1) before

```
class 클라이언트 {

    public void run() {
        MyLinkedList list = new MyLinkedList();
        doSomething1(list);
        doSomething2(list);
    }

    public void doSomething1(MyLinkedList list) {
        list.add(3);
        list.remove(4);
    }

    public void doSomething2(MyLinkedList list) {
        list.add(5);
        list.remove(6);
    }
}

class MyLinkedList {
    public void add(int value) {
            ......
        }

        public void remove(int value) [
            ......
        }
}
```

2) 클라이언트에 구현

```
class 클라이언트 {
    Lock lock = new Lock();

    public void run() {
        MyLinkedList list = new MyLinkedList();
        doSomething1(list);
        doSomething2(list);
    }

    public void doSomething1(MyLinkedList list) {
        lock.lock();
        list.add(3);
        list.remove(4);
        lock.unlock();
    }

    public void doSomething2(MyLinkedList list) {
        lock.lock();
        list.add(5);
        list.remove(6);
        lock.unlock();
    }
}

class MyLinkedList {
    public void add(int value) {
        ......
    }

    public void remove(int value) [
        ......
    }
}

class Lock {
    public void lock() { ...... }
    public void unlock() { ...... }
}
```

3) 서비스 클래스에 구현

```
class 클라이언트 {

    public void run() {
        MyLinkedList list = new MyLinkedList();
        doSomething1(list);
        doSomething2(list);
    }

    public void doSomething1(MyLinkedList list) {
        list.add(3);
        list.remove(4);
    }

    public void doSomething2(MyLinkedList list) {
        list.add(5);
        list.remove(6);
    }
}

class MyLinkedList {
    Lock lock = new Lock();

    public void add(int value) {
        lock.lock();
        ......
        lock.unlock();
    }

    public void remove(int value) [
        lock.lock();
        ......
        lock.unlock();
    }
}

class Lock {
    public void lock() { ..... }
    public void unlock() { ..... }
}
```

4) 동기화 제어 객체 분리

```
class 클라이언트 {

    public void run() {
        Synchronization list = new Synchronization ();
        doSomething1(list);
        doSomething2(list);
    }

    public void doSomething1(Synchronization list) {
        list.add(3);
        list.remove(4);
    }

    public void doSomething2(Synchronization list) {
        list.add(5);
        list.remove(6);
    }
}

class Synchronization {
    MyLinkedList list = new MyLinkedList();
    Lock lock = new Lock();

    public void add(int value) {
        lock.lock();
        list.add(value);
        lock.unlock();
    }

    public void remove(int value) [
        lock.lock();
        list.remove(value);
        lock.unlock();
    }
}

class MyLinkedList {
    public void add(int value) {
        ......
    }

    public void remove(int value) [
        ......
    }
}

class Lock {
    public void lock() { ..... }
    public void unlock() { ..... }
}
```

5) 다형성 구현 #1

클라이언트의 세금계산 메소드 호출을 다형성 호출로 개선

```
class 클라이언트 {

    public void run() {
        MyList list = new SynchronizedList();
        doSomething1(list);
        doSomething2(list);
    }

    public void doSomething1(MyList list) {
        list.add(3);
        list.remove(4);
    }

    public void doSomething2(MyList list) {
        list.add(5);
        list.remove(6);
    }
}

interface MyList {
    void add(int value);
    void remove(int value);
}

class SynchronizedList implements MyList {
    MyLinkedList list = new MyLinkedList();
    Lock lock = new Lock();

    public void add(int value) {
        lock.lock();
        list.add(value);
        lock.unlock();
    }

    public void remove(int value) [
        lock.lock();
        list.remove(value);
        lock.unlock();
    }
}

class MyLinkedList implements MyList {
    public void add(int value) {
        ......
    }

    public void remove(int value) [
        ......
    }
}

class Lock {
    public void lock() { ..... }
    public void unlock() { ..... }
}
```

6) 다형성 구현 #2

SynchronizedList 클래스의 MyLinkedList 메소드 호출을 다형성 호출로 개선

```
class 클라이언트 {

    public void run() {
        MyList list = new SynchronizedList();
        doSomething1(list);
        doSomething2(list);
    }

    public void doSomething1(MyList list) {
        list.add(3);
        list.remove(4);
    }

    public void doSomething2(MyList list) {
        list.add(5);
        list.remove(6);
    }
}

interface MyList {
    void add(int value);
    void remove(int value);
}

class SynchronizedList implements MyList {
    MyList list = new MyLinkedList();
    Lock lock = new Lock();

    public void add(int value) {
        lock.lock();
        list.add(value);
        lock.unlock();
    }

    public void remove(int value) [
        lock.lock();
        list.remove(value);
        lock.unlock();
    }
}

class MyLinkedList implements MyList {
    public void add(int value) {
        ......
    }

    public void remove(int value) [
        ......
    }
}

class Lock {
    public void lock() { ..... }
    public void unlock() { ..... }
}
```

7) Proxy 패턴

```
class 클라이언트 {
    MyList list

    public 클라이언트(MyList list) {
        this.list = list;
    }

    public void run() {
        doSomething1(list);
        doSomething2(list);
    }

    public void doSomething1(MyList list) {
        list.add(3);
        list.remove(4);
    }

    public void doSomething2(MyList list) {
        list.add(5);
        list.remove(6);
    }
}

interface MyList {
    void add(int value);
    void remove(int value);
}

class SynchronizedList implements MyList {
    MyList list;
    Lock lock = new Lock();

    public SynchronizedList(MyList list) {
        this.list = list;
    }

    public void add(int value) {
        lock.lock();
        list.add(value);
        lock.unlock();
    }

    public void remove(int value) [
        lock.lock();
        list.remove(value);
        lock.unlock();
    }
}

class MyLinkedList implements MyList {
    public void add(int value) {
        ......
    }

    public void remove(int value) [
        ......
    }
```

```
}

class MyArrayList implements MyList {
    public void add(int value) {
        ......
    }

    public void remove(int value) [
        ......
    }
}

///// 객체 조립 사례
클라이언트 client1 = new 클라이언트(new MyLinkedList());
client1.run();

클라이언트 client2 = new 클라이언트(new MySynchronizedList(new MyLinkedList()));
client2.run();

클라이언트 client3 = new 클라이언트(new MySynchronizedList(new MyArrayList()));
client3.run();
```

클라이언트 코드는 MyList 인터페이스만 참조한다.
   아래 계층의 클래스를 참조하지 않는다.
   MySynchronized, MyLinkedList, MyArrayList 클래스를 참조하지 않는다.

MySynchronized 클래스는 MyList 인터페이스만 참조한다.
   아래 계층의 클래스를 참조하지 않는다.
   MyLinkedList, MyArrayList 클래스를 참조하지 않는다.

8) java 언어의 synchronized method

```
class SynchronizedList implements MyList {
    MyList list;
    Lock lock = new Lock();

    public SynchronizedList(MyList list) {
        this.list = list;
    }

    public void add(int value) {
        lock.lock();
        list.add(value);
        lock.unlock();
    }

    public void remove(int value) [
        lock.lock();
        list.remove(value);
        lock.unlock();
    }
}
```

java 언어의 synchronized method 문법을 활용하여 다음과 같이 구현할 수 있다.

```
class SynchronizedList implements MyList {
    MyList list;

    public SynchronizedList(MyList list) {
        this.list = list;
    }

    public synchronized void add(int value) {
        list.add(value);
    }

    public synchronized void remove(int value) [
        list.remove(value);
    }
}
```