OPEN STREAM

213-05.スレッド

Make IT your Success

目次

スレッド
- スレッドの概念
- プログラムは動く？誰が動かしている？
- スレッドとは
- スレッドとプロセス(とマルチコアCPUとハードウェアマルチスレッディング)

スレッドの生成と開始
- 新しいスレッドを生成するには？
- 新しく生成したスレッドに何かやらせるには？

オーバーライドしたrun()

スレッドの状態

Singletonパターン

synchronizedメソッド

synchronized(){}ブロック

スレッドの状態を変える命令

この講義について

目的：

- スレッドの生成・開始など、使用方法を学ぶ。
- synchronizedの機能、使用方法について学ぶ。

ゴール：

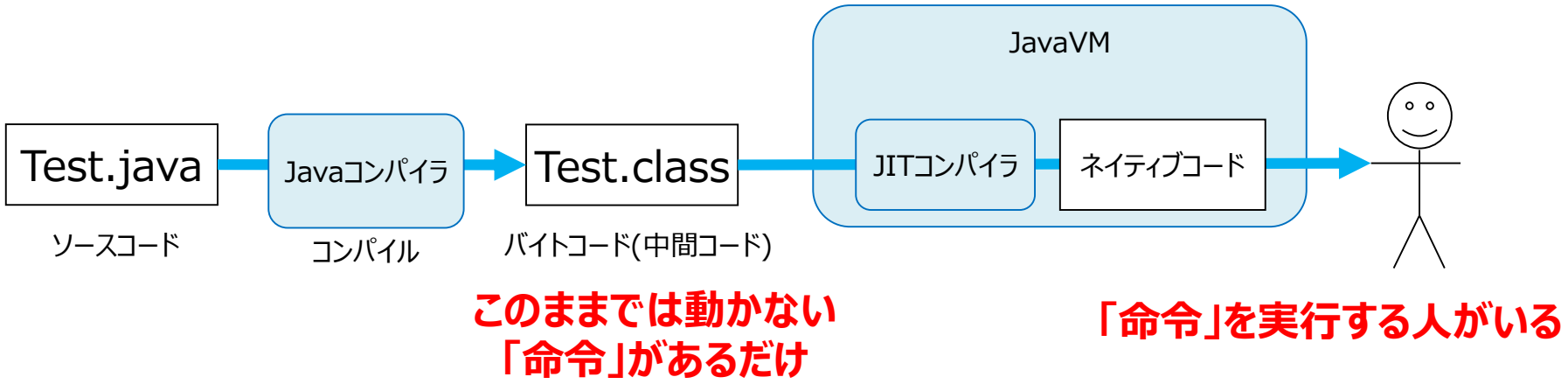
- スレッドとは何か、概念を理解すること。
- Threadクラスの継承とRunnableインタフェースの実装の使い分けができること。

スレッド

スレッドの概念

- 「プログラム」は動かない
- プログラムは命令が書いてあるものに過ぎない　この命令を動かす人は別にいる
- 命令を動かす人はOSの機能を利用することになる
- プログラムを動かす人＝スレッド

プログラムは動く？



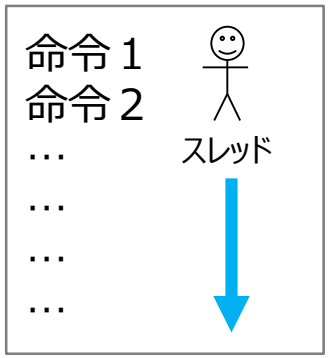
「プログラム」自体は動かない。
プログラムは命令が書いてあるだけ。 この命令を動かすものは別にいる。
命令を動かすもの(命令を実行する人)はOSの機能を利用する

