

情報処理技法 (Javaプログラミング)2

第1回 前期の復習

人間科学科コミュニケーション専攻
白銀 純子

第1回の内容

- ◆ オリエンテーション
- ◆ 前期の復習

授業目標

- ◆ 簡単なアルゴリズムを理解すること
- ◆ オブジェクト指向プログラミングの基礎を理解すること
- ◆ GUIの作成方法について学習し、ソフトウェアの動作の概略を理解すること

参考書・連絡先・資料置き場

◆ 参考書

- ◆ 基礎講座 Java, 白銀純子, 毎日コミュニケーションズ, 2010

◆ 連絡先

研究室: **8号館4階8413室**

メールアドレス: ***junko@lab.twcu.ac.jp***

※質問は、メールか研究室にどうぞ

◆ 授業Webページ

<http://www.cis.twcu.ac.jp/~junko/Programming/>

- ◆ 講義中に見せる資料
- ◆ 演習問題の解答例や課題, etc.

成績評価とレポート

- ◆ 成績評価

出席: 30%, レポート+期末試験: 70%

- ◆ レポート

- ◆ 期末試験(持ち込み全て可で、実技試験がメイン)

履修の前提条件

- ◆ 情報処理技法 (Javaプログラミング) 1を履修しているか、または同等の知識およびスキルを持っていること
 - ◆ 持っていないと、授業について来れないため

前期の復習

プログラミング言語

- ◆ コンピュータに命令を伝えるための言語(人間が理解できる言葉)
 - ◆ コンピュータが理解できる言葉: 機械語
- ◆ 誰がいつ解釈しても意味が同じ
 - ◆ 1つの文が幾通りにも解釈できると、コンピュータは処理できない
(コンピュータは、1文1文直訳しかしない)
- ◆ 文法規則を厳密に定義

プログラミング言語で書かれた命令書を機械語に訳すことで、
コンピュータに命令を実行させる

命令書の機械語への訳し方

- ◆ 命令書を最初から最後まで機械語に翻訳し、翻訳した結果のファイルをコンピュータに渡す
- ◆ 命令書を最初から1行1行読んで機械語に訳す(通訳する)

プログラミング言語にまつわる用語

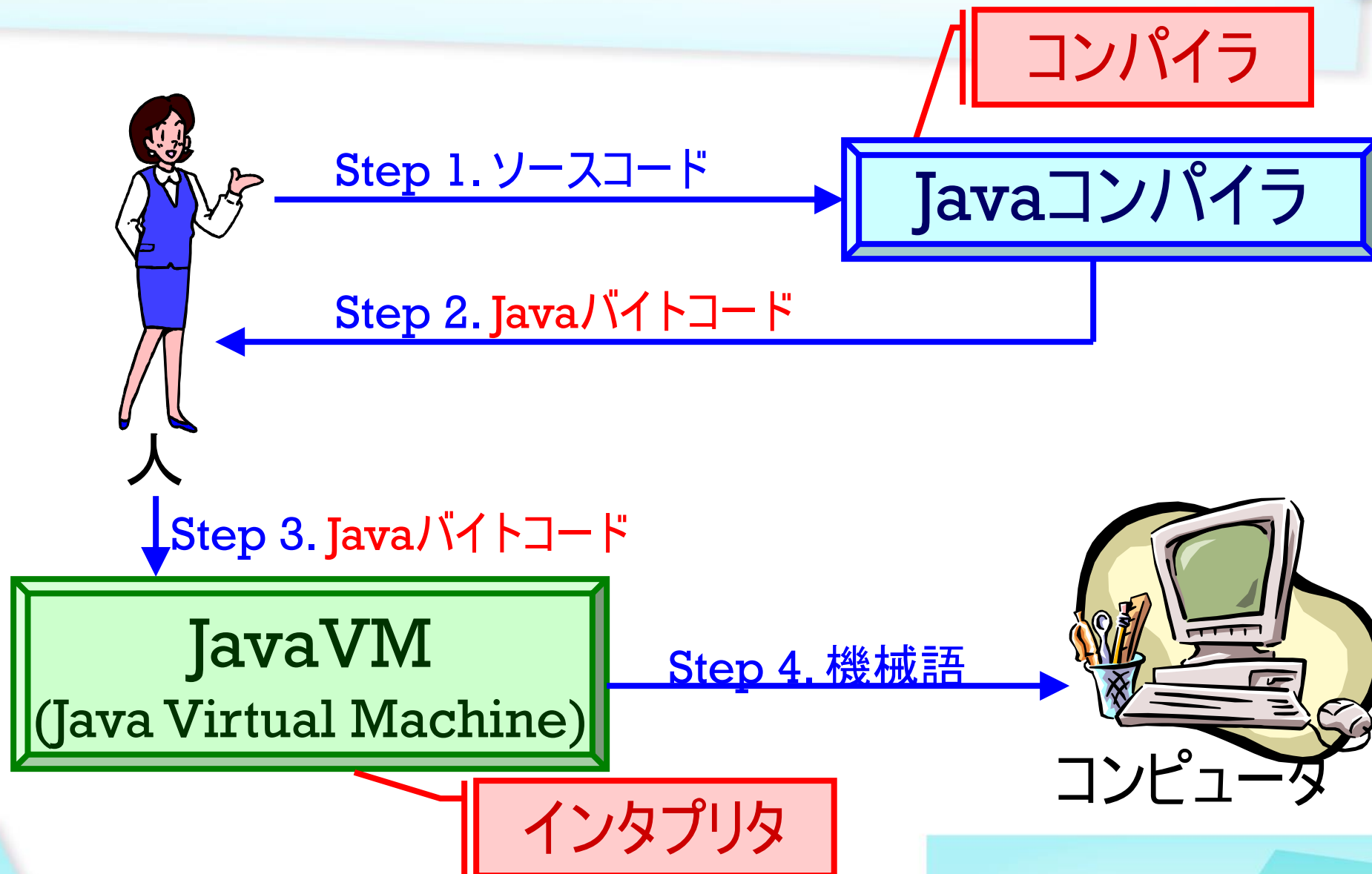
- ◆ **コンパイル**: プログラミング言語で書かれた命令書を機械語に翻訳すること
- ◆ **コンパイラ**: 命令書をコンパイルするためのソフトウェア
- ◆ **インタプリタ**: 命令書を通訳するためのソフトウェア
- ◆ **命令書**: ソースコード(プログラム)
- ◆ **ソースコードを作成すること**: プログラミング
- ◆ **ソースコードをコンパイルしたもの**: オブジェクトコード(プログラム)

Javaって何?

