제 44강

스레드3

교재: p251~256

목차

복습

스레드란?

: 하나의 프로세스 안 에서 두 가지 이상의 일을 하도록 하는 것

<용어 정리>

- 프로세스(Process) : 실행 중인 프로그램
- 스레드(Thread) : 프로세스에서 작업을 수행하는 것
- 멀티 스레드 프로세스(Multi-Thread Process)
  : 두 가지 이상의 작업을 하는 프로세스

복습

[스레드 생성 방법]

- **자바 스레드로 작동할 작업이 무엇인지 코드로 작성**

1) Thread 클래스 상속
2) Runnable 인터페이스 구현

- **스레드 코드가 실행할 수 있도록 JVM 한테 요청**

1) 인스턴스 생성 후 start( ) 호출
2) 인스턴스 생성 후 Thread( ) 매개변수 생성자에 인자값 전달, start( ) 호출

복습

1) Thread 클래스 상속

```
class T1 extends Thread{
        public void run( ){
                        //작업할 내용
                }
}
```

```
Th1 t1 = new Th1( );
t1. start( );
```

2) Runnable 인터페이스 구현

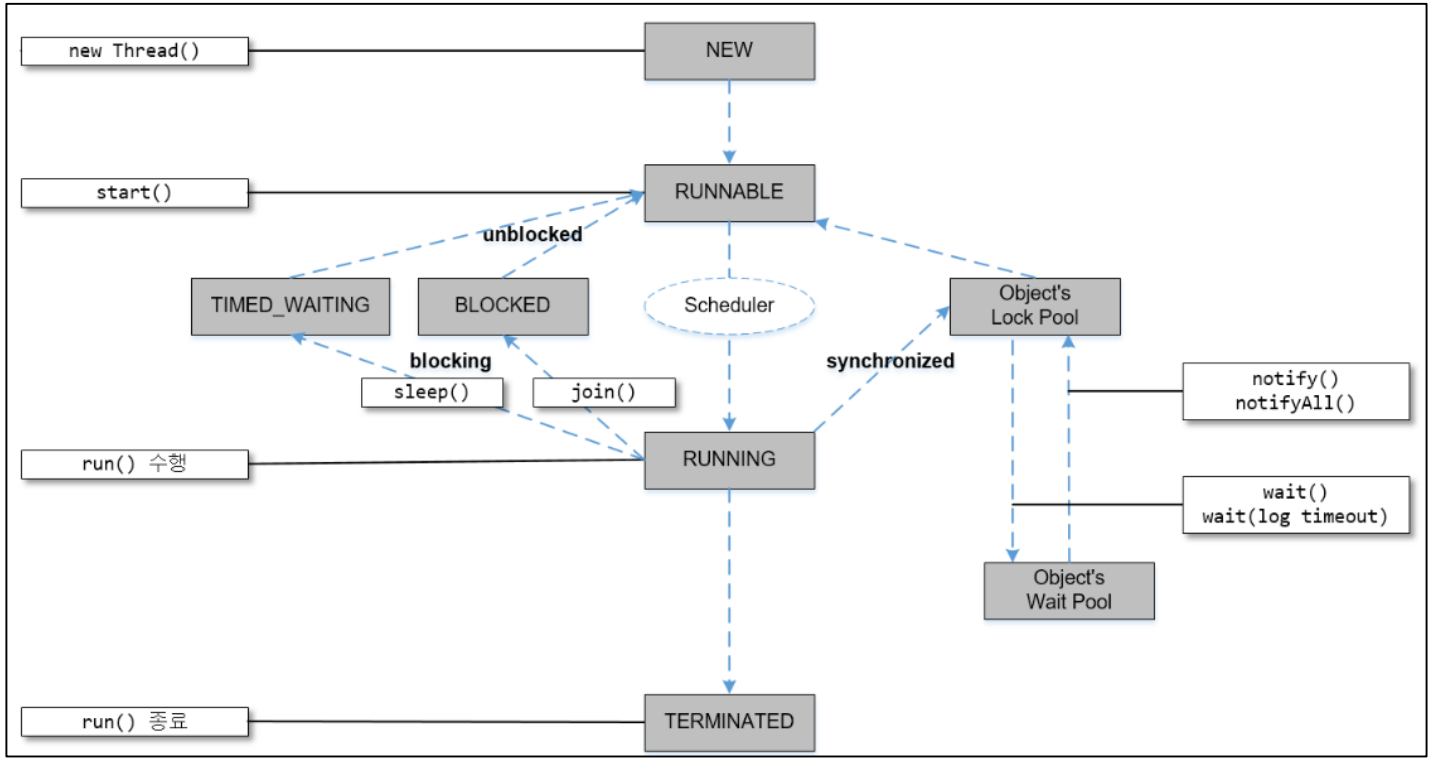
```
class T2 implements Runnable{
        public void run( ){
                        //작업할 내용
                }
}
```

```
Th2 t2 = new Th2( )
Thread t = new Thread(t2);

t.start( );
```

복습

스레드의 생명주기(Thread Life Cycle)



출처.https://codedragon.tistory.com/3526

1. 스레드의 동기화

스레드의 동기화?

