第1週　導入

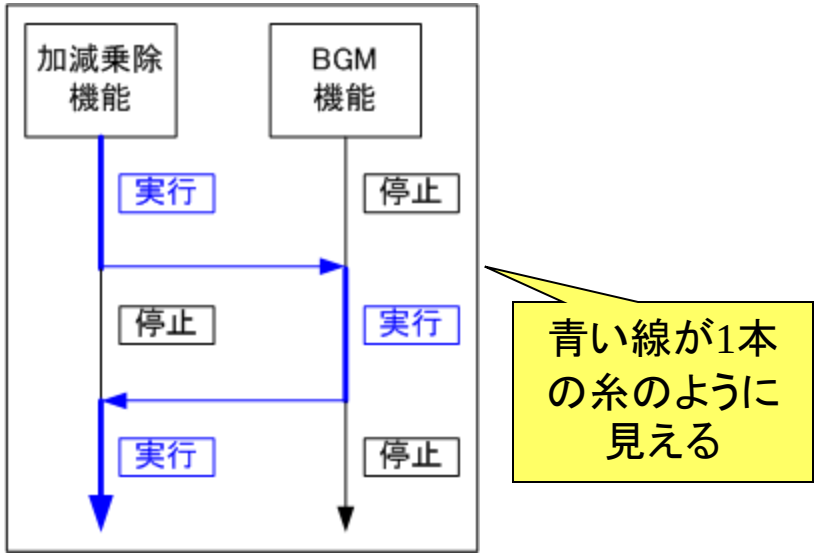
- マルチスレッドとは何か
- スレッドのライフサイクル
- クラスThreadのメソッド
- スレッドの作り方基本の使い方サンプル
- 演習1、課題1

スレッドとは何か(1)

スレッド(thread)：もともとは「糸」という意味。プログラムの実行単位のこと。
スレッドはプログラムの一部の機能を実行する。

今まで作成したプログラムは処理を1つ1つ順番に実行していた。
→シングルスレッド
(仮にBGMが流れる電卓アプレットを作成したとすると(いらないけど・・・)
そのアプレットは加減乗除の計算を行っているときBGMが停止してしまう。逆にBGMが流れていると加減乗除ができなくなってしまう→使いものにならない)

BGM機能付き電卓(シングルスレッド)



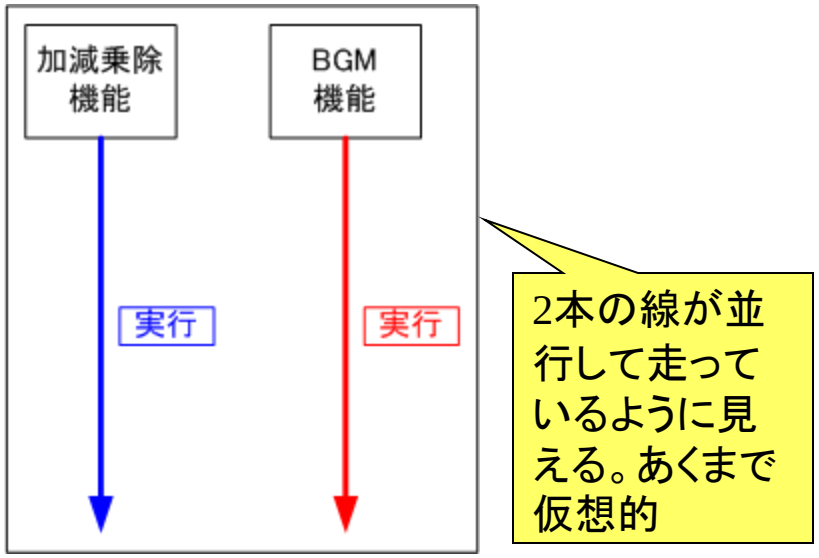
スレッドとは何か(2)

複数の処理を並行して実行するためにはどうしたらいいか？

BGMを聴きながら電卓で加減乗除ができるようにするには？
→マルチスレッドを使う

マルチスレッド：　同時に複数のスレッドが存在し、それらのスレッドが並行して動作すること。仮想的に複数の処理が並行して動作しているように見える。実際はCPUを短時間で切り替えて使っている。

