컴프2 11주차 : 식사하는 철학자들

과제 목표 : 스레드로 프로그래밍하면서 데드락을 해결한다.

학번 : 201902722

이름 : 유형곤

큰 과제 : 데드락 없이 모든 철학자들이 스파게티를 먹어야 한다.

해결 과정의 흐름 : 철학자들은 왼쪽 젓가락을 들고 -> 오른쪽 젓가락을 들고 -> 먹고 -> 왼쪽 젓가락을 내려놓고 -> 오른쪽 젓가락을 내려놓고 -> 생각 하는 순서로 행동하는데, 여기서 데드락은 모든 사람이 왼쪽 젓가락을 들었을 때, 오른쪽 젓가락을 무한정 기다리기 때문에 발생한다. 따라서 여기서는 젓가락을 대기하는 시간을 제한하여 (timeout) 데드락 문제를 해결한다.

작은 과제 : 젓가락 클래스를 구현한다.

```
public class Chopstick {
    int stickNum;
    ReentrantLock lock;

    public Chopstick(int stickNum) {
        this.stickNum = stickNum;
        this.lock = new ReentrantLock();
    }


    public int getStickNum() {
        return stickNum;
    }
}
```

Int stickNum은 각 젓가락의 고유 번호를, ReentrantLock lock은 젓가락이라는 자원을 관리하기 위한 일종의 락(lock)입니다. 그 외에도 생성자와 stickNum을 반환하는 getStickNum()이 있습니다.

작은 과제 : 철학자를 구현한다.

여기서 철학자는 하나의 스레드이므로, Runnable 인터페이스를 구현하여 만들었습니다.

```
public class PhilosopherRunnable implements Runnable {
    String name;
    Chopstick left, right;
    boolean isFull = false;

    public PhilosopherRunnable(String name, Chopstick left, Chopstick right) {
        this.name = name;
        this.left = left;
        this.right = right;
    }

    @Override
    public void run() {
        while(!isFull) {
            try {
                if(getStick(left) && getStick(right)) {
                    eat();
                    returnSticks();
                    think();
                } else {
                    returnSticks();
                }

            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

PhilosopherRunnable은 세 가지 멤버변수가 있습니다.

String name : 철학자의 이름

Chopstick left, right : 각각 왼쪽, 오른쪽 젓가락

boolean isFull : 식사 여부

run() 메서드는 식사에 성공할 때까지 반복문을 도는데, 여기서 getStick(left) && getStick(right)가 참이라는 것은 시간 내에 왼쪽, 오른쪽 젓가락을 모두 드는 데 성공했다는 의미입니다. 만약 실패했다면 모든 젓가락을 내려놓고 다시 시도합니다.

```
public void returnSticks() {
    if(left.lock.isHeldByCurrentThread()) {
        left.lock.unlock();
        System.out.printf("%s가 %d번 젓가락을 놓았습니다!\n", name, left.getStickNum());
    }
    if(right.lock.isHeldByCurrentThread()) {
        right.lock.unlock();
        System.out.printf("%s가 %d번 젓가락을 놓았습니다!\n", name, right.getStickNum());
    }
}

public boolean getStick(Chopstick stick) throws InterruptedException {
    if(stick.lock.tryLock(100, TimeUnit.MILLISECONDS)) {
        System.out.printf("%s가 %d번 젓가락을 들었습니다!\n", name, stick.getStickNum());
        return true;
    } else {
        return false;
    }
}

public void eat() {
    System.out.printf("%s가 스파게티를 먹는중...\n", name);
    isFull = true;
}
```

returnSticks 메서드는 왼쪽, 오른쪽 젓가락을 내려놓습니다.

getStick 메서드는 100ms 안에 젓가락을 드는 경우 true, 그게 아닌 경우 false를 반환합니다.

eat 메서드는 isFull을 true로 변경하며 식사가 끝났음을 확인할 수 있습니다.

```
public void think() {
    try {
        System.out.printf("%s가 생각중...\n", name);
        Thread.sleep((int)(Math.random() * 100));
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Think 메서드는 생각합니다. (0~100ms 사이의 시간을 기다립니다.)

```java
public class Philosophers {
    public static void main(String[] args) {
        Chopstick[] sticks = new Chopstick[5];
        for(int i = 0; i < sticks.length; i++) {
            sticks[i] = new Chopstick(i);
        }

        Thread[] philosophers = new Thread[5];
        for(int i = 0; i < philosophers.length; i++) {
            philosophers[i] = new Thread(new PhilosopherRunnable("철학자 " + (i+1), sticks[i % 5], sticks[(i+1) % 5]));
            philosophers[i].start();
        }
    }
}
```

메인 함수에서는 각각의 젓가락과 철학자를 생성하고 철학자를 실행합니다.