

プログラミング基礎 I

第11回 メソッド応用

九州産業大学 理工学部

pk@is.kyusan-u.ac.jp

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・・・
 - 表示が変わるタイミングでは必ず実行して確認すること！！！！
- インデントに注意！
 - **インデントが崩れたら、Ctrl + A → Ctrl + I**
- エラーが出たら、エラーの内容を確認してみる
 - まずはエラーの内容を見て、自分で解決できないか考えてみよう
 - それでも分からなければ遠慮なく質問しましょう

今回の重要点

- メソッド
 - 処理をまとめて名前をつけたもの
- メソッド宣言：メソッドを作る

```
返回值 メソッドの名前(引数) {  
    処理を記述;  
    return 返回值;  
}
```

引数：メソッドを呼び出すときにメソッドに渡す値

- メソッド呼び出し：宣言したメソッドを使う
変数 = メソッド名(引数); // 返回值を受け取る

引数と返り値

- 引数：メソッドに入力する値
- 返り値：メソッドから返される値

メソッドに何を入力すると何が出力されるのか

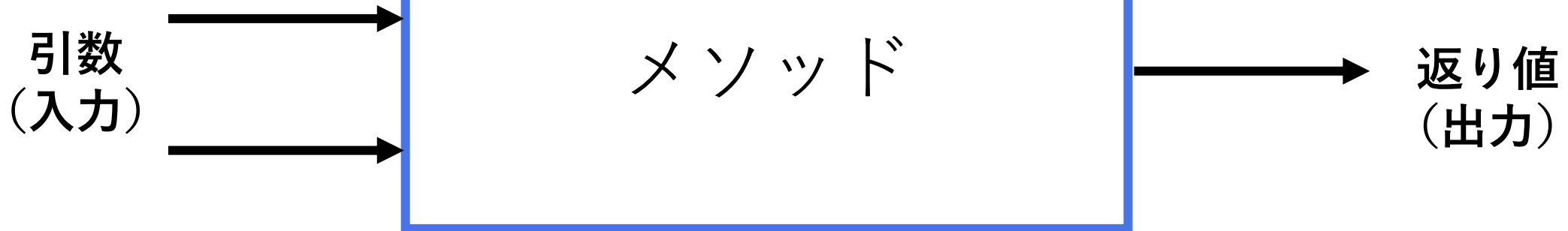
メソッド呼び出しでは、入出力の関係を理解することが重要



メソッドを自分で作ることが出来る

- メソッドは自分で作ることも出来る
 - メソッドを作ること：メソッド宣言

今回は、メソッドの中身を作っていく



同じような処理を何度も記述するケース

- 例) 区切り線を表示する

```
//for文で20回繰り返し、- を20個連結して表示<
for(int i = 0; i < 20; i++) {<
    System.out.print("-");<
}<
System.out.println();<
```

- 例) 配列の全要素を表示する

```
for(int i = 0; i < scores.length; i++) {<
    System.out.print(scores[i] + ", ");<
}<
System.out.println();<
```

同じような処理を何度も記述するケース

- 配列内の全要素の合計値を計算

```
int sum = 0;
for(int i = 0; i < scores.length; i++) {
    sum += scores[i];
}
System.out.println(sum);
```

同じような処理を何度も記述するケース

- 例) クラス型変数のインスタンスのフィールドを表示

```
//15.student1, student2 のフィールドを、下記のように表示<
String tarouInfo = "";<
tarouInfo += student1.studentNumber;<
tarouInfo += " " + student1.person.name + "の";<
tarouInfo += student1.subject1.name + "の教室:";<
tarouInfo += student1.subject1.className;<
System.out.println(tarouInfo);<
<
String student2Info = "";<
student2Info += student2.studentNumber;<
student2Info += " " + student2.person.name + "の";<
student2Info += student2.subject1.name + "の教室:";<
student2Info += student2.subject1.className;<
System.out.println(student2Info);<
<
math.className = "OA教室1";<
<
tarouInfo = "";<
tarouInfo += student1.studentNumber;<
tarouInfo += " " + student1.person.name + "の";<
tarouInfo += student1.subject1.name + "の教室:";<
tarouInfo += student1.subject1.className;<
System.out.println(tarouInfo);<
<
student2Info = "";<
student2Info += student2.studentNumber;<
student2Info += " " + student2.person.name + "の";<
student2Info += student2.subject1.name + "の教室:";<
student2Info += student2.subject1.className;<
System.out.println(student2Info);<
```

同じ様な処理を何度も記述する

- 面倒だと思ったことはないですか？
 - 変数名、配列名が違うだけの処理を、何度も何度も書かないといけないの？
- コピー＆ペーストミスをしたことはありませんか？
 - 同じ様な処理だから、コピーして貼り付けて、変数名を変えれば動く。楽をしよう！！
 - 間違えた・・・

