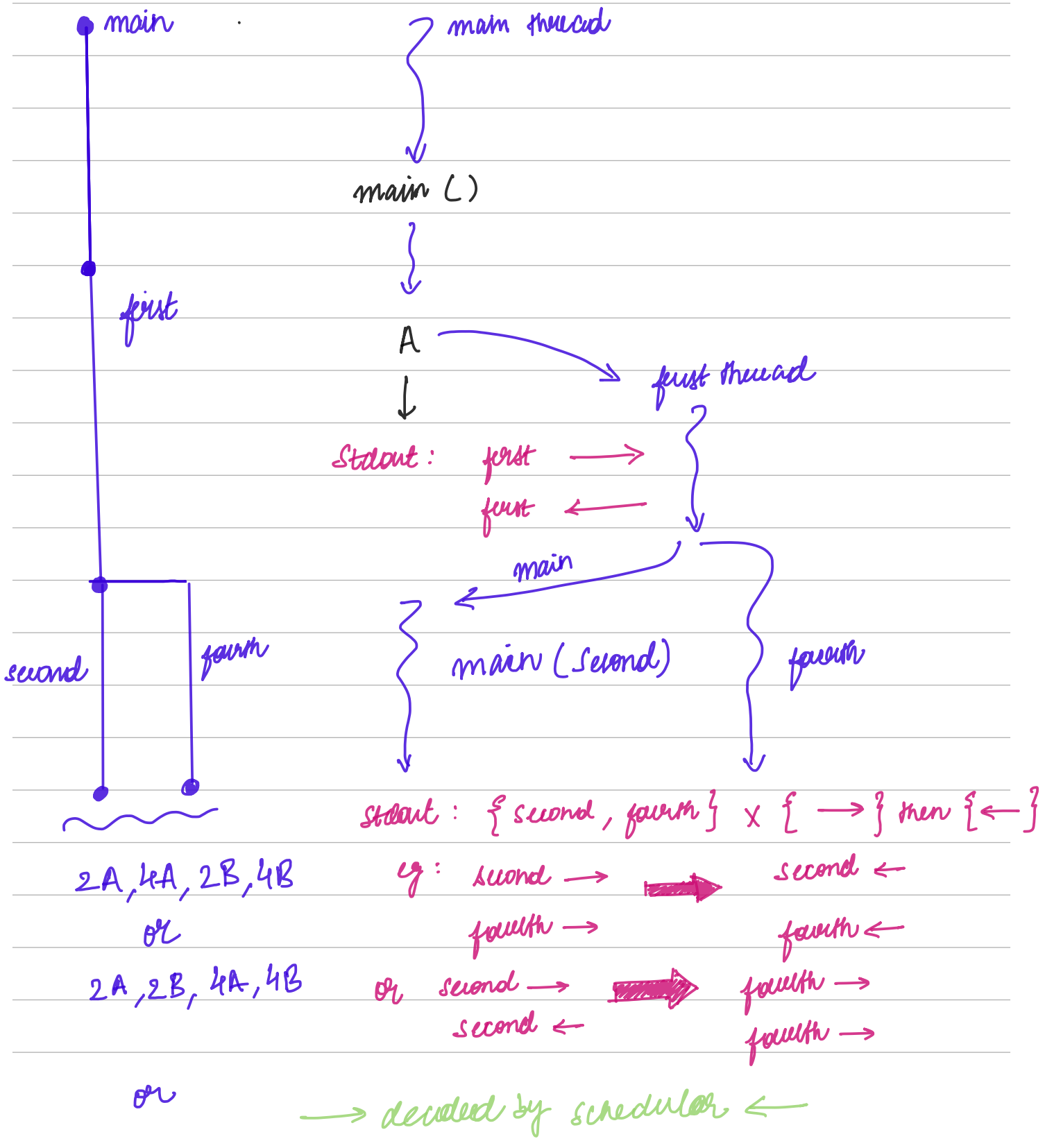
Shreesh Tripathi
Kyle Barke

Homework - 10.1

Q1)

```
aT4_0.run();   // first    (A)
aT4_1.run();   // second   (B)
aT4_2.start(); // third    (C)
```



