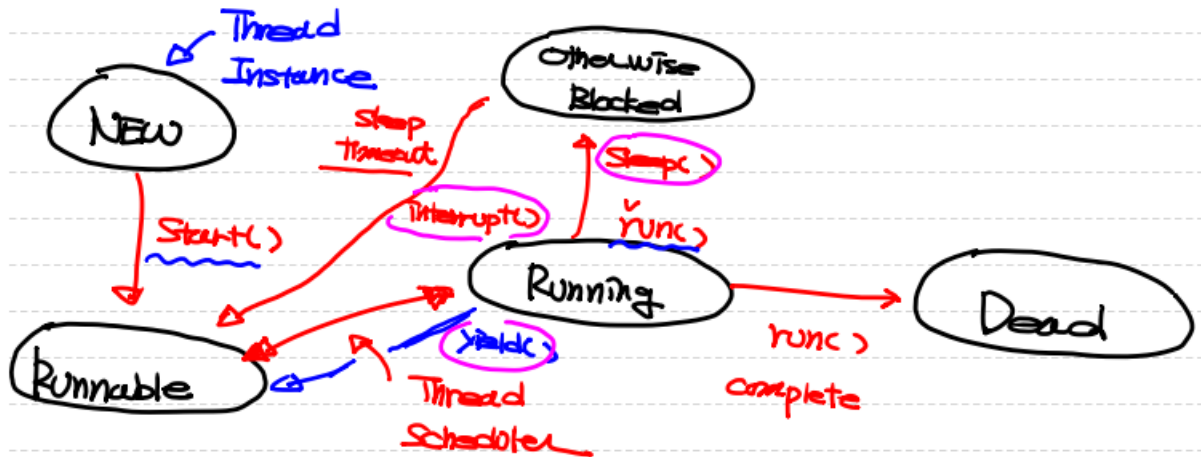




DAY 5



JOIN

- `join()` ⇒ instance method
 - 특정 Thread를 지정해서 내가 수행하고있는 중간에 들어오게 할 수 있다.
- `sleep()` ⇒ static method

`Join()` , `join(초)`

```
package lecture0715;

class ThreadEx_08_01 extends Thread {
    final static int MAX_MEMORY = 1000;
    // 상수 만들때는 모두 대문자, Snake 표기법이 일반적이다.
    int usedMemory = 0;

    @Override
    public void run() {
        while (true) {
            try {
                Thread.sleep(10000); // 5초 동안 잔다.
            } catch (Exception e) {
                System.out.println("interrupt( )에 의해서 깨어났습니다.");
            }
            gc();
            System.out.println("Memory 청소 완료!\n" + "현재 사용가능한 Memory :" + freeMemory());
        }
    }

    private void gc() {
        // 내부에서만 사용하는 logic용 method이기 때문에 private로 쓰는게 좋다.
        usedMemory -= 300;
        if (usedMemory < 0) {
```

```

        usedMemory = 0;
    }
}

public int totalMemory() {
    return MAX_MEMORY;
}

public int freeMemory() {
    return MAX_MEMORY - usedMemory;
}
}

public class ThreadExam08 {
    public static void main(String[] args) {
        ThreadEx_08_01 t1 = new ThreadEx_08_01();
        t1.setDaemon(true);
        t1.start();

        int requiredMemory = 0;
        for (int i = 0; i < 20; i++) {
            requiredMemory = (int) (Math.random() * 10) * 20;
            // Math.random() => 0.0보다 크거나 같고 1.0보다는 작은 랜덤한 숫자가 나온다.
            // Math.random() * 10 => 10을 곱하니까 0~10 사이의 랜덤한 숫자가 나온다.
            // (int)로 type casting했으니까 0~10사이의 정수가 나온다.

            // 필요한 Memory가 사용할 수 있는 양보다 크거나
            // 현재의 전체 Memory양의 60%이상을 사용하고 있을때 gc를 실행

            if (requiredMemory > t1.freeMemory() || t1.freeMemory() < t1.totalMemory() * 0.4) {
                t1.interrupt(); // gc()실행이 끝날때까지 기다리지 않는다.
                try {
                    t1.join(100);
                } catch (Exception e) {
                    // TODO: handle exception
                }
            }
            t1.usedMemory += requiredMemory;
            System.out.println("사용된 Memory : " + t1.usedMemory);
        }
    }
}

```

```

사용된 Memory : 520
사용된 Memory : 520
사용된 Memory : 640
interrupt( )에 의해서 깨어났습니다.
Memory 청소 완료! 현재 사용가능한 Memory :660
사용된 Memory : 520
사용된 Memory : 640
interrupt( )에 의해서 깨어났습니다.
Memory 청소 완료! 현재 사용가능한 Memory :660
사용된 Memory : 460
사용된 Memory : 460
사용된 Memory : 640
interrupt( )에 의해서 깨어났습니다.

