Java講座2017

～第1回～

飛ばしながら基礎

クラスとかもやるよ

0. はじめに

- 今回は知っていることが多いと思うので割と飛ばし気味でいきます
- もし本当に初心者で全く分からない等あればどんどん聞いてください
- 適宜、練習用にクラス作ってください
- 演習のときは指示してますが、それ以外はとくにしてないので…

1. ひな形

- プログラムの中身を書く前にこれだけはまず書く！ってやつ

- 導入の資料でプロジェクトを作った人は「Sample01」を使う
- 作ってない人は分かりやすい名前でプロジェクトとクラスを準備

*Sample01.java　Ex01.java　Sample02.java

```
1  package lesson01;
2
3  import javax.swing.JOptionPane;
4
5  public class Sample01 {
6      public static void main(String[] args) {
7
8      }
9  }
```

- ↑がひな形。これを書いてからmain()の中にプログラム本体を書くのが基本
- Import javax.swing.JOptionPane;　はダイアログ入出力を使うためのもの

2. 変数の型

- Javaでは変数を扱う際に変数名の前に型を指定する必要がある

型名	
int	整数値（例　0, 2, -59,　32467とか）
double	小数値（例　3.14, -76.1275とか）
String	文字列（例　こんにちは, 森永真由美とか）
boolean	真偽値（例　true, false）

- 上記表は一部なので気になる人は調べましょう！！
- あれ？floatは？boolは？

  →floatはdoubleと同じで実数値ですが、doubleの方が表示できる桁が大きいです

  →C#でいうboolはJavaではbooleanです。覚えて！

3. 入出力

- 自分は講義でやった、入力はダイアログ、出力はダイアログまたはコンソールという形でやります
- 俺はコンソール一筋だ！！！！！と言う人がいたらそれでどうぞ！！

- まずは出力

- 右の例の7行目

  System.out.println(文字列)

  でコンソール画面に文字列を表示

- 8行目

  JOptionPane.showMessageDialog

  (null, 文字列)

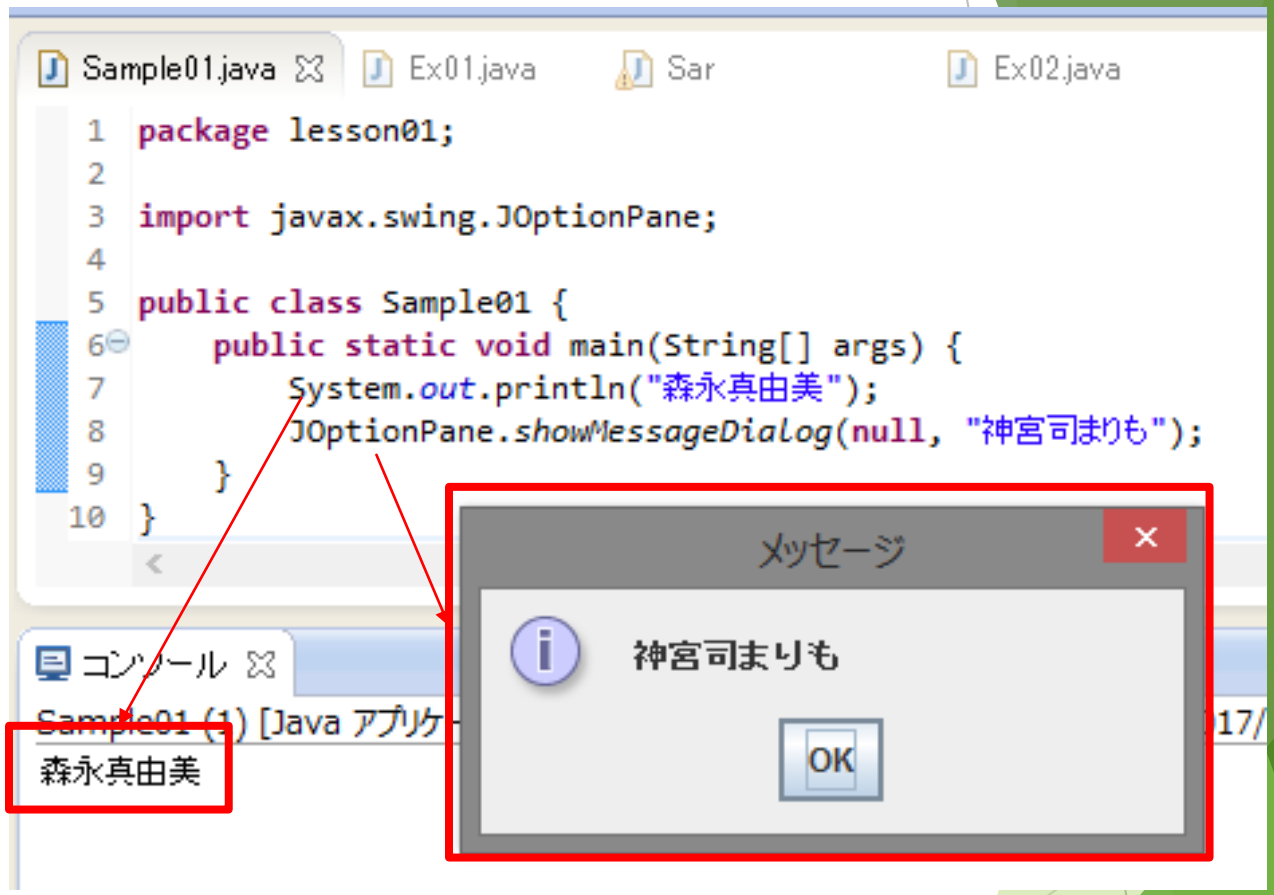
  でダイアログに文字列を表示

  なぜ（null, 文字列）なのか？

  →ダイアログのフレーム（枠）が

  　標準のものでいいよってだけ

- つぎは入力



- 7行目

  JOotionPane.showInputDialog

  (文字列)

