

# VOP F18 Re-Eksamen 5. september 2018


---

Dette opgavesæt indeholder 3 programmeringsopgaver, som vægtes med i alt 70 point. De sidste 30 point er optjent med de tællende aktiviteter, der er udført i løbet af semesteret.

Opgave 1 tæller op til 20 point og løses udelukkende i javaFx.

Opgave 2 tæller op til 30 point og indeholder integration mellem en klasse med File I/O og javaFx brugerfladen.

Opgave 3 tæller op til 20 point og løses uafhængig af brugerfladen.

**Vigtigt:** Der kan være problemer med at eksekvere klasser med en `main()`-metode under et JavaFx projekt. Brug  , **Shift-F11** eller højre klik på projektet og vælg **Clean and Build**, inden **run** kaldes direkte på en Java `main()`-klasse.

**Hint:** Giv dig tid til at gennemlæse hele opgavesættet, inden du går i gang med løsningerne. Du bør dog starte med at opsætte projektet, som beskrevet i Opgave 1a.

**Aflevering:** Ved afslutningen af eksamen skal løsningen afleveres på BlackBoard:

- I NetBeans markeres projektet, som indeholder løsningen
- Vælg menuen *File -> Export Projekt -> To ZIP...*
- På den fremkomne pop-up dialog, vælges et passende sted at gemme projektet i feltet *Build ZIP:*
- Sørg for at filen får det rigtige navn. Fx "abcd17 VOP ReEksamenF18.zip"
- Upload filen

God fornøjelse.

## Opg. 1 Oprettelse af Java FX-projekt med animeret tekst 20 point

### Opg. 1a

### Oprettelse af projekt

(5 point)

**Formål:** At oprette eksamensprojektet som en JavaFX applikation, navngivet korrekt og forberedt til de øvrige eksamensopgaver.

1. Opret et projekt i NetBeans af typen *JavaFX FXML Application*:
  - a. *Project Name* skal begynde med dit SDU-brugernavn, fx "abcde17 VOP ReEksamenF18".
  - b. Marker checkboksen *Create Application Class*.
2. Start *SceneBuilderen* ved at dobbeltklikke på den dannede *FXML*-fil og slet den *Button* og *Label* som er dannet til *Hello World* eksemplet.
3. Det dannede *Anchorpane* kan med fordel gøres større, fx ved at sætte *Pref Width = 500* og *Pref Height = 325* under *Layout*, eller ved at trække i et af hjørnerne med musen.
4. Sæt et *TabPane* på brugerfladen og tilpas dets størrelse med *Fit to Parent*.
5. Pak den udleverede zip-fil ud og kopier mappen *population* til projektets *src*-mappe.
6. Kopier *Danske Byer.txt* til projekt-mappen (udenfor *src*-mappen).

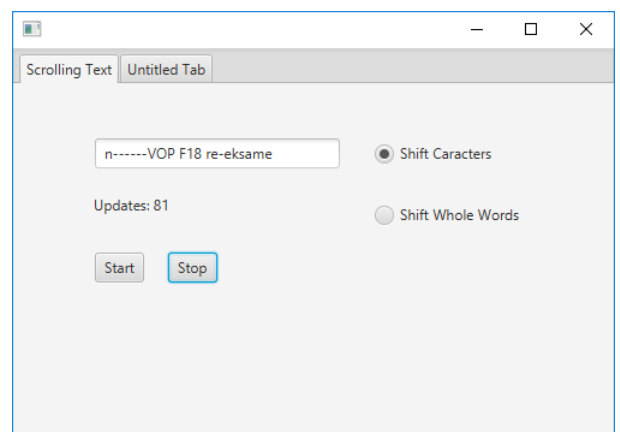
### Opg. 1b Brugerflade til glidende tekst

(15 point)

Navngiv den første Tab "Scrolling Text" eller lignende.

Brugerfladen skal indeholde:

- 1 *TextField* til indskrivning af tekst ("VOP F18 re-eksamen" i eksemplet)
- 1 *Label* til notering af hvor mange opdateringer en tråd har lavet
- 2 *Buttons* til Start og Stop
- 2 *RadioButtons* til valg af metode



Implementer to indre klasser i Controller klasse:

- *CharacterScroller* implements *Runnable*: Teksten fra *TextField* skal i et loop venstre-rotere ét tegn ad gangen, således at første tegn flyttes til sidst i teksten
- *WordScroller* implements *Runnable*: Teksten skal splittes til et array, indeholdende tekstens hele ord (split på mellemrums-tegnet). Genskriv teksten i Tekstfeltet, startende med 2. ord i arrayet og sluttende med det første ord. Husk mellemrum mellem ordene, når teksten skrives.

For begge gælder at roteringerne skal foretages i et loop, at der ventes 200 ms mellem hvert gennemløb og at antallet af gennemløb registreres i *Updates*-labelen.

