
# TDDC77 - Objektorienterad programmering

# Laboration 4

Institutionen för datavetenskap
Linköpings universitet

Den här laborationen består av flera deluppgifter. Flera av deluppgifterna ska redovisas så spara koden från en del innan ni börjar ändra/skriva över den i nästa del.

## MultiplicationTable

I den här deluppgiften ska ni deklarera en klass som heter `MultiplicationTable`. Klassen har en konstruktor som tar emot antalet rader och antalet kolumner som argument. Det ska finnas en metod som heter `print()` som skriver ut följande tabell:

```
  * |  0  1  2  3  4  5  6  7  8
----+--------------------------
  0 |  0  0  0  0  0  0  0  0  0
  1 |  0  1  2  3  4  5  6  7  8
  2 |  0  2  4  6  8 10 12 14 16
  3 |  0  3  6  9 12 15 18 21 24
  4 |  0  4  8 12 16 20 24 28 32
  5 |  0  5 10 15 20 25 30 35 40
  6 |  0  6 12 18 24 30 36 42 48
  7 |  0  7 14 21 28 35 42 49 56
  8 |  0  8 16 24 32 40 48 56 64
  9 |  0  9 18 27 36 45 54 63 72
```

Alla kolumner i tabellen ska vara av samma bredd. Bredden ska vara större än bredden på det elementet som tar största platsen i tabellen, oavsett räknesätt (nästa delmoment).

> Note: Se gärna följande tutorial om utskrift formatering:
> https://dzone.com/articles/java-string-format-examples

## Arv

Antag att vi vill lägga till tabeller för fler räknesätt t.ex. addition. Man skulle då kunna tänka sig att vi har en abstract superklass `ArithmeticTable` som definierar en abstrakt metod `double evaluate(int, int)` som utför en aritmetisk heltalsoperation på sina två argument och returnerar resultatet.

Inför den nya superklassen och visa hur den ska användas genom att deklarera subtyperna `MultiplicationTable`, `AdditionTable`, `SubtractionTable` och `DivisionTable`. Hur kan vi lösa problemet med utskrift av rätt operatortecken och rätt utskriftsbredd på ett smidigt sätt?

## Komposition/Inkapsling

Istället för arv kan vi lösa föregående deluppgiften med inkapsling. Istället för att `ArithmeticTable` har en abstrakt metod så skickar vi med ett objekt som är ansvarigt för att beräkna cellvärdena. Inför först följande interface:

```
public interface Operation {
   public char symbol();
   public int width(int rows, int cols);
   public double evaluate(int a, int b);
}
```

Nu kan vi byta konstruktoren till `ArithmeticTable` till:

```
public ArithmeticTable(Operation op, int r, int c);
```

> Note: ArithmeticTable är inte abstract längre.

Istället anropar metoden `evaluate` som ligger i `ArithmeticTable` den metoden `evaluate` på den inkapslade operationen. Instansiera ett antal objekt av typen `Operation`: `addition`, `substraction`, `multiplication`, `division`.

> Note: Arv kan användas för att skapa ovannämnda Operationerna eller man kan instansiera interface:et Operation fyra gånger genom att använda *anonymous implementer classes*.

```
//AdditionOperation ärver från interfacet Operation.
AdditionOperation addition = new AdditionOperation(/* kan finnas
argument om det behövs*/);
ArithmeticTable tab = new ArithmeticTable(addition, 9, 8);
tab.evaluate(7, 9); //addition
```

## Enumeration

Skapa en uppräkning `EnumOperation` som implementerar `Operation` och definierar värdena: `ADDITION`, `SUBTRACTION`, `MULTIPLICATION`, `DIVISION`. Utgå från följande mall:

```
public enum EnumOperation implements Operation {
     // Er kod ska in här
     private final char symbol;

     private EnumOperation(char symbol) {
         this.symbol = symbol;
     }
     public char symbol() {
         return symbol;
     }
}
```

Note: Varje värde ska implementera/override:a evaluate() på eget sätt.