```java
@SneakyThrows
@Override
public void run() {
    for (;;) {
        try {
            lock.lock();

            ThreadWorker.test += 1;
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }

        Thread.sleep( millis: 3000);
    }
}
```

Run매소드 실행

For(;;) → 무한루프 안에

Lock 걸었음.

그 안에 test+1했음

Try ~ catch를 통해 예외 처리

Unlock 했음.

그리고 3초 홀드 후 → 다시 for문 시작부터 실행

무한루프 !!

그 중에 get~ 매소드를 호출했는데, 만약 lock상태라면?

Unlock 이후 그 test 값을 호출 할 수 있게 한다.

(→ThreadWorker.test를 한 이유 : 전역변수를 처리할 때 명시적으로 표현하기 위해)

Thread.sleep이 try문 안에 있는 것과 밖에 있는 것에 대한 차이를 잘 모르겠습니다.

Try 밖에 있는 Thread.sleep구현

1. 무한루프 시작 → lock 걸고 test+1한다.
2. 예외처리 후 unlock으로 lock을 풀어준다.
3. 3초 대기한다.
4. 다시 lock걸고 test+1 ~~~ 반복

Try 안에 있는 Thread.sleep구현

1. 무한루프 시작 → lock걸고 test+1한다.
2. 3초 대기
3. 예외 처리 후 unlock으로 lock을 풀어준다.
4. 다시 lock걸고 test+1 ~~~ 반복

이렇게만 따지면 큰 차이는 없어 보입니다. try안에 Thread.sleep가 있다면 lock이 풀리자마자 다시 lock에 걸려서 값이 나오지 못하고, 무한 버퍼링이 되는걸까요?

내가 바꾼 매소드 (문제은행 7, 2번) _ 질문

```java
package com.example.demo;

import com.example.demo.utility.thread.ThreadManager;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        ThreadManager tm = new ThreadManager();
        tm.start();

        SpringApplication.run(DemoApplication.class, args);
    }

}
```

변경 한 부분 (메인)

메인에서 ThreadManager 생성,

Start() → 구동

```java
@RequestMapping("/homework2")
public String homework2(Model model) throws InterruptedException {
    logger.info("client entered/homework2");


    model.addAttribute( attributeName: "test", ThreadManager.getTest() );

    return "25thHW/homework2";
}
```

변경 한 부분 (controller 클래스)

ThreadManager의 getTest매소드 호출

```java
public ThreadManager()  {
    test = 0;
    lock = new ReentrantLock();
}

public static int getTest(){
    return test;
}


public void plusOne() throws InterruptedException {
    boolean addOne = true;
    while (addOne){
        test++;
        Thread.sleep( millis: 3000);
        if(test>100){
            addOne = false;
        }
    }
}

@Override
public void run() {
    // TODO Auto-generated method stub
    try {
        lock.lock();
        plusOne();
        } catch (InterruptedException ex) {
        ex.printStackTrace();
    }finally {
        lock.unlock();
```

변경 한 부분 (ThreadManager 클래스)

해당 클래스에서는 getTest부분만 추가했을 뿐인데 구동이 문제없이 되고 있습니다.

앞서 선생님께서 lock안에 Thread.sleep을 넣게 되면 test가 100이 될때까지 계속 버퍼링이 돌게 될 것이라고 말씀 주셨는데

어떻게 제대로 구현이 되는지 잘 모르겠습니다...!