

Javaプログラミング (3)

クラス、インスタンス (1)

成蹊大学理工学部
情報科学科

制御構造

- 様々な制御構造について学ぶ
 - if
 - for
 - switch
 - while

switch 文

```
switch (式) {  
    case 定数式1:  
        処理1  
        break;  
    case 定数式2:  
        処理2  
        break;  
    default:  
        処理3  
        break;  
}
```

- まず、(式)が評価され、その結果が定数式1の値に等しければ処理1が実行され、定数式2に等しければ、処理2が実行される。どの場合にも当てはまらなければ、defaultの処理3が実行される。
- 定数式には、**整数型の定数**を使うこと。(注)String型, double型は使えない
- 各処理の最後には、break文をいれる。これは「ここで処理を終わります」という意味

JDK7以降のswitch文

文字列でも利用できるようになった.

```
String str = "あいうえお";
```

```
switch (str) {
```

```
    case "あいうえお":
```

```
        System.out.println("String switch");
```

```
        break;
```

```
    default:
```

```
        System.out.println("default");
```

```
}
```

switch 文の例1

- 多くの選択肢から1つを選んで実行する

```
switch (n) {  
    case 1:  
        System.out.println("オレンジジュースです");  
        break;  
    case 2:  
        System.out.println("コーヒーです");  
        break;  
    default:  
        System.out.println("どちらでもありません");  
        break;  
}
```

switch 文の例2(文字による分岐)

```
String input = new String ("abc");  
char c = input.charAt(0);  
    switch (c) {  
    case 'a':  
        System.out.println("オレンジジュースです");  
        break;  
    case 'b':  
        System.out.println("コーヒーです");  
        break;  
    default:  
        System.out.println("どちらでもありません");  
        break;  
    }
```

← ゼロ番目の文字 'a' が c に代入される

break文がない場合

```
switch (c) {  
    case '1':  
    case 'a':  
        System.out.println("オレンジジュースです");  
        break;  
    case '2':  
    case 'b':  
        System.out.println("コーヒーです");  
        break;  
    default:  
        System.out.println("どちらでもありません");  
        break;  
}
```

- ・ '1'の時も 'a'の時も同じ処理が行われる.
- ・ break文を忘れると, 次のcaseのところまで進んでしまうので, 忘れないように!
- ・ バグ発見, エラー処理のためにもdefaultはつけるようにしましょう.

while 文

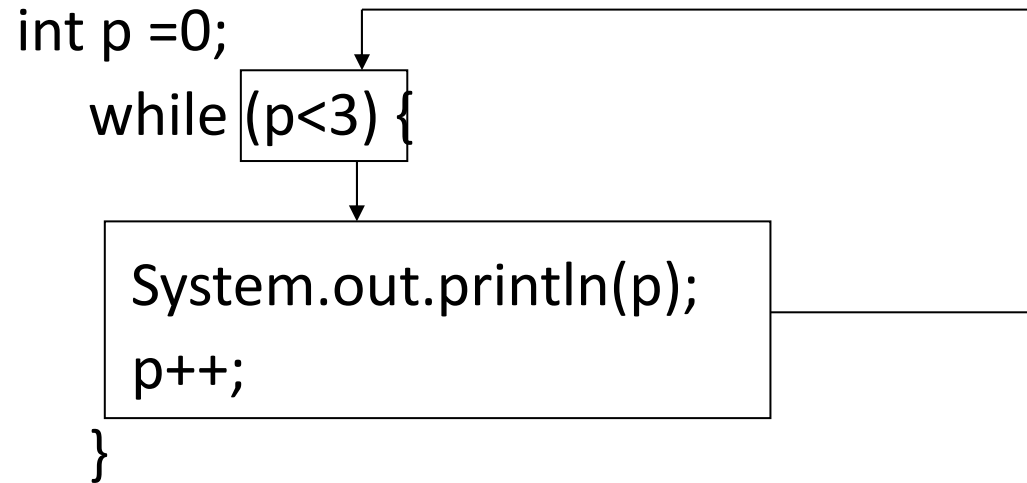
- ある処理を繰り返し行う

```
while (条件式) {  
    繰り返す処理  
}
```