メソッドを使えると楽

- メソッド呼び出し、楽だなと思いませんでしたか？
 - 最大値を求める処理は、 `Math.max` を呼び出すだけ
 - 絶対値を求める処理は、 `Math.abs` を呼び出すだけ
- メソッド呼び出しなら、
引数を入れて、返り値を受け取るだけ
 - 何度も同じことをする場合には、
メソッドを使ったほうが圧倒的に楽

例) 絶対値を求める処理

- 変数 a,b,c,d,e を絶対値にせよ

```
int a = sc.nextInt(); int b = sc.nextInt(); int c = sc.nextInt(); int d = sc.nextInt(); int
e = sc.nextInt();
// aの絶対値を計算
if(a < 0) {
    a *= -1;
}
// bの絶対値を計算
if(b < 0) {
    b *= -1;
}
// cの絶対値を計算
if(c < 0) {
    c *= -1;
}
// dの絶対値を計算
if(d < 0) {
    d *= -1;
}
// eの絶対値を計算
if(e < 0) {
    e *= -1;
}
```

例) 絶対値を求める処理(メソッド利用)

- 何回も同じ処理を記述する場合は、メソッドを利用出来ると短く記述できる

```
int a = sc.nextInt(); int b = sc.nextInt(); int c = sc.nextInt();  
int d = sc.nextInt(); int e = sc.nextInt();  
// aの絶対値を計算  
a = Math.abs(a);  
// bの絶対値を計算  
b = Math.abs(b);  
// cの絶対値を計算  
c = Math.abs(c);  
// dの絶対値を計算  
d = Math.abs(d);  
// eの絶対値を計算  
e = Math.abs(e);
```

この授業で扱うメソッドの種類

- **インスタンスメソッド**

- クラスから生成したインスタンスを使って呼び出すメソッド
- この科目では、主にインスタンスメソッドの扱いについて学ぶ
- プログラミング基礎 II ですぐにインスタンスメソッドを学ぶため
 - 他のメソッド（クラスメソッド，staticメソッド）については、インスタンスメソッドを修得してから覚えよう

インスタンスメソッドの宣言

- インスタンスメソッドの宣言

```
class クラス名 {
```

```
    戻り値 メソッド名(引数の型 引数名1, 引数の型 引数名2, ...){
```

```
        処理;
```

```
        return 戻り値;
```

```
    }
```

```
}
```

```
class MyMathClass {
```

```
    int abs(int a) {
```

```
        if(a < 0) {
```

```
            a *= -1;
```

```
        }
```

```
        return a;
```

```
    }
```

```
}
```

引数と返り値

- **引数**：メソッドに渡す値
 - メソッドを作る際には、**メソッド外から与えられる値**
 - 引数はいくつあってもいい、無くてもいい
- **返り値**：メソッドから返される値のこと
 - **呼び出した側に返す値**
 - `int`, `double`, `String`, クラス型 等
 - 返り値がないメソッドもある (`void`メソッド)

インスタンスメソッドの呼び出し

- インスタンス名.メソッド名(引数1,...);

```
MyMathClass myMath = new MyMathClass();  
System.out.println(myMath.abs(-5));
```

- フィールドへのアクセスと同じように、
. (ピリオド) で繋いで呼び出すことが出来る

インスタンスメソッドを作ってみよう

- クラスを宣言
 - インスタンスメソッドを扱うには、まずクラスが必要

```
1 class MyMathClass {  
2       
3       
4       
5 }  
6 public class Main11p1_25RS999 {  
7       
8     public static void main(String[] args) {  
9         // TODO 自動生成されたメソッド・スタブ  
10      
11    }  
12   
13 }
```

インスタンスメソッドを宣言

- MyMathClass 内に、下記のメソッドを宣言
受け取った引数の絶対値を返すメソッド

メソッド名：abs

引数：int型の a

返り値：int型

```
1 class MyMathClass {  
2     int abs(int a) {  
3         if(a < 0) {  
4             a *= -1;  
5         }  
6         return a;  
7     }  
8 }
```

return; 返り値を返す

- return; は値を返す命令
- 返り値があるメソッドは、必ず値を返さなければならない
 - つまり、必ず return が必要

どの書き方でも結果は同じ

```
int abs(int a) {  
    if(a < 0) {  
        a *= -1;  
    }  
    return a;  
}
```

