

```
package Nineteenth;
```

```
public class FailedBank {  
    private int money = 100000;  
  
    public int getMoney() {  
        return money;  
    }  
  
    public void setMoney(int money) {  
        this.money = money;  
    }  
  
    public void plusMoney(int plus) {  
        int m = getMoney();  
  
        try {  
            Thread.sleep(80);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
  
        setMoney(m + plus);  
    }  
  
    public void minusMoney(int minus) {  
        int m = getMoney();  
  
        try {  
            Thread.sleep(50);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
  
        setMoney(m - minus);  
    }  
}
```

→ Critical Section

⇒ 두개의 Thread에서 이 값을 공유함.

Thread Sleep을 줘서만

Sleep 이후 경쟁하는 과정에서

데이터 무결성이 깨졌다.

```

3 class A extends Thread {
4     public void run() {
5         for(int i = 0; i < 100; i++) {
6             WhyThreadMutex.fb.plusMoney(3000);
7         }
8
9         System.out.println("plusMoney(3000): " +
10             WhyThreadMutex.fb.getMoney());
11     }
12 }

```

①

→ Thread 실행
Method

```

13
14 class B extends Thread {
15     public void run() {
16         for(int i = 0; i < 100; i++) {
17             WhyThreadMutex.fb.minusMoney(1000);
18         }
19
20         System.out.println("minusMoney(1000): " +
21             WhyThreadMutex.fb.getMoney());
22     }
23 }

```

②

WhyThreadMutex 안의
Failed Bank 클래스 안에 있는
minus Money 메소드?

```

24
25 public class WhyThreadMutex {
26     public static FailedBank fb = new FailedBank();
27
28     public static void main(String[] args) {
29         System.out.println("원금: " + fb.getMoney());
30
31         A a = new A();
32         B b = new B();
33
34         a.start();
35         b.start();
36
37         // 사실상 이 문제를 해결하기 위해 도입해야 하는 것이
38         // Lock Mechanism(Mutex, Semaphore, Spinlock) 이다.
39         // 이들은 결론적으로 지금 거론한 확장성 사용증을 표기하는 것이다.
40         // 자바는 스핀락 사용이 쉽지 않음
41         // (일반적으로 스핀락 코드는 어셈블리어를 통해 구현됨)

```

(나만 아는 것)

```
3 import java.util.Random;
4 import java.util.concurrent.Semaphore;
5
6 public class ItemEnhanceThreadTest {
7     private Semaphore sem = new Semaphore(3);
8
9     public void enter(Item item) {
10         try {
11             sem.acquire();
12         } catch (InterruptedException e) {
13             return;
14         }
15
16         try {
17             item.enhance(this);
18         } finally {
19             sem.release();
20         }
21     }
22
23     public static class Item {
24         private static int cnt = 0;
25         private int num = ++cnt;
26
27         private void enhance(ItemEnhanceThreadTest iet) {
28             System.out.println("강화 시작!");
29
30             Random rand = new Random();
31
32             try {
33                 Thread.sleep(rand.nextInt(5000) + 5000);
34             } catch (InterruptedException e) {
35
36             }
37
38             System.out.println(num + "번 아이템 강화 종료!");
39         }
40     }
41 }
```

Semaphore 개수
방의 개수

Semaphore로
수행할 내용.

++cnt를 앞에 붙이는 이유가 무엇일까?

```

44 // -----
45 // | lock = 1 |   특정한 메모리 주소 공간 (SL)
46 // -----
47
48 // SL에 값이 1 이나 ? 아니냐 ?
49 // 1이면 현재 Lock 상태니까 들어오지마!
50 // 0이면 현재 Unlock 상태니까 먼저 들어오는 녀미 일자!

