インターネットプログラミング
2403教室
第7回
2015/11/11

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

近藤君

- 他岩井研の精鋭

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
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/2015_schedule_t.pdf

- スケジュール †
- (1)第１回　Java理解度チェック
- (2) 第２回　Javaプログラミング基礎1
- (3) 第３回　Javaプログラミング基礎2　TCP/IPの復習　TCPサーバ
- (4) 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) 第５回　UDP通信
- (6) 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。

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)

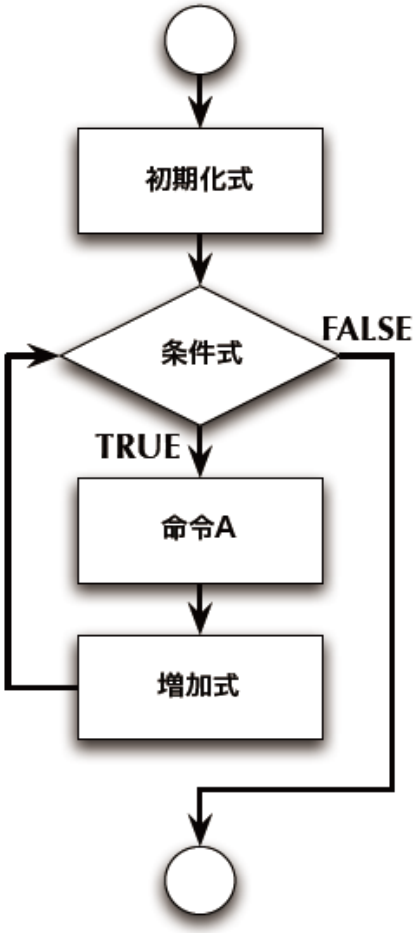
```
int i;
i=1;
while(i<=4) {
   内容
   i=i+1;
}
```

```
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; 次の一歩 ) {
        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):
```

拡張**for文**EnhancedForStatement

- for (データ型 変数名: コレクション)
  { 実行する文1;
   実行する文2;
  ...
   }

- int data[] = {85, 72, 89};
  for (int seiseki: data)
  {
      System.out.println(seiseki);
  }

拡張for文(Enhanced)の流れ

- 1)配列dataを宣言し初期化
  2)配列から要素に含まれる値を1つ取り出し変数seisekiに代入
  3)変数seisekiを出力
  4)配列から要素に含まれる値を1つ取り出し変数seisekiに代入
  5)変数seisekiを出力
  6)配列から要素に含まれる値を1つ取り出し変数seisekiに代入
  7)変数seisekiを出力
  8)配列から全ての値を取り出したので繰り返しを終了

IteratorSample

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
public class IteratorSample {
public static void main(String[] args) {
List list = new ArrayList();
 for (int i = 0; i < 10; i++) {
  list.add(new Integer(i));
  }
  Iterator it = list.iterator();
  while (it.hasNext()) {
    Integer tmp = (Integer)it.next(); System.out.println(tmp);
  }
  }
}
```

EnhancedForSample

```
import java.util.ArrayList;
import java.util.List;
    public class EnhancedForSample {

    public static void main(String[] args) {
    List<Integer> list = new ArrayList<Integer>();
    for (int i = 0; i < 10; i++) {
      list.add(new Integer(i));
     }
    for (Integer i : list) {
       System.out.println(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

WhileTest

```
public class  WhileTest {

  public static void main(String[] args) {
    int i = 0;
    int j = 0;

    while(j < 101) {
      i += j;
      System.out.println("i = " + i );
      j++;
    }
    System.out.println("while: "+ i);
  }
}
```

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

```
返り値  メソッド1 {
  XXXXXXXXXX
}
```

```
返り値  メソッド2 {
  XXXXXXXXXX
}
```

```
返り値  メソッド3 {
  XXXXXXXXXX
}
```

```
返り値  メソッド4 {
  XXXXXXXXXX
}
```

いくつでも
作ってよい。

public ?

