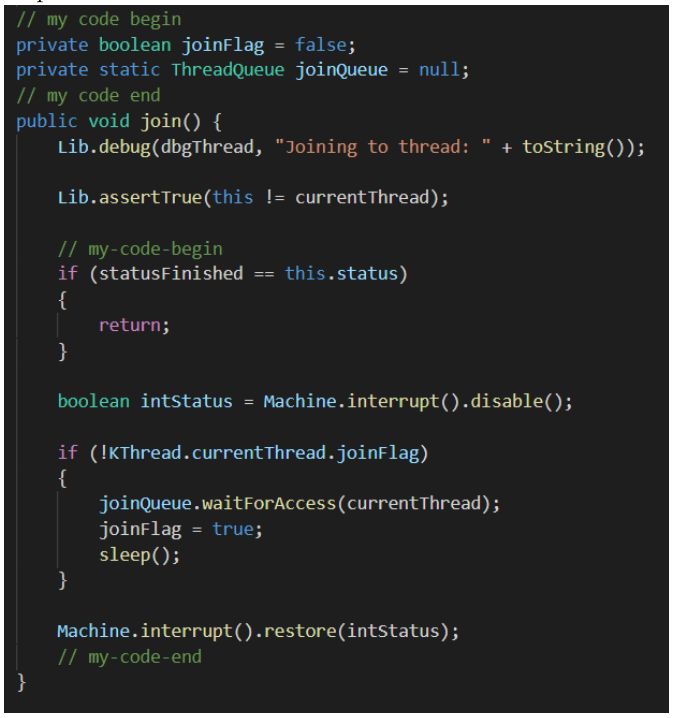
Operating System: Project 1

Note that all code written by me is marked with ”my-code-begin” and ”my-code-end”. Test code is marked with ”my test code begin” and ”my test code end”. I’ll try to give necessary explanation and I’m not showing every line of my code here.

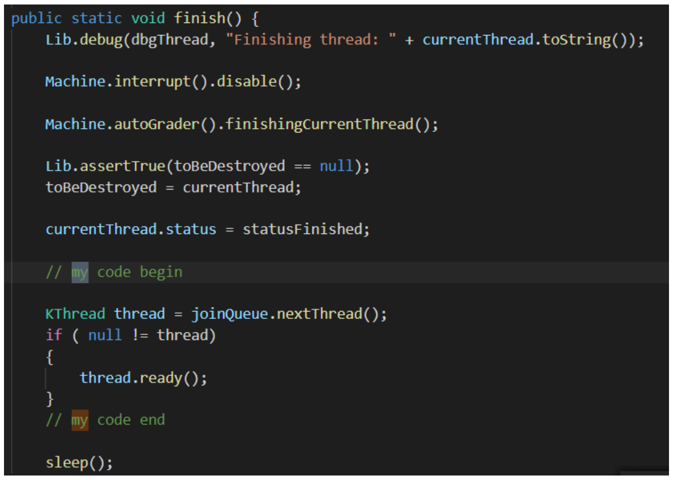
Task 1: Join

Implementation:



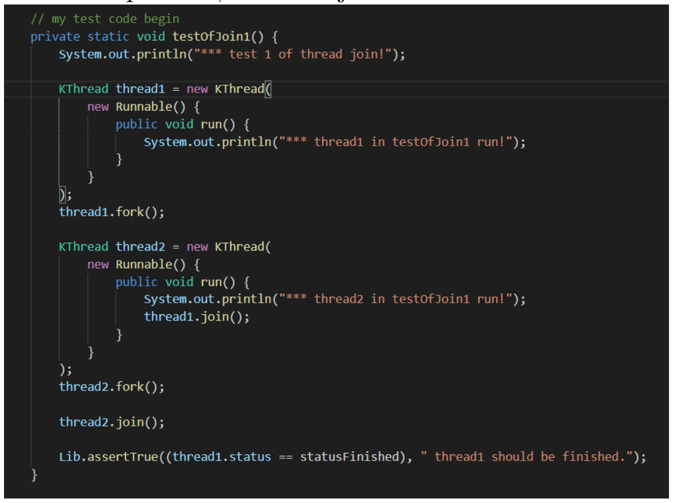
```java
// my code begin
private boolean joinFlag = false;
private static ThreadQueue joinQueue = null;
// my code end
public void join() {
    Lib.debug(dbgThread, "Joining to thread: " + toString());

    Lib.assertTrue(this != currentThread);

    // my-code-begin
    if (statusFinished == this.status)
    {
        return;
    }

    boolean intStatus = Machine.interrupt().disable();

    if (!KThread.currentThread.joinFlag)
    {
        joinQueue.waitForAccess(currentThread);
        joinFlag = true;
        sleep();
    }

    Machine.interrupt().restore(intStatus);
    // my-code-end
}
```

joinQueue is a queue to hold the thread which called join, waiting to wake this thread again.
joinFlag is to check whether a thread is ever joined by another, which is referred to when finishing.



