2-5.制御文(ループ文)

提出する 評定を受ける

ループ文

はじめに

繰り返すこと、と言えば意味はスッと入るかと思います。
朝が来て夜が来る、また次の日には、朝が来て夜が来る、、、
そんな大きなくくりであったり、
水道の蛇口をひねったままにすれば水が出続けるのであったり、
扇風機や観覧車が回っているのであったり。

普段は意識していないかもしれませんが、私たちの身近にある機能になります！

Step1: 概念を知る

ループ処理の主な機能は、 **指定された回数分や特定の条件下にある間、処理を実行し続けること** です。

Javaにおいては、何度も同じ処理を繰り返したい場合に、
繰り返しの処理が制御文内で書くことができるfor文、while文という構文が用意されています。

Javaに携わる上では頻繁に使用する機能になります。
さっそく見ていきましょう！

Step2: 【繰返しの制御①】　for文

for文は、ループカウンタなどを使用して、定められた回数だけ同じ処理を繰り返したいときに使用します。
ループカウンタと、添え字による配列要素へのアクセスは非常に相性がいいため、配列の中身を単純に１つ１つ参照する場合などによく使用されます。

```
for (初期化式; 条件式; 変化式) {
    /* ··· 繰り返し行う処理 ··· */
}
```

【for文を使ったプログラム例】

```
public class Main {

    public static void main(String[] args) {

        // int配列の宣言と初期化
        int[] arr = {5, 8, 2, 6};

        // 配列arrの中身を全て表示する
        for (int i = 0; i < arr.length; i++) {
            System.out.println("arr[" + i + "] … " + arr[i]);
        }

    }
}
```

【実行結果】

```
arr[0] … 5
arr[1] … 8
arr[2] … 2
arr[3] … 6
```

解説

i は 0 から始まり、for文内の処理（println()による表示）を
実行しながら 1 ずつ増えていき、arr.length（＝4） に
なったところでループを終了する。
※lengthは配列内の要素数を指定できる **属性(プロパティ)** です。

Step3-1:【繰返しの制御②】　while文

ループ文では、for文以外にも、while文というものが存在します。

```
while (継続条件) {
   /* ··· 繰り返し行う処理 ··· */
}
```

【1から10までを出力させるwhile文】

```
public class Main {

    public static void main(String[] args) {

        // 変数の初期化
        int i = 1;

        //1から10までの数字を出力
        while (i  <= 10){
          System.out.println( i );
          i++;
        }

    }
}
```

解説

while (i <= 10)の()内が継続条件になる。
今回の場合だとiが10以下の場合、処理が継続される。

注意点

i++;　←ここを忘れると **無限ループ** になってしまうので注意しましょう。
※無限ループについては下記で解説しております。

【while文を使ったプログラム例 （WhileSample.java）】

```
import java.util.Scanner;

public class WhileSample {
    public static void main(String[] args) {
        int inputNum = 0;       // ユーザ入力値
        int sum = 0;        // 合計値

        // ユーザ入力読み込みオブジェクトを生成
        Scanner scanner = new Scanner(System.in);

        // 初回の数値入力
        System.out.print("数値入力（-1で終了）：");
        inputNum = scanner.nextInt();

        // ユーザから「-1」が入力されるまで、入力された数値をsum変数に加算
        while (inputNum != -1) {
            // 入力値をsumに加算
            sum += inputNum;

            // ２回目以降の数値入力
            System.out.print("数値入力（-1で終了）：");
            inputNum = scanner.nextInt();
        }

        // 合計値を表示
        System.out.println("入力された数値の合計は " + sum + " です。");
    }
}
```

【実行結果】

```
数値入力（-1で終了）：3
数値入力（-1で終了）：1
数値入力（-1で終了）：2
数値入力（-1で終了）：-1
入力された数値の合計は 6 です。
```

解説

上記のwhile文サンプルプログラムでは、ユーザからの数値の入力をするために、
Javaの標準クラスライブラリとして提供されている **Scanner クラス** というものを使用しています。

Scannerクラス を使用するために、
`import`宣言と`new`演算子によるScannerのインスタンスの生成というものを行なっています。
実際にユーザからのキーボードの入力を受け付けて数値として変数に格納する処理は、`inputNum = scanner.nextInt();`という文です。
この文を実行することで、画面の表示がユーザの入力待ちの状態になります。
その状態でユーザが数値を入力して **Enterキー** を押下すると、
入力された数値が`inputNum`変数に格納され、プログラムが次の行の処理に進みます。