スレッドとはプログラムを実行する人、一人を指す。
実行する人は増やすことができる。

スレッドとは

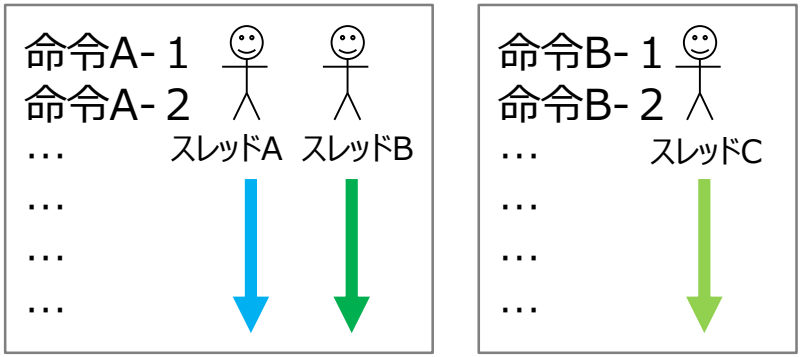
プログラムを動かす人(1つの単位)です。詳しくは、命令を実行しているCPUを利用する単位です。
プログラムの命令を順番に実行してゆきます。
スレッドは一つの命令の実行が終わらないと、次の命令には進めません。

一つのプログラムで複数のスレッドを生成し、
同時に複数のスレッドに命令を実行させる事もできます。
これを**マルチスレッドプログラミング**をといいます。



プログラム

スレッドが上から順番に命令を実行してゆく
命令1が終わったら、命令2へ
一つの命令の実行が終わらないと
次の実行へは進まない



プログラム

一つのプログラムの中で、
同時に複数のスレッドが
命令を別々に実行する事もできます。
スレッドA、スレッドB、スレッドCは
別々に命令を実行します。
(スレッド自身は1つの命令が終わらないと
次の命令に進めません)

スレッドとプロセス(とマルチコアCPUとハードウェアマルチスレッディング)

1つのCPUが並行して命令を実行するしくみ

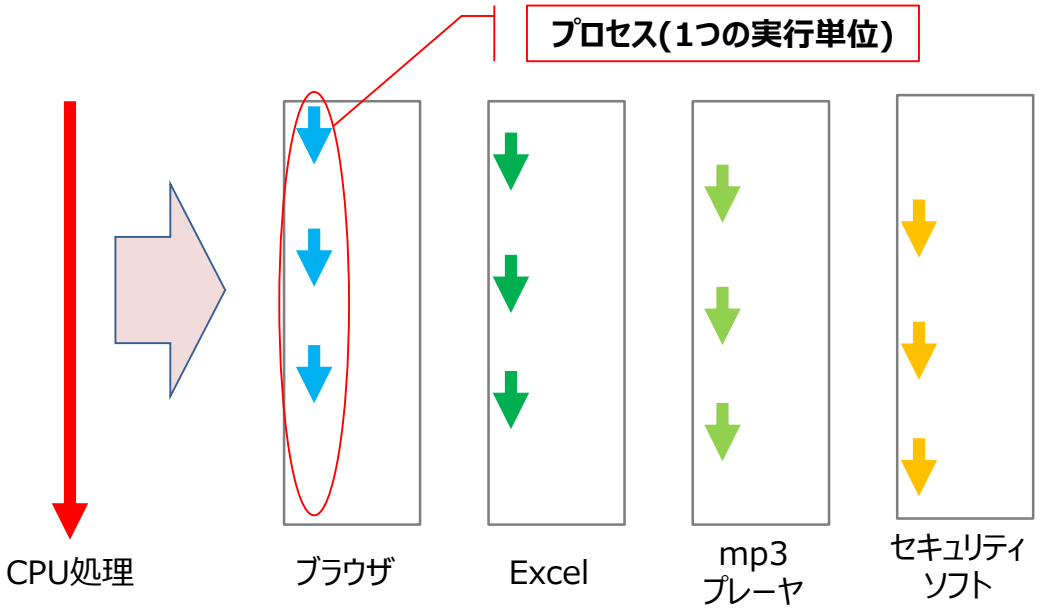
1つのCPUは1つの演算装置を持っています。1つの演算装置は同時に1つの命令しか実行できません。つまり(通常は)1つのCPUでは1つの命令を実行し、その命令が実行終了しないと、次の命令は実行できません。（並行して複数の命令を実行できない）

しかし、PCではブラウザを立ち上げながらオフィスソフトを利用したり、複数のアプリケーションを同時に実行しています。(マルチタスク)

WindowsなどOS上でCPU処理を細かく分断し、複数のアプリケーションを順番に実行する事でマルチタスクを疑似的に実現しています。

この一つの実行単位を**プロセス**といいます。(アプリケーションの実行がプロセスととらえて良いです)

