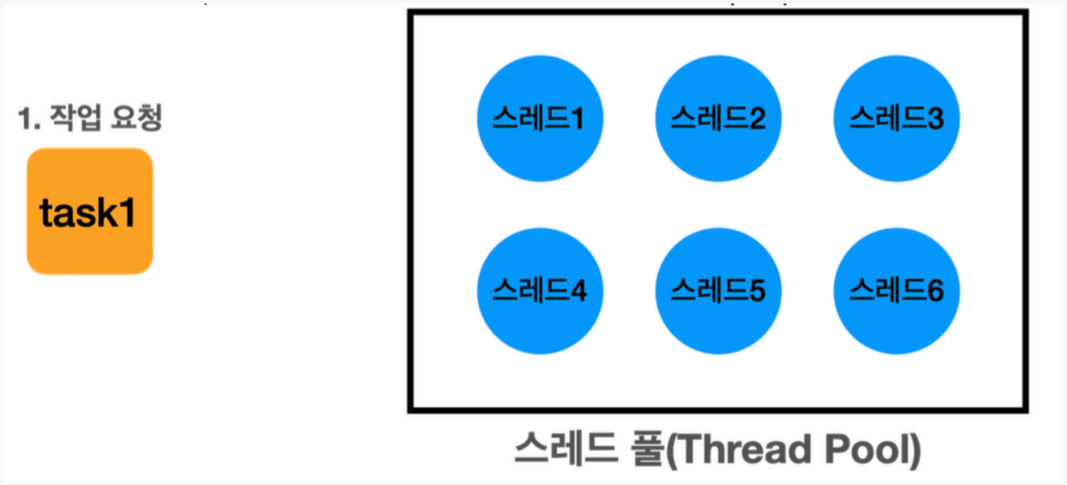
5주차 스터디

목차

스레드를 직접 사용할 때의 문제점

- 스레드 생성 시간으로 인한 성능 문제
  - 메모리 할당
  - 운영체제 자원 사용
  - 운영체제 스케줄러 설정
- 스레드 관리 문제
  - 스레드 개수 관리
  - 스레드 종류 문제
- Runnable 인터페이스의 불편함
  - 반환 값이 없다
  - 예외 처리

스레드 풀



스레드 풀

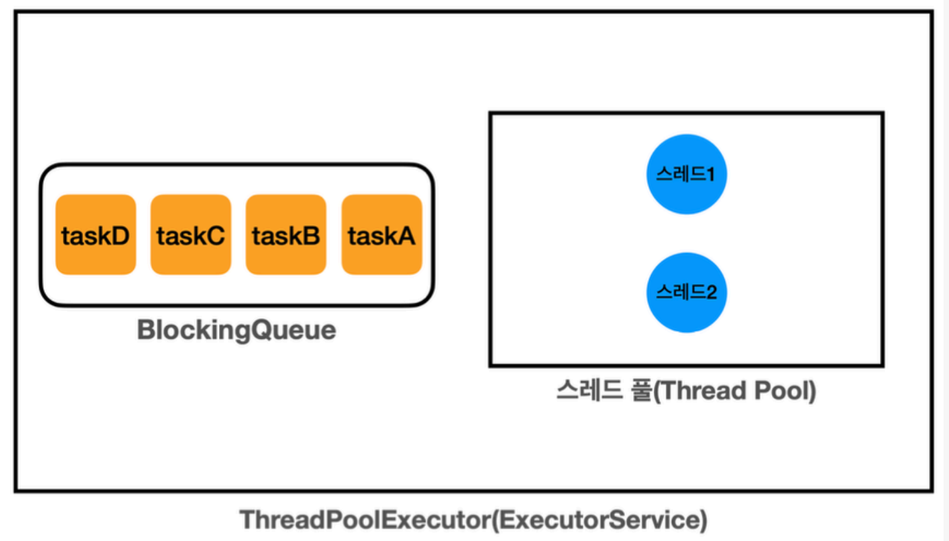
- 스레드를 관리하는 스레드 풀
- 스레드는 스레드 풀에서 대기하며 쉰다.
- 필요한 만큼만 스레드를 만들 수 있고, 또 관리할 수 있다.
- 스레드를 재사용할 수 있어서, 재사용시 스레드의 생성 시간을 절약할 수 있다.