いきなり初めて見るキーワードがあちこちに出てきてびっくりしたかもしれませんが、 細かいところはとりあえず置いておいて、
今はこのプログラムが「どのように動くか」を理解するようにしてください。
特にここでは、`while`文による繰り返しの動きと、`inputNum`や`sum`変数の値の変化を読み取ることができればOKです。

for文との違い

同じループ文ですが、まったく同じかと言われるとそうではありません。
以下に使用ケースを記述します。

・for文
繰り返す回数が明確な場合に使用
→小学校の算数の授業で習った九九をマス目で作成するなど（あるいは、九九の結果値を作成）
（九九なので 9 と言う数字がキモ！

・while文
繰り返す回数が定まらない(前もって分からない)場合に使用
→ユーザーの入力値を受け取るまでは処理を繰り返す、といったような処理を行う場合
水道の蛇口をひねって水を出しっぱなしにし、バケツの8分目まで水が溜まったら、蛇口をひねって戻す。
水道は、ユーザー（蛇口をひねる人）が、水を止めるまでは出続ける仕様になってますよね？
`while`文 でも似たようなイメージを持つと、 **繰り返す回数が定まらない場合** という具体例が見えやすくなります！

Step3-2: 【繰返しの制御③】　do-while文

do-while文は、while文と同様、括弧内の継続条件によって同じ処理を繰り返したいときに使用します。
括弧内の継続条件を満たしている間、処理を繰り返します。

for文もwhile文もdo-while文も繰り返し処理を行うというのは同じですが、
forやwhileは前判定であり、do-whileは後判定であるという違いがあります。

前判定のループでは、もし最初の継続条件判定で判定結果が偽だった場合は、一度もループ内の処理を行わずループを終了することになります。
それに対して、
後判定のループでは、まず最初にループ内の処理を１回実行してから、継続条件の判定を行います。

つまり、forやwhileは **0回以上の繰り返し処理** 、do-whileは **1回以上の繰り返し処理** で使用します。
do-while文を使用する場面はそんなに多くありません。
最低一回は処理を行ってから繰り返しの判定を行いたい という場面に遭遇したらdo-while文の使用を検討してみてください。

```
do {

   /* ･･･ 繰り返し行う処理 ･･･ */

} while (継続条件);
```

【do-while文を使ったプログラム例 （DoWhileSample.java）】

```
import java.util.Scanner;

public class DoWhileSample {
    public static void main(String[] args) {
        int inputNum = 0;       // ユーザ入力値
        int sum = 0;        // 合計値

        // ユーザ入力読み込みオブジェクトを生成
        Scanner scanner = new Scanner(System.in);

        // 初期値を入力
        System.out.print("初期値入力：");
        inputNum = scanner.nextInt();
        System.out.println("初期値は " + inputNum + " です。\n");

        // ユーザから「-1」が入力されるまで、入力された数値をsum変数に加算
        do {
            // 入力値をsumに加算
            sum += inputNum;

            // 加算値を入力
            System.out.print("加算値入力（-1で終了）：");
            inputNum = scanner.nextInt();

        } while (inputNum != -1);

        // 合計値を表示
        System.out.println("入力された数値の合計は " + sum + " です。");
    }
}
```

【実行結果】

```
初期値入力：50
初期値は 50 です。

加算値入力（-1で終了）：2
加算値入力（-1で終了）：8
加算値入力（-1で終了）：6
加算値入力（-1で終了）：-1
入力された数値の合計は 66 です。
```

break文 と continue文

繰り返し処理のfor文とwhile文では、break文とcontinue文によって、ループの流れを変えることができます。
break文を記述すると、その時点でループ処理から抜けます。例えば下記のプログラムの場合、

```
for (int i = 0; i < 5; i++) {

    break;

}
```

ループに入るといきなりbreak文が出てくるので、そこでループ処理は終了してしまいます。
しかし、これではループの記述をしている意味がありませんね。
通常は、ある条件に当てはまったときだけbreak文を実行してループから抜けるというように記述します。

```
for (int i = 0; i < 5; i++) {

    if (何かの条件) {
        break;      // 条件に一致したらループから抜ける
    }
}
```