引数自体を上書きして返す
渡された値が変わっても問題ない場合はこれでOK

```
int abs(int a) {  
    int returnVal = a;  
    if(a < 0) {  
        returnVal *= -1;  
    }  
    return returnVal;  
}
```

return する用の変数を宣言しておき、
その変数に返り値を代入、
最後にreturnする

返す値が分かりやすい

```
int abs(int a) {  
    if(a < 0) {  
        return -a;  
    }  
    return a;  
}
```

条件に応じてreturnする
(returnは複数行記述しても良い)

return以後のプログラムは実行されない点に注意

インスタンスメソッドを呼び出す

- MyMathClass型の変数を宣言、インスタンスを生成して代入

```
9 public class Main11p1_25RS999 {  
10     public static void main(String[] args) {  
11         MyMathClass myMath = new MyMathClass();  
12     }  
13 }  
14  
15 }
```

- myMathのabsを呼び出し

```
9 public class Main11p1_25RS999 {  
10     public static void main(String[] args) {  
11         MyMathClass myMath = new MyMathClass();  
12           
13         System.out.println(myMath.abs(-8));  
14     }  
15 }  
16 }
```

メソッドの返り値受け取り

```
MyMathClass myMath = new MyMathClass();
```

```
// メソッド呼び出しの入れ子で表示  
System.out.println(myMath.abs(-8));
```

```
// 変数に返り値を格納してから表示  
int absTest = myMath.abs(-8);  
System.out.println(absTest);
```

インスタンス変数は使いまわしていい

- どちらでも構わないが、
後で返り値を利用するなら、変数に格納しておいたほうがいい

別のメソッドを追加する

- メソッドは複数宣言できる

```
1 class MyMathClass {  
2     int abs(int a) {  
3         if(a < 0) {  
4             a *= -1;  
5         }  
6         return a;  
7     }  
8     int max(int a, int b) {  
9         if(a > b) {  
10            return a;  
11        } else {  
12            return b;  
13        }  
14    }  
15 }
```

1つ目のメソッド

2つ目のメソッド

メソッドの中にメソッドを宣言しないように注意

Math.maxメソッド呼び出し

- 引数が複数あるメソッドは、カンマで区切って入力

```
int a = -8;  
int b = 6;  
int maxAb = myMath.max(a, b);  
System.out.println(maxAb);
```

Mathクラスにないメソッドを作成してみよう

- Math.floorメソッド: 小数点以下を切り捨てて返すメソッド
 - 小数点何位以下を切り捨て、は工夫が必要

```
double pi = 3.1415926535;  
double p = Math.pow(10, 3);  
double f = Math.floor(pi * p)/p;  
System.out.print(f);  
//一度,1000倍して,小数点を切り捨てて,1000で割る
```

- 指定桁数以下の小数点を切り捨てるメソッドは、Javaの標準機能では存在しない

指定桁数以下の小数点を切り捨てるメソッド

- メソッド名：floorDigit

引数：double型の val, int型の digit

返り値：double型

引数が複数の場合、 , で区切って指定する

```
double floorDigit(double val, int digit) {  
    double p = Math.pow(10, digit-1);  
    double returnVal = Math.floor(val * p)/p;  
    return returnVal;  
}
```

floorDigitメソッド呼び出し

- 引数の入力順に注意
 - 引数：double型の val, int型の digit

```
double pi = 3.1415926535;  
System.out.println(myMath.floorDigit(pi, 4));
```

出力結果：3.141

- 引数の順番を間違えると・・・

```
double pi = 3.1415926535; ◀  
System.out.println(myMath.floorDigit(4, pi)); ◀
```

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
 型 MyMathClass のメソッド floorDigit(double, int) は引数 (int, double) に適用できません

at Main11p1_25RS999.main(Main11p1_25RS999.java:56)

引数の順番に注意！

- 引数は、
宣言部と呼び出し部で同じ入力順でなければならない
- メソッド呼び出しをする際には、
必ず、**宣言部の引数の順番を確認**すること

参照型を引数や返り値にしたメソッド

- 参照型
 - 配列、クラス(自分で宣言するものも含む)、String型 (Stringも実は参照型)
 - 参照型も引数や返り値に指定できる

参照型を引数に持つメソッド例

- メソッド名：getArrayString

引数：int型の 配列 array

返り値：String型

- 引数で受け取った配列の 全要素を, で繋いだ文字列を返す

```
String getArrayString(int [] array) {  
    String returnStr = "";  
    for(int i = 0; i < array.length; i++) {  
        returnStr += array[i] + ", ";  
    }  
    return returnStr;  
}
```

配列を扱うメソッドをMyMathクラスに追加

```
class MyMathClass {  
    String getArrayString(int [] array) {  
        String returnStr = "";  
        for(int i = 0; i < array.length; i++) {  
            returnStr += array[i] + ", ";  
        }  
        return returnStr;  
    }  
}
```

