

インターネットプログラミング  
2401教室  
第6回  
2014/10/22

岩井将行

# 授業資料

- <http://www.cps.im.dendai.ac.jp>

# 講師紹介

- <http://cps.im.dendai.ac.jp/index.php?Members%2Fiwai>
- 岩井研究室
- <http://cps.im.dendai.ac.jp>
- 岩井研究室の研究分野
- <http://cps.im.dendai.ac.jp/index.php?Research%2FTopics>
- 連絡先 1号館11F 11107b
- iwaiあっと im.dendai

# TA・SA・副手

- 岩井研の精鋭

tjm

- miwa

# 成績

- 毎回取り組み姿勢(出席)
  - 毎回課題(演習のみ)
  - 中間試験(座学)
  - 最終試験(座学)
  - 最終課題(演習のみ)
- 
- ★演習は演習最終発表会を加味

# 講義内容

[第1回](#) Java理解度チェック

[第2回](#) Javaプログラミング基礎2

[第3回](#)

TCP/IPの復習 TCPサーバ

[第4回](#)

TCPクライアント/サーバ通信 チャットプログラム

[第5回](#)

UDP通信

[第6回](#)

中間学力考査（持ち込み不可 紙提出）

[第7回](#)

スレッド基礎 サーバのスレッド化 マルチスレッド

[第8回](#)

デザインパターン

ファクトリメソッド シングルトン

[第9回](#)

ノンブロッキングI/O Javaプログラミング応用

[第10回](#)

マルチスレッド スレッドプール

[第11回](#)

Webクライアント

[第12回](#)

WEBサーバ,プロジェクト設計

[第13回](#)

プロジェクト実装1

[第14回](#)

2014/10/22  
プロジェクト実装2

[第15回](#)

学力考査（持ち込み可 プログラミング提出）

# 授業予定日日程

- [http://www.soe.dendai.ac.jp/kyomu/portal/2014\\_schedule\\_t.pdf](http://www.soe.dendai.ac.jp/kyomu/portal/2014_schedule_t.pdf)
- スケジュール 十
- (1)9/17 第1回 Java理解度チェック
- (2)9/24 第2回 Javaプログラミング基礎1
- (3)10/1 第3回 Javaプログラミング基礎2 TCP/IPの復習  
TCPサーバ
- (4)10/8 第4回 TCPクライアント/サーバ通信 チャットプログラム
- (5)10/15 第5回 UDP通信
- (6)10/22 第6回 中間学力考査（持ち込み不可 紙提出）

# 概要

- クライアント／サーバモデル、TCP/IPネットワークのアプリケーションプログラミングインタフェースの基本および、ネットワークアプリケーションを効率的に動作させるためのマルチスレッドプログラミングを講義する。この基本の後、チャット等の対話型アプリケーション、Twitter4J等のアプリケーション開発の実例を講義する。



# ゴール

- 通信ネットワークを利用したアプリケーションソフトウェアを、TCP/IP を意識したレベルで作成できる力を養成することを目標とする。

# Java プログラムの基本

クラス

```
public class Hello {
```

```
    public static void main(String[] args){
```

```
        // この中に、処理内容を書きます
```

```
    }
```

```
}
```

メソッド

注意1: Hello の部分がプログラム名

注意2: String は、文字列。String[] 文字列の配列

注意3: args は、コマンド引数の配列

args[0], args[1]

# 繰り返し (for文)

```
int i;  
for(i=1; i<=4; i=i+1) {  
    内容  
}
```



```
変数の宣言;  
for(初期化式; 条件式; 増加式) {  
    内容  
}
```

# 繰り返し (for文)

- 変数の宣言
  - 繰り返しの回数をメモしておく変数を用意する
- 初期化式
  - 変数の最初の数字は何か
- 条件式
  - 変数がどうなっている間、続けるか
- 増加式
  - 一回繰り返すごとに、変数をどうするか

```
変数の宣言;  
for(初期化式; 条件式; 増加式) {  
    内容  
}
```

# 繰り返し (for文)

- 変数の宣言
  - 整数型の名前がiという変数を用意
- 初期化式
  - 変数iの最初の数字は1
- 条件式
  - 変数iが4以下の間、続ける
- 増加式
  - 一回繰り返すごとに、変数iに1を足したものを、変数iに入れる

```
int i;  
for(i=1; i<=4; i=i+1) {  
    内容  
}
```

