情報理工学部　実世界情報コース

プログラミング演習１

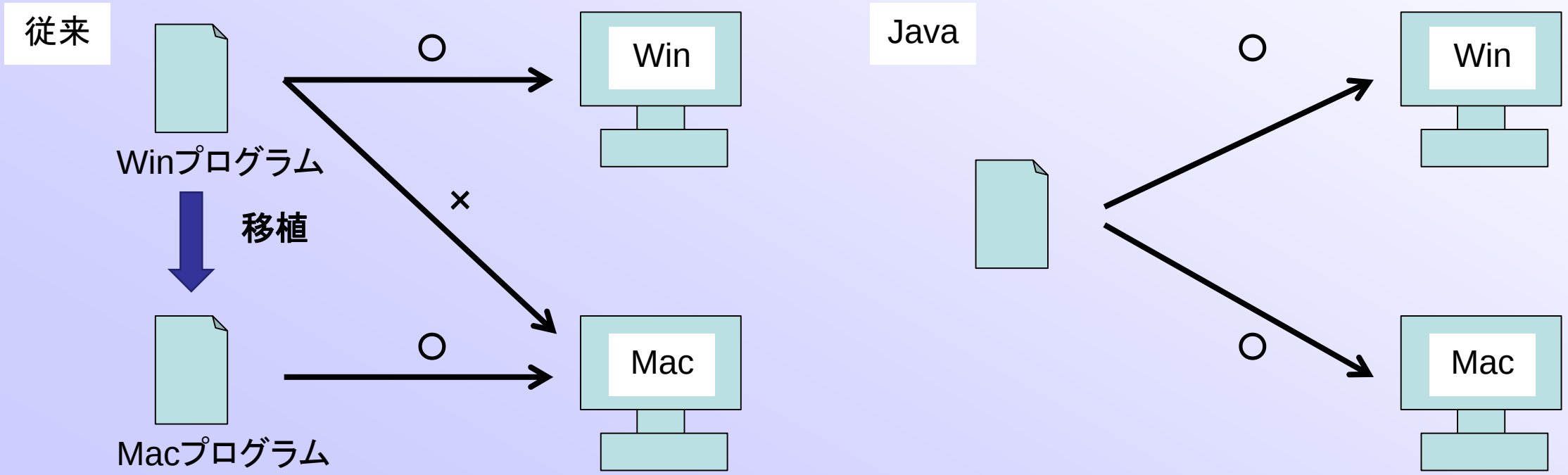
Java（１）

Javaとは

- Java言語は**プログラミング言語**の1種である
- プログラミング言語とは、人間がコンピュータに指示を与えるための言語である
- プログラミング言語には大きく、2種類ある
  - **構造化プログラミング言語**（C言語、BASIC言語など）
  - **オブジェクト指向言語**（Java言語、Python言語、C#言語など）
- Java言語の特徴は、**Webブラウザ**で動作する**アプレット**が作成できる

Javaの特徴

- Javaのプログラムは**OS(Operation System)に依存しない**
- 1度プログラムを作ると移植する必要がない**Write once, run anywhere**の特徴がある
- 依存しないのは、ハードウェア上に**Java Virtual Machine(Java VM)**と呼ばれる仮想環境があるためである。

