条件分岐とは

プログラミングでは「ある条件を満たすときだけこの処理を実行したい」という場面が沢山出てくる

そこで使うのが条件分岐というもの。

Javaで条件分岐を使うときには、if文がよく使われる

条件分岐とは、条件によって処理を分けること、ユーザーの操作やシステムの状況に対して、正しく動作するソフトウェアを開発するためには、条件分岐が欠かせない。

例えば、条件分岐は身近なサービスにも使われていて、Googleのログイン画面では以下のような条件分岐がされている

* パスワードが合っている場合→アカウント画面を表示
* パスワードが間違っている場合→エラーメッセージを表示

ユーザーによってはパスワードの入力を間違えてしまうケースもある、そのような場合に間違えてログイン成功時のアカウント画面を見せたらまずい事になる

しかし、上記のように条件分岐を行えば、パスワードが合っていた時にだけログイン可能となる、ユーザーの操作内容に応じて、ソフトウェアの動作を切り替えられる。

Javaで条件分岐させる時の書き方として、主に２種類ある

・if文

・switch文

if文は自由度が高く、基本的にどんな条件分岐でも実装可能、状況によってはswich文を使った方がスマートな形で条件分岐を行える

if文の書き方

条件分岐の中でもポピュラーなif文は、大半のプログラミング言語で使えます。Javaでif文を使うときの最もシンプルな書き方として、まず以下を押さえましょう。

if( 条件式 ) {

条件式がtrue（真）のときの処理

}

上記のif文は「もし条件式が成り立つならば、{ }内の処理を行う」という意味

条件式とは、処理を行う条件を満たしているか判定するための式のこと。

条件式は、条件が成り立てばtrue（真）、成り立たなければfalse（偽）という結果を導き出します。true・falseは、[4章](https://terakoya.sejuku.net/programs/128/chapters/1717)で学んだ論理型（boolean型）のデータです。

if文は、条件式がtrueのときだけ{ }内の処理を行い、falseなら処理を行いません。

if文の基本的な処理

例として、以下のプログラムをみてください。これは、年齢が18歳未満のときは「未成年です」、18歳以上のときは「成人です」と表示するプログラムです

int age = 17; // 年齢

//【条件式1】年齢が18未満かどうか

if( age < 18 ) {

// 年齢が18歳未満の場合

System.**out**.println("未成年です");

}

//【条件式2】年齢が18歳以上かどうか

if( 18 <= age ) {

// 年齢が18歳以上の場合

System.out.println("成人です");

}

上記では、2つのif文で条件分岐しています。それぞれの条件式は以下のとおりです。

* 【条件式1】変数age（年齢）が18未満かどうか
* 【条件式2】変数age（年齢）が18以上かどうか

ageは最初に17としているため、成り立つのは条件式1のみです。そのため、条件式1の処理だけが実行され、条件式2の処理は実行されません。

上記の条件式1と条件式2は表裏一体で、必ずどちらか一方だけが成り立ちます。このようなケースでは、以下のように**else句**を使うことで1つにまとめることも可能です。

if( 条件式 ) {

条件式がtrue（真）のときの処理

} else {

条件式がfalse（偽）のときの処理

}

条件式が成り立つときは最初の{ }内が実行され、成り立たないときは2つ目の{ }内が実行されます。先ほどのif文2つをelse句でまとめると、以下のようになります。

//年齢が18未満かどうか判定

if( age < 18 ) {

// 年齢が18歳未満の場合

System.out.println("未成年です");

} else {

// 年齢が18歳以上の場合

System.out.println("成人です");

}

このようにif文を使うことで、条件が成り立つとき・成り立たないときで処理を切り替えられます。Javaに限らずプログラミングではif文をよく使う

条件式に使う比較演算子

条件式で使われていた「<」や「<=」は、**比較演算子**と呼ばれます。「A < B」のように比較演算子の両隣にデータを書くことで、2データの大小関係などを比較できます。

Javaの条件式に使う、主な比較演算子は以下のとおりです。

| **比較演算子** | **処理の内容** |
| --- | --- |
| == | 2つの値が等しい場合はtrueを返す（等価演算子）。 |
| != | 2つの値が等しくない場合はtrueを返す。 |
| > | 左辺の値が右辺の値よりも大きい場合はtrueを返す。 |
| >= | 左辺の値が右辺の値以上の場合はtrueを返す。 |
| < | 左辺の値が右辺の値よりも小さい場合はtrueを返す。 |
| <= | 左辺の値が右辺の値以下の場合はtrueを返す。 |