# 繰り返し (while文)

```
int i;  
i=1;  
while(i<=4) {  
    内容  
    i=i+1;  
}
```



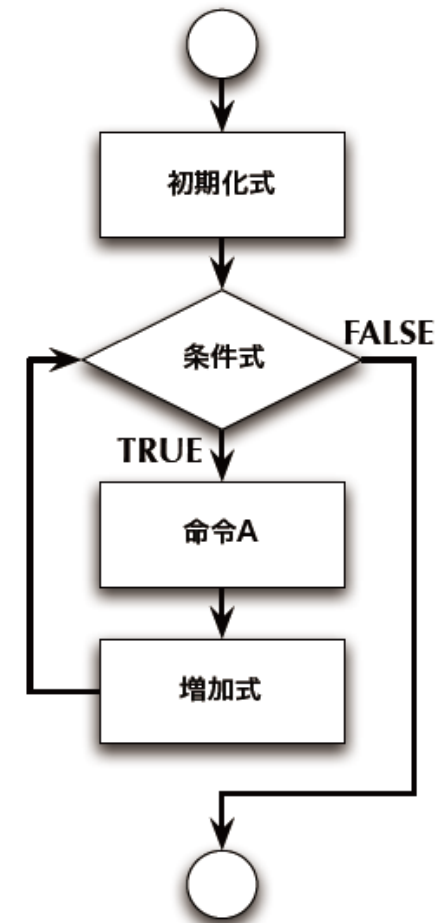
```
変数の宣言;  
初期化式;  
while(条件式) {  
    内容  
    増加式;  
}
```

# フローチャート (繰り返し)

```
int i;  
for(i=1; i<=4; i=i+1) {  
    命令A  
}
```



```
変数の宣言;  
for(初期化式; 条件式; 増加式) {  
    命令A  
}
```



# 省略演算

- 足し算

- $i=i+1;$

- $i+=1;$

- $i++;$

- 引き算

- $i=i-1;$

- $i-=1;$

- $i--;$

- 掛け算

- $i=i*2;$

- $i*=2;$

- 割り算(切捨て)

- $i=i/10;$

- あまりの計算

- $p=i\%2;$

- もし*i*が偶数なら $p==0;$

- 奇数なら $p==1$



# for文

```
for ( 初期化; 条件式; 次の一步 ) {  
    繰り返す処理  
}
```

```
for (int i = 0; i < 3; i++) {  
    System.out.println(i);  
}
```

により、

0

1

2

が表示される。

# 変数の有効範囲(スコープ)

```
for (int i = 0; i < 3; i++) {  
    System.out.println(i);  
}
```

System.out.println("i = " + i): ← iは有効範囲外なので  
コンパイルエラー。

## 解決策

```
int i;  
for (i = 0; i < 3; i++) {  
    System.out.println(i);  
}  
System.out.println("i = " + i):
```

# while 文

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

## null (ナル、ヌル)の意味

```
while (line != null) {
```

null 入力の終わりに達したときの特殊なオブジェクト

# 繰り返し文の練習問題

1 から 100 までの整数を足し合わせる

(1) その1: for文を使う

(2) その2: while文を使う

CountTest.java

WhileTest.java

次週以降

CountTenRunnable.java

CountTesterTweThreads.java

# Class

クラスは、物の設計図。  
中に変数やメソッドが定義される。

オブジェクトは、クラス定義に基づく実際の物。プログラム上は、変数。

例： Automobile というクラスを定義する。  
Automobileクラスで、volkesWagen,  
audi, volvo というオブジェクトを作る。

# method

車というクラスを定義する。メソッドとして、

乗る、止める  
を用意する。

ポルシェというオブジェクトを車という  
クラスで生成すると、

ポルシェ. 乗る、  
ポルシェ. 止める  
というメソッドが使える。

# 定義する場所

## クラス

```
戻り値 メソッド1 {  
  XXXXXXXXXXXX  
}
```

```
戻り値 メソッド2 {  
  XXXXXXXXXXXX  
}
```

```
戻り値 メソッド3 {  
  XXXXXXXXXXXX  
}
```

```
戻り値 メソッド4 {  
  XXXXXXXXXXXX  
}
```