Afhængig af hvilken af de to *RadioButtons*, der er valgt, skal klik på Start medføre at der oprettes en *Thread* med et objekt af en af de to *Runnable* klasser.

Det skal sikres at tråden standser, hvis programmet lukkes mens den kører.

Ved klik på Stop, skal tråden standses vha. et *Interrupt*.

## Opg. 2 Indbyggere i danske byer

30 point

I denne opgave skal der udvikles et program, som kan analysere befolkningsudviklingen fra 2012 til 2018 i de 32 største danske byer.

Udleverede filer: DanishTown.java (kode skelet)  
TownStatistics.java (kode skelet)  
Danske Byer.txt

Danske Byer.txt er en `"/"-separeret tekstfil`. Den indeholder befolkningstallene fra 2012 og 2018 i danske byer med mere end 20.000 indbyggere. Formatet af hver linje er:

Placering/bynavn/befolkning2012/befolkning2018

Eksempelvis viser øverste linje oplysningerne om hovedstadsområdet:

1/Hovedstadsområdet/1213822/1308893

dvs. Hovedstadsområdet har haft flest indbyggere i 2018. I 2012 var der 1213822 indbyggere og i 2018 1308893 indbyggere. I denne opgave ignorerer vi placeringerne, dvs. den første oplysninger i hver linje.

### a. Implementering af DanishTown

15 point

DanishTown.java skal implementeres så:

- Interfacet `Comparable<E>` implementeres
- Der skal være 3 felter:
  - Byens navn (`String`)
  - Befolkningstal i 2012 (`int`)
  - Befolkningstal i 2018 (`int`)
- De 3 felter skal tildeles værdier fra parametre til constructoren
- Derudover skal der være 3 metoder:
  - `private int getDiff()`, som returnerer befolkningsudviklingen fra 2012 til 2018
  - `public String toString()`. Skal vise byens navn og udviklingen i befolkningstal, fx: "Odense dif: 9414" og "Herning dif: -2316". Den returnerede streng skal starte eller slutte med et linjeskifte ("`\n`").
  - `public int compareTo(DanishTown o)`, fra `Comparable`, skal implementeres så instanser af `DanishTown` kan sorteres efter **stigende** befolkningsudvikling. Hvis to byer har samme befolkningsudvikling, skal de sorteres på bynavn.

### b. Implementering af TownStatistics

10 point

TownStatistics.java skal implementeres så:

- variablen `private List<DanishTown> townList` initialiseres i constructoren til enten `LinkedList` eller `ArrayList`. Vælg den implementation som mest effektivt kan sorteres.
- `public void readFile(String fileName)` skal implementeres så:
  - Hvis `townList` allerede indeholder data, skal den tømmes.
  - Filen læses én linje ad gangen.
  - Af oplysningerne om *bynavn*, *befolkning2012* og *befolkning2018* dannes en instans af `DanishTown`, som tilføjes `townList`.
  - `Exceptions` skal fanges inde i metoden og filen lukkes korrekt efter indlæsning.

- `public void sort()` skal sortere listen ved brug af `compareTo()`-metoden i `DanishTown`.  
*Hint: Benyt en af standard sorteringsalgoritmerne i `java.util.Collections`.*

Herunder vises dele af et muligt resultat efter eksekvering af `main()`-metoden i `TownStatistics`:

<p>Unsorted TownStatistics:</p> <p>Hovedstadsområdet dif: 95071, Aarhus dif: 20864, Odense dif: 9412, Aalborg dif: 9339, Esbjerg dif: 819, .... O. S. V ...</p>	<p>Sorted TownStatistics:</p> <p>Herning dif: -2316, Frederikshavn dif: 128, Skive dif: 247, Svendborg dif: 427, Sønderborg dif: 481, .... O. S. V ...</p>
---	--

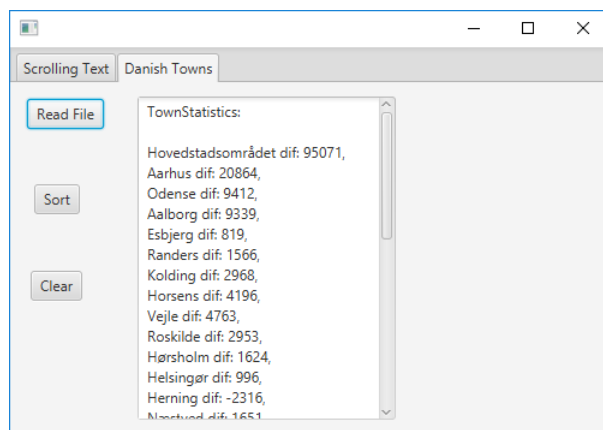
### c. Brugerflade til befolkningsstatistik

5 point

Der skal bruges 3 Buttons og et TextArea. Alle 4 komponenter skal være synlige i Controlleren.

