Oefeningen Hoofdstuk 4

Maak een In IntelliJ een module aan met naam “H4”.

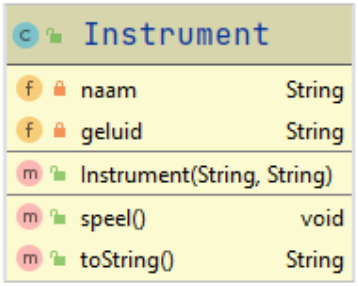
Per oefening maak je een aparte package. Voor oefening 1 geef je deze als naam “be.pxl.h4.oef1”.

Oefening 1 - PXL BAND

In deze oefening stel je een muziekgroep samen. We maken de klassen *Band*, *Muzikant* en *Instrument*. Deze klassen gebruiken elkaar om de functionaliteit van het programma uit te voeren. Hergebruik zoals steeds zoveel mogelijk code die je al hebt!

Instrument

Maak een klasse Instrument op basis van onderstaand UML diagram.

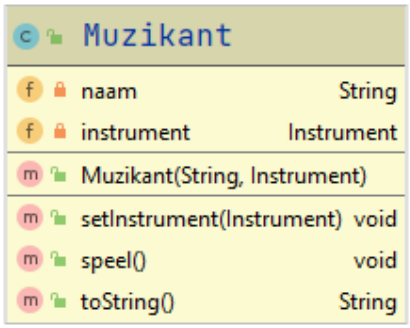


Enkele opmerkingen:
- de methode *speel()* print het geluid van het instrument af in de console.
- de methode *toString()* geeft enkel de naam van het instrument terug.

Maak een klasse BandApp met een main-methode waarin je een Instrument-object aanmaakt en alle methoden even uittest.

Muzikant

Maak een klasse Muzikant op basis van onderstaand UML diagram.



- de methode *speel()* moet de *speel()*-methode van het instrument oproepen.
- de methode *toString()* geeft de gegevens van de muzikant als volgt:

```
Sam [Gitaar]
```

Maak in de main-methode een Muzikant-object aan en test alle methoden uit.

Band

1. Maak een klasse Band met de eigenschappen *naam* (String) en *leden* (array van *Muzikant* objecten).
2. Voorzie een constructor waarmee je een Band kan aanmaken door een waarde voor beide eigenschappen mee te geven.
3. Maak ook hier een *speel()* methode. Aan deze methode moet je een parameter *lengte* (int) meegeven. De methode print het volgende uit:

```
PXL-Digital in concert!
Boenk baf boenk baf
Pftoooooee
Bow bow bow
Bow bow bow
Wawawaaw
```

   Er wordt een aantal keer een **willekeurige** muzikant uit de array met *leden* geselecteerd, die dan zijn instrument speelt. Het aantal wordt hierbij aangegeven door de parameter *lengte*.

4. De *toString()* methode geeft de gegevens van de *Band* als volgt terug:

```
PXL-Digital
Sam [Gitaar]
Heidi [Drum]
Ingrid [Bas]
Nele [Klarinet]
Francis [Triangel]
```

   Maak hiervoor gebruik van de StringBuilder klasse.

BandApp

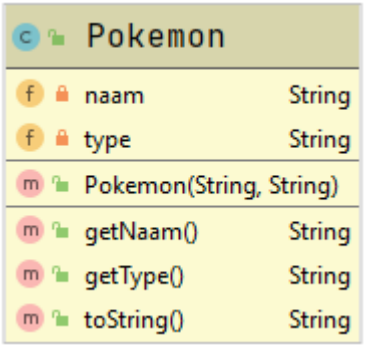
Breidt je main methode uit zodat er een aantal Muzikanten aangemaakt worden met verschillende Instrumenten. Sla ze op ze in een array en maak dan een Band aan met een originele naam en de array van Muzikanten. Print de gegevens van de band uit met de *toString()* en laat de band dan spelen voor 20 tellen.

Oefening 2 – Pokémon

In deze oefening simuleer je enkele Pokemon-trainers die op jacht gaan om Pokemon te vangen. We maken klassen voor een *Pokemon*, een *Pokedex* om de gevangen Pokemon in op te slaan en natuurlijk ook een *Trainer* klasse. Hergebruik zoals steeds zoveel mogelijk code die je al hebt!

