Module 19 Concurrent Development

Module title	Concurrent Development
Module NFQ level (only if an NFQ level can be demonstrated)	7
Module number/reference	BSCH-CD
Parent programme(s)	Bachelor of Science (Honours) in Computing Science
Stage of parent programme	Award stage
Semester (semester1/semester2 if applicable)	Semester 1
Module credit units (FET/HET/ECTS)	ECTS
Module credit number of units	10
List the teaching and learning modes	Direct, Blended
Entry requirements (statement of knowledge, skill and competence)	Learners must have achieved programme entry requirements.
Pre-requisite module titles	BSCH-CP, BSCH-OOP, BSCH-SD1, BSCH-SD2
Co-requisite module titles	None
Is this a capstone module? (Yes or No)	No
Specification of the qualifications (academic, pedagogical and professional/occupational) and experience required of staff (staff includes workplace personnel who are responsible for learners such as apprentices, trainees and learners in clinical placements)	Qualified to as least a Bachelor of Science (Honours) level in Computer Science or equivalent and with a Certificate in Training and Education (30 ECTS at level 9 on the NFQ) or equivalent.
Maximum number of learners per centre (or instance of the module)	60
Duration of the module	One Academic Semester, 12 weeks teaching
Average (over the duration of the module) of the contact hours per week	5
Module-specific physical resources and support required per centre (or instance of the module)	One class room with capacity for 60 learners along with one computer lab with capacity for 25 learners for each group of 25 learners

Analysis of required learning effort		
	Minimum ratio teacher / learner	**Hours**
Effort while in contact with staff		
Classroom and demonstrations	1:60	24
Monitoring and small-group teaching	1:25	36
Other (specify)		
Independent Learning		
Directed e-learning		
Independent Learning		90
Other hours (worksheets and assignments)		100
Work-based learning – learning effort		
Total Effort		250

Allocation of marks (within the module)					
	Continuous assessment	**Supervised project**	**Proctored practical examination**	**Proctored written examination**	**Total**
Percentage contribution	60%			40%	100%

Module aims and objectives

This module builds on the work completed in programming modules completed in stages 1 and 2 and will apply the methods learned there to the study of Concurrent Development. Concurrency complicates the field of programming because processes are non-deterministic and to write concurrent systems that are correct we must understand how to manage limited shared resources. Learners gain an understanding of the need for, and advantages of, concurrent and parallel systems; a mastery of a new programming paradigm that is different from that of the single threaded one; a description of how processes and threads are managed in multi processor, multi core machines; an understanding and mastery of the many classical problems arising with concurrent and parallel tasks; an awareness of the need for such issues as fairness, process synchronisation, deadlock avoidance, etc; and an ability to write concurrent and parallel programs to solve real world problems.

Minimum intended module learning outcomes

On successful completion of this module, the learner will be able to:

1. Discuss the advantages of concurrent programming over single threaded sequential programming
2. Use threading to distribute the workload of a single task and optimise performance of algorithms
3. Explain the necessity for synchronisation when sharing resources and solve problems that require synchronisation
4. Explain race conditions, deadlocks and strategies for deadlock avoidance

5. Use condition variables to solve limited resource sharing problems
6. Describe the role of semaphores and events in concurrent systems and solve problems requiring both semaphores and events as part of the solution
7. Describe the difference between the shared memory model for threads and the distributed memory model for processes
8. Describe the client server architecture and use it to solve communication between distributed processes
9. Explain the semantics of streams and show how they can be harnessed to deliver multi-threaded parallel services

Rationale for inclusion of the module in the programme and its contribution to the overall MIPLOs

This module takes an object-oriented approach to the study of concurrent development and provides learners with the experience of applying this methodology to a complex field of programming. The module provides learners with the tools necessary to build concurrent models that optimise the performance of solutions to computational problems by harnessing the full power of the underlying hardware architecture. It provides a detailed study of the classical issues surrounding the control of non-deterministic processes. Appendix 1 of the programme document maps MIPLOs to the modules through which they are delivered.

