UNIVERSITÄT BASEL

University of Basel

Dozenten
Prof. Dr. Thomas Vetter
Dr. Marcel Lüthi
Departement
Mathematik und Informatik
Spiegelgasse 1
CH – 4051 Basel

Assistent
Dr. Marcel Lüthi
Departement
Mathematik und Informatik
Spiegelgasse 1
CH – 4051 Basel

Tutoren / Tutorinnen
Claudia Grundke
Viktor Gsteiger
Simon Dold
Timo Steinebrunner
Alexander Rovner
Nikodem Kernbach
Lukas Stöckli

Erweiterte Grundlagen der Programmierung (45398-01)

Blatt 12 [10 Punkte]

Vorbesprechung	09. Dezember - 13. Dezember
Abgabe	20. Dezember

Allgemeine Hinweise

- Wir empfehlen Ihnen, dass Sie im Buch "Sprechen Sie Java" bis und mit Kapitel 21 lesen.

Voraussetzung

- Es gelten dieselben Voraussetzungen wie für Übungsblatt 1. Wenn Sie sich betreffend der Umgebung oder der automatisierten Tests noch unsicher sind, lesen Sie bitte nochmals sorgfältig das Infoblatt oder fragen Sie die Tutoren.
- Die Zip-Datei, die auch dieses Übungsblatt enthält, muss entpackt werden. Es enthält die gesamte Übungsumgebung inklusive der automatisierten Tests. Schreiben Sie Ihre Lösungen in die dafür vorgesehenen Dateien, wie in der jeweiligen Übungsaufgabe angegeben.

Empfohlenes Vorgehen

- Wechseln Sie in den Ordner `src/main/java`. Dort finden Sie die Dateien, in welche Sie Ihren Java Code schreiben.
- Schreiben Sie Ihr Programm, kompilieren Sie dieses mit dem Java Compiler `javac` und führen Sie es mit `java` aus, wie es in der Vorlesung gezeigt wurde.
- Wenn Sie denken, dass alles in Ordnung ist, wechseln Sie zurück ins Übungsverzeichnis `uebung12` und führen da `gradlew test` aus, um zu überprüfen ob Ihre Lösung die automatisierten Tests besteht. Überprüfen Sie auch Ihren Codestil mit `gradlew checkstyleMain`. Falls Ihr Code den Vorgaben entspricht, erhalten Sie einen Bonuspunkt.

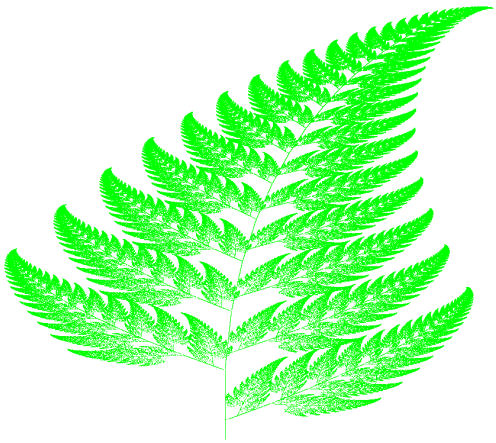
Abgabe

Ergänzen Sie die Datei email.txt mit Ihrer Unibas E-Mail Adresse. Erstellen Sie eine Zip-Datei der gesamten Übungsumgebung (also des Verzeichnisses `uebung12`) und laden Sie dieses auf Courses (https://courses.cs.unibas.ch/) hoch.

Aufgabe 1 - Barnsley Farn [3 Punkte]

Sie finden im Verzeichnis `src/main/java/fern` die Klasse `BarnsleyFern`.

Ihre Aufgabe ist es ein Programm zu schreiben, welches mittels den unten beschriebenen Regeln ein Farn wie auf dem Bild gezeigt produziert. Dabei lernen Sie die Klassen `java.awt.image.BufferedImage` und `java.util.Random` kennen. Lesen Sie die Java API-Dokumentation der enstprechenden Klassen.