Pokemon

Maak een klasse Pokemon op basis van onderstaand UML diagram.

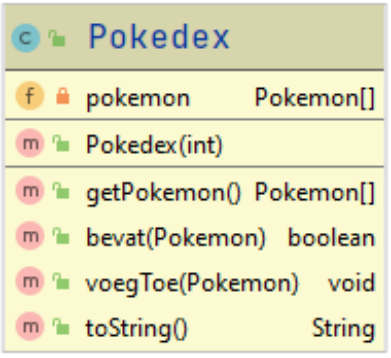


De methode *toString()* geeft het volgende terug:

```
Charmander (Fire)
```

Pokedex

De klasse Pokedex heeft als eigenschap een lijst (array) van Pokemon objecten. Aan de constructor kan je een *int* meegeven als parameter, die de grootte van de array bepaalt. Zorg er dus voor dat het aanmaken van de array pas in de constructor gebeurt.



De methode ***bevat()*** krijgt een Pokemon object als parameter en geeft *true* of *false* terug afhankelijk van of de gegeven Pokemon reeds aanwezig is in de lijst of niet. Check hiervoor de naam van de Pokemon. Hou er rekening mee dat er ook *null* waarden in de array kunnen zitten.

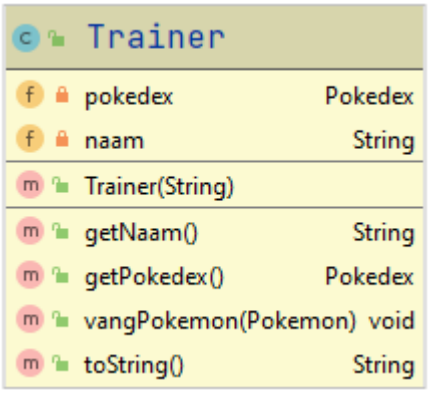
voegToe() voegt de gegeven Pokemon toe op de eerste beschikbare lege plaats (*null*) in de lijst.

De ***toString()*** geeft een String terug met een oplijsting van de Pokemon, in volgend formaat:

```
- Oddish (Grass)
- Diglett (Ground)
- Bulbasaur (Grass)
- Rattata (Normal)
```

Trainer

De klasse Trainer heeft een String en een Pokedex object als eigenschappen.



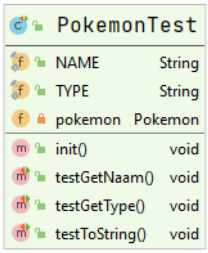
De constructor krijgt enkel de naam als parameter, maar maakt wel standaard een Pokedex object van grootte 10 (*zie constructor Pokedex*).

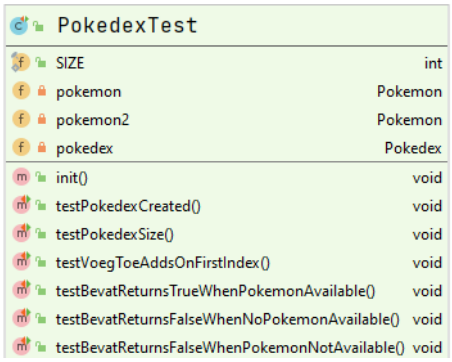
De methode *vangPokemon()* voegt de gegeven Pokemon toe aan de Pokedex, maar wel alleen als deze daar nog niet in zit. Gebruik bestaande methodes waar mogelijk.

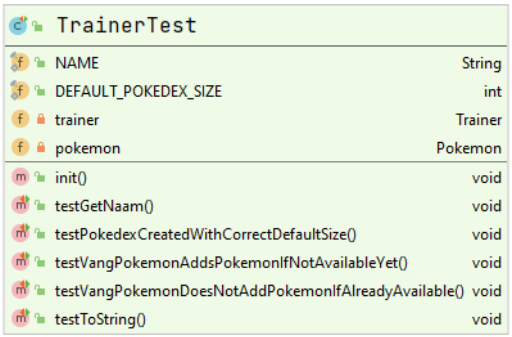
De *toString()* methode geeft enkel de naam van de trainer terug.

Unit tests

Er zijn enkele unit tests voorzien die je kan toevoegen en uitvoeren, om te checken of alles werkt zoals beschreven staat. Wanneer dit het geval is, kan je verder naar het laatste onderdeel van deze oefening.