- 条件式はboolean型の式 (if文と同じ)
- 条件式を評価し, その値がtrueである間処理を繰り返す
- 無限ループにするには条件式としてtrueを使う

```
while (true) {  
    繰り返す処理  
}
```


while文の例



- (1) 変数pを宣言, 0で初期化
- (2) 条件式においてpが3よりも小さいか調べる
- (3) 条件が満たされれば, pをプリント
- (4) pの値を1増やす

forやwhileでのbreak文

- forやwhile文の中で処理を中断するときには, break文を用いる

```
while (...) {
```

```
...
```

```
  if () {
```

```
    break;
```

```
  }
```

```
  ...
```

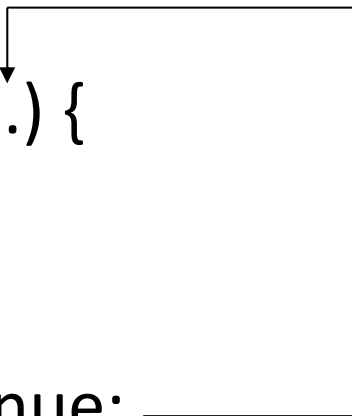
```
  }
```

→ while文を中断して脱出する

continue文

- ループを次に進める

```
while (...) {  
  ...  
  if ( ...) {  
    continue;  
  }  
  ...  
}
```



while文の残りを中断し、
繰り返しを次に進める

配列の復習

- 英語, 数学, 理科の平均点を計算するプログラムを考える.
- 関連した情報をうまくまとめておく方法はないのだろうか?

```
1: public class Average {
2:     public static void main(String[] args) {
3:         double average = 0.0;
4:         average = calAverage();
5:         System.out.println("3教科の平均点は "+average);
6:     }
7:
8:     public static double calAverage() {
9:         double result = 0.0;
10:        int[] score;
11:        score = new int[3]; //配列の宣言
12:        score[0] = 63; //英語の点数
13:        score[1] = 92; //数学の点数
14:        score[2] = 75; //理科の点数
15:        result = (score[0]+score[1]+score[2])/3.0;
16:        return result;
17:    }
18: }
```

