

Javaプログラミング I

2回目 “ようこそJavaへ”

■ 今日の講義で学ぶ内容 ■

- 画面へのメッセージの表示
- 文字や文字列、数値を表現するリテラル
- 制御コードを表すエスケープシーケンス

画面出力の基本形

ソースファイル名：クラス名.java

```
class クラス名
{
    public static void main(String[] args)
    {
        System.out.println("ここに出力したい文字列1行目");
        System.out.println("ここに出力したい文字列2行目");
        :
    }
}
```

同じ 大文字, 小文字a区別)

ソースコード例

ソースファイル名：Sample2_1.java

拡張子は小文字で .java

```
// 画面に文字列を出力するコード
class Sample2_1
{
    public static void main(String[] args)
    {
        System.out.println("ようこそJavaへ!");
        System.out.println("Javaをはじめましょう!");
    }
}
```

タブ

実行画面

```
ようこそJavaへ!
Javaをはじめましょう!
```

いろいろな出力方法

1. `System.out.println("ここに出力したい文字列");` ↵

ここに出力したい文字列 が画面に表示された後、行末に改行が挿入されます

2. `System.out.print("ここに出力したい文字列");`

ここに出力したい文字列 が画面に表示された後、行末に改行が挿入されません

3. `System.out.printf("ここに出力したい文字列");`

ここに出力したい文字列 が画面に表示された後、行末に改行が挿入されません
※C言語 printf()関数と同じふるまいをします

4. `System.out.format("ここに出力したい文字列");`

ここに出力したい文字列 が画面に表示された後、行末に改行が挿入されません
※System.out.printf()と同じふるまいをします

改行?)
○○○○ ↵
○○○○

○○○○○○○○
|
改行なし

ソースコード例

ソースファイル名 : Sample2_2.java

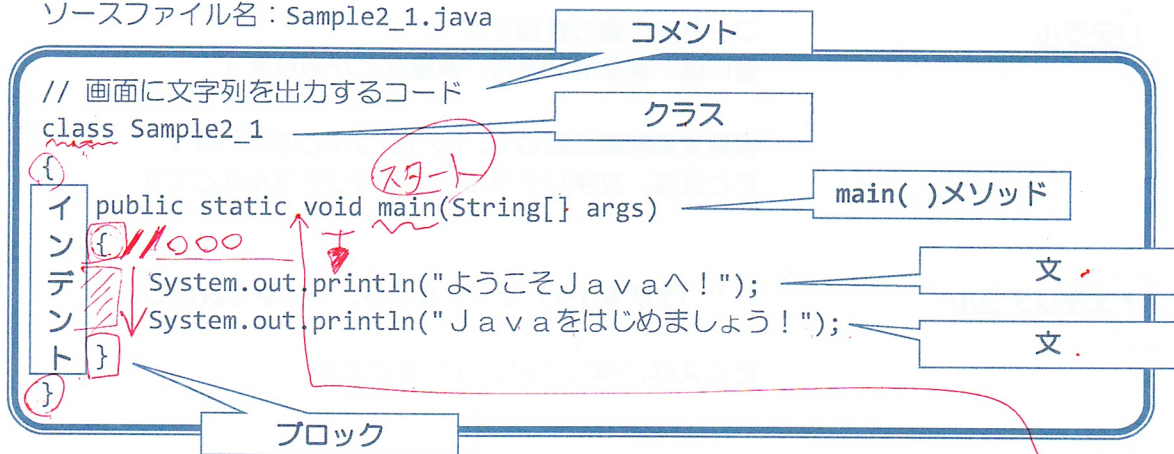
```
class Sample2_2
{
    public static void main(String[] args)
    {
        System.out.println("1. println()による出力 (改行) ");
        System.out.print("2. print()による出力");
        System.out.printf("3. printf()による出力");
        System.out.format("4. format()による出力");
    }
}
```

実行画面

1. println()による出力 (改行) ↵
2. print()による出力 3. printf()による出力 4. format()による出力 ○○○○ - ..

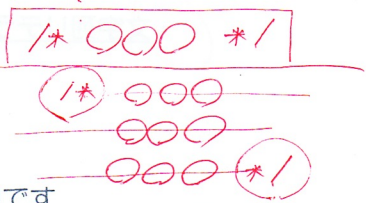
コードの内容

ソースファイル名 : Sample2_1.java



コメント

- // (ダブルスラッシュ) からその行の最後まで、または /* */ で囲まれた行 (複数行でも可) です
- コンパイル時には無視されるコードです
- 処理の内容などメモを記入しておくのに大変に便利です



クラス

- キーワード **class** がついた { } (括弧) 内をいいます
- Java のソースコードは少なくとも1つのクラスから成ります

📄 一般に Java のソースコードは複数のクラスをもちますが、ここでは1つのクラスをもつ場合を主に解説していきます

ブロック

- { } (括弧) で囲まれた処理の集まり部分です
- キーワード **class** がつく { } と区別します

main()メソッド

- public static void main(String[] args) で始まるブロックです
- Java ではここからプログラムの処理が始まります

文

- 最後に ; (セミコロン) がついた個々の単一の処理や命令です
- ブロックは複数の文をもつことができます
- 文は上から下へ順番に実行されます

📄 文には単一の処理や命令の他に **if 文** や **for 文** などがあります
また、文にはブロックを置くこともできます
これについては if 文の回で詳しく解説します