Executor

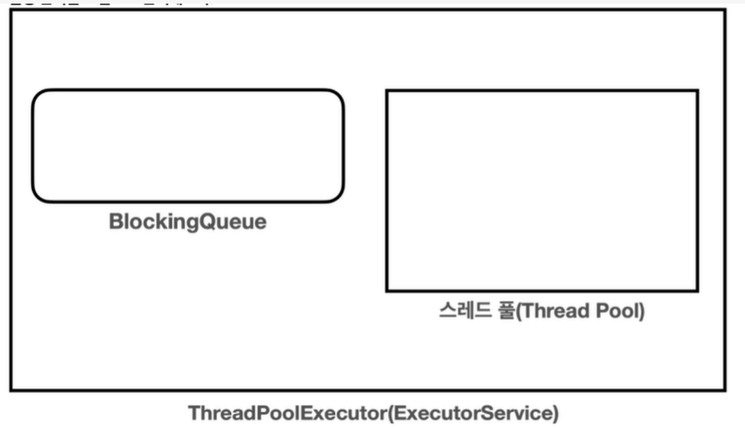
```
public class ExecutorBasicMain {

    public static void main(String[] args) {
        ExecutorService es = new ThreadPoolExecutor(2, 2, 0,
TimeUnit.MILLISECONDS, new LinkedBlockingQueue<>());
        log("== 초기 상태 ==");
        printState(es);
        es.execute(new RunnableTask("taskA"));
        es.execute(new RunnableTask("taskB"));
        es.execute(new RunnableTask("taskC"));
        es.execute(new RunnableTask("taskD"));
        log("== 작업 수행 중 ==");
        printState(es);

        sleep(3000);
        log("== 작업 수행 완료 ==");
        printState(es);

        es.close();
        log("== showdown 완료 ==");
        printState(es);
    }
}
```

ThreadPoolExecutor 생성자

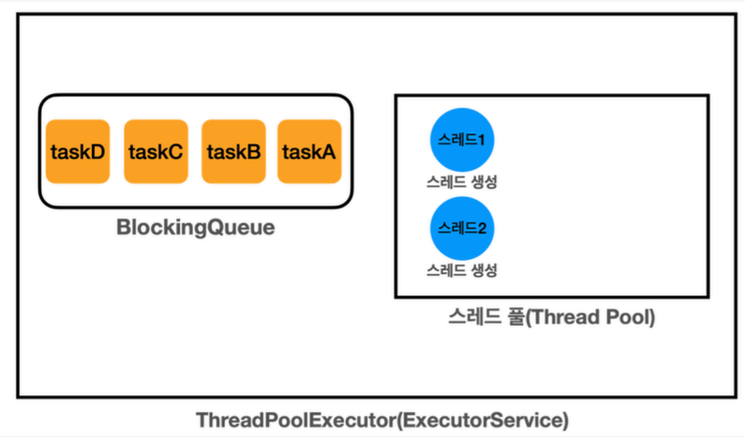
- corePoolSize: 스레드 풀에서 관리되는 기본 스레드의 수
- maximumPoolSize: 스레드 풀에서 관리되는 최대 스레드 수
- keepAliveTime, TimeUnit unit: 기본 스레드 수를 초과해서 만들어진 스레드가 생존할 수 있는 대기 시간
- BlockingQueue workQueue: 작업을 보관할 블로킹 큐



