Laboration 5

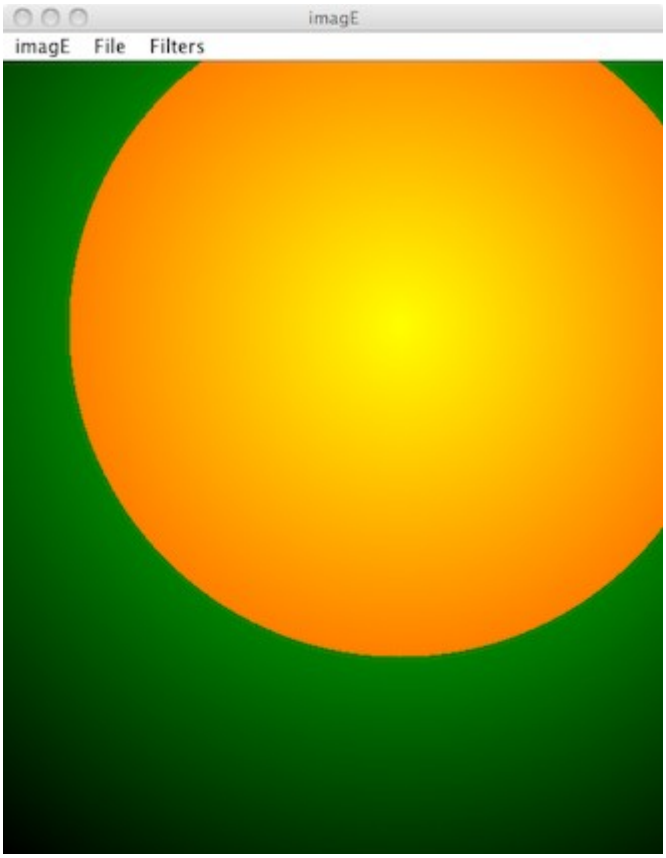
1. Syfte

I denna laboration ska vi återigen behandla digital transformering av data; denna gång gäller det bilder. Återigen får ni grundstommen till ett program och ska komplettera detta med ett par små klasser. Och slutligen, återigen är den mängd kod ni ska skriva inte stor; när man väl förstått hur det hela hänger ihop är det bara fråga om några tiotal rader.
Syftet med denna laboration är att få övning i att använda fler-dimensionella fält.

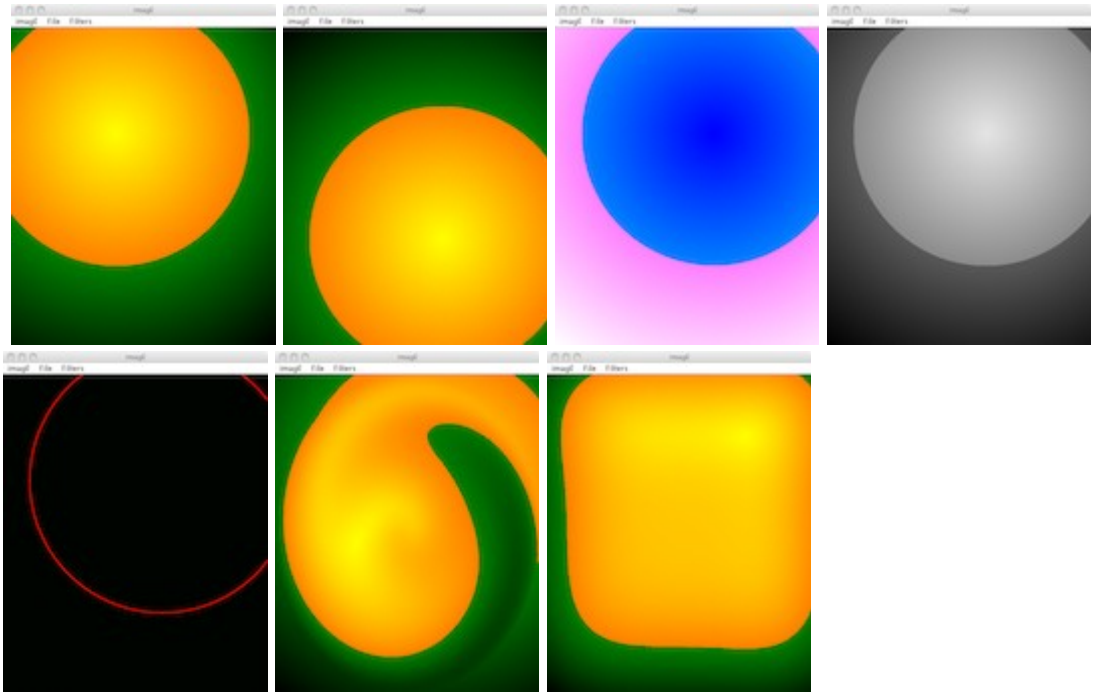
2. Förberedelser

Börja med att ladda ner lab5.zip till er arbetskatalog; uppackning ger en katalog lab5 med ett antal filer.