いくつでも  
作ってよい。

# public ?

## クラス

```
public 返回值 メソッド1 {  
    XXXXXXXXXXXX  
}
```

```
public 返回值 メソッド2 {  
    XXXXXXXXXXXX  
}
```

```
返回值 メソッド3 {  
    XXXXXXXXXXXX  
}
```

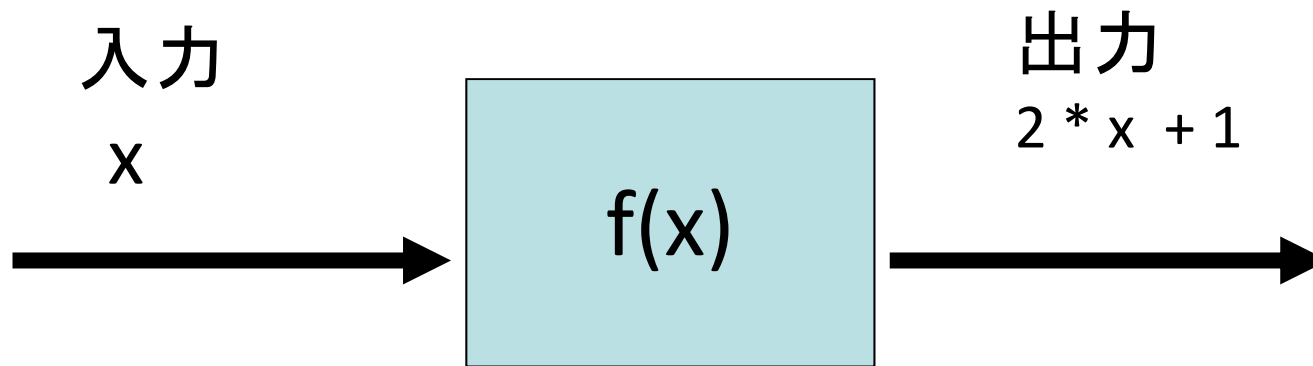
```
返回值 メソッド4 {  
    XXXXXXXXXXXX  
}
```



# 関数

- 関数

$$f(x) = 2 * x + 1$$

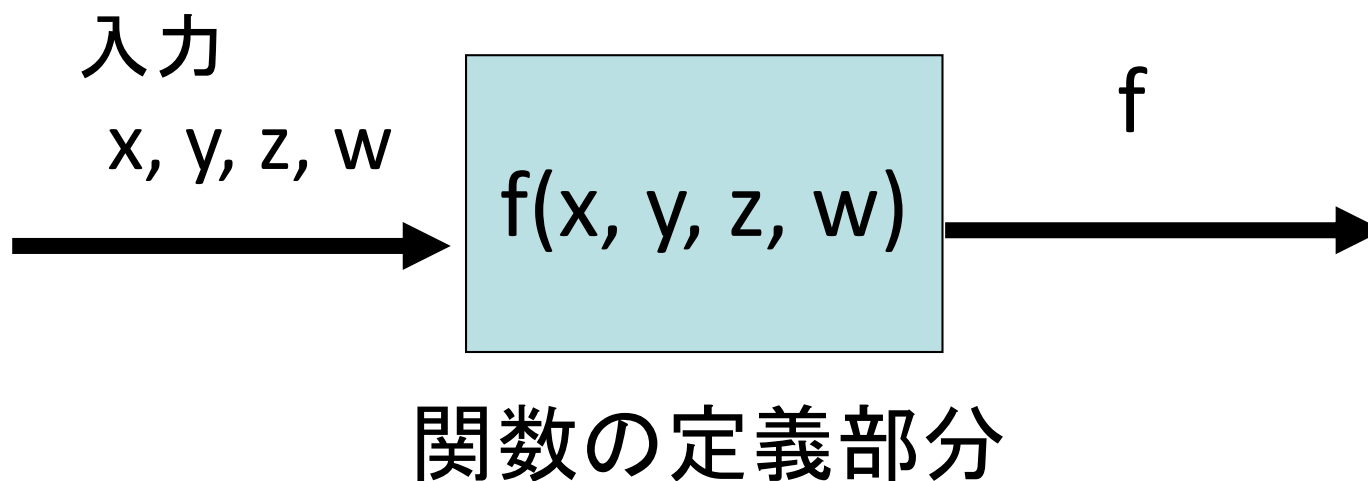


関数の定義部分

しかし、関数の入力はいくらでもあってよい。

- 関数

$$f(x, y, z, w) = 2 * x + y - z + \lfloor w \rfloor$$



# メソッドで引数がたくさんあるとき

```
int  calcComplex(int x, int y, int z,  
                float w) {  
    if ( x > y ) {  
        return z;  
    } else {  
        return (int)w;  
    }  
}
```