なお「trueを返す」とは、条件式から「true（成り立つ）という結果が導き出される」ことを意味します。これはif文に限らず、以下のようなプログラムでも簡単に確認できます。

System.out.println(1 != 2); // 条件式1

System.out.println(1 == 2); // 条件式2

「1 != 2」は「1と2は等しくない」、「1 == 2」は「1と2は等しい」という意味です。条件式1は成り立つためtrueが返され、条件式2は成り立たないためfalseが返されます。

補足1：論理型のデータは比較演算子なしで条件式に使える

true（真）またはfalse（偽）だけを表す論理型（boolean型）を学びました。論理型のデータは、比較演算子なしで条件式に使えます。

たとえば、成人かどうかを表す論理型の変数「isAdult」があるとしましょう。変数isAdultがtrueなら成人、falseなら未成年を意味します。

先ほどのif文で変数isAdultを使うとすれば、以下のような書き換えが可能です。

boolean isAdult = false; // 成人かどうか（true：成人／false：未成年）

if(isAdult) {

// 年齢が18歳以上の場合

System.**out**.println("成人です");

} else {

// 年齢が18歳未満の場合

System.out.println("未成年です");

}

整数型の変数age（年齢）で条件式を判定する場合は、比較演算子で「18」と比較する必要がありました。しかし論理型の変数isAdultであれば、変数名を( )内に書くだけでOKです。

このとき、変数isAdultがtrueなら最初の{ }内の処理が、falseなら2つ目の{ }内の処理が実行されます。先ほどとは反対に、成人の処理が先になっている点に注意が必要です。

なお、boolean型変数の頭に「!」（エクスクラメーションマーク）を付けると、条件式を反転できます。つまり、「boolean型変数がfalseかどうか」という条件です。

たとえば上記の条件式に「!」を使うと、条件式は「変数isAdultがfalseかどうか」、つまり「未成年かどうか」という条件になります。直前の例と処理が逆になりますね。

if(!isAdult) {

// 年齢が18歳未満の場合

System.out.println("未成年です");

} else {

// 年齢が18歳以上の場合

System.out.println("成人です");

}

変数isAdultがtrueなら、条件式は成り立たないことになり、2つ目の{ }内が実行されます。

このように論理型のデータを使うときは、比較演算子を使わず条件分岐が可能です。

補足2：浮動小数点型のデータをそのまま比較するのはNG

if文などの条件式で浮動小数点型のデータをそのまま比較するのは避けましょう。4章で学んだとおり、浮動小数点型は誤差が発生する場合があり、正確に比較できないためです。

たとえば以下の式では、float型とdouble型の変数に対して、それぞれ「12.3」を代入。そして、「2つの変数の値が等しいかどうか」という条件式の結果を表示しています。

float valFloat = 12.3F;

double valDouble = 12.3;

System.out.println(valFloat == valDouble);

どちらの変数にも「12.3」が入っているため、条件式が成り立ち、trueが表示されそうですよね。しかし実行すると、以下のようにfalse（成り立たない）と表示されます。

これは、一見同じに見える「12.3」でも、わずかな誤差が発生しているためです。このように、浮動小数点型には誤差が付き物のため、そのままでは厳密な比較ができません。

浮動小数点型の2データを厳密に比較したい場合、以下のような方法があります。現時点では「こんな方法もあるんだ」程度に受け取っておけばOKです。

比較する2データの差を取り、誤差が許容範囲であることを確認する

2データを整数型に変換してから比較する（小数部分が無視できる場合）

数値を文字列のように扱える「BigDecimalクラス」を使用する

参考までに、1つ目の方法で誤差をケアするサンプルコードを載せておきます。valFloatとvalDoubleの差をabs()メソッドで求め、許容範囲（0.0001）以内かチェックする例です。

余力のある人は、同じようにソースコードを書いて実行してみるとよいでしょう。

float valFloat = 12.3F;

double valDouble = 12.3;

