

■ 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");
        :
    }
}
```

ソースコード例

ソースファイル名：Sample2\_1.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!");
    }
}
```

実行画面

```
ようこそJavaへ!
Javaをはじめましょう!
```

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()関数と同様に、`¥n` による改行が可能です

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

`System.out.printf()`とまったく同じふるまいをします

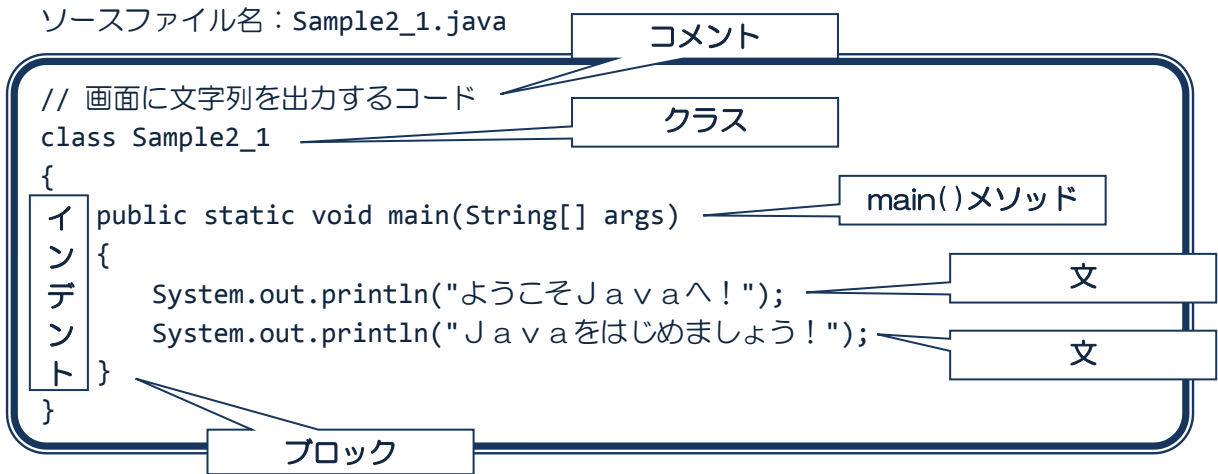
#### ソースコード例

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（行末に改行あり）");
        System.out.print("2. print()による出力（行末に改行なし）");
        System.out.printf("3. printf()による出力¥n");
        System.out.format("4. format()による出力¥n");
    }
}
```

#### 実行画面

1. println()による出力（行末に改行あり）
2. print()による出力（行末に改行なし）
3. printf()による出力
4. format()による出力



- 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

## 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);
    }
}
```

## 実行画面

```
A
Hello
123
0.24
true
```

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現します


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

 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

## 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("水平タブいれませす¥t 水平タブいれませした");
        System.out.println("改行します¥n 改行ませした");
        System.out.println("復帰します¥r 復帰ませした");
        System.out.println("ダブルクォートを表示ませませ");
        System.out.println('¥');
        System.out.println("円マークを表示ませませ");
        System.out.println('¥¥');
    }
}
```

## 実行画面

```
バックスペースしまバックスペースませした
水平タブいれませす      水平タブいれませした
改行ませませ
改行ませませ
復帰ませませ
ダブルクォートを表示ませませ
"
円マークを表示ませませ
¥
```

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      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      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);
    }
}
```



## 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  $1.2 \times 10^2$  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  $1.0 \times 10^{-2}$  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 -> ");
        System.out.print(1.2e+2);
        System.out.print(", 1.0e-2 -> ");
        System.out.print(1.0e-2);
    }
}
```

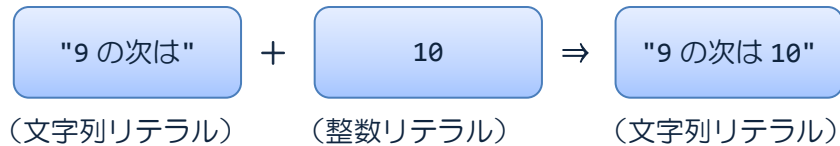
## 実行画面

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」となり連結されます。



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。

