IN1010

Repetisjon Uke 12

Rekursjon

(Basert på Dag Langmyhrs presentasjon)

11. april, 2025

Hva er rekursjon



Når vi har en bit av kode som kaller seg selv, så kaller vi det for *rekursjon*.

Det kan være en naturlig løsning hvis man…

* vil finne ulike veier til en løsning
* har noe som bør løses hierarkisk
* kan dele noe i mindre deler

Hva er rekursjon



Rekursjon kan også erstattes med løkker, men ofte kan vi få mer oversiktlig og kortere kode ved å bruke rekursjon.

Hva er rekursjon



Det er noen ulemper, slik som bruk av minne og mer tid. Noen ganger kan det være at det ikke er like enkelt å forstå hva som skjer.

Simpel rekursjon

```
class Rekursiv {

    static int rekursiv(int n) {
        if (n < 1) return 0;
        System.out.printf("%d ", n);
        return rekursiv(n – 1);
    }

    public static void main(String[] args) {
        System.out.println(rekursiv(10));
    }

}
```

Skriver ut tallene fra 10 til 0

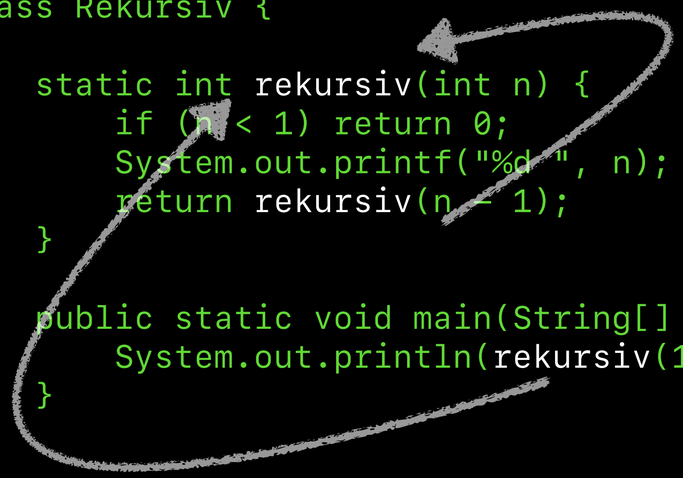
Simpel rekursjon

```
class Rekursiv {

    static int rekursiv(int n) {
        if (n < 1) return 0;
        System.out.printf("%d ", n);
        return rekursiv(n - 1);
    }

    public static void main(String[] args) {
        System.out.println(rekursiv(10));
    }

}
```



Den vanligste måten er en direkte rekursjon, hvor samme metode kaller seg selv

Simpel rekursjon

```
class Rekursiv {

    static int rekursiv(int n) {
        if (n < 1) return 0;
        System.out.printf("%d ", n);
        return rekursiv(n – 1);
    }

    public static void main(String[] args) {
        System.out.println(rekursiv(10));
    }

}
```

Vi begynner med å definere et trivielt eller enkelt tilfelle (base case)

Dette sørger for at vi ikke gjentar i det uendelige

Simpel rekursjon

```
class Rekursiv {

    static int rekursiv(int n) {
        if (n < 1) return 0;
        System.out.printf("%d ", n);
        return rekursiv(n - 1);
    }

    public static void main(String[] args) {
        System.out.println(rekursiv(10));
    }

}
```

Den rekursive delen består av en operasjon og det rekursive kallet

Her er operasjonen simpel: `n - 1`

Simpel rekursjon

```
class Rekursiv {

    static int rekursiv(int n) {
        if (n > 10) return 0;
        System.out.printf("%d ", n);
        return rekursiv(n - 1);
    }

    public static void main(String[] args) {
        System.out.println(rekursiv(10));
    }

}
```

```
Exception in thread "main"
java.lang.StackOverflowError
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
      ⋮     ⋮     ⋮     ⋮     ⋮     ⋮
```

Hvis den trivielle delen mangler, eller ikke er korrekt skrevet, så vil vi få en

stack overflow feil

Simpel rekursjon

```
class Rekursiv {

    static int rekursiv(int n) {
        if (n > 10) return 0;
        System.out.printf("%d ", n);
        return rekursiv(n - 1);
    }

    public static void main(String[] args) {
        System.out.println(rekursiv(10));
    }

}
```

```
Exception in thread "main"
java.lang.StackOverflowError
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
   at Rekursiv.rekursiv(Rekursiv.java:6)
      ⋮     ⋮     ⋮     ⋮     ⋮     ⋮
```