  で入力ダイアログを表示

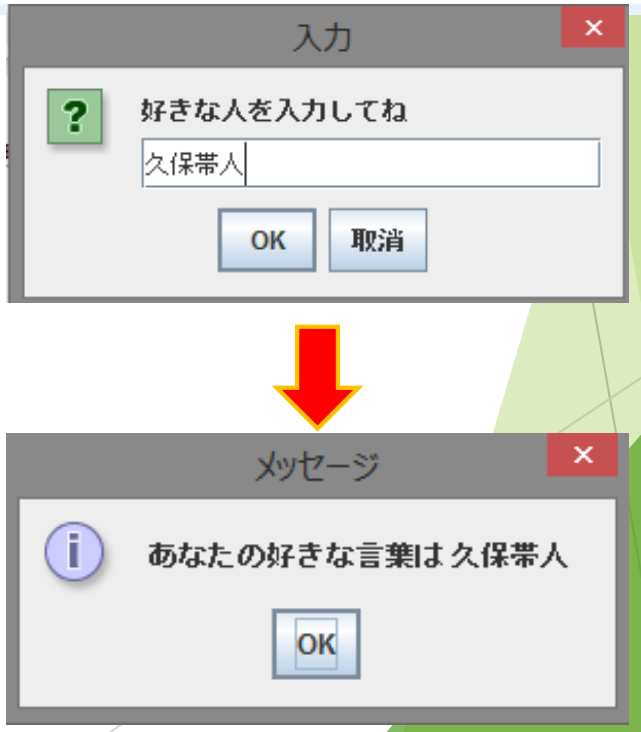
  この時の文字列は入力ダイアログに表示したい文章

  ここで入力されたものは文字列として扱われるため、一度Stringに入れる必要がある

- 8行目で、7行目で入力したものをダイアログに出力する

Sample01.java | Ex01.java | Sample02.java | Ex02.java

```
1  package lesson01;
2
3  import javax.swing.JOptionPane;
4
5  public class Sample01 {
6      public static void main(String[] args) {
7          String message = JOptionPane.showInputDialog("好きな人を入力してね");
8          JOptionPane.showMessageDialog(null, message);
9      }
```



- 数字（整数や実数）を入力したいとき

Sample01.java　Ex01.java　Sample02.java　Ex02.java

```
1  package lesson01;
2
3  import javax.swing.JOptionPane;
4
5  public class Sample01 {
6      public static void main(String[] args) {
7          String message = JOptionPane.showInputDialog("好きな数字(整数)を入力してね");
8          int num = Integer.parseInt(message);
9          System.out.println("文字列の時" + (message + 3));
10         System.out.println("整数の時" + (num + 3));
11     }
12 }
```

コンソール

```
<終了> Sample01 (1) [Java アプリケーション] C:¥Software¥Java¥jdk¥bin¥javaw.exe (2017/07/02 22:44:07)
文字列の時23
整数の時5
```

入力：2
文字列の時 ”2”に+3で23
整数の時 2+3 = 5

- 一度Stringに入れた後変換する

整数→Integer.parseInt(文字列(数値))

実数→Double.parsDouble(文字列(数値))

とすることで文字列から変換が可能です

- いちいちSystem.out.println()とかJOptionPane.showInputDialog()と打つのめんどくさくね？

- System.out.println()
  →「sysout」でCTRL + スペース

- JOptionPane.showInputDialog()
  →「jop」でCTRL + スペース、から「.sh」でCTRL + スペース

- という感じで楽しましょう
- 環境設定によってはできないかも。その場合は見てみます
- （だいたいCTRL + スペースで何とかなる？）

4. 演算子

- 他の言語と一緒

演算子名	
+	加算、文字列結合
-	減算
*	乗算
/	除算
%	余り

- その他シフト演算子とかもちゃんとあるよ

5. 条件分岐

- 他の言語とほぼ同じ ifとswitch

- if(条件文){
  命令
  }else if(条件文){
  命令
  }else{
  命令
  }

```
16      int x=0;
17      if(x==0){
18          System.out.println("あ");
19      }else if(x==1){
20          System.out.println("い");
21      }else{
22          System.out.println("う");
23      }
```

- switch(式){

  case 定数1:

  　　処理;

  　　break;

  case 定数2:

  　　処理;

  　　break;

  …

  default:

  　　処理;

  　　break;

  }

Sample01.java　Ex01.java　Sample02.java

```
1  package lesson01;
2
3  public class Sample01 {
4      public static void main(String[] args) {
5          int x = 0;
6          switch (x) {
7          case 0:
8              System.out.println("あ");
9              break;
10         case 1:
11             System.out.println("い");
12             break;
13         default:
14             System.out.println("う");
15             break;
16         }
17     }
18 }
```

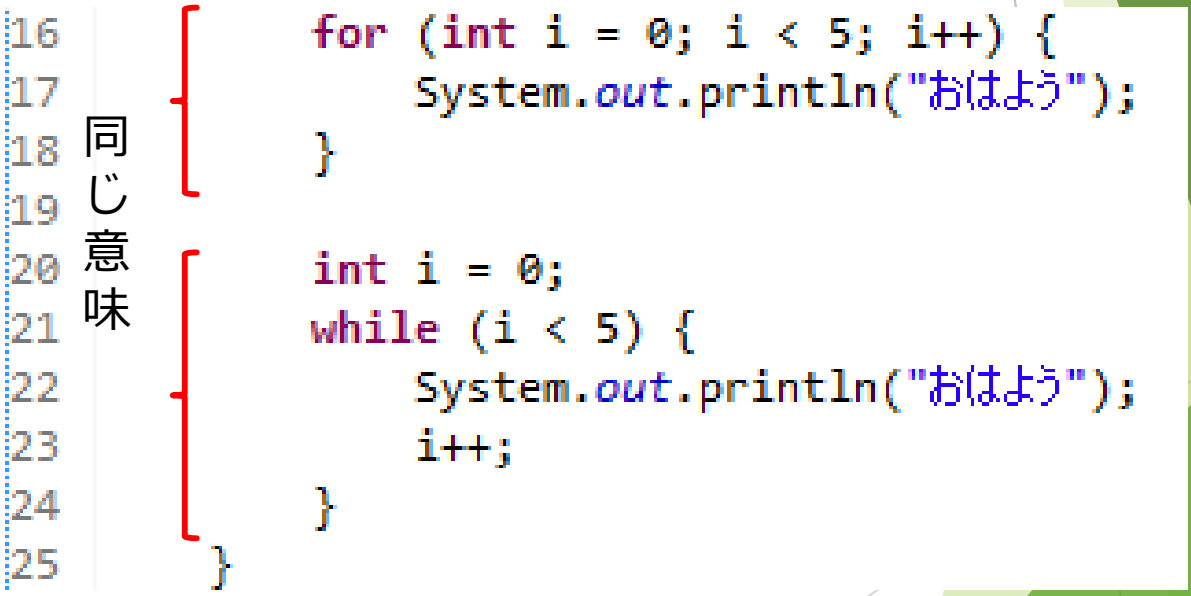
- 式の値が定数1, 2…の場合にその中の処理が行われます
- どの定数にも式が合わない場合default内の処理を行います