- Sie sollen ein Bild vom Typ `java.awt.image.BufferedImage` der Grösse 1024× 1024 erstellen.
- Der Hintergrund vom Bild soll weiss sein.
- Sie sollen eine Hilfsklasse `Punkt` nutzen, welche als innere Klasse von der Klasse `BarnsleyFern` implementiert ist.
- Implementieren Sie die Regeln f_1 - f_4, die nachfolgend beschrieben sind. Sie können diese Testen, indem Sie die Kommentare um die Tests in `src/test/java/BarnsleyFernTests` entfernen und diese ausführen.
- Sie beginnen die Iteration mit dem Punkt `(0, 0)`.
- Danach sollen Sie update Regeln f_1 - f_4 mindestens 1'000'000 mal anwenden. Datei sollen in jeder Iteration jeweils zufällig eine Regel nach folgenden Wahrscheinlichkeiten ausgewählt werden:
  - f_1 mit Wahrscheinlichkeit 0.01
  - f_2 mit Wahrscheinlichkeit 0.85
  - f_3 mit Wahrscheinlichkeit 0.07
  - f_4 mit Wahrscheinlichkeit 0.07
- Nutzen Sie die Methode `nextDouble` der Klasse `java.util.Random` um Zufallszahlen zu erzeugen.

- Nutzen Sie folgende Formel um aus den Punktkoordinaten (x, y) die entsprechende Pixelkoordinate (i, j) im Bild zu berechnen:

$$i = 1024 \cdot (x + 3)/6,\ j = 1024 - 1024 \cdot (y + 2)/14$$

- Färben Sie den entsprechenden Pixel im Bild grün ein.

Regeln:

Die x und y Koordinaten von Punkt p_{n+1} sollen nach folgenden Regeln aus den Koordinaten von Punkt $p_n = (x, y)$ berechnet werden.

f_1: $(x, y) \mapsto (0, 0.16y_n)$

f_2: $(x, y) \mapsto (0.85x + 0.04y, -0.04x_n + 0.85y_n + 1.6)$

f_3: $(x, y) \mapsto (0.2x_n - 0.26y_n, 0.23x_n + 0.22y_n + 1.6)$

f_4: $(x, y) \mapsto (-0.15x_n + 0.28y_n, 0.26x_n + 0.24y_n + 0.44)$

Aufgabe 2 - Lambdas und Streams [4 Punkte]

In dieser Aufgabe arbeiten Sie mit dem Paket *java.util.stream* und lernen verschiedenen FunktionsInterfaces wie `Consumer, Predicate, Function` sowie die Klasse `Comparator` kennen.

Streams sind ein Konzept aus der funktionalen Programmierung. Sie dienen dazu eine Reihe von Operationen auf einer Liste von Objekten hintereinander auszuführen. Dabei können Sie sich vorstellen, dass man mit einer initialen Liste von Objekten beginnt. Jede Operation auf dem Stream erstellt eine neue Liste anhand der vorherigen Liste und der Operation selbst. Dabei nimmt jede Operation einen Parameter von einem genau definierten Funktionsinterface entgegen. In der Aufgabe werden folgende Operationen verwendet:

- **filter** - Entscheidet für jedes Element ob es in die neue Liste übernommen wird oder nicht. Filter erwartet einen Parameter der das Funktionsinterface *Predicat<T>* erfüllt.
- **map** - Erstellt aus jedem Objekt ein neues Objekt. Map erwartet einen Parameter der das Funktionsinterface *Function<T,R>* erfüllt.
- **sorted** - Sortiert eine ganze Liste nach einer Regel. Sorted erwartet einen Parameter der das Funktionsinterface *Comparator<T>* erfüllt.
- **foreach** - Macht etwas für jedes Element ohne eine neue Liste zu erstellen. Foreach erwartet einen Parameter der das Funktionsinterface *Consumer<T>* erfüllt.

Sie finden im Verzeichnis `src/main/java/streams` die Klasse `LambdasAndStreams`. Implementieren Sie die fehlenden Methoden. Lesen Sie dazu auch die API Dokumentation für die verwendeten Klassen. Testen Sie ihre Implementation mittels den automatisierten Tests.

Aufgabe 3 - Funktionale Bilder [3 Punkte]

In dieser Aufgabe erzeugen Sie Bilder, indem Sie mathematische Funktionen auswerten. Die Idee stammt ursprünglich von Conal Eliot. Unter folgender URL finden Sie eine Galerie mit vielen Beispielbildern: `http://conal.net/Pan/Gallery/`.

