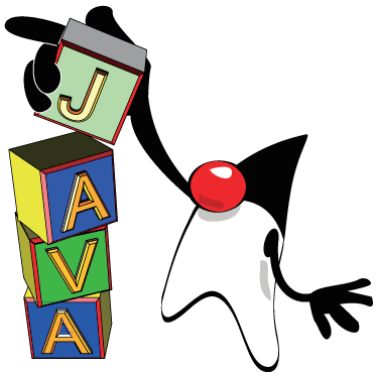
令和 6 年度文部科学省委託「専修学校による地域産業中核的人材養成」事業

情報 DX エンジニア教材資料

Java プログラミング

IT 分野 DX 人材養成のモデルプログラム開発と実証事業

Java
プログラミング



●商標

・Java は、Oracle Corporation およびその子会社、関連会社の米国及びその他の国における登録商標です。

・Windows は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。

・本書では、®、©、TM などのマークは省略しています。

目次

はじめに Java の概要 …… 6

① Java の特徴 …… 6
② Java の開発環境構築 …… 7
③ Java の開発手順 …… 9

第1部 はじめての Java プログラミング

第1章 Java プログラムの基本 …… 20

1―1―1 プログラムの基本形と標準出力 …… 20
1―1―2 変数 …… 26

第2章 演算子 …… 34

1―2―1 様々な演算子 …… 34
1―2―2 入力の読込みと例外処理 …… 40

第3章 配列 …… 48

1―3―1 一次元配列 …… 48
1―3―2 二次元配列 …… 56

第4章 制御構造（選択） …… 64

1―4―1 選択構造① …… 64
1―4―2 選択構造② …… 72

第5章 制御構造（反復） …… 80

1―5―1 反復構造① …… 80
1―5―2 反復構造② …… 90

第6章 メソッド …… 98

1―6―1 メソッドの基本 …… 98
1―6―2 メソッドのオーバロード …… 106

第2部 オブジェクト指向プログラミング①

第1章 オブジェクト指向とクラス定義 …… 114

2—1—1 オブジェクト指向とは? …… 114
2—1—2 クラス定義 …… 120

第2章 オブジェクトの生成とインスタンスの利用 …… 126

2—2—1 オブジェクトの生成 …… 126
2—2—2 インスタンスの利用 …… 132

第3章 コンストラクタ …… 138

2—3—1 コンストラクタの利用 …… 138
2—3—2 this キーワード …… 146

第4章 アクセスレベルとパッケージング …… 152

2—4—1 アクセス修飾子 …… 152
2—4—2 パッケージング …… 160

第5章 参照型の一次元配列と ArrayList クラス …… 166

2—5—1 参照型の一次元配列 …… 166
2—5—2 ArrayList クラス …… 174

第3部 オブジェクト指向プログラミング②

第1章 クラスの拡張 …… 184

3−1−1 継承 …… 184

3−1−2 サブクラスの定義とインスタンス化 …… 190

第2章 オーバライド …… 198

3−2−1 メソッドの再定義 …… 198

3−2−2 super( ) …… 206

第3章 インタフェース …… 212

3−3−1 インタフェースとは? …… 212

3−3−2 インタフェースの拡張 …… 220

第4章 ポリモフィズム …… 226

3−4−1 ポリモフィズムとは? …… 226

3−4−2 ポリモフィズムの実現 …… 230

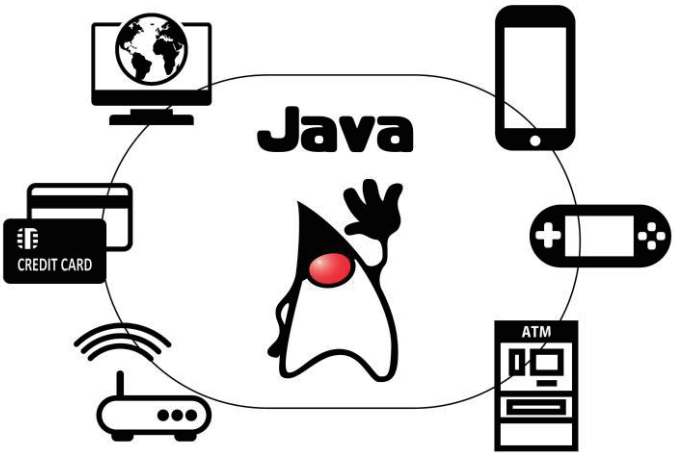
索引 …… 242

はじめに　Javaの概要

Javaのマスコット“デューク”です！
まずは，Javaの魅力を紹介します！

1　Java の特徴

Java は，プログラミング言語の一つです。スマートフォン用アプリ，ゲーム，ATM，ネットワーク機器，IC カード，Web アプリなど，様々なソフトウェアを開発することができます。



(1)　だれでも自由に無料で使える

Java は，だれでも自由に無料で使うことができます。また，プログラムを作るための道具は，インターネットからダウンロードして無料で利用することができます。

(2)　Java プログラムはどこでも動く

Java で作成したプログラムは，PC，タブレット，スマートフォンなどで動かすことができます。

(3)　オブジェクト指向に対応

Java は，現実の世界と同じように部品の組合せでソフトウェアを作る考え方（オブジェクト指向）に対応しています。

2 Java の開発環境構築

(1) Eclipse とは

Eclipse は，Eclipse プロジェクトが無償で配布している統合開発環境です。Java の開発では事実上の標準とされています。

(2) Eclipse のインストール

Windows を使用している PC に，Eclipse と Eclipse 日本語化プラグイン Pleiades（MergeDoc プロジェクトが無償で配布）がセットになった「Pleiades All in One」をインストールします。

・記載している内容は，2020 年 3 月 1 日時点の情報です。

● Eclipse のダウンロード

MergeDoc プロジェクトのダウンロードサイトからファイルをダウンロードします。

① Web ブラウザソフトを起動し，次の URL にアクセスします。

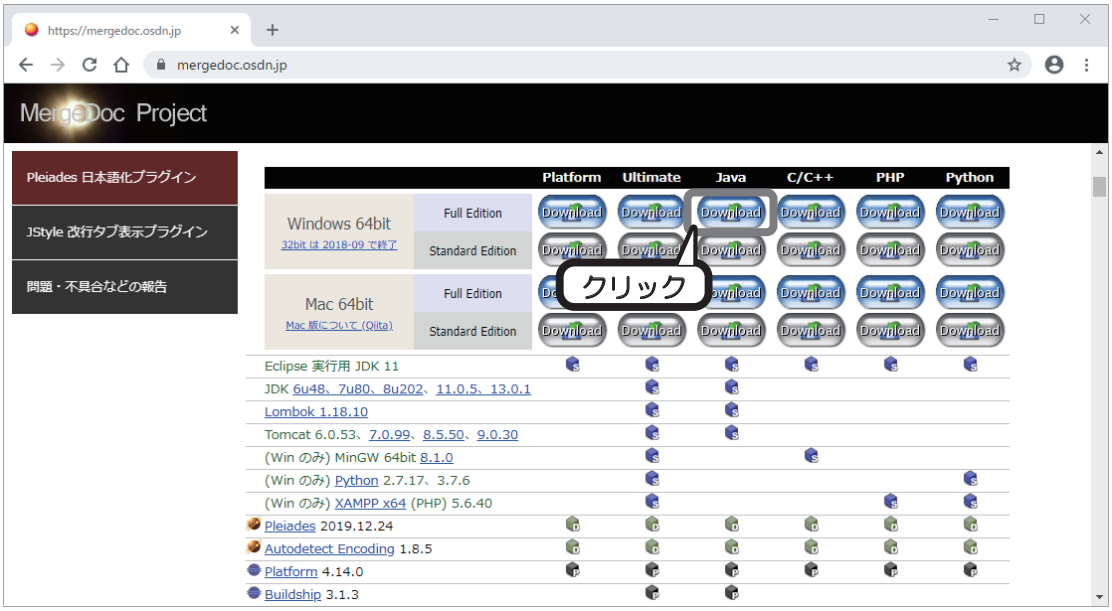
・URL

http://mergedoc.osdn.jp/

② 「Pleiades All in One ダウンロード」の「Eclipse 2019-12」をクリックします。

・32 ビット OS の場合は，「Eclipse 2018-09」をクリックします。

③　「Pleiades All in One Eclipse ダウンロード」の中の「Java Windows 64bit Full Edition」をクリックします。



④　クリック後は，画面の指示に従ってファイルをダウンロードします。

● Eclipse のインストール

①　「pleiades-2019-12-java-win-64bit-jre_XXXXXXXX.zip」を右クリックし，「プログラムから開く」の「エクスプローラー」を選択します。

②　「pleiades」フォルダをデスクトップにコピーします。

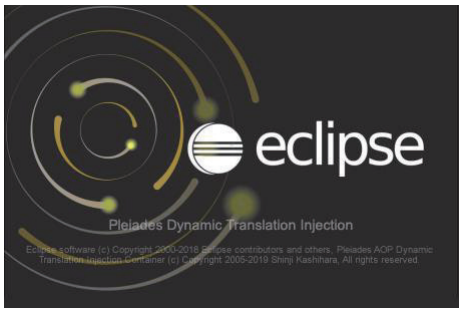
⚠
- コピーが完了するまでは，コピー元の「pleiades」フォルダが格納されているフォルダは閉じないでください。
- 「ユーザー アカウント制御」が表示される場合は「はい」ボタンをクリックしてください。

③　デスクトップの「pleiades」フォルダを「C:¥Program Files」に移動します。

3 Java の開発手順

(1) Eclipse の起動

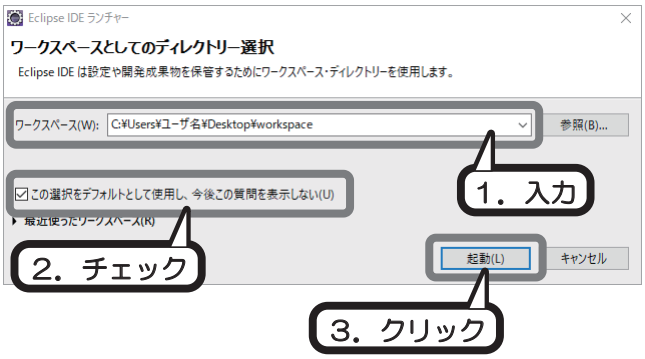
「C:¥Program Files¥pleiades¥eclipse」フォルダの「eclipse.exe」をダブルクリックして Eclipse を起動します。



● ワークスペースの選択

① 「ワークスペース(W)」に適切なパスを入力します。

② 「この選択をデフォルトとして使用し，今後この質問を表示しない(U)」にチェックを入れて，「起動(L)」ボタンをクリックします。



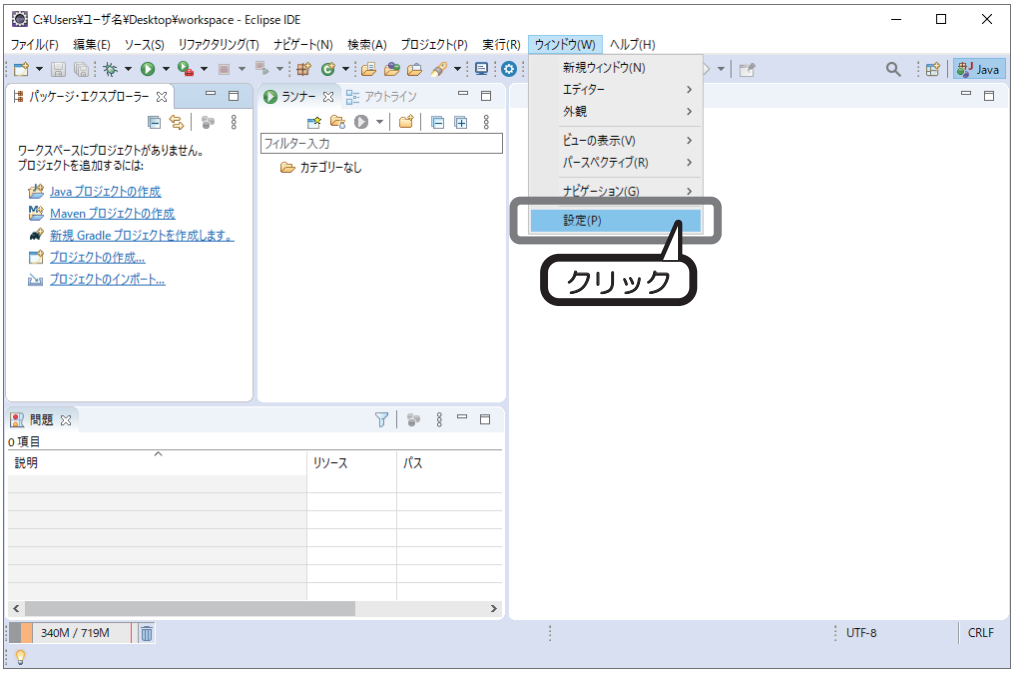
・ワークスペース

C:¥Users¥ユーザ名¥Desktop¥workspace

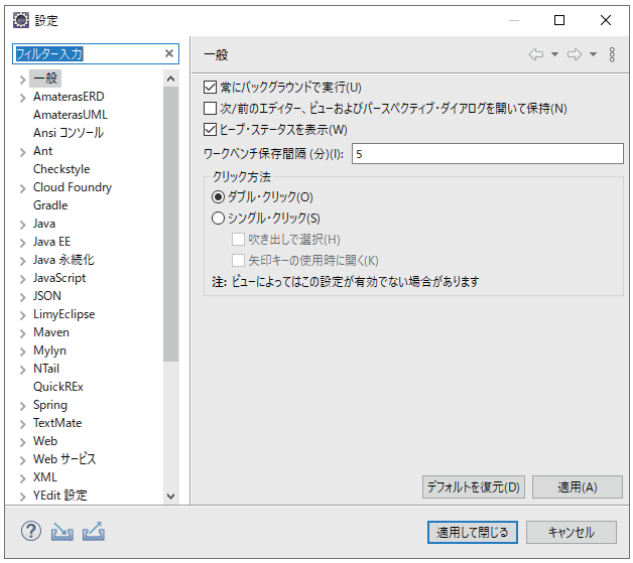
(2)　Eclipse の環境設定

●　ワークスペースの設定

①　「ウィンドウ(W)」メニューの「設定(P)」をクリックします。

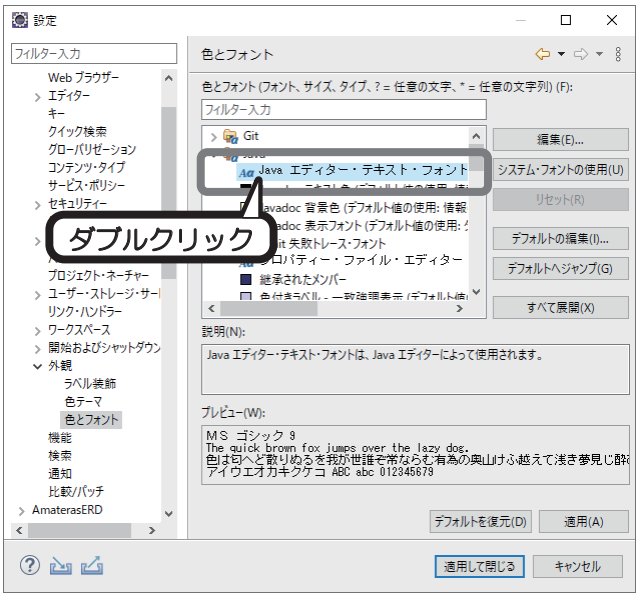


「設定」ウィンドウが表示されます。



② 「一般」をダブルクリックして開き，「外観」をダブルクリックして「色とフォント」をクリックします。

③ 「Java」をダブルクリックして「Java エディター・テキスト・フォント（デフォルト値の使用：テキストフォント）」をダブルクリックします。



「フォント」ウィンドウが表示されるので，フォント名，スタイル，サイズ名を変更します。

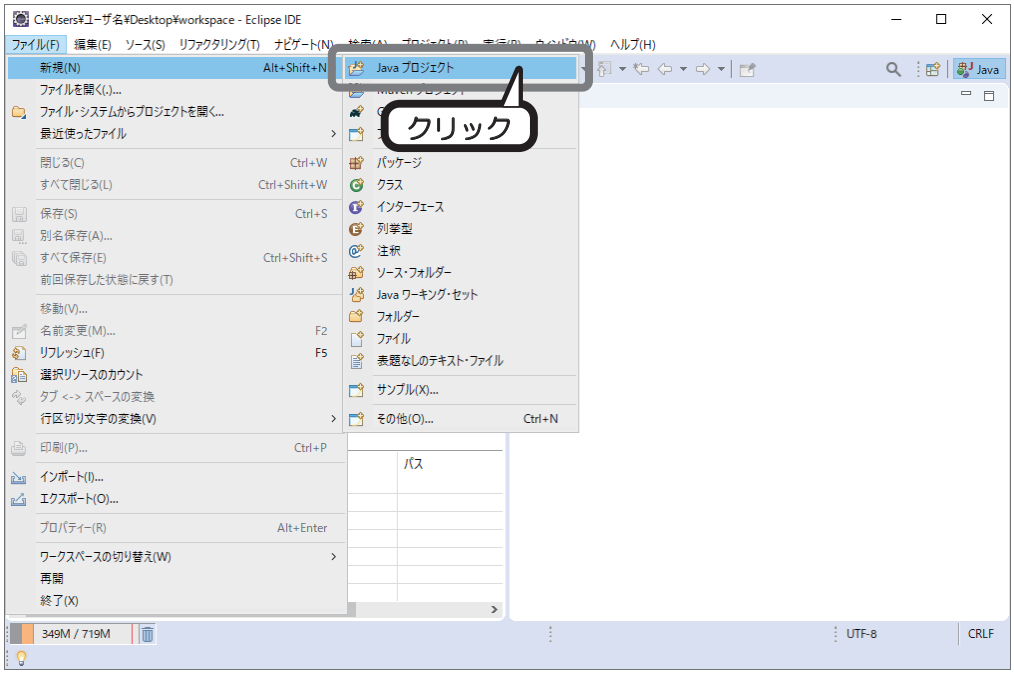
④ 「デバッグ」をダブルクリックして「コンソール・フォント（デフォルト値の使用：テキストフォント）」をダブルクリックします。

「フォント」ウィンドウが表示されるので，フォント名，スタイル，サイズ名を変更します。

(3) Java プロジェクトの作成

● 新規 Java プロジェクトの作成

① 「ファイル(F)」メニューの「新規(N)」をポイントして，「Java プロジェクト」をクリックします。



⚠

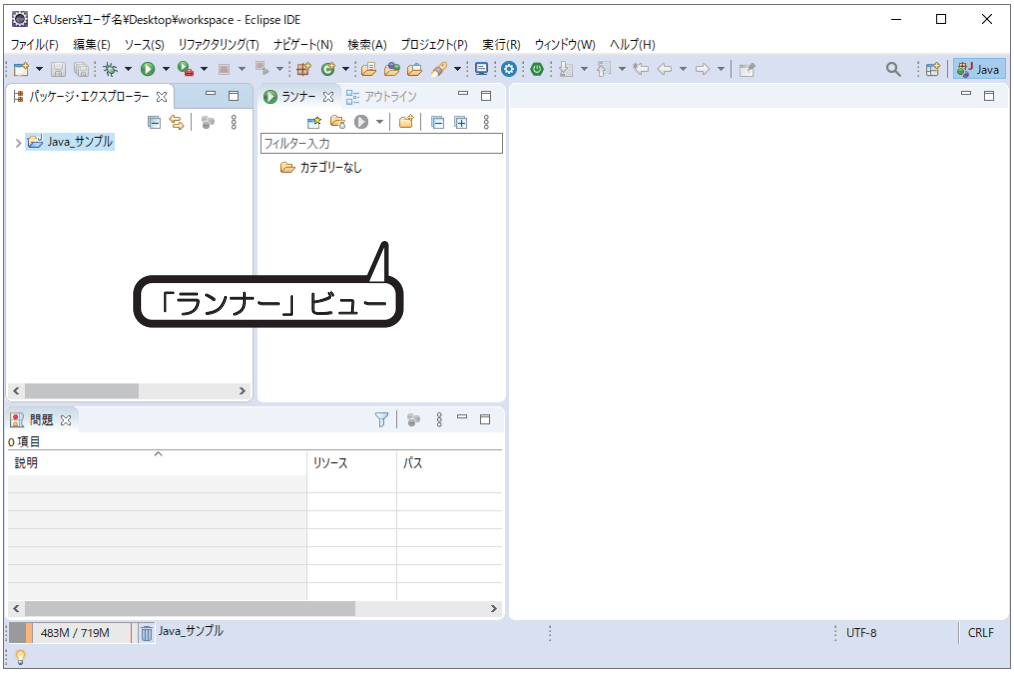
・「Java プロジェクト」がない場合は，次の手順で「Java プロジェクト」を選択します。

①「ファイル(F)」メニューの「新規(N)」をポイントして，
「プロジェクト(R)...」をクリックする

②「Java」をダブルクリックし，「Java プロジェクト」を選択して
「次へ(N)>」ボタンをクリックする

② 「プロジェクト名(P)」に適切なプロジェクト名を入力して，「完了(F)」ボタンをクリックします。

※「新規 module-info.java」ウィンドウが表示されますが，「作成しない(D)」をクリックします。

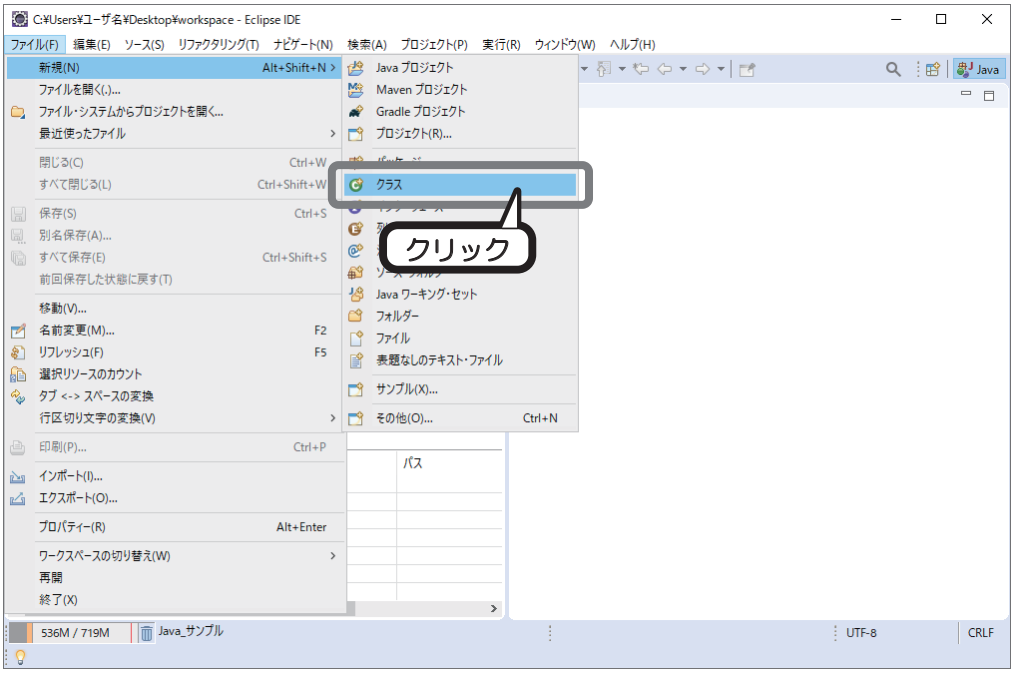


※「ランナー」ビュー，「アウトライン」ビューは，「×」ボタンをクリックして閉じてください。

(4) 新規 Java クラスの作成と Java プログラムの実行

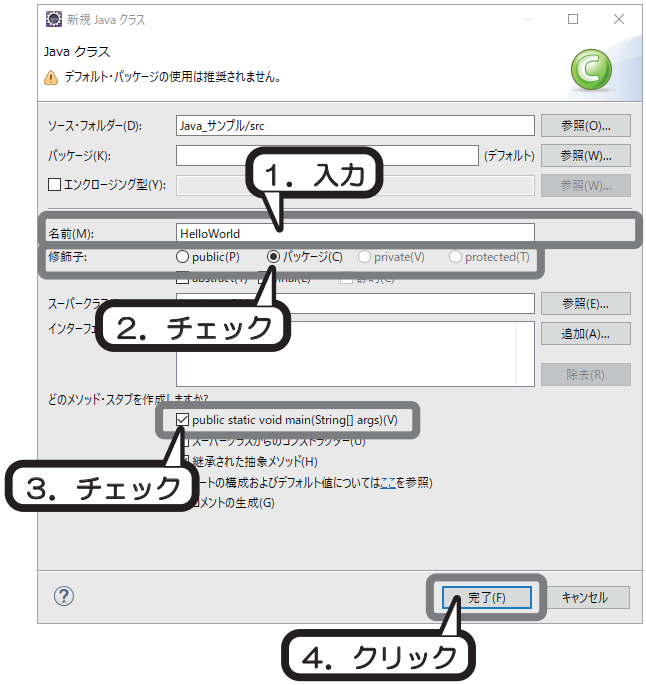
● 新規 Java クラスの作成

① 「ファイル(F)」メニューの「新規(N)」をポイントして，「クラス」をクリックします。



② 「名前(M)」に適切なクラス名を入力し，「修飾子」にチェックを入れて，適切な修飾子を指定します。

③ 「public static void main(String[] args)(V)」にチェックを入れて，「完了(F)」ボタンをクリックします。



④ 「エディター」ビューに表示されたクラス内にプログラムを追加します。

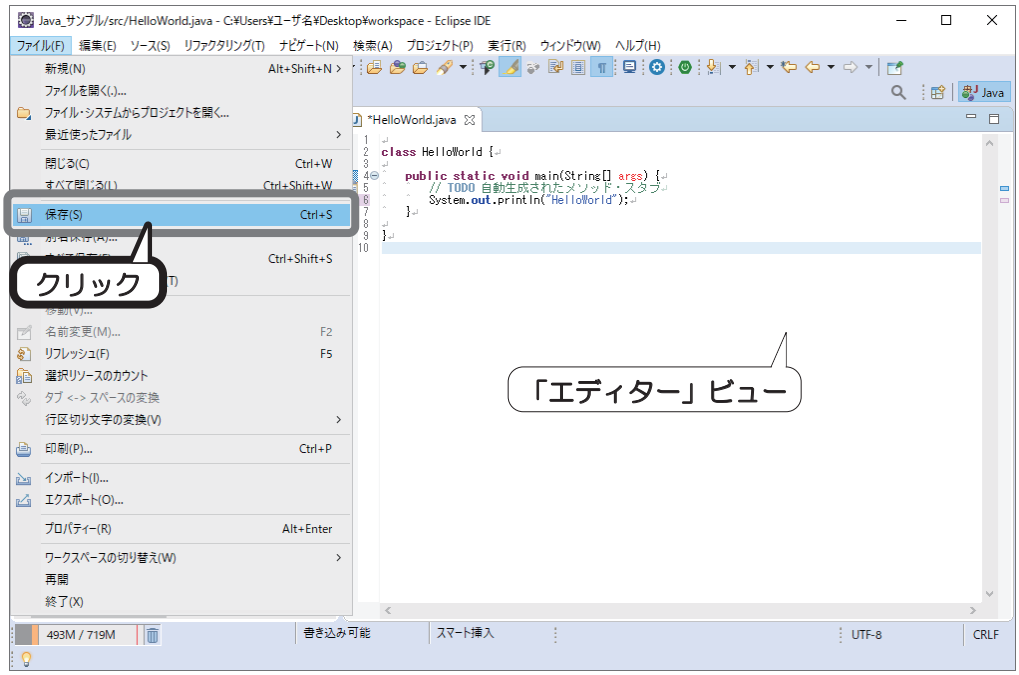
[サンプルプログラム]

HelloWorld.java

```
1
2  class HelloWorld {
3
4      public static void main(String[] args) {
5          // TODO 自動生成されたメソッド・スタブ
6          System.out.println("HelloWorld");
7      }
8
9  }
```

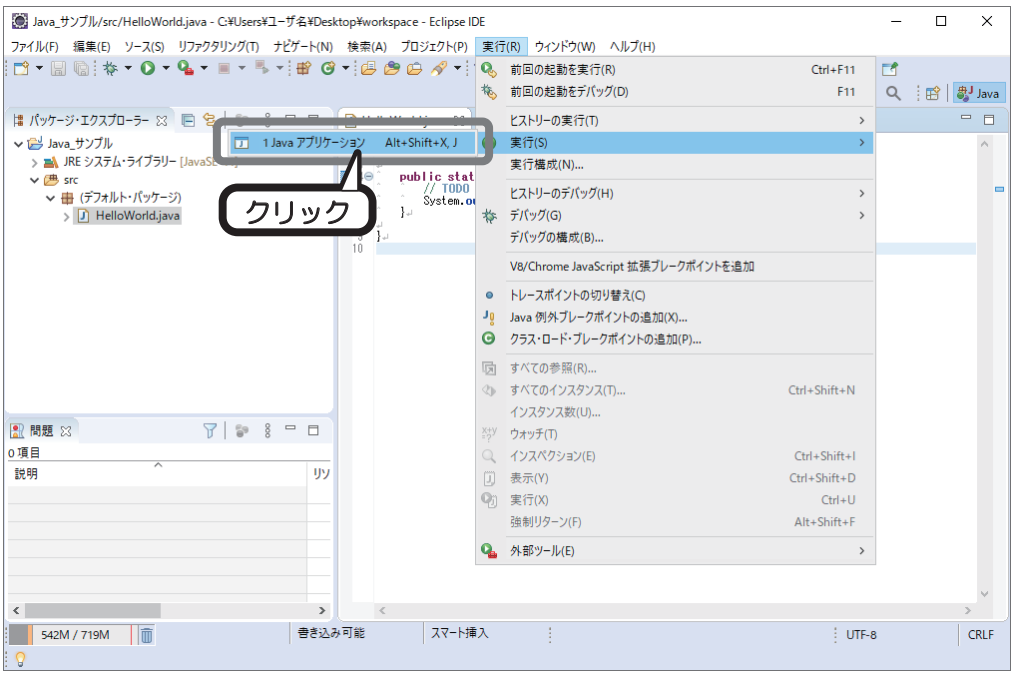
このプログラムを追加する

⑤　「ファイル(F)」メニューの「保存(S)」をクリックします。

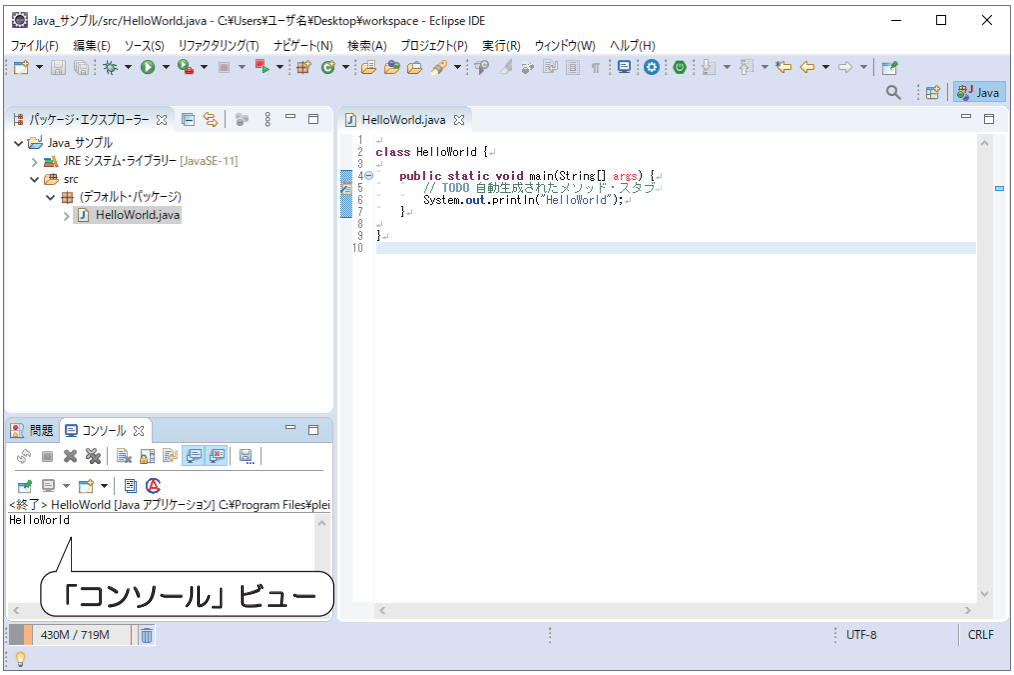


● Java プログラムの実行

①　「エディター」ビューをクリックし，「実行(R)」メニューの「実行(S)」をポイントして，「Java アプリケーション」をクリックします。



② 実行結果が「コンソール」ビューに表示されます。



・２回目以降は， ボタンを押して実行することができます。

⑸ その他

● Java プロジェクトへの作成済プログラムの追加

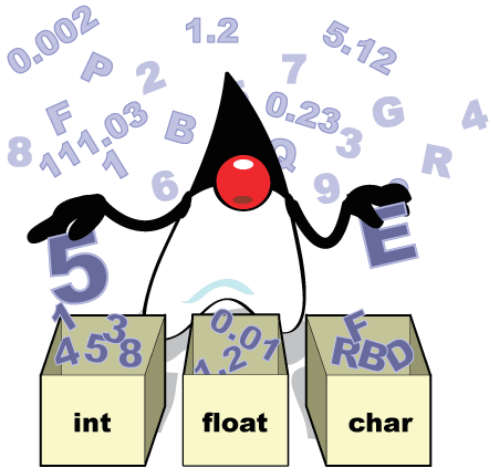
作成済のプログラムをプロジェクトに追加する場合は，拡張子「.java」のファイルをコピーして，Java プロジェクト内の「src」階層に貼り付けます。

プログラムがパッケージングされている場合は，パッケージのフォルダごと「src」階層に貼り付けます。

※パッケージングについては，第２部・第４章で解説します。

はじめての Javaプログラミング

第1部



第1章 Javaプログラムの基本

デュークのアドバイス

はじめてみれば超簡単！
今日から君もJavaプログラマー！

1-1-1 プログラムの基本形と標準出力

例題をとおして，Java プログラムの基本形について学習します。また，ディスプレイに出力する場合の標準出力についても学習します。

例題1-1-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
今日からスタート！
Javaプログラミング
```

クラス定義

クラス名　：　Example111

メソッド名　：　main メソッド

処理内容

ディスプレイに，メッセージ“今日からスタート！”を表示します。
ディスプレイに，メッセージ“Java プログラミング”を表示します。

[プログラム例 1-1-1]

Example111.java

```
1  class Example111 {
2      public static void main(String[] args) {
3          System.out.println("今日からスタート！");
4          System.out.println("Java プログラミング");
5      }
6  }
```

実行結果

```
今日からスタート！
Javaプログラミング
```

[作業手順]

① Eclipse の「Java_第 1 部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第 1 部/src
名前	Example111
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 1-1-1 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

・保存後，Eclipse のプログラムに×印があった場合は，文法エラー（コンパイルエラー）です。エラーを修正してから，再度プログラムを保存してください。

③ Eclipse の ボタンを押して，プログラムを実行します。

例題1-1-1解説

(1) プログラムの基本形

● クラス定義

Java では，**クラス**という形でプログラムを記述します。

記述方法（クラス定義）

```
class クラス名 {
    :
}
```

class に続けて，任意のクラス名を記述します。

・クラス名には，英数字，「_（アンダースコア）」，「$（ドル記号）」を使うことができます。ただし，クラス名の先頭文字に数字を使うことはできません。
・クラス名が英字の場合，一般的に先頭文字は大文字にします。
・「{」と「}」の間に記述された範囲を，**ブロック**と呼びます。

・「class」，クラス名，スペース，「{」，「}」は，半角で入力してください。入力しない場合は，コンパイルエラーになります。

クラスの範囲は，「{」から「}」までの記述全体で，「{」と「}」も含まれます。プログラム例 1-1-1 では，クラスの範囲は次に示す１～６行目です。

Example111.java（抜粋）

```
1  class Example111 {
:      :
6  }
```

クラスのブロック（「{」と「}」の間）に，メソッドや変数を記述します。

● main メソッド

main メソッドは，Java プログラムで最初に実行されるメソッドです。

記述方法（main メソッド）

```
class クラス名 {
    public static void main(String[] args) {
        具体的な処理内容
    }
}
```

main メソッドは，クラスのブロック内に記述します。

・メソッドとは，処理のまとまりのことです。
※メソッドの詳細については，第１部・第６章で解説します。

・スペース，「(」，「)」，「[」，「]」，「{」，「}」は，半角で入力してください。半角で入力しない場合は，コンパイルエラーになります。
・大文字，小文字の区別があるので，間違えるとコンパイルエラーになる場合や，実行できない場合があります。

main メソッドの範囲は，クラスのブロック内の「{」から「}」までの記述全体で，「{」と「}」も含まれます。
プログラム例 1-1-1 では，main メソッドの範囲は次に示す２～５行目です。

Example111.java（抜粋）

```
1  class Example111 {
2      public static void main(String[] args) {
:          :
5      }
6  }
```

メソッドの範囲内の「{」と「}」の間に，具体的な処理内容を記述します。

(2) 標準出力

Eclipse のコンソールに出力することを，**標準出力**と言います。Java プログラムでは，main メソッドの{ }内に標準出力の処理を記述します。

● １行出力後，改行する場合

「System.out.println()」を使用して出力します。

記述方法（１行出力後，改行する場合）

```
System.out.println(出力内容);
```

記述例（１行出力後，改行する場合）

```
System.out.println("文字列");
System.out.println(100);
System.out.println(3.14);
```

実行結果

```
文字列
100
3.14
```

● １行出力後，改行しない場合

「System.out.print()」を使用して出力します。

記述方法（１行出力後，改行しない場合）

```
System.out.print(出力内容);
```

記述例（１行出力後，改行しない場合）

```
System.out.print("文字列");
System.out.print(100);
System.out.print(3.14);
```

実行結果

```
文字列1003.14
```

・具体的な処理内容は，一行ごとに「;（セミコロン）」で閉じます。
・文字列は，「″（ダブルクォーテーション）」と「″」の間に記述します。

・「.（ドット）」，「(」，「)」，「″」，「’（シングルクォーテーション）」，「;」は，半角で入力してください。
・大文字，小文字の区別があるので，間違えるとコンパイルエラーになります。

練習問題1-1-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
身長：170.5cm
体重：60.5kg
年齢：21歳
趣味：スポーツ観戦
```

クラス定義

クラス名　　：　Practice111

メソッド名　：　main メソッド

処理内容

ディスプレイに，身長，体重，年齢，趣味を表示します。

main メソッドの処理条件

1．文字列「身長：170.5cm」をディスプレイに表示します。
2．文字列「体重：60.5kg」をディスプレイに表示します。
3．文字列「年齢：21 歳」をディスプレイに表示します。
4．文字列「趣味：スポーツ観戦」をディスプレイに表示します。

1-1-2 変数

例題をとおして，変数について学習します。また，注釈（コメント）についても学習します。

例題1-1-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
1月は，英語で January です。
```

クラス定義

クラス名　：　Example112

メソッド名　：　main メソッド

処理内容

ディスプレイに，メッセージ“1 月は，英語で January です。”を表示します。

[プログラム例 1-1-2]

Example112.java

```
1  class Example112 {
2      public static void main(String[] args) {
3          int month;
4          String name;
5          month = 1;
6          name = "January";
7          System.out.print(month + "月は，英語で ");
8          System.out.println(name + " です。");
9      }
10 }
```

実行結果

```
1月は，英語で January です。
```

[作業手順]

① Eclipse の「Java_第 1 部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第 1 部/src
名前	Example112
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

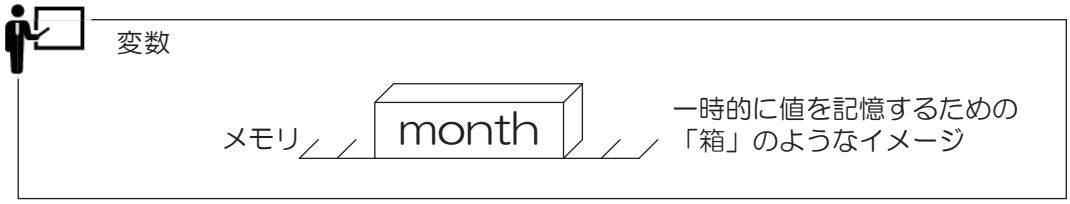
② プログラム例 1-1-2 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，プログラムを実行します。

例題1-1-2解説

(1) 変数

プログラム実行中，一時的に値を記憶するために，メモリ上に用意する場所を**変数**といいます。



プログラム例 1-1-2 では，「month」，「name」が変数です。

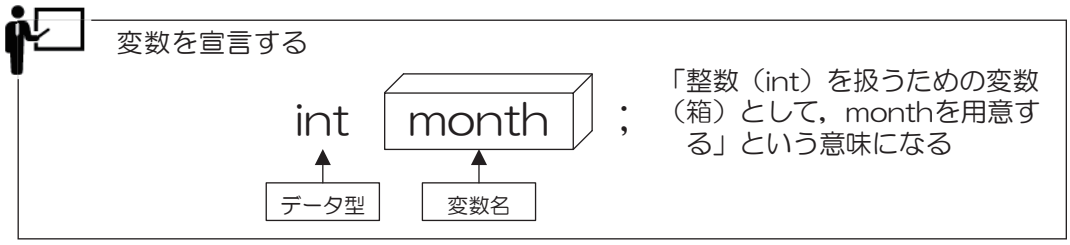
● 変数の宣言

変数を利用するには，変数を宣言します。変数の宣言によって，コンピュータが変数を用意します。

記述方法（変数の宣言）

```
データ型 変数名;
```

- **データ型**とは，変数に入れる値の種類のことです。
- 変数名には，英数字，「_（アンダースコア）」，「$（ドル記号）」を使うことができます。ただし，変数名の先頭文字に数字を使うことはできません。
- 予約されているキーワードは，変数名として使うことはできません。
  ※予約されているキーワードについては，P.32 を参照してください。
- 一般的に，変数名の先頭文字は小文字にします。



変数は，まとめて宣言することもできます。

記述方法（変数をまとめて宣言する）

```
データ型 変数名１，変数名２;
```

記述例（変数をまとめて宣言する）

```
int month, day;
```

● 変数の宣言とデータ型

変数を宣言する際に記述するデータ型には，大きく分けて次の二つがあります。

・基本データ型

基本データ型は，数値や文字などの基本的なデータを表すデータ型です。

・参照型

参照型は，インスタンスを参照するためのデータ型です。文字列を扱う String 型や配列型などがあります。

※参照型の詳細については，第２部・第２章で解説します。

基本データ型には，次のような種類があります。

〈基本データ型一覧〉

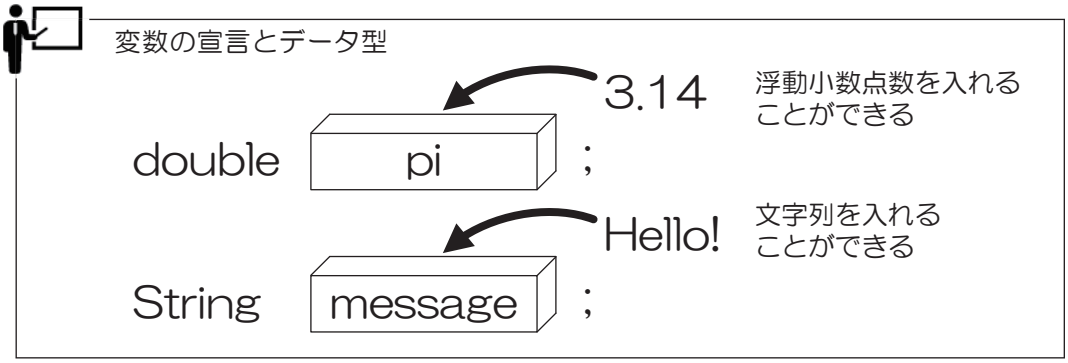
データ型	意味	記憶できる範囲
byte	整数型	−128 ～ 127
short		−32,768 ～ 32,767
int		−2,147,483,648 ～ 2,147,483,647
long		−9,223,372,036,854,775,808 ～ 9,223,372,036,854,775,807
float	浮動小数点数型	$-3.4028235\times10^{+38}$ ～ $3.4028235\times10^{+38}$
double		$-1.7976931348623157\times10^{+308}$ ～ $1.7976931348623157\times10^{+308}$
char	文字型	１文字（Unicode）
boolean	真偽値型	true（真）または false（偽）

記述例（浮動小数点数を扱う変数の宣言）

```
double pi;
```

記述例（文字列を扱う変数の宣言）

```
String message;
```



● **変数への値の代入**

変数に値を入れることを“変数に値を代入する”といいます。変数に値を代入するには，**代入演算子**「=」を使います。

記述方法（変数への値の代入）

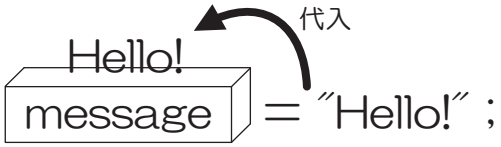
```
変数名 = 値;
```

記述例（変数への値の代入）

```
message = "Hello!";
```

変数に値を代入する

プログラム実行中一時的に
値を記憶しておく



プログラム例 1-1-2 では，int 型の変数 month に，整数「1」を代入しています。また，String 型の変数 name には，文字列「January」を代入しています。

Example112.java（抜粋）

```
5          month = 1;
6          name = "January";
```

・変数に代入した値は，プログラムの中で何度でも使うことができます。

・変数には，宣言する際に指定したデータ型以外の値を代入すると，コンパイルエラーになることがあります。

変数の宣言と同時に，値を代入することもできます。

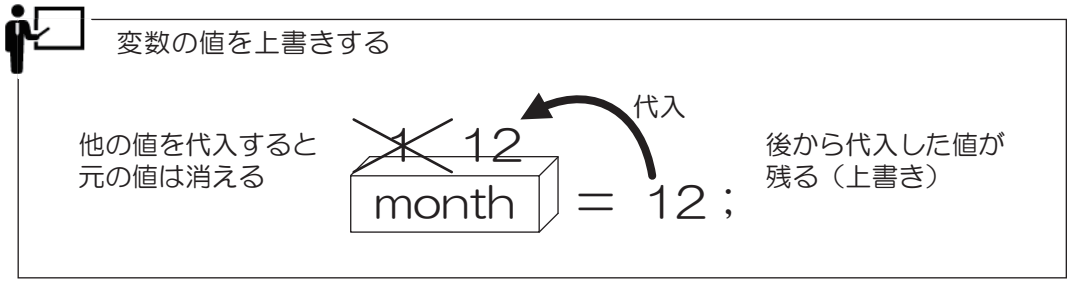
記述方法（変数の宣言と同時に値を代入する）

```
データ型 変数名 = 値;
```

記述例（変数の宣言と同時に値を代入する）

```
String message = "Hello!";
```

変数に値を代入した後，他の値を代入すると元の値は消え，後から代入した値が残ります（上書き）。



● **変数の値の表示**

変数の値を表示するには，「System.out.print()」または「System.out.println()」の( )内に対象の変数を指定します。

記述方法（変数の値を表示する）

```
System.out.println(変数名);
```

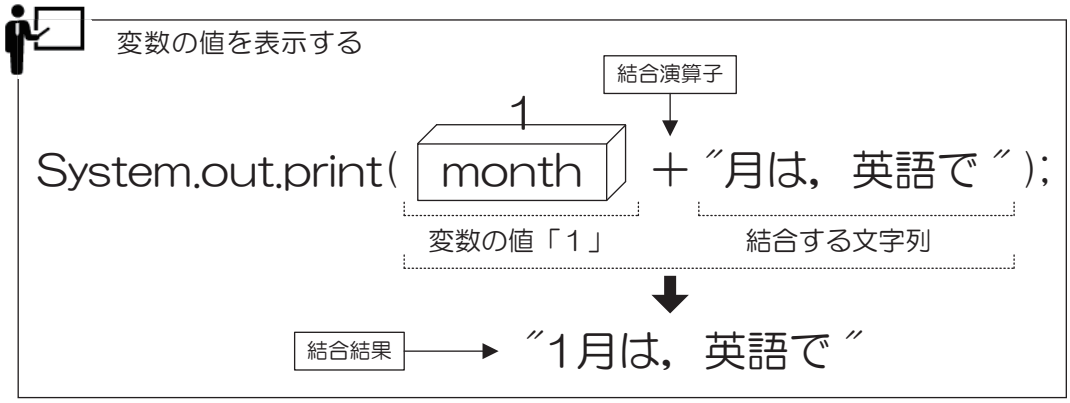
記述例（変数の値を表示する）

```
System.out.println(message);
```

プログラム例 1-1-2 では，変数 month の値と文字列「月は，英語で 」を結合して，「System.out.print()」の( )の間に指定しています。また，変数 name の値と文字列「 です。」を結合して，「System.out.println()」の( )の間に指定しています。

Example112.java（抜粋）

```
 7          System.out.print(month + "月は，英語で ");
 8          System.out.println(name + " です。");
```



・Java プログラムでは，加算に使う「+」を，文字列と他のデータを結合する演算子としても使用します。

● Java プログラムで予約されているキーワード

Java プログラムで予約さているキーワードには，次のようなものがあります。

〈予約されているキーワード一覧〉

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	false	final	finally
float	for	goto	if	implements
import	instanceof	int	interface	long
native	new	null	package	private
protected	public	return	short	static
strictfp	super	switch	synchronized	this
throw	throws	transient	true	try
void	volatile	while		

以上のキーワードは，変数名として使うことはできません。

⑵ 注釈（コメント）

プログラムの中に，「//」を使って，１行分の注釈（コメント）を記述することができます。

記述例（１行分の注釈（コメント））

