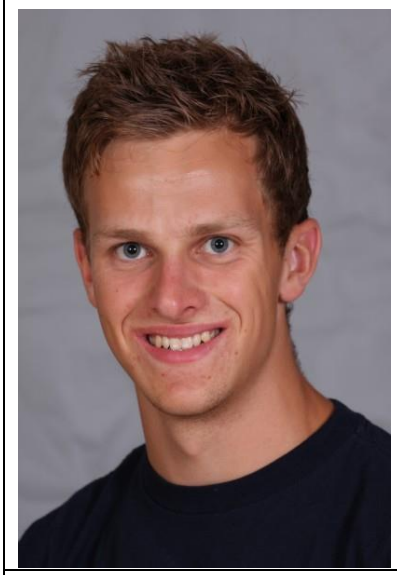
Mads Bornebusch, s1233627
Kristian Sloth Lauszus, s123808
Simon Piotr Almind Patrzalek, s123401

14. April, 2013

Afleveringsopgave 4

02102 Indledende Programmering

Mads Bornebusch s1233627



Kristian Sloth Lauszus s123808

Simon Patrzalek s123808

Rapporten og øvelsen har været et samarbejde mellem Mads Friis Bornebusch, Kristian Sloth Lauszus og Simon Patrzalek.
Vi har alle lavet opgaverne hver for sig og har derefter tjekket om vi havde de samme løsninger. Herefter har vi gennemgået en selektion af de bedste løsninger. Det er så disse som er kommet med i denne rapport.

Mads Bornebusch, s1233627
Kristian Sloth Lauszus, s123808
Simon Piotr Almind Patrzalek, s123401

14. April, 2013

Opgave 1

I denne opgave var det meningen at vi skulle benytte vores færdigheder inden for brugen af klasser. Det skulle vel og mærke være klasser vi selv lavede. Man skulle lave en klasse som indeholdt alt fra konstruktører til metoder, som kunne bruges i klassifikation af artikler. Dette skulle i sidste ende munde ud i klientkode, som skulle være i stand til at oprette en artikel, hvor der adskillige parametre beskrev denne artikel.

Abstraktionsniveauet var dog lavt i starten. Her skulle vi blot lave en klasse til som kunne klassificere et forlag med to parametre, nemlig navn af forlaget og hvilket land forlaget hørte til. Dette foregik ganske enkelt ved at oprette en separat klasse, som havde en konstruktør og en toString-metode. Herefter skulle vi bruge en klasse som kunne behandle tidsskrifter, med tidskriftets titel som input til konstruktøren. Efter dette skulle der laves metoder som tillod at ændre ISSN-nummeret og forlaget af tidsskriftet. Her brugte vi at vi allerede havde defineret forlag som en type. Hermed opnåede vi en kobling mellem klasserne. Ligeledes lavede vi også en toString-metode for Tidsskrift-klassen.

Tidsskrift-klassen indeholdte ikke nogle metoder til at lagre information på listeform, men ved Artikel-klassen brugte vi lister til at lagre information. En artikel ville således bestå af en liste med fire felter, henholdsvis for forfattere, titel, tidsskrift og referenceliste. Her havde vi igen mere kobling med klasserne, da vi allerede havde defineret tidsskrift-klassen, som så igen refererede til Forlag-klassen.

```
public Artikel(List<String> forfattere, String titel, Tidsskrift tidsskrift) {
        this.forfattere = forfattere;
        this.titel = titel;
        this.tidsskrift = tidsskrift;
}
```

Figur 1 - Konstruktøren for Artikel-klassen

```
public String toString(boolean getReferences) {
        String output = "";
        int size = forfattere.size();
        for(String forfatter : forfattere) {
                output += forfatter;
                size--;
                if(size == 1) // Put & between the two last writers
                        output += " & ";
                else if(size > 1) // Put a comma between writers
                        output += ", ";
        }
        output += ": \"" + titel + "\". ";
        output += tidsskrift;

        if(getReferences && !referenceliste.isEmpty()) { // Print the reference list
                output += "\nReference List: ";
                for(Artikel artikel : referenceliste) {
                        output += artikel.toString(false);
                        output += " ";
                }
                output += "\n";
        }

        return output;
}
```

Figur 2 - toString-metoden for Artikel-klassen

Mads Bornebusch, s1233627
Kristian Sloth Lauszus, s123808
Simon Piotr Almind Patrzalek, s123401

