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

    public void plusMoney(int plus){
        int m = this.getMoney();
        try{
            Thread.sleep( millis: 0); // 기다리기 귀찮아서 sleep시간 0으로
        } catch (InterruptedException e){
            e.printStackTrace(); //erorr 발생시 어디서 error 발생했는지 출력
        }
    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);
    }
    public int getMoney() { return money; }
    public void setMoney(int money) { this.money = money; }
}
```

2021.06.04 Java

<thread 사용시 동기화 문제 발생 예제>

Q. 사용하는 컴퓨터가 듀얼 코어라면 java에서도
Thread의 객체를 2개까지만 만들 수 있고
쿼드 코어라면 4개. 헥사 코어라면 6개. 옥타코어라면 8개 까지만
만들 수 있는건가요??

<< 사실상 thread.sleep이 error가 날 일은 없어서
printStackTrace() method가 실행될 일은 없긴하다.

```java
class FirstThread extends Thread{
    //Thread를 상속받기 때문에 FirstThread에 대한 생성자를 호출하지 않아도
    //동작되게 되어 있음.
    public void run() {
        for (int i = 0; i < 1000; i++) {
            _1st_ThreadError.myBank.plusMoney(1000);
            System.out.println("plusMoney(1000) = " + _1st_ThreadError.myBank.getMoney());
        }
    }
}
```

Thread를 상속 받는
FirstThread /SecondThread class 생성.

FirstThread의 thread는 1000원을 더하는
method를 1000번 실행하는데 사용됨.

```java
class SecondThread extends Thread{
    public void run() {
        for (int i = 0; i < 1000; i++) {
            _1st_ThreadError.myBank.minusMoney(1000);
            System.out.println("minusMoney(1000) = " + _1st_ThreadError.myBank.getMoney());
        }
    }
```

SecondThread의 thread는 1000원을 빼는
method를 1000번 실행하는데 사용됨.

```java
public class _1st_ThreadError {
    public static Bank myBank = new Bank();
    // myBank를 전역으로 공유하기 위해 작성한 code(static)
    // static 빼보면 위에 class들에서 erorr나는 것 볼 수 있음.
    public static void main(String[] args) {
        System.out.println("원금: " + myBank.getMoney());

        FirstThread t1 = new FirstThread();
        SecondThread t2 = new SecondThread();

        t1.start();
        t2.start();
    }
```

main에서 FirstThread / SecondThread의 객체 t1 / t2 생성하고 t1.start( ) /t2.start 호출
))Bank class의 money는 critical section이 됨.

```
plusMoney(1000) = -81000
minusMoney(1000) = -88000
plusMoney(1000) = -87000
plusMoney(1000) = -87000
plusMoney(1000) = -86000
plusMoney(1000) = -85000
plusMoney(1000) = -84000
plusMoney(1000) = -83000
minusMoney(1000) = -88000
minusMoney(1000) = -84000
minusMoney(1000) = -85000

Process finished with exit
```

```
minusMoney(1000) = 161000
minusMoney(1000) = 160000
minusMoney(1000) = 159000
minusMoney(1000) = 158000
minusMoney(1000) = 157000
minusMoney(1000) = 156000
minusMoney(1000) = 155000
minusMoney(1000) = 154000
minusMoney(1000) = 153000
minusMoney(1000) = 152000
minusMoney(1000) = 151000
minusMoney(1000) = 150000

Process finished with exit
```

문제가 없다면

각각 1000번씩 실행되기 때문에 마지막 money의 값은 그대로 1000000이 되어야 하는데

매번 할 때마다 그 값이 달라짐 ›› 데이터의 무결성 깨짐.

›› Bank class의 this.setMoney(m + plus)

Bank class의 this.setMoney(m + minus) 쪽에 lock이 필요함.

```
See Also: run(), stop()
public synchronized void start() {
    /**
     * This method is not invoked for the main
     * group threads created/set up by the VM.
     * to this method in the future may have to
     *
     * A zero status value corresponds to state
     */
```

Thread.start()가 synchronized로 보호되고 있음에도 불구하고 이런 문제가 발생하기 때문에 lock 필요.

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

public class Counter {
    private int count = 1;
    private Lock lock = new ReentrantLock();
    //Thread를 사용할 때 lock을 걸려면
    // ReentrantLock을 사용하여 재진입 가능한 형태로 만들어줘야 한다.
    public void increment(){
        try{
            lock.lock();
            count++;
        } finally {
            // 성공적으로 처리했던, 실패했던
            // finally는 무조건 실행된다.
            lock.unlock();;// 따라서 내부에서 문제가 생겨도 lock은 해제를 하겠다는 코드
        }
    }
    public void decrement(){
        try{
            lock.lock();
            count--;
        } finally{
            lock.unlock();
        }
    }
    public int getCount() { return count; }
```

〈lock 사용 예제〉

public class Counter에서는

--

정수 count를 ++하는 method increment()

정수 count를 --하는 method decrement()

각각에 ++ / --가 실행되기 전에 ReentrantLock을 걸어준다.

--

와 같은 코딩이 되어 있음.

```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("increasing");
            } else {
                    this.counter.decrement();
                    System.out.println("decreasing");
            }
        }
    }
}
```

Counter class형 datatype의 변수 counter

public class Worker는 Runnable을 implement한다.

run()에서

increment가 true일 시

count값을 1씩 증가시키는 counter.increment()를 실행

false일 때는 decrement() 실행.

```java
public class _2nd_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 substracter = new Thread(new Worker(counter, increment: false, count: 1000));
        substracter.start();

        adder.join();
        substracter.join();

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

increment: true로 increment() 를 1000번(count:1000) 하는 thread 'adder' 생성

increment: false로 decrement() 를 1000번(count:1000) 하는 thread 'substracter' 생성

Q. join() method가 하는 일은 뭔가요..?

```
"C:\Program Files
First count: 1
increasing
decreasing
decreasing
decreasing
decreasing
decreasing
decreasing
decreasing
decreasing
decreasing
decreasing
decreasing
decreasing
decreasing
decreasing
```

```
increasing
increasing
increasing
decreasing
decreasing
decreasing
increasing
increasing
decreasing
decreasing
decreasing
increasing
increasing
increasing
increasing
increasing
decreasing
decreasing
Final count: 1
```

ReentrantLock을 이용해서 한 thread가 작업할 때는 다른 thread는 critical section에 접근 못 하도록 lock을 했기 때문에 increment / decrement를 각각 1000번씩 돌리면 결국 count의 값은 1에서 시작해서 마지막에 1이 된다.

)) thread ((를 쓰는 이유는 뭔가??

빠르기 때문에.

속도가 상관 없다면. 그냥 병렬로 thread들 사용 할 필요 없이

하나로만 순차적으로 작업한다면 데이터무결성이 깨질일도 없음.

그럼에도 불구하고 thread를 사용하는 이유는 thread 이용하면 '빠르다'

)) class에 public 붙일 수 있는 경우는 ?

project상에 file로 class를 만드는 경우에 사용.