- ◆ プログラミング言語の1つ
- ◆ コンピュータやOSに依存せず、実行可能

コンパイルしたプログラムは、通常は、OSが異なると実行できない

Javaのしくみ



Javaプログラムの実行方法

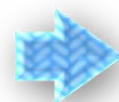
- ◆ Step1: Jeditなどでソースコードを作成する

- ◆ ファイル名は、必ず拡張子を「**.java**」とすること

- ◆ Step2: ソースコードをコンパイルする

(コマンド名: **javac**, 引数: ソースコードのファイル名)

% **javac** ファイル名 **.java**



「ファイル名 **.class**」というファイルが作成される

- ◆ Step3: JavaバイトコードをJavaVMで実行する

(コマンド名: **java**, 引数: 拡張子なしのファイル名)

% **java** ファイル名

拡張子なし

データ型と変数

プログラムで扱うデータ

- ◆ プログラムでは、データは必ず「箱」に入れて扱う
 - ◆ 例えば、「りんご1個の値段を扱うための箱」、「りんごの生産地の名前を扱うための箱」
- ◆ 「箱」: **変数**と呼ぶ
- ◆ 「変数」には名前を付ける
 - ◆ Ex. 「りんご1個の値段を扱うための変数」: applePrice
 - ◆ Ex. 「りんごの生産地名を扱うための変数」: applePlace

データの種類

◆ プログラムで扱うことのできるデータの種類

- ◆ 整数(**int**)
- ◆ 小数(**float**, **double**)
- ◆ 文字列(**String**)

} 「データ型」と呼ぶ

◆ 変数は、名前を付けたときに、どの種類のデータを入れるのかを決めておく

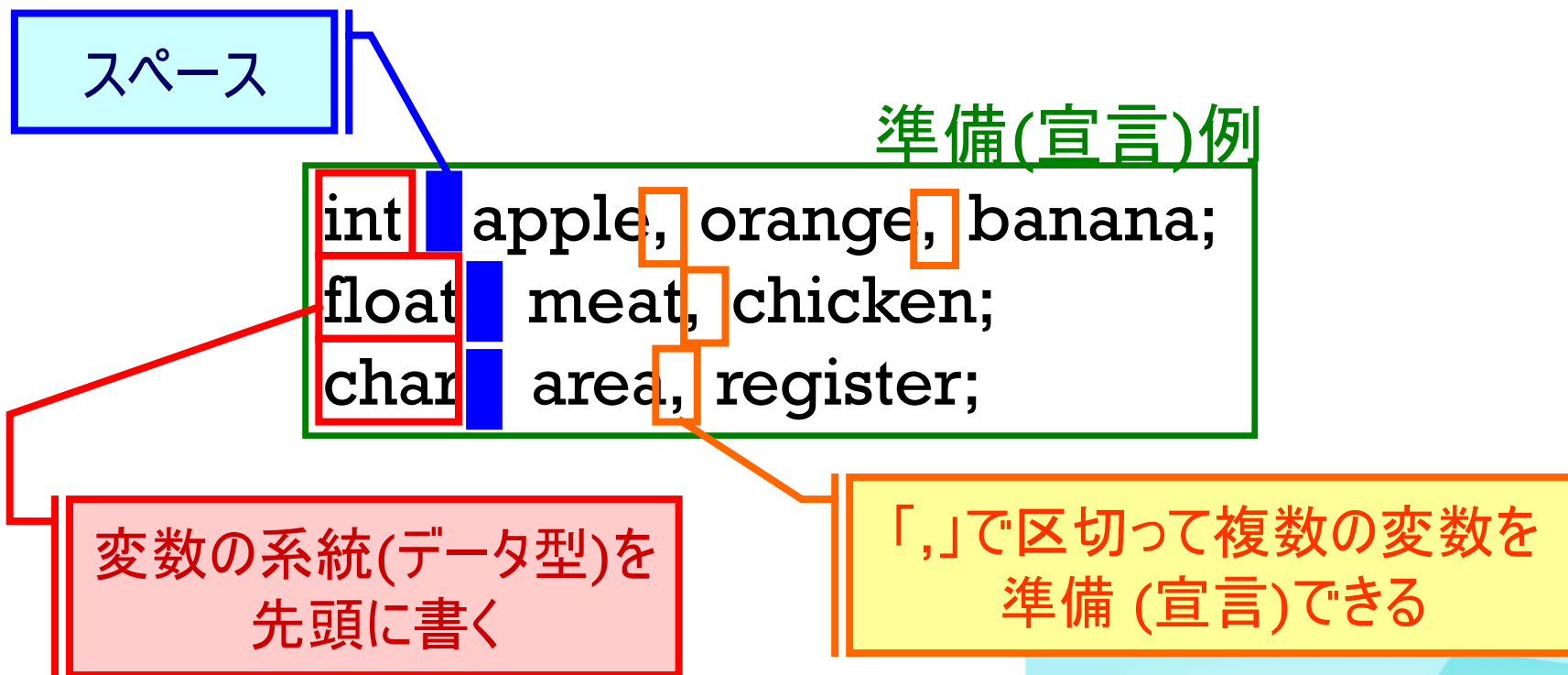
- ◆ りんご1個の値段を扱うための変数: applePrice - int型
- ◆ りんごの生産地名を扱うための変数:
applePlace - String型

「変数の宣言」と呼ぶ

変数の宣言

◆ 変数を使う(データを入れるなど)前に、
変数を準備する必要

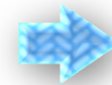
変数を「宣言する」という



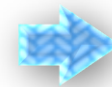
変数の値

- ◆ 変数(箱)に値(データ)を入れて扱う
- ◆ 「=」で値を決める  「代入する」という
= 箱の中に具体的なデータを入れること
- ◆ 用意した変数に初めて値を入れること: 初期化

購入したりんごの値段が200円だった場合

 `applePrice = 200;`

りんごの生産地が青森だった場合

 `applePlace = "青森";`

数の計算

◆ 変数(箱)の中には、計算結果や処理結果を入れることもできる

◆ 足し算: +

◆ 引き算: -

◆ かけ算: *

◆ 割り算(商): /

◆ 割り算(余り): %

例えば...代金計算
(支払い金額: result)

