税額は" + fmt.format(eTaxAmount) + "円");
    }
}

public  class   Samp8   {
    public  static  void    main(String args[]){
        NewTaxPayer eData = new NewTaxPayer(1503, "遠山 絵梨香", 6000000, 4260000);
        eData.printTaxPayer();
    }
}
```

[出力例]

★★★遠山 絵梨香★★★

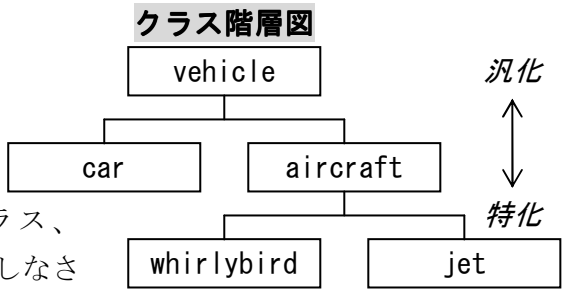
給与収入額は 6,000,000 円

課税所得額は 4,260,000 円

所得税額は 522,000 円

【考察】

【例題 9】



［ファイル名］

P232app.java, P221vehicle.java

［プログラムの説明］

テキスト・P230～232 app クラス、aircraft クラス、whirlybirdクラス及びjetクラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P232app"と、クラス名"aircraft"はクラス名"P231aircraft"と、クラス名"whirlybird"はクラス名"P231whirlybird"と、またクラス名"jet"はクラス名"P231jet"とすること。

なお、vehicleクラスとcarクラスは、テキスト・P221の"P221vehicle.java"を使用する。

【考察】

【例題 10】

［ファイル名］

P233app.java

［プログラムの説明］

テキスト・P233～P234 app クラスと各 a～d クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P233app"と、各クラス名"a"～"d"はクラス名"P233a"～"P233d"とすること。

【考察】

● 重 要 ●

Javaの継承におけるコンストラクタの呼び出し順序は、サブクラスの最上位階層のスーパークラスのコンストラクタから呼び出され、各クラスを初期化する。

【例題 11】

［ファイル名］

P235app.java

[プログラムの説明]

テキスト・P235～P236 app クラス、animal クラス及び fish クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P235app"と、クラス名"animal"はクラス名"P235animal"と、クラス名"fish"はクラス名"P235fish"とすること。

【考察】

● 重 要 ●

オーバーライド(再定義)とは、スーパークラスに定義されているインスタンスメソッドと同じシグネチャを持つインスタンスメソッドをサブクラスで定義することである。

メソッドの**隠蔽**(hide)と**オーバーライド**(over ride)

スーパークラス / サブクラス	クラスメソッド	インスタンスメソッド
クラスメソッド	隠蔽(hide)される。	コンパイルエラーとなる。
インスタンスメソッド	コンパイルエラーとなる。	オーバーライドされる

【例題 12】

[ファイル名]

P237app.java

[プログラムの説明]

テキスト・P237～P238 app クラス、animal クラス及び fish クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P237app"と、クラス名"animal"はクラス名"P237animal"と、クラス名"fish"はクラス名"P237fish"とすること。

【考察】

【例題 13】

[ファイル名]

Samp13.java

[プログラム]

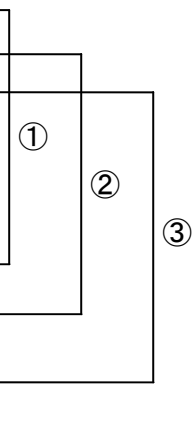
```
class   class1  {                       //  スーパークラスの定義
    public  void    override(){
        System.out.println("override: class1");
    }
```

```
    public  static  void    hide(){   //  クラスメソッドの定義
        System.out.println("hide : class1");      ←――――② 隠蔽
    }
    public  void    testSuper(){
        System.out.println("testSuper : class1");
    }
}

class   class2  extends class1  {     //  サブクラスの定義
    public  void    override(){
        System.out.println("override: class2");  ←――――① オーバーライド
    }
    public  static  void    hide(){
        System.out.println("hide : class2");
    }
    public  void    testSuper(){
        super.testSuper();                                  }←――――③ オーバーライド
        System.out.println("testSuper : class2");           }
    }
}

public  class   Samp13  {
    public  static  void    main(String args[]){
        class2  c2 = new class2();
        class1  c1 = c2;

        c1.override();
        c1.hide();
        c2.testSuper();
    }
}
```



[実行例]