Information provided to learners about the module

Learners receive a programme handbook to include module descriptor, module learning outcomes (MIMLO), class plan, assignment briefs, assessment strategy and reading materials.

Module content, organisation and structure

Fundamentals

- Motivation for concurrency; simple examples; determinism versus non-determinism – advantages and disadvantages
- Process versus threads; priority of processes; process creation and destruction (Fork in Unix, Task in Ada, thread in java); processes sharing memory; dynamic process creation; facilities for concurrency provided by programming languages and operating systems; C#, Windows, Linux and Unix; Java virtual machine; writing threads in Java

Divide & Conquer algorithms

- Distributing workload by dividing linear spaces; using threadpools to optimise performance; using divide and conquer algorithms such as ForkJoinPool in Java.

Resource sharing

- Mutual exclusion; intrinsic locking; coarse grained versus fine-grained locking; deadlocks and deadlock avoidance; starvation; condition variables and sharing limited resources; semaphores; Classic problems: dining philosophers; readers/writers, producer/consumer, bounded buffer; resource allocation; synchronisation; controlling threads with thread queues. Solving problems with Semaphores and event barriers.

Communicating processes (processes without shared memory)

- Distributed memory model; Pipes; channels; message passing; remote procedure call. Distributed Topologies.
- Server design and the implementation of client server architecture over sockets. Deploying services on a client server architecture. Supporting distributed processing using server model.

Functional Programming and Concurrent Streams

- Streams and the delivery of parallel services to application programs.
- Message passing systems based on Actors (Scala, F# and Erlang) Avoiding race conditions with the use of immutable state.
- Communication protocols for actors. Programming with Actors

Module teaching and learning (including formative assessment) strategy

The module is delivered through a combination of lectures and practical lab programming sessions. The learners complete a series of worksheets throughout the module that are directly related to the material covered in lectures. The emphasis is on developing sound software engineering skills in practical programming based on theoretical knowledge.

Assessment consists of a series of continuous assignments and a final examination. Each week learners are required to complete a series of programming tasks that relate to the material covered in lectures. The practical lab sessions are used to enforce concepts covered in the lectures and the worksheets are used to ensure that learners are keeping up with the material as it is delivered. Lab sessions are also used to deal with issues emerging from the worksheets. All work submitted by learners is assessed and comments are given to individual learners.

Timetabling, learner effort and credit

The module is timetabled as one 2-hour lecture and two 1.5-hour labs per week.

Continuous assessment spreads the learner effort to focus on the aspects of the course under discussion.

There are 60 contact hours made up of 12 lectures delivered over 12 weeks with classes taking place in a classroom. There are also 24 lab sessions delivered over 12 weeks taking place in a fully equipped computer lab. The learner will need 100 hours of independent effort to further develop the skills and knowledge gained through the contact hours. An additional 90 hours are set aside for learners to work on worksheets and assignments that must be completed for the module.

The team believes that 250 hours of learner effort are required by learners to achieve the MIMLOs and justify the award of 10 ECTS credits at this stage of the programme.

Work-based learning and practice-placement

There is no work based learning or practice placement involved in the module.

E-learning

The college VLE is used to disseminate notes, advice, and online resources to support the learners. The learners are also given access to Lynda.com as a resource for reference.

Module physical resource requirements

Requirements are for a classroom for 60 learners equipped with a projector, and a 25-seater computer lab for practical sessions with access to Java and a suitable development environment (for example Notepad++) (this may change should better techonologies arise).

Reading lists and other information resources

Recommended Text

Benmammar, B. (2017) *Concurrent, Real-Time and Distributed Programming in Java*. Newark: John Wiley & Sons, Incorporated.

Goetz, B. (2006) *Java Concurrency in Practice*. Upper Saddle River: Addison-Wesley.

Secondary Reading

Lea, D. (2002) *Concurrent Programming in Java: Design Principles and Patterns*. Reading: Addison-Wesley.

Burns, A. and Wellings, A. J. (2007) *Concurrent and Real-time Programming in Ada*. Cambridge: Cambridge University Press.

