Javaプログラミング（第2回）

成蹊大学理工学部
情報科学科

基本型の種類

- boolean：論理型(trueまたはfalse)
  boolean b = false;
- char:整数型（文字型），符号なし16ビット
- byte:整数型，符号付き8ビット
- short：整数型，符号付き16ビット
- int:整数型，符号付き32ビット
- long:整数型，符号付64ビット
- float:単精度浮動小数点型
- double:倍精度浮動小数点数型
  double d = 15.0;

整数型と浮動少数点数型

整数型として宣言

```
int x = 15;
int y = 10;
System.out.println((x+y)/2);
```

→　12（小数点以下が切り捨てられる）

浮動小数点数型として宣言

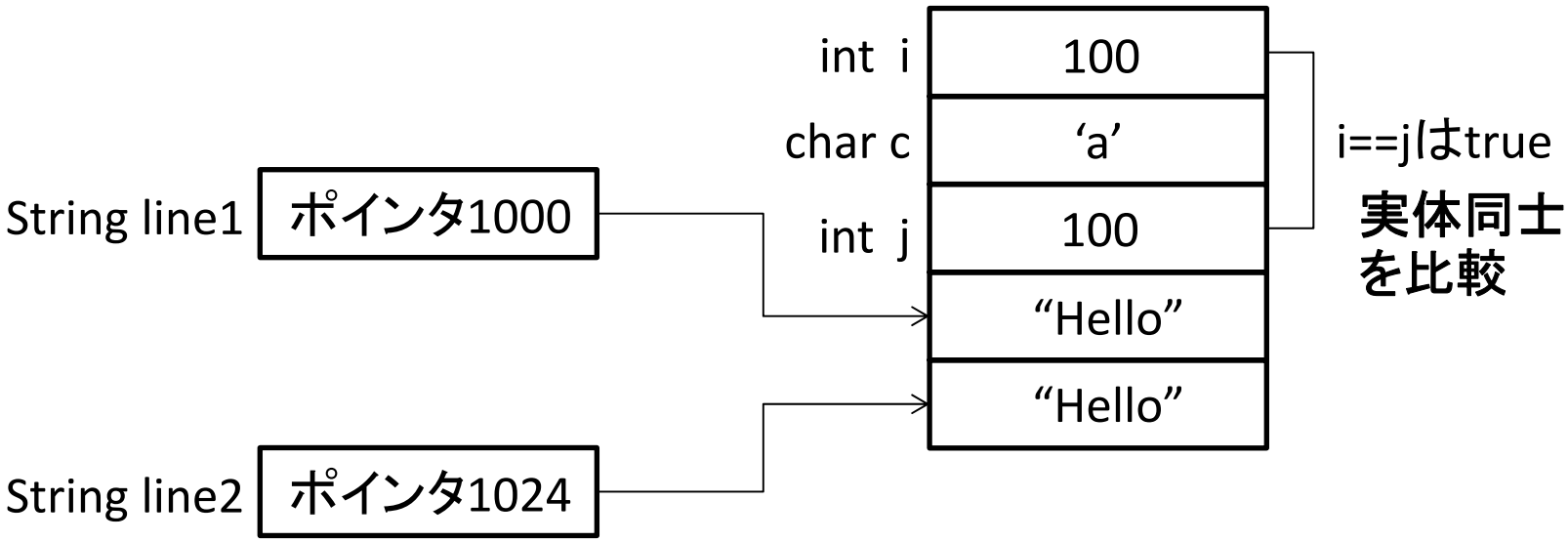
```
double x = 15.0; (15と書いてもよい)
double y = 10.0;
System.out.println((x+y)/2);
```

→12.5（小数点以下が切り捨てられない）

基本型と参照型

- 基本型 (primitive type)：論理値と数値
  - メモリ上に領域が確保され，具体的な値（リテラル）が代入されている
- 参照型 (reference type)：クラス，インタフェース，配列
  - メモリ上の領域には，別の領域に存在する実体へのポインタ（識別する為の ID）が代入されている
  - 文字列あらわすStringはクラスなので，参照型となる
    String型の初期化
    String str = new String(“こんにちは”);

基本型と参照型(Cont.)



line1==line2はfalse!!

ポインタ同士が比較される

ポインタが指している中身を比較すべき

line1.equals(line2)とすればtrue!!

