

Objekt-Orienteret Programmering

Metoder

Aslak Johansen asjo@mmmi.sdu.dk
Peter Nellesmann pmn@mmmi.sdu.dk

September 18, 2023

Part 0:
Funktioner i Matematik

Funktioner i Matematik ▷ Definition

En udregning der tager noget:

- ▶ ***Input:*** En række af navngivne parametre der tildeles værdier.
- ▶ ***Output:*** En enkelt værdi.

Hver værdi er et tal.

Vi siger at funktionen *mapper* inputtet til outputtet, eller at funktionen repræsenterer en *mapping* fra input til output.

Funktioner i Matematik ▷ Gymnasiematematik ▷ Lineære Funktioner

Lineære funktioner er funktioner der tager formen:

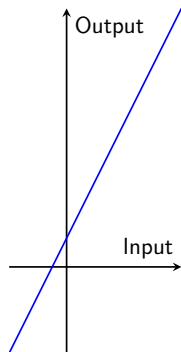
$$f(x) = a \cdot x + b \quad (1)$$

En konkret lineær funktion erstatter a og b i (1) med specifikke værdier:

$$f(x) = 2 \cdot x + 1$$

I Java er en ækvivalent:

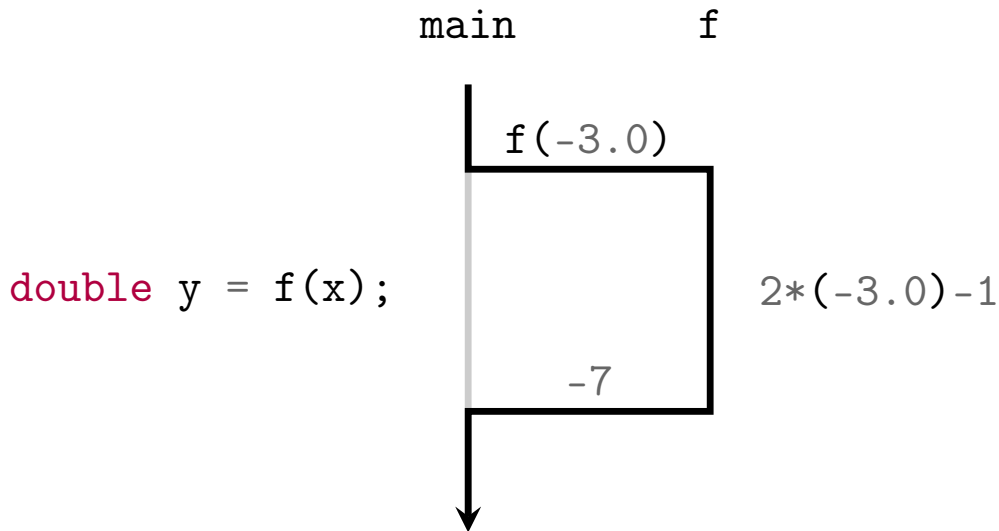
```
public static double f (double x) {  
    return 2*x-1;  
}
```



Funktioner i Matematik ▷ Gymnasiematematik ▷ Lineære Funktioner

```
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);  
        }  
    }  
}
```

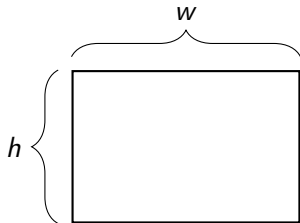
```
f(-3.0) = -7.0  
f(-2.5) = -6.0  
f(-2.0) = -5.0  
f(-1.5) = -4.0  
f(-1.0) = -3.0  
f(-0.5) = -2.0  
f(0.0) = -1.0  
f(0.5) = 0.0  
f(1.0) = 1.0  
f(1.5) = 2.0  
f(2.0) = 3.0  
f(2.5) = 4.0  
f(3.0) = 5.0
```



Funktioner i Matematik ▷ Gymnasiematematik ▷ Areal af Rektangel

Arealet af et rektangel udregnes ud fra dets bredde og højde:

$$area(w, h) = w \cdot h$$



I Java er en ækvivalent:

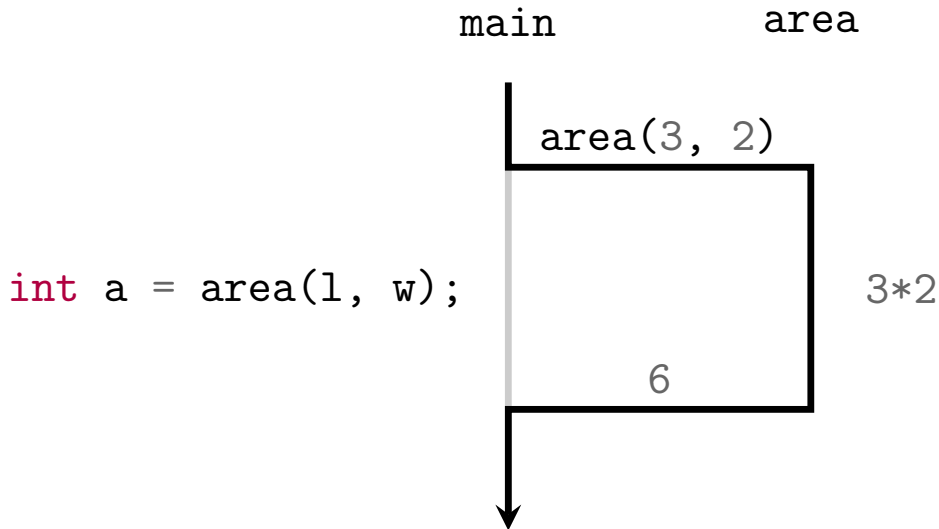
```
public static int area (int w, int h) {  
    return w*h;  
}
```

Funktioner i Matematik ▷ Gymnasiematematik ▷ Areal af Rektangel

```
public class RectangleArea {  
    public static int area (int w, int h) {  
        return w*h;  
    }  
  
    public static void main (String[] args) {  
        for (int h=0 ; h<=4 ; h+=1) {  
            for (int w=0 ; w<=8 ; w+=1) {  
                int a = area(w, h);  
                System.out.printf(" %2d", a);  
            }  
            System.out.println("");  
        }  
    }  
}
```

0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8
0	2	4	6	8	10	12	14	16
0	3	6	9	12	15	18	21	24
0	4	8	12	16	20	24	28	32

Funktioner i Matematik ▷ Gymnasiematematik ▷ Areal af Rektangel



Part 1:

Metoder i Java

Metoder i Java ▷ Introduktion

Metoder er en sproglig konstruktion der muliggør genbrug af kode:

- ▶ Lignende kodelumper med ens ansvar identificeres.
- ▶ En generel *parameteriseret* udgave defineres i form af en metode.
- ▶ Denne udgave udgør en abstraktion.
- ▶ De identificerede kodelumper kan da hver erstattes af et kald til denne metode.
- ▶ **Resultat:** Delt logik kan defineres ét sted i stedet for at være spredt rundt som kopier i ens kode.
 - ▶ Hvorfor er dét at spredte kopier et problem?

Metoder i Java ▷ En Metodes Anatomi

```
public static int max (int num1, int num2) {  
    int result;  
  
    if (num1 > num2) {  
        result = num1;  
    } else {  
        result = num2;  
    }  
  
    return result;  
}
```

The diagram illustrates the components of a Java method definition:

- modifiers**: `public` and `static`
- returtype**: `int`
- metodenavn**: `max`
- parameter liste**: `(int num1, int num2)`
- signatur**: The combination of the return type and parameter list, `int (int num1, int num2)`.
- metodens krop**: The body of the method, enclosed in curly braces, containing the implementation logic.
- return statement**: `return result;`

Metoder i Java ▷ Anatomi ▷ Parametre

Kaldes ofte for *argumenter*.

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

Parametrene (med deres angivne rækkefølge og type) definerer sammen med metodens navn metodens *signatur* (som vi var inde på under metodens anatomi).

Metoder i Java ▷ Anatomi ▷ Parametre

Kombinationen af metodens navn og rækkefølgen af parametertyper kaldes for metodens *signatur*.

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

Signatur: max × (int, int)

Metoder i Java ▷ Metode Overloading

Man kan have flere metoder med samme navn så længe deres signaturer er unikke:

- ▶ **Typer af parametre**
- ▶ **Antal af parametre**

```
// signatur: max * (int, int)  
public static int max (int num1,  
                      int num2) {  
    return (num1 > num2 ? num1 : num2);  
}
```

```
// signatur: max * (double, double)  
public static double max (double num1,  
                        double num2) {  
    return (num1 > num2 ? num1 : num2);  
}
```

```
// signatur: main * (String[])  
public static  
void main (String[] args) {  
    int t1 = 1;  
    int t2 = 0;  
    double n1 = 1.0;  
    double n2 = 3.0;  
  
    System.out.println(max(t1, t2));  
    System.out.println(max(n1, n2));  
}
```