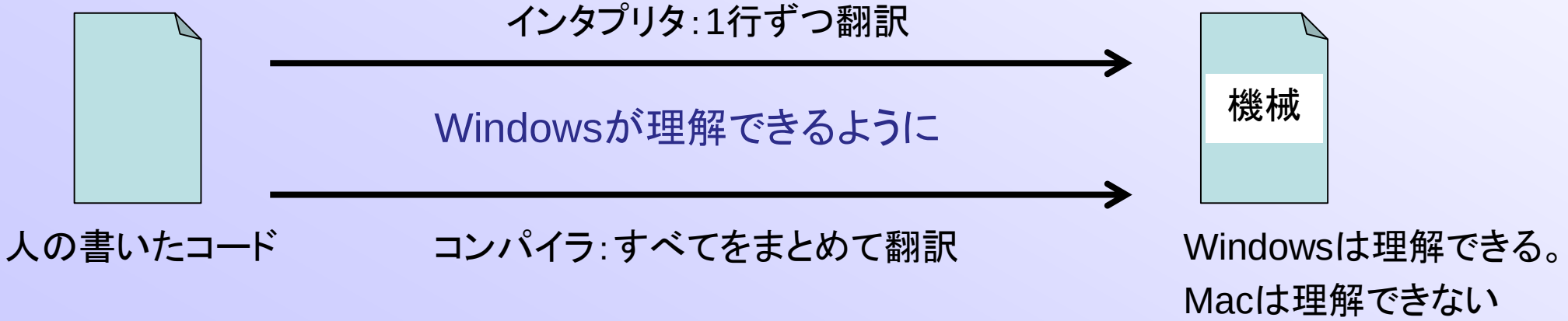


プログラムの動作

- コンピュータは**機械語（マシン語）**しか理解ができない
  - 人間が書いたコードをコンピュータが理解できるように翻訳する必要がある。

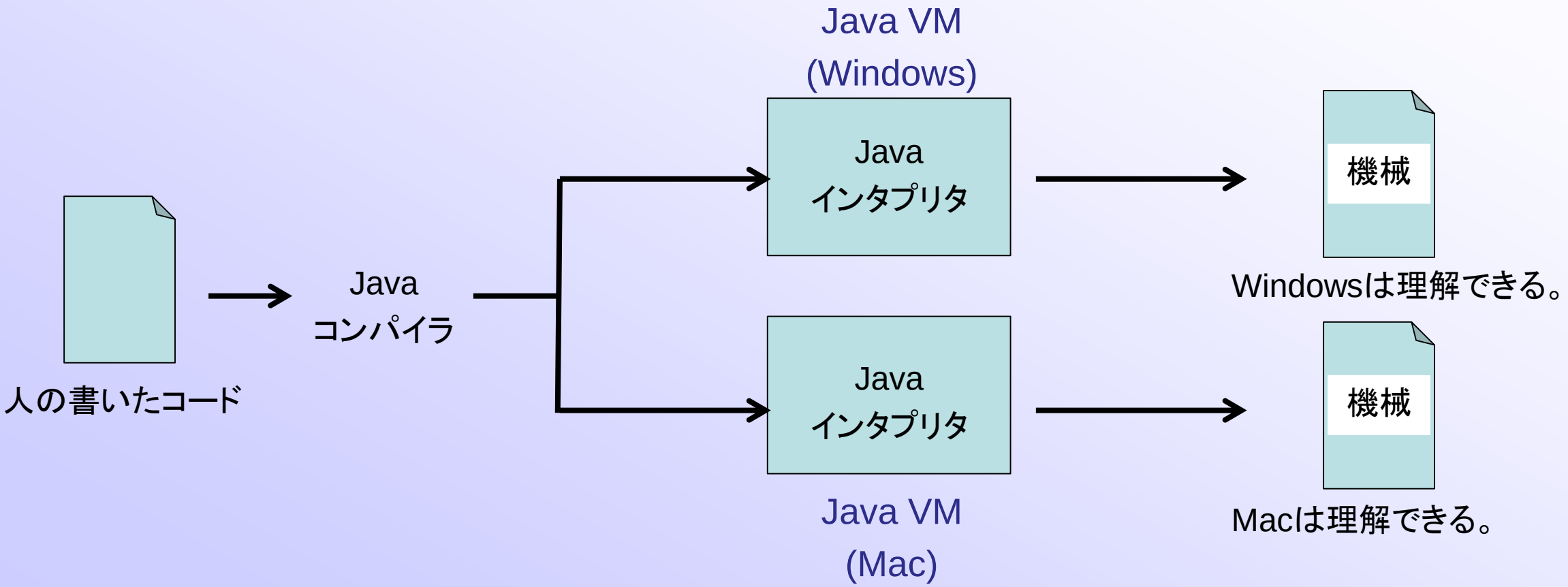
- このような変換（翻訳）を**インタプリタ**や**コンパイラ**という
  - インタプリタ：リアルタイムに翻訳をする
  - コンパイラ：すべての言語をまとめて翻訳する

  ※Pythonはインタプリタ上で動くことを想定しており、ではJupyter Notebookで逐次翻訳されていた



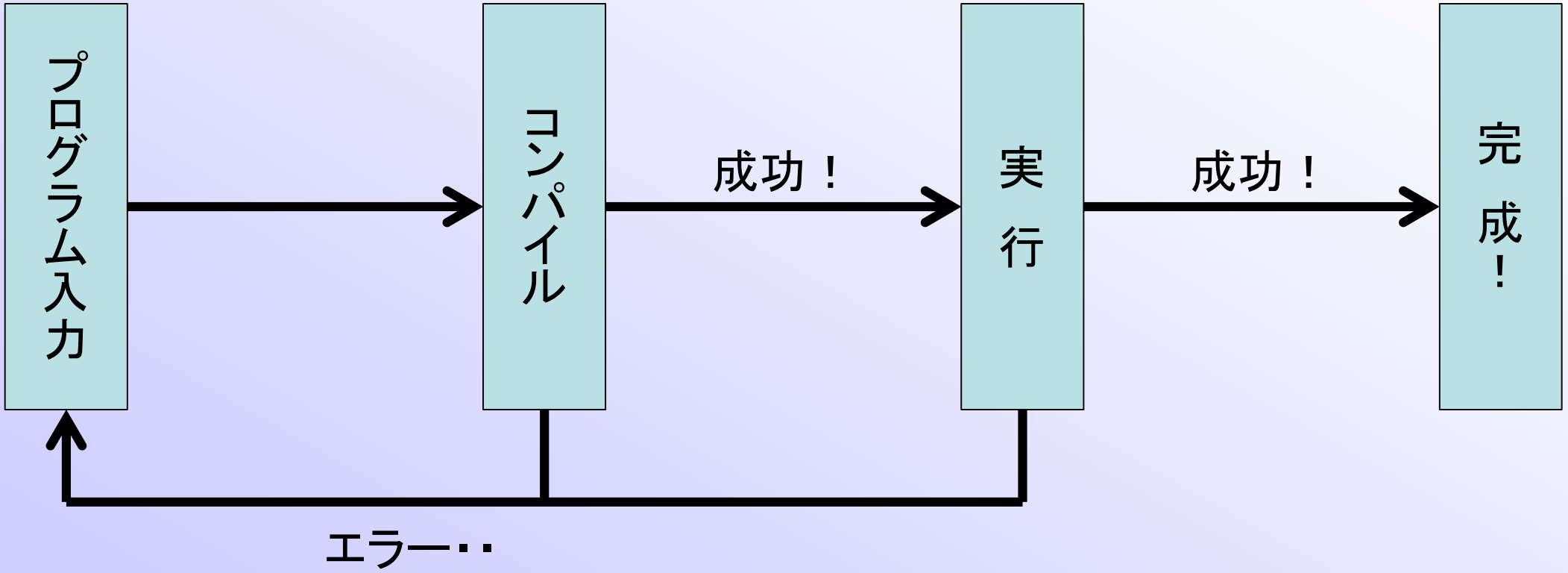
Javaプログラムの実行

- JavaはWindowsやMacなどのハードウェア上ではなく、**仮想環境（Java VM）**で変換してくれる



プログラムが実行されるまで

- プログラムの実行するためには以下の流れになる



- JavaはPythonと違い、プログラムが完成（実行）する前にエラーをチェックする

Javaでコンパイル

- Javaプログラムのコンパイルの方法

  ＞javac ファイル名.java

  –エラーが出る場合は、何がエラーか表示されているので、英語だからと言って、読むことはあきらめない

- Javaプログラムを実行する方法

  ＞java ファイル名（クラス名）

基本のプログラムの形と出力

- Javaプログラムを書くときは、ベースとして次のような形になる
- Javaのクラスは、**ファイル名と同じ**にしなければエラーとなる

ファイル名.java

```
1  public class ファイル名 {
2          public static void main(String[] args)
3          {
4                  プログラムの内容（処理）;
5          }
6  }
```

