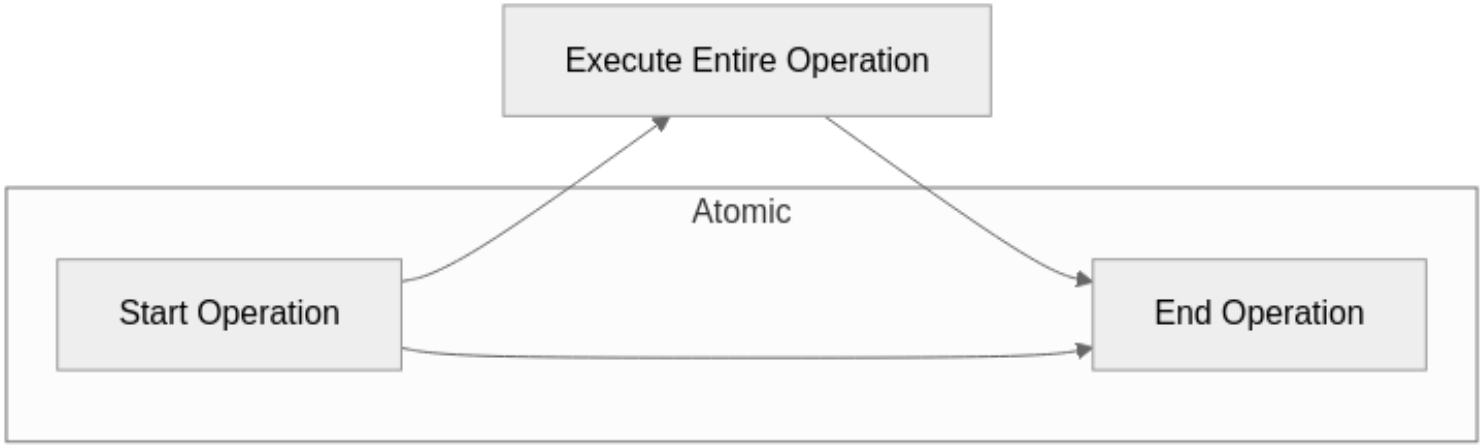
Thread Safety

What Is Atomicity?

Atomicity means operations occur as one indivisible step.



Example: UnsafeCountingFactorizer

```
@NotThreadSafe
public class UnsafeCountingFactorizer implements Servlet {
    private long count = 0;
    public void service(ServletRequest req, ServletResponse
        ++count;
    }
}
```

Example: Atomic Using AtomicLong

```
@ThreadSafe
public class CountingFactorizer implements Servlet {
    private final AtomicLong count = new AtomicLong(0);
    public void service(ServletRequest req, ServletResponse
        count.incrementAndGet();
    }
}
```

Example: Synchronized Counter

```
public class SynchronizedCounter {
    private int count = 0;
    public synchronized void increment() { count++; }
    public synchronized int get() { return count; }
}
```

Example: Atomic Compound Action (BankAccount)

```
public class BankAccount {
    private int balance;
    private int tx;
    public synchronized void deposit(int amt) {
        balance += amt;
        tx++;
    }
}
```

Example: Lazy Init Race

```
@NotThreadSafe
public class LazyInitRace {
    private ExpensiveObject instance;
    public ExpensiveObject getInstance() {
        if (instance == null)
            instance = new ExpensiveObject();
        return instance;
    }
}
```

Example: Safe Lazy Init

```
public synchronized ExpensiveObject getInstance() {
    if (instance == null)
        instance = new ExpensiveObject();
    return instance;
}
```

Example: AtomicReference Not Enough

```
AtomicReference<String> first = ...;
AtomicReference<String> last = ...;
public void update(String f, String l) {
    first.set(f);
    last.set(l);
}
```

Summary

- Single-variable atomicity: use atomic classes
- Multi-variable atomicity: use locks
- Check-then-act: needs atomicity - use locks