Magee, J. and Kramer, J. (2006) *Concurrency: State Models & Java Programs*. Chichester: John Wiley & Sons

Odersky, M., Spoon, L. and Venners, B. (2016) *Programming in Scala*. Walnut Creek: Artima Press.

Lectures on Programming Paradigms with Scala, Tony Mullins (2012), Griffith College Dublin.

Specifications for module staffing requirements

For each instance of the module, one lecturer qualified to at least Bachelor of Science (Honours) in Computer Science or equivalent, and with a Certificate in Training and Education (30 ECTS at level 9 on the NFQ) or equivalent.. Industry experience would be a benefit but is not a requirement.

Learners also benefit from the support of the programme director, programme administrator, learner representative and the Student Union and Counselling Service.

Module Assessment Strategy

The assignments constitute the overall grade achieved, and are based on each individual learner's work. The continuous assessments provide for ongoing feedback to the learner and relates to the module curriculum.

No.	Description	MIMLOs	Weighting
1	Series of weighted worksheets Worksheet 1: Learning outcomes 2 Worksheet 2: Learning outcomes 2 Worksheet 3: Learning outcomes 2,3,4,5 Worksheet 4: Learning outcomes 3,6 Worksheet 5: Learning outcomes 2,3,4,6,8 Worksheet 6: Learning outcomes 2,3,4,9	1-9	60%
2	Written exam that tests the theoretical aspects of the module	1-9	40%

All repeat work is capped at 40%.

Sample assessment materials

Note: All assignment briefs are subject to change in order to maintain current content.

Assignment 1

Please complete the problems listed below. This assignment forms part of the assessment for this module and you are required to upload your solution in the template file available on Moodle. This file should be used to write your solution and should be uploaded to Moodle on or before midnight on Sunday next. You must include as header your **name** and **student number**.

Question 1

Write a program that uses 4 threads that each toss a die a given number of times. In both cases the result of each toss is stored in a shared array. The array is deemed to be large enough to store the result of each throw and each thread should only write to its own array segment. Once the threads have completed their work then the main program counts the frequency of each throw and prints it on the screen.

Question 2

Given below is a single threaded program that computes the index of the **leftmost zero** in a huge array. Your task is to write a parallel solution that distributes the workload fairly over 4 threads. It is important to try to write an optimal solution. **Note**: it is not required to optimize the initializtion phase by using threads.

```
public class FindLeftmostZero {
   static final int N = 10000000;
  public static void main(String[] args) {
    int data[] = new int[N];
    //assume occurrence of zero equally likely for all numbers generated
    for(int j = 0; j < N; j++)
      data[j] = (int)(Math.random()*N);
    int index = 0;
    while(index < data.length && data[index] != 0) index++;
    if(index == data.length)
      System.out.println("No zero");
    else
      System.out.println(index);
  }
}
```

Assignment 2

Please complete the problems listed below. This assignment forms part of the assessment for this module and you are required to upload your solution in the template file available on Moodle. This file should be used to write your solution and should be uploaded to Moodle on or before midnight on Sunday next. You must include as header your name and student number.

Question 1

MergeSort continuously divides the data into segments until segments of size 1 are reached. It then begins the merging phase. This is the expensive part. Improvements could be made if we could reduce the cost of merging. It turns out that *InsertionSort* is very efficient for small data sequences (say sequences of 100 values) where the data is partially ordered in the correct order and the displacement is small.
The idea is to combine *MergeSort* and *InsertionSort* to reduce the over head of merging. To do this we terminate the *MergeSort* division when segments of some given size are reached, use *InsertionSort* to sort the segments and then do the merging as before.
Your task is to implement this solution using the ForkJoin framework. You should test it with integer 10000000 integer array.

```
static void mergeSort(int f[], int lb, int ub){
    //termination reached when a segment of size 1 reached - lb+1 = ub
    if(lb+1 < ub){
        int mid = (lb+ub)/2;
        mergeSort(f,lb,mid);
        mergeSort(f,mid,ub);
        merge(f,lb,mid,ub);
    }
}
static void merge(int f[], int p, int q, int r){
    //p<=q<=r
    int i = p; int j = q;
    //use temp array to store merged sub-sequence
    int temp[] = new int[r-p]; int t = 0;
    while(i < q && j < r){
        if(f[i] <= f[j]){
            temp[t]=f[i];i++;t++;
        }
        else{
            temp[t] = f[j]; j++; t++;
        }
    }
    //tag on remaining sequence
    while(i < q){ temp[t]=f[i];i++;t++;}
    while(j < r){ temp[t] = f[j]; j++; t++;}
    //copy temp back to f
    i = p; t = 0;
    while(t < temp.length){ f[i] = temp[t]; i++; t++;}
}
}
```