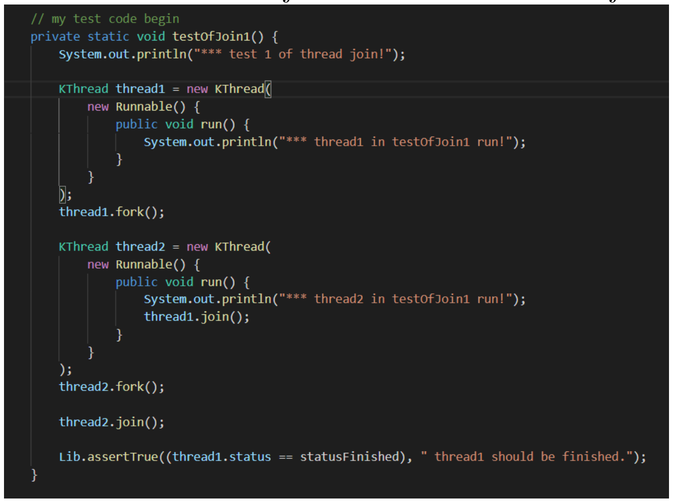
```
public static void finish() {
    Lib.debug(dbgThread, "Finishing thread: " + currentThread.toString());

    Machine.interrupt().disable();

    Machine.autoGrader().finishingCurrentThread();

    Lib.assertTrue(toBeDestroyed == null);
    toBeDestroyed = currentThread;

    currentThread.status = statusFinished;

    // my code begin

    KThread thread = joinQueue.nextThread();
    if ( null != thread)
    {
        thread.ready();
    }
    // my code end

    sleep();
```

Test1: simple case, a thread join another



```
    // my test code begin
    private static void testOfJoin1() {
        System.out.println("*** test 1 of thread join!");

        KThread thread1 = new KThread(
            new Runnable() {
                public void run() {
                    System.out.println("*** thread1 in testOfJoin1 run!");
                }
            }
        );
        thread1.fork();

        KThread thread2 = new KThread(
            new Runnable() {
                public void run() {
                    System.out.println("*** thread2 in testOfJoin1 run!");
                    thread1.join();
                }
            }
        );
        thread2.fork();

        thread2.join();

        Lib.assertTrue((thread1.status == statusFinished), " thread1 should be finished.");
    }
```

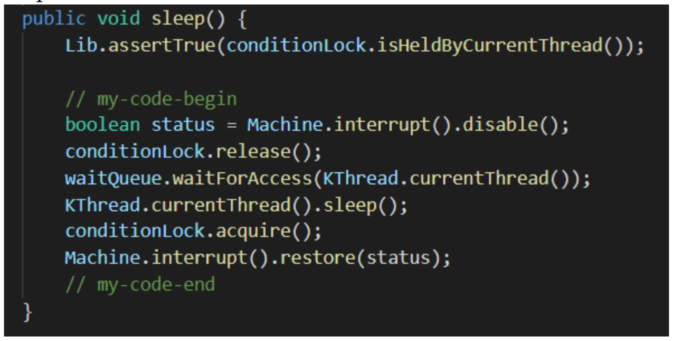
Test2: a thread tries to join a thread which tries to join another



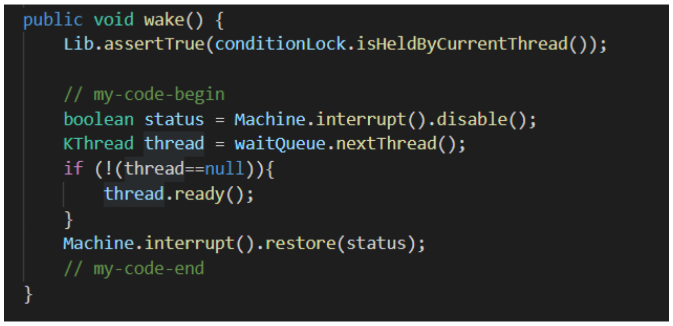
```
    // my test code begin
    private static void testOfJoin1() {
        System.out.println("*** test 1 of thread join!");

        KThread thread1 = new KThread(
            new Runnable() {
                public void run() {
                    System.out.println("*** thread1 in testOfJoin1 run!");
                }
            }
        );
        thread1.fork();

        KThread thread2 = new KThread(
            new Runnable() {
                public void run() {
                    System.out.println("*** thread2 in testOfJoin1 run!");
                    thread1.join();
                }
            }
        );
        thread2.fork();

        thread2.join();

        Lib.assertTrue((thread1.status == statusFinished), " thread1 should be finished.");
    }
```