スレッドはアプリケーション上での実行単位です。



CPU処理を細かく分断し疑似的にマルチタスクを実現している

スレッド⊆プロセス

スレッドとプロセス(とマルチコアCPUとハードウェアマルチスレッディング)

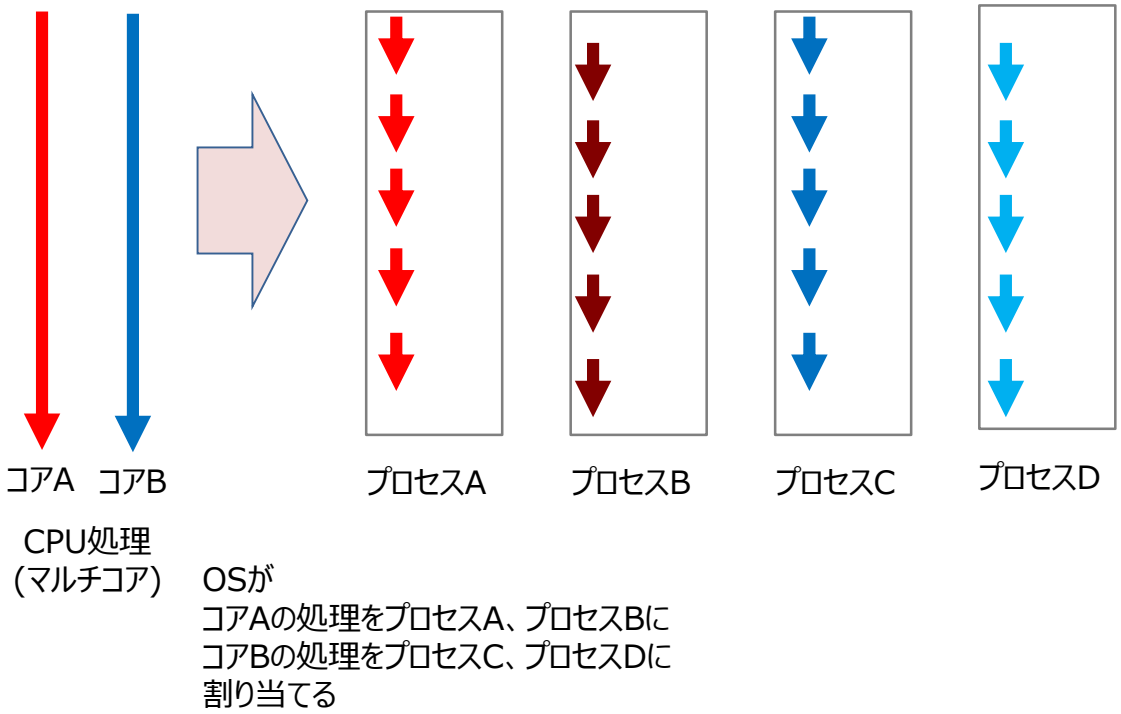
階層イメージ

CPU
↓↑
OS
↓↑
マルチタスク
プロセス(タスク)　プロセス(タスク)　プロセス(タスク)
↓↑
マルチスレッド
スレッド　スレッド　スレッド　スレッド

スレッドとプロセス(とマルチコアCPUとハードウェアマルチスレッディング)

現在のPCはマルチコアCPUを使用しています。マルチコアCPUとは複数のプロセサ(演算装置と制御装置)を持つCPUです。(複数のCPUがあるのと同じ)マルチコア対応のOSは、複数のCPU処理を複数のプロセスに割り当てます。

一つのコア(一つのCPU)で複数の処理を行う機能をハードウエアマルチスレッディングといいます。
(インテル社のハイパースレッディング・テクノロジーが一般的な呼び方になっています)
総コア数 x 1コア当りのスレッド数がCPU処理の総数になります。
(OSはこの総数を各プロセスに割り当てます)



スレッドの生成と開始

新しいスレッドを生成するには？

- ThreadTest1
  →スレッドを一人生成して、そのスレッドを開始して、main()メソッドを終了。

- mainスレッド
  →最初に自動で生まれる。
  →main()メソッドを実行する。

- ２人目以降のスレッドを生成するには？
  →スレッドを生成する命令を他のスレッド（＝最初はmainスレッドしかいない）が実行することによって新しいスレッドが誕生する。
  →スレッドを生成する命令
  Threadクラスのstart()メソッド。
  これ以外にスレッドを生成する命令は存在しない。
  start()メソッドはnativeメソッドであり、要するにスレッドを生成して開始させることはjavaにはできず、OSのスレッドの機能を呼び出すことになる。