ポインタの指している実体が比較される

自動型変換

- 型の大きさ
  byte -> short -> int -> long -> float -> double
  　　char ->
  – 変数を別の型に代入するときには，型変換が必要
- 拡張型変換：小さい型を大きい型に代入するときは自動的に型変換される
  – char 型は int, long, float, double型に代入可能

```
int i = 10;
float f = i; // int 型は float 型に代入可能
char c = 'a';
int i = c; // char 型は int 型に代入可能
```

演算時の自動型変換

- 一つの式中に異なる型が混在する場合，自動的な型変換が起こる
  - 小さい型は大きい型に合わせて解釈される
  - 整数よりも浮動小数点数が優先される
  - char, byte, short は int 型に合わせられる

```
int i = 100;
double d = 3.14;
System.out.println(i * d);
// (int × double) の結果は double と解釈される
```

演算時の自動型変換 (Cont.)

1988*105/100を正しく計算するプログラムは?

```
int   i = 1988;
double  d1 = i * 105/100;
double  d2 = i * 1.05; // => 正しい結果
double  d3 = i / 100 * 105;
```

キャスト

- 縮小型変換：大きい型を小さい型に代入するときは，演算精度の低下が起こるため、自動型変換はされない
- 明示的な型変換であるキャストが必要

```
double  d=3.1415;
int  i=10;
i= (int) d; // i=d; はエラー！
```

メソッドの定義

- mainメソッドに全ての処理を書くのではなく，ある程度処理をまとめて，別のメソッドを作る

```
public class Discount {
   public static void main (String[] args) {
     int p, q;
     p=10000;
     q=halve(p);   //pを引数としてメソッドの呼び出し
     System.out.println("元の値段が"+p+"円なら，半額だと"+q+"円です．");
   }

   public static  int   halve    (int n) {
     return n/2;   //nを2で割った答えを戻り値とする
   }
}
```

halveという名前のメソッド

戻り値の型がint

int型の引数を1つ取る

戻り値を示す

メソッドの定義 (Cont.)

- 戻り値のないメソッド

```
public static void myPrint (int x) {
    System.out.println(x); //プリントするだけで，戻り値はない
}
```

- 引数の数が複数になる場合

int x, yのように略さない

```
public static void myDate (int x, int y) {
    System.out.println(“今日は”+x+”月”+y+”日です．”);
}
```

メソッド名のつけ方

- 数字を名前の最初に使ってはいけない
  - 1234method　×
- _（アンダーバー）は使えるが，-（ハイフン）は使ってはならない
  - my_method　○　my-method　×
- 予約語を使ってはならない
  - public　×
- 一般的な方法
  - 小文字で始める
  - 単語の区切りを大文字か_にする.
    - myMethod, my_method

制御構造

- 様々な制御構造について学ぶ
  - if
  - for
  - while (次週)
  - Switch(次週）

if 文

```
if (条件式1）{
    条件式1を満たす場合
}  else if (条件式2) {
    条件式1は満たさず，条件式2を満たす場合
} else {
    条件式1，2ともに満たさない場合
}
```

（例）

```
if (p>=50）{
    System.out.println(“傘を持っていきます”);
}    else if (p>=30) {
     System.out.println(“できれば傘を持っていきます”);
} else  {
    System.out.println(“傘を持っていきません”);
}
```