// 0.0001までの誤差は許容範囲とみなし、等しいと判定する

if( Math.abs(valFloat - valDouble) <= 0.0001 ) {

System.**out**.println("2つの変数値は等しいです(誤差は許容範囲)");

} else {

System.out.println("2つの変数値は等しくありません");

}

なお、abs()メソッドは絶対値（マイナスを取った値）を求めるもの、詳しくは後ほど解説

上記を実行すると、誤差が許容されて、2データは同じ値と判定されます。

ほかにも浮動小数点型を正確に比較する方法はありますが、長くなるためここでは割愛します。浮動小数点型のデータは、正確に比較するのが難しいと覚えておきましょう。

if文を書いてみよう

では、実際にif文を書いてみましょう。簡単なくじ引きプログラムを作成します。0〜9の乱数（ランダムな数）を生成し、乱数に応じて当たりはずれを表示するプログラムです。

ここで作成する条件分岐は、以下の3パターンです（else ifについては後述）。

ifのみを記述するパターン（もし○○なら●●する）

ifとelseを記述するパターン（もし○○なら●●し、それ以外なら▲▲する）

ifとelse if、elseを記述するパターン（もし○○なら●●し、そうでなく□□なら■■し、それ以外なら▲▲する）

順番にやっていきましょう。

ifのみを記述するパターン（もし○○なら●●する）

まずは、値が9なら「大当たりです」と表示するプログラムを作成します。大まかな処理の流れは以下のとおりです。

0～9の乱数を生成し、変数randNumに代入する

変数randNumの値が9であれば、「大当たりです」という文字列を出力する

Javaには、数学的な処理を簡単に行えるMathクラスという便利なクラスがあります。乱数の生成には、Mathクラスのrandom()メソッドを使います

「Math.random()」のように書くと、0～1の範囲にある小数がランダムで出力されます。たとえば、Math.random()の結果を3回表示したときの結果は、以下のようになりました。

0.4312996933409997

0.7710649699670171

0.5742445873408349

実行するたびに、出力される小数の値が変わるのです。小数第一位は0～9のいずれかとなるため、上記の乱数を10倍して小数部分を切り捨てれば、0～9の乱数が得られます。

では、実践しましょう。以下のようにソースコードを書いてください。

package text;

public class section07 {

public static void main(String[] args) {

// ０～９のいずれかをランダムに生成（乱数を１０倍して小数部分を切り捨て

int randNum = (int)(Math.**random**() \* 10);

// 乱数randNumの値を出力

System.**out**.println(randNum);

// 当たりはずれの判定

if( randNum == 9 ) {

System.**out**.println("大当たりです");

}

}

}

実行結果

**9**

**大当たりです**

以下の箇所では、乱数を10倍した結果から整数部分だけを抽出するために、int型でキャストしています

int randNum = (int)(Math.random() \* 10);

実行のたびに表示される数字は0〜9の範囲で変わります。9が表示されるまで、繰り返しプログラムを実行してください。

変数randNumの値が9以外のときは、何も表示されません。「randNum == 9」（randNumが9に等しい）が成り立たず、if文の{ }内を通らないためです。

しかし、変数randNumの値が9になると、以下のように「大当たりです」と表示されます。

if文の条件式が成り立つとき・成り立たないときで、処理が分岐することを確認できました。

ifとelseを記述するパターン（もし○○なら●●し、それ以外なら▲▲する）

次は前述のelse句を用いて、if文の条件式が成り立たないときに「はずれです」と表示するソースコードを書きましょう。if文の部分にelse句がつくこと以外は、先ほどと同様です。

**package text.section07;**

**public class IfSyntax02 {**

**public static void main(String[] args) {**

**// TODO 自動生成されたメソッド・スタブ**

**// ０～９のいずれかをランダムに生成**

**int randNum = (int)(Math.random() \* 10);**

**// 乱数randNumの値を出力**

**System.out.println(randNum);**

**// 当たりはずれの判定**

**if( randNum == 9 ) {**

**System.out.println("大当たりです");**

**} else {**

**System.out.println("はずれです");**

**}**

**}**

**}**

実行結果（randNumが9のとき）

9

大当たりです

実行結果（randNumが0～8のとき）

5

はずれです

このように、条件式が成り立たないときも処理したい場合は、if文にelse句を組み合わせましょう。

ifとelse if、elseを記述するパターン（もし○○なら●●し、そうでなく□□なら■■し、それ以外なら▲▲する）

最後に、前項のプログラムに条件をプラスしましょう。「大当たり」「はずれ」はそのままに、変数randNumが8のときは「当たりです」と表示するようにします。

大まかに、以下のような条件分岐となります。

変数randNumが9のときは、「大当たりです」と表示する

変数randNumが8のときは、「当たりです」と表示する（これから追加）

それ以外のときは、「はずれです」と表示する

2つ以上の条件式で分岐させたい場合は、以下のようにelse if句を使います。ifとelseは1つの条件分岐で一度しか記述できませんが、else if句はいくつでも追加できます。

