8주차: 멀티쓰레드 프로그래밍

https://github.com/KMUCS-MoGakKo/Java_Study/issues/17

멀티쓰레드... 너무나도 컴퓨터 구조와 운영체제가 생각난다...

- **Thread 클래스와 Runnable 인터페이스**

오라클 공식 문서에서 Thread 클래스와 Runnable 인터페이스를 살펴보았다.

Module java.base
Package java.lang

Class Thread

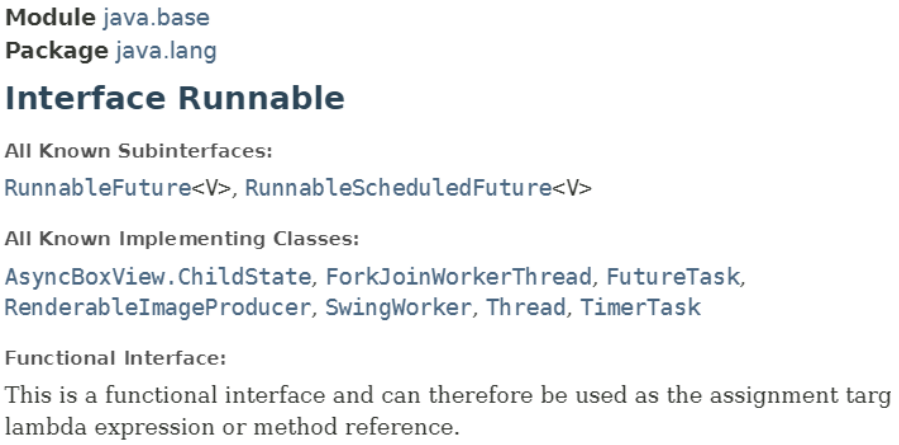
java.lang.Object
  java.lang.Thread

All Implemented Interfaces:

Runnable

Direct Known Subclasses:

ForkJoinWorkerThread



Module java.base
Package java.lang

Interface Runnable

All Known Subinterfaces:

RunnableFuture<V>, RunnableScheduledFuture<V>

All Known Implementing Classes:

AsyncBoxView.ChildState, ForkJoinWorkerThread, FutureTask, RenderableImageProducer, SwingWorker, Thread, TimerTask

Functional Interface:

This is a functional interface and can therefore be used as the assignment targ lambda expression or method reference.

Thread 클래스 문서를 읽어보면 다음과 같다.

1. Thread 는 생성자와 Thread.Builder 로 쓰레드를 생성할 수 있고, 이들은 Runnable task 를 실행해준다.
2. Thread 는 고유 식별자와 이름이 있다. 식별자는 쓰레드가 생성될 때 함께 생성되고 변경 불가능하다. 이름은 바꿀 수 있다.

Thread.Builder 가 뭔가 하고 알아봤더니,

Module java.base
Package java.lang

Interface Thread.Builder

All Known Subinterfaces:

`Thread.Builder.OfPlatform`[PREVIEW], `Thread.Builder.OfVirtual`[PREVIEW]

Enclosing class:

`Thread`

음. 이것만 봐선 뭔지 모르겠다.

> **`Builder` is a preview API of the Java platform.**
> *Programs can only use `Builder` when preview features are enabled.*
> *Preview features may be removed in a future release, or upgraded to permanent features of the Java platform.*

A builder for `Thread` and `ThreadFactory` objects.

`Builder` defines methods to set `Thread` properties such as the thread `name`. This includes properties that would otherwise be inherited. Once set, a `Thread` or `ThreadFactory` is created with the following methods:

- The unstarted method creates a new *unstarted* `Thread` to run a task. The `Thread`'s `start` method must be invoked to schedule the thread to execute.
- The start method creates a new `Thread` to run a task and schedules the thread to execute.
- The factory method creates a `ThreadFactory`.

A `Thread.Builder` is not thread safe. The `ThreadFactory` returned by the builder's `factory()` method is thread safe.

Unless otherwise specified, passing a null argument to a method in this interface causes a `NullPointerException` to be thrown.