# メソッド分け

- 合成関数

$$f(x) = 2 * x + 1$$

$$h(x) = 3 * (2 * x + 1) + 5$$

のとき、 $h(x) = (g \circ f)(x)$

```
int h(int x) {  
    return 3 * (2 * x + 1) + 5;  
}
```



```
int h(int x) {  
    return 3 * g(x) + 5;  
}  
  
int g(int x) {  
    return 2 * x + 1;  
}
```

Javaプログラミングも同じ。メソッドとして独立させた方がよいかどうか、よく考える。

# メソッドの形式

公開するか  
否か

クラス  
メソッドとす  
る

戻り値の型

```
public static int メソッド名(引数宣言) {
```

## メソッドの中身

```
    return (戻り値);  
}
```

# void

関数によっては、返り値がいらないものもある。そのときには、返り値なし (void) を指定する。

前回作成した、drawBar に返り値は必要なかった。

引数がない場合もある。

# 型

int 整数

float 浮動小数点数 (実数)

char 文字型

等

# メソッドの引数

戻り値   メソッド名(型 変数名1,  
                  型 変数名2,  
                  型 変数名3,  
                  型 変数名4  
                  .....) {

メソッドの本体

}



# Javaのメソッドの引数

```
戻り値   メソッド名(型 変数名1,  
            型 変数名2,  
            型 変数名3,  
            型 変数名4  
            .....) {
```

メソッドの本体

クラス(既に定められたものでも、  
自分で定めたものでも)の名前でもよい

```
}
```

# メソッド呼び出し

本来は、

```
g.drawString(XXXXXXXXXXXXXXXXXX);
```

のように、

```
オブジェクト.メソッド名(引数...);
```

と書く。

## メソッド呼び出し(2)

しかし、自分で定義したクラスの中のメソッドを呼び出すときは、  
オブジェクト。

なしに、  
メソッド名(引数...);  
でよい。

例:

```
drawBar(XXXXXXXXXXXXX);
```

# methodとクラス

- Heikin.java と Kamoku.java
- Heikin と Kamoku クラスを作る
  - public class Heikin
  - class Kamoku
- Heikin クラス
  - Kamokuクラスのインスタンスとして、englishとmath を作る
  - english の name に "英語" を設定する
  - english の score に 80 を設定する
  - math も english と同様に (name→数学, score→70)
  - 英語と数学のscoreを読み出して、平均値を表示する
- Kamoku クラス
  - String name
  - setScore というメソッドを定義する。score に値を設定する。
  - getScore というメソッドを定義する。scoreを返す。

# 定数の宣言

C++/C では、#define 文を使用した。

(例)

```
#define WIDTH 80
```

Javaでは、final static で修飾する。

(例)

```
public final static int WIDTH = 80;  
public final static String school = "dendai";
```

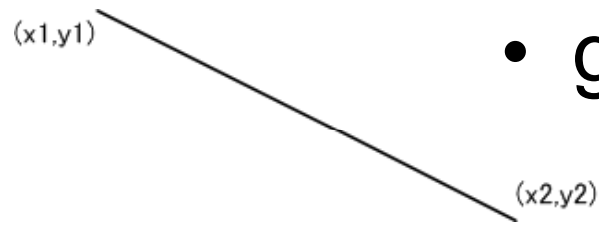
# Graphics

Graphics というクラスには、  
drawString, drawCircle 等の  
メソッドが定義されている。

Graphics クラスである g という  
オブジェクトに対して、  
g.drawString、  
g.drawCircle  
という形でメソッドを呼び出せる。