if( 条件式A ) {

条件式Aがtrue（真）のときの処理

} else if( 条件式B ) {

条件式Aはfalse（偽）だが、条件式Bはtrue（真）のときの処理

} else {

条件式Aも条件式Bもfalse（偽）のときの処理

}

今回の場合は、変数randNumが8のときに「当たりです」と表示する条件分岐をelse if句で追加します。以下のようにソースコードを書きましょう。

package text.section07;

public class IfSyntax03 {

public static void main(String[] args) {

// **TODO** 自動生成されたメソッド・スタブ

// ０～９のいずれかをランダムに生成

int randNum = (int)(Math.**random**() \* 10);

// 乱数randNumの値を出力

System.**out**.println(randNum);

// 当たりはずれの判定

if( randNum == 9) {

System.**out**.println("大当たりです");

} else if (randNum == 8) {

System.**out**.println("当たりです");

} else {

System.**out**.println("はずれです");

}

}

}

これまでと同じように、繰り返し実行してみましょう。変数randNumの値によって「大当たりです」「当たりです」「はずれです」の3種類が表示されればOKです。

実行結果（randNumが9のとき）

9

大当たりです

実行結果（randNumが8のとき）

8

当たりです

実行結果（randNumが0～7のとき）

5

はずれです

このように、複数の条件式で分岐させる場合はif-else文にelse if句を組み合わせましょう。

「かつ」「または」で複数の条件式を組み合わせる方法

これまでは、if文の( )内に記述する条件式が1つだけでした。しかし、以下のように複数の条件式を組み合わせたい場合もあります。

男性かつ20歳以上のとき

月曜日または水曜日のとき

「かつ（and条件）」「または（or条件）」のように複数の条件式を組み合わせるときは、論理演算子を使います。主な論理演算子は、以下の2つです。

| **論理演算子** | **処理の内容** |
| --- | --- |
| && | 条件式2つがともに成り立てばtrueを返す（AかつB）。 |
| || | 条件式2つの片方でも成り立てばtrueを返す（AまたはB）。 |

以下のようにソースコードを書いてください。random()メソッドを用いて0〜4の乱数を生成し、「かつ」「または」の条件分岐を行っています。

package text.section07;

public class IfSyntax04 {

public static void main(String[] args) {

// **TODO** 自動生成されたメソッド・スタブ

// ０～４のいずれかをランダムに生成

int randNum = (int)(Math.**random**() \* 5);

// 乱数randNumの値を出力

System.**out**.println(randNum);

// randNumが１より大きいかつ３より小さい

if( (1 < randNum) && (randNum < 3 ) ) {

System.**out**.println("randNumは１より大きい、かつ３より小さい値です");

} else {

System.**out**.println("and条件が成り立ちません");

}

// randNumが１に等しいまたは３に等しい

if((randNum == 1 ) || (randNum == 3)) {

System.**out**.println("randNumは１、または３です");

} else {

System.**out**.println("or条件が成り立ちません");

}

}

}

条件式が複数ある場合は、以下のように括弧でまとめると読みやすくなります。「&&」や「||」は優先順位が低い（処理される順番が遅い）ため、なくても誤動作にはなりません。