새로운 친구가 나왔다. Thread.Builder 를 검색했더니 **빌더 패턴**이란걸 알려준다.

크게 중요한 내용은 아닌 듯 하니 관련 링크만 첨부하고 넘어가도록 하자!

결론만 말하자면, 생성할 때 디자인과 생성의 표현을 분리하고자 하는 의도에서 나온 것이라고 한다.

+ 빌더 패턴이란 : https://lemontia.tistory.com/483
+ 빌더 패턴을 사용해야하는 이유 : https://mangkyu.tistory.com/163

Creating and starting threads

`Thread` defines public constructors for creating platform threads and the `start` method to schedule threads to execute. `Thread` may be extended for customization and other advanced reasons although most applications should have little need to do this.

`Thread` defines a `Thread.Builder`[PREVIEW] API for creating and starting both platform and virtual threads. The following are examples that use the builder:

다음은 쓰레드 생성에 관한 항목이다.

쓰레드는 2가지 방법으로 생성할 수 있다.

1. 직접 Thread 클래스를 상속받아 생성

```
class CustomThread extends Thread {
	@Override
	public void run() { ... }
}
```

Thread 클래스를 상속 받아, `run` 메소드를 오버라이딩하여 쓰레드를 생성한다. 쓰레드를 실행하면 `run` 메소드를 호출하기 때문에 `run` 메소드를 오버라이딩하여 커스텀하면 된다.

2. Runnable 인터페이스를 구현하여 생성

두 번째 방법은 Runnable 인터페이스를 구현한 클래스를 Thread 생성자에 인자로 전달하는 방법이다.

```
class CustomRunnable implements Runnable {
	@Override
	public void run() { ... }
}
```

Custom한 클래스를 생성했다면 실행하는 방법은 아래와 같다.

```
public class TestThread {
	public static void main(String[] args) {

		// 1. Thread 클래스 상속 받아 구현
		Thread t1 = new CustomThread();

		// 2. Runnable 인터페이스
		Thread t2 = new Thread(new CustomRunnable());

		t1.start();
		t2.start();
	}
}
```

1번의 경우, **CustomThread 클래스**를 객체로 만들고, 2번은 Thread 객체를 만들고 인자로 **CustomRunnable 클래스**를 전달하면 된다. 수행은 둘 다 `start` 메소드로 할 수 있다.
(이 부분은 운영체제에서 배웠던 기억이 난다. `run()` 만으로는 쓰레드를 생성할 수 없어 `start()` 가 있어야만 쓰레드가 생성되고, `run` 메소드가 실행이 된다.)

Thread 와 Runnable 을 비교한 것을 표로 정리하면 아래와 같다.

	Runnable	Thread
람다 가능	O	X
상속 필요	X	O
자원 사용량	적음	많음

이미지 출처: https://mangkyu.tistory.com/258

딱 보기에도 Runnable 을 상속 받는 것이 여러모로 효율적으로 보인다. 실제로 Runnable 을 상속 받아 실행하는 것을 추천하는 편이다. Thread 관련 기능의 확장이 필요한 경우에는 Thread 클래스를 상속받아 구현해야 할 때도 있다고는 하지만, 대부분 Runnable 인터페이스를 사용하면 해결 가능하다.

다음은 쓰레드의 메서들에 대한 설명이다. 주요 메서드 3개만 보고 넘어가자.

