

プログラミング基礎 I

第2回

条件分岐

九州産業大学理工学部

pk@is.kyusan-u.ac.jp

第2回講義概要

- 条件分岐
 - if文
 - if-else文
 - if-else if-else文
 - switch文
- 演算子
 - 関係演算子
 - 論理演算子

テキスト（必須）

教科書がないと
解けない問題があります

- 見ひらきで学べるJavaプログラミング
 - 講義中の説明の他、授業後の復習、事前の予習、演習課題などで使用します
 - 必ず用意しておくこと

書籍情報

書名	見ひらきで学べるJavaプログラミング
著者名	古井陽之助、神屋郁子、下川俊彦、合志和晃
出版年	2019年8月31日
出版社名	近代科学社
ISBNコード	978-4-7649-0597-9
本体価格	2,400円（税抜）



プログラムを書く上での注意

- **一区切りごとに実行して、エラーがないか確認する！！**
 - エラーを放置して先に進むと、どんどんエラーが増えていく・・・
 - 慣れないうちは、一行書くごとに実行、くらいで丁度いい
- **半角/全角に注意！！**
 - 文字列を入力するとき以外は、「日本語入力オフ」で入力する
- エラーが出たら、**エラーの内容を確認**してみる
 - ～～を変数に解決できません：その名前の変数がない
 - 構文エラーがあります。“}” を挿入して...：閉じ括弧が足りない
 - **括弧は開いたら閉じる！**、対応があることを確認するのが大事

変数

- データを格納する入れ物
- 変化するデータを扱うのに使う
- 自分で変数を作ることできる

変数を作る（変数宣言）

- 変数宣言の構文

型 変数の名前;

- 型** 格納するデータの種類

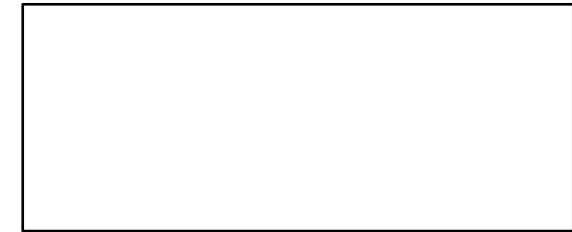
- 変数の名前** 自分でつける

- 例 整数型の変数score

```
int score;
```

整数型の変数x、y

```
int x, y; （複数まとめて）
```



score



x



y

型の種類 教科書P30

分類	名前	値
整数型	byte	8ビット整数 -128～127
	short	16ビット整数 -32768～32767
	int	32ビット整数 -2147483648～2147483647
	long	64ビット整数 -9223372036854775808～ 9223372036854775807
小数型	float	32ビット小数
	double	64ビット小数
文字型	char	1文字
文字列型	String	文字列
真偽値型	boolean	true(真)/false(偽)

値を格納する（代入）

- 代入の構文

変数の名前 = データ;

- データは具体的な数値であったり式であったり
- 例 scoreという変数に100を代入

score = 100;

100

score

変数の初期化

- 変数の宣言と初期値（変数に最初に格納される値）の格納
- 初期化の構文

型 変数の名前 = データ;

- 例：整数型の変数scoreを宣言し、100で初期化

```
int score = 100;
```

算術演算子 教科書P34

演算子	意味	例
+	加算 (足し算)	$5+3 \rightarrow 8$
-	減算 (引き算)	$10-3 \rightarrow 7$
*	乗算 (掛け算)	$3*2 \rightarrow 6$
/	除算 (割り算) (※整数演算では商は整数)	$3.2/2 \rightarrow 1.6$ $9/2 \rightarrow 4$
%	剰余 (割り算の余り)	$9\%2 \rightarrow 1$

変数の型についての注意

- 下記のプログラムの結果はどうなるか？

```
int x = 5;
```

```
int y = x/2;
```

```
System.out.println(y);
```

- 暗算してみると、 $5 \div 2$ なので2.5だが、
実行してみると、2と表示される
 - Javaでは、整数型と整数型の計算結果は整数型となるため、
2.5 の小数点以下が切り捨てられて 2 と表示される