```

Mutex

```

52 // -----
53 // | lock = 1 |   특정한 메모리 주소 공간 (SL1)
54 // -----
55 // | lock = 2 |   특정한 메모리 주소 공간 (SL2)
56 // -----
57 // | lock = 3 |   특정한 메모리 주소 공간 (SL3)
58 // -----

```

Semaphore

```

59
60 // SL1, 2, 3중 빈게 있냐 ? 없냐 ?
61 // 있으면 먼저 들어오는 녀미 일자!
62 // 없으면 못들어오니까 돌아가렴 ~
63
64 // synchronized 같은 경우에는 Mutex 개념으로
65 // Thread 여러개가 동시다발적으로 구동이 불가능한 반면
66 // Semaphore는 Mutex List 같은 개념으로
67 // Thread 여러개가 동시다발적으로 구동이 가능하다.

```

```

68
69 // Mutex와 Semaphore의 차이점이 무엇인가 ?
70 // 대기열이 있다 없다.

```

```

71 public static void main(String[] args) {
72     final ItemEnhanceThreadTest iet = new ItemEnhanceThreadTest();
73
74     for(int i = 0; i < 6; i++) {
75         Thread t = new Thread(new Runnable() {
76             @Override
77             public void run() {
78                 Item item = new Item();
79                 iet.enter(item);
80             }
81         });
82         t.start();

```

Start로 Thread를 실행할 수 있는 공간 생성.

↓
run 안에 있는 method 실행.

```

package Nineteenth;
2
public class PerfSyncBank {
3
4     private int money = 100000;
5
6     private String plusMsg = "";
7     private String minusMsg = "";
8
9     public int getMoney() {
10         return money;
11     }
12
13     public void setMoney(int money) {
14         this.money = money;
15     }
16
17     public void plusMoney(int plus) { Critical Section.
18         int m = getMoney();
19
20         // Memory Hierarchy(메모리 계층 구조)
21         // 용량이 크면 클 수록 동작 속도가 느리고
22         // 속도가 빠르면 빠를수록 용량이 작다.
23         // 결론적으로 I/O 를 최소화하면 어떤 프로그램이든 속도가 엄청나게 빨라진다.
24
25         // 최적화의 순서
26         // 1) I/O 최소화
27         // 2) 알고리즘 & 자료구조
28         // 3) 마이크로 아키텍처적 접근(CPU 파이프라인, 캐시, 어셈블리어 등등)
29
30         // 결국 모니터에 출력하는 행위 자체가 I/O 를 빈번하게 발생시키는 것이므로
31         // I/O를 최소화 하고자 한다면 한 번에 모아서 출력하면 된다.
32         // I/O를 block 단위로 모아 출력하는 방식은 잠시 보류!
33
34         //System.out.println("m = " + m);
35         //plusMsg += "m = " + m + "\n";
36
37         // 4096 / 16 = 2^12 / 2^4 = 2^8 = 256
38
39         setMoney(m + plus);
40     }

```

```
42 // synchronized 키워드를 통해서
43 // 강제로 Thread 간의 순서를 조정해줄 수 있다.
44 // 그러므로 서로 화장실 들어가겠다고 싸우는
45 // Race Condition을 무마할 수 있다.
```

```
46 public void minusMoney(int minus) {
47     int m = getMoney();
48
49     //System.out.println("m = " + m);
50     //minusMsg += "m = " + m + "\n";
51
52     setMoney(m - minus);
53 }
```

```
54
55 public String getPlusMsg() {
56     return plusMsg;
57 }
```

```
58
59 public String getMinusMsg() {
60     return minusMsg;
61 }
62 }
```

