Afleveringsopgave 4

Afleveres senest:
Søndag d. 3. april 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).

Opgave 1

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).

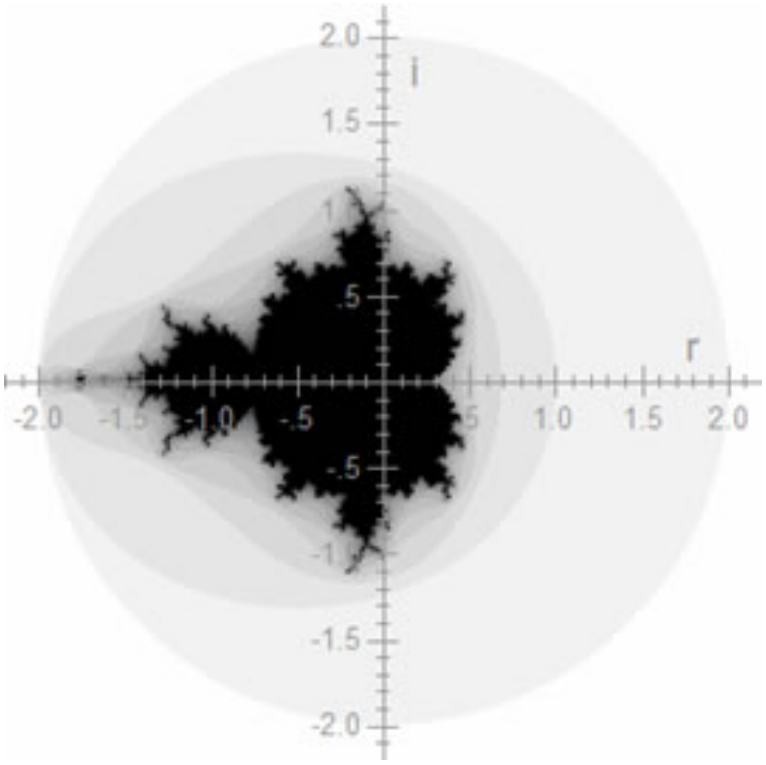
Opgave 2

Denne opgave omhandler *fraktaler*. Fraktaler er særlige geometriske mønstre som udmærker sig ved at være bygget op af delmønstre som alle er formindskede kopier af det overordnede mønster (eller approksimerede kopier). Begrebet *fraktal* blev introduceret af den franske matematiker Benoît Mandelbrot i 1975. En af de mest kendte fraktaler, *Mandelbrot-mængden*, er opkaldt efter ham. Mandelbrot-mængden er i virkeligheden blot en mængde af komplekse tal, men når vi aftegner mængden af disse tal i den komplekse talplan får vi smukke fraktale mønstre. Denne opgaven går ud på at lave et Java-program som kan tegne forskellige udsnit af Mandelbrot-mængden.

Et simpelt billede af Mandelbrot-mængden er givet i figur 1. Mandelbrot-mængden kan ikke beskrives ved en enkel matematisk ligning. Den er i stedet defineret ved en *algoritme*, og er derfor primært velegnet til at blive udregnet af en computer. Reglen for at afgøre om et komplekst tal z_0 er i Mandelbrot-mængden er følgende. Vi betragter en følge af komplekse tal $z_1, z_2, z_3, z_4, \ldots$ defineret ud fra z_0 på følgende måde:

$$
\begin{aligned}
z_1 &= (z_0)^2 + z_0 \\
z_2 &= (z_1)^2 + z_0 \\
z_3 &= (z_2)^2 + z_0 \\
z_4 &= (z_3)^2 + z_0 \\
&\vdots
\end{aligned}
$$

Det er altså således at vi for alle positive heltal i sætter $z_i = (z_{i-1})^2 + z_0$. Vi siger nu at tallet z_0 *ikke* tilhører Mandelbrot-mængden hvis talfølgen $|z_1|, |z_2|, |z_3|, |z_4|, \ldots$ går mod uendelig, dvs. hvis tallene i følgen bliver ved med at vokse. Hvis talfølgen til gengæld er begrænset, det vil sige, hvis tallene stopper med at vokse eller bliver mindre og mindre, så *er* tallet i Mandelbrot-mængden.



Figur 1: Koordinater for Mandelbrot-mængden i den komplekse talplan.

