Dozent
Prof. Dr. Thomas Vetter
Dep. Mathematik und Informatik
Spiegelgasse 1
CH – 4051 Basel

Assistenten
Bernhard Egger
Andreas Forster

Tutoren
Sein Coray
Jonas Finkler
Eddie Joseph
Loris Sauter
Linard Schwendener
Florian Spiess

Webseite
http://informatik.unibas.ch/hs2016/erweiterte-grundlagen-der-programmierung/

Erweiterte Grundlagen der Programmierung (45398-01)

Blatt 7 [0 Punkte]

Vorbesprechung	07. Nov - 11. Nov
Abgabe	14. Nov - 18. Nov (vor dem Tutorat)

Aufgabe 1: Mandelbrotmenge (Praxis) [8 Punkte]

Die Lösung dieser Aufgabe wird später erneut verwendet.
In dieser Aufgabe schreiben Sie ein Programm, dass die Mandelbrotmenge darstellt. Die Mandelbrotmenge ist definiert, als die Teilmenge der komplexen Zahlen c, für die die Folge $z_0 = 0, z_{n+1} = z_n^2 + c$ beschränkt ist. D.h., die Mandelbrotmenge besteht aus den Zahlen, für die es eine Konstante k gibt, so dass alle Elemente der Folge $z_n(c)$ kleiner k sind.

$$\{c \in \mathbb{C} \mid \exists k \forall n : z_n < k \text{ mit } z_0 = 0, z_{n+1} = z_n^2 + c\} \tag{1}$$

Man kann diese Menge darstellen, indem man die zu ihr gehörenden Punkte der komplexen Zahlenebene einfärbt. Um die obige Definition für eine gegebene Zahl c zu testen, müsste man *alle* Elemente der Folge z_n betrachten, was in der Praxis natürlich nicht möglich ist. Darum stellen wir stattdessen die "Fluchtgeschwindigkeit" der Folge $z_n(c)$ dar. Die "Fluchtgeschwindigkeit" definieren wir als das kleinste n von $z_n(c)$, so dass $|z_n(c)| > 2$.

Schreiben Sie unter Verwendung der auf der Webseite gegebenen Klassen ein Programm, dass Abbildungen der Mandelbrotmenge erzeugt. Das Programm soll über die Pixel eines Bildes laufen, und jedem Pixel eine komplexe Zahl entsprechend der (x, y) Position des Pixels zuordnen. Für jeden Pixel berechnen sie die Fluchtgeschwindigkeit, und färben den Pixel entsprechend ein.

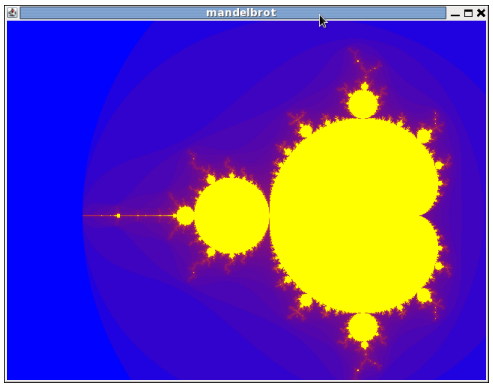
Der zu implementierende Funktionsaufruf in den vorgegebenen Klassen ist

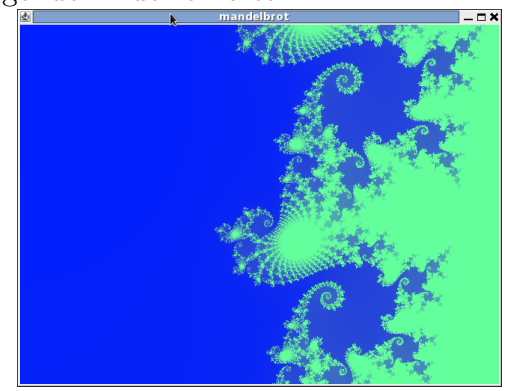
```
show_mandelbrot_inplace(Complex origin, double stepsize, int maxiter)
```

Sei `origin` die Zahl $a + bi$ (wobei $i = \sqrt{-1}$ die imaginäre Einheit ist), dann ordnen Sie dem Pixel (x, y) die Zahl $c = (a + \texttt{stepsize} \cdot x) + (b + \texttt{stepsize} \cdot y)i$ zu, und betrachten Sie höchstens `maxiter` Elemente der Folge $z_n(c)$.

Färben Sie die Pixel mit Hilfe der Klasse `ColorPalette` ein, welche Ihnen eine Instanz der Klasse `java.awt.Color` zurück gibt. Ein Aufruf ist im Quellcode von der Vorlesungswebseite zu finden. Benutzen Sie die Werte dieser Farbe um im ImageWindow die

Pixel einzufärben. Sie sollten in etwa folgende Bilder erhalten.