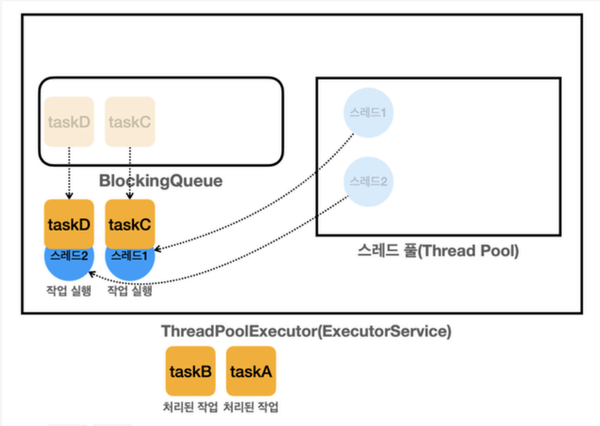
Executor



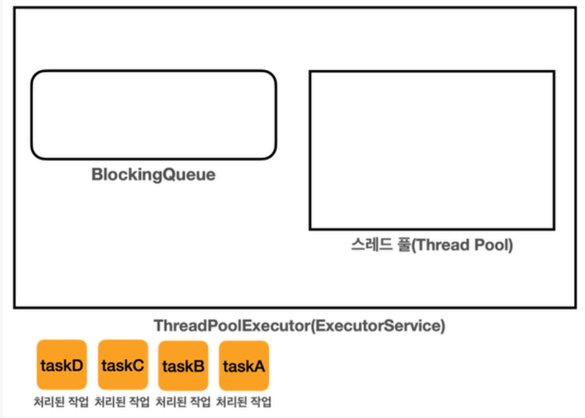
- ThreadPoolExecutor를 생성한 시점에 스레드 풀에 스레드를 미리 만들어두지는 않는다.



- es.execute(작업)를 호출하면 내부에서 BlockingQueue에 작업을 보관한다.
- corePoolSize에 지정한 수 만큼 스레드를 스레드 풀에 만든다.

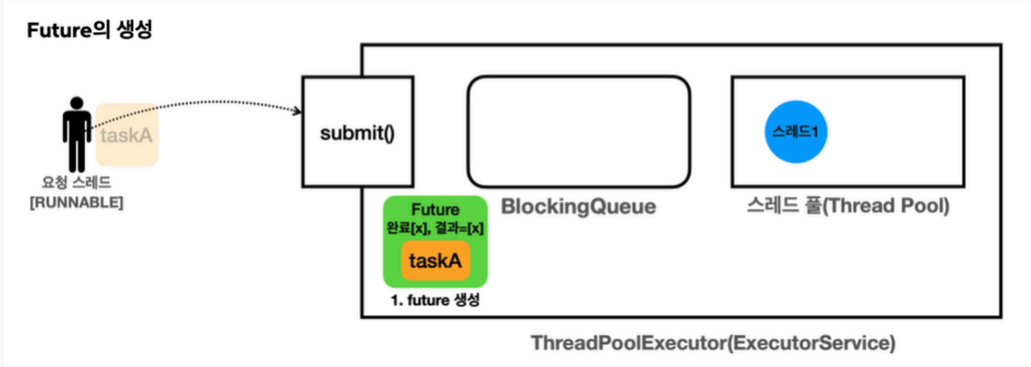


- 큐에 대기중인 작업이 있으면 스레드 풀에 있는 스레드가 해당 작업을 수행한다.
- 작업이 완료되면 스레드 풀에 스레드를 반납하고 스레드는 대기(WAITING) 상태로 스레드 풀에 대기한다.

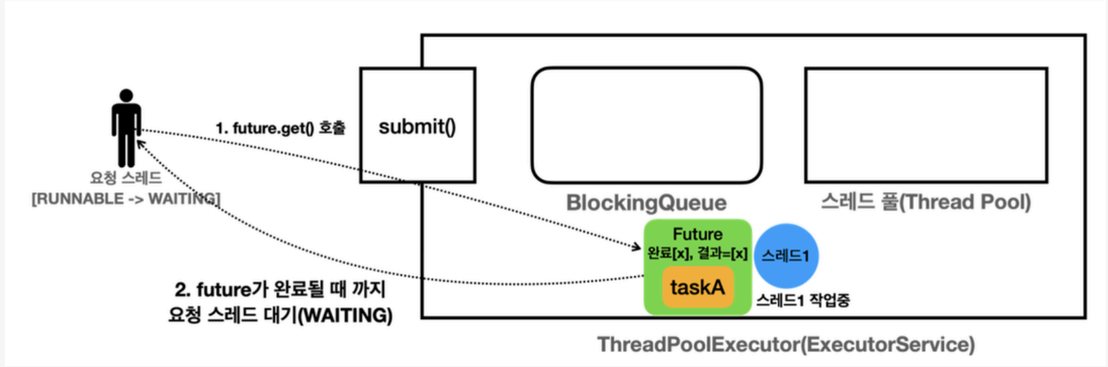


- close()를 호출하면 ThreadPoolExecutor가 종료된다. 이때 스레드 풀에 대기하는 스레드도 함께 제거된다.

