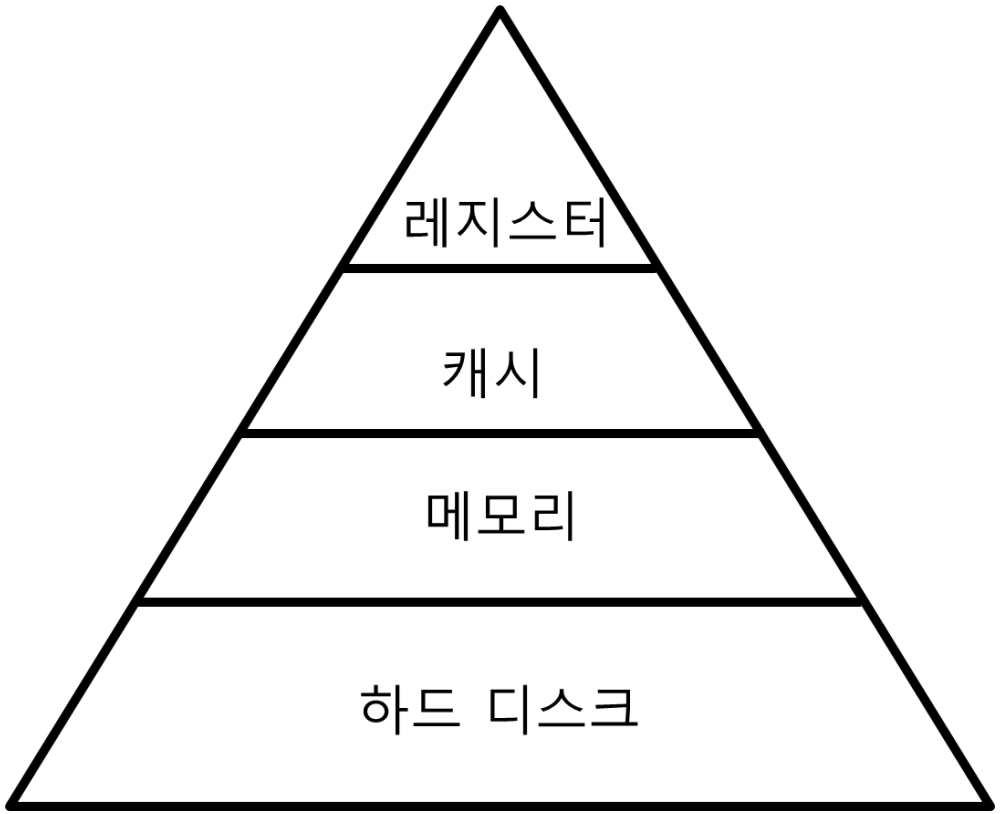
(디지털컨버전스)스마트 콘텐츠와 웹 융합응용SW개발자 양성과정

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 –Joongyeon Kim(김중연)
jjjr69@naver.com

2021년 6월 4일 질문노트

[김중연]



레지스터와 캐시는 CPU 내부에 존재한다. 당연히 CPU는 아주 빠르게 접근할 수 있다

메모리는 CPU 외부에 존재한다. 레지스터와 캐시보다 더 느리게 접근 할 수 밖에 없다. 하드 디스크는 CPU가 직접 접근할 방법조차 없다

CPU가 하드 디스크에 접근하기 위해서는 하드 디스크의 데이터를 메모리로 이동시키고, 메모리에서 접근해야 한다. 아주 느린 접근 방에 불가능하다.

```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; }
}
```

```java
public class Worker implements Runnable {
    private Counter counter;  //Counter 클래스에 들어있는 counter파일을 불러온다
    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");
            }
        }
    }
}
```

```java
public class BankLockTest {
    public static void main(String[] args) throws InterruptedException {
                                    // throws InterruptedException이 붙은 대표적인 메소드는 다음과 같다.
                                    //java.lang.Object 클래스의 wait 메소드
                                    //java.lang.Object 클래스의 sleep 메소드
                                    //java.lang.Object 클래스의 join 메소드
        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());
    }
}
```

InterruptedException은 자바 스레드의 인터럽트 메커니즘의 일부이다. 자바에서는 스레드에 하던 일을 멈추라는 신호를 보내기 위해 인터럽트를 사용한다.

