

< 24th_ThreadRefactor >

[Bank6ProbIntegration (main)]

```
public class Bank6ProbIntegration {  
    public static void main(String[] args) throws InterruptedException {  
  
        ThreadManager tm1 = new ThreadManager( threadNum: 6, parameter1: 2, parameter2: 2, ThreadCalculation.SQUARE);  
        System.out.println("계산된 값 = " + tm1.calcArea());  
  
        ThreadManager tm2 = new ThreadManager( threadNum: 4, parameter1: 0, parameter2: 3, ThreadCalculation.QUADRATIC);  
        System.out.println("계산된 값 = " + tm2.calcArea());  
    }  
}
```

1. main 클래스에서 ThreadManager 타입의 객체 tm1과 tm2의 생성자를 호출하며 threadNum, parameter1, parameter2, 서비스 코드(SQUARE or QUADRATIC) 값을 매개변수로 넘겨준다.

- ThreadManager 클래스의 필드

```
public class ThreadManager {  
    private ThreadRectangle[] tRect;  
    // BlaBla[] blabla;  
    private ThreadQuadraticEquation[] tqe;  
  
    private int threadNum;  
    private int serviceCode;
```

serviceCode가 SQUARE일 경우 분할된 스레드를 처리할 변수 tRect
serviceCode가 QUADRATIC일 경우 분할된 스레드를 처리할 변수 tqe
연산 작업을 몇 개의 스레드로 나눠 수행할 것인지 결정하는 변수 threadNum

- ThreadManager 생성자

```
public ThreadManager (int threadNum, int parameter1,  
                      int parameter2, int serviceCode) {  
  
    this.threadNum = threadNum;  
    this.serviceCode = serviceCode;  
  
    checkService(threadNum);  
    serviceAlloc(parameter1, parameter2);  
}
```

main으로부터 threadNum, parameter, serviceCode를 받아온다.

main으로부터 받아온 threadNum과 serviceCode를 ThreadManager 클래스의 변수에 대입한다.

threadNum을 매개변수로 넘겨주며 checkService 메서드를 호출한다.

parameter1과 parameter2를 매개변수로 넘겨주며 serviceAlloc 메서드를 호출한다.

- checkService

```
public void checkService (int threadNum) {  
    switch (serviceCode) {  
        case ThreadCalculation.SQUARE:  
            tRect = new ThreadRectangle[threadNum];  
            break;  
  
        case ThreadCalculation.QUADRATIC:  
            tqe = new ThreadQuadraticEquation[threadNum];  
            break;  
    }  
}
```

switch문을 통해 serviceCode에 해당하는 case를 수행한다.

[ThreadCalculation 클래스]

```
public class ThreadCalculation {  
    static final int SQUARE = 0;  
    static final int QUADRATIC = 1;  
}
```

이 클래스의 변수 SQUARE와 QUADRATIC은 static으로 선언된 전역변수

1) 서비스 코드 SQUARE = 0

```
case ThreadCalculation.SQUARE:
    tRect = new ThreadRectangle[threadNum];
    break;
```

threadNum만큼의 크기를 가지는 ThreadRectangle 타입의 배열 tRect를 생성한다.

2) 서비스 코드 QUADRATIC =1

```
case ThreadCalculation.QUADRATIC:
    tqe = new ThreadQuadraticEquation[threadNum];
    break;
```

threadNum만큼의 크기를 가지는 ThreadQuadraticEquation 타입의 배열 tqe를 생성한다.

- serviceAlloc

```
public void serviceAlloc (int parameter1, int parameter2) {
    switch (serviceCode) {
```

parameter1, parameter2를 받아온다.

지정된 서비스코드에 해당하는 case를 수행한다.

1) 서비스 코드 SQUARE = 0

```
case ThreadCalculation.SQUARE:
    ThreadCommon.threadNum = threadNum;
    ThreadCommon.calcRealTotal(parameter1);
    ThreadCommon.threadCnt = 0;

    for (int i = 0; i < threadNum; i++) {
        tRect[i] = new ThreadRectangle(threadNum, parameter1, parameter2);
    }
    break;
```

Threadcommon 클래스의 threadNum 변수에 현재 클래스의 threadNum(6)을 대입한다.

parameter1을 넘겨주며 Threadcommon 클래스의 calcRealTotal 메소드를 호출한다.

Threadcommon 클래스의 threadCnt 변수를 0으로 초기화한다.