PokemonApp

Als alles naar behoren werkt, kan je de PokemonApp uit de startbestanden kopiëren en uitvoeren. Controleer zeker of de package naam correct is, als er zich fouten zouden voordoen.

De output zou er ongeveer als volgt moeten uitzien:

(door de random factor zal jouw output niet exact hetzelfde zijn)

```
Sam heeft volgende Pokemon gevangen:
- Charmander (Fire)
- Squirtle (Water)
- Ekans (Poison)

Ash heeft volgende Pokemon gevangen:
- Caterpie (Bug)
- Ekans (Poison)
- Diglett (Ground)
- Bulbasaur (Grass)
- Rattata (Normal)

Brock heeft volgende Pokemon gevangen:
- Sandshrew (Ground)
- Charmander (Fire)
- Oddish (Grass)
- Squirtle (Water)

Misty heeft volgende Pokemon gevangen:
- Ekans (Poison)
- Jigglypuff (Fairy)
- Sandshrew (Ground)
- Diglett (Ground)
```

Oefening 3

In deze oefening pakken we het wat anders aan. In plaats van alle details te geven over de klassen, eigenschappen en methoden die jullie moeten implementeren, laten we dit deze keer over aan jou. We beschrijven de applicatie die we als resultaat willen, de rest probeer jij zo goed mogelijk op een object georiënteerde manier op te lossen.

Omschrijving: Bouw een applicatie waarmee we een lijst van lectoren kunnen bijhouden en afprinten. Van elke lector willen we bijhouden wat zijn/haar voor- en achternaam is, welke vakken hij/zij geeft en het aanstellingspercentage van de lector. Elk vak bevat een naam, een (vak)code en het aantal studiepunten.

Restricties:

- Een vak kan nooit meer dan 18 studiepunten zijn.
- Een lector mag nooit meer dan 100% aanstellingspercentage hebben.
- Een lector mag niet meer dan 5 vakken geven.
- Bij ongeldige bewerkingen die bovenstaande restricties overtreden, moet er een gepaste melding getoond worden.

Voorbeeld output:

```
Leraar Daems Greta is aangesteld voor 70%
Volgende vakken behoren tot het takenpakket:
     41TIN1130    Java Essentials          6
     41TIN1300    IT Essentials            6
     42TIN1220    .Net Advanced            3
     42TIN1230    Java Advanced            3
     43AON3120    Programming Expert       3
```

Oefening 4

In deze oefening worden een aantal klassen aangemaakt om persoonsgegevens en adressen bij te houden zodat we uiteindelijk 2 personen kunnen laten huwen en onder andere met 1 bewerking ervoor zorgen dat ze op hetzelfde adres wonen.

De oefening wordt stapsgewijs opgebouwd, er zijn unit tests voorzien om de toegevoegde code te testen.

Vermijd om 2 keer dezelfde code te schrijven en hergebruik zoveel mogelijk code die je al hebt!

klasse Gemeente

1. Maak een klasse Gemeente met de volgende kenmerken: postcode (int), gemeenteNaam (String). De kenmerken mogen van buiten de klasse niet toegankelijk zijn.
2. Voorzie een constructor om een gemeente te creëren door een waarde mee te geven voor beide eigenschappen.
3. Voorzie getters en setters om de waarde te kunnen opvragen en wijzigen.
   - Voorzie een controle op de postcode, deze moet 4 posities lang zijn. Indien ze te lang is worden de eerste 4 posities genomen. Indien ze te kort is wordt ze achteraan aangevuld met nullen.
4. Voorzie een methode toString() die de gemeentegegevens teruggeeft zoals: `3500 Hasselt`

> **Extra**
>
> Zorg ervoor dat de gemeentenaam omgevormd wordt volgens volgende regels:
> - Eerste letter wordt een hoofdletter.
> - Verder worden alle letters omgezet naar kleine letters behalve tekens na een spatie of '-', deze blijven zoals ze ingevoerd werden.
> - Cijfers worden verwijderd.
>
> Voorbeeld: hE78rk-7dE-Stad
> wordt omgevormd tot: Herk-de-Stad

Voeg de testklasse GemeenteTest toe (voorzien in startbestanden) en test de klasse Gemeente.

klasse Adres

1. Maak een klasse Adres met de volgende kenmerken: straat (String), huisNummer (String), gemeente (Gemeente). De kenmerken mogen van buiten de klasse niet toegankelijk zijn.
2. Voorzie een constructor om een adres te creëren door 4 waarden in te geven, vb "Sparreweg", "18", 3500, "Hasselt".