【break文を使ったプログラム例 （BreakSample.java）】

```
import java.util.Scanner;

public class BreakSample {
    public static void main(String[] args) {
        int inputNum = 0;       // ユーザ入力値
        int sum = 0;        // 合計値

        // ユーザ入力読み込みオブジェクトを生成
        Scanner scanner = new Scanner(System.in);

        // 初回の数値入力
        System.out.print("数値入力（-1で終了）：");
        inputNum = scanner.nextInt();

        // ユーザから「-1」が入力されるまで、入力された数値をsum変数に加算
        while (inputNum != -1) {
            // 入力値をsumに加算
            sum += inputNum;

            // 合計が10を超えた場合も、ループを終了する
            if (sum > 10) {
                System.out.println("合計が10を超えたので数値入力を終了します。");
                break;
            }

            // ２回目以降の数値入力
            System.out.print("数値入力（-1で終了）：");
            inputNum = scanner.nextInt();
        }
        scanner.close();
        // 合計値を表示
        System.out.println("入力された数値の合計は " + sum + " です。");
    }
}
```

【実行結果】

```
数値入力（-1で終了）：5
数値入力（-1で終了）：3
数値入力（-1で終了）：1
数値入力（-1で終了）：4
合計が10を超えたので数値入力を終了します。
入力された数値の合計は 13 です。
```

【continue文は、その時点で後の処理をスキップします。】

```
while (条件式) {

    [ある処理A]

    continue;

    [ある処理B]

}
```

上記のプログラムの場合、[ある処理A]を行ってcontinue文に来ると、
[ある処理B]をスキップしてwhile文の条件比較に戻ります。
そして条件が合えばループ内の[ある処理A]を行って・・・（以降繰り返し）・・・　となります。
しかし、これだと[ある処理B]は絶対に実行されません。
これも意味のないロジックです。 通常は、continue文もbreak文と同じく、
ある条件に当てはまったときだけcontinue文を実行するというように記述します。

```
while (条件式) {

    [ある処理A]

    if (何かの条件) {
        continue;       /* 条件に一致したら[ある処理B]はスキップする */
    }

    [ある処理B]

}
```

【continue文を使ったプログラム例 （ContinueSample.java）】

```
public class ContinueSample {
    public static void main(String[] args) {
        int[] nums = {7, 3, -8, 1, 5};
        int sum = 0;

        // numsの要素のうち、正の値のみをsum変数に加算
        for (int i = 0; i < 5; i++) {
            // 負の値の場合は、以降の処理をスキップ
            if (nums[i] < 0) {
                System.out.println("負数（" + nums[i] + "）は無視！");
                continue;
            }

            // sum変数に加算
            sum += nums[i];
            System.out.println(nums[i] + "を加算しました。");
        }

        // 合計値を表示
        System.out.println("numsの合計(正数のみ)は " + sum + " です。");
    }
}
```

【実行結果】

```
7を加算しました。
3を加算しました。
負数（-8）は無視！
1を加算しました。
5を加算しました。
numsの合計(正数のみ)は 16 です。
```

Step4: 無限ループ（永久ループ）

ループ文において、継続条件が常に真である場合、そのループは **無限ループ（永久ループ）** となります。
無限ループ は、プログラマのミスによって引き起こされる場合（=バグ）もありますが、
意図的に **無限ループ** するようにプログラムを記述する場合もあります。
その場合は、ある条件が成立したときに `break`や `return` を実行してループやメソッド処理の **ブロック** を抜けるようにします。
例えば、繰り返す回数が決まっておらず、
繰り返し処理を実行するうちに ある条件が成立したときにだけループを終了したい場合などに **無限ループ** を使用します。

`while文`の場合は、`while (true)`として、継続条件に `true` を指定することで **無限ループ** させることができます。
これは、`while文`の条件式に `true`（真） が直接指定されているため、
継続条件が `false`（偽） になることがないので **無限ループ** となります。
`for文`の場合は、`for (;;)`と記述すると **無限ループ** させることができます。
継続条件が省略されているため、判定が行われないので、 **無限ループ** となります。