14. April, 2013

Opagave 2 – Mandelbrot

Mandelbrotmængden består af de komplekse tal hvor følgende iteration ikke går mod uendelig:

$$z_{n+1} = z_n + z_0$$

Hvis man som z_0 sætter alle tal i den komplekse plan ind, undersøger om de er med i mængden eller ej og til sidst plotter dem der er, får man mandelbrotmængden. Når man undersøger om et tal er med i mængden, itererer man ovenstående funktion og hvis tallet kommer længere end 2 fra 0+0i er det ikke med i mængden da det så vil gå mod uendelig.
På næsten samme måde kan man for hvert tal i mængden finde dets Juliamængde. Juliamængden for et givet komplekst tal c, findes ved at iterere følgende funktion hvor der som z_n indsættes alle tal i den komplekse talplan:

$$z_{n+1} = z_n + c$$

Tallet c er altså det samme ved alle iterationer. Tallet er på samme måde med i mængden hvis det ikke går mod uendelig, og dette finder man ligeledes ud af ved at tjekke om tallet kommer længere end 2 fra 0+0i.
I programmet er funktionen der itererer tallene for at finde juliamængden givet som:

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

På samme måde bliver talle itereret for at finde ud af om de er med i Mandelbrotmængden. Her bliver z_0 dog lagt til i stedet for tallet *juliaConstant*. Det er spændende at se på juliamængden for forskellige tal da denne er forskellig afhængigt af om de er med i Mandelbrotmængden eller ej. Hvis et tal er med i Mandelbrotmængden bliver tallets juliamængde en sammenhængende mængde. Hvis tallet derimod ikke er med i mandelbrotmængden bliver Juliamængden en støvfraktal. En interessant ting ved Juliamængderne er, at hvis de på noget sted ikke er sammenhængende så er der intet af mængden der er sammenhængende. Dette kan vises matematisk, men det er ikke formålet her. Fraktler som Mandelbrotmængden og Juliamængderne har mange spændende egenskaber og indeholder mange flotte mønstre. I vores program har vi derfor valgt at fokusere på at gøre det let tilgængeligt at udforske den. Programmet er uafhængigt af konsollen og alt kan styres direkte fra stdDraw-vinduet.
Nedenfor er en række keyboardkommandoer til programmet anført.

Keyboard commands		
Key	**Betydning**	**Hvad sker der**
h	HELP!	En boks med nedenstående kommandoer vises
i	Iterations	Spørger brugeren om et nyt af iterationer
w	Where am I?	Giver koordinaterne på centrum
c	Color	Spørger brugeren om et nyt navn på en color-fil
g	Go to	Giver brugeren mulighed for at indtaste centrum og sidelængde
+	Zoom in	Zoomer ind til dobbelt størrelse
-	Zoom out	Zoomer ud til halv størrelse
j	Julia set	Draws the Julia set for the complex number at the center
m	Mandelbrot set	Draws the Mandelbrot set

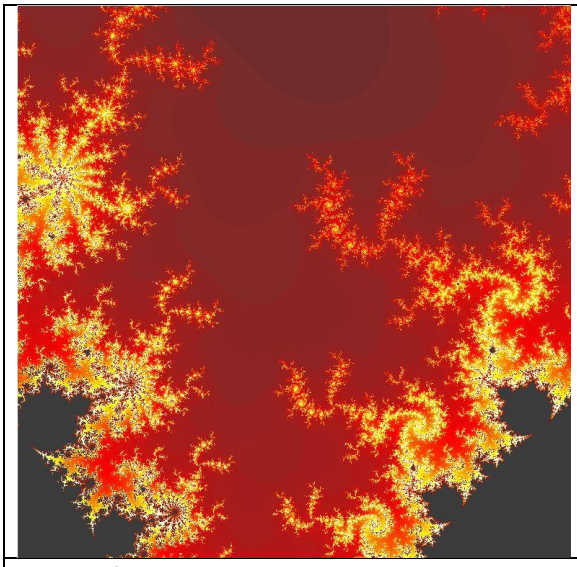
Ud over disse kommandoer har vi implementeret styring med musen. Et tryk med musen vil sætte et Både tastetryk og styring med musen er implementeret ved hjælp af StdDraw-klassen. Nedenfor er den kode vist der kører så længe brugeren ikke trykker på musen eller keyboardet:

```
while(!(StdDraw.mousePressed()|| StdDraw.hasNextKeyTyped() ));
            if(StdDraw.mousePressed()){
                        int corX = (int)Math.round(StdDraw.mouseX());
                        int corY = (int)Math.round(StdDraw.mouseY());
            }
            if (StdDraw.hasNextKeyTyped()){
                        key = Character.toLowerCase(StdDraw.nextKeyTyped());
            }
            while(StdDraw.mousePressed() || StdDraw.hasNextKeyTyped());
```

Hvis der bliver trykket med musen bliver cursorens koordinater lagt over i to variable. Input fra tastaturet bliver lagt over i en variabel af typen Char. Når brugeren ikke trykker på noget længere går koden videre og tester for hvad der blev trykket på. Afhængigt af inputtet gør programmet de forskellige ting fra tabellen ovenfor.