呼び出し (getArrayString)

```
int [] testArray = {1,-2,3,-4,5};  
System.out.println("testArray(絶対値変換前の配列)");  
System.out.println(myMath.getArrayString(testArray));
```

メソッドに渡すときには、配列名だけを記述する
よくある間違い：testArray[0] のように要素を渡そうとしている
(testArray[0] は int型、getArrayStringの引数は **int型の配列**)

String型の値が返るので、
System.out.println() に入れ子にして表示出来る

参照型を引数にする際の注意（参照渡し）

- 参照型
 - 配列、クラス(自分で宣言するものも含む)、String型（Stringも実は参照型）
- 参照型を引数としてメソッドに渡す場合、引数の値を変更すると、呼び出し側にも影響が出る

例) 配列の全要素を絶対値にして返す

- メソッド名 : `getAbsArray`

引数 : `int`型の配列 `array`

返り値 : `int`型配列

```
int [] getAbsArray(int [] array) {  
    //ひとまず宣言部のみ記載  
  
}
```

配列を返り値にする場合は、
型 [] 名前(引数の型 名前) の形

間違い例

- 引数の配列に直接代入すると・・・

```
int [] getAbsArray(int [] array) {  
    for(int i = 0; i < array.length; i++) {  
        if(array[i] < 0) { // 0未満なら  
            array[i] *= -1; // -1倍して代入 (符号反転)  
        }  
    }  
    return array; // 引数のarrayを返す  
}
```

```
class MyMathClass {  
    int [] getAbsArray(int [] array) {  
        for(int i = 0; i < array.length; i++) {  
            if(array[i] < 0) {  
                array[i] *= -1;  
            }  
        }  
        return array;  
    }  
}
```

MyMathClass のメソッドに追加

呼び出してみると・・・

```
int [] testArray = {1,-2,3,-4,5};
System.out.println("testArray(絶対値変換前の配列)");
System.out.println(myMath.getArrayString(testArray));

int [] absTestArray = myMath.getAbsArray(testArray);

System.out.println("absTestArray(絶対値変換後の配列)");
System.out.println(myMath.getArrayString(absTestArray));

System.out.println("testArray(絶対値変換前の配列)");
System.out.println(myMath.getArrayString(testArray));
```

```
testArray(絶対値変換前の配列)
1, -2, 3, -4, 5,
absTestArray(絶対値変換後の配列)
1, 2, 3, 4, 5,
testArray(絶対値変換前の配列)
1, 2, 3, 4, 5,
```

コンソール

```
testArray(絶対値変換前の配列)
1, -2, 3, -4, 5,
```

```
absTestArrayに絶対値に変換した配列を代入
(testArrayは変更していないつもり)
```

```
absTestArray(絶対値変換後の配列)
1, 2, 3, 4, 5,
```

```
testArray(絶対値変換前の配列)
1, 2, 3, 4, 5,
```

**testArrayの要素の値も
変わってしまった！**

参照渡し

- 参照型の変数をメソッドの引数として渡すことを、参照渡し という
 - 参照渡しをする場合には、引数の値を変更しないように注意

```
int [] getAbsArray(int [] array) {  
    // returnする用の、同じ要素数の配列を新しく生成  
    int [] returnArray = new int[array.length];  
    for(int i = 0; i < returnArray.length; i++) {  
        if(array[i] < 0) {  
            returnArray[i] = array[i] * -1;  
        } else {  
            returnArray[i] = array[i];  
        }  
    }  
    return returnArray; // 新しく生成した配列を返す  
}
```

引数には代入しない

参照型変数を扱う場合の注意

```
int [] arrayA = {1, 2, 3, 4, 5};
int [] arrayB = arrayA; // arrayAにarrayBをコピー
System.out.print("arrayAを出力 : ");
for(int i: arrayA) {
    System.out.print(i);
}
System.out.println();

System.out.print("arrayBを出力 : ");
for(int i: arrayB) {
    System.out.print(i);
}
System.out.println();

arrayA[0] = 9999; // arrayAの要素を変更(ArrayBは変更していない)
System.out.print("arrayAを出力 : ");
for(int i: arrayA) {
    System.out.print(i);
}
System.out.println();

System.out.print("arrayBを出力 : ");
for(int i: arrayB) {
    System.out.print(i);
}
System.out.println();
```

arrayAを出力 : 12345
arrayBを出力 : 12345
arrayAを出力 : 99992345
arrayBを出力 : 99992345

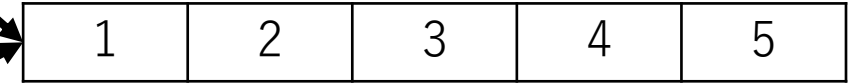
arrayBを変更した覚えがないのに、
arrayBも変わってしまった・・・

```
int [] arrayA = {1, 2, 3, 4, 5}; arrayA
```