한 스레드가 다른 스레드를 인터럽트 할 수 있고, 각 스레드는 자신이 인터럽트 되었는지 확인할 수 있다. 스레드가 자기 자신을 인터럽트 할 수도 있다.

대부분의 경우 인터럽트는 하던 일을 멈추라는 신호이며, 해당 스레드는 이를 적절히 처리해야 한다.

요약

- 인터럽트는 스레드를 종료하기 위한 메커니즘이다.
- 테스트 코드나 장난감 코드가 아니라면 인터럽트를 적절히 처리하도록 하자.

```java
class Bank {
    // 만 단위로 10억임
    private int money = 100000;

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

    public void plusMoney(int plus) {     //돈을 더하는 스레드를 만듬
        int m = this.getMoney();  // 변수 m에 getMoney값을 대입한다

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

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

class FirstThread extends Thread {  //스레드를 만듬
```

```
    public void run () {
        for (int i = 0; i < 10; i++) {
            BankBombEventTest.myBank.plusMoney(1000);
            System.out.println("plusMoney(1000) = " + BankBombEventTest.myBank.getMoney());
        }
    }
}

class SecondThread extends Thread {
    public void run () {
        for (int i = 0; i < 10; 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());  //데이터의 무결성이 깨졌다!  전역변수money를 두 스레드가 번갈아 마구잡이로 사용함

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

        first.start();   //스레드는 스타트를 해줘야 시작한다!
        second.start();
    }
}
```

```java
class ParallelThread implements Runnable {
    final int MAX = 4;
    final int MAX_LOOP = 20;

    private static int[] parallelProcessingArr;
    private int threadLocalIdx;

    public ParallelThread(int threadLocalIdx) {
        this.threadLocalIdx = threadLocalIdx;
        parallelProcessingArr = new int[MAX];

        for (int i = 0; i < MAX; i++) {  //0~3번 스레드의 값인 1을 출력함
            parallelProcessingArr[i] = 1;
        }
    }

    @Override
    public void run() {
        for (int i = 0; i < MAX_LOOP; i++) {  //
            for (int j = 0; j < MAX_LOOP; j++) {
                parallelProcessingArr[threadLocalIdx]++;  //
                parallelProcessingArr[threadLocalIdx]--;  //
                System.out.printf("안녕 나는 %d 번 스레드야! 현재값은 %d 이다!\n",
                        threadLocalIdx, parallelProcessingArr[threadLocalIdx]);
            }
        }
    }

    public static int[] getParallelProcessingArr() { return parallelProcessingArr; }
}
```

```java
public class ParallelConceptTest {
    public static void main(String[] args) throws InterruptedException {
        System.out.println("지금부터 병렬 처리를 시작합니다.");

        Thread[] pt = new Thread[4];

        long startTime = System.currentTimeMillis();  // 스톱워치 시작

        for (int i = 0; i < 4; i++) {
            pt[i] = new Thread(new ParallelThread(i));  //스레드 출력시작
            pt[i].start();
        }

        for (int i = 0; i < 4; i++) {
            pt[i].join();               //join을 넣으면 쓰레드가 종료되기 전까지 아랫부분의 출력을 멈춘다
        }

        for (int i = 0; i < 4; i++) {
            System.out.println("최종 결과 확인: " + ParallelThread.getParallelProcessingArr()[i]);
        }

        long endTime = System.currentTimeMillis();  // 스톱워치 끝

        System.out.println(endTime - startTime + "ms");
    }
}
```