!) SQUARE의 경우 parameter1 = 밑변, parameter2 = 높이

2) 서비스 코드 QUADRATIC =1

```
case ThreadCalculation.QUADRATIC:
    ThreadCommon.threadNum = threadNum;
    ThreadCommon.calcRealTotal(x parameter2 - parameter1);
    ThreadCommon.threadCnt = 0;

    for (int i = 0; i < threadNum; i++) {
        tqe[i] = new ThreadQuadraticEquation(threadNum, parameter1, parameter2);
    }
    break;
```

Threadcommon 클래스의 threadNum 변수에 현재 클래스의 threadNum을 대입한다.
x축의 길이(parameter2 - parameter1)를 넘겨주며 Threadcommon 클래스의
calcRealTotal 메소드를 호출한다.

Threadcommon 클래스의 threadCnt 변수를 0으로 초기화한다.

!) QUADRATIC 경우 parameter1 = 시작구간, parameter2 = 종료구간

- ThreadCommon 클래스의 calcRealTotal 메서드

```
public static void calcRealTotal (int x) {
    realTotal = (int) (Math.ceil(x / dx));
    System.out.println("realTotal = " + realTotal);

    totalRemain = realTotal % threadNum;
    xEnd = new int[threadNum];
    xStart = new int[threadNum];

    for (int i = 0; i < threadNum; i++) {
        xStart[i] = 0;
        xEnd[i] = 0;
    }
}
```

static으로 선언되어 다른 클래스에서 호출 가능, 호출될 때 x값을 받아온다.

ThreadManager클래스의 serviceAlloc 메서드를 통해 지정된 threadNum(6, 4)

받아온 x값(2 / 3)을 float형 전역변수 dx(0.001)로 나눠 realTotal에 대입 (2000 , 3000)

(x를 0.001로 나눴을 때 몇 개로 쪼개지는지)

realTotal(2000, 3000)과 threadNum(6)의 나머지 연산 수행해 totalRemain에 대입(2, 0)

!) critical section을 해결하기 위해 스레드 개수만큼의 배열을 만들어 영향을 받지 않게 한다.

: threadNum(6,4)의 크기를 가지는 int형 배열 xEnd와 xStart 생성

for문을 통해 ThreadNum만큼 반복을 수행

반복문 내에서는 Xstart배열과 xEnd배열의 각 인덱스 값을 0으로 초기화한다.

for문을 통해 threadNum만큼 반복을 수행한다. // 스레드 객체를 생성하는 루프

1) 서비스 코드 SQUARE = 0

```
for (int i = 0; i < threadNum; i++) {  
    tRect[i] = new ThreadRectangle(threadNum, parameter1, parameter2);  
}
```

2) 서비스 코드 QUADRATIC =1

```
for (int i = 0; i < threadNum; i++) {  
    tqe[i] = new ThreadQuadraticEquation(threadNum, parameter1, parameter2);  
}
```

===== Loop =====

루프마다 threadNum, parameter1, parameter2를 넘겨주며
서비스 코드에 해당하는 배열의 각 인덱스에 객체를 생성한다.

- ThreadRectangle 클래스의 생성자 // extends ThreadCommon

```
public ThreadRectangle (int threadNum, int x, int y) { super(x, y); }
```

받아온 x,y (2,2) 값을 전달하며 부모 클래스인 ThreadCommon의 생성자를 호출한다.

- ThreadQuadraticEquation 클래스의 생성자 // extends ThreadCommon

```
public ThreadQuadraticEquation(int threadNum, int x, int y) { super(x, y); }
```

- ThreadCommon 클래스의 생성자

```
public ThreadCommon(int x, int y) {  
    this.x = x;  
    this.y = y;  
  
    // 1. realTotal값을 통해서 threadNum으로 나눴을때 몫이 얼마가 나오는지 판정한다.  
    // 2. 나머지값을 판정한다.  
    // 3. 각 threadId 들에게 나머지가 0이 될 때까지 1개씩 나눠준다.  
    calcEachThreadTotal();  
  
    localThreadId = threadCnt++;  
}
```

x와 y를 받아와 ThreadCommon 클래스의 x, y 에 대입한다.

calcEachThreadTotal 메서드를 호출한다.

localThreadId에 threadCnt 값을 대입하고, threadCnt를 1증가시킨다.

(분할된 각각의 스레드에 Id 부여하기 위해)

