BILLEDREDIGERING (IMAGES)

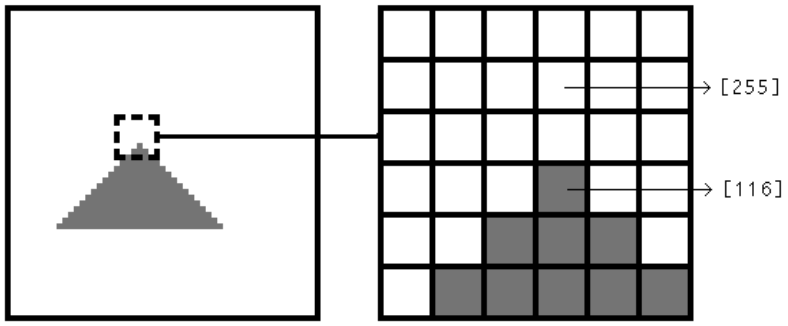
I dette projekt skal I redigere gråtonebilleder ved hjælp af nogle af de teknikker, der blev beskrevet i en forelæsning.

Hent BlueJ-projektet ***Image*** (zip) og husk at pakke det ud, før I går i gang. Projektet indeholder fire klasser:

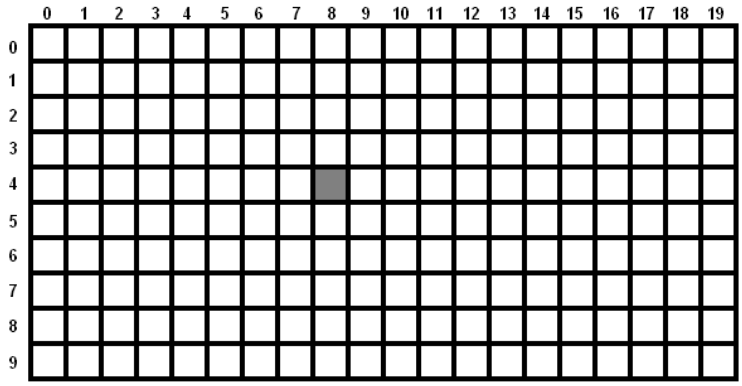
- ***Image*** og ***Pixel*** klaserne bruges til at repræsentere et billede og dets pixels (som beskrevet herunder).
- ***Filters*** klassen indeholder metoder til at manipulere billeder (f.eks. gøre billedet mørkere eller spejle det). Konstruktørerne og metoderne i denne klasse har pt. tomme kroppe (bortset fra at metoderne indeholder et dummy ***return*** statement så de kompilerer korrekt). Det er jeres opgave at implementere disse metoder, dvs. udfylde kroppene på passende vis. Parametrenes navne samt den kommentar, der er knyttet til hver parameter, fortæller, hvad den pågældende parameter bruges til.
- ***FiltersTest*** klassen indeholder et antal klassemetoder, som gør det let for instruktorerne (og jer selv) at tjekke, om de metoder, som I har implementeret i ***Filters*** klassen, fungerer korrekt.

Repræsentation af billeder

Et billede repræsenteres ved hjælp af pixels. Hver pixel indeholder en gråtone, som er specificeret via et heltal mellem 0 og 255. 0 er helt sort, og 255 er helt hvid. Tal derimellem angiver forskellige toner af grå. Herunder ses et simpelt billede, der forestiller en trekant. Til højre for billedet er en forstørrelse af et område af billedet, hvor man kan se de enkelte pixels, og hvilke talværdier de har.