Task 2: Condition Variables 2

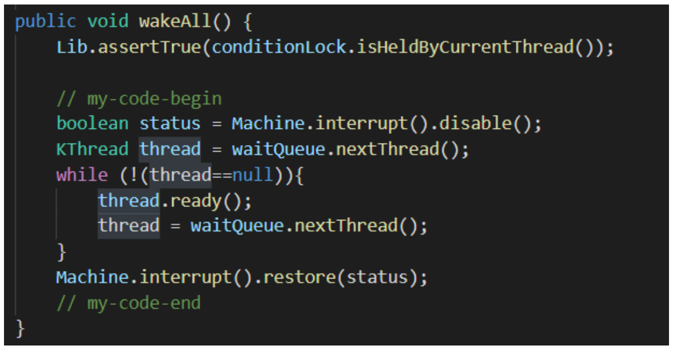
The Implementation generally follows the slides.
Implementation:



```java
public void sleep() {
    Lib.assertTrue(conditionLock.isHeldByCurrentThread());

    // my-code-begin
    boolean status = Machine.interrupt().disable();
    conditionLock.release();
    waitQueue.waitForAccess(KThread.currentThread());
    KThread.currentThread().sleep();
    conditionLock.acquire();
    Machine.interrupt().restore(status);
    // my-code-end
}
```



```java
public void wake() {
    Lib.assertTrue(conditionLock.isHeldByCurrentThread());

    // my-code-begin
    boolean status = Machine.interrupt().disable();
    KThread thread = waitQueue.nextThread();
    if (!(thread==null)){
        thread.ready();
    }
    Machine.interrupt().restore(status);
    // my-code-end
}
```



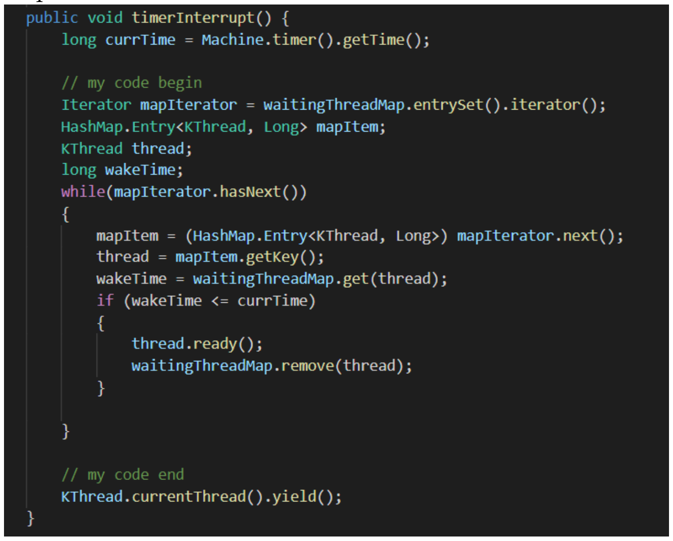
```java
public void wakeAll() {
    Lib.assertTrue(conditionLock.isHeldByCurrentThread());

    // my-code-begin
    boolean status = Machine.interrupt().disable();
    KThread thread = waitQueue.nextThread();
    while (!(thread==null)){
        thread.ready();
        thread = waitQueue.nextThread();
    }
    Machine.interrupt().restore(status);
    // my-code-end
}
```

Test1(code is too long to fit in here): test sleep and wake function
Two threads take turns trying to print sentences. Printing operation is put in critical section. Each thread wakes another and sleep after one printing operation.

Test2: test sleep and wakeall function
Three threads take turns trying to print sentences. Printing operation is put in critical section. Each thread wakes all sleeping threads and sleep after one printing operation.