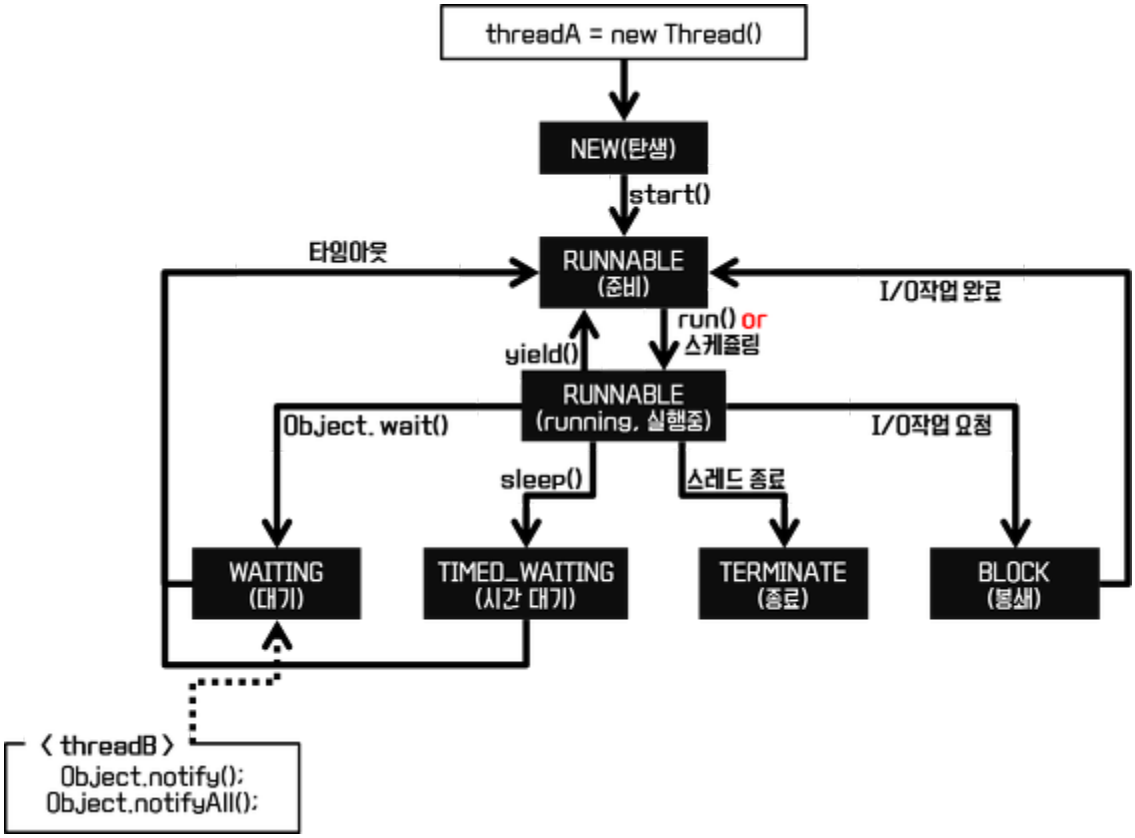
메소드명	설명
sleep	현재 쓰레드를 멈춘다. 자원을 놓아주지는 않고, 제어권을 넘겨주므로 데드락이 발생할 수 있다. sleep() 을 사용할 땐 항상 try-catch 로묶어줘야 한다.
interrupt	현재 수행중인 쓰레드를 InterruptedException 예외을 발생시켜 중단시킨다. Interrupt가 발생한 쓰레드는 예외를 catch하여 다른 작업을 할 수 있다.
join	다른 쓰레드의 작업이 끝날 때 까지 기다리게 한다. 쓰레드의 순서를 제어할 때 사용할 수 있다.

- 쓰레드의 상태

https://raccoonjy.tistory.com/15
- 스레드의 상태 6가지

NEW	스레드가 생성되었지만 스레드가 아직 실행할 준비가 되지 않았음
RUNNABLE	스레드가 실행되고 있거나 실행준비되어 스케쥴링은 기달리는 상태
WAITING	다른 스레드가 notify(), notifyAll()을 불러주기 기다리고 있는 상태(동기화)
TIMED_WAITING	스레드가 sleep(n) 호출로 인해 n 밀리초동안 잠을 자고 있는 상태
BLOCK	스레드가 I/O 작업을 요청하면 자동으로 스레드를 BLOCK 상태로 만든다.
TERMINATED	스레드가 종료한 상태

- 스레드 상태는 JVM에 의해 기록 관리된다.



참고 링크

- https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Thread.html
- https://mangkyu.tistory.com/163
- https://huzz.tistory.com/3