Ekstra funktioner ved tastetryk

Funktionen help er bare en *PLAIN_MESSAGE* fra `JOptionPane` der indeholder en tekst der forklarer funktionerne. På samme måde vil "w" give en boks der fortæller koordinaterne på det nuværende centrum.
For at kunne ændre det maksimalt tilladte antal af iterationer var vi nødt til at ændre hvordan det nuværende antal af iterationer blev omsat til en farve. Hvis vi dividerer det aktuelle antal iterationer med `(double)maxIterations/((double)COLORDEPTH-1)` vil man altid få et tal der ligger mellem 0 og *COLORDEPTH*. Dette tal er så det der bliver brugt til at vælge en farve. Hvis der ikke er valgt nogen farve-fil bliver dette antal givete videre både som R, G og B-værdi for farven. Man får således en farveskala af gråtoner.



Detalje fra Mandelbrotmængden koordinaterne er ca -0.04+0.8i, billedet er tegnet med 720 iterationer og farverne fra volcano.mnd

Når der trykkes på "c" spørges brugeren om navnet på en farvefil. Hvis denne fil findes, loades den og bruges til at farvelægge mandelbrotmængden. Da antallet af iterationer gemmes i et array er det forholdsvist hurtigt at ændre farve når man først har beregnet antallet af iterationer for alle tal i det udsnit af den komplekse talplan der vises.
Når der trykkes på "g" kommer der tre dialogbokse op der giver brugeren mulighed for at indtaste koordinater på centrum af kanvasset og sidelængden af det udsnit der skal vises. Det er således muligt hurtigt at gå til et bestemt udsnit af mandelbrotmængden uden at man skal zoome hele vejen ind.
Zoom-funktionen zoomer for hvert tryk på "+" ind så arealet af det udsnit der vises er halvt så stort som arealet af det udsnit der var vist tidligere. Når man zoomer ind deles sidelængden derfor med $\sqrt{2}$.
Koden der zoomer ind er vist nedenfor:

```
            sidelength /= Math.sqrt(2);
            redrawAll(colors);
```

Metoden `redrawAll(colors)` beregner og tegner udsnittet forfra med de samme farver.

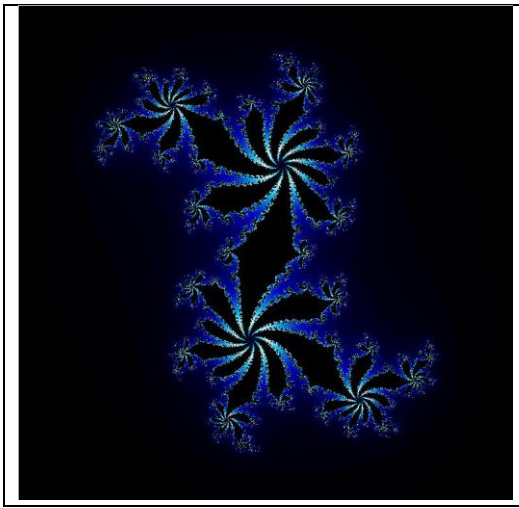
Juliamængder

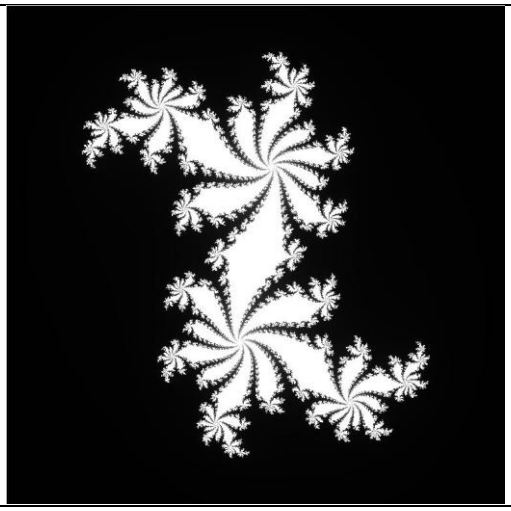
Når man trykker på "j" tegnes Juliamængden for det komplekse tal der var centrum på kanvasset. Hvis man derfor vil have Juliamængden for et bestemt punkt kan man derfor bruge "g" eller et klik med musen for at

gå dertil. Når man skal tilbage til Mandelbrotmængden trykker man ”m” og man kommer så tilbage til samme udsnit som man var i før man fik vist Juliamængden. Dette gøres med nedenstående kode:

```
juliaConstant = new Complex(centerRe , centerIm);
sidelengthMandel = sidelength;
centerReMandel = centerRe;
centerImMandel = centerIm;
mandelbrot = false;
```

