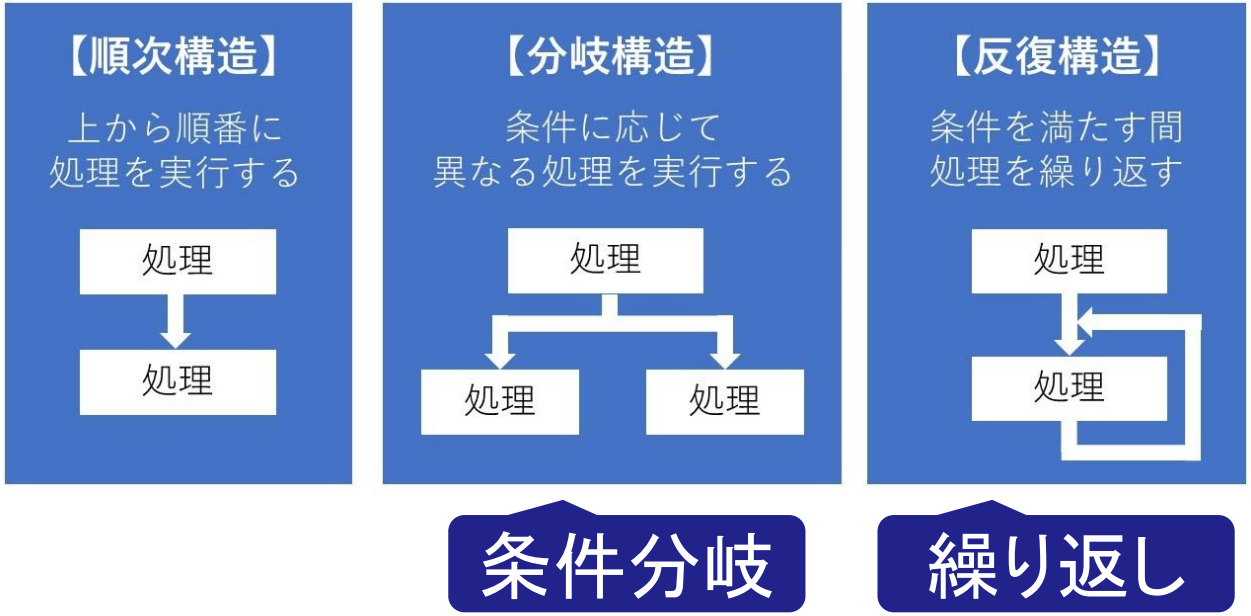
第8章　条件分岐

目次

制御構造と制御文

プログラムの処理の流れを「制御構造」といい、
「順次構造」「分岐構造」「反復構造」の3種類がある。
分岐構造や反復構造を実現するための構文を「制御文」と言う。



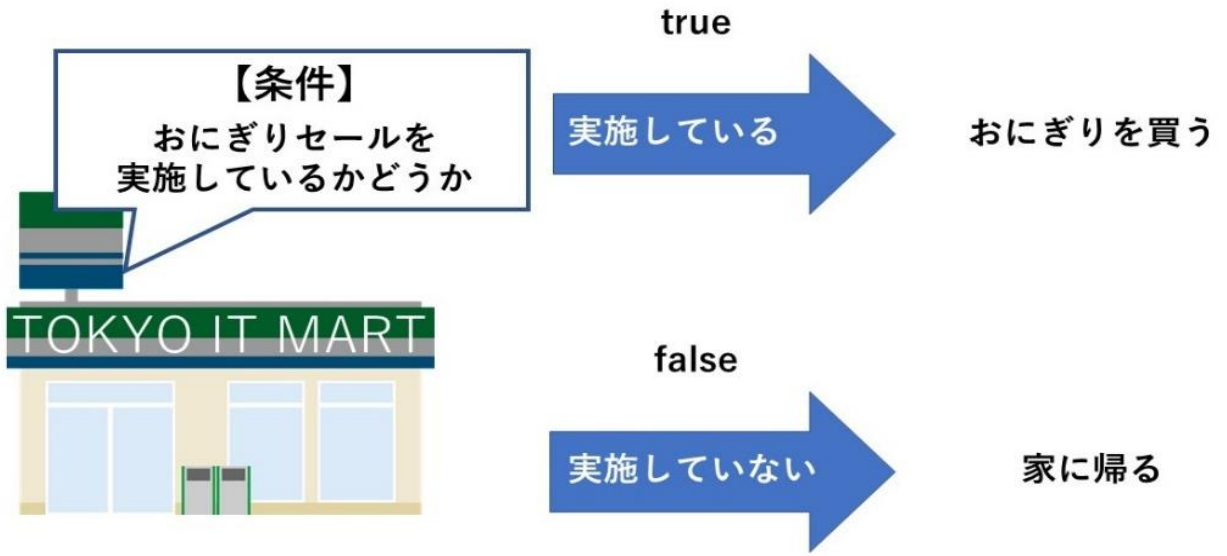
条件とは

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。

条件：おにぎりセールを実施しているかどうか

条件とは

Java条件は式として扱われ評価される。
その条件を満たす場合は「true（真）」、