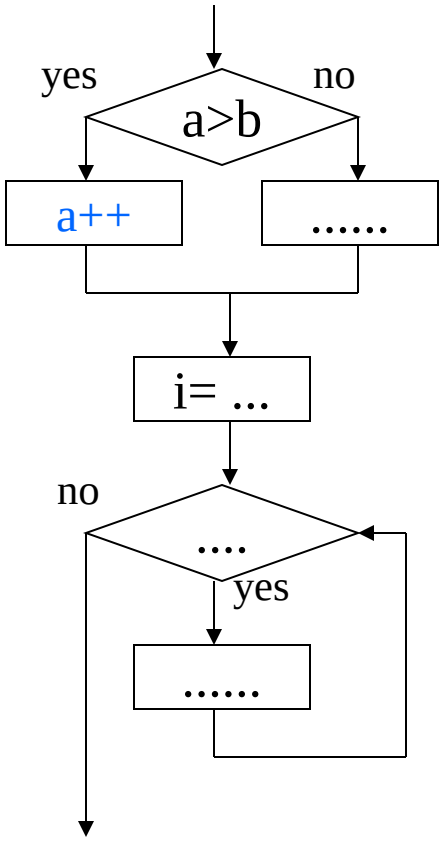
BGM機能付き電卓(マルチスレッド)



いままでやってきたプログラム

```
main(){
  int a, b, i;
  .....
  if(a>b){
     a++;
  }else{
     ........
  }
  for(i=... ){
     .....
  }
}
```

フローチャート



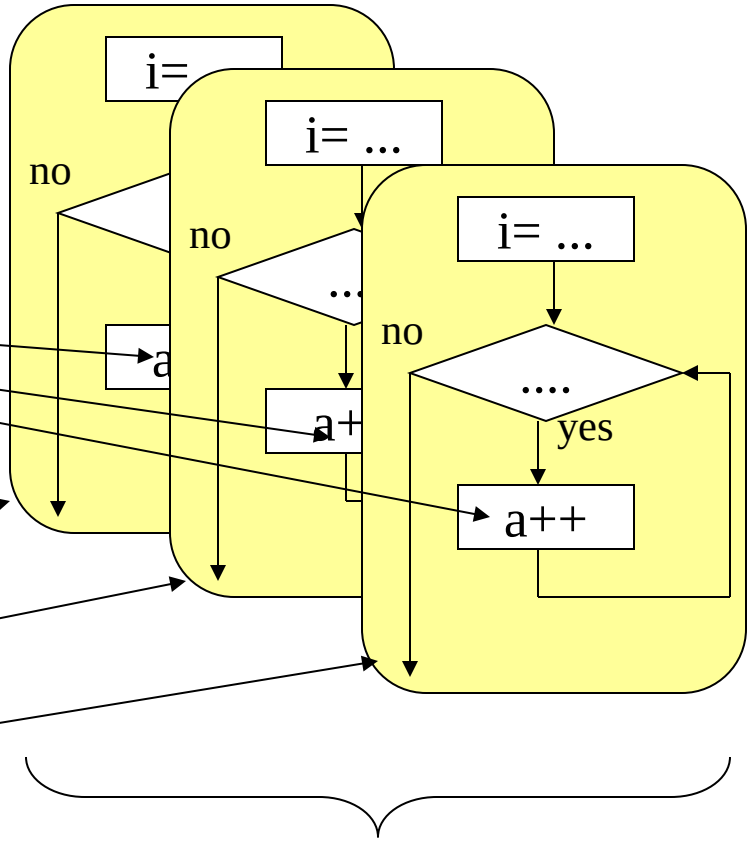
いままで学んできたプログラムは，ある一瞬には，ある1つの命令しか実行されていなかった.

処理の流れは一本である.

スレッドを利用したプログラム

```
class MyThread extends Thread{
static int a=0;
... run(){
    for(int i....){
        a++;
    }
}

.... main(....){
   .....
   new MyThread().start();
   new MyThread().start();
   new MyThread().start();
....
}
}
```



独立並行して動く.
変数も共有できる.

