

インターネットプログラミング
2403教室
第8回
2015/11/17

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](#)

2015/11/17
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 十
- (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。

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。

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

}

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(XXXXXXXXXXXX);
```

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";
```

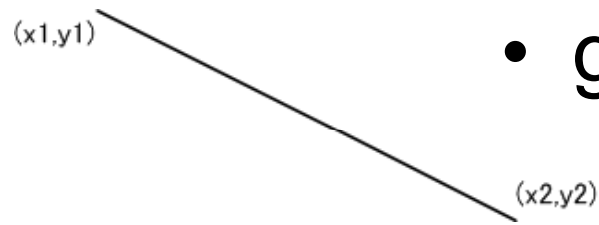
Graphics

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。

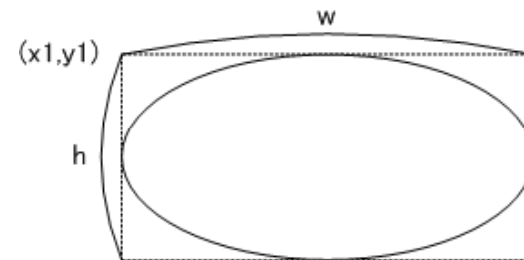
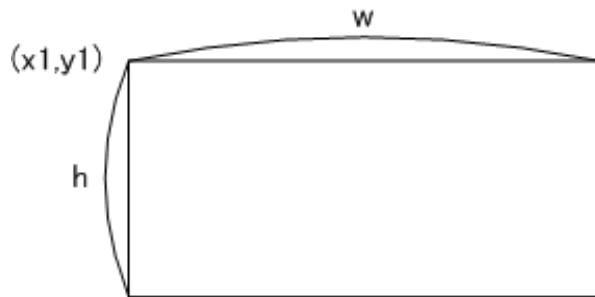
Graphicsのmethod

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

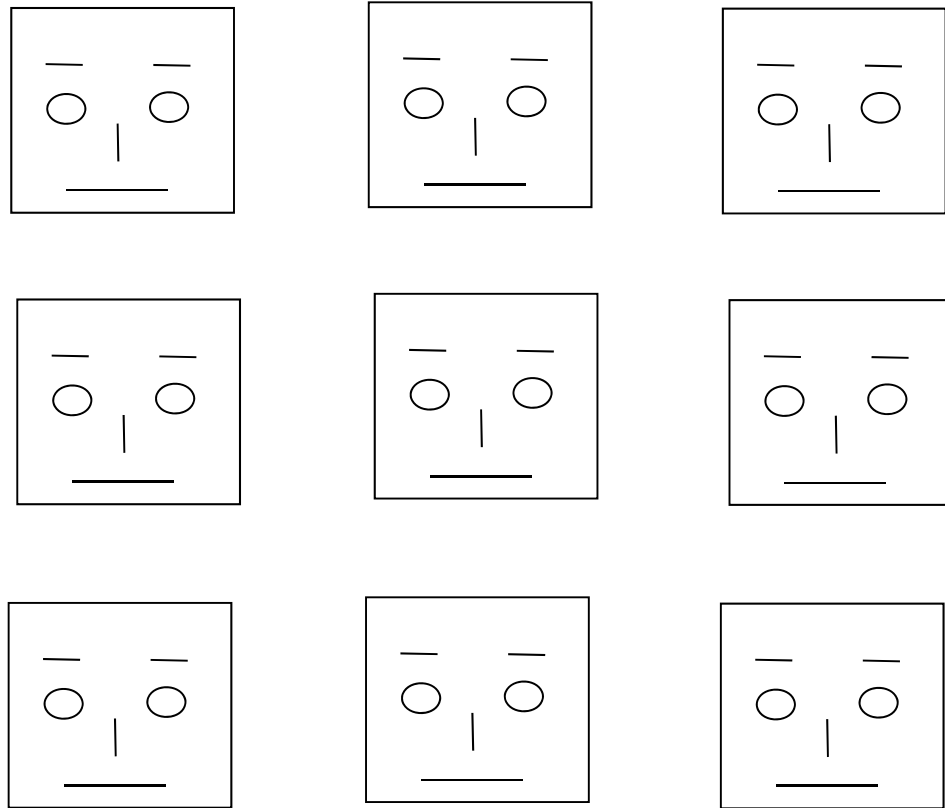


- `g.drawOval(x,y,w,h);`

- `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);  
}
```

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  
}
```