クラス

```
public 返り値  メソッド1 {
  XXXXXXXXXX
}
```

```
public  返り値  メソッド2 {
  XXXXXXXXXX
}
```

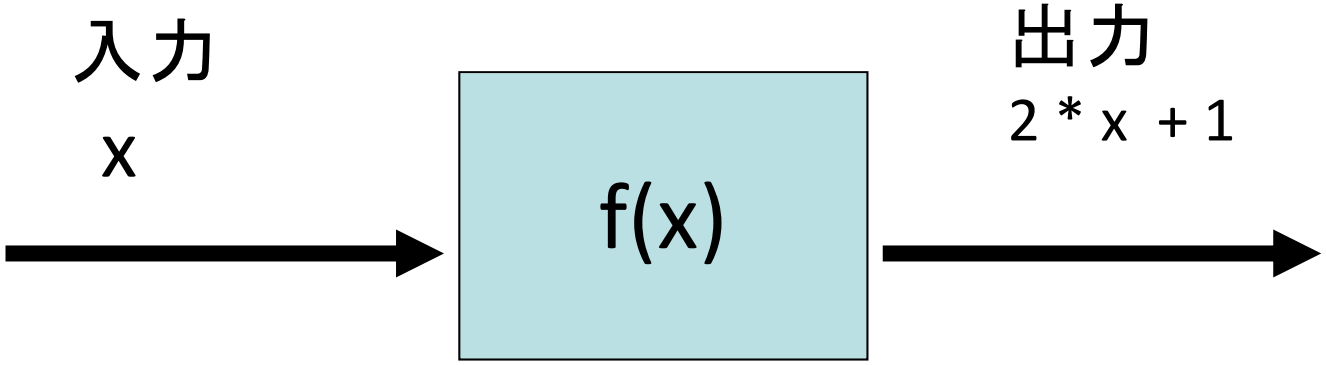
```
返り値  メソッド3 {
  XXXXXXXXXX
}
```

```
返り値  メソッド4 {
  XXXXXXXXXX
}
```

関数

- 関数

f(x) = 2 * x + 1

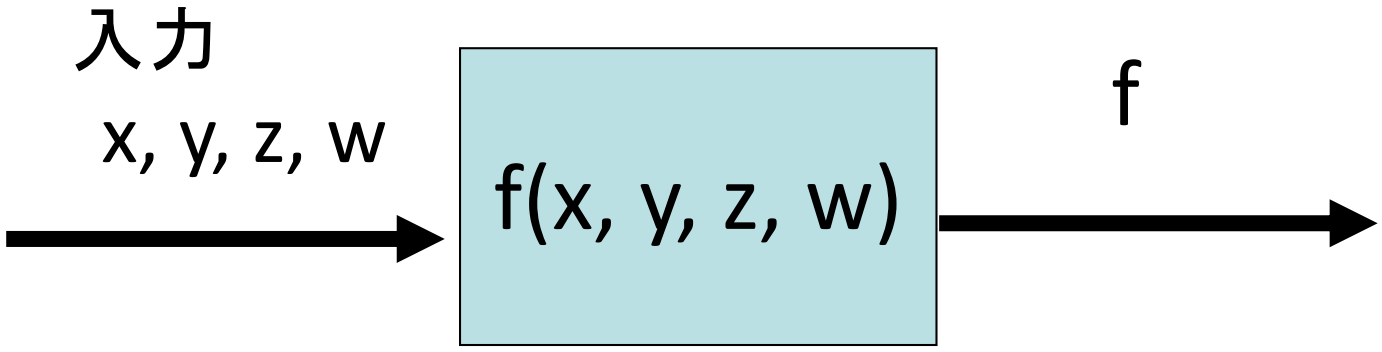


関数の定義部分

しかし、関数の入力はたくさん あってよい。

- 関数

f(x, y, z, w) = 2 * x + y - z
+ ∟w」



関数の定義部分

Calc3MethodTest.java

```
public class Calc3MethodTest {

	public static void main(String[] args){
		int x = Integer.parseInt(args[0]);
		int y = Integer.parseInt(args[1]);
		printResult(func(x,y));
		// 上の1行は、int z = func(x,y);
		//		printResult(z);
		//  の2行を縮めたもの
	}

	//返り値は整数型
	public static int func(int a, int b) {
		return 2 * a + b;
	}

	//返り値は無い
	public static void printResult(int z){
		System.out.println("計算結果は" + z + "です。");
	}

}
```

