**[ 24 ] Java Thread**

목표 : 멀티 프로세스와 스레드를 구분할 수 있다.

멀티 태스킹 개념을 이해한다.

Thread의 life cycle을 이해한다.

동기화(synchronized) 개념을 이해한다.

자바에서 활용하는 Thread를 이해하고 멀티 스레드 문법으로 구현할 수 있다.

1. 프로세스와 스레드란?

실행 가능한 보조기억장치(HDD, SDD)에 있는 파일을 우리는 프로그램이라 부릅니다. 이러한 프로그램을 실행하게 되면, 이 프로그램은 메모리에 상주하게 되고 우리는 이를 프로세스라 부릅니다. Windows는 멀티프로세스를 지원하고 있죠. 쉽게 말해서 하나의 CPU(컴퓨터)가 동시에 여러 프로그램을 동시에 실행할 수 있습니다. 이것이 멀티 프로세스입니다.

멀티프로세스 시스템에서 각 프로세스는 동시에 실행되는 것처럼 보이지만, CPU가 하나이기 때문에 실행시간을 잘게 나누어 프로세스가 돌아가는데 각 프로세스들이 돌아가면서 CPU를 점유하는데 프로세스간 CPU점유작업인 스케줄링을 한다.

그럼 **스레드(멀티스레드)란 무엇일까요? 하나의 프로그램이 동시에 여러 개의 일을 수행할 수 있도록 해 주는 것입니다**. 위에서 말한 하나의 프로세스에서 다시 여러 가지 일을 하는 것을 뜻 합니다. 예를 들어 채팅프로그램은 파일을 전송하면서 채팅을 할 수 있습니다. 동일한 채팅프로그램에서 파일전송과 채팅을 동시에 하고 있는 것 입니다. 멀티스레드이죠.

JAVA는 기본적으로 멀티스레드를 지원하기 위해 java.lang.Thread 클래스를 제공하고 있습니다.

멀티프로세스 vs. 멀티쓰레드

하나의 새로운 프로세스를 생성하는 멀티프로세스 보다 하나의 새로운 쓰레드를 생성하는 것이 더 적은 비용이 듭니다.

스레드는 두 가지 형태로 존재 합니다. 하나는 n개의 스레드가 객체 하나를 공유하는 방식이고, 또 다른 하나는 n개의 스레드가 각각 객체 n개를 사용하는 방식 입니다.

(멀티스레드의 장점)

자원을 보다 효율적으로 사용할 수 있다.

사용자에 대한 응답성이 향상된다.

작업이 분리되어 코드가 간결해 진다.

(멀티스레드의 단점)

동기화(synchronization)에 주의해야 한다.

교착상태(dead-lock)가 발생하지 않도록 주의해야 한다.

각 쓰레드가 효율적으로 고르게 실행될 수 있게 해야 한다(☞프로그래밍할 때 고려해야 할 사항들이 많다)

1. 스레드의 생명주기
   1. 생성
   2. 실행
   3. 대기
   4. 완료
2. 멀티 스레드 문법
3. Runable 인터페이스 구현을 통한 Thread

**public** **class** ThreadEx **implements** Runnable{

@Override

**public** **void** run() {

System.***out***.println(Thread.*currentThread*().getName());

System.***out***.println("ThreadEx");

**for**(**int** i=0 ; i<10 ; i++) {

System.***out***.println(Thread.*currentThread*().getName()+" : i="+i);

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) { }

}

}

}

**public** **class** ThreadExTestMain {

**public** **static** **void** main(String[] args) {

ThreadEx threadEx = **new** ThreadEx();

Thread thread1 = **new** Thread(threadEx, "A");

Thread1.start();

Thread thread2 = **new** Thread(threadEx, "B");

thread2.start();

System.***out***.println(Thread.*currentThread*().getName());

System.***out***.println("main()");

}

}

1. Thread 클래스 상속을 통한 Thread

**public** **class** ThreadEx **extends** Thread{

@Override

**public** **void** run() {

System.***out***.println(Thread.*currentThread*().getName());//현재구동되는쓰레드명 System.***out***.println("ThreadEx");

**for**(**int** i=0 ; i<10 ; i++) {

System.***out***.println(Thread.*currentThread*().getName()+" : i="+i);

