Synchronized

Synchronized

synchonized는스스로LOCK을건다.가장 확실하지만 무식한 방법에 가깝다.
1.synchronized를 사용해 함수자체에 LOCK을 건다.
2.synchronized를 사용하는 대신 객체를 하나 추가로 만든다.
3.synchronized함수는 두 가지 문제가있다.
1. 함수가 lock이 걸린다.
2. 함수를 포함한 객체(this)에 lock이 걸린다.
-> (객체를 유연하게 LOCK하기 위해 OBJECT사용하면됨) 예제확인

Solution : 함수단위가아닌synchronized를객체(this)에LOCK을건다. SyncBlock 예제 확인

```java
public class BasicSynchronization {
    private String mMessage;

    public static void main(String[] args) {
        BasicSynchronization temp = new BasicSynchronization();

        System.out.println("Test Start");

        new Thread(() ->{
            for (int i = 0; i < 100; i++) {
                temp.callMe( whoCallMe: "Thread_1");
            }
        }).start();

        new Thread(() ->{
            for (int i = 0; i < 100; i++) {
                temp.callMe( whoCallMe: "Thread_2");
            }
        }).start();
        System.out.println("Test end");
    }
    public synchronized void callMe(String whoCallMe) {
        mMessage = whoCallMe;

        try {
            long sleep = (long) (Math.random() * 100);
            Thread.sleep(sleep);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        if (!mMessage.equals(whoCallMe)) {
            System.out.println(whoCallMe + " | " + mMessage);
        }
```

```java
public class BasicSynchronization2 {
    private String mMessage;

    public static void main(String[] args) {
        BasicSynchronization temp1 = new BasicSynchronization();
        BasicSynchronization temp2 = new BasicSynchronization();

        System.out.println("Test Start");

        new Thread(() ->{
            for (int i = 0; i < 100; i++) {
                temp1.callMe( whoCallMe: "Thread_1");
            }
        }).start();

        new Thread(() ->{
            for (int i = 0; i < 100; i++) {
                temp2.callMe( whoCallMe: "Thread_2");
            }
        }).start();
        System.out.println("Test end");
    }
    public void callMe(String whoCallMe) {
        mMessage = whoCallMe;

        try {
            long sleep = (long) (Math.random() * 100);
            Thread.sleep(sleep);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        if (!mMessage.equals(whoCallMe)) {
            System.out.println(whoCallMe + " | " + mMessage);
```

예상 하겠지만 로그는 반복수만큼 출력된다.
좌측은 synchronized를 빼면 로그가 많이 출력될것이다. 우측은 객체가 다르기때문에 문제없다.

Solution_1

```java
//특정 값에 BLOCK을 걸어서 100개만 출력되는것을 확인할수있다.
import java.util.ArrayList;

public class SyncBlock1 {
    public ArrayList<Integer> mList = new ArrayList<>();

    public static void main(String[] args) throws InterruptedException {
        SyncBlock1 syncblock1 = new SyncBlock1();

        System.out.println("Test start");

        Thread t1 = new Thread(() ->{
            for (int i = 0; i < 100; i++) {
                syncblock1.add(i);
            }
        });

        Thread t2 = new Thread(() ->{
            for (int i = 0; i < 100; i++) {
                syncblock1.add(i);
            }
        });

        t1.start();
        t2.start();

        t1.join();
        t2.join();

        System.out.println(syncblock1.mList.size());
        System.out.println("Test end");
    }
```

특정 부분에 BLOCK을 설정한다.

```java
    public void add(int val) {
        /*code for synchronization is not needed*/
        synchronized (this) {
            if (!mList.contains(val)) {
                mList.add(val);
            }
        }
    }


}
```

Solution_2

```java
//LOCK의 주체가 this이기때문에 this로 걸려있는 동기화 block은 해당 lock이 풀릴때까지 대기해야한다.
import java.util.HashMap;

public class SyncBlock2 {
    private HashMap<String, String> abMap = new HashMap<>();
    private HashMap<String, String> cdMap = new HashMap<>();

    public static void main(String[] args) {
        SyncBlock2 syncBlock2 = new SyncBlock2();
        System.out.println("Test start");

        new Thread(() -> {
            for (int i = 0; i < 100; i++) {
                syncBlock2.putAB("a", "b");
                syncBlock2.getC( key: "c");
            }
        }).start();

        new Thread(() -> {
            for (int i = 0; i < 100; i++) {
                syncBlock2.putCD("c", "d");
                syncBlock2.getA( key: "a");
            }
        }).start();
        System.out.println("Test end");
    }
```

여기서 this로 인한 문제

```java
// this에 걸려버리면 자체 LOCK이 풀릴때까지 대기해야한다.
// 정작 LOCK이 필요한건 같은 hasmap을 동시에 접근하는경우 putAB와 getA 또 putCD getC가 각각 LOCK 이걸려야 효율적이다.
public void putAB(String key, String value) {
    synchronized (this) {
        abMap.put(key, value);
    }
}

public String getC(String key) {
    synchronized (this) {
        return cdMap.get(key);
    }
}
public void putCD(String key, String value) {
    synchronized (this) {
        abMap.put(key, value);
    }
}

public String getA(String key) {
    synchronized (this) {
        return cdMap.get(key);
    }
}
```

Solution_3

```java
//this가 아닌 object1과 object2 객체를 만들어 this가 아닌 동시에 lock 걸려야 하는 부분을 따로 지정해 줄 수 있다!!!

import java.util.HashMap;

public class SyncBlock3 {
    private HashMap<String, String> abMap = new HashMap<>();
    private HashMap<String, String> cdMap = new HashMap<>();

    private final Object object1 = new Object();
    private final Object object2 = new Object();

    public static void main(String[] args) {
        SyncBlock2 syncBlock2 = new SyncBlock2();
        System.out.println("Test start");

        new Thread(() -> {
            for (int i = 0; i < 100; i++) {
                syncBlock2.putAB("a", "b");
                syncBlock2.getC( key: "c");
            }
        }).start();

        new Thread(() -> {
            for (int i = 0; i < 100; i++) {
                syncBlock2.putCD("c", "d");
                syncBlock2.getA( key: "a");
            }
        }).start();
        System.out.println("Test end");
    }
```

여기서 OBJECT로 해결!!

```java
}
// this에 걸려버리면 자체 LOCK이 풀릴때까지 대기해야한다.
// 정작 LOCK이 필요한건 같은 hasmap을 동시에 접근하는경우 putAB와 getA 또 putCD getC가 각각 LOCK 이걸려야 효율적이다.
public void putAB(String key, String value) {
    synchronized (object1) {
        abMap.put(key, value);
    }
}

public String getC(String key) {
    synchronized (object1) {
        return cdMap.get(key);
    }
}
public void putCD(String key, String value) {
    synchronized (object2) {
        abMap.put(key, value);
    }
}

public String getA(String key) {
    synchronized (object2) {
        return cdMap.get(key);
    }
}
```