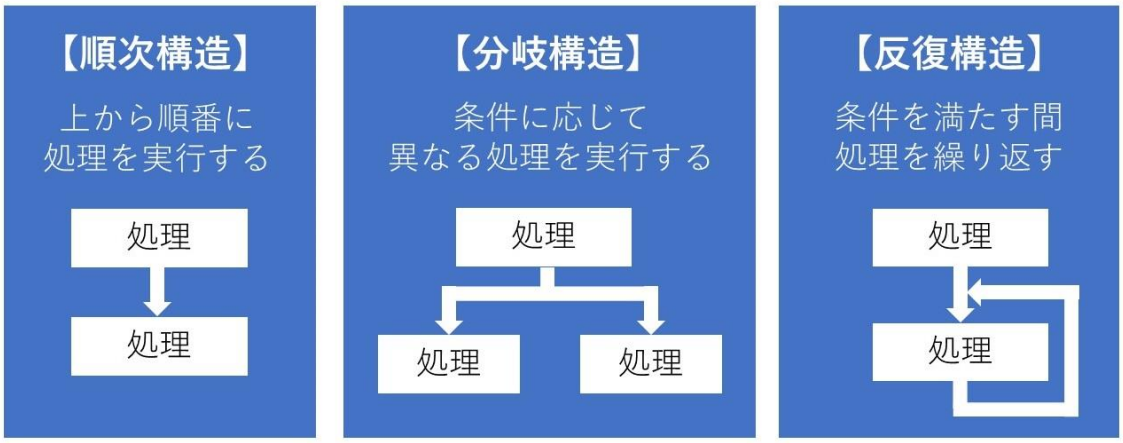
第 8 章 条件分岐

制御構造と制御文

プログラムの処理の流れは「制御構造」といい、制御構造には「順次構造」「分岐構造」「反復構造」の 3 種類があります。

【図 8-1 制御構造の種類】



これまでの章では、順次構造の単純な処理を中心に紹介しました。しかし、開発現場では、分岐構造や反復構造を使用した複雑な処理を実装しなければならないこともあります。分岐構造や反復構造を実現するための構文を「制御文」といいます。また、本書では分岐構造の処理を「条件分岐」、反復構造の処理を「繰り返し」と呼称します。

本章では分岐構造の構文を扱い、反復構造の構文については次章で説明を行います。

条件と関係演算子

1. 条件とは

はじめに、分岐構造の構文で必要となる条件と関係演算子について紹介します。

たとえば、次のような場面をイメージしてください。

> もし、おにぎりセールを実施していたらおにぎりを買う。
> おにぎりセールを実施していなかったら家に帰る。

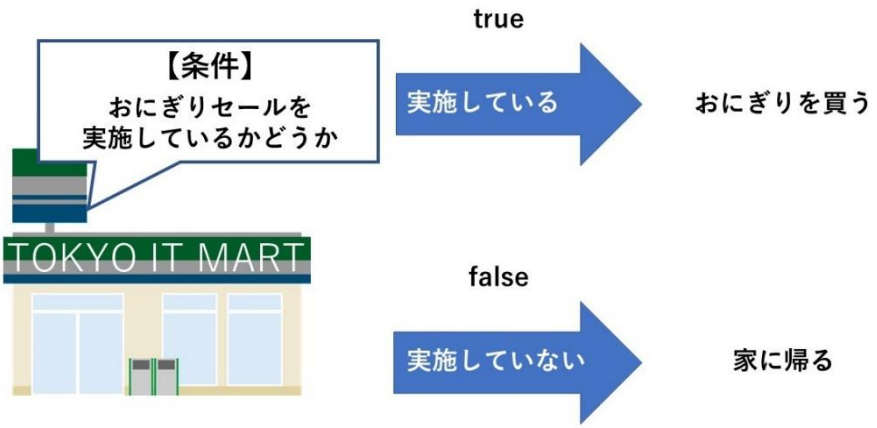
このような日常生活で起こり得る「条件に応じて動作を決める」ということを、プログラム上でも実行することができます。おにぎりを買う場合をプログラミングの考え方に置き換えてみると、おにぎりの購入処理を行うか否かを検討するための条件は下記のような内容になります。

> おにぎりセールを実施しているかどうか

Java では、条件は式として扱われ評価されます。条件は、その条件を満たす場合は「true（真）」を、満たさない場合は「false（偽）」という値 2 つを評価値として導き出します。したがって、上記の条件の評価値に応じて下記の 2 パターンの処理を実行できます。

> おにぎりセールを実施している（true） ⇒ おにぎりを買う
> おにぎりセールを実施していない（false） ⇒ 家に帰る

【図 8-2 条件の評価値による処理の変化】



2. 関係演算子とは

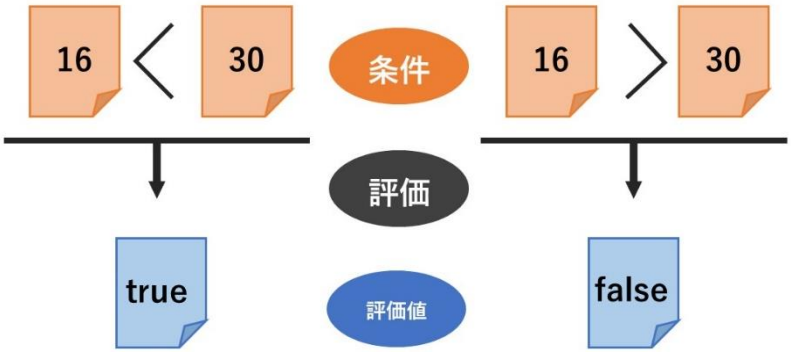
続いて条件の記述方法について紹介します。条件の基本的な記述形式は、値を比較する形のものです。比較対象となる値（オペランド）を下表で紹介する関係演算子を使用して比較します。

【表 8-1 関係演算子】

演算子	意味	記述例
==	左辺と右辺は等しい	a == b
!=	左辺と右辺は等しくない	a != b
>	左辺は右辺より大きい	a > b
>=	左辺は右辺より大きい、または等しい	a >= b
<	左辺は右辺より小さい	a < b
<=	左辺は右辺より小さい、または等しい	a <= b

関係演算子を使用した条件は評価値（条件を満たすかどうか）として「true」または「false」を導き出します。また、オペランドには数値などを使用できます。

【図 8-3 関係演算子】



ちなみに、「左辺と右辺が等しいか」を判定する演算子として、数学の世界では「=」が使用されますが、Java では「==」が使用されます。「=」は代入演算子として機能しますので、間違えないように注意しましょう。

それでは、関係演算子を使用した条件の書き方の練習として、次のサンプルコードを作成してください。

【Sample0801 関係演算子を使う】
■ 作成するファイル
/java_sample/src/lesson08/Sample0801.java

Sample0801.java

```
package lesson08;

public class Sample0801 {
    public static void main(String[] args) {
        int a = 20;
        int b = 40;

        System.out.println("a:" + a + " b:" + b);
        System.out.println("a == b:" + (a == b));
        System.out.println("a != b:" + (a != b));
        System.out.println("a < b:" + (a < b));
        System.out.println("b < a:" + (b < a));
        System.out.println("a >= 10:" + (a >= 10));
        System.out.println("a >= 20:" + (a >= 20));
    }
}
```

【実行結果】

```
a:20 b:40
a == b:false
a != b:true
a < b:true
b < a:false
a >= 10:true
a >= 20:true
```

条件の評価値を上から順番に確認してみましょう。変数 a には 20、b には 40 という異なる数値が代入されているため、「a == b」の評価値は「false」、「a != b」の評価値は「true」となります。また、a の値より b の値の方が大きいため、「a＜b」の評価値は「true」、「b＜a」の評価値は「false」となります。そして最後に、a の値は 10 より大きく 20 と等しいため、「a >= 10」と「a >= 20」のどちらも「true」となります。演算子「>=」と「<=」は、両辺の値が等しい場合も true と判定されます。

重要
条件は、関係演算子を使って記述します。

if 文

1. if 文とは

ここでは条件分岐の実装方法を紹介します。Java では、条件に応じて処理を実行させる文を条件判断文（conditional statement）といいます。条件判断文の代表的なものに if 文があります。if 文とは、「もし○○だったら、▲▲する」というように、条件を満たした場合に特定の処理を実行する文です。

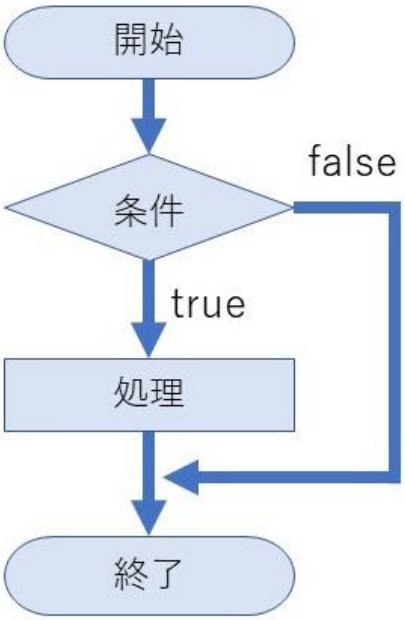
＜構文 if 文>

```
if（条件）{
    処理;
}
```

if 文は、「()」内の条件を満たしている場合（true の場合）のみ、ブロック内の処理を実行します。なお、ブロック内には複数行の処理を記述することができ、ブロック内の処理は上から順番に実行されます。条件を満たさない場合は、ブロック内の処理は実行されず、ブロックの次に記述された処理に移ります。

if 文の処理をフローチャートで表現すると、次のようになります。

【図 8-4 フローチャート if 文】



たとえば、先ほどのおにぎり購入処理を if 文で表現すると次のようになります。

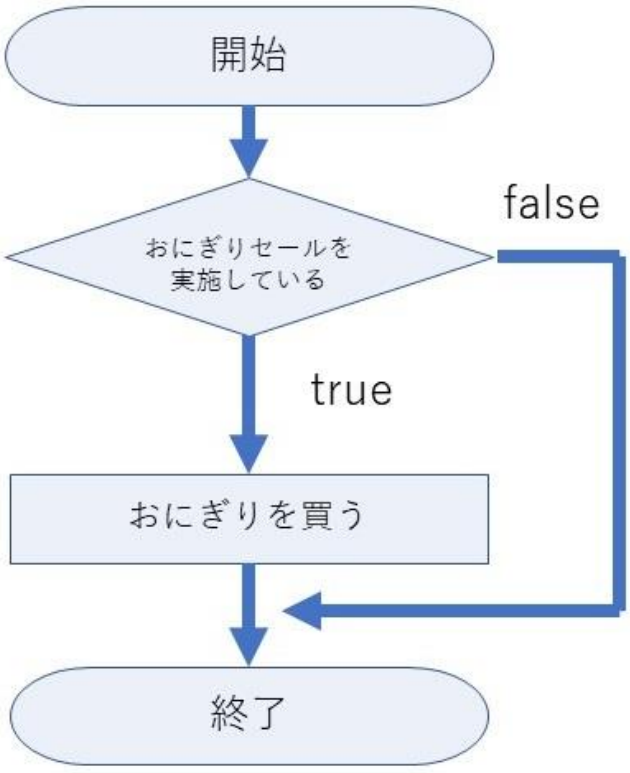
```
もし、おにぎりセールを実施していたら、おにぎりを買う。
```

if 文で表現した内容

```
if（おにぎりセールを実施している）{
    おにぎりを買う;
}
```

そして、これをフローチャートで表現すると下記のようになります。

【図 8-5 if 文の使用例のフローチャート】



それでは、if 文を実装する練習として、下記のサンプルコードを作成してください。

【Sample0802 if 文を使う】
■ 作成するファイル
/java_sample/src/lesson08/Sample0802.java

Sample0802.java

```
package lesson08;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample0802 {
    public static void main(String[] args) throws IOException {
        System.out.println("整数を入力してください。");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();
        int res = Integer.parseInt(str);

        if (res == 5) {
            // resの値が5だった場合、以下の処理を実行
            System.out.println("5が入力されました。");
            System.out.println("式の評価値は「true」です。");
        }

        System.out.println("システムを終了します。");
    }
}
```

【実行結果（5 を入力した場合)】

```
整数を入力してください。
5 ↲
5が入力されました。
式の評価値は「true」です。
システムを終了します。
```

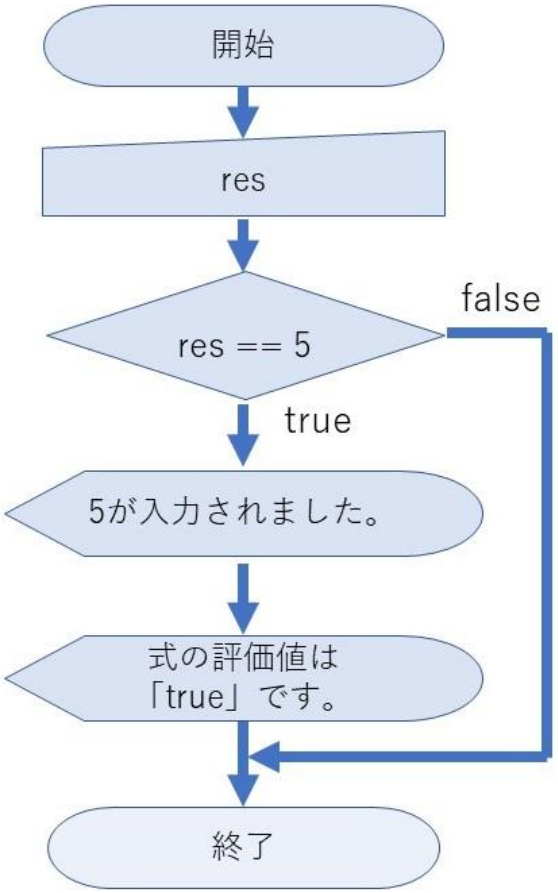
【実行結果（5 以外の整数を入力した場合)】

```
整数を入力してください。
9 ↲
システムを終了します。
```

このサンプルコードでは、標準入力した整数が 5 だった場合のみ、特定の処理を実行します。5 を入力した場合、if 文の条件の評価値は「true」となり、if 文のブロック内の処理が実行されます。 また、5 以外の整数を入力した場合は、if 文の条件の評価値は「false」となります。その場合、if 文のブロック内の処理は実行されず、ブロックの下の処理が実行されます。

以上の処理をフローチャートで表すと下記のようになります。

【図 8-6 Sample0802 の条件分岐のフローチャート】



重要
if 文を使って、分岐（条件を満たした場合にだけ特定の処理を実行）させることができます。

2. if 文の注意点

if 文を記述する際は変数のスコープに注意してください。

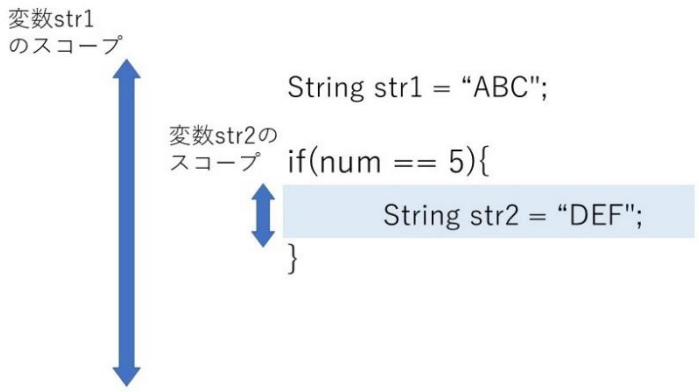
ブロック内で宣言した変数は、ブロック内の処理が終了すると利用できなくなります。変数が利用可能な範囲（有効範囲）のことをスコープ（scope）といいます。スコープを意識せずにソースコードを作成すると、想定外の結果となる場合があります。例として、下記のサンプルコードを見てください。

```
public static void main(String[] args) throws IOException {
    int num = 5;
    String str1 = "ABC";

    if (num == 5) {
        String str2 = "DEF";
    }
    System.out.println(str1);
    System.out.println(str2);
}
```

変数 str2 は if 文のブロック内で宣言される変数です。つまり、str2 のスコープは if 文のブロック内のみということです。そのため、str2 を if 文のブロック外（スコープ外）で使用しようとすると、コンパイルエラーになってしまいます。もし、str2 も if 文のブロック外で使用したいという場合は、変数 str1 と同じように、ブロックの外側で変数の宣言を行いましょう。

ちなみに、スコープ外で変数を使用しようとした場合、「str2 を変数に解決できません」といったメッセージのコンパイルエラーが発生します。このようなコンパイルエラーが発生した場合は、変数のスコープに問題がないか確認しましょう。

