JAヴぁ講座　第三回

大橋　山崎

本日の内容

・今までのまとめ集（好きな時に見る用）
・クラス、インスタンスについて。（喋ります用）
・発展問題的な問題（出来る人からどんどんやっちゃってください）

※注意※
教科書とか見ないでうろ覚えでスライドつくったので
用語の使い方とか適当です。間違ってない自信はないので、
雰囲気で理解して、盲信しないでください。
あと、コードは適当です。そのまま書いたら多分エラーになるかも。

まとめ集

入力・宣言１

```
//変数の宣言
String　僕のパンツの色　＝　“ピンク”;
    //Stringは本当は new String();してる
int number1 = 2014;
double number2 = 3.1415;
String input = JOptionpane.showInputDialog(“入力してください”);
int value = Integer.parseInt(input);//Stringの数をint型にしてくれる
```

・「型名　変数名　＝　変数に入れる値(または値を返すメソッド)；」
・変数名はひらがなでも漢字でもギリシャ文字（λとかψ）でも大丈夫

入力・宣言２

参照型の宣言

int[] array1 = new int[10];//配列の生成

Int[] array2 = {1,2,3,4,5,6,7,8,9,0};//配列の生成＆格納

Dog dog = new Dog();//インスタンスの生成

Dog[] mydog = new Dog[3];

・型名　変数名　＝　new 型名();

・型名[] 変数名　＝　new 型名[大きさ];

出力

System.outprintln(僕のパンツの色);//コンソールに出る//改行する
System.outprint(number1)//コンソールに出る//改行しない
JOptionpane.showMessageDialog(null,number1);//ダイアログに出る

System.outがあるならsystem.inもあるだろ！ってのは妥当な考え方。
だけど、system.outほど何も考えずに簡単には使えない。希ガス。
Scanner とか使うと思う。

演算子1

＋は足す演算子。- は引く演算子。*はかける演算子。/は割る演算子
計算する項を()で囲むことで優先的に計算してくれる。
%は割った余りを導く演算子。いわゆるmod
&&　は　Andの意味の論理演算子。
|| は　Orの意味の論理演算子。
＝ は「左辺の内容を右辺に代入」の意味
＝＝ は「右辺と左辺が等しい」の意味の論理演算子
！＝ は「右辺と左辺が等しくない」の意味の論理演算子
！は真偽をひっくり返す。
　⇒Booleanメソッドを起動するとき、メソッド名の先頭に！をつけると
真偽が逆転する。　　（例） if(!return_True() == false){//実行される}

演算子2

①数系変数 + 数系変数 は和を表す。
String型変数 + 数系変数 は合体する。
例）

```
int a = 7,b = 3;
```

Sysout(“a + b =”+ a + b);とsysout(“a + b = ”+(a + b));の違い。
左記はa + b = 73になって、右記はa + b = 10 になる。
②変数名++;はインクリメント。変数名--;はデクリメント。
変数名　+= 3; は変数名 = 変数名 + 3;の意味。
そんな感じで、 -= や　*= や　/= や %= が使える。（便利ってだけ）
③3項演算子　(条件式?trueの時の処理:falseの時の処理)
詳しくはサンプルコードで

入出力・演算子のサンプルコード
暇なときにでも
コピー＆ペーストして動作確認してみて。
クラス名は「asobi」になってます

```
public class asobi {


public static void main(String[] args) {
new asobi().start();
}


void start() {
String 名前 = "デデンネ";
String name = new String("ピカチュウ");
int[] array1 = new int[2];
array1[0] = 0;
array1[1] = 1;
int[] array2 = { 1, 2 };
System.out.println(名前 + "," + name + "," + array1[0] + "," + array2[1]);
double a = 3.8;
double b = 6.2;
int c = 5;
System.out.println("a + b =" + a + b);
System.out.println("a + b =" + (a + b));
if ((!returnTrue() == false && returnTrue()) && (!false)) {
System.out.println(false);
}
c++;
c--;
c += 1;
c -= 1;
c *= 2;
```

