**第11章　メンバ変数とメンバメソッド**

**メンバ変数とメンバメソッドに関する基本的な考え方について説明します。**

**メンバ変数とメンバメソッド**

クラスは「メンバ変数」と「メンバメソッド」の要素で構成されています。

**1.メンバ変数とは**

メンバ変数とは、クラス内に直接書かれる変数のことで、クラス内のメソッドから参照可能な変数で、フィールド変数またはフィールドと呼ばれます。

以下の例では、2行目の「int channelNo;」がメンバ変数です。

[Television.java]

* class Television{
* int channelNo;
* void setChannel(int newChannelNo){
* channelNo = newChannelNo;
* }
* void dispChannel(){
* System.out.println("現在のチャンネルは " + channelNo + " です");
* }
* }

**1-1.メンバ変数の定義**

メンバ変数の定義は以下のように宣言します。

* [修飾子] データ型 変数名;
* 例：
* private int number;
* pribate Stirng name;
* private int price;
* private boolean stock;

メンバ変数は、int、doubleなどの基本型だけでなく、クラス型、配列型の変数も定義できます。

また、メンバ変数はオブジェクトの内部情報のため、[**private**](※1)をつけて非公開にします。

(※1)[private]は「宣言したクラスの中でだけしようできる」という意味の**アクセス修飾子**です。アクセス修飾子については次の第9章で学習します。

**1-2.メンバ変数に値を格納する**

メンバ変数が定義されている同じクラスの中で値を格納する場合は普通の変数と同じように扱うことができます。

以下の例では4行目の「set Channel」メソッドでほかの変数の値を格納していますが、

channelNo = 1;

のように値を格納することができます。

[Television.java]

* public class Television{
* int channelNo;
* void setChannel(int newChannelNo){
* this.channelNo = newChannelNo;
* }
* void dispChannel(){
* System.out.println("現在のチャンネルは " + channelNo + " です");
* }
* }

また5行目の部分の変数の記述は、メンバ変数が定義された同じクラス内の場合、「this」を省略して書くことが可能です。

さらに、メンバ変数が定義されているクラスの**オブジェクト**を使って値を格納することも可能です。

[Ctest.java]

* public class Ctest{
* public static void main(String args[]){
* Television tv1 = new Television();
* tv1.channelNo = 1;
* tv1.dispChannel();
* }
* }

この[Ctest]クラスでは、3行目で[Television]クラスをもとに[tv1]というオブジェクトを作り、そのオブジェクトを使ってメンバ変数に値を格納しています。

4行目のように「**オブジェクト名.メンバ変数名 = 値;**」と記述して、値を格納します。

* **メンバ変数とローカル変数の違い**

ローカル変数とは、メソッド内で定義される変数のことです。

このローカル変数の**有効範囲(スコープ)**はメソッド内だけになるため、別のメソッドからアクセスすることはできません。

また、同じスコープ内で同じ名前のローカル変数を定義することはできません。

[JmSaple.java]

* public class JSample {
* int memberVar = 10;
* void methodA() {
* int localVarA = 20;
* System.out.println("メンバ変数(memberVar)：" + this.memberVar);
* System.out.println("ローカル変数(localVarA)：" + localVarA);
* }
* void methodB() {
* int localVarB = 30;
* }
* }

メンバ変数は2行目で宣言されている[int memberVar]。  
ローカル変数は、5行目で「methodA」メソッド内に宣言されている[localVarA]と

11行目で「methodB」メソッド内に宣言されている[localVarB]です。

5行目の[memberVar]の出力は、メンバ変数なのでmethodAから参照可能となります。

もし仮に、6行目が以下のように書かれていた場合、[localVarB]は「methodA」内(4～8行目)に定義したローカル変数ではないのでmethodAから参照不可となります。

System.out.println("ローカル変数(localVarB)：" + localVarB);

**2.メンバメソッドとは**

メンバメソッドとはクラスの中で定義されているメソッドのことで、そのクラスの中で何かの動作をするためのものです。

メソッドを呼び出すときに引数を指定したり、メソッドからの戻り値を設定することもできます。

**2-1.メソッドの定義**

* [修飾子] 戻り値のデータ型 メソッド名(引数1, 引数2, ....){
* }

何も値を返さない場合は、戻り値のデータ型に**void型**を指定するという決まりになっています。そのため戻り値を返さない場合は、戻り値のデータ型を「**void**」と記述して下さい。