型変換（キャスト） 教科書P31,37

- 値（数値や変数に代入されている値）を他の型に変換できる
- 数値型のキャスト

- (型名)値;
int x = 5;
int y = x / 2;
System.out.println(y); // 出力結果：2
double z = (**double**)x / 2; //整数型のxをdouble型に変換
System.out.println(z); //出力結果：2.5

小数型 / 整数型 とすると、小数点以下の計算もできるようになる

条件分岐 教科書P44

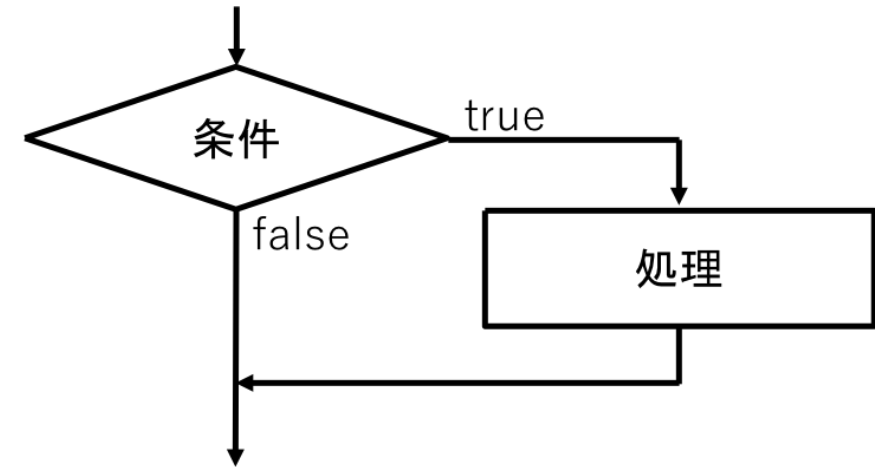
- もし〇〇なら××する(条件によって行動を変える)

```
if(条件式){
```

```
    処理
```

```
}
```

条件がtrue(正しい)なら処理を実行



- 例：明日、もし晴れたらドライブに行こう

```
if (tenki == HARE) {  
    drive();  
}
```

開き括弧と閉じ括弧の間に条件が成り立つときの処理を書く

↑ 閉じ括弧までが一つのまとまり

比較演算子

教科書P46

演算子	数学での書き方	意味	aが5の時の値
$a < 5$	$a < 5$	aは5より小さい、aは5未満	false
$a \leq 5$	$a \leq 5$	aは5以下	true
$a > 5$	$a > 5$	aは5より大きい	false
$a \geq 5$	$a \geq 5$	aは5以上	true
$a == 5$	$a = 5$	aは5と等しい	true
$a != 5$	$a \neq 5$	aは5と等しくない	false

== と = の違い

- == (論理演算子) は左辺と右辺の値が等しいか調べる
(等しければtrue、等しくなければfalse)
 - `System.out.println(a==10);`
 - a が 5 のとき実行するとfalseと表示される
- = (代入演算子) は、左辺の変数に右辺の値を格納する
 - `a = 10;`
 - 実行すると a の値が10になる

よくある間違い

```
if ( count >= 10 ) ; {  
    System.out.println("出席OK");  
}
```

if(式)文の処理文が;のみと解釈されるので、System.out..は常に実行される。

```
if ( count >= 10 ) {  
    System.out.println("出席OK");  
    if ( count < 10 ) {  
        System.out.println("出席不足");  
    }  
}
```

1つ目のif文の
閉じ括弧がない！

もし **count >= 10** のときで、さらにもし、**count < 10** なら・・・
あり得ない条件になってしまっている。

論理演算子

教科書P48

演算子	意味	利用例
a && b	a かつ b aとbの両方が成り立つ	a >= 10 && a < 20 aは10以上20未満
a b	a または b aかbの少なくとも一方が成り立つ	a < 0 a >= 20 aは0未満または20以上

|| は縦棒(パイプ)の記号 2 つ
Shift + ¥ で入力する

&や|(一つ) は論理演算子の && || とは全く別の演算子であることに注意
これら 2 つはビット演算(論理積、論理和)を行う際に使用する。

複数の条件式をつなげる（論理演算子）

- 2つの条件式の両方が成り立つときに処理を実行
 - 2つの条件式の間に **&&(かつ)** を入れて記述
 - aが**70以上80未満**のときtrue $a \geq 70 \ \&\& \ a < 80$
`if(a >= 70 && a < 80)`
- 2つの条件式の少なくとも一方が成り立つときに処理を実行
 - 2つの条件式の間に **||(または)** を入れて記述
 - aが**0未満**または**100以上**のときtrue $a < 0 \ || \ a \geq 100$
`if(a < 0 || a >= 100)`

