制御の流れ２

前回までに「条件判断処理」を習った。今回からは制御の流れを変更するもう一つの処理である「ループ処理」を習っていく。併せて、多重ループについても学ぶ。

ループ処理（繰り返し処理）

ループ処理はある条件を定め、その条件が成立している間は繰り返し処理を行う方法である。日常的な出来事に照らし合わせれば、例えば、

「お金がある限り買い物を続ける」
「品数が 10 品になるまで買い物を続ける」

といった事柄が繰り返し処理に当たる。Java でこれらの繰り返し処理を実現する文として、**while 文**、**for 文**、**do-while 文**といった文が用意されている。

「お金がある限り」や「品数が 10 品になるまで」などのループを続けるための条件のことを継続条件という。これらの継続条件の判定方法には**見張り方式**と**カウンタ方式**がある。見張り方式はある条件が満たされている間は処理を繰り返す方式であり、「お金がある限り」や「品数が 10 品になるまで」などがこの方式の継続条件に当たる。また、カウンタ方式ではループ内にループした数をカウントする変数を設定し、この変数を用いて継続条件を決める方式である。そのため、「品数が 10 品になるまで」といった条件はカウンタ方式でも示すことができる。そのため、繰り返す数があらかじめ決まっている場合にはカウンタ方式が一般的に利用される。

while 文、for 文、do-while 文の書式とフローチャートを以下に示す。

書式

while(**条件式**){
　　文；
}

for(**式 1**； **式 2**； **式 3**){
　　文；
}

do{ 　　**文**； } while(**条件式**)；

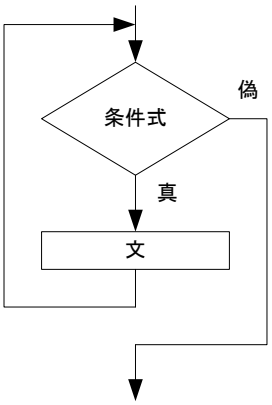

図１　while 文

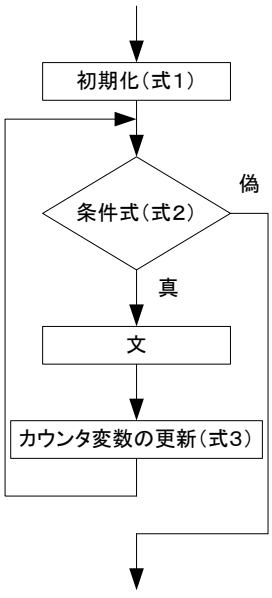

図２　for 文

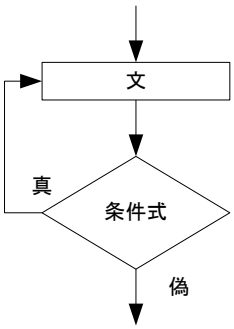

図３　do-while 文

while 文

while 文は「**条件式が“真”であれば本体を実行し続ける。条件式が“偽”になれば終了する**」という処理に使われる。そのため、前判定繰り返しともいう。また、本体とは while に続く{}で囲まれたブロックに記述された文の総称であり、while 文に限らず、for 文や do-while 文でも同様である。

以下に使用例を示す。

プログラム例１

```
import java.util.Scanner;

public class Prog05_01
{
        public static void main(String[ ] args)
        {
                Scanner stdIn = new Scanner(System.in);

                int num = -1;
                int total = 0;

                while( num != 0 )
                {
                        System.out.print("数値を入力してください:");
                        num = stdIn.nextInt();
                        total = total + num;
                }

                System.out.println("入力した値の合計値は" + total + "です");
        }
}
```

使用例のプログラムはキーボードから数値を入力し、入力された数値の合計値を求めて表示するプログラムである。このプログラムではキーボードから 0 を入力すると、total = total + num ; が実行された後にループが終了し、そこまでに入力された数値の合計が出力される。

このプログラムで重要なのは、初期化と呼ばれる次の処理である。

```
int num = -1;
int total = 0;
```

変数は宣言されると値を格納される場所が確保されるが、その中身である値はコンパイラによって自動的に決まる。そのため、値を入れておかなければ、異常な値からプログラムがスタートしてしまう。このプログラムの場合、

```
while( num != 0 )  や  total = total + num;
```

で変数を利用しているので、あらかじめ値を入れておかないと正しく動作しない。

for 文

for 文は繰り返しの回数が予め分かっているときに一般的に使われる。「**初期値は〇〇、条件式が“真”であれば本体を実行し、条件を再設定して繰り返す。条件式が“偽”になれば終了する**」という構文に使われる。これも前判定繰り返しである。

以下に使用例を示す。

プログラム例２

```
import java.util.Scanner;

public class Prog05_02
{
	public static void main(String[ ] args)
	{
		Scanner stdIn = new Scanner(System.in);

		int i;
		int num;
		int total = 0;

		for( i = 0; i < 5; i++ )
		{
			System.out.print("数値を入力してください:");
			num = stdIn.nextInt();
			total = total + num;
		}

		System.out.println("入力した値の合計値は" + total + "です");
	}
}
```