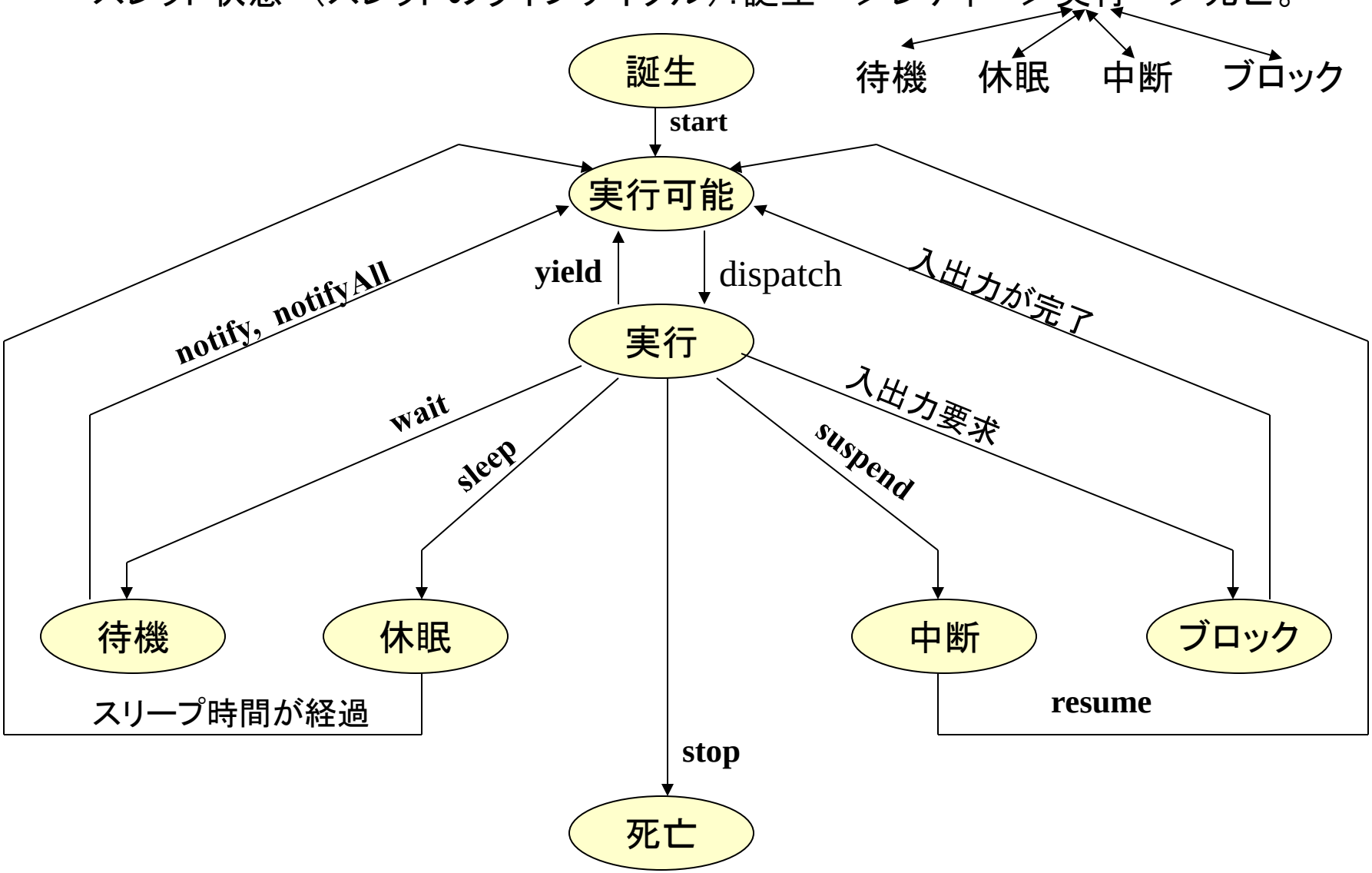
現実世界のケース

https://cis.k.hosei.ac.jp/~rhuang/Miccl/Java3/SushiApplet/index.html

二つの職人スレッド
いくつの客スレッド
➔ 独立並行して動く

スレッドのライフサイクル
(Life cycle of a thread)

スレッド状態 （スレッドのライフサイクル）: 誕生＝＞レディ＝＞実行＝＞死亡。



スレッドの定義の仕方

キーワード	スレッドクラス名前	キーワード	キーワード
class	thread_name	extends	Thread {

```
    ……  // 必要な変数とコンストラクタの宣言
    public void run() {
        ……
    }
}
```

run():抽象メソッドin Runnable
必ずrun()メソッドの再定義が
必要です。

スレッド
Count10Thread
の定義

```
class Count10Thread extends Thread{
  public Count10Thread(String threadName) {
    super(threadName);  }
  public void run(){
    for (int i = 0; i < 10; i++) {
      System.out.println(getName()+" run:¥ti=" + i);}}
}
public class Count10ThreadTest {
  public static void main(String[] args) {
    Count10Thread ctt1 = new Count10Thread("Thread1");
    Count10Thread ctt2 = new Count10Thread("Thread2");
    ctt1.start();  ctt2.start();
    }
}
```

Count10Threadのスレッド。
runメソッドが終ると消滅。

スレッドの名前を取得する

スレッドctt1/2
のrunメソッド
の実行開始

出力結果 (例)???