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 ○ 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(XXXXXXXXXXXXX);

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(XXXXXXXXXXX);

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";
```

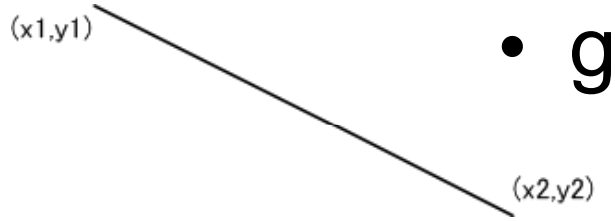
Graphics

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。

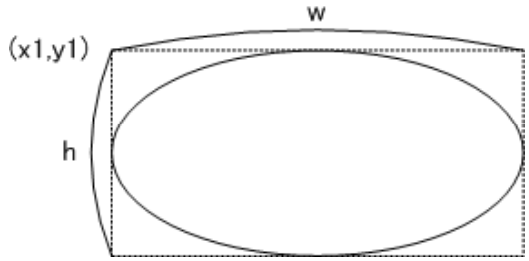
Graphics クラスである g という
オブジェクトに対して、
　g.drawString、
　g.drawCircle
という形でメソッドを呼び出せる。

Graphicsのmethod

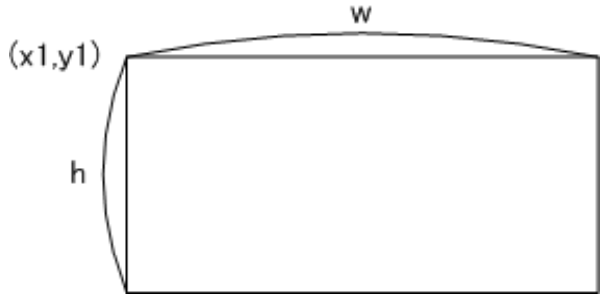
- g.drawLine(x1,y1,x2,y2);



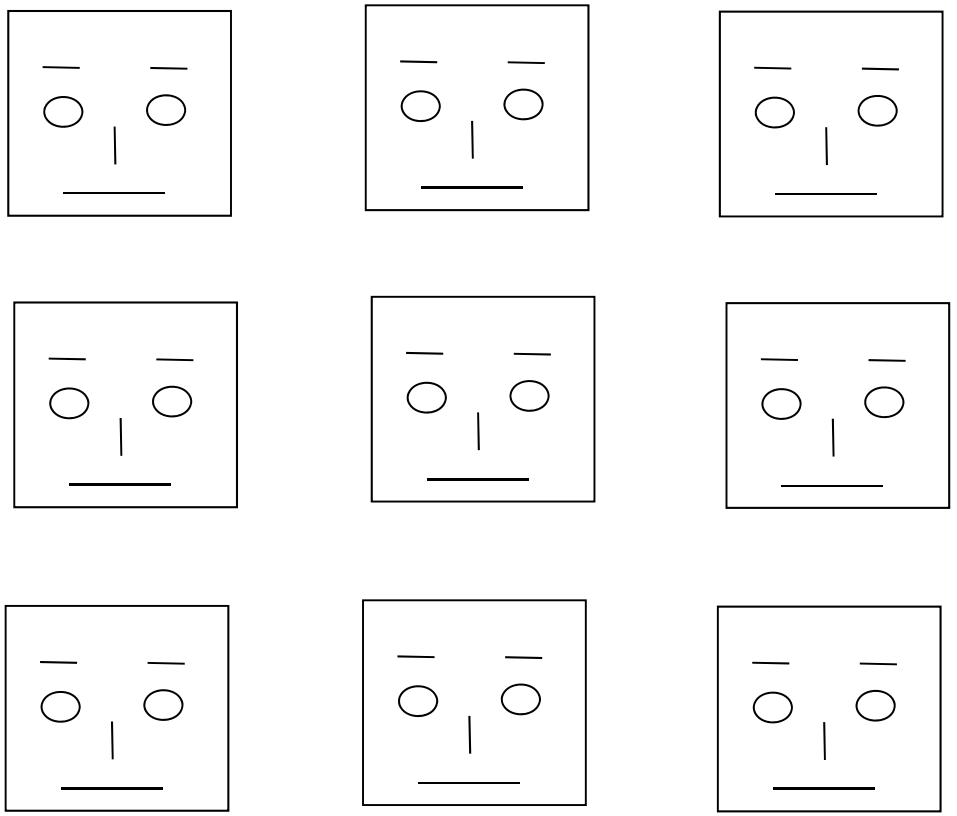
- g.drawOval(x,y,w,h);



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



課題3:faceを沢山つくってみようFaceMain.javaを改造してFacesMain.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);
}

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
}
```

