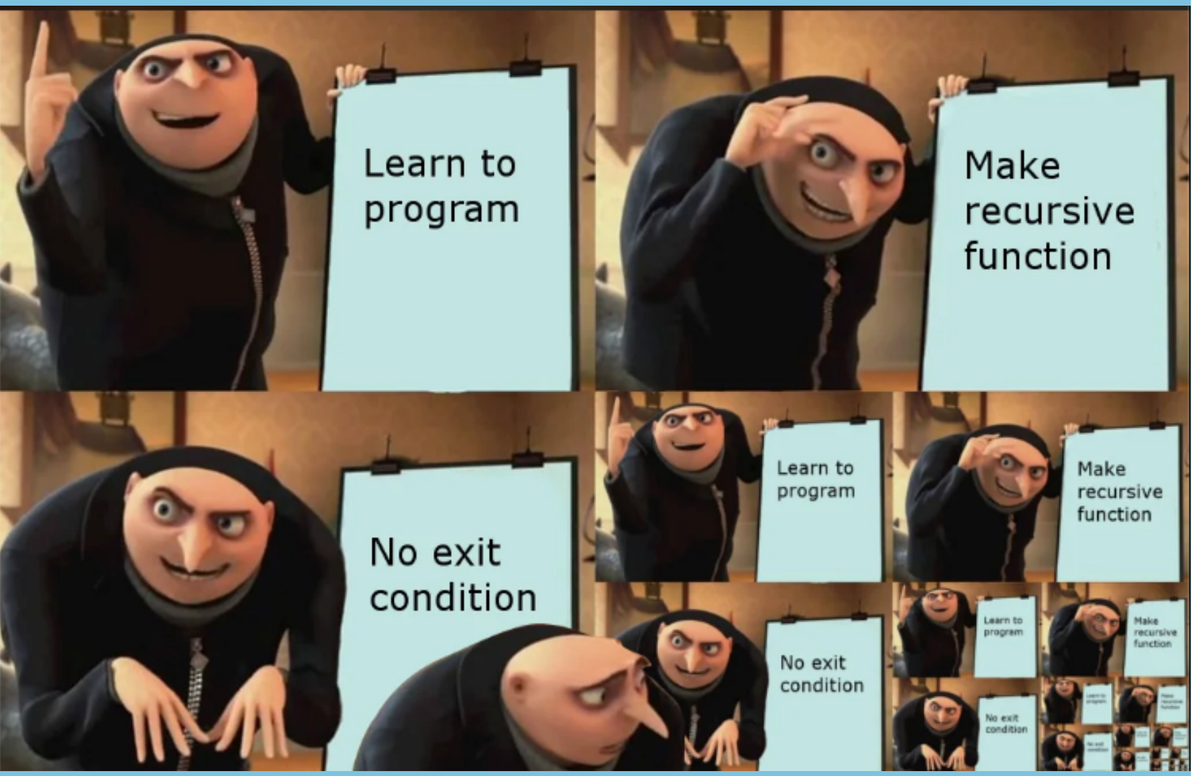
Uke 7 - Rekursjon

Gruppe 11 - jonasbny@ifi.uio.no

Agenda



Comparable

Iterator

Rekursjon

basissteget

rekursive kall

Hva vi skal gjøre neste uke

Comparable

Comparable er et interface

Inneholder en metode: compareTo()

```
public int compareTo(Object obj);
```

forste*.compareTo(*andre) skal returnere:
0 dersom *forste* og *andre* er **like**
positivt tall dersom *forste* er **større** enn *andre*
negativt tall dersom *forste* er **mindre** enn *andre*

*Du bestemmer selv hva det vil si om to objekter er like, mindre enn, eller større enn hverandre

Comparable

Når vi lager egne beholdere kan vi “tvinge” elementene i beholderen til å *måtte* implementere interfacet Comparable

```
public class prioritetsKø<T extends Comparable<T>>{
    T first;

    public void settInn(T newElement){

        // ...

        if(newElement.compareTo(first) > 0){
            // ...
        }

        // ...
    }
}
```

Iterable og Iterator

Vi kan også designe beholderen våres slik at det går an å iterere gjennom den med en **for-each-**løkke

for(E element : beholder) {...}

Iterable: et interface som representerer en samling av elementer som kan bli iterert over

Iterator: et interface som gir metoder for å traversere gjennom elementer av en **iterable**



Rekursjon

En rekursiv metode er en metode som **kaller på seg selv**

Den kalles for mindre versjoner av det opprinnelige problemet

Dette fortsettes helt til en **base case** nås

Løsningen på det opprinnelige problemet bygges ved å kombinere løsningene på de mindre versjonene

```java
public static int Fibonacci(int n){
    // base case: hvis n er mindre enn eller lik 1
    if(n <= 1) return n;

    // rekursive kallet:
    return Fibonacci(n - 1) + Fibonacci(n - 2);

}
```