満たさない場合は「false(偽)」を評価値として返す。



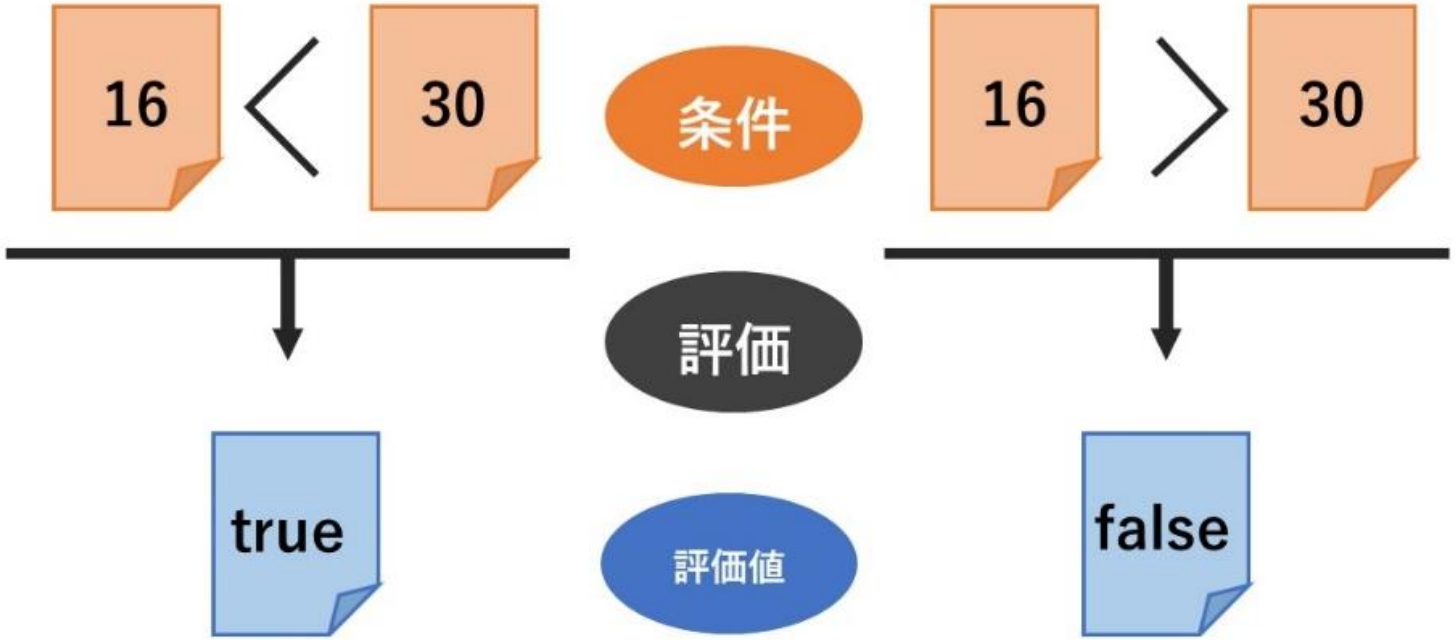
関係演算子とは

値を比較する演算子のこと。

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

関係演算子とは

評価値（条件を満たすかどうか）として
「true」または「false」を返す。



【Sample0801 関係演算子を使う】を作成しましょう

Sample0801のポイント①

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。
関係演算子の評価値としてtrue、falseが出力される。