さらに内部で
メソッドに分ける。

オブジェクト指向

クラスとインスタンス

- クラス

インスタンス

ClassA

- ClassA
- InnerClassB
- ClassC

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 返り値(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

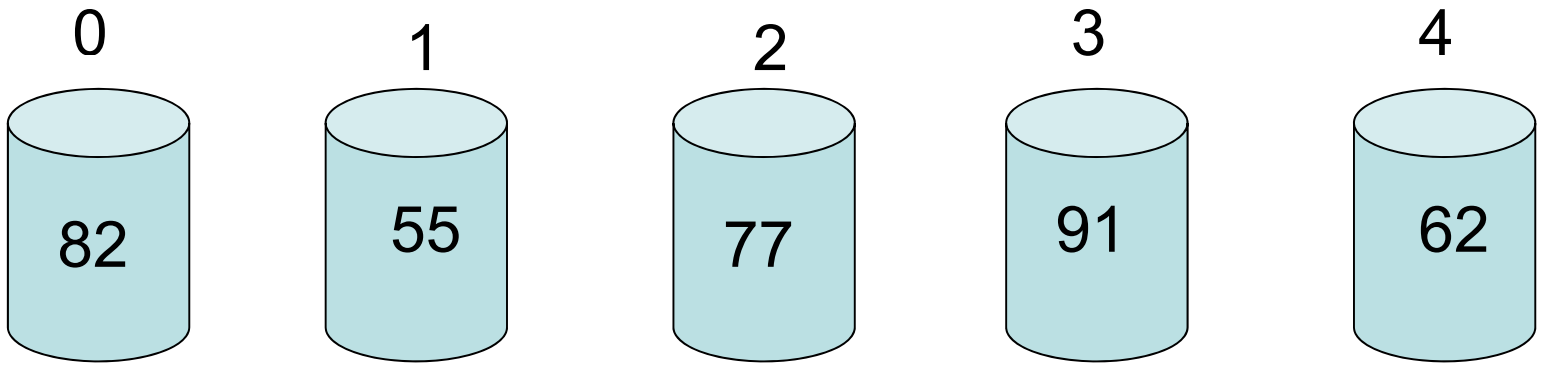
```
public class HeikinB {
	public static final int N=100;
	Kamoku[] english = new Kamoku[N];
	String kamokuname="不明";

	HeikinB(String s){
		this.kamokuname=s;

	}
	void initalizeScores(){

		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.initalizeScores();
		heikinB.printAverage();
	}
}
```

MovingBall
Thread,Runnable
RandSwitch
配列

配列

int xData;
という宣言をすると、xDataという名前の整数型の変数が使える。
５人の学生がいて、各学生のコンピュータ基礎の点数を保存したいときにはどうしたらよいか？
int mathScore;
では 1つの変数。→ 5人分.