配列

・値の列。
・インデックスは0から始まる。つまり、一番目がarray[0]に格納される。
・for文と相性が良い。
・配列名.lengthで配列の長さを得ることが出来る。
・Stringの入力.split(“,”)で入力結果をカンマで区切って配列に格納する

宣言形式：double[] array = new double[3];みたいな
格納の仕方: array[0] = 0.0;
　　array[1] = 3.14;みたいな

For文

・基本的に

for(for文内で使う変数宣言;for文が回る条件;回る度に行う処理){

For文によって回される命令文;

}

・具体的に

For(int i = 0; i < 3;i++){

Sysout((i+1)+”回目”);//1回目,2回目,3回目と変わっていく

}

For文

・応用?

```
int i= 0;
for(;i  < 3;){
i++;
}
```

つまり、for文の中は回る条件さえ書いてあれば十分ってこと。
二つ以上変数を宣言してもいいし、
二つ以上再帰時の処理を書いてもいい。

・だから、書こうと思えばfor文は1行で書ける。（サンプルコード参照）

While文

・回る回数があんま良く分からないfor文みたいなもの。

while(回り続ける条件){

処理

}

・回り続ける条件が満たされている間はずっと回り続ける。

そのため、条件をtrueとした場合、無限ループになるのでbreakが必要。

・breakは、その時点で強制的にwhile文を抜けることを意味し

continueは、その時点で強制的にwhile文の最初の処理に再帰させることを意味する。

While文とif文

・応用例(１や２以外の値が入力されても対応が出来るようになる)←対話型プログラムってこと

```
While(true){
int input = 入力（告白されちゃった！どうしよう?!(1)OKする(2)断る）;
If(input == 1)
Sysout(私も好きです);
Break;
}else if(input == 2){
Sysout(ごめんなさい);
Break;
}else{
Sysout(どっちか選べよこのやろう);//もっかいやりなおし
}
```

配列、for文,while文,if文サンプルコード（割とクソコード）

```
import javax.swing.JOptionPane;

public class asobi2 {

public static void main(String[] args) {
new asobi2().START();
}

void START() {
// .lengthと.splitについて
String a = "1,2,3,4";
String[] aa = a.split(",");
System.out.println(aa[0] + "," + aa.length);// 1,4
// for文について
int k = 0;// 初期化
for (; k < 2;) {
System.out.println(k + "< 2 だからfor文は回る");
k++;
}
// for文を一行にまとめる(例：1～10の総和を取りたい)
int[] b = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int sum = 0;
for (int i = 0; i < b.length; sum += b[i], i++)
;// for文の中身が空である時はセミコロンで閉じる
// 再帰するときの命令を書く順番が違うとエラー
System.out.println("1から10までの総和は" + sum);// 55
```

メソッド

Int add(int a,int b){ //このaとbは起動側から参照してる値にすぎない
return (a+b)　　ので、仮引数みたいに呼ばれる。
}

Int answer = add(5,8);//メソッドに送る値は実引数と呼ばれる。
Int answer2 = add(10,answer);

Int型の結果を返すからint でメソッドの型を指定している。

メソッド

Void a(){} は何も値を返さないメソッド。ただ{}内の処理を実行するだけ
Int a(){}はintの結果を返す
String a(){}はStringの結果を返す
double a(){}はdoubleの結果を返す
boolean a(){}はtrueかfalseを返す
Int[] a(){}はintの配列を返す
….的な

・return文が無いとエラーを起こす。必ずreturn しよう。
「メソッドの型」と「返す結果の型」をそろえよう。
If文を使って結果に分岐を起こしたら、基本的に分岐の分だけreturn はつくる。

知識

・変数の利用範囲
　⇒あるブロック内で宣言された変数は、そのブロック内でしか使えない

例えば、for(int i =0;略){}における変数iはfor文内でしか使われず、
For文の外では利用できない。

そのため、メソッドの外側（一番大きなブロックの中）に変数を宣言すると
その変数はどこのメソッドでも自由に使える。（だから危険な可能性も有）
　フィールドって言った気がする。