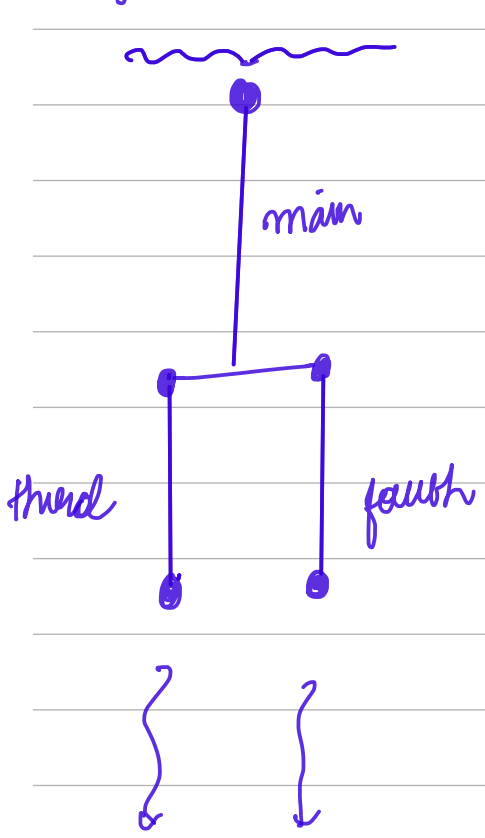
4A, 2A, 4B, 2B or fourth → fourth ←
second → second ←

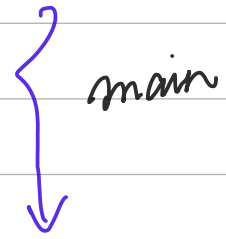
or

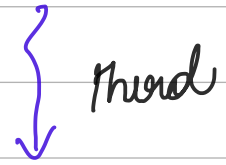
4A, 4B, 2A, 2B or fourth → second →
fourth → second ←

or 2A, 4A | 2A, 2B
then
thread 3

or second → | second ←







stdout: third →
third ←
fourth →
fourth ←

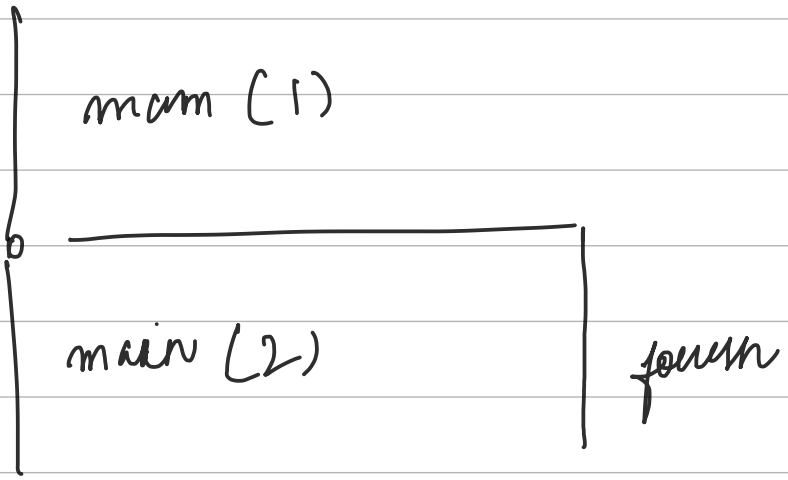
Ans output not possible bc at least only one main thread exists and until it reaches the end of first. run() another thread (fourth) is not initialized

```java
public void run () {
// This will make fourth thread finish after second thread (run by main)
if (info.equals("second")){try {Thread.sleep(1000);} catch (Exception e) {}}
System.err.println(info + " ---> ");
System.err.println(info + " <--- ");

if ( info == "first" )
    new ThreadQuestionOne("fourth").start();
}
```

Now the scheduler has two threads in queue main (running second) and fourth thread. However, due to the sleep timer in second thread the scheduler will prioritize fourth thread > main thread (running second thread) resulting in the following output:

```
first --->
first <---
fourth --->
fourth <---
second --->
second <---
third --->
third <---
```

In this case, third thread will always execute afer second thread (run by main thread) because it cant but init until main thread terminates the executation of the second thread to start the third thread and queue it at the scheduler, however, it can be executed before the fourth thread as shown later



However now thread 4 can interfere w/ main (2) or later

when third thread is also initialized

depends on
how the scheduler manages
main(second) & fourth
Or
third & fourth

```
if (info.equals("fourth")){try {Thread.sleep(1000);} catch (Exception e) {}}
```