【図 8-7 スコープの範囲】



重要
変数の宣言を行う際はスコープに注意しましょう。

コラム ブロックの省略

if 文のブロックの記載は、ブロック内の処理が 1 行のみの場合に限って省略することができます。そのため、以下に記載しているサンプルの実行結果はどちらも同じになります。

【ブロックなし】

```
if (res == 5)
    System.out.println("5 が入力されました。");
```

【ブロックあり】

```
if (res == 5) {
    System.out.println("5 が入力されました。");
}
```

しかし、「{ }」を省略するとソースコードの読み間違いなどが発生する恐れがあるため、開発現場では推奨されていません。このようなミスを防ぐために、if 文のブロックは必ず記述するようにしましょう。

「{ }」の省略により読み間違いが起こりうる例として、下記のサンプルコードを見てください。

```
if (num == 5)
    System.out.println("5 が入力されました。");
    System.out.println("式の評価値は「true」です。");
    System.out.println("システムを終了します。");
```

上記のサンプルコードは Sample0802 と同様に、変数 num の値が 5 である場合に「5 が入力されました。」と「式の評価値は『true』です。」の 2 行を出力するという意図で作成したとします。
このコードは、num の値が 5 であった場合は期待通りの結果が出力されます。

【実行結果（num の値が 5 の場合）】

```
5 が入力されました。
式の評価値は「true」です。
システムを終了します。
```

しかし、num の値が 5 以外の整数の場合は、下記のように出力されます。

【実行結果（num の値が 5 以外の整数の場合）】

```
式の評価値は「true」です。
システムを終了します。
```

「式の評価値は『true』です。」という文言は、num の値が 5 であった場合のみ出力される想定でしたが 5 以外の場合も出力されてしまっています。これは、条件を満たす場合に複数の処理を実行したいにもかかわらず、それらの処理が「{}」のブロックに囲まれていないことが原因です。ブロックの記述がないと、条件を満たす場合に実行されるのは条件の次に記述されている 1 行分の処理のみです。それより下に記述された処理は、条件の評価値にかかわらず実行されてしまいます。

仮に 1 行のみの if 文を書いた場合は、ブロック内の処理を 1 段インデントして、読み間違いが起こりにくくするといった工夫をするようにしましょう。

コラム boolean 型の変数

if 文の条件には、関係演算子の式に限らず、boolean 型の値（true か false）が評価値として出てくるものなら何でも記述することができます。そのため、下記のように boolean 型の値のみを記述することも可能です。

```
boolean flag = false;
if (flag) {
```

変数 flag に true が代入されている場合は条件を満たすと判定され、false が代入されている場合は条件を満たさないと判定されます。
また、boolean 型の変数と関係演算子を組み合わせて、下記のようにも記述できます。

```
boolean flag = false;
if (flag != true){
```

しかし、boolean 型の変数は単体で「true」か「false」の判定ができるため、わざわざ関係演算子で比較するのは冗長です。そのため、開発現場ではこのような記述は推奨されていません。

if 文のネスト

if 文の中にさらに if 文を記述することもできます。このように、処理を入れ子の形にすることをネストといいます。

<構文 if 文のネスト>

```
if（条件 1）{
    if（条件 2）{
        処理;
    }
}
```

外側の if 文の条件（条件 1）が「true」の場合、外側のブロックの処理が実行されて内側の if 文の条件（条件 2）の判定が行われます。そして、条件 2 も「true」だった場合、内側のブロックの処理が実行されます。

それでは、if 文のネストを実装する練習として、次のサンプルコードを作成してください。

【Sample0803 if 文をネストする】
■ 作成するファイル
/java_sample/src/lesson08/Sample0803.java

Sample0803.java

```
package lesson08;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample0803 {
    public static void main(String[] args) throws IOException {
        System.out.println("整数を入力してください。");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();
        int res = Integer.parseInt(str);

        if (res > 4) { // ①変数resが4より大きい
            if (res <= 10) { // ②変数resが10以下
                System.out.println(res + "は4より大きく10以下の数字です。");
            }
        }
        System.out.println("システムを終了します。");
    }
}
```

6 を入力すると、①の条件は「true」となり、②の if 文へ移動します。そして、②の条件も「true」になり、②のブロックの処理を実行します。

【実行結果（入力値が 4 より大きく、かつ 10 以下の場合）】

```
整数を入力してください。
6 ↲
6 は 4 より大きく 10 以下の数字です。
システムを終了します。
```

2 を入力すると、①の条件は「false」になります。そのため、①のブロックの処理は実行されません。

【実行結果（入力値が 4 以下、または 10 より大きい場合）】

```
整数を入力してください。
2 ↲
システムを終了します。
```

if～else 文

1. if～else 文とは

if 文は条件を満たす場合の処理のみを指定できました。では、条件を満たさない場合に特別な処理をしたい場合はどのように実装すればよいでしょうか。その場合は、if~else 文という構文を利用します。

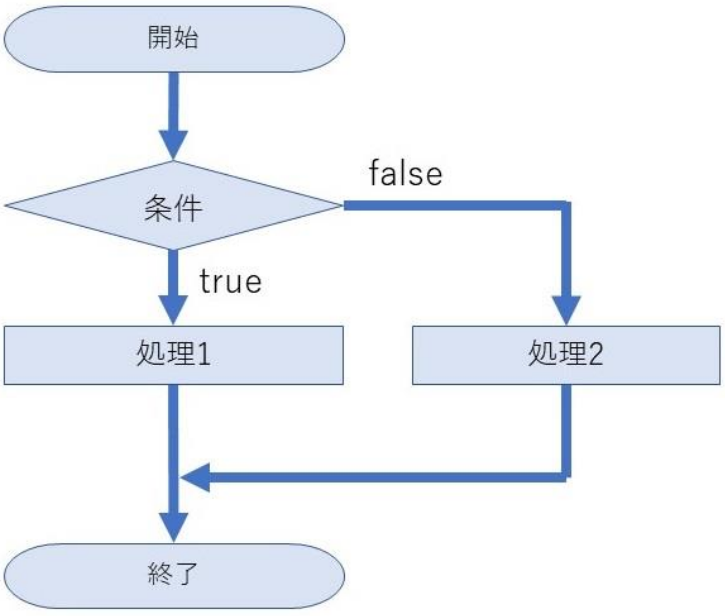
<構文 if~else 文>

```
if（条件）{
    処理 1;
} else {
    処理 2;
}
```

if~else 文では、条件を満たす場合は、if 文以下のブロックの処理を実行します。また、条件を満たさない場合は、else 以下のブロックの処理を実行します。ちなみに、各ブロック内には複数行の処理を記述することができます。

if~else 文の構文をフローチャートで表現すると、下記のようになります。

【図 8-8 if~else 文のフローチャート】



それでは次の例を元にして if~else 文の利用方法を確認しましょう。

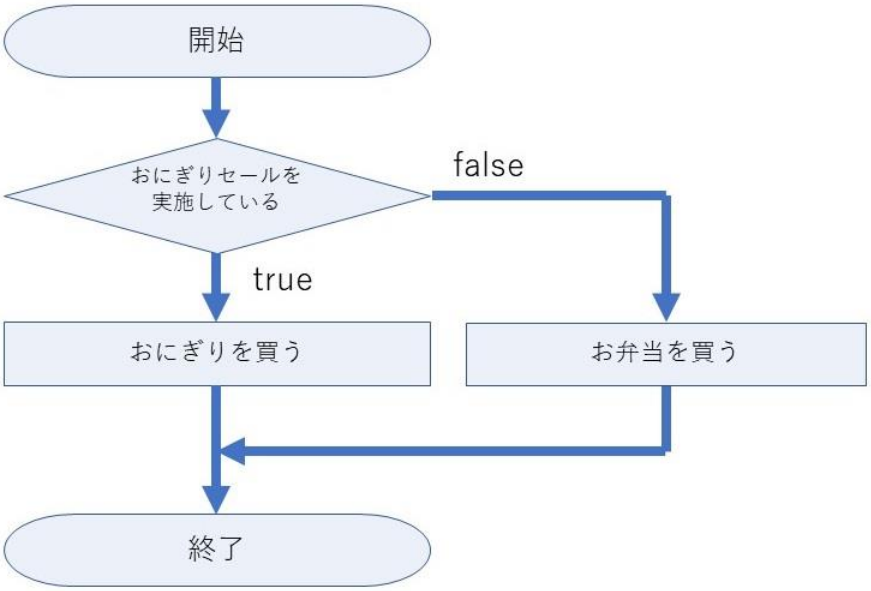
もし、おにぎりセールを実施していたら、おにぎりを買う。
おにぎりセールを実施していなかったら、お弁当を買う。

この内容では、おにぎりセールを実施しているか実施していないかでその後の行動が異なります。上記の例を if~else 文の構文に当てはめると下記のようになります。

```
if（おにぎりセールを実施している）{
    おにぎりを買う;
} else {
    お弁当を買う;
}
```

上記の if~else 文をフローチャートで表現すると、下記のようになります。

【図 8-9 if~else 文のフローチャート】



それでは、if~else 文を実装する練習として、次ページのサンプルコードを作成してください。

【Sample0804 if~else 文を使う】
■ 作成するファイル
/java_sample/src/lesson08/Sample0804.java

Sample0804.java

```
package lesson08;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample0804 {
    public static void main(String[] args) throws IOException {
        System.out.println("整数を入力してください。");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();
        int num = Integer.parseInt(str);

        if (num <= 8) {
            // ①numが8以下の場合の処理
            System.out.println(num + "は8以下の数字です。");
        } else {
            // ②numが8より大きい場合の処理
            System.out.println(num + "は8より大きい数字です。");
        }
    }
}
```

「3 を入力した場合」と「9 を入力した場合」の実行結果を確認してみましょう

【実行結果（3 を入力した場合)】

```
整数を入力してください。
3 ↲
3 は 8 以下の数字です。
```

【実行結果（9 を入力した場合)】

```
整数を入力してください。
9 ↲
9 は 8 より大きい数字です。
```

条件が「true」の場合は①の処理を、条件が「false」の場合は②の処理が実行されることが分かります。

if～else if～else 文

1. if～else if～else 文とは

複数の条件ごとに異なる処理を実行させたい場合は if~else if~else 文を使用します。この構文を利用すると複数の条件を指定することができます。

<構文 if~else if~else 文>

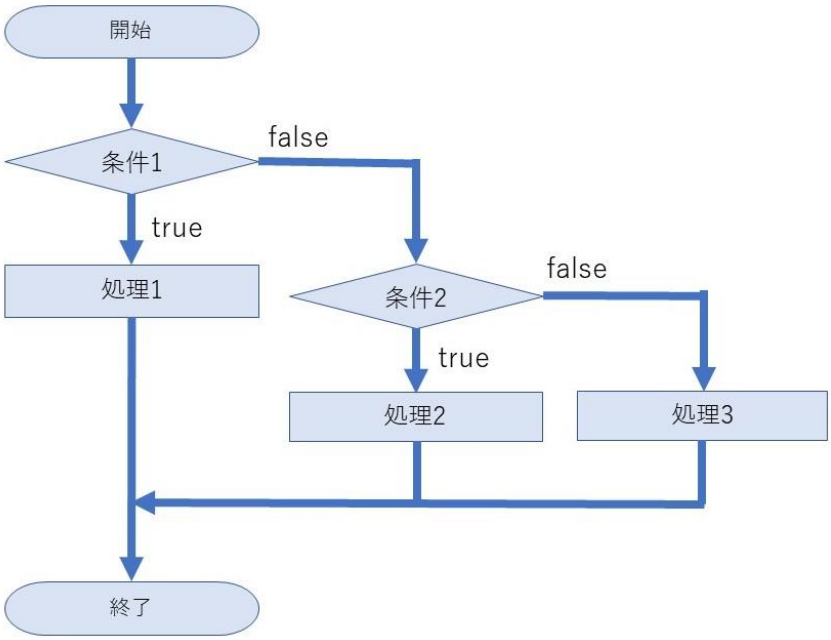
```
if(条件 1){
    処理 1;
} else if(条件 2){
    処理 2;
} else {
    処理 3;
}
```

if~else if~else 文では、はじめに if 文の条件（条件 1）の判定を行います。条件 1 が「true」の場合、処理 1 を実行して一連の条件分岐は終了します。条件 1 が「false」の場合は、条件 2 の判定を行います。条件 2 が「true」の場合は、処理 2 を実行して一連の条件分岐は終了します。もしいずれの条件も「false」の場合は else 文のブロックの処理（処理 3）が実行されます。

else if 文のブロックは続けて複数記述できるため、条件を 3 種類以上に増やすことも可能です。また、if 文や if~else 文と同様に各ブロック内には複数行の処理を記述することができます。

if~else if~else 文の構文をフローチャートで表現すると、下記のようになります。

【図 8-10 if~else if~else 文のフローチャート】



次は、下記内容を例に、if~else if~else 文の構文を見てみましょう。

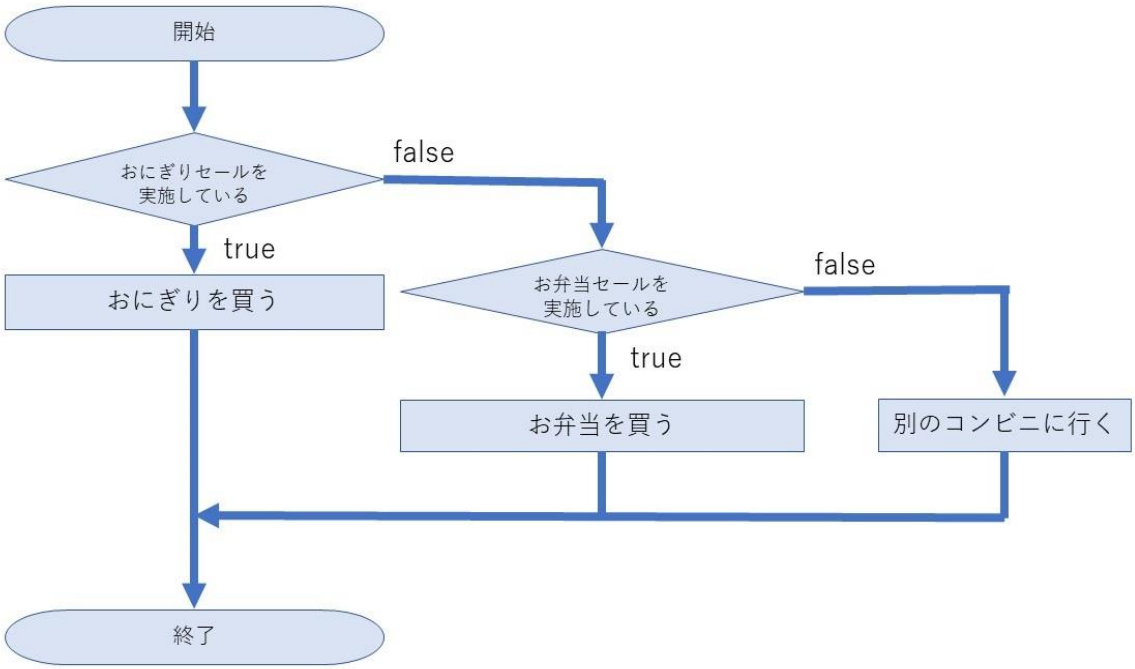
もし、おにぎりセールを実施していたら、おにぎりを買う。
おにぎりセールではなくお弁当セールを実施していたら、お弁当を買う。
どのセールも実施していなければ、別のコンビニに行く。

この内容では、おにぎりセールが実施しているか、お弁当セールを実施しているか、またはどのセールも実施していないかでその後の行動が異なっています。上記の場合を if~else if~else 文の構文にあてはめて表現すると次のようになります。

```
if（おにぎりセールを実施している）{
    おにぎりを買う;
} else if(お弁当セールを実施している){
    お弁当を買う;
} else {
    別のコンビニに行く;
}
```

また、上記の if~else if~else 文をフローチャートで表現すると下記のようになります。

【図 8-11 if~else if~else 文のフローチャート】



それでは、if~else if~else 文の練習として下記のサンプルコードを作成してください。

【Sample0805 if~else if~else 文を使う】
■ 作成するファイル
/java_sample/src/lesson08/Sample0805.java

Sample0805.java

```
package lesson08;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample0805 {
    public static void main(String[] args) throws IOException {
        System.out.println("整数を入力してください。");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();
        int num = Integer.parseInt(str);

        if (num == 4) {
            // ①numが4と等しい場合の処理
            System.out.println("4が入力されました。");
        } else if (num == 7) {
            // ②numが7と等しい場合の処理
            System.out.println("7が入力されました。");
        } else {
            // ③いずれの条件も満たさない場合の処理
            System.out.println("4と7以外の数字が入力されました。");
        }
    }
}
```