```



위의 Code에서 `join()` 을 사용하지 않았을 때는

`if(usedMemory < 0){usedMemory = 0;}` 를 했음에도 사용가능한 Memory 량이 음수가 나온다.
왜 그럴까???



`gc()` method를 이용해서 청소를 하지만 Main Thread는 `gc()` 의 실행이 끝날때까지 기다리지 않는다. 그렇기 때문에 `join()` method를 이용해서 Main Thread를 `blocked` 시키고 t1 Thread를 실행시킨다. 하지만 단순히 `join()` 을 실행하면 `while(true)` 문에서 무한루프를 돌게된다. 그래서 `join(100)` method를 사용함으로써 Main Thread를 blocked시키고 t1 Thread를 1초동안 동작시킴으로써 `gc()` Code를 수행할 수 있게 만든다음 다시 Main Thread를 동작시키는 것이다.

Thread의 동기화 문제

예를 들어 영화예매를 할 때 A,B 두사람이 동시에 P라는 좌석을 예매하려고할때

A라는 사람이 결제를 먼저 완료했으면 늦게 결제한 B는 좌석 할당이 안된다

이렇게 동시에 예매가 수행되고있을때 동기화를 시켜줘야 두 사람이 같은 좌석을 예매하는 오류가 발생하지 않는다.

Critical Section (임계영역)

한 Thread가 이 임계영역을 수행할때는 다른 Thread가 이 임계영역을 수행하지 못합니다

라고 설정해주는것.

LOCK (Monitor)

임계영역을 설정할때 필요하다.

이 공용객체에 대한 LOCK을 가져가면 다른 Thread는 LOCK를 가져갈 수 없으므로 공유객체에 접근하지못한다(blocking).

`sleep` `join` `lock` → thread 일시정지

Java에서 Lock을 얻어서 임계영역을 설정하려면 어떻게 해야할까?



`synchronized keyword` 를 사용한다.

- method 동기화
- 동기화 block을 생성한다.

```

package lecture0715;

// Thread에 의해서 공유되는 공유객체를 생성하기 위한 class
class Account {
    private int balance = 1000; // 계좌 잔액

    public int getBalance() {
        return balance;
    }

    // 출금하는 method = 동기화 시켜야 문제를 해결할 수 있다.
    // synchronized method를 붙여주면
    // 두개의 Thread가 출금 method를 동시에 처리 할 수 없게 만든다.
    public synchronized void withdraw(int money) {
        synchronized (this) { // 어떤 객체를 동기화 시켜줄지 명시해준다
            // 동기화 시켜줄 block을 지정할 수 있다.
            if (balance >= money) {
                try {
                    Thread.sleep(1000);
                    // 잔액을 바로 빼지 않고 자고있어서 자는동안 다른 Thread가 조건을 만족하니까 if문을 타고 들어와서 잔액을 뺀다.
                    // 즉 Logic은 남은 잔액이 출금하려는 잔액보다 커야만 출금을 할 수 있는데 남은 잔액이 -가 찍힐 수 있다.
                    // 두개의 Thread가 동시에 돌게 되면 이런 오류가 발생할 수 있다.

                } catch (Exception e) {
                    // TODO: handle exception
                }
                balance -= money;
            }
        }
    }
}

class ThreadEx_09 implements Runnable {
    Account acc = new Account();

    @Override
    public void run() {
        while (acc.getBalance() > 0) {
            int money = ((int) (Math.random() * 3 + 1) * 100); // 1 ~ 4 사이의 랜덤한 정수가 나온다.
            acc.withdraw(money);
            System.out.println("출금 액: " + money);
            System.out.println("남은 잔액은 : " + acc.getBalance());
        }
    }
}

public class ThreadExam09 {
    public static void main(String[] args) {
        ThreadEx_09 r = new ThreadEx_09(); // Runnable 객체 (공유객체 가지고 있다.)
        Thread t1 = new Thread(r);
        Thread t2 = new Thread(r);

        t1.start();
        t2.start();
    }
}