Rekursjon

En rekursiv metode er en metode som **kaller på seg selv**

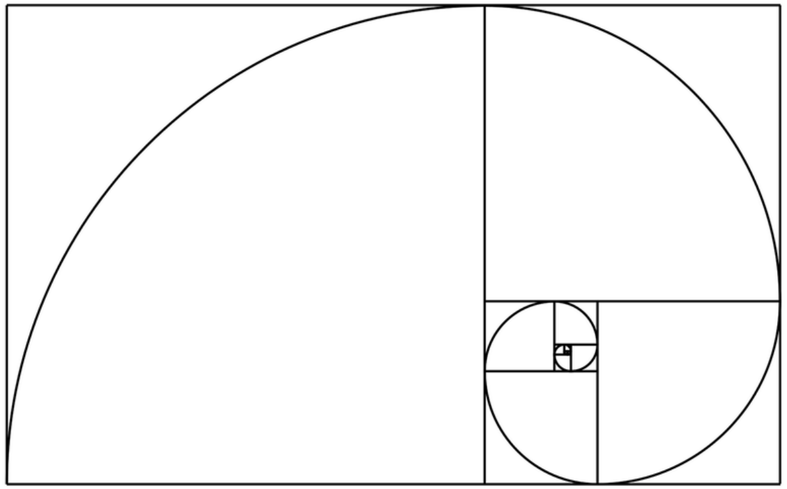
Kan erstatte iterasjon

Mye overlapp om man velger å bruke rekursjon eller iterasjon

vanlig å bruke i sorteringsalgoritmer og navigering av trær

Fordel: lettere å debugge

Ulempe: bruker mer minne



Rekursjonssteget

Når metoden kaller på seg selv, kaller vi det for rekursjonssteget (recursive step)

i rekursjonssteget må noe forandre seg, slik at vi til slutt når basistilfellet (base case)

For hvert steg vil vi at problemet blir litt lettere

En rekursiv metode kan gjøre **mer enn ett** nytt rekursivt kall i et rekursjonssteg

```
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    return x * Exponent(x, exponent - 1);
}
```

Basistilfellet (base case)

Når vi når basistilfellet, slutter vi å kalle på den rekursive metoden flere ganger

rekursjonen stopper, og kallene “nøstes opp” fra siste til første kall

vi kaller derfor basistilfellet for en “stop condition” eller stoppbetingelse

```
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    return x * Exponent(x, exponent - 1);
}
```

Hva skjer egentlig?

walk metoden:

```java
public static void walk(int steps){
    // base case:
    if(steps < 1) return;

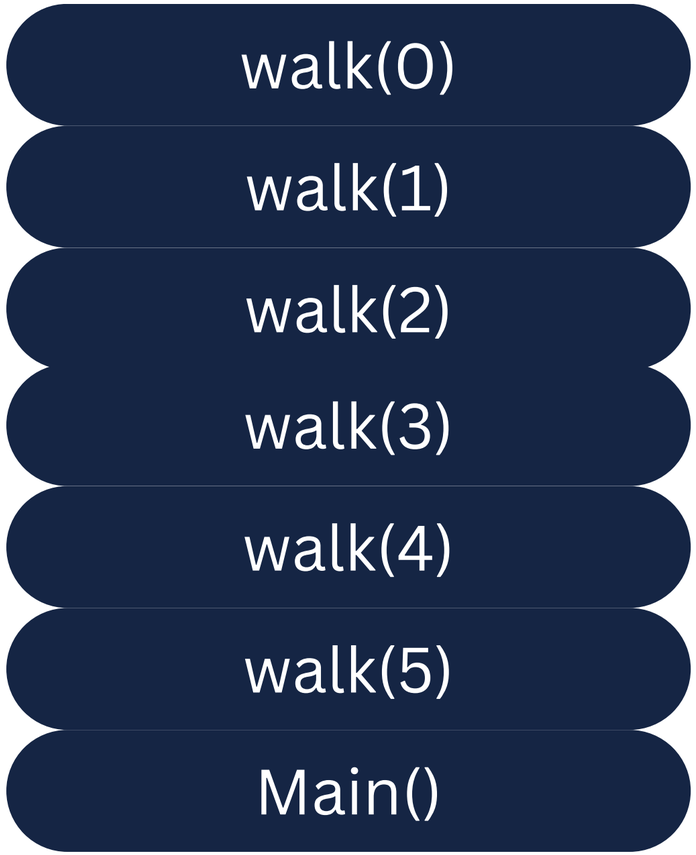
    System.out.println(x:"du gikk et steg");

    // rekursjonssteg:
    walk(steps - 1);
}

Run | Debug
public static void main(String[] args) {
    walk(steps:5);
}
```