本日の内容（クラスとインスタンス）

それと二次元配列

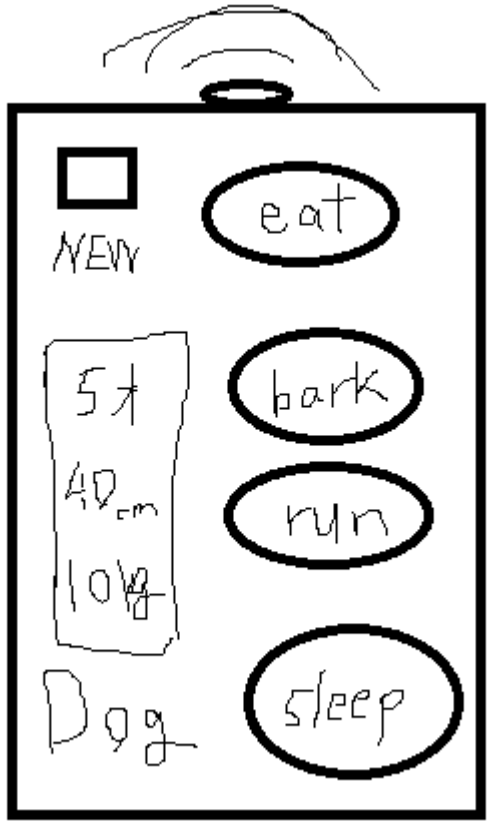
インスタンスって何よ？

• インスタンスとは参照型の変数である。
「参照」とは「ポインタ」と似た概念であるが、javaにおいては
プログラマが参照型変数に実際にどんな値が入っているかを
知る必要はない。つまり、格納されるアドレスを意識する必要が無い。

・インスタンスとはリモコンみたいなもの（個人的イメージ）
次スライドへ

インスタンスって何よ？

・何も出来ない（参照しない）リモコンAがある。
・Dogクラスには
犬Aを操るメソッドと、その犬のステータスが
記述されている。
クラスとはいわば設計図的な物。
このクラスをリモコンAに参照させる
（インスタンスを作る）と、リモコンAで犬Aを
操ることが出来る。また、犬Aの情報を
知ることが出来る。



インスタンスって何よ？

インスタンスの生成方法

Dog リモコンA(変数名) = new Dog();

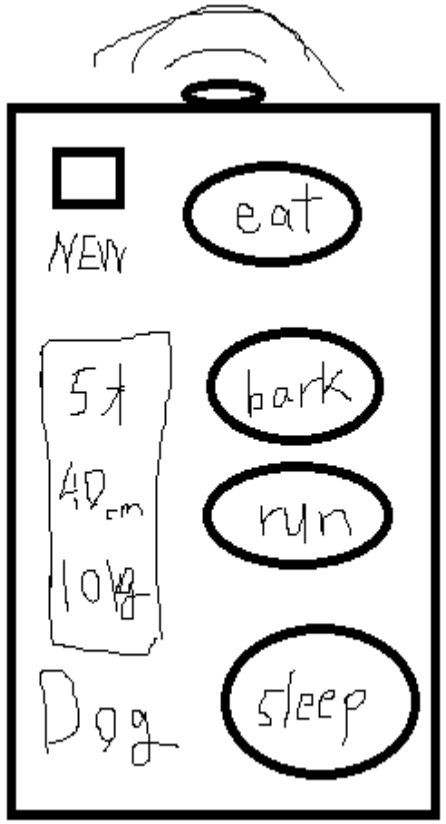
インスタンスの利用

リモコンA.eat()　⇒すると犬がご飯たべる

リモコンA.bark()　⇒すると犬が吠える

リモコンA.age　⇒すると犬の年齢が分かる（取得でき

リモコンA.weight ⇒犬の重さが分かる



インスタンスって何よ？

・もし、同様にCatクラスがあったとする。

しかし、リモコンAはDogオブジェクト（インスタンス）しか操れない。

そこで、別にリモコンBに参照させる。

　Cat リモコンB　= new Cat();

・リモコンCにもCatクラスを参照させる。

Cat リモコンC　= new Cat();