100円のりんごを10個買った場合

➡ `apple = 10;`
`result = apple * 100;`

「result」には、「1000」という
結果が入る

100円のりんごを10個,
150円のバナナを5個買った場合

➡ `apple = 10;`
`banana = 5;`
`result = apple * 100 + banana * 150;`

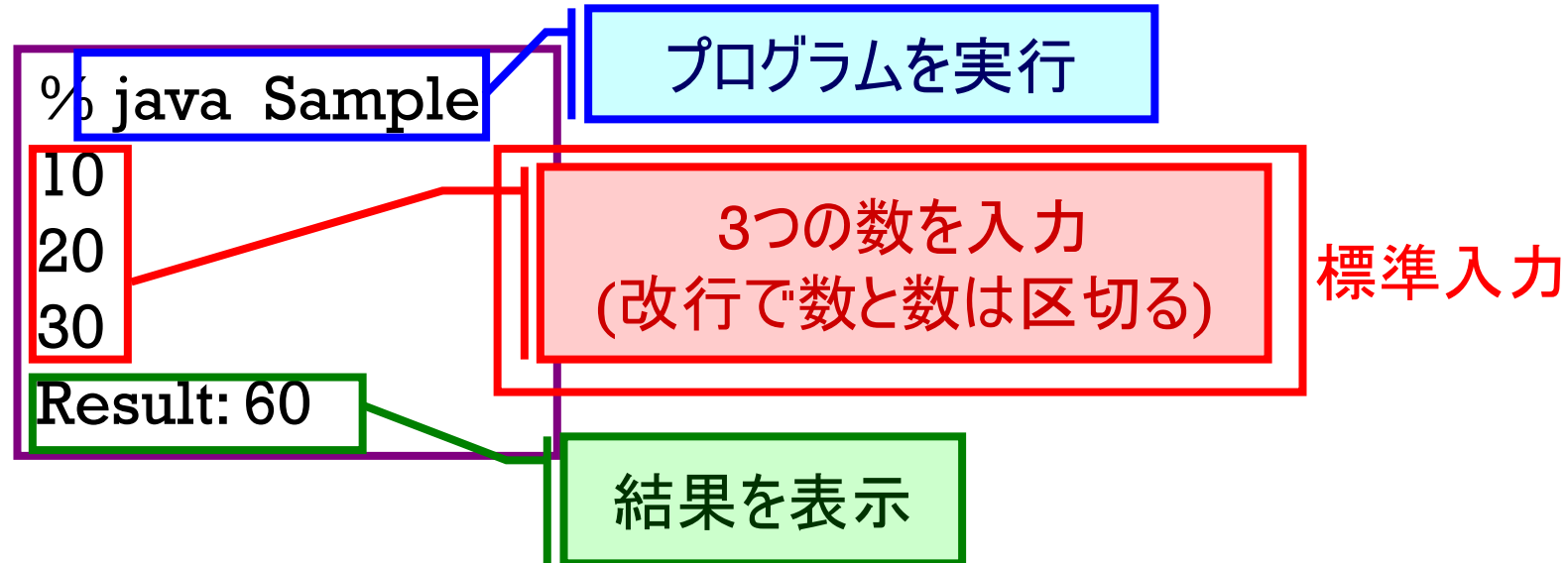
「result」には、「1750」という
結果が入る

標準入出力

標準入力とは

- ◆ プログラム実行中に、ターミナル上に値を入力し、その値をプログラムが読み取ること

例えば...数を3つ入力して、その合計を求めるプログラム(Sample.java)



標準入力のプログラム

```
import java.io.*;
import java.lang.*;
public class Standard {
    public static void main(String[] args) {
```

お約束(1)

```
    try {
```

```
        BufferedReader br =
            new BufferedReader(new InputStreamReader(System.in));
```

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

ターミナルからの入力の
読み込み部分

```
    }
```

```
    catch (IOException e) {
        System.out.println("標準入力において例外が発生しました。");
    }
```

```
    }
}
```

お約束(2)

ターミナルからの読み込み部分(1)

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

ターミナルからの入力を受け取る変数の宣言

ターミナルからの入力を読み込むメソッド
(戻り値: String型)

この式では、ターミナルから読み込んだ文字列を「str」に代入している

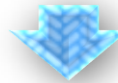
※「br.readLine()」の「br」は、「BufferedReader br」の「br」
(brでなくても良いが、この2つを合わせる必要)

ターミナルからの読み込み部分(2)

- ◆「`br.readLine()`」で読み込むのは、改行まで(データ1つ分だけ)

```
% java Sample  
10  
20  
30  
Result: 60
```

改行で区切ることで3つのデータを入力



```
str1 = br.readLine();  
<1つめのデータの処理>  
str2 = br.readLine();  
<2つめのデータの処理>  
str3 = br.readLine();  
<3つめのデータの処理>
```

入力するデータの
数だけ必要

ターミナルからの読み込み部分(3)

- ◆「`br.readLine()`」で読み込まれるものは、必ず「String」型
 - ◆ 数でない文字列を扱うときにはこれでいい
 - ◆ 入力されたものが数であっても、コンピュータは「文字の連なり」と考えていて、数値とは考えていない
- ➡ 数が入力される場合には、「それは数値である」とコンピュータに教える必要

コンピュータに「それは数値である」と教える方法は？

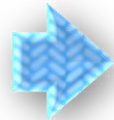
文字列を数値に変換(int型)

◆ 文字列を、「それはint型の数値である」とコンピュータに教える

ターミナルから読み込んだものが
代入されている変数

```
Integer.parseInt(str);
```

コンピュータが「文字の連なり」と考えているものを、
int型の数値であると教えるメソッド



この結果をint型の変数に代入

つまり... `num = Integer.parseInt(str);`

※「num」はint型の変数

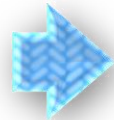
文字列を数値に変換(float型)

◆ 文字列を、「それはfloat型の数値である」とコンピュータに教える

ターミナルから読み込んだものが
代入されている変数

```
Float.parseFloat(str);
```

コンピュータが「文字の連なり」と考えているものを、
float型の数値であると教えるメソッド



この結果をfloat型の変数に代入

つまり... `num = Float.parseFloat(str);`

※「num」はfloat型の変数

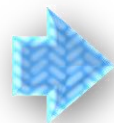
文字列を数値に変換(double型)

◆ 文字列を、「それはdouble型の数値である」とコンピュータに教える

ターミナルから読み込んだものが
代入されている変数

```
Double.parseDouble(str);
```

コンピュータが「文字の連なり」と考えているものを、
double型の数値であると教えるメソッド



この結果をdouble型の変数に代入

つまり...