3. Voorzie getters en setters om de waarde van straat en huisNummer te kunnen opvragen en wijzigen. Voorzie een get-methode om de gemeente op te halen. Je mag GEEN set-methode voor de gemeente aanmaken.
4. Voorzie een methode toString() die de gemeentegegevens teruggeeft zoals:

```
Sparreweg 18
3500 Hasselt
```

Voeg de testklasse AdresTest toe (voorzien in de startbestanden) en test de klasse Adres.

klasse Persoon

1. Maak een klasse Persoon met de volgende kenmerken: naam (String), voornaam (String), geboortedatum (LocalDate) en adres (Adres). De kenmerken mogen van buiten de klasse niet toegankelijk zijn.
2. Voorzie een constructor om een persoon aan te maken met volgende parameters:
   "Vervoort", "Lies", 29, 11, 1990, "Sparreweg", "18", 3500, "Hasselt".
   Voorzie een tweede constructor om een persoon aan te maken door naam, voornaam, geboortedatum en adres mee te geven.
   Roep op een zinvolle manier 1 constructor op vanuit de andere.
3. Voorzie get- en set-methoden voor naam en voornaam en adres.
   Voorzie een get-methode voor geboortedatum. Je mag GEEN set-methode voorzien voor de geboortedatum.
4. Voorzie een methode toString() die de persoonsgegevens als volgt teruggeeft:

```
Lies Vervoort
Sparreweg 18
3500 Hasselt
```

Voeg de testklasse PersoonTest toe (voorzien in de startbestanden) en test de klasse Persoon.

klasse Huwelijk

1. Maak een klasse Huwelijk waarin je 2 partners en een huwelijksdatum bijhoudt.
2. De initialisatie van een huwelijk-object gebeurt door opgave van 2 personen en 3 getallen (waaruit de datum samengesteld wordt).
   Wijzig het adres van de tweede partner in het adres van de eerste partner.
   Geef een melding van initialisatie zoals in volgend voorbeeld: `Niels Bex en Lies Vervoort zijn gehuwd op 05 mei 2021. Proficiat!`
3. Zorg ervoor dat van een huwelijk de 2 partner-objecten en het datum-object kunnen opgevraagd worden.
4. Maak een methode adresWijziging() die bv. volgende waarden binnenkrijgt als parameter: “Lentestraat”, “15C”, 3500, “Hasselt” en stel het adres van beide partners in op dit nieuwe adres.
5. Voorzie een methode print() die de gegevens als volgt afdrukt:

```
Lies Vervoort
Lentestraat 15C
3500 Hasselt
Geboren op 29 november 1990

Jo Bex
Lentestraat 15C
3500 Hasselt
Geboren op 12 juni 1992

Het huwelijk vond plaats op 05 mei 2021
```

Voeg de testklasse HuwelijkTest toe (voorzien in de startbestanden) en test de klasse Huwelijk.

Klasse HuwelijkApp

Maak in de klasse HuwelijkApp een main methode aan. Maak in de main-methode 2 Persoon-objecten aan en test de methoden uit. Wijzig de gemeentenaam van het adres van een persoon.

Maak een Huwelijk-object aan. Vraag het huwelijksjaar van het huwelijk op. Druk de persoonsgegevens van `partner1` af.

Doe tenslotte een adreswijziging. Druk alle gegevens van het huwelijk af.

Extra oefeningen Hoofdstuk 4

Extra oefening - Beeldverwerking

In deze oefening creëren we enkele klassen om een afbeelding in op te slaan. Nadien kunnen jullie **enkele *utility* klassen toevoegen** die een echte afbeelding op jullie harde schijf kunnen inladen en opslaan. De *utility* klassen vinden jullie tussen de startbestanden.