インデント

- 行頭での字下げです
- ソースコードを読みやすくします
- ブロック毎にインデント を付けると見やすくなります

コード上の

リテラル

文字や文字列、数値の表記

リテラル

コード内の値の表現です
値には、文字や文字列、数値などがあります

表現する対象に応じて、ooリテラルと呼ばれます
たとえば、文字リテラル、文字列リテラルなどです

• 文字リテラル

' ' (シングルクォート) で文字を囲み、一文字を表現します

たとえば、'A'、'b'、'c' などです

• 文字列リテラル

" " (ダブルクォート) で文字列を囲み、文字列を表現します

たとえば、"Hello"、"こんにちは" などです



"A" は大丈夫ですが、'Hello' は

「文字リテラルが閉じられていません」

というコンパイルエラーになります

一文字は文字列の特別な場合と考えることができますので、

文字列リテラルで表現することができます

しかし、文字リテラルは一文字である必要があります

• 整数リテラル

整数をそのまま記述し、整数を表現します

たとえば、123、-23 などです

• 浮動小数点数リテラル

実数値をそのまま記述し、実数値を表現します

たとえば、3.14、-1.2、0.24 などです

• 論理値リテラル

true 又は false を記述し、論理の真と偽を表現します



この他、空リテラルがありますが、Java プログラミング IIで詳しく解説します

• 空リテラル

null を記述し、空の参照を表現します



このマークの内容は発展的なものです

ソースコード例

ソースファイル名: Sample2_3.java

```
class Sample2_3
{
    public static void main(String[] args)
    {
        System.out.println('A');
        System.out.println("Hello");
        System.out.println(123);
        System.out.println(0.24);
        System.out.println(true);
    }
}
```

メソッド

printlnの呼び出し

{ } 組

① { } ② } ③ < コード

インクリメント
tab
キー

実行画面


```
A
|Hello
|123
|0.24
|true
```

各行の
組と書いた
後で、中に
コードと書いて
いく。

エスケープシーケンス

エスケープシーケンス ¥ (円マーク) をつけた2つの文字により表現される一文字です

たとえば、'¥n'、'¥t' などです

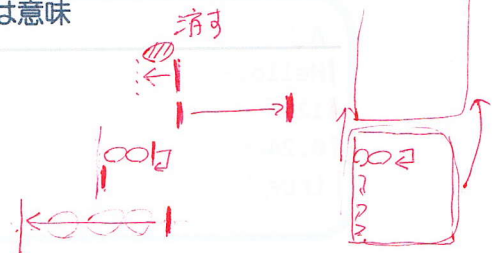
 エスケープシーケンスは一文字の表現であるため、文字リテラルで表現することができます


エスケープシーケンスは、改行やタブなどの機能を表示します

たとえば、`System.out.print('¥n');` で改行が行われます

次のような種類があります


表記	表記の機能、または意味
¥b	バックスペース
¥t	水平タブ
¥n	改行
¥f	改ページ
¥r	復帰
¥"	" ←ダブルクォート自身の表現に用います
¥'	' ←シングルクォート自身の表現に用います
¥¥	¥ ←円マーク自身の表現に用います



 エスケープシーケンスは、A や B などの一文字と同様に、文字列を構成する文字として用いることができ、文字列リテラルに含めることができます

たとえば、"`Hello¥n`" です

¥n, a, b

 一文字は文字列の特別な場合と考えることができますので、1つのエスケープシーケンスを"`¥n`"のように文字列リテラルで表現しても問題はありません

ソースコード例

ソースファイル名: Sample2_4.java

```
class Sample2_4
{
    public static void main(String[] args)
    {
        System.out.println("バックスペースします\b バックスペースしました");
        System.out.println("水平タブいれます\t 水平タブいれました");
        System.out.println("改行します\n 改行しました");
        System.out.println("復帰します\r 復帰しました");
        System.out.println("ダブルクォートを表示します");
        System.out.println('"');
        System.out.println("円マークを表示します");
        System.out.println('¥');
    }
}
```

¥f 改行-シ、\r、\n等の
出力機器で
有効。

実行画面

```
バックスペースしまバックスペースしました ↵
| 水平タブいれます|—————>| 水平タブいれました ↵
| 改行します ↵
| 改行しました ↵
| 復帰しました ↵
| ダブルクォートを表示します ↵
| " ↵
| 円マークを表示します ↵
| ¥ ↵
```