　⇒このとき、リモコンBとリモコンCは同じ役割・性質を担うかもしれないが、参照は別々である。（デフォルトコンストラクタで一致しただけ）

インスタンスって何よ？

・「リモコンB = リモコンC;」という代入操作を行うことで参照が一致する

・「リモコンC = null ; 」とすると、そのリモコンは何も参照しなくなることを
意味する。つまり、リモコンっちゃリモコンだけど、ボタンが一個もない、情報も文字も何もないただの箱になるってこと。

・リモコンBが今まで参照していたオブジェクトは、リモコンCの参照の
代入によって上書きされてしまった。つまり、リモコンBが今まで参照しつづけたアドレスにあるCatオブジェクトはもう利用されることがない。
　⇒ガーベージコレクションによって破棄される。

とりあえず、インスタンスで覚えてほしい事

・クラスとは、オブジェクトを作るのに使う設計図。
・インスタンス＝オブジェクトである。
・インスタンスが持っている変数、状態、をインスタンス変数と呼び
インスタンスが持っているメソッドをインスタンスメソッドと呼ぶ。
・クラスファイルには普通クラスはひとつ。
・クラス名（型名） 変数名　＝　new クラス名();で生成する。
・変数名.nameとか変数名[3].attack();みたいに、
.(どっと)演算子で、インスタンスのデータを利用できる。

複合データとしてのインスタンス

・インスタンス変数はStringだろうがintだろうがdoubleだろうが
いくらでも複雑に保持することが可能。
　⇒配列は「同じ型のデータ」しか保持できなかった。
　　⇒そこで、インスタンス変数を便利な格納表みたいに扱おうという
応用が出来る。

例）String name(←名前を保持する変数)
Double height (←身長を保持する変数)
Int age(←年齢を保持する変数)
をフィールドに持つクラスをインスタンスとして2つくらい生成する。

複合データとしてのインスタンス

Class Human{
さっきの3つのフィールド変数を持ってる。
}

Mainメソッド側{
Human[] human = new Human[2];//生成
Human[0].name = arice;
Human[1].name = bob;
Human[0].height = 149.7;
Human[1].height = 178.3;
Human[0].age = 17;
Human[1].age = 23;//格納
}

Hukugouデータとしてのインスタンス

もしくは、
Mainメソッド側{
Human Arice = new Human();//生成
Human Bob = new Human();//生成
Arice.name = arice;
Bob.name = bob;
Arice.height = 149.7;
Bob.height = 178.3;
Arice.age = 17;//格納
Bob.age = 23;//格納
}
とするのもアリだけど、インスタンスの配列にした方がfor文と相性が良いという点と、
メモリ側においても、ユーザ側においても管理がしやすいという点で、インスタンスを配列に入れるのが
Betterだと思う。

二次元配列。

今までの配列は、まあ１次元だったのです。
なぜなら、大きさ増やしても横方向にしかインクリメントされないから。
じゃあ横方向にインクリメントし終えたら、
縦方向にもインクリメントすればよくない？
ってことで二次元配列が出来ます。
宣言はこんな感じ。Int[][] array = new int[2][4];
どっちが縦でどっちが横かなんてのは考える必要ない。
どうせ高次元になったらわけわかんなくなるし。

二次元配列

値の格納の仕方
Array[0][0] = 2;array[0][1] = 4;って一々やるのはめんどい。
For文の中にfor文を入れることで、回す操作を回すことになるので
きれいに書くことが出来る。

```
For(int I = 0;I < 2;i++){
        for(int j = 0;j < 4;j++){
                array[i][j] = 数字;
        }
```

}こうやると、i=0の間にjが0から3まで回って、
回り終わったら、iがインクリメントされて、またjが回って…..の繰り返しで
上手い事二次元配列に値が格納されていく。

二次元配列

ということは、
[]の数を増やした分だけfor文の中にfor文書けば、効率よく
多次元の配列を考えることが可能になる。

```
Int[][][][] array = new int[1][2][3][4];
 for(){
    for(){
      for(){
       for(){
        for(){
}}}}みたいな
```

二次元配列で覚えてほしい事

・for文の中にfor文を書くことで「回す操作」を「回す」ことが出来る。
・int[][] arrayみたいに多次元にわたって配列を作ることが可能である。