“call stack”:

Top -->



Oppgave

Hva blir å skje her?

```java
public static void walk(int steps){
    // base case:
    if(steps < 1) return;

    System.out.println(x:"du gikk et steg");

    // rekursjonssteg:
    walk(steps + 1);
}

Run | Debug
public static void main(String[] args) {
    walk(steps:5);
}
```

Oppgave

```
Exception in thread "main" java.lang.StackOverflowError
        at rekursjon.walk(rekursjon.java:4)
        at rekursjon.walk(rekursjon.java:4)
```

Hva skjer egentlig?

Exponent metoden:

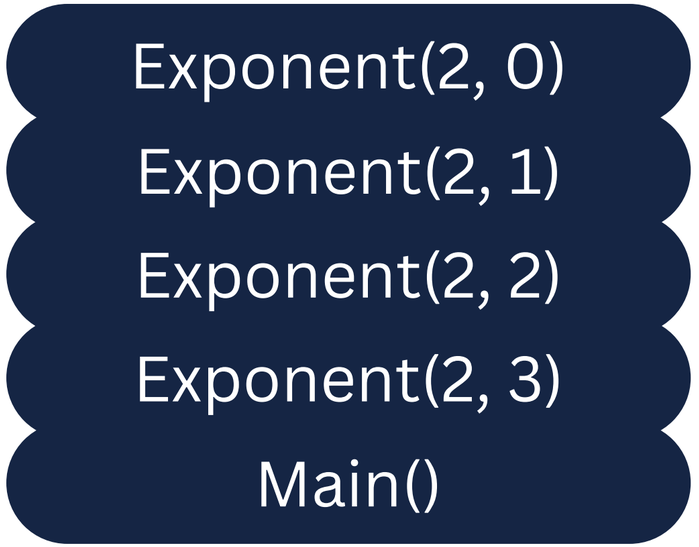
```java
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    return x * Exponent(x, exponent - 1);
}

Run | Debug
public static void main(String[] args) {
    System.out.println(Exponent(x:2, exponent:3));
}
```

"call stack":

Top -->



1

Hva skjer egentlig?

Exponent metoden:

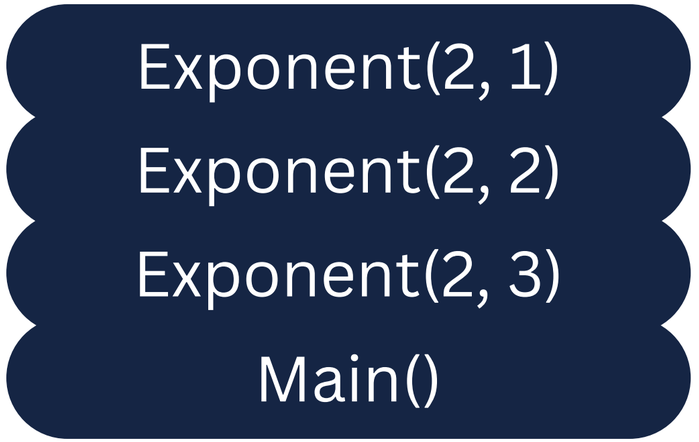
```java
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    return x * Exponent(x, exponent - 1);
}

Run | Debug
public static void main(String[] args) {
    System.out.println(Exponent(x:2, exponent:3));
}
```

"call stack":

Top -->



2

Hva skjer egentlig?

Exponent metoden:

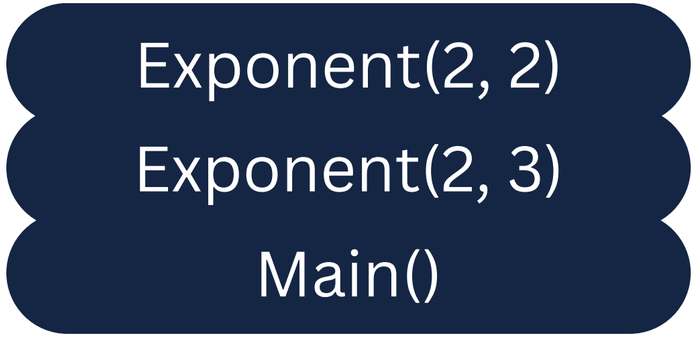
```java
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    return x * Exponent(x, exponent - 1);
}

Run | Debug
public static void main(String[] args) {
    System.out.println(Exponent(x:2, exponent:3));
}
```

"call stack":

Top -->



4

Hva skjer egentlig?