Metoder i Java ▷ Metode Overloading

Man kan have flere metoder med samme navn så længe deres signaturer er unikke:

- ▶ Typer af parametre
- ▶ **Antal af parametre**

```
// signatur: max * (int, int)
public static int max (int num1,
                      int num2) {
    return (num1 > num2 ? num1 : num2);
}

// signatur: max * (int, int, int)
public static int max (int num1,
                      int num2,
                      int num3) {
    return max((num1 > num2 ? num1 : num2),
              num3);
}
```

```
// signatur: main * (String[])
public static
void main (String[] args) {
    int t1 = 1;
    int t2 = 0;
    int t3 = 4;
    double n1 = 1.0;
    double n2 = 3.0;

    System.out.println(max(t1, t2));
    System.out.println(max(t1, t2, t3));
}
```


Metoder i Java ▷ Anatomi ▷ Krop

En metodes krop er en block der indeholder en sekvens af statements.

En metode ...

- ▶ har adgang til de værdier som metodens parametre er bundet til under kaldet.
- ▶ kan producere en værdi som den returnerer.
- ▶ kan ændre på variable udenfor funktionen.

```
public static int[] fill (int[] array, int value) {  
    for (int i=0 ; i<array.length ; i++) {  
        array[i] = value;  
    }  
    return array;  
}
```

Metoder i Java ▷ Anatomi ▷ Krop

En metodes krop er en block der indeholder en sekvens af statements.

En metode ...

- ▶ har adgang til de værdier som metodens parametre er bundet til under kaldet.
- ▶ kan producere en værdi som den returnerer.
- ▶ kan ændre på variable udenfor funktionen.

```
public static int[] fill (int[] array, int value) {  
    for (int i=0 ; i<array.length ; i++) {  
        array[i] = value;  
    }  
}
```

Metoder i Java ▷ Anatomi ▷ Return Statement

Udførelsen af en metode kan afsluttes på et vilkårligt sted med et `return` statement.

Typeangivelsen `void` bruges til at indikere, at der *ikke* skal returneres en værdi.

Hvis en metode har en retur type anderledes end `void` så

- ▶ skal alle mulige veje igennem metoden ende i et `return` statement.
- ▶ skal alle `return` statements have et expression der evaluerer til en værdi der kan implicit castes til retur typen.

Dette er den typiske måde at overføre en værdi fra den kaldte til den kaldende metode.


Part 2: Metode Kald

Metode Kald ▷ Parameterbinding

Vi *kalder* en metode ved at skrive navnet på den efterfulgt af et set parenteser.

- ▶ Tomme parenteser hvis metoden ikke har nogen parameter.
- ▶ Parenteser med kommaseparerede udtryk hvis metoden har parametre.
 - ▶ Rigtigt antal.
 - ▶ Rigtig(e) type(r).

```
public static void main (String[] args) {  
    int[] intArr = new int[12];  
    int[] oneArr = fill(intArr, 1);  
}  
  
public static int[] fill (int[] array, int value) {  
    for (int i=0 ; i<array.length ; i++) {  
        array[i] = value;  
    }  
  
    return array;  
}
```



The diagram consists of two curved arrows. One arrow originates from the `intArr` parameter in the `fill(intArr, 1)` call within the `main` method and points to the `int[] array` parameter in the `fill` method definition. The second arrow originates from the `1` argument in the `fill` call and points to the `int value` parameter in the `fill` definition.

Metode Kald ▷ Parameteroverførsel

Vi kan overføre værdier fra den kaldende metode til den kaldte metode.

- ▶ Kaldes for *value passing*.

Når en værdi overføres fra den kaldende metode (på engelsk "the caller") til den kaldte metode (på engelsk "the callee") 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.**

```
public static void main (String[] args) {  
    int[] intArr = new int[12];  
    int i = -1;  
    fill(intArr, i);  
}
```

reference værdi

Metode Kald ▷ Metodekald som Expression

```
public static int max (int[] array) {  
    int max = -1;  
    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");  
}
```

Output ved kørsel: No month is 32 days long

Metode Kald ▷ Metodekald som Statement

```
public static void prettyPrintArray (int[] array) {  
    System.out.print("[");  
    for (int i=0 ; i<array.length ; i++) {  
        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);  
}
```

Output ved kørsel: [31,28,31,30,31,30,31,31,30,31,30,31]

Metode Kald ▷ Metodekald med Sideeffekt

```
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");
}
```

Output ved kørsel: 100 people has checked in

Part 3:

Modularisering med Metoder

Modularisering med Metoder

Metoder kan gøre kode genbrugelig, uden de restriktioner på flow som kendes fra branches og loops.

