45

Java の覚え書き

マルチスレッド

スレッドとは 1 つのプログラム上で動作するある特定の処理のことを言います。
1 つのプログラム上でいくつもの処理を同時に実行しているかのように見せたい時に使用します。
このようなプログラムのことをマルチスレッドプログラムと言います。

スレッドを作るには２つの方法があります。

方法 A

Thread クラスの拡張クラスを宣言する

class ClassName extends Thread

Thread クラスを継承する事で、
スレッド機能を簡単に実装する事ができますが、
継承済みのクラスにはこの方法は利用できません。

方法 B

Runnable インタフェースを宣言する

class ClassName implements Runnable

インタフェースを利用するので、
既に継承済みのクラスでも実装できます。
ただし、Thread 機能を持つ訳ではないので、
Thread クラス経由で実行する必要があります。

46

Java の覚え書き

マルチスレッドプログラムの記述例

方法 A（extends Thread）

```
public class Execute {
        public static void main(String[] args) {
                MyClass a = new MyClassA ();
                a.start();

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

class MyClassA extends Thread {
        public void run() {
                for (int i=0; i<3; i++) {
                        System.out.println("run: i = " + i);
                }
        }
}
```

MyClass のインスタンスから直接実行

スレッド #1 の処理

run() メソッドにスレッド #2 の処理を記述

方法 B（implements Runnable）

```
public class Execute {
        public static void main(String[] args) {
                MyClass b = new MyClassB ();
                Thread th = new Thread(b);
                th.start();

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

class MyClassB implements Runnable {
        public void run() {
                for (int i=0; i<3; i++) {
                        System.out.println("run: i = " + i);
                }
        }
}
```

