

2025년 상반기 K-디지털 트레이닝

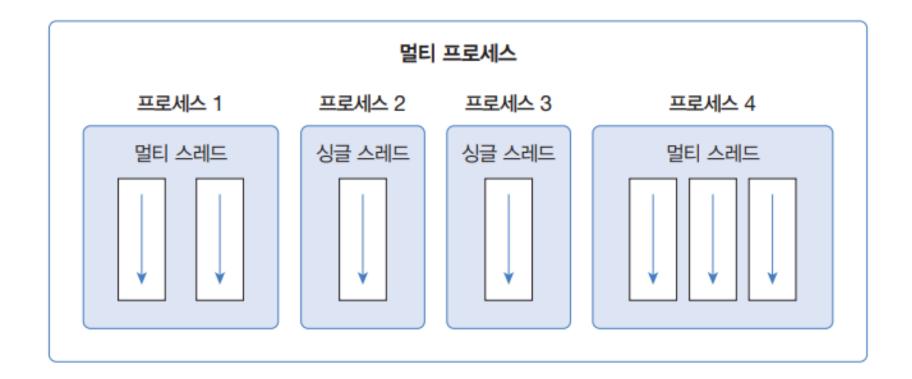
멀티 스레드

[KB] IT's Your Life



💟 멀티 프로세스와 멀티 스레드

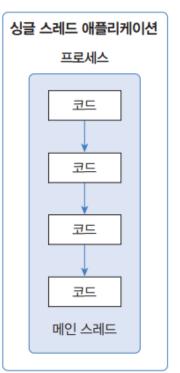
- 프로세스: 운영체제는 실행 중인 프로그램을 관리
- 멀티 태스킹: 두 가지 이상의 작업을 동시에 처리하는 것
- 스레드: 코드의 실행 흐름
- 멀티 스레드: 두 개의 코드 실행 흐름. 두 가지 이상의 작업을 처리
- 멀티 프로세스 = 프로그램 단위의 멀티 태스킹 / 멀티 스레드 = 프로그램 내부에서의 멀티 태스킹



- 메이 스레드 앱 실행 동시에 저절로 생성되고 스진행
 - 메인 스레드는 main() 메소드의 첫 코드부터 순차적으로 실행
 - main() 메소드의 마지막 코드를 실행하거나 return 문을 만나면 실행을 종료
 - 메인 스레드는 추가 작업 스레드들을 만들어서 실행시킬 수 있음
 - 메인 스레드가 작업 스레드보다 먼저 종료되더라도 작업 스레드가 계속 실행 중이라면 프로세스는 종료되지 않음

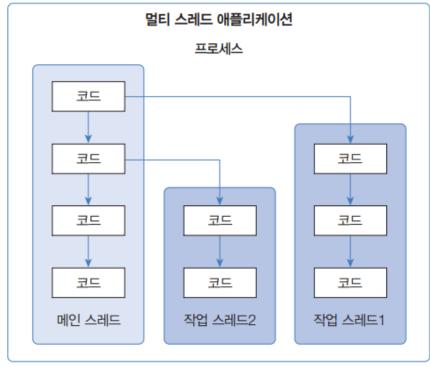
메인스레드는 main메서드를 실행시키는 역할

```
public static void main(String[] args) {
  String data = null;
  if(...) {
  while(...) {
  System.out.println("...");
```



다른 쓰레드는 명시적으로 생성해야하고 워커라고 부름

자바의 디폴트는 는 모든 쓰레드가 다 종료해야 앱이 종료됨 C++은 메인 쓰레드만 종료되도 앱이 종료됨.

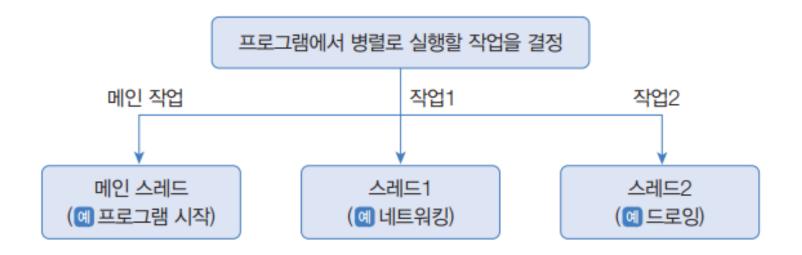


♡ 작업 스레드

멀티 쓰레드 가 자주 등장하는 영역 : 네트워크 앱. 서버. VOD(스트리밍)클라이언트앱.