if( (1 < randNum) && (randNum < 3) ) { 2

実行する前に、and条件・or条件を満たすケースを整理しましょう。randNumが0〜4の乱数だと考えれば、それぞれを満たすのは以下のケースです。

and条件（1より大きい、かつ3より小さい）→randNumが2のとき

or条件(1に等しい、または3に等しい）→randNumが1または3のとき

上記を踏まえて、プログラムを何回か実行しましょう。以下のように、3種類の表示結果となればOKです。

変数randNumの値が2のとき（and条件を満たすが、or条件は満たさない）

2

randNumは1より大きい、かつ3より小さい値です

or条件が成り立ちません

変数randNumの値が1、3のとき（or条件を満たすが、and条件は満たさない）

1

and条件が成り立ちません

randNumは1、または3です

変数randNumの値がそれ以外（0、4）のとき（どちらの条件も満たさない）

0

and条件が成り立ちません

or条件が成り立ちません

複数の条件式を組み合わせるときは、このように「&&」や「||」を使いましょう。

まとめ

用語

条件分岐：条件によって処理を分けること

条件式：条件が成り立つかどうかを判定する式のこと

成り立つならtrue（真）、成り立たないならfalse（偽）を返す

条件式には比較演算子を使う

浮動小数点型のデータをそのまま比較するのはNG

比較演算子 処理の内容

== 2つの値が等しい場合はtrueを返す（等価演算子）。

!= 2つの値が等しくない場合はtrueを返す。

> 左辺の値が右辺の値よりも大きい場合はtrueを返す。

>= 左辺の値が右辺の値以上の場合はtrueを返す。

< 左辺の値が右辺の値よりも小さい場合はtrueを返す。

<= 左辺の値が右辺の値以下の場合はtrueを返す。

文法

条件分岐は主にif文・switch文を使う

if文の最もシンプルな書き方（条件式がtrueのときのみ処理を行う）

if( 条件式 ) {

条件式がtrue（真）のときの処理

}

条件式がfalseのときも処理を行いたい場合

if( 条件式 ) {

条件式がtrue（真）のときの処理

} else {

条件式がfalse（偽）のときの処理

}

複数の条件式で条件分岐したい場合（条件式の数に応じてelse if句を追加）

if( 条件式A ) {

条件式Aがtrue（真）のときの処理

} else if( 条件式B ) {

条件式Aはfalse（偽）だが、条件式Bはtrue（真）のときの処理

} else {

条件式Aも条件式Bもfalse（偽）のときの処理

}

複数の条件式を組み合わせるときは論理演算子を使う

論理演算子 処理の内容

&& 条件式2つがともに成り立てばtrueを返す（AかつB）。

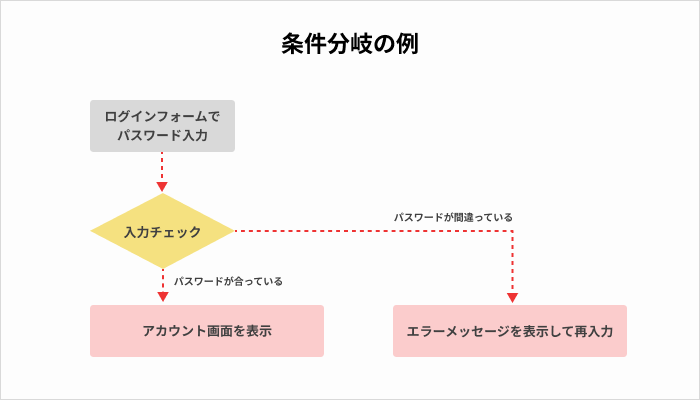
|| 条件式2つの片方でも成り立てばtrueを返す（AまたはB）。

switch文

f文と同じく、switch文も条件分岐構文の1つです。switch文を学ぶ前に、まずは条件分岐とif文について復習しましょう。

条件分岐とは、条件によって処理を分けることでしたね。以下のように、ユーザーの操作などに応じて正しく処理を切り替えることが可能です。

* パスワードが合っている場合→アカウント画面を表示
* パスワードが間違っている場合→エラーメッセージを表示



条件分岐構文として、if文を学びましたね。if文は「条件式がtrue（成り立つ）か、false（成り立たない）か」によって、実行する処理を切り替える構文でした。

if文は「〜より大きい」「〜に等しい」など、条件式にバリエーションを付けられる自由度の高さが強みです。基本的にどのような条件分岐でも、if文を使えば実装できます。

また、以下のようにelse if句を使うことで、複数条件での分岐も可能です。

if( 条件式A ) {

条件式Aがtrue（真）のときの処理

} else if( 条件式B ) {

条件式Aはfalse（偽）だが、条件式Bはtrue（真）のときの処理

} else {

条件式Aも条件式Bもfalse（偽）のときの処理

}

しかし、if文にも欠点はあります。条件数が増えるほどelse if句もたくさん必要になり、以下のような問題が生じるのです。すべての条件分岐に、if文が適しているとは限りません。