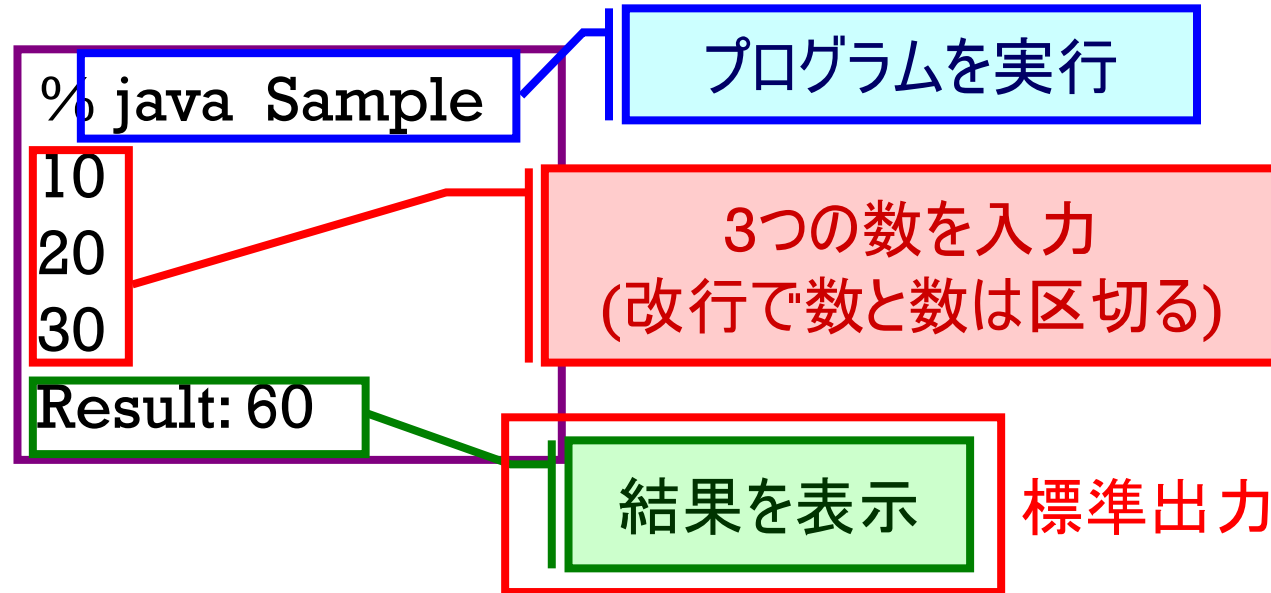
```
num = Double.parseDouble(str);
```

※「num」はdouble型の変数

標準出力とは

- ◆ プログラム実行中に、ターミナル上に何らかの文字列を表示させること

例えば...数を3つ入力して、その合計を求めるプログラム(Sample.java)



標準出力の方法(1)

- `System.out.println(文字列や数値);`
 - ✓ 出力部分(「文字列や数値」の部分)の後に改行が入る
- `System.out.print(文字列や数値)`
 - ✓ 出力部分(「文字列や数値」の部分)の後に改行が入らない

標準出力に処理結果を
表示(出力)する命令

「文字列や数値」の部分: String型のデータの作り方と同じ

- 出力したい文字列や変数を「+」でつなげて書く
- 変数でない文字列は、「"」で囲んで書く

標準出力の方法(2)

- ◆「Please input a number」を表示したい場合
(全て変数でない文字列)

➡ `System.out.println("Please input a number.");`

- ◆「3 apples」と表示したい場合(「3」は変数「num」の値,
「apples」は変数でない文字列)

➡ `System.out.println(num + " apples");`

変数(「」では囲まない)

変数でない(「」で囲む)

条件によって処理内容を変更する～if～

条件分岐(その1)(1)

◆ もし...ならば、～をする

boolean型で
結果が出るもの

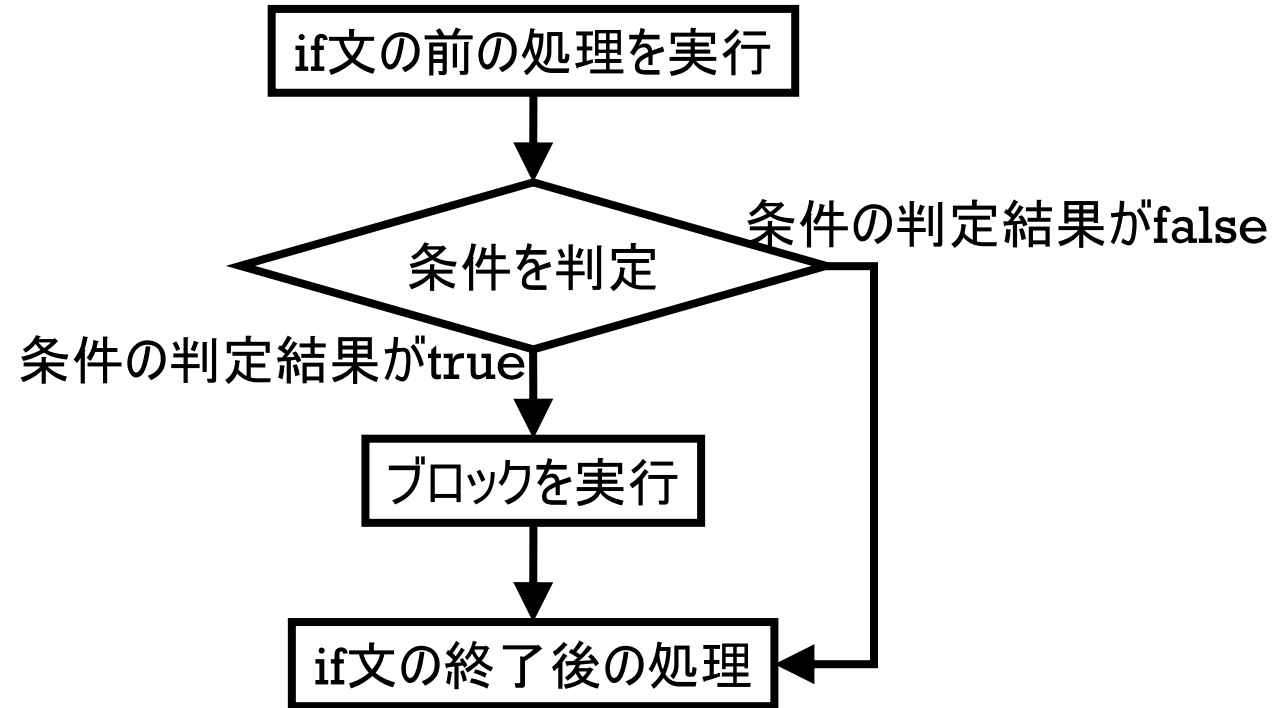
「条件」がtrueのときにすることは、
このカッコ内に書くこと

```
if (条件) {  
    「条件」がtrueのときにすること  
}
```

条件分岐(その1)(2)

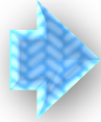
```
if (条件) {  
  文1;  
  文2;  
  文3;  
}
```

ブロック



条件分岐(その1)(例)