6. 繰り返し

- これもいつも通りforとwhileで行います

- for(初期値; 継続条件; 値の加算){
  繰り返す命令
  }

- while(継続条件)(){
  繰り返す命令
  }



- break文とcontinue文

- break文（switch文とか繰り返し以外でもそこそこ使う）

  →繰り返し途中でその繰り返しを抜けるとき使う

- continue文

  →繰り返し途中でそれ以降を無視して次の繰り返しに進むとき

Sample01.java　Ex01.java　Sample02.java　Ex02.jav

```
1  package lesson01;
2
3  public class Sample01 {
4      public static void main(String[] args) {
5          for (int i = 0; i < 5; i++) {
6              if (i == 2) {
7                  break;
8              }
9              System.out.println("今" + i + "回目");
10         }
11     }
12 }
```

コンソール

```
<終了> Sample01 (1) [Java アプリケーション] C:¥Software¥Java¥jdk¥bin
今0回目
今1回目
```

Sample01.java　Ex01.java　Sample02.java　Ex02.java

```
1  package lesson01;
2
3  public class Sample01 {
4      public static void main(String[] args) {
5          for (int i = 0; i < 5; i++) {
6              if (i == 2) {
7                  continue;
8              }
9              System.out.println("今" + i + "回目");
10         }
11     }
12 }
```

コンソール

```
<終了> Sample01 (1) [Java アプリケーション] C:¥Software¥Java¥jdk¥bin¥
今0回目
今1回目
今3回目
今4回目
```

7. 疑似乱数

- ランダムに数字を生み出したいときに使う（じゃんけんとかサイコロとか）

- Math.random()

  →0.0以上1.0未満の実数を適当に出す命令

- ↓こんな感じで使う

```
public static void main(String[] args) {
    double random = Math.random() * 6;
    int dice = (int) random + 1;
    JOptionPane.showMessageDialog(null, dice);
}
```

- randomに0.0~6.0(未満)の実数が入る
- diceにr整数値に変換したrandomを入れ、+1する

→(int)(実数値)　とすることで実数値をint型に変えられます

このとき小数点以下は切り捨て

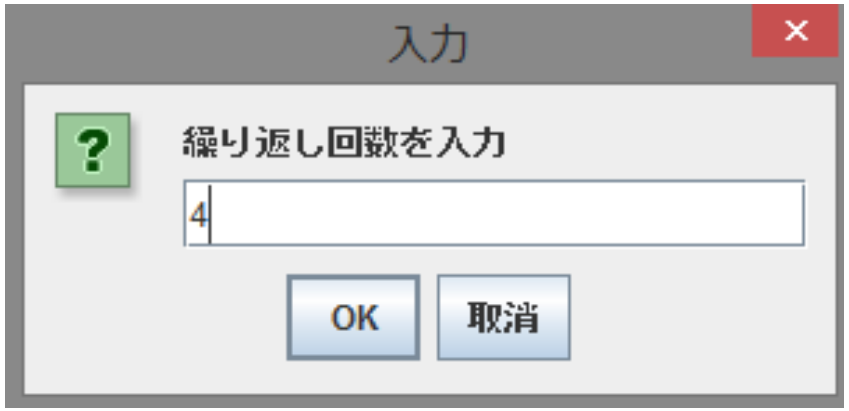
つまり、random：0.0~6.0(未満) → dice：0~5 → +1して

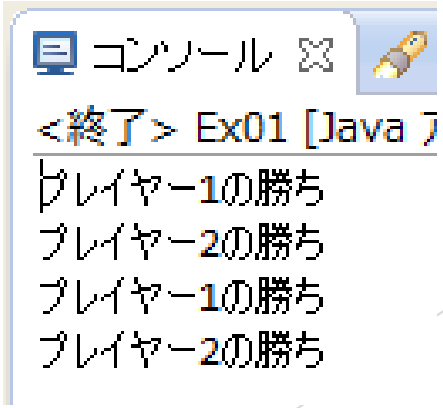
1, 2, 3, 4, 5, 6のどれかが入る！！

- diceを表示する

演習①

- 新規クラス「Ex01」を作成
- 2人でさいころを振って出た値が大きい方を勝ちとするゲーム
- このゲームをコンピュータに複数回させ、勝敗を表示するプログラムを書く
- このときじゃんけんの回数はユーザーが入力するものとする





8. 関数

- 関数（メソッド）はプログラムを小さく分割したときの部品みたいなもの
- コードが何千行とかになったときmain関数だけに書いてたらわけわかんない
- 必要なパーツごとに区切って関数化すると見やすいし分かりやすい！！

- 例をみる

- 演習①の解答例

- 回数入力と勝負を行う部分を
  inputNum, playGameとして関数化

- →のコードは間違いだらけ
  ①変数のスコープ
  ②引数
  ③戻り値

```java
package lesson01;

import javax.swing.JOptionPane;

public class Sample01 {
    public static void main(String[] args) {
        inputNum();
        playGame();
    }

    static void inputNum() {
        String str = JOptionPane.showInputDialog("繰り返し回数を入力");
        int num = Integer.parseInt(str);
    }

    static void playGame() {
        for (int i = 0; i < num; i++) {
            int p1 = (int) (Math.random() * 6 + 1);
            int p2 = (int) (Math.random() * 6 + 1);
            if (p1 > p2) {
                System.out.println("プレイヤー1の勝ち");
            } else if (p1 < p2) {
                System.out.println("プレイヤー2の勝ち");
            } else {
                System.out.println("引き分け");
            }
        }
    }
}
```