最初にif文の条件判定。
当てはまらない場合、下の判定へ
else if文の記述回数に制限はない。
if/elseは一回のみ。

複数の条件毎に異なる処理を実行したい場合に使用。

このサンプルコードを実行して 4 を入力した場合、1 つ目の条件が「true」となるため、①の処理が実行されます。

【実行結果（4 を入力した場合）】

```
整数を入力してください。
4 ↲
4が入力されました。
```

7 を入力した場合は 2 つ目の条件が「true」となるため、②の処理が実行されます。

【実行結果（7 を入力した場合）】

```
整数を入力してください。
7 ↲
7 が入力されました。
```

4 と 7 以外の数字を入力した場合、いずれの条件も満たさないため、③の処理が実行されます。

【実行結果（4 と 7 以外の数字を入力した場合）】

```
整数を入力してください。
2 ↲
4 と 7 以外の数字が入力されました。
```

重要
if～else if～else 文を使うことで、3 つ以上のルートに分岐させることができます。

switch 文

1. switch 文とは

switch 文は、if 文と同じく条件分岐の構文です。switch 文を使うことで複数の条件ごとに個別の処理を実行できます。

<構文 switch 文>

```
switch (式) {
    case 値1:
        処理1;
        break;
    case 値2:
        処理2;
        break;
    default:
        処理3;
        break;
}
```

switch 文の書き方は、まず予約語「switch」の右側にある「()」内に式を記述し、続けて「{}」を利用してブロックを記述します。ブロック内には式の評価に応じた処理を行う case 文を「case 値:」の形式で記述します。case 文は複数記述することができ、それら 1 つずつが 1 つの条件として機能します。なお、case 文は 3 つ以上記述することも可能です。

ブロック内の最後にはどの case にも当てはまらなかった場合の処理の記述を行う default 文を「default:」の形式で記述します。default 文は if～else if～else 文でいう「else 文」にあたる処理です。また、default 文の記載は必須ではないため、case 文に該当しない場合の処理が必要ない（if～else 文の else にあたる処理が必要ない）場合は省略しても問題ありません。

case 文と default 文の下には、その条件を満たした場合に実行する処理を記述します。処理は複数行記述することができます。注意点として、case 文と default 文の末尾には「:」（コロン）を忘れずに記述してください。また、各処理の最後には break 文を「break;」の形式で記述しましょう。break 文については後程詳しく紹介します。

続いて switch 文の処理の流れについて確認しましょう。switch 文では、「()」内の式の評価値が、case 文に記述された値と一致する場合、その case 文の処理を break 文まで実行します。そして、switch 文の一連の処理は終了します。

1 つ目の case 文が条件を満たさない場合は、2 つ目の case 文での判定に移ります。そして、同じ要領で 3 つ目以降の case 文の判定も行われます。もし、すべての case 文が条件を満たさない場合は default 文の下に記述された処理を実行します。

処理の流れを確認してみると、switch 文は if～else if～else 文に近い処理を行っていることが分かります。たとえば、次の switch 文と if～else if～else 文は同じ流れで処理が行われます。

<switch 文>

```
switch(曜日){
    case 月:
        友達とごはんに行く;
        break;
    case 火:
        ショッピングする;
        break;
    case 水:
        ピアノのレッスンに行く;
        break;
    default:
        図書館で勉強する;
        break;
}
```

<if～else if～else 文>

```
if (曜日 == 月) {
    友達とごはんに行く;
} else if (曜日 == 火) {
    ショッピングする;
} else if (曜日 == 水) {
    ピアノのレッスンに行く;
} else {
    図書館で勉強する;
}
```

ただし、switch 文と if～else if～else 文には文法上の違いがあり、必ずしも同じ処理を実装できるとは限りません。具体的な switch 文と if～else if～else 文の違いには次のようなものがあります。

■ 比較方法の違い

switch 文は、case 文の値と等しいか（または同じ内容か）で条件を判定します。つまり、関係演算子でいうところの「==」を使った比較しか行えません。「>」や「!=」に該当する比較は、switch 文では行えません。

■ 比較できる型の違い

switch 文は、「整数型（byte、short、int）」と「文字型（char）」の比較は行えますが、浮動小数点型（float、double）や論理型（boolean）での比較は行えません。ちなみに、switch 文では文字列型（String 型）での比較も行うことができます（Java7 以降のみ）。

■ 比較できる値の違い
switch 文では、比較する値として null 値を使用することができません。
null 値については Java テキスト(下巻)で詳しく紹介します。

■ 条件分岐の使い分け
if 文と switch 文の使い分けについては 223 ページにて紹介しています。

重要

switch 文を使って、if 文よりもシンプルに複数の分岐を記載することができます。

・条件毎に異なる処理を実行したい場合に使用。
・case文を記述できる回数に上限はない。
・全ての条件を満たさない場合の処理はdefault文の下に記述する。

デメリット：
値の不一致、大小比較などの条件では比較できない

それでは、switch 文を実装する練習として、下記のサンプルコードを作成してください。

【Sample0806 switch 文を使う】
■ 作成するファイル
/java_sample/src/lesson08/Sample0806.java

Sample0806.java

```
package lesson08;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample0806 {
    public static void main(String[] args) throws IOException {
        System.out.println("整数を入力してください。");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();
        int num = Integer.parseInt(str);

        switch (num) {
        case 4:
            // 変数numが4の場合の処理
            System.out.println("4が入力されました。");
            break;
        case 7:
            // 変数numが7の場合の処理
            System.out.println("7が入力されました。");
            break;
        default:
            // 変数numが上記以外の場合
            System.out.println("4と7以外の数字が入力されました。");
            break;
        }
    }
}
```

上記のサンプルコードは、Sample0805 と同じ流れで処理されるプログラムです。
4 を入力した場合、1 つ目の case 文の条件を満たすため、1 番目の case の処理が実行されます。

【実行結果（4 を入力した場合）】

```
整数を入力してください。
4 ↲
4 が入力されました。
```

7 を入力した場合、2 つ目の case 文の条件を満たすため、2 番目の case の処理が実行されます。

【実行結果（7 を入力した場合）】

```
整数を入力してください。
7 ↲
7 が入力されました。
```

4 と 7 以外の数字を入力した場合、いずれの case 文の条件も満たさないため、default の処理が実行されます。

【実行結果（4 と 7 以外の数字を入力した場合）】

```
整数を入力してください。
2 ↲
4 と 7 以外の数字が入力されました。
```

2. switch 文の注意点

switch 文を使うときは、以下の点に注意してください。

① 「switch」の直後に条件は書けない（例 num == 1）
② 「case」の横には値を書き、その後ろに「:（コロン）」を記述する（「;（セミコロン）」ではない）。
③ 「case」以降の処理の末尾に忘れずに「break 文」を記述する。

上記の中でも特に注意すべき点が「break 文の書き忘れ」です。break 文を書き忘れるとどのようなことが起こるのでしょうか。次のページのサンプルコードを実行して確認してみましょう。

【Sample0807 break 文の省略】
■ 作成するファイル
/java_sample/src/lesson08/Sample0807.java

Sample0807.java

```
package lesson08;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample0807 {
    public static void main(String[] args) throws IOException {
        System.out.println("整数を入力してください。");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();
        int num = Integer.parseInt(str);

        switch (num) {
        case 4:
            System.out.println("4が入力されました。");
        case 7:
            System.out.println("7が入力されました。");
        }
    }
}
```

4 を入力してみましょう。すると、1 つ目の case 文以降の処理もすべて実行されていることが分かります。

【実行結果】

```
整数を入力してください。
4 ↲
4 が入力されました。
7 が入力されました。
```

break 文は「ブロック内の処理の流れを強制的に中断する」という機能を持っています。そのため、break 文を書き忘れると、処理が中断されずに最後まで進んでしまいます。

重要
switch 文では break 文を書き忘れないように気をつけましょう。

論理演算子

1. 論理演算子とは

これまで条件分岐の方法についてご紹介してきましたが、論理演算子を利用することで複数の条件を組み合わせることができます。それでは次の例について見てみましょう。

> **今日が土曜日であり、かつ、お金があったら、**
> **国内旅行に行く。**

このように 2 つの条件を組み合わせて、より複雑な条件を使いたい場合には論理演算子を使います。上記の例を論理演算子「&&」を使用した条件として表現すると下記のようになります。

> **（ 今日が土曜日である ）&&（ お金がある ）**

「&&」は、左辺と右辺の条件が両方ともに「true」の場合に、全体を「true」と評価します。上記の例は、「今日が土曜日である」かつ「お金がある」ときに全体の評価が「true」となります。どちらか一方でも「false」の場合は、全体の評価は「false」となります。

論理演算子は次の表で紹介するようにいくつかの種類があります。

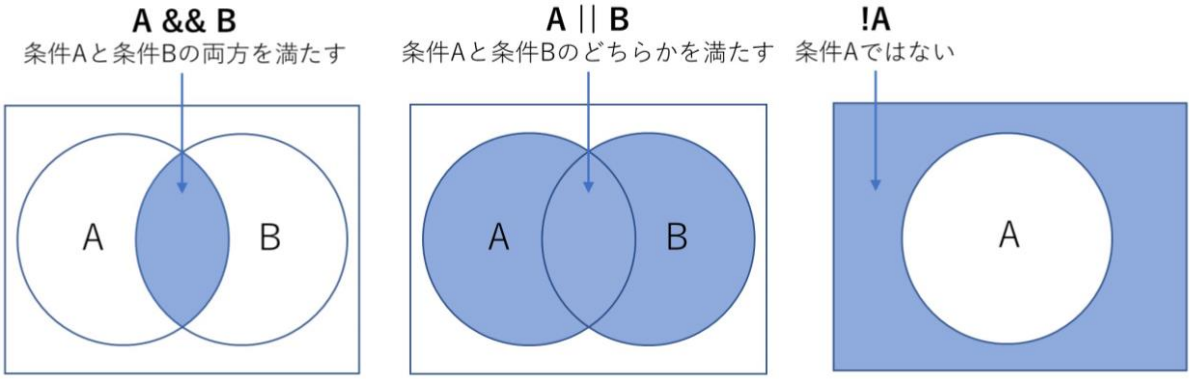
【表 8-2 論理演算子】

演算子	働き	意味	記述例
&&	かつ （論理積）	左辺と右辺の条件がどちらも true の場合、全体の評価は true	(a >= 4) && (a < 20)
||	または （論理和）	左辺と右辺の条件のいずれかが true の場合、全体の評価は true	(a == 3) || (a == 23)
!	～でない （論理否定）	条件が false の場合、全体の評価は true	!(a == 28)

論理和の演算子として利用する「|」は「パイプライン」または「パイプ」と呼びます。各論理演算子は図で表すと図 8-12 のようになりますので併せて確認しておきましょう。

OR
AとBのいずれか一方でも満たしている場合、条件全体を満たしていると判定
AND
AとBいずれの条件も見たいしている場合、条件全体を満たしていると判定

【図 8-12 論理演算子】



それでは、論理演算子の記述例を見てみましょう。

```
① 7==4 && 6>2
② 5<=8 || 9<2
③ 5<=4 || 9<20
④ !(7==8)
```

「&&」は、左辺と右辺両方ともに「true」となる場合のみ、全体の評価が「true」となります。したがって、条件①の評価は「false」となります。

「||」は、左辺と右辺のどちらかが「true」となれば、全体の評価が「true」となります。条件②では左辺が「true」であるため、全体の評価は「true」となります。条件③では右辺が「true」になるため、全体の評価は「true」となります。

「!」は、オペランドである条件を 1 つだけ必要とする単項演算子です。条件④ではオペランドの条件が「false」であるため、全体の評価は「true」となります。

それでは、次のページのサンプルコードを作成して実際に論理演算子を利用してみましょう。

【Sample0808 論理演算子を使う】
■ 作成するファイル
/java_sample/src/lesson08/Sample0808.java

Sample0808.java

```
package lesson08;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample0808 {
    public static void main(String[] args) throws IOException {
        System.out.println("整数を入力してください。");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();
        int num = Integer.parseInt(str);

        if (4 < num && num <= 10) {
            // ①numが4より大きく、かつ10以下の場合の処理
            System.out.println(num + "は4より大きく10以下の数字です。");
        } else if (num <= 4) {
            // ②numが4以下の場合の処理
            System.out.println(num + "は4以下の数字です。");
        } else {
            // ③それ以外の場合の処理
            System.out.println(num + "は10より大きい数字です。");
        }
    }
}
```

サンプルコードの実行後に 4 より大きくかつ 10 以下の数字を入力した場合は、1 つ目の条件を満たすため①の処理が実行されます。

【実行結果（4 より大きく、かつ 10 以下の数字を入力した場合)】

```
整数を入力してください。
5 ↲
5 は 4 より大きく 10 以下の数字です。
```

4 以下の数字を入力した場合、2 つ目の条件を満たすため、②の処理が実行されます。

【実行結果（4 以下の数字を入力した場合）】

```
整数を入力してください。
2 ↲
2 は 4 以下の数字です。
```

すべての条件を満たさない数字（10 より大きい数字）を入力した場合、③の処理が実行されます。

【実行結果（すべての条件を満たさない数字を入力した場合）】

```
整数を入力してください。
20 ↲
20 は 10 より大きい数字です。
```

上記のサンプルコードでは「&&」を使って条件を作成しました。

```
(4 < num && num <= 10)
```

この条件は、「if 文のネスト」の節で作成した Sample0803 中の条件と同じ条件分岐を行います。Sample0803 では、ネストにより複雑な構造の条件を記述しました。しかし、論理演算子を利用することで、上記のように 1 つの条件としてまとめて記述することができます。このように複雑な条件をより簡単な形で記述できる点が論理演算子を使用する大きなメリットとなります。

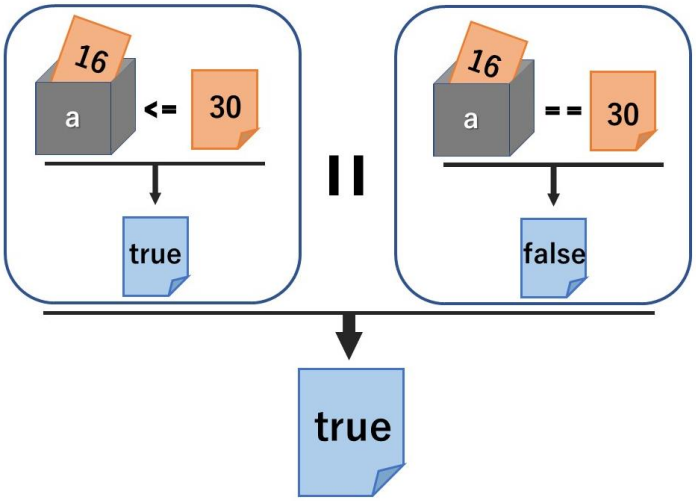
重要
論理演算子を使って、2 つ以上の条件を組み合わせた、
より複雑な条件を作ることができます。

2. 論理演算子の評価の仕組み

2 つのオペランド（条件）を必要とする論理演算子は、左辺の評価結果によって右辺の評価を行うか行わないかが決まります。

「&&」の場合は、左辺が「true」の場合のみ右辺の評価を行います。左辺が「false」だった場合、右辺が「true」であっても全体の評価が「false」になることは変わらないためです。このような評価の仕組みを「短絡評価」といいます。同様に「||」の場合は、左辺が「false」の場合のみ右辺の評価を行います。

【図 8-13 論理演算子】



また、論理否定の演算子「!」は、下記のように、boolean 型の変数をオペランドとしているときによく使用されます。

```
boolean flag = false;

if (!flag) {
    System.out.println(flag);
}
```

上記の条件は「flag != true」という条件式と同じ意味となります。boolean 型の変数と「!」を使用することで、「変数の値が true ではないか（false であるか）」の条件を簡潔に記述することができます。

条件演算子

下記のように「ある値と等しい場合に特定の値を、等しくない場合には別の値を変数に代入する」といった簡単な条件分岐の場合は、条件演算子「?:」を使用して簡潔な文に置き換えられます。

```
if (num == 0) {
    str = "A";
} else {
    str = "B";
}
```

<構文 条件演算子>

条件 ? true のときの式 : false のときの式 ;

条件演算子は、3 つのオペランドを取る演算子です。1 つ目には「条件」、2 つ目には「条件の評価が true の場合に実行したい式」、3 つ目には「条件の評価が false の場合に実行したい式」を記述します。
条件演算子は、条件の評価値により 2 つの式のどちらか一方が実行されます。そして、実行された式の評価値が条件演算子自体の評価値となります。