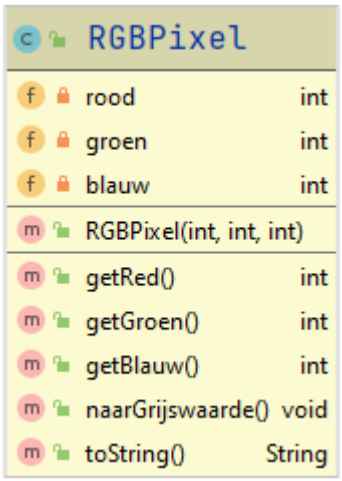
Daardoor kunnen we dan, indien je onderstaande klassen juist implementeert, bewerkingen uitvoeren op individuele pixels in de afbeelding en effecten toevoegen aan jouw foto's.

Neem de tijd om de hele opgave stap voor stap te volgen!

Deze oefening is extra en het bewerken van beelden valt buiten de 'te kennen' leerstof! Je moet kunnen werken met 2-dimensionale arrays.

Klasse RGBPixel

Maak de klasse RGBPixel op basis van onderstaand UML diagram.



De eigenschappen stellen de verschillende kleurwaarden voor binnen 1 pixel. De kleurwaarden liggen steeds tussen 0 en 255. Dit moet je niet checken in de klasse.

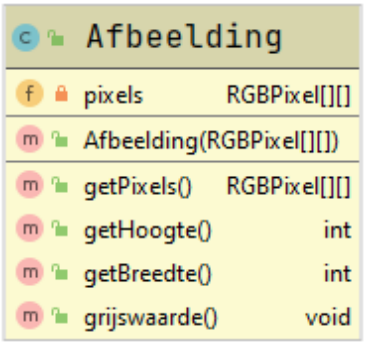
De methode *naarGrijswaarde()* neemt het gemiddelde van de *rood, groen* en *blauw* waarde en slaat het resultaat op in alle drie eigenschappen. Dit verandert de pixel in een grijze pixel. Het gemiddelde moet naar beneden afgerond worden/

De methode *toString()* geeft de gegevens van de klasse als volgt terug:

```
(255, 127, 127)
```

Klasse Afbeelding

Maak de klasse Afbeelding op basis van onderstaand UML diagram. De pixels worden, net als in een echte afbeelding, opgeslagen in een 2D array.



De methode *getHoogte()* geeft het aantal rijen terug, *getBreedte()* het aantal kolommen.

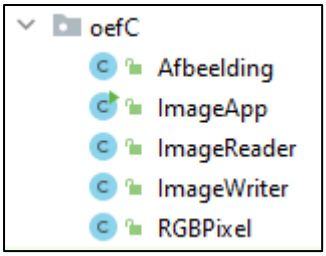
De methode *grijswaarde()* roept voor elke pixel de *naarGrijswaarde()* methode op.

Unit tests

Voer de voorziene unit tests uit, om te controleren of de bovenstaande klassen correct zijn geïmplementeerd. Tot dit niet het geval is, kan je de laatste stap van deze oefening niet uitvoeren. In het vorige hoofdstuk hebben we besproken hoe je tests kan uitvoeren.

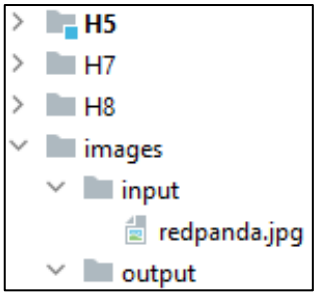
Beelden inladen en opslaan

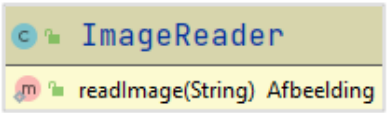
Wanneer je zeker bent dat *Afbeelding* en *RGBPixel* correct werken, kan je de twee *utility* klassen toevoegen aan je project. **Voeg dus de klassen *ImageReader* en *ImageWriter* toe** aan dezelfde package als waar je *Afbeelding* en *RGBPixel* aangemaakt hebt. (*be.pxl.h5.oefC*)

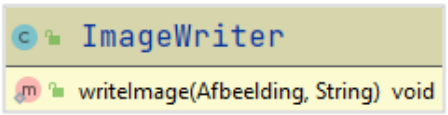


Maak ook een *ImageApp* klasse aan, met daarin een main-methode. Het resultaat moet er als volgt uit zien.

Kopieer de folder *images* uit de startbestanden en plaats deze in je hoofdproject (bv. *JavaEssentials*).