例: 買いたい服の値段が5000円以内ならば、服を買う



```
if (買いたい服の値段 <= 5000円) {  
    服を買う;  
}
```

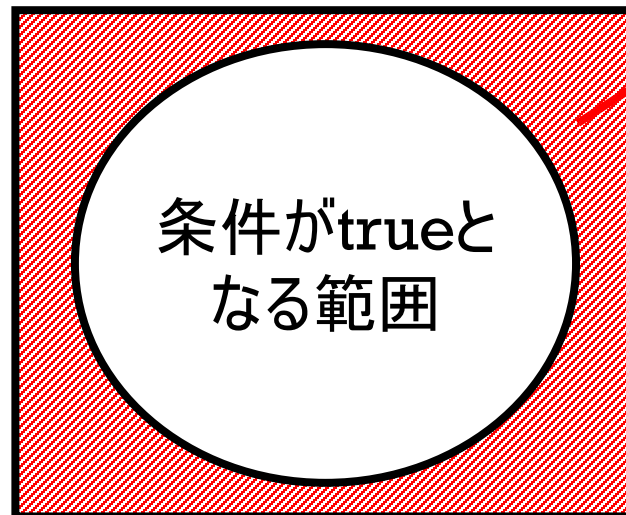
服の変数を「cloth」(int型)とすると...

```
if (cloth <= 5000) {  
    服を買う;  
}
```

条件分岐(その2)(1)

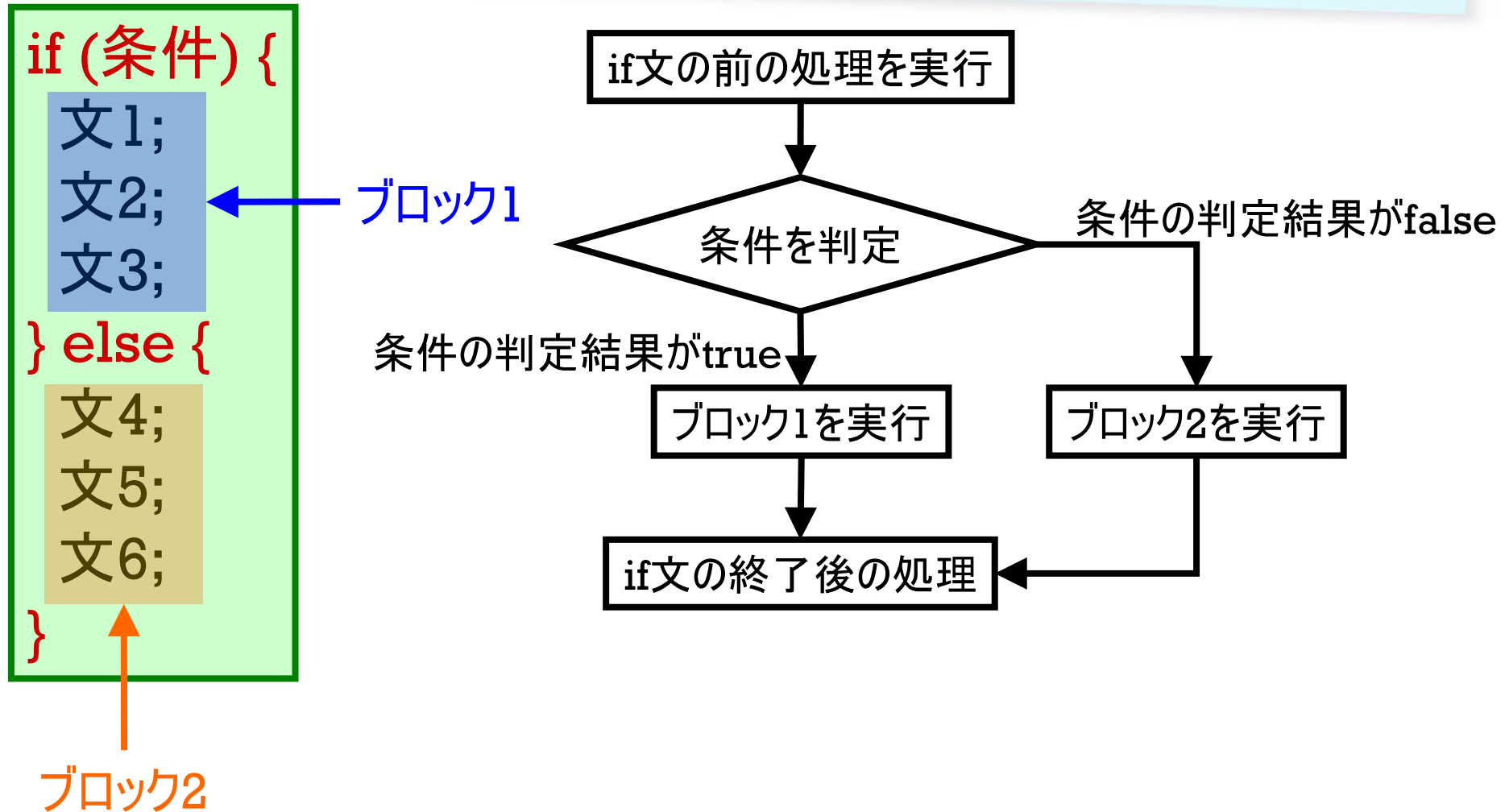
- ◆ もし...ならば～をし、そうでなければ---をする

```
if (条件) {  
    「条件」がtrueのときにすること  
}  
else {  
    「条件」がfalseのときにすること  
}
```



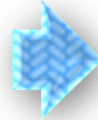
「else」の部分が
実行される範囲

条件分岐(その2)(2)



条件分岐(その2)(例)

例: 買いたい服の値段が5000円以内ならば服を買い、
そうでなければ服を買うのをあきらめる



```
if (買いたい服の値段 <= 5000円) {  
    服を買う;  
} else {  
    服を買うのをあきらめる;  
}
```

買いたい服の値段 > 5000円
のときにすること

服の変数を「cloth」(int型)と
すると...

```
if (cloth <= 5000) {  
    服を買う;  
} else {  
    服を買うのをあきらめる;  
}
```

条件分岐(発展)(1)

◆ もしAならばXをし、AでなくてBならばYをし、AでもBでもなければZをする

```
if (条件A) {  
    Xをする;  
} else if (条件B) {  
    Yをする;  
} else {  
    Zをする;  
}
```