条件演算子を使用すると上記のソースコードは下記のように記述することができます。

```
str = (num == 0) ? "A" :"B";
```

重要
条件演算子「? :」を使うことで簡単な条件の処理を簡潔に記述できます。

章のまとめ

本章では、以下のことを学びました。

- ❑ 関係演算子を使うと条件を作成できます。
- ❑ if 文を使って条件に応じた処理を行うことができます。
- ❑ if~else 文、if~else if~else 文などを使って様々な条件に応じた処理を行うことができます。
- ❑ switch 文を使って複数の値に応じた処理を実行できます。
- ❑ 論理演算子を使って複雑な条件を作成できます。
- ❑ 条件演算子「? :」を使って簡単な条件に応じた処理を記述できます。

練習問題

1. 下記の実行結果になるように、switch 文を用いたプログラムを作成してください。

Mをコンソール入力した場合 ⇒ 「性別を男性で登録しました。」
F をコンソール入力した場合 ⇒ 「性別を女性で登録しました。」

■ 作成するファイル
/java_practice/src/lesson08/Practice0801.java

【実行結果】

```
性別を入力してください。
「M」か「F」を入力してください。
F ↲
性別を女性で登録しました。
```

2. 下記の実行結果になるように、if 文を用いてコードを作成してください。

うるう年の場合 ⇒ 「うるう年です。」
うるう年ではない場合 ⇒ 「うるう年ではありません。」

■ 作成するファイル
/java_practice/src/lesson08/Practice0802.java

【実行結果】

```
西暦年を入力してください。
2020 ↲
うるう年です。
```

■ うるう年を判断する条件
① 西暦年が 4 で割り切れて、かつ西暦年が 100 で割り切れない
② 西暦年が 400 で割り切れる

3. 下記の実行結果になるように、switch 文を用いたプログラムを作成してください。

1 の場合 ⇒ 「月曜日はカルビ丼です。」
2 の場合 ⇒ 「火曜日はケチャップオムライスです。」
3 の場合 ⇒ 「水曜日はカルボナーラです。」

■ 作成するファイル
/java_practice/src/lesson08/Practice0803.java

【実行結果】

```
［日替わりランチ］ 1:月曜日　2:火曜日　3:水曜日
気になる日替わりランチの曜日を数字で入力してください。
2 ↲
火曜日はケチャップオムライスです。
```

✓ 解答

1.

```
package lesson08;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Practice0801 {
    public static void main(String[] args) throws IOException {
        System.out.println("性別を入力してください。");
        System.out.println("「M」か「F」を入力してください。");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String ans = reader.readLine();

        switch (ans) {
        case "M":
            // 変数ansの値が「M」の場合
            System.out.println("性別を男性で登録しました。");
            break;
        case "F":
            // 変数ansの値が「F」の場合
            System.out.println("性別を女性で登録しました。");
            break;
        }
    }
}
```

2.

```
package lesson08;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Practice0802 {
    public static void main(String[] args) throws IOException {
        System.out.println("西暦年を入力してください。");

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();
        int year = Integer.parseInt(str);

        if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {
            System.out.println("うるう年です。");
        } else {
            System.out.println("うるう年ではありません。");
        }
    }
}
```

問題文の解説からうるう年の条件は下記のように表現することができます。

```
(4 で割り切れる && 100 で割り切れない) || (400 で割り切れる)
```

この条件をソースコードとして記述すると下記のようになります。

```
(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)
```

うるう年を判断する条件は他にも記述方法があるので気になる方は調べてみてください。

3.

```
package lesson08;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Practice0803 {
    public static void main(String[] args) throws IOException {
        System.out.println("[日替わりランチ]　1:月曜日　2:火曜日　3:水曜日");
        System.out.println("気になる日替わりランチの曜日を数字で入力してください。");

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();
        int day = Integer.parseInt(str);

        switch (day) {
        case 1:
            // day が 1（月曜日）だった場合の処理
            System.out.println("月曜日はカルビ丼です。");
            break;
        case 2:
            // day が 2（火曜日）だった場合の処理
            System.out.println("火曜日はケチャップオムライスです。");
            break;
        case 3:
            // day が 3（水曜日）だった場合の処理
            System.out.println("水曜日はカルボナーラです。");
            break;
        }
    }
}
```

第９章 繰り返し

for 文

1. for 文とは

本章では「繰り返し」について紹介します。まずは、下記内容を例として for 文について紹介します。

```
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
```

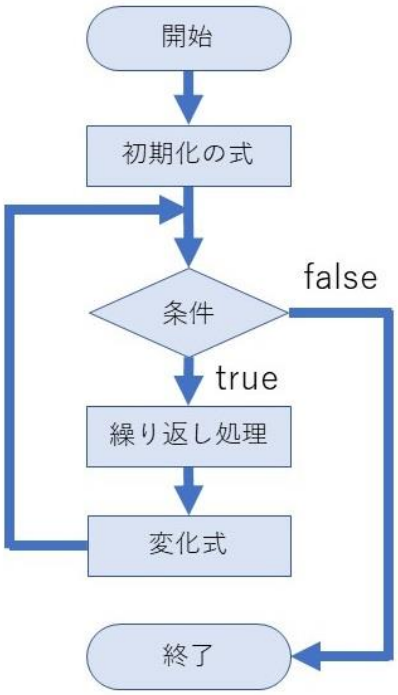
これは、買ったおにぎりの個数が 5 つになるまで、おにぎりを買い続けるという処理になります。このように、私たちは日常生活の中でも何かの処理を繰り返し行っており、こうした繰り返し行う動作は for 文を使用することでプログラムでも表現することができます。

<構文 for 文>

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

for 文のブロック内の処理は繰り返し実行されます。
for 文をフローチャートで表現すると次のように表すことができます。

【図 9-1 for 文のフローチャート】



「for」の後ろの「()」内には、セミコロン区切りで 3 つの式（①初期化の式、②条件式、③変化式）を記述します。

① 初期化の式

初期化の式はブロック内の繰り返し処理が始まる際に「最初に 1 回だけ実行される処理」です。一般的には「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します。このような変数をループカウンタと呼びます。ループカウンタの変数名は「i」「j」「k」を順に使用していくのが慣例となっています。

② 条件式

条件式はブロック内の処理を実行する前に評価される式で、繰り返しを継続するか否かを判断します。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します。評価が「true」の場合、ブロック内の処理を 1 回分実行します。

③ 変化式

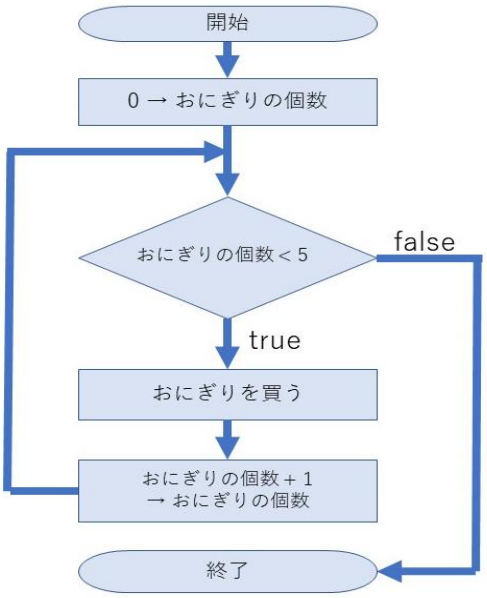
ブロック内の処理が完了した直後に実行される処理です。一般的には、「i++」のようにループカウンタの値を変化させる処理を記述します。

先ほどのおにぎりを購入する例を for 文で表現すると、次のようになります。

```
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;
}
```

この for 文をフローチャートで表現すると次のように表すことができます。

【図 9-2 for 文のフローチャート】



それでは、for 文を実装する練習として、下記のサンプルコードを作成してください。

【Sample0901 for 文を使う】
■ 作成するファイル
/java_sample/src/lesson09/Sample0901.java

Sample0901.java

```
package lesson09;

public class Sample0901 {
    public static void main(String[] args) {
        // iが5未満の場合、繰り返す
        for (int i = 0; i < 5; i++) {
            System.out.println("こんにちは");
        }
        System.out.println("繰り返しが終了しました。");
    }
}
```

【実行結果】

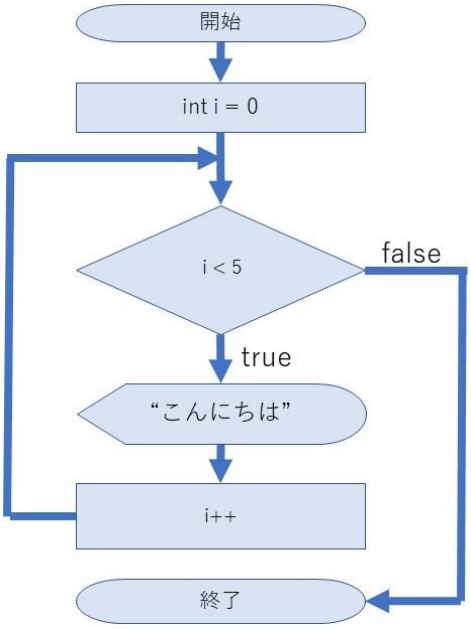
```
こんにちは
こんにちは
こんにちは
こんにちは
こんにちは
繰り返しが終了しました。
```

上記のサンプルコードは以下の流れで処理が実行されます。
なお、ループカウンタの名前は「i」です。

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。

この処理をフローチャートで表現すると次のように表すことができます。

【図 9-3 Sample0901 の繰り返しのフローチャート】



重要
「for 文」を使うと繰り返し処理を記述することができます。

2. ループカウンタの注意点

ループカウンタを使用する際は以下の 2 点に注意しましょう。

① ループカウンタの名前は自由に指定可能

ループカウンタは変数です。そのため、識別子のルールに則した名前なら自由に付けられます。ただし、慣習として「i」を使用することが多いため、特別な理由がない限りは、ループカウンタ名には「i」を記述することをお勧めします。

また、for 文の中に for 文を記述することもできます（for 文の入れ子構造については後述します）。その際、一番外側の for 文のループカウンタは i、その中の for 文のループカウンタは j、さらにその中は k というようにループカウンタ名を記述します。

② ループカウンタはブロック内でのみ利用可能

変数と同じようにループカウンタも for 文のブロック内で利用することができます。しかし、このループカウンタは for 文のブロック外では使えません。for 文が終了すると以降の処理ではループカウンタを利用することができないため注意しましょう。

3. ループカウンタの応用

ループカウンタを使用することで、for 文での繰り返し回数を出力させることもできます。
下記のサンプルコードを作成してください。

【Sample0902 繰り返し回数を出力】
■ 作成するファイル
/java_sample/src/lesson09/Sample0902.java

Sample0902.java

```
package lesson09;

public class Sample0902 {
    public static void main(String[] args) {
        // iが5未満の場合、繰り返す
        for (int i = 0; i < 5; i++) {
            System.out.println((i + 1) + "回繰り返しました。");
        }
        System.out.println("繰り返しを終了しました。");
    }
}
```

【実行結果】

```
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。
5回繰り返しました。
繰り返しを終了しました。
```

こちらのサンプルコードは繰り返し処理でループカウンタ i の値を出力しています。今回はループカウンタの初期値は 0 となっているため、その値に 1 を加算して繰り返した回数を表現しています。

ループカウンタの注意点でも触れましたが、ループカウンタ i は for 文のブロック内でのみ利用することができます。もし、ブロック外でも使用したい場合は、下記のように for 文が始まる前にループカウンタ i を宣言してください。

```
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。”);
```

また、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可能です。
たとえば、下記のような配列の要素を出力する処理を考えてみましょう。

```
int[] height = {162, 177, 154, 185};
System.out.println("1 人目の身長は" + height[0] + "cm です。");
System.out.println("2 人目の身長は" + height[1] + "cm です。");
System.out.println("3 人目の身長は" + height[2] + "cm です。");
System.out.println("4 人目の身長は" + height[3] + "cm です。");
```

上記のような記述方法では、標準出力の処理を配列の要素数分記述しなければならないため作成にとても手間がかかってしまいますが、この処理はループカウンタを使用することでもっと簡潔な形に書き直すことができます。下記のサンプルコードを作成して実際にそれを確認してみましょう。

【Sample0903 繰り返し文を配列に用いる】
■ 作成するファイル
/java_sample/src/lesson09/Sample0903.java

Sample0903.java

```
package lesson09;

public class Sample0903 {
    public static void main(String[] args) {
        int[] height = { 162, 177, 154, 185 };

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

【実行結果】

```
1人目の身長は162です。
2人目の身長は177です。
3人目の身長は154です。
4人目の身長は185です。
```

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ます。このとき、配列の添字は「0」から始まるため i の初期値も 0 にしましょう。そして、何人目の身長であるかを出力する際は「(i + 1)人目」と記述します。

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

条件式ではループカウンタ i と配列の要素数を比較しています。そのため、i の値が要素数未満であることが繰り返しの条件となります。

このように条件に「配列変数名.length」を使用すると配列の長さを取得できるため、配列の長さが変わったとしても条件の記述を変えることなく柔軟に繰り返し回数を調整することができます。

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

ちなみに、「i <= height.length」と記述すると i に要素数と同じ値が代入されるため、繰り返し処理の中で添字が存在しない配列の要素を呼び出してエラーとなってしまうため気を付けてください。

重要
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。

4. for 文の応用

for 文の応用として、入力された数字の回数だけ繰り返しを行うプログラムを作成しましょう。下記のサンプルコードを作成してください。

【Sample0904 入力した数だけ記号を出力する】
■ 作成するファイル
/java_sample/src/lesson09/Sample0904.java

Sample0904.java

```
package lesson09;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample0904 {
    public static void main(String[] args) throws IOException {
        System.out.println("いくつ*を出力しますか？");

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();
        int num = Integer.parseInt(str);

        // iが入力値以下の場合、繰り返す
        for (int i = 1; i <= num; i++) {
            System.out.print("*");
        }
    }
}
```

サンプルコードを実行するとコンソールで入力した数だけ「*」が出力されます。

【実行結果】

```
いくつ*を出力しますか？
15 ↲
***************
```

続いて 1 から入力した数までの総和を求めるプログラムを作成してみましょう。
次のサンプルコードを作成してください。

【Sample0905 入力した数までの合計を求める】
■ 作成するファイル
/java_sample/src/lesson09/Sample0905.java

Sample0905.java

```
package lesson09;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample0905 {
    public static void main(String[] args) throws IOException {
        System.out.println("いくつまでの合計を求めますか？");

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();
        int num = Integer.parseInt(str);

        // 総和を保存する変数
        int sum = 0;

        //  iが入力値以下の場合、繰り返す
        for (int i = 1; i <= num; i++) {
            // sumにiを加算代入する
            sum += i;
        }

        System.out.println("1から" + num + "までの合計は" + sum + "です。");
    }
}
```

【実行結果】

```
いくつまでの合計を求めますか？
10 ↲
1 から 10 までの合計は 55 です。
```

先ほどと同じように、ユーザが入力した数まで繰り返し処理を行っています。ここで重要なのは、変数 sum にループカウンタ i の値を加算代入している処理です。変数 i の値は「i++」により 1 つずつ加算されます。そのため、1 から入力値までの総和を求めることができます。

【図 9-4 入力した数までの合計を求める】

	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

コラム 繰り返しの方法

for 文の後に続く「()」には、様々な記述パターンがあります。繰り返しを行いたいときに、どういう条件なのかを考えて記述しましょう。たとえば、5 回の繰り返しを行う記述方法でも一例として以下のようなパターンが考えられます。

```
for(int i = 0; i < 5; i++){
    …
}
```
← 0~4 の整数 i を昇順に処理する

```
for(int i = 1; i <= 5; i++){
    …
}
```
← 1～5 の整数 i を昇順に処理する

```
for(int i = 5; 1 <= i; i--){
    …
}
```
← 5～1 の整数 i を降順に処理する

5. 拡張 for 文

拡張 for 文とは、1 回の繰り返しごとに配列の要素を先頭から順番に取り出して処理することのできる構文です。

＜構文 拡張 for 文>

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

下記のサンプルコードを見てください。

```
int[] tests ={45, 80, 76, 56, 55};

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

上記の for 文は拡張 for 文を使用してより簡潔に記述することができます。
それでは拡張 for 文の動作を確認するために下記のサンプルコードを作成してみましょう。

【Sample0906 配列を拡張 for 文で使う】
■ 作成するファイル
/java_sample/src/lesson09/Sample0906.java

Sample0906.java

```
package lesson09;

public class Sample0906 {
    public static void main(String[] args) {
        int[] tests = { 45, 80, 76, 56, 55 };

        // 1回の繰り返しごとに、要素の値が変数valueに代入される
        for (int value : tests) {
            System.out.println(value);
        }
    }
}
```