Exponent metoden:

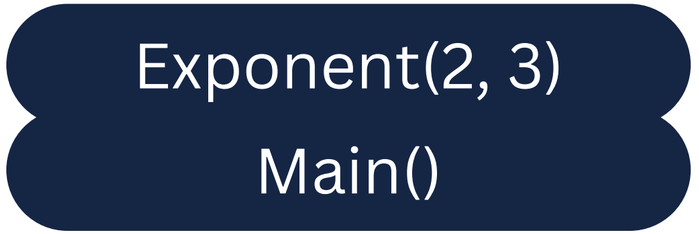
```java
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    return x * Exponent(x, exponent - 1);
}

Run | Debug
public static void main(String[] args) {
    System.out.println(Exponent(x:2, exponent:3));
}
```

“call stack”:

Top -->



8

Hva skjer egentlig?

Exponent metoden:

```java
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    return x * Exponent(x, exponent - 1);
}

Run | Debug
public static void main(String[] args) {
    System.out.println(Exponent(x:2, exponent:3));
}
```

“call stack”:

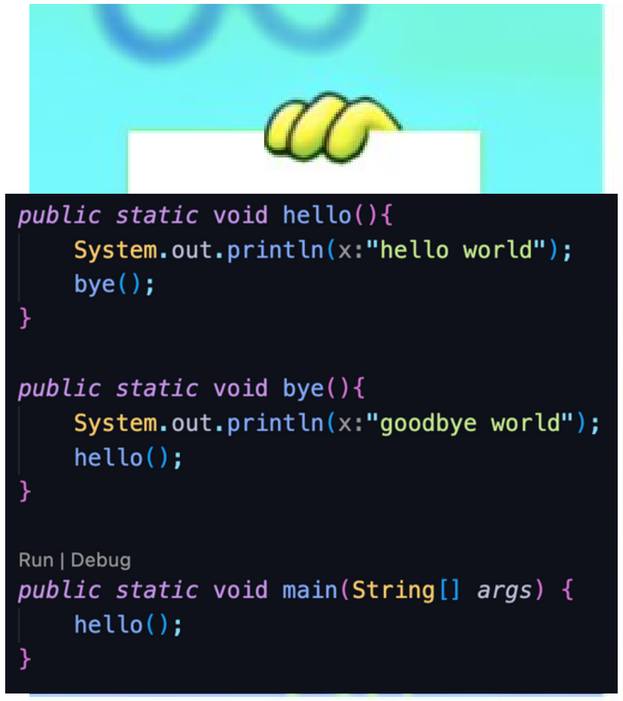
Terminal: 8

Oppsummert

En metode som kaller på seg selv er rekursiv

Rekursjon kan virke forvirrende, prøv å visualiser hva som skjer

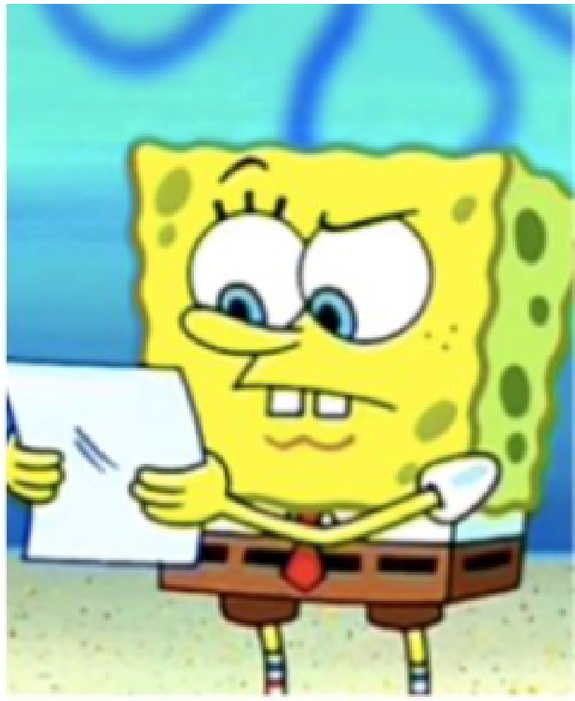
Husk å definere de to komponentene:
- -Basistilfellet (base case)
- -Rekursjonssteget (recursion step)



```
public static void hello(){
    System.out.println(x:"hello world");
    bye();
}

public static void bye(){
    System.out.println(x:"goodbye world");
    hello();
}

Run | Debug
public static void main(String[] args) {
    hello();
}
```







Oppgaver