よくある間違い

- ~以上~以下のような場合に、
下記のように書くことは出来ない
`if(70 <= a <= 80)`
 - `if(a >= 70 && a < 80)` 正しい記述
- 言語によっては上記の記載も可能だが、
Javaでは分けて記述する必要がある

if-else文 教科書P50

- もし〇〇なら××する、そうでなければ△△する

if(条件式){

処理1

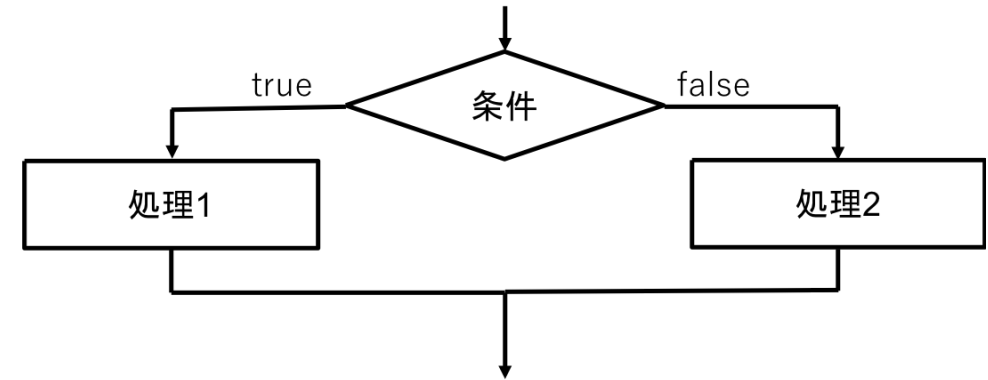
式がtrueなら処理1を実行

} else {

falseなら処理2を実行

処理2

}



- 例：もし晴れたらドライブに、そうでなければ本を読もう

```
if (tenki == HARE) {
```

```
    drive();
```

```
} else {
```

```
    read();
```