```
month = 1;      // 変数 month に整数を代入する
```

また，「/*」と「*/」の間に，複数行の文字列を注釈（コメント）として記述することができます。

記述例（複数行の注釈（コメント））

```
/*  このクラスは，変数の基本を確認するクラスです。
    変数で使用するデータ型についても確認します。      */
class Example112 {
```

・注釈（コメント）を記述すると，プログラムの内容がわかりやすくなります。

・注釈（コメント）として記述された文字列は，プログラムとしては認識されません。

練習問題1-1-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
今日は，8月11日
山の日です。
```

クラス定義

クラス名　　：　Practice112

メソッド名　：　main メソッド

処理内容

ディスプレイに，メッセージ“今日は，8 月 11 日”を表示します。
ディスプレイに，メッセージ“山の日です。”を表示します。

main メソッドの処理条件

1．変数 month に，整数「8」を代入します。
2．変数 day に，整数「11」を代入します。
3．変数 message に，文字列「山の日」を代入します。
4．文字列「今日は，」，変数 month の値，文字列「月」を結合して，ディスプレイに表示します。
5．変数 day の値と文字列「日」を結合して，ディスプレイに表示します。
6．変数 message の値と文字列「です。」を結合して，ディスプレイに表示します。

第2章 演算子

1-2-1 様々な演算子

例題をとおして，様々な演算子について学習します。

例題1-2-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
2000円の税込み価格は2200.0円です。
```

クラス定義

クラス名　：　Example121

メソッド名　：　main メソッド

処理内容

ディスプレイに，価格とその税込み価格を表示します。

[プログラム例 1-2-1]

Example121.java

```
1  class Example121 {
2      public static void main(String[] args) {
3          int data = 2000;
4          double price = data * 1.1;
5          System.out.print(data + "円の税込み価格は");
6          System.out.println(price + "円です。");
7      }
8  }
```

実行結果

```
2000円の税込み価格は2200.0円です。
```

[作業手順]

① Eclipse の「Java_第 1 部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第 1 部/src
名前	Example121
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 1-2-1 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，プログラムを実行します。

例題1-2-1解説

(1) 算術演算子

演算に使用できる算術演算子には，次のような種類があります。

<算術演算子>

演算子	意味	使用例	演算結果
+	加算	3 + 2	5
-	減算	3 - 2	1
*	乗算	3 * 2	6
/	除算	3 / 2	1
%	剰余算	3 % 2	1

プログラム例 1-2-1 では，演算子「*」を使用しています。

Example121.java（抜粋）

```
4          double price = data * 1.1;
```

・整数に浮動小数点数を掛けた場合，演算結果は浮動小数点数になります。その場合，演算結果を整数型の変数に代入することはできません。
・整数を整数で割った場合，演算結果は整数部分だけになります。

・一つの式の中で，複数の算術演算子を使用する場合は，加算，減算よりも乗算，除算，剰余算の優先順位が高くなります。

記述例（複数の演算子を使用する）

```
double price = 500 + data * 1.1;
```

上記の演算では，優先順位が高い乗算「data * 1.1」が先に実行され，その結果を整数「500」に加算します。

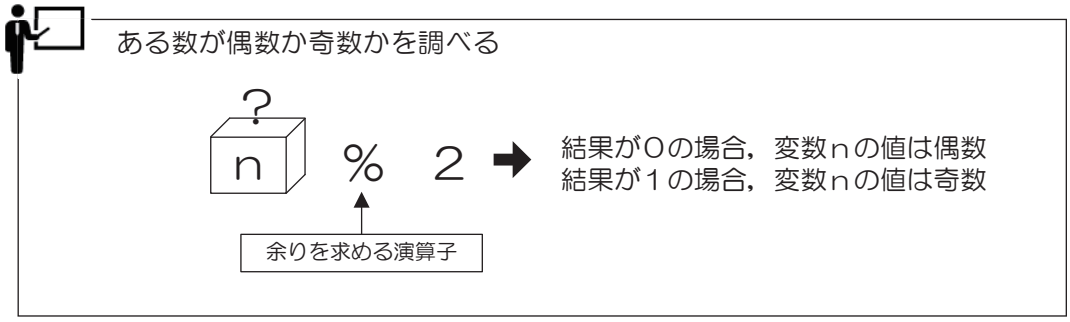
演算の優先順位を変えたい場合は，( )を使用します。

記述例（演算の優先順位を変える）

```
double price = (500 + data) * 1.1;
```

上記の演算は，優先順位が高い( )内の「500 + data」が先に実行され，その結果に浮動小数点数「1.1」を乗算します。

・剰余算に使用する「%」は，ある数を割った余りを求める演算子です。ある数が偶数か奇数かを調べる場合などに使用します。



⑵　複合代入演算子

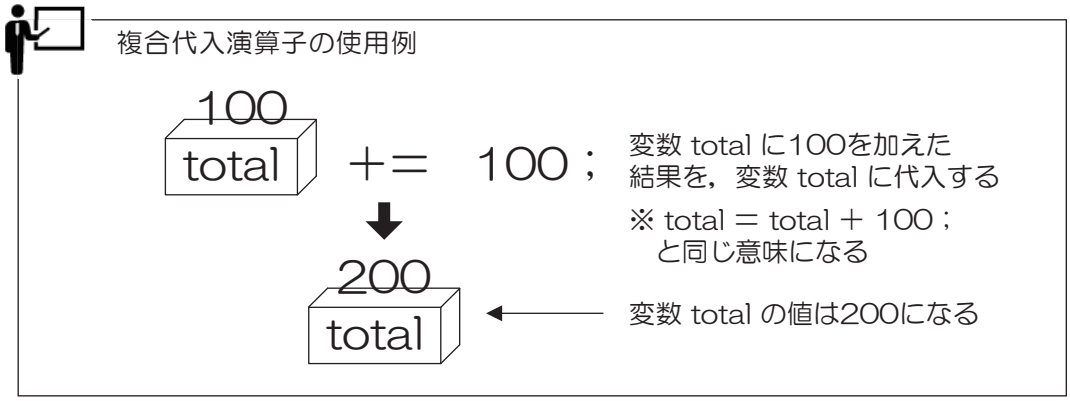
複合代入演算子は，算術演算子と代入演算子を組み合わせた演算子です。複合代入演算子には，次のような種類があります。

<複合代入演算子>

演算子	使用例	意味
+=	n += 2	変数ｎに 2 を加えた結果を，変数変数ｎに代入する
-=	n -= 2	変数ｎから 2 を引いた結果を，変数ｎに代入する
*=	n *= 2	変数ｎに 2 を掛けた結果を，変数ｎに代入する
/=	n /= 2	変数ｎを 2 で割った結果を，変数ｎに代入する
%=	n %= 2	変数ｎを 2 で割った余りを，変数ｎに代入する

記述例（複合代入演算子）

```
total += 100;
```



⑶　インクリメント演算子，デクリメント演算子

インクリメント演算子とは，変数の値を１加算する演算子です。また，**デクリメント演算子**とは，変数の値を１減算する演算子です。

●　インクリメント演算子

インクリメント演算には，演算子「++」を使用します。

記述方法（インクリメント演算子）

変数名**++;**　または　**++**変数名**;**

記述例（インクリメント演算子）

```
count++;
```

変数 count の値が０の場合，インクリメント演算の結果は１になります。

●　デクリメント演算子

デクリメント演算には，演算子「--」を使用します。

記述方法（デクリメント演算子）

変数名**--;**　または　**--**変数名**;**

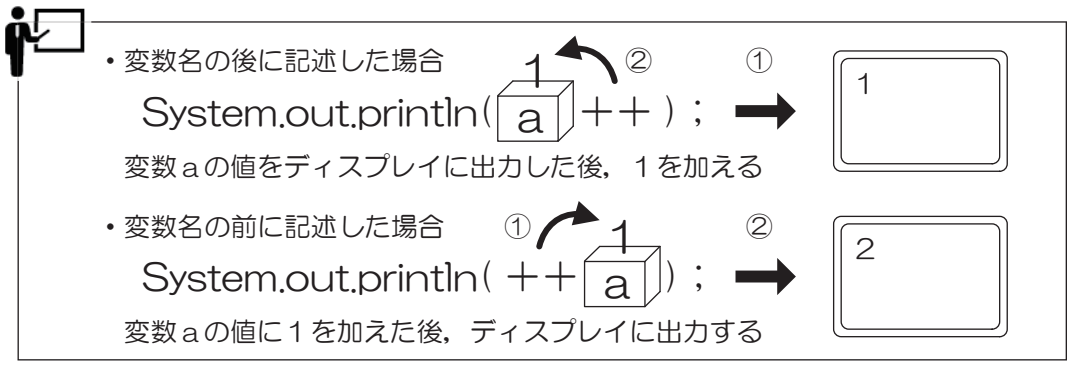
記述例（デクリメント演算子）

```
count--;
```

変数 count の値が０の場合，デクリメント演算の結果は-１になります。

・インクリメント演算子およびデクリメント演算子は，変数名の前に記述しても後に記述しても，演算結果は同じです。

・インクリメント演算子およびデクリメント演算子を，標準出力などの他の処理と一緒に使用すると，変数名の後に記述した場合と前に記述した場合で演算結果が異なる場合があります。



練習問題1-2-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
1人分料金：1800円
2人分料金：3600円
3人分料金：5400円
```

クラス定義

クラス名　　：　Practice121

メソッド名　：　main メソッド

処理内容

ディスプレイに，１人分料金を表示します。
ディスプレイに，２人分料金を表示します。
ディスプレイに，３人分料金を表示します。

main メソッドの処理条件

１．変数 price に，整数「1800」を代入します。
２．変数 total に，整数「0」を代入します。
３．変数 count に，整数「0」を代入します。
４．変数 count の値に１を加えます。
５．変数 total に，変数 price の値と変数 count の値を掛けた結果を代入します。
６．変数 count の値，文字列「人分料金：」，変数 total の値，文字列「円」を結合して，ディスプレイに表示します。
７．変数 count の値に１を加えます。
８．変数 total に，変数 price の値と変数 count の値を掛けた結果を代入します。
９．変数 count の値，文字列「人分料金：」，変数 total の値，文字列「円」を結合して，ディスプレイに表示します。
10．変数 count の値に１を加えます。
11．変数 total に，変数 price の値と変数 count の値を掛けた結果を代入します。
12．変数 count の値，文字列「人分料金：」，変数 total の値，文字列「円」を結合して，ディスプレイに表示します。

1-2-2　入力の読込みと例外処理

例題をとおして，キーボードからの入力の読込みについて学習します。また，プログラム実行時の例外処理の基本についても学習します。

例題1-2-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
チケット枚数を入力してください > 5
大人5枚の料金：3000円
```

クラス定義

クラス名　　：　Example122

メソッド名　：　main メソッド

処理内容

ディスプレイに，メッセージ“チケット枚数を入力してください > ”を表示します。
キーボードから枚数を入力して，Enter キーを押します。
ディスプレイに，入力した枚数分の料金を表示します。

[プログラム例 1-2-2]

Example122.java

```
1  import java.util.Scanner;
2  class Example122 {
3      public static void main(String[] args) {
4          Scanner sc = new Scanner(System.in);
5          System.out.print("チケット枚数を入力してください > ");
6          int count = sc.nextInt();
7          int money = 600 * count;
8          System.out.println
9              ("大人" + count + "枚の料金：" + money + "円");
10         sc.close();
11     }
12 }
```

実行結果

```
チケット枚数を入力してください > 5
大人5枚の料金：3000円
```

[作業手順]

① Eclipse の「Java_第 1 部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

<「新規 Java クラス」ウィンドウで入力する値>

入力項目	入力値
ソース・フォルダー	Java_第 1 部/src
名前	Example122
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 1-2-2 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，プログラムを実行します。

例題1-2-2解説

(1) キーボードからの入力の読込み

キーボードからの入力を読み込むには，java.util パッケージの **Scanner クラス**を使用します。Scanner クラスは，Java 言語に元々用意されています。

※パッケージの詳細については，第２部・第４章で解説します。

Scanner クラスの使用手順は，次のとおりです。

① Scanner クラスをインポートする。
※インポートの詳細については，第２部・第４章で解説します。
② Scanner クラスをインスタンス化する。
※インスタンス化の詳細については，第２部・第２章で解説します。
③ Scanner クラスのメソッドを使ってキーボードからの入力を読み込み，読み込んだ値を変数に代入する。
④Scanner クラスのインスタンスをクローズする。

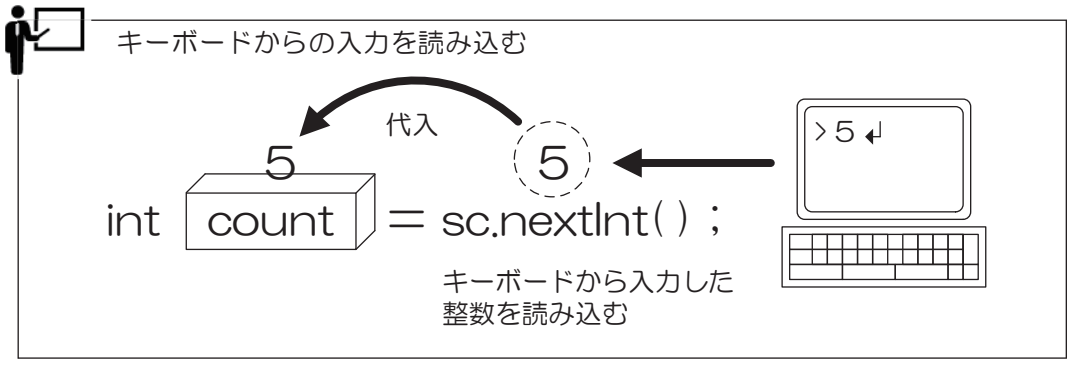
記述方法（Scanner クラスの使用手順）

```
①  import java.util.Scanner;
②  Scanner sc = new Scanner(System.in);
③  int count = sc.nextInt();
④  sc.close();
```

・Scanner クラスのインポートは，クラスの宣言よりも前に記述します。記述しない場合は，コンパイルエラーになります。

・Scanner クラスのメソッドを使用するには，Scanner クラスをインスタンス化します。

Scanner クラスの **nextInt メソッド**は，キーボードから入力された整数を読み込み，読み込んだ値を int 型で返します。



Scanner クラスには，読み込むデータ型に応じて次のようなメソッドが用意されています。

〈Scanner クラスの主なメソッド〉

メソッド名	読み込むデータ型	使用例
next	String 型	Scanner sc = new Scanner(System.in); String data = sc.next();
nextInt	int 型	Scanner sc = new Scanner(System.in); int data = sc.nextInt();
nextDouble	double 型	Scanner sc = new Scanner(System.in); double data = sc.nextDouble();

※使用例は，Scanner クラスのインポートを省略しています。

・上記のメソッドは，適切なデータ型の値を読み込まなかった場合，例外が発生します。

(2) プログラム実行時の例外処理

● 例外とは

例外とは，プログラム実行時に発生する予期せぬエラーです。Java プログラムで発生する主な例外の種類と発生するケースは，次のとおりです。

〈主な例外の種類と発生するケース〉

例外名	発生するケース
ArithmeticException	整数を 0 で除算した場合
ArrayIndexOutOfBoundsException	存在しない添字を使って配列にアクセスした場合
InputMismatchException	入力された値が，予想されるデータ型と一致しなかった場合
NullPointerException	インスタンスを参照していない変数を使って，メソッドを呼び出した場合
NumberFormatException	整数形式ではない文字列を整数に変換した場合

Scanner クラスのメソッドを使って，適切なデータ型の値を読み込まなかった場合は，InputMismatchException が発生します。

● 例外処理

Java プログラムでは，**例外処理**（例外が発生する可能性のある処理と，実際に例外が発生した場合の処理）を記述しておくことができます。

プログラム例 1-2-2 に例外処理を記述すると，次のようになります。

Example122_try_catch.java（例外処理を記述した場合）

```
 1 import java.util.Scanner;
 2 class Example122_try_catch {
 3     public static void main(String[] args) {
 4         try {
 5             Scanner sc = new Scanner(System.in);
 6             System.out.print("チケット枚数を入力してください > ");
 7             int count = sc.nextInt();
 8             int money = 600 * count;
 9             System.out.println
10                 ("大人" + count + "枚の料金：" + money + "円");
11             sc.close();
12         } catch(Exception e) {
13             System.out.println("例外発生");
14         }
15         System.out.println("終了");
16     }
17 }
```

実行結果

```
チケット枚数を入力してください > abc
例外発生
終了
```

記述方法（例外処理）

```
try {
    例外が発生する可能性がある処理
} catch(Exception e) {
    実際に例外が発生した場合に実行する処理
}
```

• try ブロック

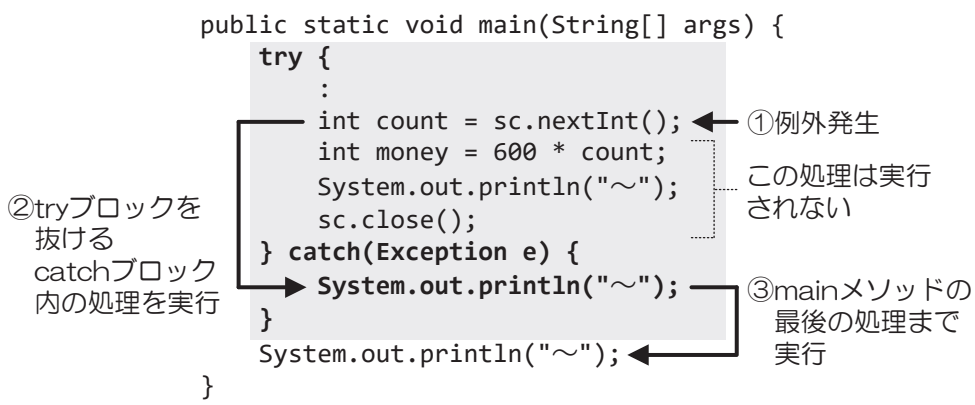
例外が発生する可能性のある処理を，**try ブロック**内に記述します。

• catch ブロック

実際に例外が発生した場合に実行する処理を，**catch ブロック**内に記述します。

・例外処理を記述しておくと，実際に例外が発生した場合もプログラムが強制終了されずに，main メソッドの最後の処理まで実行します。

実際に例外が発生した場合の処理の流れ



参考：String 型から int 型への変換

String 型（文字列）として扱っている数値を int 型（整数）に変換するには，java.lang パッケージの **Integer クラス**を使用します。Integer クラスは，Java 言語に元々用意されています。

記述例（String 型から int 型への変換）

```
//String 型（文字列）として扱っている数値を int 型（整数）に変換する
int price = Integer.parseInt("1000");
```

Integer クラスの **parseInt メソッド**は，( )内に指定した String 型の数値を int 型に変換して返します。

・Integer クラスは，インポートする必要はありません。
・parseInt メソッドは，int 型に変換できない値を( )内に指定した場合は，例外（NumberFormatException）が発生します。

・parseInt メソッドは，数値を String 型（文字列）として扱うことの多い GUI（Graphical User Interface）アプリケーションなどでよく使用されます。

練習問題1-2-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果①

実行結果

```
金額を入力してください > 23456
10000円紙幣：2枚
残り：3456円
```

実行結果②

実行結果

```
金額を入力してください > abc
エラーが発生しました！
```

クラス定義

クラス名　　：　Practice122

メソッド名　：　main メソッド

処理内容①

ディスプレイに，メッセージ“金額を入力してください > ”を表示します。
キーボードから金額を入力して，Enter キーを押します。
ディスプレイに，入力した金額に対する 10000 円紙幣の枚数を表示します。また，残りの金額も表示します。

処理内容②

キーボードから整数ではない文字列を入力して，Enter キーを押します。
ディスプレイに，メッセージ“エラーが発生しました！”を表示します。

main メソッドの処理条件

１．文字列「金額を入力してください ＞ 」を，ディスプレイに表示します。
２．変数 money に，キーボードから読み込んだ整数を代入します。
３．変数 bill に，変数 money の値を整数「10000」で割った結果を代入します。
４．変数 rem に，変数 money の値を整数「10000」で割った余りを代入します。
５．文字列「10000 円紙幣：」，変数 bill の値，文字列「枚」を結合して，ディスプレイに表示します。
６．文字列「残り：」，変数 rem の値，文字列「円」を結合して，ディスプレイに表示します。

※例外が発生した場合は，文字列「エラーが発生しました！」をディスプレイに表示します。

第3章 配列

1-3-1 一次元配列

例題をとおして，一次元配列について学習します。

例題1-3-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
大　人：1800円
高校生：1200円
中学生：600円
```

クラス定義

クラス名　　：　Example131

メソッド名　：　main メソッド

処理内容

ディスプレイに，大人の料金を表示します。
ディスプレイに，高校生の料金を表示します。
ディスプレイに，中学生の料金を表示します。

[プログラム例 1-3-1]

Example131.java

```
1  class Example131 {
2      public static void main(String[] args) {
3          int[] ticket = new int[3];
4          ticket[0] = 1800;
5          ticket[1] = 1200;
6          ticket[2] = 600;
7          System.out.println("大　人：" + ticket[0] + "円");
8          System.out.println("高校生：" + ticket[1] + "円");
9          System.out.println("中学生：" + ticket[2] + "円");
10     }
11 }
```

実行結果

```
大　人：1800円
高校生：1200円
中学生：600円
```

[作業手順]

① Eclipse の「Java_第 1 部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

<「新規 Java クラス」ウィンドウで入力する値>

入力項目	入力値
ソース・フォルダー	Java_第 1 部/src
名前	Example131
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 1-3-1 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，プログラムを実行します。

例題1-3-1解説

(1) 一次元配列インスタンスの生成

同じデータ型の変数をひとまとめにして扱う仕組みを**配列**といいます。変数が一列に並んだ状態の配列を，**一次元配列**といいます。

一次元配列を使用するには，「new」演算子を使ってインスタンスを生成します。

記述方法（一次元配列インスタンスの生成）

```
データ型[] 配列変数名 = new データ型[要素数];
```

記述例（int 型の一次元配列インスタンスの生成）

```
int[] score = new int[3];
```

記述例（String 型の一次元配列インスタンスの生成）

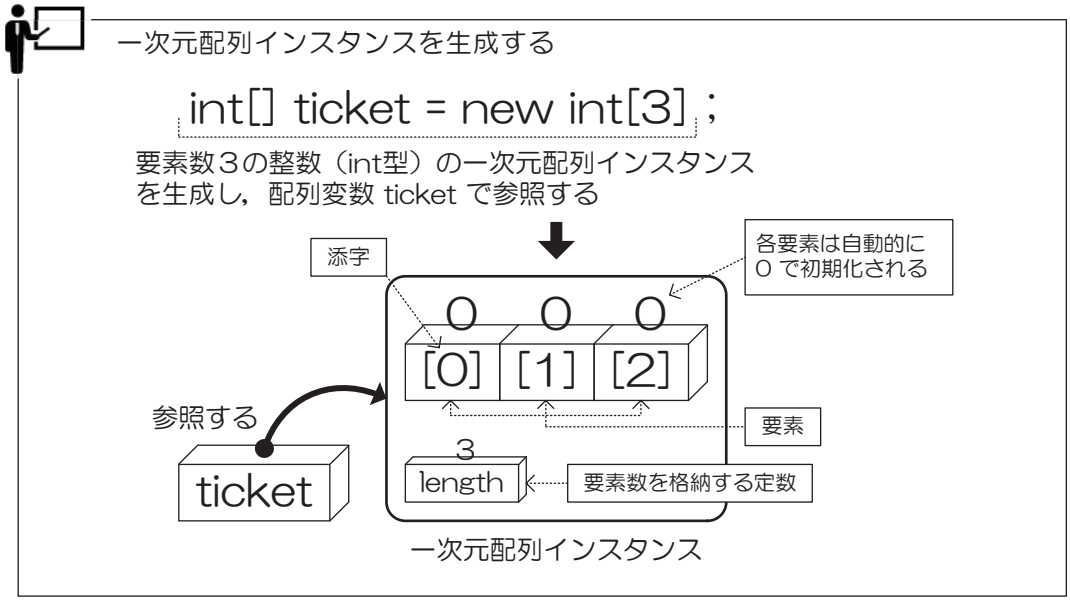
```
String[] message = new String[5];
```

・配列が持つ各変数を**要素**といいます。
・要素には0からはじまる番号が付きます。この番号を**添字**（またはインデックス）といいます。

プログラム例 1-3-1 では，「int[] ticket = new int[3];」が一次元配列インスタンスの生成です。

Example131.java（抜粋）

```
3          int[] ticket = new int[3];
```



・配列変数名に指定した変数は，生成された一次元配列インスタンスを**参照**します。
・一次元配列インスタンスが持つ定数 length には，配列の要素数を格納しています。

一次元配列の要素数は，定数 length を使用して調べることができます。

記述方法（一次元配列の要素数を調べる）

```
配列変数名.length
```

記述例（一次元配列の要素数を表示する）

```
System.out.println("要素数：" + ticket.length);
```

上記の記述例では，配列変数名 ticket が要素数３の一次元配列インスタンスを参照している場合，ディスプレイに「要素数：3」と表示されます。

・生成された一次元配列インスタンスの各要素は，自動的に初期化されます。初期化の際に代入される初期値は次のとおりです。

<一次元配列インスタンス生成時の要素の初期値>

データ型	初期値
整数型（byte，short，int，long）	0
浮動小数点数型（float，double）	0.0
文字型（char）	'¥u0000'
真偽値型（boolean）	false
参照型（String など）	null

・文字型の初期値'¥u0000'は，“文字なし”を示す値です。
・真偽値型の初期値 false は，“偽”を示す値です。
・参照型の初期値 null は，“参照なし”を示す値です。

一次元配列インスタンスの生成は，配列変数の宣言とインスタンスの生成を分けて記述することもできます。

記述方法（配列変数の宣言とインスタンスの生成を分けて行う）

```
データ型[] 配列変数名;
配列変数名 = new データ型[要素数];
```

記述例（配列変数の宣言とインスタンスの生成を分けて行う）

```
int[] score;
score = new int[3];
```

(2) 一次元配列の処理

● 一次元配列への値の代入

一次元配列の各要素に値を代入（上書き）するには，添字を指定して代入式を記述します。

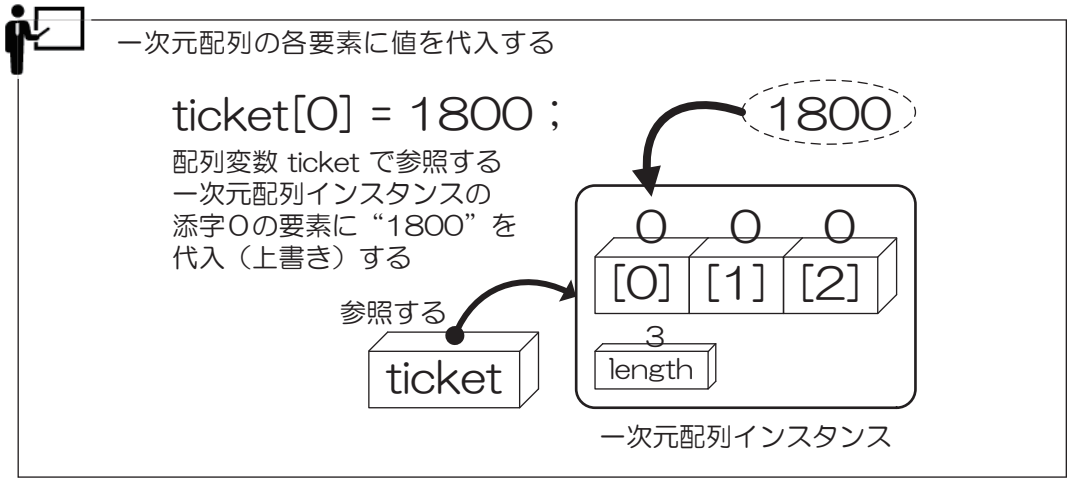
記述方法（一次元配列への値の代入）

```
配列変数名[添字] = 値;
```

プログラム例 1-3-1 では，「ticket[0] = 1800;」，「ticket[1] = 1200;」，「ticket[2] = 600;」が，一次元配列の各要素への値の代入です。

Example131.java（抜粋）

```
4          ticket[0] = 1800;
5          ticket[1] = 1200;
6          ticket[2] = 600;
```



● 一次元配列の初期化

一次元配列は，配列変数の宣言と同時に，各要素に値を代入することができます。これを**一次元配列の初期化**といいます。

記述方法（一次元配列の初期化）

```
データ型[] 配列変数名 = {値, …};
```

記述例（一次元配列の初期化）

```
int[] ticket = {1800, 1200, 600};
```

・一次元配列の初期化では，配列変数の宣言と代入式を分けて記述することはできません。記述した場合は，コンパイルエラーになります。

記述例（一次元配列の初期化：コンパイルエラー）

```
int[] ticket;
ticket = {1800, 1200, 600};
```

● 一次元配列の値の利用

一次元配列の各要素の値は，添字を指定することで，ディスプレイ表示や演算で利用することができます。

記述例（一次元配列の要素の値を演算で利用する）

```
double price = ticket[1] * 1.1;
```

プログラム例 1-3-1 では，一次元配列の要素０～２の値をディスプレイに表示しています。

Example131.java（抜粋）

```
 7         System.out.println("大　人：" + ticket[0] + "円");
 8         System.out.println("高校生：" + ticket[1] + "円");
 9         System.out.println("中学生：" + ticket[2] + "円");
```

練習問題1-3-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果①

実行結果

```
０：大人，１：高校生，２：中学生 > 1
料金：1200円
```

実行結果②

実行結果

```
０：大人，１：高校生，２：中学生 > abc
エラーが発生しました！
```

クラス定義

クラス名　　：　Practice131

メソッド名　：　main メソッド

処理内容①

ディスプレイに，メッセージ“０：大人，１：高校生，２：中学生 > ”を表示します。

キーボードから番号を入力して，Enter キーを押します。

ディスプレイに，キーボードから入力した番号の料金を表示します。

処理内容②

キーボードから整数ではない文字列を入力して，Enter キーを押します。

ディスプレイに，メッセージ“エラーが発生しました！”を表示します。

main メソッドの処理条件

１．一次元配列を参照する変数 ticket を，整数「1800」，「1200」，「600」で初期化します。
２．文字列「０：大人，１：高校生，２：中学生 > 」を，ディスプレイに表示します。
３．変数 num に，キーボードから読み込んだ整数を代入します。
４．文字列「料金：」，配列要素 ticket[num]の値，文字列「円」を結合して，ディスプレイに表示します。

※例外が発生した場合は，文字列「エラーが発生しました！」をディスプレイに表示します。

1-3-2 二次元配列

例題をとおして，二次元配列について学習します。

例題1-3-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
ID：P0111，氏名：鈴木晶子
ID：G1010，氏名：高橋大介
```

クラス定義

クラス名　　：　Example132

メソッド名　：　main メソッド

処理内容

ディスプレイに，１人目の ID と氏名を表示します。
ディスプレイに，２人目の ID と氏名を表示します。

[プログラム例 1-3-2]

Example132.java

```
1  class Example132 {
2      public static void main(String[] args) {
3          String[][] user = new String[2][2];
4          user[0][0] = "P0111";
5          user[0][1] = "鈴木晶子";
6          user[1][0] = "G1010";
7          user[1][1] = "高橋大介";
8          System.out.print("ID：" + user[0][0] + ", ");
9          System.out.println("氏名：" + user[0][1]);
10         System.out.print("ID：" + user[1][0] + ", ");
11         System.out.println("氏名：" + user[1][1]);
12     }
13 }
```

実行結果

```
ID：P0111, 氏名：鈴木晶子
ID：G1010, 氏名：高橋大介
```

[作業手順]

① Eclipse の「Java_第 1 部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第 1 部/src
名前	Example132
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 1-3-2 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，プログラムを実行します。

例題1-3-2解説

(1) 二次元配列インスタンスの生成

行と列の表形式で値を扱う配列を**二次元配列**といいます。データベースのデータのように，１件の情報が複数の項目で構成されるデータを扱う際に便利です。

二次元配列を使用するには，二次元配列インスタンスを生成します。

記述方法（二次元配列インスタンスの生成）

```
データ型[][] 配列変数名 = new データ型[行数][列数];
```

記述例（int 型の二次元配列インスタンスの生成）

```
int[][] score = new int[2][3];
```

記述例（String 型の二次元配列インスタンスの生成）

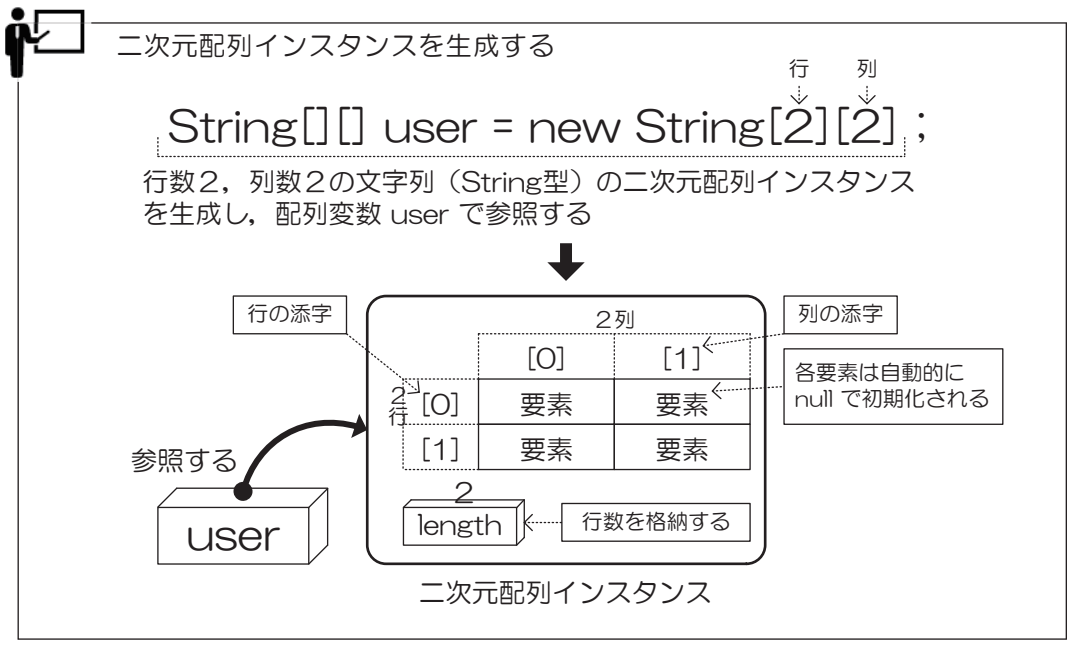
```
String[][] season = new String[4][2];
```

・二次元配列が持つ要素には，行の添字と列の添字が付きます。
・添字は，いずれも０からはじまります。

プログラム例 1-3-2 では，「String[][] user = new String[2][2];」が二次元配列インスタンスの生成です。

Example132.java（抜粋）

```
3          String[][] user = new String[2][2];
```



・二次元配列インスタンスが持つ定数 length には，行数を格納しています。

二次元配列の行数は，定数 length を使用して調べることができます。

記述方法（二次元配列の行数を調べる）

```
配列変数名.length
```

記述例（二次元配列の行数を表示する）

```
System.out.println("行数：" + user.length);
```

上記の記述例では，配列変数名 user が２行２列の二次元配列インスタンスを参照している場合，ディスプレイに「行数：2」と表示されます。

また，各行の列数を調べる場合は，次のように記述します。

記述方法（二次元配列の各行の列数を調べる）

```
配列変数名[行の添字].length
```

記述例（二次元配列の各行の列数を表示する）

```
System.out.println("０行目の列数：" + user[0].length);
```

上記の記述例では，配列変数名 user が２行２列の二次元配列インスタンスを参照している場合，ディスプレイに「０行目の列数：2」と表示されます。

・一次元配列と同様，二次元配列インスタンスの生成は，配列変数の宣言とインスタンスの生成を分けて記述することもできます。

(2) 二次元配列の処理

● 二次元配列への値の代入

二次元配列の各要素に値を代入（上書き）するには，行の添字と列の添字を指定して代入式を記述します。

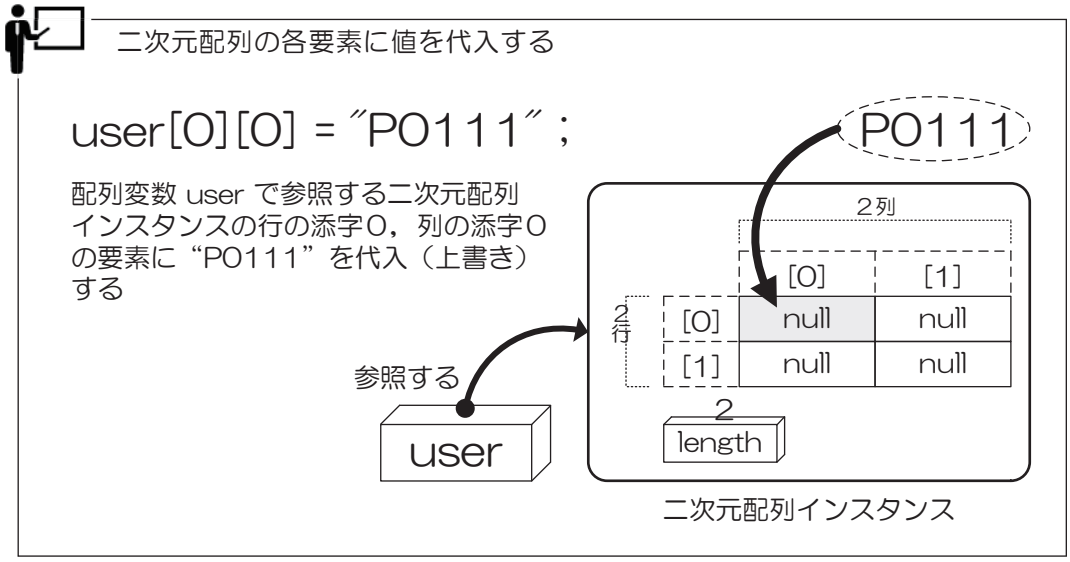
記述方法（二次元配列への値の代入）

```
配列変数名[行の添字][列の添字] = 値;
```

プログラム例 1-3-2 では，「user[0][0] = ″P0111″;」，「user[0][1] = ″鈴木晶子″;」，「user[1][0] = ″G1010″;」，「user[1][1] = ″高橋大介″;」が，二次元配列の各要素への値の代入です。

Example132.java（抜粋）

```
4           user[0][0] = "P0111";
5           user[0][1] = "鈴木晶子";
6           user[1][0] = "G1010";
7           user[1][1] = "高橋大介";
```



● 二次元配列の初期化

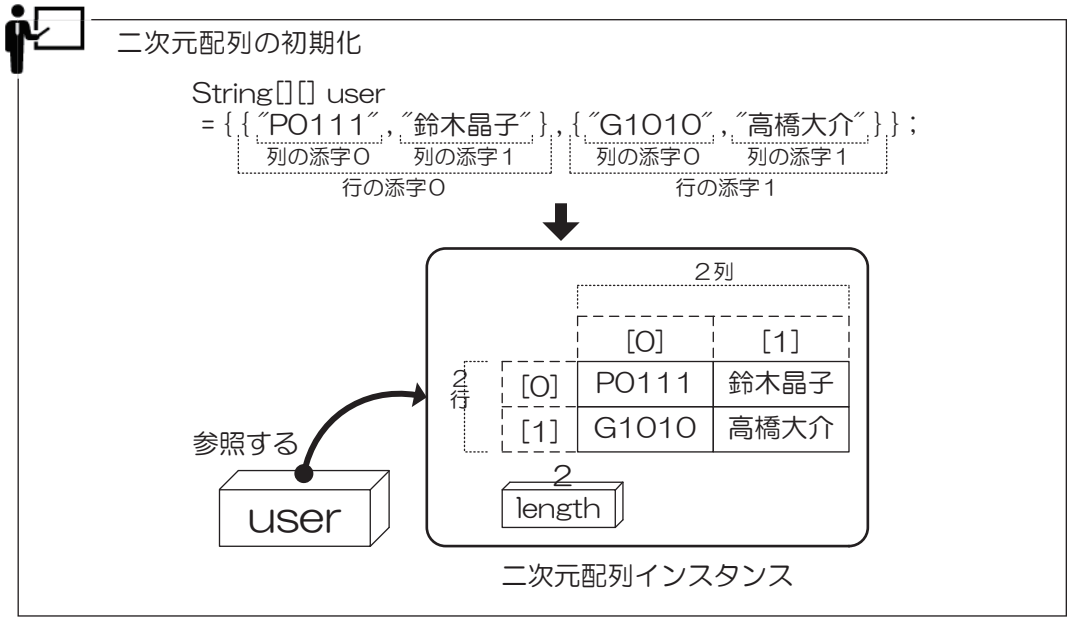
二次元配列は，配列変数の宣言と同時に，各要素に値を代入して初期化することができます。

記述方法（二次元配列の初期化）

```
データ型[][] 配列変数名 = {{値，…}，{値，…}，…};
```

記述例（二次元配列の初期化）

```
String[][] user = {{"P0111", "鈴木晶子"}, {"G1010", "高橋大介"}};
```



・二次元配列の初期化では，配列変数の宣言と代入式を分けて記述することはできません。記述した場合は，コンパイルエラーになります。

● 二次元配列の値の利用

二次元配列の各要素の値は，行の添字と列の添字を指定することで，ディスプレイ表示や演算で利用することができます。

プログラム例 1-3-2 では，二次元配列の要素の値を次のようにディスプレイに表示しています。

Example132.java（抜粋）

```
 8          System.out.print("ID：" + user[0][0] + ", ");
 9          System.out.println("氏名：" + user[0][1]);
10          System.out.print("ID：" + user[1][0] + ", ");
11          System.out.println("氏名：" + user[1][1]);
```

練習問題1-3-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果①

実行結果

```
<レンタカー料金>
０：コンパクト，１：スタンダード，２：プレミアム > 1
０：６時間，１：12時間，２：24時間 > 1
料金：8500円
```

実行結果②

実行結果

```
<レンタカー料金>
０：コンパクト，１：スタンダード，２：プレミアム > abc
エラーが発生しました！
```

クラス定義

クラス名　　：　Practice132

メソッド名　：　main メソッド

処理内容①

ディスプレイに，メッセージ“<レンタカー料金>”を表示します。
ディスプレイに，メッセージ“０：コンパクト，１：スタンダード，２：プレミアム > ”を表示します。
キーボードから番号を入力して，Enter キーを押します。
ディスプレイに，メッセージ“０：６時間，１：12 時間，２：24 時間 > ”を表示します。
キーボードから番号を入力して，Enter キーを押します。
ディスプレイに，キーボードから入力した番号のレンタカー料金を表示します。

処理内容②

キーボードから整数ではない文字列を入力して，Enter キーを押します。
ディスプレイに，メッセージ“エラーが発生しました！”を表示します。

main メソッドの処理条件

１．二次元配列を参照する変数 price を，次の値で初期化します。

	[0]	[1]	[2]
[0]	6500	7000	8000
[1]	7500	8500	9500
[2]	10000	11500	14000

２．文字列「<レンタカー料金>」を，ディスプレイに表示します。
３．文字列「０：コンパクト，１：スタンダード，２：プレミアム > 」を，ディスプレイに表示します。
４．変数 num に，キーボードから読み込んだ整数を代入します。
５．文字列「０：６時間，１：12 時間，２：24 時間 > 」を，ディスプレイに表示します。
６．変数 hour に，キーボードから読み込んだ整数を代入します。
７．文字列「料金：」，配列要素 price[num][hour]の値，文字列「円」を結合して，ディスプレイに表示します。

※例外が発生した場合は，文字列「エラーが発生しました！」をディスプレイに表示します。

第4章 制御構造(選択)

1-4-1 選択構造①

例題をとおして，一つの条件式を使った選択構造の if 文，if～else 文について学習します。また，この選択構造で使用する比較演算子とネスト（入れ子）についても学習します。

例題1-4-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
パスワード（整数４桁）> 7777
認証OK
```

クラス定義

クラス名　：　Example141

メソッド名　：　main メソッド

処理内容

ディスプレイに，メッセージ“パスワード（整数４桁）> ”を表示します。
キーボードから４桁の整数を入力して，Enter キーを押します。
ディスプレイに，メッセージ“認証 OK”を表示します。
※パスワードが正しくない場合は，メッセージを表示しません。

［プログラム例 1-4-1］

Example141.java

```
1  import java.util.Scanner;
2  class Example141 {
3      public static void main(String[] args) {
4          Scanner sc = new Scanner(System.in);
5          System.out.print("パスワード（整数４桁）> ");
6          int pass = sc.nextInt();
7          if(pass == 7777) {
8              System.out.println("認証 OK");
9          }
10         sc.close();
11     }
12 }
```

実行結果

```
パスワード（整数４桁）> 7777
認証OK
```

［作業手順］

① Eclipse の「Java_第１部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

<「新規 Java クラス」ウィンドウで入力する値>

入力項目	入力値
ソース・フォルダー	Java_第１部/src
名前	Example141
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 1-4-1 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，プログラムを実行します。

例題1-4-1解説

(1) if 文

条件によって実行する処理を分ける仕組みを，**選択構造**といいます。処理を分ける条件式が一つの場合は，**if 文**で記述します。

記述方法（if 文）

```
if(条件式) {
    条件式が true（真）だった場合に実行する処理
}
```

記述例（if 文）

```
if(count > 0) {
    average = total / count;
}
```

if 文の条件式が正しい場合は，「{」から「}」の間の処理を実行します。条件式が正しくない場合は，「{」から「}」の間の処理を実行しません。

・条件式が正しい場合を true（真），正しくない場合を false（偽）といいます。
・条件式を記述する際は，比較演算子を使用します。

● 比較演算子

条件式を記述する際に使用する**比較演算子**には，次のような種類があります。

<比較演算子>

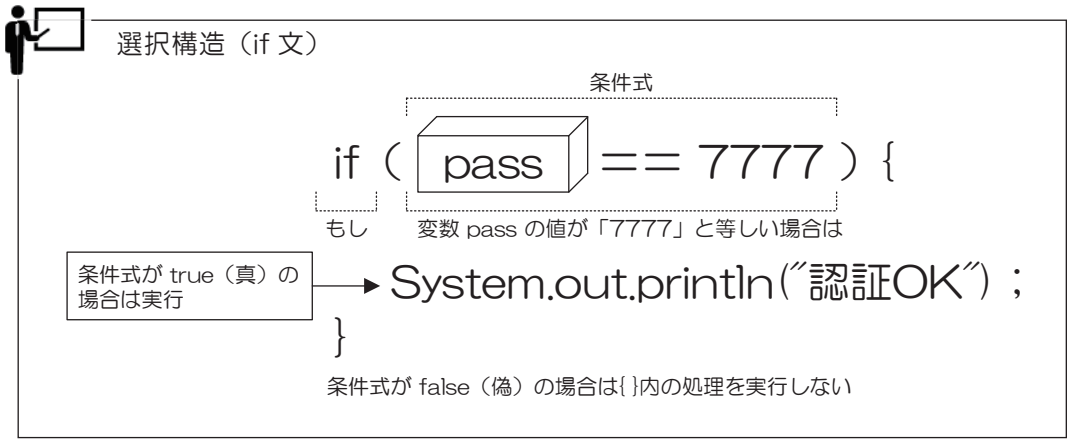
演算子	意味	記述例	記述例の意味
==	等しい	a == 10	変数 a の値は 10 と等しい
!=	等しくない	a != 10	変数 a の値は 10 と等しくない
<	より小さい	a < 10	変数 a の値は 10 より小さい
>	より大きい	a > 10	変数 a の値は 10 より大きい
<=	以下	a <= 10	変数 a の値は 10 以下
>=	以上	a >= 10	変数 a の値は 10 以上

・条件式が文字列（String 型）の場合は，比較演算子では正しく評価されません。

プログラム例 1-4-1 では，「pass == 7777」が条件式です。条件式が true（真）の場合は，「System.out.println(″認証 OK″);」を実行します。

Example141.java（抜粋）

```
 7          if(pass == 7777) {
 8              System.out.println("認証 OK");
 9          }
```



・条件式が true（真）の場合に実行する処理は，「{」から「}」の間に複数行記述することができます。

● **文字列の比較**

文字列が等しいかどうかを比較する場合は，String クラスの **equals メソッド**を使って条件式を記述します。

記述方法（equals メソッド）

変数名.equals("変数の文字列と比較する文字列")

String クラスの equals メソッドは，変数の文字列と( )内の文字列を比較し，等しい場合は true（真），等しくない場合は false（偽）を返します。