Task 3: Alarm

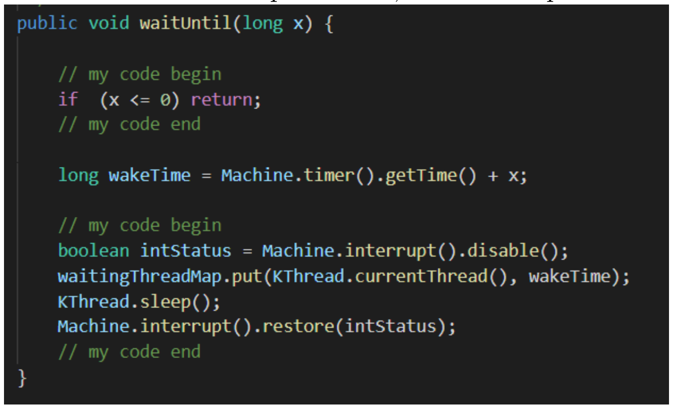
Implementation:



```
public void timerInterrupt() {
    long currTime = Machine.timer().getTime();

    // my code begin
    Iterator mapIterator = waitingThreadMap.entrySet().iterator();
    HashMap.Entry<KThread, Long> mapItem;
    KThread thread;
    long wakeTime;
    while(mapIterator.hasNext())
    {
        mapItem = (HashMap.Entry<KThread, Long>) mapIterator.next();
        thread = mapItem.getKey();
        wakeTime = waitingThreadMap.get(thread);
        if (wakeTime <= currTime)
        {
            thread.ready();
            waitingThreadMap.remove(thread);
        }

    }

    // my code end
    KThread.currentThread().yield();
}
```

a hashmap is used to keep those threads that are to wake up automatically at some point, along with their corresponding wake up time.

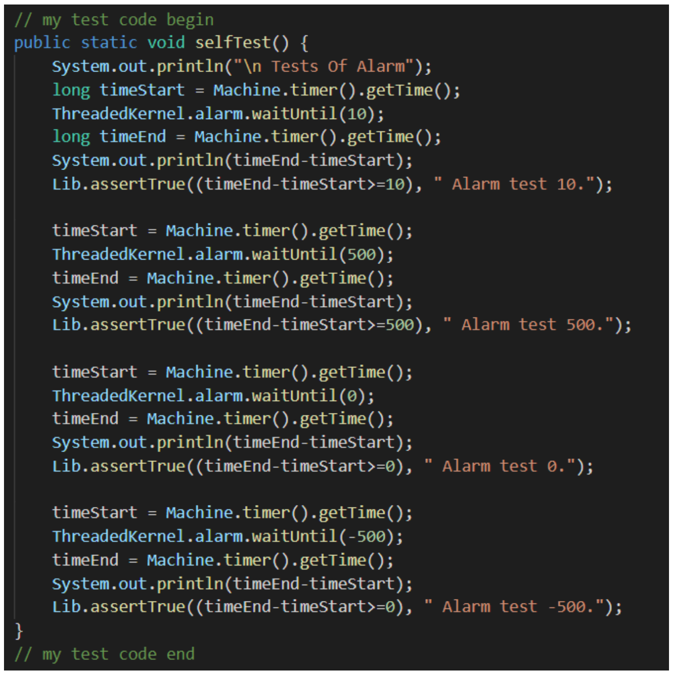
Each time timerInterrupt is called, this hashmap is checked throughly to check if any thread finishes sleeping.



```
public void waitUntil(long x) {

    // my code begin
    if  (x <= 0) return;
    // my code end

    long wakeTime = Machine.timer().getTime() + x;

    // my code begin
    boolean intStatus = Machine.interrupt().disable();
    waitingThreadMap.put(KThread.currentThread(), wakeTime);
    KThread.sleep();
    Machine.interrupt().restore(intStatus);
    // my code end
}
```

Test:

Task 3: Alarm continued on next page...