In dieser Aufgabe beschränken wir uns auf die einfachsten Bilder, nämlich einfache Schwarzweissbilder die auf Euklidischen Koordinaten definiert werden.[1] Ein Bild I ist also eine mathematische Funktion

$$I : \mathbb{R}^2 \to \{0, 1\}$$

die jedem Punkt $p = (x, y) \in \mathbb{R}^2$ den Wert $I(p) \in \{0, 1\}$ zuweist. Ein nur weisses Bild ist zum Beispiel durch die Funktion

$$I : (x, y) \mapsto 0$$

dargestellt. Das Bild, welches den Einheitskreis um den Ursprung zeigt, entspricht der Funktion

$$I : (x, y) \mapsto \begin{cases} 1 & \text{falls } \sqrt{x^2 + y^2} < 1 \\ 0 & \text{sonst} \end{cases} .$$

Das vertikale Streifen in der Abbildung unten links wäre demnach durch die Funktion

$$I : (x, y) \mapsto \begin{cases} 1 & \text{falls } |x| < 0.5 \\ 0 & \text{sonst.} \end{cases}$$

gegeben.

Wir können Variationen von Mustern erzeugen, indem wir eine Koordinatentransformation t definieren und damit das Gebiet, auf welchem unser Bild definiert ist, transformieren. Wir erhalten ein neues Bild I' durch die Komposition vom Bild I mit der Koordinatentransformation t:

$$I'(p) = (I \circ t)(p) = I(t(p)).$$

Eine einfaches Beispiel einer Koordinatentransformation ist eine Translation, die wie folgt definiert ist

$$(x, y) \mapsto (x + t_x, y + t_y)$$

wobei t_x, t_y Parameter sind, die die Grösse der Translation definieren.

Sie finden im Verzeichnis `src/main/java/images` die Klasse `FunctionalImage`, welche diese Idee mit Hilfe von boolschen Funktionen (also Funktionen vom Typ `Function<Point, Boolean>`) umsetzt. Die Methode `render` wertet die repräsentierte Funktion auf der Domain $[-1, 1] \times [-1, 1]$ aus und erzeugt daraus ein Bild.

- Implementieren Sie die Methode `createStrip` welche einen vertikalen Streifen der Breite 1 in der Mitte des Bildes zeichnet (siehe Bild unten links).
- Implementieren Sie die Methode `compose`, welches eine Funktion (Koordinatentransformation) vom Typ `Function<Point, Point>` entgegennimmt und daraus ein neues Bild durch Komposition dieser Funktionen mit dem Bild erzeugt. Für ein gegebenes Bild I und Koordinatentransformation t soll also $I \circ t$) berechnet werden.

[1] Conal Elliot beschreibt auch wie man interessante Bilder via Polarkoordinaten erzeugt, wie man Farbe dazunimmt und wie man Animationen macht.

- Implementieren Sie dann die Methode `rotate`, welche einen Parameter θ entgegennimmt und eine Funktion vom Type `Function<Point, Point>` zurückgibt. Die Funktion, die von `rotate(`θ`)` zurückgegeben wird, soll für jeden Punkt eine Rotation um den Nullpunkt um den Winkel θ (in Radians) ausführt. Nutzen Sie dabei folgende Formel um die Rotation zu implementieren:

$$(x, y) \mapsto (x\cos(\theta) - y\sin(\theta), y\cos(\theta) + x\sin(\theta))$$

  Durch Komposition vom Bild mit `rotate(`$\pi/4$`)` sollten Sie einen um 45° gedrehten Streifen erhalten, wie im Bild unten (Mitte) dargestellt.

- Implementieren Sie die Methode `swirl(r)`, welche einen Punkt nach folgender Formel transformiert:

$$p \mapsto \text{rotate}\underbrace{(dist(p) \cdot 2 \cdot \pi / r)}_{\text{Rotationsparameter}}(p).$$

  Dabei bezeichnet $dist(p)$ die Euklidische Distanz vom Punkt zum Koordinatenursprung und rotate(θ) ist die oben implementierte Rotationsmethode und r ist ein Parameter welcher vom Benutzer gewählt werden kann. Wenn Sie nun auch diese Funktion mit $r = 1$ ausführen, sollten Sie ein Bild wie unten rechts erhalten.

Hinweis: Zu dieser Aufgabe gibt es keine automatisierten Tests. Sie erkennen, ob ihr Programm korrekt ist, indem Sie die generierten Bilder anschauen.