Lad os tage et par konkrete eksempler. Lad $z_0 = 0.3 + 0.8i$. Da kan vi udregne følgen af komplekse tal $z_1, z_2, z_3, z_4, \ldots$:

$$
\begin{aligned}
z_1 &= (z_0)^2 + z_0 = -.25 + 1.28i \\
z_2 &= (z_1)^2 + z_0 = -1.28 + 0.16i \\
z_3 &= (z_2)^2 + z_0 = 1.9 + 0.39i \\
z_4 &= (z_3)^2 + z_0 = 3.77 + 2.29i \\
z_5 &= (z_4)^2 + z_0 = 9.23 + 18.05 \\
z_6 &= (z_5)^2 + z_0 = -240.19 + 334.06i \\
z_7 &= (z_6)^2 + z_0 = 53903.6 - 16047.12i \\
z_8 &= (z_7)^2 + z_0 = -2285650426 + 1729995129i
\end{aligned}
$$

Som det ses går det stille og roligt med de første par udregninger, men pludselig begynder følgen at stige meget kraftigt. Det kan vises at så snart følgen rammer et tal z_i med $|z_i| > 2$ vil følgen *nødvendigvis* gå mod uendelig, det vil sige, tallet vi startede med er *ikke* med i Mandelbrot-mængden.

Lad os som et andet eksempel se på tallet $z_0 = -0.5$. Udregner vi følgen $z_1, z_2, z_3, z_4, \ldots$ ud fra

dette tal fås:

$$z_1 = (z_0)^2 + z_0 = -0.25$$
$$z_2 = (z_1)^2 + z_0 = -0.44$$
$$z_3 = (z_2)^2 + z_0 = -0.31$$
$$z_4 = (z_3)^2 + z_0 = -0.40$$
$$z_5 = (z_4)^2 + z_0 = -0.34$$
$$z_6 = (z_5)^2 + z_0 = -0.39$$

Som det ses bliver følgen her ved med at antage ret små værdier. Det er derfor *mest sandsynligt* at tallet -0.5 *er* med i Mandelbrot-mængden. Men i princippet kan vi ikke være sikre, for følgen kunne pludselig begynde at stige på et senere tidspunkt. Det er altså ikke altid vi ved om et tal er med i Mandelbrot-mængden eller ej. Men vi kan tilnærme mængden ved at udregne f.eks. de første 100 tal i følgen, det vil sige tallene $z_1, z_2, \ldots, z_{100}$, og så checke om vi undervejs møder et tal z_i med $|z_i| > 2$. Hvis vi gør det *ved* vi at tallet z_0 vi startede med ikke er med i mængden. Hvis ikke vi undervejs møder et tal med modulus større end 2 "gætter" vi på at tallet *er* med i mængden.

1. Skriv en klasse `Complex` til repræsentation af komplekse tal i Java.

   Objekter af typen `Complex` kan med fordel beskrives igennem to felter `re` og `im` af typen `double`. De to felter `re` og `im` repræsenterer da henholdsvis realdelen og imaginærdelen af det komplekse tal. De to felter bør deklareres som private af hensyn til indkapslingen af klassen (eksempelvis hvis man på et tidspunkt ønsker at skifte til polær repræsentation i implementationen).

   Jeres klasse skal som minimum have følgende metoder:

   - `public Complex()`. En konstruktør til at konstruere et nyt `Complex`-objekt repræsenterende det komplekse tal 0.
   - `public Complex(double re, double im)`. En konstruktør til at konstruere et nyt `Complex`-objekt med realdel `re` og imaginærdel `im`.
   - `public Complex(Complex z)`. En konstruktør til at konstruere et nyt `Complex`-objekt repræsenterende det samme komplekse tal som `z`.
   - `public double getRe()`. Returnerer realdelen af `Complex`-objektet.
   - `public double getIm()`. Returnerer imaginærdelen af `Complex`-objektet.
   - `public double abs()`. Returnerer modulus af `Complex`-objektet. Modulus af et komplekst tal $a + ib$ er $\sqrt{a^2 + b^2}$.
   - `public Complex plus(Complex other)`. Returnerer et `Complex`-objekt repræsenterende summen af det aktuelle `Complex`-objekt og `Complex`-objektet `other`.

     Husk på at regnereglen for addition af komplekse tal ser ud på følgende måde:

     $$(a + bi) + (c + di) = (a + c) + (b + d)i.$$

- `public Complex times(Complex other)`. Returnerer et `Complex`-objekt repræsenterende produktet af det aktuelle `Complex`-objekt og `Complex`-objektet `other`.

