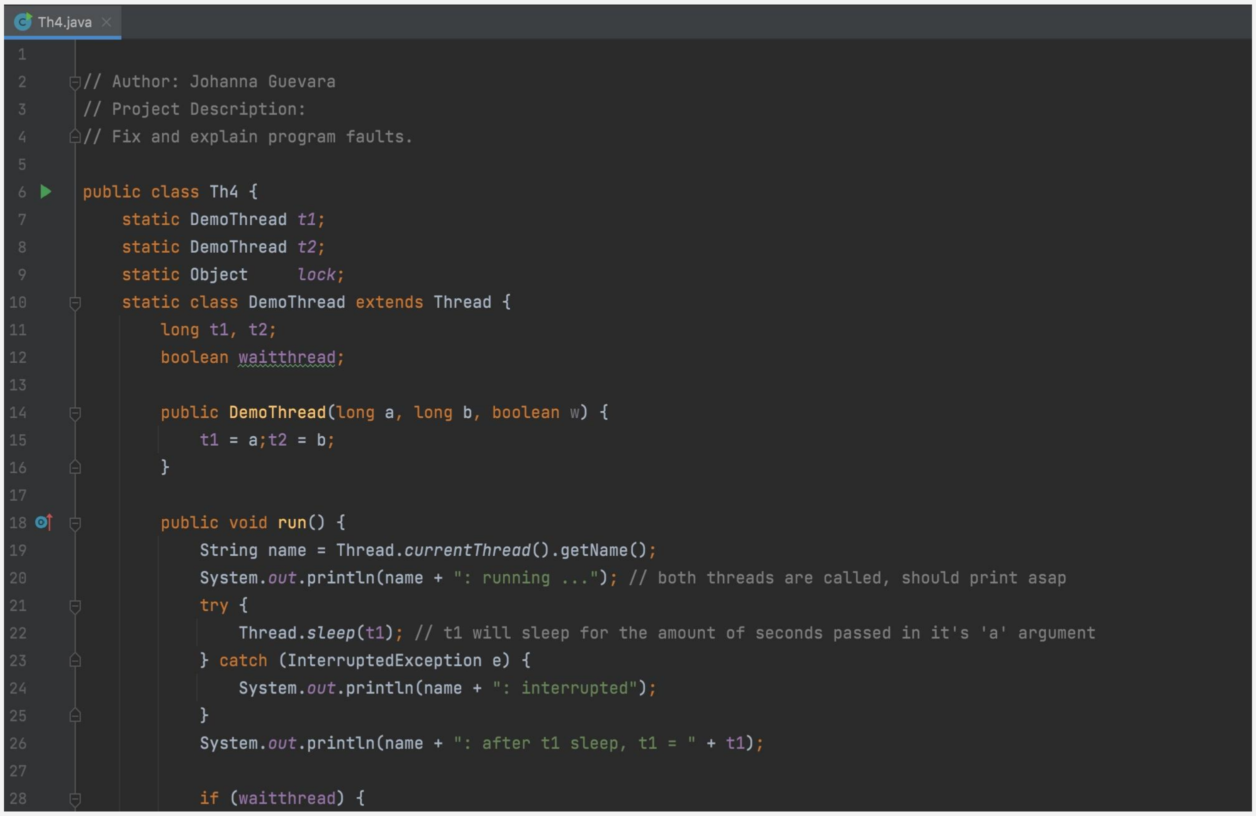
Assignment 3: Threads

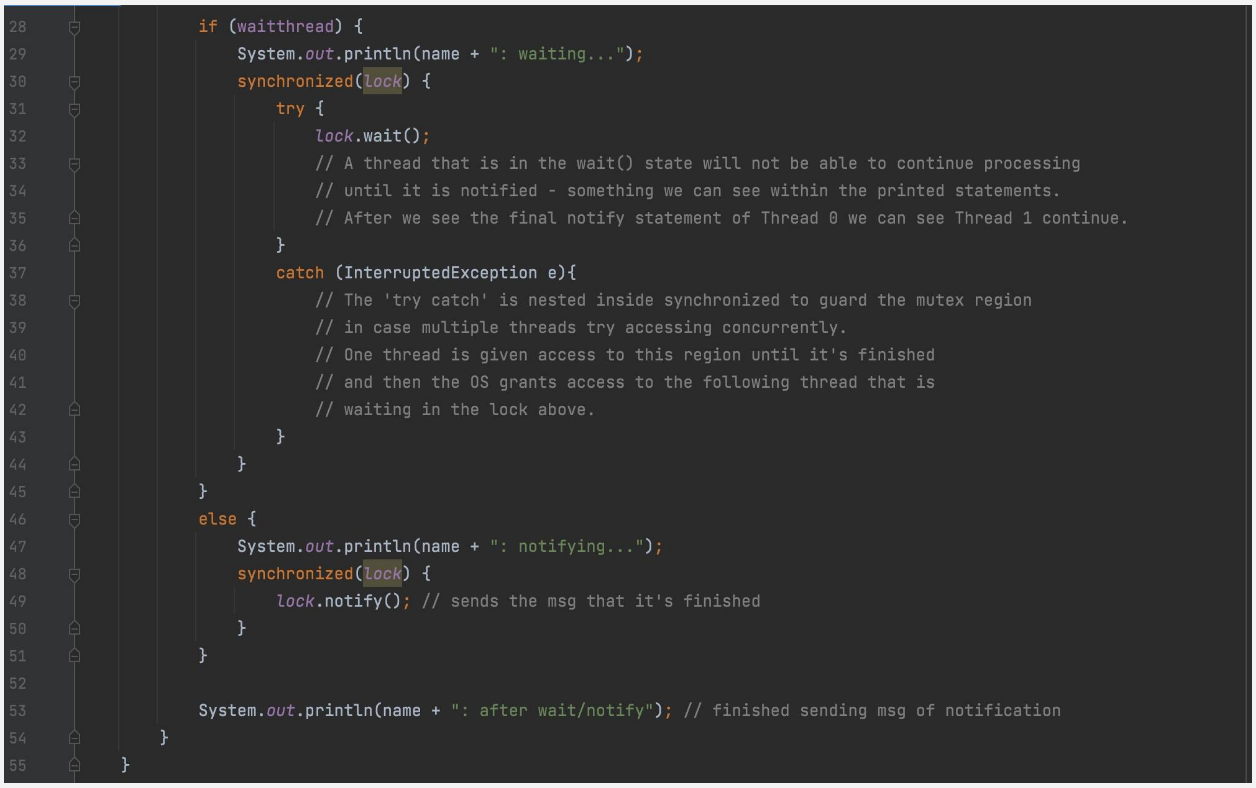
Instructions

It missed some codes, please fix it and add // for all lines and explain what the program does.

