Question 1: Java and Synchronization

Question 1.1: Java Threads Startup (1 point)

In a Java program, there is one main thread that is automatically started when a program starts: it executes the main method of the program. This method can start other threads. Can those threads also start other threads? Provide a short explanation.

Svar:

Ja, threads kan start opp andre threads. Det er ingenting i javas class Thread som sier at threads ikke kan starte andre threads og threads brukes til å fordele oppgaver og lar de kjøres parallelt.

Question 1.2: Java Threads Execution on Multicore (5 points)

Explain briefly how 100 Java threads can appear to execute concurrently asif the CPU had 128 cores – despite it having only 4 cores.

Svar:

Hvis vi har flere tråder enn kjerner, vil klokka i maskinen sørge for at trådene av og til avbrytes og en annen tråd får kjøretid på kjernen. Dette kalles Time scheduling og opprettholder illusjonen av at vi har mange cores. Altså i stedet for at vi først gjør 1 oppgave og så en annen, så gjør vi si halve 1, så halve 2 , så resten av 1 og tilslutt resten av 2. Vi vet ikke hvor lang oppgavene er så mens de har arbeidstid på kjerne så kan de bli ferdig og da begynner kjernen på en ny oppgave. Dette gjør at tråder kan bli ferdig i litt tilfeldig rekkefølge.

Question 1.3 Java Synchronized (12 points)

There are alternatives to using the Java keyword synchronized including many very specialized ways of synchronization. Pick your favorite alternative to synchronized and explain why you prefer it and how it can replace synchronized. Your preference can be based on any type of argument: be it performance, usability, convenience, or a specialized use. The most important part of your answer will be your presentation of the advantages and disadvantages in connection with any specific application of your choice.

Svar:

Min favoritt synchronized alternativ er CyclicBarrier. Jeg like CyclicBarrier fordi konseptet bak hvordan den funker er simpel og dette hjelper meg med å sette opp logikken for parallelliseringen når det er passende for oppgaven. Måten CyclicBarrier synkroniserer på er at det lages et CyclicBarrier objekt som trådene synkroniseres på. Trådene kan kalle på .await() på CyclicBarrier objektet. Når N antall tråder har kalt på .await() slippes alle tråder videre, N blir bestemt når CyclicBarrier objektet lages. Denne grensen/barrieren stopper tråder fra å gå for langt fram i oppgaven sin, som kan være nyttig hvis vi senere trenger noe informasjon som først skal håndteres av en av de andre trådene. Enkelt fortalt så samler det opp trådene og lager knutepunkter hvor vi vet at tidligere oppgaver gitt til trådene er gjort så for eksempel hvis tråd 0

trenger noe info som tråd 1 først jobber med, så kan vi setter opp en barrier så tråd 1 jobber på dataen før barrieren, mens tråd 0 jobber på den etter og dermed oppnår synkronisering. Fordelene med CyclicBarrier er at CyclicBarrier kan bli gjenbrukt ved å resettes i motsetning til CountDownLatch. En annen fordel er at CyclicBarrier kan ta en Runnable oppgave å gjøre når barrieren brytes som gjør det lett å deler opp oppgaver på data og når barrieren brytes gjør den siste inn av trådene denne runnable oppgaven til for eksempel å kombinere dataene fra arbeidet trådene gjorde for de ventet ved barrier objektet. Man kan også hente data om hvor mange tråder som venter og hvor mange som trengs for å bryte barrieren. Noen ulemper med CyclicBarrier er at det er ikke sånn at alle synkronisering tilfeller passer eller er optimale ved bruk av CyclicBarrier. Et eksempel på dette er at vi har en array hvor x antall tråder skal gjøre en oppgave og så skrive på et felt i en array, dette kan gjøres med CyclicBarrier, men er mer effektivt å gjøre med en Semaphore. En kort oppsummering er at CyclicBarrier er bra når du trenger at trådene dine er ferdig med en del oppgaver før de fortsetter videre, men mindre bra på å håndtere jobb på samme delte plasser mellom flere tråder.