- プログラムで使用する変数にはスコープなるものがあります
- 簡単に言うと変数が活動できる範囲です

```
void inputNum() {
    String str = JOptionPane.showInputDialog("繰り返し回数を入力");
    int num = Integer.parseInt(str);
}

void playGame() {
    for (int i = 0; i < num; i++) {
        int p1 = (int) (Math.random() * 6 + 1);
        int p2 = (int) (Math.random() * 6 + 1);
        if (p1 > p2) {
            System.out.println("プレイヤー1の勝ち");
        } else if (p1 < p2) {
            System.out.println("プレイヤー2の勝ち");
        } else {
            System.out.println("引き分け");
        }
    }
}
```

```
String str = JOptionPane.showInputDialog("繰り返し回数を入力");
int num = Integer.parseInt(str);

for (int i = 0; i < num; i++) {
    int p1 = (int) (Math.random() * 6 + 1);
    int p2 = (int) (Math.random() * 6 + 1);
    if (p1 > p2) {
        System.out.println("プレイヤー1の勝ち");
    } else if (p1 < p2) {
        System.out.println("プレイヤー2の勝ち");
    } else {
        System.out.println("引き分け");
    }
}
```

- それぞれ赤いかっこがnumの生存域です
- 基本的に変数は記述された関数内でのみ生きています
- そのためinputNumで書いたnumはplayGameで使用できません

- 違う関数の値を使いたい！というときに登場するものが引数と戻り値です

- 関数は呼び出すときに引数として値を渡すことができます

```
public static void main(
    int n = inputNum();
    playGame(n);
}
```

nという整数
値を渡す

```
static void playGame(int num) {
    for (int i = 0; i < num; i++) {
```

受け取ったnをnum
という名前で使う

- ↑の感じで関数を呼ぶ側呼ばれる側両方で引数設定することで値の引き渡しが可能になります

- 最後にint n = inputNum();について

- 関数は特定の値を返すことであたかもその値であるかのように振る舞えます？

- まずは型
  - void　→　何も返さない　　returnが要らない
  - int　→　整数値を返す
  - double　→　実数値を返す　　returnが必要
  - boolean　→　真偽値を返す

- inputNumもplayGameもvoidです

```
public static void main(
    int n = inputNum();
    playGame(n);
}
```

- 今nに整数値を入れたい　→　inputNum()が整数値みたいになればいい
- inputNumをvoidからintに変える

- 変えると

```
int inputNum() {
    String str = JOptionPane.showInputDialog("繰り返し回数を入力");
    int num = Integer.parseInt(str);
}
```

- エラーが出ます
- これは関数がint型になっているのに何の結果も返していないからです
- intとした場合は関数の最後にreturnとしてintの結果を返す必要があります

```
int inputNum() {
    String str = JOptionPane.showInputDialog("繰り返し回数を入力");
    int num = Integer.parseInt(str);
    return num;
}
```

- 最後のreturn num;　という1行を加えることでintであるnumを返し、

inputNumは整数的に扱うことができるようになります

returnは　値を渡す力を持ったbreak　だと思ってください

- 全体を修正したものがこれ
- →なら解答例のときと同様に動きます

- こんな感じで関数を使うときは気を付けることが多いですが、コードが長ければとてつもなく重宝するようになるので積極的に使いましょう

```java
package lesson01;

import javax.swing.JOptionPane;

public class Sample01 {
    public static void main(String[] args) {
        int n = inputNum();
        playGame(n);
    }

    static int inputNum() {
        String str = JOptionPane.showInputDialog("繰り返し回数を入力");
        int num = Integer.parseInt(str);
        return num;
    }

    static void playGame(int num) {
        for (int i = 0; i < num; i++) {
            int p1 = (int) (Math.random() * 6 + 1);
            int p2 = (int) (Math.random() * 6 + 1);
            if (p1 > p2) {
                System.out.println("プレイヤー1の勝ち");
            } else if (p1 < p2) {
                System.out.println("プレイヤー2の勝ち");
            } else {
                System.out.println("引き分け");
            }
        }
    }
}
```

9. 配列

- 1つの変数名に複数のデータを入れられる

- 型[]　変数名　＝　型[配列の長さ]　or　型[]　変数名　＝　{要素, 要素, …}

```
void start() {
    String[] array=new String[3];
    array[0]="ソル";
    array[1]="マーズ";
    array[2]="ルナ";

    String[] array2={"ソル","マーズ","ルナ"};
}
```