- 出力にはprintもしくはprintlnを使用する。

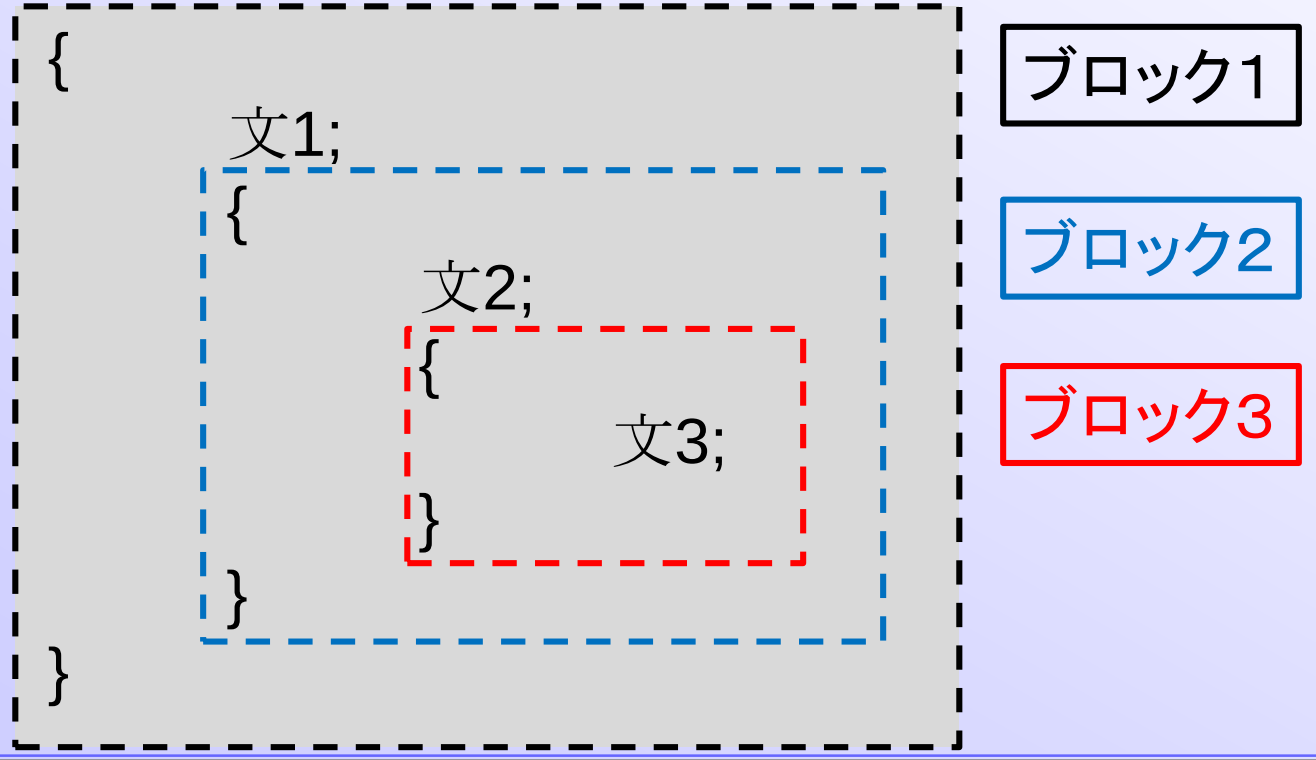
```
1  // 改行されない
2  System.out.print(“Hello World”);
3
4  // 改行される
5  System.out.println(“Hello World”);
```

Javaは「“”」と「‘’」は異なる。

Javaの特徴

- 1行のコードは**セミコロン「;」**で終わる必要がある

- 1つのまとまりは、**中括弧「{　　}」**である。
  - インデントは見やすくするため。※Pythonではインデントでひとまとまり



Javaのコメントについて

- Javaプログラムでコメントを利用する場合は次のように書きます

```
//1行だけがコメントになります。

/* この中にあるものはコメントになります。複数行でも構いません。 */
```

- Javaには、Javadocとう仕組みがあり、プログラムの仕様をHTMLファイルとしてせいせいできる

```
/** この範囲内がjavadocによるコメントになります。 */
```

- Javadocのタグ

@author: 著者名
@return: メソッドの戻り値の方と範囲
@since: プログラムの最初のバージョン
@param: パラメータに関する情報
@see: プログラムに関連するキーワード
@version: プログラムの現在のバージョン

基本データ型と変数

- 基本データ型は次のとおりである

型の名前	種類	特徴
整数型	short, byte, int, long	負を含めた整数
浮動小数点型	float, double	負を含めた少数
文字型	char	1文字を扱う
論理型	boolean	TrueまたはFalse

- 基本型のほかに参照型もある
  - Stringなど

- **Pythonと違い、入れたい値によって宣言をしなければならない**

整数型と少数型

- 整数型の詳細は次の通り

型名	種類
byte	-128～127
short	-32,768～32,767
int	-2,147,483,648～2,147,183,647
long	-9,223,372,036,854,775,808 ～9,223,372,036,854,775,807

- int型で宣言しても、値が許容範囲ならbyte型などに自動変化する
  - 明示的に変換（**キャスト**）するなら、値の前に「（型）数値」とする
- long型を明示するには語尾「**L**」か「**l**」を書く（例：12345L）
- 整数型は数値の前に「**0**」を入れると8進数、「**0x**」を入れると16進数になる

- 浮動小数型の詳細は次の通り

型名	種類
float	$-3.40282347 \times 10^{38}$～$3.40282347 \times 10^{38}$
double	$-1.7976931348623157 \times 10^{308}$～ $1.7976931348623157 \times 10^{308}$