Thread のインスタンスを経由して実行

スレッド #1 の処理

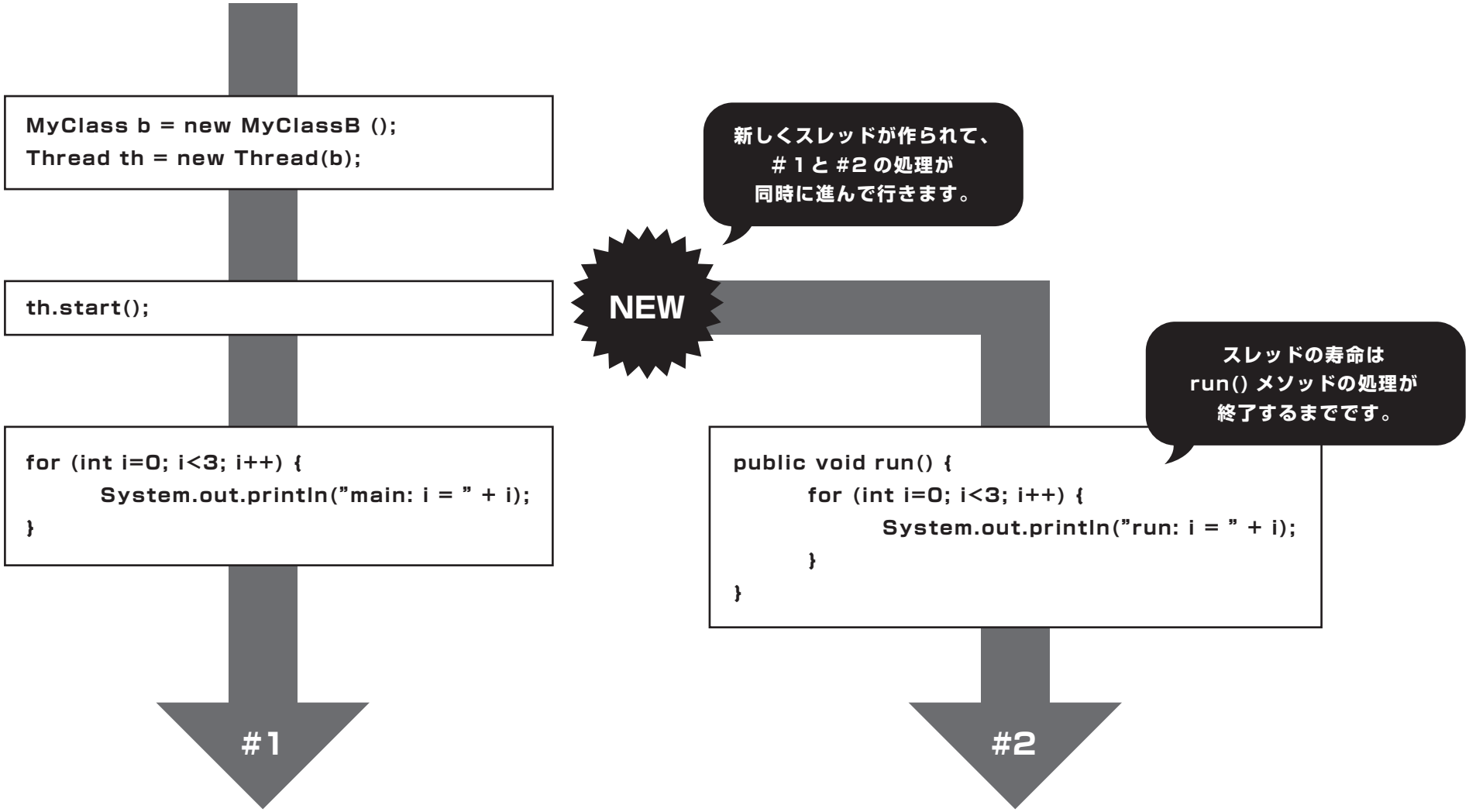
run() メソッドにスレッド #2 の処理を記述

プログラムの実行結果

```
>java Execute
main: i = 0
main: i = 1
run: i = 0
main: = 2
run: i = 1
run: i = 2
```

スレッドの実行順は Java が独断的に決めてしまうため、
main で始まる文字列と run で始まる文字列が混じるように表示されます。
（プログラムの実行結果は実行のたびに異なります）

マルチスレッドプログラムの実行時の流れ



48

Java の覚え書き

スレッドの同期（排他制御）

マルチスレッドプログラミングでは同時に複数のスレッドが並行して実行されることになります。
よって、１つのオブジェクトに複数のスレッドから同時にアクセスする可能性が出てきます。
あるスレッドがあるオブジェクトをデータを読み取っている最中に
他のスレッドがそのデータの書き換えを行うなど、プログラマが意図していない結果になってしまう場合があります。
このようなことが起こらないように、synchronized と呼ばれる機構が用意されています。