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

条件は、関係演算子を使って記述します。

if文とは

条件を満たした場合に特定の処理を実行する文。

```
if (条件) {
    処理;
}
```

( )内の条件を満たしている場合（true）:
→{ }ブロック内の処理を実行する。

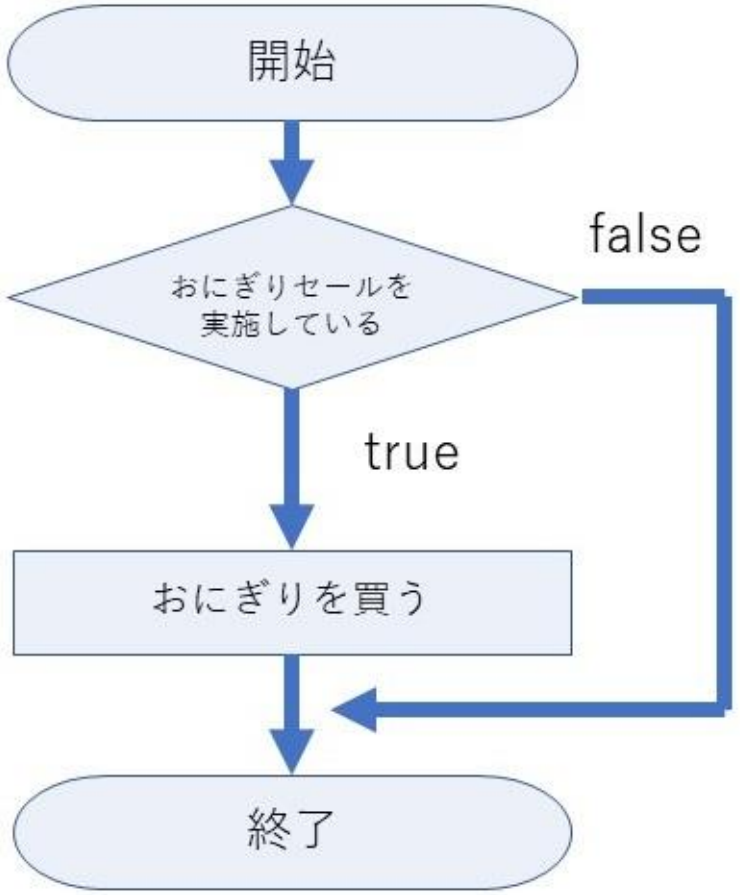
( )内の条件を満たさない場合（false）:
→{ }ブロック内の処理は実行されず、
ブロックの次に記述された処理に移る。