Abstract Method (抽象メソッド)

- キーワードとして “*abstract*” を付ける
- メソッドの中身は空

abstract void function (…) ;/*{}*/

key word　return type　name　parameter body(空)

- 抽象メソッド(*abstract* method)を含むクラスは、抽象クラス(*abstract* class)でなければならない

```
abstract class ClassName{
    abstract void FunctionName();
}
```

Abstract Class (抽象クラス)

- インスタンスの生成はできない

```
abstract class AClass{}
AClass aClass = new AClass();
```

- 親クラスとしてテンプレートのように使用される

```
abstract class AClass{}
public class BClass extends AClass{}
```

Interface (インターフェース)

- クラスや他のインターフェースを作成するためのテンプレートとして使用される

  interface *Interface1*{}

  public class ClassName implements *Interface1*{}

- インスタンスの生成はできない

  Interface1 interface1 = *new* Interface1();

- 初期化されないメンバ変数の宣言はできない

  interface Interface1{

  int var;

  }

- メソッドはすべて抽象メソッド(*abstract* method)となる

  interface Interface1{

  (abstract) void function();

  }

マルチスレッド処理機能です。これはJava言語の特徴の一つです。

Javaでは、スレッドというのは、一つのオブジェクトです。

クラススレッドのコンストラクタ
public Thread()
public Thread(String threadName)
……

クラススレッドのインスタンス
Thread thread1, thread2;
thread1 = new Thread();
thread2 = new Thread(“student”);

```
java.lang.Object
  |
  +--java.lang.Thread
```

Java API

スレッドの例の出力結果

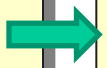
```
class Count10Thread extends Thread{
  public Count10Thread(String threadName) {
    super(threadName);  }
public void run(){
    for (int i = 0; i < 10; i++) {
      System.out.println(getName()+" run:¥ti=" + i);

      try {
         // 5000ミリ秒間スレッドを停止します
         sleep(5000);
         //最大5000ミリ秒間スレッドを停止します
         // sleep((int) ( Math.random() * 5000 ));
       } catch (InterruptedException e) {
       }

     }
   }
}
public class Count10ThreadTest {
  public static void main(String[] args) {
    Count10Thread ctt1 = new Count10Thread("Thread1");
    Count10Thread ctt2 = new Count10Thread("Thread2");
    ctt1.start();  ctt2.start();
    }
}
```

出力結果 (例)

```
Thread1 run:  i=0
Thread1 run:  i=1
Thread1 run:  i=2
Thread1 run:  i=3
Thread1 run:  i=4
Thread1 run:  i=5
Thread1 run:  i=6
Thread1 run:  i=7
Thread1 run:  i=8
Thread1 run:  i=9
Thread2 run:  i=0
Thread2 run:  i=1
Thread2 run:  i=2
Thread2 run:  i=3
Thread2 run:  i=4
Thread2 run:  i=5
Thread2 run:  i=6
Thread2 run:  i=7
Thread2 run:  i=8
Thread2 run:  i=9
```



```
Thread1 run:  i=0
Thread2 run:  i=0
Thread2 run:  i=1
Thread2 run:  i=2
Thread1 run:  i=1
Thread1 run:  i=2
Thread1 run:  i=3
Thread1 run:  i=4
Thread1 run:  i=5
Thread1 run:  i=6
Thread1 run:  i=7
Thread1 run:  i=8
Thread1 run:  i=9
Thread2 run:  i=3
Thread2 run:  i=4
Thread2 run:  i=5
Thread2 run:  i=6
Thread2 run:  i=7
Thread2 run:  i=8
Thread2 run:  i=9
```

Think about

1. No sleep time
2. A fixed sleep time
3. A maximum sleep time

スレッドクラスのメソッド
(The Thread methods)