Question 2: Join using Semaphores
Question 2.1 Join Replacement (16 points)
You are to achieve the effect of Java's join, but instead of using join, you should use semaphores as in the Java class Semaphore.The idea is to modify the JoinP Java program given below to NOT use join, but instead use semaphores. Your task is now to write a version of JoinP where you have the join with some Java code that makes the program work like the original JoinP program without using join but instead using semaphores.Provide the resulting program along with a short explanation of your solution.Here is the JoinP program:

```
import java.util.concurrent.*;
class JoinP {
        public static void main(String[] args) {
            int numberofthreads = 10;
            Thread[] t = new Thread[numberofthreads];

            for (int j = 0; j < numberofthreads; j++) {
              (t[j] = new Thread( new ExThread() )).start();
            }

            try {
              for (int k = 0; k < numberofthreads; k++) t[k].join();
            } catch (Exception e) { return; }

        }
        static class ExThread implements Runnable {
            public void run() {
              try {
                TimeUnit.SECONDS.sleep(10);
```

```
                } catch (Exception e) { return;};
              }
          }
}
```

Svar Code:

```
import java.util.concurrent.*;
class SemaphoresJoinP {
  public static void main(String[] args) {
       System.out.println("Start of main");
       int numberofthreads = 10;
       Thread[] t = new Thread[numberofthreads];
       Semaphore sem = new Semaphore(numberofthreads );
       for (int j = 0; j < numberofthreads; j++) {
          (t[j] = new Thread( new ExThread(sem) )).start();
       }
       try {
         TimeUnit.SECONDS.sleep(1);
       } catch (Exception e) {};
       while(sem.availablePermits() < numberofthreads ){
         try {
         System.out.println("waiting ...");
         TimeUnit.SECONDS.sleep(1);
         } catch (Exception e) {};
       }
       System.out.println("End of main");
  }
  static class ExThread implements Runnable {
       Semaphore sem;
       public ExThread(Semaphore sem){
         this.sem = sem;
       }
       public void run() {
          try {
            sem.acquire();
            System.out.println(Thread.currentThread().getName());
            TimeUnit.SECONDS.sleep(10);
            System.out.println(Thread.currentThread().getName() +" is done Sleeping");
            sem.release();
          } catch (Exception e) { return;};

       }
  }
}
```

Forklaring:
Tanker er at join venter til tråden er dø før den fortsetter videre og for loopen passer pa at alle 10 blir sjekket. Koden over erstatter join med semaphore som har “numberofthreads” tillatelser. Hver tråd henter en tillatelse, sover og gir fra seg tillatelsen. Programmet sover 1 sekund så trådene får tid til å hente tillatelsene sine og kjører en while loop som holder programmet i den til alle tillatelsene har kommet tilbake til semaphoresen. Tanken var at hvis vi bruker semaphore sin tillatelse til å se om alle er ferdige siden tråden holder på tillatelsen til den er ferdig med jobben.

Question 2.2 Test Case (12 points)
You are to write a Java program that demonstrates a test case for the program that you wrote in 2.1. Explain the test that you chose and why you think it shows that your program from 2.1 works – at least for your chosen test case (it does not – at all – have to be comprehensive – just show a typical case). Each thread could, for illustration, print what it does at each step – be sure to include an id of the thread doing the printing.Hint: You can “schedule” when threads actively try to do stuff by delaying them using, e.g.,TimeUnit.SECONDS.sleep(10);Provide the program and its output and any comments that you might have.