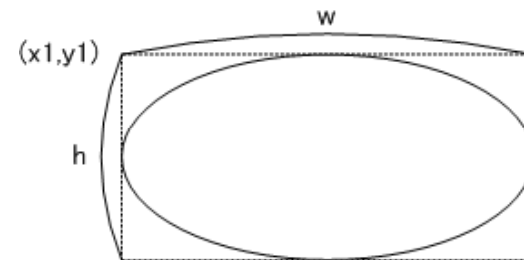
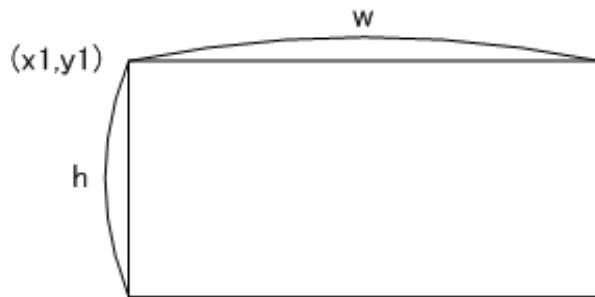
# Graphicsのmethod

- `g.drawLine(x1,y1,x2,y2);`



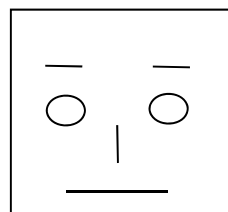
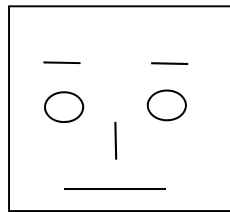
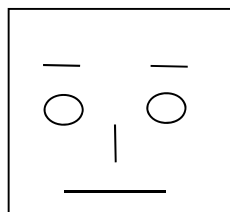
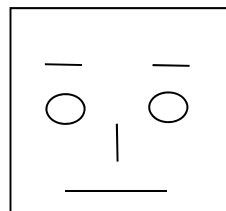
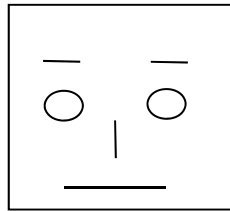
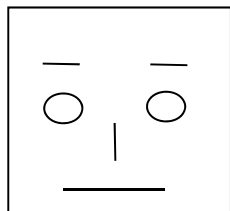
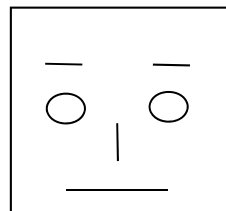
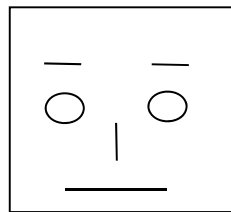
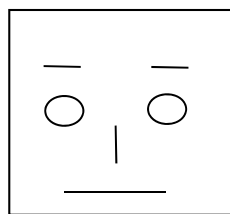
- `g.drawOval(x,y,w,h);`

- `g.drawRect(x,y,w,h);`



課題faceを沢山つくってみよう  
Face.javaを改造してFaces.java  
を提出してください。

課題faceを沢山つくってみようFace.javaを改造して  
Faces.javaを提出してください。二重の繰り返し文を  
用いること。例





# Face ヒント1

```
void drawFace(Graphics g,  
               int xStart,  
               int yStart) {
```

左隅の座標を (xStart, yStart) と  
して、一つの顔を描くメソッドを記述する。

```
}
```

# Faceヒント2

paint メソッドの中には、

```
for(int i = 0; i < 3; i++) {  
    for(int j=0; j < 3; j++) {  
        drawFace(g, 20 + 80 *i,  
                20 + 80 *j);  
    }  
}
```

80 という数字は仮。顔の大きさを考えて計算する。

しかし、数字決め打ちは避けたい

```
for(int i = 0; i < 3; i++) {  
    for(int j=0; j < 3; j++) {  
        drawFace(g, 20 + step *i,  
                20 + step *j);  
    }  
}
```

# 眉毛や鼻の形を自由に変えたい

```
void drawFace(Graphics g, int xStart, int yStart) {  
    drawFrame(g, xStart, yStart);  
    drawEyeBrow(g, xStart, yStart);  
    drawEye(g, xStart, yStart);  
    drawNose(g, xStart, yStart);  
    drawMouth(g, xStart, yStart);  
}
```

さらに内部で  
メソッドに分ける。

```
void drawFrame(Graphics g, int xStart, int yStart) {  
    記述  
}  
void drawEyeBrow(Graphics g, int xStart, int yStart) {  
    記述  
}  
void drawEye(Graphics g, int xStart, int yStart) {  
    記述  
}  
void drawNose(Graphics g, int xStart, int yStart) {  
    記述  
}  
void drawMouth(Graphics g, int xStart, int yStart) {  
    記述  
}
```