変数名は別でも、実体は同じデータ

```
int [] arrayB = arrayA; arrayB
```

arrayAと同じ場所を参照



配列などの参照型の変数を = で代入すると、
データの実体があるアドレスへの参照が代入される

同じところを指しているので、
どちらかを変更するともう一方も変更されてしまう

シャローコピー（浅いコピー）

参照している先のコピー（実体は同じ）、見せかけの複製を作る 38

String型も、実は参照型

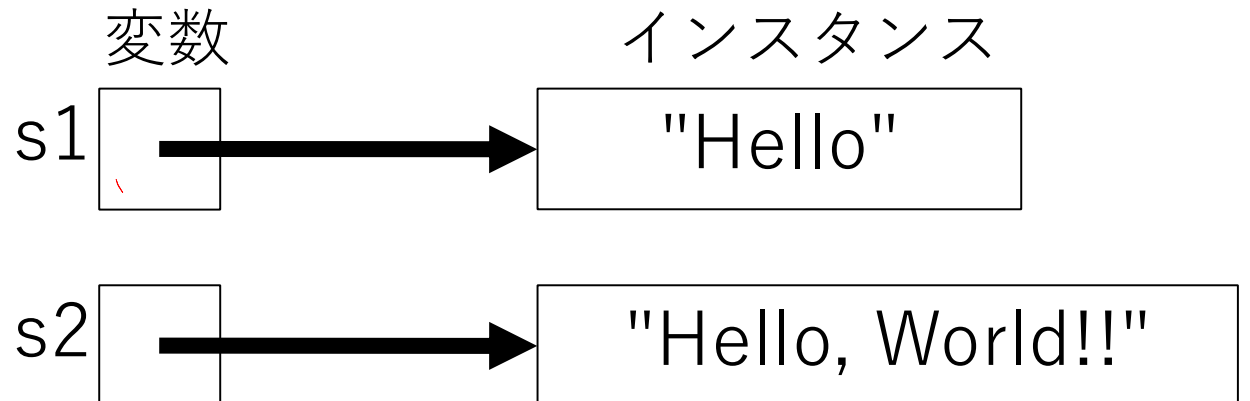
- 変数に直接データは格納されない

```
String s1 = "Hello";
```

s1 "Hello"

実は、
この書き方は
正確ではない。

文字列の長さで
メモリの使用量も違う。
変数に直接入らない。



String型扱い時の注意

- == で比較したら何故駄目なのか？

```
Scanner sc = new Scanner(System.in);
String s1 = "Hello";
String s2 = sc.next(); // Helloと入力
if(s1==s2) {
    System.out.println("s1とs2は同じ文字列");
}
```

Hello

s1とs2は同じ文字列 と表示されない

== では、等しいと判定できていない

String型の比較は、equalsメソッドを利用する

```
Scanner sc = new Scanner(System.in);
String s1 = "Hello";
String s2 = sc.next(); // Helloと入力
if(s1.equals(s2)) {
    System.out.println("s1とs2は同じ文字列");
}
```