記述例（文字列の比較）

```
if(id.equals("P0111") == true) {
    :
}
```

・上記の記述例の条件式の「== true」は，省略することもできます。

(2) if～else 文

if 文の条件式が正しくない場合の処理を，**else 文**で記述することができます。この選択構造を二分岐といいます。

記述方法（if～else 文）

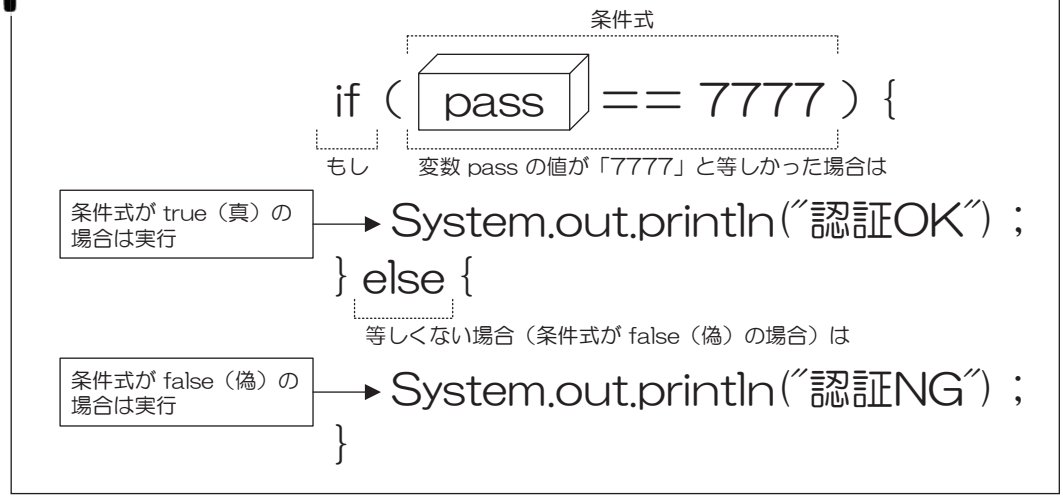
```
if(条件式) {
    条件式が true（真） の場合に実行する処理
} else {
    条件式が false（偽）の場合に実行する処理
}
```

記述例（if～else 文）

```
if(pass == 7777) {
    System.out.println("認証 OK");
} else {
    System.out.println("認証 NG");
}
```

if 文の条件式が正しい場合は，if 文の「{」から「}」の間の処理を実行します。条件式が正しくない場合は，else 文の「{」から「}」の間の処理を実行します。

if～else 文



・条件式が false（偽）の場合に実行する処理は，「{」から「}」の間に複数行記述することができます。

⑶　ネスト（入れ子）

条件によって実行する処理を分けた後に，別の選択構造を使用する状態を**ネスト**または**入れ子**といいます。

記述方法（ネスト）

```
if(条件式１) {
    if(条件式２) {
        条件式１が true（真）で，条件式２が true（真） の場合に
        実行する処理
    } else {
        条件式１が true（真）で，条件式２が false（偽）の場合に
        実行する処理
    }
}
```

記述例（ネスト）

```
if(id.equals("P0111") == true) {
    if(pass == 7777) {
        System.out.println("認証 OK");
    } else {
        System.out.println("パスワードが違います");
    }
} else {
    System.out.println("ID が違います");
}
```

・ネストは，else 文の「{」と「}」の間に記述することもできます。

・ネストは，いくつ重ねても文法上の問題はありませんが，あまり重ねすぎると処理がわかりにくくなるので注意してください。

練習問題1-4-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果①

実行結果

```
入力（1～10）> 5
入力値は範囲内です。
```

実行結果②

実行結果

```
入力（1～10）> 11
入力値は範囲外です。
```

実行結果③

実行結果

```
入力（1～10）> abc
入力値は不正です。
```

クラス定義

クラス名　　：　Practice141

メソッド名　：　main メソッド

処理内容①・②

ディスプレイに，メッセージ“入力（1～10）>　”を表示します。
キーボードから整数を入力して，Enter キーを押します。
ディスプレイに，キーボードから入力した整数に対するメッセージを表示します。

処理内容③

キーボードから整数ではない文字列を入力して，Enter キーを押します。
ディスプレイに，メッセージ“入力値は不正です。”を表示します。

main メソッドの処理条件

１．文字列「入力（1～10）＞」をディスプレイに表示します。
２．変数 num に，キーボードから読み込んだ整数を代入します。
３．変数 num の値が，１以上 10 以下の場合は，文字列「入力値は範囲内です。」をディスプレイに表示します。そうでない場合は，文字列「入力値は範囲外です。」をディスプレイに表示します。

※例外が発生した場合は，文字列「入力値は不正です。」をディスプレイに表示します。

1-4-2 選択構造②

例題をとおして，複数の条件式を使った選択構造の else if 文，switch 文ついて学習します。また，この選択構造で使用する論理演算子についても学習します。

例題1-4-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
誕生年（西暦４桁）> 1995
平成生まれです。
```

クラス定義

クラス名　：　Example142

メソッド名　：　main メソッド

処理内容

ディスプレイに，メッセージ“誕生年（西暦４桁）> ”を表示します。
キーボードから４桁の整数を入力して，Enter キーを押します。
ディスプレイに，入力した西暦に対するメッセージを表示します。
※1927 未満の整数を入力した場合は，メッセージを表示しません。

[プログラム例 1-4-2]

Example142.java

```
1  import java.util.Scanner;
2  class Example142 {
3      public static void main(String[] args) {
4          Scanner sc = new Scanner(System.in);
5          System.out.print("誕生年（西暦４桁）> ");
6          int year = sc.nextInt();
7          if(year >= 1989) {
8              System.out.println("平成生まれです。");
9          } else if(year >= 1927) {
10             System.out.println("昭和生まれです。");
11         }
12         sc.close();
13     }
14 }
```

実行結果

```
誕生年（西暦４桁）> 1995
平成生まれです。
```

[作業手順]

① Eclipse の「Java_第１部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第１部/src
名前	Example142
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 1-4-2 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，プログラムを実行します。

例題1-4-2解説

(1) else if 文

else if 文を使って条件式を追加することにより，分岐を増やすことができます。

記述方法（if～else if 文）

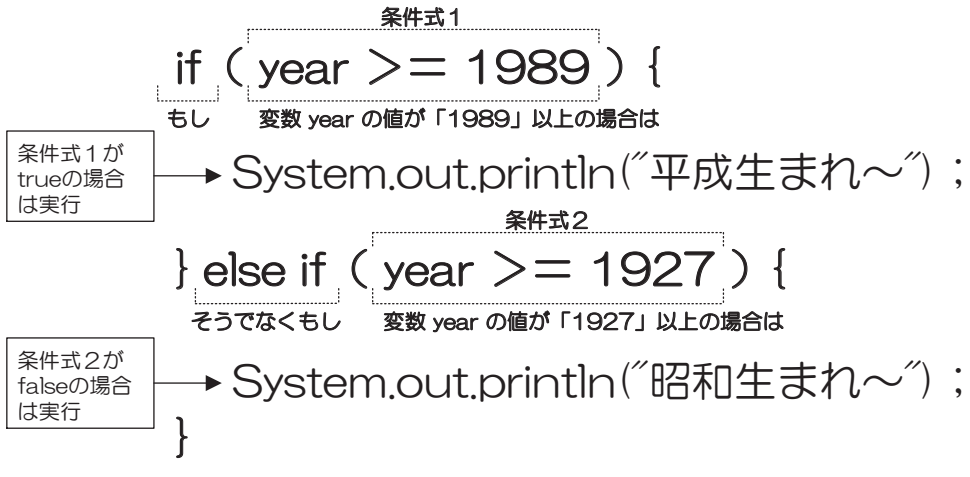
```
if(条件式１) {
    条件式１が true（真）の場合に実行する処理
} else if(条件式２) {
    条件式２が true（真）の場合に実行する処理
}
```

記述例（if～else if 文）

```
if(age < 20) {
    System.out.println("未成年");
} else if(age >= 20) {
    System.out.println("成人");
}
```

条件式１が false（偽）で，条件式２が true（真）の場合は，else if 文の「{」から「}」の間の処理を実行します。

if～else if 文



・else if 文は，複数記述することができます。

● 論理演算子

複数の条件式をまとめて記述する際に使用する**論理演算子**には，次のような種類があります。

〈論理演算子〉

演算子	意味	記述例	記述例の意味
&&	かつ	n >= 1 && n <= 10	変数 n の値は 1 以上 かつ 変数 n の値は 10 以下
||	または	n < 1 || n > 10	変数 n の値は 1 より小さい または 変数 n の値は 10 より大きい
!	ではない	!(n == 10)	変数 n の値は 10 と等しくない

・二つの条件式をまとめて記述する例

記述例（&&）

```
if(id.equals("P0111") && pass == 7777) {
    System.out.println("認証 OK");
}
```

記述例（||）

```
if(id == null || id.equals("")) {
    System.out.println("入力値が不正です。");
}
```

記述例（!）

```
if(!(num >= 1)) {
    System.out.println("入力値が不正です。");
}
```

・三つ以上の条件式をまとめて記述する例

記述例（三つ以上の条件式をまとめて記述する）

```
if(name.equals("") || address.equals("") || phone.equals("")) {
    System.out.println("未入力の項目があります。");
}
```

・論理演算子「&&」は，左辺の条件式が false の場合，右辺の条件式は評価されません。
・論理演算子「||」は，左辺の条件式が true の場合，右辺の条件式は評価されません。

(2) switch 文

switch 文を使って，変数の値によって複数の処理に分岐することができます。

記述方法（switch 文）

```
switch(変数名) {
case 値１:
    変数の値が「値１」の場合の処理
    break;
case 値２:
    変数の値が「値２」の場合の処理
    break;
  :
default:
    それ以外の場合の処理
}
```

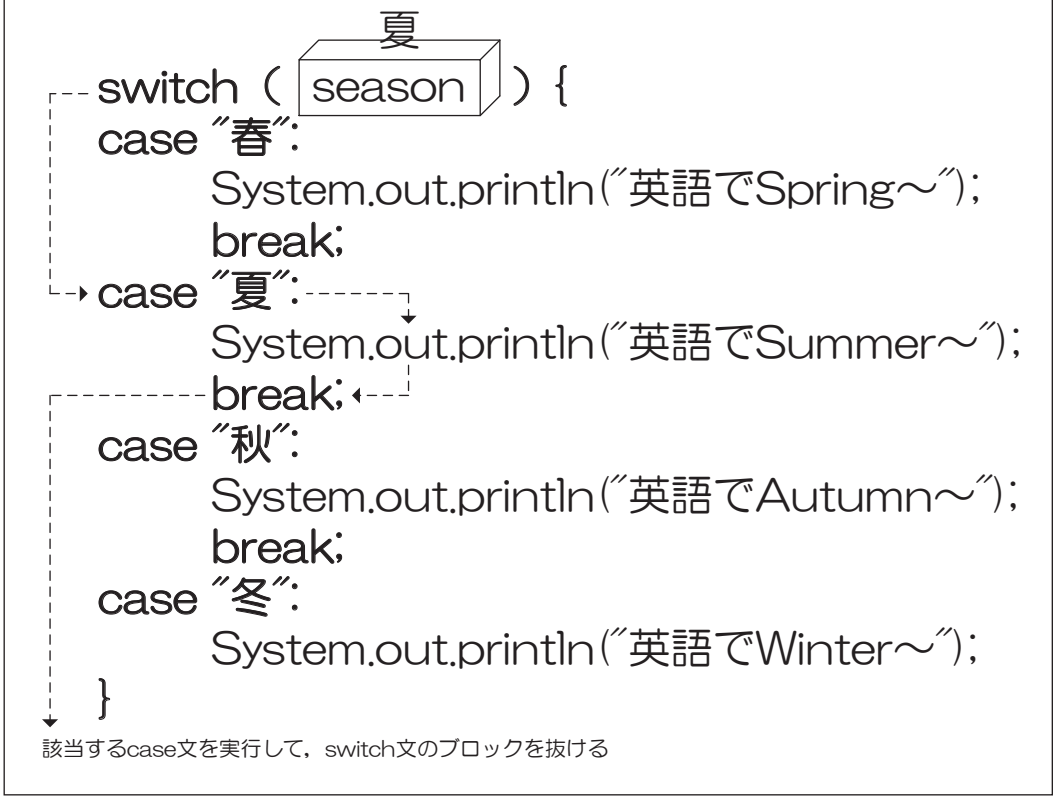
・case 文は，switch 文の「{」から「}」の間に複数記述することができます。
・switch 文で利用できる変数のデータ型は，byte 型，short 型，int 型，char 型，String 型，列挙型（本テキストでは説明していない）のいずれかです。
・default 文は，省略することもできます。

記述例（switch 文）

```
switch(season) {
case "春":
    System.out.println("英語で Spring です。");
    break;
case "夏":
    System.out.println("英語で Summer です。");
    break;
case "秋":
    System.out.println("英語で Autumn です。");
    break;
case "冬":
    System.out.println("英語で Winter です。");
    break;
default:
    System.out.println("季節ではありません。");
}
```

・各 case の break 文を省略した場合は，次の case 文の処理も実行します。

switch 文



・switch 文を使うと，変数の値によって分岐する選択構造をわかりやすく記述することができます。
・処理が複数に分岐する選択構造を，多分岐といいます。

練習問題1-4-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果①

実行結果

```
<パスワード登録>
整数４桁以上８桁以下 > 1234
パスワードを登録しました。
```

実行結果②

実行結果

```
<パスワード登録>
整数４桁以上８桁以下 > 999
もう一度やり直してください。
```

実行結果③

実行結果

```
<パスワード登録>
整数４桁以上８桁以下 > 100000000
もう一度やり直してください。
```

実行結果④

実行結果

```
<パスワード登録>
整数４桁以上８桁以下 > abc
入力値は誤りです。
```

クラス定義

クラス名　　：　Practice142

メソッド名　：　main メソッド

処理内容①

ディスプレイに，メッセージ“<パスワード登録>”を表示します。
ディスプレイに，メッセージ“整数４桁以上８桁以下 > ”を表示します。
キーボードから整数を入力して，Enter キーを押します。
ディスプレイに，メッセージ“パスワードを登録しました。”を表示します。

処理内容②・③

キーボードから，範囲外の整数を入力して，Enter キーを押します。
ディスプレイに，メッセージ“もう一度やり直してください。”を表示します。

処理内容④

キーボードから整数ではない文字列を入力して，Enter キーを押します。
ディスプレイに，メッセージ“入力値は誤りです。”を表示します。

main メソッドの処理条件

１．文字列「<パスワード登録>」を，ディスプレイに表示します。
２．文字列「整数４桁以上８桁以下 > 」を，ディスプレイに表示します。
３．変数 pass に，キーボードから読み込んだ整数を代入します。
４．変数 pass の値が整数「1000」以上でかつ「99999999」以下だった場合は，文字列「パスワードを登録しました。」をディスプレイに表示します。そうでない場合は，文字列「もう一度やり直してください。」をディスプレイに表示します。

※例外が発生した場合は，文字列「入力値は誤りです。」をディスプレイに表示します。

第5章 制御構造(反復)

1-5-1 反復構造①

例題をとおして，反復構造の while 文について学習します。また，反復構造で使用するカウンタ，break 文，continue 文についても学習します。

例題1-5-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
カウント数 > 2
カウント：0
カウント：1
```

クラス定義

クラス名　：　Example151

メソッド名　：　main メソッド

処理内容

ディスプレイに，メッセージ“カウント数 > ”を表示します。
キーボードから整数を入力して，Enter キーを押します。
ディスプレイに，「０～入力した整数－１」の間の数値を，改行しながら表示します。
※int 型の整数以外の値を入力した場合は，例外が発生します。

［プログラム例 1-5-1］

Example151.java

```
 1  import java.util.Scanner;
 2  class Example151 {
 3      public static void main(String[] args) {
 4          Scanner sc = new Scanner(System.in);
 5          System.out.print("カウント数 > ");
 6          int num = sc.nextInt();
 7          int i = 0;
 8          while(i < num) {
 9              System.out.println("カウント：" + i);
10              i++;
11          }
12          sc.close();
13      }
14  }
```

実行結果

```
カウント数 > 2
カウント：0
カウント：1
```

［作業手順］

① Eclipse の「Java_第 1 部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第 1 部/src
名前	Example151
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 1-5-1 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，プログラムを実行します。

例題1-5-1解説

(1) while 文

条件式が true（真）の間，処理を繰り返し実行する仕組みを，**反復構造**または**繰返し構造**といいます。反復構造は **while 文**で記述します。

記述方法（while 文）

```
while(継続条件) {
    継続条件が true（真）の間実行する処理
}
```

継続条件が true（真）の間，「{」と「}」の間の処理を実行します。

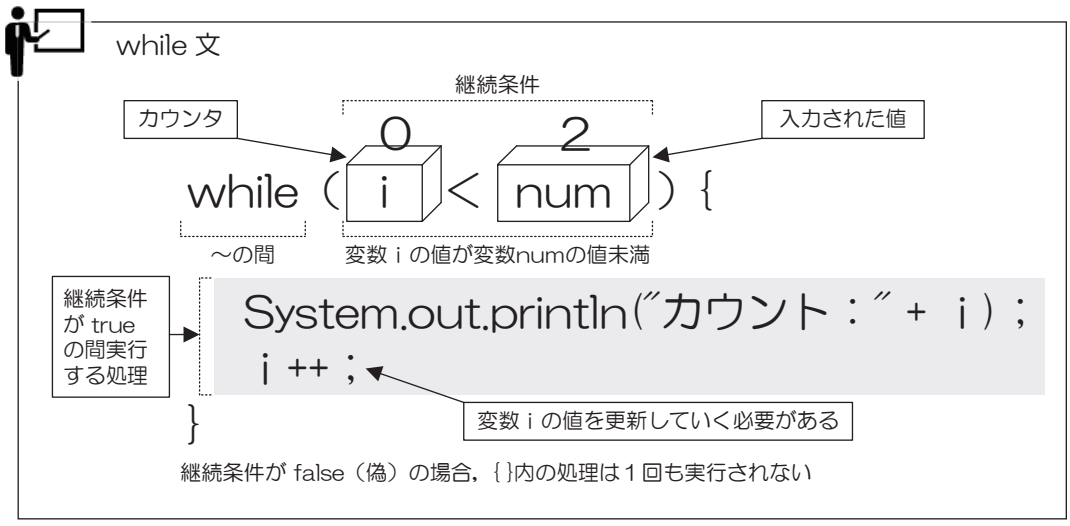
・「while」は，“～の間”という意味です。
・処理を繰り返すための条件を**継続条件**といいます。反復構造の条件式は，基本的に継続条件を使います。

・継続条件が false（偽）の場合，{ }内の処理は１回も実行されません。

プログラム例 1-5-1 では，「i ＜ num」が継続条件です。変数 i の値が変数 num の値以下である間，処理を繰り返し実行します。

Example151.java（抜粋）

```
 8          while(i < num) {
 9              System.out.println("カウント：" + i);
10              i++;
11          }
```



・継続条件が true（真）のままでは，繰返し処理は終了しません。

継続条件「i ＜ num」が false（偽）になるには，変数 i の値を更新していく必要があります。

● カウンタ

・初期値の設定

プログラム例 1-5-1 の変数 i は，これまでの繰返し回数を数えるために使用する変数です。このように，回数や個数を数えるために使用する変数のことを，**カウンタ**といいます。

・カウンタの宣言と初期化

カウンタには，数えはじめの値として初期値の設定が必要です。

記述方法（カウンタ）

```
int カウンタ;
カウンタ = 初期値;
  または
int カウンタ = 初期値;
```

・変数に初期値を設定することを，変数の初期化といいます。

プログラム例 1-5-1 では，while 文の直前の処理「int i = 0;」で，カウンタ i の宣言と初期化を行っています。

Example151.java（抜粋）

```
7           int i = 0;
```

・カウンタの値を１ずつ増やす

カウンタの値を１ずつ増やしていくことで，これまでの繰返し回数を数えます。

記述方法（カウンタの値を１ずつ増やす）

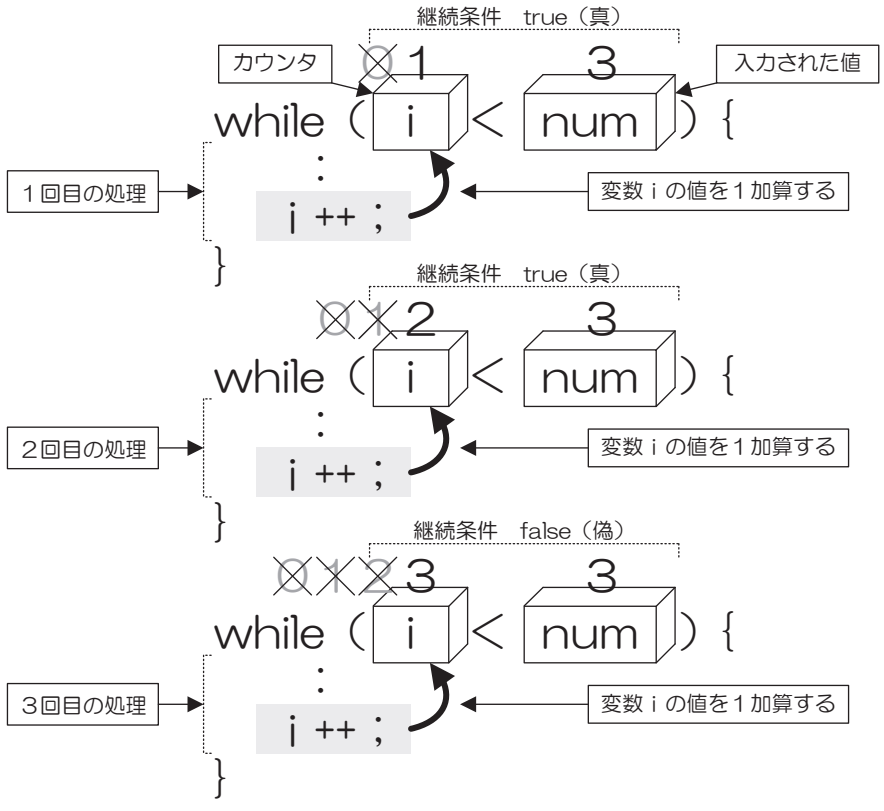
```
カウンタ++;
```

プログラム例 1-5-1 のカウンタ i は，繰返し対象の処理「i++;」によって，値を１ずつ増やします。

Example151.java（抜粋）

```
 8          while(i < num) {
 9              System.out.println("カウント：" + i);
10              i++;
11          }
```

カウンタの値を１ずつ増やす（３を入力した場合）



※３回目の処理で変数 i の値が３になり，継続条件が false（偽）になる

処理を繰り返すたびに「i++;」を実行することで，カウンタ i の値が１，２，３…と１ずつ増えていきます。このカウンタ i の値を利用して，これまでの繰返しの回数を数えていきます。

・カウンタの値を増やすことを，**カウントアップ**といいます。
・カウンタの値を減らすことを，**カウントダウン**といいます。

・反復構造で，カウントアップまたはカウントダウンの仕組みがない場合，継続条件が false（偽）にならないため，繰返し処理は終了しません。

・永久に終了しない繰返し処理を，**永久ループ**または**無限ループ**といいます。

● do～while 文

do～while 文は，継続条件の判定が繰返し処理の後で行われる反復構造です。

記述方法（do～while 文）

```
do {
    継続条件が true（真）の間実行する処理
} while(継続条件);
```

記述例（do～while 文）

```
do {
    num = sc.nextInt();
    sum = sum + num;
} while(num != 0);
```

do～while 文は，まず「{」と「}」の間の処理を実行します。その後で，継続条件の判定が行われます。

・do～while 文は，継続条件が false（偽）の場合でも，{ }内の処理を必ず 1 回は実行します。

(2) break 文

break 文を使って，繰返し処理を途中で抜けることができます。

記述方法（break 文）

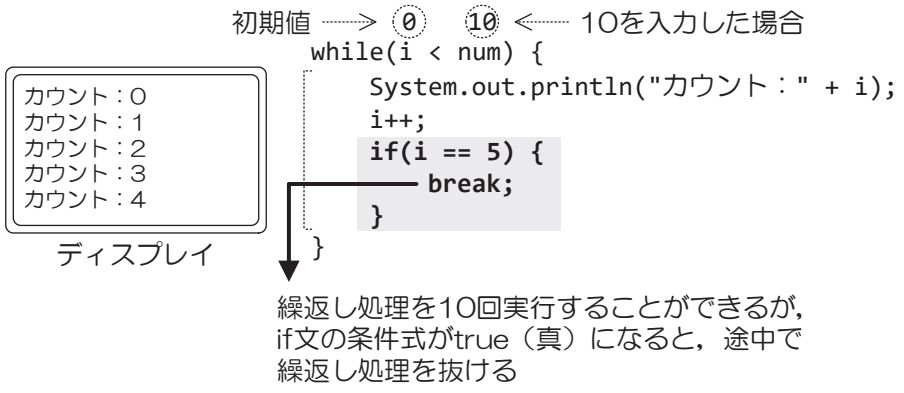
```
while(継続条件) {
    繰返し処理
    if(条件式) {
        break;
    }
}
```

if 文の条件式が true（真）の場合は，break 文によって while 文のブロックを抜けます。

記述例（break 文）

```
while(i < num) {
    System.out.println("カウント：" + i);
    i++;
    if(i == 5) {
        break;
    }
}
```

break 文（繰返し処理を途中で抜ける）



(3) continue 文

continue 文を使って，繰返し処理をスキップする（while 文の次の繰返し処理に進む）ことができます。

記述方法（continue 文）

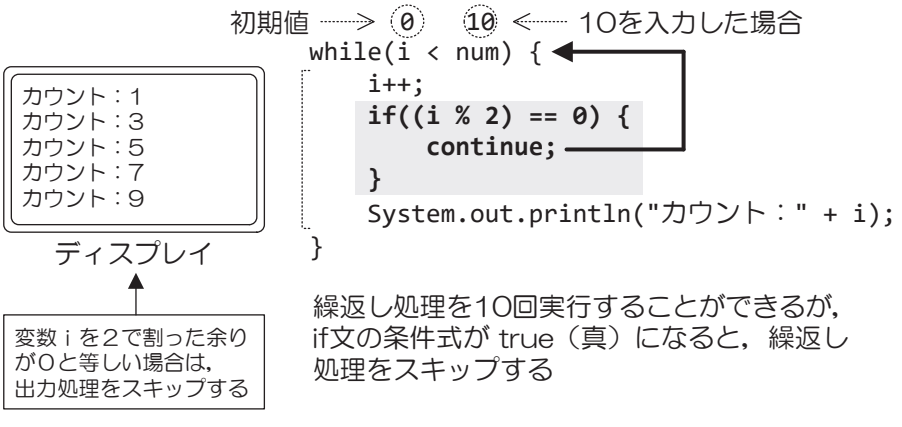
```
while(継続条件) {
    if(条件式) {
        continue;
    }
    繰返し処理
}
```

if 文の条件式が true（真）の場合は，continue 文によって繰返し処理をスキップします。

記述例（continue 文）

```
while(i < num) {
    i++;
    if((i % 2) == 0) {
        continue;
    }
    System.out.println("カウント：" + i);
}
```

continue 文（繰返し処理をスキップする）



練習問題1-5-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果①

実行結果

```
整数を入力 > 10
1から10までの合計：55
```

実行結果②

実行結果

```
整数を入力 > abc
入力値は不正です。
```

クラス定義

クラス名　　：　Practice151

メソッド名　：　main メソッド

処理内容①

ディスプレイに，メッセージ“整数を入力 > ”を表示します。
キーボードから整数を入力して，Enter キーを押します。
ディスプレイに，“１～入力した整数”の合計を表示します。

処理内容②

キーボードから整数ではない文字列を入力して，Enter キーを押します。
ディスプレイに，メッセージ“入力値は不正です。”を表示します。

main メソッドの処理条件

１．文字列「整数を入力 ＞ 」をディスプレイに表示します。
２．変数 num に，キーボードから読み込んだ整数を代入します。
３．カウンタ i に，整数「1」を代入します。
４．変数 sum に，整数「0」を代入します。
５．カウンタ i の値が変数 num の値以下の間，次の処理を繰り返します。
　・変数 sum に，“変数 sum＋カウンタ i の値”の結果を代入します。
　・カウンタ i の値を 1 加算します。
６．文字列「1 から」，変数 num の値，文字列「までの合計」，変数 sum の値を結合して，ディスプレイに表示します。

※例外が発生した場合は，文字列「入力値は不正です。」をディスプレイに表示します。

1-5-2 反復構造②

例題をとおして，for 文と拡張 for 文について学習します。

例題1-5-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
1人分料金：1800円
2人分料金：3600円
3人分料金：5400円
```

クラス定義

クラス名　　：　Example152

メソッド名　：　main メソッド

処理内容

ディスプレイに，１人分の料金を表示します。
ディスプレイに，２人分の料金を表示します。
ディスプレイに，３人分の料金を表示します。

[プログラム例 1-5-2]

Example152.java

```
1  class Example152 {
2      public static void main(String[] args) {
3          int price = 1800;
4          for(int i = 1; i <= 3; i++) {
5              int total = price * i;
6              System.out.println(i + "人分料金：" + total + "円");
7          }
8      }
9  }
```

実行結果

```
1人分料金：1800円
2人分料金：3600円
3人分料金：5400円
```

[作業手順]

① Eclipse の「Java_第 1 部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第 1 部/src
名前	Example152
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 1-5-2 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，プログラムを実行します。

例題1-5-2解説

(1) for 文

カウンタを利用した反復構造は，for 文で記述することができます。

記述方法（for 文）

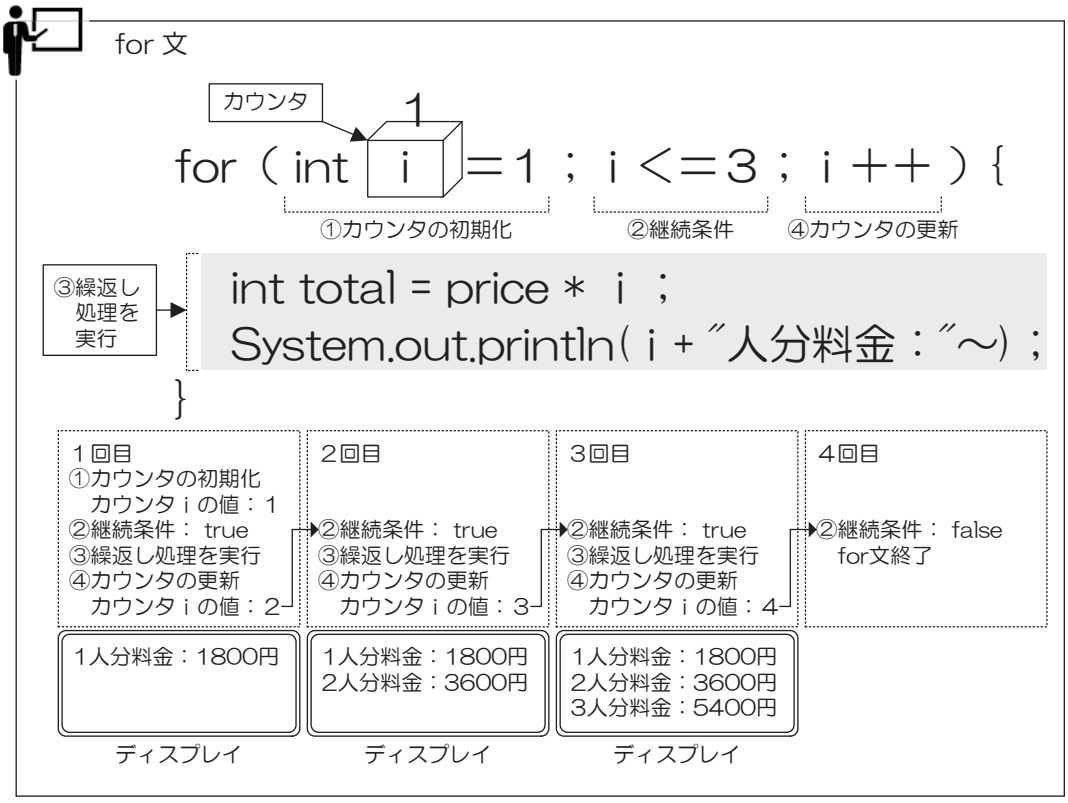
```
for(カウンタの初期化; 継続条件; カウンタの更新) {
    継続条件が true（真）の間実行する処理
}
```

Example152.java（抜粋）

```
4          for(int i = 1; i <= 3; i++) {
5              int total = price * i;
6              System.out.println(i + "人分料金：" + total + "円");
7          }
```

継続条件が true（真）の間，カウンタの値を更新しながら処理を繰り返し実行します。

・カウンタの初期化，継続条件，カウンタの更新を 1 か所にまとめて記述するため，while 文よりもコンパクトに記述することができます。



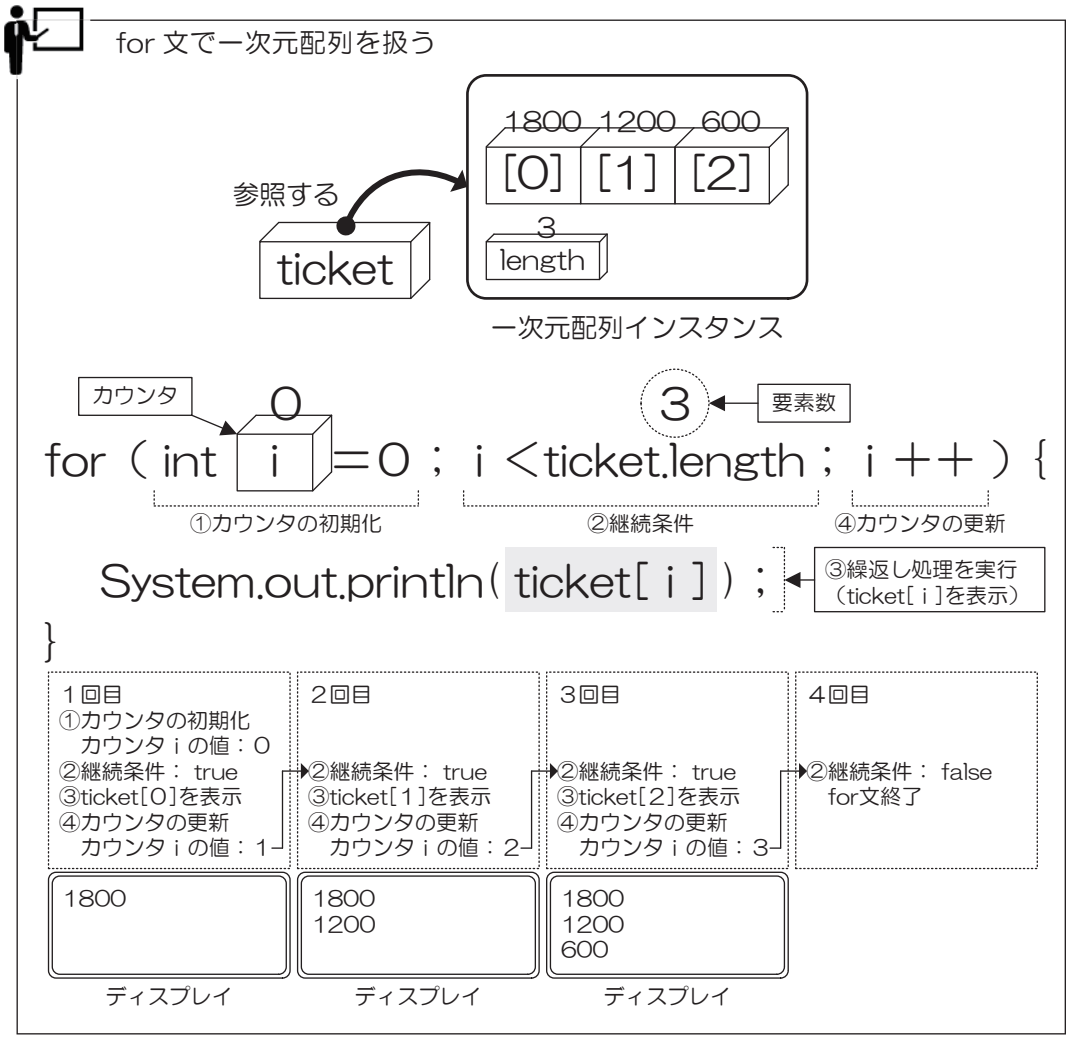
・for 文は，カウンタを利用して繰返し回数を数える反復構造に適しています。
また，配列を扱う反復構造にも適しています。

● for 文で一次元配列を扱う

for 文のカウンタを利用して，一次元配列の要素を扱うことができます。

記述例（for 文で一次元配列を扱う）

```
int[] ticket = {1800, 1200, 600};
for(int i = 0; i < ticket.length; i++) {
    System.out.println(ticket[i]);
}
```



・for 文で一次元配列の要素を扱う際の継続条件には，一次元配列の持つ定数 length を使います。

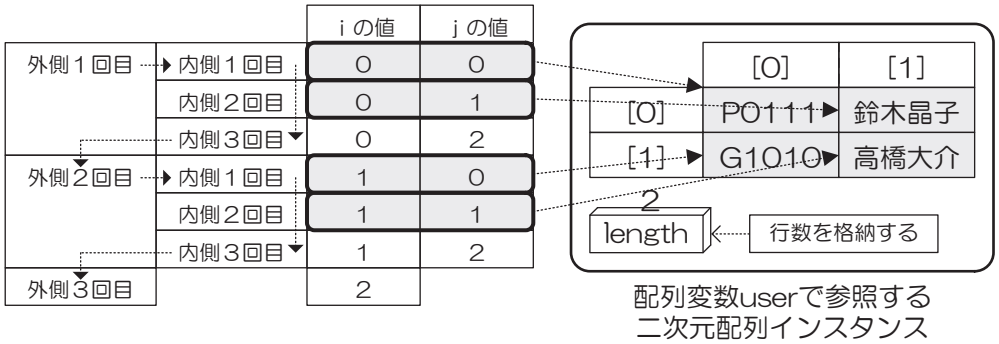
● 二重ループを使って二次元配列を扱う

二重ループを使って反復構造を記述することにより，二次元配列の要素を扱うことができます。

記述例（二重ループを使って二次元配列を扱う）

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

二重ループを使って二次元配列を扱う



● ラベルを使って二重ループを途中で抜ける

内側の繰返し処理で break 文を使って二重ループを抜ける場合は，**ラベル**を使います。

記述例（ラベルを使って二重ループを途中で抜ける）

```
outer: for(int i = 0; i < user.length; i++) {
    for(int j = 0; j < user[i].length; j++) {
        System.out.print(user[i][j]);
        if(i == 0 && j == 1) {
            break outer;
        }
    }
    System.out.println();
}
```

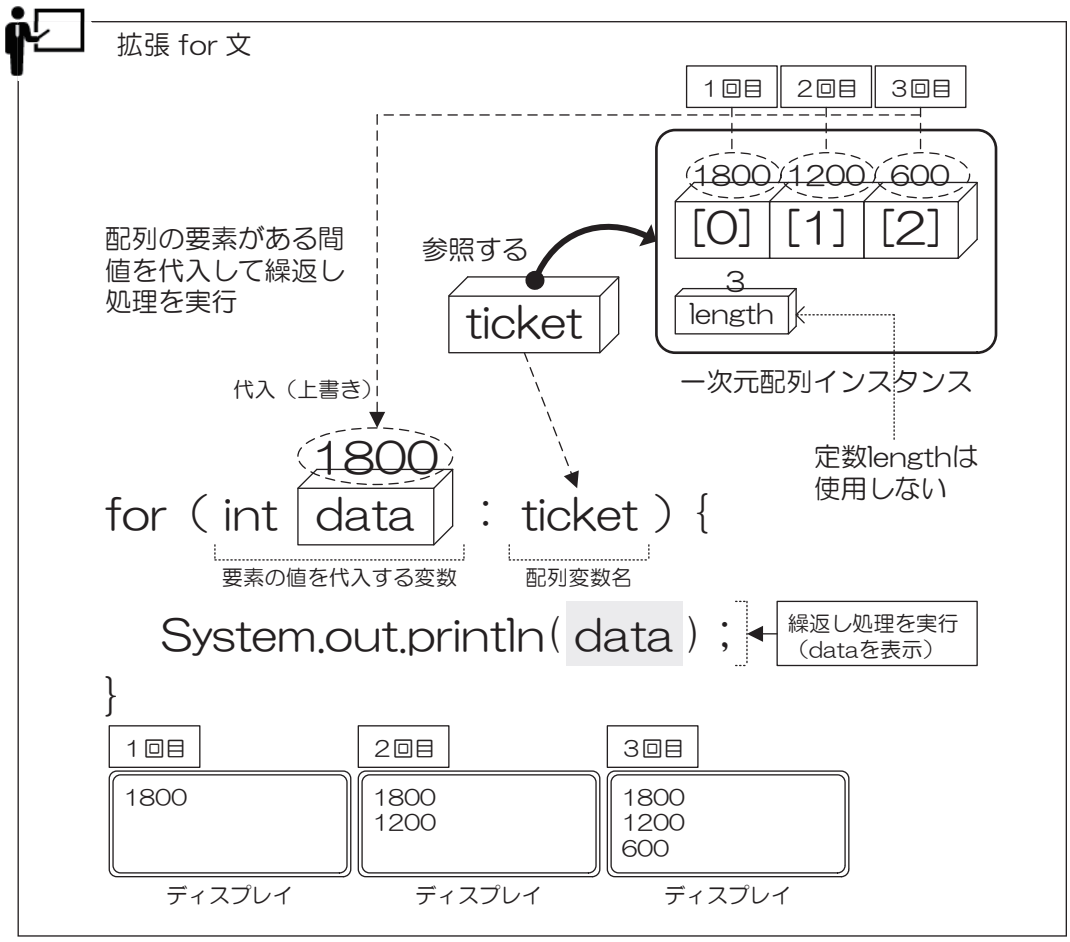
・ラベル名は，外側の繰返し処理の for 文の前に記述します。
・ラベル名は，任意で決めることができます。命名規則は，変数名と同じです。
・ラベルは，continue 文でも使用することができます。

(2) 拡張 for 文

拡張 for 文を使って反復構造を記述することにより，配列の全ての要素を扱うことができます。

記述例（拡張 for 文）

```
int[] ticket = {1800, 1200, 600};
for(int data : ticket) {
    System.out.println(data);
}
```



・拡張 for 文は，配列の要素がある間「:（コロン）」の左辺に宣言した変数に値を代入（上書き）して，繰返し処理を実行します。
・拡張 for 文は，ArrayList に対して使用することもできます。
※ArrayList の詳細については，第２部・第５章で解説します。

練習問題1-5-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果①

実行結果

```
データ入力1件目 > 1000
データ入力2件目 > 2000
データ入力3件目 > 3000
1件目：1000
2件目：2000
3件目：3000
データ合計：6000
```

実行結果②

実行結果

```
データ入力1件目 > abc
入力値は不正です。
```

クラス定義

クラス名　　：　Practice152

メソッド名　：　main メソッド

処理内容①

ディスプレイに，メッセージ“データ入力 1 件目 > ”を表示します。
キーボードからデータを入力して，Enter キーを押します。
ディスプレイに，メッセージ“データ入力 2 件目 > ”を表示します。
キーボードからデータを入力して，Enter キーを押します。
ディスプレイに，メッセージ“データ入力 3 件目 > ”を表示します。
キーボードからデータを入力して，Enter キーを押します。
ディスプレイに，入力した 3 件のデータとそのデータの合計を表示します。

処理内容②

キーボードから整数ではない文字列を入力して，Enter キーを押します。
ディスプレイに，メッセージ“入力値は不正です。”を表示します。

main メソッドの処理条件

１．一次元配列を参照する変数 sales に，要素数３の一次元配列インスタンスを生成して代入します。

２．カウンタ i の値が，“０～配列変数 sales が参照する一次元配列インスタンスの要素数未満”の間，次の処理を繰り返します。

・文字列「データ入力」，カウンタ i の値＋１の結果，文字列「件目 ＞ 」を結合して，ディスプレイに表示します。

・配列要素 sales[i]に，キーボードから読み込んだ整数を代入します。

３．変数 total に，整数「0」を代入します。

４．カウンタ i の値が，“０～配列変数 sales が参照する一次元配列インスタンスの要素数未満”の間，次の処理を繰り返します。

・カウンタ i の値＋１の結果，文字列「件目：」，配列要素 sales[i]の値を結合して，ディスプレイに表示します。

・変数 total に，変数 total の値＋配列要素 sales[i]の結果を代入します。

５．文字列「データ合計：」と変数 total の値を結合して，ディスプレイに表示します。

※例外が発生した場合は，文字列「入力値は不正です。」をディスプレイに表示します。

第6章 メソッド

1-6-1 メソッドの基本

例題をとおして，メソッドの定義・呼出しについて学習します。また，データ型の変換についても学習します。

例題1-6-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
金額 > 2000
税込金額：2200.0円
```

クラス定義

クラス名　：　Example161

メソッド名　：　main メソッド
　　tax メソッド

処理内容

ディスプレイに，メッセージ“金額 > ”を表示します。
キーボードから金額を入力して，Enter キーを押します。
ディスプレイに，入力した金額の税込金額を表示します。
※int 型の整数以外の値を入力した場合は，例外が発生します。

[プログラム例 1-6-1]

Example161.java

```
1  import java.util.Scanner;
2  class Example161 {
3      public static void main(String[] args) {
4          Scanner sc = new Scanner(System.in);
5          System.out.print("金額 > ");
6          int num = sc.nextInt();
7          double result = tax(num);
8          System.out.println("税込金額：" + result + "円");
9          sc.close();
10     }
11     static double tax(int data) {
12         double ans = data * 1.1;
13         return ans;
14     }
15 }
```

実行結果

```
金額 > 2000
税込金額：2200.0円
```

[作業手順]

① Eclipse の「Java_第 1 部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第 1 部/src
名前	Example161
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 1-6-1 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押してプログラムを保存し， ボタンを押して，プログラムを実行します。

例題1-6-1解説

(1) メソッドの定義

mainメソッドから利用されるメソッドの定義方法は，次のとおりです。

記述方法（mainメソッドから利用されるメソッドの定義）

```
class クラス名 {
    :
    static 返却値のデータ型 メソッド名
                                (データ型 引数を受け取る変数名，…) {
        具体的な処理内容
        return 返却値;
    }
}
```

・メソッドは，値を受け取り，決められた処理を実行して，結果を返します。
・staticは，“インスタンスを生成しなくても利用できる”ということを意味する修飾子です。
※インスタンスの詳細については，第２部・第２章で解説します。

プログラム例1-6-1では，mainメソッドから利用されるメソッドの定義は，次に示す11～14行目で行っています。

Example161.java（抜粋）

```
11      static double tax(int data) {
12          double ans = data * 1.1;
13          return ans;
14      }
```

・メソッドがメソッドの呼出し元から受け取る値を**引数（ひきすう）**といいます。メソッドは，引数を処理してその結果を返します。
・メソッドの呼出し元に返すメソッドの処理結果を，**返却値**または**戻り値**といいます。返却値は，**return文**で記述します。

プログラム例1-6-1のtaxメソッドの場合は，「return ans;」が返却値を返す命令です。また，( )内の「int data」が引数を受け取る変数の宣言です。

・引数を受け取る変数は，( )内に「,（カンマ）」で区切って複数宣言することができます。

(2) メソッドの呼出し

メソッドの呼出し元では，返却値を受け取る変数を用意します。

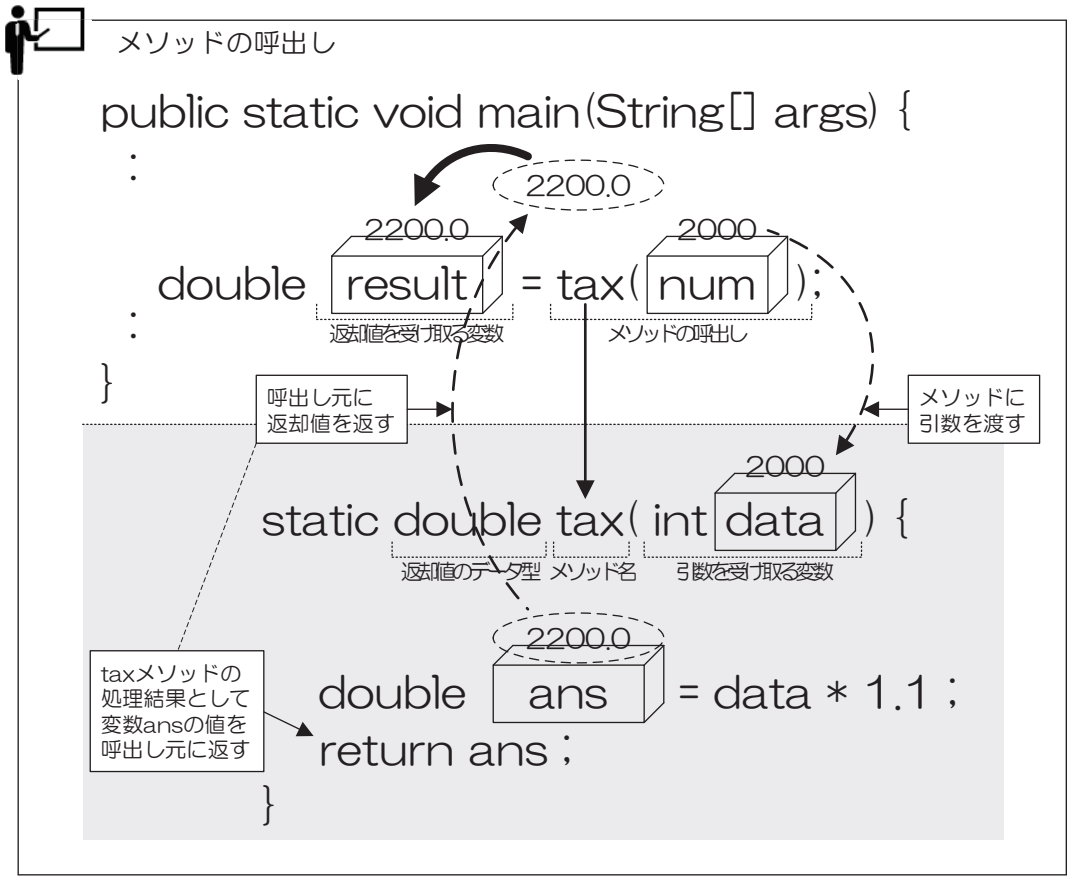
記述方法（メソッドの呼出し）

```
返却値を受け取る変数名 = メソッド名(引数);
```