Centrum gemmes først som konstanten *juliaConstant* og derefter gemmes alle værdier i variable. Når man tegner Mandelbrotmængden igen indlæses værdierne tilbage til de variable der benyttes til at tegne med så man ser samme udsnit. På denne måde kan gå på opdagelse i forskellige Juliamængder der ligger tæt på hinanden, f.eks. på randen af mandelbrotmængden.





Juliamængden vist med farverne fra filen blues.mnd og uden farver.

Mads Bornebusch, s1233627
Kristian Sloth Lauszus, s123808
Simon Piotr Almind Patrzalek, s123401

14. April, 2013

Opgave 3 – Conways Game of Life

I denne opgave skulle vi lave et program, som skulle kunne simulere primitivt liv. Simpelt forklaret gik det ud på at en celle, som enten kunne være død eller levende, havde en tilstand som var bestemt af dens omkringliggende celler. For hvert "tik" af uret ændrede cellen således tilstand, hvis antallet af dens naboer havde ændret sig siden sidste tik. Basalt foregik det ved at indlæse en matrix som indeholdt en masse 1-taller og 0-taller. Et 1-tal står for en levende celle. Herefter blev matricen behandlet og de nye værdier af cellerne indlæst i matricen, hvorefter denne blev brugt til at vise ændringen grafisk.

I opgaveformuleringen lød det at vi skulle skrive to klasser for at kunne løse opgaven. En GameOfLife-klasse og en GameOfLifeMain-klasse. GameOfLife-klassen skulle indeholde de metoder som blev i behandlingen af matricen, som indeholdte information over de levende og døde celler. Main-klassen skulle indeholde metoden til at visse ændringen i populationen grafisk.

I GameOfLife-klassen sørgede metoden nextState for at generere matricen som blev brugt til næste "tik". Denne bliver ganske enkelt løbet igennem ved hjælp af to for-loops, da den var todimensionel. Metoden nextState havde hjælpemetoden liveNeighbours, som kontrollede hvor mange levende celler en given celle havde som nabo. På baggrund af hvilken værdi liveNeighbours returnerede, erklærede nextState så en given celle for død eller levende.

```
public int liveNeighbours(int x, int y) {
                              int liveNeighbours = 0;

                              for(int i = -1; i <= 1; i++) {
                                            int corY = y+i;
                                            for(int j = -1; j <= 1; j++) {
                                                          int corX = x+j;
                                                          if(corX >= 0 && corY >= 0 && corX < gridSize &&
corY < gridSize && (corX != x || corY != y)) {
                                                                        if(state[corY][corX] == 1)
                                                                                      liveNeighbours++;
                                                          }
                                            }
                              }
                              return liveNeighbours;
                }
```

Figur 3 liveNeighbours-hjælpemetoden

```
                public void nextState() {
                              int[][] newState = new int[gridSize][gridSize];
                              for(int y = 0; y < gridSize; y++) {
                                            for(int x = 0; x < gridSize; x++) {
                                                          int liveNeighbours = liveNeighbours(x,y);
                                                          if(liveNeighbours < 2 || liveNeighbours > 3)
                                                                        newState[y][x] = 0;
                                                          else if(liveNeighbours == 3)
                                                                        newState[y][x] = 1;
                                                          else
                                                                        newState[y][x] = state[y][x];
                                            }
                              }
                              state = newState.clone();
                }
```

Figur 4 next-State metoden

Vores udvidelse til denne opgave var at implementerer en såkaldt JFileChooser der åbner en dialogboks ved programmets start der lader brugeren vælge en start tilstand ud fra en tekstfil. Hvis brugeren annullerede dialogboksen blev der blot genereret et random tilstand.

```
JFileChooser chooser = new JFileChooser(new File(System.getProperty("user.dir")));
int returnVal = chooser.showOpenDialog(null);

GameOfLife gol;
if(returnVal != JFileChooser.APPROVE_OPTION)
                gol = new GameOfLife(gridSize);
else
                gol = new GameOfLife(openMatrix(chooser.getSelectedFile().getPath()));
```

Figur 5 Implementeringen af JFileChooser