  Husk på at regnereglen for multiplikation af komplekse tal ser ud på følgende måde:

$$(a+bi)\cdot(c+di) = (ac-bd)+(bc+ad)i$$

- `public String toString()`. Returnerer en streng der på passende vis repræsenterer `Complex`-objektet (f.eks. på formen `"a + ib"`).

2. Afprøv jeres `Complex`-klasse ved at lave et lille klient-program `ComplexTest` der afprøver de forskellige metoder fra `Complex`-klassen. I behøver ikke dokumentere `ComplexTest` i rapporten, men kan nøjes med et par kommentarer i selve klient-programmet.

   I kan f.eks. teste at der gælder følgende lighed:

$$(1+2i)+(4+5i) = (5+7i).$$

   Det kan I gøre ved at give følgende kommandoer:

```
Complex z1 = new Complex(1,2);  // z1 = 1 + 2i
Complex z2 = new Complex(4,5);  // z2 = 4 + 5i
System.out.println(z1.plus(z2));  // print z1 + z2
```

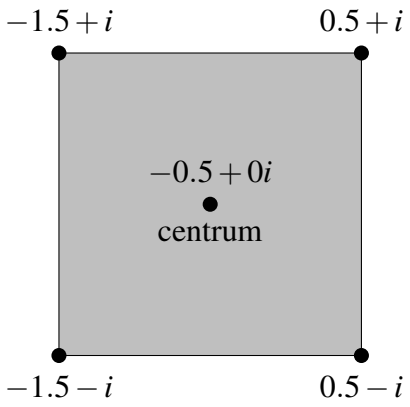
   Hvis jeres klasse fungerer efter hensigten skulle den sidste linje printe `5.0 + 7.0i` på konsollen. Bemærk at ovenstående stump kode både tester konstruktøren `Complex(double re, double im)` og instansmetoderne `plus` og `toString`. For at teste instansmetoden `times` kan man til ovenstående stump kode tilføje:

```
System.out.println(z1.times(z2));  // print z1 * z2
```

   Nu skulle programmet så gerne printe `-6.0 + 13.0i`, som er produktet af de to komplekse tal $1+2i$ og $4+5i$.

3. Betragt følgende metode som anvender `Complex`-klassen:

```
public static int iterate(Complex z0) {
   Complex z = new Complex(z0);
   for (int i = 0; i < MAX; i++) {
      if (z.abs() > 2.0) {
         return i;
      }
      z = z.times(z).plus(z0);
   }
   return MAX;
}
```



Figur 2: Udsnit af den komplekse plan med centrum i $-0.5+0i$ og sidelængde 2.

I ovenstående kode antages `MAX` at være en klasse-konstant af typen `int`.

Metoden `iterate` ovenfor kan benyttes til at tegne approksimationer af Mandelbrot-mængden. I skal skrive et program `Mandelbrot` som benytter `Complex` og `StdDraw` til at tegne Mandelbrot-mængden. Herunder beskrives hvordan programmet skal fungere.

Start med i programmet `Mandelbrot` at definere værdien af klassekonstanten `MAX`. Værdien af `MAX` giver et mål for hvor præcis approksimationen af Mandelbrot-mængden bliver. Jo større værdi `MAX` har, des mere præcis bliver approksimationen. Men store værdier af `MAX` betyder også at udregningerne tager længere tid. Arbejd med små værdier af `MAX` i begyndelsen, f.eks. `MAX = 20`.

Programmet `Mandelbrot` skal indlæse tre reelle tal fra konsollen. Disse tal angiver hvilket kvadratiske udsnit af den komplekse talplan vi ønsker at se på Mandelbrot-mængden i. De første to tal angiver centrum af udsnittet, og det tredie tal angiver sidelængden af udsnittet. Hvis eksempelvis der bliver indlæst de tre tal `-0.5`, `0` og `2` betyder det at vi ser på det udsnit som har centrum i $-0.5+0i$ og som har sidelængden 2. Dette er illustreret på figur 2.