- 値の表記の仕方は次の通り
  - 0.0123（そのまま表記）
  - 1.23×10^{-2}（指数表記）
  - 1.23**E**-2、1.23**e**-2（科学表記）

- 浮動小数型も語尾を書く
  - **float型：「F」「f」**（例：0.123f、3.1415F）
  - **double型：「D」「d」**（例：0.123d、3.1415D）

文字型と論理型

- 1文字を扱うための型を「文字型」といい「char」で書きます。
  ※文字列は「char」では宣言できません
- Javaは基本的に「**Unicode**」である。
- 文字で表現できないもの（タブなど）は、**エスケープ・シーケンス**や**Unicodeエスケープ**を使う

文字	エスケープ・シーケンス	Unicode シーケンス
バックスペース	¥b	¥u0008
水平タブ	¥t	¥u0009
改行	¥n	¥u000a
改ページ	¥f	¥u000c
復帰	¥r	¥u000d
ダブルクォート	¥”	¥u0022
シングルクォート	¥’	¥u0027
￥マーク（バックスラッシュ）	¥¥	¥u005c

- 論理型は、「true」と「false」の2種類のみ保持する
- 型は「boolean」である。

- if分などの条件文などで利用する

変数

- JavaはPythonと違い、変数を宣言するときに型を指定する必要がある。

```
型　識別子;
（例）int number; ・・・・int型のnumberという名前の変数
```

- 定数を宣言する場合は、型の前に「final」とつける

```
final 型　識別子;
（例）final int number; ・・・・int型のnumberという名前の変数
```

- 同じ型であれば、カンマで区切って変数を複数宣言できる

```
final 型　識別子1, 識別子2, …;
（例）int num1, num2, num3;
```

変数に使えない文字列

- 変数の名前のルール
  1. **1文字目は必ず文字**にする
  2. 使えない記号（！や＃）などがある。
  3. **予約語**（あらかじめ、特別な意味のある単語）は利用できない

abstract	continue	for	new	switch
assert	default	if	package	synchronized
boolean	do	goto	private	this
break	double	implements	protected	throw
byte	enum	import	public	throws
case	else	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

- Javaでは、2つ以上の単語がくっついている変数は後ろが大文字（例：newRoom）
- 定数は大文字で2つ以上の単語がくっついている場合はアンダーバーでつなぐ（例：MAX_SCORE）

変数の初期化

- 定義した変数や定数に値を代入するには次のように書く

```
int number;
number = 3;
```

```
final int TEISU;
TEISU = 3;
```

- 宣言時に初期値を与えることもできる

```
int num1 = 1, num2 = 5, num3 = 4;
```

```
final int TEISU1 = 0, TEISU2 = 5;
```

- 2つ以上の値を次のようにして代入することもできる

```
int x, y, z;
x = y = z = 3; //x, y, z をすべて3にする
```

変数の有効範囲（スコープ）

- 変数には有効範囲があり、この有効範囲を**スコープ**という

```
1  {
2          int a = 3;
3          int b = 5;
4
5          {
6                  int a = 2;
7                  int c = 5;
8                  System.out.println(a+b);
9                  System.out.println(a+c);
10         }
11
12         System.out.println(a+b);
13         System.out.println(a+c);
14 }
```

青ブロックでは，変数としてaとbを宣言している
(1) 青ブロックは，中にあるブロック（赤ブロック）で宣言している変数は使えない

赤ブロックでは，変数としてaとcを宣言している
(1) 青ブロックのaと赤ブロックのaは違うもの
(2) 青ブロックは赤ブロックで定義している変数を使用できる

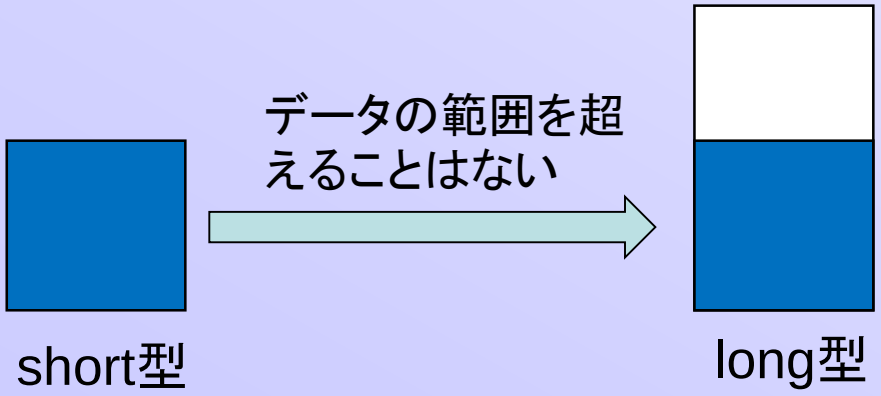
上にブロック（青ブロック）で宣言しているのでOK

赤ブロックで宣言しているのでcは赤ブロックしか使えないので**コンパイルエラー**

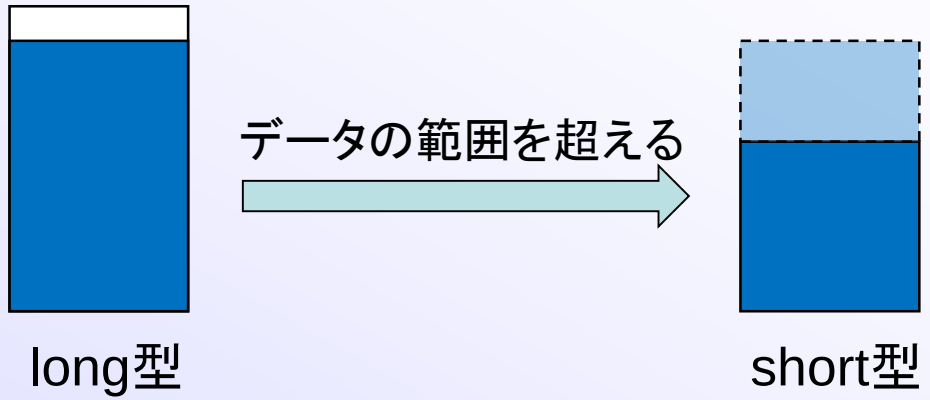
ワイドニング変換・ナローイング変換・キャスト（型の変換）

- 一度宣言した型を変更することができる
  - 型の取りうる範囲を考える必要がある
    - double > float > long > int > short > byte
    - charはintに変換できる
  - 右から左（ex: int→long）へ変換することを**ワイドニング変換**という

```
1  long l_num; //long型変換の宣言
2  short s_num;   //short型変換の宣言
3  l_num = s_num;   //shor型→long型
```