【実行結果】

```
45
80
76
56
55
```

拡張 for 文では、1 回目の繰り返しでは添字 0 番の要素の値を取り出し、2 回目の繰り返しでは添字 1 番の要素の値を取り出し、「:」の左側で宣言した変数に代入します。そして、その変数はブロック内で使用することができます。

for 文とは異なり、拡張 for 文はループカウンタや添字の指定が不要なため、より簡潔にコードを記載することができます。

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ます。for 文のネストの構文は下記の通りです。

<構文 for 文のネスト>

```
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;
    }
}
```

それでは下記のサンプルコードを作成してください。

【Sample0907 for 文のネスト】
■ 作成するファイル
/java_sample/src/lesson09/Sample0907.java

Sample0907.java

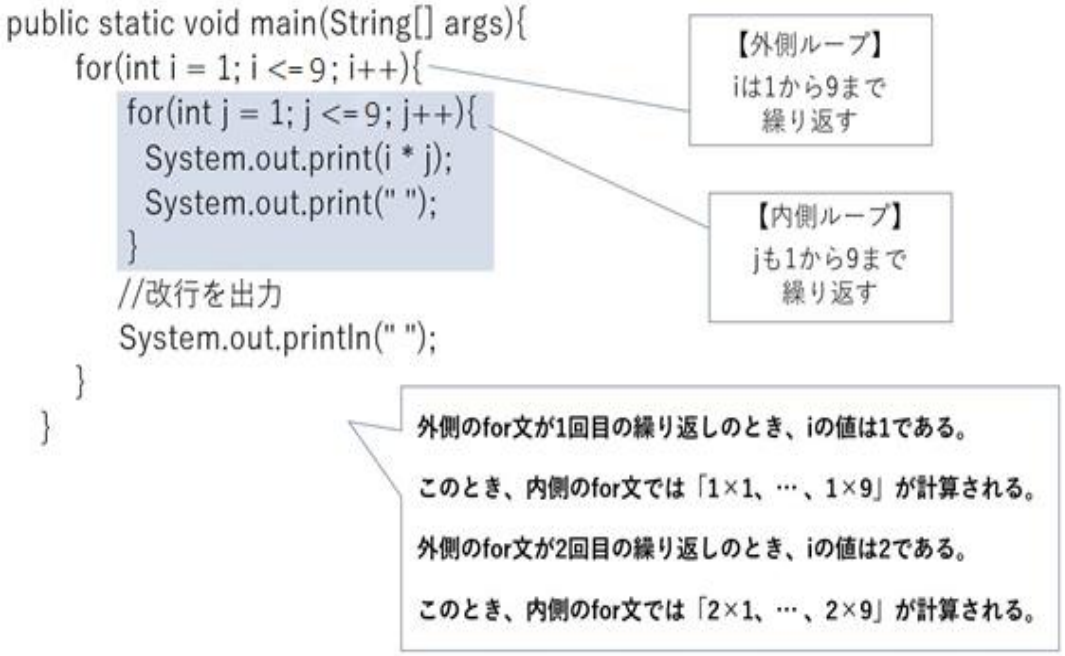
```
package lesson09;

public class Sample0907 {
    public static void main(String[] args) {
        for (int i = 1; i <= 9; i++) {
            for (int j = 1; j <= 9; j++) {
                System.out.print(i * j);
                System.out.print(" ");
            }
            System.out.println(" ");
        }
    }
}
```

【実行結果】

```
1 2 3 4 5 6 7 8 9
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

上記のサンプルコードでは九九を出力しています。内側の for 文の繰り返し 1 回につき、掛け算を行います。そして、内側の for 文が終了すると外側の for 文の繰り返しが 1 回分終了します。このような流れで処理が進み、計算は外側の for 文が終わるまで繰り返し行われます。

【図 9-5 for 文のネスト】



重要
for 文をネストすると多重の繰り返し処理を記述できます。

while 文

1. while 文とは

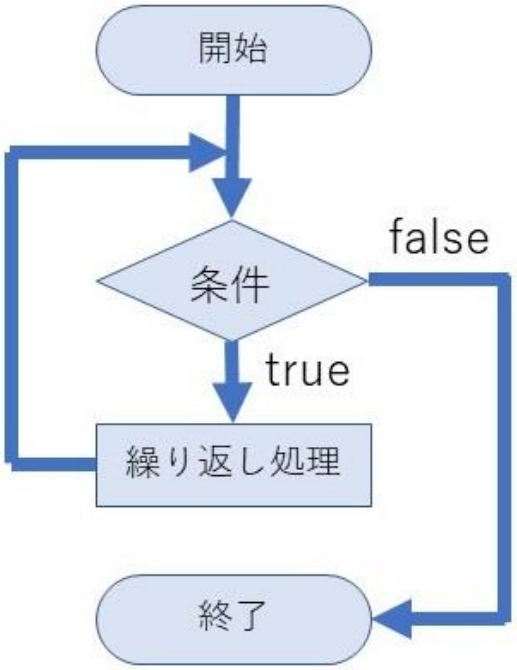
while 文は for 文と同じく繰り返し処理を行うための構文です。

<構文 while 文>

```
while(条件){
    繰り返し処理;
}
```

while 文は、指定された条件が満たされている限りブロック内の処理を繰り返し実行します。
while 文の構文をフローチャートで表現すると次のように表すことができます。

【図 9-6 while 文のフローチャート】



それでは while 文の構文について、おにぎりの購入を行う場合を例にして確認してみましょう。

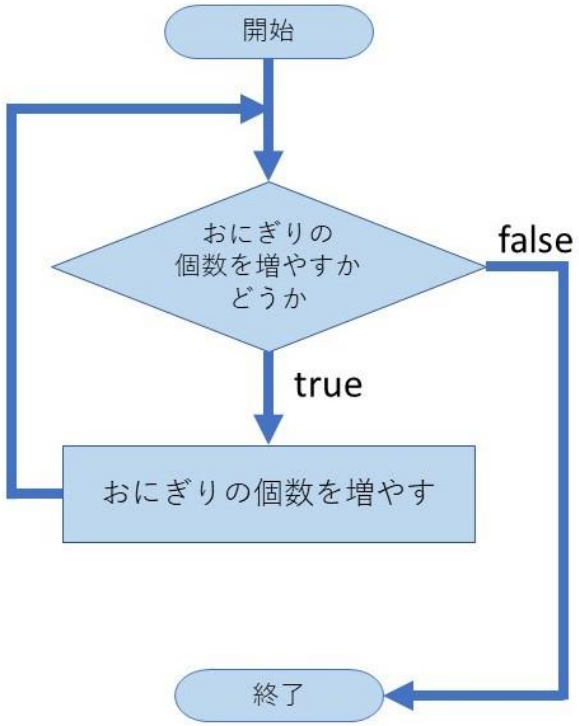
購入するおにぎりの個数を増やすかどうか
⇒いいえを選ぶまでおにぎりの個数を増やす

前ページの内容を while 文の構文に当てはめると次のようになります

```
while(おにぎりの個数を増やすかどうか){

    おにぎりの個数を増やす;
}
```

while 文では、「おにぎりの個数が 5 個未満である」という条件が false になるまで処理を繰り返します。これをフローチャートで表現すると次のように表すことができます。

【図 9-7 while 文】



それでは、while 文の練習として下記のサンプルコードを作成してください。

【Sample0908 while 文を使う】
■ 作成するファイル
/java_sample/src/lesson09/Sample0908.java

Sample0908.java

```
package lesson09;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample0908 {
    public static void main(String[] args) throws IOException {
         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int i = 0;
        System.out.println("おにぎりを購入しますか?");
        System.out.print("はい：0、いいえ：1 >");
        String choise = br.readLine();
        int buyFlag = Integer.parseInt(choise);
        // buyFlag が 0 の場合、繰り返す
        while (buyFlag == 0) {
            i++;
            System.out.println("購入するおにぎりの個数を 1 つ増やしますか？");
            System.out.print("はい：0、いいえ：1 >");
            choise = br.readLine();
            buyFlag = Integer.parseInt(choise);
        }

        System.out.println("購入したおにぎりの個数は" + i + "個です。");
    }
}
```

【実行結果】

```
おにぎりを購入しますか?
はい：0、いいえ：1 >0
購入するおにぎりの個数を 1 つ増やしますか？
はい：0、いいえ：1 >0
購入するおにぎりの個数を 1 つ増やしますか？
はい：0、いいえ：1 >1
購入したおにぎりの個数は 2 個です。
```

この while 文では、条件である「buyFlag == 0」が「false」になるまで処理を繰り返しています。

```
while (buyFlag == 0) {
            i++;
            System.out.println("購入するおにぎりの個数を1つ増やしますか？");
            System.out.print("はい：0、いいえ：1 >");
            choise = br.readLine();
            buyFlag = Integer.parseInt(choise);
        }
```

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、入力された選択肢を int 型に変換し、buyFlag に代入しています。

2. 無限ループ

繰り返し文では、条件の評価が常に「true」だと、処理が無限に繰り返されます。このことを「無限ループ」といいます。例として次のサンプルコードを見てください。

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

上記のサンプルコードは、条件が i < 5 の while 文から最終的に条件が false になるための処理である「i++」を抜いたものです。そのため、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」となります。

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かけてしまう恐れがあります。また、プログラムが強制的に止まってしまうこともあるため、無限ループにならないように気を付けて実装を行いましょう。

重要
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

1. do~while 文とは

do~while 文も、while 文と同じように繰り返しを行う構文です。while 文と同様に条件が「true」となっている間に繰り返し処理が実行されます。

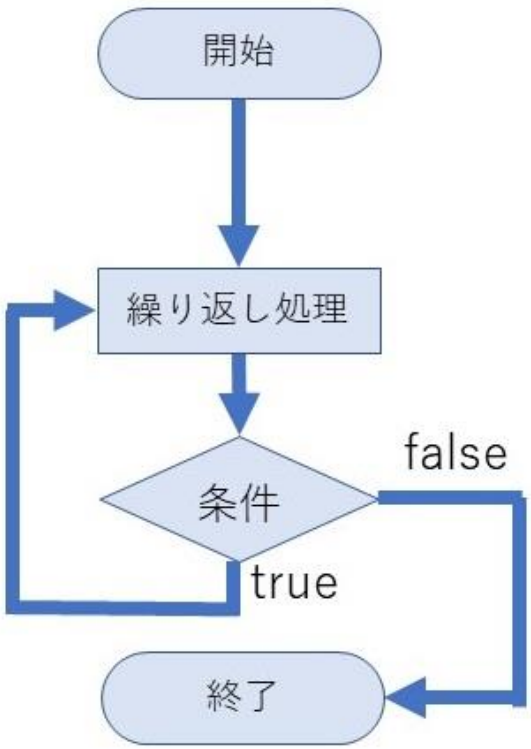
<構文 do~while 文>

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

while 文では繰り返し処理よりも先に条件を評価するため、1 回目の条件の評価で条件が「false」の場合、1 度も繰り返し処理は実行されません。それに対して、do~while 文は最初の 1 回は必ず繰り返し処理を実行します。そして、繰り返し処理が終了したタイミングで条件の評価を行い、条件が「true」の場合は再度繰り返し処理を実行します。

do~while 文の構文をフローチャートで表現すると、下記のようになります。

【図 9-8 do~while 文のフローチャート】



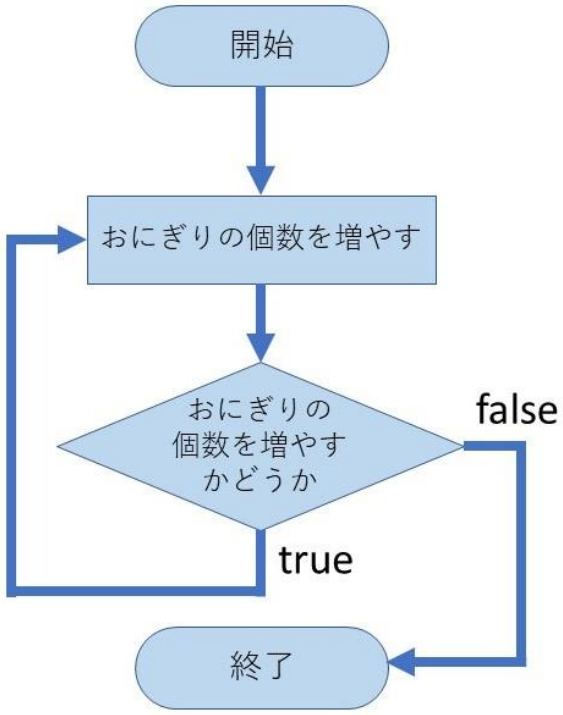
次に、おにぎりを購入する例を do~while 文に書き換えた場合の構文を確認してみましょう。

```
do {
        おにぎりの数を増やす;
        おにぎりの数を増やすのか選択;
} while（おにぎりの個数をさらに増やすかどうか）;
```

while 文と同様に、おにぎりの個数を増やし続けるという処理を繰り返します。しかし、do~while 文の場合、最初の一回目はおにぎりの数がどのくらいあるかにかかわらず繰り返し処理を実行します。

上記の内容をフローチャートで表現すると、次のように表すことができます。

【図 9-9 do~while 文】



それでは、do~while 文を実装する練習として、下記のサンプルコードを作成してください。

【Sample0909 do~while 文を使う】
■ 作成するファイル
/java_sample/src/lesson09/Sample0909.java

Sample0909.java

```
package lesson09;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample0909 {
          public static void main(String[] args) throws IOException {
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            int i = 0;
            int buyFlag = 0;
            System.out.println("おにぎりを購入します。");

            do {
                i++;
                System.out.println("購入するおにぎりの個数を１つ増やしますか？");
                System.out.print("はい：0、いいえ：1 >");
                String choise = br.readLine();
                buyFlag = Integer.parseInt(choise);
                // buyFlag が０の場合、繰り返す
               } while (buyFlag == 0);
                System.out.println("購入したおにぎりの個数は" + i + "個です。");
          }
}
```

【実行結果】

```
おにぎりを購入します。
購入するおにぎりの個数を1つ増やしますか？
はい：0、いいえ：1 >0
購入するおにぎりの個数を1つ増やしますか？
はい：0、いいえ：1 >0
購入するおにぎりの個数を1つ増やしますか？
はい：0、いいえ：1 >1
購入したおにぎりの個数は3個です。
```

上記のサンプルコードでは、do~while 文を使って 3 回繰り返し処理を実行しています。

重要

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回繰り返し処理を実行します。

処理の流れの変更

1. break 文とは

繰り返し処理の流れを変更したい場合は break 文や continue 文を使用します。まずは、break 文から確認していきましょう。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ちます。ここでは繰り返しの場合における break 文の利用方法について紹介します。

＜構文 break 文>

```
break;
```

それでは、下記のサンプルコードを作成してください。

【Sample0910 break 文で処理を中断する】
■ 作成するファイル
/java_sample/src/lesson09/Sample0910.java

Sample0910.java

```
package lesson09;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample0910 {
    public static void main(String[] args) throws IOException {
        System.out.println("何回目の繰り返しで中止しますか？(1～10)");

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String str = reader.readLine();
        int num = Integer.parseInt(str);

        for (int i = 1; i <= 10; i++) {
            System.out.println(i + "回繰り返しました。");
            if (i == num) {
                // 変数iが変数numと等しい場合、繰り返しを中断する
                break;
            }
        }

        System.out.println("繰り返しを中断しました。");
    }
}
```

こちらのサンプルコードの for 文は合計 10 回繰り返す条件となっています。しかし、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します。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ります。

【実行結果】

```
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。
```

ちなみに、繰り返しのブロックをネストしている場合は、内側のブロックで break 文を実行すると、”内側の繰り返しのみ”が中断され、外側のブロックの処理に移ります。すべての繰り返し処理が中断されるわけではないので注意してください。

重要
break 文を使うことで繰り返しを強制的に中断することができます。

2. continue 文とは

続いて、continue 文を紹介します。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ちます。つまり、continue 文以降の処理がスキップされるということです。

<構文 continue 文>

```
continue;
```

それでは、次のページのサンプルコードを作成してください。

【Sample0911 continue 文で処理をスキップする】
■ 作成するファイル
/java_sample/src/lesson09/Sample0911.java

Sample0911.java

```
package lesson09;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample0911 {
    public static void main(String[] args) throws IOException {
        System.out.println("何回目の繰り返しを中断しますか？(1～10)");

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String str = reader.readLine();
        int num = Integer.parseInt(str);

        for (int i = 1; i <= 10; i++) {
            if (i == num) {
                // 残りの処理を行わずに次の繰り返しに移動する
                continue;
            }
            System.out.println(i + "回繰り返しました。");
        }
    }
}
```

上記の for 文は合計 10 回繰り返す条件となっています。しかし、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、continue 文より下に記述された処理が実行されないまま、次の繰り返しに移ります。なお、continue 文により残りの処理がスキップされた場合でも「1 回分の繰り返しが完了した」と見なされるため、次の繰り返しが開始する前に for 文の変化式が実行されます。

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ません。そして、続けて 5 回目以降の繰り返し処理が実行されたことが分かります。

【実行結果】