科目の名前と点数との
対応が分かりにくい

クラス概念

- ・ クラス: 関連した情報を1つにまとめたもの

① 共通の属性を持つ

科目クラスは科目名と点数という属性を持つ

科目
: 科目名
: 点数

犬クラスの宣言

犬
: 足が4本
: 尻尾がある

① 共通の処理(機能)を持つ

科目クラスは平均値計算という処理を持つ

科目
: 科目名
: 点数

点数の表示

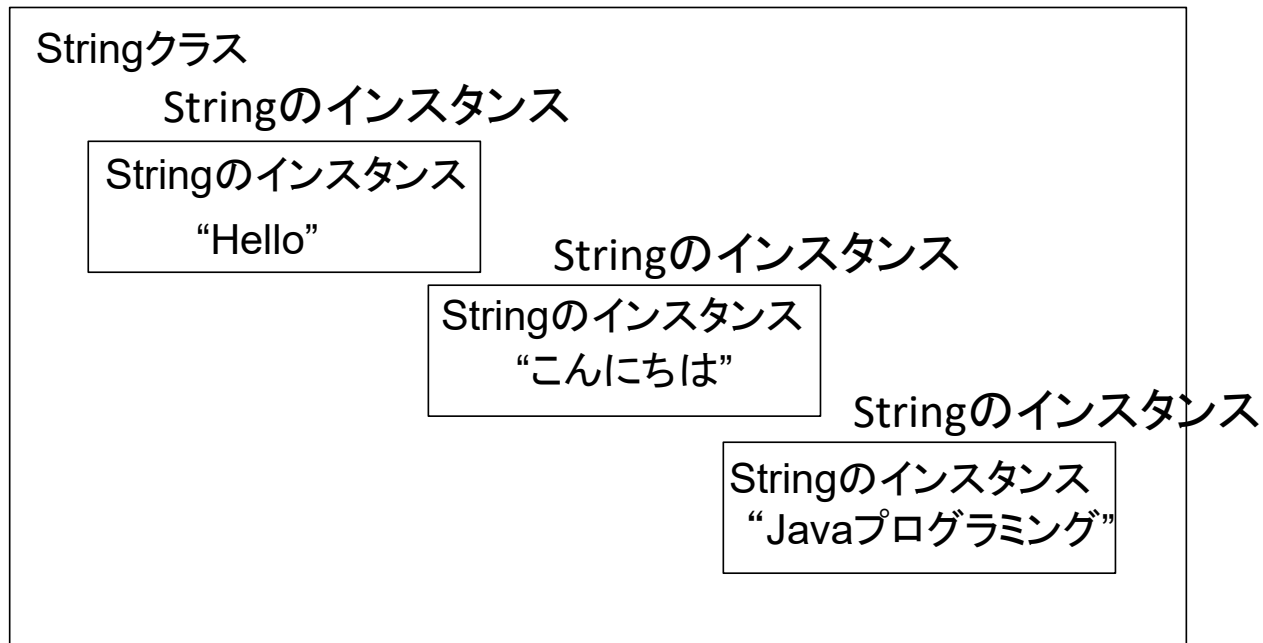
犬クラスの宣言

犬
: 足が4本
: 尻尾がある

ほえる

クラスとインスタンス

- インスタンス: クラスに属する具体的なもの(オブジェクト)
 - クラス: 科目
 - インスタンス: 英語, 数学, 理科
- クラスとは鋳型の仕様書のようなもの. インスタンスは鋳型から作られた具体的なもの
- Stringクラスとそのインスタンス



クラスの宣言と構造

クラスの宣言と構造

```
class クラス名 {  
    フィールドの宣言;  
    コンストラクタの宣言; //省略する場合もある  
    メソッドの宣言;  
}
```

- ・クラス名は大文字で始めること！
- ・フィールドとメソッドがクラスの性質を決める
 - フィールド: クラスの属性情報, データ
 - メソッド: クラスの処理・操作
- ・(例) 科目クラス
 - フィールド: 科目名, 点数
 - メソッド: 点数表示

フィールドの宣言

フィールドの宣言
型名 フィールド名;

(例) 科目名と点数をフィールドとする, **Kamoku**クラスを宣言する.

```
1: class Kamoku {  
2:     string subject;  
3:     int score;  
   ...  
}
```


コンストラクタ

- コンストラクタはインスタンスの初期化の方法を決めるもの。インスタンスを作成する鑄型

コンストラクタの宣言

```
クラス名 (型 変数名1, 型 変数名2, ...) {  
    インスタンス.フィールド1 = 変数名1;  
    インスタンス.フィールド2 = 変数名2;  
}
```

(例) Kamokuオブジェクトを作成するためのコンストラクタ.

- `this`とは現在の「この」インスタンスを指すという意味. 省略可能.
- コンストラクタはメソッドに形が似ているが、**クラス名と名前が同じであること**
、戻り値の記述がないことに注意!

```
1:      Kamoku (String kamoku, int ten) {  
2:          this.subject=kamoku;  
3:          this.score=ten;  
4:      }
```

インスタンスの生成

インスタンスの作り方

引数なしコンストラクタ： クラス名 変数 = new クラス名();

引数付きコンストラクタ： クラス名 変数 = new クラス名(変数1, 変数2, ...);

- ・ 引数なしコンストラクタはフィールドに値を代入しない.

(例) `String st = new String();`

- ・ 引数付きコンストラクタはフィールドの中身, あるいはその一部を指定する方法.