スローされうる例外

public void **start()**
runメソッドを起動し、期待された仕事を行う。
IllegalThreadStateException

public final void **stop()**
ThreadDeathオブジェクトを送出することによってスレッドを停止させる。
SecurityException

public final void **suspend()**
スレッドの実行を中断される。

public final void **resume()**
中断されたスレッドを再開させる。

public static void **sleep(long millis)**
スレッドを眠らせる。
InterruptedException

public final void **wait()**
ターゲットオブジェクトを待つためにwaitメソッドを呼び出し、待機状態に入る。

public final void **notify()**
待ち行列の先頭にあるスレッドが実行可能状態に移る。
IllegalMonitorStateException

public final void **notifyAll()**
待ち行列にあるスレッドがすべて実行可能状態に移る。

public final void **setName(String name)**
スレッドの名前を設定する。
SecurityException

public final ***String*** **getName()**
スレッドの名前を返す。

public static ***Thread*** **currentThread()**
カレントスレッドへの参照を返す。

public final ***boolean*** **isAlive()**
スレッドが生きているかを判断する。

public void run()

スレッドstart()を呼び出すとき、
run()メソッドを起動します。

簡単な例
A simple example

クラスThreadから導出したサブクラスのスレッドSleepPrintThreadを4つ生成し、その中でThreadメソッドsleepを呼び出します。これらのスレッドは0～5秒間（乱数で決定）眠ったあと、それぞれの名前を表示します。

```
class SleepPrintThread extends Thread {
  int sleepTime;
  public SleepPrintThread()
  {
    // 0～5秒間スリープ
    sleepTime = (int) ( Math.random() * 5000 );
    System.out.println( "Name: " + getName() +
                        ";  sleep: " + sleepTime );
  }

  public void run()
  {
    try {
      Thread.sleep( sleepTime );
    }
    catch ( InterruptedException exception ) {
      System.err.println( exception.toString() );
    }
    System.out.println( getName() );
  }
}
```

スレッドSleepPrintThreadのsleepメソッドの実行開始

スレッドSleepPrintThreadのgetNameメソッドを呼び出す

```
public class PrintTest {
  public static void main( String args[] )
  {
    SleepPrintThread thread1. thread2;
    SleepPrintThread thread3, thread4;

    thread1 = new SleepPrintThread();
    thread2 = new SleepPrintThread();
    thread3 = new SleepPrintThread();
    thread4 = new SleepPrintThread();

    thread1.start();
    thread2.start();
    thread3.start();
    thread4.start();
  }
}
```

四つSleepPrintThreadオブジェクトを生成するスレッド名前とスリープ時間を出力する

runメソッドが実行されると、スレッド名前を出力する

実行結果：?

簡単な例の実行結果

```
public class PrintTest {
  public static void main( String args[] )
  {
    SleepPrintThread thread1. thread2;
    SleepPrintThread thread3, thread4;

    thread1 = new SleepPrintThread();
    thread2 = new SleepPrintThread();
    thread3 = new SleepPrintThread();
    thread4 = new SleepPrintThread();


    thread1.start();
    thread2.start();
    thread3.start();
    thread4.start();
  }
}
```

結果例１:

```
Name: Thread-0;  sleep: 1774
Name: Thread-1;  sleep: 1353
Name: Thread-2;  sleep: 4627
Name: Thread-3;  sleep: 4003
Thread-1
Thread-0
Thread-3
Thread-2
```

結果例２:

```
Name: Thread-0;  sleep: 2258
Name: Thread-1;  sleep: 91
Name: Thread-2;  sleep: 2823
Name: Thread-3;  sleep: 1369
Thread-1
Thread-3
Thread-0
Thread-2
```