Ladda in filen Main.java i jGrasp. Kompilera och kör programmet. Ett fönster öppnas med nedanstående utseende:



Detta är ett enkelt program för att manipulera bilder genom att använda olika filter. Huvuddelen av programmets fönster visar en initial bild. Interaktionen med programmet sker helt genom de tre menyerna. imagE-menyn erbjuder två alternativ: att få upp en informationsruta om programmet eller att avsluta detsamma. File-menyn ger möjlighet att läsa in en ny bild eller spara den befintliga. Den mest intressanta menyn är Filter-menyn. Här erbjuds ett antal olika sätt att filtrera bilden. Resultatet av användning av de olika filtren visas i nedanstående bilder:



Vi uppmanar er att experimentera litet med programmet. Några exempel på bilder bifogas i zip-filen, men det går också bra att använda egna bilder i formatet JPEG.

3. Snabbkurs i digital bildrepresentation

En digital bild är en rektangulär matris av pixlar (eng. pixels, för picture elements). För varje pixel beskrivs dess intensitet i de tre färgkomponenterna rött, grönt och blått. Dessa intensiteter är heltal i intervallet 0 till 255 (åtta bitar). En pixels färg beskrivs alltså fullständigt av ett RGB-värde, bestående av dessa tre heltal. Några exempel på färger är följande:

Färg	RGB-värde	Färg	RGB-värde
Svart	(0,0,0)	Vitt	(255,255,255)
Rött	(255,0,0)	Cyan	(0,255,255)
Grönt	(0,255,0)	Magenta	(255,0,255)
Blått	(0,0,255)	Gult	(255,255,0)

Detta innebär att vi i ett program kan representera en bild med en variabel av typen `int[][][]`, dvs ett fält med tre index. För en bild med bredden WIDTH pixlar och höjden HEIGHT pixlar kan variabeln lämpligen skapas med satsen

```
int[][][] imArray = new int[WIDTH][HEIGHT][3];
```

För att förtydliga ytterligare: heltalet `imArray[37][124][1]` anger intensiteten av grönt i den pixel som återfinns 37 pixlar åt höger och 124 pixlar nedåt från bildens övre vänstra hörn (detta hörn är origo i bildens koordinatsystem).

Hur fyller man nu detta fält med en bild? En möjlighet är att skapa en helt artificiell bild genom att i en nästad for-loop fylla matrisen:

```
for (int x=0; x<imArray.length; x++)
    for (int y=0; y<imArray[0].length; y++) {
        imArray[x][y][0] = ...
        imArray[x][y][1] = ...
        imArray[x][y][2] = ...
    }
```

Detta är det sätt som initialbilden skapas i programmet (i metoden initialImage i klassen imagEFrame) där olika färger väljs beroende på avståndet från pixeln (300,200).

Men det normala sättet att skapa en bild är att läsa in den från en fil i JPEG-format; denna fil har i sin tur sannolikt skapats av en digitalkamera. Det finns stöd i Javas standardbibliotek för att läsa och skriva JPEG-filer till och från objekt av Javas standardklass `BufferedImage`. Bilder av denna klass kan i sin tur visas i fönster på skärmen av ett Javaprogram.

En BufferedImage är "nästan" ett sådant fält som vi beskriver ovan. Den skiljer sig på två viktiga sätt:

- en pixel representeras inte av ett fält av tre int-värden utan alla dessa placeras i ett int-värde; intensiteterna är små heltal och består av åtta bitar var, så de ryms gott och väl i en 32-bitars int. På detta sätt reduceras minnesåtgången för en bild radikalt jämfört med vår representation.
- Det tvådimensionella fält som innehåller pixelvärdena är en privat tillståndsvariabel i en BufferedImage. Som användare kan vi bara begära av bilden att få RGB-värdet för en given pixel genom att anropa dess metod

  `int getRGB(int x, int y).`

  BufferedImage fungerar alltså här på samma sätt som de modellklasser vi sett tidigare för svartvita bilder och i Game of Life.

Trots onödigt stort minnesutnyttjande ska vi alltså manipulera bilder som fält av typ `int[][][]`. Den givna klassen `Conversions` tar hand om översättning i bägge riktningar mellan `BufferedImage` och `int[][][]` genom metoderna `image2Matrix` respektive `matrix2Image`.

3. Filter

Om ni experimenterat med programmet har ni sett olika sätt att manipulera, eller filtrera, en bild. Här följer en kort beskrivning av de olika filtren.

Filter utan parameter

Dessa filter implementerar gränssnittet `ImageFilter`:

```
public interface ImageFilter {


    public String getMenuName();

    public void apply(int[][][] src, int[][][] dest);

}
```

Metoden apply applicerar filtret i fråga på bilden i src och lämnar resultatet i dest. Bägge fälten förutsätts vara redan skapade och lika stora.

