**LETTERKENNY INSTITUTE OF TECHNOLOGY  
ASSIGNMENT COVER SHEET**

Student’s name: Dietmar Steiner (L00131070)

Word Count:

I confirm that the work submitted has been produced solely through my own efforts.

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_17/11/2015\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lecturer’s name: Nigel Kelvey

Assessment Title: Performance Based Programming Assignment

Work to be submitted to: Nigel Kelvey

Date for submission of work:

Place and Time for submitting work:

Notes

Penalties: The total marks available for an assignment is reduced by 15% for work submitted up to one week late. The total marks available are reduced by 30% for work up to two weeks late. Assignment work received more than two weeks late will receive a mark of zero. (Incidents of alleged plagiarism and cheating are dealt with in accordance with the Institute’s Assessment Regulations).

Plagiarism: Presenting the ideas etc. of someone else without proper acknowledgement (see section L1 paragraph 5).

Cheating: The use of unauthorised material in a test, exam etc., unauthorised access to test matter, unauthorised collusion, dishonest behaviour in respect of assessments, and deliberate plagiarism (see section L1 paragraph8).

Continuous Assessment: For students repeating an examination, marks awarded for continuous assessment, shall normally be carried forward from the original examination to the repeat examination.

Workflow Job Queue

Contents

[Introduction 2](#_Toc435620385)

[Requirements 2](#_Toc435620386)

[Report 3](#_Toc435620387)

[Semaphores 3](#_Toc435620388)

[Threads 3](#_Toc435620389)

[Synchronisation 3](#_Toc435620390)

[Exceptions 3](#_Toc435620391)

[Documentation 3](#_Toc435620392)

[Conclusion 3](#_Toc435620393)

[Testing 3](#_Toc435620394)

[Unit Test JobQueueTest 3](#_Toc435620395)

[TestJob 6](#_Toc435620396)

[TestErrorJob 6](#_Toc435620397)

[Conclusion 7](#_Toc435620398)

[Appendix 7](#_Toc435620399)

[Code 7](#_Toc435620400)

[Class JobQueue 7](#_Toc435620401)

[Class QueueThread 11](#_Toc435620402)

[Interface Job 12](#_Toc435620403)

[Test Log File 14](#_Toc435620404)

# Introduction

In Workflow Systems (Workflow Management Coalition, 1995) a workflow contains a number of jobs to be carried out. To facilitate distribution and resource management Job Queues are used. Job Queues contains jobs from many workflows and are a means to use computer resources efficiently by being job unaware. The job knows what to do. Workflow jobs are sent to a queue to be executed. The queue then takes a job out of the queue when resources are available and executes the job.

Requirements   
Only a certain amount of threads can be started by the queue to not overload the system.  
Only a certain amount of jobs waiting is allowed. If the number of jobs waiting is at the maximum amount the queue does not accept further jobs to be added to the queue.  
When the queue is shutdown all jobs within the queue have to be finished before the shutdown.

# Report

The assignment is to create a software artefact with three items from the list given selected.

This Assignment selected Semaphores, Threads and Synchronisation.

## Semaphores

The artefact uses 2 Semaphores for:

1. Limiting the number of jobs, which can be submitted to the queue (method add(job)).
2. Holding Threads in a waiting condition until a job becomes available (method getJob()).

## Threads

The artefact uses 10 threads to execute the jobs in the queue.

## Synchronisation

The artefact uses a synchronised block in the setRunning method to ensure the state of running is manipulated only by one thread at the time and after the state of the instance is valid.

## Exceptions

Exceptions are handled by a special function. If a different handling is required the class can be sub classed and the handler function overwritten.

## Documentation

The documentation of the code is contained in the code.

# Conclusion

The challenge in this assignment is the testing as in concurrent applications, there are not many moments where the state of the object under test can be safely described. In my testing code I tried to establish a stable environment by waiting for a second.

