자바의 **멀티스레드**

가. **멀티스레드**란 **하나의 프로그램내에서 여러 개의 작은 작업이 동시에 수행되는 것을 말한다.**

나. 멀티스레드 구현 방법

첫째, Thread 클래스 상속받는법.

Thread 클래스를 상속받아 멀티스레드를 구현하면 방법은 쉽지만, **단일 상속만 가능하다는 단점이 있다.**

둘째, **Runnable 인터페이스를 상속받는 법**

다중 상속을 받을 수 있다는 장점이 있다. 하지만 **멀티스레드를 시작하는 start() 메서드가 없기 때문에 이 인터페이스를 상속받은 자식클래스 객체를 Thread 클래스 생성자 인자값으로 넘겨줘서 다시 한번더 객체를 생성해서 start()메서드를 호출해야 한다는 단점이 있다.**

다. **멀티스레드에서 start() 메서드를 호출하면 멀티스레드가 시작되면서 run() 메서드를 자동 호출**한다. 명시적인 run()메서드를 호출하지 않아도 start() 메서드만 호출하면 run() 메서드는 자동 호출하도록 자바는 설계 되어져 있다.

바로 자동 호출되는 **run()메서드 내에 멀티스레드 문장을 구현**한다.

쓰레드 스케줄링 메서드

1. **sleep(), suspend(), wait(),join() : 쓰레드를 일시정지** 시킨다.

2. **resume(), notify(), interrupt() : 일시정지 상태를 벗어나 다시 실행대기 상태**로 만든다.

3. **yield() : 실행중에 자신에게 주어진 실행시간을 다른 쓰레드에게 양보하고 자신은 실행대기 상태**가 된다.

4. **stop() : 쓰레드를 즉시 종료** 시킨다.

5. **sleep(long millis) ->밀리세컨드 1000분의 1초 단위. 일정시간동안 쓰레드를 멈추게 한다.**

6. **sleep()에 의해서 일시정지 상태가 된 쓰레드는 지정된 시간이 다 되거나 interrupt() 가 호출되면 InterruptedException 예외가 발생해 다시 실행대기 상태**가 된다.

7. **suspend()에 의해서 일시 정지된 쓰레드는 resume()을 호출해야 다시 실행대기 상태**가 된다.

8. **suspend(), resume(), stop()** 은 자바 API를 보시면 전에는 사용되었지만 **앞으로 사용하지 않을 것을 권장**한다는 뜻의 **@Deprecated 애너테이션으로 정의** 되어 있다. 이 3개의 메서드는 쓰레드를 교착상태(dead-lock)로 만들기 쉽게 한다. **교착상태란 두 쓰레드가 자원을 점유한 상태에서 서로 상대편이 점유한 자원을 사용하려고 기다리느라 진행이 멈춰있는 상태를 말한다.**

쓰레드의 동기화

1. **동기화**란 **한 쓰레드가 진행 중인 작업을 다른 쓰레드가 간섭하지 못하도록 막는 것**을 말한다.

2. **하나의 쓰레드에 의해서만 처리할 수 있도록 하는 영역을 임계영역**이라 한다.

3. **임계영역을 지정**하기 위해서는 **하나의 쓰레드가 이 영역에 진입할 때 락을 걸어서 다른 쓰레드가 수행되지 못하도록 하고, 이 영역에서 벗어날 경우 락을 풀어서 다른 쓰레드가 수행하도록 한다**. 즉 **임계영역 내에서는 한 순간에 하나의 쓰레드만 동작하도록 제약**을 주어야 한다. 이러한 **제약을 위해서는 자바에서는 동기화 기법**을 제공하는데 **하나의 쓰레드만 동작하도록 하고자 하는 메서드나 블록에 synchronized 로 지정**한다.