(例) `String st = new String("Hello");`

- ・ 1つのクラスにコンストラクタを複数定義してもよい.
- ・ デフォルトコンストラクタ：コンストラクタが宣言されていない場合は, 引数なしのコンストラクタが自動的に作成される.

(例)英語, 数学, 理科の3教科についての情報をKamokuのインスタンスとして作成する.

```
Kamoku english = new Kamoku("英語", 63);  
Kamoku mathematics = new Kamoku("数学", 92);  
Kamoku science = new Kamoku("理科", 75);
```

メソッドの宣言

- メソッドの宣言
 - クラス宣言内でメソッドも宣言する
 - メソッドの記述方法は、第2回の学習内容を参照のこと.

```
void scorePrint () {  
    System.out.println(this.subject+"の点数は"+this.score+"です");  
}
```

フィールドやメソッドへのアクセス

フィールドの参照
インスタンス.フィールド

(例)

englishオブジェクトの科目名を参照するには

```
String name = english.subject;
```

englishオブジェクトの点数を参照するには,

```
int i = english.score;
```

メソッドの利用
インスタンス.メソッド(引数列)

(例)

englishオブジェクトに点数を表示するメソッドscorePrint()を適用するには

```
english.scorePrint();
```

☆フィールドの参照とメソッドの利用は書き方がとても似ているので注意!

科目クラスの全体像

- (例) 科目名とその点数をプリントするメソッドscorePrintを加えると, Kamokuクラスは以下のようになる.

```
class Kamoku {  
    String subject;  
    int score;  
    Kamoku (String kamoku, int ten) {  
        this.subject=kamoku;  
        this.score=ten;  
    }  
    void scorePrint () {  
        System.out.println(this.subject+"の点数は"+this.score+"です");  
    }  
}
```

フィールド

コンストラクタ

メソッド

科目クラスの使用例

```
public class Heikin2 {  
    public static void main (String[] args) {  
        Kamoku english = new Kamoku("英語", 63);  
        Kamoku mathematics = new Kamoku("数学", 92);  
        Kamoku science = new Kamoku("理科", 75);  
        english.scorePrint();  
        mathematics.scorePrint();  
        science.scorePrint();  
    }  
}
```

Heikin2から
Kamokuクラス
を利用

} ←ここでHeikin2は終了.

```
class Kamoku {  
    String subject;  
    int score;  
    Kamoku (String kamoku, int ten) {  
        this.subject=kamoku;  
        this.score=ten;  
    }  
    public void scorePrint () {  
        System.out.println(this.subject+"の点数は"+this.score+"です");  
    }  
}
```

Heikin2実行結果

英語の点数は63です
数学の点数は92です
理科の点数は75です

四角形クラスの使用例

```
public class RectangleSample {  
    public static void main(String[] args) {  
        MyRectangle mr1 = new MyRectangle();  
        mr1.printMyRectangle();  
        MyRectangle mr2 = new MyRectangle(30, 50);  
        mr2.printMyRectangle();  
        System.out.println("mr1の面積  
="+mr1.getHeight()+"*"+mr1.getWidth()+"="+mr1.getArea());  
    }  
}
```

実行結果

```
横: 10 たて20  
横: 30 たて50  
mr1の面積=20*10=200
```


四角形クラスの例

//クラスの宣言

```
class MyRectangle {
```

```
    //フィールドの宣言
```

```
    private int width;
```

```
    private int height;
```

フィールドを**private**にしたので、ほかのクラスからはフィールドに直接アクセスできなくなる

```
    //引数なしコンストラクタ
```

```
    MyRectangle () {
```

```
        width=10;
```

```
        height=20;
```

```
    }
```

```
    //引数つきコンストラクタ
```

```
    MyRectangle(int w, int h) {
```

```
        width=w;
```

```
        height=h;
```

```
    }
```

アクセスメソッド

フィールドの情報を隠蔽し(クラスの中身はユーザに知らせない), アクセスメソッド経由でフィールド情報の取得・変更を行うほうがよい.

...