○ 멀티 스레드 프로그램을 개발 시 먼저 몇 개의 작업을 병렬로 실행할지 결정하고 각 작업별로 스레 드를 생성

기본적으로 자바는 입출력할 때 동기식으로 운영한다. JS와 다르게



- Thread 클래스로 직접 생성
 - Runnable 구현 객체를 매개값으로 갖는 생성자를 호출

Thread클래스에는 RUN이라는 메소드가 있다. 작업쓰레드는 이 Thread클래스의 인스턴스다!!

작업쓰레드는Run메소드를 실행하는 것이 역할이다. 실행이 끝나면 해당 쓰레드는 종료.

```
추상 메서드 run이 정의되어있다
   Thread thread = new Thread(Runnable target);
위임 패턴이죠?
                                         생성자나 메소드로 자신의 필드에
                                         들어갈 객체(동작할)를 넘겨주죠?
2번 기법
    class Task implements Runnable {
      @Override
      public void run() {
        //스레드가 실행할 코드
```

개발자의 역할은 이run 메소드를 어떻게 정의할지 정하고 구현하는 것

1. 상속받아서 override한 방법

2. 이미 다른걸 상속 받아서 쓰레드 클래스 ㅁ상속 못할 때는 위임. runnalbe 인터페이스를 구현하면서

Thread 클래스로 직접 생성

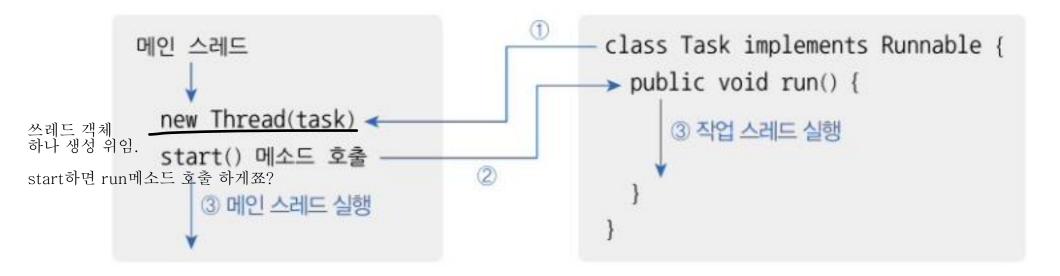
○ Runnable 구현 객체를 매개값으로 갖는 생성자를 호출

```
Runnable task= new Task();
Thread thread = new Thread(task);
                                   익명 구현 객체 기법이죠?
Thread thread = new Thread( new Runnable() {
  @Override
  public void run() {
   //스레드가 실행할 코드
});
thread.start(); 쓰레드 기동 코드. 여기서부터 run메소드 실행
```