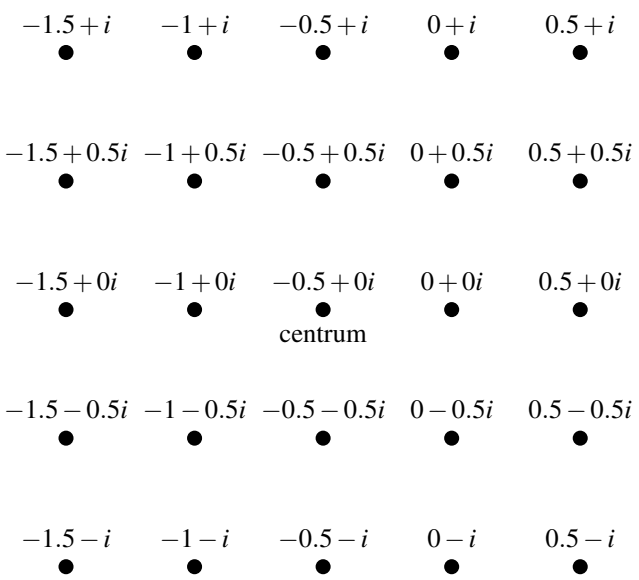
Figur 1 viser koordinaterne af Mandelbrot-mængden i den komplekse talplan. Som det ses ligger Mandelbrot-mængden i det store og hele indenfor det udsnit der har centrum i $-0.5+0i$ og sidelængde 2.

Det af brugeren valgte udsnit betragtes nu som et punkt-grid. På figur 3 er vist et 5 gange 5 grid på udsnittet med centrum i $-0.5+0i$ og sidelængde 2. De angivne komplekse tal på figuren svarer til koordinaterne for hver af de 25 punkter i griddet. Størrelsen af griddet — dvs. antallet af punkter i højden og bredden — kan gemmes i en klasse-konstant. Sørg for at vælge et relativt lille tal i begyndelsen, f.eks. 5 eller 10.

Lad nu g betegne grid-størrelsen, dvs. antallet af punkter på hver led i griddet. Lad x_0+y_0i betegne centrum for griddet, og lad s betegne griddets sidelængde. Det er da således at punktet i række j og søjle k af griddet har koordinaterne svarende til følgende komplekse tal:

$$\left(x_0-\frac{s}{2}+\frac{s\cdot j}{g-1}\right)+\left(y_0-\frac{s}{2}+\frac{s\cdot k}{g-1}\right)i.$$

Hvis vi eksempelvis som i figur 3 lader $g=5$, $x_0=-0.5$, $y_0=0$ og $s=2$ får vi specielt



Figur 3: Et 5 gange 5 punkt-grid på udsnittet med centrum i $-0.5+0i$ og sidelængde 2.

at punktet i række j og søjle k af griddet har koordinater svarende til følgende komplekse tal:

$$\left(-0.5-\frac{2}{2}+\frac{2\cdot j}{5-1}\right)+\left(0-\frac{2}{2}+\frac{2\cdot k}{5-1}\right)i=(-1.5+0.5j)+(-1+0.5k)\,i.$$

Sætter vi $j=k=0$ fås det komplekse tal $-1.5-i$ svarende til punktet i nederste venstre hjørne, jvf. figur 3 (bemærk at vi tæller rækker og søjler fra 0 og op, ligesom i Java). Sætter vi $j=k=4$ fås det komplekse tal $0.5+i$ svarende til punktet i øverste højre hjørne, jvf. igen figur 3.

Det er nu sådan at et punktet $x+yi$ i griddet skal farvelægges hvis og kun hvis metoden `iterate` anvendt på $x+yi$ returnerer konstanten `MAX`. Figur 4 viser et simpelt eksempel på griddet fra figur 3. Klassekonstanten `MAX` er her sat til 20.

Det midterste punkt på figuren er $-0.5+0i$. Det er farvelagt fordi `iterate(z)` returnerer 20 når `z` har værdien $-0.5+0i$. Med andre ord gælder der at følgende udtryk returnerer værdien 20:

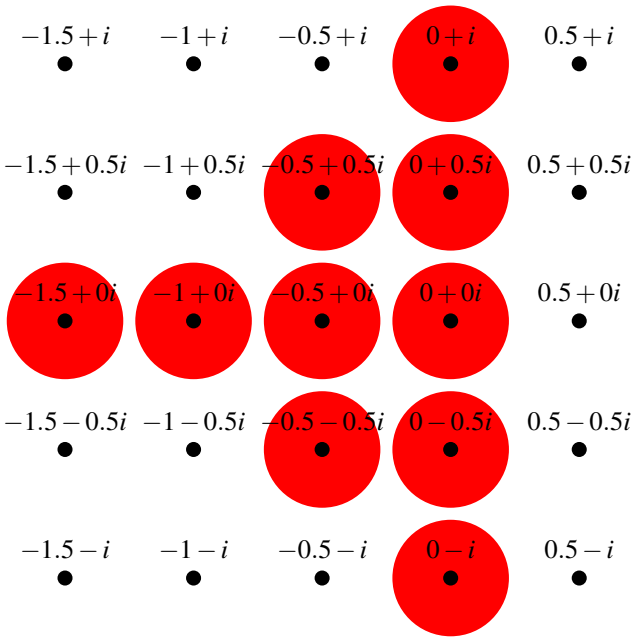
```
iterate(new Complex(-0.5,0))
```

Punktet i øverste højre hjørne er $0.5+i$. Det er ikke farvelagt, fordi `iterate(z)` returnerer tallet 1 når `z` er $0.5+i$. Med andre ord gælder der at følgende udtryk returnerer værdien 1:

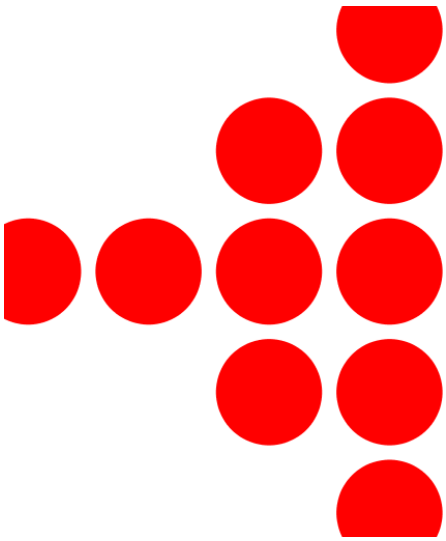
```
iterate(new Complex(0.5,1))
```

Da 1 er mindre end `MAX` skal det tilsvarende punkt ikke farvelægges.

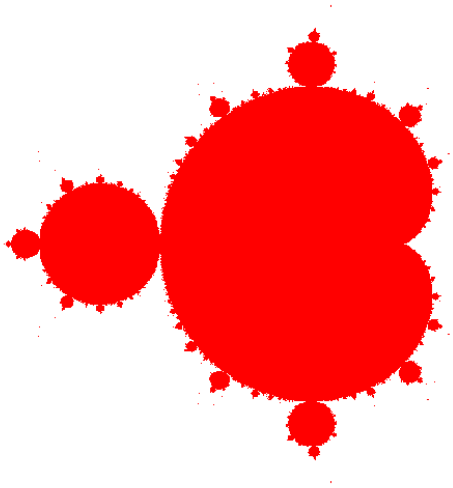
Benyt `StdDraw` til at lave farvelægningen af de enkelte punkter. Figur 5 viser et eksempel på hvordan outputtet kan se ud når vi anvender samme grid-størrelse, sidelængde og



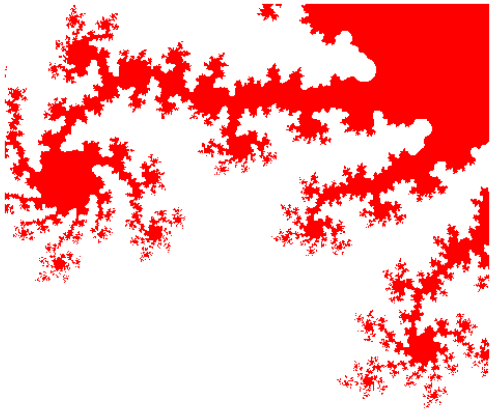
Figur 4: Simpel Mandelbrot med gridstørrelse 5, `MAX = 20`, centrum i $-0.5+0i$ og sidelængde 2.



Figur 5: Simpel Mandelbrot med gridstørrelse 5, `MAX = 20`, centrum i $-0.5+0i$ og sidelængde 2.



Figur 6: Simpel Mandelbrot med gridstørrelse 512, `MAX = 255`, centrum i $-0.5+0i$ og sidelængde 2.



Figur 7: Simpel Mandelbrot med gridstørrelse 512, `MAX = 50`, centrum i $0.10259-0.641i$ og sidelængde 0.0086.

