```java
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);
    }
}
```

Critica section

⇒ 두개의 Thread에서 이 값을 공유함.

Thread Sleep을 줬지만

Sleep이후 계산하는 과정에서

데이터 무결성이 깨졌다.

```java
class A extends Thread {
    public void run() {
        for(int i = 0; i < 100; i++) {
            WhyThreadMutex.fb.plusMoney(3000);
        }

        System.out.println("plusMoney(3000): " +
                WhyThreadMutex.fb.getMoney());
    }
}

class B extends Thread {
    public void run() {
        for(int i = 0; i < 100; i++) {
            WhyThreadMutex.fb.minusMoney(1000);
        }

        System.out.println("minusMoney(1000): " +
                WhyThreadMutex.fb.getMoney());
    }
}

public class WhyThreadMutex {
    public static FailedBank fb = new FailedBank();

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

        A a = new A();
        B b = new B();

        a.start();
        b.start();

        // 사실상 이 문제를 해결하기 위해 도입해야 하는 것이
        // Lock Mechanism(Mutex, Semaphore, Spinlock) 이다.
        // 이들은 결론적으로 지금 거론한 화장실 사용중을 표기하는 것이다.
        // 자바는 스핀락 사용이 쉽지 않음
        // (일반적으로 스핀락 코드는 어셈블리어를 통해 구현됨)
```

1

Thread 상속받는 method

2

WhyThreadMutex 안에있는 FailedBank 클래스 안에 있는 minusMoney 메소드 ?

```java
import java.util.Random;
import java.util.concurrent.Semaphore;

public class ItemEnhanceThreadTest {
    private Semaphore sem = new Semaphore(3)

    public void enter(Item item) {
        try {
            sem.acquire();
        } catch (InterruptedException e) {
            return;
        }

        try {
            item.enhance(this);
        } finally {
            sem.release();
        }
    }

    public static class Item {
        private static int cnt = 0;
        private int num = ++cnt;

        private void enhance(ItemEnhanceThreadTest iet) {
            System.out.println("강화 시작!");

            Random rand = new Random();

            try {
                Thread.sleep(rand.nextInt(5000) + 5000);
            } catch (InterruptedException e) {

            }

            System.out.println(num + "번 아이템 강화 종료!");
        }
    }
```

(내맘대로 주석)

Semaphore 갯수 → 방의 갯수

Semaphore로 수행할 내용.

++을 앞에 붙여도 이렇게 문제없을까?

```
// ------------
// | lock = 1 |  특정한 메모리 주소 공간 (SL)
// ------------

// SL에 값이 1 이냐 ? 아니냐 ?
// 1이면 현재 Lock 상태니까 들어오지마!
// 0이면 현재 Unlock 상태니까 먼저 들어오는 넘이 임자!

// ------------
// | lock = 1 |  특정한 메모리 주소 공간 (SL1)
// ------------
// | lock = 2 |  특정한 메모리 주소 공간 (SL2)
// ------------
// | lock = 3 |  특정한 메모리 주소 공간 (SL3)
// ------------

// SL1, 2, 3중 빈게 있냐 ? 없냐 ?
// 있으면 먼저 들어오는 넘이 임자!
// 없으면 못들어오니까 돌아가렴 ~

// synchronized 같은 경우에는 Mutex 개념으로
// Thread 여러개가 동시다발적으로 구동이 불가능한 반면
// Semaphore는 Mutex List 같은 개념으로
// Thread 여러개가 동시다발적으로 구동이 가능하다.

// Mutex와 Semaphore의 차이점이 무엇인가 ?
// 대기열이 있다 없다.
public static void main(String[] args) {
    final ItemEnhanceThreadTest iet = new ItemEnhanceThreadTest();

    for(int i = 0; i < 6; i++) {
        Thread t = new Thread(new Runnable() {
            @Override
            public void run() {
                Item item = new Item();
                iet.enter(item);
            }
        });
        t.start();
```