Question 2

Given below is a class Point that is thread safe. Your task is to write a class, called CollectionPoint, that manages a collection of Point instances. This class *owns* the points under its control and must synchronize concurrent activity. Hence, the class must be thread safe. The class CollectionPoint should use an ArrayList to store points and must provide the following interface methods: add, that adds a new point to the collection; search, that searches for a point in the class and returns true or false; getAllX(int x) that returns a list of all points whose x-ordinate matches x; toString, that the list of points as a String. It should also be possible to replace an existing point with a new one.

The class Point listed below is immutable and, hence, does not require synchronization.

```
final class Point{
        private final double x, y;
        public Point(double x0, double y0){x = x0; y = y0;}
        public double x(){return x;}
        public double y(){return y;}
        public String toString(){return "("+x+","+y+")";}
}
```

Assignment 3

Please complete the problems listed below. This assignment forms part of the assessment for this module and you are required to upload your solution on Moodle. You must include as header your name and student number.

Question 1

A platform has space for at most 100 people at any one time. People are only admitted when the platform is open and the number of persons does not exceed the prescribed limit. Using condition variables write a class that could be used to control access to the platform. By creating multiple threads to represent people accessing the platform write a simulator for your control.

Question 2

A thread pipeline is a sequence of threads linked together with a chain of buffers. Each thread in the pipeline reads from the buffer preceding it in the chain and may write to the buffer following it in the chain. The diagram described in your lecture illustrated a pipeline of threads with connecting buffers. There is a single thread (`main`) that is the source of data in the pipe and it writes to the first buffer in the chain. The `Buffer` class, listed below, is used to provide nodes in the chain.

Your task is to construct a pipeline that will sort a list of integer values. The pipe will consist of *10* threads only. Each thread will sort its own list of values. The source will generate *1000000* random integer values guaranteed to be in the range *0..99999*. Each thread in the pipe will handle values in a given range: *thread0* values of x such that *0 <= x < 10000*, *thread1* values of x such that *10000 <= x < 20000*, etc. All we know is that the data is random in a given range and, consequently, we don't know how many values a given thread may have to deal with. A suggestion is to use an `ArrayList` to store values for the individual threads. This has the added advantage that you can use the `Collection.sort` algorithm to do the sort for you. When the sort is complete each thread should copy its data to a global shared array that is sorted.

Assignment 4

Please complete the problems listed below. This assignment forms part of the assessment for this module and you are required to upload your solution on Moodle. You must include as header your name and student number.
This is an assignment based on the Semaphore class discussed in the lecture and you may only use it to answer these questions.

Question 1
Given N threads ensure that they execute in order 0..N-1. Each thread should print a message listing its number in the sequence when it gets to execute. The values must be in order.

Question 2
A class that forces all threads to wait for an event is required. This class is to have two methods: waitEvent() that forces threads to wait and releaseAll() that releases all waiting threads. The class should be called WaitEventBarrier. It takes no arguments.
Write a simple test in main that creates a WaitEventBarrier and a number of threads that are forced to wait until released by main.