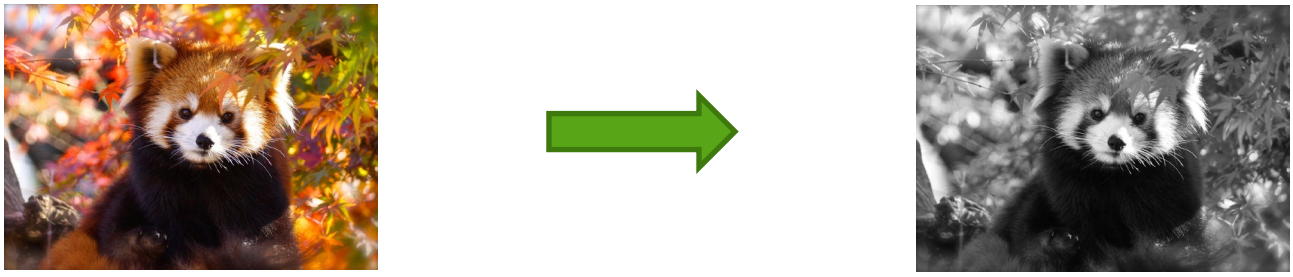




Uit de UML diagrammen van de twee toegevoegde klassen, kan je opmaken hoe je er mee kan werken. Let er op dat ze enkel een klasse methode bevatten!
De String parameters bevatten de bestandsnamen van de afbeeldingen die ingelezen of opgeslagen moeten worden.
readImage zoekt de bestanden in de folder ***images/input***
writeImage slaat de bestanden op in ***images/output***

De afbeelding in bovenstaande voorbeeld spreek je dus aan met de String ***"redpanda.jpg"***.

Probeer nu een afbeelding in te laden, voer er de methode *grijswaarde()* methode op uit en sla het resultaat weer op.



Extra bovenop de extra oefening!! Meer effecten!

Je hebt nu alle nodige code en tools in handen om heel wat andere leuke effecten uit te proberen op je foto's. Je kan eender welke waarde aan de pixels geven en nadien het resultaat opslaan d.m.v. de *utility* klasse.
Hieronder vind je enkele voorbeelden en wat uitleg over hoe ze te implementeren. Je kan zeker ook zelf op zoek gaan naar filters, of gewoon wat uitproberen.

Je kan hiervoor aanpassingen doen in zowel de *Afbeelding* als de *RGBPixel* klasse.

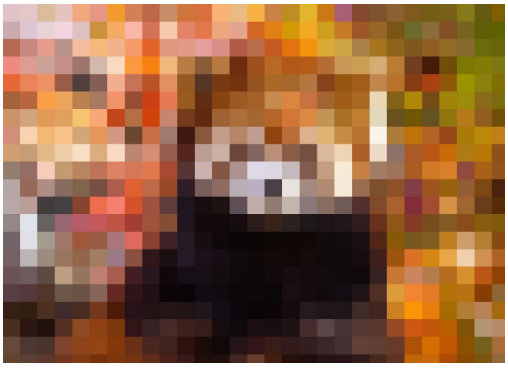
Laat je creativiteit dus maar de vrije loop!

INVERTEREN

Kleuren inverteren houdt in dat je alle kleuren (*rood, groen, blauw*) vervangt door hun verschil met 255. (*255 = de maximale kleurwaarde*) Zo zal een waarden van 50 bijvoorbeeld veranderen in 205. (*255-50*)