Hello

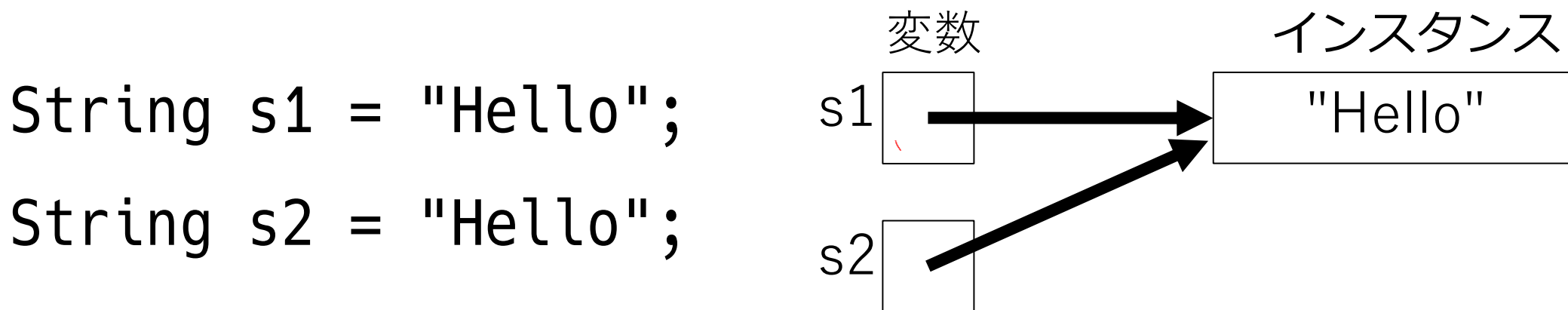
s1とs2は同じ文字列

equalsを使えば問題ない

- String型を比較する際には、.equalsメソッドを利用

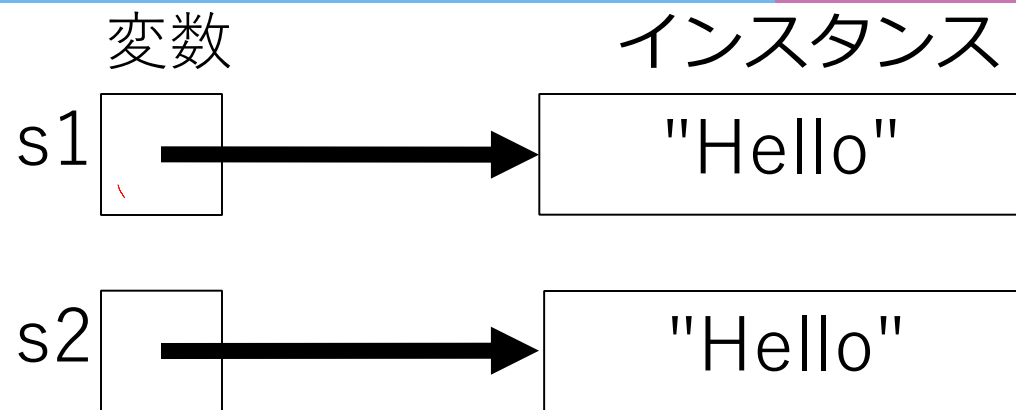
String型インスタンスの生成タイミング

- 同じ文字列を **プログラムに直接記述** すると、
既に同じ文字列リテラルが定義されていたときには、
そのオブジェクトへの参照を利用する



プログラム内に直接記述した文字列：**文字列リテラル**

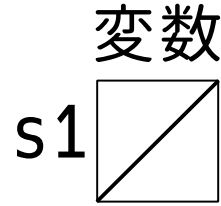
```
String s1 = "Hello";  
String s2 = sc.next( );
```



- `sc.next( )` メソッドを呼び出すと、新しく `String` 型のインスタンスを生成する
 - 参照先が異なるので、`==` では文字列の比較が出来なくなる

String型のnewでの生成

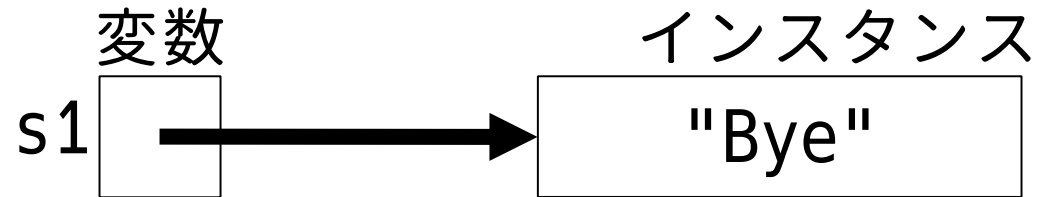
```
String s1;
```



この時点ではまだ実体は存在していない

```
s1 = new String("Bye");
```

newで生成するとインスタンスが出来る

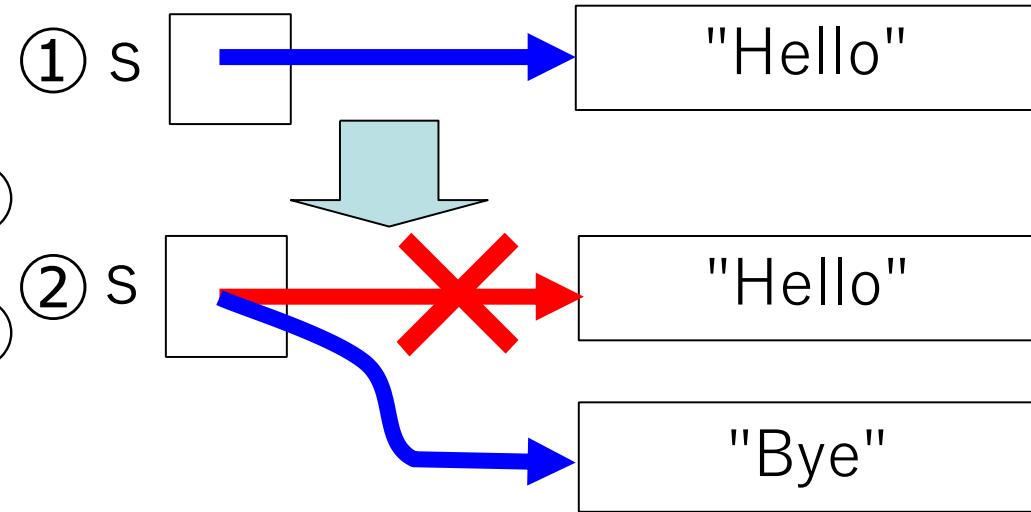


- String型はnew演算子なしで生成できるが、内部では new されている。
`s1 = "Bye";` これからもこちらの書き方で良いが、内容は理解しておくこと。