- arrayもarray2も作り方が違うだけで意味は同じ

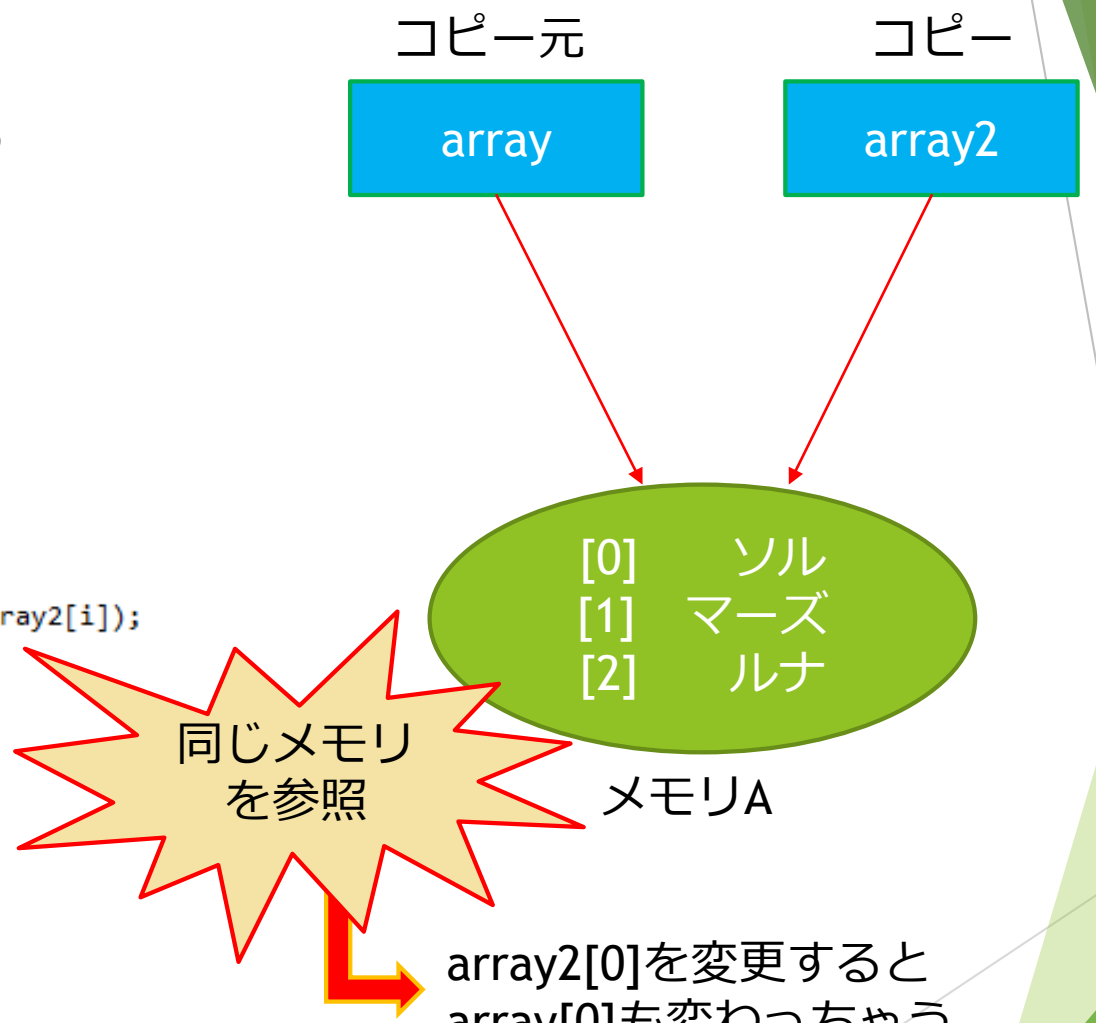
- シャローコピーとディープコピーに気を付ける

- シャローコピー

→新規配列に既存配列をコピーする

```
String[] array = new String[3];
array[0] = "ソル";
array[1] = "マーズ";
array[2] = "ルナ";

String[] array2 = new String[3];
array2 = array;
array2[0] = "ソーラーアクエリオン";

for (int i = 0; i < array.length; i++) {
    System.out.println(array[i] + " , " + array2[i]);
}
```

ソーラーアクエリオン , ソーラーアクエリオン
マーズ , マーズ
ルナ , ルナ

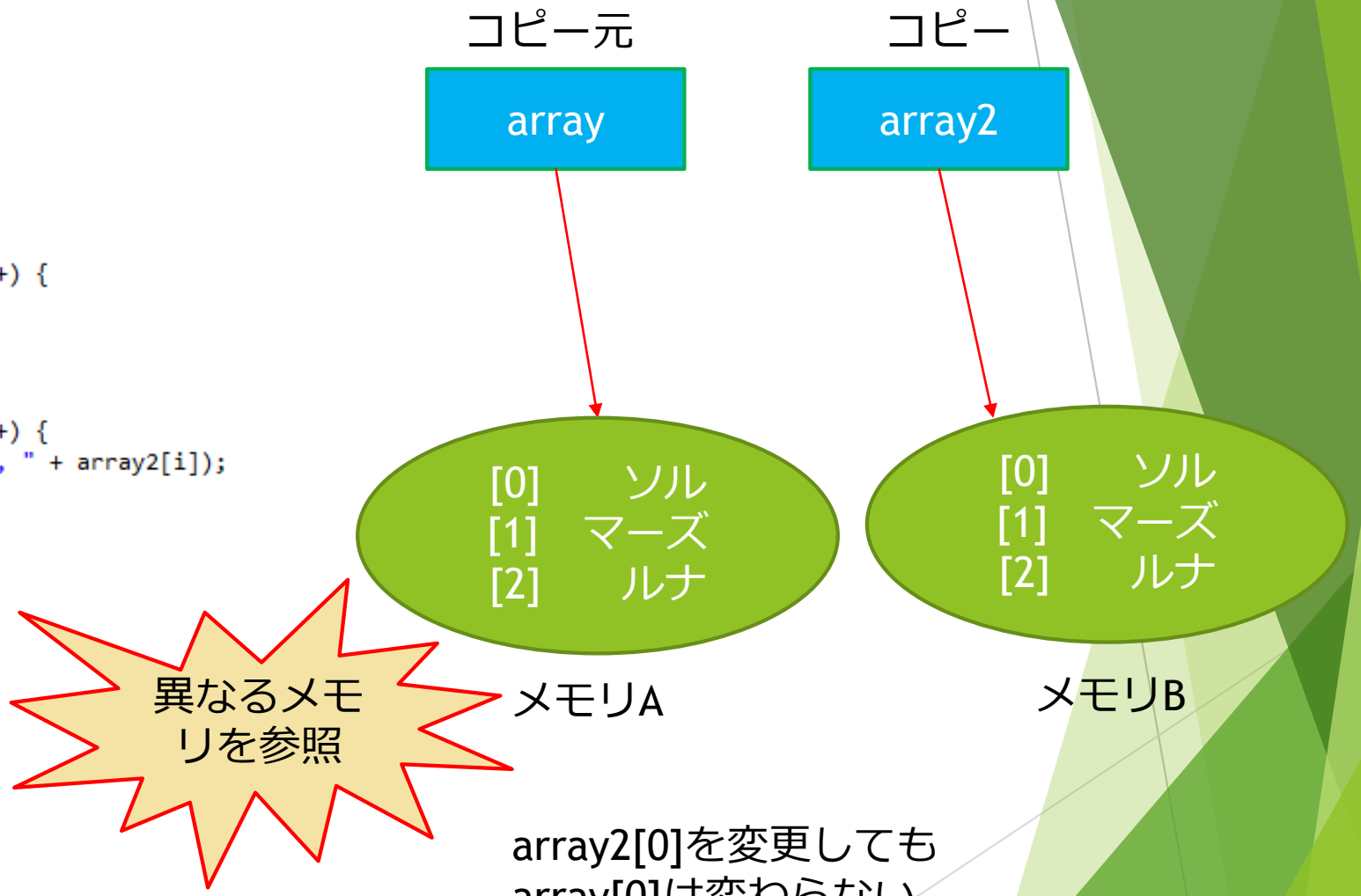


- ディープコピー

→新規配列の要素に既存配列の要素をコピーする

```
String[] array = new String[3];
array[0] = "ソル";
array[1] = "マーズ";
array[2] = "ルナ";

String[] array2 = new String[3];
for (int i = 0; i < array.length; i++) {
    array2[i] = array[i];
}
array2[0] = "ソーラーアクエリオン";

for (int i = 0; i < array.length; i++) {
    System.out.println(array[i] + " , " + array2[i]);
}
```