おにぎりセールの例をif文で表現した場合

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 (おにぎりセールを実施している){
    おにぎりを買う;
}
```

おにぎりセールの例をフローチャートで表現した場合



【Sample0802 if文を使う】を作成しましょう

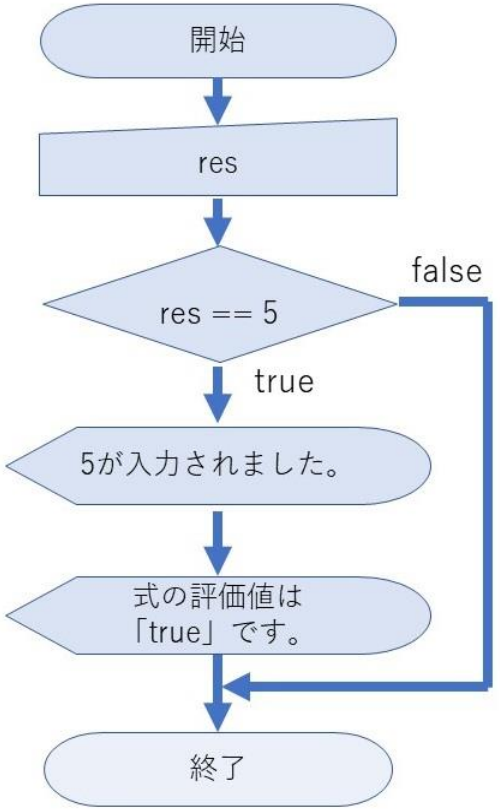
Sample0802のポイント①

5を入力した場合：
→if文の条件の評価値は「true」
→if文のブロック内の処理が実行される。

5以外の整数を入力した場合：
→if文の条件の評価値は「false」
→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る。

Sample0802のポイント②

フローチャート



if文の注意点

変数のスコープに注意。
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る。

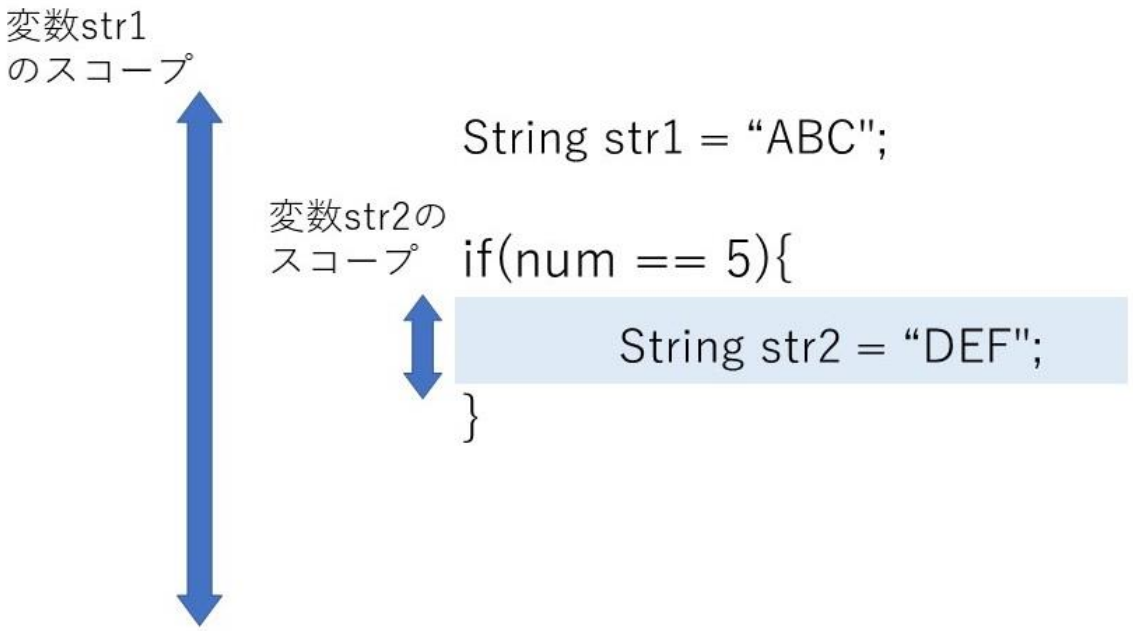
スコープ（scope）:
変数が利用可能な範囲（有効範囲）のこと。

if文の注意点

```
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);
}
```

コンパイルエラー：
str2のスコープ外

if文の注意点



ブロック内は、インデントを使って読みやすくしましょう。
「 { } 」をつけて if 文の構文がどこなのか分かりやすくしましょう。
変数の宣言を行う際はスコープに注意しましょう。

if文のネスト

if文の中にさらにif文を記述することも可能。
処理を入れ子の形にすることを「ネスト」と言う。

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

【Sample0803 if文をネストする】を作成しましょう

Sample0803のポイント①

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

```
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。
```

Sample0803のポイント②

2を入力すると、①の条件は「false」となり、
①のブロックの処理は実行されない。

