第9章　繰り返し

目次

for文とは

for文を利用して、繰り返しをプログラムで表現できる。

```
for(初期化の式; 条件; 変化式){
    繰り返し処理;
}
```

for文の()には、①初期化、②条件式、③変化式を記述

① 初期化の式

ブロック内の繰り返し処理が始まる際に1回だけ実行される処理。
一般的には、ループカウンタ(繰り返した回数を保存するための変数)を初期化する式を記述する。
ループカウンタの変数名は、「i、j、k」を順に使用する。(慣例)

```
for(初期化の式; 条件; 変化式){
    繰り返し処理;
}
```

for文の()には、①初期化、②条件式、③変化式を記述

② 条件

ブロック内の処理を実行する前に評価され、
この繰り返しを継続するかを判断する。
一般的には、ループカウンタの数値が、
ある数値を超過するかどうかを評価する条件を記述する。
評価が「true」の場合、ブロック内の処理を1回分実行する。

```
for(初期化の式; 条件; 変化式){
    繰り返し処理;
}
```

for文の()には、①初期化、②条件式、③変化式を記述

③ 変化式

ブロック内の処理が完了した直後に、実行される処理。
一般的には、「i++」のようにループカウンタの値を
変化させる処理を記述する。

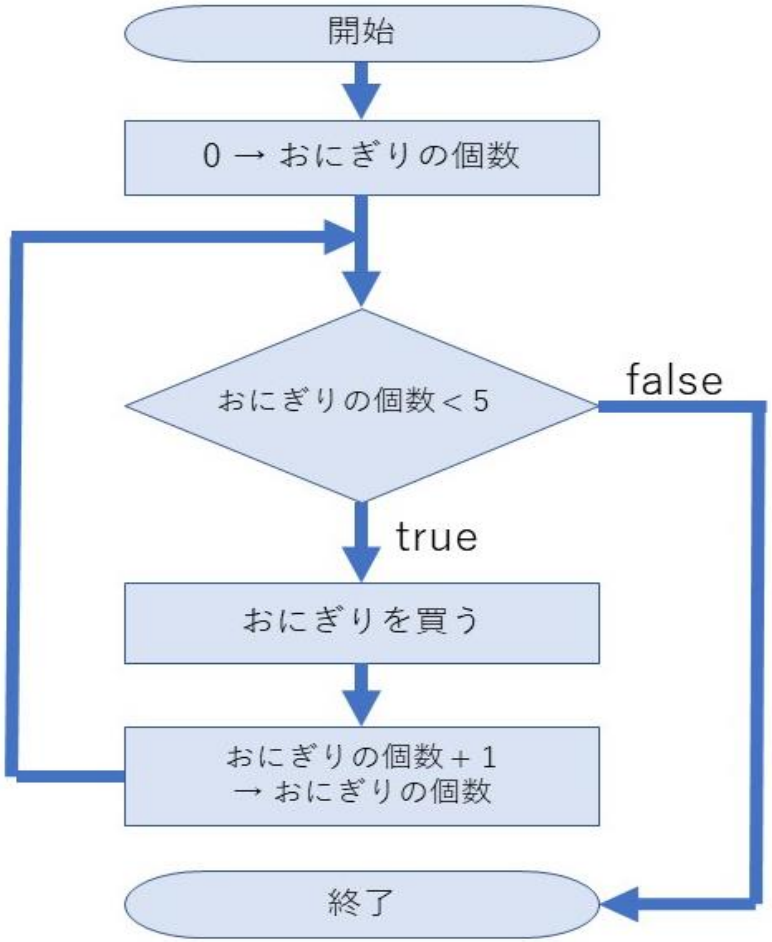
```
for(初期化の式; 条件; 変化式){
    繰り返し処理;
}
```

for文の()には、①初期化、②条件式、③変化式を記述

おにぎりの個数が5個になるまで
⇒おにぎりを買い続ける


```
for(おにぎりの個数は0個; おにぎりの個数が5個未満である; 個数を1個ずつ増やす){
    おにぎりを買う;
}
```