# Testing

## Unit Test JobQueueTest

package jobqueue;

import java.util.ArrayList;

import java.util.concurrent.Semaphore;

import java.util.logging.Level;

import java.util.logging.Logger;

import junit.framework.TestCase;

/\*\*

\*

\* @author L00131070

\*/

public class JobQueueTest extends TestCase {

@Override

protected void setUp() throws Exception {

super.setUp();

}

@Override

protected void tearDown() throws Exception {

super.tearDown();

}

/\*\*

\* Test of constructor method, of class JobQueue.

\*

\* @throws InterruptedException

\*

\*/

public void testJobQueue() throws InterruptedException {

System.out.println("testJobQueue");

JobQueue queue = new JobQueue();

assertTrue(queue != null);

assertTrue(queue.isRunning());

Thread.sleep(500);

System.out.println("Queue shutdown");

queue.setRunning(false);

assertTrue(!queue.isRunning());

Thread.sleep(500);

System.out.println("Queue startup");

queue.setRunning(true);

assertTrue(queue.isRunning());

System.out.println("Test testConstructor successful");

}

/\*\*

\* Test of add method, of class JobQueue.

\*

\* @throws InterruptedException

\*

\*/

public void testAdd() throws InterruptedException {

System.out.println("testAdd");

JobQueue queue = new JobQueue();

if (queue.add(null)) {

fail(" NULL JOB");

}

for (int i = 0; i < 40; i++) {

Job job = new TestJob();

boolean test = queue.add(job);

if (!test) {

assertTrue(!test && i == 20);

System.out.println(i + " " + test + " waiting 1s");

Thread.sleep(1000);

test = queue.add(job);

}

}

queue.setRunning(false);

System.out.println("Test testAdd successful");

}

/\*\*

\* Test of setRunning method, of class JobQueue.

\*

\* @throws InterruptedException

\*

\*/

public void testSetRunning() throws InterruptedException {

System.out.println("testSetRunning");

JobQueue queue = new JobQueue();

assertTrue(queue.isRunning());

Thread.sleep(1000);

queue.setRunning(false);

assertTrue(!queue.isRunning());

assertTrue(queue.getWaitingThreads().getQueueLength() == 0);

assertTrue(queue.getFullQueue().availablePermits() == 0);

assertTrue(queue.getThreads().isEmpty());

assertTrue(queue.getQueue().isEmpty());

queue.setRunning(true);

assertTrue(queue.isRunning());

Thread.sleep(1000);

assertTrue(queue.getWaitingThreads().getQueueLength() == 10);

assertTrue(queue.getFullQueue().availablePermits() == 20);

assertTrue(queue.getThreads().size() == 10);

assertTrue(queue.getQueue().isEmpty());

System.out.println("Test testSetRunning successful");

}

/\*\*

\* Test of handling exceptions thrown from doJob of the job interface in

\* class JobQueue Thread.

\*/

public void testErrorJob() {

System.out.println("testErrorJob");

JobQueue queue = new JobQueue();

queue.add(new TestErrorJob());

queue.setRunning(false);

System.out.println("Test testErrorJob successful");

}

/\*\*

\* Test of handleThrowables method, of class JobQueue.

\*/

public void testHandleThrowables() {

System.out.println("Test testHandleThrowables");

Throwable ex = new Exception("TestException");

JobQueue instance = new JobQueue();

instance.handleThrowables(ex);

System.err.flush();

System.out.println("Test testHandleThrowables successful");

}

}

## TestJob

**package** jobqueue;

**class** TestJob **implements** Job {

**private** **static** **int** *nextId* = 1;

**private** **int** id;

TestJob() {

id = *nextId*++;

}

@Override

**public** **void** doJob() {

**try** {

System.***out***.println("start job: " + id);

Thread.*sleep*(1000);

System.***out***.println("end job: "+id);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

@Override

**public** **void** setError(Throwable error) {

System.***out***.println("ERROR: "+error.getMessage());

}

}

## TestErrorJob

**package** jobqueue;

**class** TestErrorJob **implements** Job {

**private** **static** **int** *nextId* = 1;

**private** **int** id;

TestErrorJob() {

id = *nextId*++;

}

@Override

**public** **void** doJob() {

System.***out***.println("start errorjob: " + id);

String test = **null**;

test.length();

System.***out***.println("end errorjob: " + id);

}

@Override

**public** **void** setError(Throwable error) {

System.***out***.println("ERROR: "+error.getMessage());

}

}

# Conclusion

Concurrently operating threads cannot be predicted as to when something is executed, as one can see in the test log file. Testing can be quite challenging as it is difficult to predict what to test when and make a correct decision if the test was successful.