```
整数を入力してください。
2 ↲
システムを終了します。
```

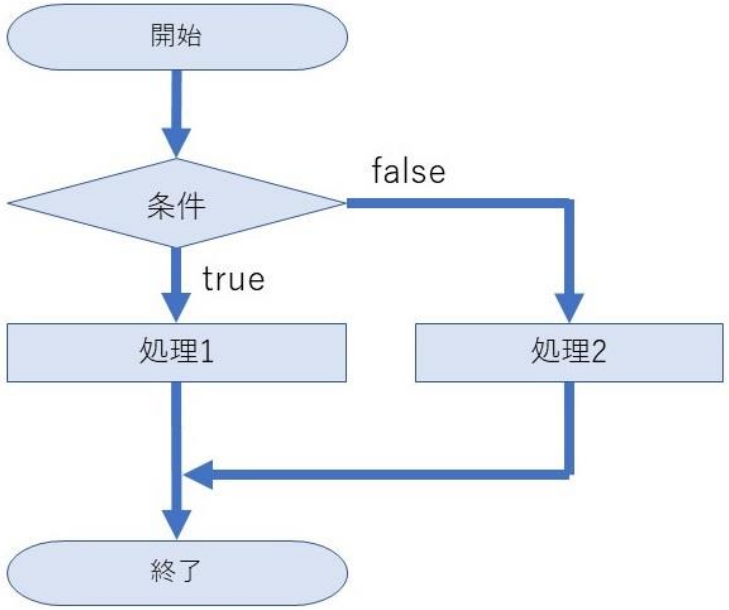
if~else文

条件を満たさない場合に特別な処理をしたい場合、
「if~else文」という構文を利用する。

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

if~else文

条件を満たす場合は、if文以下のブロックの処理を実行する。
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する。

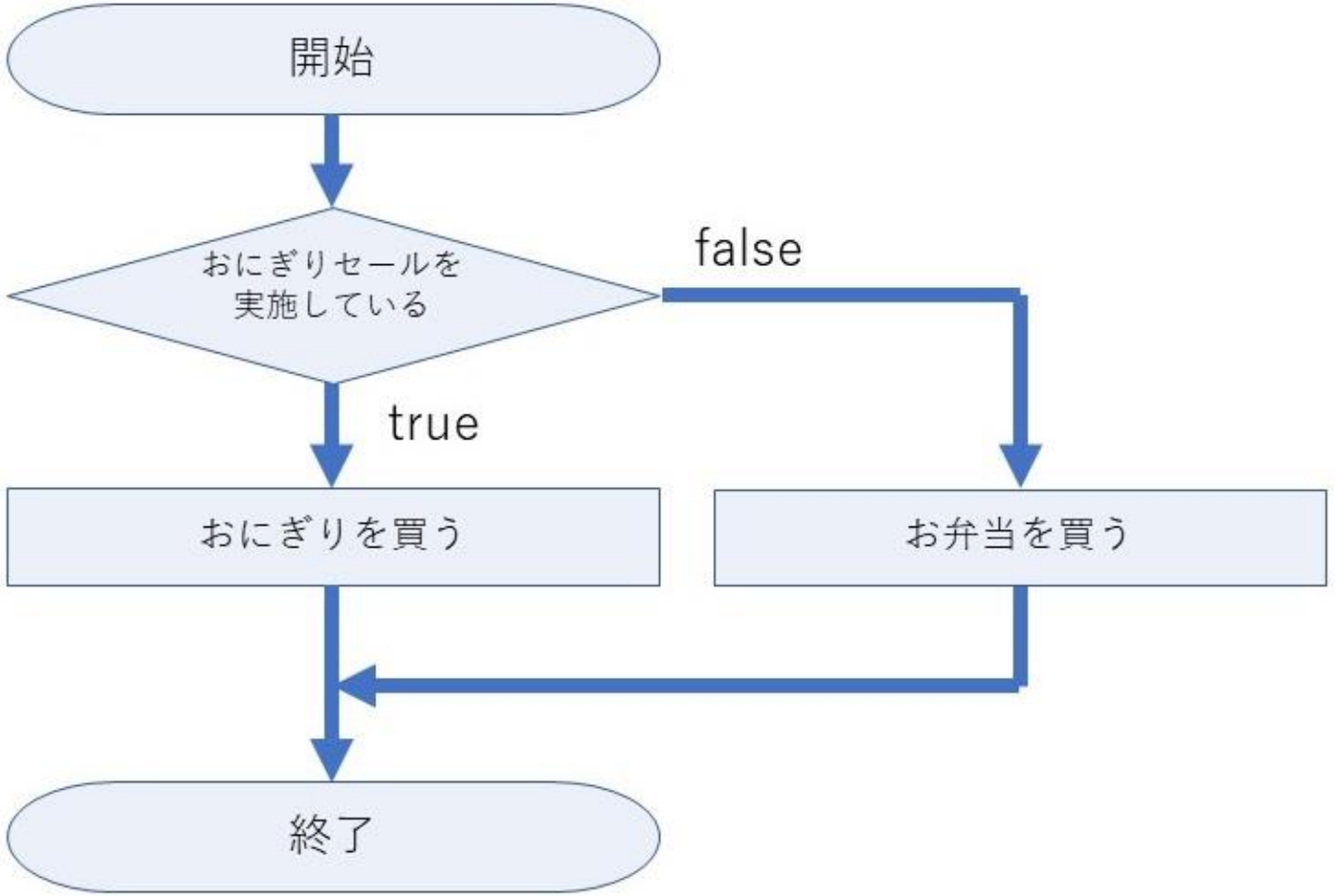


if~else文

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。