ちなみに、`do-while文`においても、継続条件の条件式に`true`を指定することで **無限ループ** にすることができますが、これはあまりお勧めしません。
`do-while文`は、あくまで「後判定」を行う場合にのみ記述するほうが良いです。
無限ループ は条件式が常に真のため、条件式の判定を後判定にする意味がありませんので、
`do-while文`でなく一般的なループである`while文`で記述するほうが好ましいです。

```
// whileの無限ループ
while (true) {

    [ある処理A]

    if (ループ脱出条件) {
        break;
    }

    [ある処理B]
}

// forの無限ループ
for (;;) {

    [ある処理A]

    if (ループ脱出条件) {
        break;
    }

    [ある処理B]
}
```

【無限ループのプログラム例 （EndlessLoopSample.java）】

```
import java.util.Scanner;

public class EndlessLoopSample {
    public static void main(String[] args) {
        int inputNum = 0;       // ユーザ入力値
        int sum = 0;        // 合計値

        // ユーザ入力読み込みオブジェクトを生成
        Scanner scanner = new Scanner(System.in);

        // ユーザから「-1」が入力されるまで、入力された数値をsum変数に加算
        while (true) {
            // 数値を入力
            System.out.print("数値入力（-1で終了）：");
            inputNum = scanner.nextInt();

            // -1 が入力されたらループを脱出
            if (inputNum == -1) {
                break;
            }

            // 入力値をsumに加算
            sum += inputNum;
        }

        // 合計値を表示
        System.out.println("入力された数値の合計は " + sum + " です。");
    }
}
```

【実行結果】

```
数値入力（-1で終了）：5
数値入力（-1で終了）：2
数値入力（-1で終了）：8
数値入力（-1で終了）：-1
入力された数値の合計は 15 です。
```

多重ループ

ループ文の中に更にループ文を記述することができます。
そのようなループを２重ループや３重ループ、あるいは多重ループと呼びます。

以下の九九表を表示する例では、外側でループカウンタ i によるループを行い、 その内側でループカウンタ j によるループを行っており、２重ループになっています。
実行結果を見ると分かる通り、外側の i が 1 つ増加する度に、毎回内側の j が 1 から 9 まで値を変えて九九の計算をしています。
また、内側の j が 9 まで回った後、外側の i を 1 つ増やす直前に `System.out.println();`で改行を出力しています。
これにより、一つの段を表示し終わったタイミングで改行が行われることになります。

【多重ループのプログラム例 （九九表を表示）】

```
public class Main {

    public static void main(String[] args) {

        // 九九表を表示
        for (int i = 1; i <= 9; i++) {
            // 1つの段を表示
            for (int j = 1; j <= 9; j++) {
                System.out.print(i + "x" + j + "=" + (i * j) + "  ");
            }

            // 1つの段を表示し終わったところで、改行する
            System.out.println();
        }

    }
}
```

【実行結果】

```
1x1=1 1x2=2 1x3=3 1x4=4 1x5=5 1x6=6 1x7=7 1x8=8 1x9=9
2x1=2 2x2=4 2x3=6 2x4=8 2x5=10 2x6=12 2x7=14 2x8=16 2x9=18
3x1=3 3x2=6 3x3=9 3x4=12 3x5=15 3x6=18 3x7=21 3x8=24 3x9=27
4x1=4 4x2=8 4x3=12 4x4=16 4x5=20 4x6=24 4x7=28 4x8=32 4x9=36
5x1=5 5x2=10 5x3=15 5x4=20 5x5=25 5x6=30 5x7=35 5x8=40 5x9=45
6x1=6 6x2=12 6x3=18 6x4=24 6x5=30 6x6=36 6x7=42 6x8=48 6x9=54
7x1=7 7x2=14 7x3=21 7x4=28 7x5=35 7x6=42 7x7=49 7x8=56 7x9=63
8x1=8 8x2=16 8x3=24 8x4=32 8x5=40 8x6=48 8x7=56 8x8=64 8x9=72
9x1=9 9x2=18 9x3=27 9x4=36 9x5=45 9x6=54 9x7=63 9x8=72 9x9=81
```

提出課題

提出ファイル

・Task2_5.java

2-5/Task2_5.java に課題がありますので、指示に従って記述して下さい。

評定概要

学生から秘匿	No
参加者	80
提出	54
要評定	0