* ソースコードの可読性が落ちる（読みづらくなる）
* 同じ変数が何度も条件式に登場して冗長になる（無駄が多くなる）

例として、if文の欠点が顕著に出ている以下のソースコードをみてください。これは、順位を表す変数「ranking」の値を参照し、適切なメダルの色を表示するプログラムです。

int ranking = 1; // 順位

//順位に応じてメダルの色を判定

if( ranking == 1 ) {

System.**out**.println("金メダル");

} else if( ranking == 2 ) {

System.**out**.println("銀メダル");

} else if( ranking == 3 ) {

System.**out**.println("銅メダル");

} else {

System.out.println("参加賞");

}

「ranking == 1」「ranking == 2」「ranking == 3」と、条件式に何度も変数rankingが登場しています。もし変数名を変更したい場合、すべての条件式を変更しなければなりません。

また、中括弧{ } が分岐の数だけあって、見た目もごちゃごちゃしています。このように、1つの変数に対して複数の条件分岐が発生するケースには、if文は向かないのです。

上記のようなプログラムで条件分岐をさせるなら、switch文のほうが適しています。

switch文の書き方

switch文は、1つの変数をもとに条件分岐させる構文です。switch文の基本的な書き方は、以下のとおり。case句（caseで始まる部分）は、条件の数だけ追加しましょう。

switch(変数名) {

case 値1 -> 変数が値1に等しいときの処理;

case 値2 -> 変数が値2に等しいときの処理;

default -> 変数がどの値とも一致しないときの処理;

}

switch文は、( )内で指定した変数の値と、各case句に指定された値を上から順番に比較します。そして、変数と同じ値が見つかり次第、そのcase句の処理を実行するのです。

どのcase句とも値が一致しなかった場合は、最後のdefault句にある処理を実行します。switch文のdefault句は、if文の最後に書くelse句に近いものです。

前節のif文で条件分岐したプログラムをswitch文で書くと、以下のようになります。

int ranking = 1; // 順位

switch(ranking) {

case 1 -> System.out.println("金メダル");

case 2 -> System.out.println("銀メダル");

case 3 -> System.out.println("銅メダル");

default -> System.out.println("参加賞");

}

if文では9行もあった条件分岐が、5行に減りました。また、中括弧{ }も変数rankingも、1度だけしか登場しません。switch文のほうが、スマートに書けていますよね。

このように、1つの変数の値に応じて処理を切り替えたい場合は、switch文が便利です。

補足：以前のswitch文の書き方も知っておこう

Javaは今でもアップデートが行われている言語であり、文法も継続的に改良されています。実はswitch文も、アップデートにともない改良が行われました。

ひと昔前のswitch文は、以下のような書き方でした。case句の書き方が先ほどとは異なり、全体的に少し複雑なことがわかりますね。

switch(ranking) {

case 1:

System.out.println("金メダル");

break;

case 2:

System.out.println("銀メダル");

break;

case 3:

System.out.println("銅メダル");

break;

default:

System.out.println("参加賞");

break;

}

また「break;」はbreak文と呼ばれるもので、1つのcaseの処理が終わったときにswitch文を終わらせるために必要です。各caseの最後には、基本的にbreak文を書く必要があります。

たとえば「case 1:」のbreak文を書き忘れると、case 1:の処理後もswitch文の処理が終わりません。「case 2:」以降の処理まで意図せず行ってしまいます（フォールスルー）。

こうしたリスクや欠点を改良するために、2020年3月リリースのJava 14以降では大幅に仕様が変わりました。「case 1 ->」のように書くことで、break文が不要となったのです。

2023年3月時点でサポートされている最新バージョンはJava 20です。そのため、Javaのバージョンがよほど古くない限りは、基本的に新しいswitch文の書き方を使えます。

今では新しい書き方が推奨されているため、古い書き方をする必要はありません。ただし、チーム開発では古いソースコードを目にする場合もあるため、存在は知っておきましょう。

switch文を書いてみよう

では、実際にswitch文を書いてみましょう。ここでは、if文との違いがわかりやすいように、前章で作成した「くじ引きプログラム」をswitch文に置き換えてみます。