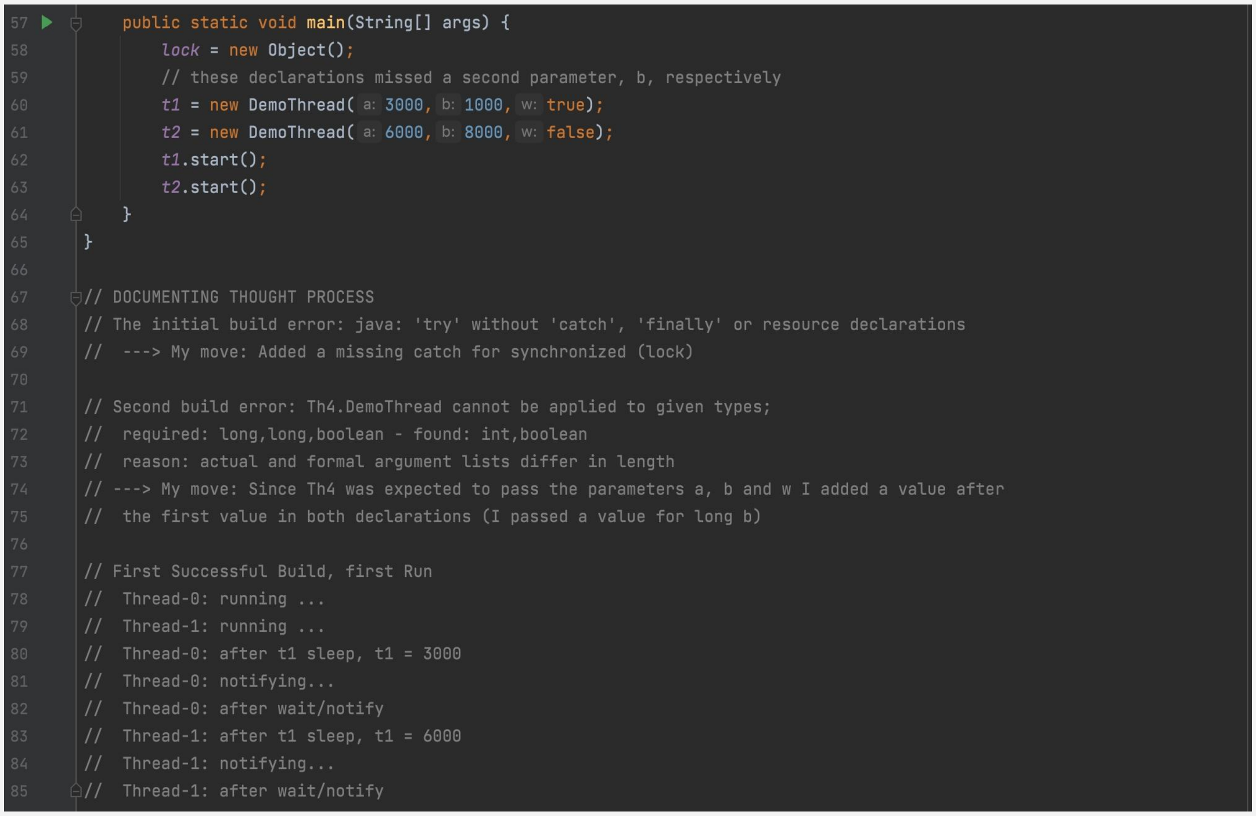
Code



```java

// Author: Johanna Guevara
// Project Description:
// Fix and explain program faults.

public class Th4 {
    static DemoThread t1;
    static DemoThread t2;
    static Object     lock;
    static class DemoThread extends Thread {
        long t1, t2;
        boolean waitthread;

        public DemoThread(long a, long b, boolean w) {
            t1 = a;t2 = b;
        }

        public void run() {
            String name = Thread.currentThread().getName();
            System.out.println(name + ": running ..."); // both threads are called, should print asap
            try {
                Thread.sleep(t1); // t1 will sleep for the amount of seconds passed in it's 'a' argument
            } catch (InterruptedException e) {
                System.out.println(name + ": interrupted");
            }
            System.out.println(name + ": after t1 sleep, t1 = " + t1);

            if (waitthread) {
```



```
            if (waitthread) {
                System.out.println(name + ": waiting...");
                synchronized(lock) {
                    try {
                        lock.wait();
                        // A thread that is in the wait() state will not be able to continue processing
                        // until it is notified - something we can see within the printed statements.
                        // After we see the final notify statement of Thread 0 we can see Thread 1 continue.
                    }
                    catch (InterruptedException e){
                        // The 'try catch' is nested inside synchronized to guard the mutex region
                        // in case multiple threads try accessing concurrently.
                        // One thread is given access to this region until it's finished
                        // and then the OS grants access to the following thread that is
                        // waiting in the lock above.
                    }
                }
            }
            else {
                System.out.println(name + ": notifying...");
                synchronized(lock) {