For exception handling the artefact implements an exception handler method with a default behaviour. Subclasses can override this function to implement their own behaviour.

# Appendix

## Code

### Class JobQueue

Class JobQueue

/\*\*

\*

\*

\*/

package jobqueue;

import java.util.ArrayList;

import java.util.concurrent.ConcurrentLinkedQueue;

import java.util.concurrent.Semaphore;

import java.util.concurrent.TimeUnit;

/\*\*

\* This class is a service to run given jobs concurrently.<br>

\* The queue starts 10 threads to service the jobs in the queue.<br>

\* A maximum number of 20 jobs can be in the service either in the queue or being executed.<br>

\* If the service is full the add procedure waits for 500ms for a space to become available then returns.<br>

\*

\* @author L00131070

\*

\*/

public class JobQueue {

/\*\*

\* Defines the maximum number of jobs allowed in the queue environment. This

\* includes running jobs.

\*/

public static final int MAXJOBS = 20;

/\*\*

\* Defines the number of concurrent threads will be created for job

\* execution.

\*/

public static final int MAXTHREADS = 10;

/\*\*

\* Defines the ms wait time for add(job).

\*/

public static final int ADDWAIT = 500;

private final Semaphore waitingThreads;

private final Semaphore fullQueue;

private volatile boolean running = false;

private final ArrayList<Thread> threads = new ArrayList<>(MAXTHREADS);

private final ConcurrentLinkedQueue<Job> queue = new ConcurrentLinkedQueue<>();

/\*\*

\* Returns the getJob Semaphore

\*

\* @return the waitingThreads Semaphore

\*/

protected Semaphore getWaitingThreads() {

return waitingThreads;

}

/\*\*

\* Returns the add Semaphore

\*

\* @return the fullQueue Semaphore

\*/

protected Semaphore getFullQueue() {

return fullQueue;

}

/\*\*

\* Returns the Running state of the Queue

\*

\* @return the running state of the queue

\*/

protected final boolean isRunning() {

return running;

}

/\*\*

\* Returns the Thread ArrayList

\* @return the threads array

\*/

protected ArrayList<Thread> getThreads() {

return threads;

}

/\*\*

\* Returns the LinkedQueue

\* @return the queue

\*/

protected ConcurrentLinkedQueue<Job> getQueue() {

return queue;

}

/\*\*

\* Creates a JobQueue

\*/

public JobQueue() {

waitingThreads = new Semaphore(0);

fullQueue = new Semaphore(MAXJOBS, true);

setRunning(true);

}

/\*\*

\* Waits for a job to be available in the queue and returns the job or if it

\* returns null to end the thread

\*

\* @return the job to be executed or null to initiate ending of the thread

\*/

protected Job getJob() {

try {

waitingThreads.acquire();

} catch (InterruptedException e) {

handleThrowables(e);

}

return queue.poll();

}

/\*\*

\* Adds the given job to the queue, if there is space

\*

\* @param job the job to add

\* @return true when successfully added or false when there is no space left

\* in the queue

\*/

public boolean add(Job job) {

if (null == job) {

return false;

}

try {

if (!fullQueue.tryAcquire(1, ADDWAIT, TimeUnit.MILLISECONDS)) {

return false;

}

} catch (InterruptedException e) {

handleThrowables(e);

return false;

}

queue.add(job);

waitingThreads.release();

return true;

}

/\*\*

\* starts up the queue operation or shuts it down dependent on running

\*

\* @param running if true starts the threads or if false ends all threads.