プログラム例 1-6-1 では，main メソッドからのメソッドの呼出しは，次に示す 7 行目です。

Example161.java（抜粋）

```
7         double result = tax(num);
```



main メソッドから tax メソッドを呼び出す際に，変数 num の値を引数として渡します。tax メソッドは，変数 data で引数を受け取り，変数 ans に代入した値を返却値として呼出し元に渡します。呼出し元では，返却値を変数 result に代入します。

● 様々なメソッドの定義と呼出し

・引数なし，返却値なし

記述例（引数なし，返却値なし）

```
<呼出し元>
  displayTaxRate();
<メソッドの定義>
  static void displayTaxRate() {
      double taxRate = 0.1;
      System.out.println("消費税率：" + taxRate);
  }
```

・void は“返却値なし”を意味します。返却値のデータ型に void と記述したメソッドは，返却値を渡すことができません。

・引数なし，返却値あり

記述例（引数なし，返却値あり）

```
<呼出し元>
  String message = getMessage();
<メソッドの定義>
  static String getMessage() {
      return "Good morning";
  }
```

・呼出し元が返却値を受け取る変数のデータ型と，メソッド定義の返却値のデータ型は一致していないといけません。一致しない場合，コンパイルエラーになります。

・引数あり，返却値あり

記述例（引数あり，返却値あり）

```
<呼出し元>
  int total = calcTotal(1000, 2000, 3000);
<メソッドの定義>
  static int calcTotal(int data1, int data2, int data3) {
      return data1 + data2 + data3;
  }
```

・呼出し元の引数の数，データ型，順番は，メソッド定義と一致していないといけません。一致しない場合，コンパイルエラーになります。

⑶　データ型の変換

●　キャスト（明示的型変換）

キャストとは，明示的にデータ型を変換する仕組みです。キャストを利用することで，double 型の値を int 型に変換することができます。

記述方法（キャスト）

```
int 変数名 = (int)double 型の値;
```

プログラム例 1-6-1 の tax メソッドの場合，キャストを利用して double 型の演算結果の小数点以下の値を切り捨て，int 型の変数に代入することができます。

記述例（キャスト）

```
static int tax(int data) {
    int ans = (int)(data * 1.1);
    return ans;
}
※メソッドの戻り値の型を変える必要があります。
```

●　暗黙的型変換

基本データ型の値は，異なるデータ型の変数に値を代入することができます。これを**暗黙的型変換**といいます。

<暗黙的型変換のルール>

データ型	代入することができるデータ型
double 型	・float 型と double 型の値 ・すべての整数型（byte 型，short 型，int 型，long 型）の値
float 型	・float 型の値 ・すべての整数型（byte 型，short 型，int 型，long 型）の値
long 型	・すべての整数型（byte 型，short 型，int 型，long 型）の値
int 型	・byte 型，short 型，int 型の値
short 型	・byte 型と short 型の値
byte 型	・byte 型の値

記述例（暗黙的型変換）

```
double data = 1000;
```

上記のルールに反した場合は，コンパイルエラーになります。

記述例（暗黙的型変換：コンパイルエラー）

```
int data = 3.14;
```

練習問題1-6-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果①

実行結果

```
ユーザーID：> G1010
パスワード：> 8888
ログインしました。
```

実行結果②

実行結果

```
ユーザーID：> G0101
パスワード：> 7777
IDまたはパスワードが違います。
```

実行結果③

実行結果

```
ユーザーID：> G1010
パスワード：> abc
入力値は不正です。
```

クラス定義

クラス名　　：　Practice161

メソッド名　：　main メソッド
　　loginCheck メソッド

処理内容①・②

ディスプレイに，メッセージ“ユーザーID：> ”を表示します。
キーボードからユーザーID を入力して，Enter キーを押します。
ディスプレイに，メッセージ“パスワード：> ”を表示します。
キーボードからパスワードを入力して，Enter キーを押します。
ディスプレイに，入力したユーザーID とパスワードの認証結果を表示します。

処理内容③

ディスプレイに，メッセージ“ユーザーID：＞ ”を表示します。
キーボードからユーザーIDを入力して，Enterキーを押します。
ディスプレイに，メッセージ“パスワード：＞ ”を表示します。
キーボードから整数ではない文字列を入力して，Enterキーを押します。
ディスプレイに，メッセージ“入力値は不正です。”を表示します。

mainメソッドの処理条件

１．文字列「ユーザーID：＞ 」をディスプレイに表示します。
２．変数userIdに，キーボードから読み込んだ文字列を代入します。
３．文字列「パスワード：＞ 」をディスプレイに表示します。
４．変数passwordに，キーボードから読み込んだ整数を代入します。
５．loginCheckメソッドを呼び出し，返却値を変数resultに代入します。
　loginCheckメソッドに指定する引数は次のとおりです。
　第１引数：　変数userId
　第２引数：　変数password
６．変数resultの値をディスプレイに表示します。

※例外が発生した場合は，文字列「入力値は不正です。」をディスプレイに表示します。
※キーボードから入力された文字列を読み込むには，Scannerクラスのnextメソッドを使います。

loginCheckメソッドの処理条件

メソッド定義：　static String loginCheck(String id, int pass)

１．変数idの値が文字列「G1010」と等しく，かつ変数passの値が整数「8888」と等しい場合は，文字列「ログインしました。」を返します。そうでない場合は，文字列「IDまたはパスワードが違います。」を返します。

1-6-2 メソッドのオーバロード

例題をとおして，メソッドのオーバロード（多重定義）・呼出しについて学習します。また，メソッドの可変長引数についても学習します。

例題1-6-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
合計料金：1800円
合計料金：2100円
```

クラス定義

クラス名　：　Example162

メソッド名　：　main メソッド
　　charge メソッド

処理内容

ディスプレイに，メッセージ“合計金額：1800 円”を表示します。
ディスプレイに，メッセージ“合計金額：2100 円”を表示します。

[プログラム例 1-6-2]

Example162.java

```
1  class Example162 {
2      public static void main(String[] args) {
3          charge(3);
4          charge(700, 3);
5      }
6      static void charge(int count) {
7          int money = 600 * count;
8          System.out.println("合計料金：" + money + "円");
9      }
10     static void charge(int price, int count) {
11         int money = price * count;
12         System.out.println("合計料金：" + money + "円");
13     }
14 }
```

実行結果

```
合計料金：1800円
合計料金：2100円
```

[作業手順]

① Eclipse の「Java_第 1 部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第 1 部/src
名前	Example162
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 1-6-2 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押してプログラムを保存し， ボタンを押して，プログラムを実行します。

例題1-6-2解説

(1) メソッドのオーバロード

同じ名前のメソッドを複数定義することを，**オーバロード（多重定義）**といいます。

プログラム例 1-6-2 では，メソッドのオーバロードは，次に示す 6 ～13 行目で行っています。

Example162.java（抜粋）

```
 6      static void charge(int count) {
 7          int money = 600 * count;
 8          System.out.println("合計料金：" + money + "円");
 9      }
10      static void charge(int price, int count) {
11          int money = price * count;
12          System.out.println("合計料金：" + money + "円");
13      }
```

・オーバロードは，次の条件のいずれかを満たす場合に成立します。
①引数を受け取る変数の数が異なる
②引数を受け取る変数のデータ型が異なる
・オーバロードをするメソッド定義の数に制限はありません。

● オーバロードされたメソッドの呼出し

オーバロードされたメソッドの呼出し元は，引数の数とデータ型が一致するメソッドを呼び出します。

プログラム例 1-6-2 では，オーバロードされたメソッドの呼出しは，次に示す 3，4 行目で行っています。

Example162.java（抜粋）

```
 3          charge(3);
 4          charge(700, 3);
```

charge メソッドを呼び出す際に，引数を一つだけ指定した場合は，引数を一つ受け取る charge メソッドが呼び出されます。引数を二つ指定した場合は，引数を二つ受け取る charge メソッドが呼び出されます。

(2) 可変長引数

可変長引数を使用すると，メソッドの引数の数を任意の数にすることができます。

記述例（可変長引数）

```
static int calcTotal(int... data) {
    int total = 0;
    for(int price : data) {
        total = total + price;
    }
    return total;
}
```

・可変長引数を定義したメソッドでは，受け取った引数を配列として扱います。
・メソッドの呼出し元は，引数をいくつでも指定することができます。

・引数を受け取る変数の宣言が複数ある場合，可変長引数を宣言できるのは，最後の引数だけです。

可変長引数を使用したメソッドの呼出しは，次のようになります。

記述例（可変長引数を使用したメソッドの呼出し）

```
//指定する引数の数を任意の数にすることができる
int total = calcTotal(100, 200, 300);
```

参考：アサーション

アサーションは，不正なプログラム（バグ）を防ぐための機能です。開発中にこの機能を使うことで，メソッドの事前条件や事後条件をチェックすることができます。

記述例（アサーション）

```
static void charge(int count) {
    assert count >= 0;
    System.out.println("合計料金：" + (600 * count) + "円");
}
```

assert の後に記述した条件式が false の場合，AssertionError が発生します。

Eclipse でアサーションの機能を有効にするには，メニューの「実行」の「実行構成」の「引数」タブの「VM 引数」に「-ea」と入力して，「実行」ボタンを押します。

練習問題1-6-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果①

実行結果

```
<ユーザー登録>
氏名（必須）> 鈴木晶子
年齢（０：登録しない）> 24
氏名：鈴木晶子を登録しました。
年齢：24を登録しました。
```

実行結果②

実行結果

```
<ユーザー登録>
氏名（必須）> 鈴木晶子
年齢（０：登録しない）> 0
氏名：鈴木晶子を登録しました。
```

クラス定義

クラス名　　：　Practice162

メソッド名　：　main メソッド
　　register メソッド

処理内容①

ディスプレイに，メッセージ“<ユーザー登録>”を表示します。
ディスプレイに，メッセージ“氏名（必須）　>　”を表示します。
キーボードから氏名を入力して，Enter キーを押します。
ディスプレイに，メッセージ“年齢（０：登録しない）>　”を表示します。
キーボードから年齢を入力して，Enter キーを押します。
ディスプレイに，入力した氏名と年齢を表示します。

処理内容②

ディスプレイに，メッセージ“<ユーザー登録>”を表示します。
ディスプレイに，メッセージ“氏名（必須） > ”を表示します。
キーボードから氏名を入力して，Enter キーを押します。
ディスプレイに，メッセージ“年齢（０：登録しない）> ”を表示します。
キーボードから０を入力して，Enter キーを押します。
ディスプレイに，入力した氏名とメッセージを表示します。

main メソッドの処理条件

１．文字列「<ユーザー登録>」をディスプレイに表示します。
２．文字列「氏名（必須）> 」をディスプレイに表示します。
３．変数 name に，キーボードから読み込んだ文字列を代入します。
４．文字列「年齢（０：登録しない）> 」をディスプレイに表示します。
５．変数 age に，キーボードから読み込んだ整数を代入します。
６．変数 age の値が整数「0」と等しくない場合は，引数として変数 name と age を指定して register メソッドを呼び出します。そうでない場合は，引数として変数 name を指定して register メソッドを呼び出します。

※例外が発生した場合は，文字列「入力値は不正です。」をディスプレイに表示します。
※キーボードから入力された文字列を読み込むには，Scanner クラスの next メソッドを使用します。

register メソッド①の処理条件

メソッド定義： static void register(String name)

１．文字列「氏名：」，変数 name，文字列「を登録しました。」を結合して，ディスプレイに表示します。

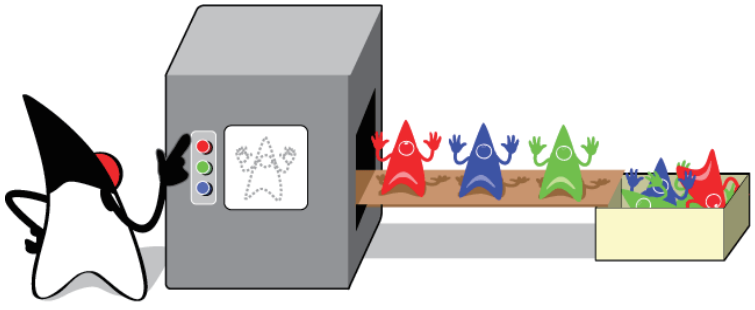
register メソッド②の処理条件

メソッド定義： static void register(String name, int age)

１．文字列「氏名：」，変数 name，文字列「を登録しました。」を結合して，ディスプレイに表示します。
２．文字列「年齢：」，変数 age，文字列「を登録しました。」を結合して，ディスプレイに表示します。

オブジェクト指向プログラミング①

第2部



第1章 オブジェクト指向とクラス定義

デュークのアドバイス

オブジェクト指向の考え方は，今では当たり前！
ステップアップでバッチリ理解！

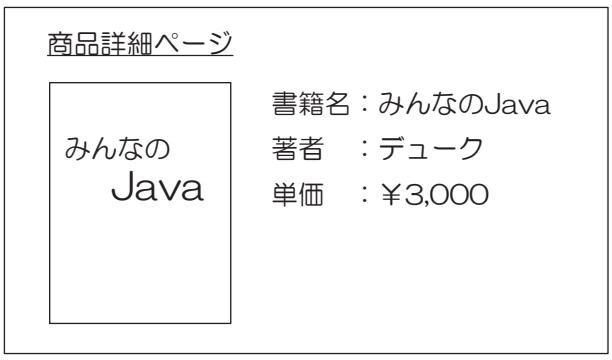
2-1-1 オブジェクト指向とは?

例題をとおして，オブジェクト指向について学習します。また，カプセル化と情報隠蔽についても学習します。

例題2-1-1

・インターネットで書籍を販売することになった。次の画面イメージを確認して，商品詳細ページの表示に必要な書籍情報オブジェクトをモデリングしなさい。

画面イメージ



処理内容

「商品詳細ページ」に，“書籍名：みんなの Java”，“著者：デューク”，“単価：￥3, 000”を表示します。

※一般的には，文字だけでなく画像も関連するデータですが，ここでは省略します。

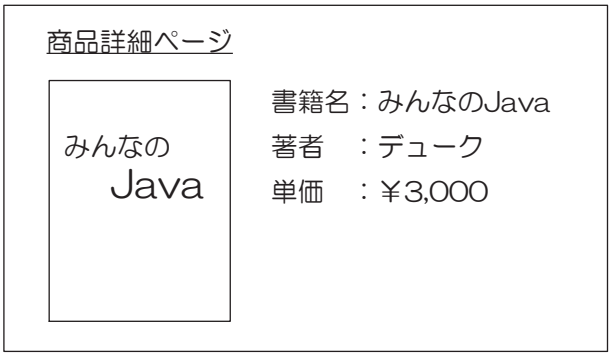
[モデリング例 2-1-1]

書籍情報オブジェクト

書籍情報オブジェクト

データ
書籍名：みんなのJava
著者　：デューク
単価　：￥3,000
機　能
書籍情報を商品詳細ページに表示する

画面イメージ



[作業手順]

① 商品詳細ページの表示に必要なデータ「書籍名：みんなの Java」，「著者：デューク」，「単価：￥3, 000」を記述します。

② 商品詳細ページの表示に必要な機能「書籍情報を商品詳細ページに表示する」を記述します。

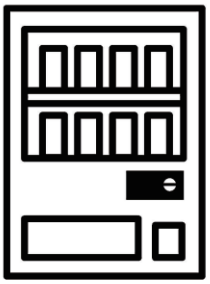
・機能名は汎用的な表現にします。例えば，「みんなの Java の情報を表示する」などは，好ましくありません。

例題2-1-1解説

(1) オブジェクト指向

オブジェクト指向とは，作成（開発）する対象（ものや事柄）を**オブジェクト（もの）**としてとらえる考え方です。

自動販売機を開発する場合



- 自動販売機をオブジェクトとしてとらえる
- 自動販売機オブジェクトがどんな性質を持っているのか考える

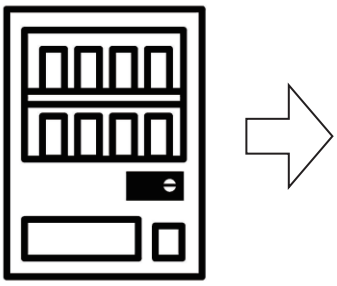
対象オブジェクトがどんな性質を持っているのかは，実際の開発でも当たり前のように考えられています。

・コンピュータ上で動くシステムについても，現実世界で扱うオブジェクトと同じようにとらえます。

● オブジェクトのモデリング

オブジェクト指向では，オブジェクトが持つ性質を**データ**と**機能**に分けてモデリングします。

オブジェクトのモデリング



自動販売機オブジェクト

データ（属性）
商品名　：コーヒー
価格　　：130円
支払金額：200円
機　能（操作）
商品を出す
おつりを計算する

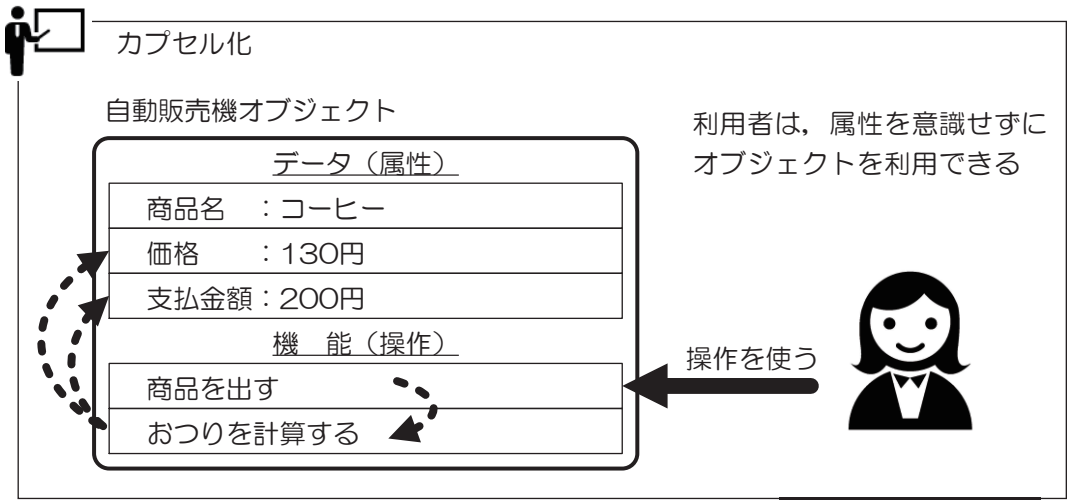
・オブジェクト指向では，オブジェクトのモデリングで分けたデータのことを**属性**，機能のことを**操作**と呼びます。

⑵　カプセル化と情報隠蔽

●　カプセル化

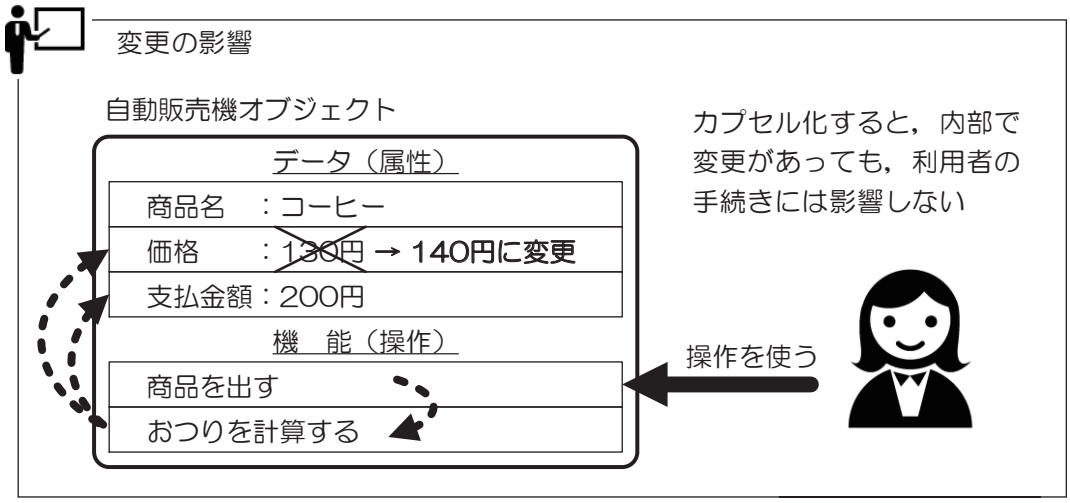
オブジェクトの持つ属性と操作が一体となって振る舞う状態を，**カプセル化**といいます。

利用者は，属性を意識せずにオブジェクトを利用することができます。



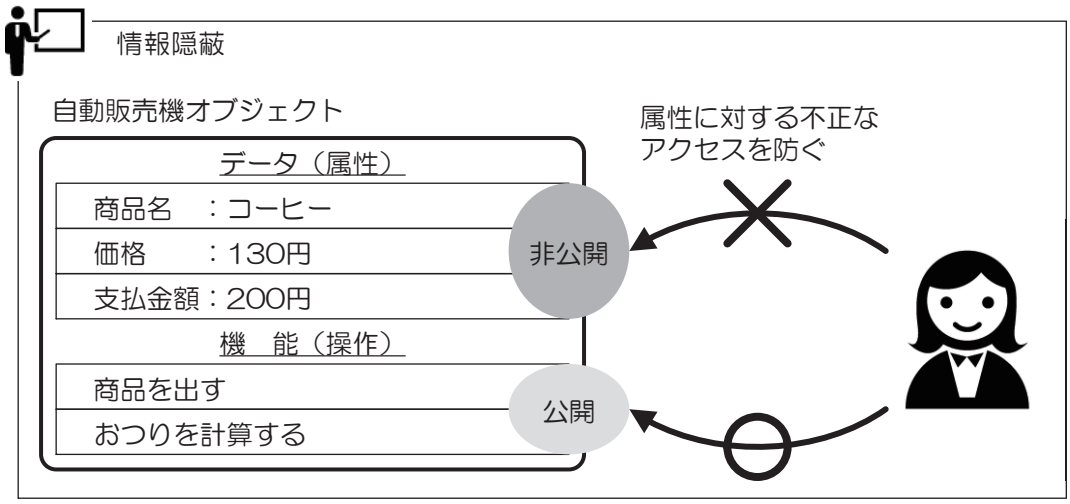
●　変更の影響

カプセル化されたオブジェクトは，内部で属性値や操作の詳細に変更があっても，利用者の手続きには影響しません。



● **情報隠蔽**

オブジェクトの持つ属性を非公開にし，操作のみを公開することを**情報隠蔽**といいます。



・情報隠蔽によって，属性に対する不正なアクセスを防ぐことができます。

・オブジェクトをモデリングした結果は図表にまとめます。
・一般的に，オブジェクト指向における図の表記法はUML（統一モデリング言語）に従います。本テキストに掲載した図は，オリジナルの表記法です。

練習問題 2-1-1

・インターネットで会員情報を確認するための Web ページを，作成することになった。次の画面イメージを確認して，会員情報・確認ページの表示に必要なオブジェクトをモデリングしなさい。

画面イメージ

会員情報・確認ページ

氏名	：	神田　二郎
メールアドレス	：	kanda@java.co.jp
登録年月日	：	2014/04/01
会員年数	：	6年

処理内容

「会員情報・確認ページ」に，“氏名：神田　二郎”，“メールアドレス：kanda@java.co.jp”，“登録年月日：2014/04/01”，“会員年数：6 年”を表示します。

「会員年数」は，「登録年月日」から計算して求めます。

会員情報オブジェクト

データ（属性）
機　能（操作）
会員情報を表示する

2-1-2 クラス定義

例題をとおして，クラス定義の基本について学習します。また，インスタンス変数とメソッドについても学習します。

例題2-1-2

・次のオブジェクトを確認して，適切なクラスを定義しなさい。

オブジェクト

顧客オブジェクト

データ（属性）
ＩＤ：2222
氏名：淡路　花子
機能（操作）
顧客情報を表示する

クラス定義

Customer
int id String name
void display()

displayメソッドの処理条件

・インスタンス変数idとnameの値をディスプレイに表示します。

[プログラム例 2-1-2]

Customer.java

```
1  class Customer {
2      int id;
3      String name;
4      void display() {
5          System.out.println("ID   : " + id);
6          System.out.println("氏名 : " + name);
7      }
8  }
```

・このプログラムは実行できません。

[作業手順]

① Eclipse の「Java_第２部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第２部/src
名前	Customer
修飾子	「パッケージ」にチェックを入れる

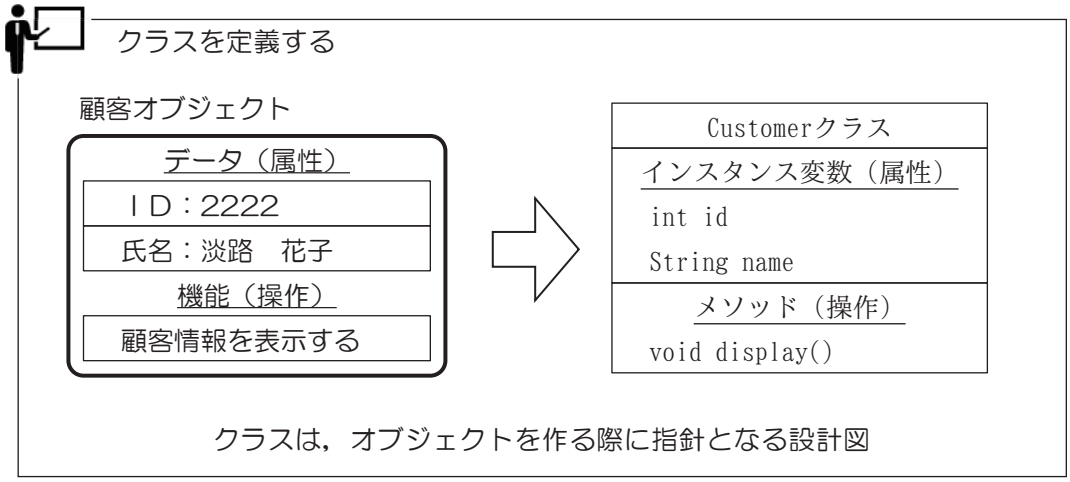
※これ以外の項目には，入力しません。

② プログラム例 2-1-2 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

例題2-1-2解説

(1) クラス定義の基本

オブジェクト指向では，オブジェクトの構造を抽象化して**クラス**を定義します。
クラスは，オブジェクトを作る際に指針となる設計図のようなものです。
※「抽象化」とは，必要な情報だけを抽出して一般化することです。



・クラスからオブジェクトを生成することを，**インスタンス化**といいます。
・クラスから生成されたオブジェクトを，**インスタンス**といいます。
※インスタンス化の詳細については，第２部・第２章で解説します。

・一般的に，クラスの表記はUML（統一モデリング言語）に含まれるクラス図の表記法に従います。本テキストのクラスの表記はオリジナルの表記法です。

記述方法（クラス定義）

```
class クラス名 {
    :
}
```

classに続けて，任意のクラス名を記述します。
プログラム例2-1-2では，クラスの範囲は次に示す１～８行目です。

Customer.java（抜粋）

```
1  class Customer {
:      :
8  }
```

⑵　インスタンス変数とメソッド

●　インスタンス変数の宣言

クラスに関連するデータ（属性）を記憶するための変数を，**インスタンス変数**といいます。

記述方法（インスタンス変数の宣言）

```
class クラス名 {
    //インスタンス変数の宣言
    データ型 変数名１;
    データ型 変数名２;
     :
}
```

インスタンス変数は，クラスのブロック内に宣言します。
プログラム例 2-1-2 では，インスタンス変数の宣言は次に示す２，３行目です。

Customer.java（抜粋）

```
 2      int id;
 3      String name;
```

インスタンス変数は，まとめて宣言することもできます。

記述例（インスタンス変数をまとめて宣言する）

```
class Customer {
    int id, age;
    String name, address;
     :
}
```

インスタンス変数の宣言と同時に，値を代入することもできます。

記述例（インスタンス変数の宣言と同時に，値を代入する）

```
class Customer {
    int id = 2222;
    String name = "淡路　花子";
     :
}
```

● メソッドの定義

インスタンスの利用者から利用されるメソッドの定義方法は，次のとおりです。

記述方法（メソッドの定義）

```
class クラス名 {
    :
    //メソッドの定義
    返却値のデータ型 メソッド名(データ型 引数を受け取る変数名，…) {
        具体的な処理内容
        return 返却値;
    }
    :
}
```

メソッドは，クラスのブロック内に定義します。
プログラム例 2-1-2 では，メソッドの定義は次に示す４～７行目です。

Customer.java（抜粋）

```
4      void display() {
5          System.out.println("ID   :" + id);
6          System.out.println("氏名：" + name);
7      }
```

プログラム例 2-1-2 の display メソッドの場合は，引数のないメソッドです。また，返却値もありません。

・メソッドは，同じクラス内のインスタンス変数を利用することができます。

練習問題2-1-2

・次のオブジェクトを確認して，適切なクラスを定義しなさい。

オブジェクト

製品オブジェクト

データ（属性）
ナンバー：T001
製品名　：トートバッグ
価格　　：8000円
数量　　：5個
機能（操作）
製品情報を表示する

クラス定義

Product
String no String name int price int quantity
void display()

display メソッドの処理条件

メソッド定義：　　void display()

1．インスタンス変数 price の値と quantity の値を掛けた結果を，変数 total に代入します。

2．インスタンス変数 no，name，price，quantity，total の値をディスプレイに表示します。

※このプログラムは実行できません。

第2章 オブジェクトの生成とインスタンスの利用

デュークのアドバイス

オブジェクトの利用は，イメージが大事！
手順を覚えて着実にマスター！

2-2-1 オブジェクトの生成

例題をとおして，オブジェクトの生成について学習します。また，複数のインスタンスの生成についても学習します。

例題2-2-1

・プログラム例 2-1-2 で作成した Customer クラスの，インスタンスを生成するプログラムを作成しなさい。

Customer.java（プログラム例 2-1-2 で作成済）

```
1  class Customer {
2      int id;
3      String name;
4      void display() {
5          System.out.println("ID   :" + id);
6          System.out.println("氏名：" + name);
7      }
8  }
```

クラス定義

クラス名　　：　Example221

メソッド名　：　main メソッド

[プログラム例 2-2-1]

Example221.java

```
1  class Example221 {
2      public static void main(String[] args) {
3          Customer cust = new Customer();
4      }
5  }
```

実行結果

（何も表示されません）

・このプログラムは，実行してもなにも表示されません。

[作業手順]

① Eclipse の「Java_第２部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第２部/src
名前	Example221
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 2-2-1 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

・保存後，「ローカル変数 cust の値は使用されていません」という警告メッセージが出ますが，問題ありません。

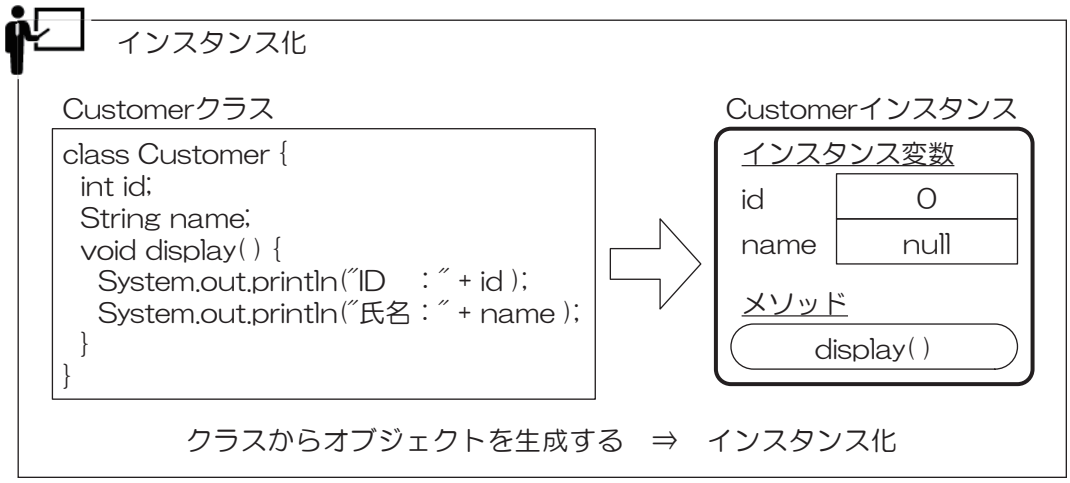
③ Eclipse の ボタンを押して，プログラムを実行します。

例題2-2-1解説

(1) オブジェクトの生成

● **インスタンス化**

クラスからオブジェクトを生成することを，**インスタンス化**といいます。また，クラスから生成されたオブジェクトを**インスタンス**といいます。



・クラスに定義したインスタンス変数やメソッドは，インスタンス化して初めて利用することができます。

インスタンス化の命令には，new を使います。

記述方法（インスタンス化）

```
class クラス名 {
    public static void main(String[] args) {
        参照型 参照変数名 = new 参照型();
    }
}
```

new の後に指定した参照型のクラスがインスタンス化されます。

・**参照型**とは，インスタンスを参照する変数に指定するデータ型です。
・参照型には，作成済のクラス名を指定します。

参照変数には，インスタンスの参照を代入します。

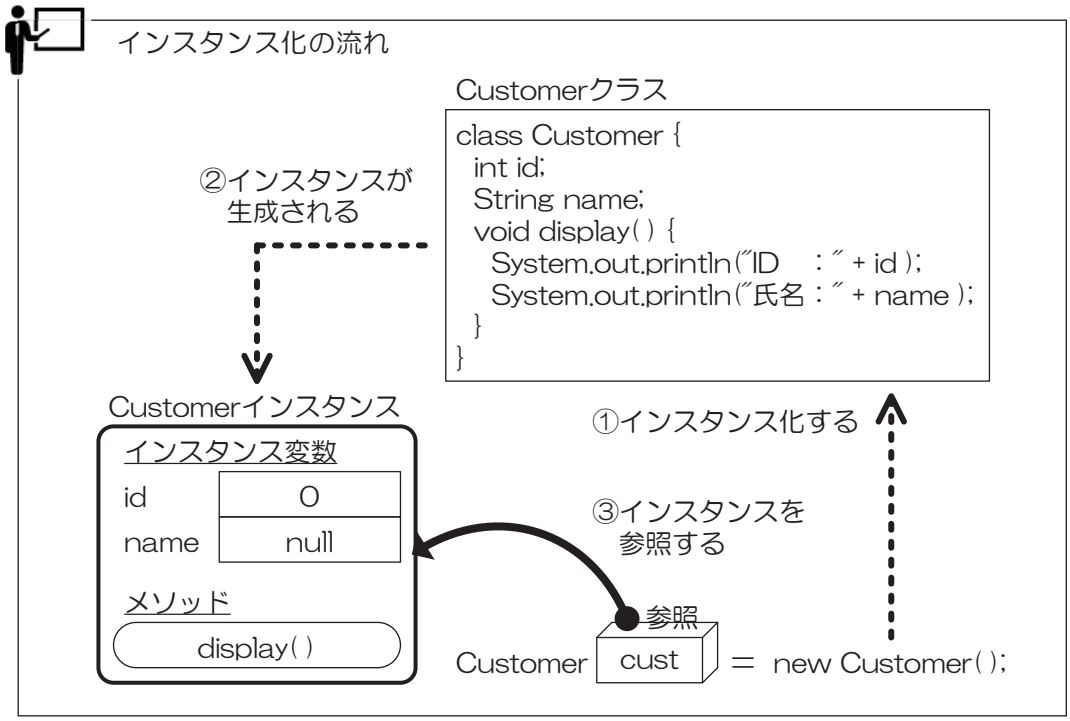
・コンピュータの内部では，インスタンスは主記憶装置上に記憶されます。
・**参照**とは，主記憶装置上のインスタンスの位置情報（アドレス）です。

プログラム例 2-2-1 では，インスタンス化は次に示す３行目です。

Example221.java（抜粋）

```
3          Customer cust = new Customer();
```

Customer クラス型の変数 cust に，Customer クラスのインスタンスの参照を代入しています。



・生成されたインスタンスの各インスタンス変数は，自動的に初期化されます。
・インスタンス化の際に代入される初期値は次のとおりです。

<インスタンス生成時の変数の初期値>

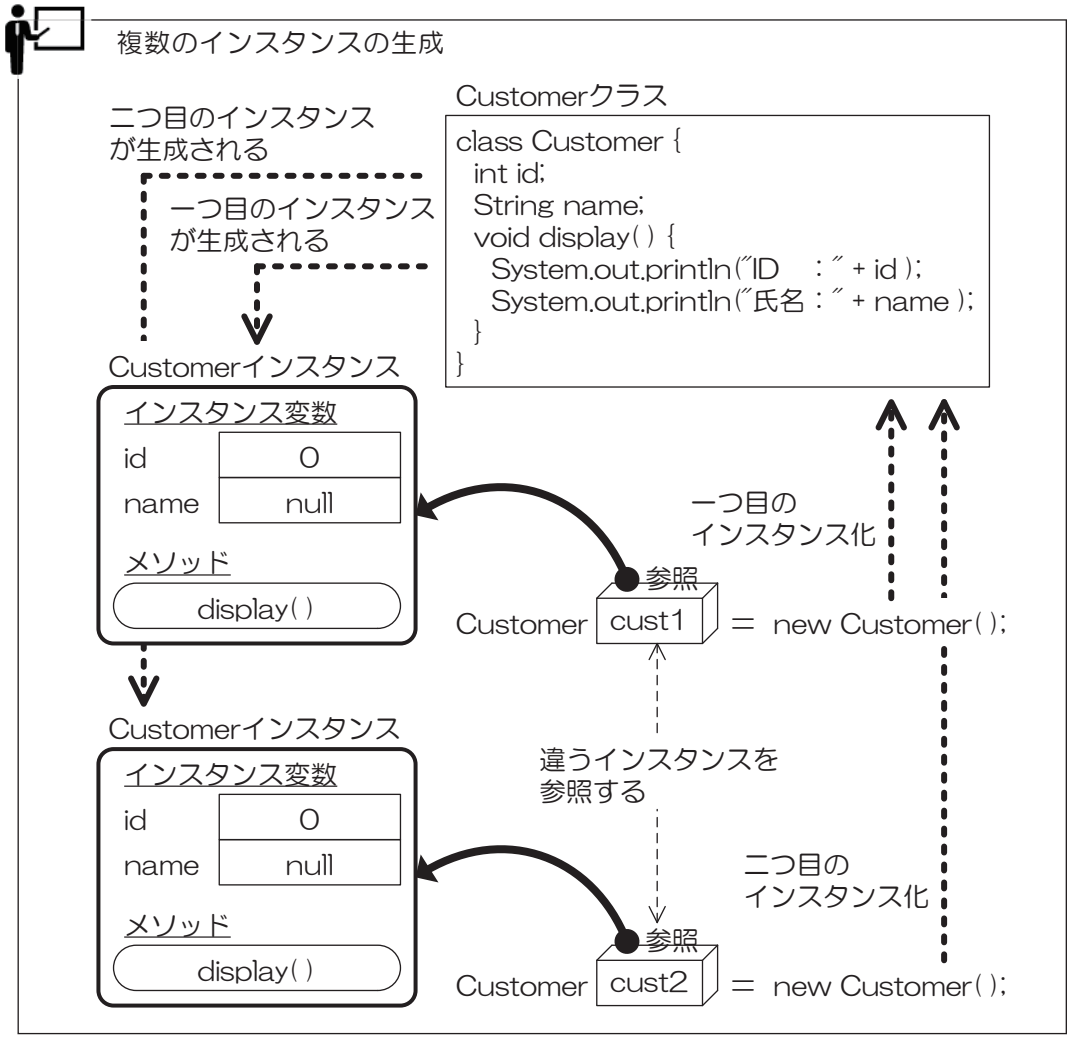
データ型	初期値
整数型（byte，short，int，long）	0
浮動小数点数型（float，double）	0.0
文字型（char）	'¥u0000'
真偽値型（boolean）	false
参照型（String など）	null

(2) 複数のインスタンスの生成

一つのクラスから，複数のインスタンスを生成することができます。

記述例（複数のインスタンスの生成）

```
//一つ目のインスタンスの生成
Customer cust1 = new Customer();
//二つ目のインスタンスの生成
Customer cust2 = new Customer();
```



・生成された二つのインスタンスは，性質の異なる二つのオブジェクトとして扱うことができます。

・インスタンスを参照する変数に，他のインスタンスの参照を代入すると元の参照は消え，後から代入した参照が残ります（上書き）。

練習問題 2-2-1

・練習問題 2-1-2 で定義した Product クラスの，インスタンスを生成するプログラムを作成しなさい。
※保存後，文法エラー（コンパイルエラー）がないことを確認してください。

Product.java（練習問題 2-1-2 で定義済）

```
 1  class Product {
 2      String no;
 3      String name;
 4      int price;
 5      int quantity;
 6      void display() {
 7          int total = price * quantity;
 8          System.out.println("ナンバー：" + no);
 9          System.out.println("製品名　：" + name);
10          System.out.println("価格　　：" + price + "円");
11          System.out.println("数量　　：" + quantity + "個");
12          System.out.println("合計金額：" + total + "円");
13      }
14  }
```

クラス定義
　クラス名　　：　Practice221
　メソッド名　：　main メソッド

main メソッドの処理条件
・Product クラス型の変数 pro に，Product クラスのインスタンスを生成してその参照を代入します。

2-2-2 インスタンスの利用

例題をとおして，インスタンスの利用について学習します。また，static 変数についても学習します。

例題 2-2-2

・次の実行結果になるように，プログラムを作成しなさい。
※プログラム例 2-1-2（Customer.java）を利用して作成します。

実行結果

```
ID  ：3333
氏名：神保　三郎
```

クラス定義

クラス名　　：　Example222

メソッド名　：　main メソッド

Customer.java（プログラム例 2-1-2 で作成済）

```
1  class Customer {
2      int id;
3      String name;
4      void display() {
5          System.out.println("ID  ：" + id);
6          System.out.println("氏名：" + name);
7      }
8  }
```

［プログラム例 2-2-2］

Example222.java

```
1  class Example222 {
2      public static void main(String[] args) {
3          Customer cust = new Customer();
4          cust.id = 3333;
5          cust.name = "神保　三郎";
6          cust.display();
7      }
8  }
```

実行結果

```
ID　：3333
氏名：神保　三郎
```

［作業手順］

① Eclipse の「Java_第２部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第２部/src
名前	Example222
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 2-2-2 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，プログラムを実行します。

例題 2-2-2 解説

(1) インスタンスの利用

参照変数が参照するインスタンスを利用するには，「.（ドット）」を使用します。

記述方法（インスタンスの利用）

参照変数名.インスタンス変数またはメソッドの呼出し

・「.（ドット）」を“～の中の”と読むと，イメージがつかみやすくなります。

● インスタンス変数への値の代入

インスタンス変数に値を代入するには，「参照変数.」に続けてインスタンス変数を指定し，代入式を記述します。

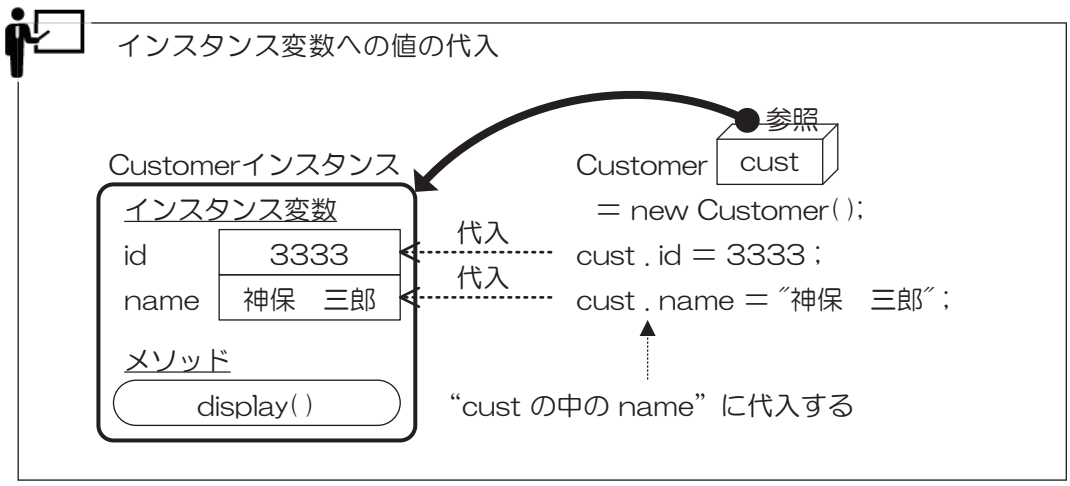
記述方法（インスタンス変数への値の代入）

参照変数名.インスタンス変数名 = 値;

プログラム例 2-2-2 では，インスタンス変数への値の代入は次に示す４，５行目です。

Example222.java（抜粋）

```
4          cust.id = 3333;
5          cust.name = "神保　三郎";
```



● メソッドの呼出し

メソッドを呼び出すには，「参照変数.」に続けてメソッドの呼出しを記述します。

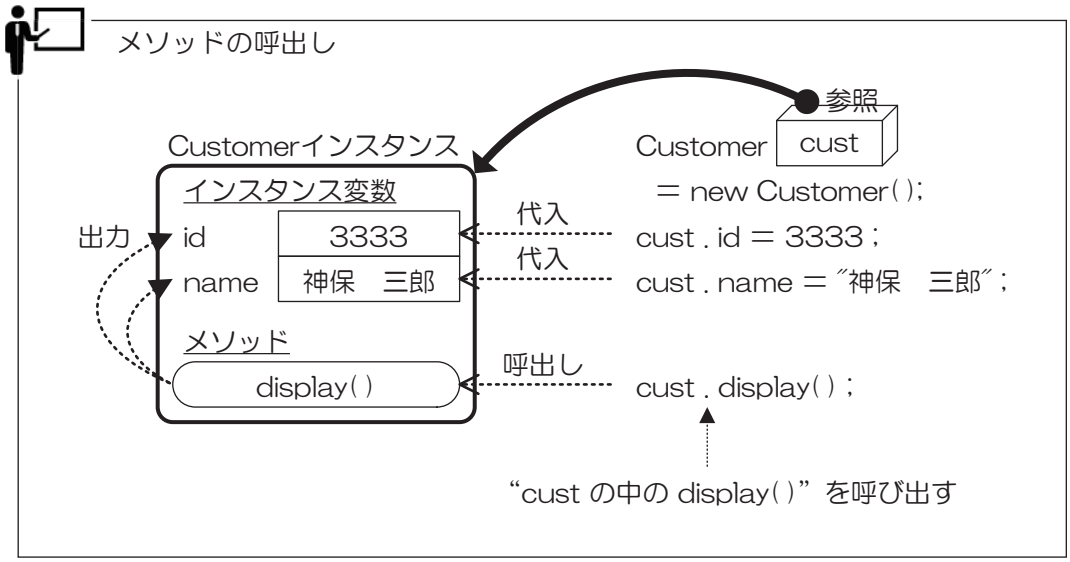
記述方法（メソッドの呼出し（返却値なし））

```
参照変数名.メソッド名(引数);
```

プログラム例 2-2-2 では，メソッドの呼出しは次に示す 6 行目です。

Example222.java（抜粋）

```
6          cust.display();
```



プログラム例 2-2-2 の display メソッドは，引数のないメソッドです。また，返却値もありません。

返却値のあるメソッドの呼出しは，次のようになります。

記述方法（メソッドの呼出し（返却値あり））

```
返却値を受け取る変数名 = 参照変数名.メソッド名(引数);
```

記述例（メソッドの呼出し（返却値あり））

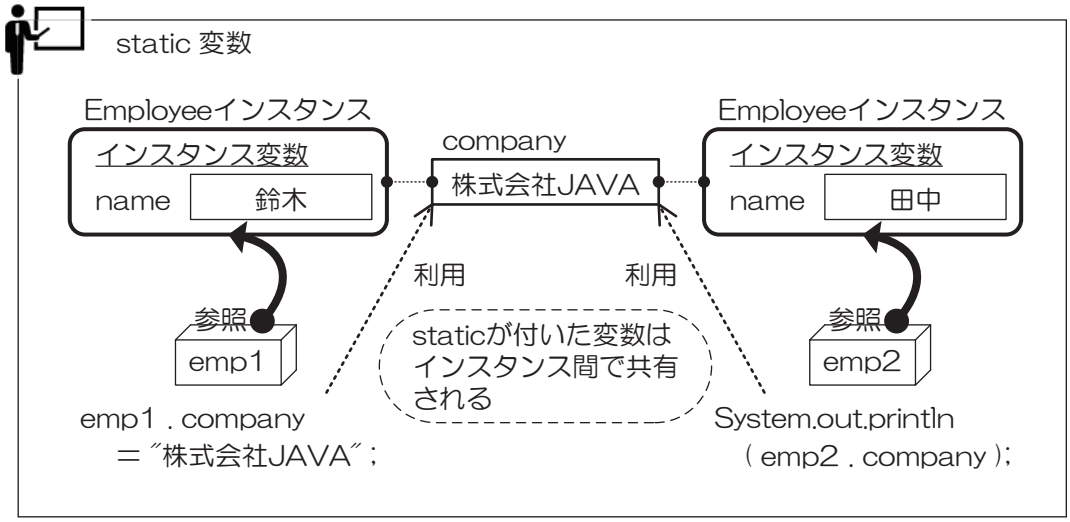
```
Employee emp = new Employee();    //インスタンス化
String name = emp.getName();      //メソッドの呼出し
```

(2) static 変数

static がついた変数は，インスタンス間で共有されます。

記述例（static 変数の宣言）