```
override: class2
hide : class1
testSuper : class1
testSuper : class2
```

【考察】

【例題 14】

[ファイル名]

P240app.java

［プログラムの説明］

テキスト・P240～P241 app クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P240app"とすること。

また、クラス名"P240app"と、テキスト・P239～P240 のクラス名"aircraft"、クラス名"whirlybird"及びクラス名"jet"をファイル名"P240app.java"に保存すること。ただし、各クラス名の先頭には"P240"を付けること。

なお、vehicle クラスと car クラスは、テキスト・P221 の"P221vehicle.java"を使用する。

【解説】－ 代入互換性(assignment compatible)

代入互換性(assignment compatible)とは、左辺の変数の型に右辺の式の型が代入可能である性質を言う。

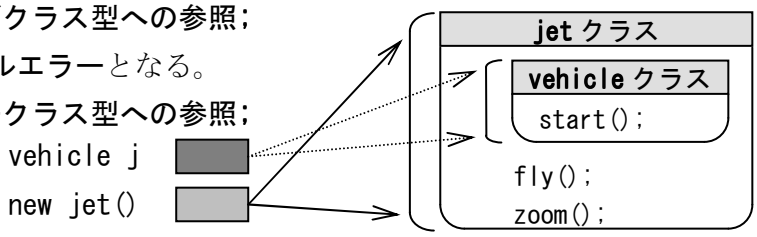
継承関係にある**スーパークラス**と**サブクラス**には、次のような**代入互換性**がある。

スーパークラス型の変数 = サブクラス型への参照;

なお、次のような代入文は**コンパイルエラー**となる。

サブクラス型の変数 = スーパークラス型への参照;

↑ これは、できない。



【考察】

・ 動作を確認後、P241 のコメントアウト部分をコメントを外して再コンパイルしなさい。

【例題 15】

［ファイル名］

P243app.java

［プログラムの説明］

テキスト・P242～P244 app クラスと各 a～d クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P243app"と、各クラス名"a"～"d"はクラス名"P242a"～"P242d"とすること。

【解説】－ 動的メソッドディスパッチ(dynamic method dispatching)

通常、呼び出すメソッドの型の解決はコンパイル時に行われるが、Java ではプログラムの実行中に特定のオブジェクト変数に格納するオブジェクトの型を指定できる。

この性質を**実行時ポリモーフィズム**(runtime polymorphism)と言い、この仕組みを**実行時バインディング**(late binding)と言う。

【考察】

抽象クラス(abstract class)

クラスの宣言時に、アクセス指定子 abstract を指定したクラスを**抽象クラス**(abstract class)と言う。

抽象クラスの性質は、次のとおりである。

(1) **抽象クラス**からは、直接オブジェクトを生成できない。
(2) **抽象クラス**は、**抽象メソッド**(abstract method)を持つことができる。
(3) **抽象クラス**は、**データメンバ/フィールド**、**コンストラクタ**及び**非抽象メソッド**を持つことができる。
(4) **抽象クラス**に、アクセス指定子 final を同時に指定できない。

抽象メソッドとは、**戻り値**と**シグネチャ**の宣言のみを行ない、メソッド本体の実装を定義しないメソッドである。**抽象メソッド**の性質は、次のとおりである。 ⇒ 純粋仮想関数

(1) **抽象メソッド**の宣言は、**抽象クラス**内でのみ記述できる。
(2) **抽象メソッド**の本体の実装は、**抽象クラス**を**継承**した**サブクラス**中で定義する。
(3) メソッドの宣言時に、メソッド本体の実装を定義しない場合、暗黙的に**抽象メソッド**と見なされる。
(4) **抽象メソッド**に、アクセス指定子 static と final を同時に指定できない。

【例題 16】

[ファイル名]

P246app.java

[プログラムの説明]

テキスト・P245～P246 app クラス、a クラス及び b クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P246app"と、クラス名"a"はクラス名"P245a"と、クラス名"b"はクラス名"P246b"とすること。

【考察】

- 動作を確認後、P246app クラスの"P246b　b1 = new P246b;"の直下に、"P245a　　a1 = new P245a();"を追加コーディングし、再コンパイルしなさい。
- 動作を確認後、P246b クラスの getData()メソッドの本体定義の直下に、"abstract String　setData();"を追加コーディングし、再コンパイルしなさい。

.....

.....

.....

.....

final **クラス**と final **メソッド**

クラスの宣言時に、アクセス指定子 final を指定したクラスを final **クラス**(final class)と言う。final **クラス**は、**継承**を許可しない。

メソッドの宣言時に、アクセス指定子 final を指定したメソッドを final **メソッド**(final method)と言う。final **メソッド**は、**オーバーライド**と**隠蔽**を許可しない。

【例題 17】

［ファイル名］

P248app.java

［プログラムの説明］

テキスト・P247～P248 app クラス、animal クラス及び fish クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P248app"と、クラス名"animal"はクラス名"P247animal"と、クラス名"fish"はクラス名"P247fish"とすること。

【考察】

【例題 18】

［ファイル名］

P249app.java

［プログラムの説明］

テキスト・P248～P249 app クラス、animal クラス及び fish クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P249app"と、クラス名"animal"はクラス名"P248animal"と、クラス名"fish"はクラス名"P249fish"とすること。

【考察】

【例題 19】

［ファイル名］

P250app.java

［プログラムの説明］

テキスト・P250 app クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P250app"とすること。

【解説】－ 名前付き定数(final field)

Java では、データメンバ/フィールドの宣言時にアクセス指定子 final を付けると、**名前付き定数**(final field)として扱われる。

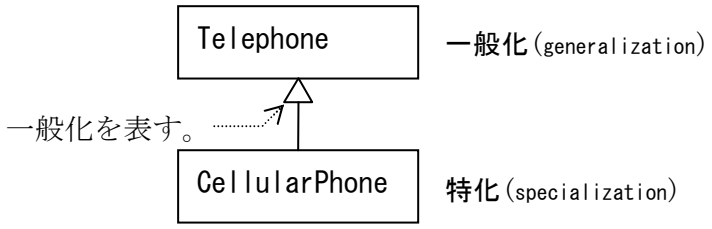
【考察】

is-a 関係と has-a 関係

「**is-a の関係**」は、"**オブジェクト A はオブジェクト B である**"といった be 動詞的な関係で、別のクラスの性質（属性、操作、関係）を引き継ぐ形で再利用する方法である。⇒ **継承**

このとき、"**オブジェクト B**"を**一般化**された側と言い、共通的な性質に着目して抽象化したクラスである。⇒ **スーパークラス**(super class)、**基本クラス**(base class)

また、"**オブジェクト A**"を**特化**された側と言い、**基本クラス**の性質として共通化できなかった性質(差分)だけを定義したクラスである。⇒ **サブクラス**(sub class)、**派生クラス**(derived class)



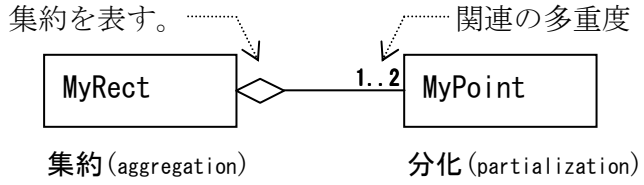
"CellularPhone クラスは、Telephone クラスを継承する"

『"has-a の関係"は、クラス内に同じ特性のものが複数存在する場合があるが、"is-a の関係"では、クラス内に同じ特性のものが複数存在し得ない。』

また、OOP(Object Oriented Programming)における再利用の実装に一つに、「has-a **の関係**」の再利用がある。⇒ part-of 関係

「has-a **の関係**」は、"**オブジェクト A はオブジェクト B を持つ**"といった have 動詞的な関係で、別の定義済みクラス型のインスタンスをクラスのデータメンバとして再利用する方法である。

このとき、"**オブジェクト A**"を**集約**された側と言い、全体を意味する。また、"**オブジェクト B**"を**分化**された側と言い、全体を構成する要素を意味する。⇒ 集約と分化の関係、部品性



"MyRect クラスは、MyPoint クラスを持つ"

【例題 20】

[ファイル名]

P250app.java

[プログラムの説明]

テキスト・P250～P251 app クラス、a クラス及び b クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P251app"と、クラス名"a"はクラス名"P250a"と、クラス名"b"はクラス名"P251b"とすること。

【考察】

【例題 21】

［ファイル名］

P252app.java

［プログラムの説明］

テキスト・P251～P252 app クラス、a クラス及び b クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P252app"と、クラス名"a"はクラス名"P251a"と、クラス名"b"はクラス名"P252b"とすること。

【考察】

【例題 22】

［ファイル名］

P252app.java

［プログラムの説明］

テキスト・P253～P255 app クラス、a クラス、b クラス及び c クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P254app"と、クラス名"a"はクラス名"P253a"と、クラス名"b"はクラス名"P253b"と、クラス名"c"はクラス名"P254c"とすること。

【解説】

"java.lang.object"は、Java の中で「最も汎用的なクラス」であり、Java の階層の最上位に位置する。すなわち、Java ではすべてのクラスは、このクラス"Object"から自動的に派生されていることになる。

【考察】

2 インタフェース(interface)

インタフェースは、定数と**抽象メソッド**のみを宣言したクラスの仕様である。

【構文】