VERDUISTEREN

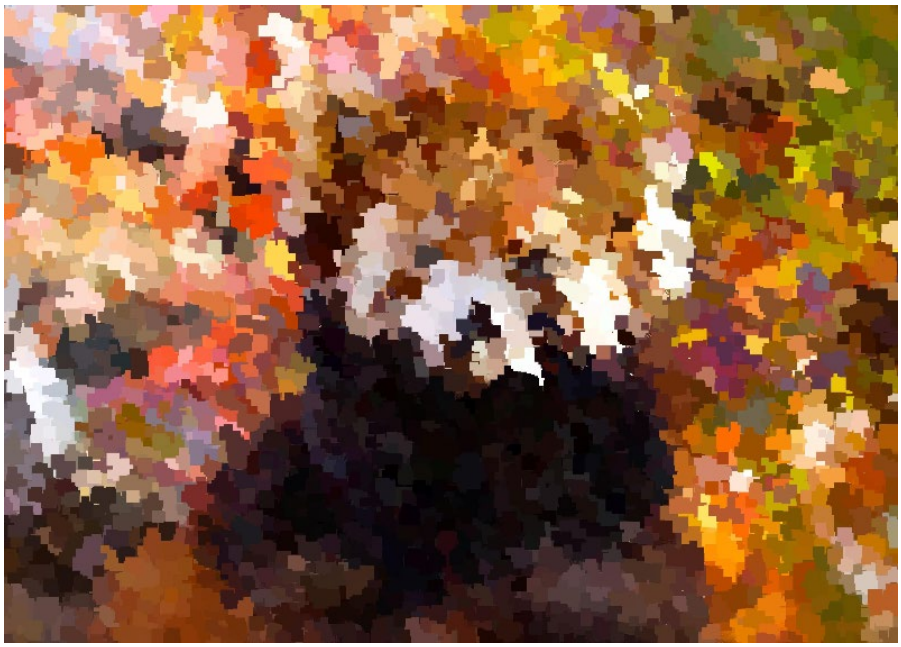
Je kan de kleurwaarden van elke pixel vermenigvuldigen met een factor kleiner dan 1 (*bv. 0.6 in het voorbeeld*) om de hele afbeelding donkerder te maken.



PIXELATE

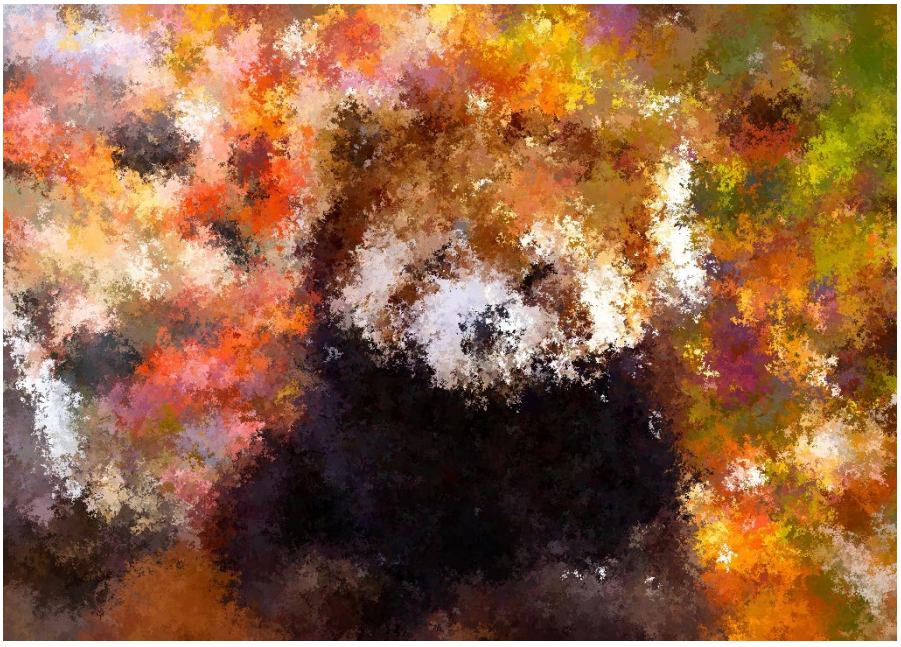
Als je in subregio's van pixels (*bv. 80x80*) alle kleurwaarden uitmiddelt en deze gemiddelden voor alle pixels in die subregio instelt, krijg je een pixel blockeffect. Je gaat dus alle *rood*-waarden optellen en delen door het aantal pixels, waarna je dat resultaat instelt als *rood* waarde voor alle pixels in de regio. Hetzelfde doe je voor *groen* en *blauw*.

Ter inspiratie, om nog wat ideeën op te doen:



DOTTED

- Selecteer random pixel
- Kopieer kleur in vierkant rond deze pixel
- Grootte vierkant ook random, met maximum
- Herhaal 20000 keer



RANDOM PATH

- Selecteer random pixel
- Laat vanuit deze pixel een pad in willekeurige richtingen lopen, waarbij je elke pixel op het pad dezelfde kleur geeft als de originele pixel
- Herhaal 100000 keer, lengte van pad 100 pixels

Ook online kan je zeker nog voorbeelden vinden van andere filters of bewerkingen die je op een collectie van pixels kan uitvoeren.

Laat je mooiste resultaten zeker eens zien aan je lector!