4. 동기화 기법
  가. 메서드 전체를 동기화 처리
```
public synchronized void 메서드명(){
 //임계영역
}
```

  나. 특정한 영역을 임계 영역으로 지정
```
synchronized(객체의 참조변수){
  //임계영역
}
```

5. **동기화 영역내에서 한 쓰레드가 락을 보유하고 계속해서 작업을 진행할 상황이 아니면 일단 락을 풀고 기다리게 하는 경우가 발생**한다. 예를 들면 **출금계좌에 잔고가 부족한대 한 쓰레드가 락을 가진 상태로 돈이 입금될 때까지 무한정 기다리면 다른 쓰레드는 모두 해당객체의 락을 기다리느라 아무런 작업을 못하는 경우가 발생**한다.
이럴 경우 일단 wait()를 호출하여 락을 반납하고 기다리게 한다. 그러면 **다른 쓰레드가 락을 얻어 해당 객체에 대해서 작업을 수행**할 수 있게 된다.
나중에 **다시 작업할 상황이 되면 notify()를 호출해서 중단했던 쓰레드가 다시 락을 얻어 작업을 진행**할 수 있게 한다.

6. **notify()를 호출했다면 wait()에 의해서 락을 반납하고 대기실에서 대기 중인 쓰레드 중에서 하나를 임의로 선택해서 통지할 뿐, 락을 반납한 해당 쓰레드를 선택해서 통지 할수 없다**. 운 좋게 락을 반납한 해당 스레드가 통지를 받으면 다행인대, **그렇지 않**

으면 계속 통지를 받지 못하고 무한정 기다리게 되는 현상이 발생한다. 이런 현상을 '**기아 현상**' 이라 한다. 이 현상을 방지하기 위해 **notify() 대신 notifyAll()을 사용**해야 한다. 이 메서드는 대기실에 대기중인 모든 쓰레드에게 통지를 하기 때문에 기아 현상은 막았지만,통지 받은 쓰레드가 서로 락을 얻기 위해 경쟁을 하게 된다. 이처럼 **여러 쓰레드가 서로 락을 얻기 위해 경쟁**하는 것을 '**경쟁 상태**' 라 한다.
이 경쟁 상태를 개선하기 위해서 각 쓰레드를 구분해서 통지하는 것이 필요하다.
Lock과 컨디션을 이용하면 부분적인 선별적 통지가 가능하다. 컨디션을 사용하면 wait() 대신 await()를 사용하면 되고, notify() 대신 signal()을 이용한다.

데몬 스(쓰)레드

1. 데몬(daemon) 스레드는 주 스레드의 작업을 돕는 보조적인 역할을 수행하는 스레드이다. 주 스레드가 종료되면 데몬 스레드는 강제적으로 자동 종료된다.

2. 데몬 스레드 적용 예)
  가. 워드 프로세서 자동 저장
  나. 미디어 플레이어의 동영상 및 음악 재생 등

3. 스레드의 데몬을 만들기 위해서는 주 스레드가 데몬이 될 스레드의 setDaemon(true)를 호출해야 한다. 예외 오류를 발생하기 않기 위해서는 start() 메서드 호출 전에 이 메서드를 호출해야 한다.

4. 현재 실행중인 스레드가 데몬 스레드 인지 아닌지를 판별하는 방법은 isDaemon() 메서드를 호출해 반환값이 true이면 데몬 스레드이다.