//メソッドの宣言

```
int getWidth() {  
    return width;  
}  
int getHeight() {  
    return height;  
}
```

アクセスメソッド

```
int getArea() {  
    return width * height;  
}  
void printMyRectangle () {  
    System.out.println("横: "+this.width+" たて"+this.height);  
}
```

}

Heikin.java再考

配列を使って実装したHeikin.javaをクラスを使って書き直してみる.
学生ごとにデータをまとめるためにStudentクラスを新たに宣言する

```
class Student {  
    private String name;  
    private Kamoku[] scores;  
    private double average;  
  
    Student (String nm, Kamoku[] ten) {  
        this.name=nm;  
        this.scores=ten;  
    }  
  
    Kamoku[] getScores() {  
        return scores;  
    }  
    void setAverage(double av){  
        this.average=av;  
    }  
}
```

つづく

Heikin.java再考(Cont.)

```
double calAverage() {  
    Kamoku[] kojinten=this.scores;  
    int sum=0;  
    for (int i =0; i<kojinten.length;i++) {  
        int ten=kojinten[i].score;  
        sum+=ten;  
    }  
    return sum/(double)kojinten.length;  
}  
void printData() {  
    String message = new String(this.name);  
    message = message+": ";  
    for (int i=0;i<this.scores.length;i++) {  
        Kamoku km = this.scores[i];  
        message=message+km.subject+"="+km.score+" ";  
    }  
    message = message+"平均="+this.average;  
    System.out.println(message);  
}  
}
```

Heikin.java再考(Cont.)

```
public class NewHeikin {  
    public static void main(String[] args) {  
        //名前リストを作成  
        String[] names = {"山田","田中","鈴木","高橋"};  
        //科目のリストを作成  
        String[] subjects = {"数学","理科","英語"};  
        //点数のデータを作成  
        int[][] scores = {  
                                {63, 90, 75},  
                                {85, 100, 95},  
                                {78, 80, 82},  
                                {10, 10, 10}  
        };  
    }  
};
```

ここまでは同じ

Heikin.java再考(Cont.)

//学生のリストを作成

```
Student[] slist= new Student[names.length];
for (int i=0;i<names.length;i++) {
    String nm = names[i]; //名前
    Kamoku[] kmlist= new Kamoku[subjects.length];
    for(int j=0;j<subjects.length;j++){
        Kamoku km = new Kamoku(subjects[j], scores[i][j]);
        kmlist[j]=km;
    }
    Student st=new Student(nm, kmlist);
    slist[i]=st;
}
//学生ごとに3教科の平均値を計算し, プリントアウト
for (int i= 0; i<slist.length;i++) {
    double average = slist[i].calAverage();
    slist[i].setAverage(average);
    slist[i].printData();
}
}
```

標準入力からの入力

```
import java.io.*;
```

```
public class RectangleSample {  
    public static void main(String[] args) {  
        BufferedReader reader = new BufferedReader (new  
            InputStreamReader(System.in));  
        try {  
            System.out.println("横を入力");  
            String yoko = reader.readLine();  
            System.out.println("たてを入力");  
            String tate = reader.readLine();  
            int yokolnt = Integer.parseInt(yoko);  
            int tateInt = Integer.parseInt(tate);  
            MyRectangle mr3 = new MyRectangle(yokolnt, tateInt);  
            System.out.println("面積="+mr3.getArea());  
        } catch (IOException e) {  
            System.out.println(e);  
        }  
    }  
}
```

横を入力
30 ←入力
たてを入力
40 ←入力
面積=1200

標準入力の方法

- reader: BufferedReader型の変数
- BufferedReader: データを行単位で読み込みを行うためのクラス
- InputStreamReader: 入力の読み込みを行うクラス
- System.in: キーボードからの入力
- readLine: 1行分の文字列を取得するメソッド

Javaで円周率を使う

- MathクラスはPIというフィールドを持つ
- したがって、円周率を変数として保存したい場合

```
double pi = Math.PI;
```