スレッドの生成と開始

新しく生成したスレッドに何かやらせるには？

- run()メソッドをオーバーライド。run()メソッド＝スレッドの人生が書かれている。
- スレッドは start()がよばれると、自分のrun()メソッドに書かれている自分の人生を実行する。run()が終了すると スレッドの人生は終わる。

ThreadTest2.java

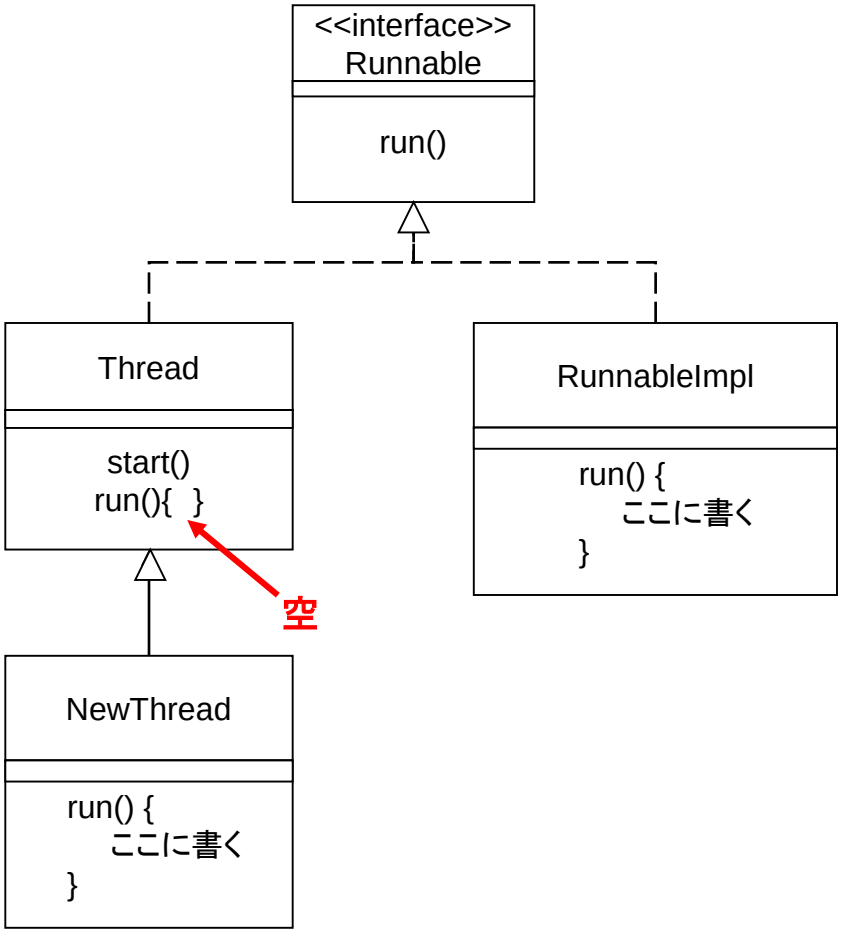
- デフォルトの名前、main, Thread-0, Thread-1,…
- Thread#getName(), Thread#setName()
- スレッドの名前を調べる。

```
Thread.currentThread().getName();
```

オーバーライドしたrun()

run()