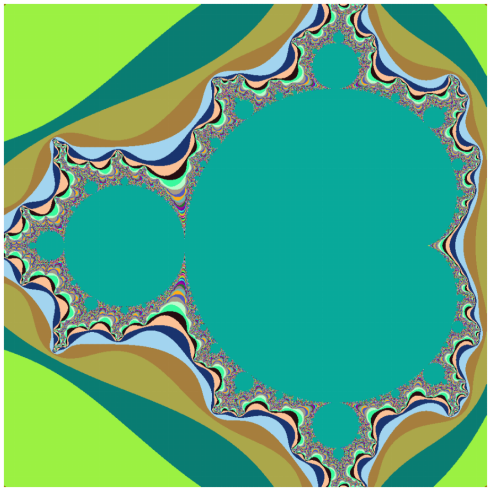
centrum som ovenfor (sammenlign figur 5 med figur 4). Et punkt kan farvelægges med metoden `point` (benyt evt. også metoden `setPenRadius`). Start med små grids som i figur 5. Når det fungerer tilfredsstillende kan I prøve med større grids. Figur 6 viser et eksempel med en grid-størrelse på 512 og `MAX = 255`.

Når I har skrevet klassen færdig så afprøv den med forskellige værdier af `MAX` og forskellige udsnit af den komplekse plan. Figur 7 viser udsnittet med centrum i $0.10259-0.641i$ og sidelængden 0.0086, og med `MAX = 50`. En grid-størrelse på 512 er generelt passende. Gem nogle af eksemplerne som png- eller jpg-filer og inkludér dem i jeres rapport.

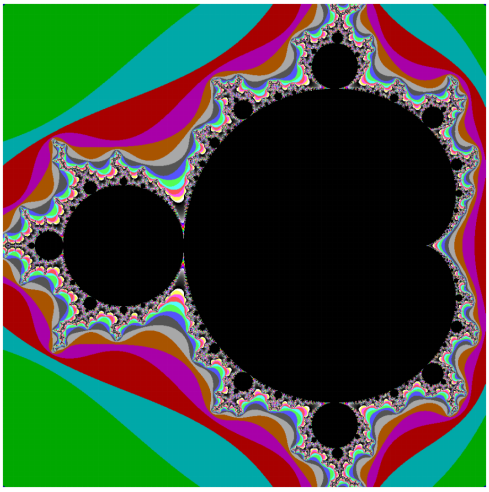
Tekniske hints: Når man bruger `StdDraw` på almindelig vis tegnes grafikken med det samme når tegne-metoderne (`line`, `point`, osv.) kaldes. Når man gerne vil vise Mandelbrotmængder i store grids begynder dette at være en lidt for tidskrævende proces. Man kan i stedet generere al grafikken inden man viser resultatet på skærmen. Det gøre ved at kalde metoden `StdDraw.show(0)` én gang *før* alle tegne-kommandoerne og så igen én gang *efter* alle tegne-kommandoerne. F.eks. vil følgende lille cirkel-program først tegne en cirkel på skærmen når alle punkterne langs cirklen er blevet genereret:

```
StdDraw.setXscale(-1,1);
StdDraw.setYscale(-1,1);
StdDraw.show(0);
double n = 100;
for (int i = 0; i < n*2*Math.PI; i++) {
   StdDraw.point(Math.cos(i/n), Math.sin(i/n));
}
StdDraw.show(0);
```

`StdDraw` benytter som standard en figur-størrelse på 512 gange 512 pixels. Hvis I vil operere med større grids end dette skal I således ændre på figur-størrelsen. Hvis



Figur 8: Farvet Mandelbrot med gridstørrelse 1000, `MAX = 255`, centrum i $-0.5+0i$ og sidelængde 2. Farverne er tilfældigt valgte.



Figur 9: Farvet Mandelbrot med gridstørrelse 1000, `MAX = 255`, centrum i $-0.5+0i$ og sidelængde 2. Farverne er fra filen `mandel.mnd`.

