Uke 8 - Repetisjon

Gruppe 11 - jonasbny@ifi.uio.no

Agenda



Primitive typer
static
Referanser / pekere
Arv og komposisjon
Lenkeliste
Datastruktur
Klassehierarki
comparable og compareTo
Rekursjon

Primitive typer

I java er “alt” objekter, bortsett fra de primitive typene

A primitive data type specifies the size and type of variable values, and it has no additional methods.

Det finnes åtte primitive typer:
-Byte, short, int, long

-float, double

-boolean, char



Primitive typer

Primitive typer og datastrukturer

“Wrapper klasser” lar deg bruke primitive typer som objekter



static

static betyr at metoden eller variabelen
hører til klassen og ikke et individuelt objekt

Det er derfor tilgjengelig selv før
et objekt er opprettet

int antallHunder



1 1 1

static int antallHunder



1 2 3

Referanser / pekere

En referanse peker til et objekt

Flere variabler kan peke til samme objekt

Et objekt kan peke på et annet objekt

Bytte pekere



```java
public static void main(String[] args) {
    int x = 3;
    int y = 4;
    variabelBytte(x, y);
    System.out.println(x + " " +y);
}

public static void variabelBytte(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
```

```java
public static void main(String[] args) {

    int fire = 3;
    int to = 4;
    int fem = 5;

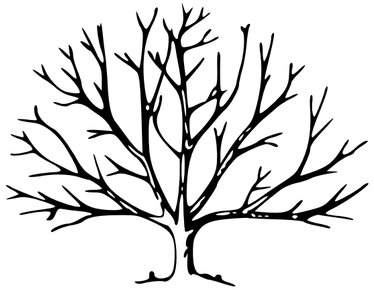
    fire = to;
    to = fem;
    fem = fire;

    // Ka skjer her?!?!?!?
}
```

