

# Java講習

後期第4回

## 前回やったこと

- ・インスタンスメソッド・コンストラクタ
- ・クラス型の変数の中身は「**インスタンスの場所**」
- ・攻撃メソッドを作った(少し補足します)



## 前回の補足：攻撃メソッドの使い方

前回バタバタしたので, attack()メソッドの動作を確認しておきます

この場合m1がe1に攻撃することになります

```
Mikata m1 = new Mikata("KNIGHT", 100, 20, 80, 15, 12);  
Enemy e1 = new Enemy("ENEMY", 400, 30, 10, 30, 15);  
  
// (Mikata.class) void attack(Enemy enemy);  
m1.attack(e1);  
System.out.println();
```

## 前回の補足：攻撃メソッドの使い方

このように出力されます. ちゃんとHPが減っていますね！

```
KNIGHT HP : 100, MP : 20, ATK : 80, DEF : 15, SPEED : 12
ENEMY HP : 400, MP : 30, ATK : 10, DEF : 30, SPEED : 15
```

KNIGHTはENEMYに50ダメージを与えた  
ENEMYのHPは残り350

```
KNIGHT HP : 100, MP : 20, ATK : 80, DEF : 15, SPEED : 12
ENEMY HP : 350, MP : 30, ATK : 10, DEF : 30, SPEED : 15
```

# 今回やること

1. Javaのライブラリ
2. ライブラリの例 - Mathクラス
3. 色指定printfを作ろう！
4. コマンド選択を実装する



# ライブラリとは？

プログラミングを勉強していると,

「**同じような処理**」を「**何回も**」書く必要があるのが分かってきます

例) 数学関連(累乗, 組み合わせ, 乱数, 暗号化)

データ構造関連(ソート, リスト, 木構造, ハッシュ)

入出力関連(キーボードの入力, マウスの入力, スピーカーの出力)

ネットワーク関連

GUI関連

「これをいちいち作るんですか？」



# ライブラリとは？

そういった「汎用的な」処理を先人たちがまとめて1つにしたものを  
**ライブラリ**といいます(クラスやメソッドの集まり)

Javaはライブラリが豊富な言語として名を馳せていま"した"

PythonやJavascriptといったスクリプト言語や, Java互換のあるKotlinやScalaの登場によって影が薄くなりつつある



# APIリファレンス

ライブラリに「どんなメソッド・定数があるのか」を調べるには  
**APIリファレンス**を見るのが一番早いです

static long	<code>max(long a, long b)</code> 2つのlong値のうち大きいほうを返します。
static double	<code>min(double a, double b)</code> 2つのdouble値のうち小さいほうを返します。

こんな感じで、ライブラリに存在するメソッドの情報が書かれています。

どの言語でもこういったものがあるので  
**必ず読みましょう！**



# import文

javaでライブラリを使うにはimport文を使う

```
import <パッケージ名やクラス名>;
```

パッケージはクラスをさらに包むものだと思います

まあ「**習うより慣れよ**」なので早速使ってみましょう



# ライブラリを使った開発のプロセス

1. 「こういう機能が欲しい！」
2. どんなことが出来るのかAPIリファレンスを見る
3. その中から使えそうものを探す
4. (サードパーティ製の場合は)ライブラリを使えるように環境構築する
5. importする
6. 使う(必要に応じて改造する)



# Mathクラス

数学の関数などを扱うにはMathクラスを使います

ブラウザで「java math」と調べると多分一番上に出てきます

java.lang

クラス**Math**


java.lang.Object

java.lang.Math

---

```
public final class Math
    extends Object
```

Mathクラスは、指数関数、対数関数、平方根、および三角関数といった基本的な数値処理を実行するためのメソッドを含んでいます。



# Mathクラス

何をimportすればいいのかはここに書いてあります.

## クラスMath

java.lang.Object

java.lang.Math

つまりimport **java.lang.Math**;と書けばMathクラスを読み込める

# APIドキュメントの見方

例えばaのb乗した数を求めるメソッドを探しましょう

```
static double
```

```
pow(double a, double b)
```

```
1番目の引数を、2番目の引数で累乗した値を返します。
```

それっぽいのがありました

(累乗は英語で"power"(それだけじゃないけど))



# APIドキュメントの見方

```
static double pow(double a, double b)
```

**〜返り値** **引数**  
1番目の引数を、2番目の引数で累乗した値を返します。

先頭についてるdoubleは返り値, (double a, double b)は引数を表す.