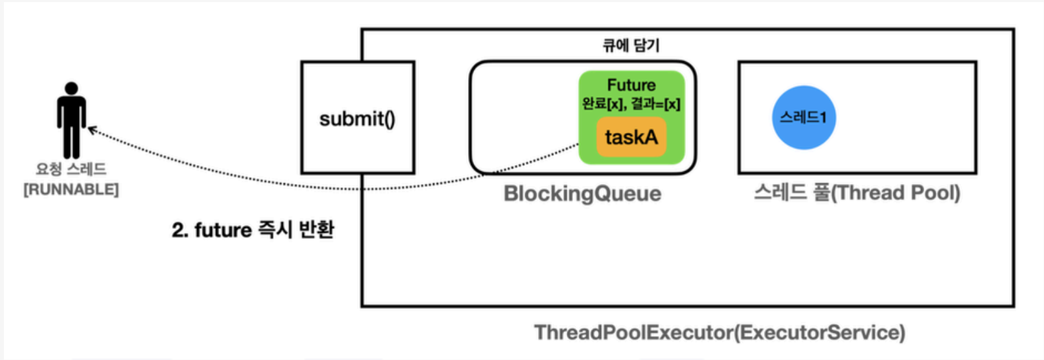
Future 실행 흐름



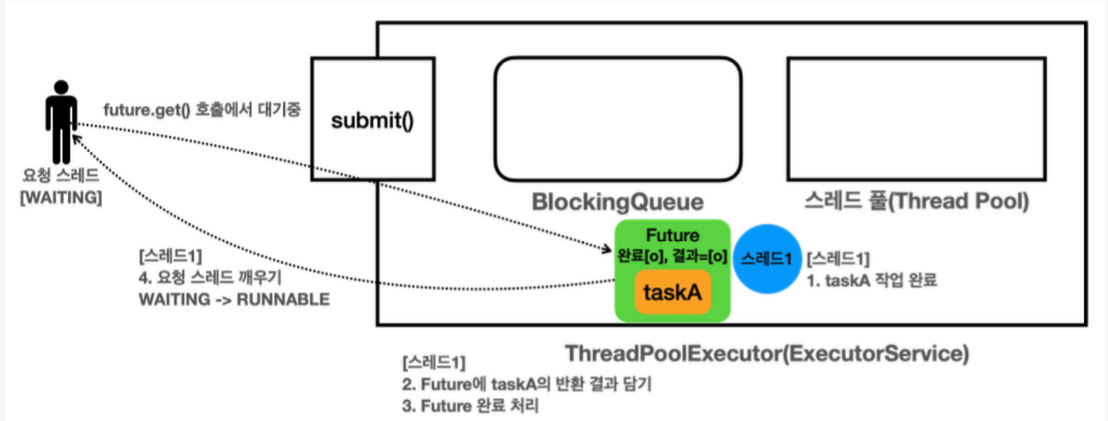
- es.submit(new MyCallable()) 호출



- 요청 스레드가 future.get()을 호출하면 Future가 완료 상태가 될 때 까지 대기
- 요청 스레드의 상태는 RUNNABLE -> WAITING