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 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");  
    }  
}
```

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

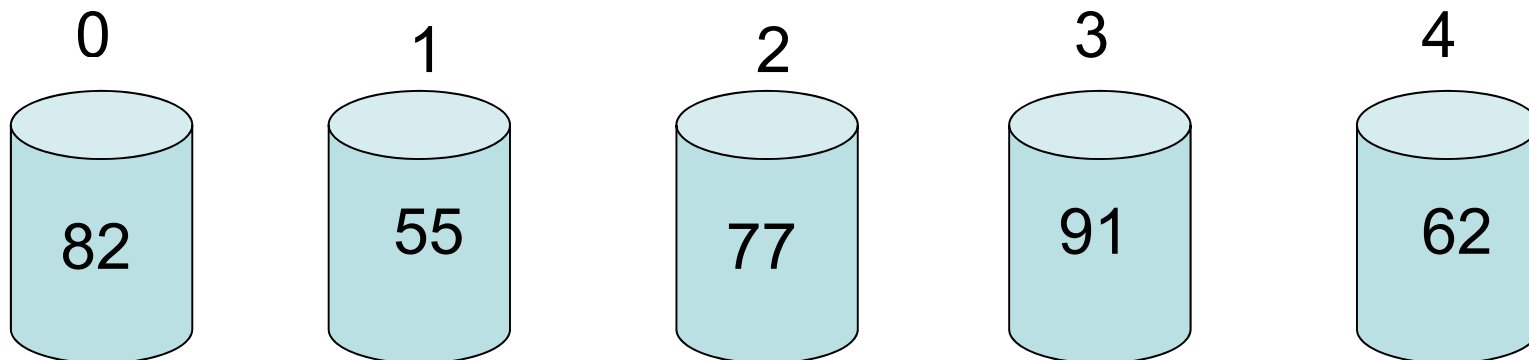
```
int xData;
```

という宣言をすると、xDataという名前の整数型の変数を使える。

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[] fobjns = 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;
        fobjns[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て**抽象メソッドを実装**します。



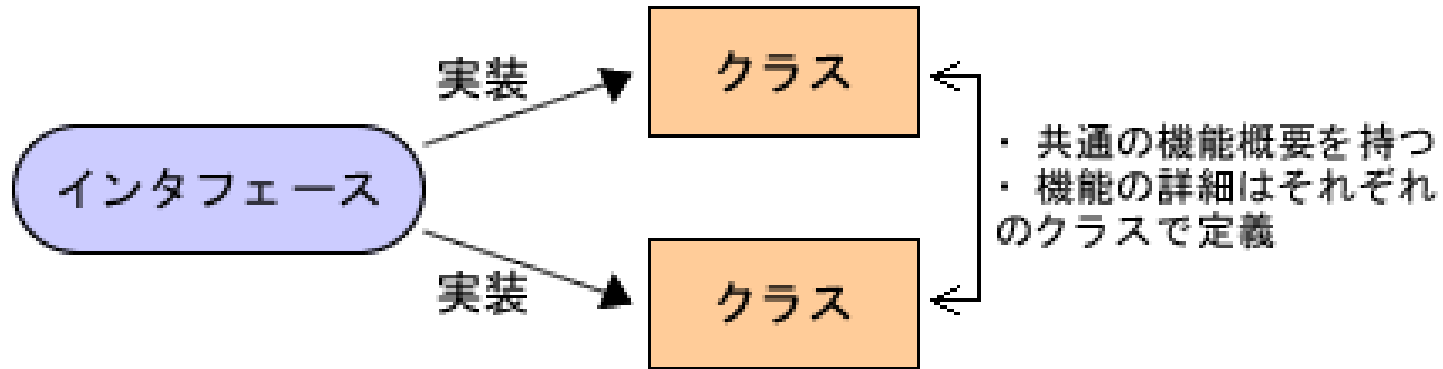
インタフェースの複数実装

- クラスの場合は、単一のクラスしか継承 (extends) できませんが、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。