Arv og komposisjon

Arv:
Noe et objekt **er**

Komposisjon:
Noe et objekt **har**



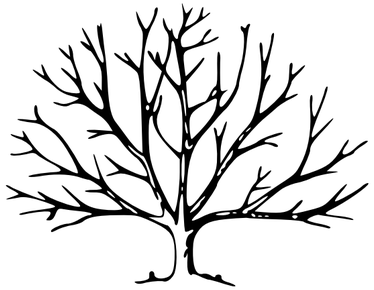
Kan gjøre at vi dupliserer mindre kode

Kan gi oss en mer intuitiv måte å modellere data på

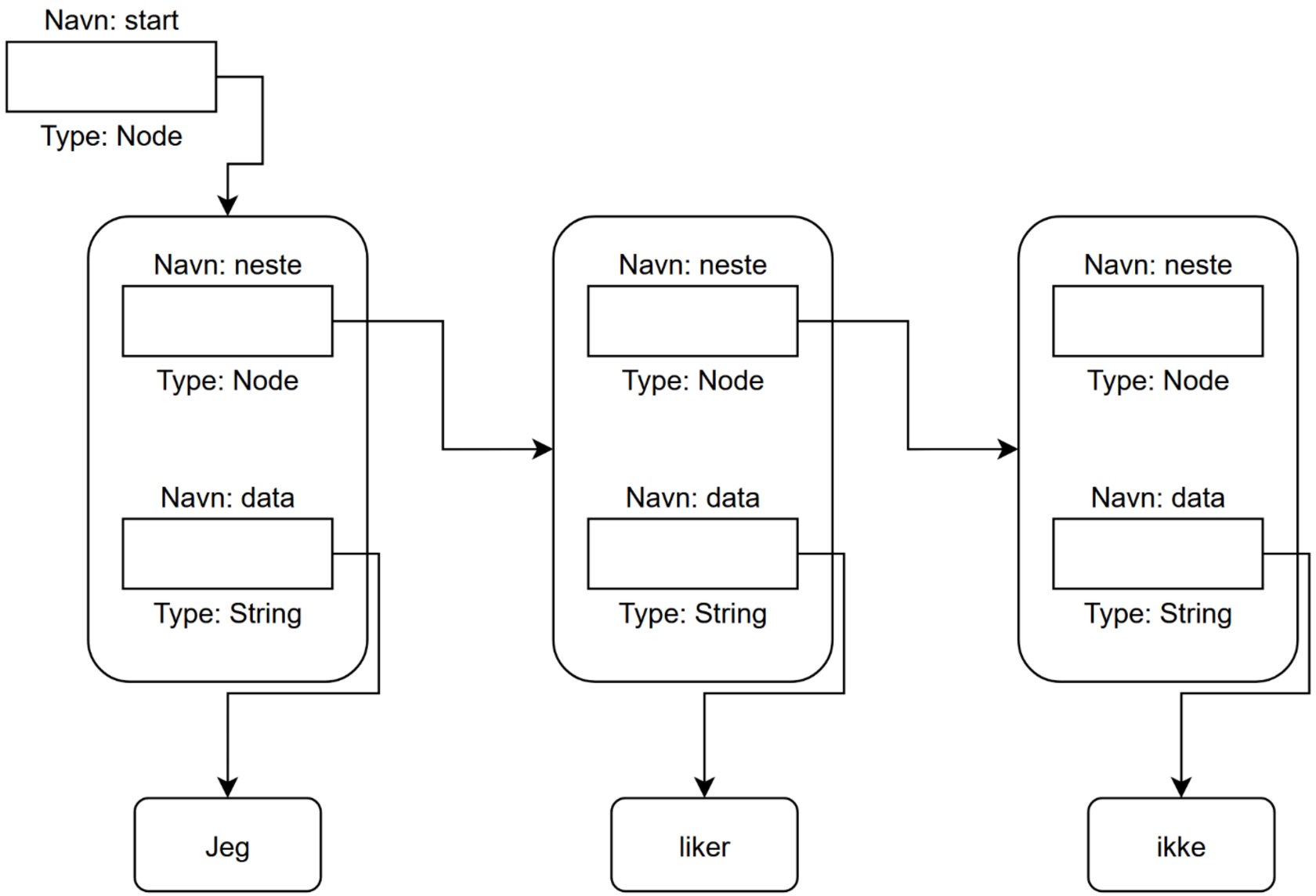
Arv og komposisjon

En taxi **er** en bil

En bil **har** en motor



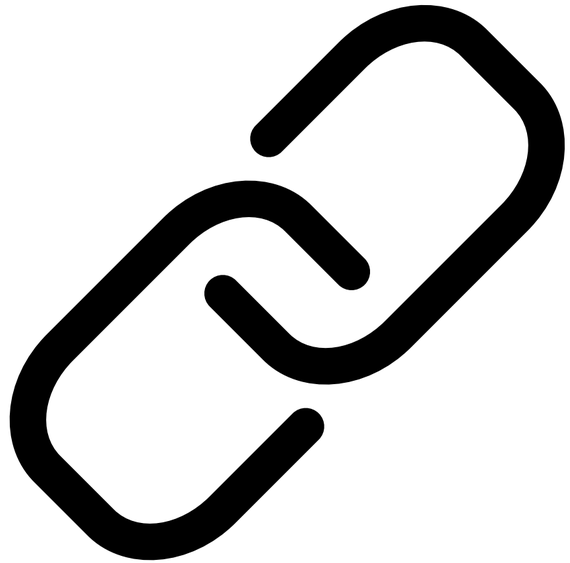
Lenkeliste


Navn: start
Type: Node
Navn: neste
Type: Node
Navn: data
Type: String
Navn: neste
Type: Node
Navn: data
Type: String
Navn: neste
Type: Node
Navn: data
Type: String
Jeg
liker
ikke

Lenkeliste

For å slippe å hekte masse noder sammen i main, lager vi et eget objekt som håndterer nodene

Tilbyr de metodene vi behøver for å hente ut og sette inn
  -og andre ting som size()



Datastrukturtegning

Visualisering av koden som kjøres

lenkeliste eksempel



Klassehierarki

Oversikt over hva som arver fra hva og hvordan interface vi implementerer

beholdere eksempel



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){
            // ...
        }

        // ...
    }
}
```

Comparable

men??
class prioritetsKoe<T extends Comparable<T>>?
Er ikke Comparable<T> et interface?
Betyr det da at T også er nødt til å være et interface?

Nei!

“extends” her er en placeholder for *enten* extends *eller* implements

Java gidder ikke å lage et eget kodeord som kommer i tillegg til disse to, og bruker bare “extends”

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

    public void settInn(T newElement){

        // ...

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

        // ...
    }
}
```