**例**

[JSample11\_1.java]

* package JmSaple;
* public class JSample11\_1 {
* public static void main(String[] args) {
* departure();
* arrive();
* annouce();
* }
* private static void departure() {
* System.out.println("出発します。");
* }
* private static void arrive() {
* System.out.println("到着します。");
* }
* private static void annouce() {
* departure();
* arrive();
* }
* }

**実行結果**

出発します。

到着します。

出発します。

到着します。

**2-2.メソッドの呼び出し**

メンバメソッドの呼び出し方は、メンバ変数の場合と同様に、同じクラス内から呼び出す場合は**「this.メンバメソッド名(引数)」**と記述し、

メンバメソッドが定義されているクラスの外からオブジェクトを使って呼び出す場合には**「オブジェクト名.メンバメソッド名(引数)」**の形式で呼び出します。

下記の例に、mainメソッドで、 myMethod()メソッドを呼び出します。

**例**

[JSample11\_2.java]

* package JSample;
* public class JSample11\_2 {
* static void myMethod() {
* System.out.println("Hello World!");
* }
* public static void main(String[] args) {
* myMethod();
* }
* }

**実行結果**

Hello World!

**・引数を使ってメソッドに値を渡す**  
メソッドを呼び出す時に呼び出し元から値をメソッドに渡すことができます。

**例**

[JSample11\_3.java]

* package JSample;
* public class JSample11\_3 {
* public static void main(String[] args) {
* int english = 58;
* check("英語", english);
* }
* private static void check(String language, int score) {
* System.out.print("この応募者の" + language + "能力は");
* if (score > 70) {
* System.out.println("ビジネスレベルです。");
* } else {
* System.out.println("日常会話レベルです。");
* }
* }
* }

**実行結果**

この応募者の英語能力は日常会話レベルです。

この例では、6行目で「check」メソッドを呼び出しています。

「check」メソッドは9～16行目で、「string型」の「"英語"」という値を[String language]に、

「int型」の「english」という値を[int score]にそれぞれ渡し、処理が行われます。

この場合、[score]に渡された[english]という変数には「58」という値が代入されていたため、if文関係式のfalseとなり、戻り値は「"日常会話レベルです。"」になります。

**・return文**

return文は、メソッドを終了するとともに、戻り値を指定する働きがあります。

下の(1)の例では、10行目でreturn文が使用されています。ここで「multiply」メソッドを終了し、[n \* 2]を戻り値として返しています。

**（１）** メソッドから呼び出し元に値を返す場合。

**例**

[JSample11\_4.java]

* package JSample;
* public class JSample11\_4 {
* public static void main(String[] args) {
* int result;
* result = multiply(5);
* System.out.println(result);
* }
* private static int multiply(int n) {
* return n \* 2;
* }
* }

**実行結果**

10

この例では、6行目で[result]という変数に、「multiply」メソッドに「5」という値を渡した結果の戻り値を返すという指示を与えています。

6行目で指定したint型の「5」という値は、multiplyメソッドの「int n」に代入されます。multiplyメソッドでは、そのnの値を2倍し(10行目)、その値を返し、6行目の[result]という変数に代入します。

**（２）** return文がメソッドの中で実行されると、それ以降の処理を実行せずに呼び出し元へ処理を移すことができます。

**例**

[JSample11\_5.java]

* package JSample;
* public class JSample11\_5 {
* public static void main(String[] args) {
* int data[];
* test(15, 4);
* test(7, 0);
* }
* private static void test(int n1, int n2) {
* if (n2 == 0) {
* System.out.println("0で割ることはできません");
* return;
* }
* System.out.println(n1 + " / " + n2 + " = " + (n1 / n2));
* return;
* }
* }