작업 스레드 생성과 실행

Thread 클래스로 직접 생성

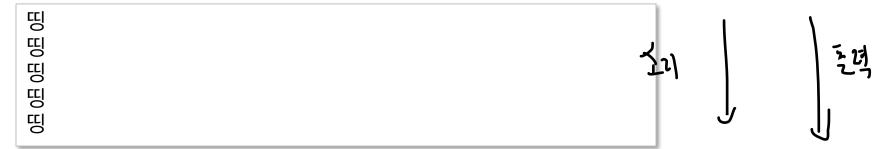
o Runnable 구현 객체를 매개값으로 갖는 생성자를 호출



```
package ch14.sec03.exam01;
           GUI프로그래밍 라이브러리
import java.awt.Toolkit;
public class BeepPrintExample {
 public static void main(String[] args) {
   Toolkit toolkit = Toolkit.getDefaultToolkit();
   for(int i=0; i<5; i++) {
    toolkit.beep(); // 비프음 발생
    try { Thread.sleep(500); } catch(Exception e) {} // 0.5초간 일시 정지
   for(int i=0; i<5; i++) {
    System.out.println("띵");
    try { Thread.sleep(500); } catch(Exception e) {} // 0.5초간 일시 정지
                                           띵
                                           띵
                                           띵
                                           띵
                                           띵
```

```
package ch14.sec03.exam02;
                                 쓰레드 관련은 lang패키지에 들어있는 기본클래스다
import java.awt.Toolkit;
public class BeepPrintExample {
 public static void main(String[] args) {
   Thread thread = new Thread(new Runnable() { // 작업 스레드 생성
    @Override
   → public void run() {
      Toolkit toolkit = Toolkit.getDefaultToolkit();
      for(int i=0; i<5; i++) {
        toolkit.beep();
        try { Thread.sleep(500); } catch(Exception e) {}
   });
                                          thread.run 직접 호출하면 메인쓰레드에 맡기게 된 결과가 나온다
   thread.start(); // 작업 스레드 실행
```

```
for(int i=0; i<5; i++) {
 System.out.println("띵");
 try { Thread.sleep(500); } catch(Exception e) {}
```



Thread 자식 클래스로 생성

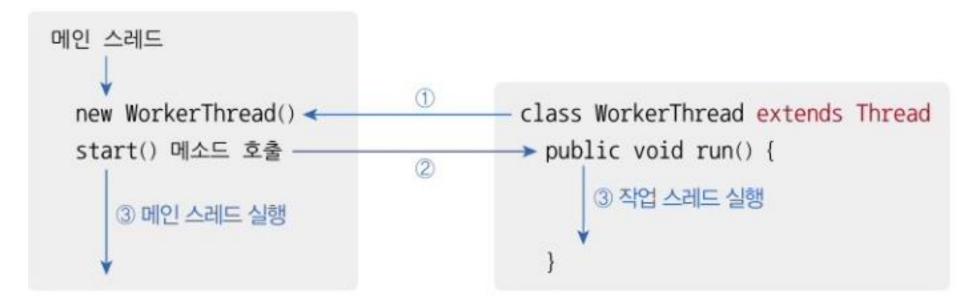
- Thread 클래스를 상속한 다음 run() 메소드를 재정의
 - 스레드가 실행할 코드를 작성
- 객체를 생성 혹은 Thread 익명 자식 객체를 사용 가능

아예 정의한것

```
public class WorkerThread extends Thread {
  @Override
  public void run() {
    //스레드가 실행할 코드
//스레드 객체 생성
Thread thread = new WorkerThread();
thread.start();
```

```
익명 객체로 일회용을 정의및 생성한것
Thread thread = new Thread() {
 @Override
  public void run() {
    //스레드가 실행할 코드
thread.start();
```

☑ Thread 자식 클래스로 생성



```
Thread thread = new Thread() {
  @Override
  public void run() {
    //스레드가 실행할 코드
  }
};
thread.start();
```

작업 스레드 생성과 실행

BeepPrintExample.java

```
package ch14.sec03.exam03;
import java.awt.Toolkit;
public class BeepPrintExample {
 public static void main(String[] args) {
   Thread thread = new Thread() { 익명 객체로 상속 받은 일회용 객체 만들었죠?
     @Override
   → public void run() {
       Toolkit toolkit = Toolkit.getDefaultToolkit();
       for(int i=0; i<5; i++) {
        toolkit.beep();
        try { Thread.sleep(500); } catch(Exception e) {}
   thread.start();
```

상속 버전, 위임 버전 뭘써야 해

위임이 더 좋아.