→ Critical Section.

```

class X extends Thread {
    public void run() {
        for(int i = 0; i < 1000000; i++) {
            // 현재 케이스는 실제 Critical Section에만 강제 동기화를 걸었다.
            // 그러므로 여러 태스크들이 동시에 접근할 수 있는 영역을
            // 부분적으로 안전하게 보호한 반면
            // 이전의 예제는 매서드 전체를 보호했다.
            // 그러므로 당연히 Critical Section만 방어할때에 비해 성능이 저하된다.
            synchronized (PerfSyncBankTest.psb) {
                PerfSyncBankTest.psb.plusMoney(3000);
            }
        }

        System.out.println(PerfSyncBankTest.psb.getPlusMsg());

        System.out.println("plusMoney(): " +
            PerfSyncBankTest.psb.getMoney());
    }
}

class Y extends Thread {
    public void run() {
        for(int i = 0; i < 1000000; i++) {
            synchronized (PerfSyncBankTest.psb) {
                PerfSyncBankTest.psb.minusMoney(1000);
            }
        }

        System.out.println(PerfSyncBankTest.psb.getMinusMsg());

        System.out.println("minusMoney(): " +
            PerfSyncBankTest.psb.getMoney());
    }
}

```

Mutex를 사야에서 사용하는 방법.

② Critical Section에서 이슈 발생
부분에서 동전화 처리.

속도↑ 용량↓

```
38 public class PerfSyncBankTest {
39     public static PerfSyncBank psb = new PerfSyncBank();
40
41     public static void main(String[] args) throws InterruptedException {
42         System.out.println("원금: " + psb.getMoney());
43
44         X x = new X();
45         Y y = new Y();
46
47         x.start();
48         y.start();
49     }
50 }
```



```

1 package Nineteenth;
2
3 public class SynchronizedBank {
4     private int money = 100000;
5
6     public int getMoney() {
7         return money;
8     }
9
10    public void setMoney(int money) {
11        this.money = money;
12    }
13
14    public synchronized void plusMoney(int plus) {
15        int m = getMoney();
16
17        /*
18        try {
19            Thread.sleep(80);
20        } catch (InterruptedException e) {
21            e.printStackTrace();
22        }
23        */
24
25        System.out.println("m = " + m);
26
27        setMoney(m + plus);
28    }
29
30    // synchronized 키워드를 통해서
31    // 강제로 Thread 간의 순서를 조정해줄 수 있다.
32    // 그러므로 서로 화장실 들어가겠다고 싸우는
33    // Race Condition을 무마할 수 있다.
34    public synchronized void minusMoney(int minus) {
35        int m = getMoney();

```

Mutex를 자바에서 사용하는법.

① 클래스 전체를 동기화하는방법

↑ ?

순서를 조정해줄 수 있다는 말은
언제 쓰여진 것이 먼저 순서인가요?

```
37      /*
38      try {
39          Thread.sleep(50);
40      } catch (InterruptedException e) {
41          e.printStackTrace();
42      }
43      */
44
45      System.out.println("m = " + m);
46
47      setMoney(m - minus);
48  }
49 }
```

```
3 class C extends Thread {
4     public void run() {
5         for(int i = 0; i < 1000000; i++) {
6             SynchronizedBankTest.sb.plusMoney(3000);
7         }
8
9         //SynchronizedBankTest.sb.plusMoney(3000);
10
11        System.out.println("plusMoney(3000): " +
12            SynchronizedBankTest.sb.getMoney());
13    }
14 }
15
16 class D extends Thread {
17     public void run() {
18         for(int i = 0; i < 1000000; i++) {
19             SynchronizedBankTest.sb.minusMoney(1000);
20         }
21
22         //SynchronizedBankTest.sb.minusMoney(1000);
23
24        System.out.println("minusMoney(1000): " +
25            SynchronizedBankTest.sb.getMoney());
26    }
27 }
28
29 public class SynchronizedBankTest {
30     public static SynchronizedBank sb = new SynchronizedBank();
31
32     public static void main(String[] args) throws InterruptedException {
33         System.out.println("원금: " + sb.getMoney());
34
35         C c = new C();
36         D d = new D();
37
38         c.start();
39         d.start();
40     }
41 }
```

Class 전체를 동기화
→ 이처럼 D가 늘어나면
속도↓, 용량↑