(디지털 컨버전스) 스마트 콘텐츠와 웹 융합 응용 SW개발자 양성과정

훈련기간 : 2021.05.07 ~ 2021.12.08

KH정보교육원 | 전승리

멀티 Thread

```java
class Bank {
    private int money = 100000;

    public int getMoney() {
        return money;
    }
    public void setMoney(int money) {
        this.money = money;
    }

    public void plusMoney(int plus) {
        int m = this.getMoney();

        try {
            Thread.sleep( millis: 0);
        } catch (InterruptedException e) {
            // 에러 발생하면 어디서 에러가 났는지 출력해줘 ~
            e.printStackTrace();
        }
        this.setMoney(m + plus);
    }
    public void minusMoney(int minus) {
        int m = this.getMoney();

        try {
            Thread.sleep( millis: 0);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        this.setMoney(m - minus);
    }
}
```

Critical section : private int money

1번째 Thread

```java
class FirstThread extends Thread {
    public void run () {
        for (int i = 0; i < 1000; i++) {
            BankBombEventTest.myBank.plusMoney(1000);
            System.out.println("plusMoney(1000) = " + BankBombEventTest.myBank.getMoney());
        }
    }
}
```

2번째 Thread

```java
class SecondThread extends Thread {
    public void run () {
        for (int i = 0; i < 1000; i++) {
            BankBombEventTest.myBank.minusMoney(1000);
            System.out.println("minusMoney(1000) = " + BankBombEventTest.myBank.getMoney());
        }
    }
}

public class BankBombEventTest {
    public static Bank myBank = new Bank();

    public static void main(String[] args) {
        System.out.println("원금: " + myBank.getMoney());

        FirstThread first = new FirstThread();
        SecondThread second = new SecondThread();

        first.start();
        second.start();
    }
}
```

두개의 Thread가 critical section에 해당하는 money에 접근한다.
critical section을 보호해주는 쉴드가 없기때문에 값이 매번 바뀌게 된다.

printStackTrace : 예외 발생당시 호출스택에 있던 메서드의 정보와 예외메시지 출력기능

Lock을 이용해서 임계영역보호

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

public class Counter {
    private int count = 1;
    // Thread를 사용할 때 Lock을 걸려면
    // ReentrantLock을 사용하여 재진입이 가능한 형태로 만들어줘야 한다.
    private Lock lock = new ReentrantLock();

    public void increment () {
        try {
            lock.lock();
            count++;
        } finally {
            // 성공적으로 처리했던, 실패를 했던
            // finally는 무조건 실행된다.
            // 그러므로 내부에서 문제가 생겨도 Lock은 해제를 한다는 뜻
            // ex) 화장실에서 문 잠그고 죽음 ???
            lock.unlock();
        }
    }
    public void decrement () {
        try {
            lock.lock();
            count--;
        } finally {
            lock.unlock();
        }
    }
    public int getCount () {
        return count;
    }
}
```

Critical section : private int count

lock을 걸고 숫자 증가후 unlock

lock을 걸고 숫자감소 후 unlock

Join : 쓰레드가 끝나기를 기다린다.

Thread 생성용 클래스

```java
public class Worker implements Runnable {
    private Counter counter;
    private boolean increment;
    private int count;

    public Worker(Counter counter, boolean increment, int count) {
        this.counter = counter;
        this.increment = increment;
        this.count = count;
    }

    @Override
    public void run() {
        for (int i = 0; i < this.count; i++) {
            if (increment) {
                this.counter.increment();
                System.out.println("I'm increment");
            } else {
                this.counter.decrement();
                System.out.println("I'm decrement");
            }
        }
    }
}
```

lock 설정

```
First count: 1
I'm decrement
I'm decrement
I'm decrement
I'm decrement
I'm decrement
I'm decrement
I'm increment
I'm increment
I'm increment
I'm increment
I'm increment
I'm increment
I'm increment
I'm increment
Final count: 1
```

```java
public class BankLockTest {
    public static void main(String[] args) throws InterruptedException {
        Counter counter = new Counter();
        System.out.println("First count: " + counter.getCount());

        Thread adder = new Thread(new Worker(counter,  increment: true,  count: 1000));
        adder.start();

        Thread subtracter = new Thread(new Worker(counter,  increment: false,  count: 1000));
        subtracter.start();

        adder.join();
        subtracter.join();

        System.out.println("Final count: " + counter.getCount());
    }
}
```

멀티와 싱글 스레드 속도차이

```
public class SequencePerformanceTest {
    final static int ZERO = 0;
    final static int END5 = 1000000000;
    final static int START5 = 800000001;
    final static int END4 = 800000000;
    final static int START4 = 600000001;
    final static int END3 = 600000000;
    final static int START3 = 400000001;
    final static int END2 = 400000000;
    final static int START2 = 200000001;
    final static int END = 200000000;
    final static int START = 1;
    final static double COEFFICIENT = Math.pow(10,
    final static double DEG2RAD = 180.0;
```