# クラスとインスタンス

- クラス



## インスタンス



# Kamoku.java

```
class Kamoku {
    String name;
    int score;

    Kamoku(int s) { // これがコンストラクタ。特殊なメソッド。クラス名と同じ。
        score = s;
    }

    // setScore というメソッドを定義する。
    // score に値を設定する。
    public void setScore(int num){
        score = num;
    }

    // getScore というメソッドを定義する。
    // scoreを返す。
    public int getScore() {
        return score;
    }
}

// メソッド 関数のこと
// public 戻り値(戻り値return value) 関数名() {
//     中に具体的な処理を書く
// }
```

```
public class HeikinA {  
    public static void main(String[] args){  
  
    // Kamokuクラスのインスタンスとして、englishを作る  
        Kamoku english = new Kamoku(80);  
    // 同様に、mathインスタンスをつくる。  
        Kamoku math = new Kamoku(70);  
  
    // english の name に "英語" を設定する  
        english.name = "英語";  
        int a = english.getScore();  
        System.out.println("英語の点は" + a + "ですね");  
        a = math.getScore();  
        System.out.println("数学の点は" + a + "ですね");  
    }  
}
```

# 【難】HeikinB objectの配列

```
public class HeikinB {
    public static final int N=100;
    Kamoku[] english = new Kamoku[N];
    String kamokuname="不明";

    HeikinB(String s){
        this.kamokuname=s;
    }
    void initializeScores(){
        for (int i = 0; i < N; i++) {
            int score = 50+(int)
(Math.random() * 50);
            english[i] = new Kamoku(score);
            english[i].name = kamokuname +
i;
        }
    }
}
```

```
void printAverage(){
    double sum=0;
    for (int i = 0; i < N; i++) {
        int score =
english[i].getScore();

        System.out.println(english[i].name + "の点
は" + score+"点");
        sum+=score;
    }

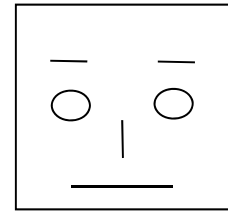
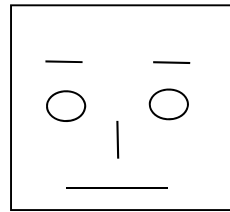
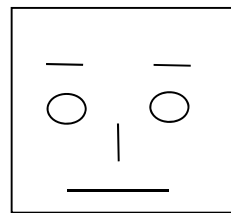
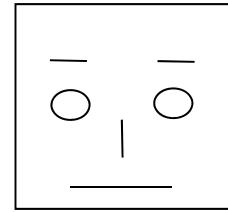
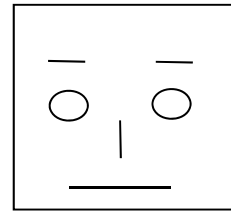
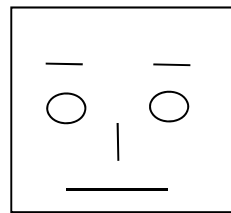
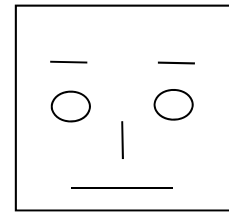
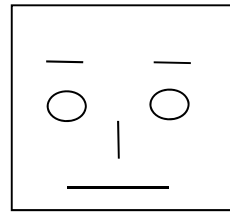
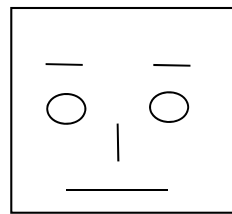
    double average=sum/N;
    System.out.println(this.kamokuname+"の"
+ "平均の点は"
+average+"点");
}

public static void main(String[] args) {

    HeikinB heikinB=new HeikinB("英語");
    heikinB.initializeScores();
    heikinB.printAverage();
}
}
```



課題faceを沢山つくってみよう  
FacesMain.javaを改造してobject指向で  
FacesMain.javaを提出してください。  
なるべくなら二重の繰り返し文を用いること。例



# Face ヒント1

```
void drawFace(Graphics g,  
               int xStart,  
               int yStart) {
```

左隅の座標を (xStart, yStart) と  
して、一つの顔を描くメソッドを記述する。

```
}
```