synchronized メソッド

```
synchronized void methodName() {
      …
}
```

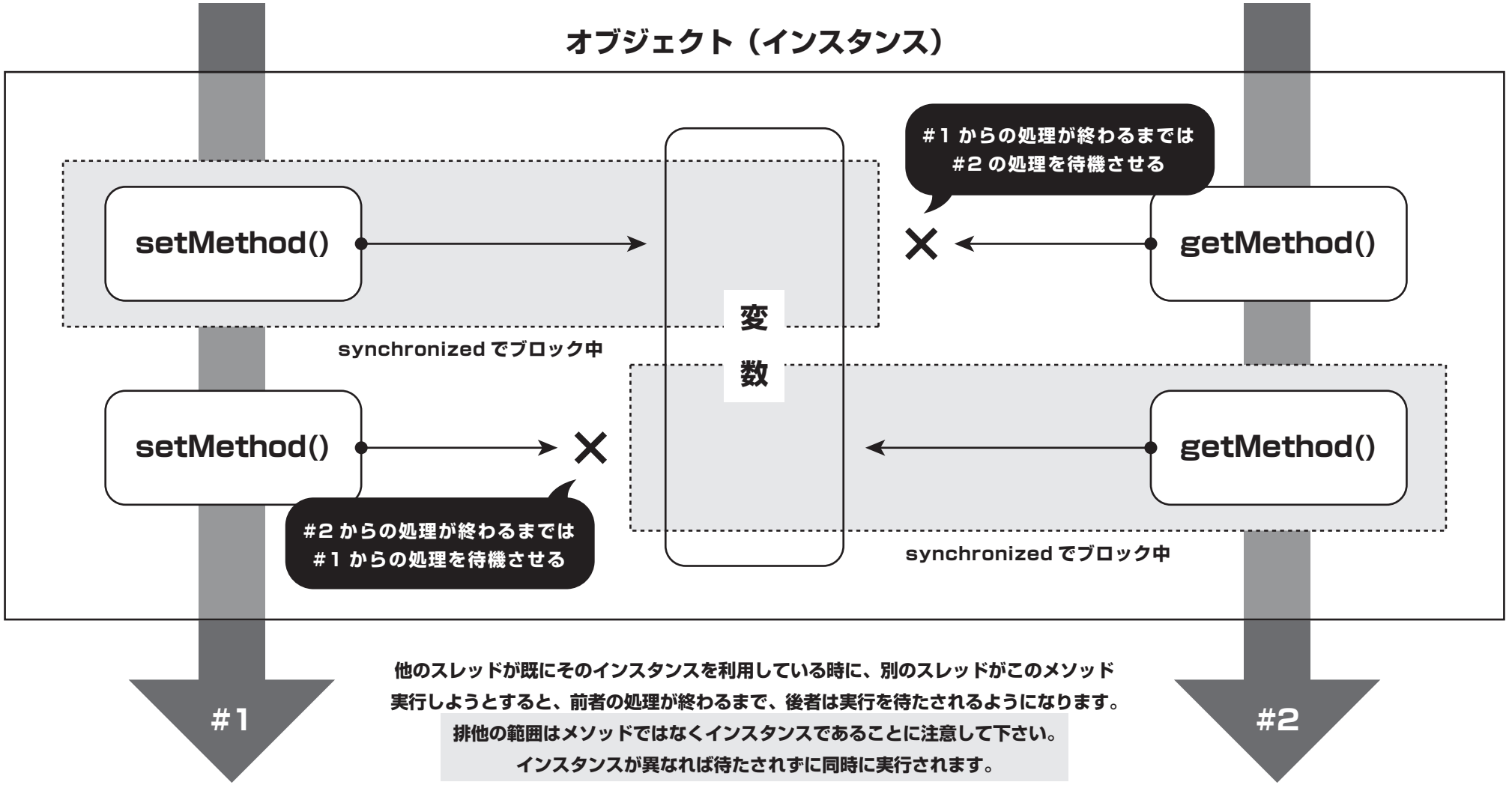
オブジェクトで同期を取りたいメソッドに
synchronized 修飾子をつける

synchronized ブロック

```
…
synchronized（ロックオブジェクト）{
      …
}
…
```

オブジェクトで同期を取りたい部分に
synchronized ブロックで囲う

synchronized の動作イメージ



他のスレッドが既にそのインスタンスを利用している時に、別のスレッドがこのメソッド実行しようとすると、前者の処理が終わるまで、後者は実行を待たされるようになります。

排他の範囲はメソッドではなくインスタンスであることに注意して下さい。
インスタンスが異なれば待たされずに同時に実行されます。

50

Java の覚え書き

スレッドを一時停止する

```
Thread.sleep( 停止時間 );
```

スレッドを一時停止したいときには
Thread.sleep メソッドを使用します。
Thread.sleep を利用する場面としては次ような時があります。

スレッドの動作が早すぎて不都合がある時。

時間間隔をあけたい時。

Thread.sleep の引数には停止時間を設定しますが、
ミリ秒単位なので注意しましょう。
（1 秒 = 1000 ミリ秒）

51

Java の覚え書き

スレッドの終了を待つ

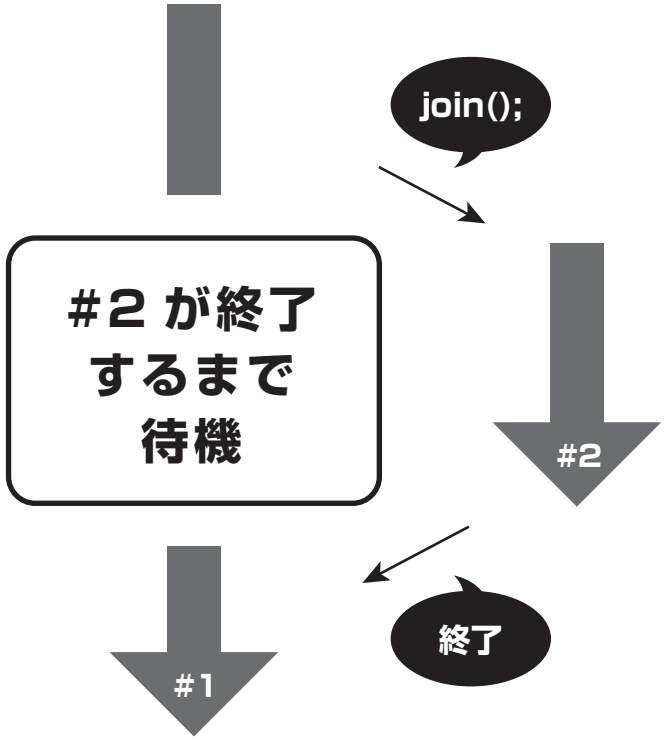
インスタンス変数名 .join();