```
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;
}
```

if~else文



【Sample0804 if~else文を使う】を作成しましょう

Sample0804のポイント

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。
```

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。
```

if～else if～else文とは

複数の条件ごと異なる処理を実行させたい場合は
「if~else if~else文」を使用する。

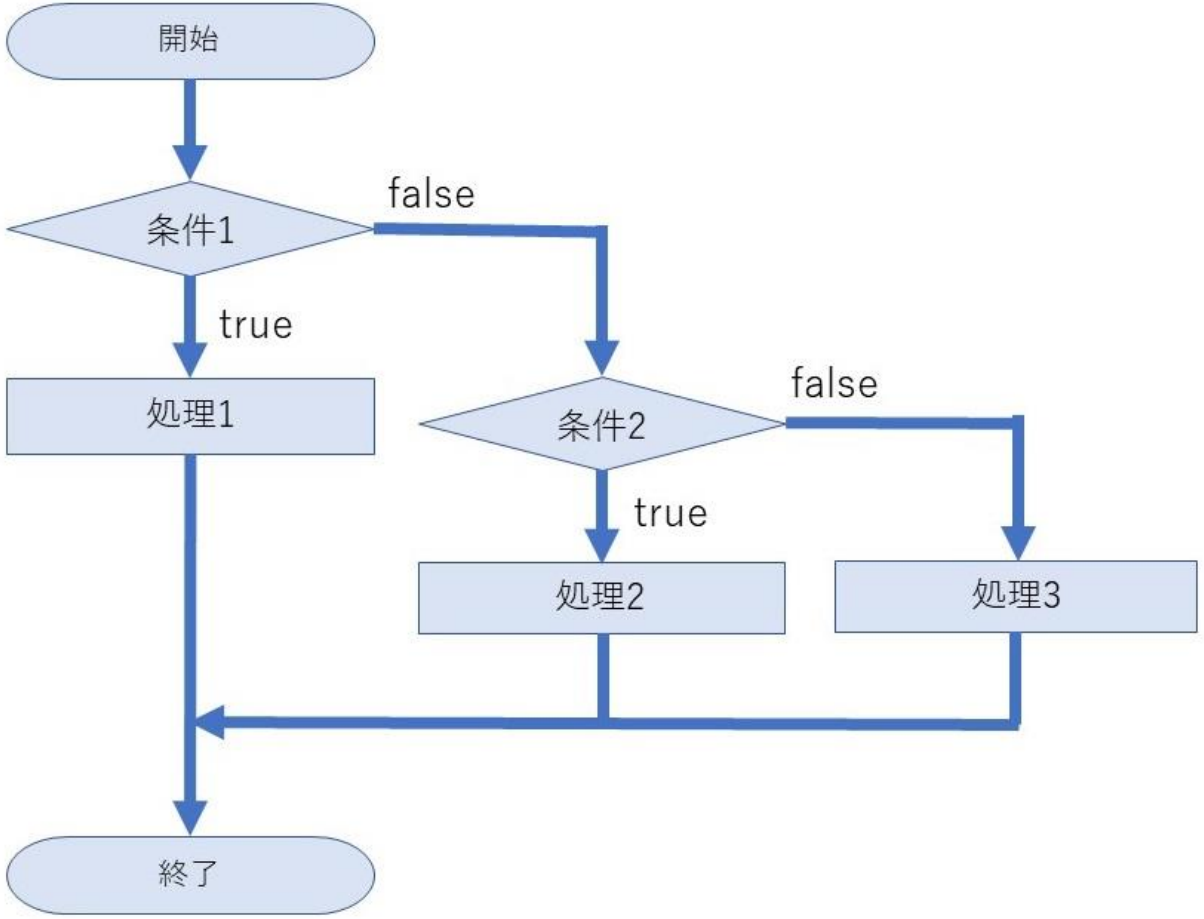
```
if(条件1){
    処理1;
} else if(条件2){
    処理2;
} else {
    処理3;
}
```

if～else if～else文とは

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。