```
class Employee {
    String name;                //インスタンス固有
    static String company;      //インスタンス間で共有
}
```



・static が付いた変数を **static 変数**または**クラス変数**といいます。

・static が付いていない変数（インスタンス変数）は，インスタンス固有の値を記憶します。

static 変数は，他のクラスから「**クラス名.static 変数名**」で利用することもできます。

記述例（static 変数の利用）

```
System.out.println(Employee.company);  //static 変数 company の値を出力
```

練習問題2-2-2

・次の実行結果になるように，プログラムを作成しなさい。
※練習問題 2-1-2 で定義した「Product.java」を利用します。

実行結果

```
ナンバー：T001
製品名　：トートバッグ
価格　　：8000円
数量　　：5個
合計金額：40000円
```

クラス定義

クラス名　　：　Practice222
メソッド名　：　main メソッド

main メソッドの処理条件

1．Product クラス型の変数 pro に，Product クラスのインスタンスを生成してその参照を代入します。
※Product クラスは，練習問題 2-1-2 で定義済です。
2．変数 pro が参照するインスタンスのインスタンス変数 no に，文字列「T001」を代入します。
3．変数 pro が参照するインスタンスのインスタンス変数 name に，文字列「トートバッグ」を代入します。
4．変数 pro が参照するインスタンスのインスタンス変数 price に，整数「8000」を代入します。
5．変数 pro が参照するインスタンスのインスタンス変数 quantity に，整数「5」を代入します。
6．変数 pro が参照するインスタンスの display メソッドを呼び出します。

第3章 コンストラクタ

2-3-1 コンストラクタの利用

例題をとおして，コンストラクタの定義・呼出しについて学習します。また，コンストラクタのオーバロードについても学習します。

例題2-3-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
口座番号：1234567
口座名義：神田　二郎
```

クラス定義①

Account
int number String name
Account(int nu, String na) void display()

クラス定義②

クラス名　　：　Example231

メソッド名　：　main メソッド

※Account クラスを利用します。

[プログラム例 2-3-1]

Account.java

```
1  class Account {
2      int number;
3      String name;
4      Account(int nu, String na) {
5          number = nu;
6          name = na;
7      }
8      void display() {
9          System.out.println("口座番号：" + number);
10         System.out.println("口座名義：" + name);
11     }
12 }
```

Example231.java

```
1  class Example231 {
2      public static void main(String[] args) {
3          Account ac = new Account(1234567, "神田　二郎");
4          ac.display();
5      }
6  }
```

実行結果

```
口座番号：1234567
口座名義：神田　二郎
```

［作業手順］

① Eclipse の「Java_第２部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第２部/src
名前	Account
修飾子	「パッケージ」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 2-3-1（Account.java）を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の「Java_第２部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第２部/src
名前	Example231
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

④ プログラム例 2-3-1（Example231.java）を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

⑤ Eclipse の ボタンを押して，プログラムを実行します。

例題2-3-1解説

(1) コンストラクタの定義・呼出し

コンストラクタは，クラスがインスタンス化される際に自動的に呼び出される特別なメソッドです。

● コンストラクタの定義

コンストラクタは，インスタンス化されるクラスに定義します。

記述方法（コンストラクタの定義）

```
class クラス名 {
    :
    コンストラクタ名(データ型 引数を受け取る変数名，…) {
        具体的な処理内容
    }
    :
}
```

・コンストラクタ名には，クラス名を指定します。
・コンストラクタは，引数を受け取って決められた処理を実行します。

・コンストラクタの定義では，返却値のデータ型は記述しません。
・コンストラクタは，返却値がありません。

プログラム例2-3-1（Account.java）では，コンストラクタの定義は次に示す４～７行目です。

Account.java（抜粋）

```
4      Account(int nu, String na) {
5          number = nu;
6          name = na;
7      }
```

・コンストラクタの具体的な処理内容に決まりはありません。
・一般的に，インスタンス変数の初期化を行います。

● コンストラクタの呼出し

コンストラクタは，クラスがインスタンス化される際に自動的に呼び出されます。

記述方法（コンストラクタの呼出し）

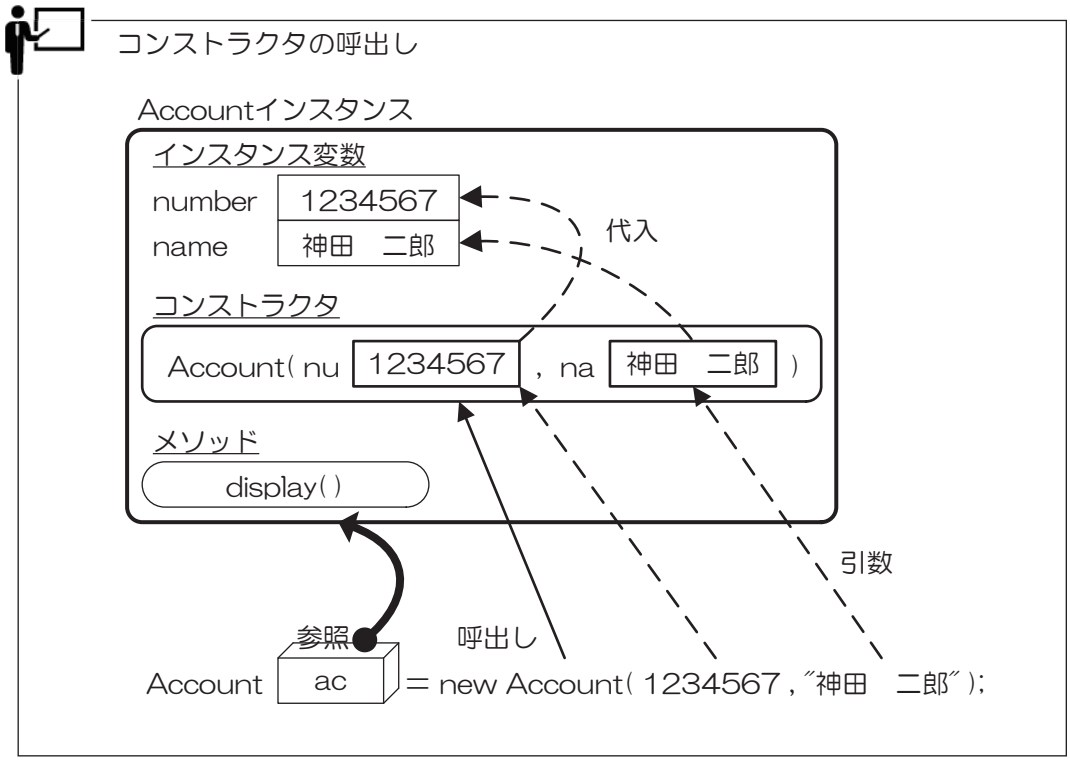
```
参照型 参照変数名 = new コンストラクタ名(引数, …);
```

・コンストラクタ名には，クラス名を指定します。

プログラム例 2-3-1（Example231.java）では，コンストラクタの呼出しは次に示す３行目です。

Example231.java（抜粋）

```
3        Account ac = new Account(1234567, "神田　二郎");
```

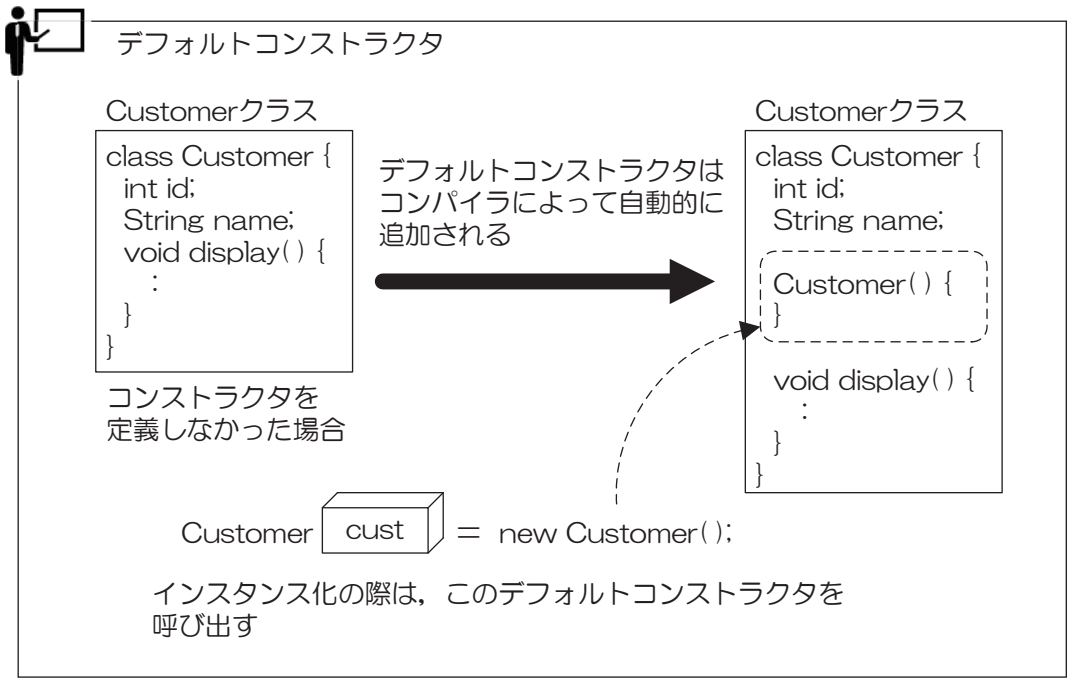


Account クラスのコンストラクタを呼び出す際に，整数「1234567」を第１引数として，文字列「神田　二郎」を第２引数として渡します。コンストラクタは，第１引数を変数 nu で受け取り，インスタンス変数 number に代入します。また，第２引数を変数 na で受け取り，インスタンス変数 name に代入します。

・呼出し元の引数の個数，データ型，順番は，コンストラクタの定義と一致していないといけません。一致しない場合，コンパイルエラーになります。

● デフォルトコンストラクタ

デフォルトコンストラクタは，インスタンス化されるクラスにコンストラクタが定義されていない場合に，自動的に追加されるコンストラクタです。



・デフォルトコンストラクタは，コンパイラによって自動的に追加されます。
・インスタンス化の際は，このデフォルトコンストラクタを呼び出します。

・コンストラクタを定義した場合は，デフォルトコンストラクタは追加されません。

⑵　コンストラクタのオーバロード

コンストラクタは，**オーバロード**（多重定義）することができます。

記述例（コンストラクタのオーバロード）

```
class Account {
    int number;
    String name;
    Account(int nu) {
        number = nu;
        name = "未登録";
    }
    Account(int nu, String na) {
        number = nu;
        name = na;
    }
   :
}
```

記述例では，引数を一つ受け取るコンストラクタと，引数を二つ受け取るコンストラクタをオーバロードしています。

引数を一つ受け取るコンストラクタを使用した場合は，インスタンス変数 number には引数 nu の値が代入されます。インスタンス変数 name には，文字列「未登録」が代入されます。

・オーバロードは，次の条件のいずれかを満たす場合に成立します。
　①引数を受け取る変数の個数が異なる
　②引数を受け取る変数のデータ型が異なる
・オーバロードするコンストラクタの定義の個数に制限はありません。

●　オーバロードされたコンストラクタの呼出し

オーバロードされたコンストラクタの呼出し元は，引数の個数とデータ型が一致するコンストラクタを呼び出します。

記述例（コンストラクタの呼出し）

```
Account ac1 = new Account(9876543);
Account ac2 = new Account(1234567, "神田　二郎");
```

Account クラスのコンストラクタを呼び出す際に，引数を一つだけ指定した場合は，引数を一つ受け取るコンストラクタが呼び出されます。引数を二つ指定した場合は，引数を二つ受け取るコンストラクタが呼び出されます。

練習問題２-３-１

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
書籍名：みんなのJava
著者　：デューク
単価　：3000円
```

クラス定義①

Book
String title String author int price
Book(String t, String a, int p) void display()

コンストラクタの処理条件

・引数 t の値をインスタンス変数 title に代入します。
・引数 a の値をインスタンス変数 author に代入します。
・引数 p の値をインスタンス変数 price に代入します。

display メソッドの処理条件

・インスタンス変数 title，author，price の値をディスプレイに表示します。

クラス定義②

クラス名　　：　Practice231
メソッド名　：　main メソッド

main メソッドの処理条件

１．Book クラスのコンストラクタを呼び出し，生成した Book クラスのインスタンスの参照を Book クラス型の変数 b に代入します。
　コンストラクタに指定する引数は次のとおりです。
　第１引数：　文字列「みんなの Java」
　第２引数：　文字列「デューク」
　第３引数：　整数「3000」
２．変数 b が参照するインスタンスの display メソッドを呼び出します。

2-3-2 this キーワード

例題をとおして，this キーワードについて学習します。

例題 2-3-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
品名：手ごねハンバーグ
単価：590円
```

クラス定義①

Menu
String name int price
Menu(String name, int price) void display()

クラス定義②

クラス名　　：　Example232

メソッド名　：　main メソッド

※Menu クラスを利用します。

[プログラム例 2-3-2]

Menu.java

```
1  class Menu {
2      String name;
3      int price;
4      Menu(String name, int price) {
5          this.name = name;
6          this.price = price;
7      }
8      void display() {
9          System.out.println("品名：" + name);
10         System.out.println("単価：" + price + "円");
11     }
12 }
```

Example232.java

```
1  class Example232 {
2      public static void main(String[] args) {
3          Menu me = new Menu("手ごねハンバーグ", 590);
4          me.display();
5      }
6  }
```

実行結果

```
品名：手ごねハンバーグ
単価：590円
```

［作業手順］

① Eclipse の「Java_第２部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第２部/src
名前	Menu
修飾子	「パッケージ」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 2-3-2（Menu.java）を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の「Java_第２部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第２部/src
名前	Example232
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

④ プログラム例 2-3-2（Example232.java）を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

⑤ Eclipse の ボタンを押して，プログラムを実行します。

例題2-3-2解説

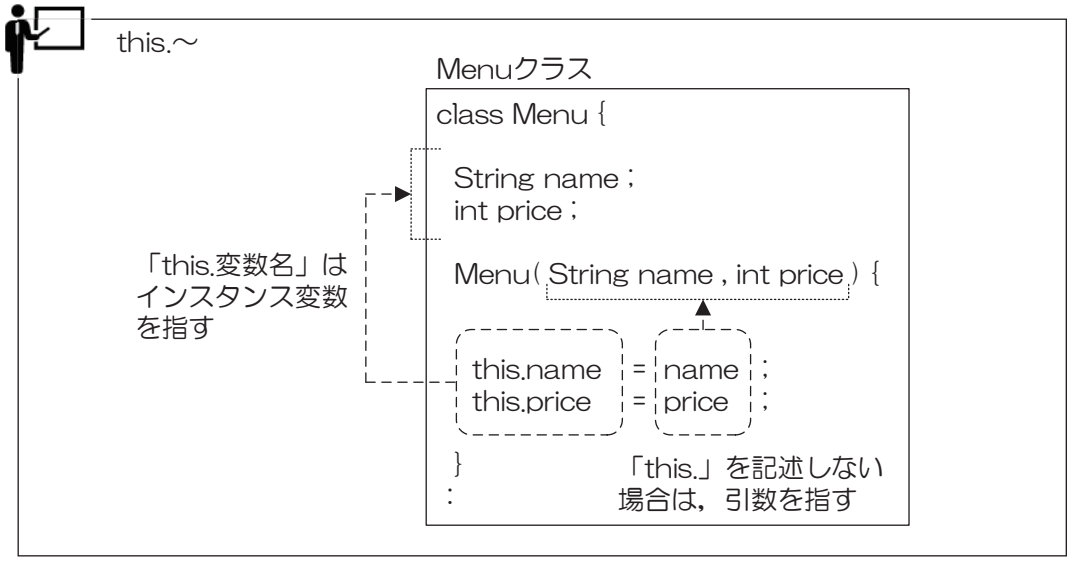
(1) this.～

this は，このクラスのインスタンスを参照している特別な変数です。「this.」に続けてインスタンス変数やメソッドを指定します。

プログラム例 2-3-2（Menu.java）では，this の使用は次に示す５，６行目です。

Menu.java（抜粋）

```
1  class Menu {
2      String name;
3      int price;
4      Menu(String name, int price) {
5          this.name = name;
6          this.price = price;
7      }
```



- 「this.変数名」はインスタンス変数を指します。
- 「this.」を記述せずに変数名を指定した場合は，引数を指します。

- プログラム例 2-3-2（Menu.java）で，「this.」を記述せずに「name = name;」と記述した場合は，引数の値はインスタンス変数に代入されません。
- this は，読取り専用です。

(2) this( )

コンストラクタがオーバロード（多重定義）されている場合は，「this()」を利用して同じクラス内の他のコンストラクタを呼び出すことができます。

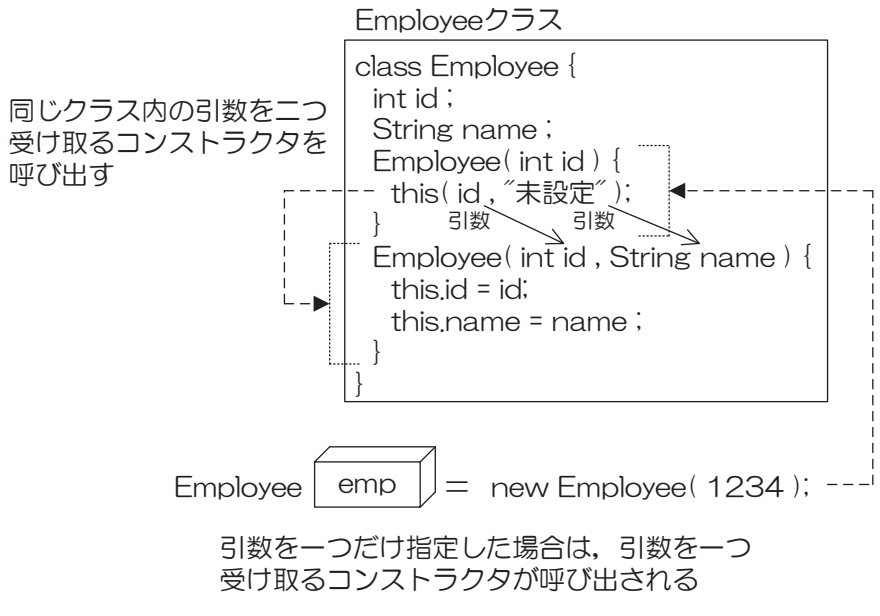
記述方法（this()）

```
this(引数, …);
```

記述例（this()）

```
class Employee {
    int id;
    String name;
    Employee(int id) {
        this(id, "未設定");
    }
    Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }
}
```

this( )



・「this()」は，コンストラクタ内の先頭行に記述します。先頭行以外に記述した場合は，コンパイルエラーになります。

練習問題2-3-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
商品番号：58
商品名　：未登録
単価　　：0円
```

クラス定義①

Item
int no String name int price
① Item(int no) ② Item(int no, String name, int price) void display()

コンストラクタ①の処理条件

・引数 no の値をインスタンス変数 no に代入します。
・文字列「未登録」をインスタンス変数 name に代入します。
・整数「0」をインスタンス変数 price に代入します。

コンストラクタ②の処理条件

・引数 no の値をインスタンス変数 no に代入します。
・引数 name の値をインスタンス変数 name に代入します。
・引数 price の値をインスタンス変数 price に代入します。

display メソッドの処理条件

・インスタンス変数 no，name，price の値をディスプレイに表示します。

クラス定義②

クラス名　　：　Practice232
メソッド名　：　main メソッド

main メソッドの処理条件

1．Item クラスのコンストラクタを呼び出し，生成した Item クラスのインスタンスの参照を Item クラス型の変数 i1 に代入します。
　コンストラクタに指定する引数は次のとおりです。
　引数：　整数「58」
2．変数 i1 が参照するインスタンスの display メソッドを呼び出します。

第4章 アクセスレベルとパッケージング

2-4-1 アクセス修飾子

例題をとおして，アクセス修飾子について学習します。また，JavaBeans 仕様についても学習します。

例題2-4-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
会員番号：7777
```

クラス定義①

Member
- int no
+ Member(int no) + int getNo() + void setNo(int no)

クラス定義②

クラス名　　：　Example241

メソッド名　：　main メソッド

※Member クラスを利用します。

[プログラム例 2-4-1]

Member.java

```
1  class Member {
2      private int no;
3      public Member(int no) {
4          this.no = no;
5      }
6      public int getNo() {
7          return no;
8      }
9      public void setNo(int no) {
10         this.no = no;
11     }
12 }
```

Example241.java

```
1  class Example241 {
2      public static void main(String[] args) {
3          Member mem = new Member(7777);
4          int no = mem.getNo();
5          System.out.println("会員番号：" + no);
6      }
7  }
```

実行結果

```
会員番号：7777
```

［作業手順］

① Eclipse の「Java_第２部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第２部/src
名前	Member
修飾子	「パッケージ」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 2-4-1（Member.java）を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の「Java_第２部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第２部/src
名前	Example241
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

④ プログラム例 2-4-1（Example241.java）を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

⑤ Eclipse の ボタンを押して，プログラムを実行します。

例題2-4-1解説

(1) アクセス修飾子

アクセス修飾子を利用して，インスタンス化されるクラスにアクセス制御を設定することができます。

アクセス修飾子には，次のような種類があります。

<アクセス修飾子>

種類	意味	設定できる要素
public	・全てのクラスからアクセスできる	・クラス ・インスタンス変数 ・static 変数 ・コンストラクタ ・メソッド
protected	・同じパッケージ内のクラスからアクセスできる ・異なるパッケージのサブクラスからアクセスできる ・異なるパッケージのサブクラス以外のクラスからはアクセスできない	・インスタンス変数 ・static 変数 ・コンストラクタ ・メソッド
デフォルト （指定なし）	・同じパッケージ内のクラスからアクセスできる ・異なるパッケージのクラスからはアクセスできない	・クラス ・インスタンス変数 ・static 変数 ・コンストラクタ ・メソッド
private	・同じクラス内からアクセスできる ・他のクラスからはアクセスできない	・インスタンス変数 ・static 変数 ・コンストラクタ ・メソッド

※サブクラスの詳細については，第３部・第１章で解説します。

アクセスレベルは，public → protected → デフォルト → private の順に厳しくなります。

・一般的に，インスタンス変数には private を設定し，コンストラクタ，メソッドには public を設定します。

・プログラム例 2-4-1 の「Member.java」と「Example241.java」は同じパッケージ内にあるため，public が設定されていなくてもアクセスすることができます。
※パッケージの詳細については，次節で解説します。

・UML（統一モデリング言語）に含まれるクラス図の表記法では，public は「+」，protected は「#」，デフォルト（指定なし）は「~」，private は「-」で表します。

● private

private は，同じクラス内からのアクセスだけを許可する修飾子です。

記述方法（private）

```
//インスタンス変数に設定する場合
private String password;
//メソッドに設定する場合
private int total(int data1, int data2) {
  :
}
```

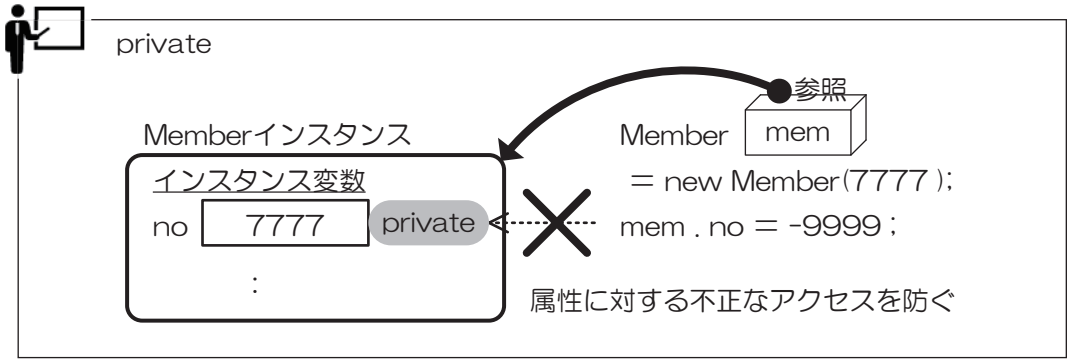
・private は，インスタンス変数，static 変数，コンストラクタ，メソッドに設定することができます。

プログラム例 2-4-1（Member.java）では，private の設定は次に示す２行目です。

Member.java（抜粋）

```
2     private int no;
```

・private が設定されたインスタンス変数，static 変数，コンストラクタ，メソッドには，他のクラスからアクセスすることはできません。



・インスタンス変数に private を設定することにより，オブジェクトの**情報隠蔽**を実現することができます。

● public

public は，全てのクラスからのアクセスを許可する修飾子です。

記述方法（public）

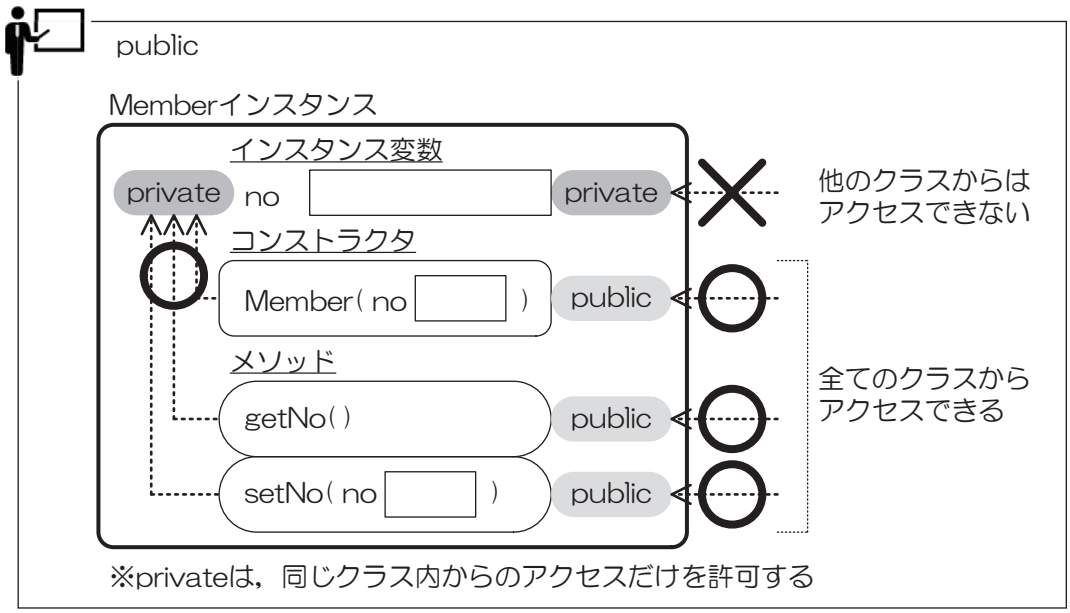
```
//インスタンス変数に設定する場合
public String department = "開発部";
//メソッドに設定する場合
public void display() {
  :
}
```

・public は，クラス，インスタンス変数，static 変数，コンストラクタ，メソッドに設定することができます。

プログラム例 2-4-1（Member.java）では，public の設定は次に示す 3 ～11 行目です。

Member.java（抜粋）

```
 3      public Member(int no) {
 4          this.no = no;
 5      }
 6      public int getNo() {
 7          return no;
 8      }
 9      public void setNo(int no) {
10          this.no = no;
11      }
```



(2) JavaBeans 仕様

JavaBeans 仕様とは，再利用性の高いクラスを定義するために考えられた，コーディング規約の一つです。

● JavaBeans 仕様のルール

JavaBeans 仕様の主なルールは，次のとおりです。

〈JavaBeans 仕様のルール（一部）〉

・クラスのアクセスレベルには，public を設定する。
・クラス名の最後には，「Bean」を付ける。
・インスタンス変数のアクセスレベルには，private を設定する。
・引数のないコンストラクタを定義する。
・コンストラクタのアクセスレベルには，public を設定する。
・インスタンス変数の値を取得する **getter（ゲッタ）**を定義する。
　getter 名は，インスタンス変数名の先頭を大文字にして「get」を付ける。
　（例）　getName
・インスタンス変数に値を設定する **setter（セッタ）**を定義する。
　setter 名は，インスタンス変数名の先頭を大文字にして「set」を付ける。
　（例）　setName
・getter および setter のアクセスレベルには，public を設定する。

・getter および setter は，インスタンス変数にアクセスするためのメソッドです。
・getter および setter をあわせて**アクセッサ**といいます。

記述例（JavaBeans 仕様）

```
public class EmployeeBean {
    private int id;
    public EmployeeBean() {
    }
    public EmployeeBean(int id) {
        this.id = id;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
}
```

練習問題2-4-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
メールアドレス > awaji@java.co.jp
パスワード     > qwerty
メールアドレス：awaji@java.co.jp
パスワード     ：qwerty
```

クラス定義①

Login
- String mail - String pass
+ Login(String mail, String pass) + String getMail() + String getPass()

コンストラクタの処理条件

・引数 mail の値をインスタンス変数 mail に代入します。
・引数 pass の値をインスタンス変数 pass に代入します。

getMail メソッドの処理条件

・インスタンス変数 mail の値を返します。

getPass メソッドの処理条件

・インスタンス変数 pass の値を返します。

クラス定義②

クラス名　　：　Practice241
インポート　：　java.util.Scanner
メソッド名　：　main メソッド

main メソッドの処理条件

1．文字列「メールアドレス ＞ 」を，ディスプレイに表示します。
2．変数 mail1 に，キーボードから読み込んだ文字列を代入します。
3．文字列「パスワード　　＞ 」を，ディスプレイに表示します。
4．変数 pass1 に，キーボードから読み込んだ文字列を代入します。
5．Login クラスのコンストラクタを呼び出し，生成した Login クラスのインスタンスの参照を Login クラス型の変数 log に代入します。
　コンストラクタに指定する引数は次のとおりです。
　第１引数：　変数 mail1
　第２引数：　変数 pass1
6．変数 log が参照するインスタンスの getMail メソッドを呼び出し，返却値を変数 mail2 に代入します。
7．変数 log が参照するインスタンスの getPass メソッドを呼び出し，返却値を変数 pass2 に代入します。
8．変数 mail2 と pass2 の値を，ディスプレイに表示します。ただし，例外が発生した場合は，文字列「エラーが発生しました！」をディスプレイに表示します。

2-4-2 パッケージング

例題をとおして，パッケージングについて学習します。また，パッケージングされたクラスの利用についても学習します。

例題2-4-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
ギフトコード：1242
ギフト名　　：バラのアレンジメント
```

クラス定義①

パッケージ名　：　data

Gift
- int code - String name
+ Gift(int code, String name) + int getCode() + String getName()

クラス定義②

クラス名　　：　Example242

インポート　：　data.Gift

メソッド名　：　main メソッド

※Gift クラスを利用します。

[プログラム例 2-4-2]

Gift.java

```
1  package data;
2  public class Gift {
3      private int code;
4      private String name;
5      public Gift(int code, String name) {
6          this.code = code;
7          this.name = name;
8      }
9      public int getCode() {
10         return code;
11     }
12     public String getName() {
13         return name;
14     }
15 }
```

Example242.java

```
1  import data.Gift;
2  class Example242 {
3      public static void main(String[] args) {
4          Gift gift = new Gift(1242, "バラのアレンジメント");
5          int code = gift.getCode();
6          String name = gift.getName();
7          System.out.println("ギフトコード：" + code);
8          System.out.println("ギフト名　　：" + name);
9      }
10 }
```

実行結果

```
ギフトコード：1242
ギフト名　　：バラのアレンジメント
```

［作業手順］

① Eclipse の「Java_第２部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第２部/src
パッケージ	data
名前	Gift
修飾子	「public」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 2-4-2（Gift.java）を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の「Java_第２部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第２部/src
名前	Example242
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

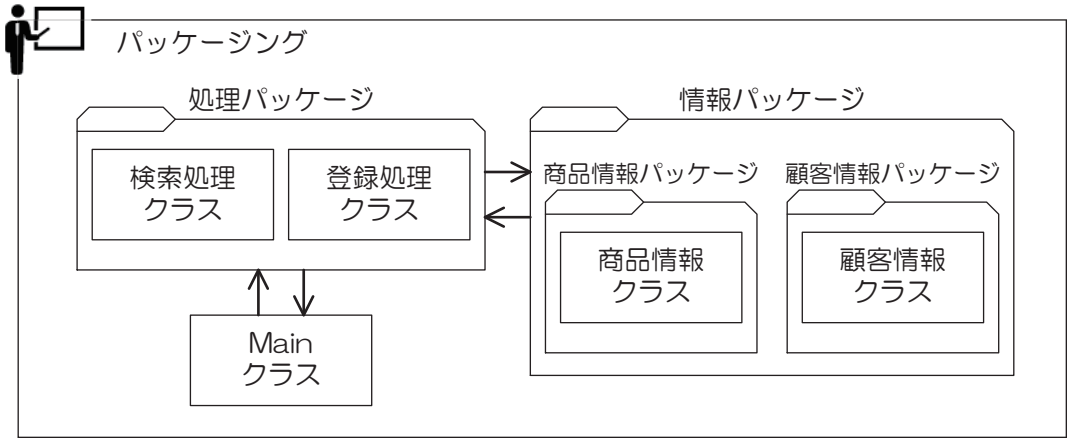
④ プログラム例 2-4-2（Example242.java）を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

⑤ Eclipse の ボタンを押して，プログラムを実行します。

例題 2-4-2 解説

(1) パッケージング

階層構造を作成してクラスを管理する仕組みを，**パッケージング**といいます。



・一般的に，クラスはパッケージングして使用します。
・パッケージングは，クラスの役割を考慮して行います。

● package 文

クラスをパッケージングするには，**package 文**を記述します。

記述方法（package 文）

```
package パッケージ名;
//サブパッケージを作成する場合
package パッケージ名.サブパッケージ名;
```

・package 文は，プログラムの先頭行に記述します。
・パッケージ名には，任意の名前を記述します。予約されているキーワードは，パッケージ名として使うことはできません。
・一般的に，パッケージ名は小文字にします。

プログラム例 2-4-2（Gift.java）では，package 文は次に示す 1 行目です。

Gift.java（抜粋）

```
1  package data;
```

・package 文を記述したクラスは，パッケージと同じ名前のディレクトリに配置します。
・パッケージングされたクラスに public が指定されていない場合は，他のパッケージから利用することはできません。

(2) パッケージングされたクラスの利用

パッケージングされたクラスを利用する場合は，クラスをインポートします。

● import 文

クラスをインポートするには，import 文を記述します。

記述方法（import 文）

```
import パッケージ名.クラス名;
//サブパッケージがある場合
import パッケージ名.サブパッケージ名.クラス名;
```

・import 文は，クラスの定義文の前に記述します。
・import 文は，複数行記述することができます。

プログラム例 2-4-2（Example242.java）では，import 文は次に示す 1 行目です。

Example242.java（抜粋）

```
1  import data.Gift;
```

● 同じパッケージ内の複数のクラスの利用

同じパッケージ内の複数のクラスを利用する場合は，「**＊（アスタリスク）**」を使ってクラス名を省略することができます。

記述方法（同じパッケージ内の複数のクラスの利用）

```
import パッケージ名.*;
//サブパッケージがある場合
import パッケージ名.サブパッケージ名.*;
```

記述例（同じパッケージ内の複数のクラスの利用）

```
import data.*;
```

・「＊（アスタリスク）」は，“パッケージに含まれるすべてのクラス”を意味します。

・「＊（アスタリスク）」を使って，パッケージ名やサブパッケージ名を省略することはできません。

練習問題2-4-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
社員ID > 1000
社員名 > 田中一郎
社員ID：1000
社員名：田中一郎
```

クラス定義①

パッケージ名　：　model

Employee
- int id - String name
+ Employee(int id, String name) + int getId() + String getName()

クラス定義②

クラス名　　：　Practice242
インポート　：　model.Employee
　　java.util.Scanner
メソッド名　：　main メソッド

コンストラクタの処理条件

・引数 id の値をインスタンス変数 id に代入します。
・引数 name の値をインスタンス変数 name に代入します。

getId メソッドの処理条件

・インスタンス変数 id の値を返します。

getName メソッドの処理条件

・インスタンス変数 name の値を返します。

main メソッドの処理条件

1．文字列「社員 ID > 」を，ディスプレイに表示します。
2．変数 id1 に，キーボードから読み込んだ整数を代入します。
3．文字列「社員名 > 」を，ディスプレイに表示します。
4．変数 name1 に，キーボードから読み込んだ文字列を代入します。
5．Employee クラスのコンストラクタを呼び出し，生成した Employee クラスのインスタンスの参照を Employee クラス型の変数 emp に代入します。
　コンストラクタに指定する引数は次のとおりです。
　第１引数：　変数 id1
　第２引数：　変数 name1
6．変数 emp が参照するインスタンスの getId メソッドを呼び出し，返却値を変数 id2 に代入します。
7．変数 emp が参照するインスタンスの getName メソッドを呼び出し，返却値を変数 name2 に代入します。
8．変数 id2 と name2 の値を，ディスプレイに表示します。ただし，例外が発生した場合は，文字列「エラーが発生しました！」をディスプレイに表示します。

第5章 参照型の一次元配列とArrayListクラス

デュークのアドバイス

配列とArrayListでインスタンスを一括管理！
イメージしながらスキルアップ！

2-5-1 参照型の一次元配列

例題をとおして，参照型の一次元配列について学習します。また，一次元配列の各要素のインスタンスの利用についても学習します。

例題2-5-1

・次の実行結果になるように，プログラムを作成しなさい。
※プログラム例2-4-2（Gift.java）を利用して作成します。

実行結果

```
ギフトコード：1134
ギフト名　　：カーネーションのブーケ
ギフトコード：1242
ギフト名　　：バラのアレンジメント
```

クラス定義①

パッケージ名　：　data

Gift
- int code - String name
+ Gift(int code, String name) + int getCode() + String getName()

クラス定義②

クラス名　　：　Example251

インポート　：　data.Gift

メソッド名　：　main メソッド

※Gift クラスを利用します。

［プログラム例 2-5-1］

Example251.java

```
 1 import data.Gift;
 2 class Example251 {
 3     public static void main(String[] args) {
 4         Gift[] aryGift = new Gift[2];
 5         aryGift[0] = new Gift(1134, "カーネーションのブーケ");
 6         aryGift[1] = new Gift(1242, "バラのアレンジメント");
 7         int code1 = aryGift[0].getCode();
 8         String name1 = aryGift[0].getName();
 9         int code2 = aryGift[1].getCode();
10         String name2 = aryGift[1].getName();
11         System.out.println("ギフトコード：" + code1);
12         System.out.println("ギフト名　　：" + name1);
13         System.out.println("ギフトコード：" + code2);
14         System.out.println("ギフト名　　：" + name2);
15     }
16 }
```

［作業手順］

① Eclipse の「Java_第２部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第２部/src
名前	Example251
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 2-5-1 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，プログラムを実行します。

例題2-5-1解説

(1) 参照型の一次元配列

クラスから生成される複数のインスタンスを，一次元配列で扱うことができます。

● 参照型の一次元配列インスタンスの生成

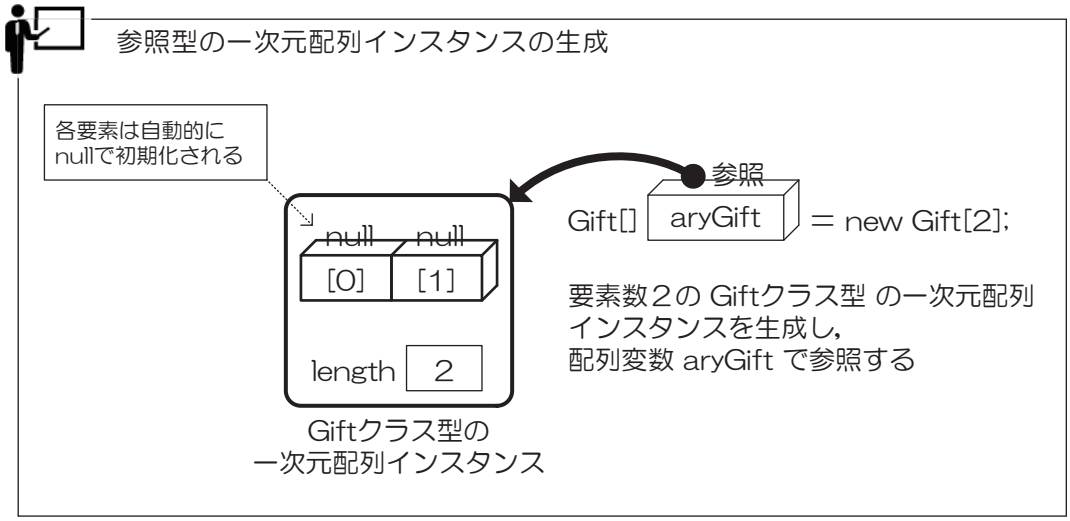
記述方法（参照型の一次元配列インスタンスの生成）

```
参照型[] 配列変数名 = new 参照型[要素数];
```

プログラム例 2-5-1 では，参照型の一次元配列インスタンスの生成は次に示す４行目です。

Example251.java（抜粋）

```
4          Gift[] aryGift = new Gift[2];
```



・生成された参照型の一次元配列インスタンスの各要素は，クラスから生成したインスタンスを参照します。
・生成された参照型の一次元配列インスタンスの各要素は，自動的に null で初期化されます。

・参照型の一次元配列インスタンスを生成しただけでは，各要素が参照するインスタンスは生成されません。

● 一次元配列へのインスタンスの参照の代入

一次元配列インスタンスの各要素に，クラスから生成したインスタンスの参照を代入（上書き）するには，添字を指定してインスタンスの生成式を記述します。

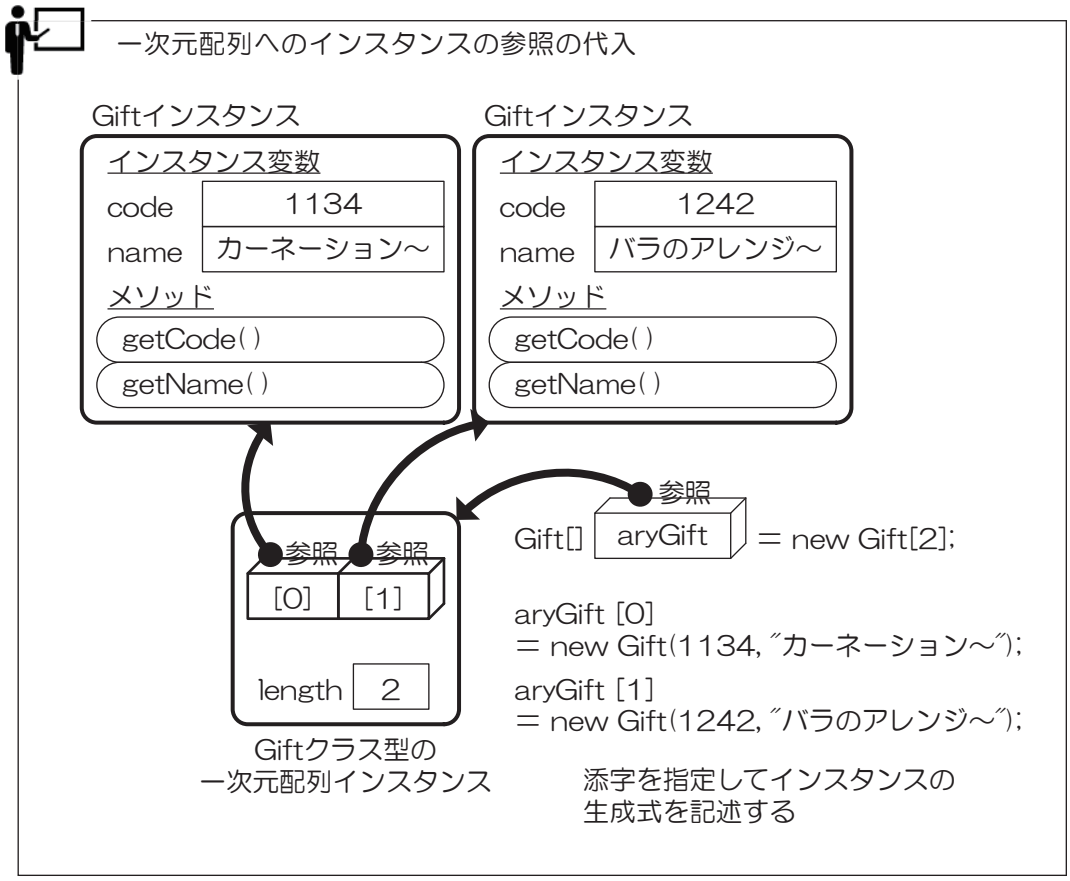
記述方法（一次元配列へのインスタンスの参照の代入）

```
配列変数名[添字] = new 参照型(引数, …);
```

プログラム例 2-5-1 では，一次元配列へのインスタンスの参照の代入は次に示す５，６行目です。

Example251.java（抜粋）

```
5          aryGift[0] = new Gift(1134, "カーネーションのブーケ");
6          aryGift[1] = new Gift(1242, "バラのアレンジメント");
```



Gift クラスのコンストラクタを呼び出してインスタンスを生成し，その参照を Gift クラス型の一次元配列インスタンスに代入します。

・new は，一次元配列インスタンスの生成と，クラスからのインスタンスの生成で使用します。new の意味を間違えないように，注意してください。

(2) 一次元配列の各要素のインスタンスの利用

一次元配列インスタンスの各要素が参照するインスタンスは，添字を指定することで利用することができます。

メソッドを呼び出す方法は，次のとおりです。

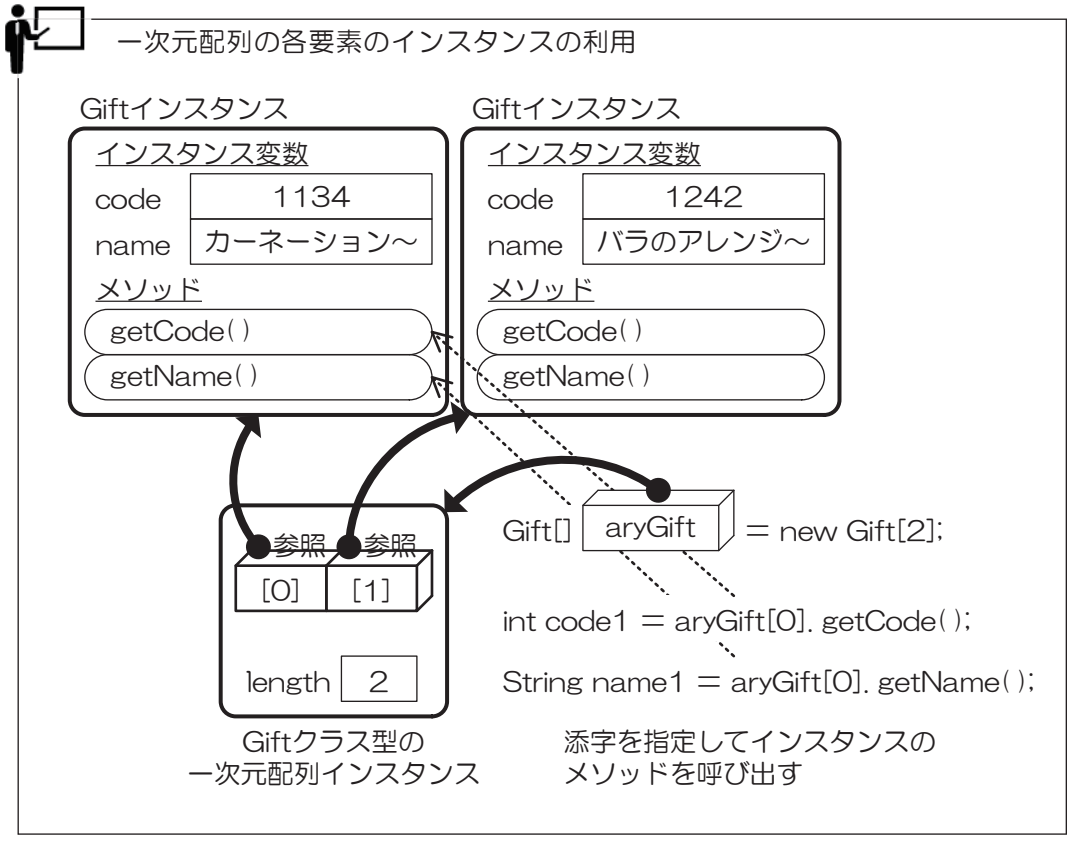
記述方法（一次元配列の各要素のインスタンスの利用）