: 멀티 스레드로 작업 시, 스레드 간 작업이 서로 간섭이 되지 않도록 하는 것

멀티 스레드의 문제점?

멀티 스레드 기법은 자원을 공유하는데, 이 때 동시에 같은 자원을 처리한다면,
자원 값이 실제 처리 해야 하는 작업보다 더 많은 작업이 발생할 수 있다.

<학교 앞>

<은행 앞>

1. 스레드의 동기화

<실습> ShareBank.java

엄마랑 아들이 동시에 돈을 뽑는 작업을 스레드를 사용해, 멀티태스킹 으로 구현했을 때,
발생하는 문제를 실습을 통해 확인해보자.

```
class ATM implements Runnable{
   int money = 10000;
   public void run() {
               for (int i = 0; i < 5; i++) {
                   withdraw(1000);
                   printInfo();
                   try {
                       Thread.sleep(1000);
                   } catch (InterruptedException e) {  }   }   }
   void withdraw(int money) {
       Thread.yield();
       this.money -= money;
       System.out.println(Thread.currentThread().getName() + "이(가) " + money + "원 출금");
   }
   void printInfo() {
       System.out.println("현재 잔액" + money + "원");
   }}
```

다른 스레드에게 작업 양도

```
public class ShareBank {
    public static void main(String[] args) {
공유자원 ATM atm = new ATM();
        Thread mom = new Thread(atm,"엄마");
        Thread son = new Thread(atm,"아들");

        mom.start();
        son.start();

    }
}
```

1. 스레드의 동기화

<실습> ShareBank.java

엄마랑 아들이 동시에 돈을 뽑는 작업을 스레드를 사용해, 멀티태스킹으로 구현했을 때,
발생하는 문제를 실습을 통해 확인해보자.

```
엄마이(가) 1000원 출금
현재 잔액8000원
아들이(가) 1000원 출금
현재 잔액8000원
엄마이(가) 1000원 출금
현재 잔액7000원
아들이(가) 1000원 출금
현재 잔액7000원
엄마이(가) 1000원 출금
현재 잔액6000원
아들이(가) 1000원 출금
현재 잔액5000원
아들이(가) 1000원 출금
현재 잔액3000원
엄마이(가) 1000원 출금
현재 잔액3000원
아들이(가) 1000원 출금
현재 잔액2000원
엄마이(가) 1000원 출금
현재 잔액1000원
```

엄마가 돈을 뽑고 아들이 돈을 뽑는다면,
2000원이 감소 해야하는데, 감소가 되지 않는 것을 확인할 수 있다!

1. 스레드의 동기화

<실습> Exam-98.java

한정된 티켓 수량을 멀티 태스킹으로 티켓팅을 했을 때 발생할 수 있는 문제점을 실습을 통해 확인하기.

```
class MyThreadB implements Runnable{
     Ticketing ticket = new Ticketing();
    public void run() {
        ticket.ticketing();
    }
}
class Ticketing{
    int ticketNumber=1;
    public void ticketing() {
        if(ticketNumber>0) {
            System.out.println(Thread.currentThread().getName()+"가 티켓팅 성공!");
            ticketNumber--;
        }
        else {
            System.out.println(Thread.currentThread().getName()+"가 티켓팅 실패!");
        }
        System.out.println(Thread.currentThread().getName()+"가 티켓팅 시도 후 티켓수:"+ticketNumber);

        }
    }
```

```
public class Synchronized1 {
    public static void main(String[] args) {
            MyThreadB s1=new MyThreadB();
            Thread t1=new Thread(s1,"A");
            Thread t2=new Thread(s1,"B");
            Thread t3=new Thread(s1,"C");

            t1.start();
            t2.start();
            t3.start();

    }

}
```

1. 스레드의 동기화

<실습> Exam-98.java

한정된 티켓 수량을 멀티 태스킹으로 티켓팅을 했을 때 발생할 수 있는 문제점을
실습을 통해 확인하기.