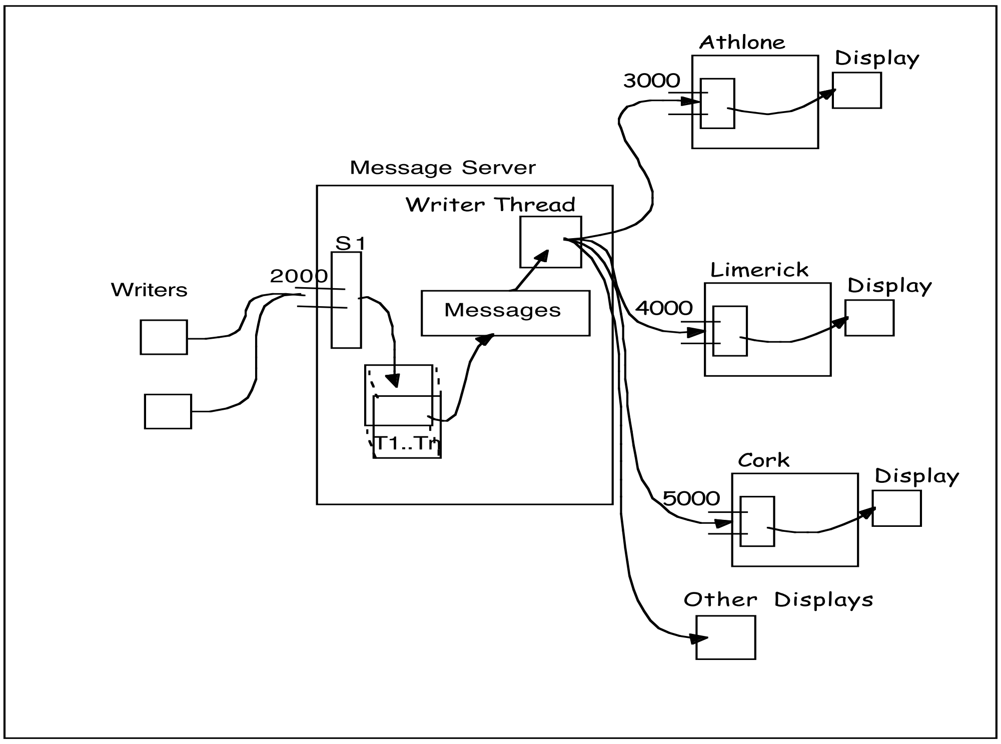
Question 3
A LatchBarrier is a control that forces *N* threads to rendezvous at a given point. When all threads reach the barrier then they are all released. The threads are automatically released by the barrier when the N^{th} thread invokes the public method wait on the barrier. This LatchBarrier has no reset method and, hence, once it releases threads it no longer works as a barrier. Note the class only has a constructor that takes *N* as argument and a wait method.
Note: this class differs from the one in **Question 2** because it has no release method but it does know the number of threads that will rendezvous.

Assignment 5

Please complete the problem described below. This assignment forms part of the assessment for this module and you are required to upload your solution on Moodle. You must include as header your name and student number.

A message server uploads and stores messages to be displayed on motorway notice boards. Write clients input messages and send them to the server. Each message consists of the text to be displayed and a list of the distributed notice boards that must display it. The server has a single write thread that automatically distributes messages to the appropriate designated display sites. Each site manages its own display and acts as a server listening for messages from the central server.
In reality each of the distributed sites would have separate IP addresses for the machines managing the local displays. But in this exercise we model the entire system in terms of a local host. Sites are identified by their port numbers. Each site has a name and a port number. The server only needs the unique port number to send a message to a site. The diagram below describes the entire system. The server listens for messages on port 2000 and the write thread writes to one or more sites using the required port numbers. The diagram just lists Athlone on port 3000, Limerick on port 4000 and Cork on port 5000. Other ports can also be added if you wish but the minimum 3 must be supported by your system.

Your task is to build a working model of this system.



