



Compilers

Java Threads

Java Threads

- Java has concurrency built in through threads
 - Each thread has its own program counter & stack



- Thread objects have class Thread
 - Start and stop methods

- Synchronization obtains a lock on the object:

→ synchronized (x) { e }

- In synchronized methods, this is locked

synchronized f() { }

scheduler

→ pick a thread
• execute 1 statement

lock x
evaluate e
unlock x

Java Threads

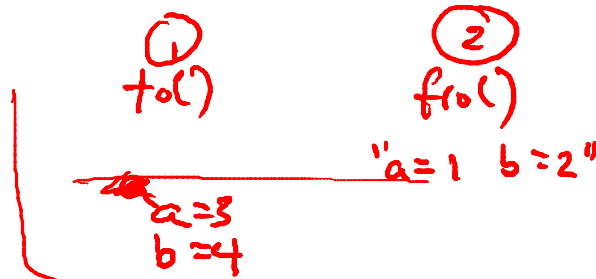
```
class Simple {
```

```
    int a = 1, b = 2;
```

```
    void to() { a = 3; b = 4; }
```

```
    void fro() { println("a= " + a + ", b= " + b); }
```

```
}
```



Thread 1 (to()):
a=3
b=4

Thread 2 (fro()):
"a=3 b=4"

Thread 1 (to()):
a=3
b=4

Thread 2 (fro()):
"a=3 b=2"

Two threads call to() and fro(). What is printed?

Java Threads

s

```
class Simple {  
    int a = 1, b = 2;
```

```
    ➔ void synchronized to() { a = 3; b = 4; }
```

```
    ➔ void fro() { println("a= " + a + ", b=" + b); }  
}
```

①	②
to()	fro()
→ lock s	
a=3	
<hr/>	
"a=3 b=2"	
<hr/>	
b=4	
unlock s	

Two threads call `to()` and `fro()`. What is printed?

```
class Simple {
```

```
    int a = 1, b = 2;
```

```
    → void synchronized to() { a = 3; b = 4; }
```

```
    → void synchronized fro() {println("a= " + a + ", b=" + b); }  
}
```

"a = 1 b = 2"
fro() before to()

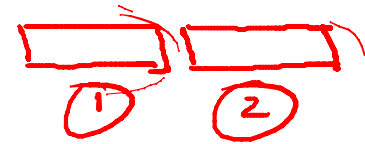
"a = 3 b = 4"
to() before fro()

Two threads call `to()` and `fro()`. What is printed?

double → ^① a = 3.14, ^② a = 2.78
high a = ...
low a = ...
a = 3.14 or 2.78
3.78? → No?

- Even without synchronization, a variable should only hold values written by some thread

"out of thin air value"



→ – Writes of values are atomic

→ – Violated for doubles, though

volatile

- Java concurrency semantics are difficult to understand in detail, particularly as to how they might be implemented on certain machines