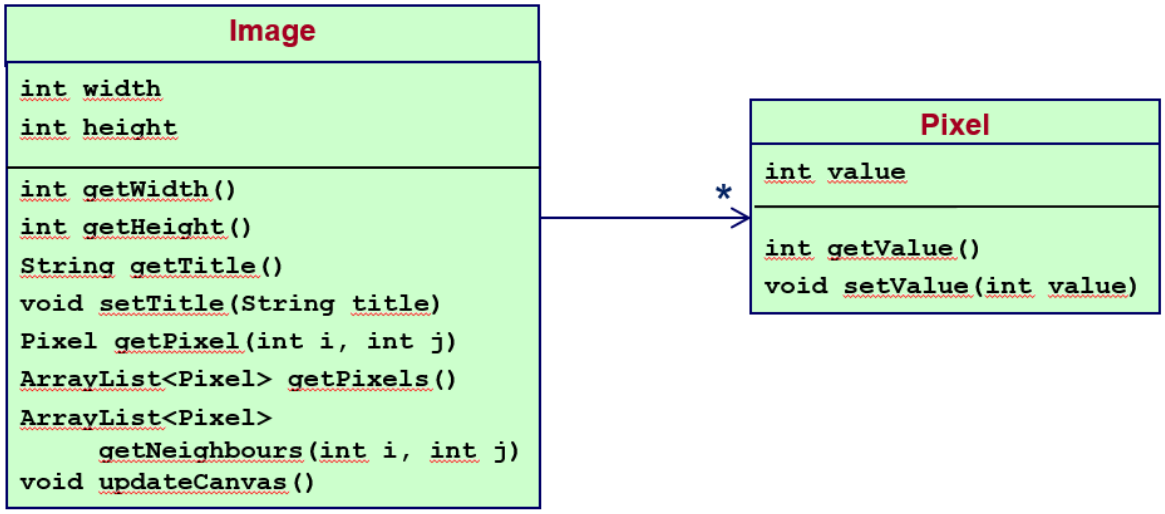


Billedets højde og bredde bestemmer det totale antal pixels i repræsentationen af billedet. Eksempelvis vil et billede, som er 20 pixels bredt og 10 pixels højt, indeholde 200 pixels:



De enkelte pixels kan tilgås via deres koordinater. Ovenfor er pixelen med koordinaterne (8, 4) markeret. Bemærk at koordinaterne begge starter med 0, og at anden koordinaten øges, når man går nedad.

Image og ***Pixel*** klasserne er specificeret i nedenstående UML-klassediagram.



Pixel klassen repræsenterer en pixel. Den er meget simpel, og indeholder kun to metoder, ***getValue*** og ***setValue***, til henholdsvis at aflæse og ændre gråtonen af den pågældende pixel. I må ikke ændre koden i denne klasse.

Image klassen repræsenterer et billede bestående af mange pixels, som repræsenteres via et 2-dimensionalt array af Pixels. Denne klasse er langt mere kompleks, men I kan heldigvis bruge den uden at forstå implementationen – på samme vis som I bruger klasser fra Java API'en. I må ikke ændre koden i denne klasse.

Image klassen har tre konstruktører, hvor man enten kan specificere:

- et filnavn (hvorefter billedet kopieres fra filen),
- en bredde, højde og titel (hvorefter man får et helt sort billede med den specificerede størrelse og titel),
- et Image objekt (hvorefter billedet bliver en kopi af dette).

Herudover er der yderligere to konstruktører (med en boolsk parameter ***visible***). Dem kan I ignorere (de bruges af test metoderne).

Opret et ***Image*** objekt ved hjælp af den første konstruktør og filnavnet ***"dog.png"***. Så skulle der gerne dukke et billede op af en hund. Filen ***dog.png*** ligger i projektmappen, og hvis I lægger andre billeder heri, kan I også bruge dem (forudsat at de er i et gængs billedformat, såsom PNG, JPG, BMP eller GIF).

Luk hundebilledet og opret et nyt ***Image*** objekt ved hjælp af den anden konstruktør. Giv det størrelsen 200 x 300 pixels og navnet ***"test"***. Du får da et billede, der er helt sort. Brug ***getPixel*** og ***getValue*** metoderne til at hente gråtonen på en af de midterste pixels i billedet. Brug dernæst ***setValue*** til at ændre gråtonen på pixelen til 255 (hvid). Læg mærke til, at der ikke sker noget med billedet. Det ændrer sig først, når I kalder metoden ***updateCanvas***, som sørger for at gentegne billedet med de opdaterede pixels. Derefter skulle I gerne se en hvid pixel midt i det sorte billede.