- まず最初に「条件A」を判断し、「条件A」が正しければXをする。
- 「条件A」が正しくなければ「条件B」を判断し、「条件B」が正しければYをする。

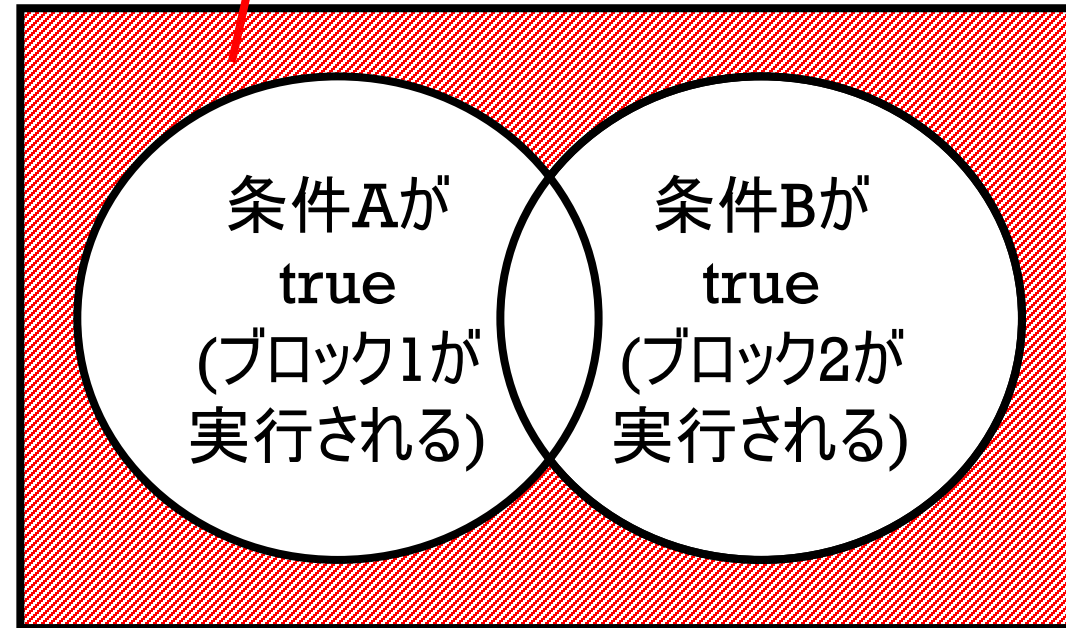
➡ AとBがどちらも正しい場合でも、Xをする(Yはしない)

※「else if」で、いくつでも条件をつけることができる

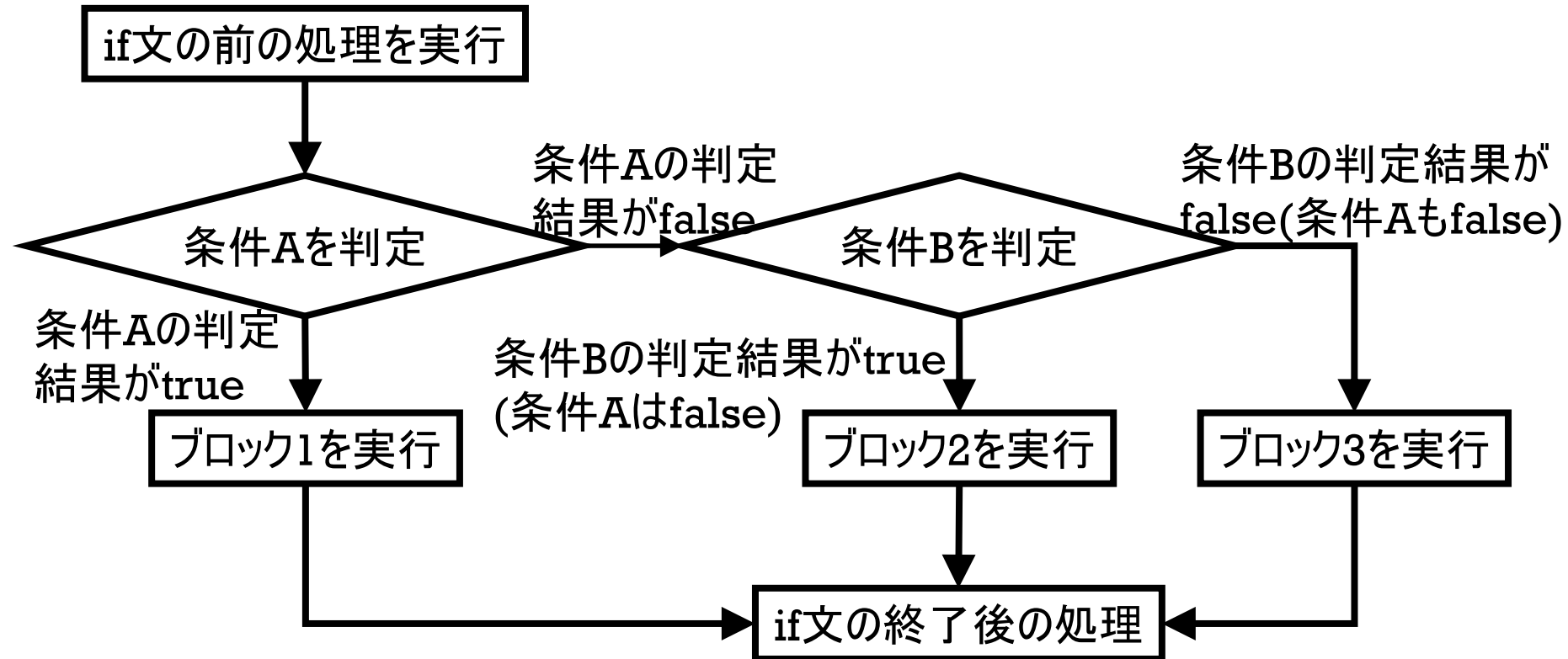
条件分岐(発展)(2)

```
if (条件A) {  
  文1;  
  文2; ← ブロック1  
  文3;  
} else if (条件B) {  
  文4;  
  文5; ← ブロック2  
  文6;  
} else {  
  文7;  
  文8; ← ブロック3  
  文9;  
}
```

「else」の部分(ブロック3)が
実行される範囲



条件分岐(発展)(3)



条件分岐(発展)(例)

例: レストランAがすいていればAで食事をし、AがすいていなくてBがすいていればBで食事をする。AもBもすいていなければ、あきらめて帰る



```
if (レストランAがすいている) {  
    レストランAで食事をする;  
} else if (レストランBがすいている) {  
    レストランBで食事をする;  
} else {  
    あきらめて帰る;  
}
```

同じ処理を何回も繰り返す～for, while～

for文

主に配列を扱う場合など、繰り返す数が決まっているときに利用

for文の書き方と動作

```
for(開始時の式; 条件; 繰り返し時の式) {  
    文1;  
    文2;  
    .....  
}
```