Hvert kall som er rekursivt lagrer lokale variabler og litt om metodekallet

Det betyr at vi kan heller ikke ha en uendelig rekursjon

Fibonacci

```
class Fibonacci {

    static int fibonacci(int n) {
        if (n <= 1) return n;
        return fibonacci(n - 1) + fibonacci(n - 2);
    }

    public static void main(String[] args) {
        try {
            int tall = Integer.parseInt(args[0]);
            System.out.printf("Fibonacci-sekvensen til %d: %d%n", tall, fibonacci(tall));
        } catch (NumberFormatException | ArrayIndexOutOfBoundsException e) {
            System.err.println("Oppgi et heltall");
        }
    }

}
```

Tallsekvensen 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Fibonacci

```
class Fibonacci {

    static int fibonacci(int n) {
        if (n <= 1) return n;
        return fibonacci(n - 1) + fibonacci(n - 2);
    }

    public static void main(String[] args) {
        try {
            int tall = Integer.parseInt(args[0]);
            System.out.printf("Fibonacci-sekvensen til %d: %d%n", tall, fibonacci(tall));
        } catch (NumberFormatException | ArrayIndexOutOfBoundsException e) {
            System.err.println("Oppgi et heltall");
        }
    }

}
```

Vi begynner med det trivielle tilfellet

Fibonacci

```
class Fibonacci {

    static int fibonacci(int n) {
        if (n <= 1) return n;
        return fibonacci(n - 1) + fibonacci(n - 2);
    }

    public static void main(String[] args) {
        try {
            int tall = Integer.parseInt(args[0]);
            System.out.printf("Fibonacci-sekvensen til %d: %d%n", tall, fibonacci(tall));
        } catch (NumberFormatException | ArrayIndexOutOfBoundsException e) {
            System.err.println("Oppgi et heltall");
        }
    }

}
```

Så den rekursive delen: Fibonacci består av summen av de to foregående;

$$F_1 = 1 \,|\, F_n = F_{n-1} + F_{n-2} \,|\, n > 1$$

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

```
n = 4
```

Hvordan fungerer det: Hvis vi skal finne for 4, så er n lik 4 i første kallet

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

```
n = 3
```

```
fibonacci(n = 4)
```

Så kaller vi på metoden på nytt, men med n-1

Metode-kallet blir lagret på "stakken" (illustrert på høyre side)

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 2

fibonacci(n = 3)

fibonacci(n = 4)

Det gjentar vi

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 1

fibonacci(n = 2)

fibonacci(n = 3)

fibonacci(n = 4)

Det gjentar vi…igjen

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 1

fibonacci(n = 1)

fibonacci(n = 2)

fibonacci(n = 3)

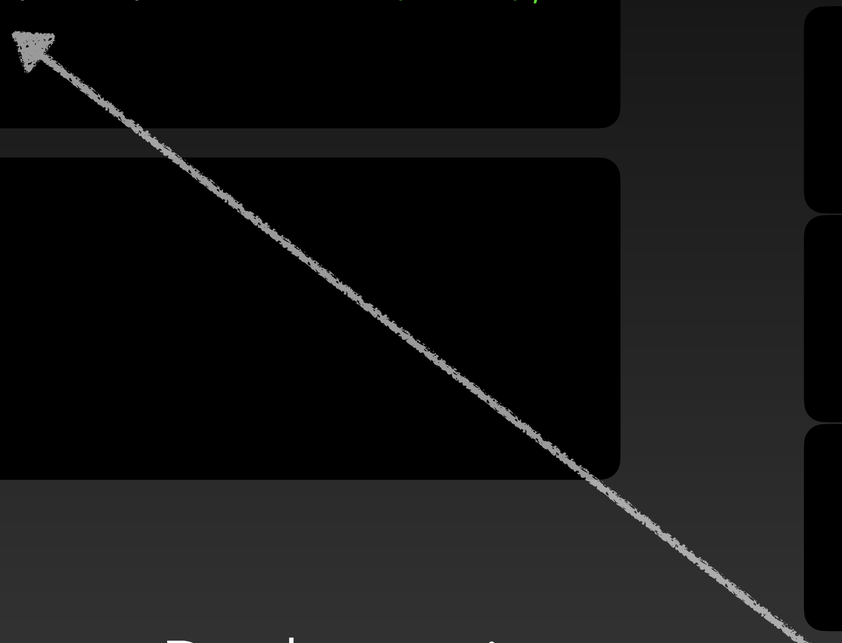
fibonacci(n = 4)