Mutex

Semaphore

Start로 Thread를 실행할 수 있는 공간 생성.
↓
run 안에 있는 method 실행.

```java
package Nineteenth;

public class PerfSyncBank {
    private int money = 100000;

    private String plusMsg = "";
    private String minusMsg = "";

    public int getMoney() {
        return money;
    }

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

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

        // Memory Hierarchy(메모리 계층 구조)
        // 용량이 크면 클 수록 동작 속도가 느리고
        // 속도가 빠르면 빠를수록 용량이 작다.
        // 결론적으로 I/O 를 최소화하면 어떤 프로그램이던 속도가 엄청나게 빨라진다.

        // 최적화의 순서
        // 1) I/O 최소화
        // 2) 알고리즘 & 자료구조
        // 3) 마이크로 아키텍처적 접근(CPU 파이프라인, 캐시, 어셈블리어 등등)

        // 결국 모니터에 출력하는 행위 자체가 I/O 를 빈번하게 발생시키는 것이므로
        // I/O를 최소화 하고자 한다면 한 번에 모아서 출력하면 된다.
        // I/O를 block 단위로 모아 출력하는 방식은 잠시 보류!

        //System.out.println("m = " + m);
        //plusMsg += "m = " + m + "\n";

        // 4096 / 16 = 2^12 / 2^4 = 2^8 = 256

        setMoney(m + plus);
    }
```

Critical Section.

```
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  }
```

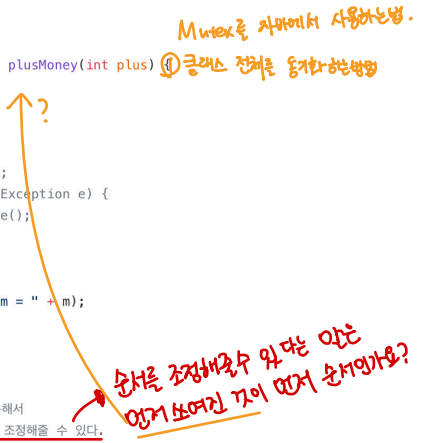
Critical Section.

```java
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 자바에서 사용하는 방법.
② Critical Section에 에서 이상 발생 부분에서 동기화 처리.
속도↑ 용량↓

```java
public class PerfSyncBankTest {
    public static PerfSyncBank psb = new PerfSyncBank();

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

        X x = new X();
        Y y = new Y();

        x.start();
        y.start();
    }
}
```

```java
package Nineteenth;

public class SynchronizedBank {
    private int money = 100000;

    public int getMoney() {
        return money;
    }

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

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

        /*
        try {
            Thread.sleep(80);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
         */

        System.out.println("m = " + m);

        setMoney(m + plus);
    }

    // synchronized 키워드를 통해서
    // 강제로 Thread 간의 순서를 조정해줄 수 있다.
    // 그러므로 서로 화장실 들어가겠다고 싸우는
    // Race Condition을 무마할 수 있다.
    public synchronized void minusMoney(int minus) {
        int m = getMoney();
```



```
        /*
        try {
            Thread.sleep(50);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
         */

        System.out.println("m = " + m);

        setMoney(m - minus);
    }
}
```

```java
class C extends Thread {
    public void run() {
        for(int i = 0; i < 1000000; i++) {
            SynchronizedBankTest.sb.plusMoney(3000);
        }

        //SynchronizedBankTest.sb.plusMoney(3000);

        System.out.println("plusMoney(3000): " +
                SynchronizedBankTest.sb.getMoney());
    }
}

class D extends Thread {
    public void run() {
        for(int i = 0; i < 1000000; i++) {
            SynchronizedBankTest.sb.minusMoney(1000);
        }

        //SynchronizedBankTest.sb.minusMoney(1000);

        System.out.println("minusMoney(1000): " +
                SynchronizedBankTest.sb.getMoney());
    }
}

public class SynchronizedBankTest {
    public static SynchronizedBank sb = new SynchronizedBank();

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

        C c = new C();
        D d = new D();

        c.start();
        d.start();
    }
}
```