```
//返却値がないメソッドの場合
配列変数名[添字].メソッド名(引数，…);
//返却値があるメソッドの場合
返却値を受け取る変数名 = 配列変数名[添字].メソッド名(引数，…);
```

プログラム例 2-5-1 では，一次元配列の各要素のインスタンスの利用は次に示す 7 ～ 10 行目です。

Example251.java（抜粋）

```
 7          int code1 = aryGift[0].getCode();
 8          String name1 = aryGift[0].getName();
 9          int code2 = aryGift[1].getCode();
10          String name2 = aryGift[1].getName();
```



● 各要素のインスタンスを扱う for 文

for 文のカウンタを利用して，一次元配列インスタンスの各要素が参照するインスタンスを扱うことができます。

プログラム例 2-5-1 で for 文を使うと，次のようになります。

記述例（各要素のインスタンスを扱う for 文）

```
for(int i = 0; i < aryGift.length; i++) {
    int code = aryGift[i].getCode();
    String name = aryGift[i].getName();
    System.out.println("ギフトコード：" + code);
    System.out.println("ギフト名　　：" + name);
}
```

また，プログラム例 2-5-1 で拡張 for 文を使うと，次のようになります。

記述例（各要素のインスタンスを扱う拡張 for 文）

```
for(Gift gift : aryGift) {
    int code = gift.getCode();
    String name = gift.getName();
    System.out.println("ギフトコード：" + code);
    System.out.println("ギフト名　　：" + name);
}
```

練習問題2-5-1

・次の実行結果になるように，プログラムを作成しなさい。
※練習問題 2-4-2 で作成した「Employee.java」を利用して作成します。

実行結果

```
社員ID：1111
社員名：神田　二郎
社員ID：2222
社員名：淡路　花子
社員ID：3333
社員名：神保　三郎
```

クラス定義①

パッケージ名　：　model

Employee
- int id - String name
+ Employee(String id, String name) + int getId() + String getName()

クラス定義②

クラス名　　：　Practice251
インポート　：　model.Employee
メソッド名　：　main メソッド

main メソッドの処理条件

１．要素数３の Employee クラス型の一次元配列インスタンスを生成し，生成した一次元配列インスタンスの参照を配列変数 aryEmp に代入します。

２．Employee クラスのコンストラクタを呼び出し，生成した Employee クラスのインスタンスの参照を配列要素 aryEmp[0]に代入します。
　コンストラクタに指定する引数は次のとおりです。
　第１引数：　整数「1111」
　第２引数：　文字列「神田　二郎」

３．Employee クラスのコンストラクタを呼び出し，生成した Employee クラスのインスタンスの参照を配列要素 aryEmp[1]に代入します。
　コンストラクタに指定する引数は次のとおりです。
　第１引数：　整数「2222」
　第２引数：　文字列「淡路　花子」

４．Employee クラスのコンストラクタを呼び出し，生成した Employee クラスのインスタンスの参照を配列要素 aryEmp[2]に代入します。
　コンストラクタに指定する引数は次のとおりです。
　第１引数：　整数「3333」
　第２引数：　文字列「神保　三郎」

５．カウンタ i の値が，“０～配列変数 aryEmp が参照する一次元配列インスタンスの要素数未満”の間，次の処理を繰り返します。
　・配列要素 aryEmp[i]が参照するインスタンスの getId メソッドを呼び出し，返却値を変数 id に代入します。
　・配列要素 aryEmp[i]が参照するインスタンスの getName メソッドを呼び出し，返却値を変数 name に代入します。
　・変数 id と name の値を，ディスプレイに表示します。

2-5-2 ArrayList クラス

例題をとおして，ArrayList クラスについて学習します。また，ArrayList インスタンスの操作についても学習します。

例題 2-5-2

・次の実行結果になるように，プログラムを作成しなさい。
※プログラム例 2-4-2（Gift.java）を利用して作成します。

実行結果

```
ギフトコード：594
ギフト名　　：キキョウの花束
ギフトコード：954
ギフト名　　：ユリのブーケ
```

クラス定義①

パッケージ名　：　data

Gift
- int code - String name
+ Gift(int code, String name) + int getCode() + String getName()

クラス定義②

クラス名　　：　Example252

インポート　：　data.Gift

メソッド名　：　main メソッド

※Gift クラスを利用します。

［プログラム例 2-5-2］

Example252.java

```
1  import java.util.ArrayList;
2  import data.Gift;
3  class Example252 {
4      public static void main(String[] args) {
5          ArrayList<Gift> list = new ArrayList<Gift>();
6          Gift gift1 = new Gift(594, "キキョウの花束");
7          Gift gift2 = new Gift(954, "ユリのブーケ");
8          list.add(gift1);
9          list.add(gift2);
10         for(int i = 0; i < list.size(); i++) {
11             Gift gift = list.get(i);
12             System.out.println("ギフトコード：" + gift.getCode());
13             System.out.println("ギフト名　　：" + gift.getName());
14         }
15     }
16 }
```

［作業手順］

① Eclipse の「Java_第２部」プロジェクトの「src」階層に，新規 Java クラスを作成します。

〈「新規 Java クラス」ウィンドウで入力する値〉

入力項目	入力値
ソース・フォルダー	Java_第２部/src
名前	Example252
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 2-5-2 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

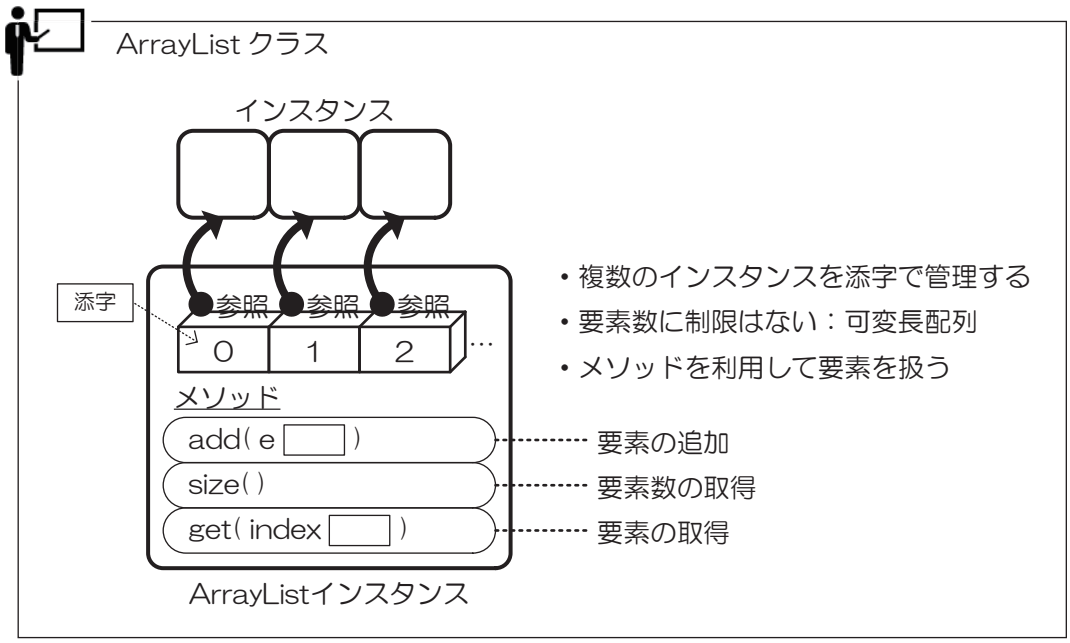
③ Eclipse の ボタンを押して，プログラムを実行します。

例題 2-5-2 解説

(1) ArrayList クラス

ArrayList クラスは，複数のインスタンスを**可変長配列**として管理するために用意されたリスト構造を実現するクラスです。

ArrayList クラスは，インスタンス化して利用します。



・ArrayList クラスは，Java 言語に元々用意されています。
・ArrayList インスタンス内で管理する要素数に制限はありません。

・ArrayList インスタンス内で扱うことができる要素は，クラスから生成されたインスタンスだけです。基本データ型の値を扱うことはできません。

● ArrayList クラスのインポート

ArrayList クラスを利用するには，import 文を記述します。

プログラム例 2-5-2 では，ArrayList クラスのインポートは次に示す 1 行目です。

Example252.java（抜粋）

```
1  import java.util.ArrayList;
```

・ArrayList クラスは，**java.util パッケージ**に含まれています。
・java.util パッケージは，Java 言語に元々用意されています。

● ArrayList インスタンスの生成

ArrayList クラスをインスタンス化するには，次のように記述します。

記述方法（ArrayList インスタンスの生成）

```
ArrayList<参照型> 参照変数名 = new ArrayList<参照型>();
```

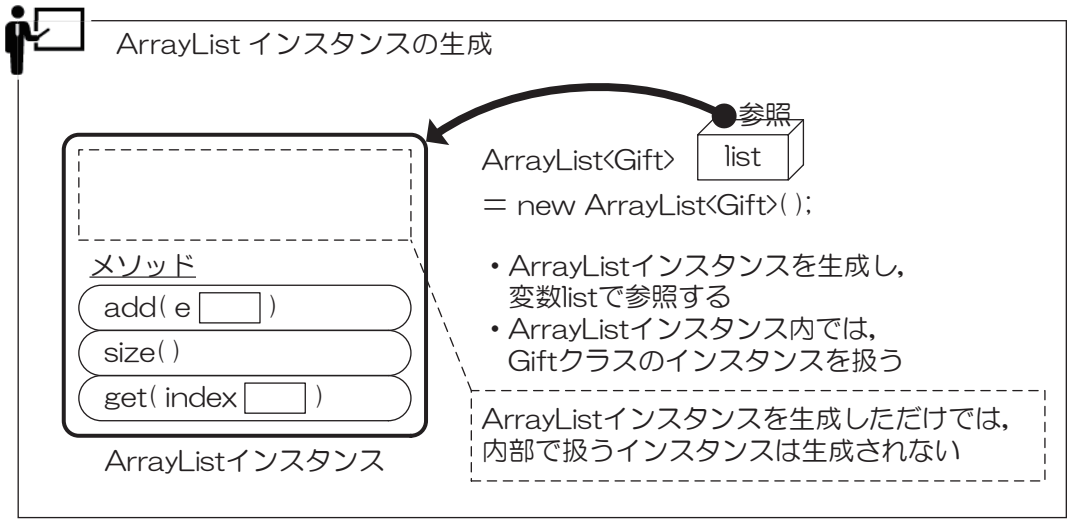
・「<参照型>」には，ArrayList インスタンス内で扱うインスタンスのデータ型を指定します。

・「<」と「>」の間に，基本データ型を指定することはできません。
・「<参照型>」に指定したデータ型以外のデータは，ArrayList インスタンス内では扱うことができません。

プログラム例 2-5-2 では，ArrayList インスタンスの生成は次に示す 5 行目です。

Example252.java（抜粋）

```
5          ArrayList<Gift> list = new ArrayList<Gift>();
```



・ArrayList クラスには，引数なしのコンストラクタが定義されています。上記のインスタンス化の際は，引数なしのコンストラクタが呼び出されます。

・ArrayList インスタンスを生成しただけでは，ArrayList インスタンス内で扱うインスタンスは生成されません。

(2) ArrayList インスタンスの操作

ArrayList インスタンスは，ArrayList クラスに定義されたメソッドを利用して操作します。

● ArrayList インスタンスへの要素の追加

ArrayList インスタンスに要素を追加するには，**add メソッド**を使用します。

記述方法（ArrayList インスタンスへの要素の追加）

```
ArrayList インスタンスの参照変数名.add(インスタンスの参照);
```

プログラム例 2-5-2 では，ArrayList インスタンスへの要素の追加は次に示す８，９行目です。

Example252.java（抜粋）

```
 8          list.add(gift1);
 9          list.add(gift2);
```

プログラム例 2-5-2 では，add メソッドの引数に Gift インスタンスの参照変数を指定しています。

● ArrayList インスタンスの要素数の取得

ArrayList インスタンスの要素数を取得するには，**size メソッド**を使用します。

記述方法（ArrayList インスタンスの要素数の取得）

```
int 返却値を受け取る変数名 = ArrayList インスタンスの参照変数名.size();
```

プログラム例 2-5-2 では，ArrayList インスタンスからの要素の取得は次に示す 10 行目です。

Example252.java（抜粋）

```
10          for(int i = 0; i < list.size(); i++) {
```

プログラム例 2-5-2 では，size メソッドの返却値を受け取る変数は使用せず，返却値は継続条件で直接利用しています。

● ArrayList インスタンスからの要素の取得

ArrayList インスタンスから要素を取得するには，**get メソッド**を使用します。

記述方法（ArrayList インスタンスからの要素の取得）

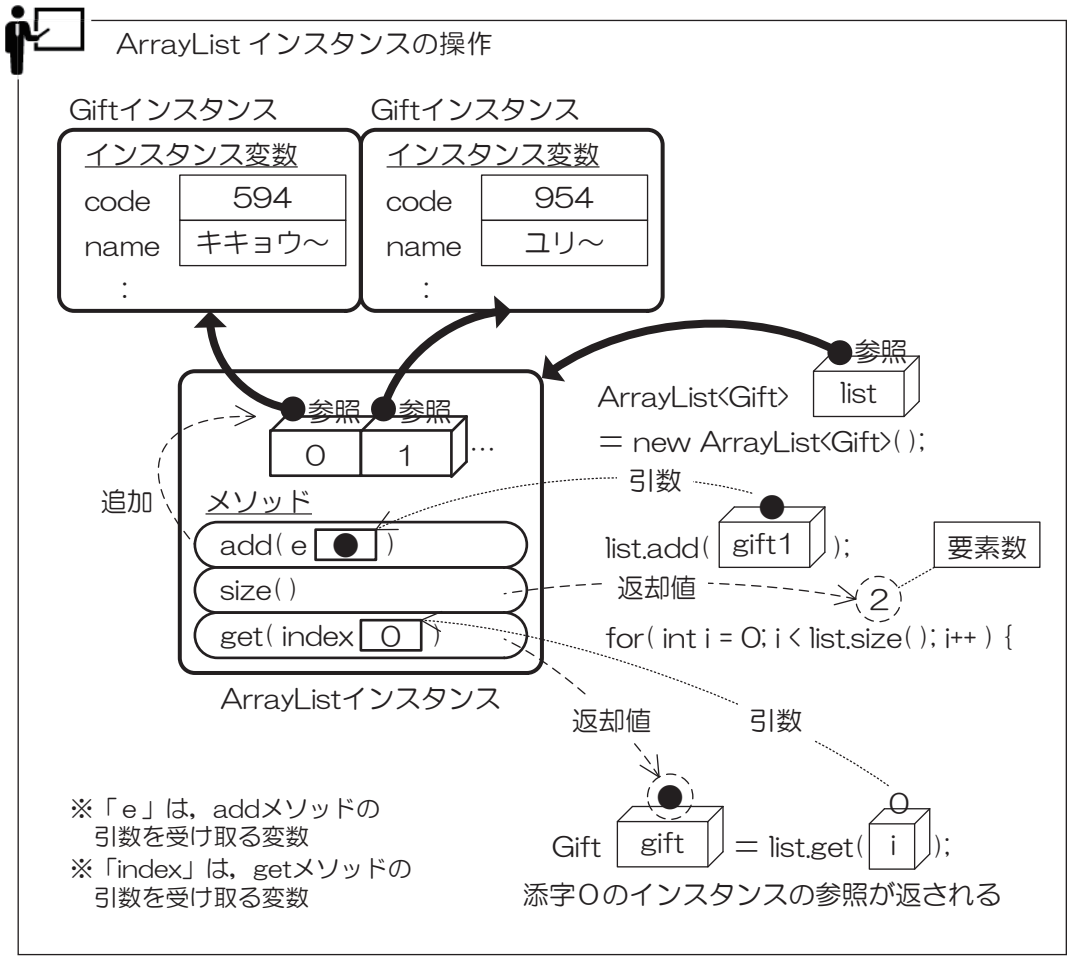
```
参照型 参照変数名 = ArrayList インスタンスの参照変数名.get(添字);
```

プログラム例 2-5-2 では，ArrayList インスタンスからの要素の取得は次に示す 11 行目です。

Example252.java（抜粋）

```
11              Gift gift = list.get(i);
```

プログラム例 2-5-2 では，get メソッドの引数にカウンタ i の値を添字として指定しています。



・ArrayList クラスには，上記以外にも多くのメソッドが定義されています。

練習問題2-5-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
注文番号：780
顧客名　：神田　二郎
注文番号：800
顧客名　：淡路　花子
注文番号：813
顧客名　：神保　三郎
```

クラス定義①

パッケージ名　：　data

Order
- int no - String custName
+ Order(int no, String custName) + int getNo() + String getCustName()

コンストラクタの処理条件

・引数 no の値をインスタンス変数 no に代入します。

・引数 custName の値をインスタンス変数 custName に代入します。

getNo メソッドの処理条件

・インスタンス変数 no の値を返します。

getCustName メソッドの処理条件

・インスタンス変数 custName の値を返します。

クラス定義②

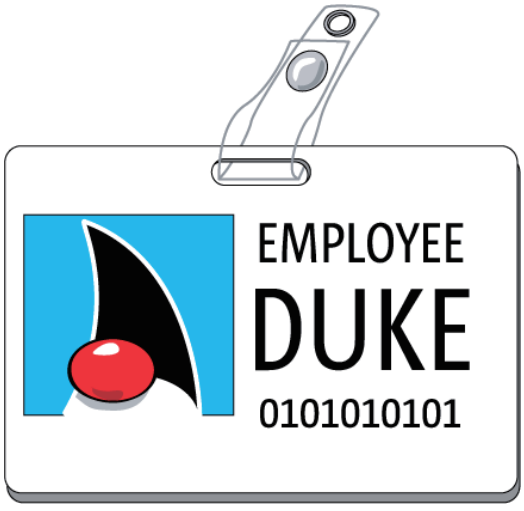
クラス名　　：　Practice252
インポート　：　java.util.ArrayList
　　data.Order
メソッド名　：　main メソッド

main メソッドの処理条件

１．ArrayList クラスのコンストラクタを呼び出し，生成した ArrayList クラスのインスタンスの参照を変数 list に代入します。
　ArrayList インスタンス内で扱うデータ型は，次のとおりです。
　データ型：　Order
２．Order クラスのコンストラクタを呼び出し，生成した Order クラスのインスタンスの参照を変数 o1 に代入します。
　コンストラクタに指定する引数は次のとおりです。
　第１引数：　整数「780」
　第２引数：　文字列「神田　二郎」
３．Order クラスのコンストラクタを呼び出し，生成した Order クラスのインスタンスの参照を変数 o2 に代入します。
　コンストラクタに指定する引数は次のとおりです。
　第１引数：　整数「800」
　第２引数：　文字列「淡路　花子」
４．Order クラスのコンストラクタを呼び出し，生成した Order クラスのインスタンスの参照を変数 o3 に代入します。
　コンストラクタに指定する引数は次のとおりです。
　第１引数：　整数「813」
　第２引数：　文字列「神保　三郎」
５．変数 list が参照する ArrayList インスタンスの add メソッドを呼び出します。
　add メソッドに指定する引数は次のとおりです。
　引数：　変数 o1
　※変数 o2，o3 についても同様の手順で，add メソッドの引数に指定します。
６．カウンタ i の値が，“０～変数 list が参照する ArrayList インスタンスの要素数未満”の間，次の処理を繰り返します。
　・変数 list が参照する ArrayList インスタンスの get メソッドを呼び出し，返却値を Order クラス型の変数 order に代入します。
　　get メソッドに指定する引数は次のとおりです。
　　引数：　カウンタ i
　・変数 order が参照するインスタンスの getNo メソッドを呼び出し，返却値を変数 no に代入します。
　・変数 order が参照するインスタンスの getCustName メソッドを呼び出し，返却値を変数 custName に代入します。
　・変数 no と custName の値を，ディスプレイに表示します。

オブジェクト指向プログラミング②

第3部



第1章 クラスの拡張

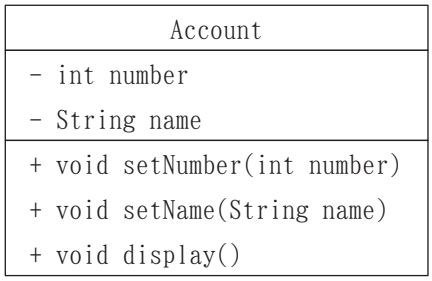
3-1-1 継承

例題をとおして，継承について学習します。また，差分コーディングについても学習します。

例題3-1-1

・インターネットで銀行口座の取引サービスを提供することになり，新たにメールアドレスを管理することになった。
次のクラスを継承して，WebAccount クラスを定義しなさい。

クラス図

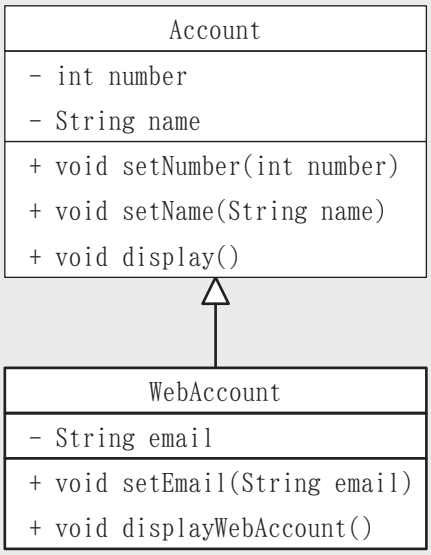


クラス定義（サブクラス）

クラス名	：	WebAccount
インスタンス変数名	：	email
メソッド名	：	setEmail メソッド
		displayWebAccount メソッド

［クラス定義例 3-1-1］

WebAccount クラス



［作業手順］

① WebAccount クラスの属性として，メールアドレスを記憶するインスタンス変数「- String email」を記述します。

② WebAccount クラスの操作として，インスタンス変数にメールアドレスを設定するメソッド「+ void setEmail(String email)」を記述します。

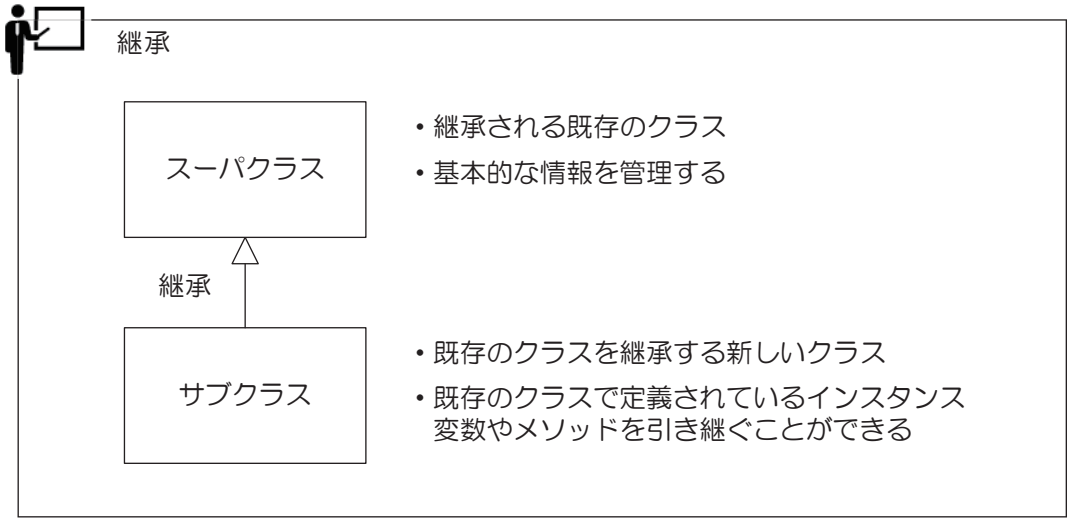
③ WebAccount クラスの操作として，ディスプレイにメールアドレスを表示するメソッド「+ void displayWebAccount()」を記述します。

④ WebAccount クラスから Account クラスに対して白抜きの実線矢印「 ——▷ 」を記入します。

例題3-1-1解説

(1) 継承

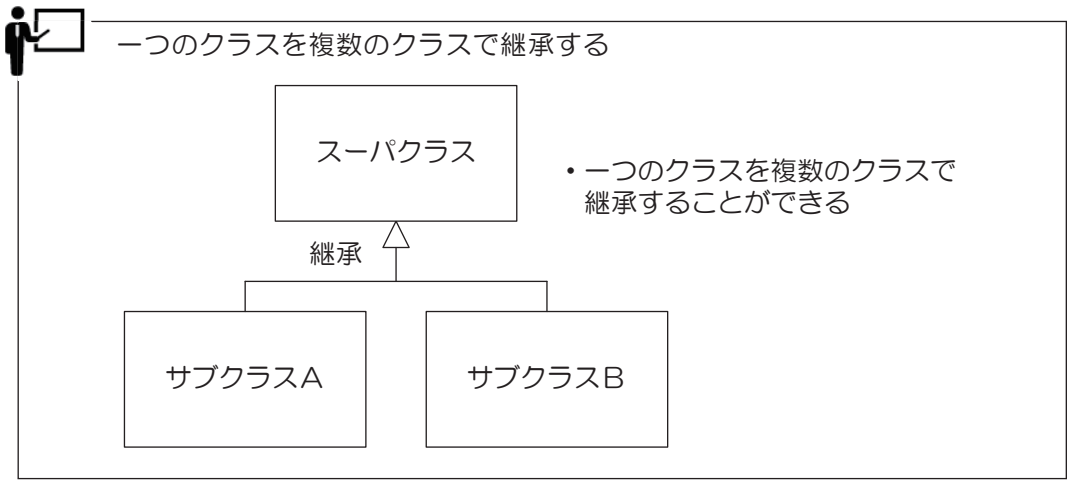
既存のクラスを引き継いで新しいクラスを定義することを**継承**といいます。



既存のクラスで定義されているインスタンス変数やメソッドは，既存のクラスを継承する新しいクラスで引き継ぐことができます。

・継承される既存のクラスを**スーパクラス**といいます。
・既存のクラスを継承する新しいクラスを**サブクラス**といいます。

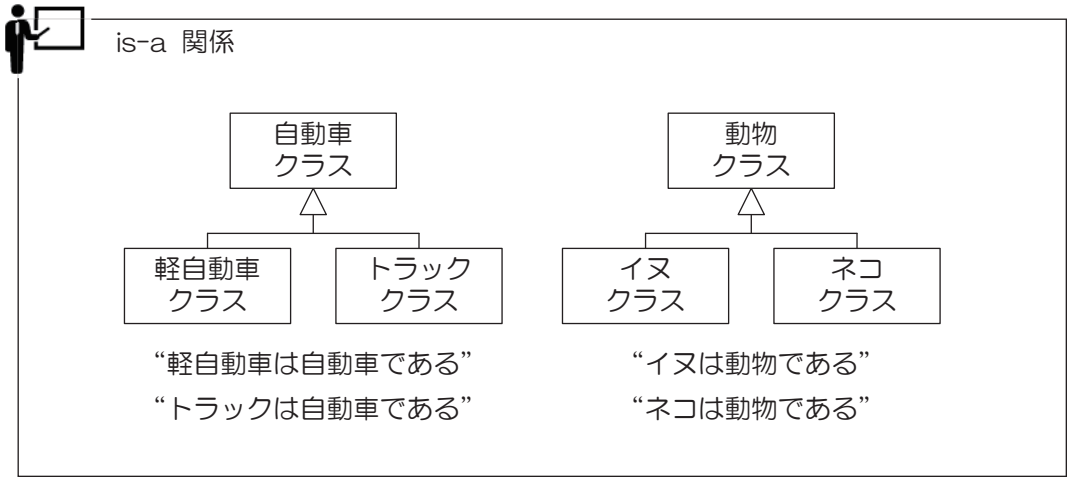
また，一つのクラスを複数のクラスで継承することができます。



・UML に含まれるクラス図の表記法では，サブクラスからスーパクラスに向けて白抜きの実線矢印を記入することで継承を表します。

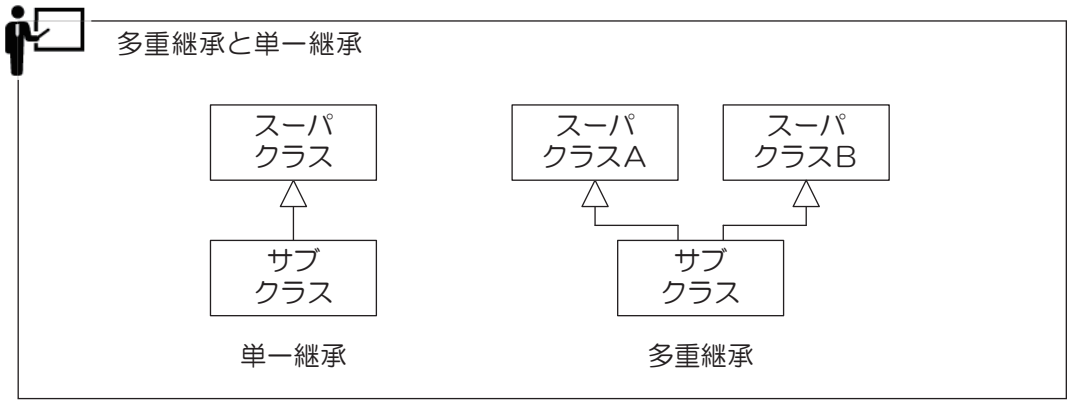
● **is-a 関係**

スーパクラスとサブクラスの間には，“サブクラスはスーパクラスである”という意味の「**is-a 関係**」が成り立ちます。



● **単一継承と多重継承**

サブクラスが一つのスーパクラスを継承することを**単一継承**といいます。また，複数のスーパクラスを継承することを**多重継承**といいます。

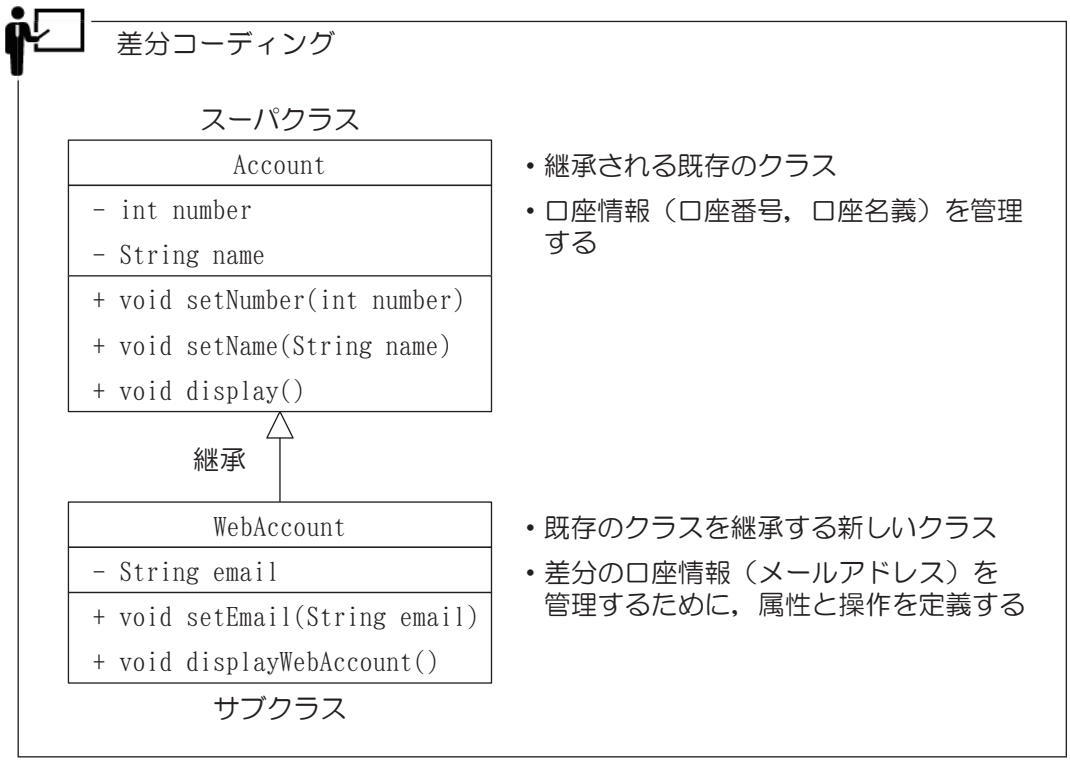


・Java では，多重継承は許されていません。

⑵ 差分コーディング

既存のクラスを修正することなく，新しいクラスで差分の属性や操作だけを効率よく定義する方法を，**差分コーディング**といいます。

クラス定義例 3-1-1 では，サブクラス（WebAccount クラス）に差分のインスタンス変数 email，setEmail メソッド，displayWebAccount メソッドを定義していますが，これが差分コーディングです。



練習問題３-１-１

・営業部の社員情報として，新たに担当地域名を管理することになった。
次のクラスを継承して，Sales クラスを定義しなさい。

クラス図

Employee
- int id - String name
+ void setId(int id) + void setName(String name) + void display()

クラス定義（サブクラス）

クラス名　　：　Sales
インスタンス変数名　：　area
メソッド名　　：　setArea メソッド
　　displaySales メソッド

Employee
- int id - String name
+ void setId(int id) + void setName(String name) + void display()

3-1-2 サブクラスの定義とインスタンス化

例題をとおして，サブクラスの定義について学習します。また，サブクラスのインスタンス化とその利用についても学習します。

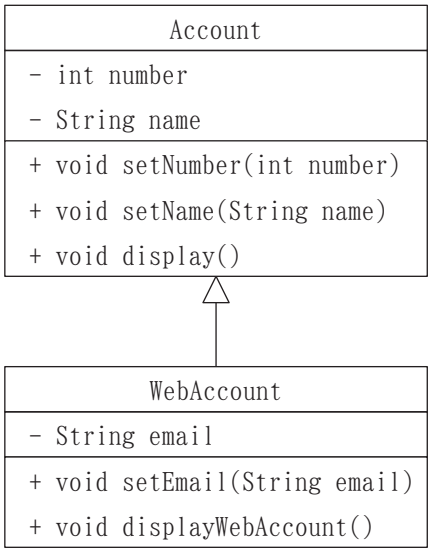
例題3-1-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
口座番号：1732050
口座名義：神保　三郎
アドレス：jinbo@java.co.jp
```

クラス図



クラス定義

クラス名　　：　Example312

インポート　：　model.WebAccount

メソッド名　：　main メソッド

※WebAccount クラスを利用します。

[プログラム例 3-1-2]

Account.java

```
1  package model;
2  public class Account {
3      private int number;
4      private String name;
5      public void setNumber(int number) {
6          this.number = number;
7      }
8      public void setName(String name) {
9          this.name = name;
10     }
11     public void display() {
12         System.out.println("口座番号：" + number);
13         System.out.println("口座名義：" + name);
14     }
15 }
```

WebAccount.java

```
1  package model;
2  public class WebAccount extends Account {
3      private String email;
4      public void setEmail(String email) {
5          this.email = email;
6      }
7      public void displayWebAccount() {
8          System.out.println("アドレス：" + email);
9      }
10 }
```

Example312.java

```
1  import model.WebAccount;
2  class Example312 {
3      public static void main(String[] args) {
4          WebAccount webAc = new WebAccount();
5          webAc.setNumber(1732050);
6          webAc.setName("神保　三郎");
7          webAc.display();
8          webAc.setEmail("jinbo@java.co.jp");
9          webAc.displayWebAccount();
10     }
11 }
```

［作業手順］

① Eclipse の「Java_第３部」プロジェクトの「src」階層に，新規 Java クラスを三つ作成します。

１．＜「新規 Java クラス」ウィンドウで入力する値（その１）＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
パッケージ	model
名前	Account
修飾子	「public」にチェックを入れる

※これ以外の項目には，入力しません。

２．＜「新規 Java クラス」ウィンドウで入力する値（その２）＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
パッケージ	model
名前	WebAccount
修飾子	「public」にチェックを入れる
スーパークラス	model.Account

※これ以外の項目には，入力しません。

３．＜「新規 Java クラス」ウィンドウで入力する値（その３）＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
名前	Example312
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 3-1-2 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，Example312 クラスを実行します。

例題 3-1-2 解説

(1) サブクラスの定義

サブクラスを定義するには，extends を使います。

記述方法（サブクラスの定義）

```
アクセス修飾子 class サブクラス名 extends スーパクラス名 {
```

・extends の後に指定できるクラス名は一つだけです。Java では，多重継承は許されていません。

プログラム例 3-1-2（WebAccount.java）では，サブクラスの定義は次に示す 2 行目です。

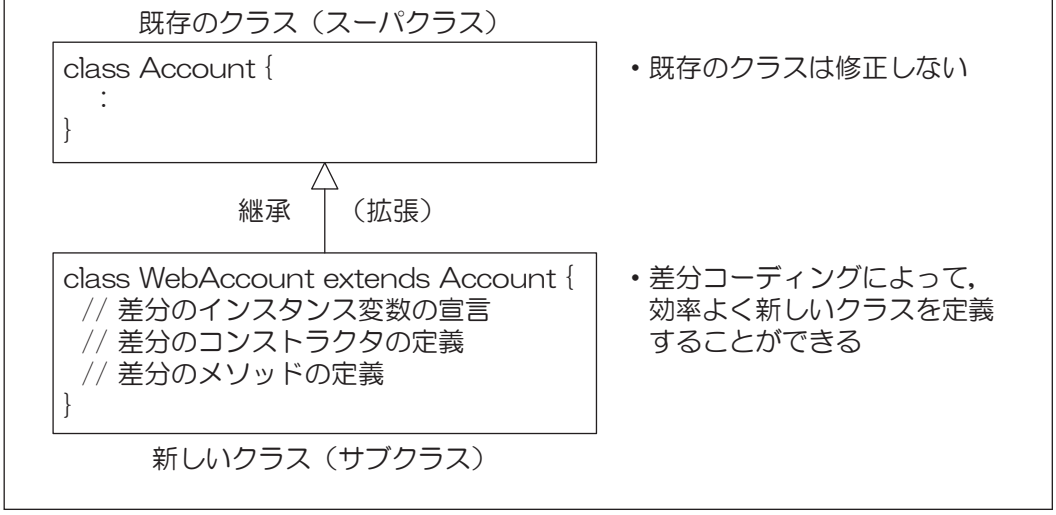
WebAccount.java（抜粋）

```
2  public class WebAccount extends Account {
```

● 継承のメリット

継承は，既存のクラスを拡張して新しいクラスを定義する仕組みです。**差分コーディング**によって，効率よく新しいクラスを定義することができます。

継承のメリット

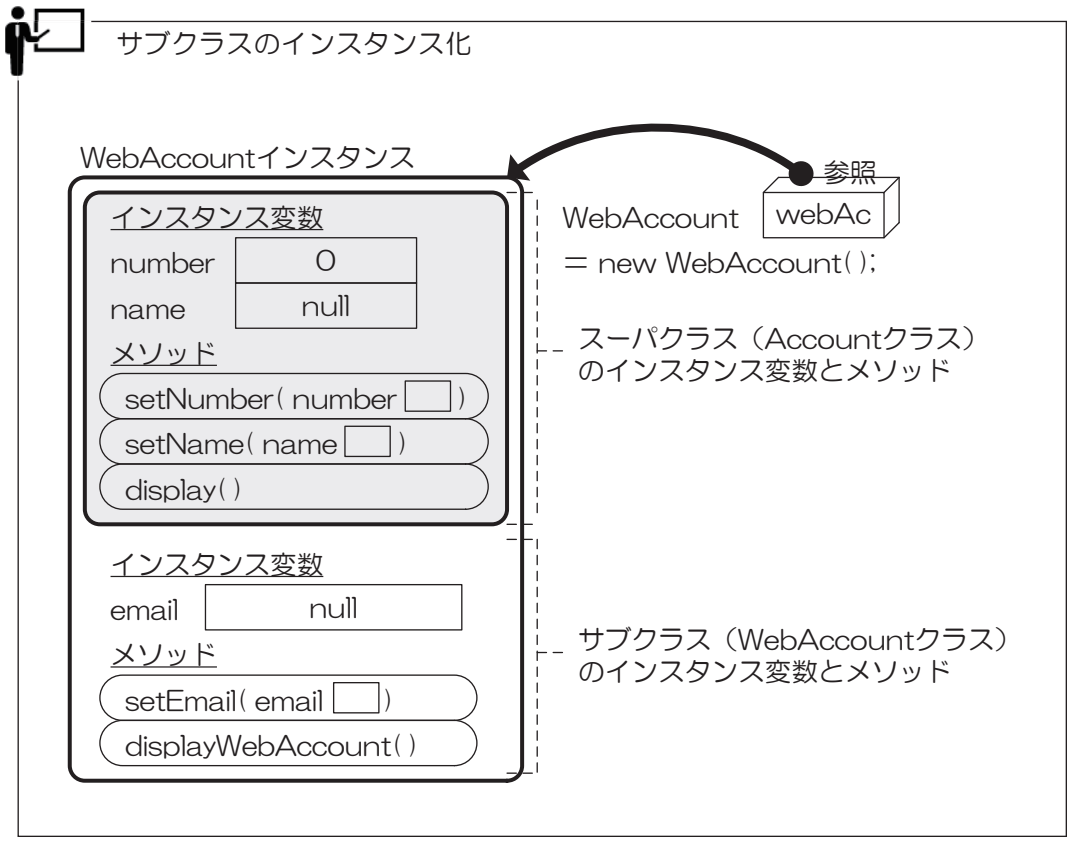


(2) サブクラスのインスタンス化

プログラム例 3-1-2（Example312.java）では，サブクラスのインスタンス化は次に示す４行目です。

Example312.java（抜粋）

```
4          WebAccount webAc = new WebAccount();
```



・サブクラスをインスタンス化すると，スーパクラスのインスタンス変数やメソッドを含むインスタンスが生成されます。

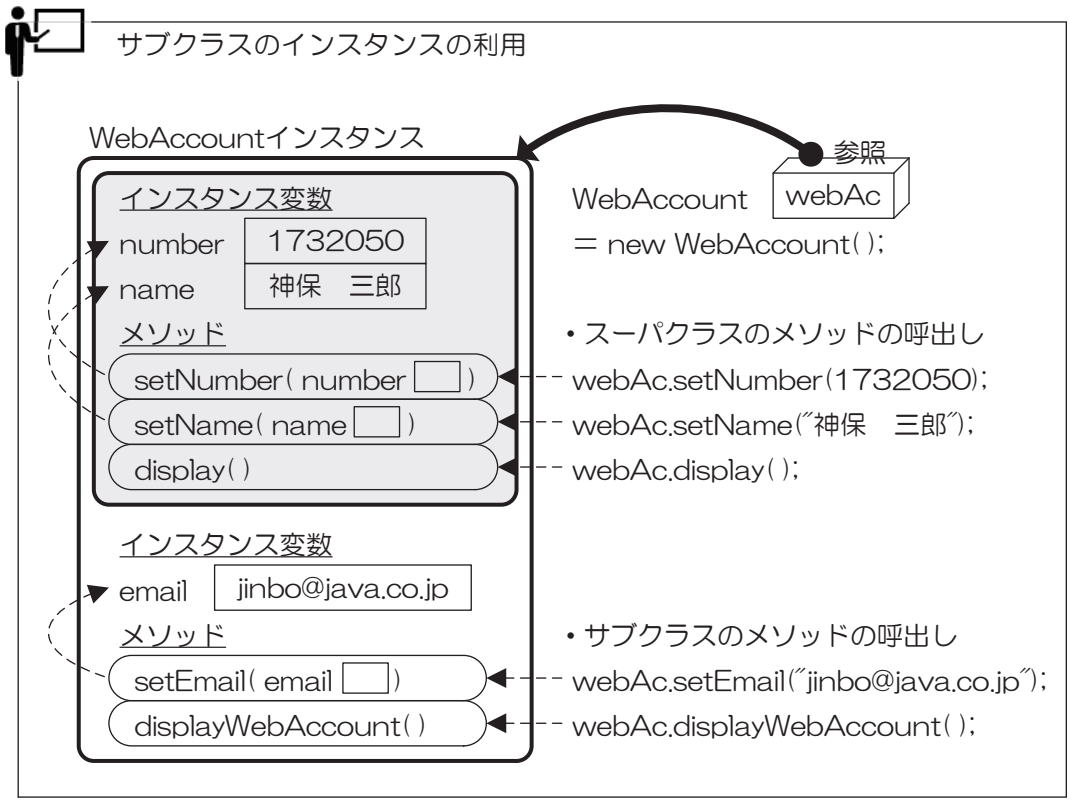
⑶　サブクラスのインスタンスの利用

サブクラスのインスタンスの参照を利用して，スーパクラスに定義されたメソッドとサブクラスに定義されたメソッドを呼び出すことができます。

プログラム例 3-1-2（Example312.java）では，メソッドの呼出しは次に示す５～９行目です。

Example312.java（抜粋）

```
5          webAc.setNumber(1732050);
6          webAc.setName("神保　三郎");
7          webAc.display();
8          webAc.setEmail("jinbo@java.co.jp");
9          webAc.displayWebAccount();
```



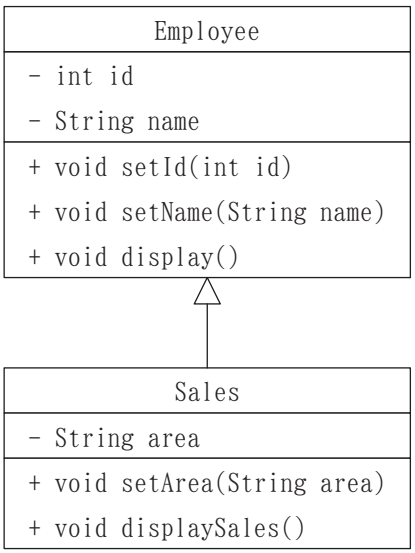
練習問題３-１-２

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
社員ID      > 2222
社員名      > 淡路花子
担当地域名 > 東京都千代田区
社員ID：2222
社員名：淡路花子
担当地域名：東京都千代田区
```

クラス図



クラス定義①（スーパクラス）

パッケージ名　：　model
クラス名　　：　Employee

クラス定義②（サブクラス）

パッケージ名　　：　model
クラス名　　：　Sales
スーパクラス名　：　model.Employee

クラス定義③

クラス名　　：　Practice312
インポート　：　java.util.Scanner
　　model.Sales
メソッド名　：　main メソッド

main メソッドの処理条件

１．文字列「社員 ID　　＞」を，ディスプレイに表示します。
２．変数 id に，キーボードから読み込んだ整数を代入します。
３．文字列「社員名　　＞」を，ディスプレイに表示します。
４．変数 name に，キーボードから読み込んだ文字列を代入します。
５．文字列「担当地域名 ＞」を，ディスプレイに表示します。
６．変数 area に，キーボードから読み込んだ文字列を代入します。
７．Sales クラスのコンストラクタを呼び出し，生成した Sales クラスのインスタンスの参照を変数 sal に代入します。
８．変数 sal が参照するインスタンスの setId メソッドを呼び出します。
　　setId メソッドに指定する引数は次のとおりです。
　　引数：　変数 id
９．変数 sal が参照するインスタンスの setName メソッドを呼び出します。
　　setName メソッドに指定する引数は次のとおりです。
　　引数：　変数 name
10．変数 sal が参照するインスタンスの display メソッドを呼び出します。
11．変数 sal が参照するインスタンスの setArea メソッドを呼び出します。
　　setArea メソッドに指定する引数は次のとおりです。
　　引数：　変数 area
12．変数 sal が参照するインスタンスの displaySales メソッドを呼び出します。

※例外が発生した場合は，文字列「エラーが発生しました！」をディスプレイに表示します。

第2章 オーバライド

3-2-1 メソッドの再定義

例題をとおして，メソッドの再定義について学習します。また，「super.～」についても学習します。

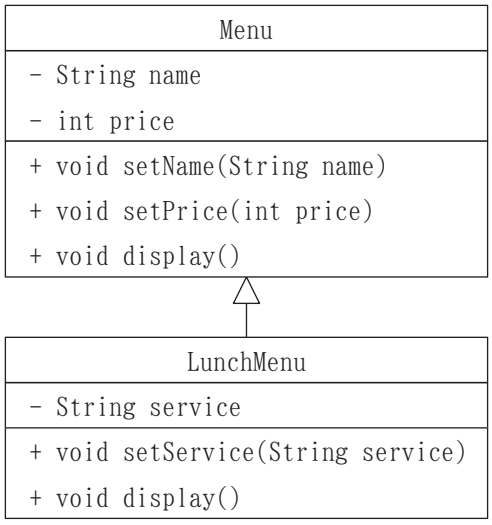
例題3-2-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
品名　　：チキンステーキ
単価　　：690円
サービス：ドリンクバー
```

クラス図



クラス定義

クラス名　　：　Example321

インポート　：　model.LunchMenu

メソッド名　：　main メソッド

※LunchMenu クラスを利用します。

［プログラム例 3-2-1］

Menu.java

```
1  package model;
2  public class Menu {
3      private String name;
4      private int price;
5      public void setName(String name) {
6          this.name = name;
7      }
8      public void setPrice(int price) {
9          this.price = price;
10     }
11     public void display() {
12         System.out.println("品名    ：" + name);
13         System.out.println("単価    ：" + price + "円");
14     }
15 }
```

LunchMenu.java

```
1  package model;
2  public class LunchMenu extends Menu {
3      private String service;
4      public void setService(String service) {
5          this.service = service;
6      }
7      public void display() {
8          super.display();
9          System.out.println("サービス：" + service);
10     }
11 }
```

Example321.java