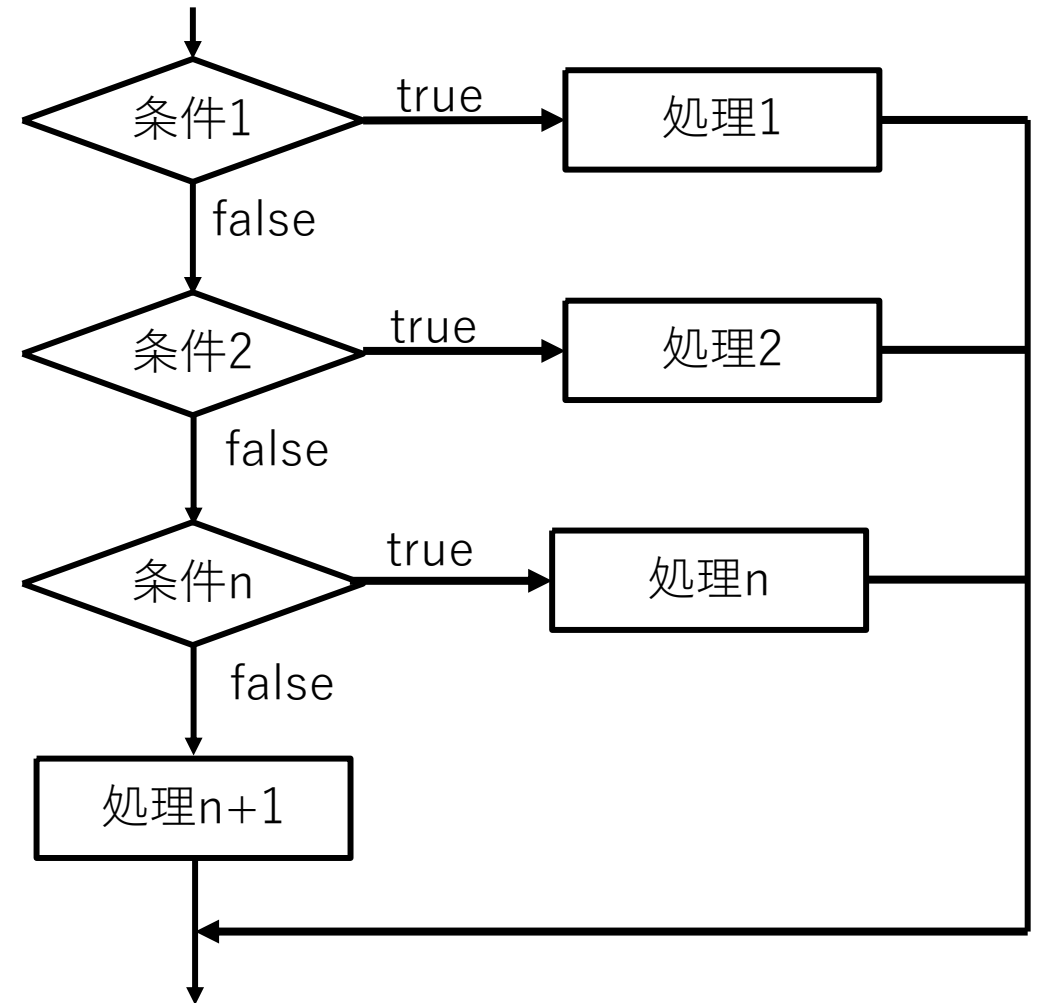
```
}
```

if-else if-else 3条件以上の分岐

教科書P53

- 条件が3つ以上あり、分岐する場合に使用

```
if ( 条件式1 ) {  
    処理1  
} else if ( 条件式2 ) {  
    処理2  
} else if ( ... ) {  
    ...  
} else if ( 条件式n ) {  
    処理n  
} else {  
    処理n+1  
}
```



正しく条件分岐できていますか？

```
if( x < 0 ) {           //①
    System.out.println(x+"は負の値");
} if ( x == 0 ) {       //②
    System.out.println(x+"はゼロ");
} else {                //③
    System.out.println(x+"は正の値")
}
```

x が -5 のとき…

- ①の条件を **満たす**
- ②の条件を **満たさない**
- ③の条件を **満たす**

-5は負の値
-5は正の値 と表示

偶数、奇数、〇〇の倍数 教科書P47

- 余りを求める演算子(**%**)を使う
- 偶数：2で割り切れる→2で割ると余りが0と等しい
 - **xが偶数**なら $x \% 2 == 0$
- 奇数：2で割り切れない→2で割ると余りが0でない
 - **xが奇数**なら $x \% 2 != 0$ ($x \% 2 == 1$ でも可)
- 〇〇の倍数：〇〇で割り切れる→〇〇で割ると余りが0
 - **xが3の倍数**なら $x \% 3 == 0$

偶数, 奇数の表示

- xの値が偶数なら「偶数」、奇数なら「奇数」と表示

```
if ( x % 2 == 0 ) { // もしも、xを2で割った余りが0ならば
    System.out.println("偶数");
}
if ( x % 2 != 0 ) { //もしも、xを2で割った余りが0でないなら
    System.out.println("奇数");
}
```

偶数, 奇数の表示をif-elseで

- xの値が偶数なら「偶数」、奇数なら「奇数」と表示
 - 整数は偶数か奇数しか存在しないので、偶数でないときを「それ以外」(else)とおける

```
if ( x % 2 == 0 ) {  
    System.out.println(x+"は偶数");  
} else {  
    System.out.println(x+"は奇数");  
}
```

3条件分岐の例（成績評価）

- scoreが 60未満ならC, 60以上70未満ならB, それ以上ならAと表示

```
if ( score < 60 ) {  
    System.out.println("C");  
} else if ( score < 70 ) {  
    System.out.println("B");  
} else {  
    System.out.println("A");  
}
```