```
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;
}
```

if～else if～else文とは



【Sample0805 if~else if~else文を使う】を作成しましょう

Sample0805のポイント

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

```
整数を入力してください。
7 ↲
7が入力されました。
```

Sample0805のポイント

4と7以外を入力した場合

```
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。
```

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

switch文は、if文と同じく、条件分岐の構文である。

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

switch文とif文の違い

・比較文法

→switch文は、case文の値と等しいか（または同じ内容か）で条件を判定する。「==」を使った比較しか行えない。

・比較できる型

→浮動小数点型（float、double）や論理型（boolean）での比較は行えない。

・比較できる値 → 比較する値としてnull値を使用できない。

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。

【Sample0806 switch文を使う】を作成しましょう

Sample0806のポイント

4を入力した場合、1つ目のcase文の条件を満たすため、
①の処理が実行される。

```
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の処理が実行される。

```
整数を入力してください。
7 ↲
7が入力されました。
```

Sample0806のポイント

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、③の処理が実行される。

```
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。
```

switch文の注意点

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。
④ default文は省略できる。

【Sample0807 break文の省略】を作成しましょう

Sample0807のポイント

4を入力すると、「case1：」以降の処理も
全て実行されていることが分かる。

```
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。
4と7以外の数字が入力されました。
```

switch文ではbreak文を書き忘れないように気をつけましょう。

論理演算子とは

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利用する。

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。

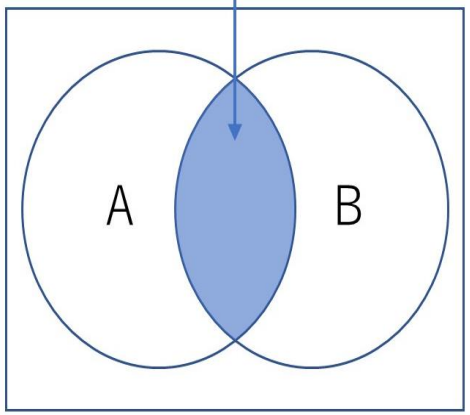
（今日が土曜日である）&&（お金がある）

論理演算子の種類

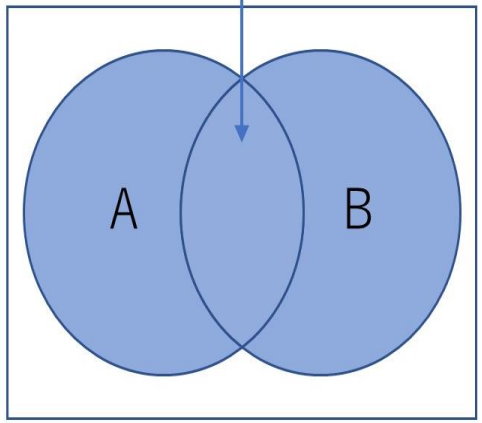
演算子	働き	意味	記述例
&&	かつ （論理積）	左辺と右辺の条件がどちらもtrueの場合、全体の評価はtrue	(a >= 4) && (a < 20)
||	または （論理和）	左辺と右辺の条件のいずれかがtrueの場合、全体の評価はtrue	(a == 3) || (a == 23)
!	～でない （論理否定）	条件がfalseの場合、 全体の評価はtrue	!(a == 28)

