入門編で覚えて欲しいこと

プログラミングにはいくつか書き方のルールがあります。

それほど多くないので、入門編を学習しながら、しっかり覚えておきましょう。

ここで紹介している内容をちゃんと覚えているか、なんとなく使っているかで、プログラムが書けるかどうかが変わってきます。

変数宣言①

変数は宣言していないと使えません。

■書式
変数の型 変数名1[, **変数名2**, **変数名3**…];　　※[]内は省略可能

■使用例
// int(数値)型の変数 a を宣言
int a;

// String(文字列)型の変数 b を宣言
String b;

// int(数値)型の変数 a と b と c を宣言
int a, **b**, **c**;

// Randomクラスの変数 rand を宣言
Random rand;

同じ変数の型であれば、複数の変数を
まとめて宣言することができます

クラス(Class)も変数の型に
することができます

変数宣言②

変数は宣言と同時に、初期値を代入することができます。

■書式

```
変数の型 変数名 = 初期値;
```

■使用例

```
// int(数値)型の変数 a を宣言して、初期値を 10 にする
int a = 10;

// String(文字列)型の変数 b を宣言して、初期値を “山田” にする
String b = “山田”;

// Randomクラスの変数 rand を宣言して、初期値としてインスタンスを生成する
Random rand = new Random();
```

クラス(Class)の場合は、new で新しいインスタンスを生成することが多いです

※クラスに関しては中級以降なので、それはまで、そういうものかーという認識でも構いません

if文①

if 文を使うことで、条件によって行う処理を変えることができます。

■書式

```
if (条件式) {
    条件式が満たされている場合の処理
} else {
    条件式が満たされていない場合の処理
}
```

※ else は省略可能

■使用例

```
if (a < 10) {
    System.out.println(“aは10より小さい”);
}
```

```
if (a < 10) {
    System.out.println(“aは10より小さい”);
} else {
    System.out.println(“aは10以上”);
}
```

if文②

条件式には「または(OR条件)」や「かつ(AND条件)」といった、組み合わせの条件式を書くこともできます。

■書式

```
if (条件式1 || 条件式2) {
    条件式１または条件式２が満たされている場合の処理
} else {
    条件式１も条件式２も満たされていない場合の処理
}
```

■使用例

```
if (animal == “犬” || animal ==“猫”) {
    System.out.println(“犬または猫”);
}

if (animal == “犬” || animal ==“猫”) {
    System.out.println(“犬または猫”);
} else {
    System.out.println(“犬でも猫でもない”);
}
```

■書式

```
if (条件式1 && 条件式2) {
    条件式１と条件式２の両方が満たされている場合の処理
} else {
    条件式１と条件式２の両方が満たされていない場合の処理
}
```

■使用例

```
if (job == “会社員” && age >= 40) {
    System.out.println(“会社員で40歳以上”);
}

if (job == “会社員” && age >= 40) {
    System.out.println(“会社員で40歳以上”);
} else {
    System.out.println(“会社員ではないか、40歳以上ではない”);
}
```

複雑になるので、最初は使わなくても構わないですが、|| や && は複数記述することもできることは覚えておきましょう。

```
// aが10より小さくて、bは4か5の場合
if (a < 10 && (b == 4 || b == 5)) { … }
```

if文③

else if を使うと、複数の条件判定を連続で行うことができます。

■使用例

```
if (animal == “犬”) {
    System.out.println(“犬です”);
} else if (animal == “猫”) {
    System.out.println(“猫です”);
} else if (animal == “猿”) {
    System.out.println(“猿です”);
} else {
    System.out.println(“犬でも猫でも猿でもありません”);
}
```

if文④

if をネストさせて(重ねて)使うと、複雑な条件判断もできます。

■使用例

```
// job は職業、age は年齢、married は既婚かどうかのフラグ
if (job == “会社員”) {
    if (age <= 25) {
        System.out.println(“25歳以下の会社員”);
    } else if (age <= 40) {
        if (married == true) {
            System.out.println(“26～40歳の既婚の会社員”);
        }  else {
            System.out.println(“26～ 40歳の未婚の会社員”);
        }
} else {
        System.out.println(“41歳以上の会社員”);
    }
} else {
    System.out.println(“会社員ではない”);
}
```