다음 예제를 보면 메인 스레드가 주 스레드이고, AutoSaveThread는 데몬 스레드이다. 1초 간격으로 save()메서드를 호출해서 데몬 스레드 AutoSaveThread를 실행시킨다. 그리고 메인 스레드가 3초 후 종료되면 AutoSaveThread도 따라서 종료된다.

```
package 데몬_스레드풀_21가상스레드;

public class AutoSaveThread extends Thread {

        public void save() {
          System.out.println("작업 내용을 저장함.");
        }

        @Override
        public void run() {
                while(true) {
                        try {
                                Thread.sleep(1000);//1초 간격으로 save()메서드를
호출해서 AutoSaveThread 데몬 스레드를 실행
```

```
                        }catch(InterruptedException ie) {
                                break;//무한루프 문 종료
                        }
                        save();
                }//무한루프문
        }//run()
}//AutoSaveThread class

package 데몬_스레드풀_21가상스레드;

public class DaemonExample {//주 스레드가 main(), 3초후 메인스레드가 종료되면 데몬스레드인 AutoSaveThread도 따라서 자동 종료
        public static void main(String[] args) {
      AutoSaveThread autoSaveThread = new AutoSaveThread();
      autoSaveThread.setDaemon(true);//데몬 스레드 지정, start()메서드를 호출하기 전 먼저 기술실행해야 예외에러가 안난다.
      autoSaveThread.start();//스레드 시작

      try {
          Thread.sleep(3000);//3초뒤에 메인 주 스레드 종료
      }catch(InterruptedException e) {}

      System.out.println("메인 스레드 종료");
        }
}
```

스레드 풀이란?

1. 스레드 개수가 많아지면 cpu가 바빠지고 메모리 사용량도 늘어난다. 그러면 해당 스레드 프로그램 성능도 떨어진다. 이러한 스레드 폭증을 막을려면 스레드 풀을 사용해야 한다.

2. 스레드 풀은 작업 처리에 사용되는 스레드를 제한된 개수만큼 정해놓고 작업 큐에 들어오는 작업들을 하나씩 스레드가 맡아서 처리한다. 그렇기 때문에 작업 처리 요청이 폭증 되어도 스레드의 전체 개수가 늘어나지 않기 때문에 성능은 저하 되지 않는다.

3. 스레드 풀 생성

3-1. 자바는 스레드 풀을 생성하고 사용할 수 있도록 java.util.concurrent 패키지에서 ExecutorService 인터페이스와 Executors 클래스 API를 제공한다. Executors의 다음 두 정적 메서드를 이용하면 간단하게 스레드 풀인 ExecutorService 구현 객체를 만들 수 있다.

메서드명(매개변수)	초기수	코어수	최대수
newCachedThreadPool()	0	0	Integer.MAX_VALUE
newFixedThreadPool(int nThreads)	0	생성된수	nThreads

초기수는 스레드풀이 생성될 때 기본적으로 생성되는 스레드 수를 의미하고, 코어수는 스레드가 증가된 후 사용되지 않는 스레드를 제거할 때 최소한 풀에서 유지하는 스레드 수를 말한다. 최대수는 증가된 스레드의 한도 수이다.
Integer.MAX_VALUE는 Java언어에서 사용되는 상수로, int 타입이 가질 수 있는 최대 정수 값을 나타낸다. 이 값은 2,147,483,647이다.

3-2. 스레드 풀 종료

스레드 풀의 스레드는 기본으로 데몬 스레드가 아니기 때문에 main 주스레드가 종료되더라도 작업을 처리하기 위해서 계속 실행상태로 남아 있다. 스레드 풀의 모든 스레드를 종료하려면 ExecutorService 의 다음 두 메서드를 중 하나를 실행해야 한다.

리턴 타입	메서드명(매개변수)	설명
void	shutdown()	현재 처리중인 작업뿐만 아니라 작업큐에 대기하고 있는 모든 작업을 처리한 뒤에 스레풀을종료

List<Runnable> shutdownNow() : 현재 작업 처리중인 스레드를 interrupt해서 작업을 중지시키고 스레드 풀을 종료시킨다. 리턴값은 작업 큐에 있는 미처리된 작업(Runnable) 목록이다.