번외로 람다식은 위임과 잘 어울린다

작업 스레드 생성과 실행

```
for(int i=0; i<5; i++) {
    System.out.println("띰");
    try { Thread.sleep(500); } catch(Exception e) {}
  }
}
```

<mark>♡ 작업 스레드의 이름</mark>

○ 작업 스레드 이름을 Thread-n 대신 다른 이름으로 설정하려면 Thread 클래스의 s<u>etName() 메</u> 소드 사용

```
thread.setName("스레드 이름");
```

- <mark>디버깅할 때 어떤 스레드가 작업을 하는지 조사하기 위해</mark> 주로 사용
- 어떤 스레드가 실행하고 있는지 확인하려면 정적 메소드인 currentThread()로 스레드 객체의 참조를 얻은 다음 getName() 메소드로 이름을 출력

```
Thread thread = <a href="Introdecomposition">Thread.currentThread();</a>
System.out.println(thread.getName());
```

ThreadNameExample.java

```
package ch14.sec04;
public class ThreadNameExample {
 public static void main(String[] args) {
   Thread mainThread = Thread.currentThread();
   System.out.println(mainThread.getName() + " 실행"); // 스레드 이름 리턴
   for(int i=0; i<3; i++) {
     Thread threadA = new Thread() {
      @Override
      public void run() {
        System.out.println(getName() + " 실행"); // 스레드 이름 리턴 \sqrt{}
     threadA.start();
```

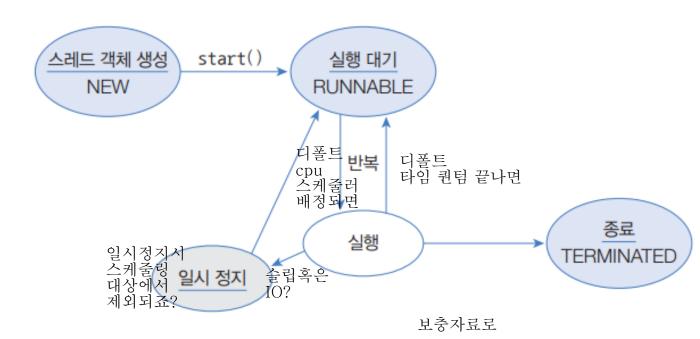
ThreadNameExample.java

```
Thread chatThread = new Thread() {
    @Override
    public void run() {
        System.out.println(getName() + " 실행"); // 스레드 이름 리턴
    }
};
chatThread.setName("chat-thread"); // 작업 스레드 이름 변경
chatThread.start();
}
```

```
main 실행
Thread-1 실행
Thread-0 실행
chat-thread 실행
Thread-2 실행
```

☑ 스레드 상태 ✓

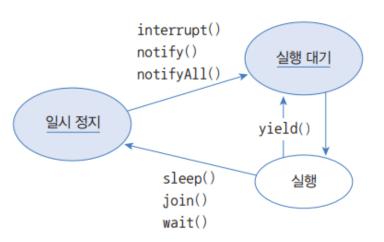
- 실행 대기 상태
 - 실행을 기다리고 있는 상태
- ㅇ 실행 상태
 - CPU 스케쥴링에 따라 CPU를 점유하고 run() 메소드를 실행.
 - 스케줄링에 의해 다시 실행 대기 상태로 돌아갔다가 다른 스레드가 실행 상태 반복
- 종료 상태:
 - 실행 상태에서 run() 메소드가 종료
 - 실행할 코드 없이 스레드의 실행을 멈춘 상태



일시 정지 상태

- 스레드가 실행할 수 없는 상태
- 스레드가 다시 실행 상태로 가기 위해서는 일시 정지 상태에서 실행 대기 상태로 가야야 함
- o Thread 클래스의 sleep() 메소드:
 - 실행 중인 스레드를 일정 시간 멈추게 함
 - 매개값 단위는 밀리세컨드(1/1000)

```
try {
  Thread.sleep(1000);
} catch(InterruptedException e) {
  // interrupt() 메소드가 호출되면 실행
```



구분	메소드	설명
일시 정지로 보냄	sleep(long millis)	주어진 시간 동안 스레드를 일시 정지 상태로 만든다. 주어진 시간이 지나면 자동적으로 실행 대기 상태가 된다.
	join()	join() 메소드를 호출한 스레드는 일시 정지 상태가 된다. 실행 대기 상태가 되려면, join() 메소드를 가진 스레드가 종료되어야 한다.
	wait()	동기화 블록 내에서 스레드를 일시 정지 상태로 만든다.
일시 정지에서 벗어남	interrupt()	일시 정지 상태일 경우, InterruptedException을 발생시켜 실행 대기 상태 또는 종료 상태로 만든다.
	notify() notifyAll()	wait() 메소드로 인해 일시 정지 상태인 스레드를 실행 대기 상태로 만든다.
실행 대기로 보냄	yield()	실행 상태에서 다른 스레드에게 실행을 양보하고 실행 대기 상태가 된다.

