Information

INF2440 Spring 2018 written exam

Date and time: June 11th, 2018 at 09:00.

Duration: 4 hours

Material allowed: All written material is allowed. No electronic aid is allowed. The lecture slides are attached as a PDF.

1 Question 1.1: Java Thread Creation

In a Java program, the programmer must create and start every thread that the program uses. Is this right?
Select an alternative:

- Yes, the program must create and start all threads.
- Yes, the program must create and start all threads in main().
- No, the first thread is created automatically and is automatically started and set to execute the main() method. The remaining threads must be created and started by one of the other threads in the program.
- No, the programmer must merely provide the code for the threads. The compiler will then automatically generate the threads at compile time and ensure that they are started when the program runs.
- No, all threads are started automatically.

Maximum marks: 4

2 Question 1.2: Java Threads Features

Which of the following statements are true for Java Threads that are created in the same program? (More than one answer is possible.)
Select one or more alternatives:

- Java threads can execute in parallel independently of one another on multi-core machines.
- If there are more threads than available cores, the program cannot be run.
- Java threads execute at the same speed because each thread is assigned to its own core .
- Java threads can access the local variable of one another.
- Java threads reside in a common part of memory and can access static variables declared in the class containing the threads.
- Java threads always execute one at a time.

Maximum marks: 6

3 Question 1.3: Java Synchronized

Can we ensure that only one thread at a time is executing a method by prefixing the method with the

synchronized keyword.

Select an alternative:

- Yes, if more than one thread calls the method, the Java system ensures that only one thread is executing the method at a time.
- Yes, except for the initial thread running main() -- it is always allowed to execute the method even if another thread is running the method at the same time.
- No, inside the method, we must add a synchronization of some kind, e.g., CyclicBarrier.
- Only if the method return type is void.

Maximum marks: 5

4 **Question 1.4: Performance of Java Threads**

Which of the following statements is closest to reality?

Select an alternative:

- Java threads can be created and run so quickly that even if we are just doing two or three integer additions, it is worth doing them in a separate thread because the thread will execute quickly on another core.
- Java threads take a lot of time to create and to synchronize so in general a thread should not be created unless it can do a lot of work, e.g., at least many thousands of additions.
- Java Threads are cheap to create as long as there are less threads than cores on the machine.
- Java threads are expensive to create but it becomes cheaper and cheaper to create threads the more threads a program creates.

Maximum marks: 5

i **Question 2: Cyclic Barriers**

In this question, you should consider the program Problem.java attached as a PDF-document.

5 **Question 2.1: Cyclic Barrier - termination with two threads**

When the program is called with the parameters "2 0", will the program always terminate? Explain your answer.

Fill in your answer here

Format | B I U x_2 x^2 | I_x | | | | | | | | Ω | | | ∑ |

Words: 0

Maximum marks: 10

6 Question 2.2: Cyclic Barrier - output with two threads

When the program is called with the parameters "2 0", will the program *always* print the same text? Explain your answer.

Fill in your answer here

Format | B I U x_2 x^2 | I_x | | | | | | | | Ω | | | ∑ | ABC |

Words: 0

Maximum marks: 10

7 Question 2.3 Cyclic Barrier: Termination with four threads

When the program is called with the parameters "2 2", will the program always terminate? Explain your answer.

Fill in your answer here

Format | B I U x_2 x^2 | I_x | | | | Ω | | | ∑ | ABC |

Words: 0

Maximum marks: 12

8

Question 2.4: Cyclic Barrier: Output with four threads

When the program is called with the paramters "2 2", will the program always print the same text? If so, show the output. If not, give two different examples of what the program prints.

Fill in your answer here

Format | B I U x_2 x^2 | I_x | | | | Ω | | | ∑ | ABC |

Words: 0

Maximum marks: 12

i

Question 3: Finding Prime Deserts

This question is based on Erathostenes Sieve that finds prime numbers up to a given number N. You should already be quite familiar with the sieve as you have implemented it in Oblig 3 (attached as PDF).

This question is about finding so-called Prime Deserts. A Prime Desert is an interval [A, B] where A < B, A and B

are prime numbers, and there is no prime between A and B. Examples of such deserts are [2, 3], [7, 11], [23, 29], and [337, 347]. The size of a Prime Desert is defined as B-A-1, i.e., the number of integers strictly between A and B. A is called the start point of the Prime Desert and B is called the end point.

