Synchronization Ⅱ - synchronized Block

쓰레드의 동기화 기법2 - synchronized 기반 동기화 블록

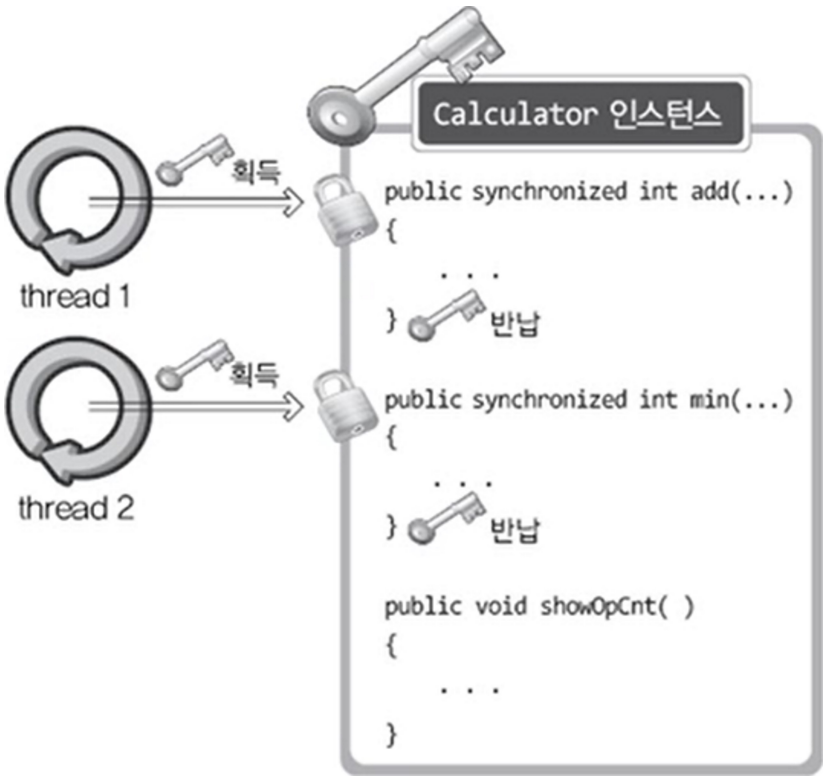
- 여러 문장을 가진 메소드를 동기화하면 메소드안의 문장 전체에 대해서 하나의 쓰레드만 실행 가능하기 때문에 엄청난 성능 저하가 일어난다. 그래서 동기화가 필요한 최소한의 문장만 Synchronized 선언 할 수 있다.

```
class Calculator{
     private int opCnt = 0;

     public int add(int n1. int n2){

          Synchronized(this){
               opCnt++;
          }

          return n1+n2;
     }

     public int min(int n1. int n2){

          Synchronized(this){
               opCnt++;
          }

          return n1-n2;
     }
}
```

---

synchronized 기반 동기화 메소드의 정확한 이해

Java의 모든 인스턴스를 대상으로 LOCK을 걸 수 있다! (자물쇠로 잠글 수 있다.)



- Synchronized 선언이 된 모든 메소드는 LOCK이 걸린다. (자물쇠가 채워진다)
- 쓰레드는 열쇠를 획득해야지만 메소드를 호출, 실행할 수 있다.

- Synchronized 메소드가 정의되어 있는 인스턴스가 열쇠 하나만을 가지고 있다!
- ★ 열쇠는 하나이기 때문에, 쓰레드가 열쇠를 획득하는 순간 다른 쓰레드는 모든 synchoronized 메소드 실행 불가!
  (모든 자물쇠를 열 수 있는 열쇠는 하나이므로)

  - Thread1이 열쇠를 획득하여 add 메소드 실행
  - min 메소드는 다른 쓰레드가 사용하지 않는데, Thread2가
    min 메소드를 실행하려고 하지만 열쇠가 없으므로 실행 불가!

- Synchronized 메소드의 호출과 종료과정에서의 메소드와 인스턴스 간의
  열쇠 획득과 반납 과정 때문에 리소스 소모가 많이 일어난다.

- Synchronized 기반 동기화 메소드는 위와 같은 여러가지 이유로 사용을 가급적 제한해야 한다.
  → Synchoronized 기반의 동기화 블록을 이용한 섬세한 제어가 필요!

```
class Calculator{
     private int opCnt = 0;

     // 열쇠와 상관없이 메소드 호출 진행 OK!
     public int add(int n1. int n2){

          // 여기에 자물쇠가 채워진다.
          Synchronized(this){
               opCnt++;
          }

          return n1+n2;
     }

     // 열쇠와 상관없이 메소드 호출 진행 OK!
     public int min(int n1. int n2){

          // 여기에 자물쇠가 채워진다.
          Synchronized(this){
               opCnt++;
          }

          return n1-n2;
     }
}
```

  - this : 열쇠정보, this는 인스턴스 자신을 의미하므로, 인스턴스 자신이 가지고 있는 열쇠를 의미한다!
    열쇠는 하나이기 때문에, 쓰레드가 열쇠를 획득하는 순간 다른 쓰레드는 모든 synchoronized 블록 실행불가!

---

synchronized 기반 동기화 블록의 예제

- 4개의 블록이 하나의 key로만 접근이 이루어지면 4개의 블록은 모두 동시에 실행 불가,
  하지만 key1, key2의 두개의 키로 해당 key영역을 제외한 나머지 영역에 영향을 미치지 않게 할 수 있다.
  (예를 들어 key1 블록이 실행되더라도 key2 블록은 영향없이 실행가능!)

- 아래의 예제는 이해를 돕기위해 key1, key2 인스턴스가 있지만 일반적으로는 this, key1, key2...로 사용한다.

(이미 해당 클래스의 인스턴스 자신(this)도 키를 하나 가지고 있으므로)

```
public class IHaveTwoNum {
	private int num1 = 0; // 동기화가 필요한 변수 1
	private int num2 = 0; // 동기화가 필요한 변수 2

	Object key1 = new Object();
	Object key2 = new Object();

	// key1 인스턴스의 열쇠로 접근 가능한 블록
	public void addOneNum1(){
		synchronized(key1) { num1 += 1; }
	}

	public void addTwoNum1(){
		synchronized(key1) { num1 += 2; }
	}

	// key2 인스턴스의 열쇠로 접근 가능한 블록
	public void addOneNum2(){
		synchronized(key2) { num2 += 1; }}

	public void addTwoNum2(){
		synchronized (key2) { num2 += 2; }
	}

	public void showAllNum(){
		System.out.println("num1 : "+num1);
		System.out.println("num2 : "+num2);
	}
}

public class AccessThread extends Thread {
	IHaveTwoNum twoNumInst;

	public AccessThread(IHaveTwoNum inst){ twoNumInst = inst; }

	@Override
	public void run(){
		twoNumInst.addOneNum1();
		twoNumInst.addTwoNum1();
		twoNumInst.addOneNum2();
		twoNumInst.addTwoNum2();
	}
}

public class SyncObjectKey {
	public static void main(String[] args) {
		IHaveTwoNum numInst = new IHaveTwoNum();

		AccessThread at1 = new AccessThread(numInst);
```

```
            AccessThread at2 = new AccessThread(numInst);

            at1.start();
            at2.start();

            try{ at1.join(); at2.join(); }catch(InterruptedException e){ e.printStackTrace(); }

            numInst.showAllNum();
        }
}
```