Svar Code:
```
import java.util.concurrent.*;
import java.util.Random;

class SemaphoresJoinPTest {
  public static void main(String[] args) {
       System.out.println("Start of main");
       int numberofthreads = 10;
       Thread[] t = new Thread[numberofthreads];
       Semaphore sem = new Semaphore(numberofthreads);
       Random randomSleep = new Random(); //for testing vil jeg ha trådene til å sove i en tilfeldig tid fra 5-10 sekunder
       System.out.println("We have " + sem.availablePermits() +" permits in the Semaphore");
       for (int j = 0; j < numberofthreads; j++) {
          (t[j] = new Thread( new ExThread(sem, 5 + randomSleep.nextInt(5)) )).start();
       }
       try {
         TimeUnit.SECONDS.sleep(1);
       } catch (Exception e) {};

       System.out.println("Waiting for threads");

       while(sem.availablePermits() < numberofthreads){   //holder oss i while til alle permits er gitt tilbake sånn alle tråder er ferdig.
         try {
```

```
            System.out.println("Permits left: " + sem.availablePermits());
            System.out.println("waiting ...");
            TimeUnit.SECONDS.sleep(1);
          } catch (Exception e) {};
        }
        System.out.println("End of main");
  }
  static class ExThread implements Runnable {
        Semaphore sem;
        int sleepTime;
        public ExThread(Semaphore sem, int sleepTime){
          this.sem = sem;
          this.sleepTime = sleepTime;
        }
        public void run() {
           try {
             System.out.println(Thread.currentThread().getName() + " try to acquire!");
             sem.acquire();
             System.out.println(Thread.currentThread().getName() + " is going to sleep for " +
sleepTime + " seconds");
             TimeUnit.SECONDS.sleep(sleepTime);
             System.out.println(Thread.currentThread().getName() +" is done Sleeping");
             sem.release();
             System.out.println(Thread.currentThread().getName() +" released permit");
           } catch (Exception e) { return;};

        }
  }
}
```

Output:

```
>java SemaphoresJoinPTest
Start of main
We have 10 permits in the Semaphore
Thread-6 try to acquire!
Thread-4 try to acquire!
Thread-2 try to acquire!
Thread-9 try to acquire!
Thread-8 try to acquire!
Thread-3 try to acquire!
Thread-7 try to acquire!
Thread-0 try to acquire!
Thread-1 try to acquire!
```

```
Thread-5 try to acquire!
Thread-7 is going to sleep for 5 seconds
Thread-1 is going to sleep for 7 seconds
Thread-9 is going to sleep for 7 seconds
Thread-2 is going to sleep for 7 seconds
Thread-3 is going to sleep for 9 seconds
Thread-5 is going to sleep for 9 seconds
Thread-8 is going to sleep for 5 seconds
Thread-4 is going to sleep for 8 seconds
Thread-0 is going to sleep for 8 seconds
Thread-6 is going to sleep for 7 seconds
Waiting for threads
Permits left: 0
waiting ...
Permits left: 0
waiting ...
Permits left: 0
waiting ...
Permits left: 0
waiting ...
Thread-7 is done Sleeping
Thread-7 released permit
Thread-8 is done Sleeping
Thread-8 released permit
Permits left: 2
waiting ...
Permits left: 2
waiting ...
Thread-1 is done Sleeping
Thread-1 released permit
Thread-2 is done Sleeping
Thread-9 is done Sleeping
Thread-9 released permit
Thread-2 released permit
Thread-6 is done Sleeping
Thread-6 released permit
Permits left: 6
waiting ...
Thread-0 is done Sleeping
Thread-0 released permit
Thread-4 is done Sleeping
Thread-4 released permit
Permits left: 8
```

waiting ...
Thread-3 is done Sleeping
Thread-3 released permit
Thread-5 is done Sleeping
Thread-5 released permit
End of main

Question 3: Double-bubblesort

Bubblesort is a sorting algorithm that sorts, e.g., an integer array A, by repeatedly going through the array from one end to the other comparing neighboring elements to each other – and swapping them when they are not in order. Double-bubblesort is a variant of bubblesort that works by comparing not two elements at a time but rather three elements at a time and swapping them so that they are in order.

Question 3.1 Sequential Double-bubblesort (8 points)

Write a sequential version of double-bubblesort in Java.Provide the Java program as your answer.