ブロック

for文の前までの処理を実行

開始時の式を実行

for文の動作

false

条件が正しい

true

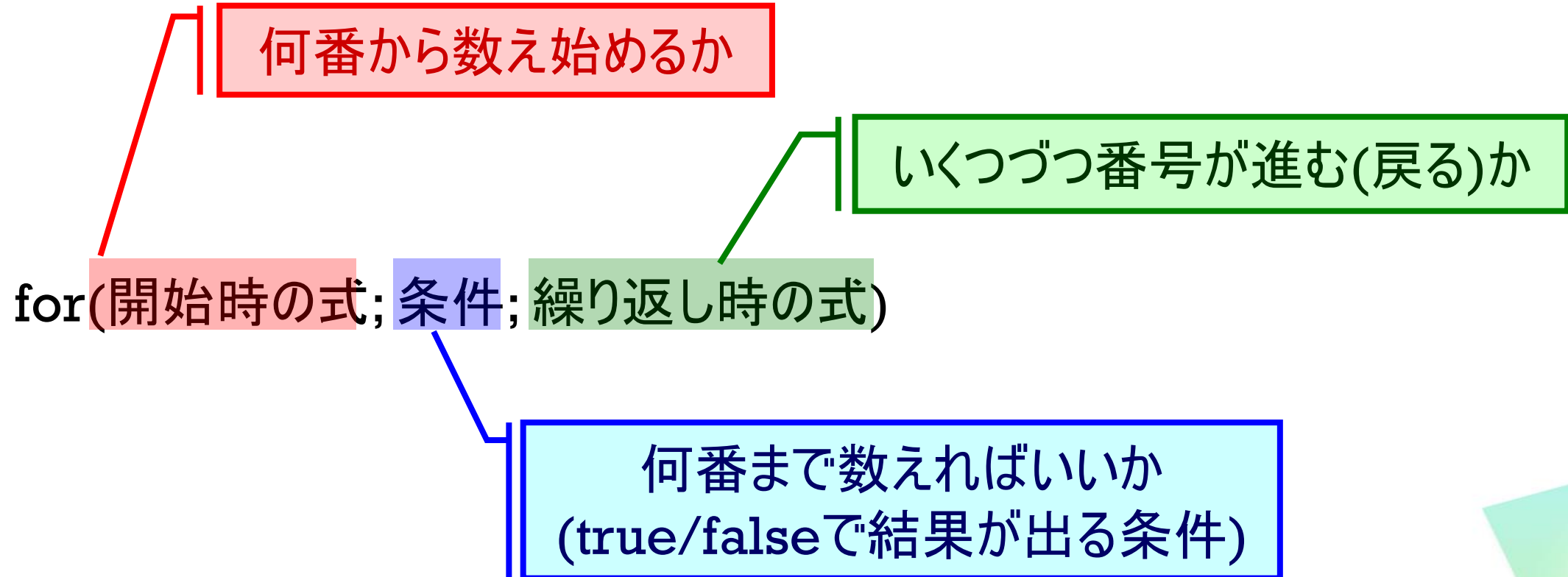
繰り返し時の式を実行

for文の後を実行

ブロックを実行

for文の一般的な使い方

- ◆ 1, 2, 3, ... や、50, 49, 48, ... のように、何かの数を数える場合に利用
- ◆ 繰り返す回数が決まっている場合に利用



※数える数は負数でもOK

for文のよくある使い方(1)

◆ 多くの数の足し算・引き算・掛け算・割り算

◆ $1 + 2 + 3 + 4 + 5 + \dots$

◆ $10 * 10 * 10 * 10 * \dots$

◆ etc.

- 数の個数が決まっている
- すべての数を同じ方法で計算する
 - ✓ Ex. すべての数を足し算する、など
 - ✓ 足し算したり掛け算したり、違う方法の計算はしない

➡ for文を使う

for文のよくある使い方(2)

◆ for分を使った計算のテンプレート

計算結果を入れる変数

初期化(どのような数で初期化するかは
計算方法によって違う)

数の個数

```
result = ▲;  
for(i = 1; i <= ●; i = i + 1) {  
    result = result ?? ■;  
}
```

オペランド(足し算/引き算/
掛け算/割り算)

1つ1つの数

for文の使い方例(1)

- ◆ 0から49まで順に数えたい場合

```
for(i = 0; i < 50; i = i + 1)
```

- ◆ 0から100までの数のうち、偶数のみを扱う場合

```
for(i = 0; i < 100; i = i + 2)
```

- ◆ 49から0まで順に数えたい場合

```
for(i = 49; i > 0; i = i - 1)
```

※「i」はint型の変数

(i, jは、ループ文で数を数えるための変数としてよく使われる)

for文の使い方例(2)

- ◆ 1から50までの数を足し合わせた数を求める

```
result = 0;  
for(i = 1; i <= 50; i = i + 1) {  
    result = result + i;  
}
```

「result=result+i」は、この式の前までの「result」の値に「i」を足し、「result」の値を新しくする

iの値が1の時: resultの値は $0 + 1$ になる
iの値が2の時: resultの値は $0 + 1 + 2$ になる
iの値が3の時: resultの値は $0 + 1 + 2 + 3$ になる
.....
iの値が50の時: resultの値は $0 + 1 + 2 + \dots + 50$ になる
iの値が51の時: ループを終わる

while文

主に、繰り返す数が決まっていないときに利用

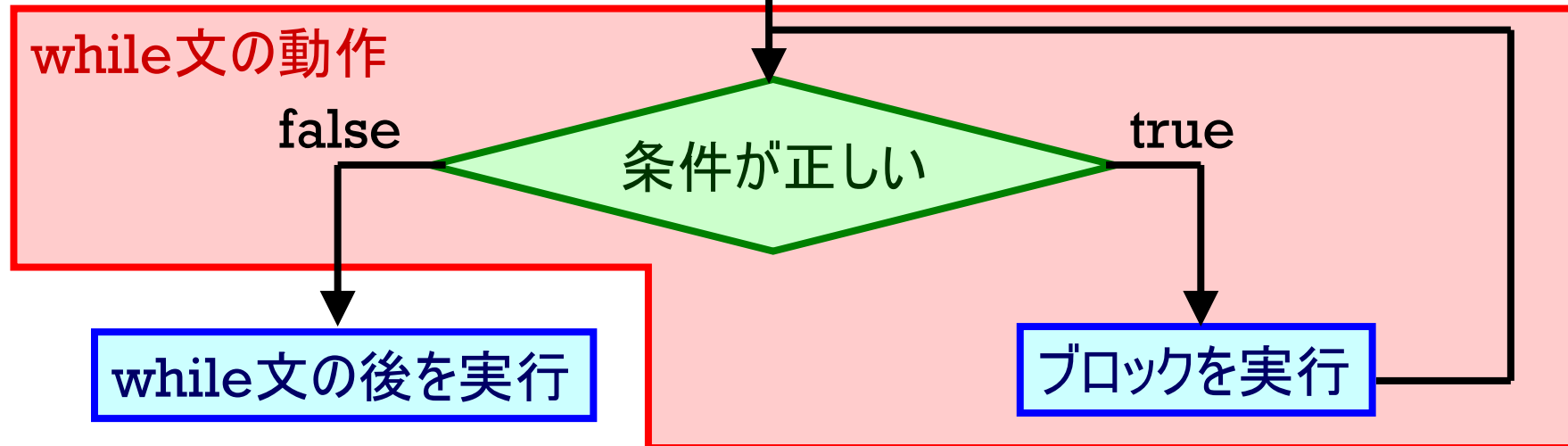
while文の書き方と動作

```
while(条件) {  
    文1;  
    文2;  
    .....  
}
```