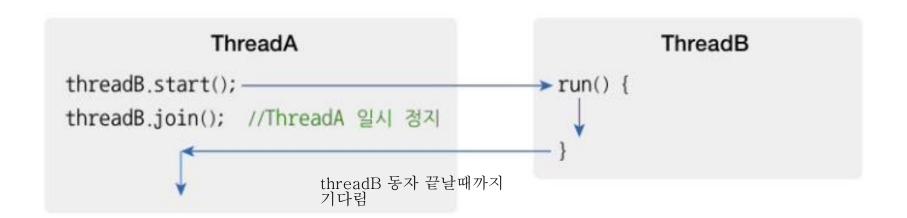
SleepExample.java

```
package ch14.sec05.exam01;
import java.awt.Toolkit;
public class SleepExample {
 public static void main(String[] args) {
   Toolkit toolkit = Toolkit.getDefaultToolkit();
   for(int i=0; i<10; i++) {
     toolkit.beep();
     try {
       Thread.sleep(3000);
     } catch(InterruptedException e) {
```

다른 스레드의 종료를 기다림

스레드의 동작 순서 제어

- 어떤 스레드 A는 특정 스레드 B의 작업 결과를 바탕으로 동작
- 스레드 A는 스레드 B의 작업이 끝날 때까지 하는 일 없이 기다려야 함
- → 어떻게 스레드 B의 작업 종료를 인지할 수 있는가
- o join() 메서드
 - 해당 스레드가 종료할 때까지 대기 하다가 스레드가 종료하면 실행을 재개



SumThread.java

```
package ch14.sec05.exam02;
public class SumThread extends Thread {
 private long sum;
 public long getSum() {
   return sum;
 public void setSum(long sum) {
   this.sum = sum;
 @Override
 public void run() {
   for(int i=1; i<=100; i++) {
     sum+=i;
```

SumThread.java

```
package ch14.sec05.exam02;

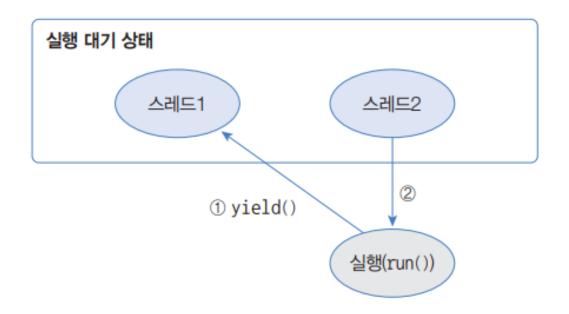
public class JoinExample {
   public static void main(String[] args) {
      SumThread sumThread = new SumThread();
      sumThread.start();
      try {
        sumThread.join();
      } catch (InterruptedException e) {
      }
      System.out.println("1~100 합: " + sumThread.getSum());
   }
}
```

1~100 합: 5050

다른 스레드에게 실행 양보

스케줄러가 참조하는 우선 순위를 변경할 수 있고 다른 쓰레드에게 양보할 수 있다

- yield() 메소드: 실행되는 스레드는 실행 대기 상태로 돌아가고, 다른 스레드가 실행되도록 양보
- 무의미한 반복을 막아 프로그램 성능 향상