Testserveren bruges ikke i denne opgave. I stedet indeholder klassen ***FiltersTest*** en række klassemetoder, der tjekker jeres løsninger ved at vise det resultat, som jeres metode producerer ved siden af det resultat, som en korrekt metode producerer. I må ikke ændre koden i denne klasse.

Opgave 1 -brighten, darken, invert

Konstruktørerne

Implementér konstruktørerne i ***Filters*** klassen. Den første skal tage et ***Image*** objekt som parameter og gemme dette i feltvariabelen ***image***. Den anden skal lave et ***Image*** objekt ud fra filens ***picture.png*** og gemme dette i feltvariabel ***image***. Ved at bruge den sidste konstruktør, sparer i lidt tid under aftestningen af jeres metoder.

brighten

Implementér ***brighten*** metoden i ***Filters*** klassen. Metoden opererer på det billede, der er gemt i feltvariablen ***image***. Den har en heltalsparameter, hvis værdi lægges til gråtoneværdien af hver enkelt pixel i billedet. Herudover ændres titlen ved hjælp af ***setTitle*** metoden så denne bliver prefixet med "brightenX-", hvor X er værdien af parameteren. Når metoden er færdig returneres det opdaterede billede som metodens returværdi.

Original

brighten

Ved at kalde metoden ***getPixels*** på et ***Image*** objekt, får man en arrayliste med alle pixels i billedet. Brug denne til at løbe alle pixels igennem og ændre deres værdi. Husk at ændre titlen og husk at kalde ***updateCanvas*** til sidst, så det viste billede bliver opdateret.

Afprøv metoden på ***picture.png*** med parameteren ***50***. Gentag kaldet et antal gange og se, hvad der sker. Brug dokumentationen af ***Pixel*** klassen til at undersøge, hvad metoden ***setValue*** gør, hvis I kalder den med en parameter, der ligger uden for intervallet ***[0, 255]***. Jeres svar skrives i filen README.TXT, som I kan editere ved at dobbeltklikke på "papiret" i øverste venstre hjørne af BlueJ-projektets klassediagram.

Test ***brighten*** metoden ved hjælp af klassemetoden ***testBrighten*** i ***FiltersTest***. Testmetoden viser to billeder på skærmen. Det venstre er det billede, som jeres ***brighten*** metode producerer, mens det højre er det billede, som den bør producere. Hvis billederne afviger fra hinanden, skal I finde fejlen og rette den.

darken

Implementér ***darken*** metoden i ***Filters*** klassen. Metoden skal fungere på samme måde som ***brighten***, men gøre billedet mørkere og prefixe titlen med "darkenX-", hvor X er værdien af parameteren.

Afprøv metoden på ***picture.png***. Hvad sker der, hvis I først laver billedet lysere og derefter mørkere? Vil I så være tilbage til det billede, som I startede med? Svaret skrives i README.TXT.

Test metoden ved hjælp af klassemetoden ***testDarken*** i ***FiltersTest***.

invert

Implementér ***invert*** metoden i ***Filters*** klassen. Metoden skal fungere på samme måde som ***brighten*** og ***darken***, men invertere billedet og prefixe titlen med "invert-". Inverteringen foretages ved at erstatte hver pixels værdi med den "modsatte" værdi, dvs. at en pixel med værdien ***47*** får værdien ***255 - 47 = 208***. Sort bliver til hvid og omvendt.

Test metoden den ved hjælp af klassemetoden ***testInvert*** i ***FiltersTest***.

Opgave 2 -mirror, flip, rotate

mirror

Implementér ***mirror*** metoden i ***Filters*** klassen. Metoden skal spejle billedet omkring den lodrette midterakse. Titlen på det nye billede skal være titlen på det gamle prefixet med ”mirror-”.

