# Threads 01

|  |  |
| --- | --- |
| **Læringsmål** | Du kan:   1. **1Pf5:** anvende centrale metoder og teknikker til at designe og konstruere programmer som samarbejdende processer/tråde 2. **1Tf1:** anvende centrale metoder og redskaber til synkronisering af processer og tråde |
| **Forventet læringsudbytte [SOLO]** | Programmeringssporet:   1. [Unistrukturel] Du kan nævne enkelte kendetegn omkring tråde og race condition |
| **Din forberedelse** | Programmeringssporet:   1. [OS] 1 (uddrag vedlagt i Planer):    * Afsnit 9.4 - 9.5, Deadlocks (side 197-206) 2. [What is a Thread?](https://www.youtube.com/watch?v=hN2Yrf4tqTY) (video: 9:45) 3. Hent [Threading in C#](http://www.albahari.info/threading/threading.pdf) (PDF)    * Læs “Introduction and Concepts” (side 4-8)    * Læs “Creating and starting Threads” (side 8-14) 4. [Thread class](https://learn.microsoft.com/da-dk/dotnet/api/system.threading.thread?view=net-6.0) 5. [Lock statement](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/lock-statement) 6. [Do not: lock(this)!](https://stackoverflow.com/questions/6815332/what-is-the-difference-between-lockthis-and-lockthislock) 7. [Implementing the Singleton Pattern in C#](https://csharpindepth.com/articles/singleton) |

Formålet med dagens øvelse er at give dig en forståelse for at kunne implementere tråde på forskellige måder samt at kunne afværge *race conditions*.

# Dagens ord:

“It is far easier to design a class to be thread-safe   
than to retrofit it for thread safety later.”  
Brian Goetz

# Øvelse 1: Terminologi

Del teamet op i to mindre grupper (3-4 pers. i hver), og brug **Ordet rundt** til at reflektere over begreberne ”Singleton Pattern”, ”Lazy Loading”, ”Thread”, ”Process”, ”Thread-safe (da: trådsikker)”, ”Lock”, ”Deadlocks”, ”Mutual Exclusion”, ”Circular wait” og ”Hold-and-wait”. Sørg for, at alle får mulighed for at tale.

*Tidsramme: 20 minutter*

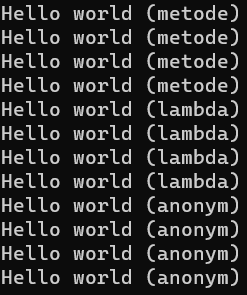
# Øvelse 2: Start af en Thread

Implementer 3 tråde, som hver skriver ”Hello world (metode/lambda/anonym)” på skærmen 4 gange ved at anvende henholdsvis en metode, et lambda-udtryk og en anonym metode som argument til Thread().

Fremgangsmåde:

1. Implementer en metode som anvendes som argument til Thread()
2. Definer et lambda udtryk som argument til Thread()
3. Definer en anonym metode som argument til Thread()

Output skulle gerne se ud som følgende:



# Øvelse 3: Benyt argument, når tråd startes

Ændr din implementering af trådene fra øvelse 2, så du kan overføre en tekststreng som et argument til trådens Start-metode. F.eks.: *t1.Start("Hello world (metode)");*

Fremgangsmåde:

1. Udvid din implementering af metoden, så den kan tage imod et objekt som argument og skrive den ud på skærmen
2. Udvid dit lambda-udtryk, så den også tager imod et objekt som argument og kan skrive den ud på skærmen
3. Udvid din anonyme metode, så den kan skrive en tekststreng ud på skærmen

Bemærk, at Start-metoden kun accepterer et argument af typen "object". Derudover er der ikke nogen forskel rent funktionelt, så du skal stadigvæk få det samme resultat printet ud til skærmen.

Øvelse 4: En betragtning, double-checked locking

Betragt nedenstående metode, der implementer double checked locking:



Besvar følgende spørgsmål i relation til kodeeksemplet:

* Er det nødvendigt at kontrollere for null i både linje 10 og 14?
  + med andre ord: kan en af linjerne undværes?
* Kan der være grunde til at have begge linjer, som de er?

# Øvelse 5: Delt ressourceproblem

Analyser nedenstående program, og forudsig, hvad der sker under udførelsen af programmet (herunder output til skærm). Du må ***ikke*** prøve at køre programmet, førend du er færdig med din forudsigelse.

class Program

{

private char \_sharedChar;

private const int SIMULATE\_WORK = 100;

static void Main(string[] args)

{

Program p = new Program();

p.Run();

}

public void Run()

{

Thread tA = new Thread(WriteA);

Thread tB = new Thread(WriteB);

tA.Name = "Thread A";

tB.Name = "Thread B";

tA.Start();

tB.Start();

tA.Join();

tB.Join();

Console.Write("\nPress a key ....");

Console.ReadKey();

}

private void WriteA()

{

for (int i = 0; i < 10; i++)

{

\_sharedChar = 'A';

Thread.Sleep(SIMULATE\_WORK);

Console.WriteLine($"{Thread.CurrentThread.Name} : {\_sharedChar}");

Thread.Yield();

}

}

private void WriteB()

{

for (int i = 0; i < 10; i++)

{

\_sharedChar = 'B';

Thread.Sleep(SIMULATE\_WORK);

Console.WriteLine($"{Thread.CurrentThread.Name} : {\_sharedChar}");

Thread.Yield();

}

}

}

Når du er klar, så udfør følgende:

* Opret et nyt konsolprojekt
* Kopier nu koden ind i dit konsol-projekt i VS
* Kør programmet

Gav det samme output, som du forudsagde?

# Øvelse 6: Delt ressourceproblem – Løsning

Du har nok bemærket, at programmet i øvelse 5 har et åbenlyst problem vedrørende den delte ressource (\_sharedChar), hvilket resulterer i, at der kan opstå forekomster såsom ”Thread B : A” og ”Thread A : B”.

Løs problemet, så der kun printes ”Thread A : A” og ”Thread B : B” ud til skærmen.

**Note:** Du må *ikke* ændre på den rækkefølge trådene starter i, i Run()-metoden – du *skal* bruge en lås.