```
access  interface  interface-name  extends  super-interface  {
    interface-body;
}
```

access　… アクセス指定子を指定する。指定できるアクセス指定子は次のとおり。

public	外部からアクセスが可能。
省略	同じパッケージのクラスからアクセスが可能。

interface-name … このインタフェースの**インタフェース**名を指定する。

super-interface … *super-interface*名を指定する。複数の**スーパーインタフェース**を指定する場合は、","で区切って指定する。

解説：*interface-body*に記述できるものは、**抽象メソッド**と**定数**に限られる。

(1) **抽象メソッド**

この**抽象メソッド**は、暗黙的にアクセス指定子"public abstract"が指定されたものとして扱われる。

(2) **定数(名前付き定数)**

この定数は、暗黙的にアクセス指定子"public static final"が指定されたものとして扱われる。

インタフェースは、実体を持たないので直接オブジェクトを生成することはできない。すなわち、**インタフェース**の**抽象メソッド**を実装したクラスを作成し、そのクラスからオブジェクトを生成することになる。

抽象クラスと**インタフェース**の比較

	抽象クラス	**インタフェース**	備考
インスタンス変数	○	×	フィールド
クラス変数	○	×	フィールド
定数	○	○	名前付き定数
コンストラクタ	○	×	
抽象メソッド	○	○	
非抽象メソッド	○	×	

インタフェースは、C++で許可していた**多重継承**の機能を、Java でも同様に提供するための手段の一つである。

● 重 要 ●

Java では、サブクラスの場合は一つのスーパークラスだけを直接拡張するが、インタフェースにはこの制約がなく複数のスーパーインタフェースを直接拡張できる。

【例題 23】

[ファイル名]

P257clicker.java, P257clicker.html

[プログラムの説明]

テキスト・P257 clicker クラスのプログラムと、次に示す HTML プログラムを作成しなさい。ただし、クラス名"clicker"はクラス名"P257clicker"とすること。

[HTML プログラム]