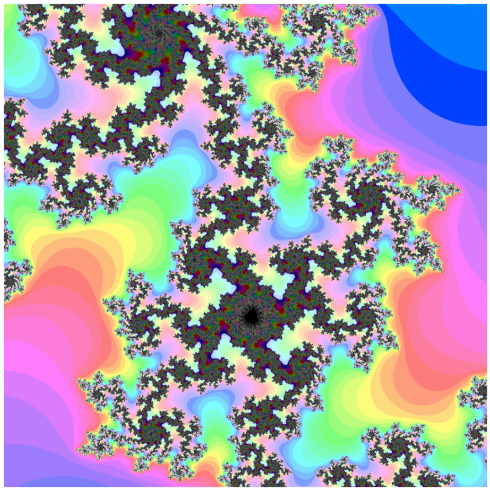
I eksempelvis ønsker en figur på 1000 gange 1000 pixels skal I blot kalde metoden `StdDraw.setCanvasSize(1000,1000)`.

4. Denne delopgave går ud på at udvide til programmet `Mandelbrot` så det giver et flerfarvet output. For nemheds skyld vil vi i det følgende antage at `MAX` er sat til den faste værdi 255. I stedet for blot at benytte en enkelt farve kan man farvelægge punkterne efter den værdi som `iterate` returnerer. Farver sættes i `StdDraw` med metoden `setPenColor`. Denne metode tager et objekt af typen `Color` som argument. Klassen `Color` i pakken `java.awt` indeholder forskellige konstruktører til at lave `Color`-objekter med. Der findes f.eks. en konstruktør af typen `Color(int r, int g, int b)` som opretter et `Color`-objekt ud fra tre heltal `r`, `g` og `b` i intervallet 0–255. Disse tre heltal angiver farvens indhold af rød, grøn og blå, henholdsvis. Så hvis man eksempelvis ønsker at at tegne noget i ren rød farve kan man give følgende kommando:

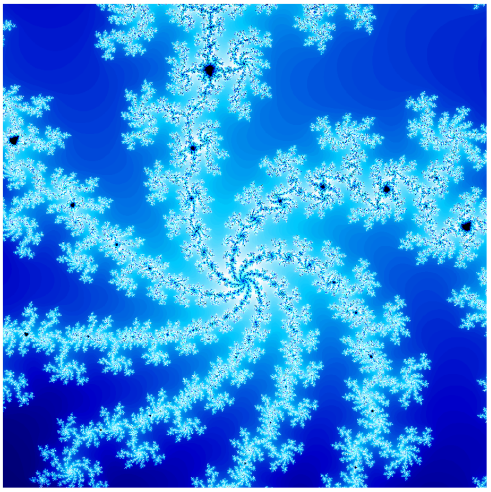
   ```
   StdDraw.setPenColor(new Color(255,0,0));
   ```

   På figur 8 er givet et eksempel på et farvet Mandelbrot hvor returværdierne fra `iterate` er benyttet til at vælge farven.

   Lav en udvidelse af jeres program `Mandelbrot` som tilknytter en tilfældig farve til hver af de 256 mulige returværdier fra `iterate` og benytter disse farver til at lave et farvelagt Mandelbrot som i figur 8. Afprøv programmet på et par forskellige udsnit.



Figur 10: Farvet Mandelbrot med gridstørrelse 1000, `MAX = 255`, centrum i $0.10087 - 0.63198i$ og sidelængde 0.0003. Farverne er fra filen `mandel.mnd`.



Figur 11: Farvet Mandelbrot med gridstørrelse 1000, `MAX = 255`, centrum i $0.10684 - 0.63675i$ og sidelængde 0.0085. Farverne er fra filen `blues.mnd`.

5. Denne delopgave går ud på at lave et farvelagt Mandelbrot som tager farveværdierne fra en fil.

   Hver linje i en sådan fil indeholder en beskrivelse af en farve angivet som tre tal i intervallet 0–255 der angiver indholdet af farverne rød, grøn og blå. Udvid programmet `Mandelbrot` så det kan indlæse en sådan fil og benytte filens farver til farvelægning. Filen indeholder netop 256 farver, så `MAX` skal igen sættes til den faste værdi 255.