  - 左から右（ex: long→short）へ変換することを**ナローイング変換**という
  - 変換できないので**コンパイルエラー**になる




```
s_num = l_num;   //ナローイング変換
```

- どうしても大きな型から小さな型に変換する場合は，**キャスト**を行う
  - ただし、超えるデータは**損失**する

```
s_num = (short) l_num;   //キャスト
```

実際の例

```
1  int a = 5;
2  double b;
3
4  double pi = 3.1415;
5  int q;
6
7  b = a; //ワイドニング変換
8  q = (int) pi; //キャスト
9
10 System.out.println(“a = “ + a);
11 System.out.println(“b = “ + b);
12
13 System.out.println(“pi = “ + pi);
14 System.out.println(“q = “ + q);
```

出力結果

```
a = 5
b = 5
pi = 3.1415
q = 3
```

int型は整数型なので、小数点が切り捨てられる
（データの損失）

演算子について

- 演算子には次のようなものがある

演算子	説　明
代入演算子	計算結果を変数に代入する演算子（＝）
算術演算子	四則演算などを行うための演算子（+ - * / %）
比較演算子	大きいや等しいなどを判断する演算子。その結果が正しい（True）か間違っている（False）を判断する（< > <= >= ==）
論理演算子	「○○**かつ**△△」や「○○**または**△△」や「～でない」などの論理的な演算（&&: かつ ||: または !: ～でない）
ビット演算子	コンピュータ内部における2進数の値を直接操作する演算子（<< >>:シフト演算子 &: And演算子 |: OR演算子 ^: XOR演算子

演算子の優先順位

- 演算子には優先順位があり、優先順位の高いものから計算される

優先順位	演　算　子
1	[] . ()
2	! ~ ++ -- +（正符号） -（負符号） ()（キャスト） new
3	* / %
4	+（加算） -（減算）
5	<< >> >>>
6	< <= > >= instanceof
7	== !=
8	&
9	^
10	|
11	&&
12	||
13	?:
14	= += -= *= /= %= &= |= <<= >>= >>>=

代入演算子

- 代入演算子は、変数に値を代入するために使用する

```
変数 = 値か変数;
```

- 型が異なるとエラーになる

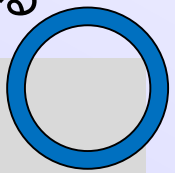
```
例）
int ans;
ans = 1.8 + 4.2; //ansは整数型で計算は少数型なので、ナローイング変換になる
System.out.println(ans);
```

キャストする

```
例）
int ans;
ans = (int)1.8 + 4.2;
System.out.println(ans);
```

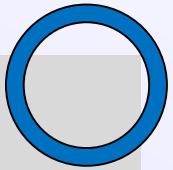
キャストしているのは、1.8のみ
コンパイルエラー

```
例）
int ans;
ans = (int)1.8 + (int) 4.2;
System.out.println(ans);
```



どちらもキャストしてから計算
結果は5(1+4になる)

```
例）
int ans;
ans = (int)(1.8 + 4.2);
System.out.println(ans);
```



計算してからキャスト
結果は6(先に計算なので6.0)

算術演算子

- 四則演算子はPythonと同じである
  - 除算（割り算）は注意する必要がある

```
例） ans = 1/3;
```

  - 通常は0.333・・・となるが、整数型なので結果は0となる
- 計算式に複数の型があれば範囲の大きなほう（精度の高いほう）を採用

```
例） ans = 1.0/3;
```

  - 1.0は少数型、3は整数型→結果は少数型になり、0.33333333となる

- 文字の演算も基本的に同じである
  - 文字と数値は数値のほうが精度が高いので、キャストをしないと数値で出力される

```
例）    ans = 'A' + 1;
        System.out.println(ans);
        System.out.println((char)ans);
結果）  66 (Aは数値で65）
        B
```

- 文字列であれば加算ができ、結果は文字列になる
  - 文字列＋文字列
    （"Hello " + "wolrd" = "Hello wolrd"）
  - 文字列＋文字
    （"Hello " + 'B' = "Hello B"）
  - 文字列＋数値
    （"Hello " + 2014 = "Hello 2014"）

比較演算子

- 四則演算子はPythonと同じである

演算子	意味	使用例	結果
<	～より小さい	1 < 2	True
<=	～以下	1 <= 2	True
>	～より大きい	1 > 2	False
>=	～以上	1 >= 2	False
==	等しい	1 == 2	False
!=	等しくない	1 != 2	True

- 異なる方の比較演算子は、型の大きいほうに自動でワイドニング変換される

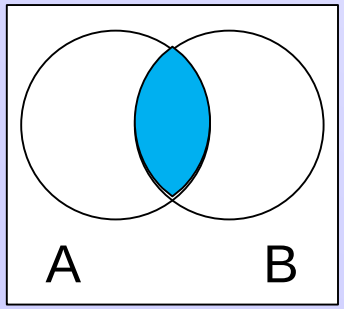
例）
1.0f < 2　　→　　1.0はfloat型、2はint型なので、より大きいfloat型に変換されて比較する

- 比較演算子は3つ以上の比較（例：a < b < c）はできない。1つずつ分割して記述する必要がある（例：a < b; b < c; a < c）。次に紹介する論理演算子と併用することが多い