```

```

출금 액: 300
남은 잔액은 : 700

```

```
출금 액: 200
남은 잔액은 : 500
출금 액: 100
남은 잔액은 : 400
출금 액: 300
남은 잔액은 : 100
출금 액: 300
남은 잔액은 : 100
남은 잔액은 : 100
출금 액: 300
출금 액: 200
남은 잔액은 : 100
남은 잔액은 : 100
출금 액: 300
남은 잔액은 : 100
출금 액: 100
남은 잔액은 : 0
출금 액: 200
남은 잔액은 : 0
```



`if (balance >= money)` 라는 if문을 통해서 현재 잔액이 출금하고자 하는 money보다 많을 때만 출금할 수 있도록 Logic을 설계했는데 실행시켜보면 남은 잔액에 - 가 찍히는 경우를 볼 수 있다. 왜 이런 문제가 발생할까?



`t1 Thread` 가 수행되고 있다고 가정하면 출금을 하기전에 `Thread.sleep(1000);` 이라는 method를 통해서 잔액을 바로 빼지 않고 1초동안 잔다. 근데 `t2 Thread` 가 `t1 Thread` 가 아직 출금을 안했으니까 if문 안에있는 조건을 만족하니까 if문을 타고 들어와서 잔액을 뺀다. 즉, `t1 Thread` 가 출금을 하는순간 남은 잔액이 0원인데 이미 `t2 Thread` 가 if문 안에 들어왔기 때문에 또 출금을 실행해서 남은 잔액이 - 가 찍히게 되는것이다. 두개의 Thread가 동시에 돌게 되면 이런 오류가 발생할 수 있다.

그렇다면 이러한 문제를 해결하기 위해서는 어떻게 해야할까??



이러한 문제를 해결하려면 `synchronized` method를 사용하면된다. `synchronized` 로 `withdraw` 라는 출금 method를 동기화 시키게 되면 `t1 Thread`가 `withdraw` 를 수행할 때 **LOCK**을 가지고 있으므로 `t2 Thread`는 **blocked** 되어서 기다리게 된다. 즉, `t1, t2 Thread`가 `withdraw method` 를 동시에 수행할 수 없게된다.

Synchronized를 이용해서 공유 데이터를 보호


Thread가 공유자원에 대한 **LOCK**을 획득한 후 오랜시간을 보내는 경우가 종종 있다.

→ 비효율적인 program이 된다.

`wait()` → **LOCK**을 놓고 Blocked Object's lock pool로 들어간다.

`notify()` → Blocked Object's lock pool에 있는 Thread가 실행될 수 있도록 block을 풀어준다. 풀게 되면 Running 상태로 가는것이 아니라 Runnable 상태로 간다.

연습문제

 Thread

Java 입출력

- IO (input/ouput)
 - 구현이 쉽다
- NIO (New IO)
 - 구현이 어렵다.

JAVA IO



Stream이라는 객체를 이용해서 입력과 출력을 처리한다.

`Java.io` 라는 **package**안에 묶여서 제공된다.

- 표준 입력
 - keyboard
 - System.in
- 표준 출력
 - monitor
 - System.out

Stream instance



Java에서 **특정 장치에서 data를 읽거나, 보낼때** 사용하는 매개 객체 **Java program**과 **file**은 **Stream**이라는 통로로 연결된다.

Stream 특징

1. **단방향**이다.
 - Stream을 생성할 때 Stream의 **종류,방향**이 결정된다.
 - **Java** → **File** 나갈때 **OuputStream**

- **File** → **Java** 들어올때 **InputStream**
2. **FIFO**(Fiirst In First Out)구조로 되어있다.
 - 먼저 들어간 data가 먼저 나온다.
 3. **결합이 가능하다.**
 - 사용하기 편한 Stream을 만들어서 사용할 수 있다.

ObjcetStream을 통해서 객체도 전달할 수있다.



단, 모든 객체가 다 되는건 아니다.

만약 instance를 생성한 class가 **Serializable interface**를 구현하고있으면 가능하다!!