
K G 아 이 티 뱅 크
JAVA
J A V A


Thread

Thread

- **실질적인 프로그램의 "실행"을 담당하고 있는 것이자 개념**
  - 프로그램 : 실행이 가능한 코드의 모음집
  - 프로세스 : 프로그램이 메모리상에서 실행된 것
  - 쓰레드 : 프로세스 내에서 실행되는, 가장 작은 실행단위
- **이러한 실행 단위를 여러개를 준비하여 동시동작을 구현함**
  - 단, 이에 따라 여러 문제가 발생하므로 테스트가 요구됨

```
class Sub extends Thread { ... }
class StandAlone implements Runnable { .... }

// Thread 구현 클래스를 상속했을 경우
Thread thread1 = new Sub();
tObject1.start();
// Runnable 인터페이스를 구현했을 경우
Runnable tObject = new StandAlone();
Thread thread2 = new Thread(tObject);
thread2.start();
```

Thread

❖ **쓰레드를 쓸 때는 run 메서드의 재정의를 반드시 요구함**

① Thread 클래스의 상속보다는 Runnable구현이 더 유연함

② 기본적인 제어는 sleep과 join을 통하여 가능함

③ 정교한 동작은 동기화 알고리즘이나 자료구조를 사용함

```
// 일반적으로 Runnable 인터페이스를 구현하여 준비함
class ThreadObject implements Runnable {
    public void run() { .... }
}
// 이 클래스로 만든 객체로 다시 Thread 객체를 생성함
Thread thread1 = new Thread(new ThreadObject());
// 단, 쓰레드의 경우도 예외처리가 필요함
try { thread1.start(); }
catch(InterruptedException e) {
    System.out.println("작업이 중단되었습니다.");
}
```

synchronized

synchronized

- **쓰레드는 생성되어 시작되면 최선의 노력을 다하여 실행됨**
  - 지나치게 빨리 실행되는 쓰레드는 의도와 다르게 작동함
  - 쓰레드의 실행을 일정한 기준으로 조절해서 통제해야 함
- **synchronized 는 이러한 쓰레드의 정교한 통제를 제공함**
  - 서로 다른 여러개의 쓰레드가 순차적으로 작동할 수 있음
  - 공유자원에 대하여 통제된 방식으로 접근할 수 있음

```
class StandAlone implements Runnable {
    Object lock;
    public void run() {
        synchronized(lock) {
            lock.notify();
            lock.notifyAll();
            lock.wait();
        }
    }
}
```

synchronized

❖ **synchronized를 사용할 때 주의해야 하는 문제가 존재함**

① Starvation : lock이 하나일 때, 무제한 대기상태가 됨

② Dead Lock : lock이 2개 이상일 때, 무제한 대기상태가 됨

❖ **이를 예방하기 위한 기본적인 철칙이 존재하며 지켜야 함**

① 하나의 lock은 시간제한 wait을 설정해주는 것이 좋음

② 여러개의 lock은 동일한 순서로 설정해줘야 함

❖ **시간제한 wait 설정하기**

```
public void run() {
    synchronized(lock) {
        lock.wait(1500);
    }
}
```

❖ **동일한 순서로 lock 설정하기**

```
public void run() {
    synchronized(lock1) {
        synchronized(lock2) {
        }
    }
}
```