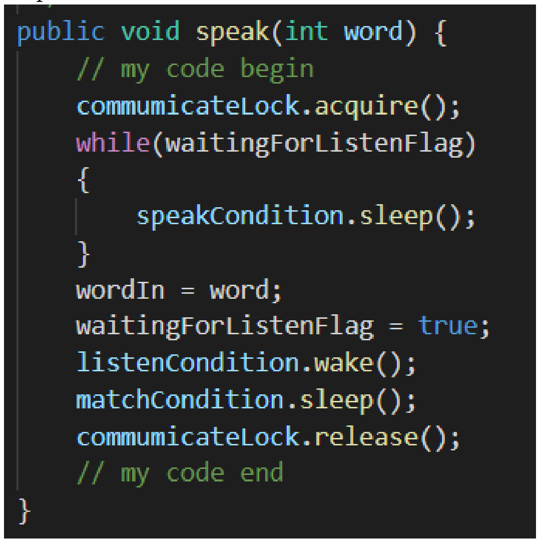


```java
// my test code begin
public static void selfTest() {
    System.out.println("\n Tests Of Alarm");
    long timeStart = Machine.timer().getTime();
    ThreadedKernel.alarm.waitUntil(10);
    long timeEnd = Machine.timer().getTime();
    System.out.println(timeEnd-timeStart);
    Lib.assertTrue((timeEnd-timeStart>=10), " Alarm test 10.");

    timeStart = Machine.timer().getTime();
    ThreadedKernel.alarm.waitUntil(500);
    timeEnd = Machine.timer().getTime();
    System.out.println(timeEnd-timeStart);
    Lib.assertTrue((timeEnd-timeStart>=500), " Alarm test 500.");

    timeStart = Machine.timer().getTime();
    ThreadedKernel.alarm.waitUntil(0);
    timeEnd = Machine.timer().getTime();
    System.out.println(timeEnd-timeStart);
    Lib.assertTrue((timeEnd-timeStart>=0), " Alarm test 0.");

    timeStart = Machine.timer().getTime();
    ThreadedKernel.alarm.waitUntil(-500);
    timeEnd = Machine.timer().getTime();
    System.out.println(timeEnd-timeStart);
    Lib.assertTrue((timeEnd-timeStart>=0), " Alarm test -500.");
}
// my test code end
```

Two are simple tests and other two tests unusual cases where wait time is zero or negative.

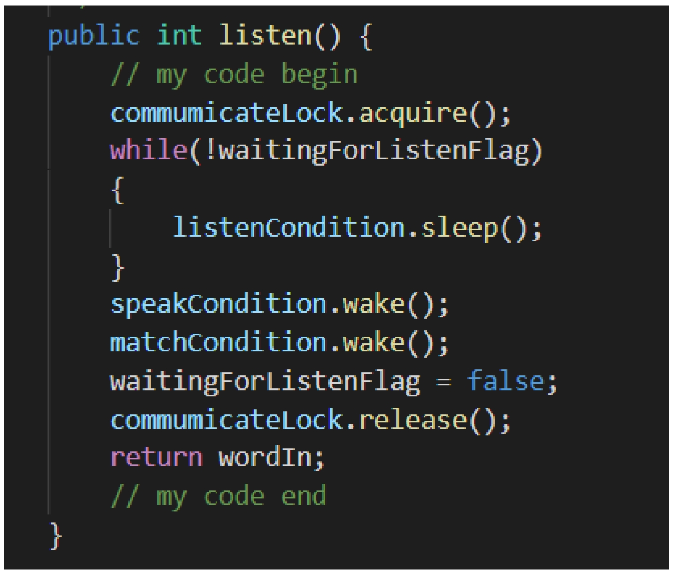
Task 4

(code is too long to fit in here)
Implementation:



```java
public void speak(int word) {
    // my code begin
    commumicateLock.acquire();
    while(waitingForListenFlag)
    {
        speakCondition.sleep();
    }
    wordIn = word;
    waitingForListenFlag = true;
    listenCondition.wake();
    matchCondition.sleep();
    commumicateLock.release();
    // my code end
}
```

Task 4 continued on next page. . .



```java
    public int listen() {
        // my code begin
        commumicateLock.acquire();
        while(!waitingForListenFlag)
        {
            listenCondition.sleep();
        }
        speakCondition.wake();
        matchCondition.wake();
        waitingForListenFlag = false;
        commumicateLock.release();
        return wordIn;
        // my code end
    }
```

int wordIn is used to transfer words. It's either empty(invalid), or storing the message of the first pending speaker.
matchCondition holds speakers waiting to be listened
listenCondition holds listeners waiting to be spoken to
matchCondition holds at most one speaker waiting to return

speaker's action:
acqiure lock
while wordIn is valid, sleep
put its message in wordIn
sleep, waiting for a listener to ask it to return
release lock

listener's action:
acqiure lock
while wordIn is invalid, sleep
read the message in wordIn
call a next speaker and ask the current speaker in matchCondition to return release lock

Test1: one speaker and one listener
Test2: two speakers and then two listeners
Test3: two listeners, four speakers and then two listeners