論理演算子

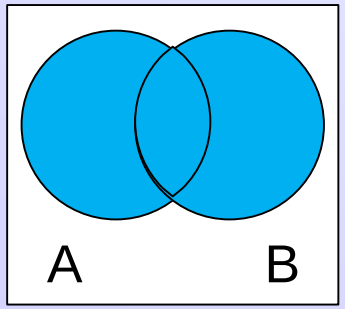
- 論理演算子の意味と図で表すと次のようになる

演算子	意味
&&	かつ
||	または
!	～でない



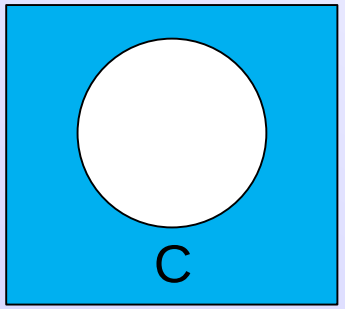
A && B

A	B	結果
False	False	False
False	True	False
True	False	False
True	True	True



A || B

A	B	結果
False	False	False
False	True	True
True	False	True
True	True	True



!C

A	結果
False	Ture
True	False

- 論理演算子では計算の途中で結果がわかってしまう場合（&&演算子でどちらかがfalse）はそれ以降の計算はしない（短絡評価と呼ぶ）

例）
(1 < 2) && (2 > 3) && (1 < 3) && (5 < 9)

優先順位を考慮しながら式を確認していく。
処理の流れとしては、

1) (1 < 2) → true
2) (2 > 3) → false ← And演算子なので、この時点で以降の計算はしないで結果「false」とする

3) (1 < 3) → true
4) (5 < 9) → true

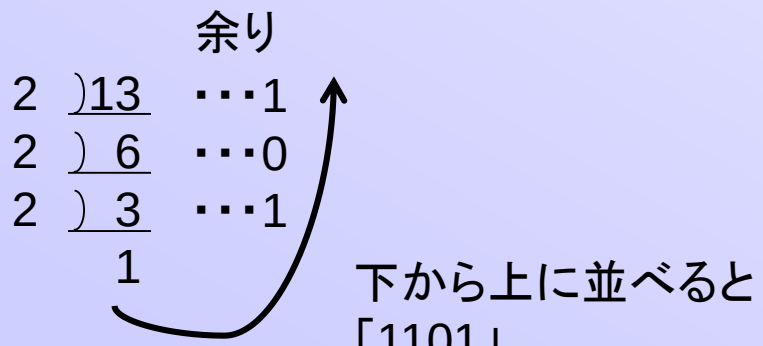
5) true && false && true && true → false

内部表現にかかわる演算子

- コンピュータは通常「0」か「1」で処理を行っている。コンピュータが扱うデータの最小単位を「**ビット（bit）**」という
  - 私たちは通常「0～9」という10個で表現している（10進数）
  - コンピュータは「0と1」の2個で表現している（2進数）

10進数	0	1	2	3	4	5	6	7	8	9	10
2進数	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010