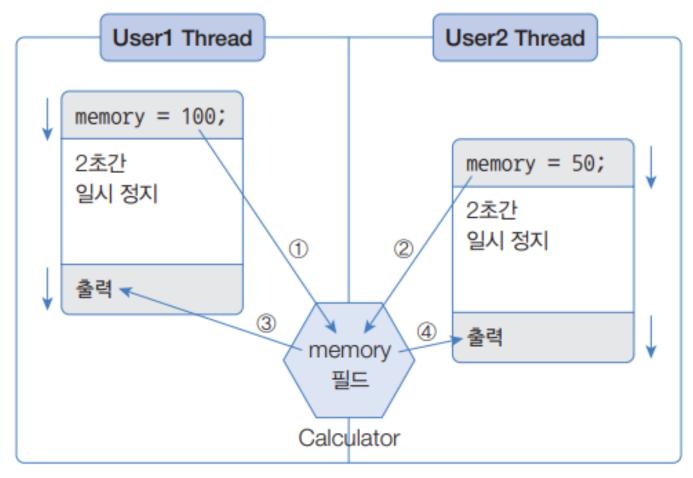


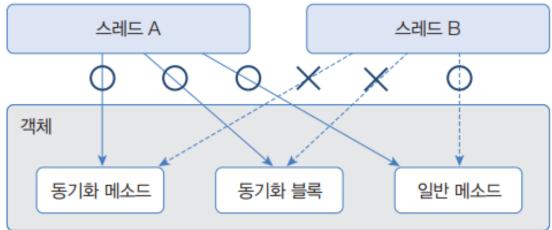
```
public void run() {
  while(true) {
   if(work) {
      System.out.println("ThreadA 작업 내용");
    } else {
      Thread.yield();
```

6 스레드 동기화

☑ 동기화 메소드와 블록

○ 스레드 작업이 끝날 때까지 객체에 잠금을 걸어 스레드가 사용 중인 객체를 다른 스레드가 변경할 수 없게 함

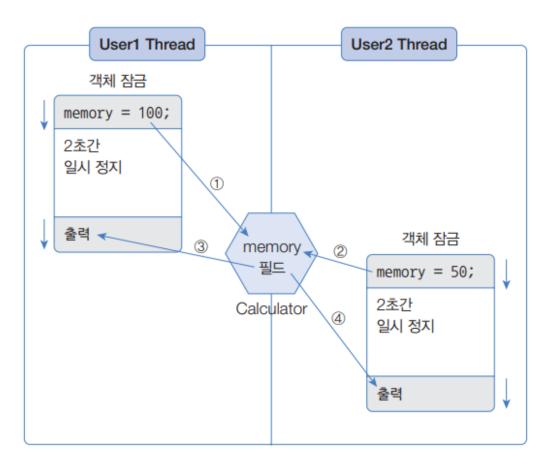




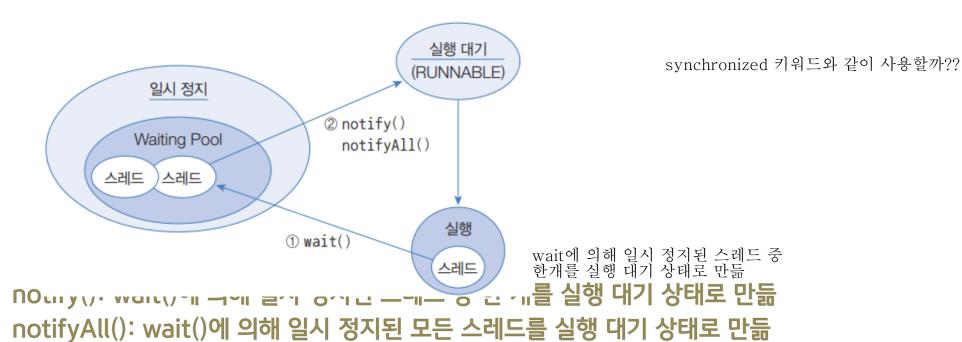
☑ 동기화 메소드 및 블록 선언

- 인스턴스와 정적 메소드에 synchronized 키워드 붙임
- 동기화 메소드를 실행 즉시 객체는 잠금이 일어나고, 메소드 실행이 끝나면 잠금 풀림
- 메소드 일부 영역 실행 시 객체 잠금을 걸고 싶다면 동기화 블록을 만듦

```
public synchronized void method() {
 //단 하나의 스레드만 실행하는 영역
public void method () {
  //여러 스레드가 실행할 수 있는 영역
  synchronized(공유객체) {
   //단 하나의 스레드만 실행하는 영역
 //여러 스레드가 실행할 수 있는 영역
```



