Basale programmeringsteknikker: Metoder

- **Teori**
  - **Gymnasie matematik, lineær funktion og areal af rektangel**
    - **Lineære funktioner:** $f(x) = ax + b$

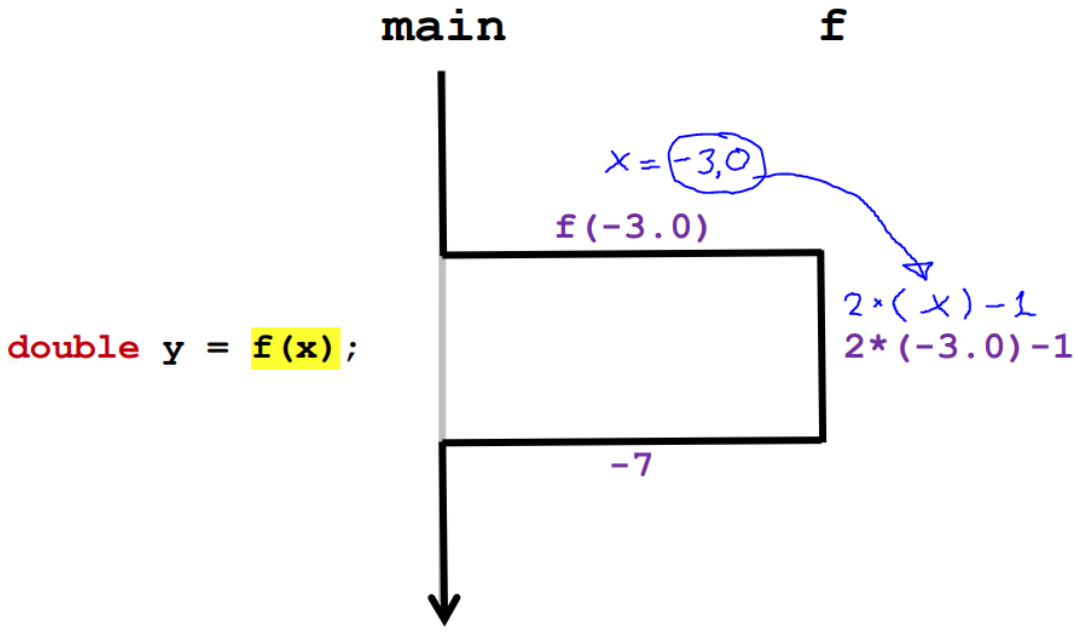
      I java:

```
//f som funktion af x, hvor f(x) så er (2*x-1)
public static double f (double x)
{
return 2*x-1;
}
```

```
class LinearFunction {
  public static double f (double x)
  {
  return 2*x-1;
  }

  public static void main (String[] args)
  {
    for (double x = -3; x <= 3; x+=0.5)
    {
    double y = f(x);
    System.out.println("f(" + x + ") = " + y);
    }
  }
}
```

      Det er foregår inde i funktionen:

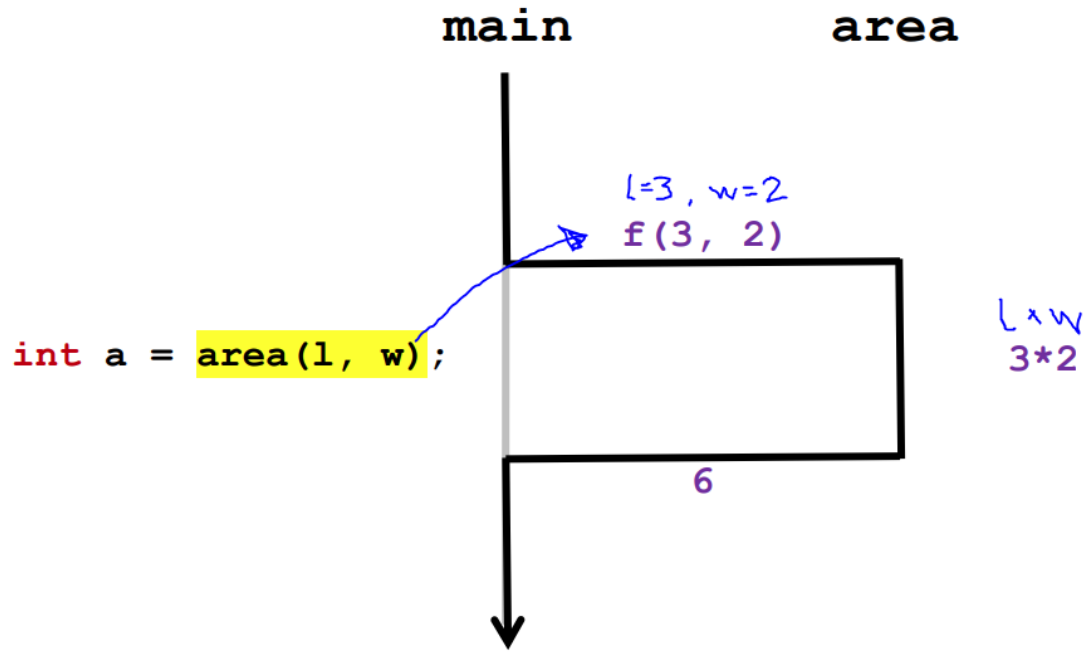


▼ **Areal af rektangel:**

Opsættes som et array af bredde gange længde:

```
public class funktion
{
    public static int area (int l, int w) {
        //Det her er metoden, som det nedenstående henter
        return l*w;
    }
    public static void main (String[] args) {
        for (int w=0 ; w<=4 ; w++) {
            for (int l=0 ; l<=8 ; l++) {
              //fordi area(l,w); er sat til at skulle ganges sammen foroven
                int a = area(l, w);
                System.out.printf(" %2d", a);
            }
            //skal være der, for at det printes pænt
            System.out.println();
        }
    }
}
```

Hvad der foregår inde i metoden:



▼ **Metode bliver brugt til:**

- Genbruge kode
  - Delt logik kan defineres ét sted, så det kan hentes igen i andre metoder, i stedet for at man skal kopiere det hele tiden
  - Ved at *parametisere* denne logik og placere den i en metode, kan de blive genbrugt ved at *kalde* denne metode.

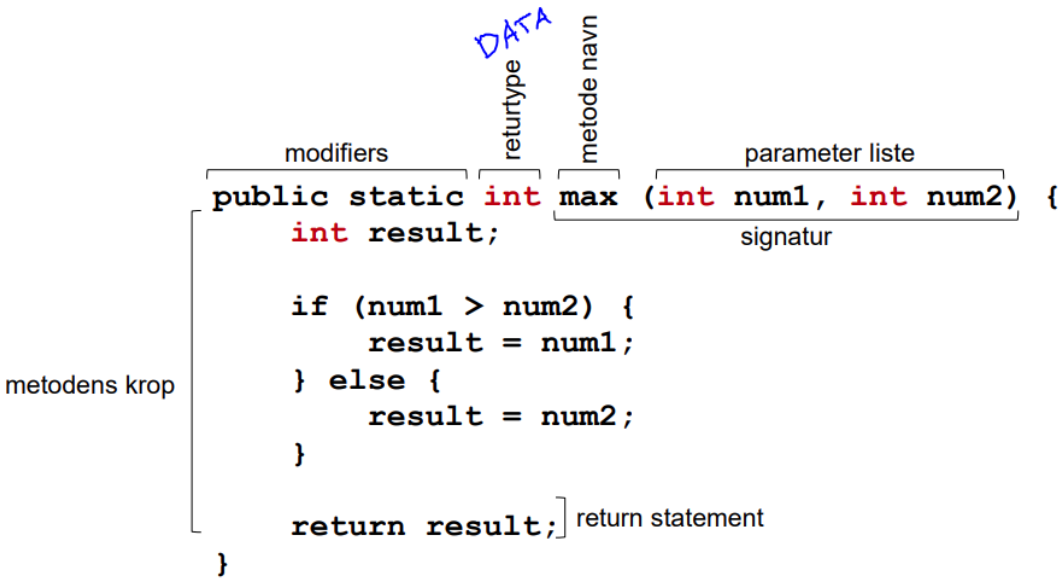
▼ **Metodens anatomi**

Parametre:

Kaldes ofte ***argumenter***

En **metode** kan defineres til at modtage en vilkårligt antal **parametre** og der er ikke begrænsninger på hvilke typer disse kan deklareres som

Parametrene (med deres angivne rækkefølge og type) definerer sammen med metodens navn **metodens signatur:**



