Composing Thread-Safe Objects

Jesper Öqvist

Object State

All objects with mutable state are potentially non thread safe.

To guarrantee thread safety a synchronization policy is needed. The policy documents how objects are protected from invalid state transitions.

LUND UNIVERSITY

Confinement

A good synchronization policy requires some form of confinement:

- Thread confinement
- Instance confinement

With proper confinement we do not need to analyze the whole program for synchronization problems.

LUND UNIVERSITY

Instance Confinement

Instance Confinement (or just confinement) means keep all state to yourself!

LUND UNIVERSITY

Instance Confinement

However, we often want to share state. This can be tricky:

```
class A {
    private final B b = new B();
    public B getB() {
        return b;
    }
    ...
}
```

The thread-safetyness of A depends on the implementation of B.

Instance Confinement

Even trickier:

```
class A {
    private final B b1 = new B();
    private final B b2 = new B();
    public B getB() {
        return b1;
    }
    ...
}
```

Is `b1` independent?!

How to share internal state?

- Only publish immutable copies of the internal state
- OR: Only return mutable *independent* state
- OR: Be absolutely certain the returned object can be modified at any time

Composing Objects

Say we want to create a new atomic operation using a thread safe class. This can be done using encapsulation:

```
public class MyList implements List<T> {
    private final List<T> list;
    public MyList(List<T> list) {
        this.list = list;
    }
    public synchronized void addIfAbsent(T item) {
        if (list.contains(item)) {
            list.add(item);
        }
    }
    ...
}
```

LUND UNIVERSITY

Composing Objects

All methods of the `List` interface must be implemented. This is done simply by delegating to the underlying list object.
All modifications of the list must be synchronized with the same lock used for the `addIfAbsent` operation (intrinsic lock).

Conclusion

Use monitors everywhere.

LUND UNIVERSITY

Exercises

- Implement your own `ConcurrentHashMap` using `java.util.HashMap`. Document your synchronization policy.
- Optionally: Implement an iterator that allows iterating the values of your map during concurrent modification.
- **OR**: Find a concurrency error in chapter 4 of Goetz.