Kombinationen af signatur og returværdi repræsenterer en kontrakt mellem de som skriver metoden og de som bruger den.

Denne adskillelse isolerer de to sider fra hinanden (så længe ingen sideeffekter modificerer delt tilstand).

Nogle klasser af fejl begrænses herved til metoden.

Part 4:

Metoder der Kalder Metoder

Modularisering med Metoder ▷ Main-Metoden

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.

```
class CLA {  
    public static void main (String[] args) {  
        for (String arg: args) {  
            System.out.println(arg);  
        }  
    }  
}
```

```
$ javac CLA.java  
$ java CLA Once upon a time ...  
Once  
upon  
a  
time  
...  
$
```

Modularisering med Metoder ▷ Metode til Metode Kald

```
public static void main (String[] args) {  
    int[] data = {1,3,5,8};  
    int maxD = maxDiff(data);  
    System.out.println("The maximum difference is " + maxD);  
}
```

```
public static int maxDiff (int[] array) {  
    int[] diffs = new int[array.length-1];  
    for (int i=0 ; i<diffs.length ; i++) {  
        diffs[i] = array[i+1] - array[i];  
    }  
    return max(diffs);  
}
```

```
public static int max (int[] array) {  
    int max = -1;  
    for (int i=0 ; i<array.length ; i++) {  
        if (array[i]>max) {  
            max = array[i];  
        }  
    }  
    return max;  
}
```

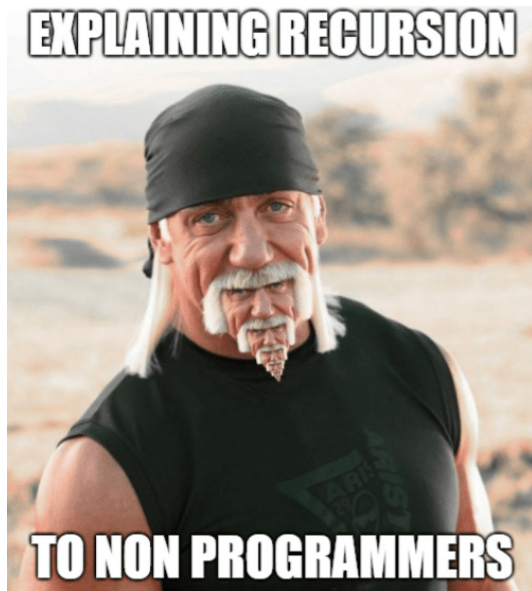
Part 5: Rekursion

Rekursion ▷ Definition

En metode der virker ved at kalde sig selv én eller flere gange kaldes en rekursiv metode.

De fleste rekursive metoder er struktureres omkring en branch, der adskilder ét eller flere basistilfælde fra hvad der kaldes et rekursionstrin.

I det rekursive trin løser metoden en delmængde af problemet ved at kalde sig selv med denne delmængde som parameter, og så integrerer den resultatet af dette med løsningen til resten af problemet for at få den fulde løsning.



Rekursion ▷ Fibonacci Sekvensen ▷ Definition

Fibonacci sekvensen er defineret som:

$$fib(n) = \begin{cases} 0, & \text{for } n = 0 \\ n, & \text{for } n = 1 \\ fib(n-1) + fib(n-2), & \text{for } n > 1 \end{cases}$$

Dette giver: 0 1 1 2 3 5 8 13 21 34 55 ...

Hvordan kan vi implementere en metode der returnerer den n'te Fibonacci tal?

Grænsefladen til denne metode må være:

```
public static int fib (int n)
```

Rekursion ▷ Fibonacci Sekvensen ▷ Løsning

```
public static int fib (int n) {
    if (n==0) {
        return 0;
    } else if (n==1) {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
}

public static void main (String[] args) {
    for (int n=0 ; n<10 ; n++) {
        System.out.println(n+": "+fib(n));
    }
}
```

0: 0
1: 1
2: 1
3: 2
4: 3
5: 5
6: 8
7: 13
8: 21
9: 34

Hvorfor er dette en god løsning? Hvorfor er dette en dårlig løsning?

Questions?