```
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。
7回繰り返しました。
8回繰り返しました。
9回繰り返しました。
10 回繰り返しました。
```

重要

continue 文を使うことで繰り返し処理を途中で中断して次の繰り返し処理に移ることができます。

制御文のまとめ

1. 制御文の使い分け

ここまで前章と本章を通して制御文についての学習を行いました。
それぞれの制御文にどのような違いがあるのかをここで一度整理しておきましょう。

■ 条件分岐（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です。この 2 つの違いは、書式の違いによる「①ソースコードの読みやすさ」と「②実行速度」の 2 つです。この 2 つの観点から、if 文と switch 文の使い分けを考えてみましょう。

【if 文と switch 文の使い分け】
if 文の場合は、式の評価が true なのか false なのかによって処理を分岐しています。それに対して、switch 文の場合は、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します。
条件分岐の処理は大きく分けると「二分岐」と呼ばれる 2 つの処理のどちらかを実行するものと、「多分岐」と呼ばれる 3 つ以上の処理のどれか 1 つを実行するものに分けることができます。if 文と switch 文の使い分けはこの「二分岐」と「多分岐」の観点で考えることができます。

① 二分岐の場合
二分岐の処理を if 文と switch 文それぞれで行うと、次のようになります。

【if 文の場合】

```
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 文の場合】

```
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;
}
```

前ページの 2 つを見て分かるように、二分岐の場合は「if 文」を使ったほうが見やすいコードとなることがよく分かります。また、実行速度に関してはどちらもほとんど変わりません。

② 多分岐の場合

多分岐の処理を if 文と switch 文それぞれで行うと、次のようになります。

【if 文の場合】

```
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 文の場合】

```
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;
}
```

多分岐の場合も switch 文のほうが行数は多いですが、ソースコードは二分岐のときよりも読みやすくなっています。また、多分岐の場合、実行速度も switch 文のほうが if 文よりも速いとされています。if 文は条件を上から順番に評価していくのに対して、switch 文は式の評価を求めるのは 1 度だけで、そのあとは同じ値を見つけるだけという違いがあるからです。そのため、多分岐のような複数回評価を行う場合には、「switch 文」を使うことが良い場合があります。ただ、実際の現場では可読性の観点から switch 文の使用を禁止している場合もあります。

このように、「いくつの条件で」分岐を行っているのかをしっかり確認することで if 文と switch 文の使い分けが行いやすくなります。

また、switch 文を記述する際の注意事項は次の通りです。

①すべての条件式が「変数 == 値」「変数 == 変数」のように
　左辺と右辺が一致するかどうかの比較を行っていること。（「>」や「!=」は使えない）

②比較する値が整数（byte 型、short 型、int 型）、
　文字列や文字（String 型、char 型）であること。（小数や真偽の値は使えない）

開発現場によっては、「switch 文は原則使用してはいけない」などのコーディング規約が設けられていることがあります。開発プロジェクトに参画した際にはコーディング規約の内容をよく確認しておくようにしましょう。

■ 繰り返し（for 文と while 文、do~while 文）

【for 文と while 文の違い】
for 文と while 文のどちらの構文を利用しても繰り返し処理を記述することは可能です。これらの違いは「繰り返し処理を行う回数が決まっているか決まっていないか」といった違いがあります。この観点から for 文と while 文の使い分けを考えてみましょう。

【while 文と do~while 文の違い】
while 文と do~while 文はどちらも while 文の特徴である「繰り返し処理を行う回数が決まっていない」という場合に使われることが多いです。しかし do~while 文は while 文と違い、条件にかかわらず、絶対に 1 回は実行されるといった違いがあります。

【for 文と while 文、do~while 文の使い分け】
for 文と while 文、do~while 文それぞれの構文は次の通りです。

【for 文の場合】

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

【while 文の場合】

```
while(条件){
    繰り返し処理;
}
```

【do~while 文の場合】

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

構文から次のようなことが分かります。

① for 文は初期化の式、条件、変化式を明記しないといけないため、条件が視覚的に分かりやすいため、「繰り返し回数が決まっている」処理に適している。
② while 文は「繰り返し回数が決まっていない」処理に適している。
③ do~while 文は「必ず一回は実行させたい」処理に適している。

このように、for 文と while 文の使い分けは「繰り返す回数が決まっているかどうか」を基準に考えてみましょう。do~while 文を使用するかは、1 回目から実行の判定を行うかで決めましょう。

2. 制御文の組み合わせ

ここでは、「配列の要素を並び替える」という処理を例として制御文の組み合わせについて学びましょう。

拡張 for 文（または for 文）と配列に加えて、Arrays.sort()メソッドという処理を組み合わせると、昇順、降順、アルファベット順などの特定のルールに従って配列の要素を並び替えることができます。このような並び替えを行う処理は一般的にソート(sort)と呼びます。

それでは下記のサンプルコードを作成してください。

【Sample0912 配列のソート】
■ 作成するファイル
/java_sample/src/lesson09/Sample0912.java

Sample0912.java

```
package lesson09;

import java.util.Arrays;

public class Sample0912 {
    public static void main(String[] args) {
        int[] numbers = { 3, 4, 2, 1, 5 };

        // 配列をソートする
        Arrays.sort(numbers);

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

【実行結果】

```
1
2
3
4
5
```

このサンプルのような記述を行うと配列を昇順に並び替えることができます。
配列をソートする場合は、「Arrays.sort(配列変数名)」と記述します。

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

また、クラスブロックの手前には「import java.util.Arrays」と記述します。

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

サンプルでは昇順のみを取り上げましたが、ソートの方法は上記以外にもあります。気になる人は適宜調べてみてください。

章のまとめ

for(初期化式;条件式;変化式)
while(条件式)
do{}while(条件式)

本章では、以下のことを学びました。

- ❑ for 文を使うと繰り返し処理を行うことができます。
- ❑ while 文を使うと繰り返し処理を行うことができます。 処理実行前にループ判定
- ❑ do~while 文を使うと繰り返し処理を行うことができます。 一度処理を実行してからループ判定
- ❑ for 文はネストすることができます。
- ❑ 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。

練習問題

1. 下記の要件に沿ったプログラムを作成してください。

- ユーザに整数を連続して入力させる。
- 「-1」が入力されると、それまでに入力した整数の合計値を実行結果の形式で出力する。

■ 作成するファイル
/java_practice/src/lesson09/Practice0901.java

【実行結果】

```
整数を入力してください。
34 ↲
56 ↲
78 ↲ ↲
-1 ↲
整数の合計値は 168 です。
```

2. 入力値の数値分、「*」を実行結果になるような形式で出力するプログラムを作成してください。

■ 作成するファイル
/java_practice/src/lesson09/Practice0902.java

【実行結果①】

```
整数を入力してください。
5 ↲
␣␣␣␣*
␣␣␣*
␣␣*
␣*
*
```

「␣」は半角スペース 1 個分を表しています。
半角スペースを複数個出力した後に * を 1 つ出力してください。

【実行結果②】

```
整数を入力してください。
3
␣␣*
␣*
*
```

3. 実行結果になるように、入力した数値が素数かどうかを判定するプログラムを作成してください。
   なお、入力できる整数は 2 以上としますが、2 未満の整数値が入力された場合の入力チェック処理は実装不要です。
   また、「素数」とは、1、および同じ値でしか割り切れない整数のことです。

   ■ 作成するファイル
   /java_practice/src/lesson09/Practice0903.java

【実行結果（素数を入力した場合）】

```
整数を入力してください。
11 ↲
入力された整数は素数です。
```

【実行結果（素数以外の数値を入力した場合）】

```
整数を入力してください。
35 ↲
入力された整数は素数ではありません。
```

✓ 解答

1.

```
package lesson09;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Practice0901 {
    public static void main(String[] args) throws IOException {
        System.out.println("整数を入力してください。");

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        int num = 0;
        int sum = 0;

        while (num != -1){
            String str = reader.readLine();
            num = Integer.parseInt(str);
            if (num != -1) {
                sum += num;
            }
        }

        System.out.println("整数の合計値は" + sum + "です。");
    }
}
```

まず、下記のように変数 num と変数 sum の初期値を「0」に設定します。

```
int num = 0;
int sum = 0;
```

そのあと、次のページのように入力値を加算する処理を記述します。num に「-1」が代入されないかぎり sum に num を加算する処理を繰り返し実行します。「-1」が入力された場合、while 文の条件が「false」となってループを抜けます。

```
while (num != -1){
    String str = reader.readLine();
    num = Integer.parseInt(str);
    if (num != -1) {
        sum += num;
    }
}
```

2.

```
package lesson09;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Practice0902 {
    public static void main(String[] args) throws IOException {
        System.out.println("整数を入力してください。");

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String str = reader.readLine();
        int num = Integer.parseInt(str);

        // for文①
        for (int i = 1; i <= num; i++) {
            // for文②
            for (int j = 1; j <= num - i; j++) {
                System.out.print(" ");
            }
            System.out.print("*¥n");
        }
    }
}
```

外側の for 文（for 文①）が改行の回数を制御し、内側の for 文（for 文②）がスペースの出力回数を制御していると考えると分かりやすいです。

for 文②では、「入力値 - i」分のスペースを出力しています。for 文②が最初に実行される時点では i は 1 のため、 num－1 回分スペースを出力します。for 文②の処理 1 回分が終わると「*」の出力と改行を実行します。

以上の処理が 1 回終わるごとに for 文①の i が 1 増えていきます。

3.

```
package lesson09;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Practice0903 {
    public static void main(String[] args) throws IOException {
        System.out.println("整数を入力してください。");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();
        int num = Integer.parseInt(str);

        int i;
        for (i = 2; i < num; i++) {
            if (num % i == 0) {
               System.out.println(“余りが 0 なので、素数ではありません”);
                break;
            }
        }

        if (i == num) {
            System.out.println("入力された整数は素数です。");
        }
    }
}
```

まずは、for 文を使って素数かどうかを判定する数字を 2 の値から開始します。for 文の中に、if 文を記述し、素数でない場合を判定します。もし、i が入力値 num より小さいときに「num ÷ i」が割り切れた場合、その入力値は「素数ではない」と判断されます。素数でないことが判明した時点で、残りの繰り返し処理は不要になりますので break 文で for 文を抜けます。

for 文のあとで、あらためて if 文で素数かどうか判定します。もし、i の値と num の値が等しかった場合は、for 文の中で break 文が実行されなかったということになり、素数であることが分かります。それ以外のケースは break 文で for 文を抜けているので素数ではありません。

```
if (i == num) {
    System.out.println("入力された整数は素数です。");
}
```

第 10 章 デバッグ①

デバッグとは

プログラムが想定通りに実行されない場合、そのプログラム中にはバグと呼ばれる想定外の動作を行っている処理が存在します。そのバグを探して修正する作業のことをデバッグといいます。

1. Eclipse でのデバッグ

Eclipse には「デバッグモード」と呼ばれる機能があります。この機能は「特定の処理(文)が実行される手前でプログラムを一時停止する」、「一つ一つの処理を手動で実行する」などの操作を可能にします。

このデバッグモードを活用することで、「特定の処理を実行した後に変数にどのような値が代入されるか」、「各処理が想定通りの流れで実行されるか」などを確認することができます。ここではデバッグモードの具体的な操作手順を紹介します。

※　操作手順の説明では Eclipse3.7 の画面を利用しています。研修で使用している Eclipse とバージョンが異なるため、若干画面レイアウトに差異がありますが項目名は同様となります。

それでは実際にデバッグを行う準備として以下のサンプルコードを作成しましょう。

【Sample1001 入力した整数値に応じて異なる文字列を標準出力する】

■ 作成するファイル

/java_sample/src/lesson10/Sample1001.java

Sample1001.java

```
package lesson10;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sample1001 {
    public static void main(String[] args) throws IOException {
        System.out.println("整数値を入力してください。");

        // 数字の入力処理
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String str = br.readLine();
        int num = Integer.parseInt(str);

        // 入力された数値を確認し、それに応じて異なる文字列を出力
        if (num < 0) {
            System.out.println(num + "：0未満の値です。");
        } else if (0 <= num && num < 10) {
            System.out.println(num + "：0以上10未満の値です。");
        }

        System.out.println("以上で処理は終了です。");
    }
}
```

デバッグモードの操作方法

続いて、先ほど作成したプログラムを使用してデバッグモードの操作方法を紹介します。デバッグモードの大まかな操作手順は以下のようになります。

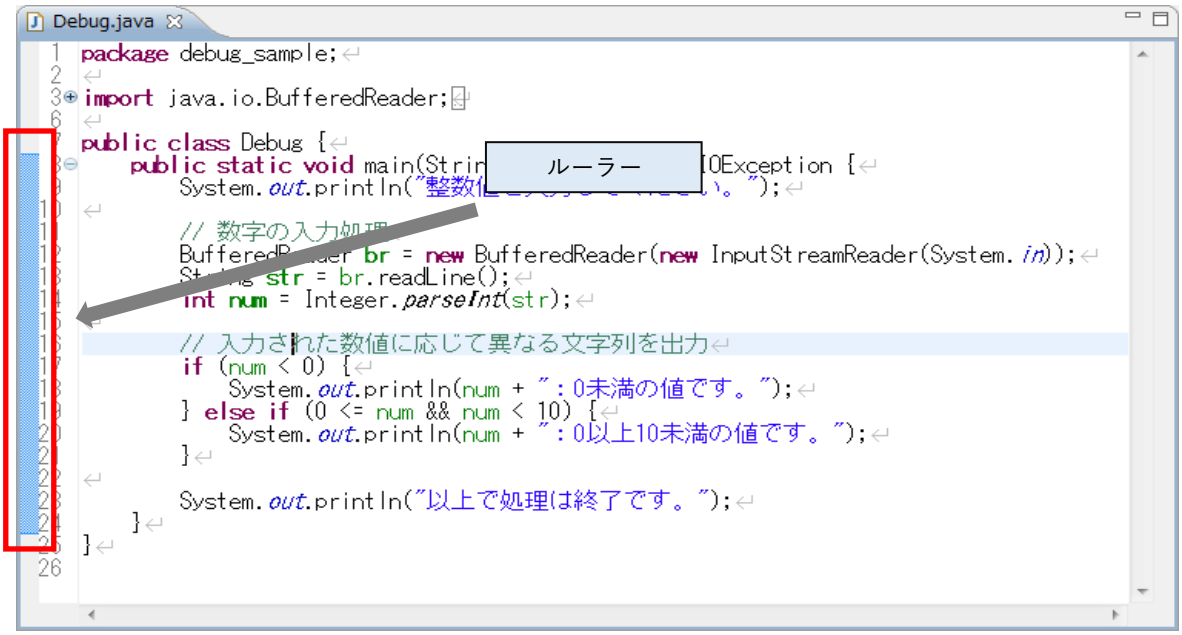
1. ソースコード中で一時停止したい処理を指定する
2. デバッグモードでプログラムを実行する
3. 一文ずつ処理を実行する
4. 実行結果を確認する
5. デバッグモードを終了する
6. バグを修正する

それでは、各手順の具体的な内容について確認していきましょう。

1. ソースコード中で一時停止したい処理を指定する

1.1 ルーラーを表示する

プログラムを実行中に一時停止させたい処理をソースコード中で指定します。ソースコード中のmain()メソッド内の適当な箇所をマウスカーソルで選択してください。すると、ソースコード画面の左端に青い帯が表示されます。この帯のことをルーラーと呼びます。

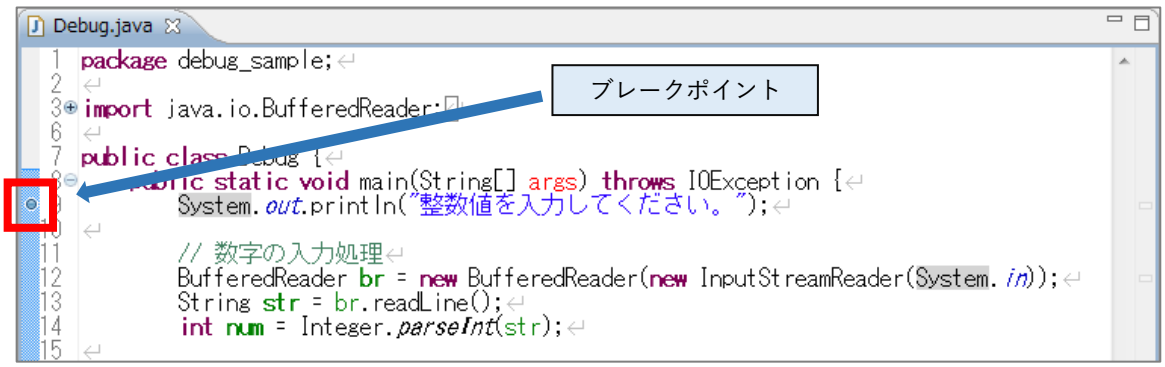


1.2 ブレークポイントを表示する

下記処理の位置でルーラーをダブルクリックしてみてください。

```
System.out.println("整数値を入力してください。");
```

すると、ルーラー上に小さく青い丸印が表示されます。この丸印のことをブレークポイントと呼びます。ブレークポイントを置いた場所で処理が一旦止まるため、ブレークポイントを置いた場所までの処理の流れや変数の状態を確認できます。