run()はThreadクラスが実装しているinterfaceのメソッド



◎ポイント

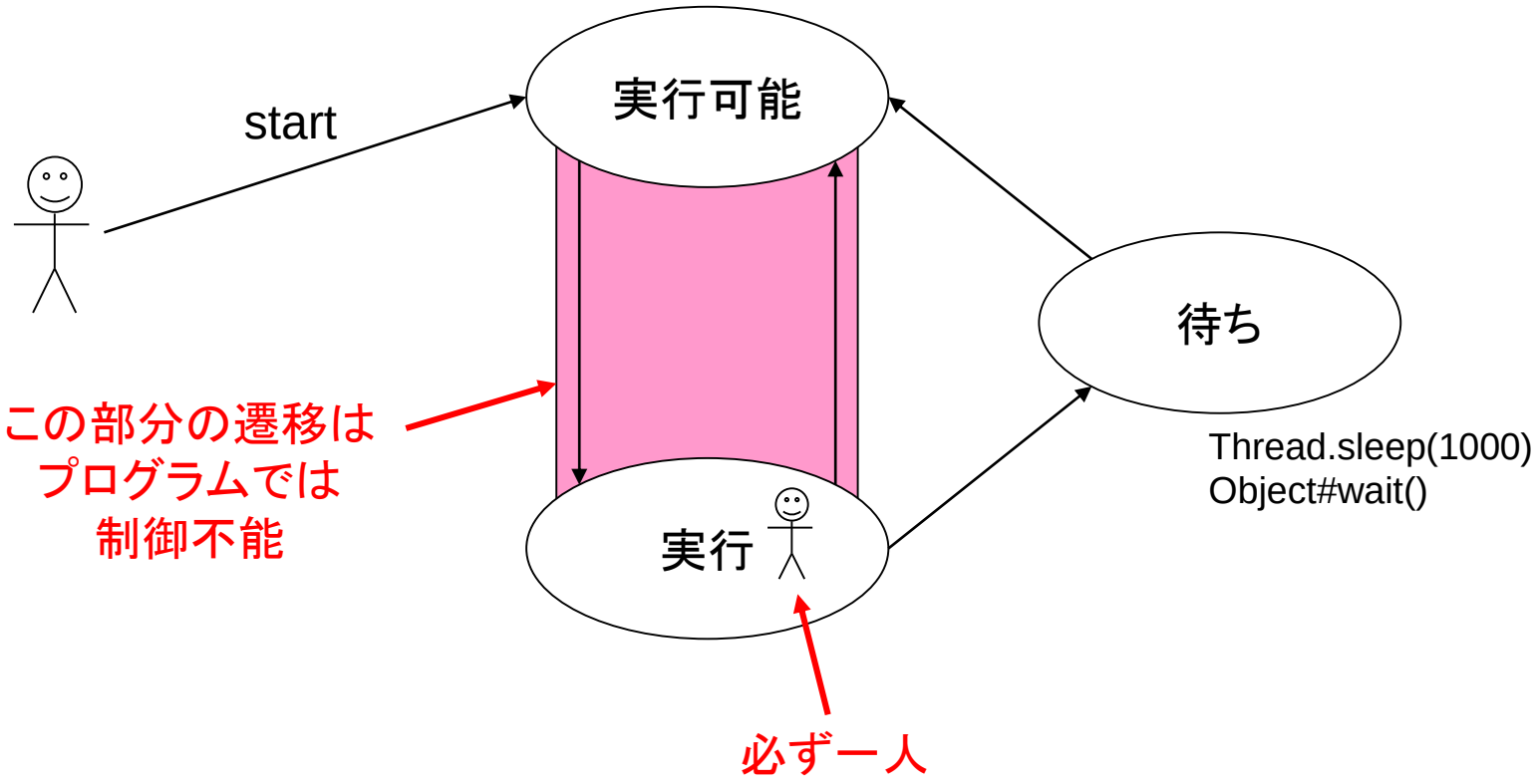
Threadクラスの継承と
Runnableインタフェースの実装
使い分けは？

・どうモデリングしたいか？
・どういうクラス名にしたいか？

例：
家事(動作そのもの)と
お母さん(実行するひと)のような。

スレッドの状態

◎ポイント
・同時に実行しているのは1人だけ(CPU数だけ)
・実行可能状態と実行の制御はプログラムでは不能

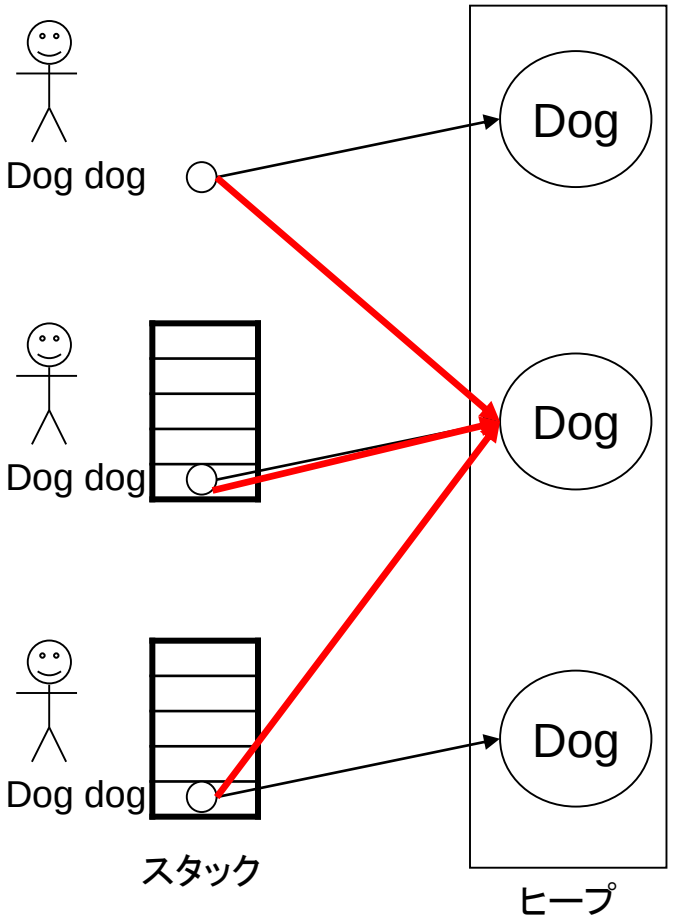


スレッドの状態

下記のrun()メソッドを複数のスレッドが同時に実行してもよいか？

```
public void run(){
        Dog dog = new Dog();
        while( true ){
                dog.walk();
                dog.feed();
        }
}
```

スレッドの状態



◎ポイント
・スタックはThreadのもちもの

もしこうなってるといけない！

Singletonパターン

Singletonパターン(GoF)とは？

- １つのクラスに対して、１つのインスタンスしか生成できないことを保証したクラスの定義の仕方。

実現方法

- 「クラスに対して１つ」をどのように実現するか？
  - クラスに１つ→クラスに一回の処理→staticフィールドの初期化→staticフィールドを初期化するタイミングでnewする。

※これテストで出るので絶対紹介すること。

・コンストラクタをprivate化

・フィールドに自分のインスタンスを持つ

・getInstanceメソッドをつける

Singletonパターン

①Dogをシングルトンにしてみる

- 最初はsynchronizedはつけない！

②マルチスレッドで動かしてみる

- さっきのThreadTest3とNewThread3で確認する
- 想定しない値(さっきの図)になることを確認する
