



HØGSKOLEN i Buskerud

**Avdeling for ingeniørutdanning
Institutt for teknologi**

Oppgavetittel: Lab

Fag(nr./navn): DOPS2021 - Operativsystemer

Gruppemedlemmer: T. Alexander Lystad

Faglærer: Karoline Moholth

Dato: 15. oktober 2009

Jeg/vi bekrefter at den innleverte besvarelsen helt og fullt er mitt/vårt verk.

.....

.....

.....

.....

Innholdsfortegnelse

Klasseinndeling.....	3
Ringbuffer.....	3
Konklusjon	3
Main.java	4
Daddy.java	5
Son.java	6
Son1.java	7
Son2.java	8
Son3.java	9
RingBuffer.java	10

Klasseinndeling

Main og *RingBuffer* er to opplagte selvstendige klasser å begynne med. Videre bør det være en klasse for hver tråd. Felles funksjonalitet for sønnene samles i en klasse som sønnene arver fra. Klassene blir da: *Main*, *RingBuffer*, *Daddy*, *Son*, *Son1*, *Son2*, *Son3*.

Ringbuffer

En rett frem-implementasjon av en ringbuffer, men hvordan bør underflow og overflow håndteres?

Dersom man er i en situasjon hvor man ønsker live streaming e.l. er det ønskelig at ubrukt data overskrives. Da må man huske på å forskyve lesepekeren i tillegg til skrivepekeren, slik at man fortsatt er FIFO. Alternativt, og mest vanlig, kan man forby skiving til buffer når den er full. Jeg velger å kaste en *BufferOverflowException* i ringbufferen, slik at den som bruker ringbufferen selv kan bestemme hvordan den bør håndteres i sin implementasjon.

Forsøk på å lese data der data ikke finnes, bør også varsles. Jeg kaster en *BufferUnderflowException*. I begge tilfeller hvor jeg bruker ringbufferen, ignorerer jeg disse exceptionene, slik at den forsøker å lese eller skrive helt til den får det til.

Det er umulig å få underflow og overflow samtidig, derfor er ikke dette en fare for vranglås.

push()- og *pop()*-metodene er *synchronized*, slik at ikke to prosesser forsøker å utføre IO-operasjoner samtidig.

Konklusjon

Jeg har lært hva en ringbuffer er, og hvordan den kan og bør funke. Jeg har også lært at selv om en feil ikke oppstår i løpet av 20 minutters kjøring, må man ta høyde for den, fordi den kan (og vil) komme senere.

Main.java

```
package no.hibu.dops2021.lab1.student124867;

public class Main {

    public static void main(String[] args) {
        new Thread(new Daddy()).start();
    }

}
```

Daddy.java

```
package no.hibu.dops2021.lab1.student124867;

public class Daddy implements Runnable {

    private RingBuffer m_ringBuffer;

    Daddy() {
        m_ringBuffer = new RingBuffer(3);
    }

    public void run() {
        new Thread(new Son1(m_ringBuffer)).start();
        new Thread(new Son2(m_ringBuffer)).start();
        new Thread(new Son3(m_ringBuffer)).start();
    }
}
```

Son.java

```
package no.hibu.dops2021.lab1.student124867;

public abstract class Son implements Runnable {

    protected RingBuffer m_ringBuffer;

    Son(RingBuffer rb) {
        m_ringBuffer = rb;
    }

    protected void sleep() {
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            //Ignore
        }
    }
}
```

Son1.java

```
package no.hibu.dops2021.lab1.student124867;

import java.util.Scanner;

public class Son1 extends Son {

    Son1(RingBuffer rb) {
        super(rb);
    }

    public void run() {
        Scanner keyboard = new Scanner(System.in);
        while (true) {
            m_ringBuffer.push(keyboard.next());
        }
    }
}
```

Son2.java

```
package no.hibu.dops2021.lab1.student124867;

import java.nio.BufferOverflowException;

public class Son2 extends Son {

    Son2(RingBuffer rb) {
        super(rb);
    }

    public void run() {
        int _i = 0;
        while (true) {
            try {
                m_ringBuffer.push(Integer.toString(_i++ % 10));
            }
            catch (BufferOverflowException e) {
                _i--; //Decrement _i if we couldn't successfully push
            }
            sleep(); //For human readability
        }
    }
}
```

Son3.java

```
package no.hibu.dops2021.lab1.student124867;

import java.nio.BufferUnderflowException;

public class Son3 extends Son {

    Son3(RingBuffer rb) {
        super(rb);
    }

    public void run() {
        while (true) {
            try {
                System.out.println(m_ringBuffer.pop());
            }
            catch(BufferUnderflowException e) {
                //Ignore
            }
            sleep(); //For human readability
        }
    }
}
```

RingBuffer.java

```
package no.hibu.dops2021.lab1.student124867;

import java.nio.BufferOverflowException;
import java.nio.BufferUnderflowException;

public class RingBuffer {

    private String[] m_aItems; //Array of items - the buffer
    private int m_n = 0;      //Number of items in the buffer
    private int m_iRead = 0;  //Read index
    private int m_iWrite = 0; //Write index
    private int m_iSize;      //Buffer size

    RingBuffer(int size) {
        if (size <= 0) { throw new IllegalArgumentException("Buffer size must exceed 0."); }
        m_iSize = size;
        m_aItems = new String[size];
    }

    synchronized public void push(String item) {
        if (m_n >= m_iSize) { throw new BufferOverflowException(); }
        m_aItems[m_iWrite] = item;
        m_n++;
        m_iWrite = (m_iWrite + 1) % m_iSize;
    }

    synchronized public String pop() throws BufferUnderflowException {
        if (m_n < 1) { throw new BufferUnderflowException(); }
        String item = m_aItems[m_iRead];
        m_n--;
        m_iRead = (m_iRead + 1) % m_iSize;
        return item;
    }
}
```