Assignment 6

Please complete the problems described below. This assignment forms part of the assessment for this module and you are required to upload your solution on Moodle. You must include as header your name and student number. **Please copy your final solutions for each question to the relevant section in the file Assignment6.java on Moodle.**

Question 1 (6 marks)

Use the class MyArrayList to implement the interface MyList listed below using parallel streams and write a test program to test the methods. The class should be thread safe.

```
interface MyList<E>{
    public void add(E x);
    public void add(List<E> lst);
    public boolean forAll(Predicate<E> pr);
    public boolean exists(Predicate<E> pr);
    public long count(Predicate<E> pr);
    public List<E> map(Function<E,E> fn);
    public List<E> filter(Predicate<E> pr);
    public List<E> mapFilter(Function<E,E> fn, Predicate<E> pr);
}

class MyArrayList<E> implements MyList<E>{
    private ArrayList<E> data = new ArrayList<>();
    …
}
```

Question 2 (4 marks)

Given on Moodle is a program called *HappyNumsParallel.java* that computes the frequency of happy numbers in first 1000000 integer values. A class called HappySad is given. Your task is to write a separate program that solves this problem using parallel streams. I have put a partial solution on Moodle – it is called *HappyNumsParallelStreams.java*. It contains the Predicate happy that can be used as part of the stream. It is important that you comment on the performances of the two programs when you have finished your work.

GRIFFITH COLLEGE DUBLIN

QUALITY AND QUALIFICATIONS IRELAND
EXAMINATION

CONCURRENT PROGRAMMING

Lecturer(s):

External Examiner(s):

Date: 9th January 2018
Time: 2.15-5.15

THIS PAPER CONSISTS OF SIX QUESTIONS
FIVE QUESTIONS TO BE ATTEMPTED
ALL QUESTIONS CARRY EQUAL MARKS

APPENDIX ATTACHED TO THE BACK OF THE EXAMINATION PAPER

QUESTION 1

(a) Explain why sharing resources among multiple threads or processes can give rise to problems. Give at least one example.

Suppose you want to share the screen with multiple threads who may all try to write to it at the same time, describe how you would control access to this device. Your answer should include a sketch of the code used by a thread to control access.

(10 marks)

(b) Write a thread called Mapper that copies data from a segment in a source array to a copy array squaring the value of each element from the source array in the process. Both the lower bound and upper bound of the segment in the source array are passed as arguments to the thread.

(7 marks)

(c) In relation to threads what is the purpose of the join method? When is a thread in the state TIMED_WAITING?

(3 marks)

Total (20 marks)

QUESTION 2

(a) Given below is a sequential program that calculates the frequency of even values in a large array. Your task is to write a parallel solution that distributes the workload over the number of processors on the machine executing your program.

```
  public static void main(String[] args) {
   int f[] = new int[1000000];
   for(int j = 0; j < f.length;j++) f[j] = (int)(Math.random()*100000);
   int freq = 0;
   for(int j = 0; j < f.length; j++)
        if(f[j] % 2 == 0) freq++;
   System.out.println(freq);
}
```

The number of processors is given by
Runtime.getRuntime().availableProcessors())

(10 marks)

(b) The ForkJoinPool in Java 7 provides a divide and conquer algorithm that can be used to implement concurrent solutions to problems. Briefly explain how it works, and use it to implement a solution to calculating the frequency of odd numbers in a huge array. (**Appendix A** to this paper provides a template for a RecursiveTask that can be used to write your solution.)

(10 marks)

Total (20 marks)

QUESTION 3

(a) Explain the semantics of the class Lock in the Java.util.concurrent package and write down a code pattern for using an object lock in a code block.

(5 marks)

(b) Class Stack<T> below describes a generic stack, i.e. a last-in-first-out list, intended for use in a concurrent environment. Only methods push and pop are given. Your task is to re-write it using the Lock class so that the shared methods are safe for concurrent access.

```
class Stack<T> {
  private T[] s = (T[])new Object[100];
  private int n = 0;
  public bolean push(T item) {
    if(n < s.length){
      s[n] = item;  n++;
      return true;
    }
    return false;
  }
  public T pop() {
   if(n > 0){
     n--;
     return (T)(s[n]);
   }
   return false;
  }
}
```

(5 marks)

(c) Explain why the use of static variables and instance variables in the definition of a class complicates things when writing methods that support concurrent access for threads.