자바 21부터 사용가능한 가상 스레드

가상 스레드는 처리량이 높은 동시 애플리케이션을 개발할 때 사용할 수 있는 경량 스레드이다.
가상 스레드는 CPU를 효율적으로 이용하면서 동시 처리량을 확장할 수 있다. 기존 스레드는 운영체제가 제공하는 플랫폼 스레드와 1:1로 매핑된다면 가상 스레드는 플랫폼 스레드와 N:1로 매핑한다. 다수의 가상 스레드가 한 개의 플랫폼 스레드와 매핑되기 때문에 플랫폼 스레드의 부족 문제를 해결할 수 있다.
가상 스레드는 cpu에서 계산을 수행하는 동안만 플랫폼 스레드를 사용한다. 가상 스레드는 블로킹 I/O 입출력이나 네트워킹을 수행하는 경우 가상 스레드는 일시 중지되지만, 플랫폼 스레드는 중지되지 않고, 다른 가상 스레드의 작업을 처리한다. 그렇기 때문에 cpu 활용도는 최적으로 끌어올리면서 높은 동시 처리량을 달성할 수 있다.

1. 가상 스레드풀 생성

가상 스레드는 메모리가 부족하지 않다면 개수를 무제한으로 사용할 수 있기 때문에 스레드풀처럼 최대 개수를 제한할 필요가 없다. 실제로 가상 스레드 풀을 생성하는 Executors의 메서드는 최대 개수를 입력받지 않는다.

구분	스레드풀 생성
기존 플랫폼스레드풀	ExecutorService platformExecutor=Executors.newFixedThreadPool (최대개수);
가상 스레드풀	ExecutorService virtualExecutor= Executors.newVirtualThreadPerTaskExecutor();

2. 가상 스레드 생성

개발자는 가상 스레드와 플랫폼 스레드 중에서 하나를 선택해서 작업을 처리해야 하는데 자바 21부터는 가상 스레드를 생성하기 위해 새로운 정적 메서드 2개가 추가되었다.

리턴타입	정적메서드
Thread	Thread.startVirtualThread(Runnable task)
Thread.Builder.OfVirtual	Thread.ofVirtual()

다음 코드는 람다식으로 작성된 Runnable 구현 객체를 매개값으로 해서 startVirtualThread()메서드를 호출한다. 이 메서드는 가상 스레드를 생성하고 바로 작업을 실행한다. 그리고 생성된 가상 스레드 객체를 리턴한다.

```
Thread thread = Thread.startVirtualThread(() -> {
     작업내용
} ) ;
```

다음 코드는 ofVirtual() 메서드로 빌더 객체를 생성하고, 람다식으로 구성된 Runnable 구현 객체를 매개값으로 해서 start() 메서드를 호출한다. 이 메서드로 가상 스레드를 생성하고 바로 내용을 실행한다.그리고 생성된 가상 스레드 객체를 반환한다.

```
Thread thread = Thread.ofVirtual()
               .start(() -> {
                 작업내용
         });
```

다음 코드는 빌드 객체의 name() 메서드로 스레드 이름을 설정하고, 마지막으로 start() 메서드를 호출해서 가상 스레드를 생성하고 실행한다.

```
Thread thread = Thread.ofVirtual()
               .name("threadName “)  //스레드 이름 설정
```

```
                .start(() -> {
                 작업내용
              });
```

```
package 데몬_스레드풀_21가상스레드;

//자바 21에서 추가된 가상 스레드 생성법 3가지
public class VirtualThreadExample {
        public static void main(String[] args) {

                //자바 21 : 첫번째 방법
                Thread.startVirtualThread(() -> {
                        System.out.println("첫번째 가상 쓰레드 생성하는 법");
                });

                //자바 21 : 두번째 방법
                Thread.ofVirtual()
                      .start(()->{
                          System.out.println("두번째 가상 쓰레드 생성하는 법
");
                      });

                //자바 21: 스레드 이름을 설정하는 세번째 방법
                Thread virtualThread03=Thread.ofVirtual()
                                .name("downloadThread") //스레드 이름 설정
                                .start(()->{
                                        System.out.println("스레드 이름을 설정
한 세번째 가상 쓰레드 생성하는 법");
                                });

                System.out.println("virtualThread03     스레드     이름:"     +
virtualThread03.getName());//스레드 이름 반환
        }
}
```