配列の宣言

- 型　配列名[] = new 型[n];
  int tensu[] = new int[100];
  型[]　配列名 = new 型[n];
  int[] tensu = new int[100];

0 ～ n-1 の n個の配列ができる。

配列の添字は0から始まる
配列 xData の大きさが3のとき、使えるのは、xData[0]、xData[1] 、xData[2]。

配列の宣言

・int mathScore[] = new int[5];
と宣言すると、

```
  mathScore[3] =  82;
for(int i = 0; i < 5; i++) {
    mathScore[i] = 10;
}
```

のような代入等が可能となる。

配列の長さ

- 配列名.length
- 例えば、xData の長さは、xData.length で求められる。

配列の初期化

配列の型[]　配列　= {要素, 要素, 要素};

例

int[]　xData = {90, 85, 65};

問題

- mathScore という大きさ5のintの配列をつくり、82, 55, 77, 91, 62 の初期値を入れる
- 全てを　＋５する
- 最高点と平均点を表示する
  - For文を使う

Objectの配列

Objectの配列：FaceObjKadaiAns.java

- FaceObjAns**[]** fobjs = **new FaceObjAns[9];**

```
for (int j = 0; j < 3; j++) {//行
    yStart = j * 220 + 50;
    for (int i = 0; i < 3; i++) {//列
        xStart = i * 220 + 40;
        fobjs[i + 3 * j] = new FaceObjAns(xStart, yStart);
    }
}
```

描画

```
// 9個数の顔を書く
for (int i = 0; i < fobjs.length; i++) {
   fobjs[i].makeFace(g);
}
```

FaceObjectのコンストラクタ

```
class FaceObjAns {
// コンストラクタ
int h;
int w;
int xStart = 0;
int yStart = 0;
public FaceObjAns(int x, int y) {
        h = 200;
        w = 200;
        this.xStart = x;
        this.yStart = y;
}
// 個々にメソッドを追加
public void makeFace(Graphics g) {
        makeRim(g);
        makeEyes(g, 20);
        makeNose(g, 40);
        makeMouth(g, 80);
}
```

Interface

- 内容に抽象メソッドしか持たないクラスのようなもの(バールのようなもの)をインタフェースと呼びます。
- クラスと並んで、パッケージのメンバーとして存在します。
- インタフェースはクラスによって実装(implements) され、
- 実装クラスはインタフェースで宣言されてい抽象メソッドを実装します。

インタフェースの複数実装

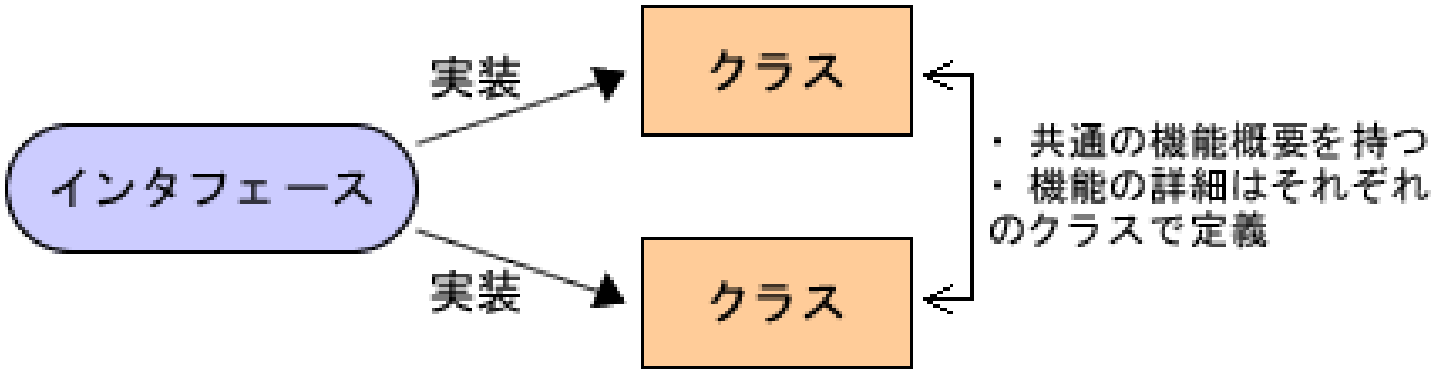
- クラスの場合は、単一のクラスしか継承 (extends) できせんが、インタフェースの場合は、複数のインタフェースを実装 (implements) することができます。