**実行結果**

15 / 4 = 3  
0で割ることはできません

この例では「test」メソッドを、6行目の「(15, 4)」と7行目の「(7, 0)」の値で利用しています。

「(15, 4)」の値で「test」メソッドを実行した際、[n2]には[4]が入るため、10行目のif文の関係式には該当せず、if文のブロックの処理は行われず(10～13行目)、15行目の出力処理が行われます。

そのため実行結果の1行目のように出力されます。

次に「(7, 0)」の値で「test」メソッドが実行され、10行目のif文の関係式に該当するため、if文のブロックの処理が行われます(10～13行目)。

このとき13行目の[return]でメソッドを終了し値が返されるため、15行目以降の処理はされ、ません。

したがって実行結果の2行目のように出力されます。

**3.コンストラクタ**

コンストラクタとは、クラスのインスタンス生成時に実行されるメソッドで、主にそのクラスの**メンバ変数を初期化する**ときに使用し、インスタンス生成時、必ず最初に呼び出されます。

以下の例では3～5行目までがコンストラクタにあたります。

* public class Calculate {
* private double tax;
* public Calculate (double x) {
* this.tax = x;
* }
* ----- その他の処理 -----
* }

**3-1.コンストラクタの定義**

コンストラクタの定義は下記のように宣言します。

コンストラクタ名はクラス名と同じにします。

* [アクセス修飾子]　コンストラクタ名　(引数型 引数名)

**3-2.複数のコンストラクタ**

コンストラクタは、引数の型や引数の数を変更することで1つのクラス内に複数宣言することができます。

具体的には、コンストラクタ内でいくつかのメンバ変数を初期化する際、

指定する初期値がある場合は指定し

指定する初期値がない場合はデフォルト値を指定する

というように、初期化処理を使い分けたい場合にコンストラクタを複数宣言します。

どのコンストラクタが呼び出されるかは呼び出し元の引数の型、引数の数により決定されます。

**例**

[SetNumber.java]

* package JSample;
* public class SetNumber {
* private int basicNum;
* private int specialNum;
* public SetNumber(int spn) {
* this.basicNum = 100;
* this.specialNum = spn;
* }
* public SetNumber() {
* this.basicNum = 100;
* this.specialNum = 500;
* }
* public static void main(String[] args) {
* SetNumber insA = new SetNumber();
* System.out.println("insAのSetNumber()でbasicNumは" + insA.basicNum+"です");
* System.out.println("insAのSetNumber()でspecialNumは" + insA.specialNum+"です");
* SetNumber insB = new SetNumber(1000);
* System.out.println("insBのSetNumber(1000)でbasicNumは" + insB.basicNum+"です");
* System.out.println("insBのSetNumber(1000)でspecialNumは" + insB.specialNum+"です");
* }
* }

**実行結果**

insAのSetNumber()でbasicNumは100です  
insAのSetNumber()でspecialNumは500です  
insBのSetNumber(1000)でbasicNumは100です  
insBのSetNumber(1000)でspecialNumは1000です

上記の例では、引数を持つコンストラクタ(7～10行目)と、引数を持たないコンストラクタ(12～15行目)の2つのコンストラクタが宣言されています。

18行目は、引数を指定していないため、12～15行目の引数を持たないコンストラクタが呼び出され、実行結果の1～2行目のように初期値が設定されます。

22行目は、int型の引数を1つ指定しているため、7～10行目のコンストラクタが呼び出され、実行結果の3～4行目のように初期値が設定されます。

**練習**

**問題**[JEx11\_1.java]

Numクラス(Num.java)を使用して、2 整数の比較は最大値を取ります、次のプログラムを完成しなさい。

* package JSample;
* public class Num {
* public int test(int n1, int n2) {
* //ToDo
* }
* }

JEx11\_1クラスを使用して、次のプログラムを完成しなさい。

* package JSample;
* import java.util.Scanner;
* public class JEx11\_1 {
* public static void main(String[] args) {
* Scanner scan = new Scanner(System.in);
* System.out.println("キーボードから数値1を入力してください。");
* int n1 = scan.nextInt();
* System.out.println("キーボードから数値2を入力してください。");
* int n2 = scan.nextInt();
* Num num = new Num();
* int k = num.test(n1, n2);
* System.out.println(n1 + " と " + n2 + " 比較的な最大値は：" + k);
* }
* }