Woodland Academy

3.3 Java　補足　スレッド

- スレッド

Shape Your Future

目次

プロセス

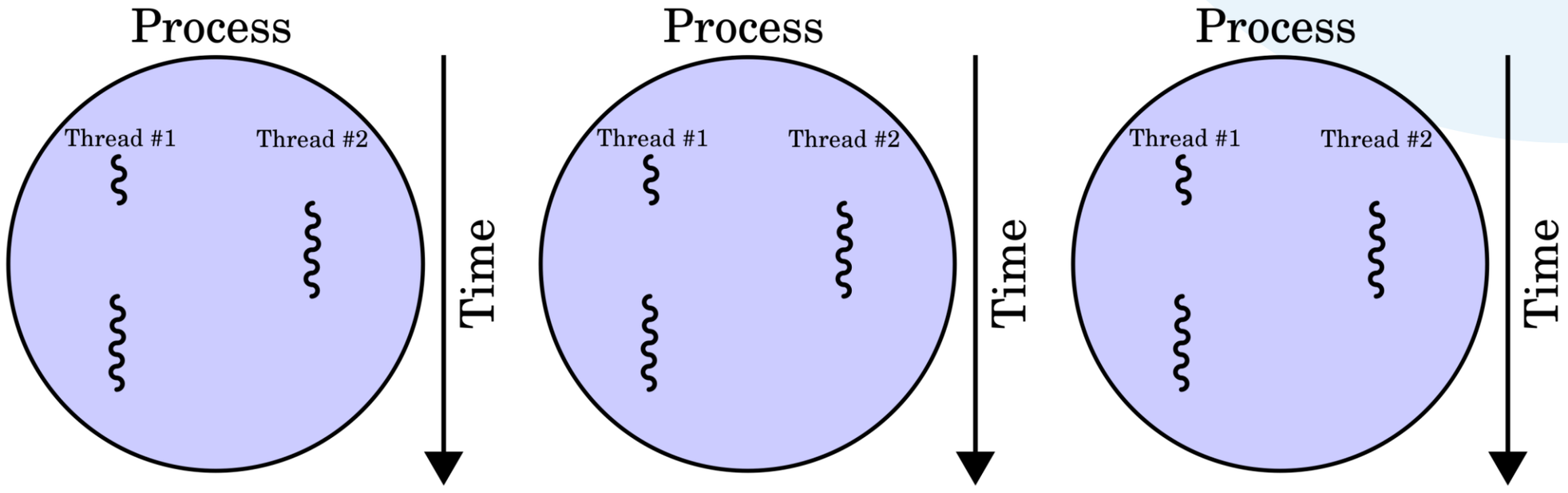
- コンピュータでは、あるプログラムを実行しようとすると、オペレーティングシステムが、そのプログラムに 1 つの**プロセス**[Process]を割り当てます。

- 複数のプロセスを同時に実行することができ、OS が自動的にそれらの実行時間を調整して、同時に実行できるようにしてくれます。

Woodland Academy

スレッド

- しかし、時には同じプログラムの中で複数の作業を同時に行う必要があります。この時は、**スレッド**[Thread]を使うべきです。プロセスと同様に、OS は複数のスレッドを同時に実行できるように、実行時間を自動的に割り当てることができます。