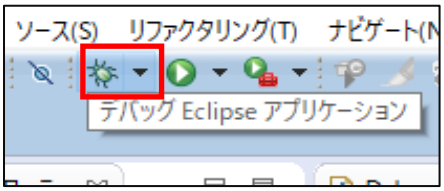


なお、ブレークポイント上で再度ダブルクリックすると、対象のブレークポイントは削除されます。必要に応じて、ブレークポイントは表示もしくは削除しましょう。また、ブレークポイントは 1 つのソースコード中に複数箇所表示することができます。ユーザは処理の流れを想定し、バグが起きた可能性のある処理にブレークポイントを打ちバグを探します。

2. デバッグモードでプログラムを実行する

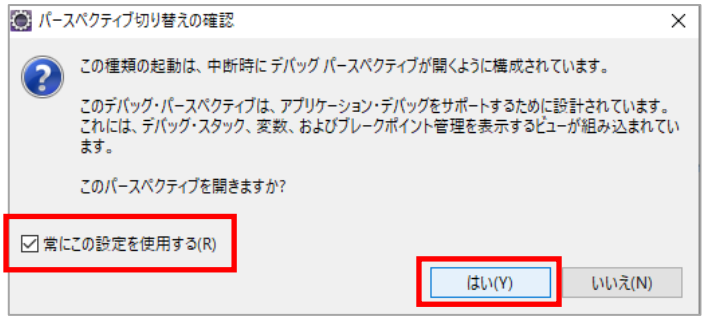
2.1 「デバッグ」ボタンを押す

ブレークポイントを置いたらソースコード中で一時停止したい処理を指定する作業は完了です。次はデバッグモードを起動し、プログラムを実行させます。Eclipse 中のツールバーに表示された「デバッグ」ボタン(下図参照)を押してください。

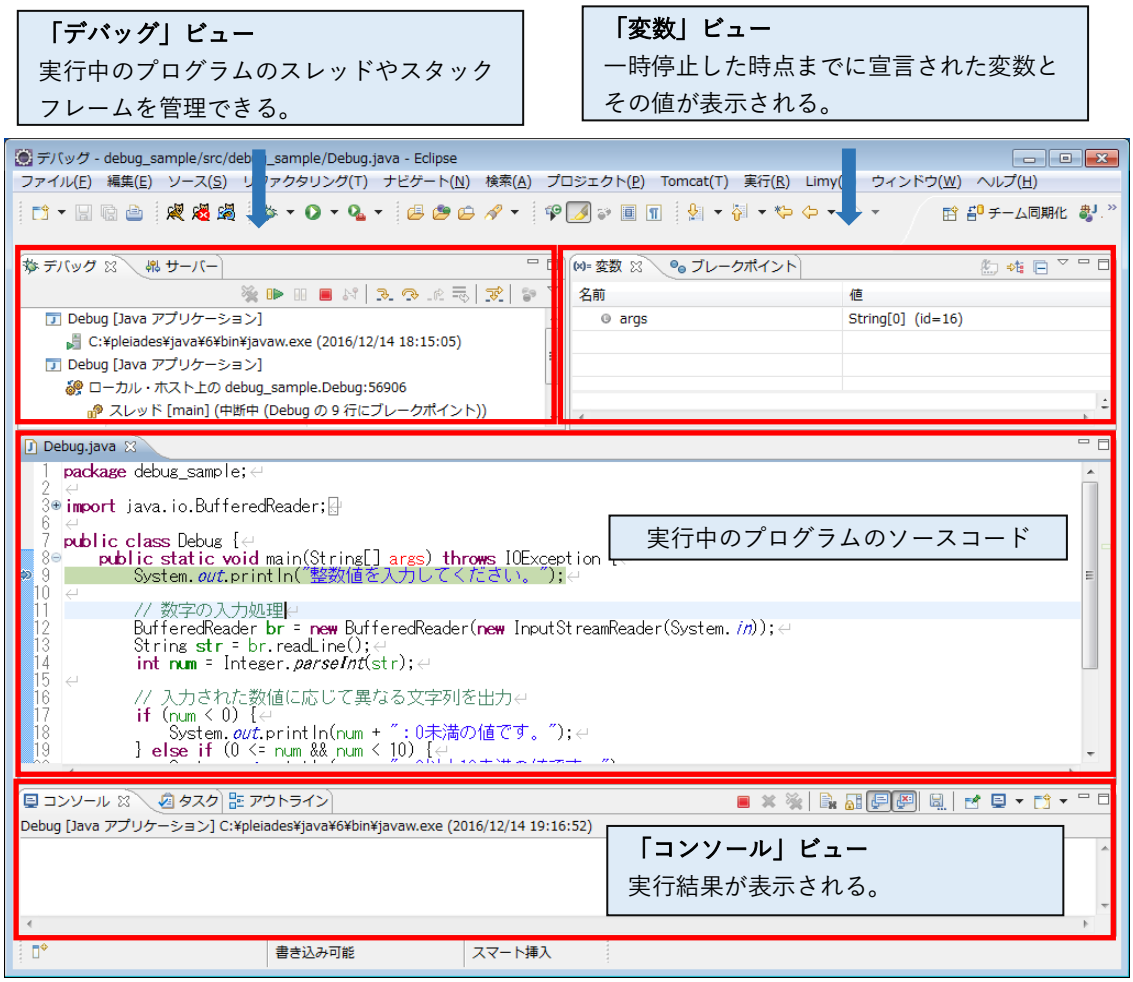


2.2 デバッグ・パースペクティブを表示する

「デバッグ」ボタンを押すと、次のような確認ダイアログが表示されます。このダイアログ中の「常にこの設定を使用する」というメッセージ横のチェックボックスにチェックを入れ、「はい」ボタンを押してください。



「はい」を押すと、Eclipse 上の画面が次の図のように切り替わります。この画面はデバッグ・パースペクティブと呼ばれ、デバッグ用の機能が集まった画面になっています。この画面中には赤枠で示したようなブロック(ビュー)などが表示されています。

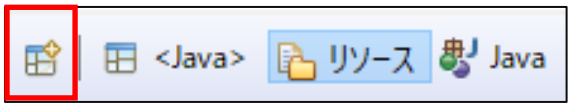


※ プログラミングの分野では、プログラム内のある処理の開始～終了までを一単位とし、それをスレッドと呼んでいます。

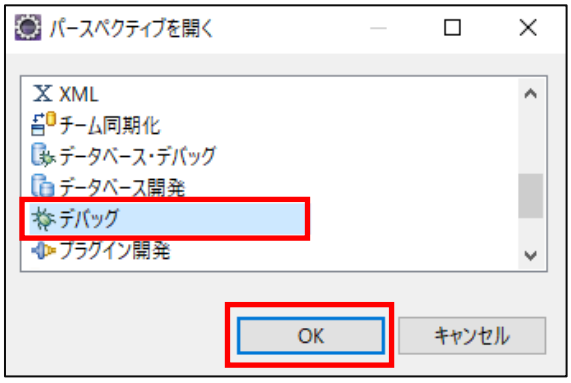
※ 上図と同じビューが表示されていない場合は、次の方法を試してください。

● デバッグ・パースペクティブの表示
「ウィンドウ」→「パースペクティブ」→「パースペクティブを開く」→「その他」→「デバッグ」の順で「デバッグ・パースペクティブ」を表示してください。

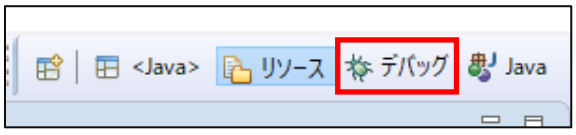
● パースペクティブアイコンの作成
① 図の赤枠で囲っているアイコン部分をクリックします。



② 「デバッグ」を選択して OK を押します。



③ デバッグアイコンが表示されます。

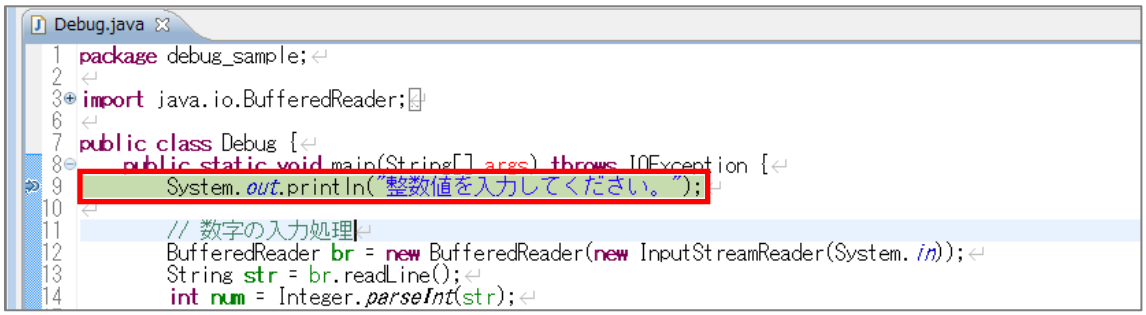


Eclipse 上の「実行」ボタンを押してプログラムを実行した場合、プログラム中の最後の処理まで自動的に処理が進みます。しかし、ブレークポイントを置いてから「デバッグ」ボタンを押した場合、プログラムは実行開始されますがブレークポイントの箇所で処理が一時停止します。

3. 一文ずつ処理を実行する

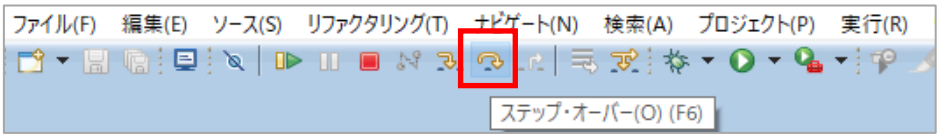
3.1 一時停止している処理を確認する

デバッグ・パースペクティブ中のソースコードを見てください。ソースコード中の一行のみ背景色が薄い緑色になっています。この箇所は、その処理を実行する直前でプログラムが一時停止していることを表しています。

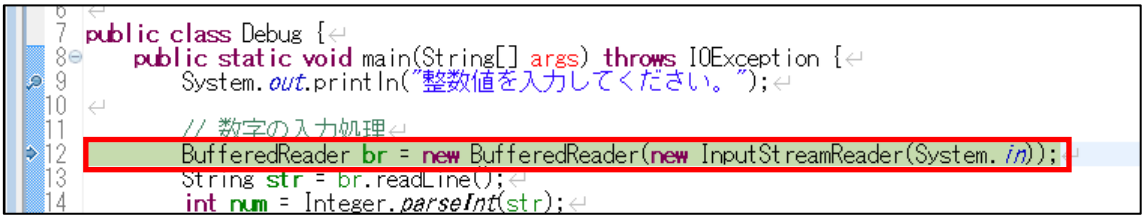


3.2 「ステップ・オーバー」機能で一文の処理を実行する

それでは、この状態から一時停止した一文の処理を実行しましょう。一文ずつ処理を手動で実行する際には「ヘッダーメニュー」下のツールバーにある「ステップ・オーバー」ボタンを押します(キーボードの[F6 キー]に対応しています)。



「ステップ・オーバー」ボタンを押したら、まずはソースコードを見てください。背景色が緑色になっている箇所が次の処理が記述された行に移りました。このことは「先ほど一時停止していた処理を実行してその後に次の処理を実行する直前でプログラムを一時停止している」ことを表しています。このように「ステップ・オーバー」ボタンを押すことで、一文ずつ処理を手動で実行することができます。



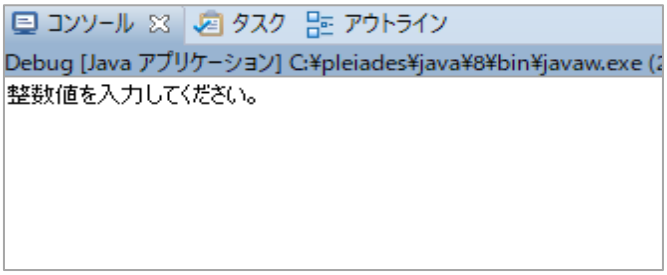
※ コメント行は処理ではないため一時停止の対象外となります。

4. 実行結果を確認する

処理を進めた後には以下のように実行結果を確認できます。

4.1 コンソール上の出力結果を確認する

次に「コンソール」ビューを見てください。最初に一時停止させた処理の結果がコンソール上に標準出力されています。このように、標準出力処理を手動で実行した場合には、その行で指定された文字列のみがコンソール上に出力されます。



※ ソースコード中の「br.readLine()」のように、ユーザによる入力操作が行われるまで処理待ちする文を手動で実行した場合、入力操作が完了するまで次の処理に進みません。もし、「ステップ・オーバー」ボタンを押しても処理が進まない場合は、現在実行しようとしている処理が入力操作を促すものであるか確認しましょう。

4.2 変数の値を確認する

「ステップ・オーバー」ボタンを押して、以下の処理を実行しましょう。この処理を実行する際にはコンソール上で適当な整数値を入力してください。

```
String str = br.readLine();
```

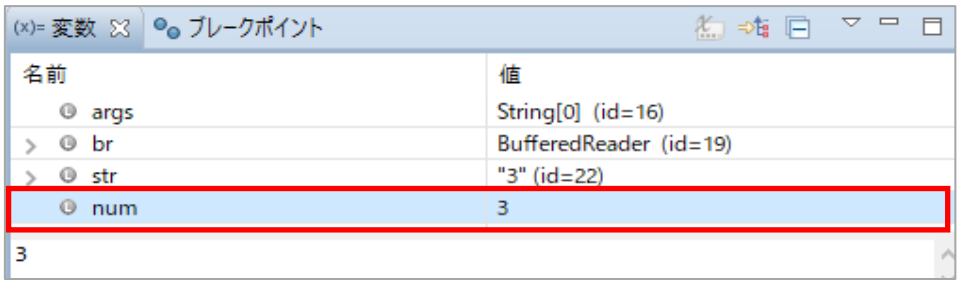
次に「ステップ・オーバー」ボタンを押して、下記処理を実行しましょう。

```
int num = Integer.parseInt(str);
```

上記の処理まで実行すると下記の処理の背景色が緑色になります。

```
if (num < 0) {
```

このとき、「変数」ビューに注目してください。「変数」ビューには、一時停止した処理の直前までに宣言された変数とその値が一覧表示されます。今回の場合、次のページの図のように変数 num に入力した整数値が代入されていることが分かります。



5. 処理の流れを確認する

「ステップ・オーバー」ボタンを押して、下記の処理を実行します。

```
if(num < 0){
```

すると次はどの行の処理で一時停止したでしょうか。周りの方と比較してみると、人によって一時停止した箇所が異なることが分かります。たとえば、「3」と入力した場合には次の処理で一時停止します。

```
}else if(0 <= num && num < 10){
```

これは if 文の条件式「num < 0」の値が false となり、次の else-if 文の条件式での処理に移ったためです。このように、デバッグモードを利用することで処理の流れを確認することができます。一連の確認作業を行った際に想定通りの処理結果にならなかった箇所があった場合、そのソースコード中に誤った処理（バグ）が含まれていると考えられます。

6. デバッグモードを終了する

前項の確認作業が一通り済んだらデバッグモードを終了させましょう。デバッグモードを終了させるには「ヘッダーメニュー」下のツールバーにある「再開」ボタン、もしくは「終了」ボタンを押します。

「再開」ボタンを押した場合、最後の処理まで自動的に進み、プログラムが終了します。

また、ソースコード中に複数箇所ブレークポイントを表示させた場合、次のブレークポイントの処理まで自動的に処理が進みます。また、「終了」ボタンを押した場合、プログラム全体の処理が強制的に中断されます。

7. バグを修正する

バグを発見したら、原因の箇所を適宜修正します。また、修正後は改めてデバッグモードを起動してプログラムを実行し、問題の処理が想定通りの結果となることも確認しましょう。デバッグモードの操作方法は以上となります。