For at spejle et billede sætter man værdien af pixel ***(i, j)*** i det nye billede lig med værdien af pixel ***(width-i-1, j)*** i det gamle billede, hvor ***width*** er bredden af billedet. Man kommer let til at overskrive pixels undervejs, og få et resultat i stil med det billede, der er vist til højre. Det er derfor nemmest at oprette et helt nyt billede, dvs. skabe en lokal variable med et nyt ***Image*** objekt (med samme størrelse som det oprindelige) og så indsætte gråtoneværdierne i det nye billede. Når man er færdig med at opdatere alle pixels, assignes billedet i den lokale variabel til klassens feltvariabel og denne returneres som metodens returværdi.

mirror

mirror (wrong)

Start med at lave en simpel implementation, der returnerer et nyt billede med samme størrelse, som det oprindelige billede (og med den modificerede titel). Der skulle så gerne dukke et sort billede op med samme størrelse som det oprindelige billede.

For at gennemløbe og behandle alle pixels kan man bruge en dobbelt ***for*** løkke, hvor den inderste løkke gennemløber alle x-koordinater og den yderste alle y-koordinator (eller omvendt). For at få fat i pixelen på en given koordinat, anvendes ***getPixel*** metoden i ***Image*** klassen. Når man har fat i en pixel kan dens gråtoneværdi aflæses/ændres ved hjælp af ***getValue*** og ***setValue*** metoderne i ***Pixel*** klassen.

Test metoden ved hjælp af klassemetoden ***testMirror*** i ***FiltersTest***.

flip

Implementér ***flip*** metoden i ***Filters*** klassen. Metoden skal fungere på samme måde som ***mirror***, men spejle billedet omkring den vandrette midterakse og prefixe titlen med ”flip-”.

Husk at færdiggøre metodens kommentar.

Test metoden ved hjælp af klassemetoden ***testFlip*** i ***FiltersTest***.

rotate

Implementér ***rotate*** metoden i ***Filters*** klassen. Metoden skal rotere billedet 90 grader i urets retning. Titlen på det nye billede skal være titlen på det gamle prefixet med ”rotate-”.

Start med at lave et nyt billede, der er lige så bredt, som det oprindelige billede var højt, og lige så højt, som det oprindelige billede var bredt (og med den modificerede titel).

Herefter skal de enkelte pixelværdier kopieres, ligesom i ***mirror*** og ***flip***. Nu skal værdien af pixel i ***(i, j)*** i det nye billede sættes lig med værdien af pixel ***(j, width-i-1)*** i det gamle billede, hvor ***width*** er bredden på det nye billede.

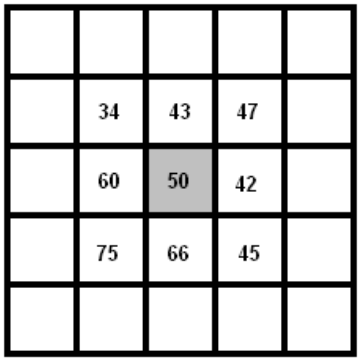
Test metoden ved hjælp af klassemetoden ***testRotate*** i ***FiltersTest***.

Overvej om I kan implementere ***mirror*** og ***flip*** uden at oprette et nyt billede. Samme spørgsmål for ***rotate***. Svaret skrives i README.TXT.

Opgave 3 - blur

blur

For at gøre et billede mere uskarpt (blurred) laver man et nyt billede, hvor gråtoneværdien for den enkelte pixel er beregnet som et gennemsnit af naboværdierne (i det oprindelige billede). Naboerne til en pixel, kan findes ved hjælp af metoden ***getNeighbours***, som returnerer en arrayliste af de omkringliggende pixels (inklusiv den pixel der angives af kaldets parametre). Bemærk, at pixels langs kanterne kun har 6 naboer, mens pixels i hjørnerne har 4.



$$\frac{34 + 43 + 47 + 60 + 50 + 42 + 75 + 66 + 45}{9} = 51$$

Implementér hjælpemetoden ***average*** i ***Filters*** klassen. Metoden tager to heltal ***i*** og ***j*** som parametre og returnerer et heltal, som er gennemsnitsværdien af de omkringliggende pixels (fundet ved hjælp af ***getNeighbours***).

