2022.1.25 (26일차) 문제은행 7 풀이

화면에 주사위(랜덤) 출력해보자.

선생님 답안

Controller (package) - Bank7Dash26thController (class)

```java
@Controller
public class Bank7Dash26thController {

    private final static Logger logger =
            LoggerFactory.getLogger(Bank7Dash26thController.class);

    @RequestMapping("/homework1")
    public String bank7Homework1 (Model model) {
        Dice dice = new Dice();
        dice.rollDice();

        model.addAttribute( attributeName: "diceNum", dice.getDiceNum());

        return "26th/homework1";
    }
```

Dice 객체 생성해서 여기서 getDiceNum 매소드를 가지고 왔다. 이것도 되는구나!!

```java
package com.example.demo.utility;

import lombok.Data;

// @Getter @Setter @RequiredArgsConstructor @ToString @EqualsAndHashCode
// @Getter: 자동 게터 만들어줌
// @Setter: 자동 세터 만들어줌
// @특정 입력이 들어오는 생성자 처리
// @toString 안만들어도 알아서 자동화
// @EqualsAndHashCode: 현재로서 신경 안써도 됨
@Data
public class Dice {
    private int diceNum;
    private final int MAX = 6;
    private final int MIN = 1;
    private final int range = MAX - MIN + 1;

    public void rollDice () { diceNum = (int) (Math.random() * range + MIN); }
}
```

처음배운 것 : @Data를 작성하면 getter,setter,toString등 쓸수있데!! 그렇다면 굳이 get매소드 안써도 된다는 뜻.

실제로도 rollDice 매소드를 만들어서, 그냥 diceNum의 값을 구했는데 → 위에 dice.getDiceNum()을 호출했고,

그 값을 문제없이 호출하고 있게 만들어줌.

내가 만든 코드를 보여줄께

```java
@Controller
public class Homework25th extends ThreadManager {

    private static final Logger logger = LoggerFactory.getLogger(Homework25th.class);

    @RequestMapping("/homework1")
    public String homework1(Model model){
        logger.info("client entered/homework1");

        final int MAX = 6;
        final int MIN = 1;
        int range = MAX - MIN +1;
        int random = (int)(Math.random()*range + MIN);

        model.addAttribute( attributeName: "random",random);

        return "25thHW/homework1";
    }
```

아쉬운점 : 클래스 사용 안하고 dice 사용 → 클래스로 썼으면 더 좋았을것.

별도 유틸리티(패키지)를 만들어서 처리 가능한 클래스 만들어서 컨트롤 함

대망의 두번째 문제 :

1초마다 1씩 증가하는 값을 만든다. 그리고 새로고침 할때마다 증가하는 그 값을 출력하도록 한다.(??)

```java
package com.example.demo;

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

/*
기본적으로 우리가 만드는 서비스라는 개념은 반드시 두 가지 개념이 존재함
서버(Server)와 고객(Client) [ 공급자와 수요자 ]
우리가 구동시킨 Spring Boot 서버 (Server)는 말 그대로 Server에 해당함
우리가 만든 Controller에 URL을 요청하는 모든 사용자들
 */
@SpringBootApplication
public class DemoApplication {

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

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

}
```

선생님이 만든 클래스

그 중 메인 클래스로 들어와봤다.

여기서 ThreadWorker의 객체를 생성해서

Start를 시킴

main으로 빼는 이유는? url접속할때마다 객체가 생성이 되기 때문에 test값이 초기화가 된다.

근데 메인을 접근하는 방법을 알아야한다.

접근하기 위해서는, get매소드를 static으로 바꿔야해? 그럼 lock도 static으로 바꿔줘야함.

뒤에 있는 내용으로 보면 더 이해가 갈 듯.

우선 main 스프링을 봤다면, controller에서 어떻게 작성했는지 확인 할 차례

```
@RequestMapping("/homework2")
public String bank7Homework2 (Model model) {
    logger.info("homework2");

    model.addAttribute( attributeName: "threadNum", ThreadWorker.getSyncLockTest());

    return "26th/homework2";
}
```

Homework2를 웹에 검색하면 ThreaWorker 클래스의 get~ 매소드 값이 나올 수 있도록 한다.