<실행결과>

```
A가 티켓팅 성공!
B가 티켓팅 성공!
B가 티켓팅 시도 후 티켓수:-1
C가 티켓팅 성공!
C가 티켓팅 시도 후 티켓수:-2
A가 티켓팅 시도 후 티켓수:0
```

티켓 수량이 0 일 때만 티켓팅이 성공해야 하므로 B,C 는 티켓팅이 되면 안되나

```
public void ticketing() {
    if(ticketNumber>0) {
        System.out.println(Thread.currentThread().getName()+"가 티켓팅 성공!");
        ticketNumber--;
```

B와 C도 티켓팅이 성공해, ticketNumber가 음수가 되는 것을 확인 할 수 있다.
이 이유는, 조건문을 걸어뒀으나, A가 티켓팅 한후 ,ticketNubmer을 감소 시키기 전에
B와 C의 티켓팅 코드가 통과했기 때문에 가능한 것을 볼 수 있다.

→ 따라서 이런 문제를 해결 하기 위해,
스레드가 동시에 자원을 접근하는 것을 제한하는
스레드 동기화 작업이 필요!

1. 스레드의 동기화

스레드 동기화(Thread Synchronized) 방법?

1) 동시에 작업하는 메서드에 synchronized 키워드 걸기

```
public synchronized void 메서드명( ){
// 수행할 작업
                                    }
```

2) synchronized 영역 지정하기

```
void 메서드명( ){
synchronized(스레드객체){
// 수행할 작업
                }
                            }
```

1. 스레드의 동기화

<실습> Synch_ShareBank.java

엄마랑 아들이 동시에 돈을 뽑는 작업을 동기화처리하여,
멀티 태스킹의 문제점 해결하기

```
class ATM implements Runnable{
   int money = 10000;
   public void run() {
               for (int i = 0; i < 5; i++) {
                   withdraw(1000);
                   printInfo();
                   try {
                       Thread.sleep(1000);
                   } catch (InterruptedException e) {  }   }   }
    synchronized void withdraw(int money) {
        Thread.yield();
        this.money -= money;
        System.out.println(Thread.currentThread().getName() + "이(가) " + money + "원 출금");
   }
   void printInfo() {
        System.out.println("현재 잔액" + money + "원");
   }}
```

다른 스레드가 접근 못하도록 동기화

```
public class ShareBank {
    public static void main(String[] args) {
공유자원 ATM atm = new ATM();
        Thread mom = new Thread(atm,"엄마");
        Thread son = new Thread(atm,"아들");

        mom.start();
        son.start();

    }
}
```

1. 스레드의 동기화

<실습> Synch_ShareBank.java

엄마랑 아들이 동시에 돈을 뽑는 작업을 동기화 처리하여,
멀티 태스킹의 문제점 해결하기

```
엄마이(가) 1000원 출금
현재 잔액9000원
아들이(가) 1000원 출금
현재 잔액8000원
엄마이(가) 1000원 출금
현재 잔액7000원
아들이(가) 1000원 출금
현재 잔액6000원
엄마이(가) 1000원 출금
현재 잔액5000원
아들이(가) 1000원 출금
현재 잔액4000원
엄마이(가) 1000원 출금
현재 잔액3000원
아들이(가) 1000원 출금
현재 잔액2000원
엄마이(가) 1000원 출금
현재 잔액1000원
아들이(가) 1000원 출금
현재 잔액0원
```

돈을 뽑는 작업을 할 경우,
다른 스레드가 돈을 뽑는 작업(withdraw( )) 를 실행하지 못하므로,
동시에 돈을 뽑아 생기는 문제가 해결되는 것을 확인할 수 있다.

1. 스레드의 동기화

<실습> Exam-98.java_동기화처리

한정된 티켓 수량을 멀티 태스킹으로 티켓팅을 동기화 처리하여
발생할 수 있는 문제점을 해결하기

```
class MyThreadB implements Runnable{
     Ticketing ticket = new Ticketing();
    public void run() {
        ticket.ticketing();
    }
}
class Ticketing{
    int ticketNumber=1;
    public void ticketing() {
        synchronized(this){
        if(ticketNumber>0) {
            System.out.println(Thread.currentThread().getName()+"가 티켓팅 성공!");
            ticketNumber--;
        }
        else {
            System.out.println(Thread.currentThread().getName()+"가 티켓팅 실패!");
        }
        System.out.println(Thread.currentThread().getName()+"가 티켓팅 시도 후 티켓수:"+ticketNumber);
        }
        }
    }
```

```
public class Synchronized1 {
    public static void main(String[] args) {
            MyThreadB s1=new MyThreadB();
            Thread t1=new Thread(s1,"A");
            Thread t2=new Thread(s1,"B");
            Thread t3=new Thread(s1,"C");

            t1.start();
            t2.start();
            t3.start();

    }

}
```