- 10進数を2進数に変換（13を2進数に）



- 2進数を10進数に変換（1011を2進数に）

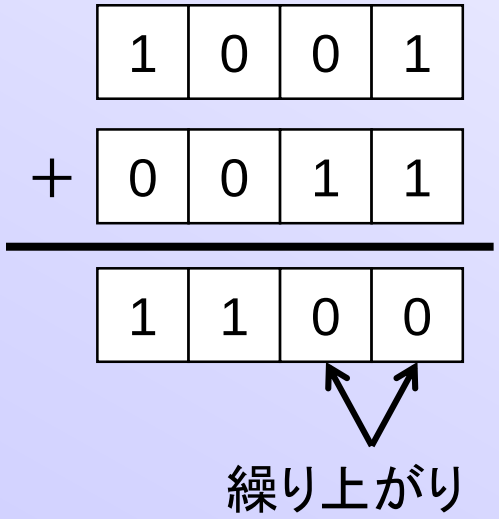
変換する場合は、右から2^0、2^1･･･とかけていく

1101
→1 × 2^3 ＋0 × 2^2 ＋ 1 × 2^1 ＋ 1 × 2^0
→8 + 0 + 2 + 1
→11

2進数の足し算・引き算

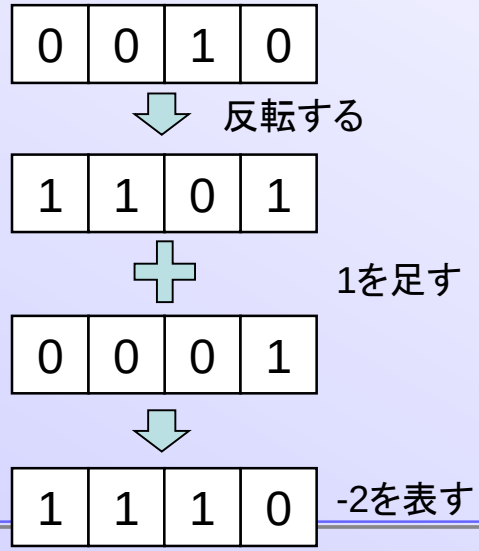
- 2進数も足し算や引き算ができる

  例）1001+0011



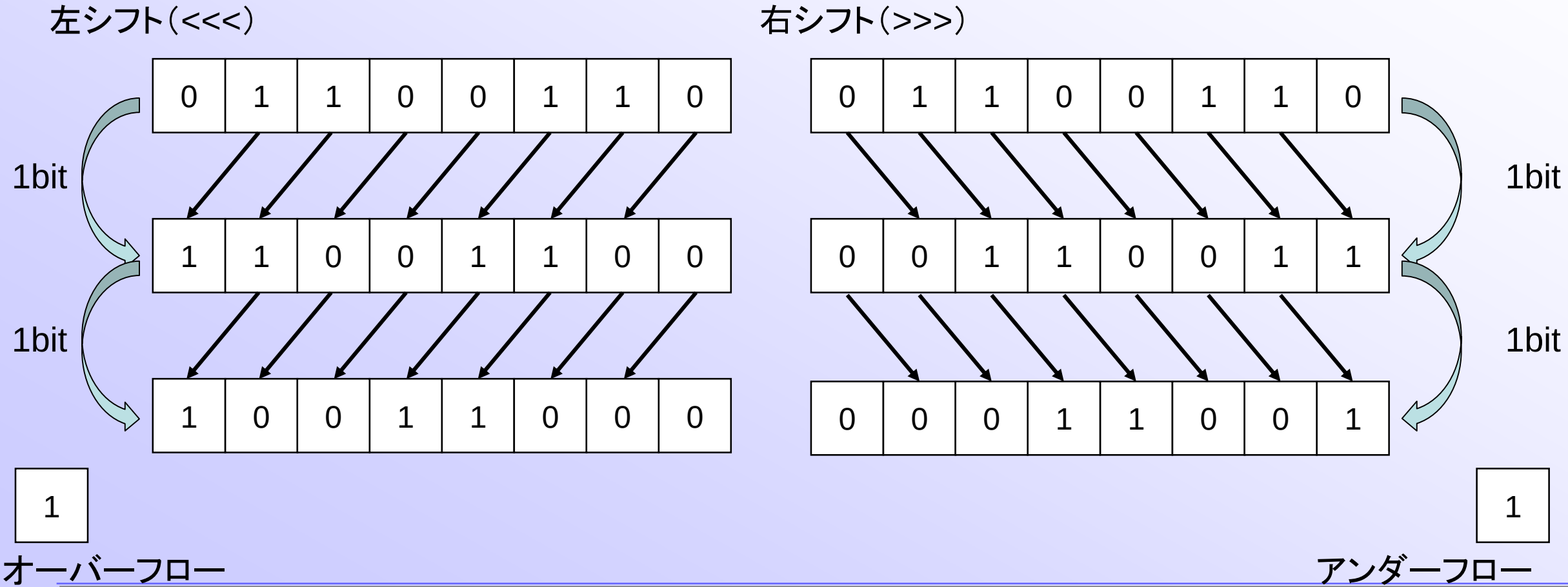
- 2進数では、負の値を考慮していないので、マイナス記号もない
- 負の値を表すために、最上位ビット（一番左）の値を符号として扱う（0なら正、1なら負）
- これを**2の補数**と呼ぶ

  例）0010（10進数で2）を-2にしたい場合



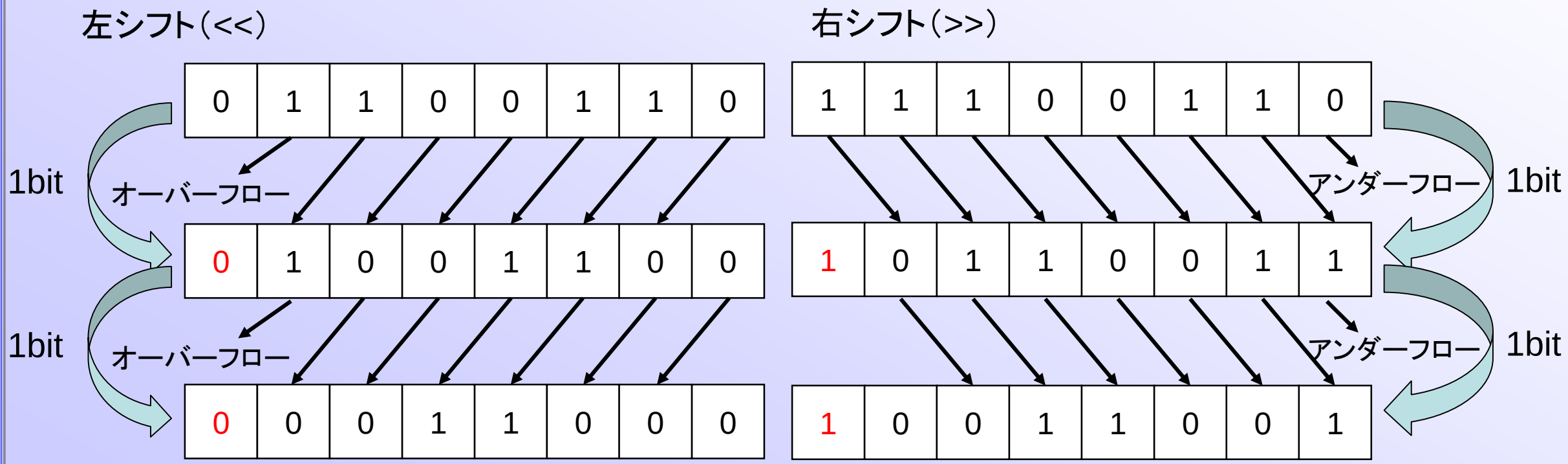
シフト演算子

- ビットを左右の桁に移動することを**シフト**といい、**左シフト**と**右シフト**がある
  - <<<や>>>を使うと、すべてのビットがシフトする



符号を考慮したシフト演算

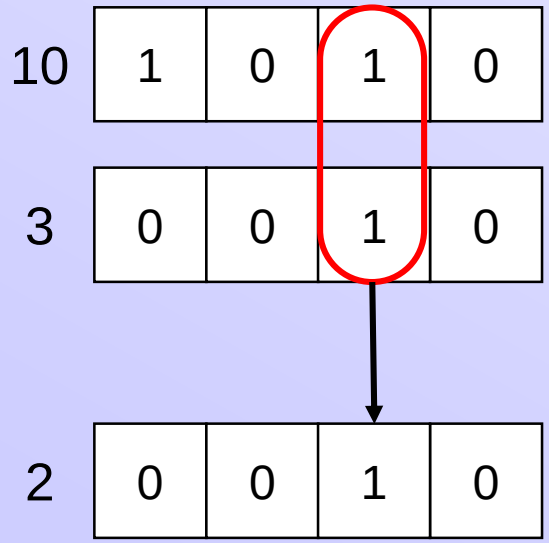
- シフト演算は最上位ビットを変化させないようにシフトする
  - 最上位ビットは符号を表すビットのため
  - 最上位ビットをシフトさせない演算子は<<や>>である