3条件以上の数値による分岐

- scoreが 60未満ならC, **60以上70未満なら**B, それ以上ならAと表示

```
if ( score < 60 ) {
```

```
    System.out.println("C");
```

```
} else if(score < 70) {
```

```
    System.out.println("B");
```

```
} else {
```

```
    System.out.println("A");
```

```
}
```

この else に入ってきている時点でscoreは60以上

問題文通りに記述すると、

```
} else if( score >= 60 && score < 70) {
```

となって、これでも正しいが、
scoreが60以上の条件は書かなくても良い。

バグの元になるので、省略出来るところは省略しよう。

if文で文字列を比較する場合の注意

- Javaでは、`==` 演算子で文字列の比較は行えない

```
9 | Scanner sc = new Scanner(System.in);  
10 |  
11 | String s = "test";  
12 | String t = sc.next();  
13 | if(s == t) {  
14 |     System.out.println("sとtは同じ文字列です");  
15 | } else {  
16 |     System.out.println("sとtは違う文字列です");  
17 | }  
18 | }  
19 | }  
20 |
```

問題 コンソール

<終了> Test (1) [Java アプリケーション] C:\eclipse202103\java\16\bin\javaw.exe (2024/09/12 9:50:24 - 9:50:28)

test

sとtは違う文字列です

同じ、"test" の文字列を入力したはずなのに、
違う文字列だと判定された

if文で文字列を比較したい場合

- String.equalsメソッドを利用する
 - (String型の) 変数名.equals(文字列 または String型変数)

```
String s = "test";  
String t = sc.next();  
if(s.equals(t)) {  
    System.out.println("sとtは同じ文字列です");  
} else {  
    System.out.println("sとtは違う文字列です");  
}
```

- == と equalsメソッドの違いについては、後の回でも解説
 - 文字列の比較をする際は、equalsを使う、と覚えておこう

boolean型変数を使った条件分岐

- `String.equals`メソッドは、文字列の内容が同じであれば **true**, そうでなければ **false** となる

```
// isMsgEqual変数にsとtの比較結果を代入
boolean isMsgEqual = s.equals(t);

if(isMsgEqual){    // isMsgEqualがtrueならば
    System.out.println("sとtは同じ文字列");
}
```

- boolean型変数の場合には、比較演算子を省略可能

オンライン小テストでのif文記述のルール

- 条件式は、主体となるものを左辺に書く

例) scoreが60未満の場合

if(score < 60) とする

if(60 > score) は採点上では×

60はscoreより大きいとなって理解しにくくなる

- 変数と値を比較する場合には、変数が左辺とする

オンライン小テストでのif文記述のルール

- 問題文にかかれている通りに記述する
scoreが60未満の場合は「不可」、そうでないときは「可」

これはOK

```
if(score < 60){  
    System.out.println("不可");  
} else {  
    System.out.println("可");  
}
```

これはNG

```
if(score >= 60){  
    System.out.println("可");  
} else {  
    System.out.println("不可");  
}
```

結果は同じだが、指示通りになっていない

if文記述の際の注意

- **{ }は省略しない**
 - 制御文 (if, for, while等) 周りは必ず { } で囲む
 - 省略すると可読性が下がり、コード変更時に誤りやすくなる
- 条件式で boolean型の変数を比較しない
 - `if( isMsgEqual == true )` のように記述しない
`if( isMsgEqual )` だけで良い (余計な式を書かない)

条件式記述時のコーディング規約

※よくあるコーディング規約の例

- 主体となるものを左辺に書く
`if(score < 60)`
- 大小を比較演算子は、`<` と `<=` のみを使う(`>`, `>=`は使わない)