Teilaufgabe 1 [3 Punkte]

Implementieren Sie die Klasse `Complex`, die die komplexen Zahlen abbildet. Eine komplexe Zahl ist eine Zahl $a + bi$ mit folgenden Rechenregeln:

$$\text{Addition:} \quad (a+bi)+(c+di) = (a+c)+(b+d)i \tag{2}$$

$$\text{Multiplikation:} \quad (a+bi)(c+di) = (ac-bd)+(ad+bc)i \tag{3}$$

$$\text{Betrag:} \quad |(a+bi)| = \sqrt{a^2+b^2} \tag{4}$$

Die Struktur der Klasse ist vorgegeben, Sie müssen noch die Methodenrümpfe einfüllen.

Beachten Sie, dass `Complex add(Complex c)`, `Complex mult(Complex c)` und `Complex sqr()` nicht den Wert der komplexen Zahl verändern, sondern ein neues Objekt zurückgeben, während die Versionen mit der Endung `inplace` den Wert des Objektes verändern, und das Objekt selbst zurückgeben.

Teilaufgabe 2 [3 Punkte]

Implementieren Sie das Zeichnen der Mandelbrotmenge. Nutzen Sie die Methoden, die den Wert der Komplexen Zahl nicht verändern, sondern immer neue Objekte zurückgeben.

Teilaufgabe 3 [2 Punkte]

Vergleichen Sie die Geschwindigkeit der Implementation aus Teilaufgabe 2 mit einer Implementation, die die inplace Varianten der Komplexen Operationen verwendet. Messen Sie die Geschwindigkeit mit Hilfe der Funktion `System.currentTimeMillis()`.

Aufgabe 2: Game of Life (Praxis) [4 Punkte]

Die Lösung dieser Aufgabe wird später erneut verwendet.

Das Spiel des Lebens (Game of Life) geht auf den Mathematiker John Conway zurück und funktioniert nach folgenden Prinzipien:

`( http://www.math.com/students/wonders/life/life.html )`

Die Grundeinheit sind Zellen, die in einer Matrix angeordnet sind. Jede Zelle kann lebendig oder tot sein. Jede Zelle hat acht Nachbarn, wobei Randzellen die Zellen des gegenüberliegenden Randes als Nachbarn haben. Der Zustand der Zellen (lebendig oder tot) ändert sich von Generation zu Generation. Die aktuelle Zellpopulation beeinflusst die darauf folgende Generation nach folgenden Regeln:

(a) Eine tote Zelle mit genau drei lebenden Nachbarn erwacht zum Leben (birth).

(b) Eine lebende Zelle mit zwei oder drei lebenden Nachbarn bleibt am Leben (survival).

(c) Alle anderen lebenden Zellen sterben (overcrowding or loneliness).

Klasse [1 Punkt]
Schreiben Sie eine Klasse `GameOfLife` welche als Feld eine Zellenpopulation als eine $size \times size$ Matrix vom Typ `boolean` speichert. Jedes Elemente hat entweder den Wert `false` (tote Zelle, '.') oder `true` (lebende Zelle, '@'). Speichern Sie auch die Grösse `size` als Feld der Klasse.

Konstruktoren [1 Punkt]
Schreiben Sie zwei Konstruktoren. Einen parameterlosen Konstruktor welcher die Feldgrösse auf 6×6 initialisiert und folgendes Muster setzt.

```
......
.@.@..
..@@..
..@...
......
......
```

Der zweite Konstruktor soll ein Parameter haben, die Feldgrösse. Der Wert jeder Zelle soll dabei zufällig gesetzt werden. Mit einer Wahrscheinlichkeit von 30% soll eine Zelle als "lebend" initialisiert werden. Verwenden Sie dazu die `java.util.Random` Klasse.

Evolution [1 Punkt]
Schreiben Sie eine Methode um einen Wert aus dem Feld abzufragen, eine Methode welche die lebenden Nachbarn zählt, und eine Methode welche das Update ausführt. Dabei soll der Zustand aller Zellen zur "gleichen" Zeit ausgeführt werden. Verwenden Sie dazu eine Kopie der Zellpopulation. Beachten Sie ferner dass der Zugriff auf eine Zelle jeweils auf der gegenüberliegenden Seite geschieht wenn auf die Nachbarn von Zellen am Rand zugegriffen wird.

Ausgabe [$\frac{1}{2}$ Punkt]
Schreiben Sie nun noch eine Methode `public String toString()` welche die Population in einem String darstellt. Um Zeilenumbrüche in einem `String` darzustellen können Sie die Zeichenfolge "\n" verwenden. Jeder Zelle soll dabei durch ein `.` oder ein `@` dargestellt werden.

Testprogram [$\frac{1}{2}$ Punkt]
Schreiben Sie ein Testprogramm `BasicGOL` welches die Klasse `GameOfLife` benützt, eine Instanz erstellt, Evolutionsschritte berechnet und das Ergebniss jeweils auf die Konsole ausgibt.