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) { }

}

}

}

**public** **class** ThreadExTestMain {

**public** **static** **void** main(String[] args) {

ThreadEx thread1 = **new** ThreadEx();

thread1.setName("A");

thread1.start();

ThreadEx thread2 = **new** ThreadEx();

thread2.setName("B");

thread2.start();

System.***out***.println(Thread.*currentThread*().getName());

System.***out***.println("main은 여기서 끝");

}

}

1. 객체1개, 스레드n개

**public** **class** ThreadEx **implements** Runnable{

**private** **int** num = 0; //값 공유

@Override

**public** **void** run() {

**for**(**int** i=0 ; i<10 ; i++) {

**if**(Thread.*currentThread*().getName().equals("A")) {

System.***out***.println("~ ~ A 수행중 ~ ~");

++num;

}

System.***out***.println("ThreadName : "+Thread.*currentThread*().getName()+"\t num : "+num);

**try** {

Thread.*sleep*(1500);

} **catch** (InterruptedException e) { }

}

}

**public** **int** getNum() {

**return** num;

}

}

**public** **class** ExObject1ThreadN {

**public** **static** **void** main(String[] args) {

ThreadEx threadEx = **new** ThreadEx();

Thread threadA = **new** Thread(threadEx, "A");

Thread threadB = **new** Thread(threadEx, "B");

threadA.start();

threadB.start();

System.***out***.println("main 함수 쓰레드 : "+Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("DONE. 이 시점의 num : "+threadEx.getNum());

}

}

1. 객체n개, 스레드n개

**public** **class** ExObjectNThreadN {

**public** **static** **void** main(String[] args) {

ThreadEx targetA = **new** ThreadEx();

ThreadEx targetB = **new** ThreadEx();

Thread threadA = **new** Thread(targetA, "A");

Thread threadB = **new** Thread(targetB, "B");

threadA.start();

threadB.start();

System.***out***.println("main 함수 쓰레드 : "+Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("DONE. 이 시점의 A측 num : "+targetA.getNum());

System.***out***.println("DONE. 이 시점의 B측num : "+targetB.getNum());

}

}

1. 선거 개표 방송 예제

선거개표방송에서 지역1, 지역2, 지역3의 개표율이 엎치락 뒤치락 하는 것을 Thread를 이용하여 만들어 보겠습니다.

**public** **class** VoteThread **implements** Runnable{

**private** **final** **int** TARGETNUM = 100;

**private** **int** count = 0;

Random random = **new** Random();

@Override

**public** **void** run() {

StringBuffer sb = **new** StringBuffer();

**while**(**true**) {

count = count + random.nextInt(10);

sb.delete(0, sb.toString().length());

**if**(count>=TARGETNUM) {

count = 100;

**for**(**int** i=0 ; i<count ; i++) {

sb.append('\*');

}

System.***out***.println(Thread.*currentThread*().getName()+"개표율 : "+count+"\t"+sb);

**break**;

}**else** {

**for**(**int** i=0 ; i<count ; i++) {

sb.append('\*');

}

System.***out***.println(Thread.*currentThread*().getName()+"개표율 : "+count+"\t"+sb);

}

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) {

System.***out***.println(e.getMessage());

}

}

}

}

**public** **class** VoteThreadMain {

**public** **static** **void** main(String[] args) {

VoteThread voteLocation1 = **new** VoteThread();

VoteThread voteLocation2 = **new** VoteThread();

VoteThread voteLocation3 = **new** VoteThread();

Thread location1 = **new** Thread(voteLocation1, "Location1");

Thread location2 = **new** Thread(voteLocation2, "Location2");

Thread location3 = **new** Thread(voteLocation3, "Location3");

location1.start();

location2.start();

location3.start();

}

}

1. Synchronized

객체에 선언되어 있는 인스턴스 변수를 스레드에서 공유하게 되어, 인스턴스의 값에 영향을 미치게 됩니다.

이런 경우를 대비해서 JAVA에서는 synchronized라는 키워드를 이용합니다. Synchronized는 먼저 수행되는 스레드의 모든 작업이 끝날 때까지 다른 스레드는 기다리도록 하는 예약어입니다.