ソル , ソーラーアクエリオン
マーズ , マーズ
ルナ , ルナ



array2[0]を変更しても
array[0]は変わらない

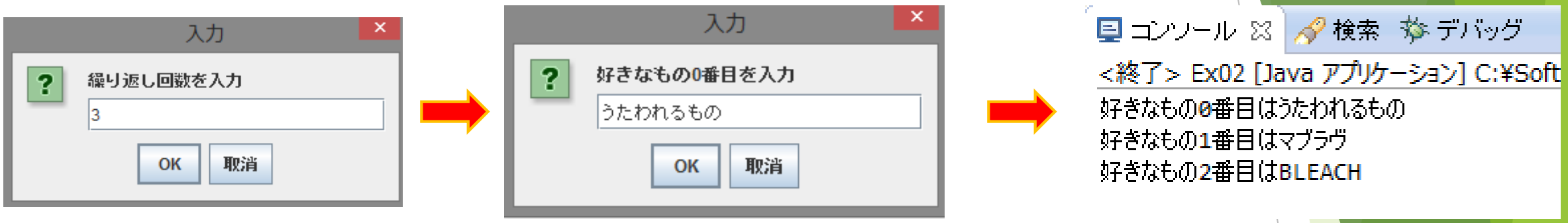
演習②

- 新規クラス「Ex02」
- ユーザが入力した長さのString配列に、ユーザが好きなものを入力
- 入力したデータを最初から順番にコンソールに表示する

- このとき、main関数に全て書かずに
  - ・繰り返し回数を入力する関数
  - ・配列に入力する関数
  - ・配列の中身を出力する関数

  をそれぞれ用意するように
- グローバル変数禁止で！！！！！！！！！！！！

- ↓の感じ



- 関数の型はvoidやint以外にString[]やint[]として配列型にもできます
- 配列の中身をfor文で探索したいとき

→繰り返し回数 = 配列の要素数にしたい

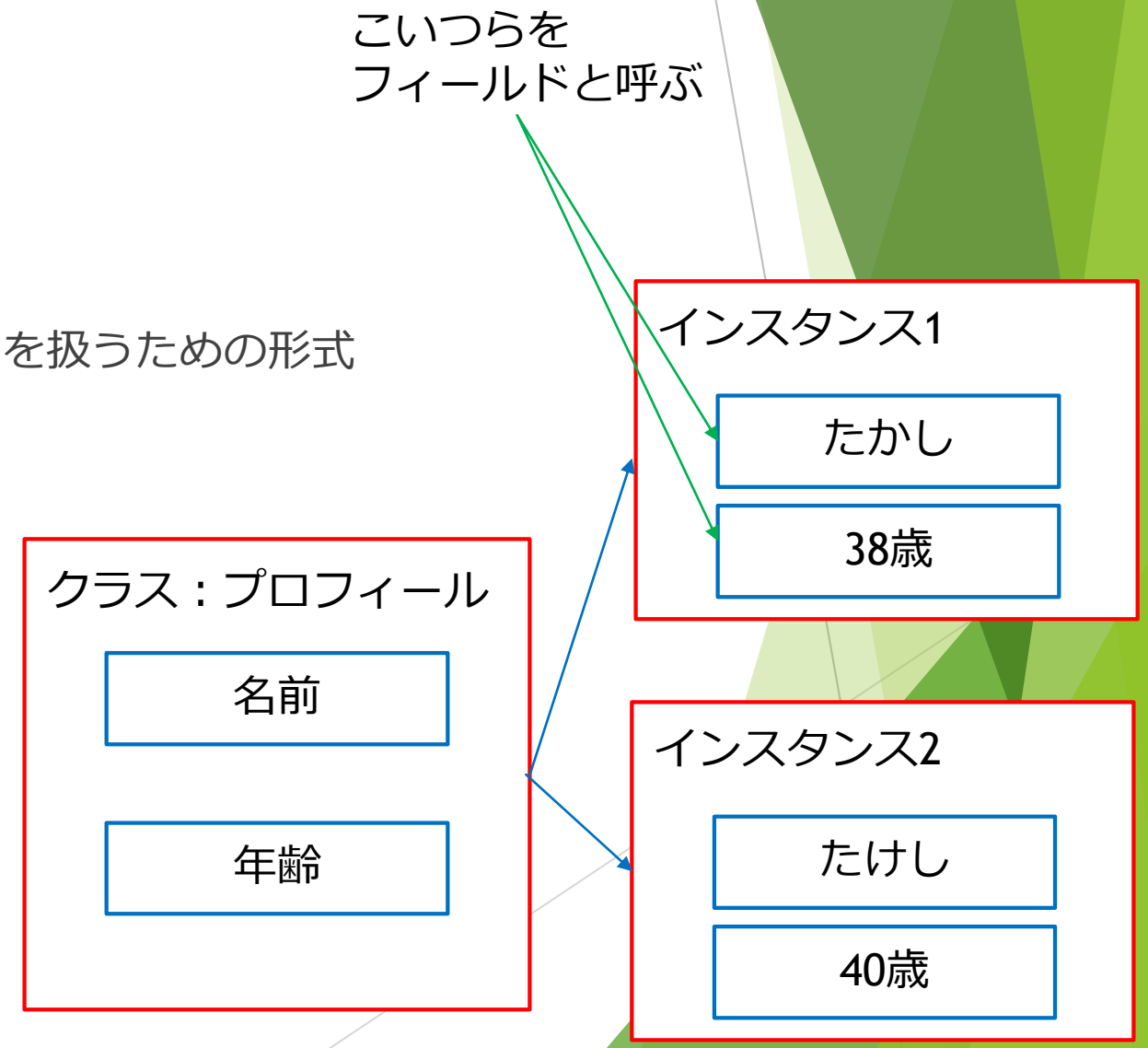
配列の要素数は　配列の変数名.length　で得られます

10. クラス

- この辺からめんどくさくなってきて嫌だ…

- クラスは様々な形のデータを集めた複合データを扱うための形式
- 実際の複合データはインスタンスとして扱う

- クラスは以下の要領で使う
  - ・クラスで複合データの形式を定義
  - ・クラスからインスタンスを生成
  - ・インスタンスから複合データを操作



- クラスの定義

```
class　クラス名{
        型　フィールド名;
        …

        戻り値の型　メソッド名(引数たち){
            処理
        }
}
```