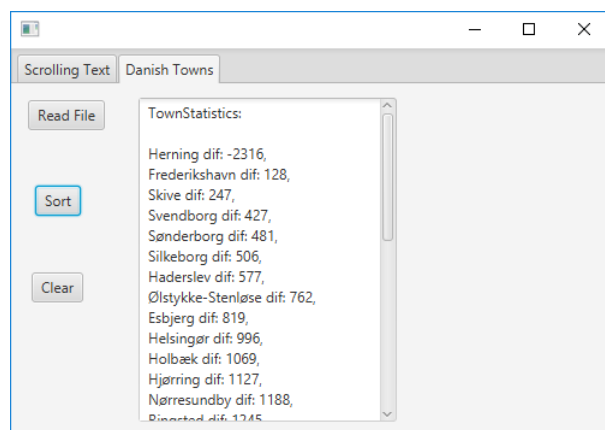
En instance af `TownStatistic` skal erklæres i `FXMLDocumentController.java` og initialiseres i `initialize()`-metoden.

Ved klik på "Read File", skal `Befolkning.txt` indlæses og vises i TekstArealet:



Klik på "Clear" skal tømme TekstArealet.

Ved klik på "Sort", skal `sort()`-metoden i `TownStatistic` kaldes og resultatet tilføjes til TekstArealet:



Hvis ikke der er klikket på "Clear" siden filen blev læst ind, skal det være muligt at scrolle gennem både det usorterede og det sorterede resultat.

### Opg 3: Kast med to terninger

20 point

I denne opgave skal der implementeres et program, som kan simulere kast med to terninger

#### a) Terning og raflebæger med 2 terninger

7 point

Opret en package `dice_statistics` til disse opgaver.

Implementer en klasse `Die.java`, som repræsenterer én terning. Klassen skal indeholde en metode med signaturen `public int roll()`, som returnerer et tilfældigt genereret heltal mellem 1 og 6, begge inclusive.

Implementer en test-klasse

`RollingDice.java`, i hvis `main()`-metode der oprettes to instancer af `Die`, kaldet `d1` og `d2`.

I en løkke skal der udføres 10 kast med de to terninger. For hver gang de to terninger kastes, skal resultaterne af de to kast samt deres sum udskrives, som det ses af denne udskrift:

```
run:
d1: 1 d2: 4 sum: 5
d1: 6 d2: 3 sum: 9
d1: 3 d2: 2 sum: 5
d1: 2 d2: 5 sum: 7
d1: 4 d2: 4 sum: 8
d1: 3 d2: 6 sum: 9
d1: 2 d2: 4 sum: 6
d1: 1 d2: 3 sum: 4
d1: 6 d2: 5 sum: 11
d1: 5 d2: 3 sum: 8
BUILD SUCCESSFUL (total time: 0 seconds)
```

#### b) Opsamling af terningkast i array

7 point

**NB:** Da vi nu skal kaste mange gange med terningerne bør linjen som udskriver de enkelte kast i opgave a udkommenteres, fx `// System.out.println(...)`.

Ved kast af to terninger er der 11 mulige udfald, nemlig tallene fra 2 til 12. Erklær et `int`-array med længden 11 og tilføj kode til løkken fra opgave a) til opsamling af disse udfald. Dvs. hvis `sum = 2` lægges der 1 til `array[0]`, hvis `sum = 3` lægges der 1 til `array[1]`, og hvis `sum = 12` lægges der 1 til `array[10]`. Efter løkken skal indholdet af arrayet udskrives.

Her er hvordan resultatet kan se ud, hvis der kastes 1000 gange:

Det ses heraf at resultaterne har været `28*2, 59*3, 84*4, ..., 26*12`

```
run:
[28, 59, 84, 120, 138, 158, 154, 98, 81, 54, 26]
BUILD SUCCESSFUL (total time: 0 seconds)
```

#### c) %-fordeling

6 point

Den procent-mæssige fordeling af de 11 mulige udfald fra opgave b) skal nu udregnes og udskrives. Her vises resultatet efter 100000 kast med de to terninger

```
run:
[2770, 5554, 8322, 11132, 13928, 16719,
13829, 11106, 8335, 5578, 2727]
Sum 2 in 2,77 %
Sum 3 in 5,55 %
Sum 4 in 8,32 %
Sum 5 in 11,13 %
Sum 6 in 13,93 %
Sum 7 in 16,72 %
Sum 8 in 13,83 %
Sum 9 in 11,11 %
Sum 10 in 8,33 %
Sum 11 in 5,58 %
Sum 12 in 2,73 %
BUILD SUCCESSFUL (total time: 0 seconds)
```