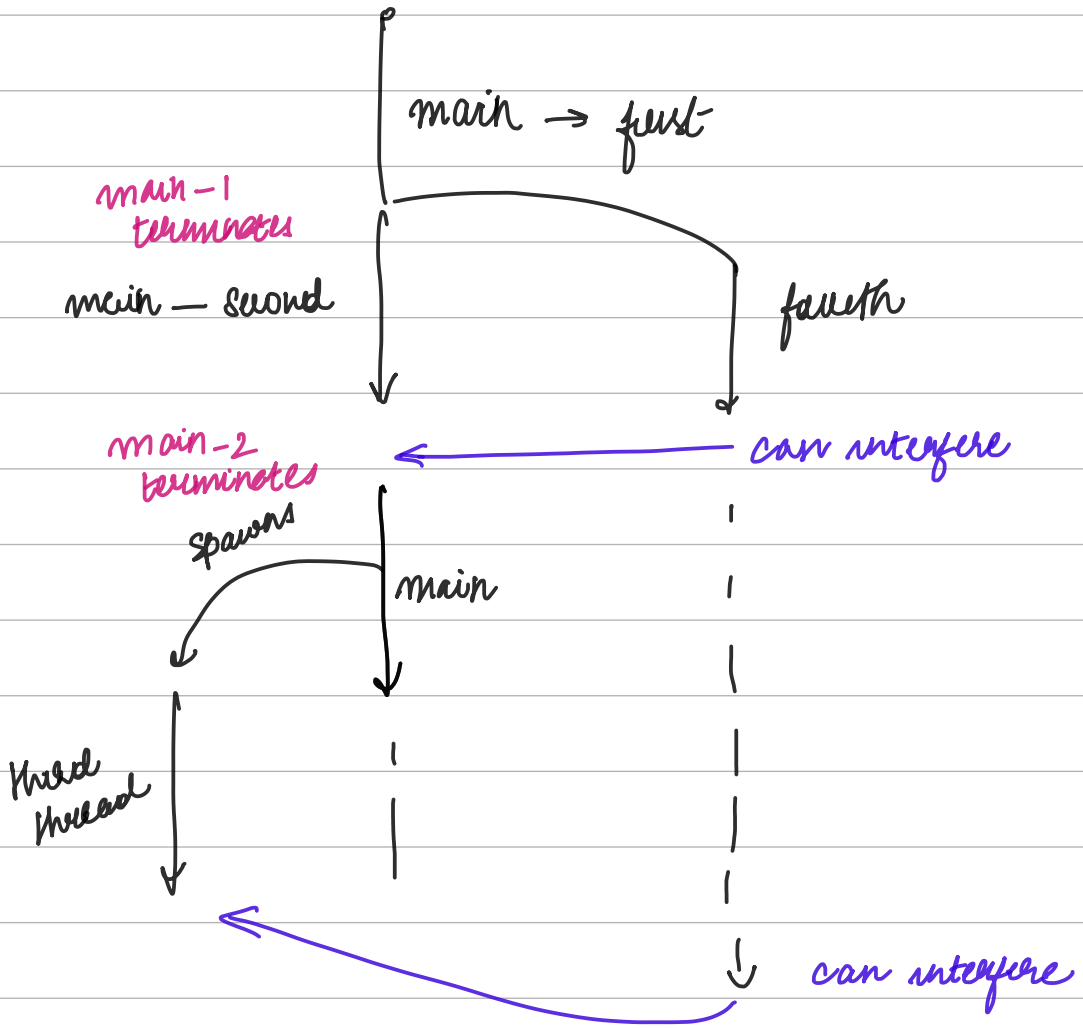
```
public void run () {
// This will make fourth thread finish last (after main - second and third thread)
if (info.equals("fourth")){try {Thread.sleep(1000);} catch (Exception e) {}}
System.err.println(info + " ---> ");
System.err.println(info + " <--- ");

if ( info == "first" )
    new ThreadQuestionOne("fourth").start();
}
```

Will cause scheduler to proritize main thread and subsequently third thread over fourth thread resulting in the following output

```
first --->
first <---
second --->
second <---
third --->
third <---
fourth --->
fourth <---
```

However, third thread will never be executed before the second thread since second thread is actually explicitly taken care of by the main thread so at a given time, scheduler only has the main thread to worry about Once the scheduler allows main thread to take care of second thread and it terminates, third thread is spawned independent of the main thread

Finally



Conclusion 1: first 1 always first

Conclusion 2: third thread always after second thread

Conclusion 3: fourth thread can clash with second thread, can be after second thread or can clash with third thread or can be after third thread

