# Afleveringsopgave 4

# Afleveres senest:
# *Søndag d. 15. november kl. 23.59*

**Jørgen Villadsen**

---

Se afleveringsopgave 1 for generel information. Bemærk at Java-filerne skal være i en ZIP-fil, men der må ikke være andre filer eller kataloger i denne (I skal ikke aflevere filer, som I har fået udleveret).

Opgaverne er ikke lige omfattende, og nogle af opgaverne kræver kendskab til emnerne, som gennemgås 27/10, men det er en god ide at kigge lidt på opgaverne inden da... :-)

---

## Opgave 1

I denne opgave skal der udvikles et ret simpelt program `Letters`, der tæller antallet af tegn angivet på kommandolinjen som argumenter. For eksempel (her er > operativsystemets prompt):

```
>java Letters abc de f gh
8
```

Det er i orden at bruge Eclipse i stedet for kommandolinjen — i så fald skal argumenterne (`abc de f gh`) angives i menuen Run / Run Configurations og så i fanebladet Arguments.

Programmet skal testes grundigt, og dokumentationen herfor skal indgå i besvarelsen.

# Opgave 2

Formålet med opgaven er at skrive et program, der kan håndtere artikler og deres referencer.

Skriv en klasse `Forlag`, hvis instanser repræsenterer forskellige forlag. Et forlag er beskrevet ved navnet på forlaget (f.eks. "University Press") og stedet, hvor forlaget ligger (f.eks. "Denmark"). Klassen skal således have to felter: `navn` og `sted`. Klassen skal også indeholde en konstruktør til at oprette et nyt forlag med forlagets navn og sted angivet som parametre.

Skriv en klasse `Tidsskrift`, hvis instanser repræsenterer tidsskrifter. Et tidsskrift er beskrevet ved en titel, et forlag og et ISSN-nummer. Klassen skal således have tre felter: `titel`, `forlag` og `issn`. ISSN-nummeret er en entydig kode, som alle tidsskrifter har (en streng). Forlaget skal have typen `Forlag`. Klassen skal også indeholde en konstruktør til at oprette et nyt tidsskrift med dets titel angivet som parameter. Tidsskriftets ISSN-nummer og forlag skal kunne sættes efterfølgende med to metoder `setIssn` og `setForlag`.

Skriv en klasse `Artikel`, hvis instanser repræsenterer artikler i tidsskrifter. En artikel er beskrevet ved en liste (et array) af forfattere, en titel, en angivelse af tidsskriftet, den er publiceret i, og en reference-liste. Reference-listen indeholder listen (arrayet) over de andre artikler, som den pågældende artikel refererer til. Klassen skal således have følgende felter: `forfattere`, `titel`, `tidsskrift` og `referenceliste`. Klassen skal også indeholde en konstruktør til at oprette en ny artikel. Konstruktøren skal tage forfattere, titel og tidsskrift som parametre, idet elementerne i referencelisten senere skal kunne sættes med en metode `setReferenceliste`, som også skal skrives.

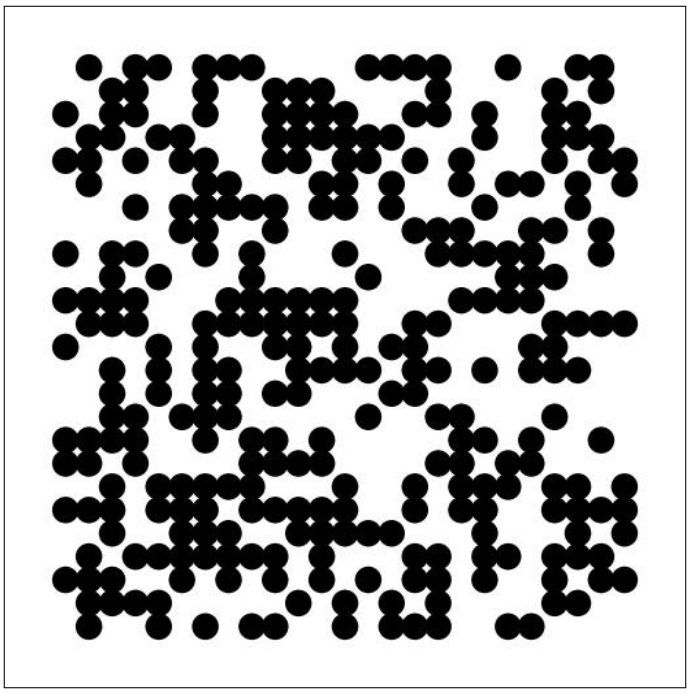
Hvis ikke andet er specificeret, så kan strenge benyttes som felters type.

