2022.1.20 복습(23일차) 문제은행[6]

문제1. 2*2 사각형 넓이 구하기 (가로를 0.001로 나눠, 세로와 곱한 후 총 합산 구하기 ->4에 근접)

```
public class Bank6Prob1 {
    public static void main(String[] args) throws InterruptedException {

        ThreadRectangle.calcEachThreadTotal();

        ThreadRectangle[] rect = new ThreadRectangle[ThreadRectangle.THREAD_MAX];

        // 스레드 준비
        for (int i = 0; i < ThreadRectangle.THREAD_MAX; i++)
        {
            rect[i] = new ThreadRectangle();
        }

        // 스레드 구동
        for (int i = 0; i < ThreadRectangle.THREAD_MAX; i++)
        {
            rect[i].run();
            rect[i].join();
        }

        // test 구동 (컴퓨터 세계의 오차의 모순을 확인)
        //rect[0].run();
        System.out.printf("%d개의 스레드가 모든 작업을 완료하였습니다.", ThreadRectangle.THREAD_MAX);
```

```
        float finalResult = 0;

        for (int i = 0; i < ThreadRectangle.THREAD_MAX; i++) {
            finalResult += rect[i].getSum();
        }

        System.out.println("최종 결과는 ? " + finalResult);
    }
}
```

1. ThreadRectangle 클래스의 calcEachThreadTotal() 매소드를 호출했다. 해당 매소드는 static으로 전역매소드라 이렇게 호출하게 되면 각 real total, total값을 알 수 있게 된다.
2. ThreadRectangle 클래스를 객체배열로 만들었다. (개수는 ThreadRectangle.THREAD_MAX → 이것도 static 이여서 전역변수 사용 가능하다) → 해당 클래스를 THREAD_MAX 개를 객체로 만들었다.
3. 각 배열 값을 실행하기 위해 for문 통해(THREAD_MAX만큼 반복) 아래와 같이

   rect[i] = new ThreadRectangle(); → 각 배열 값에 ThreadRectangle생성자를 생성해줬다.
4. 이제 쓰레드 구동할 차례이다. Run과 join을 함께 써서 → run: 실행 / join : hold 상태로 만들어 놓음.

   Join → hold로 만든 이유는. 해당 클래스의 값이 끝날때까지 없어지지 않기 위해.

   사실 저것도 run으로 쓰면 안되고, start로 써줘야한다.

   그 이유는? 밑으로 가보자.

run 매소드와 start매소드 약간 다른데, run과 start는 무슨 차이이지??

run()으로 구동되면 Thread가 관장하는 제어권에서 제어되지 않음(순차적 값 출력)

start()로 구동하면 Thread 상에서 Context Switching을 통한 경쟁이 활성화됨(뒤죽박죽)

솔직히, Thread썼는데 Context Switching 안한다? 그럼 왜써? 성능이 느려지는데?

디버깅할때는 run을 해야한데 -> 값이 문제없이 잘 나오냐?의 값을 보기 위해

start -> 에 문제가 생겼다?? critical section접근에 문제가 생겼다는 것.

run -> 문제가 생겼다? 코드가 잘못 만들어짐

새부 클래스 보도록 하자

```
public class ThreadRectangle extends Thread {
    final int Y = 2;
    final static int X = 2;
    // 사각형은 덜 쪼갤수록 정밀도가 올라가는 현상이 발생함 (float이나 double의 오차 때문에 그럼)
    final static float dx = 0.0001f;
    private int xStart, xEnd;
    static int threadCnt = 0;
    private int localThreadId;

    final static int THREAD_MAX = 4;
    static int total;

    float sum;
```

주의 : extends Thread 함.

변수 선언 /

```
    public ThreadRectangle () {
        localThreadId = threadCnt++;
        xStart = 0 + total * localThreadId;
        xEnd = total * (localThreadId + 1) - 1;

        sum = 0;

        System.out.printf("xStart = %4d, xEnd = %4d, thread ID = %d\n",
                xStart, xEnd, localThreadId);
    }

    public static void calcEachThreadTotal () {
        // ceil() 천정 함수: 즉 올림
        int realTotal = (int) (Math.ceil(X / dx));
        System.out.println("realTotal = " + realTotal);

        total = realTotal / THREAD_MAX;
        System.out.println("total = " + total);
    }

    public float getSum() {
        return sum;
    }
```

해당 클래스 생성사에서 값 초기화 해주기.

여기서 xStart, xEnd가 나오는 부분이 제일 중요함.

어떤 학생 질문 : 그런데 스레드를 왜 4개로 분할해주나요?