for文の()には、①初期化、②条件式、③変化式を記述



東京ITスクール
TOKYO IT SCHOOL

【Sample0901 for文を使う】を作成しましょう

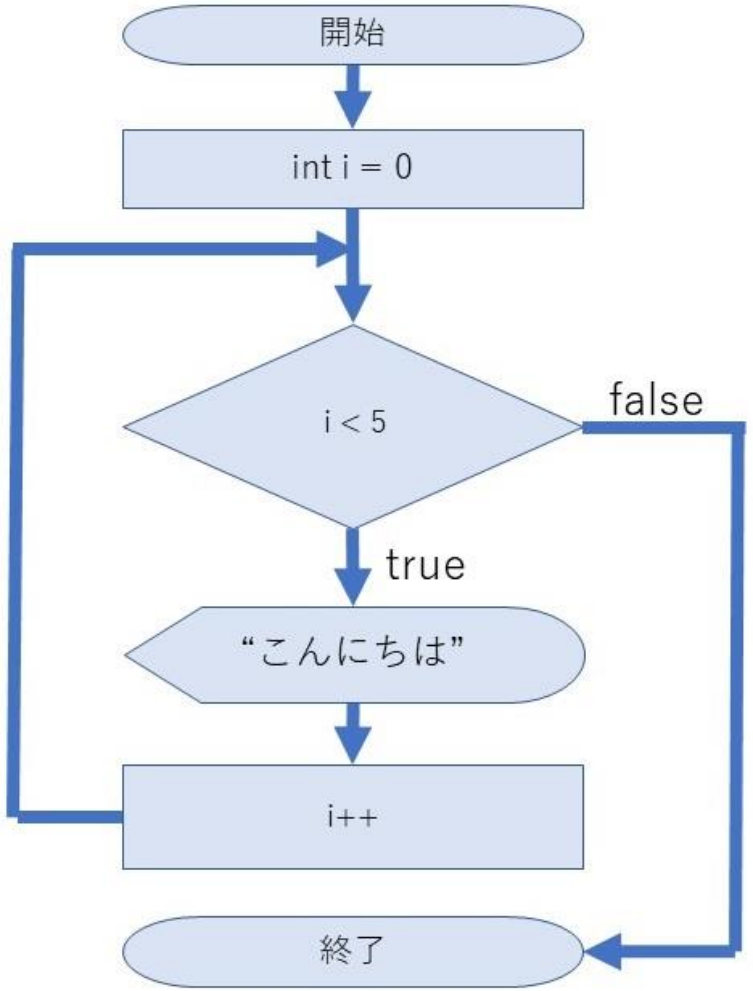
Sample0901のポイント

以下の流れで処理が実行される。

① ループカウンタiを0で初期化する。
② ループカウンタiが5未満の場合は、
ブロック内の処理を実行する。
③ ブロック内の処理が完了したら、
ループカウンタiに1を加算する。
④ ループカウンタiが5になるまで②、③を繰り返す。

for文を使うと、繰り返し処理が記述できます。

Sample0901のポイント



ループカウンタの注意点

① ループカウンタの名前は自由である。
識別子のルールに則した名前なら自由。
ただし、慣習として「i」を使用することが多い。

② ブロック内でループカウンタを使える。
変数と同じように、ループカウンタもfor文のブロック内で
使うことができる。ただし、ブロック外では使えない。

【Sample0902 繰り返し回数を出力】を作成しましょう

ループカウンタを使用し、
繰り返した回数を出力させることができる。

Sample0902のポイント

繰り返し処理でループカウンタiの値を出力している。
ブロック外でもループカウンタの値を使用したい場合、
for文が始まる前にループカウンタiを宣言する。

```
int i;
for(i = 1; i < 5; i++){
   System.out.println(i + “回繰り返しました。”);
}
System.out.println(i  + “回繰り返しました。”);
```

【Sample0903 繰り返し文を配列に用いる】を作成しましょう

ループカウンタを使用して、
配列を簡潔な記述で操作することもできる。

Sample0903のポイント

ループカウンタiを添字として使用することで、
for文で各要素を呼び出すことができる。

- 添字は「0」から始まるので、iの初期値も0にする。
- 何人目の身長であるかを出力する際は
  「(i + 1)人目」と記述する。

```
for (int i = 0; i < height.length; i++) {
   System.out.println((i + 1) + "人目の身長は" +
   height[i] + "です。");
}
```