Men nå er n <= 1

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 2



fibonacci(n = 2)

fibonacci(n = 3)

fibonacci(n = 4)

Det betyr vi returnerer 1, uten rekusivt kall

Merk at vi er tilbake til metodekjøret hvor n = 2

Fibonacci

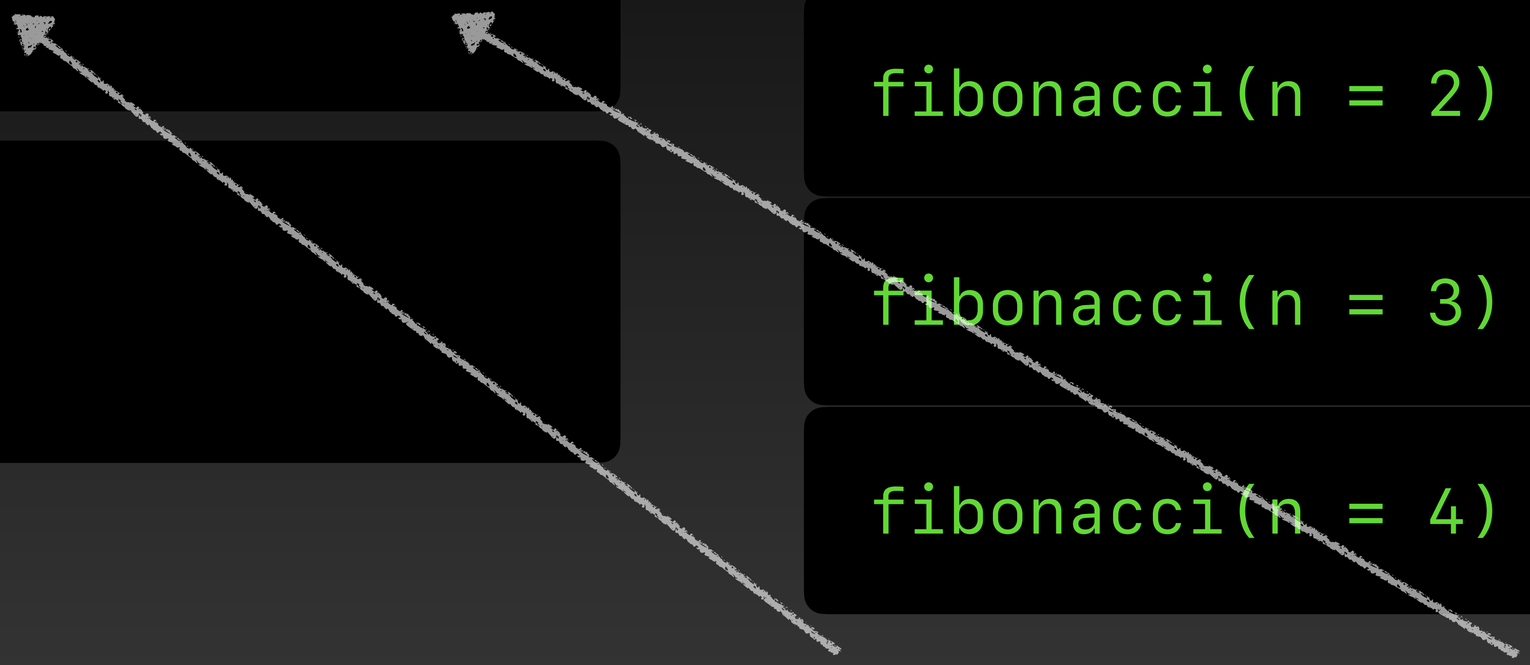
```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 2

fibonacci(n = 2)

fibonacci(n = 3)

fibonacci(n = 4)



Det første uttrykket i return er nå 1, og Java begynner på "høyre" siden

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 0

fibonacci(n = 2)

fibonacci(n = 3)

fibonacci(n = 0)

Den kaller på metoden med n - 2 = 0

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 0

fibonacci(n = 2)

fibonacci(n = 3)

fibonacci(n = 0)

Her returnerer den også 0

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 2

fibonacci(n = 2)

fibonacci(n = 3)

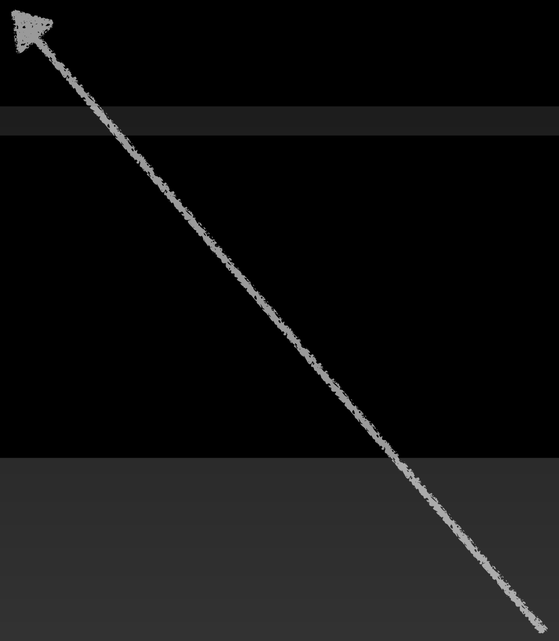
fibonacci(n = 4)

Vi er tilbake til n = 2 kallet, og returnerer 1 + 0 = 1

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 3