整数リテラルと進数表現

整数リテラルを用いて整数を表現するとき、12 や -4 のように通常は 10 進数を用います
このほかに、8 進数や 16 進数での整数の表現が可能です

10 の 8 進数表現

$$10_{(10)} = 12_{(8)}$$

012

0 で数値を始める → 8 進数表現とみなされます

例えば、`System.out.println(012);` は 10 と出力されます

10 の 16 進数表現

$$10_{(10)} = A_{(16)}$$

0xA

0x で数値を始める → 16 進数表現とみなされます

例えば、`System.out.println(0xA);` は 10 と出力されます



記号 x は大文字でも小文字でもよいです
0xA と 0xA は同じ整数 10 を表現します

10 の 10 進数表現

10

上記以外 → 10 進数表現とみなされます

例えば、`System.out.println(10);` は 10 と出力されます



019 や 0x4g はエラーとなります
これは 8 進数の各桁を 0~7 までを用いて表現するため、
また 16 進数の各桁は 0~15 であり、アルファベットで
0~9、a~f (または大文字 A~F でもよい) を用いて
表現するためです

ソースコード例

ソースファイル名: Sample2_5.java

```
class Sample2_5
{
    public static void main(String[] args)
    {
        System.out.print("10 -> ");
        System.out.print(10);
        System.out.print(", 012 -> ");
        System.out.print(012);
        System.out.print(", 0xa -> ");
        System.out.print(0xa);
    }
}
```

10 → 10, 012 → 10, 0xa → 10

(10)	(8)	(16)
0	0	0
1	1	1
2	2	2
⋮	⋮	⋮
8	10	
9	11	9
10	12	A
11		B
12		C
⋮		⋮
15		F

実行画面

10 -> 10, 012 -> 10, 0xa -> 10

浮動小数点数リテラルと指数表現

浮動小数点数リテラルを用いて実数を表現するとき、通常 3.14 や 0.2 のように書きます
しかし、非常に大きな実数や小さな実数には指数表現を用いると大変に便利です

たとえば、

1.2e+2 は 1.2×10^2 を表現します

System.out.println(1.2e+2); とすると、120.0 と出力されます

1.0e-2 は 1.0×10^{-2} を表現します

System.out.println(1.0e-2); とすると、0.01 と出力されます



記号 e は大文字でも小文字でもよいです

1.2e+2 と 1.2E+2 は同じ実数 120.0 を表現します

ソースコード例

ソースファイル名: Sample2_6.java

```
class Sample2_6
{
    public static void main(String[] args)
    {
        System.out.print(1.2e+2 -> 120.0);
        System.out.print(1.2e+2);
        System.out.print(1.0e-2 -> 0.01);
        System.out.print(1.0e-2);
    }
}
```

1.2×10^2

1.0×10^{-2}

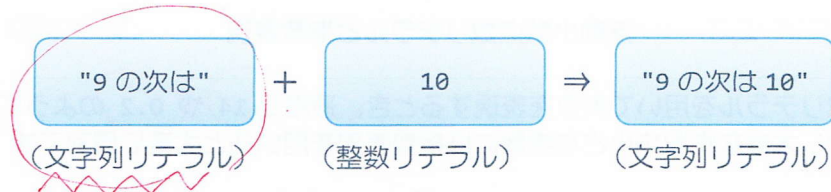
実行画面


1.2e+2 -> 120.0, 1.0e-2 -> 0.01

文字列リテラルと他のリテラルとの連結

文字列リテラルは、整数リテラルや文字リテラルなど他のリテラルと + により連結できます
連結を上手に用いると、画面出力をするコードがすっきり見やすくなります

たとえば、"9の次は" + 10 とすると、"9の次は10"と連結されます



 連結の記号 + は演算子といいます
演算子 + の詳しい解説は「演算子」の回で行います
ここでは演算子 + と連結の機能を押さえておきましょう

ソースファイル名 : Sample2_7.java

```
class Sample2_7
{
    public static void main(String[] args)
    {
        // 文字列リテラル"9の次は"と整数リテラル10を連結します
        System.out.println("9の次は"+10);

        // 文字列リテラル"πは"と浮動小数点数リテラル3.14を連結します
        System.out.println("πは"+3.14);

        // 文字列リテラル"Aの小文字は"と文字リテラル'a'を連結します
        System.out.println("Aの小文字は"+'a');

        // 文字列リテラル"trueの反対は"と論理値リテラルfalseを連結します
        System.out.println("trueの反対は"+false);
    }
}
```

実行画面

```
9の次は10
πは3.14
Aの小文字はa
trueの反対はfalse
```

■ 今日の講義のまとめ ■

- Java での画面出力の方法にはいくつかあり、行の最後に改行が挿入される場合とされない場合があります。
- Java のコードには、コメント、クラス、インデント、main()メソッド、ブロック、文という構成要素があります。
- Java のコード上での文字や文字列、数値はリテラルと呼ばれる決まった形式で表現されます。'A'などの文字リテラル、"Java"などの文字列リテラル、20などの整数リテラル、20.5などの浮動小数点数リテラルがあります。
- エスケープシーケンスは文字リテラルの特別なものであり、¥マークとアルファベットの2文字で表現されます。'¥n'は改行、'¥t'はタブを画面に出力します。
- 整数リテラルでは16などの10進数表現のほかに、020などの8進数表現や0x10などの16進数表現が可能です。
- 浮動小数点数リテラルでは通常の120.0のほかに1.2e+2のような指数表現が可能です。