Signaturen i dette tilfælde er: **(max, int, int)**

▼ **Metode overloading**

Man kan godt have flere metoder med samme **navn**, man skal bare ændre

- **Typen af parametre** eller
- Antallet af parametre

Eksempel på Metode Overloading:

```
public class funktion
{
    public static int max (int num1, int num2) {
        if (num1 > num2) {
            return num1;
        } else {
            return num2;
        }
    }
    public static double max (double num1, double num2) {
        if (num1 > num2) {
            return num1;
        } else {
            return num2;
        }
    }

    public static void main (String[] args) {
        //declare variabler med værdier, og kør dem i metoderne:
        int t1 = 1;
        int t2 = 0;
        double n1 = 1.0;
        double n2 = 3.0;
        //Java ved selv, at (int) værdien t1 = 1, skal køre i (int) metoden
        //og at (double) værdien n1=0, skal køres i (double) metoden
        System.out.println(max(t1,t2));
        System.out.println(max(n1,n2));
```

```
    }
}
```

▼ **Metode Krop**

En metodes krop indeholder en sekvens af statements

▼ **Return Statement**

Udførelsen af en metode afsluttes med et **return statement**

"Typen" **void** bruges til at indikere, at der *ikke* skal returneres en *værdi*.

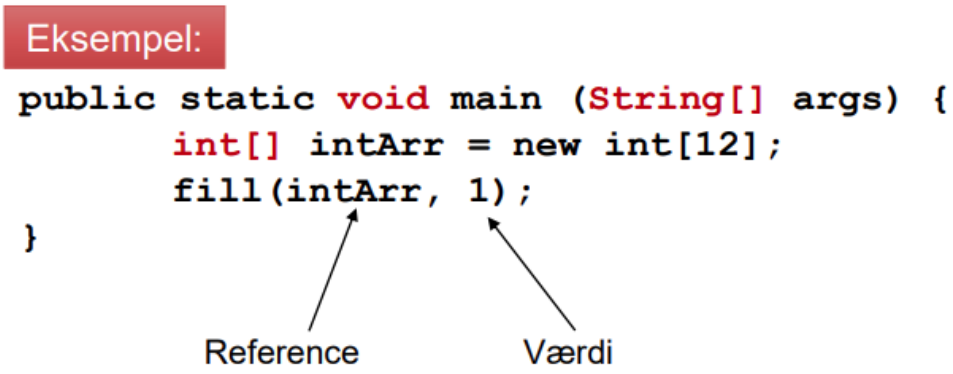
Hvis en metode har en retur type anderledes end void så:
1. skal alle mulige veje igennem metoden ende i en return statement.
2. skal alle return statements have et expression der evaluerer til en værdi af retur typen.

▼ **Overlevering af Værdier**

At overføre værdien fra den kaldende metode til den kaldte kaldes **Value Passing.**

Der gælder følgende regler:

- Ved variable af **primitive typer**, kopieres **værdien**
- Ved variable af **komplekse typer** kopieres **værdien**, men;
  - i dette tilfælde er værdien den adresse i hukommelsen, hvor den komplekse værdi er lagret



▼ **Metodekald som et Expression:**

```
public class funktion {
    public static int max(int[] array) {
        //der laves et loop, hvor værdien max bliver overskrevet, hvis der findes
        //en værdi som er højere end max, ellers returneres max
        int max = -1; //sættes til -1, fordi intet bør være lavere end det
        for (int i = 0; i < array.length; i++) {
            if (array[i] > max) {
                max = array[i];
            }
        }
        return max;
    }
```

```
    public static void main(String[] args) {
        int[] months = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
        int longer = max(months) + 1;
        System.out.println("No month is " + longer + " days long");
    }
}
```

▼ Metodekald som et Statement - void

```
public class funktion {
    public static void prettyPrintArray (int[] array) {
        System.out.print("[");
        for (int i=0 ; i<array.length ; i++) {
            //hvis i==0 printer den "" + array[i],
            // og i alle andre tilfælde hvor i!=0, printer den ","
            System.out.print((i==0 ? "" : ",")+array[i]);
        }
        System.out.println("]");
    }
    public static void main (String[] args) {
        int[] months = {31,28,31,30,31,30,31,31,30,31,30,31};
        prettyPrintArray(months);
    }
}
```

▼ Metodekald med Sideeffekt

```
public class funktion {
    public static int checkedIn = 0;
    public static void checkin () {
        checkedIn++;
    }
    public static void main (String[] args) {
        for (int i=0 ; i<100 ; i++) {
            checkin();
        }
        System.out.println(checkedIn+" people has checked in");
    }
}
```

▼ Metoder der kalder Metoder

Main metoden er speciel på den måde, at den fortæller, hvor udførelsen af programmet starter

Alle metoder er i stand til at kalde andre metoder, herunder dem selv.

Metoder kan til dels betragtes som en blok af parameteriserede statements.