```
fibonacci(n = 3)
```

```
fibonacci(n = 4)
```

Nå har vi returnert med 1 til første ledd i returen til n = 3 kallet

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 3

fibonacci(n = 3)

fibonacci(n = 4)

Vi kaller på metoden med n - 2 = 3 - 2 = 1

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 1

fibonacci(n = 3)

fibonacci(n = 1)

Denne gangen vil vi også returnere med 1

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 3

fibonacci(n = 3)

fibonacci(n = 4)

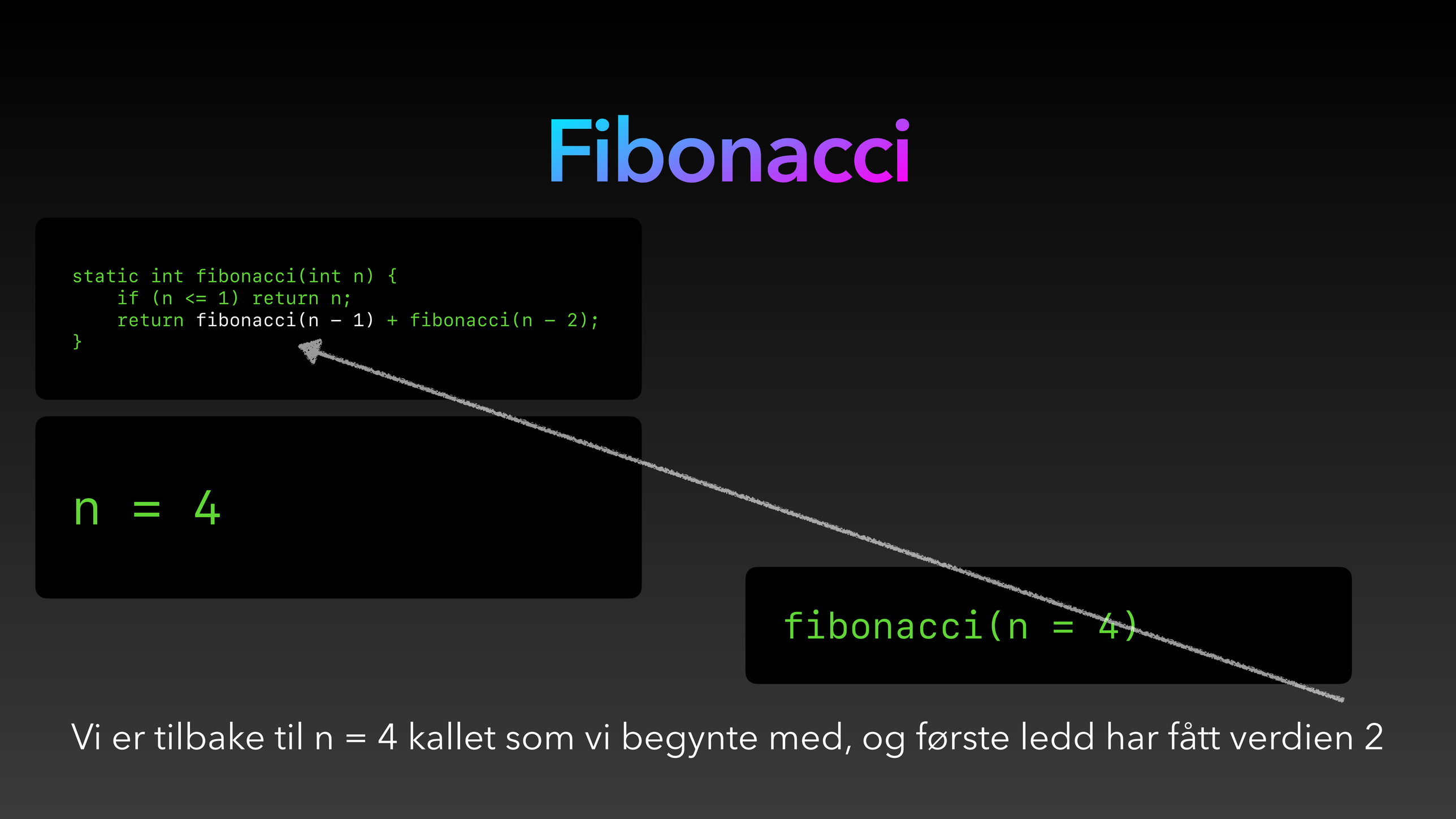
Så nå har n=3-kallet fått begge "sidene", så den kan returnere 1 + 1 = 2

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