class InterfaceImpl ***implements*** Interface1, interface2, interface

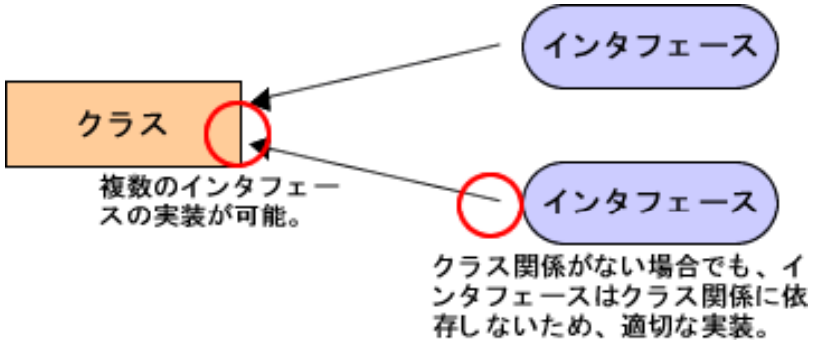
... }

interface の修飾子は public のみ。

インタフェースのメリット



- Javaは複数のスーパークラスを継承することはできません。Javaでは複数のスーパークラスの継承（多重継承）が認められていないため。



Plugin hybrid車は災害時にバッテリとして利用可能

ICar.javaインタフェース

- 車輪in getNumOfTiers()がある。
- スピートを設定する
  - void setSpeed (int sp)
  - int getSpeed()
  - void printCarName()

IElectricCharge.javaインタフェース

- void chargeBattery(int b)
- int getAllBattery()
- int consumeBattery(int b)

HybridCarImpl.java

- を実装してください。
- Yourには自分の名前を入れてください。
- 例　MasaHybridCarImpl.java

呼び出しのMainCall.javaを実装しよう。

- Hint
- MasaHybridCarImpl masaCar= new MasaHybridCarImpl();

- ICar car=(ICar) masaCar;
- car.setSpeed(); car.printCarName();
- IElectricCharge charger =(IElectricCharge) masaCar
- carger.chargeBattery(100);

Thread

Thread,Runnable

MovingBall

ThreadSleep　停止

- 300ミリ秒処理を停止する。

```
try{
   Thread.sleep(3000);
   //3000ミリ秒Sleepする
}catch(InterruptedException e){}
```