(5 marks)

(d) Given below is the class SharedArray. Explain why the method swap is deadlock prone and re-write it so that it is deadlock free.

```
class SharedArray {
        private int ff[];
        private Lock keys[];
        public SharedArray(int n){
                ff = new int[n];
                keys = new ReentrantLock[n];
                for(int j = 0; j < n; j++){
                        ff[j] = (int)(Math.random()*100);
                        keys[j] = new ReentrantLock();
                }
            }
            void swap(int j, int k) {
                keys[j].lock(); keys[k].lock();
                int t = ff[j]; ff[j] = ff[k]; ff[k] = t;
                keys[j].unlock(); keys[k].unlock();
            }
            //.....
}
```

(5 marks)

Total (20 marks)

QUESTION 4

(a) Explain the role of condition variables when writing a class that manages a shareable resource.

(5 marks)

(b) Explain why the await method of a condition variable must always be enclosed by a loop guard.

(4 marks)

(c) A server manages the allocation of 10 ports to clients. When a client requests a port from the server it receives a port number, if one is available. In the event that no port is available the client waits indefinitely for one to become free. Once a port is allocated no other client can use it. Using condition variables, implement the server class.

(11 marks)

Total (20 marks)

QUESTION 5

(a) Explain the semantics of the Semaphore class.

(4 marks)

(b) The generic class MyQueue, listed below, manages a fixed size queue of elements. This class has only two methods: join that appends new elements to the queue and get that removes and retrieves the element at the head of the queue. Your task is to re-write this class so that it can be shared among multiple threads. The semantics of the new class are that it should be thread safe and use Semaphores to control threads using its methods. A thread using the join method should wait if the queue is full and a thread using the get method should wait in the event that the queue is empty.

```
class MyQueue<T>{
 private LinkedList<T> queue = new LinkedList<T>();
```

```
  private int maxSize;
  public MyQueue(int m){maxSize = m;}
  public void join(T x){
    if(queue.size() < maxSize) queue.add(x);
  }
  public T get(){
   if(queue.size() > 0) return queue.removeFirst();
   return null;
  }
}
```

(10 marks)

(c) The following class will deadlock by invoking method t(). Why?

```
class A{
    private Semaphore sem;
    public A(){sem = new Semaphore(0);}
    synchronized void t(){
            ...
            try{sem.acquire();
            }catch(InterruptedException e){}
    }
    synchronized void t1(){
            ...
            sem.release();
    }
}
```

(3 marks)

(d) Explain how an instance of a CountDownLatch can be used to force N threads to wait for a signal to begin working.

(3 marks)

Total (20 marks)

QUESTION 6

(a) Explain how functions and streams in Java 8 facilitate programmers in stating **what** they want delivered rather **how** it is to be delivered.

(4 marks)

(b) One of the prime motivations for introducing functions and streams in the Java programming language was to provide support for parallel processing. Explain the semantics of parallel streams.

(4 marks)

(c) With reference to message passing concurrent implementations explain the difference between channel-based systems and actor-based systems. What type of message passing system does Scala use?

(5 marks)

(d) The program listed below uses 4 threads to calculate the sum of the even values in an integer array. It divides the workload equally between the 4 threads, waits for them to complete their calculations and then calculates the final result by invoking the result method of each thread. The code for the thread class Summer is also given.

Your task is to re-write this code using relevant functions and a parallel stream. See **Appendix B** for a list of relevant methods.

```
import java.util.function.*;
import java.util.stream.*;
import java.util.stream.Collectors.*;
public class Question6{
        public static void main(String args[]){
                Integer data[] = new Integer[1000];
                for(int j = 0; j < 1000; j++) data[j] = (int)(Math.random()*10000);
                Summer th[] = new Summer[4];
                for(int j = 0; j < 4; j++){
                  th[j] = new Summer(data,j*250,(j+1)*250);
                  th[j].start();
                }
                for(Summer t : th)
                  try{t.join();}
                  catch(InterruptedException e){}
                Integer k = 0;
                for(int j = 0; j < 4; j++) k += th[j].result();
                System.out.println("Sum = "+k);
        }
}
class Summer extends Thread{
        Integer[] dt;
        int lb, ub;
        Integer sum;
        Summer(Integer[] f, int l, int u){
                dt = f; lb = l; ub = u;
        }
        public void run(){
          sum = 0;
          for(int j = lb; j < ub; j++){
            if(dt[j] % 2 == 0)
               sum += dt[j];
          }
        }
        public Integer result(){return sum;}
}
```