Q2) Homework 10.2

Shreesh Tripathi
Kyle Burke

The main thread spawns a new thread with id = 1 and aStaticInt is set to 1

Since id == 1, another thread object with id = 2 is made however thread 1 takes care of the execution of thread 2 run method

But since the .run() method is used, thread 2 will not run concurrently but as a regular method call on the second thread (recently init)

Since this is a regular call, until the .run() method of thread 2 isn't executed, thread 1 constructor call wouldn't be complete

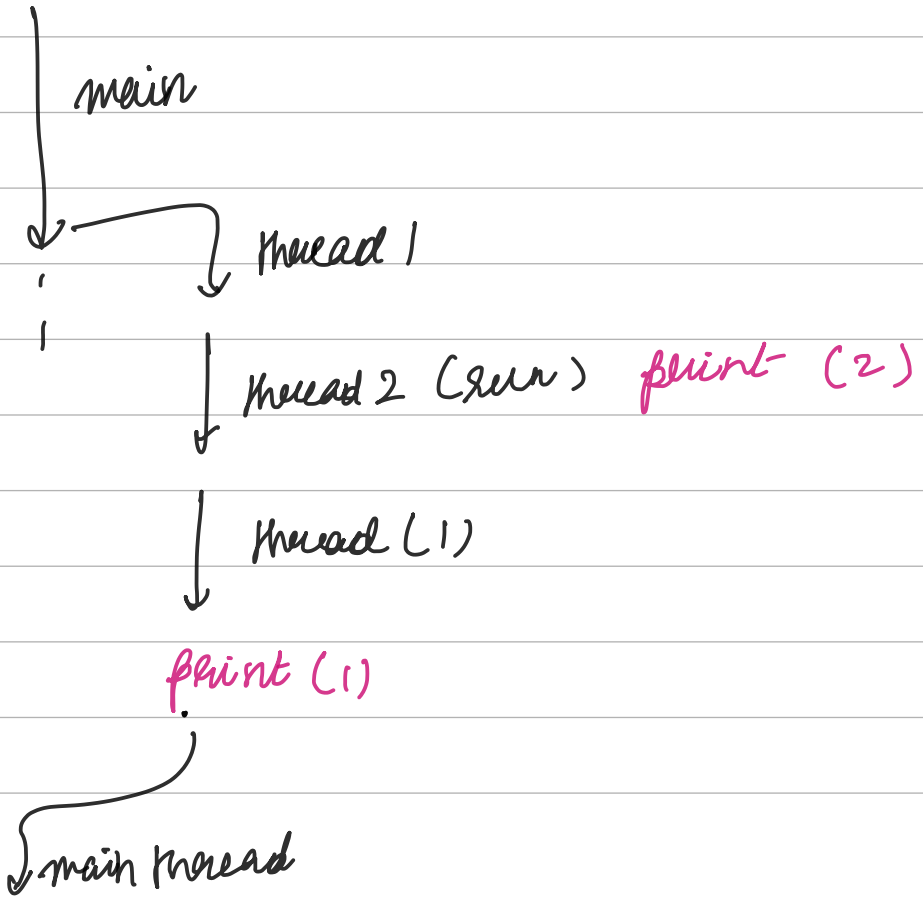
and until its not complete, thread 1's run method cannot by queued by scheduler until thread 2 run() method is complete

Thread 2 run method finally ends and thread 1 is now finally spawned as a thread other than main which is queued at the compiler and finally executed

the control flow returns to the main method and program terminates

```
2 ----->
id/aStaticInt = 2/2
2 <-----
1 ----->
id/aStaticInt = 1/2
1 <-----
```