그럼, ThreadWorker의 클래스를 보러 가봐야겠다. 저 get의 값이 어떻게 나오는지 보려면!

ThreadWorker 클래스

```
package com.example.demo.utility.thread;

import ...

public class ThreadWorker extends Thread {
    private static int test;

    private static Lock lock;

    public ThreadWorker () {
        ThreadWorker.test = 0;
        lock = new ReentrantLock();
    }

    public static int getSyncLockTest () {
        int tmp;

        lock.lock();
        tmp = ThreadWorker.test;
        lock.unlock();

        return tmp;
    }

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

여기서 봐야할 것.

ThreadWorker의 매소드에 extends Thread를 처리하여,

꼭 run 매소드를 사용해야함.

Run에서 구동할 코드를 구현해야한다.

그리고 lock을 하는 이유:

Test값이 static이다. 만약 test 값이 lock이 되지 않는다?

+1을 하는 도중에 새로고침하면? 그건 어떤 값으로 나와야하는거지?? 이러한 이유 때문에 lock을 거는 것임.

메인도 쓰레드 워커를 동작이 가능하다.

스프링 컨드롤러도 쓰레드로 동작 가능하다.

그래서 test가 크리티컬 색션이 되어서 lock를 쳐야한다.

그리고 get~매소드에서 lock은 없어도됨!

이유 : get은 그냥 값을 도출하는 매소드이기 때문에

갱신하는 상황이라는건 +=를 사용하는 상황인건가요?

-> 맞습니다. 연산하는게 아니라면 굳이 lock을 할 필요가 없음

연산이 값을 바꾸는 작업이 아니라서 lock이 없어도 구동하는데 문제가 없음.

읽어오는데는 lock이 필요가 없음 값이 바뀌는 것이 아니기 때문에.

```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?

그럼 내가 만든 엉망진창 메소드 보러가자

```
    @RequestMapping("/homework2")
    public String homework2(Model model) throws InterruptedException {
        logger.info("client entered/homework2");
        // plusOne(); & run() & start() 모두 실행했으나 웹 버퍼링.. 이유가 뭘까
        run();

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

        return "25thHW/homework2";
    }

}
```

우선 run을 여기서 구동하는것도 문제고..

(run -또는 start는 메인에서 생성자 호출한다. 앞전에 얘기 했던 것 처럼 생성자에서 호출하지 않게 된다면,

test의 값이 계속 0으로 나오게 될 것이다! 아! 그래서 run할 때 0이 계속 나왔던 거구나...!)

사실 선생님이 말한 정말 이상하다고 한 것은 이부분이다.

```
@Controller
public class Homework25th extends ThreadManager {
```

Controller가 이미 쓰레드를 가지고 있는 것이라고 한다. 그래서 extends Thread~ 이렇게 사용한다는 것은 매우 이상하다는 것.!! 주의하기

```
package com.example.demo.controller25thHW;

import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class ThreadManager extends Thread {

    static int test;
    Lock lock;

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

    public static int plusOne() throws InterruptedException {
        boolean addOne = true;
        while (addOne){
            test++;
            Thread.sleep( millis: 3000);
            if(test>100){
                addOne = false;
            }
            System.out.println(test);
        }
        return test;
    }
```

내가 만든거

Extends Thread처리 → lock / Thread.sleep사용하기 위해

생성자에서 초기화 완료

선생님 말씀으로는,

lock을 걸어두고 자니까 - 요청을했을때 받아들여지지 않는다.

lock을 풀지 않고 5분동안 아무것도 못하게 되는것임.

버퍼링이 발생하는거임.

?? 근데 음.. 이건 밑에서 다시 설명하겠음.

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

Run 매소드에서

Lock 실행

plusOne 매소드 통해 test+1 & thread.sleep을 하게 한다.

Unlock 실행

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...!