You are to write a Java program that generates a list of Prime Deserts. The list should be sorted so that the intervals come in ascending order, i.e., the starts points are in ascending order. The first Prime Desert is [2, 3] and thereafter, the next Prime Desert should be larger than the previous one throughout the list. Furthermore, you should find all Prime Deserts where the end point no greater than N, but you should exclude from the list any Prime Desert whose size is not greater than the size of the previous Prime Desert in the list. For example, that means that the start of the list is: [2,3] ,[3, 5], [7, 11], [23, 29], [89, 97],... i.e., [5, 7], [11, 13], [13, 17], [17, 19] and more are left out because their size is not greater than the previous Prime Desert in the list. To be sure, you may not leave out a Prime Desert, if it is larger than the previous in the list and smaller than the next one; for example, you cannot have a list that starts with [2, 3] followed by [23, 29].

You need not write the entire program as you can assume that you already have the code that you wrote for Oblig 3. You are welcome to base your code on Modell2 from the lectures in Week 5 (uke5).

9

Question 3.1: Sequential Program for finding Prime Deserts

Write a sequential Java program that finds the list of Prime Deserts specified in the introduction to Question 3.
Fill in your answer here

```
1
```

Maximum marks: 20

10

Question 3.2: Parallel Program to find Prime Deserts

Write a parallel Java program that finds the list of Prime Deserts specified in the introduction to Question 3. Strive to achieve a speedup over the sequential version.

Fill in your answer here

```
1
```

Maximum marks: 40

Question 4: Bubblesort

Bubblesort is a sorting algorithm that sorts, e.g., an integer array A, by comparing two neighboring elements and exchaning them, if the first one is greater than the second. The sorting consists of a number of passes. Each pass goes thru the array one element at a time and performs the exchange, if necessary. At the end of pass k, the k'th largest number is in place at the top of the array, so the next pass need not consider the top k elements of the array.

A sequential bubblesort of an integer array is given as a PDF file.

11

Question 4.1 Parallelizing Bubblesort

How can Bubblesort be parallelized? Describe the design of a solution.

Fill in your answer here

Format | B I U x_2 x^2 | I_x | | | | Ω | | | Σ | ABC |

Words: 0

Maximum marks: 20

12 Question 4.2: Parallel Bubblesort

Write a Java program implementing your design of a parallel version of Bubblesort from Question 4.1. Strive to have a speedup over the sequential version. Put any added explanation that you have as comments in the code.

Fill in your answer here

```
1
```

Maximum marks: 40

13 Question 5.1: Recursion and Parallelism

There are many ways of parallelizing a recurvise algorithm. Of the following statements, which of them is a good idea (you may select more than one):

Select one or more alternatives:

- When the recursive calls form a tree structure, it is a good idea to spawn and use threads at the bottom of the tree only.
- A simple way to parallelize a recursive program is to have one thread per recursive call. This almost always leads to achieving a good speedup.
- It is important to limit the number of threads.
- When the recursive calls form a tree structure, it is a good idea to spawn new thread at the top of the tree and limit the number of threads at the bottom of the tree.
- Recursive calls are relatively expensive compared to creating and running a thread.
- Recursive calls are several orders of magnitude cheaper than creating and running a thread.
- When the recursive calls form a tree structure, it is a good idea to replace branches at the lower levels of the tree with simple sequential solutions of the problem.
- It is important to balance the number of threads and the amount of work done by each thread.

Maximum marks: 16

Question 12
Attached

```
static void bubbleSort(int[] arr) {
   int n = arr.length;
   int temp;
   for (int i=0; i < n; i++){
      for (int j=1; j < (n-i); j++){
          if(arr[j-1] > arr[j]){
             //swap elements
             temp = arr[j-1];
             arr[j-1] = arr[j];
             arr[j] = temp;
          }

      }
   }
}
```

Question 12
Attached

```
static void bubbleSort(int[] arr) {
   int n = arr.length;
   int temp;
   for (int i=0; i < n; i++){
      for (int j=1; j < (n-i); j++){
          if(arr[j-1] > arr[j]){
             //swap elements
             temp = arr[j-1];
             arr[j-1] = arr[j];
             arr[j] = temp;
          }

      }
   }
}
```

Question 5
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 5
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 6
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 6
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 7
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 7
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 8
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 8
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```