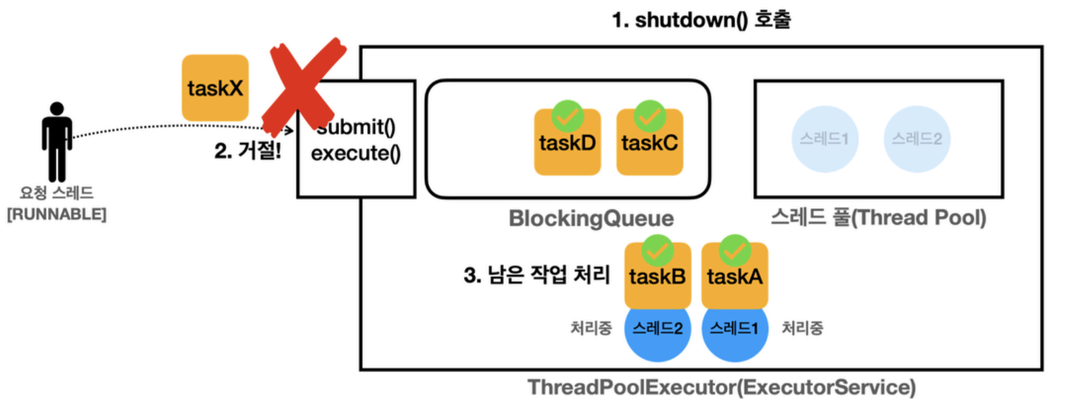


- 생성된 Future는 블로킹 큐에 저장
- 저장과 동시에 execute()를 호출한 요청 스레드에 Future를 즉시 반환

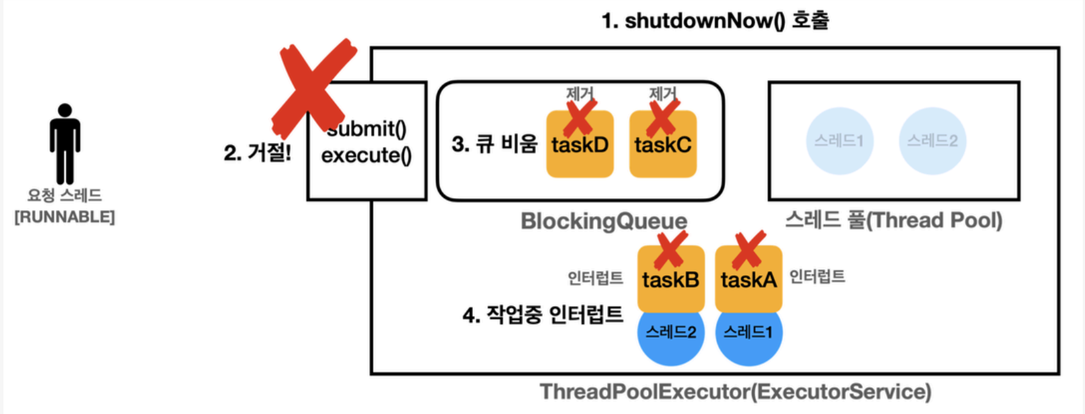


- 스레드1이 작업을 완료하면 future에 결과값을 담고 상태를 완료로 변경
- 스레드1이 대기하고 있던 요청 스레드를 깨우고 요청 스레드는 다시 작업을 진행

ExecutorService의 종료 메서드



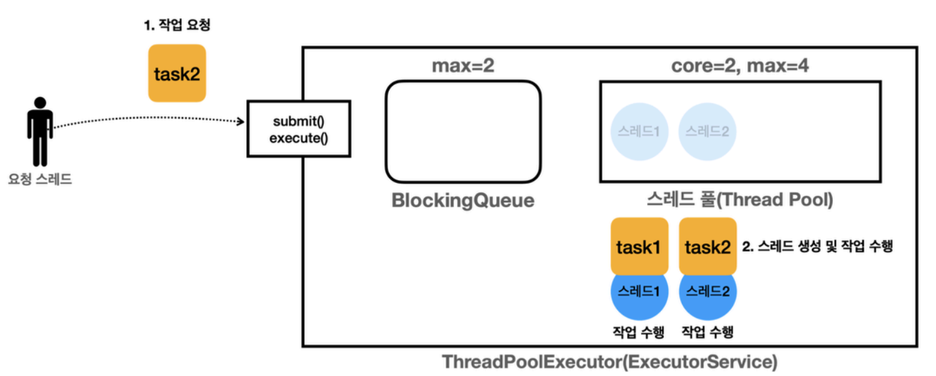
- ExecutorService는 새로운 요청을 거절
- 스레드 풀의 스레드는 처리중인 작업을 완료하고 큐에 남아있는 작업도 모두 꺼내서 완료



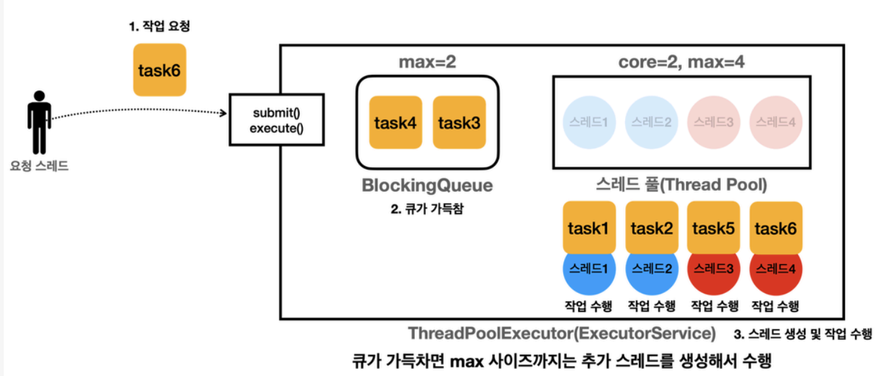
- ExecutorService는 새로운 요청을 거절
- 큐를 비우고 작업 중인 스레드에 인터럽트를 보냄

