Opgave

# MyTime


I denne opgave træner vi grundlæggende OOP, herunder klasse og attributter, dataindkapsling, konstruktør samt invarians for klassen.

**Opgaven**

Du skal skrive klassen MyTime, som kan registrere et tidspunkt. Klassen kan også bruges i programmer, hvor du skal registrere længden af en tidsperiode, altså forskellen mellem to tidspunkter.

**Del 1: Timer og minutter**

I del 1 er kravene blot:

- Et tidspunkt skal bestå af timer og minutter, fx 14:57.
- Tidspunktet skal ligge mellem kl. 0:00 og 23:59 (klassens invarians).
- Der skal være én konstruktør. Den skal modtage timer og minutter.
- Der skal være en timeAsString(), som returnerer en tekststreng på formatet TT:MM; altså timetal og minuttal – begge med to cifre – adskilt af et kolon. Eksempler: 14:57 eller 09:23 (ikke 9:23).
- Brug *ikke* toString() og udskriv *ikke* noget fra denne klasse. Andre klasser skal kalde timeAsString() og selv udskrive tidspunktet.

**Del 2: År, måned og dato samt ugedag**

Vi udvider tidspunktet til også at omfatte år, måned og dato:

- Tidspunktet skal nu bestå af år, måned, dato og de allerede oprettede timer og minutter.

- Skriv metoden isLeapYear(), som fortæller om året er et skudår.
  Den korte version af skudår er, at årstal, som er delelige med 4, er skudår.
  Den lange – og korrekte – version fremgår af https://da.wikipedia.org/wiki/Skud%C3%A5r (vi bruger den gregorianske kalender):

  > I den julianske kalender er et år skudår, hvis årstallet er deleligt med 4. Dette er i den gregorianske udvidet således at dette ikke gælder for årstal der er delelige med 100, bortset fra dem, der er delelige med 400 som alligevel er skudår. År 1900 var således ikke et skudår, men år 2000 var. Et år i den gregorianske kalender varer således, i gennemsnit over en periode på 400 år, $365 + \frac{1}{4} - \frac{1}{100} + \frac{1}{400} = 365{,}2425$ dage.

- Omskriv konstruktøren, så den nu modtager alle data. Husk fortsat at sikre klassens invarians.

- Skriv metoden dayOfWeek(), som returnerer en enum med værdien MONDAY, TUESDAY, etc. for den pågældende dato.
  Søg på internettet og se om der er en smart måde at gøre det på.

- Skriv metoden isWorkDay(), som fortæller om dagen er en arbejdsdag (mandag-fredag).

**Del 3: Tidsgymnastik (frivilligt, men stjernegod træning)**

- Skriv nogle metoder, som kan ændre tidspunktet, idet du fortsat sikrer klassens invarians:
    - AddMinutes(int), som går et antal minutter frem i tiden.
      Man må gerne kalde metoden med fx værdien 200, som så bare skal tolkes som 3 timer og 20 minutter.
      Hvis metoden kaldes med et negativt tal, skal der bare regnes bagud i tid.
    - SubtractMinutes(int), som går et antal minutter tilbage i tiden.
    - Tilsvarende kan du tilføje addHours(int), addDays(int), addMonths(int) og addYears(int).
    - Og så de tilsvarende subtract-metoder ligesom for subtractMinutes).

**Del 4: Tidsperiode**

Opret en ny klasse, som indeholder starttidspunkt og sluttidspunkt for en tidsperiode.

Opret metoderne:

- En konstruktør, modtager de to tidspunkter. Det tidligste tidspunkt skal altid være starttidspunktet – også hvis det omvendte gør sig gældende når konstruktøren kaldes.
- Gettere til de to tidspunkter.
- Frivilligt: Settere til de to tidspunkter, så længe du husker invariansen jf. første punkt.
- duration(), som returnerer forskellen mellem de to tidspunkter. Datatypen er MyTime.
  Eksempel: Hvis de to tidspunkter er 2020-10-01-10:00 og 2020-10-02-11:30, er forskellen 0000:00:01:01:30 (nemlig 1 døgn, 1 time og 30 minutter).

**Del 5: 12/24-timers-indstilling (frivilligt)**

Nu skal MyTime have en indstilling til om den fungerer som 12-timers eller 24-timers ur. Det har indflydelse på den tekststreng, som timeAsString() returnerer.

**Del 6: Sekunder og milisekunder (frivilligt)**

MyTime skal nu være endnu mere præcis.

- Tilføj sekunder og metoderne addSeconds(int) og subtractSeconds(int).
- Gør det samme med milisekunder