\*/

protected final void setRunning(boolean running) {

if (isRunning() == running) {

return;

}

synchronized (this) {

if (!running) {

this.running = false;

waitingThreads.release(MAXTHREADS);

threads.stream().forEach((thread) -> {

try {

thread.join();

} catch (InterruptedException e) {

handleThrowables(e);

}

});

waitingThreads.drainPermits();

fullQueue.drainPermits();

queue.clear();

threads.clear();

} else {

waitingThreads.drainPermits();

fullQueue.drainPermits();

queue.clear();

this.running = true;

for (int i = 0; i < threads.size(); i++) {

try {

threads.remove(i).join();

} catch (InterruptedException e) {

handleThrowables(e);

}

}

for (int i = 0; i < MAXTHREADS; i++) {

threads.add(i, new QueueThread(this));

threads.get(i).start();

}

fullQueue.release(MAXJOBS);

}

}

}

@Override

protected void finalize() {

if (isRunning()) {

setRunning(false);

}

try {

super.finalize();

} catch (Throwable ex) {

handleThrowables(ex);

}

}

/\*\*

\* All Throwables are handled using this function.

\* To have a different handling override this function.

\*

\* @param ex the Throwable thrown during the execution.

\*

\*/

protected void handleThrowables(Throwable ex) {

System.out.println(ex.getMessage());

ex.printStackTrace();

}

}

### Class QueueThread

/\*\*

\* This class is the thread class used by the JobQueue.<br>

\*

\* @author L00131070

\*

\*/

final class QueueThread extends Thread {

JobQueue queue = null;

/\*\*

\* Creates a QueueThread

\*/

QueueThread(JobQueue queue) {

this.queue = queue;

}

/\*\*

\* runs the thread by getting a job from the queue then executing the doJob on the job.

\* The Thread ends when the job gotten from the queue is null.

\*/

@Override

public void run() {

System.out.println("Thread: " + queue.getThreads().indexOf(this) + " starting");

Job job = queue.getJob();

while (null != job) {

try {

job.doJob();

} catch (Throwable error) {

job.setError(error);

}

queue.getFullQueue().release();

job = queue.getJob();

}

System.out.println("Thread: " + queue.getThreads().indexOf(this) + " ending");

}

}

### Interface Job

**package** jobqueue;

/\*\*

\* The Job Interface defines the functions to be implemented to use the JobQueue.

\*

\* **@author** L00131070

\*/

**public** **interface** Job {

/\*\*

\* The job executed by the JobQueue

\*/

**public** **void** doJob();

/\*\*

\* All errors and exceptions thrown by doJob() during the execution are sent here for processing.

\*

\* **@param** error the error/exception thrown by doJob()