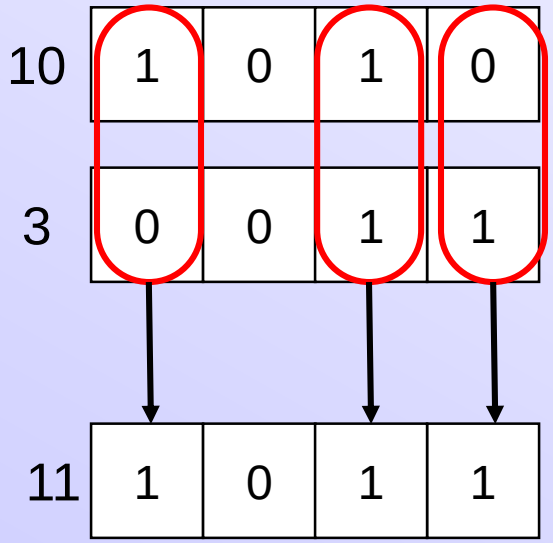
ビット演算子

- ビット演算子は**2進数**として計算する

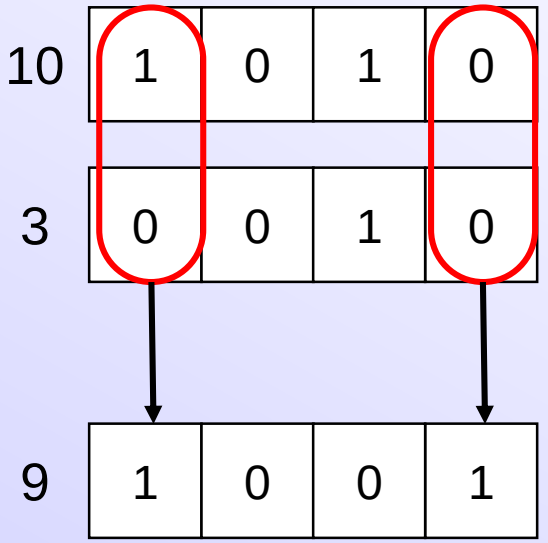
「&」:AND演算子
ある桁の両方が1のとき1
例）10 & 3



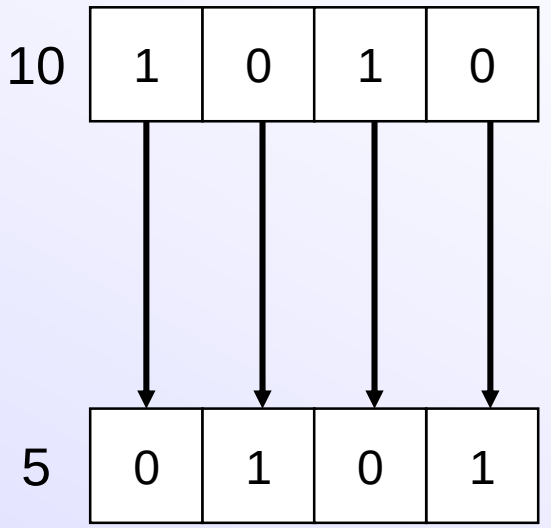
「|」:OR演算子
ある桁のどちらかが1のとき1
例）10 | 3



「^」:XOR演算子
ある桁の値が異なるとき1
例）10 & 3



「~」:NOT演算子
ビットを反転
例）10



他の演算子

- 単項演算子
  - これまでは二項演算子であった

例） x = x + 1;　←二項演算子

  - 簡略化できる

例） x += 1;　←単項演算子

- インクリメント/デクリメント演算子
  - 特に，+1や-1の場合は次のように書くこともできる

例） x = x + 1;
x++;　または　++x;

インクリメント

例） x = x - 1;
x--;　または　--x;

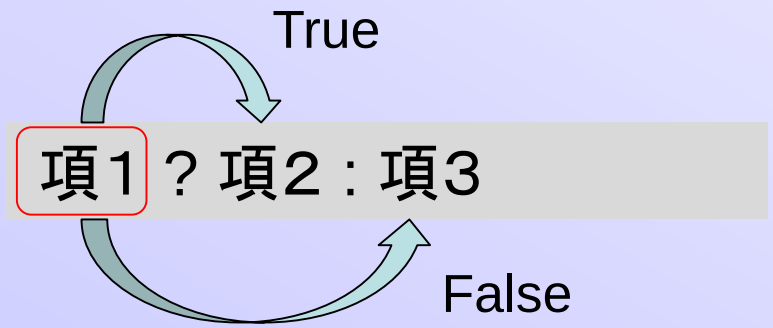
デクリメント

単項演算子	変換前		変換後
+=	a = a + b	→	a += b
-=	a = a – b	→	a –= b
*=	a = a * b	→	a *= b
/=	a = a / b	→	a /= b
%=	a = a % b	→	a %= b
&=	a = a & b	→	a &= b
|=	a = a | b	→	a |= b
^=	a = a ^ b	→	a ^= b
<<=	a = a << b	→	a <<= b
>>=	a = a >> b	→	a >>= b
>>>=	a = a >>> b	→	a >>>= b

右側は後置型、左側は前置型で、後置型は処理前に計算し、前置型は処理後で計算する

条件演算子

- 条件によって処理の内容を変えられる演算子
- 3つの項があるので、3項演算子と呼ばれる
- 条件文（if文）のような処理をする



例）絶対値を考える。

```
int x;
int a = -5;


x = a > 0 ? a : -a;
```

aが0より大きければ、xにaを代入
aが0より小さければ、xに-aを代入

この例では、a=-5なので、xに-(-5)=5を代入