論理演算子の種類

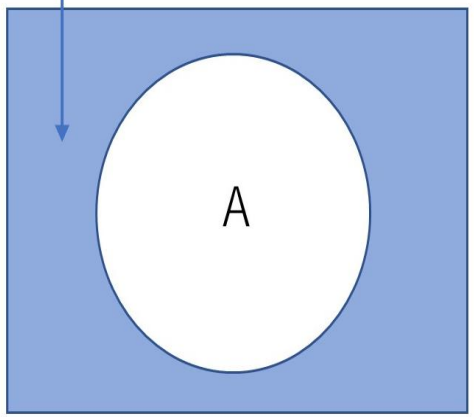
A && B
条件Aと条件Bの両方を満たす



A || B
条件Aと条件Bのどちらかを満たす



!A
条件Aではない



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

論理演算子の記述例

「&&」は、左辺と右辺両方ともに「true」となる場合のみ「true」
→条件①の評価は「false」

「||」は、左辺と右辺のどちらかが「true」となれば、「true」。
→条件②の評価は「true」
→条件③の評価は「true」

「!」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。
→条件④の評価は「true」

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

【Sample0808 論理演算子を使う】を作成しましょう

Sample0808のポイント

4より大きく、かつ10以下の数字を入力した場合、
1つ目の条件を満たすため、①の処理が実行される。

```
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の処理が実行される。

```
整数を入力してください。
2 ↲
2は4以下の数字です。
```

Sample0808のポイント

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の処理が実行される。

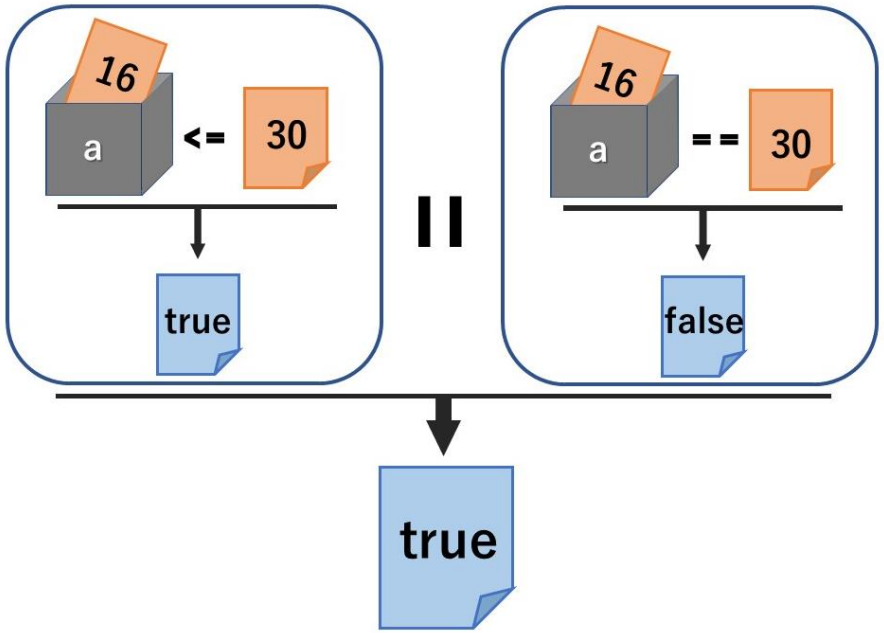
```
整数を入力してください。
20 ↲
20は10より大きい数字です。
```

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が決まる。

「&&」の場合:
左辺が「true」の場合のみ、
右辺の評価を行う。

「||」の場合:
左辺が「false」の場合のみ、
右辺の評価を行う。



論理演算子の評価の仕組み

論理否定の演算子「！」は、boolean型の変数を
オペランドとしている時によく使用される。

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

「flag != true」という条件式と同じ意味となる。
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
簡潔に記述できる。

条件演算子

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ることができる。

```
条件 ? trueのときの式1 : falseのときの式2
```

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。

条件演算子は、3つのオペランドを取る

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」を記述する。

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

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

章のまとめ

- 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、
  いろいろ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。