

# 1 Day 1

## 1.0.1 Keywords

Thread, Thread-pool, concurrency, synchronized, volatile, race-conditions, atomicity, deadlocking

## 1.1 Exercise 1

### 1.1.1 A

Synkroniseringen vil være smart, når man har med så mange tal at gøre, da det bliver meget uoverskueligt med asynkronisk threads.

Man kan eventuelt køre den første thread synkront med de to andre, da den indeholder mange tal der oftest bliver udskrevet.

### 1.1.2 B

Det første problem man kan opserve er uoverskueligheden af data, da hver thread printer data, og det ikke klart hvilken thread som printer hvilke data. Ydermere er mængden af data der bliver printet meget stor, så noget af dataen bliver ikke opserveret.

Løsningen til dette ville først og fremmest være at printe navnet på den thread der printer dataen ud. I den forstad når man i dette tilfælde har med tre threads at gøre, kan man eventuelt printe "t1: 1" for thread 1 osv.

Derudover kan man køre de tre threads synkront. Sådan opnår man klarheden af hvilken thread der printer data på daværende tidspunkt.

Eventuelt kunne man køre den første thread synkront, da dens output er meget større end de to andre, og de to andre asynkront.

Det andet problem kunne være race conditions, men jeg mener ikke dette er et problem i denne opgave.

### 1.1.3 C

Jeg mener at den løsning jeg har givet i opgave B er korrekt, da den ville give overblik over dataen der bliver printet i programmet.

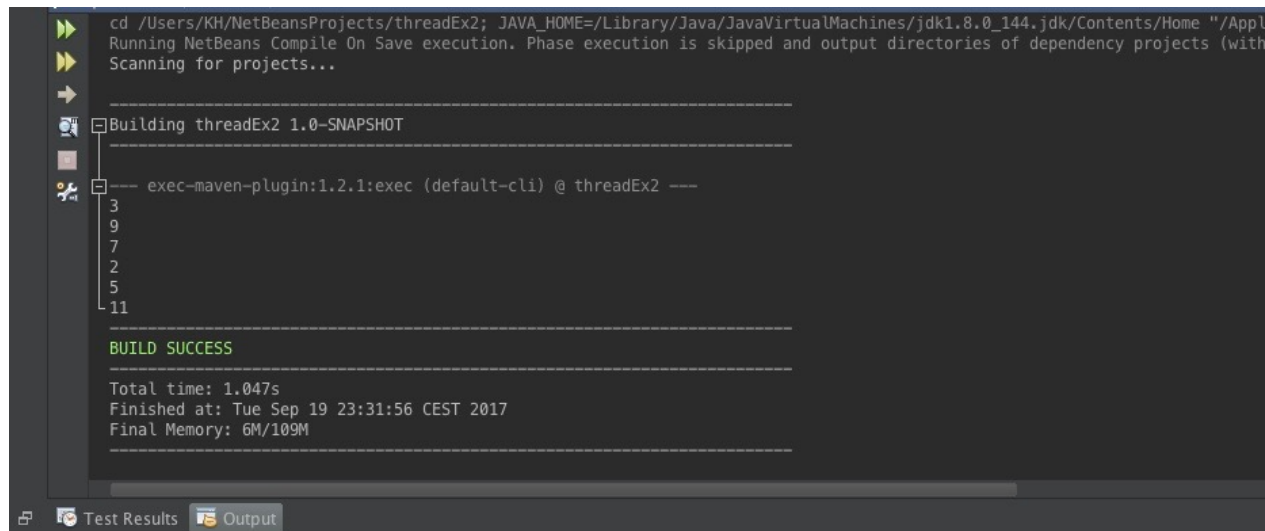
Mit argument er, da den første thread printer sin data først med sit navn før den data der bliver printet, som er en stor mængde data på en kort tidsperiode, har man her overblik over den data der bliver printet fra denne thread, og man er sikker på det er "t1", altså thread 1, der printer data.

Derefter printer de to andre threads deres navn ud med data, og da denne data kommer med pauser imellem, og i små mængder, har man overblik over hvilken thread der printer hvilken data, og dataoutputtet for hver thread er samlet til en hvis forstand.