Sample0903のポイント

条件ではループカウンタiと要素数を比較しており、
iの値が要素数未満であることが繰り返す条件となる。
「配列変数名.length」を使用することで、
配列の長さが変わったとしても、記述を変えることなく、
繰り返し回数を調整することができる。

```
for (int i = 0; i < height.length; i++) {
```

ループカウンタを使用して、
繰り返しの回数確認や配列操作が行えます。

【Sample0904 入力した数だけ記号を出力する】を作成しましょう

繰り返す回数を入力値で決められるプログラム

【Sample0905 入力した数までの合計を求める】を作成しましょう

1から入力した数までの総和を求めるプログラム

Sample0905のポイント

変数sumにループカウンタiの値を加算代入している。
変数iの値は「i++」により1つずつ加算される。
結果、1から入力値までの総和を求めることができる。

	変数sum	＋	変数i	＝	新しいsumの値 (sum += i)
1回目の繰り返し	0	＋	1	＝	1
2回目の繰り返し	1	＋	2	＝	3
3回目の繰り返し	3	＋	3	＝	6
4回目の繰り返し	6	＋	4	＝	１０
5回目の繰り返し	１０	＋	5	＝	１５
6回目の繰り返し	１５	＋	6	＝	２１

2回目以降の変数sumには、
新しいsumの値(sum += i)の値が代入されていく

拡張for文

配列の要素を1回の繰り返しごとに
先頭から順番に取り出して処理するための構文。

```
for(型 変数名 : 配列変数名){
    繰り返し処理;
}
```

【Sample0906 配列を拡張for文で使う】を作成しましょう

Sample0906のポイント

拡張for文はループカウンタや添字の指定が不要。

for文よりも簡潔なコードとなる。

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



```
for (int value : test) {
   System.out.println(value);
}
```

for文のネスト

for文をネストすると多重の繰り返しが実行できる。

```
for (初期化の式; 条件; 変化式) {
        繰り返し処理;
      for (初期化の式; 条件; 変化式) {
            繰り返し処理;
      }
}
```

【Sample0907 for文のネスト】を作成しましょう

Sample0907のポイント

内側のfor文の繰り返し
1回につき、掛け算を行う。
内側のfor文が終了すると
外側のfor文の繰り返し
1回分が終了する。

以上の流れを
外側のfor文が
終わるまで繰り返す。

```
public static void main(String[] args){
    for(int i = 1; i < 10; i++){
        for(int j = 1; j < 10; j++){
          System.out.print(i * j);
          System.out.print(" ");
        }
        //改行を出力
        System.out.println(" ");
    }
}
```