```
n = 4
```

```
fibonacci(n = 4)
```



Vi er tilbake til n = 4 kallet som vi begynte med, og første ledd har fått verdien 2

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

```
n = 4
```

```
fibonacci(n = 4)
```

Andre leddet kaller på metoden med n - 2 = 4 - 2 = 2

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 2

```
fibonacci(n = 2)
```

Nå er denne mer enn 1 og vi kaller på første leddet med n - 1 = 2 - 1 = 1

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 1

fibonacci(n = 1)

fibonacci(n = 2)

Denne returnerer 1 siden den er mindre eller lik 1

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 2

```
fibonacci(n = 2)
```

Første ledd er da 1 og kaller på andre ledd med n - 2 = 2 - 2 = 0

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 0

fibonacci(n = 0)

fibonacci(n = 2)

Også er denne mindre eller lik 1, og returnerer n som er 0

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n = 2

```
fibonacci(n = 2)
```

Tilbake ett hakk og vi er i ett n = 2 kall, med 1 i første ledd og 0 i andre

Vi returnerer 1 + 0 = 1

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

```
n = 4
```

```
fibonacci(n = 4)
```

Nå er vi tilbake i n=4 kallet som var det originale, og venstre ledd = 2, og høyre = 1

Vi kan nå returnere begge ledd som gir oss 2 + 1 = 3

Fibonacci

```
static int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

```
return 3;
```

Så da har vi rekursert ferdig og svaret blir 3

Fakultet

```
class Fakultet {

    static long fakultet(int n) {
        if (n <= 1) return 1;
        return n * fakultet(n - 1);
    }

    public static void main(String[] args) {
        try {
            int tall = Integer.parseInt(args[0]);
            System.out.printf("%d! = %d%n", tall, fakultet(tall));
        } catch (NumberFormatException | ArrayIndexOutOfBoundsException e) {
            System.err.println("Oppgi et heltall");
        }
    }
}
```

Fakultet er matematisk operasjon som egner seg for rekursjon,

det er produktet av alle positive heltall mindre eller lik n

Fakultet

```
class Fakultet {

    static long fakultet(int n) {
        if (n <= 1) return 1;
        return n * fakultet(n - 1);
    }

    public static void main(String[] args) {
        try {
            int tall = Integer.parseInt(args[0]);
            System.out.printf("%d! = %d%n", tall, fakultet(tall));
        } catch (NumberFormatException | ArrayIndexOutOfBoundsException e) {
            System.err.println("Oppgi et heltall");
        }
    }
}
```