マルチスレッドの例

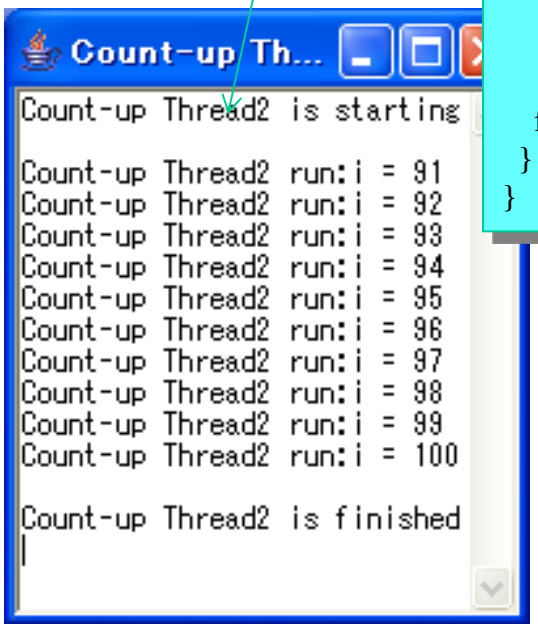
```
class Count10ThreadDown extends Thread{
  public Count10ThreadDown(String threadName){
    super(threadName);
  }
  public void run(){
    String threadName = getName();
    System.out.println (threadName + " is started!");
    for (int i = 0; i < 10; i++) {
      try {sleep((int)(Math.random()* 3000));}
      catch(InterruptedException e ) {};
      System.out.println(threadName+" run:¥ti=" + (9-i));
    }
  System.out.println(threadName+" is done!");
  }
}
```

```
public class MultiThreadDemo {
public static void main(String[] args) {
    Count10ThreadDown thread1 = new Count10ThreadDown("Count-down Thread1");
    CountNumThreadUp thread2 = new CountNumThreadUp("Count-up Thread2", 91);
    thread1.start();
    thread2.start();
}}
```

```
class CountNumThreadUp extends Thread{
  OutputWindow f;
  int k;
  public CountNumThreadUp(String threadName, int k){
    super(threadName); this.k=k;
    f = new OutputWindow(threadName);
  }
  public void run(){
    f.newline(getName()+ " is starting¥n¥n");
    for (int i = 0; i < 10; i++) {
      f.newline(getName()+" run:i = " + (k+i)+"¥n");
      try {sleep((int)(Math.random()* 3000));
          } catch(InterruptedException e )
            {};
    }
    f.newline("¥n"+getName()+ " is finished¥n");
  }
}
```

On Console Window

```
Count-down Thread1 is started!
Count-down Thread1 run:i=9
Count-down Thread1 run:i=8
Count-down Thread1 run:i=7
Count-down Thread1 run:i=6
Count-down Thread1 run:i=5
Count-down Thread1 run:i=4
Count-down Thread1 run:i=3
Count-down Thread1 run:i=2
Count-down Thread1 run:i=1
Count-down Thread1 run:i=0
Count-down Thread1 is done!
```



Two threads:

1. One displays its output on Console
2. Another is on a created window

他のクラス (new)

```
import javax.swing.*;

public class OutputWindow extends JFrame {
        private JTextArea text;

        public OutputWindow(String title) {
                super(title);
                text = new JTextArea(10, 20);
                add(text);
                setSize(250, 400);
                setVisible(true);

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }

        public void newline(String line) {

                text.append(line);
```

Work in Class

Run all examples （Four examples）
in this lecture notes

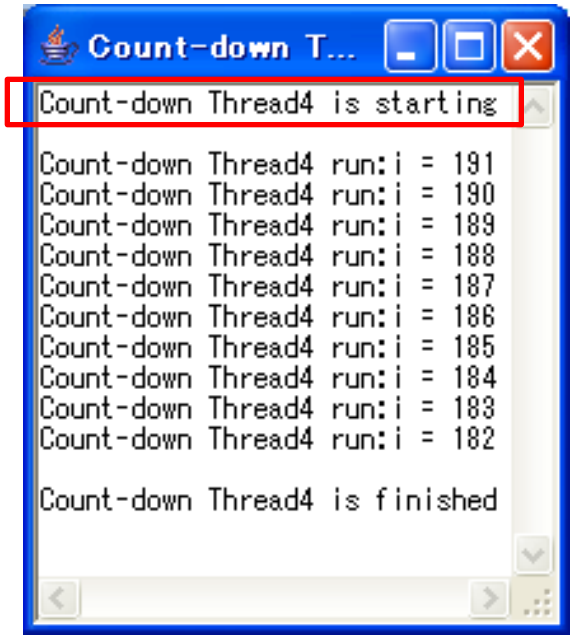
Home Work:演習1

p17~p18のMultiThread Demoのプログラムに以下のthread3,4を追加せよ

Thread3 :Count10ThreadUPスレッド(0から9までカウントアップ)