- ☑ wait()과 notify()를 이용한 스레드 제어 돌다 Object 클래스의 메서드이다. 공유 자원에 대해서. 어떤 클래스든 공유자원이 될 수 있으니까
 - 두 스레드 교대 실행 시 공유 객체는 두 스레드가 작업할 내용을 각각 동기화 메소드로 정함
 - 한 스레드 작업 완료 시 notify() 메소드를 호출해 일시 정지 상태에 있는 다른 스레드를 실행 대기 상태로 만들고, wait() 메소드를 호출하여 자신은 일시 정지 상태로 만듦



자바는 공유자원까지 지원한다. 동기화가 보장된 자료구조를 지원한다!!

💟 안전하게 스레드 종료하기

작업쓰레드는 종료되지 않고 무한 루프를 도는 경우가 맣다. 이때 상태변수를 이용하며 종료를 제어한다. 외부에서 상태변수의 값을 변하게 하고

- 스레드 강제 종료 stop() 메소드: deprecated(더 이상 사용하지 않음)
- 스레드를 안전하게 종료하려면 사용하던 리소스(파일, 네트워크 연결)를 정리하고 run() 메소드를 빨리 종료 해야 함

☑ 조건 이용

o while 문으로 반복 실행 시 조건을 이용해 run() 메소드 종료를 유도

```
public class XXXThread extends Thread {
  private boolean stop; • stop이 필드 선언

public void run() {
  while(!stop) {
    //스레드가 반복 실행하는 코드;
  }
  //스레드가 사용한 리소스 정리 • 리소스 정리 • 스레드 종료
}
```

하지만 이방법의 부작용

stop을 검사할 때까지는 진행되어야 해

나느 바로 종료 시키고 싶은ㄷ

7 스레드 안전 종료

- ☑ interrupt() 메소드 이용 앞 페이지 의 해결법. 강제로 즉시 종료 시키는 메소드
 - 스레드가 일시 정지 상태에 있을 때 InterruptedException 예외 발생
 - 예외 처리를 통해 run() 메소드를 정상 종료

```
XThread thread = new XThread();
thread.start();

"
thread.interrupt();

Thread.sleep(1); //일시 정지

}
catch(InterruptedException e) {

}
//스레드가 사용한 리소스 정리
}

**Thread.sleep(1); //일시 정지

**V제로 쓰레드를 종료시키니까 생기는 부작용

**V제로 쓰레드를 종료시키니까 생기는 부작용
```

o Thread의 interrupted()와 isInterrupted() 메소드는 interrupt() 메소드 호출 여부를 리턴

```
boolean status = Thread.interrupted();
boolean status = objThread.isInterrupted();
```

<mark>강제 종료 시키는 쓰레드 특은 무한 루프 돌며 일을</mark> 하며 계속 상주하는 쓰레드. main쓰레드가 종료되면 알아서 따라서 같이 자동으로 클린업하며 종료하게하고 싶은데? => 스레드를 데몬스레드로 설정

데몬 스레드

- 주 스레드의 작업을 돕는 보조적인 역할을 수행하는 스레드
- 주 스레드가 종료되면 데몬 스레드도 따라서 자동 종료
- 데몬 스레드 적용 예:
 - 워드프로세서의 자동 저장, 미디어플레이어의 동영상 및 음악 재생, 가비지 컬렉터
- 주 스레드가 데몬이 될 스레드의 setDaemon(true)를 호출

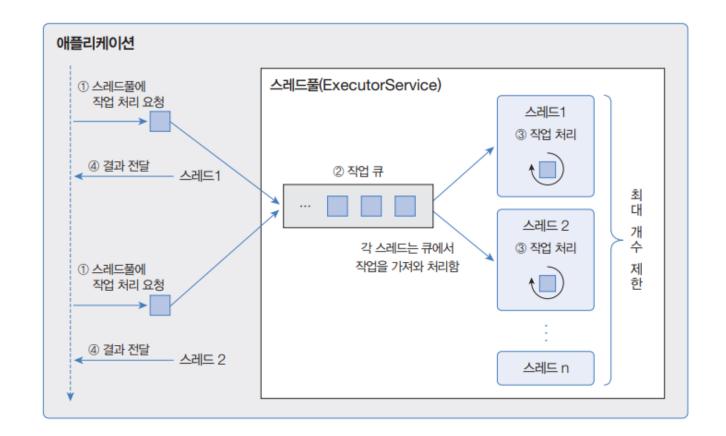
```
public static void main(String[] args) {
  AutoSaveThread thread = new AutoSaveThread();
  thread.setDaemon(true);
  thread.start();
```

풀 자체도 큐로 보관 활용함.