このメソッドを"**pow(3.0, 2.0)**"みたいに呼べば"**9.0**"みたいに返ってくる！  
(Java公式が用意したものなので, 信じて使いましょう)

# ちょっと待って

しれっとMath"クラス"といったけどライブラリじゃないの？

しかも先頭のstaticは何なのか説明してないよね？

---

static double

pow(double a, double b)

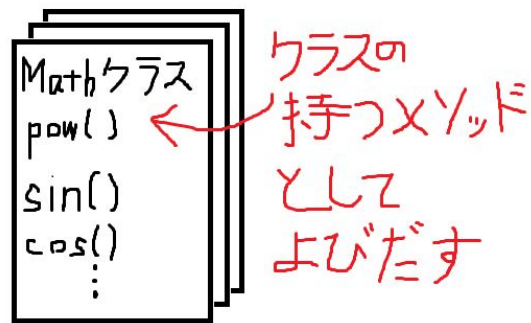
1番目の引数を、2番目の引数で累乗した値を返します。

---



# Javaのライブラリ

実は, Javaではクラスを使ってメソッドや定数をまとめている !



なので `m1.pow()` みたいな **雰囲気** で呼び出す



# staticがあるときと無いとき

メソッドの先頭のstaticの有無で呼び出し方が違う！

- ・staticが無いとき

```
Mikata m1 = new Mikata(); // インスタンスを生成し  
m1.printStatus();         // インスタンスのメソッドとして呼び出す
```

- ・staticがあるとき

```
Mikata.gameEnd(); // クラス名.メソッド名と呼び出す  
⇒インスタンス生成をしない！
```

staticの意味は詳しく解説しません (知ってたら当然だなと感じるはず)

# つまり

---

```
static double pow(double a, double b)
```

**——~返り値** **——引数**  
1番目の引数を、2番目の引数で累乗した値を返します。

---

このメソッドを使うには

**import java.lang.Math;**を記述して

**Math.pow(3.0, 2.0)**みたいに呼べば**9.0**みたいに返ってくる

ということがAPIリファレンスからわかる

## 実際に動かしてみる

```
import java.lang.Math;

public class MathTest {
    public static void main(String[] args) {
        double ans = Math.pow(3.0, 2.0);

        System.out.println(ans);
    }
}
```

```
C:\Users\user>cd C:\Users\user\Java\Java-lecture\4_library>java MathTest
9.0
```

# 用意されたものを使うのもいいですが

せっかくなので, 自分用のライブラリを作ってみましょう



# 出力をカラフルにしよう

下に変なコードがありますが、  
これはターミナルの色を変える処理をしています

```
public static void main(String[] args) {  
    System.out.printf("\u001b[00;31m");  
}
```

```
C:\Users\fnnk2\Documents\Program\Java\Java-lecture\4_library>java ColorPrint
```

```
C:\Users\fnnk2\Documents\Program\Java\Java-lecture\4_library>
```

```
C:\Users\fnnk2\Documents\Program\Java\Java-lecture\4_library>
```

```
C:\Users\fnnk2\Documents\Program\Java\Java-lecture\4_library>
```

```
C:\Users\fnnk2\Documents\Program\Java\Java-lecture\4_library>
```

# 色と文字列の対応

色と文字列の対応表はこんな感じです

```
red      = "\u001b[00;31m";  
green    = "\u001b[00;32m";  
yellow   = "\u001b[00;33m";  
purple   = "\u001b[00;34m";  
pink     = "\u001b[00;35m";  
cyan     = "\u001b[00;36m";  
def      = "\u001b[00m";
```

(**def**は元々の色に戻す文字列)



# 使い方

実際に使うときはこんな感じ...

```
System.out.printf("\u001b[00;31m"); // 赤色にする
System.out.printf("あおいろ");
System.out.printf("\u001b[00m");    // 色を戻す
```

```
C:\Java\Java-lecture\4_library>java ColorPrint
あおいろ
C:\Java\Java-lecture\4_library>
```

ちょっと行数も多くて面倒だなーという感じです

# じゃあメソッドにしようよ

ならばカラフルな出力をする沢山メソッドを作って  
クラス(ライブラリ)にまとめてしまおう！

早速ColorPrintクラスを作りました！





# 早速実装した

こんな感じにしました.

```
public class ColorPrint {  
    static final String def    = "\u001b[00m";  
    static final String red    = "\u001b[00;31m";  
  
    public static void redPrintf(String format, Object ... args) {  
        System.out.printf("%s", red);  
        System.out.printf(format, args);  
        System.out.printf("%s", def);  
    }  
}
```

**final**は変数の値が**変更不可能**であることを表すもので  
引数の"**String format, Object ... args**"はprintfと同じものです  
**staticメソッド**としていることに注意！

# 使ってみる

さっそく呼び出してみましょう

```
public static void main(String[] args) {  
    ColorPrint.redPrintf("あおいろ\n");  
    ColorPrint.redPrintf("1 = %d\n", 1);  
}
```

```
Java\Java-lecture\4_library>java ColorPrint  
あおいろ  
1 = 1
```

これは便利！



# コマンド入力ができるように改造します

やっぱりRPGといったら自分で何をするかを選択したい

色んなやり方があるけど、一番手軽なのは**キーボードの入力**を使うこと  
キーボードの入力を取得するためのクラスがJavaにはあります

```
java.util
```

```
クラスScanner
```

```
java.lang.Object
```

```
java.util.Scanner
```