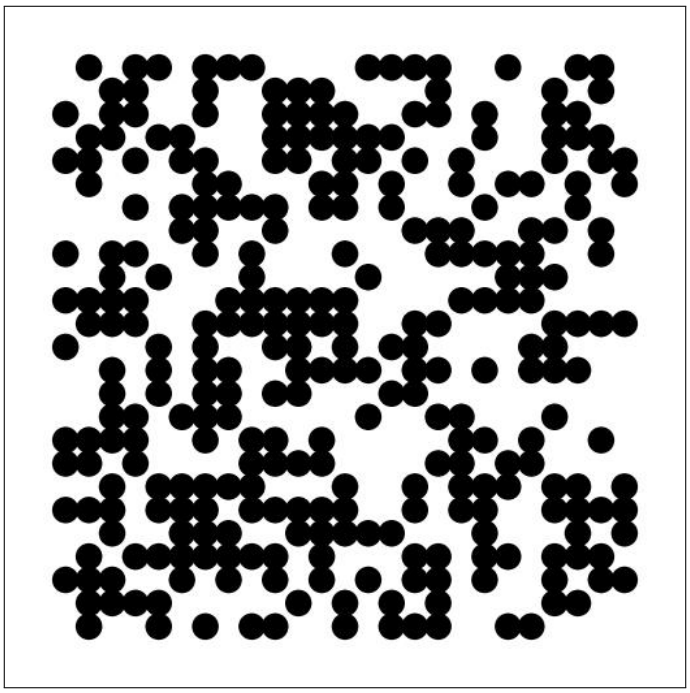
   I filen `mnd.zip` findes nogle eksempler på sådanne filer. De ligger i filerne `blues.mnd`, `chroma.mnd`, `glasses2.mnd`, `mandel.mnd`, `neon.mnd` og `volcano.mnd`. Afprøv jeres program på disse filer.

   I kan også prøve at lave jeres egne farve-filer hvis I har lyst (de skal ikke afleveres).

   Figur 9, 10 og 11 viser eksempler på output fra det udvidede program.

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.*

- Udvid programmet så man kan indtaste gridstørrelsen, figurstørrelsen, `MAX` etc.
- Udvid programmet så man med musen kan udvælge et delafsnit af Mandelbrot-mængden som man vil zoome ind på. Benyt metoderne `mousePressed()`, `mouseX()` og `mouseY()` fra `StdDraw` til formålet. Metoden `mousePressed()` returnerer en `boolean`, som angiver om brugeren har trykket med musen, og `mouseX()` og `mouseY()` angiver det pågældende koordinat som `double`. Hvis I implementerer denne udvidelse bør I nok sørge for at arbejde med en forholdsvis lille gridstørrelse, f.eks. 200, for at det ikke tager for lang tid at zoome.



Figur 12: 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 13(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:

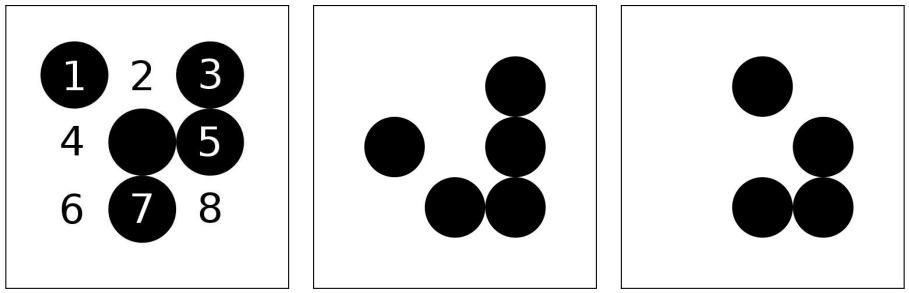
(1) En levende celle med færre end to naboer er død af ensomhed i næste tilstand.

(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.

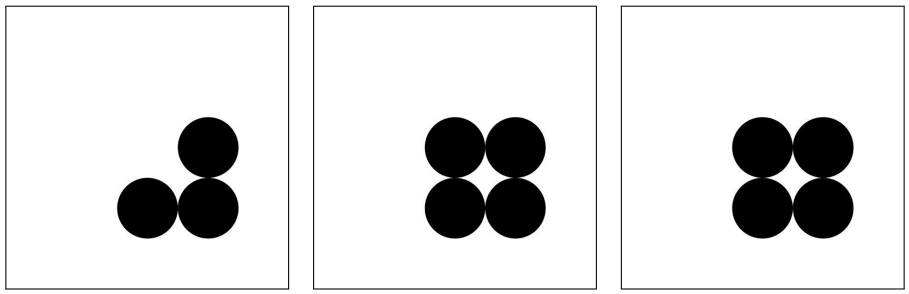
Figur 13 viser hvordan en tilstand ændrer sig trin for trin. I trin 2 er den midterste celle fra trin 1 død af pladsmangel, da den har fire levende naboer; den øverste venstre celle er død af



(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 13: 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 13(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 13(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!