```
<HTML>
<!-- Java applet (P257) -->
<HEAD>
<TITLE>APPLET</TITLE>
<BODY>
<HR>
```

```
<CENTER>
<APPLET CODE=P257clicker.class WIDTH=200 HEIGHT=200>
</APPLET>
</CENTER>
<HR>
</BODY>
</HTML>
```

【考察】

【例題 24】

［ファイル名］

Samp24.java

［プログラム］

```
// インタフェースの定義例
interface   MyFirstInterface    {
    static  final   int CONST_BIAS = 35;    //  名前付き定数の定義
    abstract    void    putPoint(int point);//  抽象メソッドの定義
}

//  FrivolousClass での MyFirstInterface の実装例
class   FrivolousClass  implements  MyFirstInterface    {
    public  void    putPoint(int point){
        int temp = point + CONST_BIAS;

        if(temp > 100)
            temp = 100;
        System.out.println(temp);
    }
}

//  ExcellenceClass での MyFirstInterface の実装例
class   ExcellenceClass implements  MyFirstInterface    {
    public  void    putPoint(int point){
        System.out.print(point);
        if(point < 40)
            System.out.println("赤点");
        else
            System.out.println();
    }
}

public  class   Samp24  {
    public  static  void    main(String args[]){
```

実装

実装

```
        FrivolousClass  fc = new FrivolousClass();

        System.out.println("★★ FrivolousClass ★★");
        fc.putPoint(72);
        fc.putPoint(85);
        fc.putPoint(35);

        ExcellenceClass ec = new ExcellenceClass();