-> 여러개 분할해서 써서 일을 더 빠르게 쓰려고

쓰레드를 쓰는 이유는 이것 뿐이다. 만약 이게 아니라면 굳이 쓰레드를 쓸 필요가 없다.

그래서 4개를 나눠서 일을 빠르게 만들려고 하는 것임.

그래서 생성자 + calcEachThreadTotal 매서드는 현재

가로 길이를 0.001로 나눈 길이 & 4개로 쪼갠 값을 체크하고 있다. 이번 수업에서 제일 어려웠고, 생각하기 힘든 것.

```java
@Override
public void run() {
    // 자기 구간을 계산하도록 한다.
    /*
    1-3. 이 아주 작은 밑변과 높이값인 2를 곱해 작은 면적값 파악
    1-4. 각각의 모든 작은 사각형들을 합산하여 4에 근접하는지 파악
     */
    float curX = xStart;
    float tmp = 0;

    for (int i = xStart; i <= xEnd; i++) {
        tmp = dx * Y;
        //System.out.printf("tmp = %.12f\n", tmp);
        sum += tmp;
        //System.out.printf("sum = %.12f\n", sum);
    }

    //System.out.printf("tmp * 500 = %.12f\n", tmp * 500);
    System.out.printf("sum = %.12f\n", sum);
}
```

Extends Thread 사용으로 override run 구동이 가능하다.

해당 매소드로 인해 메인 클래스에 start & join이 가능

해당 매소드에서 진행 되는 실제 값을 출력한다. (sum)

curX는 현재 x값(xStart – 4개로 쪼갰기 때문에 각 4개의 시작 값은 분명 다를 것이다.)

그래서 for문에서 시작을 xStart / 끝을 xEnd로 해야함.

0.001(가로) * 세로(2) = tmp

sum +=tmp → 총 합이 된다.

질문2 y = x^2공식(동일하게 x = 0.001로 잘게 나누기...! 0~3 해서 9에 근접한 값을 구해라!)

```java
public class Bank6Prob2 {
    public static void main(String[] args) throws InterruptedException {
        ThreadQuadraticEquation.calcEachThreadTotal(0, 3);

        ThreadQuadraticEquation[] rect = new ThreadQuadraticEquation[ThreadQuadraticEquation.THREAD_MAX];

        // 스레드 준비
        for (int i = 0; i < ThreadQuadraticEquation.THREAD_MAX; i++)
        {
            rect[i] = new ThreadQuadraticEquation();
        }

        // 스레드 구동
        for (int i = 0; i < ThreadQuadraticEquation.THREAD_MAX; i++)
        {
            rect[i].start();
        }

        // join()은 Thread가 끝날때까지 대기하는거라
        // 루프 돌면서 start()와 join()이 묶여 있으면
        // 사실상 순차 구동이라 봐도 무방함
        for (int i = 0; i < ThreadQuadraticEquation.THREAD_MAX; i++) {
            rect[i].join();
        }

        System.out.printf("%d개의 스레드가 모든 작업을 완료하였습니다.", ThreadQuadraticEquation.THREAD_MAX);
```

```java
        System.out.printf("%d개의 스레드가 모든 작업을 완료하였습니다.", ThreadQuadraticEquation.THREAD_MAX);

        float finalResult = 0;

        for (int i = 0; i < ThreadRectangle.THREAD_MAX; i++) {
            finalResult += rect[i].getSum();
        }

        System.out.println("최종 결과는 ? " + finalResult);
    }
}
```

1. ThreadQuadraticEquation.calcEachThreadTotal(0, 3);으로 static 매소드이기에 값을 호출했다.

   0,3으로 각각의 값을 입력해서 (0:시작값 3:마지막값을 도출하게 한다.)

2. ThreadQuadraticEquation를 객체배열로 만들었다.
3. for문으로 rect 각 배열에 ThreadQuadraticEquation 생성자 불러옴.
4. Start로 시작해놓고, join으로 main이 끝날때까지 hold 해 놓았다.
5. For문 통해 rect[].getSum()을 +해준다. 그럼 총 합을 구할 수 있음.

세부 클래스로 가보자.

```java
public class ThreadQuadraticEquation extends Thread {
    final static float dx = 0.0001f; //   0.0019 부족 오차
    private int xStart, xEnd;
    static int threadCnt = 0;
    private int localThreadId;

    final static int THREAD_MAX = 4;
    static int total;

    float sum;

    public ThreadQuadraticEquation () {
        localThreadId = threadCnt++;
        xStart = 0 + total * localThreadId;
        xEnd = total * (localThreadId + 1) - 1;

        sum = 0;

        System.out.printf("xStart = %5d, xEnd = %5d, thread ID = %d\n",
                xStart, xEnd, localThreadId);
    }
```

이번엔 y = x^2하는 값을 도출한다.