Eksempel:

```
public class funktion {
    public static int max (int[] array) {
        int max = -1;
        for (int i=0 ; i<array.length ; i++) {
            //hvis tallet på plads [i] i array er større end max, så overskriv
            //max med det
            if (array[i]>max) {
                max = array[i];
```

```
            }
        }
        return max;
    }
    public static int maxDiff (int[] array) {
        //array.length-1, fordi hvis array er 4 lang, skal der kun bruges 3 differenser
        int[] diffs = new int[array.length-1];
        for (int i=0 ; i<diffs.length ; i++) {
            //regn differensen mellem array plads [i] og den næste plads: [i+1]
            diffs[i] = array[i+1] - array[i];
        }
        //kald på max-metoden, og returner det største tal, altså den største differense
        return max(diffs);
    }
    public static void main (String[] args) {
        int[] data= {1,3,5,8};
        int maxdiff = maxDiff(data);
        System.out.println("The maximum difference is "+maxdiff);
    }

}
```

▼ **Opgaver**

8.3 Sudoku Prettyprinter

Følgende datastruktur repræsenterer en sudoku plade:

Skriv et program, hvori
1. Ovenstående struktur er defineret.
2. En metode er defineret der som parameter tager en sådan struktur og skriver den ud på skærmen.
3. En main metode kalder denne metode.

```
import java.util.*;

public class string {
    public static void prettyprint (int[][] array) {
        for (int i=0; i< array.length; i++) {
            System.out.print(Arrays.toString(array[i]));
            System.out.println();
        }
    }
    public static void main(String[] args) {
        int[][]puzzle = {
                {7, 3, 6, 4, 5, 2, 9, 8, 1},
                {1, 9, 8, 6, 3, 7, 4, 5, 2},
                {4, 2, 5, 9, 8, 1, 3, 7, 6},
                {3, 6, 4, 5, 2, 8, 1, 9, 7},
                {9, 5, 2, 7, 1, 4, 6, 3, 8},
                {8, 1, 7, 3, 9, 6, 2, 4, 5},
                {2, 8, 9, 1, 7, 3, 5, 6, 4},
                {6, 7, 3, 2, 4, 5, 8, 1, 9},
                {5, 4, 1, 8, 6, 9, 7, 2, 3},
        };
        prettyprint(puzzle);
    }
}
```

8.4 Sum

Skriv en metode, der lægger to heltal sammen. Skriv derudover et program der viser hvordan denne metode skal bruges.

```
public class sum {
    public static int f(int x, int z) {
        return x + z;
    }

    public static void main(String[] args) {
        for (int x = 0; x <= 10; x++) {
            for (int z = 0; z <= 2; z++) {
                int y = f(x, z);
                System.out.println("f(x="+ x +" & z="+ z + ") = "+f(x,z));
            }
        }
    }
}
```

8.8 Fakultet

Skriv et program, hvori
1. En metode udregner fakultet (e.g., fac(4)=4∗3∗2∗1) uden brug af et loop.
2. En main som kalder denne metode og udskriver resultatet.

```
public static int factorial(int n) {
        if (n==0)
            return 1;
        else
            return (n*factorial(n-1));
    }


    public static void main(String[] args) {
        int n = 5;
        System.out.println("Factorial of " + n + " is: " + factorial(n));
    }
```

8.9 Cirkler i Tal

Skriv et program der udregner og udskriver både arealet (π · r2) og omkredsen (2πr) af tre cirkler med radius på hhv. 1, 3 og 5.

```
public class cirkler_i_tal {
    public static double cirkelAreal (double radius){
        return (3.14159*radius*radius);
    }
    public static double cirkelOmkreds (double radius) {
        return 2*3.14159*radius;
    }

    public static void main(String[] args) {
        for(double radius = 1; radius <=5; radius += 2) {
            System.out.println("area with radius: " + radius + " = " + cirkelAreal(radius));
            System.out.println("circumference with radius: " + radius + " = " + cirkelOmkreds(radius));
            System.out.println();
        }
```

```
    }
}
```

8.5 Egen Kvadratrod

Skriv et program, hvori
1. En metode udregner kvadratroden af en double med fx 7 decimale cifre.
• Hint: Prøv jer frem for hvert ciffer, og brug et loop til at iterere over cifrene 0.000000001 til 1000000000.
2. En main som demonstrerer denne metode.

```
public static double root(int number) {
        if (number < 0)
            return -1;
        if (number == 0 || number == 1)
            return number;
        double root = 0.0d;
        double precision = 0.0000001d;
        double square = root;
        while (square < number) {
            root = root + precision;
            square = root * root;
        }
        return root;
    }

    public static void main(String[] args) {
        System.out.println(root(9));
    }
```