参照先の変更

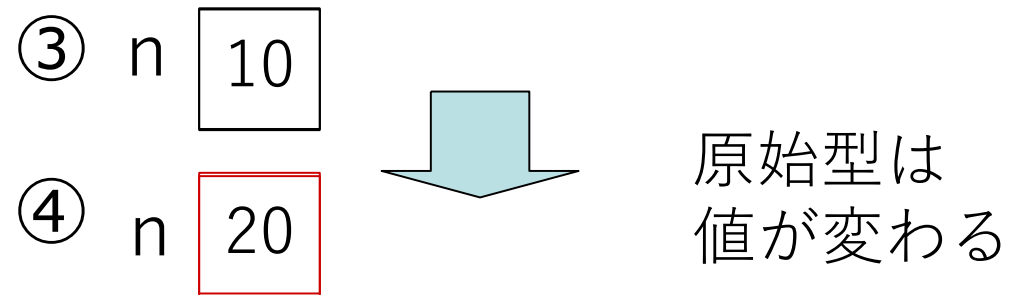
[参照型]

```
String s = new String("Hello"); // ①  
s = new String("Bye");          // ②
```



[原始型]

```
int n = 10;          // ③  
n = 20;              // ④
```

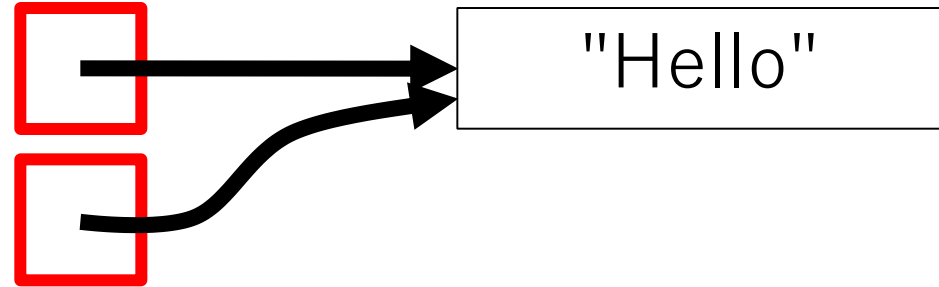


参照型のデータが等しいかどうかは 関係演算子 == ではわからない

同じオブジェクトと同じ値は異なる

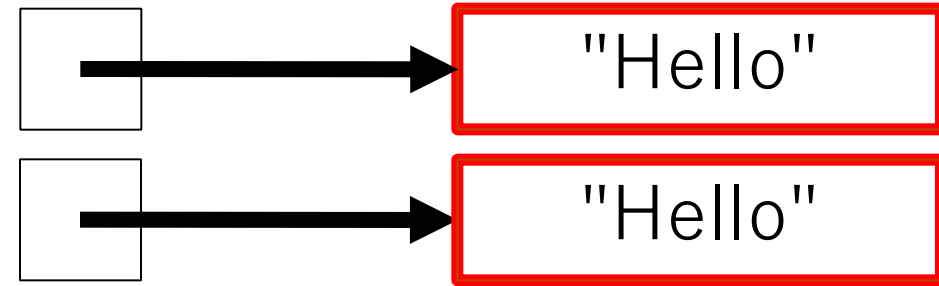
==

同じオブジェクト
(変数の値が同じ)
を参照しているならtrue



equalsメソッド

オブジェクトの値が同じなら
true



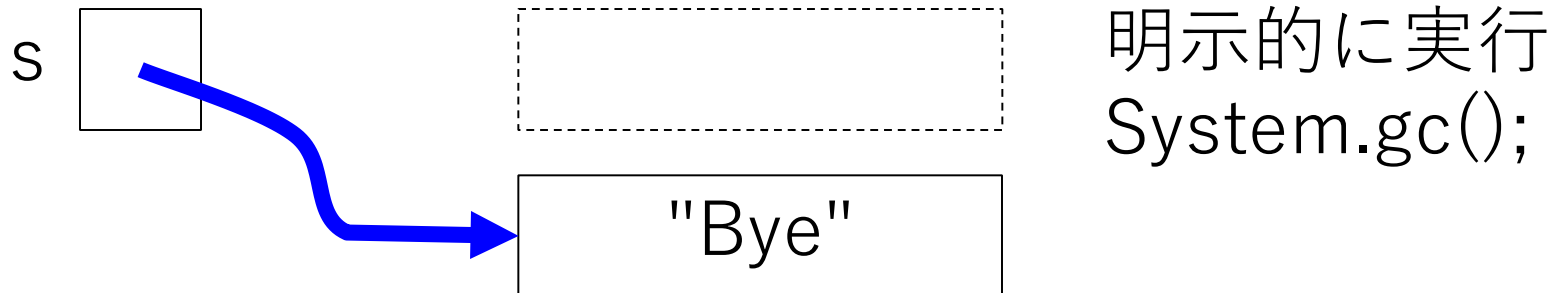
equalsメソッドは、この例ではString型のクラスのメソッド
変数名1.equals(変数名2) 例 **s1.equals(s2)**
オブジェクトの値(文字列)が等しければtrue

ガベージコレクタ

```
String s = new String("Hello");  
s = new String("Bye");
```

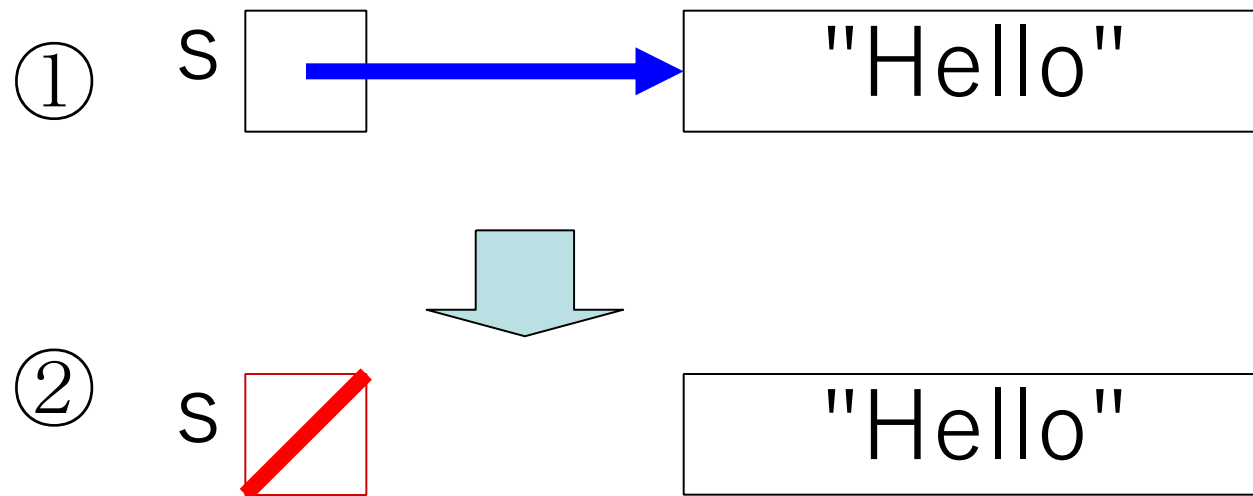


- 参照されないオブジェクトはガベージコレクタ（ごみ集め屋）が回収（その部分のメモリが使えるようになる。）



null

```
String s = new String("Hello"); // ①  