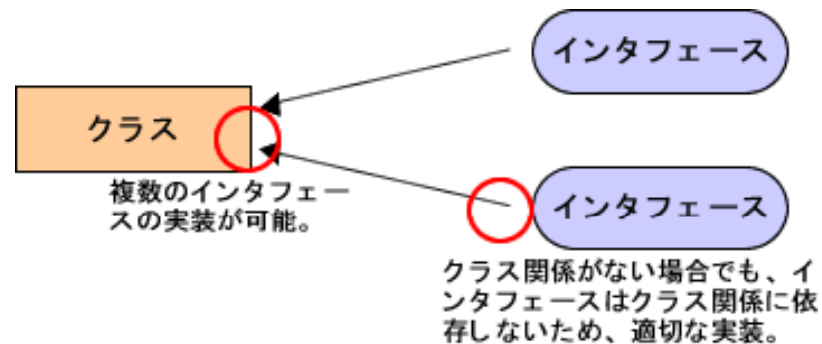
```
class InterfaceImpl implements Interface1, interface2, interface  
... }
```

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ーとして利用可能



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`
- `charger.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**

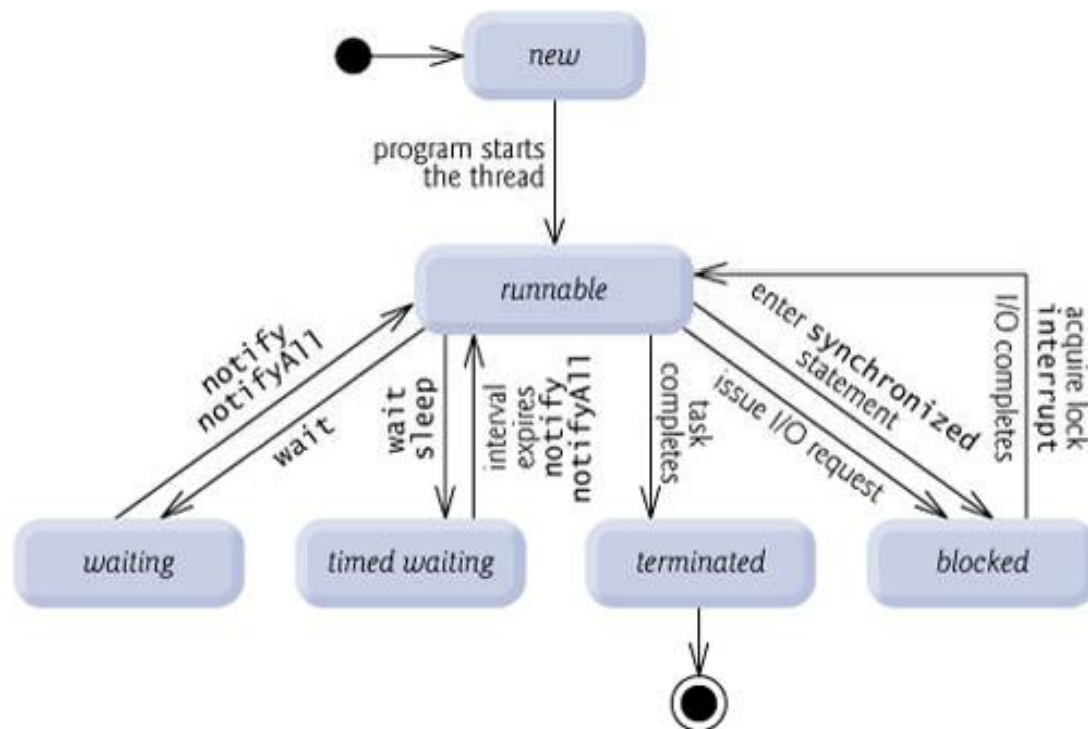
Runnableを実装したクラスをThreadクラスのコンストラクタとして渡す。start()で開始。

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

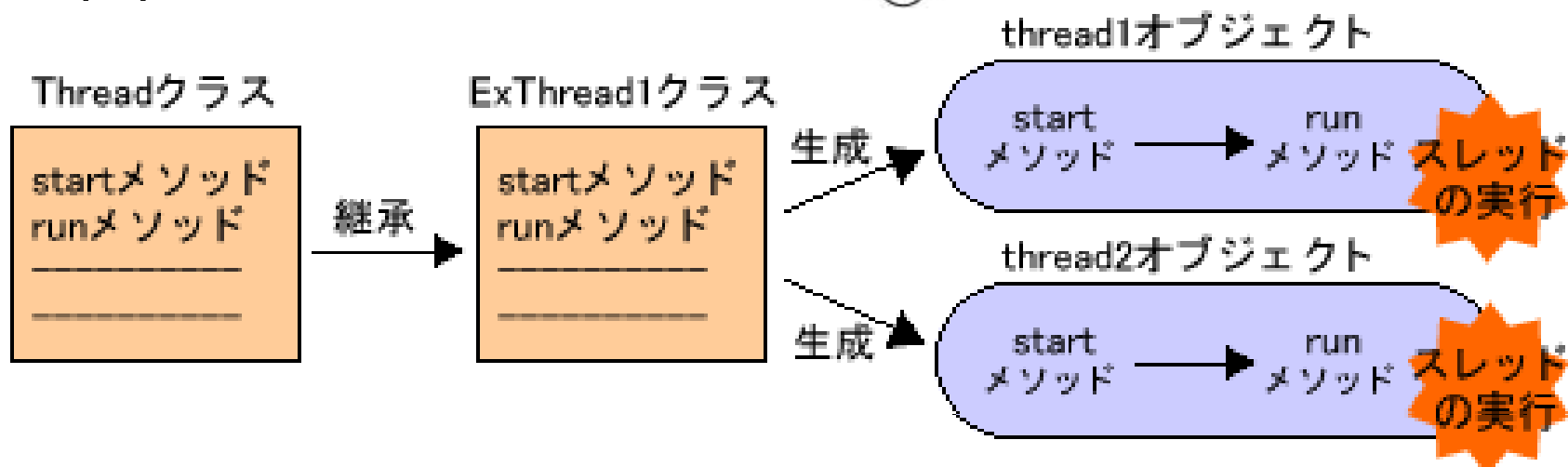
- **2) extends Thread**

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

• (1)



• (2)