```
1  import model.LunchMenu;
2  class Example321 {
3      public static void main(String[] args) {
4          LunchMenu lunchMe = new LunchMenu();
5          lunchMe.setName("チキンステーキ");
6          lunchMe.setPrice(690);
7          lunchMe.setService("ドリンクバー");
8          lunchMe.display();
9      }
10 }
```

［作業手順］

① Eclipse の「Java_第３部」プロジェクトの「src」階層に，新規 Java クラスを三つ作成します。

１．＜「新規 Java クラス」ウィンドウで入力する値（その１）＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
パッケージ	model
名前	Menu
修飾子	「public」にチェックを入れる

※これ以外の項目には，入力しません。

２．＜「新規 Java クラス」ウィンドウで入力する値（その２）＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
パッケージ	model
名前	LunchMenu
修飾子	「public」にチェックを入れる
スーパークラス	model. Menu

※これ以外の項目には，入力しません。

３．＜「新規 Java クラス」ウィンドウで入力する値（その３）＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
名前	Example321
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 3-2-1 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，Example321 クラスを実行します。

例題3-2-1解説

(1) メソッドの再定義

スーパクラスのメソッドをサブクラスで再定義することを，**オーバライド**といいます。プログラム例 3-2-1（LunchMenu.java）では，オーバライドは次に示す 7 ～10 行目です。

LunchMenu.java（抜粋）

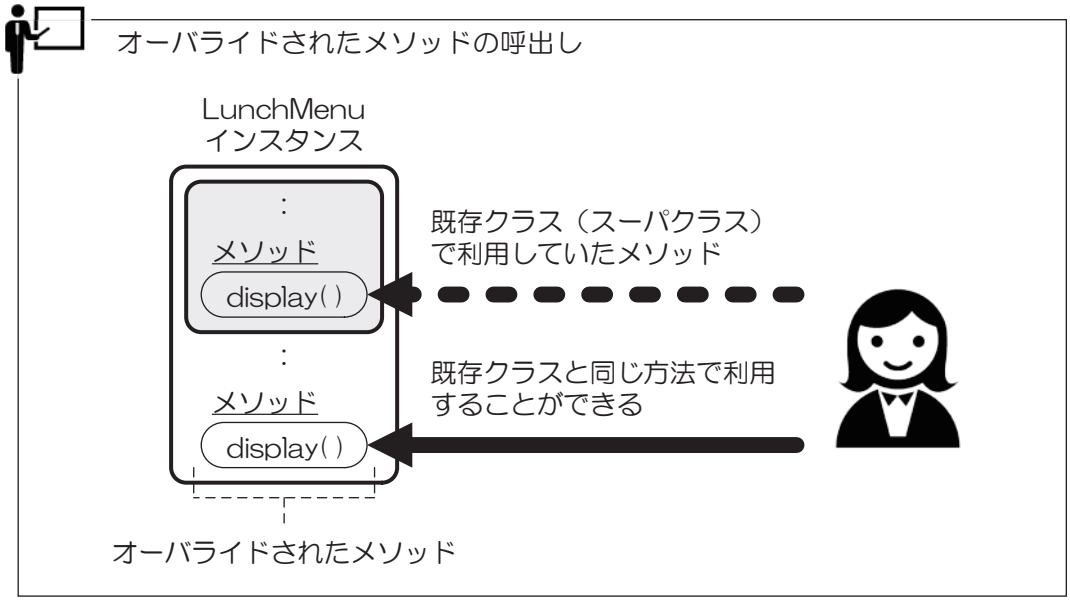
```
 7     public void display() {
 8         super.display();
 9         System.out.println("サービス：" + service);
10     }
```

・オーバライドは，次の条件を満たす場合に成立します。
①スーパクラスのメソッドと，メソッド名が同じ
②スーパクラスのメソッドと，引数の数，データ型が同じ
③スーパクラスのメソッドと，返却値のデータ型が同じ
④スーパクラスのメソッドと，アクセスレベルが同じかまたは弱いアクセスレベル（public が最も弱い）

・スーパクラスのメソッドと，メソッド名が同じで引数の数とデータ型も同じ場合，返却値のデータ型が異なるとコンパイルエラーになります。

● オーバライドのメリット

オーバライドは，既存のメソッドと同じ形式で新しいメソッドを定義する仕組みです。メソッドを利用する側にとって，使いやすい機能を提供することができます。



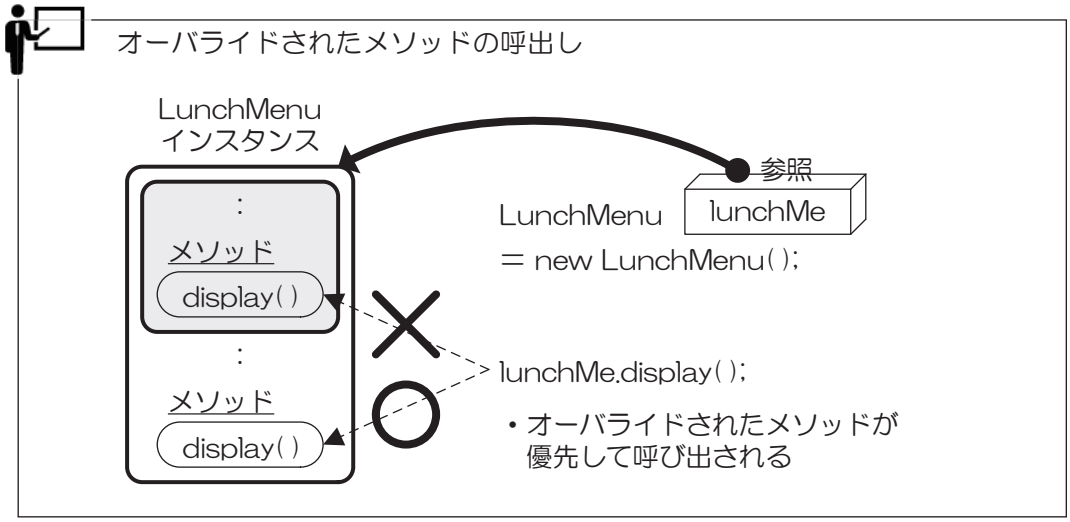
● オーバライドされたメソッドの呼出し

サブクラスのインスタンスを生成した場合は，サブクラスでオーバライドされたメソッドがスーパクラスのメソッドよりも優先して呼び出されます。

プログラム例 3-2-1（Example321.java）では，オーバライドされたメソッドの呼出しは次に示す４行目です。

Example321.java（抜粋）

```
4          LunchMenu lunchMe = new LunchMenu();
```



・サブクラスの参照を利用してメソッドを呼び出す場合は，スーパクラスのオーバライドされたメソッドを呼び出すことはできません。

参考：アノテーション

アノテーションは，JavaSE5.0（1.5）から追加された注釈の機能です。クラスやメソッドに対して，付加情報を簡単に記述することができます。

記述例（アノテーション）

```
@Override
public void display() {
    super.display();
    System.out.println("サービス：" + service);
}
```

「@」に続けて，注釈の種類を記述します。「@Override」を記述したメソッドは，コンパイラによって正しくオーバライドされているかチェックされます。

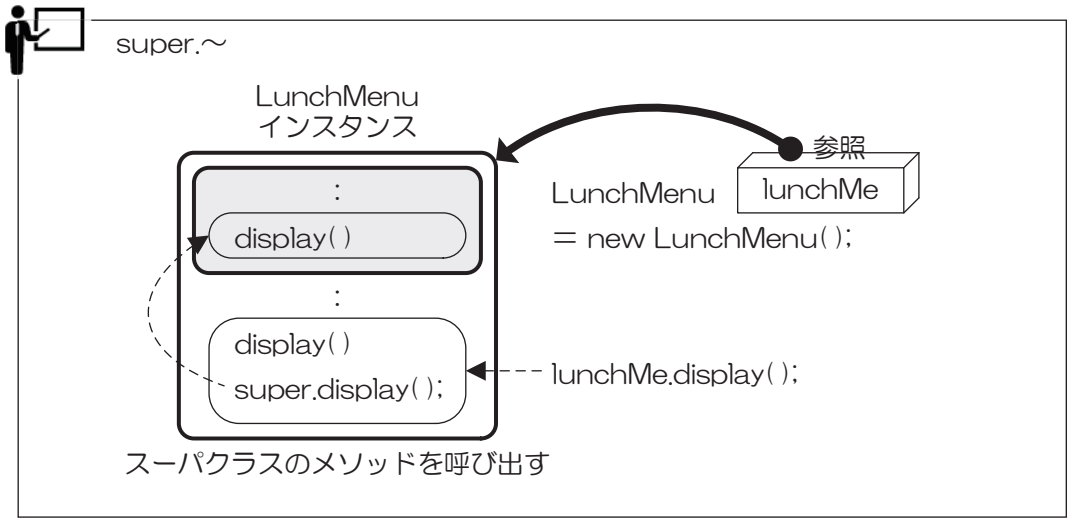
(2) super.～

super は，スーパクラスのインスタンスを参照している特別な変数です。「super.」に続けてインスタンス変数やメソッドを指定します。

プログラム例 3-2-1（LunchMenu.java）では，super は次に示す８行目です。

LunchMenu.java（抜粋）

```
8          super.display();
```



・「super.」を記述しない場合は，同じクラス内に定義された display メソッド（呼出し元のメソッド自身）を呼び出すことになります。

・スーパクラスに定義されているオーバライドされていないメソッドは，サブクラスからは「super.」を省略して呼び出すことができます。

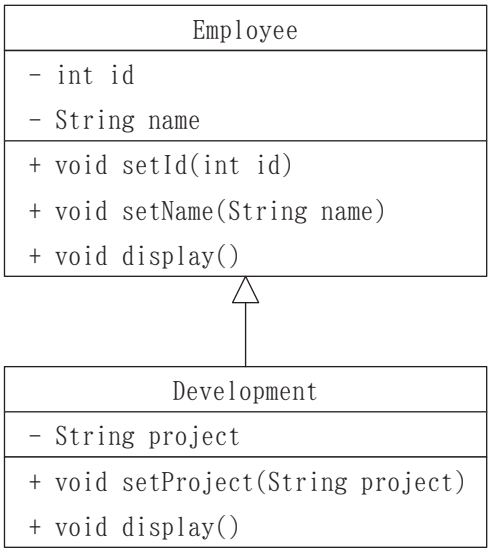
練習問題３-２-１

・次の実行結果になるように，プログラムを作成しなさい。
※練習問題 3-1-2 で作成した「Employee.java」を利用して作成します。

実行結果

```
社員ID          > 1111
社員名          > 神田二郎
プロジェクト名 > プロジェクトＸ
社員ID：1111
社員名：神田二郎
プロジェクト名：プロジェクトＸ
```

クラス図



クラス定義①（スーパクラス）

パッケージ名　：　model
クラス名　　：　Employee
※Employee クラスは，練習問題 3-1-2 で作成済です。

クラス定義②（サブクラス）

パッケージ名　　：　model
クラス名　　：　Development
スーパクラス名　：　model.Employee

クラス定義③

クラス名	：	Practice321
インポート	：	java.util.Scanner
		model.Development
メソッド名	：	main メソッド

main メソッドの処理条件

１．文字列「社員ID　　＞」を，ディスプレイに表示します。
２．変数 id に，キーボードから読み込んだ整数を代入します。
３．文字列「社員名　　＞」を，ディスプレイに表示します。
４．変数 name に，キーボードから読み込んだ文字列を代入します。
５．文字列「プロジェクト名 ＞」を，ディスプレイに表示します。
６．変数 project に，キーボードから読み込んだ文字列を代入します。
７．Development クラスのコンストラクタを呼び出し，生成した Development クラスのインスタンスの参照を変数 dev に代入します。
８．変数 dev が参照するインスタンスの setId メソッドを呼び出します。
　setId メソッドに指定する引数は次のとおりです。
　引数：　変数 id
９．変数 dev が参照するインスタンスの setName メソッドを呼び出します。
　setName メソッドに指定する引数は次のとおりです。
　引数：　変数 name
10．変数 dev が参照するインスタンスの setProject メソッドを呼び出します。
　setProject メソッドに指定する引数は次のとおりです。
　引数：　変数 project
11．変数 dev が参照するインスタンスの display メソッドを呼び出します。

※例外が発生した場合は，「エラーが発生しました！」をディスプレイに表示します。

3-2-2 super( )

例題をとおして，「super( )」について学習します。

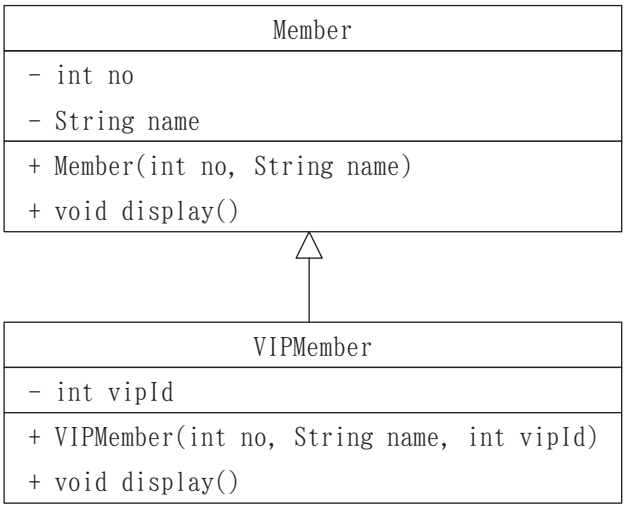
例題3-2-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
会員番号 ：7777
会員名　 ：本郷　良子
VIP会員ID：9321
```

クラス図



クラス定義

クラス名　　：　Example322

インポート　：　model.VIPMember

メソッド名　：　main メソッド

※VIPMember クラスを利用します。

[プログラム例 3-2-2]

Member.java

```
 1  package model;
 2  public class Member {
 3      private int no;
 4      private String name;
 5      public Member(int no, String name) {
 6          this.no = no;
 7          this.name = name;
 8      }
 9      public void display() {
10          System.out.println("会員番号 ：" + no);
11          System.out.println("会員名   ：" + name);
12      }
13  }
```

VIPMember.java

```
 1  package model;
 2  public class VIPMember extends Member {
 3      private int vipId;
 4      public VIPMember(int no, String name, int vipId) {
 5          super(no, name);
 6          this.vipId = vipId;
 7      }
 8      public void display() {
 9          super.display();
10          System.out.println("VIP 会員 ID：" + vipId);
11      }
12  }
```

Example322.java

```
 1  import model.VIPMember;
 2  class Example322 {
 3      public static void main(String[] args) {
 4          VIPMember vip = new VIPMember(7777, "本郷　良子", 9321);
 5          vip.display();
 6      }
 7  }
```

［作業手順］

① Eclipse の「Java_第３部」プロジェクトの「src」階層に，新規 Java クラスを三つ作成します。

１．＜「新規 Java クラス」ウィンドウで入力する値（その１）＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
パッケージ	model
名前	Member
修飾子	「public」にチェックを入れる

※これ以外の項目には，入力しません。

２．＜「新規 Java クラス」ウィンドウで入力する値（その２）＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
パッケージ	model
名前	VIPMember
修飾子	「public」にチェックを入れる
スーパークラス	model.Member

※これ以外の項目には，入力しません。

３．＜「新規 Java クラス」ウィンドウで入力する値（その３）＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
名前	Example322
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 3-2-2 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

③ Eclipse の ボタンを押して，Example322 クラスを実行します。

例題3-2-2解説

(1) super()

「super()」を利用して，スーパクラスのコンストラクタを呼び出すことができます。

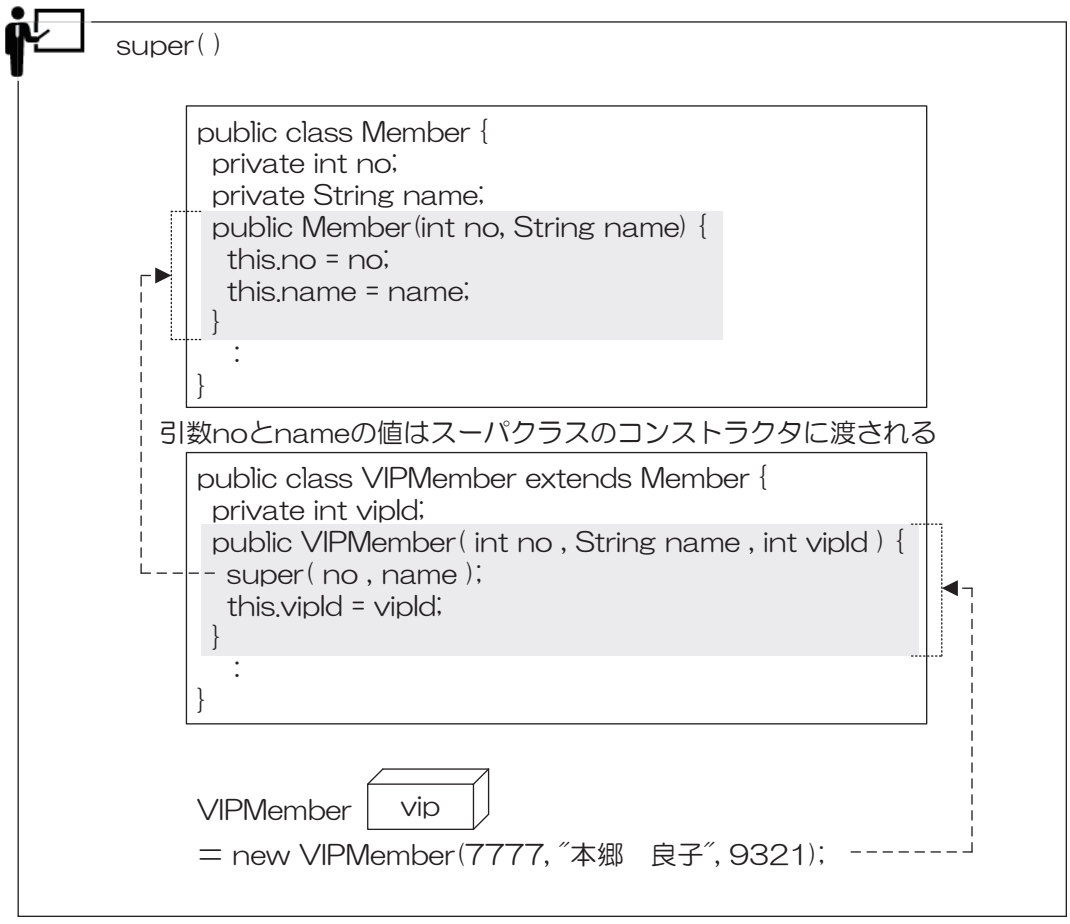
記述方法（super()）

```
super(引数, …);
```

プログラム例3-2-2（VIPMember.java）では，「super()」は次に示す５行目です。

VIPMember.java（抜粋）

```
5          super(no, name);
```



・「super()」は，コンストラクタ内の先頭行に記述します。先頭行以外に記述した場合は，コンパイルエラーになります。

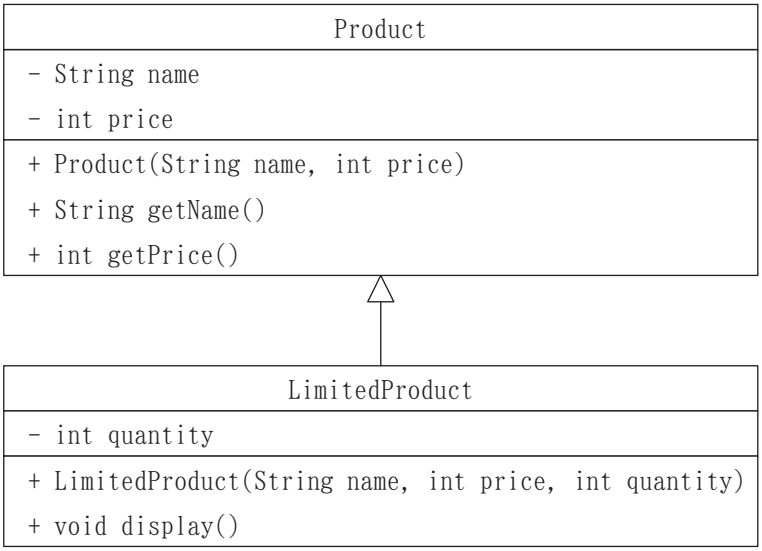
練習問題３-２-２

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
製品名    > 低反発まくら
価格      > 5000
限定数量 > 50
製品名    ：低反発まくら
価格      ：5000円
限定数量：50個
```

クラス図



クラス定義①（スーパクラス）

パッケージ名　：　model
クラス名　　：　Product

クラス定義②（サブクラス）

パッケージ名　　：　model
クラス名　　：　LimitedProduct
スーパクラス名　：　model.Product

クラス定義③

クラス名　　：　Practice322
インポート　：　java.util.Scanner
　　model.LimitedProduct
メソッド名　：　main メソッド

main メソッドの処理条件

１．文字列「製品名　　＞」を，ディスプレイに表示します。
２．変数 name に，キーボードから読み込んだ文字列を代入します。
３．文字列「価格　　＞」を，ディスプレイに表示します。
４．変数 price に，キーボードから読み込んだ整数を代入します。
５．文字列「限定数量 ＞」を，ディスプレイに表示します。
６．変数 quantity に，キーボードから読み込んだ整数を代入します。
７．LimitedProduct クラスのコンストラクタを呼び出し，生成した LimitedProduct クラスのインスタンスの参照を変数 lim に代入します。
　コンストラクタに指定する引数は次のとおりです。
　第一引数：　変数 name
　第二引数：　変数 price
　第三引数：　変数 quantity
８．変数 lim が参照するインスタンスの display メソッドを呼び出します。

第3章 インタフェース

3-3-1 インタフェースとは?

例題をとおして，インタフェースについて学習します。また，インタフェースの定義と実装についても学習します。

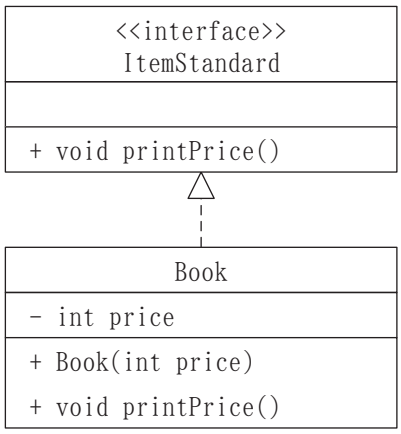
例題3-3-1

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
<書籍情報>
定価：2500円
```

クラス図



クラス定義

クラス名　：　Example331

インポート　：　model.Book

メソッド名　：　main メソッド

※Book クラスを利用します。

[プログラム例 3-3-1]

ItemStandard.java

```
1  package model;
2  public interface ItemStandard {
3      public abstract void printPrice();
4  }
```

Book.java

```
1  package model;
2  public class Book implements ItemStandard {
3      private int price;
4      public Book(int price) {
5          this.price = price;
6      }
7      public void printPrice() {
8          System.out.println("定価：" + price + "円");
9      }
10 }
```

Example331.java

```
1  import model.Book;
2  class Example331 {
3      public static void main(String[] args) {
4          System.out.println("<書籍情報>");
5          Book book = new Book(2500);
6          book.printPrice();
7      }
8  }
```

［作業手順］

① Eclipse の「Java_第３部」プロジェクトの「src」階層に，新規 Java インタフェースを作成します。

＜「新規 Java インターフェース」ウィンドウで入力する値＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
パッケージ	model
名前	ItemStandard
修飾子	「public」にチェックを入れる

※これ以外の項目には，入力しません。

② Eclipse の「Java_第３部」プロジェクトの「src」階層に，新規 Java クラスを二つ作成します。

１．＜「新規 Java クラス」ウィンドウで入力する値（その１）＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
パッケージ	model
名前	Book
修飾子	「public」にチェックを入れる

※これ以外の項目には，入力しません。

・インタフェースの指定は，「新規 Java クラス」ウィンドウで入力することもできます。

２．＜「新規 Java クラス」ウィンドウで入力する値（その２）＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
名前	Example331
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

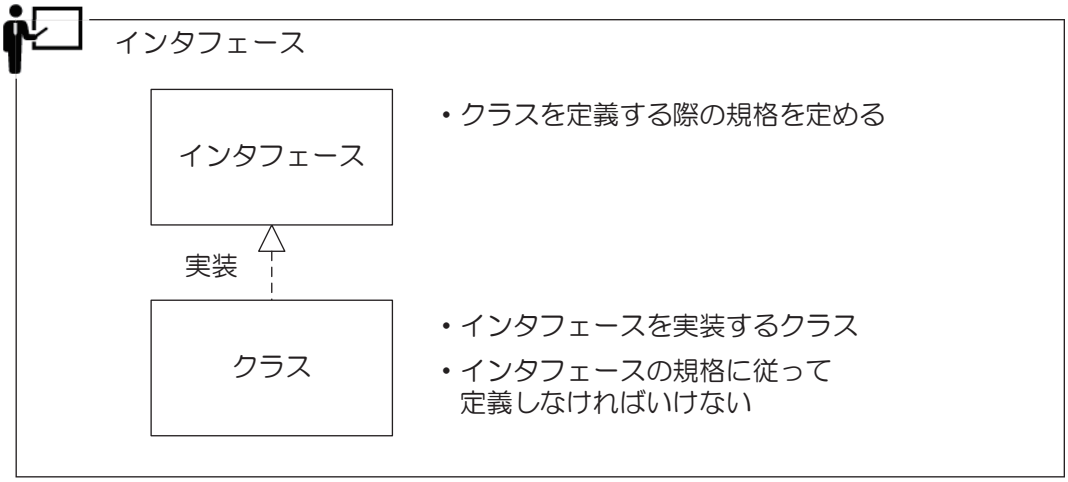
③ プログラム例 3-3-1 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

④ Eclipse の ボタンを押して，Example331 クラスを実行します。

例題3-3-1解説

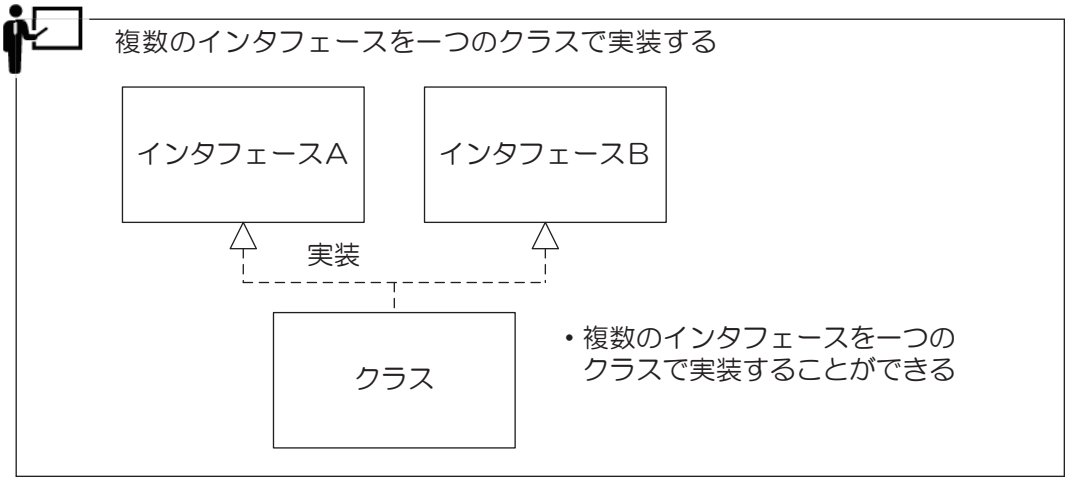
(1) インタフェース

インタフェースは，クラスを定義する際の規格を定めたものです。
インタフェースを実装するクラスは，その規格に従って定義しなければいけません。



・インタフェースで定められた規格に従ってプログラミングすることを**実装**といいます。

また，複数のインタフェースを一つのクラスで実装することができます。



・クラス図のインタフェースの表記法は次のとおりです。
①インタフェース名の上に「<<interface>>」を記述する
②抽象メソッドは斜体で記述する
※抽象メソッドの詳細については，次ページで解説します。
③インタフェースに対して白抜きの破線矢印を記入することで実装を表す

(2) インタフェースの定義

インタフェースの定義方法は，次のとおりです。

記述方法（インタフェースの定義）

```
アクセス修飾子 interface インタフェース名 {
    :
}
```

プログラム例 3-3-1（ItemStandard.java）では，インタフェースの定義は次に示す２～４行目です。

ItemStandard.java（抜粋）

```
2  public interface ItemStandard {
:      :
4  }
```

・インタフェースに指定できるアクセス修飾子は，public とデフォルト（指定なし）だけです。

● 抽象メソッド

インタフェースには，**抽象メソッド**を定義することができます。

記述方法（抽象メソッド）

```
public abstract 返却値のデータ型 メソッド名
                                (データ型 引数を受け取る変数名, …);
```

・抽象メソッドとは，返却値のデータ型，メソッド名，引数だけを定義し，処理内容を記述しないメソッドです。
・abstract は，抽象メソッドであることを表す修飾子です。

・抽象メソッドは，処理内容を記述しないため「{」と「}」は記述しません。「;」を記述して，処理内容がないことを表します。

プログラム例 3-3-1（ItemStandard.java）では，抽象メソッドの定義は次に示す３行目です。

LunchMenu.java（抜粋）

```
3      public abstract void printPrice();
```

・public と abstract は省略することもできます。省略した場合も，public と abstract を記述した場合と同じ定義となります。

● **定数の宣言**

インタフェースには，**定数**を宣言することができます。

記述例（定数の宣言）

```
public interface Figure {
    public static final double PI = 3.14;
     :
}
```

・変更することができない値を定数といいます。
・final を変数に設定することにより，２回目以降の値の上書き禁止を表します。
・final をクラスに設定することにより，継承禁止を表します。
・final をメソッドに設定することにより，オーバライド禁止を表します。

・public，static，final は省略することもできます。省略した場合も，public，static，final を記述した場合と同じ定義となります。

⑶　インタフェースの実装

インタフェースを実装するには，implements を使います。

記述方法（インタフェースの実装）

```
アクセス修飾子 class クラス名 implements インタフェース名，… {
```

プログラム例 3-3-1（Book.java）では，インタフェースの実装は次に示す２行目です。

Book.java（抜粋）

```
2  public class Book implements ItemStandard {
```

・インタフェースを実装するクラスでは，インタフェースで定義された抽象メソッドをオーバライドしなければいけません。

プログラム例 3-3-1（Book.java）では，抽象メソッドのオーバライドは次に示す７～９行目です。

Book.java（抜粋）

```
7      public void printPrice() {
8          System.out.println("定価：" + price + "円");
9      }
```

・インタフェースを実装するクラスがインタフェースで定義された抽象メソッドをオーバライドしない場合は，コンパイルエラーになります。
・インタフェースをインスタンス化することはできません。

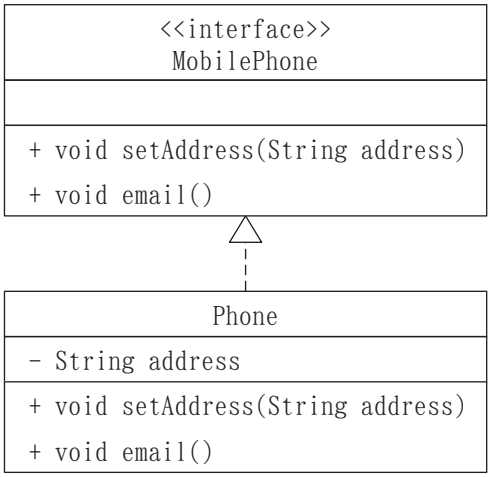
練習問題３-３-１

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
宛先 > jinbo@java.co.jp
メールを送信しました。
To：jinbo@java.co.jp
```

クラス図



インタフェース定義

パッケージ名　　：　model
インタフェース名　：　MobilePhone

クラス定義①

パッケージ名　　：　model
クラス名　　：　Phone
インタフェース名　：　model.MobilePhone

クラス定義②

クラス名　　：　Practice331
インポート　：　java.util.Scanner
　　model.Phone
メソッド名　：　main メソッド

main メソッドの処理条件

１．文字列「宛先 ＞ 」を，ディスプレイに表示します。
２．変数 address に，キーボードから読み込んだ文字列を代入します。
３．Phone クラスのコンストラクタを呼び出し，生成した Phone クラスのインスタンスの参照を変数 phone に代入します。
４．変数 phone が参照するインスタンスの setAddress メソッドを呼び出します。
　setAddress メソッドに指定する引数は次のとおりです。
　引数：　変数 address
５．変数 phone が参照するインスタンスの email メソッドを呼び出します。

※例外が発生した場合は，文字列「送信できませんでした！」をディスプレイに表示します。

3-3-2 インタフェースの拡張

例題をとおして，インタフェースの拡張について学習します。また，サブインタフェースの定義についても学習します。

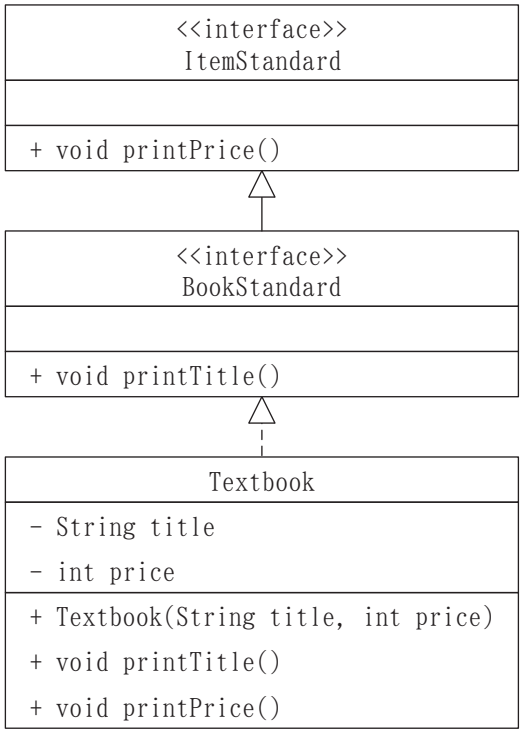
例題3-3-2

・次の実行結果になるように，プログラムを作成しなさい。
※例題 3-3-1 で定義した「ItemStandard.java」を利用します。

実行結果

```
<書籍情報>
書籍名：みんなのJava
定価　：3000円
```

クラス図



クラス定義

クラス名　　：　Example332

インポート　：　model.Textbook

メソッド名　：　main メソッド

※Textbook クラスを利用します。

[プログラム例 3-3-2]

ItemStandard.java（プログラム例 3-3-1 で作成済）

```
1  package model;
2  public interface ItemStandard {
3      public abstract void printPrice();
4  }
```

BookStandard.java

```
1  package model;
2  public interface BookStandard extends ItemStandard {
3      public abstract void printTitle();
4  }
```

Textbook.java

```
1  package model;
2  public class Textbook implements BookStandard {
3      private String title;
4      private int price;
5      public Textbook(String title, int price) {
6          this.title = title;
7          this.price = price;
8      }
9      public void printTitle() {
10         System.out.println("書籍名：" + title);
11     }
12     public void printPrice() {
13         System.out.println("定価　：" + price + "円");
14     }
15 }
```

Example332.java

```
1  import model.Textbook;
2  class Example332 {
3      public static void main(String[] args) {
4          System.out.println("<書籍情報>");
5          Textbook text = new Textbook("みんなの Java", 3000);
6          text.printTitle();
7          text.printPrice();
8      }
9  }
```

［作業手順］

① Eclipse の「Java_第３部」プロジェクトの「src」階層に，新規 Java インタフェースを作成します。

＜「新規 Java インターフェース」ウィンドウで入力する値＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
パッケージ	model
名前	BookStandard
修飾子	「public」にチェックを入れる

※これ以外の項目には，入力しません。

② Eclipse の「Java_第３部」プロジェクトの「src」階層に，新規 Java クラスを二つ作成します。

１．＜「新規 Java クラス」ウィンドウで入力する値（その１）＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
パッケージ	model
名前	Textbook
修飾子	「public」にチェックを入れる

※これ以外の項目には，入力しません。

２．＜「新規 Java クラス」ウィンドウで入力する値（その２）＞

入力項目	入力値
ソース・フォルダー	Java_第３部/src
名前	Example332
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

③ プログラム例 3-3-2 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

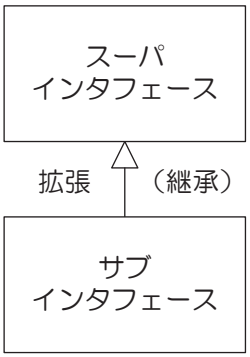
④ Eclipse の ボタンを押して，Example332 クラスを実行します。

例題3-3-2解説

(1) インタフェースの拡張

既存のインタフェースを拡張して新しいインタフェースを定義することができます。

インタフェースの拡張



・継承される既存のインタフェース
・基本的な規格を定める

・既存のインタフェースを拡張（継承）する新しいインタフェース

※白抜きの実線矢印を記入することで
インタフェースの拡張（継承）を表す

(2) サブインタフェースの定義

サブインタフェースを定義するには，extends を使います。

プログラム例 3-3-2（BookStandard.java）では，サブインタフェースの定義は次に示す２行目です。

BookStandard.java（抜粋）

```
2  public interface BookStandard extends ItemStandard {
```

・サブインタフェースを実装するクラスが，スーパインタフェースとサブインタフェースで定義された抽象メソッドをオーバライドしない場合は，コンパイルエラーになります。

プログラム例 3-3-2（Textbook.java）では，抽象メソッドのオーバライドは次に示す９～14 行目です。

Textbook.java（抜粋）

```
 9      public void printTitle() {
10          System.out.println("書籍名：" + title);
11      }
12      public void printPrice() {
13          System.out.println("定価　：" + price + "円");
14      }
```

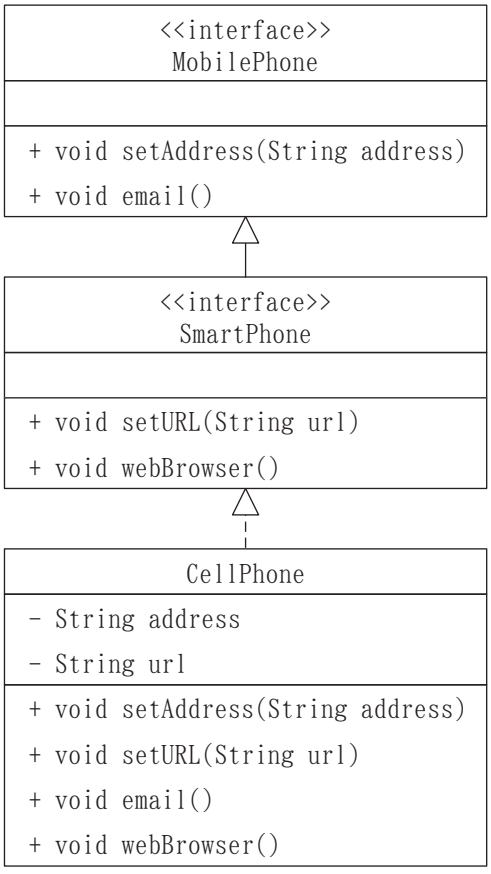
練習問題３-３-２

・次の実行結果になるように，プログラムを作成しなさい。
　※練習問題 3-3-1 で定義した「MobilePhone.java」を利用します。

実行結果

```
宛先 > jinbo@java.co.jp
メールを送信しました。
To  :jinbo@java.co.jp
アドレス > http://www.minjava.co.jp
Webページにアクセスしました。
URL：http://www.minjava.co.jp
```

クラス図



インタフェース定義

パッケージ名　　：　model
インタフェース名　　：　SmartPhone
スーパインタフェース名　：　MobilePhone

クラス定義①

パッケージ名　　：　model
クラス名　　：　CellPhone
インタフェース名　：　model.SmartPhone

クラス定義②

クラス名　　：　Practice332
インポート　：　java.util.Scanner
　　model.CellPhone
メソッド名　：　main メソッド

main メソッドの処理条件

１．文字列「宛先 ＞ 」を，ディスプレイに表示します。
２．変数 address に，キーボードから読み込んだ文字列を代入します。
３．CellPhone クラスのコンストラクタを呼び出し，生成した CellPhone クラスのインスタンスの参照を変数 phone に代入します。
４．変数 phone が参照するインスタンスの setAddress メソッドを呼び出します。
　setAddress メソッドに指定する引数は次のとおりです。
　引数：　変数 address
５．変数 phone が参照するインスタンスの email メソッドを呼び出します。
６．文字列「アドレス ＞ 」を，ディスプレイに表示します。
７．変数 url に，キーボードから読み込んだ文字列を代入します。
８．変数 phone が参照するインスタンスの setURL メソッドを呼び出します。
　setURL メソッドに指定する引数は次のとおりです。
　引数：　変数 url
９．変数 phone が参照するインスタンスの webBrowser メソッドを呼び出します。

※例外が発生した場合は，文字列「エラーが発生しました！」をディスプレイに表示します。

第4章 ポリモフィズム

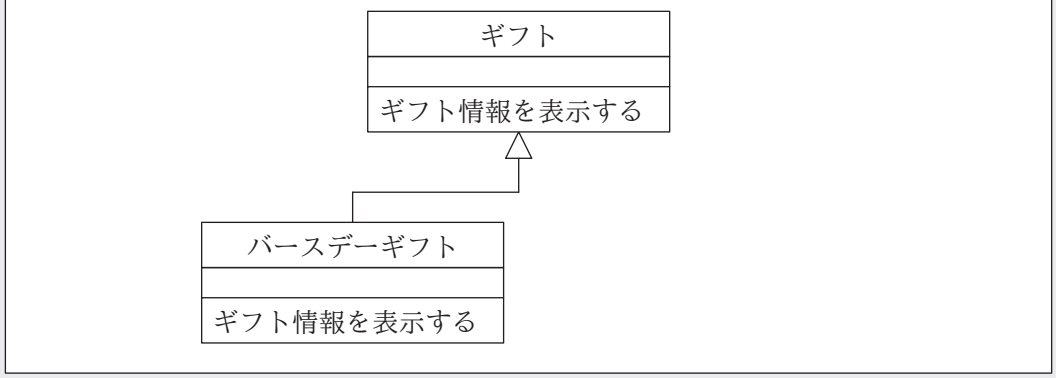
3−4−1 ポリモフィズムとは?

例題をとおして，ポリモフィズムについて学習します。また，ポリモフィズムの実現方法についても学習します。

例題3-4-1

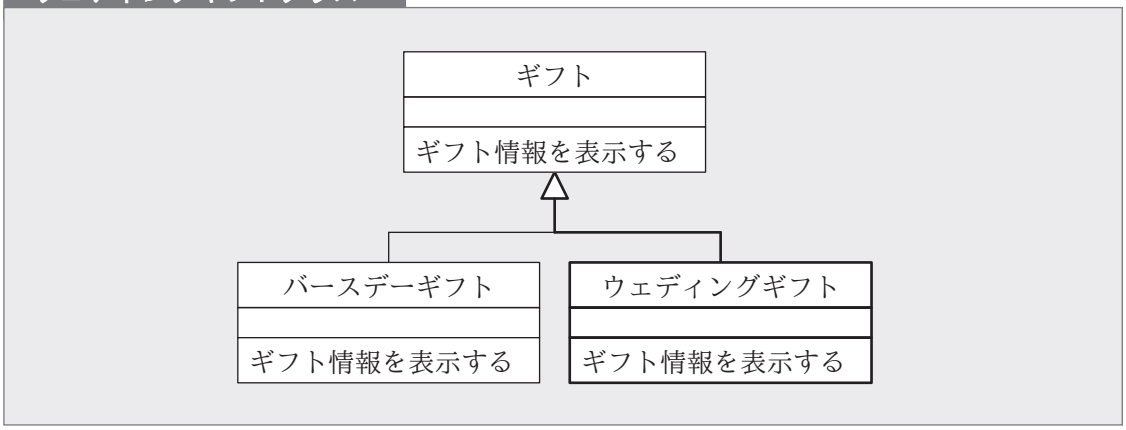
・バースデーギフトクラスを参考にして，ギフトクラスを継承するウェディングギフトクラスを定義しなさい。

クラス図



［クラス定義例 3-4-1］

ウェディングギフトクラス



［作業手順］

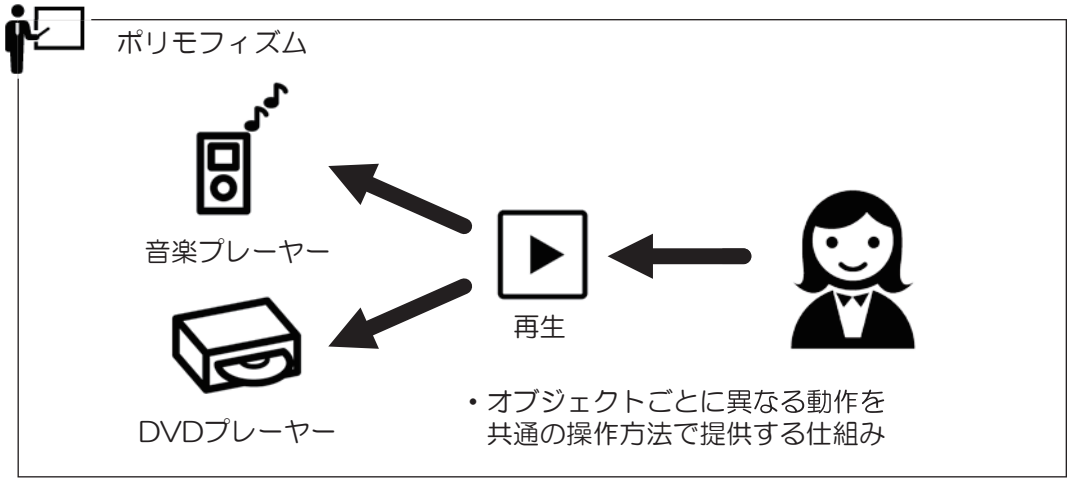
① ウェディングギフトクラスの操作として，「ギフト情報を表示する」を記述します。

② ウェディングギフトクラスからギフトクラスに対して白抜きの実線矢印「 ——▷ 」を記入します。

例題3-4-1解説

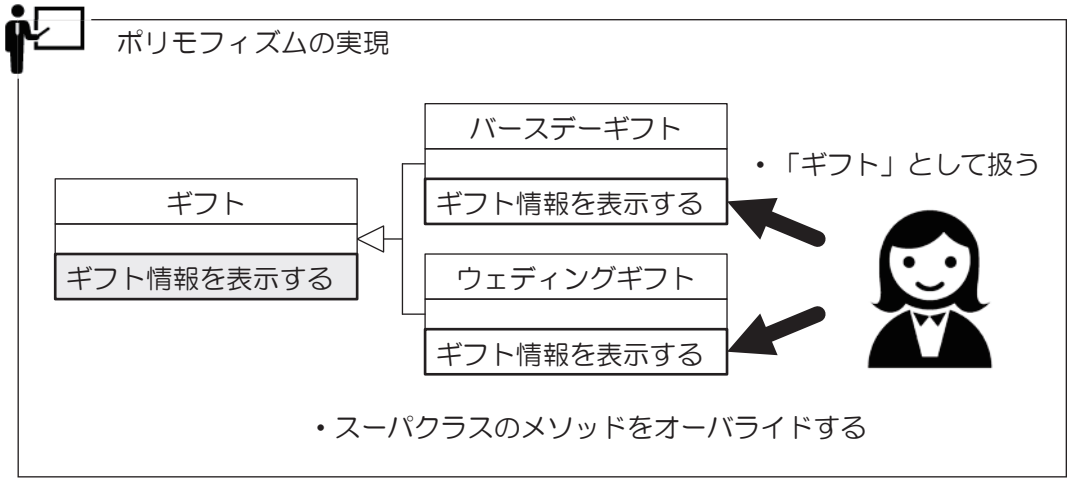
(1) ポリモフィズム

ポリモフィズム（多態性，多様性）とは，オブジェクトごとに異なる動作を共通の操作方法で提供する仕組みです。



(2) ポリモフィズムの実現方法

ポリモフィズムは，継承によって実現することができます。



サブクラスがスーパクラスのメソッドをオーバライドすることで，オブジェクトの利用者に共通の操作方法を提供することができます。

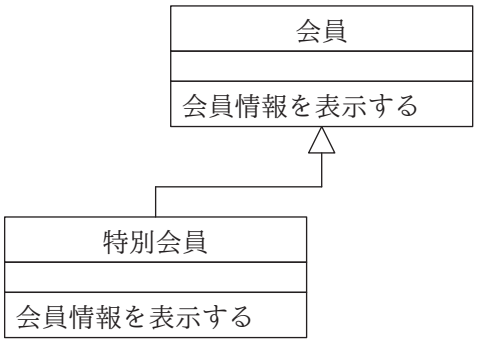
また，利用者はサブクラスのオブジェクトをスーパクラスと同じ種類のオブジェクトとして扱うことができます。

・ポリモフィズムは，インタフェースの実装によって実現することもできます。
・インタフェースの実装によって，共通の操作を持つことが保証されます。

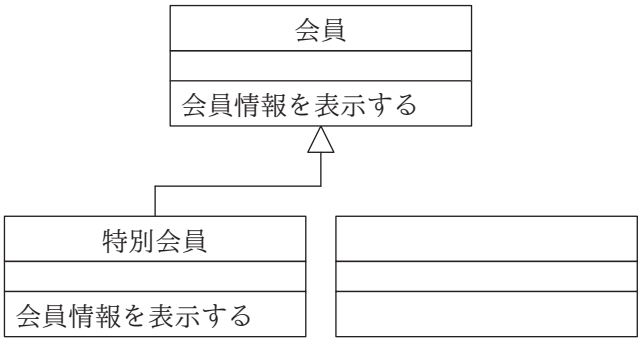
練習問題３-４-１

・特別会員クラスを参考にして，会員クラスを継承する法人会員クラスを定義しなさい。

クラス図



クラス定義（サブクラス）



3-4-2 ポリモフィズムの実現

例題をとおして，ポリモフィズムの実現について学習します。また，ポリモフィズムにおけるオブジェクトの利用方法についても学習します。

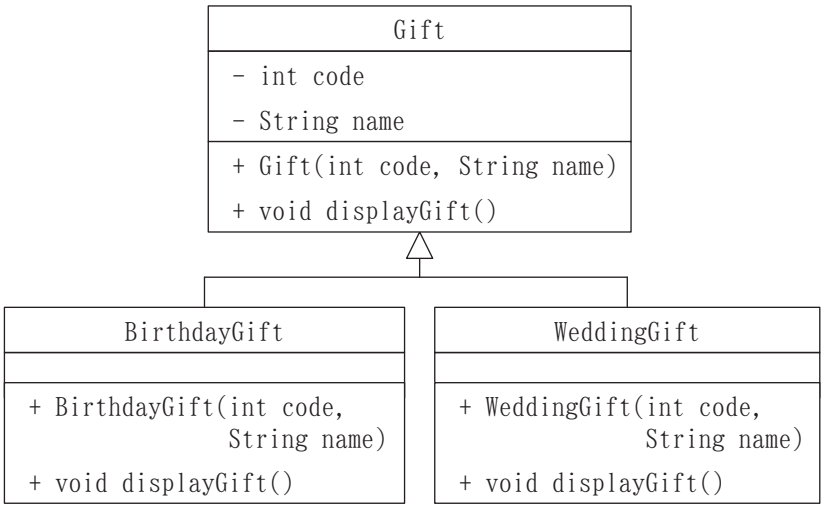
例題3-4-2

・次の実行結果になるように，プログラムを作成しなさい。

実行結果