                    lock.notify(); // sends the msg that it's finished
                }
            }

            System.out.println(name + ": after wait/notify"); // finished sending msg of notification
        }
    }
```



```
    public static void main(String[] args) {
        lock = new Object();
        // these declarations missed a second parameter, b, respectively
        t1 = new DemoThread( a: 3000, b: 1000, w: true);
        t2 = new DemoThread( a: 6000, b: 8000, w: false);
        t1.start();
        t2.start();
    }
}

// DOCUMENTING THOUGHT PROCESS
// The initial build error: java: 'try' without 'catch', 'finally' or resource declarations
//  ---> My move: Added a missing catch for synchronized (lock)

// Second build error: Th4.DemoThread cannot be applied to given types;
//  required: long,long,boolean - found: int,boolean
//  reason: actual and formal argument lists differ in length
// ---> My move: Since Th4 was expected to pass the parameters a, b and w I added a value after
//  the first value in both declarations (I passed a value for long b)

// First Successful Build, first Run
//  Thread-0: running ...
//  Thread-1: running ...
//  Thread-0: after t1 sleep, t1 = 3000
//  Thread-0: notifying...
//  Thread-0: after wait/notify
//  Thread-1: after t1 sleep, t1 = 6000
//  Thread-1: notifying...
//  Thread-1: after wait/notify
```

Explanation

This project explores concurrency within the Java language using the methods sleep(), wait(), lock(), and notify(). This program introduces two threads and creates a system of monitoring that controls the availability of the resource both are trying to access. In this case both threads t1 and t2 are called simultaneously within the main and are both attempting to finish the public void run() and access writing. Immediately t1 is stopped and forced to sleep. This means that the process will be delayed by the amount of seconds passed within the ‘a’ argument. Based on the code that was given, the 3000 will translate to 3 seconds. After that t1 runs into the wait and is locked into that state until the other thread finishes and sends the notify.