- インスタンスの生成

クラス名　変数名　=　new　クラス名(引数たち)

- 実例↓「SampleClass」というクラスを作ってください

```
package lesson01;

public class SampleClass {
    String name;
    int age;

    void setProfile(String name0, int age0) {
        name = name0;
        age = age0;
    }

    public static void main(String[] args) {
        SampleClass sc = new SampleClass();
        sc.setProfile("たかし", 38);
        System.out.println(sc.name + "さんは" + sc.age + "歳です");
    }
}
```

クラスの定義

フィールドの設定

フィールドに引数の値を設定するメソッド

インスタンス生成

設定

たかしさんは38歳です

- SampleClass型（名前と年齢の情報を持つ）のインスタンスscを生成
- scの名前には“たかし”、年齢には38を入れる
- インスタンスが持つ情報にアクセス

→変数名.フィールド名　とすることで扱うことが可能

- インスタンス生成した後にset~メソッドで引数渡すの書くのめんどくない？

  →そのためのコンストラクタ

- コンストラクタはnew ~(); でインスタンスを生成した時に自動的に呼び出されるメソッド

- クラス名　(引数たち){

  　　処理

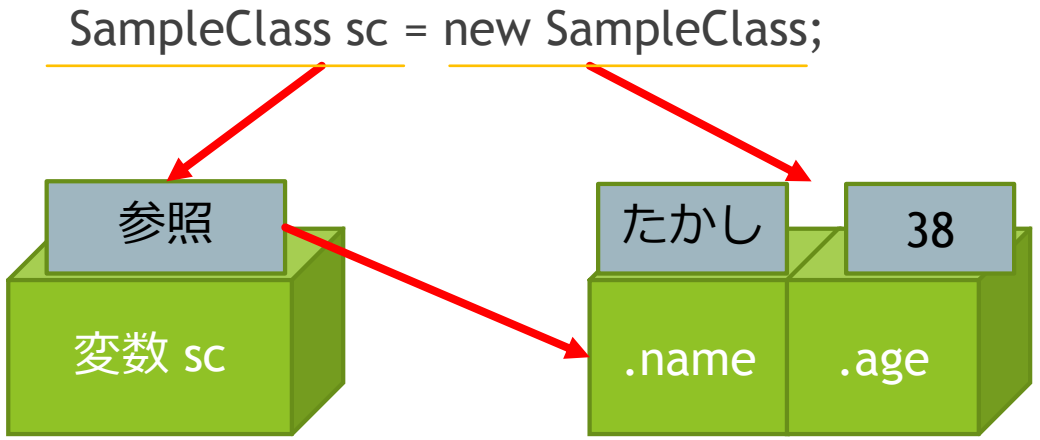
  }

  で書きます

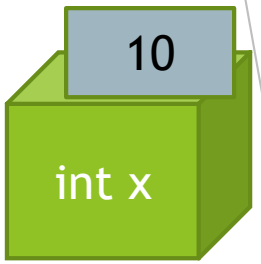
- →は先ほどの例と同じ動き
- this.は覚えよう

```
package lesson01;

public class SampleClass {
    String name;
    int age;

    SampleClass(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public static void main(String[] args) {
        SampleClass sc = new SampleClass("たかし", 38);
        System.out.println(sc.name + "さんは" + sc.age + "歳です");
    }
}
```

- おまけで、参照と実体について軽く…
- インスタンスを生成したときコンピュータのどこかに実体ができる



cf. intなど
int x = 10



- 変数が持つのはインスタンスの実体への参照であり
  変数自体は実体を持たない（実は配列と同じ仕組み）

- そのためscなどの変数をコピーすると参照先をコピーするだけとなり
  実体を変えるとそれを参照する全てに影響を与える
  先ほどの配列のコピーミスはこれにより起きます

11. カプセル化

- 何それ？
- 変数にアクセス修飾子をつけて外部クラスやメソッドからのアクセスを遮断すること
- 意味わかんないけど実はTHEオブジェクト指向な超便利な考え

- たとえば、System.out.println(文字列)とかJOptionPane.showInputDialog(文字列)とか中身がよく分からないし、アクセスもできない。でも文字列を渡すと表示してくることは分かってる
- ↑これが実はカプセル化

- とりあえずアクセス修飾子を覚える

修飾子	クラス内	パッケージ内	サブクラス	外部
public	○	○	○	○
protected	○	○	○	×
(なし)	○	○	×	×
private	○	×	×	×

- 確認のため実際にやってみる
- SampleClassを一部変更
- 変数にprivateを付け

main関数を切り取る

（使うから捨てないで）

```
package lesson01;

public class SampleClass {
    private String name;
    private int age;

    SampleClass(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

- 新規クラス「Sample03」もうある場合は違う名前で
- さきほど切り取ったmainを貼る

```
package lesson01;

public class Sample03 {
    public static void main(String[] args) {
        SampleClass sc = new SampleClass("たかし", 38);
        System.out.println(sc.name + "さんは" + sc.age + "歳です");
}
```

- するとエラーが出る
- これはSampleClassのnameとageがprivateになったため外部クラスである Sample03からアクセスできなくなったから

- privateである値の中身だけを渡す関数を用意する

- SampleClassを変更
- →の2つのメソッドにより

Sample03から直接nameとageにアクセスせずに間接的にアクセス可能になる

```
public class SampleClass {
    private String name;
    private int age;

    SampleClass(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```

- インスタンスの関数もフィールドと同様に変数名.関数名で呼ぶ

```
public class Sample03 {
    public static void main(String[] args) {
        SampleClass sc = new SampleClass("たかし", 38);
        System.out.println(sc.getName() + "さんは" + sc.getAge() + "歳です");
    }
}
```

- これでエラーがなくなり元のコードと同じ結果を得られる

- これがカプセル化
- (java関係ないけど)ゲームだとステータス（HP, MP, ATK...）などに
  直接アクセスすると変化を与える危険があるためこのように間接的にアクセスする
  方が良い（らしい）

演習③

- 新規クラス「Ex03」「Ex03Class」2つ用意
- Ex03Classに名前、英語の点数、数学の点数をprivateで定義
- Ex03で3人分の名前、英語の点数、数学の点数をインスタンスから生成
- それぞれの生徒について、英語と数学の合計点が60点以上のとき
  名前と各点数を表示するプログラムを書く