【外側ループ】
iは1から9まで
繰り返す

【内側ループ】
jも1から9まで
繰り返す

外側のfor文が1回目の繰り返しのとき、iの値は1である。

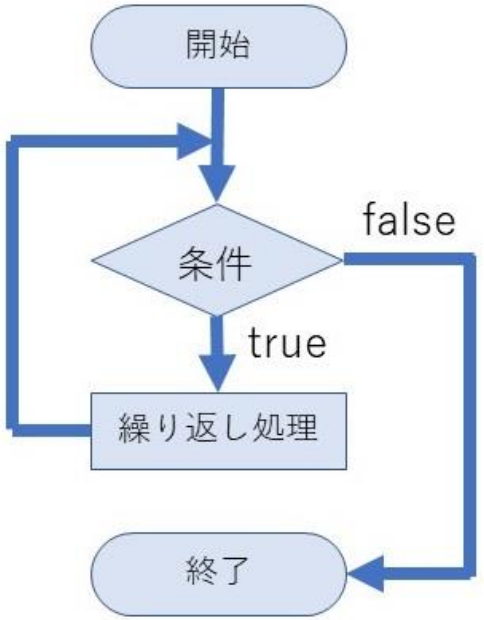
このとき、内側のfor文では「1×1、…、1×9」が計算される。

外側のfor文が2回目の繰り返しのとき、iの値は2である。

このとき、内側のfor文では「2×1、…、2×9」が計算される。

while文

```
while(条件){
    繰り返し処理;
}
```



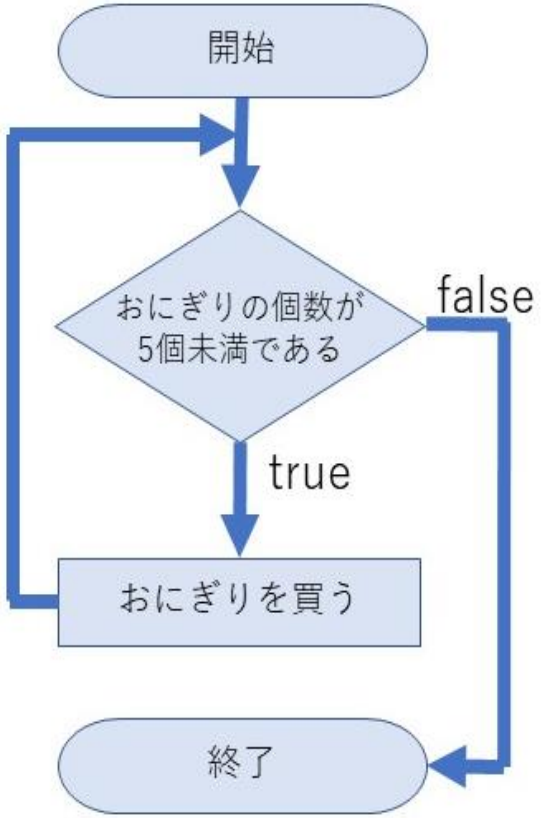
while文

おにぎりの個数が5個になるまで
⇒おにぎりを買い続ける


```
while(おにぎりの個数が5個未満である){
    おにぎりを買う;
}
```

while文

```
while(おにぎりの個数が5個未満である){
    おにぎりを買う;
}
```



【Sample0908 while文を使う】を作成しましょう

Sample0908のポイント

条件である「i < 5」が「false」になるまで繰り返している。
while文のブロック内では、最終的に繰り返しが終了するように、
「i++」で変数iの値を変化させている。

```
while (i < 5) {
    System.out.println((i + 1) + "回繰り返しました。");
    i++;
}
```

無限ループ

繰り返し文では、条件の評価が
常に「true」だと、無限に繰り返される。
無限の繰り返しのことを「無限ループ」という。

```
int i = 0;
while (i < 5) {
    System.out.println((i + 1) + "回繰り返しました");
}
```

上記は、【Sample0908】のwhile文から「i++」を抜いたもの。
何回繰り返してもiの値は常に1のままであり、
while文の条件は「true」となる。

無限ループ

無限ループはプログラムが動いているPC内のCPUやメモリに
大きな負荷を掛ける恐れがある。
また、プログラムが強制的に止まってしまうこともある。
無限ループにならないよう注意が必要。

while文を使うと繰り返し処理を記述できます。
無限ループの発生に注意する必要があります

do~while文

while文と同様、条件が「true」の間、
繰り返し処理が実行される。

```
do {
   繰り返し処理;
} while (条件式);
```

do~while文

最初の段階で条件が「false」の場合…

while文：1度も繰り返し処理は実行されない。
(繰り返し処理よりも先に、条件を評価するため)

do~while文：必ず最初の1回は繰り返し処理を実行する。
(繰り返し処理が終了後、条件の評価を行うため)

do～while文を使うと繰り返し処理を記述できます。
do～while文は、最低1回は繰り返し処理を実行します。

【Sample0909 do~while文を使う】を作成しましょう

処理の流れの変更:break文とは

break文は、繰り返しやswitch文などの
処理の流れを強制的に中断するという機能を持つ。

```
break;
```

**break文を使って、
繰り返しを強制的に中断することができます。**

【Sample0910 break文で処理を中断する】を作成しましょう

Sample0910のポイント

入力値がループカウンタiの値と一致した場合、
break文が実行され、繰り返しが強制的に終了する。
たとえば、「4」と入力した場合、
5回目以降の繰り返しは行われていないことが分かる

```
何回目の繰り返しで中止しますか？(1～10)
4 ↲
1回繰り返しました。
2回繰り返しました。
3回繰り返しました。
4回繰り返しました。
繰り返しを中断しました。
```