```
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`
- `CountTesterTwoThreads.java`

MovingBall

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

```
class MovingInnerFFrame extends Frame  
    implements Runnable {  
  
    public void run() {}  
}
```

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。

文字列型(String)からの変換

- 数値型への変換
- int型 int num =
Integer.valueOf(str).intValue();
– int num = Integer.parseInt(str);
- short型 short num =
Short.valueOf(str).shortValue();
– short num = Short.parseShort(str);
- byte型 byte num =
Byte.valueOf(str).byteValue();
– byte num = Byte.parseByte(str);

文字列からの変換

- long型への変換
- long num = Long.valueOf(str).longValue();
 - long num = Long.parseLong(str);
- float型への変換
- float num = Float.valueOf(str).floatValue();
 - float num = Float.parseFloat(str);
- double型への変換
- double num =
Double.valueOf(str).doubleValue();
 - double num = Double.parseDouble(str);

数値型からの変換

- 文字列型(String)への変換

int型	<code>String str = String.valueOf(num);</code>
short型	<code>String str = String.valueOf(num);</code>
byte型	<code>String str = String.valueOf(num);</code>
long型	<code>String str = String.valueOf(num);</code>
float型	<code>String str = String.valueOf(num);</code>
double型	<code>String str = String.valueOf(num);</code>

バイト配列型からの変換

- 文字列型(String)への変換
- String型
- `String str = new String(buf);`
- `String str = new String(buf, "Shift_JIS");`
- 数値型への変換
- `int型 int num = buf & 0xff;`
-

char型からの変換

- String型への変換
- `char c[] = {(char)(0x41)};`
- `String str = new String(c);`

文字列の変換

- バイト配列型への変換
- byte配列型 `byte[] buf = str.getBytes();`
 - `byte[] buf = str.getBytes("Shift_JIS");`
- char配列型への変換
- char配列型
 - `char c[] = str.toCharArray();`

文字列の置き換えreplaceAll

- Java の String クラスには置換メソッドとして `replaceAll()` メソッドが用意されています。
- 文字列に指定した文字列が存在した場合に、指定した別の文字列に置換を行います。
- `String str1 = "今日の天気は晴れです。";`
- `String str2 = "晴れ";`
- `String str3 = "雨";`

- `str1.replaceAll(str2, str3);`

正規表現による置換

- Javaで正規表現のエスケープを行うには
Pattern.quote(str)を使用

```
import java.util.regex.Pattern;  
String str1 = "../test.html";  
String str2 = "../";  
String str3 = "http://www.example.com/";  
str1.replaceAll(Pattern.quote( str2 ), str3);
```

文字列の一致を判定する

```
String strA = "文字列の検索サンプルです";  
String strB = "検索";  
if (strA.equals(strB)) {  
    System.out.println( "完全一致です" );  
} else {  
    System.out.println( "完全一致ではありません"  
        );  
}
```