たかし　60　80
たけし　20　30　　のとき→
たくし　60　10

```
たかしさんの英語は60点、数学は80点
たくしさんの英語は60点、数学は10点
```

- ↓これで用意したクラスを型にして配列が作れます

```
Ex03Class[] students = new Ex03Class[3];
```

12. 色々なデータ構造

- ここではリストとマップ

- リスト

→配列と同じで複数のデータを扱う際に使う

最大の違いは長さが可変であること

操作によってデータの長さが増減する（配列は生成時に固定）

- ここではリストとして2種類挙げる

  ・ArrayList

  ・LinkedList

  何が違うのか？

  →ArrayListは要素にランダムにアクセスするときに

  LinkedListは最初の要素から順番にアクセスするときに有効

- ↓詳細は実行時間について実験している方のページ参照（かまたま日記3）

  http://kamatama41.hatenablog.com/entry/20121024/1351100094

- リストの作り方

- インポートが必要

  import java.util.ArrayList;

- ArrayList<型名>　リストの名前　=　new　ArrayList<型名>();

- 要素を加える時

  リストの名前.add(インデックス, 追加する要素)

  ※インデックス指定が無いとき末尾に追加

- n番目の値を取る時

  リストの名前.get(インデックス)

- LinkedListもArrayListと同様に作ります

- インポートが必要

  import java.util.LinkedList;

- LinkedList<型名>　リストの名前　=　new　LinkedList<型名>();

- それぞれのリストがどのようなメソッドを持つかはjavaのページを見るかeclipseで確認しましょう。多すぎて説明しきれません

↓ArrayListについて

https://docs.oracle.com/javase/jp/6/api/java/util/ArrayList.html

↓LinkedListについて

https://docs.oracle.com/javase/jp/6/api/java/util/LinkedList.html

- ArrayListの例↓（LinkedListでも同様のことができます）

```java
package lesson01;

import java.util.ArrayList;

public class Sample04 {

    public static void main(String[] args) {
        ArrayList<String> array = new ArrayList<String>();
        array.add("a");
        array.add("b");
        array.add("c");

        System.out.println(array.get(1));// bがでる
        System.out.println(array.indexOf("a"));// 0がでる
    }

}
```

- リストの中身を全部表示するとき
- いちいちfor文のi番目の要素を指定してgetして…

→めんどくさい

- 拡張for文を使う！！

- for(要素の型　変数名：リスト名){

　　処理

　}

```
public static void main(String[] args) {
    ArrayList<String> array = new ArrayList<String>();
    array.add("a");
    array.add("b");
    array.add("c");
    array.add("d");

    for (int i = 0; i < array.size(); i++) {//通常パターン
        System.out.println(array.get(i));
    }

    System.out.println();// ただの改行

    for (String str : array) {//拡張for文
        System.out.println(str);
    }
}
```

- マップ

  →キーに関連付けてデータを格納するデータ構造

  講義の課題以外で使ったことないです
- ここではHashMapを挙げる

- インポートが必要

  import java.util.HashMap;

- HashMap<キーの型, データの型>　マップ名 = new HashMap<キーの型, データの型>();

- マップへの追加は

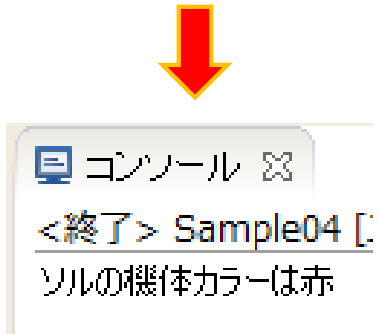
  マップ名.put(キー名, データ)

- データの取得

  マップ名.get(キー名)

- ↓例

```java
package lesson01;

import java.util.HashMap;

public class Sample04 {

    public static void main(String[] args) {
        HashMap<String, String> map = new HashMap<String, String>();
        map.put("ソル", "赤");
        map.put("マーズ", "青");
        map.put("ルナ", "緑");

        String s = "ソル";
        System.out.println(s + "の機体カラーは" + map.get(s));
    }
}
```



演習④

- Listを配列で作る
- せっかくなのでやりましょう

- 新規クラス「Ex04」と「Ex04ArrayList」をつくる

- 「Ex04」
- 右と同じものを用意

- Ex04ArrayListに
  - ・add
  - ・size
  - ・isEmpty
  - ・get

  関数を用意することで
  Ex04が動くようにする

- whileループで6回入力したとき、6人分の名前が表示されるプログラム

```
package lesson01;

import javax.swing.JOptionPane;

public class Ex04 {

    public static void main(String[] args) {
        Ex04ArrayList list = new Ex04ArrayList();
        while (true) {
            String input = JOptionPane.showInputDialog("名前を入力してください");
            if (input.equals("end")) {
                JOptionPane.showMessageDialog(null, "入力を終了します");
                break;
            }
            list.add(input);
        }

        for (int i = 0; i < list.size(); i++) {
            if (!list.isEmpty()) {
                System.out.println((i + 1) + "番目の人は" + list.get(i));
            }
        }
    }
}
```

- void add(String)　→　リストの最後にStringを追加
- int size()　→　リストのデータ長を返す
- boolen isEmpty()　→　リストが空の時trueを返す
- String get(int)　→　リストのintの要素を返す

- size ～ ADD_SIZEまでの変数はグローバル変数といい
  変数のスコープを無視してこのクラス内のどこでも
  使うことの出来る変数です

- コンストラクタで値の初期化
- 残りを実装

```
package lesson01;

public class Ex04ArrayList {
    private int size;
    private int capacity;
    private String[] arrays;
    final int INITIAL_CAPACITY = 5;
    final int ADD_SIZE = 5;

    Ex04ArrayList() {
        capacity = INITIAL_CAPACITY;
        size = 0;
        arrays = new String[capacity];
    }

    void add(String name) {

    }

    int size() {

    }

    String get(int index) {

    }

    boolean isEmpty() {

    }
```

- 本日はここまで
- 時間の都合上講義でやるほど細かく教えられないので何かあったら聞いてください

- 次回（総会後）は
  - ・嫌いな継承
  - ・つまんねぇ例外処理
  - ・苦手なファイル操作

  についてやるつもりです

お疲れさまでした！