処理の流れの変更:continue文とは

continue文は、実行中の繰り返し処理の途中で
残りの処理を行わずに、次の繰り返しに移動するという機能を持つ。

```
continue;
```

continue文を使って、繰り返し処理を途中で中断し、
次の繰り返しに移ることができます。

【Sample0911 continue文で処理をスキップする】を作成しましょう

Sample0911のポイント

入力値がループカウンタiと一致した場合、
continue文が実行され、次の繰り返しに移る。
たとえば、「4」と入力した場合、4回目の出力処理は実行されず、
続けて5回目以降の繰り返し処理が実行される。

```
何回目の繰り返しを中断しますか？(1～10)
4 ↲
1回繰り返しました。
2回繰り返しました。
3回繰り返しました。
5回繰り返しました。
6回繰り返しました。・・・
```

制御文のまとめ：制御分の使い分け：if文とswitch文

【if文とswitch文の違い】
どちらの書き方でも、同じ条件分岐を記述することは可能。
違いは、「①ソースコードの読みやすさ」と「②実行速度」の2つ。

【if文と switch文の使い分け】
if文：式の評価がtrueなのかfalseなのかによって処理を分岐。
switch文：式の評価とcaseで指定した値が
一致しているかどうかによって処理を分岐。

if文とswitch文　二分岐

```
if (条件) {
   条件の評価がtrueのときの処理文;
} else {
   条件の評価がfalseのときの処理文;
}
```

```
switch (式) {
case a:
     式の評価がaのときの処理;
     break;
default:
     式の評価がa以外のときの処理;
     break;
}
```

if文のほうが見やすい。
処理速度はif文、switch文どちらも変わらない。

if文とswitch文　多分岐

```
if (条件1) {
   条件1の評価がtrueのときの処理;
} else if (条件2) {
   条件2の評価がtrueのときの処理;
} else if (条件3) {
   条件3の評価がtrueのときの処理;
} else {
   上記以外のときの処理;
}
```

switch文が見やすい。
処理速度もswitch文のほうが速い。

```
switch (式) {
case a:
      式の評価がaのときの処理;
     break;
case b:
     式の評価がbのときの処理;
     break;
case c:
     式の評価がcのときの処理;
     break;
default:
     上記以外のときの処理;
     break;
}
```

if文とswitch文の使い分け

二分岐の場合：
→if文のほうが見やすい。処理速度は変わらない。

多分岐の場合：
→switch文のほうが見やすい。処理速度もswitch文が速い。

if文：二分岐
switch文：多分岐

制御文のまとめ：制御分の使い分け：for文とwhile文

【for文とwhile文の違い】
書式の違いによる記述の読みやすさ。

制御文のまとめ:制御分の使い分け:for文とwhile文

【for文とwhile文の使い分け】

```
for(初期化の式; 条件; 変化式){
    繰り返し処理;
}
```

```
while(条件){
    繰り返し処理;
}
```

for文:「繰り返し回数が決まっている」処理

while文:「繰り返し回数が決まっていない」処理

【Sample0912 配列のソート】を作成しましょう

制御文の組み合わせについて、
「要素を並び替える」という処理を例として学ぶ。

Sample0912のポイント

配列をソートする場合は、「Arrays.sort(配列変数名)」と記述する。

```
Arrays.sort(numbers);
```

クラスブロックの手前には「java.util.Arrays」と記述する。

```
import java.util.Arrays;
```

章のまとめ

- for文、while文、 do~while文を使うと、
  繰り返し処理を行うことができます。
- for文をネストできます。
- break文を使うと、繰り返し文やswitch文の
  ブロックを抜け出すことができます。
- continue文を使うと、繰り返し処理を中断し、
  次の繰り返し処理を行うことができます。
- if文とswitch文は、「いくつ条件があるのか」で使い分けます。
- for文とwhile文は、
  「繰り返し回数が決まっているかどうか」で使い分けます。