Implementér ***blur*** metoden i ***Filters*** klassen. Metoden skal lave billedet uskarpt, hvor hver enkelt pixels gråtoneværdi bestemmes ved hjælp af ***average***. Titlen på det nye billede skal være titlen på det gamle prefixet med ”blur-”.

Test metoden ved hjælp af klassemetoden ***testBlur*** i ***FiltersTest***.

Opgave 4 -noise, resize, rotateAC, increaseContrast
(til dem, der har mod på mere)

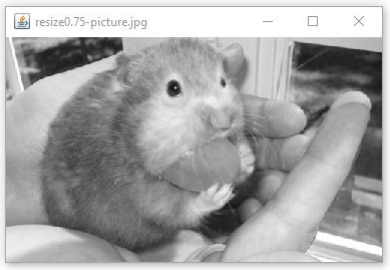
noise

Implementér ***noise*** metoden i ***Filters*** klassen. Metoden skal sløre billedet ved at tilføje støj. Metoden skal som parameter tage et heltal ***amount***, og lægger en tilfældig værdi i intervallet ***[–amount, amount]*** til gråtoneværdien af hver enkelt pixel i billedet. Titlen på det nye billede skal være titlen på det gamle prefixet med "noiseX-", hvor X er værdien af parameteren.

Afprøv metoden på ***picture.png*** (med forskellige støjgrænser).

Test metoden ved hjælp af klassemetoden ***testNoise*** i ***FiltersTest***. Denne fungerer lidt anderledes end de øvrige testmetoder. Da ***noise*** metoden er non-deterministisk, er der ikke et entydigt resultat. Testmetoden viser derfor det billede, som jeres ***noise*** metode producerer sammen med det originale billede, mens det tjekkes, at middelværdien af den tilføjede støj ligger tæt ved det forventede, og at alle de forventede støjværdi bruges et antal gange, der ligger tæt ved det forventede.

resize



resize

Et billede med ***width x height*** pixels kan skaleres med en faktor (***factor*** af type ***double***).

Herved fås et nyt billede med ***width*factor x height*factor*** pixels.

Værdien af pixel ***(i, j)*** i det nye billede sættes lig med værdien af pixel ***(i/factor, j/factor)*** i det oprindelige billede, idet de to sidstnævnte koordinater dog først nedrundes til et heltal ved at skrive (int) foran dem.

Implementér ***resize*** metoden i ***Filters*** klassen. Metoden skal skalere billedet med den angivne faktor. Titlen på det nye billede skal være titlen på det gamle prefixet med "resizeX-", hvor X værdien af parameteren.

Afprøv metoden på ***picture.png*** (med forskellige skaleringsfaktorer).

Test metoden ved hjælp af klassemetoden ***testResize*** i ***FiltersTest***.

rotateAC

Implementér ***rotateAC*** metoden i ***Filters*** klassen. Metoden skal rotere billedet 90 grader mod urets retning (anti-clockwise) og prefixe titlen med "rotateAC-".

Husk at færdiggøre metodens kommentar.

Test metoden ved hjælp af klassemetoden ***testRotateAC*** i ***FiltersTest***.

increaseContrast (svær)

Et billede med stor forskel på hvid og sort har *høj kontrast*. Et ensfarvet billede har ingen kontrast og maksimal kontrast opnås ved tilfældig støj.

I denne opgave skal I skrive en metode, der øger kontrasten på et billede. Intuitionen er, at de sorte farver skal gøres mere sorte og de hvide farver gøres mere hvide.

Lad $input \in [0, 255]$ være input heltallet (dvs. gråtoneværdien af en given pixel). Vi vil nu mappe $input$ til et tal x i intervallet $[-1, 1]$, hvor -1 svarer til helt sort og 1 svarer til helt hvid.