Vi har en triviell del

Fakultet

```
class Fakultet {

    static long fakultet(int n) {
        if (n <= 1) return 1;
        return n * fakultet(n - 1);
    }

    public static void main(String[] args) {
        try {
            int tall = Integer.parseInt(args[0]);
            System.out.printf("%d! = %d%n", tall, fakultet(tall));
        } catch (NumberFormatException | ArrayIndexOutOfBoundsException e) {
            System.err.println("Oppgi et heltall");
        }
    }
}
```

Og den rekursive delen som er $n! = n \cdot (n-1)!$

Hanois Tårn

Den oppdiktete bakgrunnen for spillet fra 1883 er:

I et tempel i Hanoi står tre stolper, og på den venstre ligger 60 ringer av ulik størrelse, sortert med den minste øverst.

Oppgaven er å flytte ringene til den høyre staken, men

- Ringene må flyttes én og én.
- En ring må aldri legges oppå en mindre ring.
- Den midtre stolpen kan benyttes underveis.

Hanois Tårn

```
class Hanoi {

    static void flytt(int n, char fra, char via, char til) {
        if (n == 1) {
            System.out.println("Flytt " + fra + " til " + til);
        } else {
            flytt(n - 1, fra, til, via);
            flytt(1, fra, via, til);
            flytt(n - 1, via, fra, til);
        }
    }

    public static void main(String[] args) {
        int antall = Integer.parseInt(args[0]);
        flytt(antall, 'A', 'B', 'C');
    }
}
```

Et klassisk eksempel på bruk av rekursjon

Hanois Tårn

```
class Hanoi {

    static void flytt(int n, char fra, char via, char til) {
        if (n == 1) {
            System.out.println("Flytt " + fra + " til " + til);
        } else {
            flytt(n - 1, fra, til, via);
            flytt(1, fra, via, til);
            flytt(n - 1, via, fra, til);
        }
    }

    public static void main(String[] args) {
        int antall = Integer.parseInt(args[0]);
        flytt(antall, 'A', 'B', 'C');
    }
}
```

Vi har det trivielle tilfellet

Hanois Tårn

```
class Hanoi {

    static void flytt(int n, char fra, char via, char til) {
        if (n == 1) {
            System.out.println("Flytt " + fra + " til " + til);
        } else {
            flytt(n - 1, fra, til, via);
            flytt(1, fra, via, til);
            flytt(n - 1, via, fra, til);
        }
    }

    public static void main(String[] args) {
        int antall = Integer.parseInt(args[0]);
        flytt(antall, 'A', 'B', 'C');
    }
}
```

Det rekursive består av flere deler

Hanois Tårn

```
class Hanoi {

    static void flytt(int n, char fra, char via, char til) {
        if (n == 1) {
            System.out.println("Flytt " + fra + " til " + til);
        } else {
            flytt(n - 1, fra, til, via);
            flytt(1, fra, via, til);
            flytt(n - 1, via, fra, til);
        }
    }

    public static void main(String[] args) {
        int antall = Integer.parseInt(args[0]);
        flytt(antall, 'A', 'B', 'C');
    }
}
```

For å kunne flytte den største ringen, må vi først flytte alle de andre ringene til hjelpestolpen

Hanois Tårn

```
class Hanoi {

    static void flytt(int n, char fra, char via, char til) {
        if (n == 1) {
            System.out.println("Flytt " + fra + " til " + til);
        } else {
            flytt(n - 1, fra, til, via);
            flytt(1, fra, via, til);
            flytt(n - 1, via, fra, til);
        }
    }

    public static void main(String[] args) {
        int antall = Integer.parseInt(args[0]);
        flytt(antall, 'A', 'B', 'C');
    }
}
```

Så kan vi flytte den største ringen

Hanois Tårn

```
class Hanoi {

    static void flytt(int n, char fra, char via, char til) {
        if (n == 1) {
            System.out.println("Flytt " + fra + " til " + til);
        } else {
            flytt(n - 1, fra, til, via);
            flytt(1, fra, via, til);
            flytt(n - 1, via, fra, til);
        }
    }

    public static void main(String[] args) {
        int antall = Integer.parseInt(args[0]);
        flytt(antall, 'A', 'B', 'C');
    }
}
```

Etterpå må vi flytte alle ringene fra hjelpestolpen over den største ringen

CM 690
CW 690