ブロック

条件の書き方はif文とまったく同じ
(true/falseで結果が出る条件)

while文の前までの処理を実行



while文の使い方例(1)

- ◆ 1から50までの数を足し合わせた数を求める

```
i = 1;  
result = 0;  
while (i <= 50) {  
    result = result + i;  
    i = i + 1;  
}
```

iの値が1の時: resultの値は $0 + 1$ になる

iの値が2の時: resultの値は $0 + 1 + 2$ になる

iの値が3の時: resultの値は $0 + 1 + 2 + 3$ になる

.....

iの値が50の時: resultの値は $0 + 1 + 2 + \dots + 50$ になる

iの値が51の時: ループを終わる

※この処理は、for文の例をwhile文に直したもの

while文の使い方例(2)

- ◆ 標準入力から文字列を入力し、その文字列の長さを標準出力で出力する
 - ◆ 「End」と入力されれば終了

```
BufferedReader br =  
    new BufferedReader(new InputStreamReader(System.in));  
str = br.readLine();  
int len;  
while (!str.equals("End")) {  
    len = str.length();  
    System.out.println("入力された文字列の長さ: " + len);  
    str = br.readLine();  
}
```

※この処理は、for文ではあまり書かない

同じ種類の変数をたくさん使う～配列～

「配列」って？

- ◆ データ型が同じで、処理方法も同じ変数をたくさん扱うときに
利用する変数

- ◆ たくさんの変数を1度にまとめて宣言する方法

⇒ 変数に番号をつけて扱う

例えば...英語の成績を表す変数を「english」

出席番号1番の生徒の成績: english[1]

出席番号2番の生徒の成績: english[2]

....

出席番号50番の生徒の成績: english[50]

⇒ 宣言する変数は「english」だけ

配列の宣言のしかた

1. 変数の名前(配列変数と呼ぶ)を決める
2. 扱う値の個数を決める
3. 配列を宣言する

書き方その1

```
データ型 変数名[];  
変数名 = new データ型[個数];
```

「[]」で、配列を宣言
するという意味

扱う値の個数を決める

書き方その2

```
データ型[] 変数名 = new データ型[個数];
```

書き方その3

```
データ型 変数名[] = new データ型[個数];
```

※書き方その1, 2, 3のどれで宣言してもかまわない

配列の使い方

- ◆ 「[]」の中に値の**番号**を書き、宣言した変数の後ろにつけて使う
 - ◆ 番号を付けた1つ1つの変数を、配列の「要素」と呼ぶ
 - ◆ 配列の要素が、通常の変数に相当

「添え字」と呼ぶ
※番号は0から数える

例えば...英語の成績を表す配列変数「english」の要素に点数を代入
出席番号1番の生徒の成績: `english[0] = 80;`
出席番号2番の生徒の成績: `english[1] = 50;`
.... 略
出席番号50番の生徒の成績: `english[49] = 75;`

配列の初期化(1)

- ◆ 最初に宣言をした後、要素に1つ1つ値を代入していく方法

最初に変数名と個数を宣言し、
1つ1つ値を代入

```
int english[ ] = new int[50];  
english[0] = 80;  
english[1] = 75;  
....  
english[49] = 50
```

※初期化: 変数を宣言した後、初めて値を代入すること

配列の初期化(2)

◆ 宣言と同時に要素に値を代入する方法

```
int english[ ] = {80, 75, ..., 50};
```

「,」(コンマ)で値を区切って、左側の値から順に0番、1番、2番...に代入される
※この場合、配列の個数は、値の個数になる(配列の個数の宣言は不要)

※初期化: 変数を宣言した後、初めて値を代入すること

配列の使い方例(1)

◆ 複数の配列で、要素の添え字が同じものを、1まとまりとして扱う

出席番号1番の生徒の得点
= 添え字が0番の要素

language[0]

math[0]

english[0]



science[0]

society[0]

出席番号2番の生徒の得点
= 添え字が1番の要素

language[1]

math[1]

english[1]

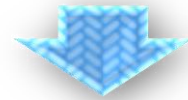


science[1]

society[1]

ループ文との組み合わせ

- ◆ 配列は、複数の同じデータ型のデータを扱うもの
 - ◆ 複数のデータに対して同じ処理を行う
- ◆ 配列は、添え字という番号をつけて管理
 - ◆ 添え字は、0, 1, 2, ...というように0番から順になっている



- ループ文と組み合わせて、1つ1つの配列の要素を処理することが多い
- ループ文の処理として、i番目の要素に対する処理を定義しておく
 - ✓ iは、ループ文で数を数えるための変数
 - ループ文で、添え字を0から順に(または最後から順に)数える

for文との組み合わせ

◆ 50人の生徒の英語の平均点を求める場合

```
sum = 0;
for(i = 0; i < 50; i = i + 1) {
    sum = sum + english[i];
}
average = sum / 50;
```

配列のi番目の要素に対する処理

iの値が0の時: sumの値は $0 + \text{english}[0]$ になる

iの値が1の時: sumの値は $0 + \text{english}[0] + \text{english}[1]$ になる

iの値が2の時: sumの値は $0 + \text{english}[0] + \text{english}[1] + \text{english}[2]$ になる

.....

iの値が49の時: sumの値は $0 + \text{english}[0] + \text{english}[1] + \dots + \text{english}[49]$ になる

iの値が50の時: ループを終わって、次の処理(平均を計算)をする

練習問題

- ◆ 標準入力から数を5つ入力し、その数の平均を求めるプログラム
- ◆ 1から10までの数を全てかけあわせた数を求めるプログラム
 - ◆ 「 $1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times 9 \times 10$ 」
- ◆ 5つの配列に友達の名前を代入しておく。そして標準入力から文字を1つ入力し、その文字で始まる友達の名前を出力するプログラム
 - ◆ 友達の名前が存在しなければ、「なし」と出力する

出席確認

- ◆ 授業のページから、本日の出席確認の復習問題に解答
 - ◆ 授業のページ: <http://www.cis.twcu.ac.jp/~junko/Programming/>
- ◆ 今後: 授業開始前に、授業のページから、前回授業の復習問題に解答
 - ◆ 13:30までに解答すれば正規出席扱い
 - ◆ 13:30を過ぎて解答すれば遅刻扱い