Grundlagen der Programmierung

Marcel Lüthi
Andreas Morel-Forster
Universität Basel
Fachbereich Informatik
HS 23

Übung 8

Voraussetzung

- Ein JDK ist installiert.
- Installierte IDE, Visual Studio Code sowie die Plugins für Java und Gradle
- Wenn Sie die Vorlesung verpasst haben, dann empfehlen wir Ihnen die Unterlagen anzuschauen.
- Die Zip-Datei, die auch dieses Übungsblatt enthält, muss entpackt werden. Es enthält die gesamte Übungsumgebung. Schreiben Sie ihre Lösungen in die dafür vorgesehenen Dateien, wie in der jeweiligen Übungsaufgabe angegeben.

Wichtiger Hinweis

- *Achten Sie auf guten Programmierstil.*

Aufgabe 8.1 (Stack, 3 Punkte)

In dieser Aufgabe implementieren Sie eine neue Datenstruktur, genannt *Stack*. Genau wie ein Array, kann ein Stack eine Sequenz von Elementen speichern. Dabei stellt der Stack jedoch nur ganz eingeschränkte Methoden zur Verfügung um Elemente darin zu speichern. Ein Stack unterstützt nämlich nur folgende Operationen:

(a) Die Operation `push` fügt ein neues Element zum Stack hinzu.

(b) Die Operation `pop` entfernt das Element, das zuletzt hinzugefügt wurde.

(c) Die Operation `size` gibt die Anzahl Elemente zurück, die gerade im Stack gespeichert sind.

Auf deutsch wird Stack auch oft Stapel genannt, da sich dieser wie ein Stapel verhält. Neue Elemente werden zuoberst auf den Stapel gelegt, und jeweils wieder von oben vom Stapel genommen. Implementieren sie diese Operationen in der Klasse `Stack`, welche Sie im Verzeichnis `src/main/java/` finden. Nutzen Sie intern ein Array um die Elemente zu speichern. Um die Implementation zu vereinfachen, können Sie beim Konstruieren des Stacks die Kapazität des Stacks angeben, also die maximale Anzahl Elemente, die im Stack gespeichert werden können.
`Tipp:` Schreiben Sie eigene kleine Testprogramme in der Main-methode um Ihre Implementation zu testen. Wenn alles funktioniert nutzen Sie auch die mitgelieferten Tests.

Aufgabe 8.2 (Komplexe Zahlen)

Eine komplexe Zahl ist eine Zahl $a + bi$ mit folgenden Rechenregeln:

$$\text{Addition:} \quad (a+bi)+(c+di) = (a+c)+(b+d)i \tag{1}$$

$$\text{Multiplikation:} \quad (a+bi)(c+di) = (ac-bd)+(ad+bc)i \tag{2}$$

$$\text{Betrag:} \quad |(a+bi)| = \sqrt{a^2+b^2} \tag{3}$$

Im Verzeichnis `src/main/java` finden Sie die Klasse `Complex`. Implementieren Sie die fehlenden Methoden.
Beachten Sie, dass Sie die Operationen Addition und Multiplikation jeweils in zwei Varianten implementieren müssen. In der einen Variante soll der berechnete Wert als neues Objekt zurückgegeben werden. In der zweiten Variante wird das aktuelle Objekt verändert.
Testen Sie Ihre Klasse mit den mitgelieferten Tests.

Aufgabe 8.3 (Mandelbrotmenge)

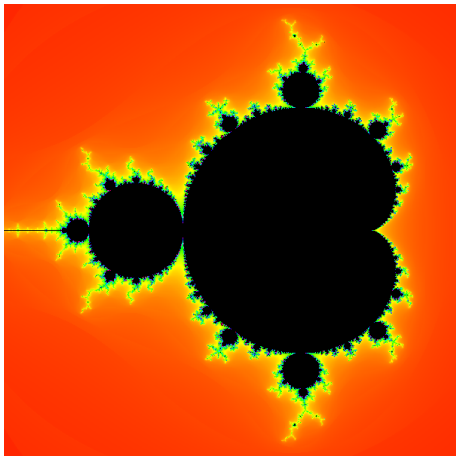
In dieser Aufgabe schreiben Sie ein Programm, dass die Mandelbrotmenge darstellt.
Die Mandelbrotmenge ist definiert als die Teilmenge der komplexen Zahlen c, für die die Folge $z_0 = 0, z_{n+1} = z_n^2 + c$ beschränkt ist. Das heisst, die Mandelbrotmenge besteht aus den Zahlen, für die es eine Konstante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

$$\{c \in \mathbb{C} \mid \exists k \forall n : z_n < k \text{ mit } z_0 = 0, z_{n+1} = z_n^2 + c\} \tag{4}$$

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 $|z_n(c)| > 2$.
Im Verzeichnis `src/main/java` finden Sie die Klasse `Mandelbrot`. Implementieren Sie die Methode `computeMandelbrot`, welche für eine gegebene Zahl die Mandelbrot Folge $z_0 = 0, z_{n+1} = z_n^2 + c$ berechnet, solange $|z_n| < 2$ ist. Dann soll ein MandelbrotResult Objekt zurück gegeben werden, welches die Resultate der Berechnung enthält.
Testen Sie Ihr Programm mit den mitgelieferten Tests.

Aufgabe 8.4 (Visualisierung der Mandelbrotmenge)

In dieser Aufgabe visualisieren Sie die Mandelbrotmenge. Dafür implementieren Sie die Methode `createMandelbrotVisualization` . Dafür laufen Sie über die Pixel eines Bildes, und ordnen jedem Pixel eine komplexe Zahl mit der Methode
`pixelPosToComplexNumber`. Berechnen Sie dann die Mandelbrotfolge mit der in der vorigen Aufgabe implementierten Methode und nutzen Sie die Hilfsklasse `ColorPalette` um eine Farbe entsprechend der Berechnung auszuwählen.
Wenn Sie das Programm kompilieren und mit den in der Main-Methode angegebenen Parametern ausführen, sollten Sie etwa folgendes Bild erhalten.



Aufgabe 8.5 (Color Klasse)

Implementieren Sie die Methode `ColorPalette.colorToGreyscale`. Diese soll eine Farbe, wie die von `ColorPalette.getColor` zurückgegeben wird, in Graustufen umwandeln. Dies schaffen Sie, indem die R G B Komponente der Farbe jeweils durch das gewichtete Mittel $R * 0.2989 + G * 0.5870 + B * 0.1140$ ersetzt wird.
Lesen Sie die Dokumentation der Klasse `Color` um die richtigen Methoden der Klasse `Color` zu finden: `https://docs.oracle.com/en/java/javase/11/docs/api/java.desktop/java/awt/Color.html`.
Testen Sie Ihre Implementation mit den mitgelieferten Tests.
Visualisieren Sie nun die Mandelbrotmenge auch als Graustufenbild.