```
<バースデーギフト>
ギフトコード：5523
ギフト名　　：リストウォッチ
<ウェディングギフト>
ギフトコード：1122
ギフト名　　：ティーカップ
```

クラス図



クラス定義

クラス名　：　Example342

インポート　：　java.util.ArrayList
model.Gift
model.BirthdayGift
model.WeddingGift

メソッド名　：　main メソッド

[プログラム例 3-4-2]

Gift.java

```
 1  package model;
 2  public class Gift {
 3      private int code;
 4      private String name;
 5      public Gift(int code, String name) {
 6          this.code = code;
 7          this.name = name;
 8      }
 9      public void displayGift() {
10          System.out.println("ギフトコード：" + code);
11          System.out.println("ギフト名    ：" + name);
12      }
13  }
```

BirthdayGift.java

```
 1  package model;
 2  public class BirthdayGift extends Gift {
 3      public BirthdayGift(int code, String name) {
 4          super(code, name);
 5      }
 6      public void displayGift() {
 7          System.out.println("<バースデーギフト>");
 8          super.displayGift();
 9      }
10  }
```

WeddingGift.java

```
 1  package model;
 2  public class WeddingGift extends Gift {
 3      public WeddingGift(int code, String name) {
 4          super(code, name);
 5      }
 6      public void displayGift() {
 7          System.out.println("<ウェディングギフト>");
 8          super.displayGift();
 9      }
10  }
```

（次のページに続く）

Example342.java

```
1  import java.util.ArrayList;
2  import model.Gift;
3  import model.BirthdayGift;
4  import model.WeddingGift;
5  class Example342 {
6      public static void main(String[] args) {
7          Gift gift1 = new BirthdayGift(5523, "リストウォッチ");
8          Gift gift2 = new WeddingGift(1122, "ティーカップ");
9          ArrayList<Gift> list = new ArrayList<Gift>();
10         list.add(gift1);
11         list.add(gift2);
12         for(Gift gift : list) {
13             gift.displayGift();
14         }
15     }
16 }
```

[作業手順]

① Eclipse の「Java_第３部」プロジェクトの「src」階層に，新規 Java クラスを四つ作成します。

1．<「新規 Java クラス」ウィンドウで入力する値（その１）>

入力項目	入力値
ソース・フォルダー	Java_第３部/src
パッケージ	model
名前	Gift
修飾子	「public」にチェックを入れる

※これ以外の項目には，入力しません。

2．<「新規 Java クラス」ウィンドウで入力する値（その２）>

入力項目	入力値
ソース・フォルダー	Java_第３部/src
パッケージ	model
名前	BirthdayGift
修飾子	「public」にチェックを入れる
スーパークラス	model.Gift

※これ以外の項目には，入力しません。

3.〈「新規 Java クラス」ウィンドウで入力する値（その3）〉

入力項目	入力値
ソース・フォルダー	Java_第３部/src
パッケージ	model
名前	WeddingGift
修飾子	「public」にチェックを入れる
スーパークラス	model.Gift

※これ以外の項目には，入力しません。

4.〈「新規 Java クラス」ウィンドウで入力する値（その4）〉

入力項目	入力値
ソース・フォルダー	Java_第３部/src
名前	Example342
修飾子	「パッケージ」にチェックを入れる
どのメソッド・スタブを作成しますか？	「public static void main(String[] args)」にチェックを入れる

※これ以外の項目には，入力しません。

② プログラム例 3-4-2 を参考にして，プログラムを入力します。入力後，Eclipse の ボタンを押して，プログラムを保存します。

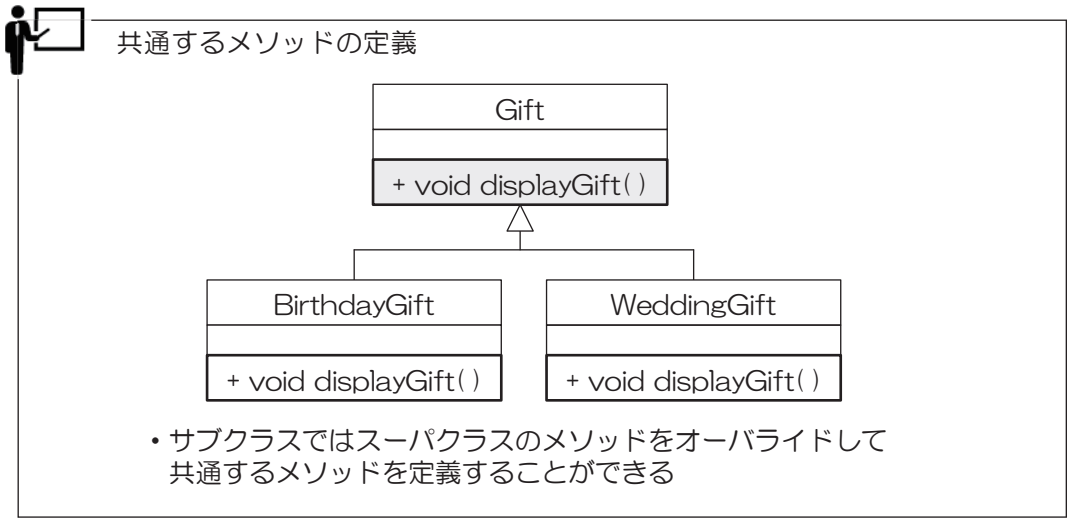
③ Eclipse の ボタンを押して，Example342 クラスを実行します。

例題 3-4-2 解説

(1) ポリモフィズムの実現

● **共通するメソッドの定義**

サブクラスではスーパクラスのメソッドをオーバライドします。これにより，サブクラスは共通するメソッドを定義することができます。



プログラム例 3-4-2 では，共通するメソッドの定義は次に示す BirthdayGift.java の６～９行目と，WeddingGift.java の６～９行目です。

BirthdayGift.java（抜粋）

```
6      public void displayGift() {
7          System.out.println("<バースデーギフト>");
8          super.displayGift();
9      }
```

WeddingGift.java（抜粋）

```
6      public void displayGift() {
7          System.out.println("<ウェディングギフト>");
8          super.displayGift();
9      }
```

(2) ポリモフィズムにおけるオブジェクトの利用方法

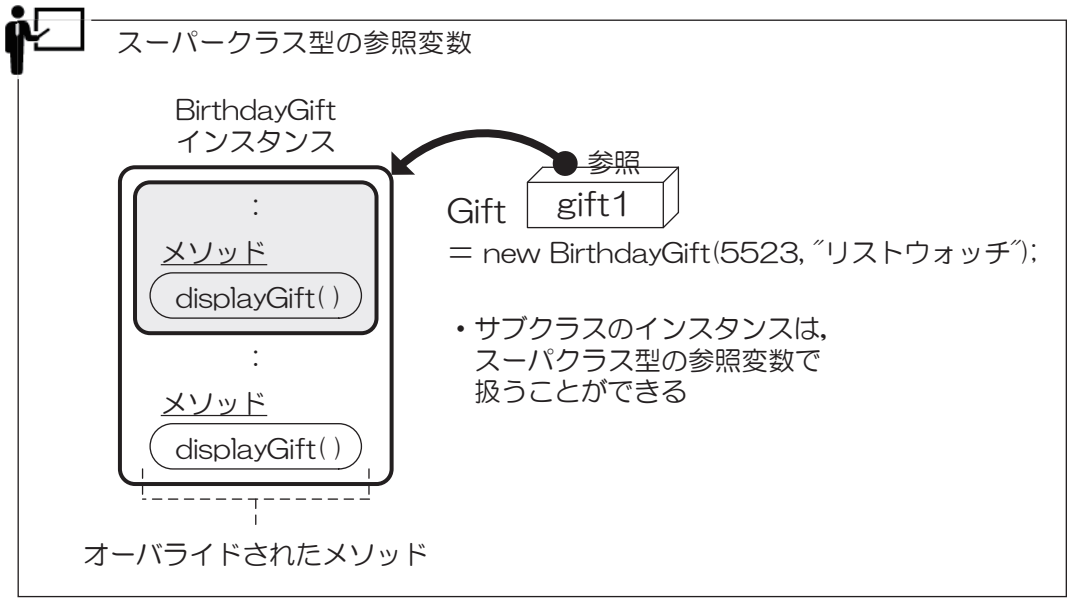
● スーパクラス型の参照変数

継承関係にあるサブクラスのインスタンスは，スーパクラス型の参照変数で扱うことができます。

プログラム例 3-4-2（Example342.java）では，スーパクラス型の参照変数は次に示す７，８行目です。

Example342.java（抜粋）

```
 7          Gift gift1 = new BirthdayGift(5523, "リストウォッチ");
 8          Gift gift2 = new WeddingGift(1122, "ティーカップ");
```



・インタフェースを実装したクラスのインスタンスは，インタフェース型の参照変数で扱うことができます。

MobilePhone インタフェース型の参照変数で，MobilePhone インタフェースを実装した Phone クラスのインスタンスを参照する場合は次のように記述します。

記述例（インタフェース型の参照変数）

```
MobilePhone phone = new Phone();
```

● ArrayList の利用

サブクラスのインスタンスは，共通のスーパクラス型で扱うことができるので，ArrayList を利用して一括管理することができます。

プログラム例 3-4-2（Example342.java）では，ArrayList インスタンスの生成は次に示す 9 行目です。

Example342.java（抜粋）

```
 9          ArrayList<Gift> list = new ArrayList<Gift>();
```

生成した ArrayList のインスタンスには，「<」と「>」の間に指定したクラスのインスタンスとサブクラスのインスタンスを追加することができます。

プログラム例 3-4-2（Example342.java）では，ArrayList インスタンスへの要素の追加は次に示す 10，11 行目です。

Example342.java（抜粋）

```
10          list.add(gift1);
11          list.add(gift2);
```

● 各要素のインスタンスを扱う拡張 for 文

拡張 for 文を利用して，ArrayList インスタンスの各要素が参照するインスタンスを扱うことができます。

・拡張 for 文は，ArrayList インスタンスの要素がある間「:（コロン）」の左側に宣言した変数に要素の参照を代入（上書き）して，繰返し処理を実行します。

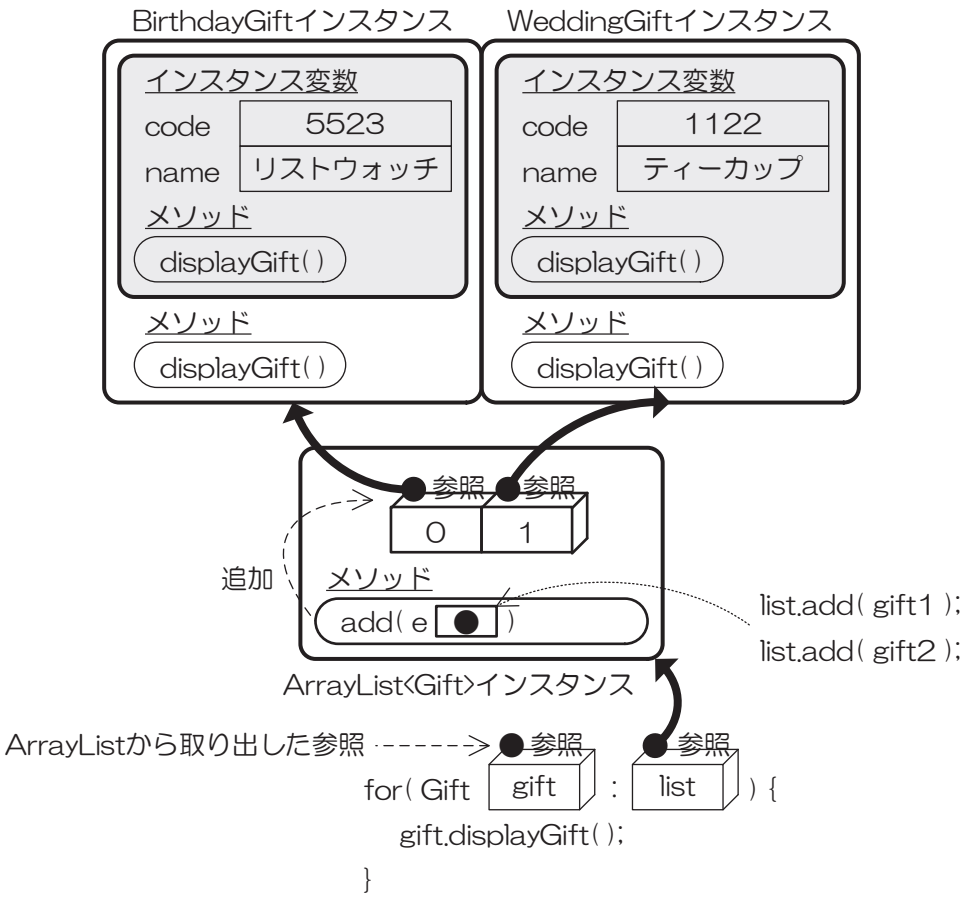
プログラム例 3-4-2（Example342.java）では，各要素のインスタンスを扱う拡張 for 文は次に示す 12～14 行目です。

Example342.java（抜粋）

```
12          for(Gift gift : list) {
13              gift.displayGift();
14          }
```

・スーパクラスである Gift クラスのメソッドをオーバライドしているため，共通のメソッドを呼び出すことができます。

各要素のインスタンスを扱う拡張 for 文



・拡張for文は，ArrayListの要素がある間インスタンスの参照を取り出し，「:（コロン）」の左側に宣言したGift型の参照変数に代入（上書き）して繰返し処理を実行する

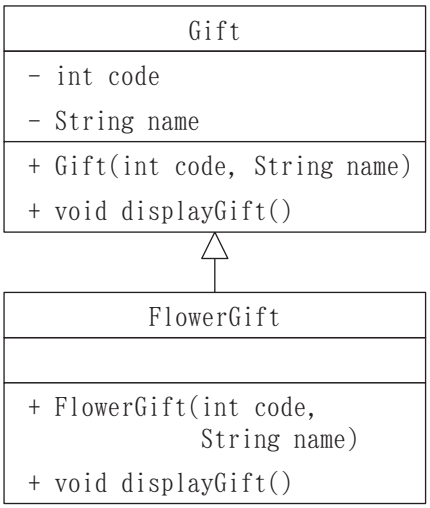
練習問題3-4-2

・次の実行結果になるように，プログラムを作成しなさい。
※例題 3-4-2 で定義した「Gift.java」，「BirthdayGift.java」，
「WeddingGift.java」を利用します。

実行結果

```
商品コード1 > 5523
商品名1 > リストウォッチ
商品コード2 > 1122
商品名2 > ティーカップ
商品コード3 > 1134
商品名3 > カーネーションのブーケ
<バースデーギフト>
ギフトコード：5523
ギフト名　　：リストウォッチ
<ウェディングギフト>
ギフトコード：1122
ギフト名　　：ティーカップ
<フラワーギフト>
ギフトコード：1134
ギフト名　　：カーネーションのブーケ
```

クラス図



クラス定義①

パッケージ名　：　model
クラス名　　：　FlowerGift
スーパクラス名　：　model.Gift

クラス定義②

クラス名　　：　Practice342
インポート　：　java.util.ArrayList, java.util.Scanner, model.Gift,
model.BirthdayGift, model.WeddingGift, model.FlowerGift
メソッド名　：　main メソッド

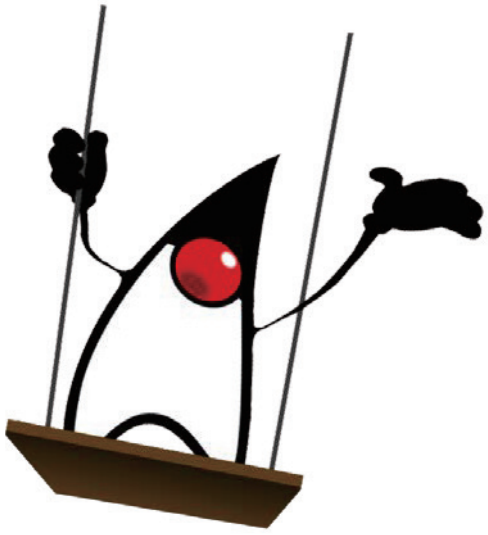
main メソッドの処理条件

１．文字列「商品コード１ ＞」を，ディスプレイに表示します。
２．変数 code1 に，キーボードから読み込んだ整数を代入します。
３．文字列「商品名１ ＞」を，ディスプレイに表示します。
４．変数 name1 に，キーボードから読み込んだ文字列を代入します。
５．BirthdayGift クラスのコンストラクタを呼び出し，生成した BirthdayGift クラスのインスタンスの参照を Gift 型の変数 gift1 に代入します。
　コンストラクタに指定する引数は次のとおりです。
　第一引数：　変数 code1
　第二引数：　変数 name1
６．文字列「商品コード２ ＞」を，ディスプレイに表示します。
７．変数 code2 に，キーボードから読み込んだ整数を代入します。
８．文字列「商品名２ ＞」を，ディスプレイに表示します。
９．変数 name2 に，キーボードから読み込んだ文字列を代入します。
10．WeddingGift クラスのコンストラクタを呼び出し，生成した WeddingGift クラスのインスタンスの参照を Gift 型の変数 gift2 に代入します。
　コンストラクタに指定する引数は次のとおりです。
　第一引数：　変数 code2
　第二引数：　変数 name2
11．文字列「商品コード３ ＞」を，ディスプレイに表示します。
12．変数 code3 に，キーボードから読み込んだ整数を代入します。
13．文字列「商品名３ ＞」を，ディスプレイに表示します。
14．変数 name3 に，キーボードから読み込んだ文字列を代入します。
15．FlowerGift クラスのコンストラクタを呼び出し，生成した FlowerGift クラスのインスタンスの参照を Gift 型の変数 gift3 に代入します。
　コンストラクタに指定する引数は次のとおりです。
　第一引数：　変数 code3
　第二引数：　変数 name3
16．ArrayList クラスのコンストラクタを呼び出し，生成した ArrayList クラスのインスタンスの参照を list に代入します。
17．変数 list が参照する ArrayList インスタンスの add メソッドを呼び出します。
　add メソッドに指定する引数は次のとおりです。
　引数：　変数 gift1
　※変数 gift2，gift3 についても同様の手順で，add メソッドの引数に指定します。
18．変数 list が参照する ArrayList インスタンスの要素がある間，次の処理を繰り返します。
　・ArrayList インスタンスから要素を取り出し，Gift 型の変数 gift に代入します。
　・変数 gift が参照するインスタンスの displayGift メソッドを呼び出します。

※例外が発生した場合は，文字列「エラーが発生しました！」をディスプレイに表示します。

索引

和文索引
欧文索引
記号索引



索引

和　文

－あ－

アクセス修飾子 ---------- 155
アクセッサ ---------- 158
アサーション ---------- 109
アノテーション ---------- 202
暗黙的型変換 ---------- 103

－い－

一次元配列 ---------- 50
一次元配列の初期化 ---------- 53
入れ子 ---------- 69
インクリメント演算子 ---------- 38
インスタンス ---------- 122, 128
インスタンス化 ---------- 122, 128
インスタンス変数 ---------- 123
インタフェース ---------- 215

－え－

永久ループ ---------- 85

－お－

オーバライド ---------- 201
オーバロード ---------- 108, 144
オブジェクト ---------- 116
オブジェクト指向 ---------- 116

－か－

カウンタ ---------- 83
カウントアップ ---------- 85
カウントダウン ---------- 85
拡張 for 文 ---------- 95
カプセル化 ---------- 117
可変長配列 ---------- 176
可変長引数 ---------- 109

－き－

基本データ型 ---------- 29
キャスト ---------- 103

－く－

クラス ---------- 22, 122
クラス変数 ---------- 136
繰返し構造 ---------- 82

－け－

継承 ---------- 186
継続条件 ---------- 82
ゲッタ ---------- 158

－こ－

コンストラクタ ---------- 141

－さ－

サブインタフェース ---------- 223
サブクラス ---------- 186
差分コーディング ---------- 188, 193
参照 ---------- 51, 128
参照型 ---------- 29, 128

－し－

実装 ---------- 215
情報隠蔽 ---------- 118, 156
真偽値型 ---------- 29

－す－

スーパインタフェース ---------- 186
スーパクラス ---------- 186

－せ－

整数型 ---------- 29
セッタ ---------- 158
選択構造 ---------- 66

－そ－

操作 ---------- 116
添字 ---------- 50
属性 ---------- 116

－た－

代入演算子 ---------- 30
多重継承 ---------- 187
単一継承 ---------- 187

－ち－
抽象メソッド ---------- 216

－て－
定数 ---------- 217
データ型 ---------- 28
デクリメント演算子 ---------- 38
デフォルトコンストラクタ ---------- 143

－に－
二次元配列 ---------- 58
二重ループ ---------- 94

－ね－
ネスト ---------- 69

－は－
配列 ---------- 50
パッケージング ---------- 163
反復構造 ---------- 82

－ひ－
比較演算子 ---------- 66
引数 ---------- 100
標準出力 ---------- 24

－ふ－
複合代入演算子 ---------- 37
浮動小数点数型 ---------- 29
ブロック ---------- 22

－へ－
返却値 ---------- 100
変数 ---------- 28

－ほ－
ポリモフィズム ---------- 228

－む－
無限ループ ---------- 85

－も－
文字型 ---------- 29
戻り値 ---------- 100

－よ－
要素 ---------- 50

－ら－
ラベル ---------- 94

－れ－
例外 ---------- 43
例外処理 ---------- 44

－ろ－
論理演算子 ---------- 75

欧　文

－A－
abstract ---------- 216
addメソッド ---------- 178
ArithmeticException ---------- 43
ArrayIndexOutOfBoundsException ------ 43
ArrayListクラス ---------- 176
assert ---------- 109

－B－
boolean ---------- 29
break文 ---------- 86
byte ---------- 29

－C－
case文 ---------- 76
catchブロック ---------- 44
char ---------- 29
class ---------- 22
continue文 ---------- 87

－D－
default文 ---------- 76
double ---------- 29
do～while文 ---------- 85

－E－
Eclipse ---------- 7
else文 ---------- 68
else if文 ---------- 74
equalsメソッド ---------- 67
Exception ---------- 44
extends ---------- 193,223

－F－
false ---------- 29,66
final ---------- 217
float ---------- 29
for文 ---------- 92

－G－
getメソッド ---------- 179
getter ---------- 158

－I－
if文 ---------- 66
import文 ---------- 164
implements ---------- 217
InputMismatchException ---------- 43
int ---------- 29
Integerクラス ---------- 45
interface ---------- 216
is-a関係 ---------- 187

－J－
JavaBeans仕様 ---------- 158
java.utilパッケージ ---------- 176

－L－
length ---------- 51
long ---------- 29

－M－
mainメソッド ---------- 23

－N－
new ---------- 50,128
nextメソッド ---------- 43
nextDoubleメソッド ---------- 43
nextIntメソッド ---------- 42,43
NumberFormatException ---------- 43
null ---------- 51,168
NullPointerException ---------- 43

－P－
package文 ---------- 163
parseIntメソッド ---------- 45
private ---------- 155,156
protected ---------- 155
public ---------- 155,157

－R－
return文 ---------- 100

－S－
Scannerクラス ---------- 42
setter ---------- 158
short ---------- 29
sizeメソッド ---------- 178
static ---------- 100,136
static変数 ---------- 136
String型 ---------- 29
super ---------- 203
super() ---------- 209
super. ---------- 203
switch文 ---------- 76
System.out.print ---------- 24
System.out.println ---------- 24

－T－
this ---------- 149
this() ---------- 150
this. ---------- 149
true ---------- 29,66
tryブロック ---------- 44

－V－
void ---------- 102

－W－
while文 ---------- 82

記　号

+ ---------- 31, 36, 155
++ ---------- 38
+= ---------- 37
- ---------- 36, 155
-- ---------- 38
-= ---------- 37
* ---------- 36, 164
*= ---------- 37
/ ---------- 36
/= ---------- 37
/**/ ---------- 32
// ---------- 32
% ---------- 36
%= ---------- 37
= ---------- 29
== ---------- 66
! ---------- 75
!= ---------- 66
< ---------- 66
<= ---------- 66
> ---------- 66
>= ---------- 66
<参照型> ---------- 177
&& ---------- 75
|| ---------- 75
---------- 155
~ ---------- 155
. ---------- 134
@Override ---------- 202

Java
プログラミング

解答例

第1部

[練習問題1-1-1]

Practice111.java

```
1  class Practice111 {
2      public static void main(String[] args) {
3          System.out.println("身長：170.5cm");
4          System.out.println("体重：60.5kg");
5          System.out.println("年齢：21歳");
6          System.out.println("趣味：スポーツ観戦");
7      }
8  }
```

[練習問題1-1-2]

Practice112.java

```
1  class Practice112 {
2      public static void main(String[] args) {
3          int month = 8;
4          int day = 11;
5          String message = "山の日";
6          System.out.print("今日は，" + month + "月");
7          System.out.println(day + "日");
8          System.out.println(message + "です。");
9      }
10 }
```

[練習問題1-2-1]

Practice121.java

```
1  class Practice121 {
2      public static void main(String[] args) {
3          int price = 1800;
4          int total = 0;
5          int count = 0;
6          count++;
7          total = price * count;
8          System.out.println(count + "人分料金：" + total + "円");
9          count++;
10         total = price * count;
11         System.out.println(count + "人分料金：" + total + "円");
12         count++;
13         total = price * count;
14         System.out.println(count + "人分料金：" + total + "円");
15     }
16 }
```

[練習問題1-2-2]

Practice122.java

```
1  import java.util.Scanner;
2  class Practice122 {
3      public static void main(String[] args) {
4          try {
5              Scanner sc = new Scanner(System.in);
6              System.out.print("金額を入力してください > ");
7              int money = sc.nextInt();
8              int bill = money / 10000;
9              int rem = money % 10000;
10             System.out.println("10000円紙幣：" + bill + "枚");
11             System.out.println("残り：" + rem + "円");
12             sc.close();
13         } catch(Exception e) {
14             System.out.println("エラーが発生しました！");
15         }
16     }
17 }
```

[練習問題1-3-1]

Practice131.java

```
1  import java.util.Scanner;
2  class Practice131 {
3      public static void main(String[] args) {
4          try {
5              int[] ticket = {1800, 1200, 600};
6              System.out.print("０：大人, １：高校生, ２：中学生 > ");
7              Scanner sc = new Scanner(System.in);
8              int num = sc.nextInt();
9              System.out.println("料金：" + ticket[num] + "円");
10             sc.close();
11         } catch(Exception e) {
12             System.out.println("エラーが発生しました！");
13         }
14     }
15 }
```

[練習問題1-3-2]

Practice132.java

```
1  import java.util.Scanner;
2  class Practice132 {
3      public static void main(String[] args) {
4          try {
5              int[][] price = {{6500, 7000, 8000},
6                               {7500, 8500, 9500},
7                               {10000, 11500, 14000}};
8              System.out.println("<レンタカー料金>");
9              System.out.print("０：コンパクト, １：スタンダード, ２：プレミアム > ");
10             Scanner sc = new Scanner(System.in);
11             int num = sc.nextInt();
12             System.out.print("０：６時間, １：12時間, ２：24時間 > ");
13             int hour = sc.nextInt();
14             System.out.println("料金：" + price[num][hour] + "円");
15             sc.close();
16         } catch(Exception e) {
17             System.out.println("エラーが発生しました！");
18         }
19     }
20 }
```

［練習問題 1-4-1］

Practice141.java

```
1  import java.util.Scanner;
2  class Practice141 {
3      public static void main(String[] args) {
4          try {
5              System.out.print("入力（1～10）> ");
6              Scanner sc = new Scanner(System.in);
7              int num = sc.nextInt();
8              if(num >= 1) {
9                  if(num <= 10) {
10                     System.out.println("入力値は範囲内です。");
11                 } else {
12                     System.out.println("入力値は範囲外です。");
13                 }
14             } else {
15                 System.out.println("入力値は範囲外です。");
16             }
17             sc.close();
18         } catch(Exception e) {
19             System.out.println("入力値は不正です。");
20         }
21     }
22 }
```

[練習問題1-4-2]

Practice142.java

```
1  import java.util.Scanner;
2  class Practice142 {
3      public static void main(String[] args) {
4          try {
5              System.out.println("<パスワード登録>");
6              System.out.print("整数４桁以上８桁以下 > ");
7              Scanner sc = new Scanner(System.in);
8              int pass = sc.nextInt();
9              if(pass >= 1000 && pass <= 99999999) {
10                 System.out.println("パスワードを登録しました。");
11             } else {
12                 System.out.println("もう一度やり直してください。");
13             }
14             sc.close();
15         } catch(Exception e) {
16             System.out.println("入力値は誤りです。");
17         }
18     }
19 }
```

[練習問題1-5-1]

Practice151.java

```
1  import java.util.Scanner;
2  class Practice151 {
3      public static void main(String[] args) {
4          try {
5              System.out.print("整数を入力 > ");
6              Scanner sc = new Scanner(System.in);
7              int num = sc.nextInt();
8              int i = 1;
9              int sum = 0;
10             while(i <= num) {
11                 sum = sum + i;
12                 i++;
13             }
14             System.out.println("1から" + num + "までの合計：" + sum);
15             sc.close();
16         } catch(Exception e) {
17             System.out.println("入力値は不正です。");
18         }
19     }
20 }
```

[練習問題1-5-2]

Practice152.java

```
1  import java.util.Scanner;
2  class Practice152 {
3      public static void main(String[] args) {
4          try{
5              int[] sales = new int[3];
6              Scanner sc = new Scanner(System.in);
7              for(int i = 0; i < sales.length; i++) {
8                  System.out.print("データ入力" + (i + 1) + "件目 > ");
9                  sales[i] = sc.nextInt();
10             }
11             int total = 0;
12             for(int i = 0; i < sales.length; i++) {
13                 System.out.println((i + 1) + "件目："+ sales[i]);
14                 total = total + sales[i];
15             }
16             System.out.println("合計：" + total);
17             sc.close();
18         } catch(Exception e) {
19             System.out.println("入力値は不正です。");
20         }
21     }
22 }
```

[練習問題1-6-1]

Practice161.java

```
1  import java.util.Scanner;
2  class Practice161 {
3      public static void main(String[] args) {
4          try{
5              Scanner sc = new Scanner(System.in);
6              System.out.print("ユーザーID：> ");
7              String userId = sc.next();
8              System.out.print("パスワード：> ");
9              int password = sc.nextInt();
10             String result = loginCheck(userId, password);
11             System.out.println(result);
12             sc.close();
13         } catch(Exception e) {
14             System.out.println("入力値は不正です。");
15         }
16     }
17     static String loginCheck(String id, int pass) {
18         if(id.equals("G1010") && pass == 8888) {
19             return "ログインしました。";
20         } else {
21             return "IDまたはパスワードが違います。";
22         }
23     }
24 }
```

[練習問題1-6-2]

Practice162.java

```
1  import java.util.Scanner;
2  class Practice162 {
3      public static void main(String[] args) {
4          try {
5              Scanner sc = new Scanner(System.in);
6              System.out.println("<ユーザー登録>");
7              System.out.print("氏名（必須）> ");
8              String name = sc.next();
9              System.out.print("年齢（0：登録しない）> ");
10             int age = sc.nextInt();
11             if(age != 0){
12                 register(name, age);
13             } else {
14                 register(name);
15             }
16             sc.close();
17         } catch(Exception e) {
18             System.out.println("入力値は不正です。");
19         }
20     }
21     static void register(String name) {
22         System.out.println("氏名：" + name + "を登録しました。");
23     }
24     static void register(String name, int age) {
25         System.out.println("氏名：" + name + "を登録しました。");
26         System.out.println("年齢：" + age + "を登録しました。");
27     }
28 }
```

第2部

[練習問題2-1-1]

会員情報オブジェクト

データ（属性）	
氏名	：神田　二郎
メールアドレス	：kanda@java.co.jp
登録年月日	：2014/04/01
会員年数	：6年
機　能（操作）	
会員年数を計算する	
会員情報を表示する	

[練習問題2-1-2]

Product. java

```
1  class Product {
2      String no;
3      String name;
4      int price;
5      int quantity;
6      void display() {
7          int total = price * quantity;
8          System.out.println("ナンバー：" + no);
9          System.out.println("製品名　：" + name);
10         System.out.println("価格　　：" + price + "円");
11         System.out.println("数量　　：" + quantity + "個");
12         System.out.println("合計金額：" + total + "円");
13     }
14 }
```

[練習問題2-2-1]

Practice221.java

```
1  class Practice221 {
2      public static void main(String[] args) {
3          Product pro = new Product();
4      }
5  }
```

[練習問題2-2-2]

Practice222.java

```
1  class Practice222 {
2      public static void main(String[] args) {
3          Product pro = new Product();
4          pro.no = "T001";
5          pro.name = "トートバッグ";
6          pro.price = 8000;
7          pro.quantity = 5;
8          pro.display();
9      }
10 }
```

[練習問題2-3-1]

Book.java

```
1  class Book {
2      String title;
3      String author;
4      int price;
5      Book(String t, String a, int p) {
6          title = t;
7          author = a;
8          price = p;
9      }
10     void display() {
11         System.out.println("書籍名：" + title);
12         System.out.println("著者　：" + author);
13         System.out.println("単価　：" + price + "円");
14     }
15 }
```

Practice231.java

```
1  class Practice231 {
2      public static void main(String[] args) {
3          Book b = new Book("みんなのJava", "デューク", 3000);
4          b.display();
5      }
6  }
```

[練習問題2-3-2]

Item.java

```
1  class Item {
2      int no;
3      String name;
4      int price;
5      Item(int no) {
6          this(no, "未登録", 0);
7      }
8      Item(int no, String name, int price) {
9          this.no = no;
10         this.name = name;
11         this.price = price;
12     }
13     void display() {
14         System.out.println("商品番号：" + no);
15         System.out.println("商品名　：" + name);
16         System.out.println("単価　　：" + price + "円");
17     }
18 }
```

Practice232.java

```
1  class Practice232 {
2      public static void main(String[] args) {
3          Item i1 = new Item(58);
4          i1.display();
5      }
6  }
```

[練習問題2-4-1]

Login.java

```
1  class Login {
2      private String mail;
3      private String pass;
4      public Login(String mail, String pass) {
5          this.mail = mail;
6          this.pass = pass;
7      }
8      public String getMail() {
9          return mail;
10     }
11     public String getPass() {
12         return pass;
13     }
14 }
```

Practice241.java

```
1  import java.util.Scanner;
2  class Practice241 {
3      public static void main(String[] args) {
4          try {
5              Scanner sc = new Scanner(System.in);
6              System.out.print("メールアドレス > ");
7              String mail1 = sc.next();
8              System.out.print("パスワード     > ");
9              String pass1 = sc.next();
10             Login log = new Login(mail1, pass1);
11             String mail2 = log.getMail();
12             String pass2 = log.getPass();
13             System.out.println("メールアドレス：" + mail2);
14             System.out.println("パスワード     :" + pass2);
15             sc.close();
16         } catch(Exception e) {
17             System.out.println("エラーが発生しました！");
18         }
19     }
20 }
```

[練習問題2-4-2]

Employee.java

```
1  package model;
2  public class Employee {
3      private int id;
4      private String name;
5      public Employee(int id, String name) {
6          this.id = id;
7          this.name = name;
8      }
9      public int getId() {
10         return id;
11     }
12     public String getName() {
13         return name;
14     }
15 }
```

Practice242.java

```
1  import java.util.Scanner;
2  import model.Employee;
3  class Practice242 {
4      public static void main(String[] args) {
5          try {
6              Scanner sc = new Scanner(System.in);
7              System.out.print("社員ID > ");
8              int id1 = sc.nextInt();
9              System.out.print("社員名 > ");
10             String name1 = sc.next();
11             Employee emp = new Employee(id1, name1);
12             int id2 = emp.getId();
13             String name2 = emp.getName();
14             System.out.println("社員ID：" + id2);
15             System.out.println("社員名：" + name2);
16             sc.close();
17         } catch(Exception e) {
18             System.out.println("エラーが発生しました！");
19         }
20     }
21 }
```

[練習問題2-5-1]

Practice251.java

```
1  import model.Employee;
2  class Practice251 {
3      public static void main(String[] args) {
4          Employee[] aryEmp = new Employee[3];
5          aryEmp[0] = new Employee(1111, "神田　二郎");
6          aryEmp[1] = new Employee(2222, "淡路　花子");
7          aryEmp[2] = new Employee(3333, "神保　三郎");
8          for(int i = 0; i < aryEmp.length; i++) {
9              int id = aryEmp[i].getId();
10             String name = aryEmp[i].getName();
11             System.out.println("社員ID：" + id);
12             System.out.println("社員名：" + name);
13         }
14         //拡張for文
15         /*
16         for(Employee emp : aryEmp) {
17             int id = emp.getId();
18             String name = emp.getName();
19             System.out.println("社員ID：" + id);
20             System.out.println("社員名：" + name);
21         }
22         */
23     }
24 }
```

[練習問題2-5-2]

Order.java

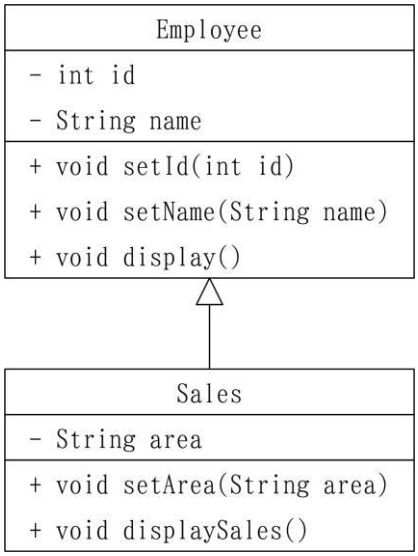
```
1  package data;
2  public class Order {
3      private int no;
4      private String custName;
5      public Order(int no, String custName) {
6          this.no = no;
7          this.custName = custName;
8      }
9      public int getNo() {
10         return no;
11     }
12     public String getCustName() {
13         return custName;
14     }
15 }
```

Practice252.java

```
1  import java.util.ArrayList;
2  import data.Order;
3  class Practice252 {
4      public static void main(String[] args) {
5          ArrayList<Order> list = new ArrayList<Order>();
6          Order o1 = new Order(780, "神田　二郎");
7          Order o2 = new Order(800, "淡路　花子");
8          Order o3 = new Order(813, "神保　三郎");
9          list.add(o1);
10         list.add(o2);
11         list.add(o3);
12         for(int i = 0; i < list.size(); i++) {
13             Order order = list.get(i);
14             int no = order.getNo();
15             String custName = order.getCustName();
16             System.out.println("注文番号：" + no);
17             System.out.println("顧客名　：" + custName);
18         }
19         //拡張for文
20         /*
21         for(Order order : list) {
22             int no = order.getNo();
23             String custName = order.getCustName();
24             System.out.println("注文番号：" + no);
25             System.out.println("顧客名　：" + custName);
26         }
27         */
28     }
29 }
```

第3部

[練習問題3-1-1]



[練習問題3-1-2]

Employee.java

```
1  package model;
2  public class Employee {
3      private int id;
4      private String name;
5      public void setId(int id) {
6          this.id = id;
7      }
8      public void setName(String name) {
9          this.name = name;
10     }
11     public void display() {
12         System.out.println("社員ID：" + id);
13         System.out.println("社員名：" + name);
14     }
15 }
```

Sales.java

```
1  package model;
2  public class Sales extends Employee {
3      private String area;
4      public void setArea(String area) {
5          this.area = area;
6      }
7      public void displaySales() {
8          System.out.println("担当地域名：" + area);
9      }
10 }
```

Practice312.java

```
1  import java.util.Scanner;
2  import model.Sales;
3  class Practice312 {
4      public static void main(String[] args) {
5          try {
6              Scanner sc = new Scanner(System.in);
7              System.out.print("社員ID     > ");
8              int id = sc.nextInt();
9              System.out.print("社員名     > ");
10             String name = sc.next();
11             System.out.print("担当地域名 > ");
12             String area = sc.next();
13             Sales sal = new Sales();
14             sal.setId(id);
15             sal.setName(name);
16             sal.display();
17             sal.setArea(area);
18             sal.displaySales();
19             sc.close();
20         } catch(Exception e) {
21             System.out.println("エラーが発生しました！");
22         }
23     }
24 }
```

[練習問題3-2-1]

Development.java

```
1  package model;
2  public class Development extends Employee {
3      private String project;
4      public void setProject(String project) {
5          this.project = project;
6      }
7      public void display() {
8          super.display();
9          System.out.println("プロジェクト名：" + project);
10     }
11 }
```

Practice321.java

```
1  import java.util.Scanner;
2  import model.Development;
3  class Practice321 {
4      public static void main(String[] args) {
5          try {
6              Scanner sc = new Scanner(System.in);
7              System.out.print("社員ID        > ");
8              int id = sc.nextInt();
9              System.out.print("社員名        > ");
10             String name = sc.next();
11             System.out.print("プロジェクト名 > ");
12             String project = sc.next();
13             Development dev = new Development();
14             dev.setId(id);
15             dev.setName(name);
16             dev.setProject(project);
17             dev.display();
18             sc.close();
19         } catch(Exception e) {
20             System.out.println("エラーが発生しました！");
21         }
22     }
23 }
```

[練習問題3-2-2]

Product.java

```
1  package model;
2  public class Product {
3      private String name;
4      private int price;
5      public Product(String name, int price) {
6          this.name = name;
7          this.price = price;
8      }
9      public String getName() {
10         return name;
11     }
12     public int getPrice() {
13         return price;
14     }
15 }
```

LimitedProduct.java

```
1  package model;
2  public class LimitedProduct extends Product {
3      private int quantity;
4      public LimitedProduct(String name, int price, int quantity) {
5          super(name, price);
6          this.quantity = quantity;
7      }
8      public void display() {
9          String name = super.getName();
10         int price = super.getPrice();
11         System.out.println("製品名   :" + name);
12         System.out.println("価格     :" + price + "円");
13         System.out.println("限定数量:" + quantity + "個");
14     }
15 }
```

Practice322.java

```
1  import java.util.Scanner;
2  import model.LimitedProduct;
3  class Practice322 {
4      public static void main(String[] args) {
5          try {
6              Scanner sc = new Scanner(System.in);
7              System.out.print("製品名   > ");
8              String name = sc.next();
9              System.out.print("価格     > ");
10             int price = sc.nextInt();
11             System.out.print("限定数量 > ");
12             int quantity = sc.nextInt();
13             LimitedProduct lim = new LimitedProduct(name, price, quantity);
14             lim.display();
15             sc.close();
16         } catch(Exception e) {
17             System.out.println("エラーが発生しました！");
18         }
19     }
20 }
```

[練習問題3-3-1]

MobilePhone.java

```
1  package model;
2  public interface MobilePhone {
3      public abstract void setAddress(String address);
4      public abstract void email();
5  }
```

Phone.java

```
1  package model;
2  public class Phone implements MobilePhone {
3      private String address;
4      public void setAddress(String address) {
5          this.address = address;
6      }
7      public void email() {
8          System.out.println("メールを送信しました。");
9          System.out.println("To：" + address);
10     }
11 }
```

Practice331.java

```
1  import java.util.Scanner;
2  import model.Phone;
3  class Practice331 {
4      public static void main(String[] args) {
5          try{
6              Scanner sc = new Scanner(System.in);
7              System.out.print("宛先 > ");
8              String address = sc.next();
9              Phone phone = new Phone();
10             phone.setAddress(address);
11             phone.email();
12             sc.close();
13         } catch(Exception e) {
14             System.out.println("送信できませんでした！");
15         }
16     }
17 }
```

[練習問題3-3-2]

SmartPhone.java

```
1  package model;
2  public interface SmartPhone extends MobilePhone{
3      public abstract void setURL(String url);
4      public abstract void webBrowser();
5  }
```

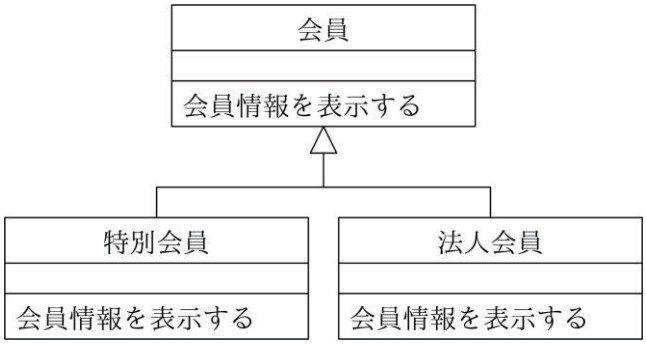
CellPhone.java

```
1  package model;
2  public class CellPhone implements SmartPhone {
3      private String address;
4      private String url;
5      public void setAddress(String address) {
6          this.address = address;
7      }
8      public void setURL(String url) {
9          this.url = url;
10     }
11     public void email() {
12         System.out.println("メールを送信しました。");
13         System.out.println("To ：" + address);
14     }
15     public void webBrowser() {
16         System.out.println("Webページにアクセスしました。");
17         System.out.println("URL：" + url);
18     }
19 }
```

Practice332.java

```
1  import java.util.Scanner;
2  import model.CellPhone;
3  public class Practice332 {
4      public static void main(String[] args) {
5          try{
6              Scanner sc = new Scanner(System.in);
7              System.out.print("宛先 > ");
8              String address = sc.next();
9              CellPhone phone = new CellPhone();
10             phone.setAddress(address);
11             phone.email();
12             System.out.print("アドレス > ");
13             String url = sc.next();
14             phone.setURL(url);
15             phone.webBrowser();
16             sc.close();
17         } catch(Exception e) {
18             System.out.println("エラーが発生しました！");
19         }
20     }
21 }
```

[練習問題3-4-1]



[練習問題3-4-2]

FlowerGift.java

```
1  package model;
2  public class FlowerGift extends Gift {
3      public FlowerGift(int code, String name) {
4          super(code, name);
5      }
6      public void displayGift() {
7          System.out.println("<フラワーギフト>");
8          super.displayGift();
9      }
10 }
```

Practice342.java

```
import java.util.ArrayList;
import java.util.Scanner;
import model.BirthdayGift;
import model.FlowerGift;
import model.Gift;
import model.WeddingGift;
class Practice342 {
    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);
            System.out.print("商品コード1 > ");
            int code1 = sc.nextInt();
            System.out.print("商品名1 > ");
            String name1 = sc.next();
            Gift gift1 = new BirthdayGift(code1, name1);
            System.out.print("商品コード2 > ");
            int code2 = sc.nextInt();
            System.out.print("商品名2 > ");
            String name2 = sc.next();
            Gift gift2 = new WeddingGift(code2, name2);
            System.out.print("商品コード3 > ");
            int code3 = sc.nextInt();
            System.out.print("商品名3 > ");
            String name3 = sc.next();
            Gift gift3 = new FlowerGift(code3, name3);
            ArrayList<Gift> list = new ArrayList<Gift>();
            list.add(gift1);
            list.add(gift2);
            list.add(gift3);
            for(Gift gift : list) {
                gift.displayGift();
            }
            sc.close();
        } catch(Exception e) {
            System.out.println("エラーが発生しました！");
        }
    }
}
```

令和 6 年度文部科学省委託「専修学校による地域産業中核的人材養成」事業
IT 分野 DX 人材養成のモデルプログラム開発と実証事業

情報 DX エンジニア教材資料
Java プログラミング

令和 7 年2月

一般社団法人全国専門学校情報教育協会
〒164-0003 東京都中野区東中野 1-57-8 辻沢ビル３F
電話：03-5332-5081 FAX.03-5332-5083

●本書の内容を無断で転記、掲載することは禁じます。