Thread4:CountNumThreadDownスレッド（指定した数から10カウントダウン。Frameで出力）

演習1の出力例




```
Count-down Thread1 is started!
Count-up Thread3 is started!
Count-up Thread3 run:i=0
Count-down Thread1 run:i=9
Count-up Thread3 run:i=1
Count-down Thread1 run:i=8
Count-up Thread3 run:i=2
Count-down Thread1 run:i=7
Count-up Thread3 run:i=3
Count-up Thread3 run:i=4
Count-down Thread1 run:i=6
Count-up Thread3 run:i=5
Count-up Thread3 run:i=6
Count-down Thread1 run:i=5
Count-up Thread3 run:i=7
Count-down Thread1 run:i=4
Count-up Thread3 run:i=8
Count-down Thread1 run:i=3
Count-down Thread1 run:i=2
Count-down Thread1 run:i=1
Count-up Thread3 run:i=9
Count-up Thread3 is done!
Count-down Thread1 run:i=0
```

Home Work:課題1

Ex1. 以下のような、つぎの2つのスレッドクラスを定義せよ。

SquareThread class:

スレッドの名前をセットするコンストラクタ

for i=1～9,

iの値とスレッドの名前、i*iの値を0～3000ミリ秒のスリープタイム

で出力する。

スレッドの名前と文字列"is done!"を出力する。

CubeThread class:

スレッドの名前をセットするコンストラクタ

for i=1～9

iの値とスレッドの名前、i*i*iの値を0～3000ミリ秒のスリープタイム

で出力する。

スレッドの名前と文字列“is done!”を出力する。

SquareThreadスレッドとCubeThreadスレッドを生成してスタートさせる

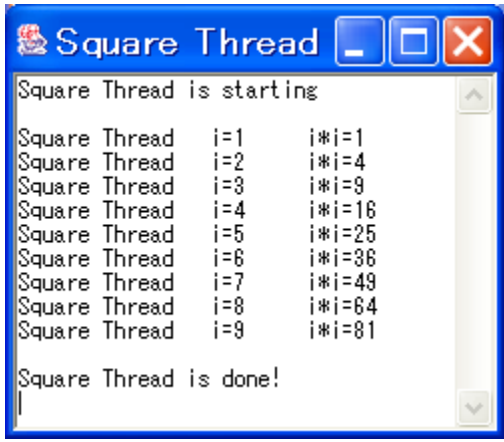
Javaアプリケーションプログラムを記述せよ。

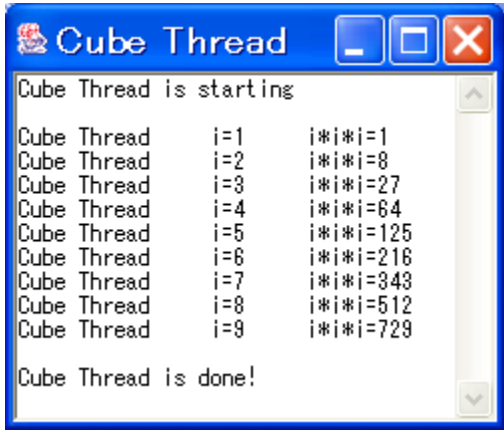
(ヒント: Count10ThreadUp　Count10ThreadDownを参照)

Ex1の出力例

```
Square Thread is started!
Cube Thread is started!
Square Thread    i=1    i*i = 1
Square Thread    i=2    i*i = 4
Square Thread    i=3    i*i = 9
Cube Thread      i=1    i*i*i = 1
Square Thread    i=4    i*i = 16
Cube Thread      i=2    i*i*i = 8
Square Thread    i=5    i*i = 25
Square Thread    i=6    i*i = 36
Cube Thread      i=3    i*i*i = 27
Square Thread    i=7    i*i = 49
Square Thread    i=8    i*i = 64
Cube Thread      i=4    i*i*i = 64
Cube Thread      i=5    i*i*i = 125
Cube Thread      i=6    i*i*i = 216
Square Thread    i=9    i*i = 81
Square Thread is done!
Cube Thread      i=7    i*i*i = 343
Cube Thread      i=8    i*i*i = 512
Cube Thread      i=9    i*i*i = 729
Cube Thread is done!
```

Ex2の出力





Ex2. Ex1をFrameを使って書き換えよ (演習1にのCountNumThreadUp と CountNumThreadDownを参照)