Metoder

Tro det eller ej men det vi har nu räcker ganska så långt

Men… Att bara använda det vi har för att göra större program gör programmen trassliga, långa och svårtlästa.

Tänk att vi ska göra det här programmet:

- Först ska vi fråga efter ett heltal och kvadrera det.
- Sen ska vi fråga efter ett heltal och dubblera det
- Sen ska vi fråga efter två heltal och addera dem.
- Sen ska vi fråga efter åtta heltal och multiplicera dem.
- All input ska dessutom valideras

Redan här börjar det bli det ganska långt och enformigt att skriva koden.

Vi kommer att tvingas skriva koden för att läsa in och validera heltal om och om och om igen….

Drömmen:

```
public static void main(String args[]){
    int tal = getInteger();
    System.out.println(”I kvadrat: ” + tal * tal);
    int tal2 = getInteger();
    System.out.println(tal2 + ” * 2 ” + ”= ” + tal2 * 2);
    …

     }
```

Där getInteger() varje gång läser in ett heltal och validerar

Lätt att förstå och se vad som händer.

Drömmen kan bli verklig med metoder!

En metod i Java har den här formen:

```
accessModifier returTyp metodNamn(dataTyp variabelNamn){
        //kod som tillhör metoden
}
```

Ex.

```
public static int getInteger (Scanner in){
    System.out.println("ange ett heltal");
    while(!in.hasNextInt()){
        in.next();
    }
    int tal = in.nextInt();
    return tal;
}
```

```
public double getMinValue(double x, double y){
    if(x < y)
        return x;
    else return y;
}
```

Ett annat exempel

```java
public static int addIntegers(int a, int b){
    int c = a + b;
    return c;
}
```

Vad händer här?

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

    int firstValue = 3;
    int secondValue = 4;

    int sum = addIntegers(firstValue, secondValue);

    System.out.println("Summan av 3 och 4 är " + sum);
}
```

Vad gör metoden:

```
public static boolean mystery(String input){
    if (input.length() >= 4){
        return true;
    } else {
        return false;
    }
}
```

Vad gör metoden?

```
public static String mystery (int a, String c ){
    String x = c.substring(0,a);
    return x;
}
public static String mystery(int a, int b, String c){
    String x = c.substring(a,b);
    return x;

}
```

Overload!

När och varför ska man använda metoder?

Återanvändning av kod: Genom att placera återkommande uppgifter i metoder kan du undvika att duplicera kod. Detta gör koden mer lättläst och underlättar underhåll.

Modularitet och struktur: Att bryta ner en stor uppgift i mindre delar med hjälp av metoder gör det enklare att förstå och underhålla koden. Varje metod kan fokusera på en specifik uppgift.

Abstraktion: Metoder möjliggör abstraktion genom att låta dig använda en metodanrop för att utföra komplexa uppgifter som användaren inte behöver förstå i detalj.

Läsbarhet och förståelse: Välskrivna metoder med tydliga namn gör det enklare att läsa och förstå koden. De ger även en hög nivå av dokumentation.

Felsökning och felhantering: Genom att isolera vissa funktioner i metoder kan du enklare isolera och felsöka problem i koden. Det gör också att felhantering blir mer strukturerad och lättare att hantera.

Keep it short!

Gör hellre många korta koncisa och lättlästa metoder än långa och snåriga!

Många funktioner är (oftast) bättre än få!

Det är fullt möjligt att anropa funktioner från funktioner – gör det!

Det går till och med att låta en funktion åberopa sig själv (rekursion).

```java
public static boolean isTheFactorialDividableby2(){
 int value = 1;
    Scanner in = new Scanner(System.in);
    int input1 = in.nextInt();
    int input2 = in.nextInt();
    int input3 = in.nextInt();
    int input4 = in.nextInt();
    int sum = input1+input2 +input3 + input4;

    for (int i = 0;i<sum; i++){
        value *= i;
    }

    if (value%2 == 0){
        return true;
    }else {
        return false;
    }
}
```

```java
public static boolean isTheFactorialDividableby2(){
    int sum = readAndAdd();
    int value = countFactorial(sum);
    boolean answer = isEven(value);
    return answer;

}
```

Vilken av dem tror ni är lättast att läsa, förstå och debugga?

Scope

Nu när vi har tittat lite på metoder så passar det att prata lite om Scope (sv: Omfång)

Med Scope så menar man det område inom en kod där en viss variabel eller funktion är tillgänglig och kan användas.

Det finns olika nivåer av Scope:

Global scope: Variabler som deklareras här är tillgängliga över hela programmet, oavsett var de kallas. Globala variabler deklareras vanligtvis utanför någon specifik funktion eller klass.

Class scope: I Java kan variabler deklareras på klassnivå. Dessa kallas medlemmar och är synliga för alla metoder och block inom klassen. Man kan även göra dem synliga utanför klassen med nyckelordet public.

Local scope: Variabler som deklareras inuti en specifik metod eller block är **bara** tillgängliga inom den metoden eller blocket. De är inte synliga utanför detta scope.

Parameter scope: Parametrar som skickas till en metod har ett eget omfång inuti den funktionen. De beter sig som lokala variabler inuti funktionen.

Package scope: I java kan man strukturera flera klasser i samma ’paket’. En klassvariabel är synlig för alla klasser i samma paket om man inte tilldelar den något annat.

Obs!: I Java kan man inte deklarera variabler globalt. Utan i just java så kan man istället deklarera klassvariabler som **public static** om man vill göra dem globala. Mer om det när vi kommer till klasser och objekt.