# Scannerクラスの使い方

Scannerは、**インスタンス化して使う**タイプのライブラリです

インスタンスメソッド**nextInt()**でキーボードから整数値を取得できます

```
import java.util.Scanner;

public class ScannerTest {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int key = sc.nextInt(); // 何かが入力されるまで停止

        System.out.printf("key = %d\n", key);

        sc.close();
    }
}
```

```
Java\Java-lecture\4_library>java ScannerTest
5
key = 5
```

# 応用

whileとif-elseを組み合わせればこんなこともできます

```
int key = -1;
while (true) {
    key = sc.nextInt(); // 何かが入力されるまで停止
    if (key == 0) {
        System.out.println("さよなら");
        break;
    } else if (key == 1) {
        System.out.println("おはよう");
    } else if (key == 2) {
        System.out.println("こんにちは");
    } else if (key == 3) {
        System.out.println("こんばんは");
    }
}
```

```
Java\Java-lecture\4_library>java ScannerTest
1
おはよう
2
こんにちは
3
こんばんは
9
0
さよなら
```

# コマンド入力の実装方針

if文の中を「**そのコマンドで行う処理**」に置き換えれば  
コマンド入力を実装できる！

```
int key = -1;
while (true) {
    key = sc.nextInt(); // 何かが入力されるまで停止
    if (key == 0) {
        // コマンド1
    } else if (key == 1) {
        // コマンド2
    } else if (key == 2) {
        // コマンド3
    } else if (key == 3) {
        // コマンド4
    }
}
```

# こんな感じにしました

簡単のため, 攻撃コマンドを選択したらm1がe1を攻撃するだけにします

printStatusメソッドは全員のステータスを表示するものです

```
Scanner sc = new Scanner(System.in);

int cmd = -1;          static final int COMMAND_ATTACK      = 1;
while (true) {         static final int COMMAND_QUIT       = 0;
    cmd = sc.nextInt();

    if (cmd == COMMAND_QUIT) {
        break;
    } else if (cmd == COMMAND_ATTACK) {
        m1.attack(e1);
        printStatus(m1, m2, m3, e1, e2);
    }
}

sc.close();
```

# さっき使ったColorPrintを使おう

printStatusメソッドやMikata.attackメソッドには,  
ColorPrintクラスのメソッドを沢山使っています, 開発らしくなってきました

```
static void printStatus(Mikata m1, Mikata m2, Mikata m3, Enemy e1, Enemy e2) {  
    ColorPrint.greenPrintf("[PARTY]\n");  
    m1.printStatus();  
    m2.printStatus();  
    m3.printStatus();  
    ColorPrint.yellowPrintf("[ENEMY]\n");  
    e1.printStatus();  
    e2.printStatus();  
    System.out.println("-----");  
}  
  
void attack(Enemy enemy) {  
    int damage = this.atk - enemy.def;  
  
    System.out.println("-----");  
    System.out.print(this.name + "は" + enemy.name + "に");  
    ColorPrint.redPrintf("%d", damage);  
    System.out.println("のダメージを与えた");  
}
```



# デモ

-----  
[PARTY]

KNIGHT HP : 100, MP : 100, ATK : 10, DEF : 15, SPEED : 12  
WIZARD HP : 85, MP : 35, ATK : 5, DEF : 15, SPEED : 20  
TANK HP : 240, MP : 0, ATK : 20, DEF : 40, SPEED : 5

[ENEMY]

ENEMY1 HP : 390, MP : 30, ATK : 10, DEF : 0, SPEED : 15  
ENEMY2 HP : 1200, MP : 0, ATK : 50, DEF : 5, SPEED : 1

-----  
2

1

-----  
KNIGHTはENEMY1に10のダメージを与えた  
ENEMY1のHPは残り380  
-----

[PARTY]

KNIGHT HP : 100, MP : 100, ATK : 10, DEF : 15, SPEED : 12  
WIZARD HP : 85, MP : 35, ATK : 5, DEF : 15, SPEED : 20  
TANK HP : 240, MP : 0, ATK : 20, DEF : 40, SPEED : 5

[ENEMY]

ENEMY1 HP : 380, MP : 30, ATK : 10, DEF : 0, SPEED : 15  
ENEMY2 HP : 1200, MP : 0, ATK : 50, DEF : 5, SPEED : 1  
-----

# まとめ

- ・staticの有無でメソッドの呼び出し方が異なる
- ・先人の作ったライブラリ群は, かなり洗練されている(安全で速いことが多い)
- ・ライブラリを使うと, 一気に**ゲーム作りっぽ**くなる！



# 演習課題

「何のコマンドを選択したら(攻撃or終了)なのか」が分かりづらいので  
使用できるコマンドを表示するprintCommandメソッドを作ってください

```
static void printCommand()
```

```
[COMMAND]
```

```
1. Attack
```

```
0. Quit
```

```
-----
```



# 参考文献

「オブジェクト指向でなぜ作るのか」平澤 章

「Java(tm) Platform, Standard Edition 8 API仕様」

<https://docs.oracle.com/javase/jp/8/docs/api/overview-summary.html>