생성자 + calcEachThreadTotal은 아까 2*2의 클래스와 비슷하다.

그 이유는 동일하게 total 값을 4개로 나눠서 그걸 Thread를 사용하여 업치락뒤치락하면서 값을 구하도록 하기 때문이다.

(빠른 결과값 도출 가능)

```java
    public static void calcEachThreadTotal (int start, int end) {
        // ceil() 천정 함수: 즉 올림
        int realTotal = (int) (Math.ceil((end - start) / dx));
        System.out.println("realTotal = " + realTotal);

        total = realTotal / THREAD_MAX;
        System.out.println("total = " + total);
    }

    public float getSum() { return sum; }
```

여기엔 start, end를 각각 메인클래스에서 받아왔다. (0,3)

0: 시작값 3 : 끝나는 값 → realTotal에서 end-start / dx → 3000

Math.ceil 을 하는 이유는 현재 값이 2999.9999가 나오기 때문

아무튼 이전과 동일한 수식으로 total값을 만들어줬음

(값이 중복되는 것은 상속을 통해 해결 가능 → 내일 할 예정)

```java
    @Override
    public void run() {
        float curX = dx * xStart;

        // y = x^2  높이값 curX * curX
        // 사각형의 넓이는 dx * y = dx * curX * curX
        for (int i = xStart; i <= xEnd; i++, curX += dx) {
            // 0.0001 = dx
            // curX = 2.5677
            // curX = 2.5677
            // 소수점 계산 특성: 소수점 4번째 자리숫자와 소수점 4번째 자리 숫자의 연산은 무엇을 만드나 ?
            // 소수점 8번째 자리의 결과를 만들어내게 됨
            // 다시 거기에 0.0001을 곱하니 소수점이 더 뒤로 밀려가게 되고
            // 데이터타입이 표시할 수 없는 소수점을 만들게 될 가능성이 높아짐
            // 이러한 사항이 한도를 넘어가게 되면 dx = 0.000001f과 같이
            // 납득하기 어려운 오차를 유발할 수 있다.
            sum += dx * curX * curX;

            System.out.printf("Thread ID = %d, sum = %.12f\n", localThreadId, sum);
        }

        System.out.printf("sum = %.12f\n", sum);
    }
```

Run은 동일하게 Thread가 extends하고 있기 때문에 실행 가능

이 계산법에서는 봐야할 것이 있음.

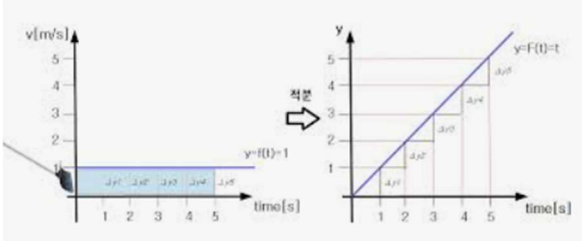
나는 y = x^2를 선으로 생각해서 그냥 x제곱만 하면되겠다!

하고 값을 실행했음. **하지만 선생님이 원하는 것은 이거였음.**

X * y = 작은 네모를 구한 후 합하기

X(0.001) * y(x^2이기 때문에 (xStart*0.001)^2) = 값.

Sum += 값.



질문들

double로는 왜 안쓰나요? 더 정확한데 메모리가 낭비되서인가요?

->속도가 느려서, 의료장비 아니면 안쓴다.

쓰레드 값이 섞기고 싶다면(이게 바로 쓰레드를 하는 이유이기 때문에 이렇게 하는게 맞다.)

값을 분산시키고(선생님이 4개로 값 나눈것처럼) -> 값을 start 시키면 각각 값이 나눠서 나올 수 있겠다.!!!

와우 매우 천재같음.

선생님 RUN으로 하면 그냥 순차적 실행인가요?아니면 CPU하나만 쓰는 건가요?

--> 맞다.

sum -> static 으로 만들어서 각 thread에서 값을 중첩하지 않도록 하는것이 -> lock을 만들 수 있는 방법이다.

static -> critical section이 된다.

--> lock이 필요하다.

lock을 쓴다 ? 할일이 없어진다. -> 성능이 떨어진다. 그러니 critical section이 있고, 값이 변경 될 가능성이 있을 때만 사용하라고~