s = null;                        // ②
```



どこも参照していない状態になる

String型を扱う上での注意（まとめ）

- 普段は、 `String s1 = "Hello";` の書き方で問題ない
- String型を比較するときは、
`s1.equals("Hello");` のようにして、
.equalsメソッドを利用する
- String型は参照型であって、
内部処理としてはインスタンスが生成されていることは知っておく

補足：オブジェクトとインスタンス

- オブジェクト

- プログラム中に存在する、処理できる全ての実体
- インスタンスもオブジェクトの一種
- Mathクラス等、インスタンス化出来ないクラスはオブジェクト

- インスタンス

- クラスから生成された実体を持つオブジェクト
- `Person p1 = new Person();` など、
クラスを元に生成される個別のオブジェクト

- (厳密には異なるが)

Javaにおいては同じものとして扱って問題ない

まとめ

- インスタンスメソッドの宣言と呼び出し

- 宣言

```
class クラス名 {  
    戻り値 メソッド名(引数の型 引数名1, 引数の型 引数名2, ...){  
        処理;  
        return 戻り値;  
    }  
}
```

- 呼び出し

```
//クラス型のインスタンス生成と代入  
型 インスタンス変数 = new 型();  
型 変数名 =インスタンス変数名.メソッド名();
```

- 参照型変数を扱う上での注意

- 参照渡し、String型の比較（equalsメソッド）