くじ引きプログラムの条件分岐をswitch文に置き換えよう

くじ引きプログラムは、0〜9の乱数（ランダムな数）を生成し、その値によって当たりはずれを判定する内容でしたね。大まかに、以下のような条件分岐となります。

変数randNumが9のときは、「大当たりです」と表示する

変数randNumが8のときは、「当たりです」と表示する

それ以外のときは、「はずれです」と表示する

ifでのソースコード

//0～9のいずれかをランダムに生成

int randNum = (int)(Math.**random**() \* 10);

//乱数randNumの値を出力

System.out.println(randNum);

//当たりはずれの判定

if( randNum == 9 ) {

System.**out**.println("大当たりです");

} else if( randNum == 8 ) {

System.**out**.println("当たりです");

} else {

System.out.println("はずれです");

}

上記プログラムの条件分岐を、switch文に置き換えてみましょう。以下のようにソースコードを書いてください。

package text.section08;

public class SwitchSyntax01 {

public static void main(String[] args) {

// **TODO** 自動生成されたメソッド・スタブ

// ０～９のいずれかをランダムに生成

int randNum = (int)(Math.**random**() \* 10);

// 乱数randNumの値を出力

System.**out**.println(randNum);

// 当たりはずれの判定

switch(randNum) {

case 9 -> System.**out**.println("大当たりです");

case 8 -> System.**out**.println("当たりです");

default -> System.**out**.println("はずれです");

}

}

}

if文での条件分岐と比べてスッキリしましたね。

なお、case句の並び順に厳格な決まりはありませんが、昇順か降順で統一しましょう。以下のようにバラバラにすると、プログラムが分かりにくくなります。

case 9 -> 〇〇;

case 1 -> △△;

case 6 -> □□;

case 2 -> ××;

このようにswitch文だと、if文と同じ条件分岐をよりスッキリ表現できます。1つの変数で条件分岐させたい場合は、switch文を有効活用しましょう。

複数の条件をまとめる場合

続いて、当たりの条件を増やしましょう。変数randNumが0を「大当たり」、1を「当たり」としてみます。これから実装するプログラムの条件分岐は、以下のとおりです。

変数randNumが0または9のときは、「大当たりです」と表示する

変数randNumが1または8のときは、「当たりです」と表示する

それ以外のときは、「はずれです」と表示する

仮にif文で大当たりの条件を増やすなら、以下のように複数の条件式を「||」でつながないといけません。この書き方では、条件数が増えると大変ですね。

if( (randNum == 0) || (randNum == 9) ) {

一方、switch文で複数条件による分岐を行う場合は、値をカンマ（,）で区切ることで1つのcase句にまとめられます。

case 値1, 値2, 値3... -> いずれかの値に一致したときの処理

では、実践しましょう。以下のようにソースコードを書いてください。

package text.section08;

public class SwitchSyntax02 {

public static void main(String[] args) {

// **TODO** 自動生成されたメソッド・スタブ

// ０～９のいずれかをランダム生成

int randNum = (int)(Math.**random**() \* 10);

// 乱数randNumの値を出力

System.**out**.println("randNum");

// 当たりはずれの判定

switch(randNum) {

case 0,9 -> System.**out**.println("大当たりです");

case 1,8 -> System.**out**.println("はずれです");

default -> System.**out**.println("はずれです");

}

}

}

先ほどと違うのは、以下の部分だけです。switch文では、これだけで分岐の条件を追加できます。

case 0, 9 -> System.out.println("大当たりです");

case 1, 8 -> System.out.println("当たりです");

1つの条件で複数の処理を行う場合

次は、大当たりのケースで「ラッキー！」と表示する処理を追加しましょう。これから実装するプログラムの条件分岐は、以下のようになります。

変数randNumが0または9のときは、「大当たりです」「ラッキー！」 と表示する

変数randNumが1または8のときは、「当たりです」と表示する

それ以外のときは、「はずれです」と表示する

1つの条件で複数の処理を行う場合は、中括弧{ }で処理を囲みましょう。 これはif文と似ていますね。

case 値 -> {

処理1;

処理2;