Dette kan gøres på følgende vis:

$$x = \frac{2 \cdot input}{255} - 1 \qquad \in [-1,1]$$

Vi ønsker nu at opløfte x i en passende potens. Dog er det upraktisk at negative værdier af x skifter fortegn afhængig af eksponenten. Derfor vil vi huske, om x var negativ eller positiv (dvs. gemme $\text{signum}(x)$) og i stedet potensopløfte $y = |x| \in [0,1]$ (altså absolutværdien af x).

Vi ønsker nu at forøge kontrasten med $amount > 0$. Den naive løsning vil være at sætte den nye værdi $y' = y^{amount}$. Dette giver dog ikke så god mening, da $amount = 0$ bør være identitets-afbildningen, mens det her er $amount = 1$.

Vi beregner i stedet $p = e^{-amount}$ og påstår, at dette er en god eksponent til y. Hvis $amount = 0$, får vi $p = 1$, og dermed identitetsafbildningen som ønsket. Positive værdier af $amount$ vil give $p < 1$, hvilket medfører at $1 \geq y' = y^p > y$, for alle y. Det er denne egenskab, der gør eksponential-funktionen til et passende valg. Uanset værdien af $amount$, får vi stadig et tal y' i intervallet $[0,1]$.

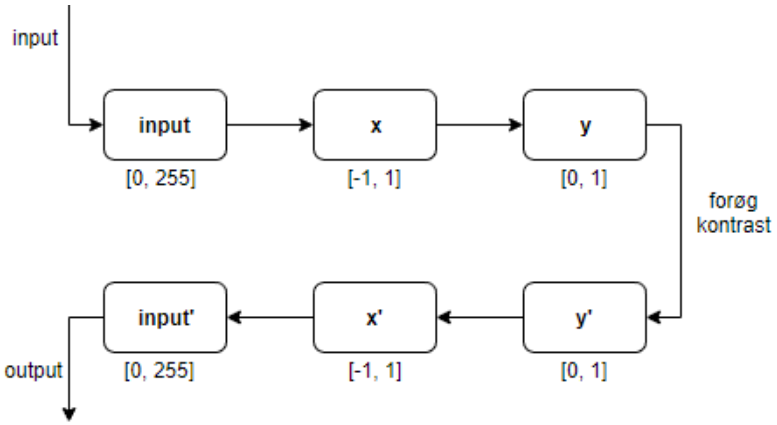
Vi skal nu mappe $y' \in [0,1]$ tilbage til intervallet $[-1, 1]$. Altså beregne $x' = \text{signum}(x) \cdot y'$. Til sidst vil vi gerne mappe x' tilbage til intervallet $[0,255]$, så vi får en pixelværdi. Dette kan gøres på følgende vis:

$$input' = \frac{x' + 1}{2} \cdot 255 \qquad \in [0,255]$$

<u>For at opsummere</u>:

Input er et heltal $input \in [0,255]$ og en parameter $amount \in \mathbb{R}$

- Beregn $x = \frac{2 \cdot input}{255} - 1$
- Beregn $y = |x|$
- Beregn $p = e^{-amount}$ og $y' = y^p$
- Beregn $x' = \text{signum}(x) \cdot y'$
- Output $input' = \frac{x'+1}{2} \cdot 255$



Implementér ***increaseContrast*** metoden i ***Filters*** klassen. Metoderne ***Math.abs***, ***Math.exp***, ***Math.pow*** og ***Math.signum*** vil være nyttige. Titlen på det nye billede skal være titlen på det gamle prefixet med "contrastX-", hvor X værdien af parameteren.

Afprøv metoden på ***picture.png*** (med forskellige værdier af ***amount***).

Er ***increaseContrast(amount)*** og ***increaseContrast(-amount)*** hinandens inverse metoder? Argumentér for dit svar i README.txt filen. Lav gerne eksperimenter for at verificere din hypotese.

Test metoden ved hjælp af klassemetoden ***testIncreaseContrast*** i ***FiltersTest***.