💟 스레드풀로 작업 처리 제한하기

그럼 몇개를 미리 해놓는게 좋을까 코어갯수만큼!!

- 작업 처리에 사용되는 스<u>레드 개수를 제한</u>하고 <u>작업 큐에 들어오는 작업들을 스레드가 하나씩 맡아 처리</u>하는 방식
- 작업 처리가 끝난 스레드는 다시 작업 큐에서 새로운 작업을 가져와 처리
- <u>작업량이 증가해도 스레드의 개수가 늘어나지 않아 애</u>플리케이션의 성능의 급격한 <u>저하 방지</u>



쓰레드 활용시 시간을 많이 잡아먹는 구간

생성 기동 하는 구간 & 종료하는 구간

셋업, 클린업 과정 때문에.

메모리 할당하고 구조 만들고 채우고 ... 비우고 메모리 반환하고...

꽤 큭다.

이를 줄이면 좋아

매번 생성 기동 종료하지 말고 미리 여러개를 생성시켜 놓고 요청이 들어올 때 배정하고 요청 완료하면 종료 말고 보관. 프로그램 종료시 종료.

갯수도 제한 하여 작업 큐에 보관하면 더 좋아 현대 서버들은 다 이기법 운용

☑ 스레드풀 생성

- o java.util.concurrent 패키지에서 ExecutorService 인터페이스와 Executors 클래스를 제공
- Executors의 다음 두 정적 메소드를 이용하면 스레드풀인 ExecutorService 구현 객체를 만들 수 있음

메소드명(매개변수)	초기수	코어 수	최대 수
newCachedThreadPool()	0	0	Integer,MAX_VALUE
newFixedThreadPool(int nThreads)	0	생성된 수	nThreads

- 초기 수: 스레드풀이 생성될 때 기본적으로 생성되는 스레드 수
- 코어 수: 스레드가 증가된 후 사용되지 않는 스레드를 제거할 때 최소한 풀에서 유지하는 스레드 수
- 최대 수: 증가되는 스레드의 한도 수

ExecutorService executorService = Executors.newCachedThreadPool();

ExecutorService executorService = Executors.newFixedThreadPool(5);

🗸 스레드풀 종료

○ 스레드풀의 스레드는 main 스레드가 종료되더라도 작업을 처리하기 위해 계속 실행 상태로 남음

리턴 타입	메소드명(매개변수)	설명
void	shutdown()	현재 처리 중인 작업뿐만 아니라 작업 큐에 대기하고 있는 모든 작업을 처리한 뒤에 스레드풀을 종료시킨다.
List(Runnable)	shutdownNow()	현재 작업 처리 중인 스레드를 interrupt해서 작업을 중지시키고 스레드 풀을 종료시킨다. 리턴값은 작업 큐에 있는 미처리된 작업(Runnable) 의 목록이다.

😕 작업 생성과 처리 요청

o 하나의 작업은 Runnable 또는 Callable 구현 클래스로 표현

Runnable 익명 구현 클래스	Callable 익명 구현 클래스
new Runnable() { @Override public void run() { //스레드가 처리할 작업 내용 }	new Callable〈T〉 { @Override public T call() throws Exception { //스레드가 처리할 작업 내용 return T;
} o	} }

리턴 타입	메소드명(매개변수)	설명
void	execute(Runnable command)	Runnable을 작업 큐에 저장작업 처리 결과를 리턴하지 않음
Future(T)	submit(Callable(T) task)	Callable을 작업 큐에 저장작업 처리 결과를 얻을 수 있도록 Future를 리턴