Klasserne skal testes grundigt. Specielt skal der skrives en klasse `ArtikelTest` med en `main`-metode, som opretter følgende:

- Forlaget *University Press*, *Denmark*.
- Tidsskrifterne *Journal of Logic* og *Brain*. Disse to tidsskrifter kommer begge fra *University Press*. ISSN-numrene kendes ikke.
- Følgende to artikler:
  - A. Abe & A. Turing: "A". Journal of Logic.
  - B. Bim: "B". Journal of Logic.

  Den første af disse artikler har en reference til den anden.

Benyt `toString`-metoder i klasserne (dog ikke i `ArtikelTest`-klassen).



Figur 1: Eksempel på mulig tilstand i Game of Life.

# Opgave 3

Denne opgave omhandler Conway's *Game of Life*. Game of Life er en simpel grafisk simulator, hvor brugeren angiver simulationens starttilstand, og derefter kan følge hvordan tilstanden udvikler sig på ofte forundrende og uforudsete måder. Game of Life er beskrevet på følgende Wikipedia-side, men de grundlæggende elementer vil blive også gennemgået i opgaveteksten.

`https://en.wikipedia.org/wiki/Conway's_Game_of_Life`

I Game of Life består en tilstand af et $n$ gange $n$ grid af felter kaldet *celler*, hvor hver celle enten er død eller levende. For at visualisere en tilstand tegnes en sort prik på koordinat $(x,y)$ hvis cellen på plads $(x,y)$ er levende (en død celle markeres ikke).

Hver celle har otte naboer således som illustreret på figur 2(a) (hver af de otte naboer til den midterste celle er givet et nummer).

På baggrund af en given tilstand kan den næste tilstand findes ud fra følgende regler:

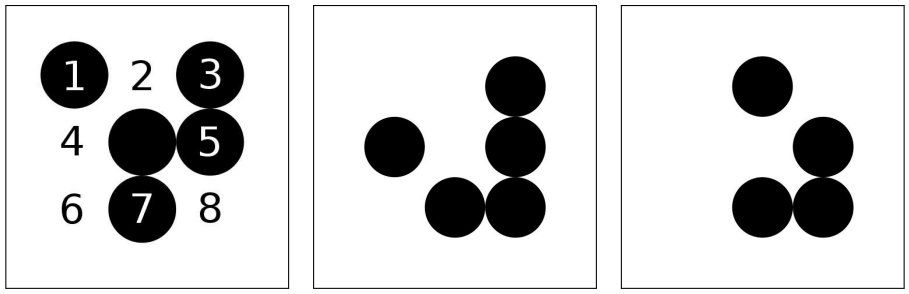
(1) En levende celle med færre end to naboer er død af ensomhed i næste tilstand.

(2) En levende celle med flere end tre naboer er død af pladsmangel i næste tilstand.

(3) En levende celle med to eller tre levende naboer lever uforandret videre i næste tilstand.

(4) En død celle med præcis tre levende naboer bliver vækket til live i næste tilstand.

Figur 2 viser hvordan en tilstand ændrer sig trin for trin. I trin 2 er den midterste celle fra trin 1 død af pladsmangel, da den har fire levende naboer; den øverste venstre celle er død af




(a) Trin 1. Den midterste celle interagerer med sine otte naboer.

(b) Trin 2

(c) Trin 3

(d) Trin 4

(e) Trin 5

(f) Trin 6

Figur 2: Eksempel på udviklingen for en given starttilstand.

ensomhed, fordi den kun har én levende nabo, mens den nederste højre celle er vækket til live, da den har præcis tre levende naboer, osv.