1. 스레드의 동기화

<실습> Exam-98.java_동기화처리

한정된 티켓 수량을 멀티 태스킹으로 티켓팅을 동기화 처리하여
발생할 수 있는 문제점을 해결하기

<실행결과>

```
A가 티켓팅 성공!
A가 티켓팅 시도 후 티켓수:0
C가 티켓팅 실패!
C가 티켓팅 시도 후 티켓수:0
B가 티켓팅 실패!
B가 티켓팅 시도 후 티켓수:0
```

동기화 처리를 통해
B와 C의 티켓팅이 실패되는 것을 확인할 수 있다.

2. wait( ) 와 notify( )

동기화에서 발생할 수 있는 문제 상황을 해결하기 위해 wait( )와 notify( ) 사용 가능!

스레드를 대기시키는 wait( )메서드와 대기중인 스레드를 깨우는 notify( )메서드를 통한 스레드 제어 알아보기!

메서드	설명
void wait( )	notify( )가 호출 될 때까지 대기
void wait(long timeout)	timeout 시간만큼 대기
notify( )	대기 중인 한 스레드만 깨움
notifyAll( )	대기 중인 모든 스레드를 깨움

2. wait( ) 와 notify( )

<실습> Exam-99.java

mom 스레드는 통장 객체에 돈을 입금하고, son 스레드는 통장 객체에 돈을 출금하는 코드

```
class Account{
    int money=0;
    //입금,출금
    public int showMoney() {
        return money;
    }
    public synchronized void setMoney() {
        try {
            Thread.sleep(1000);
        }
        catch(InterruptedException ie) {System.out.println(ie.toString());}
        this.money+=2000;
        System.out.println("어머니가 용돈을 입금했습니다. 현재 잔액: "+this.showMoney());
        this.notify();
    }
    public synchronized void getMoney() {
        while(money<=0) {
        try {
        System.out.println("통장잔고가 없어 아들 대기 중");
        this.wait();}
        catch(InterruptedException ie) {}
    }
        this.money-=2000;
        System.out.println("아들이 용돈을 출금했습니다.현재 잔액: "+this.showMoney());
    }
}
```

```
class Son extends Thread{
    private Account account;
    Son(Account account){this.account =account;}
    public void run() {
        for(int i=0;i<10;i++) {
            account.getMoney();
        }
    }
}

class Mom extends Thread{
    private Account account;
    Mom(Account account){this.account =account;}
    public void run() {
        for(int i=0;i<10;i++) {
            account.setMoney();
        }
    }
}
```

같은 자원을 전달하기 위해
생성자에서 받아온다

2. wait( ) 와 notify( )

<실습> Exam-99.java

mom 스레드는 통장 객체에 돈을 입금하고, son 스레드는 통장 객체에 돈을 출금하는 코드

```
public class Money1 {
    public static void main(String[] args) {
        Account account = new Account();
        Son son=new Son(account);
        Mom mom= new Mom(account);
        son.start();
        mom.start();
    }
}
```

공유자원

```
통장잔고가 없어 아들 대기 중
어머니가 용돈을 입금했습니다. 현재 잔액: 2000
어머니가 용돈을 입금했습니다. 현재 잔액: 4000
어머니가 용돈을 입금했습니다. 현재 잔액: 6000
어머니가 용돈을 입금했습니다. 현재 잔액: 8000
어머니가 용돈을 입금했습니다. 현재 잔액: 10000
어머니가 용돈을 입금했습니다. 현재 잔액: 12000
어머니가 용돈을 입금했습니다. 현재 잔액: 14000
어머니가 용돈을 입금했습니다. 현재 잔액: 16000
어머니가 용돈을 입금했습니다. 현재 잔액: 18000
어머니가 용돈을 입금했습니다. 현재 잔액: 20000
아들이 용돈을 출금했습니다.현재 잔액: 18000
아들이 용돈을 출금했습니다.현재 잔액: 16000
아들이 용돈을 출금했습니다.현재 잔액: 14000
아들이 용돈을 출금했습니다.현재 잔액: 12000
아들이 용돈을 출금했습니다.현재 잔액: 10000
아들이 용돈을 출금했습니다.현재 잔액: 8000
아들이 용돈을 출금했습니다.현재 잔액: 6000
아들이 용돈을 출금했습니다.현재 잔액: 4000
아들이 용돈을 출금했습니다.현재 잔액: 2000
아들이 용돈을 출금했습니다.현재 잔액: 0
```