for文

主に回数での繰り返し処理を行う場合は for 文を使います。

■書式

```
for (初期化処理; 繰り返し条件; 繰り返し時処理) {
    繰り返し条件が満たされている場合の処理
}
```

①初期化処理
②繰り返し条件の判定
③繰り返し条件が満たされている場合の処理
④繰り返し時処理
以下、②～④を繰り返す（②の条件が満たされなくなったら終了）

■使用例

```
// 1から10までを出力するプログラム
for (int i = 1; i <= 10; i++) {
    System.out.println(i);
}
```

具体例は紹介しませんが、初期化処理、繰り返し条件、繰り返し時処理は省略することができるということは覚えておきましょう。

while文①

主に条件での繰り返し処理を行う場合は while 文を使います。

■書式

```
while (条件式) {
    条件式が満たされている場合の処理
}
```

■使用例

```
// 所持金(money)が無くなるまで 100 円づつ使う
int money = 500;
while(money >= 100)
{
    System.out.println(“100円使いました”);
    money -= 100;
}
System.out.println(“お金が無くなりました”);
```

while文②

条件式を true にすると、条件式が常に満たされるので、ループから抜け出すことが無くなり、
無限ループとなります。

■書式

```
while (true) {
    処理
}
```

■使用例

```
// “無限ループ”が無限に表示されます
while(true)
{
    System.out.println(“無限ループ”);
}
```

while文③

break 文を使うことで、直前の while もしくは for の繰り返し処理から強制的に抜け出すことができます。
if 文の中に記述されていても、直前の繰り返し処理から抜け出すことができます。
抜け出せる繰り返し処理は直前の１つのみです。

■使用例

```
// 1、2、3を表示して while を抜ける
int count = 1;
while(true) {
    System.out.println(count);
    count++;
    if (count == 3) {
        break;
    }
}

// 1、2、3を表示して for を抜ける
for (int i = 1; i < 100; i++) {
    System.out.println(count);
    if (i == 3) {
        break;
    }
}
```

ArrayList

ArrayListを使うと、要素の数が変動するデータの管理ができます。

■使用例

```
// 文字列を管理する ArrayList の変数を生成する
ArrayList<String> textList = new ArrayList<String>();

// 要素を追加する：add() メソッド
textList.add(“犬”);
textList.add(“猫”);
textList.add(“猿”);

// 要素数を表示する：size() メソッド
System.out.println(“要素数は” + textList.size() + “です”);

// 要素を取得する：get() メソッド
System.out.println(“最初の要素は「” + textList.get(0) + “」です”);

// 要素を変更する：set() メソッド
textList.set(0, “Dog”);      // 0番目の要素“犬” を “Dog” に変更

// 要素を削除する：remove() メソッド
textList.remove(“猿”);      // “猿”をリストから削除する
```

ここで紹介している操作は最低限覚えておきましょう。

メソッド

メソッドを定義することで、まとまった１つの処理をまとめて行うことができるようになります。
メソッドの書式をしっかり覚えましょう。

■書式

```
戻り値の型 メソッド名([引数の型1 引数名1,引数の型2 引数名2,…]) {      ※[]内は省略可能
    処理
    return 戻り値;
}
```

■使用例

```
// 引数なし、戻り値なしの foo1 メソッド
void foo1()
{
    // 戻り値がなし(void)の場合は return は省略可
}

// 引数は整数型 a、戻り値なしの foo2 メソッド
void foo2(int a)
{
    // 戻り値がなし(void)の場合は return は省略可
}

// 引数は文字列型 s、整数型 n と m
// 戻り値は文字列型の foo3 メソッド
String foo3(String s, int n, int m)
{
    return “abc”;
}
```