Svar code:

```
import java.io.*;

class DBS{
   static void dbubbleSort(int arr[], int n){
      int i, j, temp;
      boolean swapped;
      for (i = 0; i < n - 2; i++){
         swapped = false;
         for (j = 0; j < n - i - 2; j++){
           if (arr[j] > arr[j + 2] && arr[j] > arr[j + 1]){
              // swap arr[j] and arr[j+2]
              System.out.println("Swap: " + arr[j] +" and "+ arr[j + 2]);
              temp = arr[j];
              arr[j] = arr[j + 2];
              arr[j + 2] = temp;
              swapped = true;
           }
           if (arr[j+1] > arr[j + 2]){
              // swap arr[j] and arr[j+1]
              System.out.println("Swap: " + arr[j+1] +" and "+ arr[j + 2]);
              temp = arr[j + 1];
              arr[j + 1] = arr[j + 2];
              arr[j + 2] = temp;
```

```
            swapped = true;
          }
          if (arr[j] > arr[j + 1]){
            // swap arr[j] and arr[j+1]
            System.out.println("Swap: " + arr[j] +" and "+ arr[j + 1]);
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
            swapped = true;
          }

        }
        // IF no two elements were
        // swapped by inner loop, then break
        if (swapped == false)
            break;
    }
}

// Function to print an array
static void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        System.out.print(arr[i] + " ");
    System.out.println();
}

// Driver program
public static void main(String args[])
{
    int arr[] = { 65, 37, 23, 12, 20, 10, 96 };
    int n1 = arr.length;
    int arr10[] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
    int n2 = arr10.length;
    dbubbleSort(arr, n1);
    System.out.println("Sorted array: ");
    printArray(arr, n1);

    dbubbleSort(arr10, n2);
    System.out.println("Sorted array: ");
    printArray(arr10, n2);
}
```

}

Question 3.2 Testing Double-bubblesort (5 points)
Write a simple Java test program showing that your program from 3.1 can sort an array of 10 elements containing the numbers from 1 to 10 in reverseorder. Print suitable test output from the program. Provide the program and its output.

Svar:
Code bitten er tatt fra svaret mitt i 3.1
Code:

```
    // Function to print an array
    static void printArray(int arr[], int size)
    {
        int i;
        for (i = 0; i < size; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver program
    public static void main(String args[])
    {
        int arr[] = { 65, 37, 23, 12, 20, 10, 96 };
        int n1 = arr.length;
        int arr10[] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
        int n2 = arr10.length;
        dbubbleSort(arr, n1);
        System.out.println("Sorted array: ");
        printArray(arr, n1);

        dbubbleSort(arr10, n2);
        System.out.println("Sorted array: ");
        printArray(arr10, n2);
    }
```

Output:
>java DBS
Swap: 65 and 23
Swap: 65 and 12
Swap: 37 and 12
Swap: 65 and 20

Swap: 37 and 20
Swap: 65 and 10
Swap: 37 and 10
Swap: 23 and 20
Swap: 20 and 12
Swap: 23 and 10
Swap: 20 and 10
Swap: 12 and 10
Sorted array:
10 12 20 23 37 65 96

Swap: 10 and 8
Swap: 10 and 7
Swap: 9 and 7
Swap: 10 and 6
Swap: 9 and 6
Swap: 10 and 5
Swap: 9 and 5
Swap: 10 and 4
Swap: 9 and 4
Swap: 10 and 3
Swap: 9 and 3
Swap: 10 and 2
Swap: 9 and 2
Swap: 10 and 1
Swap: 9 and 1
Swap: 8 and 6
Swap: 8 and 5
Swap: 7 and 5
Swap: 8 and 4
Swap: 7 and 4
Swap: 8 and 3
Swap: 7 and 3
Swap: 8 and 2
Swap: 7 and 2
Swap: 8 and 1
Swap: 7 and 1
Swap: 6 and 4
Swap: 6 and 3
Swap: 5 and 3
Swap: 6 and 2
Swap: 5 and 2
Swap: 6 and 1

Swap: 5 and 1
Swap: 4 and 2
Swap: 4 and 1
Swap: 3 and 1
Swap: 2 and 1
Sorted array:
1 2 3 4 5 6 7 8 9 10

Question 3.3 Parallelizing Double-bubblesort (35 points)
How can your program from 3.1 be parallelized?Describe the design of a solution that MUST be loyal to the algorithm, i.e., splitting the array into k parts that are double-bubblesorted individually then merge sorted, is not loyal as much of the speedup is gained by using merging, which is much more efficient for large arrays than any bubblesort. Hint: spend time on describing the parallelization as this is central to the course.