「または」を条件にする

- 2つの条件式を”|| “で連結させる

```
if (条件式1　||　条件式2)　{
    処理A
} else {
    処理B
}
```

（例）

```
if (p>100 || 0>p）{
    System.out.println(“降水確率は0～100の間です”);
}
```

「かつ」を条件にする

- 2つの条件式を”&&“で連結させる

```
if (条件式1　&&　条件式2)　{
    処理A
} else {
    処理B
}
```

（例）

```
if (p>10 && p<30）{
    System.out.println(“傘は持っていかなくてもよいでしょう”);
}
```

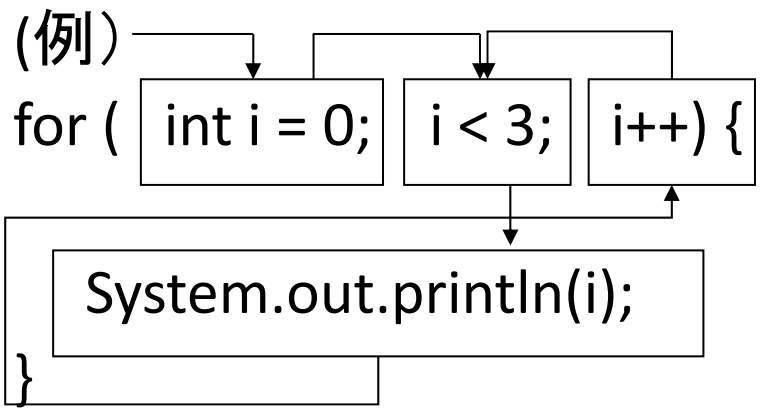
for 文

- 条件を満たす間同じ処理を繰り返す

```
for (初期化; 条件式; 次の一歩) {
    繰り返す処理
}
```

・初期化 （例）変数iをint型で宣言し，0で初期化
・条件式：繰り返しを続ける条件
・次の一歩：処理が1回終わった後に実行される．次の状態に進める．

for 文の処理の流れ



(1) 変数 i を初期化する
(2) 条件 i<3 を調べる
(3) 条件を満たしていれば，処理System.out.println(i);を行う
(4) 変数 i に1を加える

二重のfor 文

- 外側と内側で二重に繰り返し処理を行いことができる

```
for (初期化1; 条件式1; 次の一歩1) {
    処理1
    for (初期化2; 条件式2; 次の一歩2)  {
        処理2
    }
}
```

二重for 文の例

```
for (int i=0;i<10;i++) {
        System.out.print(i);
        for (int j=0;j<i;j++) {
            System.out.print("*");
        }
        System.out.print("¥n");
    }
```

・System.out.println→最後に改行がつく
・System.out.print→最後に改行がつかない
・改行記号　¥n を用いて改行を行うこともできる

変数の有効範囲（スコープ）

- 変数には有効範囲がある
  - for文では，その繰り返しの制御文内でのみ有効
  - 変数をfor文の外でも参照したい場合には，for 文の外で変数を宣言する
  - method内で定義された変数は，そのメソッド内でのみ有効

配列

形式	意味	例
型名[] 配列名;	配列の宣言	int[] score;
new 型名[要素の数];	配列の確保	score = new int[3];
配列名[添え字] = 値;	配列要素への代入	score[0] = 63;
配列の型[] = {要素0, 要素1, 要素2...};	配列の初期化	int[] score = {63, 92, 75};
配列名 = new 配列の型[] {要素0, 要素1, 要素2...};	配列にまとめて代入	score = new int[] {63, 92, 75}; (正) score = {63, 92, 75}; (誤)
配列名[添え字]	配列要素の参照	System.out.println(score[0]);

科目の平均値を求める

```
1: public class Average {
2:     public static void main(String[] args) {
3:           double average = 0.0;
4:           average = calAverage();
5:           System.out.println("3教科の平均点は "+average);
6:     }
7:. . .
```

配列による表現(1)

```
8:     public static double calAverage() {
9:             double result = 0.0;
10:            int[] score; //配列の宣言
11:            score = new int[3];  //配列の確保
12:            score[0] = 63; //配列要素への代入
13:            score[1] = 92;
14:            score[2] = 75;
15:            result = (score[0]+score[1]+score[2])/3.0;
16:            return result;
17:      }
```

配列による表現(2)

- 添え字を変数にすることもできる

```
8:    public static double calAverage() {
9:          double result = 0.0;
10:         int[] score = {63, 92, 75};
11:         int sum=0;
12:         for (int i=0;i<3;i++) {
13:               sum = sum+score[i];
14:         }
15:         result = sum/3.0;
16:         return result;
18:    }
```