別のスレッドにある処理を任せ、そのスレッドが終了したときに、
自分のスレッドの処理を再開したいという場面があります。
このようなとき、Thread クラスの join メソッドを利用します。

join メソッドには、引数の異なる３つのメソッドがあります。

```
void join()
void join(long millis)
void join(long millis, int nanos)
```

join() は該当のスレッドが終了するのを永久に待ちつづけます。
引数のあるものは、該当のスレッドが終了しなくても、指定した時間経過すれば処理を再開します。
（ただし、join(0) は join() と同じように永遠に待機します）

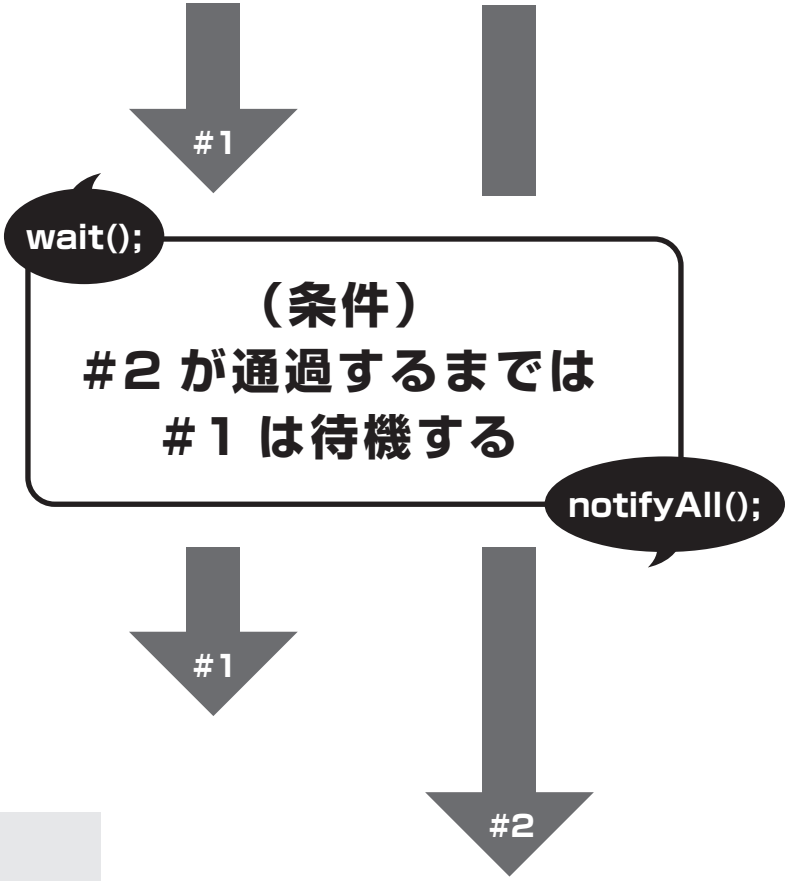


52

Java の覚え書き

スレッドを待ち合わせる

```
synchronized methodName(...) {
    while( 待機条件 ) {
        wait();
    }
    ...
    notifyAll();
}
```



スレッドをある状態で待機させ、別のスレッドから通知が来た時点で処理を再開するといった仕組みは、上記の様な構造を作る事で実現する事ができます。

void wait()	他のスレッドがこのオブジェクトの 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。
void notifyAll()	wait() メソッドによって待機中であるすべてのスレッドを再開します。