文字列を検索 indexOfメソッド

- 文字列を検索するには、indexOfメソッド
- `public int indexOf(String str)` indexOfメソッドは、ある文字列から別の文字列を左側から右側に検索し、見つかった位置を返す。該当する文字列がなかった場合は -1

```
String strA = "文字列の検索サンプルです";  
String strB = "検索";  
if (strA.indexOf(strB) != -1) {  
    System.out.println( "部分一致です" );  
} else {  
    System.out.println( "部分一致ではありません" );  
}
```

前方一致検索 startsWithメソッド

前方一致検索を行うには、startsWithメソッドを使用。

```
public boolean startsWith(String str)
```

一致した場合はtrue, 不一致ならfalse

```
String strC = "文字列の検索サンプルです";
```

```
String strD = "文字列の";
```

```
if (strC.startsWith(strD)) {
```

```
    System.out.println( "前方一致(接頭辞)です" );
```

```
} else {
```

```
    System.out.println( "前方一致ではありません" );
```

```
}
```

後方一致検索 endsWithメソッド

startsWithメソッドとは逆に後方一致検索を行うには、endsWithメソッドを使います。

```
public boolean endsWith(String str)
```

一致した場合はtrue, 不一致ならfalseになります。

```
String strX = "文字列の検索サンプルです";
```

```
String strY = "サンプルです";
```

```
if (strX.endsWith(strY)) {
```

```
    System.out.println( "後方一致(接尾辞)です" );
```

```
} else {
```

```
    System.out.println( "後方一致ではありません" );
```

```
}
```

正規表現での検索 matches

matchesメソッドは部分一致を検索するものではなく、完全一致を検索します。

```
String str51 = "文字列の検索サンプルです";
```

```
String str52 = "サンプル";
```

```
if (str51.matches(".*" + str52 + ".*")) {
```

```
    System.out.println( "部分一致です" );
```

```
} else {
```

```
    System.out.println( "部分一致ではありません" );
```

```
}
```

Javaで正規表現のエスケープ Pattern.quote(str)

```
import java.util.regex.Pattern;
String strA = "012345[a-z][0-9][a-z]012345";
String strB = "[a-z][0-9][a-z]";

if (strA.matches(".*" + Pattern.quote( strB ) + ".*")) {
    System.out.println( "部分一致です" );
} else {
    System.out.println( "部分一致ではありません" );
}
```

継承関係のプログラムを書いてみよう

- ClassExtA
 - methodA() println(“A is called”);
- ClassExtB extends ClassExtA
 - methodB() println(“B is called”);
- ClassExtC extends ClassExtB
 - methodC() println(“C is called”);
- ClassExtD
 - main ClassExtD d= new ClassExtD();
d.methodC();

実験してみよう

- ClassExtAのインスタンスをClassExtDでキャストできるか
- ClassExtCのインスタンスをClassExtA,ClassExtBでキャストできるか
- If文で演算子 `instanceOf`を使おう

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newを複数回呼び出してしまう。
Singleton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

Singleton
- singleton
- Singleton() + getInstance()

```
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。

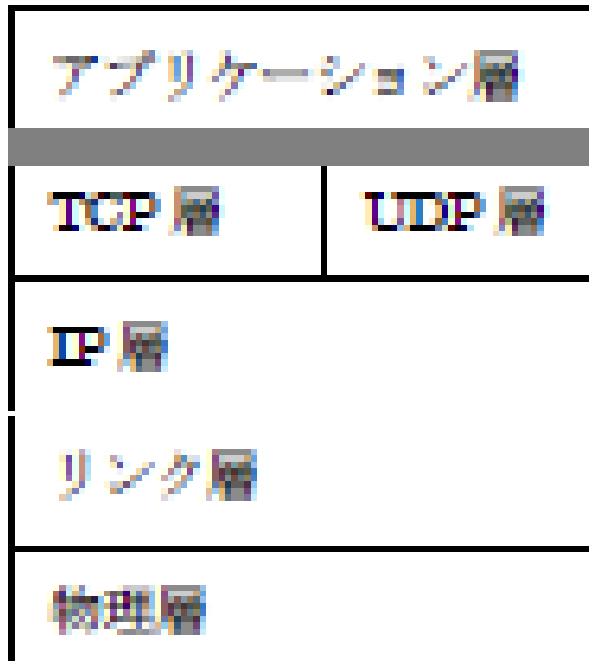
MyFirstSingletonCall

```
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



ソケットインターフェイス