1. Implementér Conway's Game of Life, hvor `StdDraw` benyttes til at visualisere hvordan en given tilstand ændrer sig — det bliver altså en slags lille animation. I bør implementere jeres løsning som mindst to klasser: `GameOfLife` og `GameOfLifeMain`.

   Klassen `GameOfLife` skal definere `GameOfLife`-objekter der repræsenterer en tilstand i en Game of Life-simulation. Klassen skal desuden tilbyde metoder til at manipulere denne tilstand, f.eks. at ændre værdien af en celle, simulere ét trin frem og lignende. Internt i `GameOfLife` foreslås det, at man repræsenterer tilstanden ved et 2-dimensionelt array af heltal (`int[][]`), hvor et 1-tal angiver en levende celle og 0 en død. Husk at et 2-dimensionelt array af heltal svarer til en matrix i matematisk forstand. Tilstanden på figur 2(a) vil således være repræsenteret ved følgende matrix (2-dimensionelle array):

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

   Jeres `GameOfLife`-klasse skal mindst have følgende to konstruktører:

   - `public GameOfLife(int n)`. En konstruktør til at generere et Game of Life med starttilstanden værende et `n` gange `n` grid af tilfældigt initialiserede celler.
   - `public GameOfLife(int[][] initialState)`. En konstruktør til at at generere et Game of Life med starttilstanden repræsenteret i arrayet `initialState`.

I klassen vil I også få brug for en række metoder på `GameOfLife`-objekter. Eksempelvis bør der være en metode `nextState()` som genererer den efterfølgende tilstand i det aktuelle `GameOfLife`. I forbindelse med at skrive metoden `nextState()` kan det være en fordel at have defineret en privat hjælpemetode `liveNeighbours(int x, int y)`, som tæller hvor mange levende naboer cellen `(x,y)` har i det aktuelle `GameOfLife`.

Klassen `GameOfLifeMain` skal indeholde klientkoden, herunder `main`-metoden, som instantierer et `GameOfLife`-objekt og udnytter de metoder som objektet tilbyder.

For at sikre at jeres animation kører flydende, skal I benytte metoden `StdDraw.show(n)`, hvor `n` er et heltal (`int`). Denne metode tegner billedet og venter derefter `n` millisekunder. Efterfølgende kald af tegne-metoder vil ikke blive vist med det samme, men først ved næste kald til `StdDraw.show(n)`.

Metoden `StdDraw.clear()` kan benyttes til at slette det tegnede.

`StdDraw` benytter som standard en figur-størrelse på 512 gange 512 pixels, men hvis I eksempelvis ønsker en figur på 1000 gange 1000 pixels, så skal I blot kalde metoden `StdDraw.setCanvasSize(1000,1000)`.

Test jeres løsning på et par forskellige `GameOfLife`-instanser.

2. Programmet `GameOfLifeMain` skal nu udvides så det kan indlæse starttilstanden fra en fil. Formatet af filen er som en matrix af 0'er og 1-taller, f.eks. følgende som svarer til tilstanden på figur 2(a):

```
1 0 1
0 1 1
0 1 0
```

I filen `gol.zip` findes nogle eksempelfiler med forskellige interessante starttilstande. De ligger i filerne `toad.gol`, `pulsar.gol`, `pentadecathlon.gol`, `glider_gun.gol` og `acorn.gol`. Afprøv jeres program på disse filer.

Herunder er der nogle eksempler på mulige udvidelser af programmet. Det er ikke påkrævet at I laver nogle af disse udvidelser. *Det er kun hvis I har ekstra tid og ønsker ekstra udfordring.*

- Modificér programmet så det holder øje med om simuleringen på et tidspunkt bliver periodisk, altså om en bestemt sekvens af tilstande begynder at gentage sig igen og igen. Få i dette tilfælde programmet til at printe perioden, dvs. antallet af tilstande som forløber imellem to gentagelser.
- Lad banen være en *torus* således at øverste kant er forbundet med nederste kant, og venstre kant med højre (ligesom i computerspillet Pac-Man).
- Introducér farver på cellerne og lav forskellige regler for overlevelse som afhænger af disse farver.
- Find eventuelt mere inspiration på Wikipedia-siden!