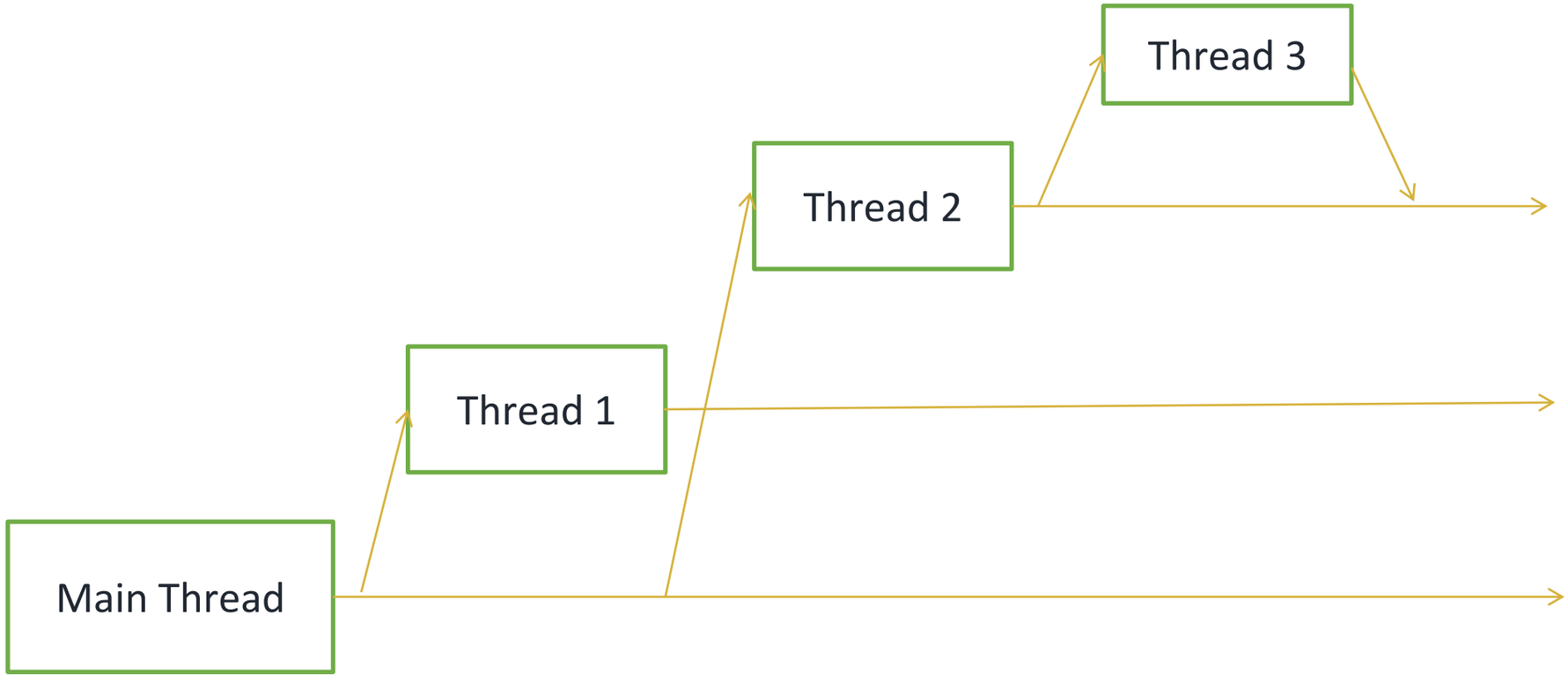


- 複数のスレッドを同時に実行させることは、**マルチスレッド**[Multithread]と言います。マルチスレッドを使うケースを考えましょう。

Java のスレッド

- プログラムが実行され始めると、すべてのコードは**メインスレッド**と呼ばれるデフォルトのスレッドで実行されます。

- メインスレッドと同時にタスクを実行させたい場合は、新しい**スレッドを作成**し、そのスレッドに**タスクを与えて実行させる**ことが必要です。



スレッドの作成

- **Thread** クラスは、Java でスレッドを表すクラスです。
- スレッドを作成するには、Thread のコンストラクタでそのオブジェクトを作成します。ドキュメントを見ると、Thread のコンストラクタは Runnable 型の引数を受け取ります：

```
Thread(Runnable target)                    Allocates a new Thread object.
```

- この **Runnable** は、**run()** というメソッドだけを持つ関数型インタフェースです：

```
1 package java.lang;
2
3 @FunctionalInterface
4 public interface Runnable {
5     public abstract void run();
6 }
```

- つまり、スレッドを生成するためには、Runnable インタフェースを実装したクラスを用意し、それをインスタンス化し、Thread のコンストラクタに渡す必要があるのです。

- ただし、Runnable は**関数型インタフェース**なので、ここは **lambda 式**が使えます（§ 3.2.2）：

```
1 Thread thread = new Thread(() -> {
2         System.out.println("Do something 1");
3         System.out.println("Do something 2");
4         System.out.println("Do something 3");
5 });
```

スレッドの実行

- 今のところ、スレッドは作成できましたが、まだ実行されていません。スレッドを実行するには、それの **start()** メソッドを呼び出す必要があります：

```
1 Thread thread = new Thread(() -> {
2         System.out.println("Do something 1");
3         System.out.println("Do something 2");
4         System.out.println("Do something 3");
5 });
6 thread.start(); // Do something 1, 2, 3
```

複数のスレッドの実行

- スレッドを使用する主な理由は**複数のタスクの同時実行**です。異なるスレッドを作成して、それらが同時に実行されるかどうかを確認しましょう。

Try

MultiThread.java

Note

スレッドの実行順番は OS に決められて、
start() の順番と**必ずしも同じわけではない。**

スレッドのメソッド

- start() メソッド以外にも、スレッドの動作を制御するための様々なメソッドが提供されています。よく使われるのは：

メソッド	機能
start()	スレッドを実行
setPriority()	スレッドの優先順位を設定
join()	スレッドが終わるのを待つ
interrupt()	スレッドを割り込み
isAlive()	スレッドが実行中かどうかを判断
sleep()	現在のスレッドを一定時間で中断する

Woodland Academy

並行処理の問題

- 実際のスレッドの順番は操作システムによって自動的に決定されるため、複数のスレッドを同時に実行すること、いわゆる**並行処理**[Concurrent Processing]が多くあります。並行処理は、予期せぬ多くの問題を引き起こす可能性があります。

- 例えば、同じ変数の値を異なるスレッドで変更しようとすると、期待するような結果が得られない恐れがあります。

Try
01011
11010
01011
Parallel.java

- なぜそのような結果が出るのかを考えましょう。

不可分性

- スレッドが期待通りに実行されない理由の大部は、コードが**不可分性**[Atomicity]（**アトミック性**、**原子性**ともいう）を満たしていないためです。不可分性とは、あるコードのブロックが常に連続的に実行され、他のスレッドより介入できないことを意味します。

- 例えば、先ほどの例では、変数がインクリメントされるたびに他の変数に影響があるコードは実行されないようにしたいです。

synchronized キーワード

- Java では、**synchronized** キーワードを使って、あるコードが同時に 1 つのスレッドしか実行できないように制御することができます。

- synchronized で修飾された同一オブジェクトのメソッドは、同時に 1 スレッドのみが実行可能です。

Try 01011 11010 01011

Parallel2.java

Woodland Academy

synchronized の他の使用法

- synchronized キーワードはメソッドを修飾する以外にも使い方がありますが、ここでは概要だけ紹介しましょう。
  - **コードブロック**の修飾：

```
1 synchronized(obj) {
2     // codes
3 }
```

　　このコードブロックを「ロック」するには、オブジェクトを使用します。 同じオブジェクトにロックされたメソッドは、同時に 1 つしか実行できません。最も一般的な使用方法は、「**this**」または「**クラス名.class**」をロックとして使います。

  - **クラスメソッド**も修飾できます。そのクラスメソッドは一度に 1 つのスレッドしか実行できません。

Woodland Academy

Q&A

まとめ

Sum Up

1.スレッドの基本概念とその使い方。

Woodland Academy

Woodland Academy

Thank you!

From Seeds to Woodland — Shape Your Future.

Shape Your Future