- calcEachThreadTotal

```
public void calcEachThreadTotal () {  
    total = realTotal / threadNum;  
  
    if (totalRemain-- > 0) {  
        total++;  
    }  
  
    System.out.println("total = " + total);  
}
```

realTotal(2000, 3000)을 threadNum(6, 4)으로 나눈 몫(333, 750)을 total에 대입한다.
totalRemain = 2 , 0
if문을 통해 현재 totalRemain이 0보다 큰지 비교하고 totalRemain값을 1 감소시킨다.
위의 조건에 해당하는 경우 total을 1증가시킨다.
(나머지를 계산 범위에 포함시키기 위해서)

```
if (localThreadId == 0) {  
    xStart[localThreadId] = 0;  
    xEnd[localThreadId] = total - 1;  
} else {  
  
    xStart[localThreadId] = xEnd[localThreadId - 1] + 1;  
    xEnd[localThreadId] = xEnd[localThreadId - 1] + total;  
}  
  
sum = 0;  
  
System.out.printf("xStart = %4d, xEnd = %4d, thread ID = %d\n",  
    xStart[localThreadId], xEnd[localThreadId], localThreadId);
```

if) localThreadId가 0인 경우
xStart배열의 localThreadId 인덱스(0)의 값 = 0
xEnd배열의 localThreadId 인덱스(0)의 값 = total(334) - 1 = 333
else)
xStart배열의 localThreadId 인덱스(1)의 값 = xEnd배열의 (localThreadId - 1) 인덱스(0)의 값 + 1
xEnd배열의 localThreadId 인덱스(1)의 값 = xEnd배열의 (localThreadId - 1) 인덱스의 값 + total

	[0]	[1]	[2]	[3]	[4]	[5]
total	334	334	333	333	333	333
xStart	0	334	668	1001	1334	1667
xEnd	333	667	1000	1333	1666	1999

sum을 0으로 초기화한다.

===== Loop 종료=====

2. main 클래스에서 tm객체들의 calcArea 메서드 호출

- ThreadManager 클래스의 calcArea 메서드

```
public float calcArea () throws InterruptedException {  
    eachThreadStartWork();  
    eachThreadWaitFinish();  
    return sumEachThreadResult();  
}
```

eachThreadStartWork 메서드를 호출해 각 스레드의 연산작업을 수행한다.

eachThreadWaitFinish 메서드를 호출해 스레드들의 작업이 끝날 때까지 대기한다.

sumEachThreadResult 메서드를 호출하고 메서드의 결과값을 main클래스로 리턴한다.

- ThreadManager 클래스의 eachThreadStartWork 메서드

```
public void eachThreadStartWork () {  
    switch (serviceCode) {  
        case ThreadCalculation.SQUARE:  
            for (int i = 0; i < threadNum; i++) {  
                tRect[i].start();  
            }  
            break;  
  
        case ThreadCalculation.QUADRATIC:  
            for (int i = 0; i < threadNum; i++) {  
                tqe[i].start();  
            }  
            break;  
    }  
}
```

지정된 서비스 코드에 해당하는 case를 수행한다.

1) 서비스 코드 SQUARE = 0

for문을 통해 threadNum만큼 반복을 수행한다.

매 루프마다 tRect 배열에 할당된 스레드 객체를 start한다.

1) 서비스 코드 QUADRATIC = 1

for문을 통해 threadNum만큼 반복을 수행한다.

매 루프마다 tqe 배열에 할당된 스레드 객체를 start한다.

- ThreadRectangle 클래스의 run 메서드

ThreadRectangle에서 run을 사용할 수 있다.

// Thread---[extends]---ThreadCommon---[extends]---ThreadRectangle

```
public void run() {
    for (int i = xStart[localThreadId]; i <= xEnd[localThreadId]; i++) {
        sum += dx * y;
        System.out.printf("Thread ID = %d, sum = %.12f\n", localThreadId, sum);
    }

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

```
// tRect = new ThreadRectangle[threadNum]
// tRect_index      [0]      [1]      [2]      [3]      [4]      [5]
// localThreadId    [0]      [1]      [2]      [3]      [4]      [5]
// total            334      334      333      333      333      333
// xStart           0       334      668      1001     1334     1667
// xEnd             333      667      1000     1333     1666     1999
```

serviceAlloc을 통해 ThreadCommon의 x와 y의 값은 2로 지정되어있다.

protected 변수인 x와 y는 상속관계를 가진 ThreadRectangle 클래스에서 사용할 수 있다.

// protected : 가문의 가보 -> 한 가족이면 사용할 수 있음, 공유해도 되는 정보

for문을 통해 xStart[현재 쓰레드ID] 인덱스의 값부터 xEnd[현재 쓰레드ID] 인덱스의 값까지 반복을 수행한다.

ex) tRect[0].start();

0.001 * 2 = sum, sum이 334번 반복누산된다.

```
sum = 0.667998492718
```


- ThreadQuadraticEquation 클래스의 run 메서드

ThreadQuadraticEquation에서 run을 사용할 수 있다.

// Thread---[extends]---ThreadCommon---[extends]---ThreadQuadraticEquation

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

    for (int i = xStart[localThreadId]; i <= xEnd[localThreadId]; i++, curX += dx) {
        sum += dx * curX * curX;
        System.out.printf("Thread ID = %d, sum = %.12f\n", localThreadId, sum);
    }

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

```
// tqe = new ThreadQuadraticEquation[threadNum]
// tqe_index          [0]      [1]      [2]      [3]
// localThreadId      [0]      [1]      [2]      [3]
// total              750      750      750      750
// xStart              0       750     1500     2250
// xEnd               749     1499     2249     2999
```

ThreadCommon 클래스의 protected 변수들을 사용해 계산

지정된 함수: $y = x^2$

각 스레드마다 x의 시작값을 지정해준다.