}

では、実践しましょう。以下のようにソースコードを書いてください。

package text.section08;

public class SwitchSyntax03 {

public static void main(String[] args) {

// **TODO** 自動生成されたメソッド・スタブ

// ０～９のいずれかをランダムに生成

int randNum = (int)(Math.**random**() \* 10);

// 乱数randNumの値を出力

System.**out**.println(randNum);

// 当たりはずれの判定

switch(randNum) {

case 0,9 -> {

System.**out**.println("大当たりです");

System.**out**.println("ラッキー！");

}

case 1,8 -> System.**out**.println("当たりです");

default -> System.**out**.println("はずれです");

}

}

}

書いたプログラムを繰り返し実行しましょう。変数randNumが0または9のときに、「大当たりです」に加えて「ラッキー！」も表示されるようになっていればOKです。

このように、case句に中括弧を使うことで複数の処理を実行できます。ただしソースコードが少し複雑になるため、1行分の処理で済むなら中括弧で囲む必要はありません。

switch式とは

Java 14以降では、switch文に加えて 「switch式」 も使えるようになりました。switch式とは、switch文の条件分岐により導き出された値を、計算式のように返せる構文です。

たとえば「1 + 1」という計算式は、結果として「2」という値を返しますよね。

a = 1 + 1;

// 以下と実質的に同じ(1 + 1の結果が2として返される)

a = 2;

これと同じように、switch式は条件分岐から値を返します。例として、以下の変数「award」に代入しているのは、switch式が返した賞品名です。

award = switch(ranking) {

case 1 -> "金メダル";

case 2 -> "銀メダル";

case 3 -> "銅メダル";

default -> "参加賞";

};

仮に変数rankingが「1」なら、switch式が導き出す値は「金メダル」です。つまり、上記のソースコードは以下と実質的に同様となります。これが、switch式の「値を返す」です。

award = "金メダル";

switch式の最も基本的な書き方は、以下のとおり。

変数名1 = switch(変数名2) {

case 値1 -> 変数2が値1に等しいときに返す値;

case 値2 -> 変数2が値2に等しいときに返す値;

・・

・・

default -> 変数2がどの値とも一致しないときに返す値;

};

switch文との違いは、大まかに以下の3点です。

「=」の右辺にswitch式が来る（値を何らかの変数に代入するため）

各case句の中身は、「処理」ではなく「値」

中括弧の最後にはセミコロン（;）が必要

では、実践しましょう。以下のようにソースコードを書いてください。

package text.section08;

public class SwitchSyntax04 {

public static void main(String[] args) {

// **TODO** 自動生成されたメソッド・スタブ

int ranking = 1; //順位

String award = ""; //商品名

//順位に応じた商品名を取得

award = switch(ranking) {

case 1 -> "金メダル";

case 2 -> "銀メダル";

case 3 -> "銅メダル";

default -> "参加証";

};

//賞品名を表示

System.**out**.println(award);

}

}

上記のプログラムでは変数rankingの初期値を「1」としているため、変数awardには「金メダル」が代入されます

このようにswitch式を使うと、条件分岐から導き出された値を変数に代入できます。変数rankingの初期値を色々と変えて、表示される賞品名が変わることを確認しましょう。

まとめ

用語

switch文：1つの変数をもとに条件分岐させる構文

switch式：条件分岐により導き出された値を、計算式のように返せる構文

文法

switch文の基本的な書き方

switch(変数名) {

case 値1 -> 変数が値1に等しいときの処理;

case 値2 -> 変数が値2に等しいときの処理;

・・・

・・・

default -> 変数がどの値とも一致しないときの処理;

}

複数の条件をまとめる場合は、カンマで区切る

case 値1, 値2, 値3... -> いずれかの値に一致したときの処理

1つの条件で複数の処理を行う場合は、中括弧でまとめる

case 値 -> {

処理1;

処理2;

}

switch式の基本的な書き方

変数名1 = switch(変数名2) {

case 値1 -> 変数2が値1に等しいときに返す値;

case 値2 -> 変数2が値2に等しいときに返す値;

・・

・・

default -> 変数2がどの値とも一致しないときに返す値;

};

switch文だと、1つの変数で条件分岐したいときにスマートな書き方が可能です。ただし、if文のように大小関係を比較したり、1つの条件分岐に複数の変数を使ったりはできません。

switch文とif文、それぞれの強みを理解することが大切です。