        System.out.println("☆☆ ExcellenceClass ☆☆");
        ec.putPoint(72);
        ec.putPoint(85);
        ec.putPoint(35);
    }
}
```

【考察】

3 内部クラス(inner class)

Java では、式またはクラス内にクラスを定義できる。このクラス内に定義されたクラスを**内部クラス**(inner class)と言う。

内部クラスは独立したクラスではあるが、組み込まれているクラスの一要素でもあることから、このクラスを含む外部のクラスのデータメンバ/フィールドやメソッドにアクセスできる。

また、**内部クラス**はコンパイルされると、「**"外部のクラス名"$"内部クラス名"**.class」と言う名前のファイルを作成する。

【例題 25】

[ファイル名]

P259app.java

[プログラムの説明]

テキスト・P259 app クラス、a クラス及び b クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P259app"と、クラス名"a"はクラス名"P259a"と、クラス名"b"はクラス名"P259b"とすること。

【考察】

【例題 26】

［ファイル名］

MyFrame.java

［プログラム］

```
import  java.lang.*;
import  java.awt.*;
import  java.awt.event.*;

public  class   MyFrame extends Frame   {
    Label       lbl1;
    MyButton    btn1;

    //  Java のエントリポイント
    public  static  void    main(String args[]){
        MyFrame mf = new MyFrame();
        mf.setSize(300, 200);
        mf.show();
    }

    public  MyFrame(){                      //  デフォルトコンストラクタ
        super();
        this.setLayout(null);
        lbl1 = new Label("Hello from Java!");
        lbl1.setBounds(20, 20, 250, 30);
        btn1 = new MyButton("CLICK");
        btn1.setBounds(100, 100, 100, 30);
        this.add(lbl1);                     //  フォームにラベルを追加
        this.add(btn1);                     //  フォームにボタンを追加

        addWindowListener(new WindowAdapter(){
            public  void    windowClosing(WindowEvent e){
①  ―→          System.exit(0);
            }
        });
    }

    //  内部クラスの定義
②  ―→class   MyButton    extends Button  {
        public  MyButton(String s){         //  引数付きコンストラクタ
            super(s);
        }

        public  boolean action(Event e, Object o){
            lbl1.setText("You are clicked.");
            return true;
        }
    }
}
```

【考察】

匿名内部クラス(anonymous inner class)

Java では、式の中に名前を持たない**内部クラス**の定義ができる。

匿名内部クラスはコンパイルされると、「**"外部のクラス名"$"番号"**.class」と言う名前のファイルを作成する。

【構文】－ 匿名内部クラス

```
new  super-class(argument-list){
        field-declaration;
        method-declaration;
}
```

super-class … このクラスの**スーパークラス**を指定する。

argument-list … **コンストラクタ**に引き渡す実引数を指定する。

field-declaration … この**内部クラス**のデータメンバ/フィールドの定義を記述する。

method-declaration … この**内部クラス**のメソッドの定義を記述する。

解説：式の中で、new 演算子によってオブジェクトを生成すると同時に、**匿名内部クラス**を定義する。

【例題 27】

［ファイル名］

P260app.java

［プログラムの説明］

テキスト・P260 app クラス、a クラス及び b クラスのプログラムを作成しなさい。ただし、クラス名"app"はクラス名"P260app"と、クラス名"a"はクラス名"P260a"と、クラス名"b"はクラス名"P260b"とすること。

【考察】

【例題 28】

［ファイル名］

MyFrame2.java

［プログラム］

```
import  java.lang.*;
import  java.awt.*;
import  java.awt.event.*;

public  class   MyFrame2    extends Frame   {
    Label   lbl1;
    Button  btn1;

    //  Java のエントリポイント
    public  static  void    main(String args[]){
        MyFrame2    mf = new MyFrame2();
        mf.setSize(300, 200);
        mf.show();
    }

    public  MyFrame2(){                     //  デフォルトコンストラクタ
        super();
        this.setLayout(null);
        lbl1 = new Label("Hello from Java!");
        lbl1.setBounds(20, 20, 250, 30);
        btn1 = new Button("CLICK");
        btn1.setBounds(100, 100, 100, 30);
        this.add(lbl1);
        this.add(btn1);

        btn1.addActionListener(new ActionListener(){
            public  void    actionPerformed(ActionEvent e){
                lbl1.setText("You are clicked.");
            }
        });

        addWindowListener(new WindowAdapter(){
            public  void    windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
    }
}
```

【考察】