```
// curX              0.001 * 0 / 0.001 * 750 / 0.001 * 1500 / 0.001 * 2250
```

for문을 통해 xStart[현재 스레드ID] 인덱스의 값부터 xEnd[현재 스레드ID] 인덱스의 값까지 반복을 수행한다.

높이가 일정한 SQUARE와 달리 이 함수는 y값이 x값에 따라 변화한다.

이를 반영하기 위해 증감식에 $curX += dx$ 를 추가해준다.

넓이에 해당하는 밑변 * 높이의 결과값을 sum에 더해준다.

```
sum = 2.670504570007
sum = 5.201043605804
sum = 0.983542442322
sum = 0.140342220664
```

- ThreadManager 클래스의 eachThreadWaitFinish 메서드

```
public void eachThreadWaitFinish () throws InterruptedException {
    switch (serviceCode) {
        case ThreadCalculation.SQUARE:
            for (int i = 0; i < threadNum; i++) {
                tRect[i].join();
            }
            break;

        case ThreadCalculation.QUADRATIC:
            for (int i = 0; i < threadNum; i++) {
                tqe[i].join();
            }
            break;
    }
}
```

지정된 서비스 코드에 해당하는 case를 수행한다.

1) 서비스 코드 SQUARE = 0

for문을 통해 threadNum만큼 반복을 수행한다.

join을 통해 매 루프마다 tRect 배열에 할당된 각 스레드 객체의 작업이 끝날 때까지 기다렸다가 다음으로 넘어간다.

2) 서비스 코드 QUADRATIC = 1

for문을 통해 threadNum만큼 반복을 수행한다.

join을 통해 매 루프마다 tqe 배열에 할당된 각 스레드 객체의 작업이 끝날 때까지 기다렸다가 다음으로 넘어간다.

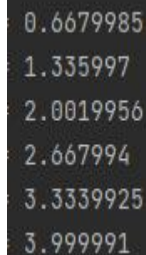
- ThreadManager 클래스의 sumEachThreadResult 메서드

```
public float sumEachThreadResult () {
    float sum = 0;

    switch (serviceCode) {
        case ThreadCalculation.SQUARE:
            for (int i = 0; i < threadNum; i++) {
                sum += tRect[i].getSum();
            }
            break;

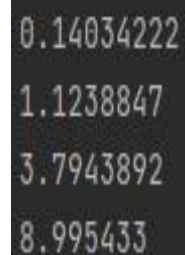
        case ThreadCalculation.QUADRATIC:
            for (int i = 0; i < threadNum; i++) {
                sum += tqe[i].getSum();
            }
            break;
    }

    return sum;
}
```



0.6679985
1.335997
2.0019956
2.667994
3.3339925
3.999991

< tRect >



0.14034222
1.1238847
3.7943892
8.995433

< tqe >

지정된 서비스 코드에 해당하는 case를 수행한다.

1) 서비스 코드 SQUARE = 0

for문을 통해 threadNum만큼 반복을 수행한다.

매 루프마다 tRect 배열에 할당된 각 스레드 객체의 getSum 메서드를 호출해 sum에 더한다.

case를 수행한 뒤 float형 변수 sum을 main으로 리턴한다.

2) 서비스 코드 QUADRATIC = 1

for문을 통해 threadNum만큼 반복을 수행한다.

매 루프마다 tqe 배열에 할당된 각 스레드 객체의 getSum 메서드를 호출해 sum에 더한다.

case를 수행한 뒤 float형 변수 sum을 main으로 리턴한다.

- getSum

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

1) 서비스 코드 SQUARE = 0

ThreadCommon 클래스에 선언되어있는 getSum 메서드를 자식 클래스인 ThreadRectangle 클래스에서 사용할 수 있다. float형 변수인 sum을 ThreadManager 클래스에 리턴한다.

2) 서비스 코드 QUADRATIC = 1

ThreadCommon 클래스에 선언되어있는 getSum 메서드를 자식 클래스인 ThreadQuadraticEquation 클래스에서 사용할 수 있다. float형 변수인 sum을 ThreadManager 클래스에 리턴한다.

- tm1.calcArea() 의 결과

```
계산된 값 = 3.999991
```

- tm2. calcArea() 의 결과

```
계산된 값 = 8.995433
```