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**

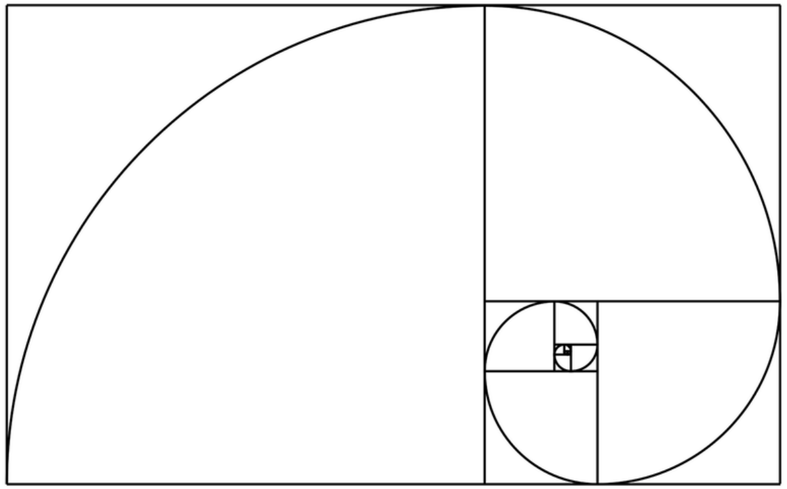
Kan erstatte iterasjon

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



Hva skjer egentlig?

“call stack”:

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));
}
```

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 -->



Hva skjer egentlig?

Exponent metoden:

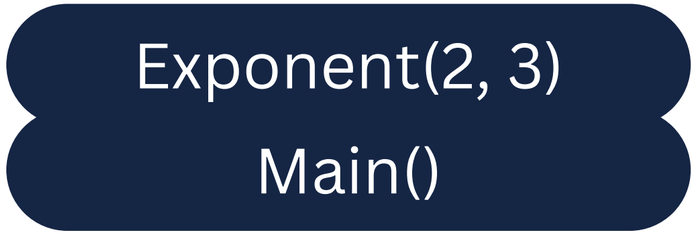
```java
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    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 -->



Hva skjer egentlig?

Exponent metoden:

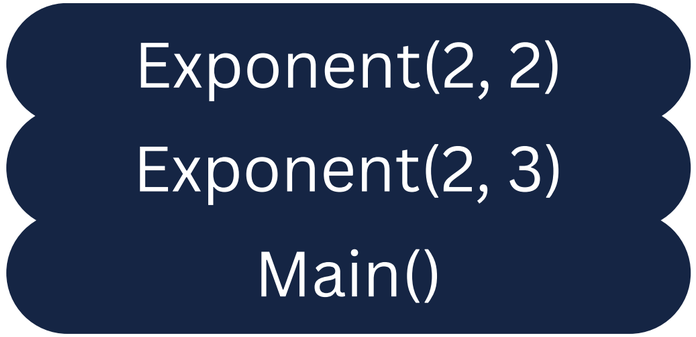
```java
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    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 -->



Hva skjer egentlig?

Exponent metoden:

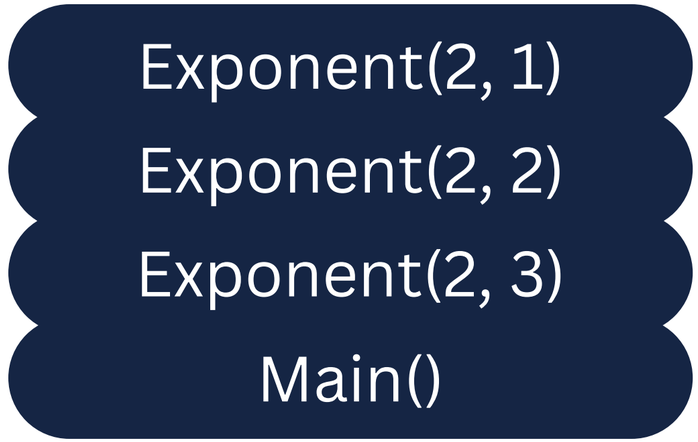
```java
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    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 -->



Hva skjer egentlig?

Exponent metoden:

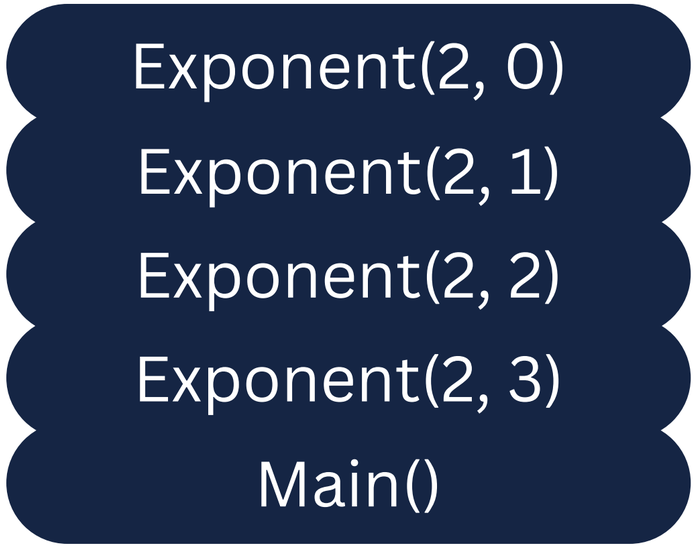
```java
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    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 -->



Hva skjer egentlig?

Exponent metoden:

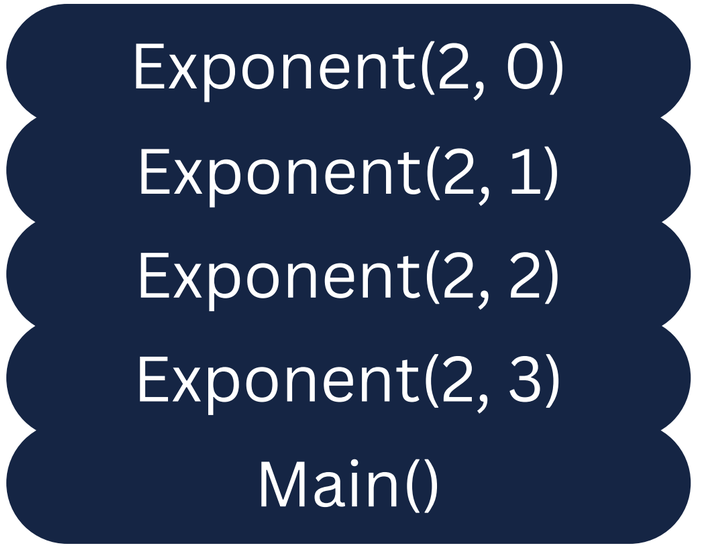
```java
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    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:

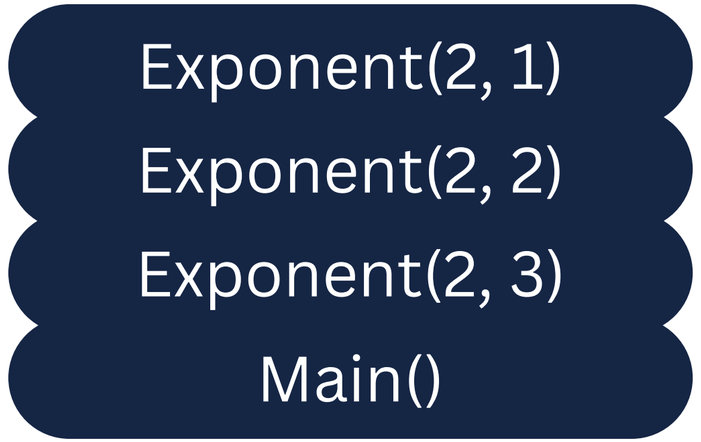
```java
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    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:

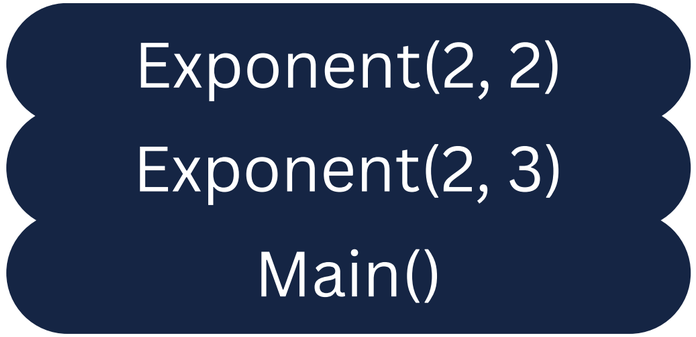
```java
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    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:

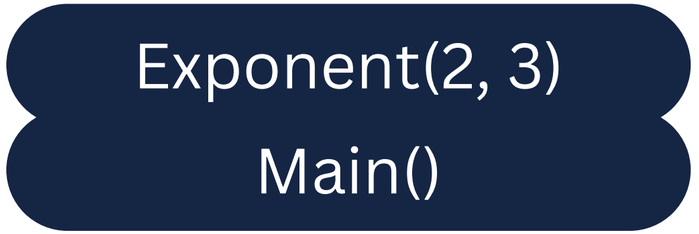
```java
public static int Exponent(int x, int exponent){
    if(exponent < 1) return 1;

    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

Menti!

Spørsmål?

Lykke til!!