`if(score < 60 && 70 <= score)`

どちらか一方が採用される事が多い

- 関係演算子の前後には半角スペースを入れる
 - 算術演算子、論理演算子も同様

switch文

教科書P54

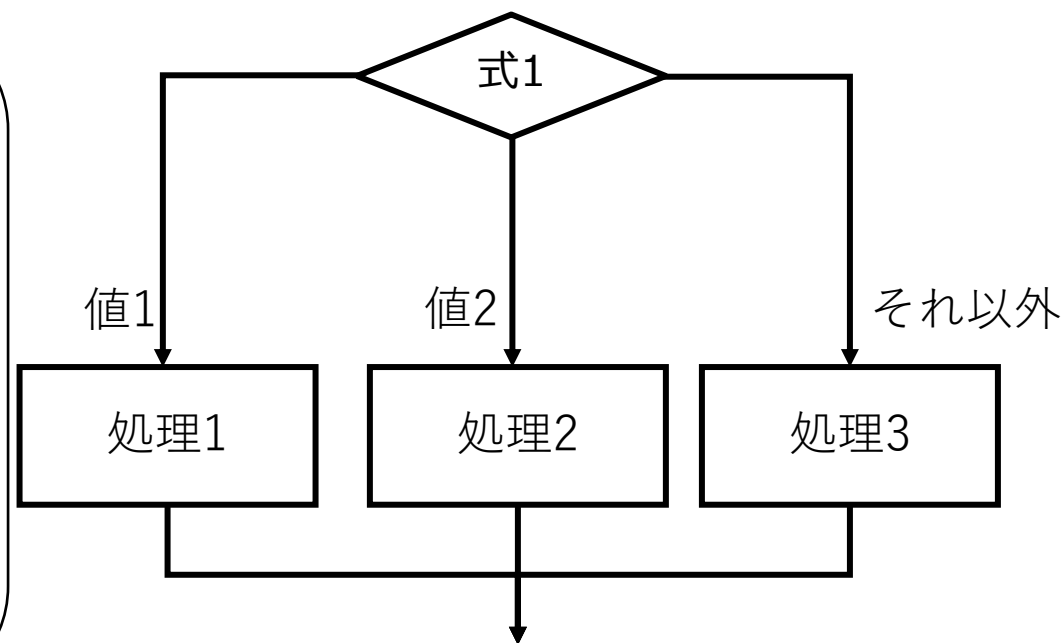
- ある式の値に応じて分岐するプログラム
 - 複雑な条件は指定できない代わりに、分岐が多い場合にシンプルに記述できる

switch文の構文

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

switch文の例

```
switch(num) {  
  case 1:  
    処理1; //numが1の時の処理  
    break;  
  case 2:  
    処理2; //numが2の時の処理  
    break;  
  default: //それ以外の時の処理  
    処理3;  
}
```



switch文の例

switch文

```
int num = sc.nextInt();
switch(num) {
    case 1:
        System.out.println("おはよう"); //numが1の時の処理
        break;
    case 2:
        System.out.println("おやすみ"); //numが2の時の処理
        break;
    default: //それ以外の時の処理
        System.out.println("さようなら");
}
```

if文 で書いた場合（同じ処理内容）

```
int num = sc.nextInt();
if(num == 1) {
    System.out.println("おはよう"); //numが1の時の処理
} else if(num == 2) {
    System.out.println("おやすみ"); //numが2の時の処理
} else {
    System.out.println("さようなら");
}
```

switch文使用時の注意

- 「変数値が~~と等しい」場合にのみ利用できる
 - 大小比較は行えない（if文を利用しましょう）
- 整数値型(int, short, byte)、文字型(char), 文字列型(String)のみ利用可能
 - 小数値型(double, float)は利用できない
 - 文字列(String型)が等しいかの比較が可能
 - Switch文では、String.equalsメソッドを使用した場合と同様の比較が行われる

switch文でよくある間違い

- break; 忘れ

```
int num = sc.nextInt();
switch(num) {
    case 1:
        System.out.println("おはよう");

    case 2:
        System.out.println("おやすみ");
        break;
    default: //それ以外の時の処理
        System.out.println("さようなら");
}
```

break; を書かなかった場合、
次のbreak; の行まで実行されてしまう

原則として、switch文には必ずbreakを
書きましょう

- おはよう
おやすみ

と表示される

まとめ

- 条件分岐
 - if文
 - if - else if 文
 - if文による文字列の比較
 - `String.equals()` メソッド
- switch文
 - switch文使用上の注意