Hogeschool PXL
Departement Digital
Opleiding Toegepaste Informatica
Academiejaar 2019-2020

PE

Vak: Java Advanced	
Resultaat	/ 4
Periode	Semester 1
Lectoren	N. Custers – S. Vanderstraeten – B. Clijsner

Opdrachten

Maak een maven project aan en plaats de startbestanden in een package *be.pxl.ja*.

Vraag 1: Functionele interface

Implementeer een generieke functionele interface **DistanceFunction** in package *be.pxl.ja.common*. De interface voorziet een abstracte methode distance. Je voorziet 1 parameter met een generiek datatype, het resultaat is een waarde van het primitieve datatype double.

Vraag 2: JUnit

Zorg ervoor dat de klasse **City** in package *be.pxl.ja.city* de generieke interface **DistanceFunction** implementeert. Adhv de lengte- en breedtegraad van 2 steden wordt de afstand tussen de 2 steden berekend. De implementatie van deze berekening vind je terug als commentaar in de klasse. Jij voorziet **2 unit testen** om deze methode te testen.

Vraag 3: Generieke methode

In de klasse **DistanceUtil** in package *be.pxl.ja.common* voorzie je een generieke hulpmethode (static) findClosest. De generieke methode heeft 2 parameters: een generieke set(!) met de naam *elements* en een generiek object *otherElement*.

De methode findClosest zoekt het object uit de lijst *elements* waarvoor de distance tot otherElement het kleinste is. Het gevonden object is de returnwaarde van de functie.

Vraag 3: DistancesBetweenCities

In het hoofdprogramma DistancesBetweenCities (in package *be.pxl.ja.city)* vind je al een aantal City-objecten terug. Plaats deze objecten in een set en sorteer alfabetisch (A->Z) op naam. Druk de namen van de gesorteerde City-objecten af.

Maak nu een nieuw City-object voor de gemeente waar je woont (zoek de lengte- en breedtegraad van je gemeente op!).

Zoek en druk af welke van de gemeenten uit de set het dichtst bij jouw gemeente ligt?

Vraag 4: Streams en lambda's

In package *be.pxl.ja.image* vind je de klassen **ImageReader** en **ImageWriter** met hulpmethoden waarmee je afbeeldingen kan lezen en schrijven.

Alle bestanden worden gelezen van of geschreven naar de folder "src/main/resources". In deze folder plaats je dus de afbeeldingen die je wil bewerken. Er is reeds 1 afbeelding (tokio.jpg) voorzien bij de startbestanden.

Bekijk even de documentatie van de klassen Path en Paths. Het path van een kleurenafbeelding wordt als argument aan de functie ImageReader.readImage() gegeven en je krijgt een List<List<RGBPixel>> terug. Ieder element in de geneste lijst geeft de rgb-waarde van een pixel van je afbeelding weer. Gebruik de functie ImageReader.readImage() nu om een afbeelding uit folder "src/main/resources" in te lezen.

Vraag 4a: Grayscale image

We gaan nu een stream implementeren om iedere RGBPixel van de ingelezen afbeelding te converteren naar een GrayscalePixel.

Implementeer hiervoor eerst in de klasse **RGBPixel** de methode *convertToGrayscale* waarbij je het gemiddelde neemt van de red-, green- en blue-waarde van je RGBPixel. Je kan dit implementeren mbv een stream als je een lijst maakt met deze 3 waarden!

Daarna kan je in het hoofdprogramma de stream voorzien om de geneste lijst te converteren. Het resultaat kan je wegschrijven naar een bestand met de naam “grayscale.jpg”. Gebruik voor het wegschrijven de methode *writeImage* in de klasse **ImageWriter**.

Vraag 4b: Fairey-cize

Onze geneste lijst met GrayscalePixel-objecten gaan we nu gebruiken om een kunstwerk van de afbeelding te maken. In 2008 maakte de straatartiest Shepard Fairey een beroemde poster van Barack Obama.

Wij gaan nu hetzelfde effect toepassen op onze foto.

Zorg hiervoor eerst dat GrayscalePixel-objecten gesorteerd kunnen worden adhv hun grayscale-waarde (laagste waarde naar hoogste waarde sorteren!).

Vervolgens zorg je ervoor dat de klasse GrayscalePixel de interface DistanceFunction implementeert om de afstand tussen 2 GrayscalePixel-objecten te berekenen. Deze afstand bereken je door de absolute waarde te nemen van het verschil van hun grayscale-waarden.

Test je implementatie van de distance-functie van de klasse GrayscalePixel met 2 relevante unit testen.

Maak nu een TreeSet aan met alle GrayscalePixel-objecten uit je geneste lijst.

De kleuren die Shepard Fairey gebruikte zijn reeds toegevoegd in de lijst *faireyColors*. Je mag indien gewenst deze kleuren veranderen.

Er is ook een hulpmethode beschikbaar waaraan je de Treeset met GrayscalePixel-objecten en de lijst met kleuren (faireyColors) kan meegeven als argumenten en die je een map geeft (Map<GrayscalePixel, RGBPixel>) die een vertaling maakt tussen de RGBPixel-objecten uit de lijst en hun best matchende grayscale-waarde.

In de map vind je bijv. de volgende vertaling terug tussen grayscale- en RGB-waarden. Deze map kan verschillen als je een andere afbeelding gebruikt of als je andere RGB-kleuren wilt gebruiken.

Grayscale-waarde	RGB-waarde
32	(0, 48, 80)
96	(218, 20, 21)
160	(112, 150, 160)
224	(250, 227, 173)

Voor ieder GrayscalePixel-object in je grayscale image bepaal je nu welke key-waarde het dichtste de greyscale-waarde van het GrayscalePixel-object benadert. De bijhorende RGBPixel gebruik je dan in de nieuwe afbeelding. Maak weer gebruik van een stream en lambda's om de nieuwe afbeelding te genereren. Schrijf de gegeneerde afbeelding weg mbv de methode writeImage in de klasse ImageWriter.

Maak nu een selfie van jezelf (of je huisdier,…) en laat je programma los op je eigen foto's!