Threadの2種類の作りかた

- 1)**implements Runnable**

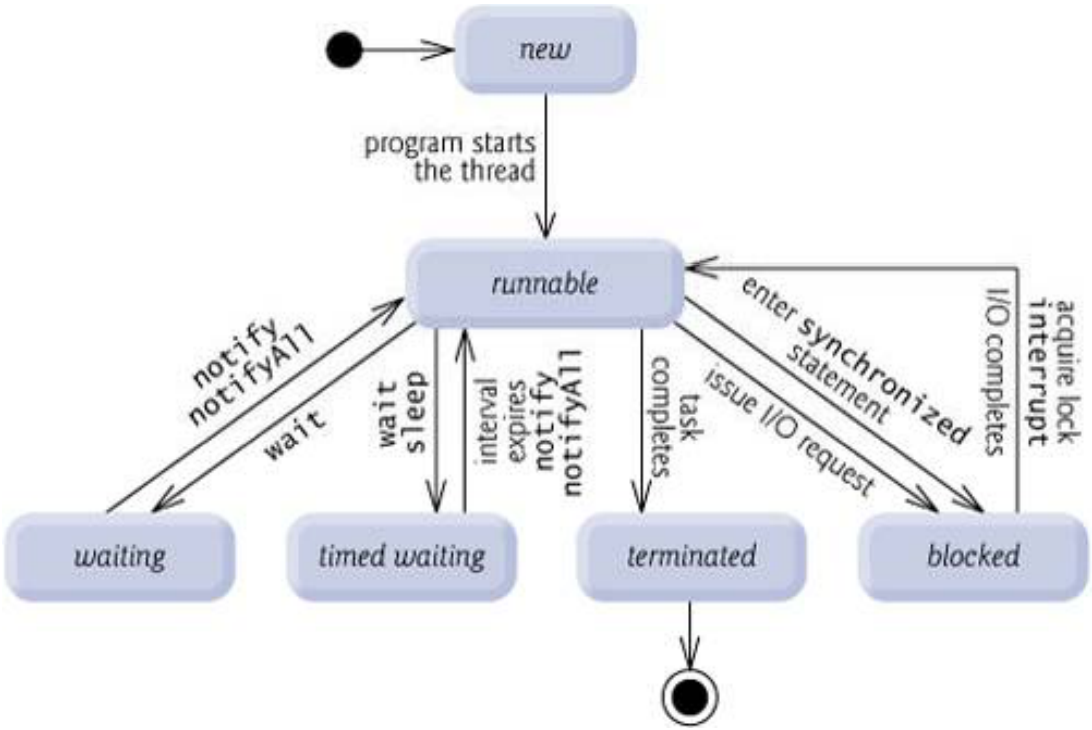
Runnableを実装したクラスをThreadクラスのコンストラクタとして渡す。start()で開始。

```
CountTenRunnable ct = new CountTenRunnable();
Thread th = new Thread(ct);
th.start();
```

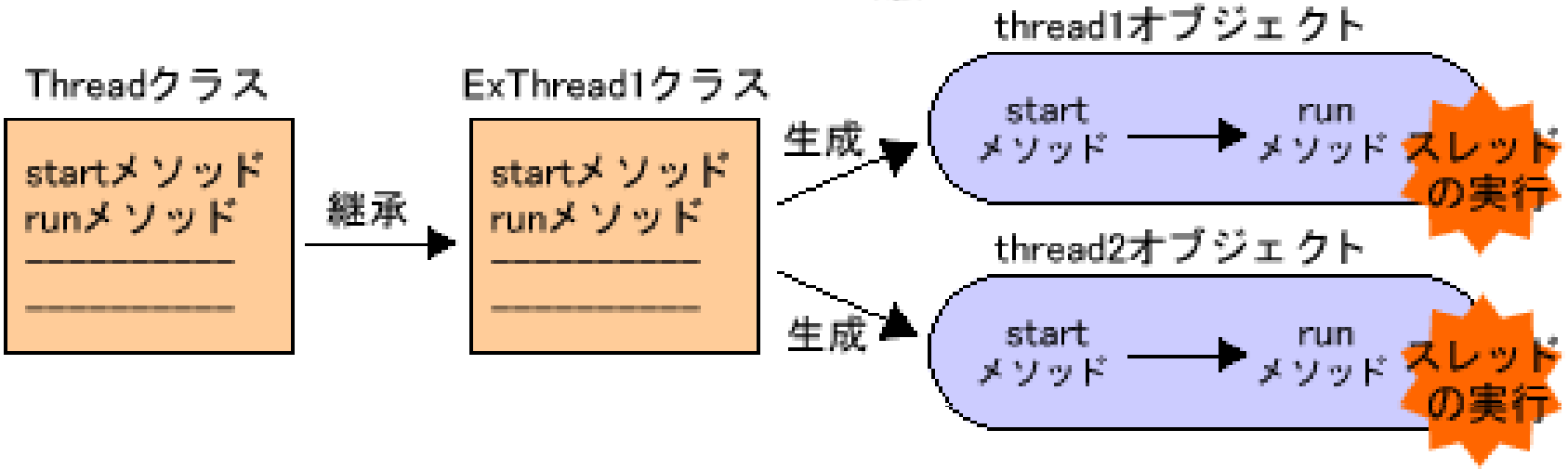
- 2)**extends Thread**

Threadを拡張したクラスをnew してstart()メソッドを呼び出す。

- (1)


new
program starts the thread
runnable
notify
notifyAll
wait
wait
sleep
interval expires
notify
notifyAll
task completes
issue I/O request
enter synchronized statement
I/O completes
acquire lock
interrupt
waiting
timed waiting
terminated
blocked