\*/

**public** **void** setError(Throwable error);

}

## Test Log File

testAdd

Thread: 0 starting

Thread: 2 starting

Thread: 4 starting

Thread: 6 starting

Thread: 8 starting

Thread: 1 starting

Thread: 3 starting

Thread: 5 starting

Thread: 7 starting

Thread: 9 starting

start job: 1

start job: 2

start job: 3

start job: 4

start job: 5

start job: 6

start job: 7

start job: 8

start job: 9

start job: 10

20 false waiting 1s

end job: 6

start job: 11

end job: 7

start job: 12

end job: 8

start job: 13

end job: 9

start job: 14

end job: 10

start job: 15

end job: 1

start job: 16

end job: 2

start job: 17

end job: 3

end job: 5

end job: 4

start job: 19

start job: 18

start job: 20

end job: 18

start job: 21

end job: 16

start job: 22

end job: 17

start job: 23

end job: 19

start job: 24

end job: 20

start job: 25

end job: 11

start job: 26

end job: 12

start job: 27

end job: 13

start job: 28

end job: 14

start job: 29

end job: 15

start job: 30

end job: 26

start job: 31

end job: 27

start job: 32

end job: 28

start job: 33

end job: 29

start job: 34

end job: 30

start job: 35

end job: 22

start job: 36

end job: 21

end job: 25

end job: 24

end job: 23

start job: 39

start job: 38

start job: 37

start job: 40

end job: 31

Thread: 1 ending

end job: 32

Thread: 3 ending

end job: 33

Thread: 5 ending

end job: 34

Thread: 7 ending

end job: 35

Thread: 9 ending

end job: 39

Thread: 8 ending

end job: 37

Thread: 4 ending

end job: 36

Thread: 0 ending

end job: 38

Thread: 6 ending

end job: 40

Thread: 2 ending

Test testAdd successful

testJobQueue

Thread: 1 starting

Thread: 3 starting

Thread: 5 starting

Thread: 7 starting

Thread: 9 starting

Thread: 0 starting

Thread: 2 starting

Thread: 4 starting

Thread: 6 starting

Thread: 8 starting

Queue shutdown

Thread: 1 ending

Thread: 3 ending

Thread: 5 ending

Thread: 7 ending

Thread: 9 ending

Thread: 0 ending

Thread: 2 ending

Thread: 4 ending

Thread: 6 ending

Thread: 8 ending

Queue startup

Test testConstructor successful

testSetRunning

Thread: 1 starting

Thread: 3 starting

Thread: 0 starting

Thread: 5 starting

Thread: 2 starting

Thread: 7 starting

Thread: 4 starting

Thread: 9 starting

Thread: 6 starting

Thread: 1 starting

Thread: 8 starting

Thread: 3 starting

Thread: 5 starting

Thread: 0 starting

Thread: 7 starting

Thread: 2 starting

Thread: 9 starting

Thread: 4 starting

Thread: 6 starting

Thread: 8 starting

Thread: 1 ending

Thread: 3 ending

Thread: 5 ending

Thread: 0 ending

Thread: 2 ending

Thread: 7 ending

Thread: 8 ending

Thread: 6 ending

Thread: 4 ending

Thread: 9 ending

Thread: 0 starting

Thread: 1 starting

Thread: 2 starting

Thread: 4 starting

Thread: 6 starting

Thread: 3 starting

Thread: 5 starting

Thread: 8 starting

Thread: 9 starting

Thread: 7 starting

Test testSetRunning successful

java.lang.Exception: TestException

Test testHandleThrowables

TestException

Test testHandleThrowables successful

testErrorJob

Thread: 1 starting

Thread: 3 starting

Thread: 5 starting

Thread: 7 starting

Thread: 9 starting

Thread: 1 starting

Thread: 3 starting

Thread: 5 starting

Thread: 7 starting

Thread: 9 starting

Thread: 0 starting

Thread: 2 starting

Thread: 4 starting

Thread: 6 starting

Thread: 8 starting

at jobqueue.JobQueueTest.testHandleThrowables(JobQueueTest.java:118)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)

at java.lang.reflect.Method.invoke(Unknown Source)

at junit.framework.TestCase.runTest(TestCase.java:176)

at junit.framework.TestCase.runBare(TestCase.java:141)

at junit.framework.TestResult$1.protect(TestResult.java:122)

at junit.framework.TestResult.runProtected(TestResult.java:142)

at junit.framework.TestResult.run(TestResult.java:125)

at junit.framework.TestCase.run(TestCase.java:129)

at junit.framework.TestSuite.runTest(TestSuite.java:252)

at junit.framework.TestSuite.run(TestSuite.java:247)

at org.eclipse.jdt.internal.junit.runner.junit3.JUnit3TestReference.run(JUnit3TestReference.java:131)

at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)

at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)

at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:675)

at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)

at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)

Thread: 0 starting

Thread: 2 starting

Thread: 4 starting

Thread: 6 starting

Thread: 8 starting

start errorjob: 1

ERROR: null

Thread: 1 ending

Thread: 3 ending

Thread: 5 ending

Thread: 7 ending

Thread: 9 ending

Thread: 0 ending

Thread: 2 ending

Thread: 4 ending

Thread: 6 ending

Thread: 8 ending

Test testErrorJob successful