(7 marks)

Total (20 marks)

Appendix A

Template for ForkJoin Pool Task

```
class <Name> extends RecursiveTask<V> {
   static final int BaseBlockSize = …;
   <Name>(…){
      //constructor
   }
   protected <V> compute() {
      if(… <= BaseBlockSize) {
         // execute sequential code
       } else {
         //divide job size
         <Name> left  = new <Name>(…);
         <Name> right = new <Name>(…);
         left.fork();
                     right.fork();
         <V> <varName> = right.join();
         <V> <varName>  = left.join();
         return <result>;
       }
    }
}
```

Appendix B
Tables Listing Function and Stream Methods

Specialized Function Types

Function Name	Argument Type	Return Type	Abstract Method Name	Purpose
Supplier<T>	None	T	get	Takes no argument and return a value of type T
Consumer<T>	T	void	accept	Consumes a value of type T
Function<T,K>	T	K	apply	A function that takes a value of type T as argument and returns a value of type K
BiFunction<T,U,R>	T,U	R	apply	A function that takes two arguments of type T, U ,and returns a value of type K
BiConsumer<T,U>	T, U	void	accept	Consumes values of type T and U
UnaryOperator<T>	T	T	apply	A function that takes a value of type T as argument and returns a value of type T
BinaryOperator<T>	T, T	T	apply	A function that takes two values of type T as argument and returns a value of type T
Predicate<T>	T	boolean	test	A function that takes a value of type T and returns a boolean value.
BiPredicate<T, U>	T, U	boolean	test	A function that takes two arguments of type T and U and returns a boolean value.

Stream Supplier Methods

Supplier Name	**Argument Type**	**Return Type**	**Semantics**
generate()	Supplier<T> f	static<T> Stream<T>	Returns an infinite stream of values where each value is generated by the Supplier function f.
iterate()	T seed, UnaryOperator <T> f	static<T> Stream<T>	Returns an infinite stream of values obtained by recursive application of the the function generating the sequence seed, f(seed), f(f(seed)), ..
limit()	long maxSize	Stream<T>	Limits the number of elements supplied through a stream to at most maxSize.
of()	T … values	static<T> Stream<T>	Returns a stream whose elements are the specified values

Stream Consumer Methods

Consumer Name	**Argument Type**	**Return Type**	**Semantics**
forEach()	Consumer<? Super T>	Void	Executes consumer function for each element supplied by the stream
allMatch()	Predicate<? Super T>	boolean	Returns true if all elements supplied by the stream satisfy the given predicate; false otherwise
anyMatch()	Predicate<? Super T>	boolean	Returns true if any element supplied by the stream satisfies the given predicate; false otherwise
count()		long	Returns the number of elements supplied by the stream

Stream Intermediate Operation Methods

Name	Argument Type	Return Type	Semantics
filter()	Predicate<? Super T>	Stream<T>	Returns a stream of those elements it receives that satisfy the given predicate.
map()	Function<? Super T, ? extends R>	<R> Stream<R>	Returns a stream of the elements it receives mapped by the given function
flatMap()	Function<? Super T, ? extends Stream<? extends R>>	<R> Stream<R>	This operation applies a function that itself returns a stream to each element of this stream.
distinct()		Stream<T>	Returns a stream consisting of the distinct elements in this stream. The definition of equality is based on the equals method implemented for the given class.
sorted()		Stream<T>	Returns a stream of the elements of the consumed stream, sorted by natural order (based on implementation of compareTo)
sorted()	Comparator(? super T compare)	Stream<T>	Returns a stream of the elements of the consumed stream, sorted by order defined by the Comparator (based on implementation of compare).