## 1.2 Exercise 2

### 1.2.1 A

Den forventede error blev fremprovokeret, men dette opstod først da flere threads blev brugt. Med to threads opstod problemet ikke, den eneste ændring der var i outputtet, var at rækkefølgen var forskelligt for hver run. Herunder ses den forventede error da seks thread blev anvendt:



```
cd /Users/KH/NetBeansProjects/threadEx2; JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home "/App
Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of dependency projects (with
Scanning for projects...

-----
Building threadEx2 1.0-SNAPSHOT
-----
exec-maven-plugin:1.2.1:exec (default-cli) @ threadEx2 ---
3
9
7
2
5
11
-----
BUILD SUCCESS
-----
Total time: 1.047s
Finished at: Tue Sep 19 23:31:56 CEST 2017
Final Memory: 6M/109M
-----
```

### 1.2.2 B

Ud af 40 forsøg med skes threads opstod erroren 1 gang.

### 1.2.3 C

Metoden er blevet atomic ved at gøre metoden synchronized, hvilket betyder at det ikke bliver kørt asynkronisk.

### 1.2.4 D

Jeg mener at gøre metoden atomic har løst problemet, da værdien ikke længere tilgås af de seks threads asynkront.

## 1.3 Exercise 3

Problemet er at Ball klassen og UI klassen køre på samme thread, så når man trykker start bliver GUI blokkeret imens Ball køre i baggrunden, hvilket gør at GUI ikke er responsiv.

Dette løses ved at oprette en ny thread for Ball hver gang man trykker start.

## 1.4 Exercise 4

## 1.5 Exercise 5

### 1.5.1 A

Man får ikke den forventede resultat.

### 1.5.2 B

Grunden til man ikke får det forventede resultat, skyldes at de 40 threads køre på samme tid, og resultere i at de opdatere variabelen på samme tid.

### 1.5.3 C

Efter synchronized blev anvendt fik man det forventede resultat, dette skyldes at synchronized giver de mange threads mulighed for at køre en ad gangen og giver derfor de mange threads lov til at køre færdig.

## 2 Day 2

## 2.1 Exercise 1

### 2.1.1 A

Grundet den længere køretid når man snakker om `tcX.run()`, skyldes at koden bliver kørt slavisk. Dette kan løses ved at bruge threads.

### 2.1.2 B

Grunden til at `extends Thread` er simpelt at bruge, skyldes at der skabes en `run` metode som overrider. Hvor i dette tilfælde findes en `run` metode, og derfor manuelt kun skal tilføje `override`.

### 2.1.3 C

Den store forskel på at køre `run()` og start af de tre threads, er at `run()` som nævnt i opg A, køre koden slavisk og resultere i at programmet køre langsomt. Ved brug af de tre threads køre det hele på en gang, og deraf får en hurtigere køre tid.

Problemet med de tre threads er at man i outputtet får `empty` og `null`. Dette skyldes at de nævnte threads ikke har talt antallet af elementer færdigt, før at main thread har nået at printe antallet af elementer.

### 2.1.4 D

Resultatet varierer når man `parallel executor(Thread)` og resultatet med `run` er konstant.

sequential: 1774979977

parallel: 1041022772

Som man kan se i resultatet er execution time på de to, er at når man køre med threads køre programmet hurtigere.

## 2.2 Exercise 2

Har læst og forstået :) (Henvises til koden)

## 2.3 Exercise 3

### 2.3.1 A

- Fordelen ved at benytte threads ved genereringen af mange tal, er at kunne generere flere tal asynkront, og dermed formindske køretiden.
- Fire threads er ikke altid det bedste at bruge, da det kommer an på hvilken processor man har. I dette tilfælde virker fire threads godt, men er ikke nødvendigvis det mest optimale.
- Producers benytter "put()" til at smide data ind, som consumers kan benytte ved at bruge "take()" sådan at dataen forsvinder fra producers, og nu benyttes i consumers
- Consumers bruger "take()" til at udtrække data.

### 2.3.2 B, C, D og F (E???)

Henvises til koden

## 2.4 Exercise 4

### 2.4.1 A, B og C

Henvises til koden

### 2.4.2 D

- De bliver ikke printet i samme rækkefølge som de blev sat ind.
- De bliver printet ud i en rækkefølge angivet af hvilke producers der bliver først færdige.
- Man kan benytte en ExecutionService med futures for at printe dem i den rækkefølge som de er tiltænkt.

### 2.4.3 E

Jeg har forsøgt at indikere at der ikke er flere docs til consumeren, men uden held. Den simple løsning vil være at pass et element fra producer til consumer om at der ikke er flere elementer.

## **3 Day 3**

### **3.1 A**

Gennemsnitligt tager det 10 sekunder at køre programmet.

### **3.2 B, C og D**

Henvises til koden.

### **3.3 E**

Tiden er blevet forbedret fra en køretid på 10 sekunder til 2 sekunder. Det er en forbedring på 500%, og da en `ExecutionService` ikke er særlig kompleks at implementere mener jeg at det er det værd.