- 確かに複数スレッドで1匹の犬を操作していることを確認する

③どうしてこうなったか行を追って見てみる

synchronizedメソッド

1. スレッドは、synchronizedメソッドに入る前に、そのメソッドの属するオブジェクトをロックする。
2. ロックは 1 人しかできない。（排他ロックという）
3. スレッドは、既にロックされているオブジェクトをロックしようとすると待たされる（＝待ち状態になる）。
4. スレッドは、synchronizedメソッドを抜けると、ロックを解放する。

synchronized(){}ブロック

1. スレッドは、***synchronizedブロック***に入る前に、**パラメータで指定された**オブジェクトをロックする。
2. ロックは１人しかできない。（排他ロックという）
3. スレッドは、既にロックされているオブジェクトをロックしようとすると待たされる（＝待ち状態になる）。
4. スレッドは、***synchronizedブロック***を抜けると、ロックを解放する。

スレッドの状態を変える命令

wait()

ほかのスレッドがこのオブジェクトの notify()メソッドまたはnotifyAll()メソッドを呼び出すまで、現在のスレッドを待機させます。

notifyAll()

このオブジェクトのモニターで待機中のすべてのスレッドを再開します

sleep()

別のスレッドがこのオブジェクトの notify()メソッドまたは notifyAll() メソッドを呼び出すか、指定された時間が経過するまで、現在のスレッドを待機させます

スレッドの生成……スレッドプール

ExecutorServiceクラスとスレッドプール

Java5以降は、スレッド生成にはExecutorServiceクラスのスレッドプールを使う。
※アプリケーション全体でのスレッド数を管理する為

Executorsクラス

ExecutorServiceオブジェクト(スレッドプール)を生成する。

newFixedThreadPoolメソッド

スレッド数を固定で決める。
→アプリケーション全体でのスレッド数を決める事ができる。

newCachedThreadPool

必要に応じてスレッドを生成する。