Svar:
For parallellisering av denne algoritmen velger vi å dele opp arrayen vi skal jobbe på mot antall tråder vi bruker. Hvis array størrelsen ikke kan deles på antall tråder, må vi sette det opp sånn at den som får siste del av arrayen får resten. Hvis vi tar 3.2 arrayen som eksempel så kan vi velge å ha 3 tråder til å håndtere de 10 array posisjonene. 10 kan ikke deles pent på 3 så tråd-1 får arrayen fra posisjon 0,1 og 2 og kjører algoritmen på det, tråd-2 får 3,4 og 5, tråd-3 får 6,7,8,9. For synkronisering av denne paralleliseringen kan vi bruke en CyclicBarrier. Vi kan gi CyclicBarrieren vår merge sort som en runnable task sånn at når alle trådene kommer til barrieren og er ferdig med å sortere sin del, begynner den siste tråden inn i barrieren å kjøre merge sort algoritmen vår. Siden vi vet at elementene fra startposisjon til sluttposisjon for hver tråds arbeidsområde er sortert etter at den har truffet barrieren og ingen av dem overlappet, så kan vi sammenlikne elementet i startposisjon med elementene i de andre trådenes sluttposisjon. Hvis den minste verdien i den sorterte array biten er større enn den største biten i de andre sorterte arraybitene så er denne biten etter bitene som har et mindre tall i siste posisjon. Hvis jeg fortsetter eksempelet så vil tråd-1 sortere 10, 9, 8 om til 8,9,10 og 8 er større enn tråd-2 og tråd-3 sine verdier i de siste posisjonene etter sorteringen (7 og 4). Merge sorten trenger da bare å sortere de sorterte elementene etter kantverdiene(første og siste element i sorterings posisjonene). Men hvis første element er mindre enn siste element i en sortert bit, men større enn første element må disse to bitene sorteres sammen. Eksempel hvis vi har en sortert bit med elementer fra 2 til 10 og en annen bit med elementer fra 7 til 15. Da må disse to bitene sorteres sammen til en større bit. Ved å ikke overlappe noen av bitene som sorteres slipper vi feil med at 2 tråder jobber og forandrer på samme felt som kan føre til at vi ikke ender opp med samme verdier i arrayen som vi startet med. Siden vi synkroniserer på CyclicBarrier vet vi at sorteringen er ferdig når vi treffer barrieren før mergesort starter.

Question 4.1 The Speed of Light in Vacuum (1 point)
What is the exact speed of light in vacuum?Give your answer in m/s and make sure it is exact.

Svar:
Det er 299 792 458 m/s

Question 4.2 Latency (5 points)
In 1988, the latency for a so-called ping request, that is the time to send a packet over the internet from one point, A, to another point B, and a reply making its way from B to A, was about 200 ms for a ping request from Scandinavia to the US West Coast – approximately 10,000 km. In 2020, the ping time is about 180 ms.Explain why the time is almost the same after 32 years---despite great improvements in transatlantic bandwidth.

Svar:
Hvis vi ser på Latency mer scientific, så er latency en funksjon av tid, avstand og lysets hastighet. Lysets hastighet i luft er litt tregere enn lysets hastighet i vakuum. Det er denne forskjellen i hastighet som vi tenker på som latency. Siden vi ikke endrer på avstanden så må vi endre på lyshastigheten(som ikke kan forandres). Så for forbedring trenger vi å komme nærmere lysets hastighet i vakuum. Dette er vanskelig og er grunnen til at vi kun har en liten forbedring.