- (2)


Threadクラス
startメソッド
runメソッド
継承
ExThread1クラス
startメソッド
runメソッド
生成
生成
thread1オブジェクト
start メソッド
run メソッド
スレッドの実行
thread2オブジェクト
start メソッド
run メソッド
スレッドの実行

```
class MainThread{
    public static void main(String args[]){
     /* 別スレッドとして動作させるオブジェクトを作成 */
     SubThread sub = new SubThread(); /* 別のスレッドを作
    成し、スレッドを開始する */
     Thread thread = new Thread(sub) thread.start();
     }
    }
    class SubThread implements Runnable{
          public void run(){ }
    }
```

動かしてみよう

- CountTest.java
- CountTenRunnable.java
- CountTesterTweThreads.java

MovingBall

```
MovingInnerFFrame f = new MovingInnerFFrame();
Thread th = new Thread(f);
th.start();


class MovingInnerFFrame extends Frame
   implements Runnable {

   public void run() {}
}
```

第5回演習課題

- MovingBallを改造し3色以上の複数のボールを表示せよ。
- 壁の跳ね返りを画面のサイズに修正せよ。

- コンソールに残り秒数カウントダウンしてゲームが終了するようにThread.sleep()メソッドを用いよ。
- なお終了時間は20秒とせよ。
- ヒント
  - Thread.sleep(1*1000);

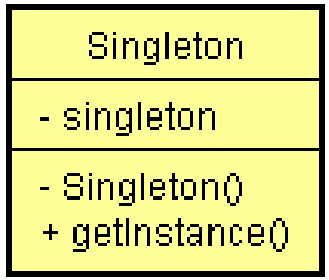
MovingBallを配列にしよう。

Singleton

Singleton

- たった一つのインスタンスしか作らせないようにするパターン。
  普通はインスタンスを沢山作る
- 場合によってはインスタンスを一つしか作らせたくない
- プログラマ任せにすると、間違ってnewを複数回呼び出してしまう。
  SingItonパターンを適用すると、指定したクラスインスタンスが1つしか存在しないことを保証する。

Singleton




```
public class MyFirstSingleton {
/* 唯一のインスタンス。*/
private static final MyFirstSingleton instance = new MyFirstSingleton();
/** * コンストラクタ。*/
private MyFirstSingleton() { }
    /** * このクラスの唯一のインスタンスを返す。*/
    public static MyFirstSingleton getInstance() {
    return instance;
    }
}
```

Singleton

- private な static 変数を定義して初期化
  インスタンスは、このクラスのロード時に一度だけ生成処理。
  コンストラクタはpublicでなく外部に公開せず
  private。外部からインスタンス生成されることを防ぐ。privateにしないと外部から new とされてしまいインスタンスが自由に作れてしまう
  getInstance() を作成し外部に公開します。
  作られた唯一のインスタンスを返却することがこのメソッドの役目。public メソッドにしてどこからでも呼び出せるように。
  生成のタイミングは、初めて getInstance() をが呼ばれた時です。

MyFirstSingletonCall

```
void Test() {
 /* new できません。コンパイルエラーとなります。 */
// MyFirstSingleton mys = new MyFirstSingleton ();
// この時点で インスタンス生成＆返却
MyFirstSingleton mys2 = MyFirstSingleton.getInstance();
// 同じインスタンス返却
MyFirstSingleton mys2 = MyFirstSingleton.getInstance();
if (mys1 == mys2) {
        System.out.println("同じインスタンスです。");
    }
}
```

動かしてみよう。

- SwingAnimationBasic
- SwingAnimationFaceObj

- Swing　より美しいグラフィックスを提供 java2D
- ちらつき防止
- http://www.java2s.com/Tutorial/Java/0240__Swing/Catalog0240__Swing.htm

Socket interface