このプログラムは while 文を用いて以下のように書き換えることができる。

プログラム例３

```
import java.util.Scanner;

public class Prog05_03
{
	public static void main(String[ ] args)
	{
		Scanner stdIn = new Scanner(System.in);

		int i;
		int num;
		int total = 0;

		i = 0;                          /* for文の式1にあたる処理 */
		while(i < 5)                    /* for文の式2にあたる処理 */
		{
			System.out.print("数値を入力してください:");
			num = stdIn.nextInt();
			total = total + num;
			i++;                        /* for文の式3にあたる処理 */
		}

		System.out.println("入力した値の合計値は" + total + "です");
	}
}
```

このプログラムの下線を引いた部分の処理を確認すると、それぞれが for 文の式１から式３の処理とまったく同じ役割を果たしていることがわかる。この for 文や while 文で用いた変数 i はルー

プの実行回数の管理に用いるため、**カウンタ変数**と呼ばれる。

do-while 文

これは後判定繰り返しで、あらかじめ繰り返し回数が分からないときに使われることが多い。「**本体を実行し、条件式が“真”であれば本体を実行し続ける。条件式が“偽”になれば終了する**」という構文に使われる。よって、do-while 文は繰り返しをする回数が決められていなくて、最低１回は繰り返すべき処理を行う場合に使用すると便利である。

以下に書式と使用例を示す。

プログラム例４

```
import java.util.Scanner;

public class Prog05_04
{
        public static void main(String[ ] args)
        {
                Scanner stdIn = new Scanner(System.in);

                int num;
                int total = 0;

                do
                {
                        System.out.print("数値を入力してください:");
                        num = stdIn.nextInt();
                        total = total + num;
                }while(num != 0);                    /* 注：最後にセミコロンをつける */

                System.out.println("入力した値の合計値は" + total + "です");
        }
}
```

着目すべきところはやはり初期化部分の

```
        int num;
        int total = 0;
```

である。while 文の場合には、

```
        int num = -1;
        int total = 0;
```

であった。do-while文では先に本体が実行されるので、num = stdIn.nextInt();で値を入力し、合計を求める処理を行ってから、継続条件の判定を行うため、初期化を行わずにint num;としても、プログラム上問題無い。それではdo-while文を用いてfor文のような処理を行わせた処理を以下に示す。

プログラム例５

```
import java.util.Scanner;

public class Prog05_05
{
        public static void main(String[ ] args)
```

```
        {
                Scanner stdIn = new Scanner(System.in);

                int i;
                int num;
                int total = 0;

                i = 0;                              /* for文の式1にあたる処理 */
                do
                {
                        System.out.print("数値を入力してください:");
                        num = stdIn.nextInt();
                        total = total + num;
                        i++;                        /* for文の式3にあたる処理 */
                }while(i < 5);                      /* for文の式2にあたる処理 */

                System.out.println("入力した値の合計値は" + total + "です");
        }
}
```

多重ループ

条件分岐処理のところでネスト（入れ子）を習ったが、ループ処理でも同じくネストを作ることができる。while 文、for 文、do-while 文によるループの内部にさらにループ構造を作ることによって、多重ループを作成する。この多重ループは次に示すように用いる。ここに示した例では２重ループであるが、もちろん３重以上のループも作成できる。また、while 文、for 文、do-while 文は自由に組み合わせることができる。

書式

```
        for(式１；式２；式３)
        {
                for(式４；式５；式６)
                {
                文１；
            }
        }
```

プログラム例６

```
        for(int i=0; i<5; i++)
        {
                for(int j=0; j<3; j++)
                {
                        System.out.println(i + "+" + j + "=" + (i+j));
                }
                System.out.println();
        }
```

演習

テキストで示した使用例６つ全て作成し、実行結果を確認しなさい。プログラム例６については動作可能なプログラムにして実行する。

課題 3

次のプログラムを作成しなさい。

Kadai3_1　for 文を用いて、1～1000 までの合計値を求め、表示する。
Kadai3_2　while 文を用いて、1～1000 までの合計値を求め、表示する。
Kadai3_3　do while 文を用いて、1～1000 までの合計値を求め、表示する。

Kadai3_4　以下のような出力をする掛け算九九の表を作成する。ただし、表示する数字は以下の出力例のように揃え、見栄えを良くすること。

出力例

```
  1  2  3  4  5  6  7  8  9
  2 ･･･････････････････････ 18
  3 ･･･････････････････････ 27
  4 ･･･････････････････････ 36
  5 ･･･････････････････････ 45
  6 ･･･････････････････････ 54
  7 ･･･････････････････････ 63
  8 ･･･････････････････････ 72
  9 ･･･････････････････････ 81
```

課題 3 には続きがある。