```
public static void main(String[] args) {
    double sum = ZERO;
    double sum2 = ZERO;
    double sum3 = ZERO;
    double sum4 = ZERO;
    double sum5 = ZERO;

    PerformanceUtil.performanceCheckStart();

    for(int i = START; i <= END; i++) {
        sum += (i * (COEFFICIENT * i)) * Math.sin(i * Math.PI / DEG2RAD);
    }

    for(int i = START2; i <= END2; i++) {
        sum2 += (i * (COEFFICIENT * i)) * Math.sin(i * Math.PI / DEG2RAD);
    }

    for(int i = START3; i <= END3; i++) {
        sum3 += (i * (COEFFICIENT * i)) * Math.sin(i * Math.PI / DEG2RAD);
    }

    04; i++) {
    NT * i)) * Math.sin(i * Math.PI / DEG2RAD);

    05; i++) {
    NT * i)) * Math.sin(i * Math.PI / DEG2RAD);
```

```
        System.out.println("sum = " + sum);
        System.out.println("sum2 = " + sum2);
        System.out.println("sum3 = " + sum3);
        System.out.println("sum4 = " + sum4);
        System.out.println("sum5 = " + sum5);

        PerformanceUtil.performanceCheckEnd();

        //System.out.println("sum = " + sum);

        PerformanceUtil.printPerformance();
    }
}
```

```
sum = 2146.7213270913494
sum2 = -9117.696529015964
sum3 = 17128.065734289175
sum4 = -16209.34218820474
sum5 = -4389.21378969557
걸린 시간: 45.409 s
```

싱글

일단 코드보다 속도위주로 확인했다.

멀티와 싱글 스레드 속도차이

```java
public class AccelThread extends OperationAccelerat
    private int localStart;
    private int localEnd;
    private int threadId;

    private double localSum = 0;

    private static double totalSum = 0;

    public AccelThread(int start, int end, int maxTh
        super(start, end, maxThreadNum);

        int total = end - start + 1;
        int threadPerData = total / maxThreadNum;

        localStart = id * threadPerData + 1;
        localEnd = localStart + threadPerData - 1;
        threadId = id;
    }
```

```java
@Override
public void run() {
    System.out.printf("threadId = %d, localStart = %d\
    System.out.printf("threadId = %d, localEnd = %d\n"

    for(int i = localStart; i <= localEnd; i++) {
        localSum += (i * (PerformanceTest.COEFFICIENT

    }

    System.out.printf("threadId = %d, localSum = %f\n"

    addAll(localSum);

    System.out.printf("threadId = %d, totalSum = %f\n", threadId, totalSum);
}
```

```java
public class PerformanceTest {
    final static int ZERO = 0;
    final static int END = 1000000000;
    final static int START = 1;
    final static double COEFFICIENT = Math.pow(10, -15);
    final static double DEG2RAD = 180.0;

    final static int MAXTHREAD = 5;

    public static void main(String[] args) throws InterruptedException {
        double sum = ZERO;

        Thread[] thr = new Thread[MAXTHREAD];

        for(int i = 0; i < MAXTHREAD; i++) {
            thr[i] = new Thread(new AccelThread(START, END, MAXTHREAD, i));
        }

        PerformanceUtil.performanceCheckStart();

        for(int i = 0; i < MAXTHREAD; i++) {
            thr[i].start();
            // sum += (i * (COEFFICIENT * i)) * Math.sin(i * Math.PI / DEG2RAD);
        }

        for(int i = 0; i < MAXTHREAD; i++) {
            thr[i].join();
        }

        PerformanceUtil.performanceCheckEnd();
        PerformanceUtil.printPerformance();
    }
}
```

```
sum = 2146.7213270913494
sum2 = -9117.696529015964
sum3 = 17128.065734289175
sum4 = -16209.34218820474
sum5 = -4389.21378969557
걸린 시간: 45.409 s
```

싱글

```
threadId = 0, localStart = 1
threadId = 4, localStart = 800000001
threadId = 4, localEnd = 1000000000
threadId = 3, localStart = 600000001
threadId = 3, localEnd = 800000000
threadId = 2, localStart = 400000001
threadId = 2, localEnd = 600000000
threadId = 1, localStart = 200000001
threadId = 1, localEnd = 400000000
threadId = 0, localEnd = 200000000
threadId = 0, localSum = 2146.721327
threadId = 0, totalSum = 2146.721327
threadId = 1, localSum = -9117.696529
threadId = 1, totalSum = -6970.975202
threadId = 4, localSum = -4389.213790
threadId = 4, totalSum = -11360.188992
threadId = 3, localSum = -16209.342188
threadId = 3, totalSum = -27569.531180
threadId = 2, localSum = 17128.065734
threadId = 2, totalSum = -10441.465446
걸린 시간: 11.391 s
```

멀티

속도차이가 확연히 차이남을 확인할수있다.