only output possible



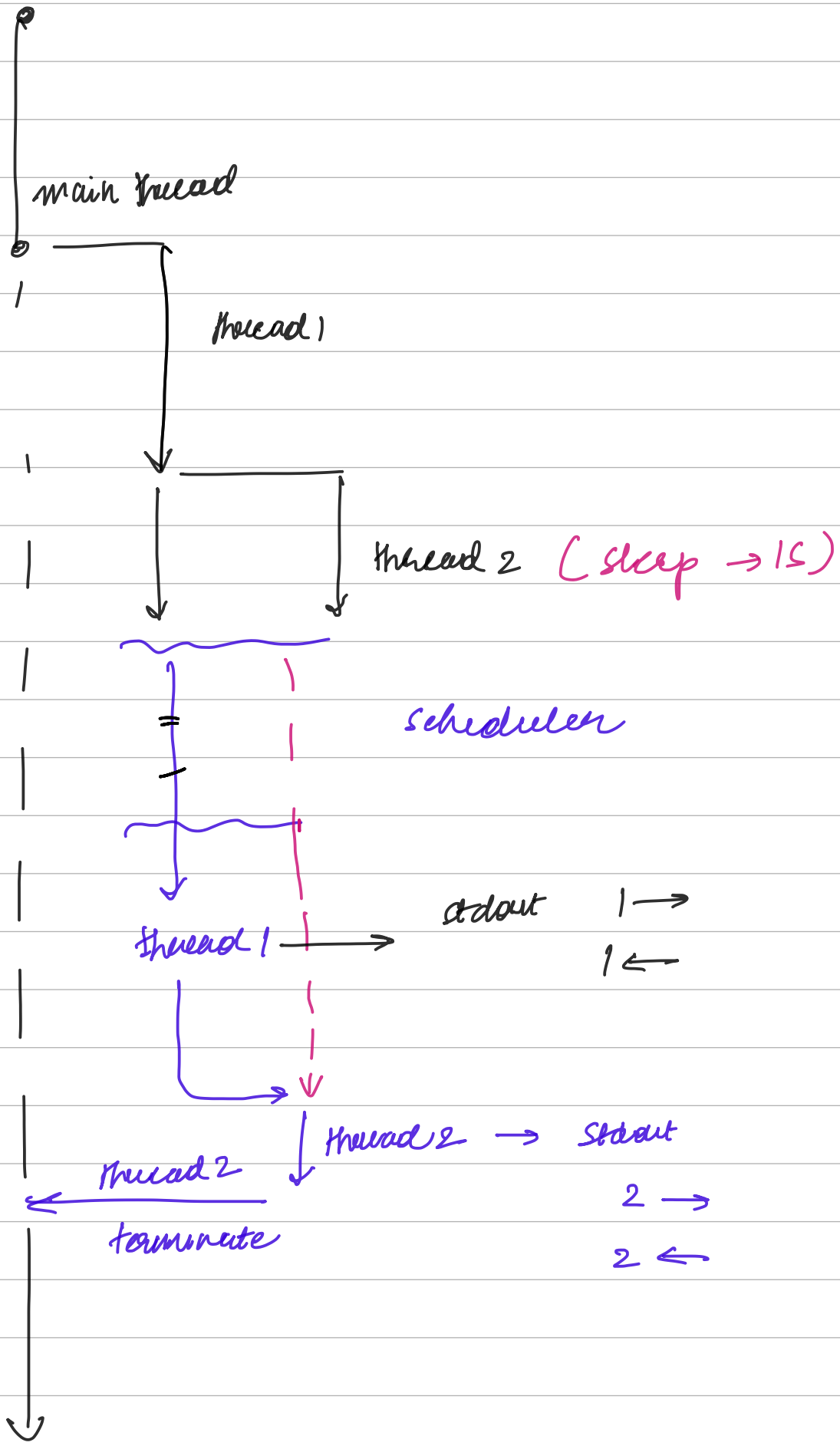
fin

```java
public class HurlyBurly extends Thread {

    static int aStaticInt;
    int id;

    HurlyBurly(int id) {
    this.id = id;
    aStaticInt = id;
    if ( id == 1 )
        // Fix to get desired output
        new HurlyBurly(2).start();
    }

    public void run()   {
    // Fix to get make it more likely to get the desired output

    if ( id == 2 ) try {Thread.sleep(1000);} catch(Exception e) {}

    System.out.println( id + " -----> ");
    System.out.println("id/aStaticInt = " + id + "/" + aStaticInt );
    System.out.println( id + " <----- ");
    }

    public static void main( String[] args ) {
    new HurlyBurly(1).start();
    }
}
```

change run → start so a new thread is spawned
2 → main → thread 1 → thread 2
make thread 2 sleep so scheduler allows thread 1

RESULT

```
1 -----> 
id/aStaticInt = 1/2
1 <-----
2 -----> 
id/aStaticInt = 2/2
2 <-----
```


main thread
thread 1
thread 2 (sleep → 1s)
scheduler
thread 1 →
stdout 1 →
1 ←
thread 2 → stdout
thread 2
terminate
2 →
2 ←