즉, 동기화란? 하나의 자원을 여러 태스크가 사용하려 할 때에, 한 시점에서 하나의 태스크만이 사용할 수 이도록 하는 것.

대부분의 응용 프로그램에서 다수개의 스레드가 공유할 수 있는 부분이 요구되며 공유부분은 상호배타적으로 사용되어야 합니다.

예를 들어 영화극장 좌석예매시스템이 있습니다. 네트워크를 이용하여 여러 매표소에서 좌석이 예매될 때, 매표소들간에 좌석 예매시스템의 좌석 정보를 동시에 접근할 수 있다면, 같은 좌석을 여러 사람에게 중복되게 예매할 수도 있습니다. 따라서 이렇게 여러 데스크들이 하나의 정보에 접근할 때에는, 한번에 하나씩의 데스크만이 접근할 수 있도록 해야 합니다.

**public** **class** ThreadTest **implements** Runnable{

**int** testNum = 0;

@Override

**public** **synchronized** **void** run() {

**for**(**int** i=0 ; i<10 ; i++){

**if**(Thread.*currentThread*().getName().equals("A")){

System.***out***.println("~ ~ ~ ~ ~ A 수행중 ~ ~ ~ ~ ~");

testNum++;

}

System.***out***.println("testNum:"+testNum+"\tThreadName:"+Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) { }

}

}

}

**public** **class** ThreadTestMain {

**public** **static** **void** main(String[] args) {

ThreadTest threadTest = **new** ThreadTest();

Thread threadA = **new** Thread(threadTest, "A");

Thread threadB = **new** Thread(threadTest, "B");

threadA.start();

threadB.start();

System.***out***.println("main스레드 : " + Thread.*currentThread*().getName());

}

}

**public** **class** ThreadTest **implements** Runnable {

**private** **int** testNum = 0;

@Override

**public** **void** run() {

**for**(**int** i=0 ; i<10 ; i++)

out();

}

**private** **synchronized** **void** out() {

**if**(Thread.*currentThread*().getName().equals("A")) {

System.***out***.println("★ ★ ★ ★ ★ A 수 행 중★ ★ ★ ★ ★");

testNum++;

}

System.***out***.println("testNum="+testNum+"\tThreadName="+Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) { }

}

}

<입출금 예제>

**public** **class** ATM {

**private** **int** account;

**public** ATM(**int** account) {

**this**.account = account;

}

**public** **synchronized** **void** deposit(**int** amount, String who) {

System.***out***.println("★★"+who+"가 입금 시작\t 현재 잔액 : "+account);

account += amount;

System.***out***.println("★★"+who+"가 입금 : "+amount+"\t현잔액 : "+account);

System.***out***.println("★★"+who+"가 입금 끝\t 현재 잔액 : "+account);

}

**public** **synchronized** **void** withdraw(**int** amount, String who) {

System.***out***.println("■■■"+who+"가 출금 시작 \t 현 잔액 : "+account);

account -= amount;

System.***out***.println("■■■"+who+"가 출금 : "+amount+"\t현잔액 : "+account);

System.***out***.println("■■■"+who+"가 출금 끝 \t 현 잔액 : "+account);

}

**public** **int** getAccount() {

**return** account;

}

}

**public** **class** ATM\_ACTION **implements** Runnable {

**private** **int** depositORwithdraw = 1;

**private** ATM obj;

**public** ATM\_ACTION(ATM obj, **int** depositORwithdraw) {

**this**.obj = obj;

**this**.depositORwithdraw = depositORwithdraw;

}

@Override

**public** **void** run() {

**for**(**int** i=0 ; i<5 ; i++) {

**if**(depositORwithdraw==1) {

obj.deposit(1000, Thread.*currentThread*().getName());

depositORwithdraw = 0;

}**else** {

obj.withdraw(500, Thread.*currentThread*().getName());

depositORwithdraw = 1;

}//if

}//for

}//run

}//class

**public** **class** ThreadTestMain {

**public** **static** **void** main(String[] args) {

ATM obj = **new** ATM(10000);

ATM\_ACTION card = **new** ATM\_ACTION(obj, 1);

Thread mom = **new** Thread(card, "엄마");

Thread dad = **new** Thread(card, "아빠");

mom.start();

dad.start();

}

}