次の表は、Eclipse のデバッグ機能のショートカットキーになります。●は覚えておくと、特に便利なものです。

【表 10-1 デバッグショートカットキー】

名前	ショートカット	効果
●ステップ・イン	F5	他のメソッドを呼び出す場合には呼び出し先のメソッドもデバッグします。また、呼び出し先のメソッドの処理も一行ずつ行います。
●ステップ・オーバー	F6	他のメソッドを呼び出す場合には呼び出し先のメソッドのデバッグをスキップします。つまり、呼び出し先のメソッドの処理は中を一行ずつ追うのではなく一括で瞬時に行います。
ステップ・リターン	F7	今のメソッドを最後まで実行して呼び出し元へ戻ります。
●再開	F8	次のブレークポイントまで処理を進めます。 次のブレークポイントがない場合は処理を終了させます。
●終了	Ctrl + F2	処理を中断してアプリケーションを終了させます。
デバッグ実行	F11	選択した変数の値をその場で確認できます (変数ビューでは値を変更できます)。
表示	Ctrl + Shift + D	選択した命令の結果を表示します。 (表示ビューでも可能)
ブレークポイントの設定/解除	Ctrl + Shift + B	デバッグしたいコードをブレークポイントに設定・解除できます。
Run to Line	Ctrl + R	カーソル行まで実行します。
次の警告 or エラーへジャンプ	Ctrl + .（ドット）	警告やエラーになっているコードを一つずつ見ることができます。
修正のヒント	Ctrl + 1	警告やエラーになっているコードで使うと、修正のヒントが表示されます。

章のまとめ

本章では、以下のことを学びました。

- ❑ バグの原因を調査・修正するのが「デバッグ」です。
- ❑ デバッグモードにより、特定の処理(文)が実行される手前でプログラムを一時停止できます。
- ❑ デバッグモードを使うことで処理を 1 行ずつ確認することができます。
- ❑ ブレークポイントは処理の流れを想定し、バグが発生しそうな処理に指定します。

練習問題

1. 以下のサンプルコードは、5 以下の整数を入力した場合に「5 以下の値です。」、6～9 を入力した場合に「6～9 までの値です。」、10 以上を入力した場合に「10 以上の値です。」と出力するプログラムです。しかし、想定通りの結果が出力されないため、デバッグモードを用いて修正してください。

   ■ 作成するファイル
   /java_practice/src/lesson10/Practice1001.java

Practice1001.java

```
package lesson10;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Practice1001 {
    public static void main(String[] args) throws IOException {
        System.out.print("整数値を入力してください>");

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String str = reader.readLine();
        int num = Integer.parseInt(str);

        // 入力された数値に応じて異なる文字列を出力
        if (num <= 5) {
            System.out.println("5以下の値です。");
        } else if (6 <= num && num < 9) {
            System.out.println("6～9までの値です。");
        } else if (10 <= num) {
            System.out.println("10以上の値です。");
        }

        System.out.println("以上で処理は終了です。");
    }
}
```

2. 3 人の生徒が 100 点満点のテストを受けました。以下のサンプルコードは、3 人のテストの点数をキーボードで入力し、3 人の点数と最高点と最低点を出力するプログラムです。しかし、想定通りの結果が出力されないため、デバッグモードを用いて修正してください。

■ 作成するファイル
/java_practice/src/lesson10/Practice1002.java

Practice1002.java

```
package lesson10;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Practice1002 {
    public static void main(String[] args) throws IOException {
        System.out.println("3人のテストの点数を入力してください。");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        int[] test = new int[3];

        for (int i = 0; i < test.length; i++) {
            String str = reader.readLine();
            int score = Integer.parseInt(str);
            test[i] = score;
        }

        int max = 100;
        for (int i = 0; i < test.length; i++) {
            if (test[i] >= max) {
                max = test[i];
            }
        }

        int min = 0;
        for (int i = 0; i < test.length; i++) {
            if (test[i] <= min) {
                min = test[i];
            }
        }

        for (int i = 0; i < test.length; i++) {
            System.out.println(i + "番目の人の点数は" + test[i] + "です。");
        }

        System.out.println("最高点は" + max + "点です。");
        System.out.println("最低点は" + min + "点です。");
    }
}
```

【実行結果】

```
3人のテストの点数を入力してください。
20 ↲
30 ↲
40 ↲
1番目の人の点数は20です。
2番目の人の点数は30です。
3番目の人の点数は40です。
最高点は40点です。
最低点は 20 点です。
```

✓ 解答

1. まずはプログラムを実行してみます。

【実行結果】

```
整数値を入力してください>9
以上で処理は終了です。
```

これは問題のある実行結果です。「9」を入力したときは「6～9 までの値です。」が出力された後に「以上で処理は終了です。」と出力されます。

ブレークポイント箇所（例）
● int num = Integer.parseInt(str);

まずはこの部分にブレークポイントを付けます。入力された値と条件に代入された値を細かく確認します。

【訂正箇所】

```
} else if (6 <= num && num < 9) {
```

この条件は「6 以上かつ 9 未満」となるので 9 を入力したときに条件に当てはまらず、「System.out.println("以上で処理は終了です。");」の処理に移ります。

【解答】

```
} else if (6 <= num && num <= 9) {
```

比較演算子の「<」を「<=」に訂正します。そうすることにより条件は「6 以上かつ 9 以下」となり 9 を入力したときに条件に当てはまり、「System.out.println("6～9 までの値です。");」の処理に移ります。

2.　まずはプログラムを実行してみます。

【実行結果】

```
3人のテストの点数を入力してください。
56 ↲
78 ↲
90 ↲
0 番目の人の点数は 56 です。
1 番目の人の点数は 78 です。
2 番目の人の点数は 90 です。
最高点 100 点です。
最低点は 0 点です。
```

この実行結果から問題点が 3 つあることが分かります。

① 入力した点数に最高点が反映されていない
② 入力した点数に最低点が反映されていない
③ 「n 番目」の始まりが 0 番目から始まっている

ブレークポイント箇所(例)
● test[i] = score;

【訂正箇所 1】

```
//最高点をチェックする条件分岐
int max = 100;                                  ──→ 問題点①
for (int i = 0; i < test.length; i++) {
    if (test[i] >= max) {
        max = test[i];
    }
}

//最低点をチェックする条件分岐                    ──→ 問題点②
int min = 0;
for (int i = 0; i < test.length; i++) {
    if (test[i] <= min) {
        min = test[i];
    }
}
```

① 「max」は最高点の変数名、「min」は最低点の変数名です。修正前の最高点の条件は「もし、test[i]（配列 test に格納された i 番目点数）が max（＝100）以上であれば max に test[i]を代入します」という条件です。この条件では 100 点を超える点数でなければ条件に当てはまりません。

【訂正】

```
int max = 0;
```

max に 0 を代入することで、入力された点数の中から一番大きい数値が max に代入されます。

② 「min」は最低点の変数名です。修正前の最低点の条件は「もし、test[i]（配列 test に格納された i 番目の点数）が min（＝0）以下であれば min に test[i]を代入します」という条件です。この条件では 0 点を下回る点数でなければ条件に当てはまりません。

【訂正】

```
int min = 100;
```

min に 100 を代入することで、配列 test に格納された点数の中で 1 番小さい値が min に代入されます。

【訂正箇所 2】

```
for (int i = 0; i < test.length; i++) {
    System.out.println(i + "番目の人の点数は" + test[i] + "です。");
}
```

問題点③

③ 「i」は配列の要素番号を表しています。配列は 0 から始まるので「i」だけでは不適切です。要素数に合わせるために「i＋1」と入力します。

```
for (int i = 0; i < test.length; i++) {
    System.out.println((i + 1) + "番目の人の点数は" + test[i] + "です。");
}
```

第 11 章 確認問題

確認問題（基礎編）

1. 次の文の空欄を埋めてください。

　数値などのデータを自由に出し入れできるのが、［　①　］です。①は、使用する際に［　②　］と名前を記述し、宣言します。①から値を取り出すためには、事前に値を入れておかなければなりません。①に値を入れる処理を［　③　］といいます。また、①の宣言と同時に③することを、［　④　］といいます。

2. 次の文の空欄を埋めてください。

　整数を表すための主要な整数型として、［　①　］、［　②　］、［　③　］、［　④　］があります。また、実数を表すための浮動小数点型には［　⑤　］、［　⑥　］があります。これらの型と文字型・論理型の総称を［　⑦　］といいます。

3. 次の一覧表の空欄を埋めてください。

論理積

x	y	x && y
false	false	①
false	true	②
true	false	③
true	true	④

論理和

x	y	x || y
false	false	⑤
false	true	⑥
true	false	⑦
true	true	⑧

4. 配列の記述方法として正しいものはどれですか。3 つ選択してください。

① int a[] = new int(2);
② int[] a = new int[3];
③ int[] a;
　a = new int[7];
④ int a[] = new int[];
⑤ int[] a = {2,4,6};

5. 以下に示すのは、変数 i の値を 3 で割った剰余に応じて、「3 で割り切れます。」「3 で割った剰余は 1 です。」「3 で割った剰余は 2 です。」のいずれかを表示するプログラムです。①～⑦の空欄を埋めてプログラムを完成させてください。

```
if (①){
    System.out.println("3 で割り切れます。");
}else if (②) {
    System.out.println("3 で割った剰余は 1 です。");
}else if (③) {
    System.out.println("3 で割った剰余は 2 です。");
}
```

```
Switch (④) {
    case ⑤:
        System.out.println("3 で割り切れます。");
        break;
    case ⑥:
        System.out.println("3 で割った剰余は 1 です。");
        break;
    case ⑦:
        System.out.println("3 で割った剰余は 2 です。");
        break;
}
```

6. ユーザに値をコンソール入力させ、実行結果のように出力させるプログラムを作成してください。

■ 作成するファイル
java_practice/src/lesson11/Practice1106.java

【実行結果】※○○は入力値です

```
入力された値は「○○」です。
```

7. 6 で入力させた値を整数値に変換するプログラムを作成してください。

■ 変更するファイル
/java_practice/src/lesson11/Practice1107.java

8. 次の式を評価した値を出力するプログラムを作成してください。

- 10 * 3
- 20 / 4
- 15 % 6

■ 作成するファイル
/java_practice/src/lesson11/Practice1108.java

確認問題（応用編）

9. for 文を用いて 5 ～20 までの値を表示してください。

■ 作成するファイル
/java_practice/src/lesson11/Practice1109.java

10. while 文を用いて 5 ～20 までの値を表示してください。

■ 作成するファイル
/java_practice/src/lesson11/Practice1110.java

11. 9 の内容を編集し、10 の値で強制的に for 文が中断するようにしてください。

■ 変更するファイル
/java_practice/src/lesson11/Practice1111.java

12. 11 の内容を編集し、8 の値をスキップするようにしてください。

■ 変更するファイル
/java_practice/src/lesson11/Practice1112.java

13. boolean 型の配列変数を用意し、要素を 3 個扱えるようにしてください。

■ 作成するファイル
/java_practice/src/lesson11/Practice1113.java

14. ユーザが入力した任意の整数値 3 つを int 型の配列に代入し、その後昇順にソートして各値を表示するプログラムを作成してください。

■ 作成するファイル
/java_practice/src/lesson11/Practice1114.java

15. boolean 型の変数 female にあらかじめ true を代入、また、int 型の変数 age にあらかじめ 25 を代入しておき、female が true かつ age が 20 以上であれば、「成人女性です。」と表示するプログラムを作成してください。

■ 作成するファイル
/java_practice/src/lesson11/Practice1115.java

✓ 解答

基礎編

1. ①変数　②型　③代入　④初期化

2. ①byte　②short　③int　④long　⑤double　⑥float　⑦基本型

3. ①false　②false　③false　④true　⑤false　⑥true　⑦true　⑧true

4. ②、③、⑤
   選択肢①は、要素を[]ではなく()で指定しているため誤りです。
   選択肢②は、配列の宣言、要素の確保を行っているため、正しいです。
   選択肢③は、2 行にわたって配列の宣言と要素の確保を行っています。
   選択肢④は、new キーワードの後に要素数を指定していないため誤りです。
   選択肢⑤は、配列の宣言、要素の確保、値の代入をまとめて記述した配列の初期化です。

5. ① i % 3 == 0　② i % 3 == 1　③ i % 3 == 2　④ i % 3　⑤ 0　⑥ 1　⑦ 2

6.
Practice1106.java

```
package lesson11;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Practice1106 {
    public static void main(String[] args) throws IOException {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();

        System.out.println("入力された値は「" + str + "」です。");
    }
}
```

7.
Practice1107.java

```
package lesson11;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Practice1107 {
    public static void main(String[] args) throws IOException
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        String str = reader.readLine();
        int num = Integer.parseInt(str);

        System.out.println("入力された値は「" + num + "」です。");
    }
}
```

8.
Practice1108.java

```
package lesson11;

public class Practice1108 {
    public static void main(String[] args) {
        System.out.println(10 * 3);
        System.out.println(20 / 4);
        System.out.println(15 % 6);
    }
}
```

【実行結果】

```
30
5
3
```

応用編

9.

Practice1109.java

```
package lesson11;

public class Practice1109 {
    public static void main(String[] args) {
        //i が 20 になるまで出力する
        for(int i = 5; i < 21; i++){
            System.out.println(i);
        }
    }
}
```

10.

Practice1110.java

```
package lesson11;

public class Practice1110 {
    public static void main(String[] args) {
        //変数 l に 5 を代入する
        int l = 5;

        //変数 l の値が 20 になるまで出力する
        while(l < 21){
            System.out.println(l);
            l++;
        }
    }
}
```

11.
Practice1111.java

```
package lesson11;

public class Practice1111 {
    public static void main(String[] args) {
        for(int i = 5; i < 21; i++){
            System.out.println(i);
            //変数 i の値が 10 になったとき強制終了する
            if(i == 10){
                break;
            }
        }
    }
}
```

実行中の繰り返し処理を強制終了したいときは break 文を使います。7～9 行目の if 文の条件判定で変数 i の値が 10 の場合、break 文が実行され、for 文から抜けます。

12.
Practice1112.java

```
package lesson11;

public class Practice1112 {
    public static void main(String[] args) {
        for(int i = 5; i < 21; i++){
            //変数 i の値が 8 になったときスキップする
            if(i == 8){
                continue;
            }
            System.out.println(i);
            //変数 i の値が 10 になったとき強制終了する
            if(i == 10){
                break;
            }
        }
    }
}
```

実行中の繰り返し処理を強制終了するのではなく、ブロック内の残りの処理をスキップして条件式に戻り、さらに繰り返し処理を続けたいときは continue 文を使います。6～8 行目の if 文の条件判定で変数 i の値が 8 の場合、continue 文が実行され、for 文の条件式に戻ります。

13.
Practice1113.java

```
package lesson11;

public class Practice1113 {
    public static void main(String[] args) {
        boolean[] bls = new boolean[3];
    }
}
```

14.
Practice1114.java

```
package lesson11;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Arrays;

public class Practice1114 {
    public static void main(String[] args) throws IOException {
        int[] numbers = new int[3];

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        for (int i = 0; i < 3; i++) {
            String str = reader.readLine();
            int num = Integer.parseInt(str);
            numbers[i] = num;
        }

        // sort()メソッドを使って値を昇順にソートする
        Arrays.sort(numbers);

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

15.
Practice1115.java

```
package lesson11;

public class Practice1115 {
    public static void main(String[] args) {
        boolean female = true;
        int age = 25;

        if(female && 20 <= age) {
            //変数 female が true かつ年齢が 20 以上の場合、「成人女性です。」と表示する
            System.out.println("成人女性です。");
        }
    }
}
```

索引

記号

0

B

C

D

F

G

I

J

L

M

P

R

S

T

U

W

い

え

お

か

き

く

こ

さ

し

す

せ

そ

た

て

と

ね

は

ひ

ふ

へ

み

む

め

も

よ

ら

り

る

ろ

参考文献

・ 中山清喬、国本大悟（2017）『スッキリわかる Java 入門 第 2 版 スッキリわかるシリーズ』インプレス
・ 高橋麻奈（2019）『やさしい Java 第 7 版』SB クリエイティブ
・ 川場隆（2013）『わかりやすい Java オブジェクト指向編』秀和システム
・ ジョゼフ・オニール（2011）『独習 Java 第 4 版』翔泳社
・ 志賀 澄人（2019）『徹底攻略 Java SE 8 Silver 問題集』インプレス
・ 安藤 正芳,”プログラミング言語利用実態調査 2021”,日経 XTECH,(2021-08-04)
https://xtech.nikkei.com/atcl/nxt/mag/nc/18/072100242/072100001/,(参照 2022-01-05).

東京ITスクール
TOKYO IT SCHOOL

Java　第 5 版

2019 年 12 月 初　版　発行
2020 年 2 月　第 2 版　発行
2021 年 2 月　第 3 版　発行
2022 年 2 月　第 4 版　発行
2022 年 12 月 第 5 版　発行

著者　　株式会社 SystemShared　東京 IT スクール