ExecutorService 스레드 풀 관리

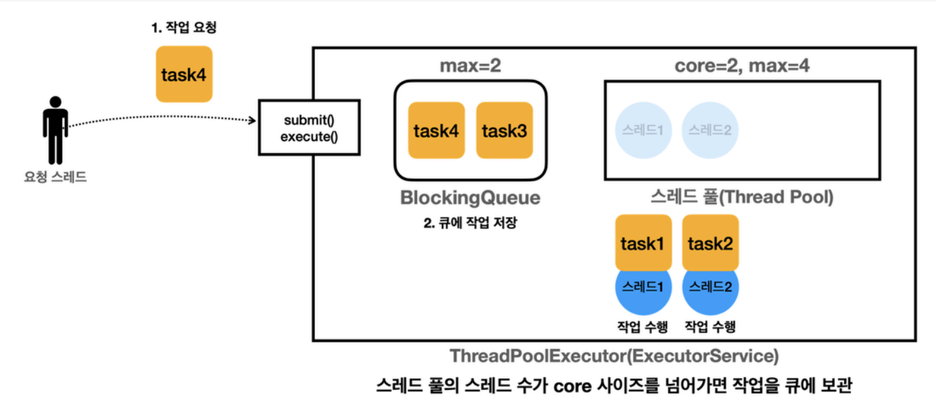
task1, task2 요청



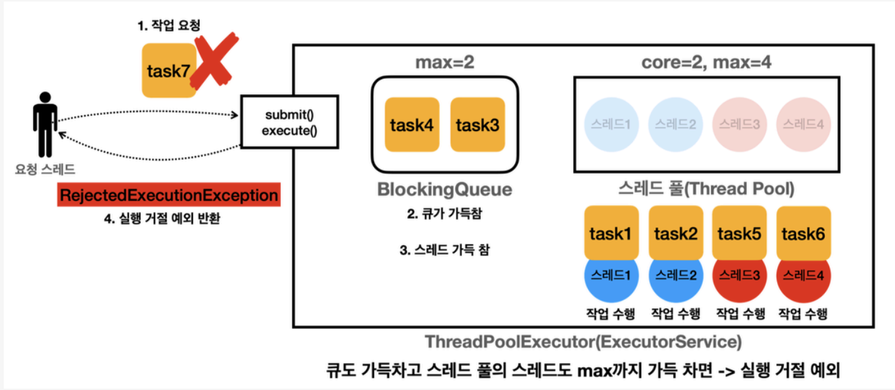
task5, task6 요청



task3, task4 요청



task7 요청



ExecutorService 스레드 풀 관리 전략

```
new ThreadPoolExecutor(1, 1,0L, TimeUnit.MILLISECONDS,
                       new LinkedBlockingQueue<Runnable>())
```

newSingleThreadPool(): 단일 스레드 풀 전략

- 기본적으로 하나의 스레드만을 사용하는 스레드 풀
- 작업 큐에 제한이 없으며 LinkedBlockingQueue를 사용
- 주로 간단한 작업이나 테스트 용도로 사용

```
new ThreadPoolExecutor(nThreads, nThreads, 0L, TimeUnit.MILLISECONDS,
                       new LinkedBlockingQueue<Runnable>())
```

newFixedThreadPool(nThreads): 고정 스레드 풀 전략

- nThreads만큼의 스레드를 고정으로 생성하여 사용하는 전략이다. 초과 스레드는 생성하지 않는다.
- 작업 큐에 제한이 없으며, LinkedBlockingQueue를 사용
- 스레드 수가 고정되어 있어 CPU 및 메모리 리소스가 예측 가능한 안정적인 방식

```
new ThreadPoolExecutor(0, Integer.MAX_VALUE, 60L, TimeUnit.SECONDS,
                       new SynchronousQueue<Runnable>());
```

newCachedThreadPool(): 캐시 스레드 풀 전략

- 기본 스레드를 사용하지 않고, 생존 주기를 가진 초과 스레드만 사용
- 큐에 작업을 저장하지 않는다. (SynchronousQueue)
- 대신에 생산자의 요청을 스레드 풀의 소비자 스레드가 직접 받아서 바로 처리한다.

감사합니다.