配列の長さを表すlength

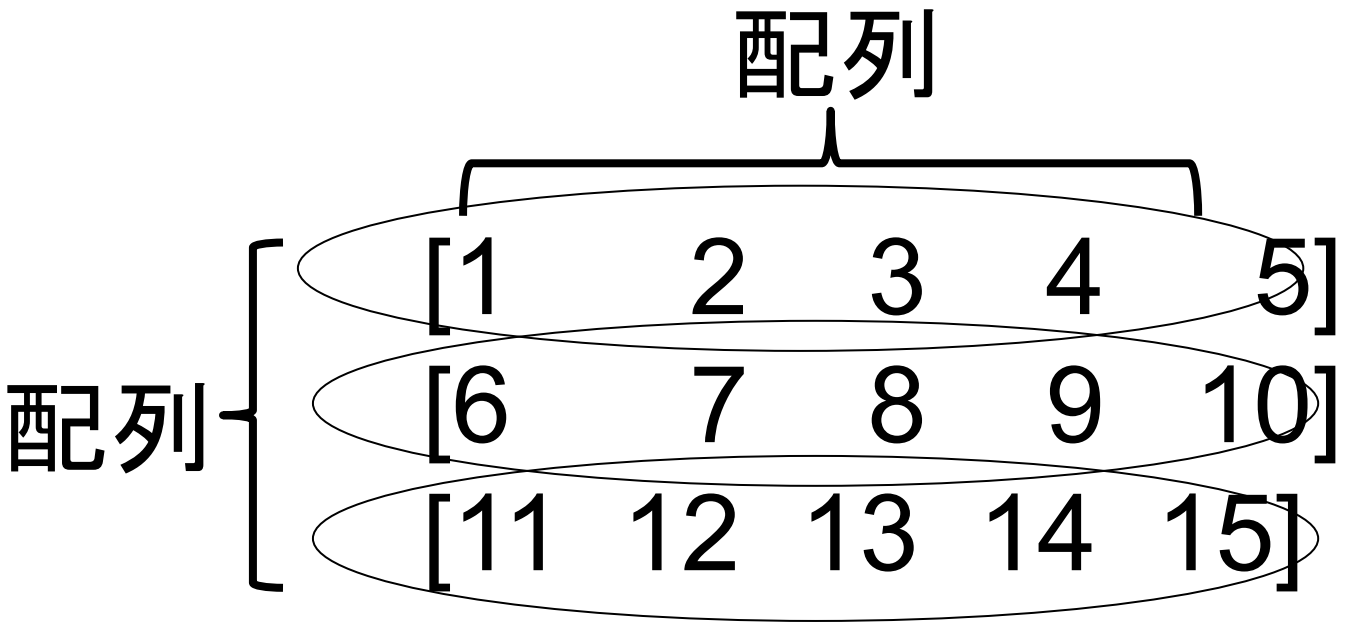
形式	意味	例
配列.length	配列の長さ	score.length

```
8:     public static double calAverage() {
9:           double result = 0.0;
10:          int[] score = {63, 92, 75};
11:          int sum=0;
12:          for (int i=0;i<score.length;i++) {
13:                sum = sum+score[i];
14:          }
15:          result = sum/3.0;
16:          return result;
18:     }
```

二次元配列

- 配列の配列を作ることができる

形式	意味	例
型名[][] 配列名;	二次元配列の宣言	int[][] score;



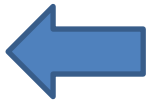
二次元配列

- 二次元配列のlengthは配列の中に配列がいくつ入っているかを表す
  - 二次元配列→score
  - score.length→score配列の中にいくつ配列があるか
  - score[0].length→最初の配列の中にいくつ要素があるか

二次元配列の例

- 4月からのカレンダーを印刷する

```
public class DoubleArray {
        public static void main(String[] args) {
        int[][] apr = {
        {26,27,28,29,30,1,2},
        {3,4,5,6,7,8,9},
        {10,11,12,13,14,15,16}
        };
        printCalender(apr);
        }
```



二次元配列

二次元配列の例

```
static void printCalender(int[][] month) {
        System.out.println("日¥t月¥t火¥t水¥t木¥t金¥t土");
        for (int i=0;i<month.length;i++) {
                int[] week= month[i];
                for (int j=0;j<week.length;j++) {
                        System.out.print(week[j]+"¥t");
                }
        System.out.println("¥n");
        }
}
```

出力例

日	月	火	水	木	金	土
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16

実行時の引数として変数を渡す方法

- 実行時に引数を指定
  > java クラスファイル名　引数1　引数2　…
- main関数でこれを受けとって処理する方法

```
public class VariableInput {
  public static void main (String[] args) {
    String variable = new String();  //String型の変数を初期化
    variable = args[0]; //最初の引数をvariableに代入
    System.out.println("代入された文字列は"+variable+"です。");
  }
}
```

★実行時に引数を与えると，1番目の引数はargs[0]，2番目の引数はargs[1]のよう自動的にargsの配列に順番に代入されてゆく

実行例

```
> java  VariableInput   123
代入された文字列は123です。
```

Stringをint型に変換する方法

- Stringをint型に変換するには，通常のキャストは異なる方法を用いる.

```
String a = new String (“123”);  //文字列123を生成
int p=0;  int型の変数pを初期化
p = Integer.parseInt(a);  //文字列”123”をint型123に変換

double q = 0.0;  //double型の変数qを初期化
q = Double.parseDouble(a);  //文字列”123”をdouble型123.0に変換
```