- **Black and White**. Bilden görs om till gråskala; dvs i varje pixel har alla tre färgkomponenterna samma värde. Detta värde är ett vägt medelvärde av de tre komponenterna. För att stämma väl med hur det mänskliga ögat uppfattar ljushet ges den gröna komponenten överlägset störst vikt vid medelvärdesbildningen.
- **Invert**. Byter varje pixels färg till dess komplementfärg.
- **Flip Horizontal**. Spegelvänder bilden horisontellt.
- **Flip Vertical**. Spegelvänder bilden vertikalt.
- **Detect Edges**. Ett mer komplicerat filter som försöker identifiera kanter i bilden. Filtret kallas Sobelfilter och är välkänt inom bildbehandling.

Filter som beror på en reell parameter

Dessa filter implementerar gränssnittet ScalableFilter, som skiljer sig från ImageFilter genom att apply tar ytterligare en parameter som anger "hur mycket" filtret ska appliceras.

```
public interface ScalableFilter {

    public String getMenuName();

    public void apply(int[][][] src, int[][][] dest, double scale);

}
```

Parametern scale antas ligga mellan -1 och 1. När man väljer ett av dessa filter tillkommer en ny komponent i användargränssnittet, en `JSlider`, med vars hjälp användaren kan ange önskad parameter.

De två givna filtren av denna typ är bägge till för att förvrida bilden och åstadkomma intressanta effekter, till exempel med porträttbilder. Det är inte viktigt för den här labben att förstå hur dessa filter fungerar; vi ger därför de beskrivningarna på en särskild sida som ni kan hitta på hemsidan för den här laborationen.

4. Er uppgift

Skärpefilter

Ni ska implementera två nya filter och lägga till dessa till programmet.

Först ska ni definiera ett filter för att öka skärpan i en bild. Detta ska göras i form av en klass SharpenFilter, som implementerar ImageFilter. Detta filter är ett exempel på en vanligt förekommande typ, så kallade faltningsfilter (eng. convolution filter). Vi betraktar först endast den röda komponenten, som i varje pixel i src är ett heltal mellan 0 och 255. Den röda komponenten i en given pixel (x,y) i dest beräknas från de röda komponenterna i (x,y) och dess närmaste grannar i src, så som illustreras av följande bild:

0	−1	0
−1	5	−1
0	−1	0

Vi multiplicerar den röda komponenten i (x,y) med 5 och subtraherar motsvarande komponent i de fyra pixlarna till höger, vänster, ovanför och under (x,y). Resultatet är det den röda komponenten i (x,y) i dest, så när som på ett problem: resultatet kan bli negativt eller större än 255. Vi måste kolla detta och sätta den röda komponenten i (x,y) till 0 i det förra fallet och till 255 i det senare. Samma transformation görs sedan för de andra två färgkomponenterna.

Denna beräkning kan inte göras för pixlarna längs kanten på bilden; dessa lämnas därför oförändrade (dvs har samma värden i `dest` som i `src`).

Sedan ni definierat klassen `SharpenFilter` ska ni lägga till det till programmet. Detta kräver tillägg av en enda rad till `ImagEFrame.java`; genom att se hur de andra filtren läggs till bör det inte vara svårt att göra detta tillägg.

Er implementation av detta filter testas lämpligen på de porträtt som finns med i zip-filen.

Slutligen noterar vi att många vanliga filter kan beskrivas som faltningsfilter genom att ersätta matrisen ovan med andra matriser. Sobelfiltret ovan består huvudsakligen av två sådana filter. Den intresserade kan hitta många ytterligare exempel genom sökning på nätet.

Ljushetsfilter

Filter med parameter behöver inte vara så komplicerade som `Swirl` och `Square`. Ni ska nu implementera ett filter som kan göra bilden ljusare eller mörkare. Den extra parametern `scale` avgör hur mycket ljusare eller mörkare. Maximalvärdet `scale=1` ska öka alla tre komponenterna i alla pixlar med 128 -- men förstås med bivillkoret att resultatet inte får bli större än 255. Minimalvärdet `scale=-1` ska minska alla tre komponenterna med 128 -- men de får inte bli negativa. För värden på scale däremellan ska ökningen/minskningen bli proportionell. För detta filter liksom för `Swirl` och `Square` är det alltså så att `scale=0` lämnar bilden oförändrad. Den nya klassen, förslagsvis kallad `BrightnessFilter` ska implementera `ScalableFilter` och läggas till programmet.

När ni gjort detta är ni klara med labb 5. För redovisningen ska ni ladda upp tre filer i Fire: era två filterfiler och den ändrade `ImagEFrame.java`.

5. Frivillig uppgift

När ni experimenterat med programmet har ni förmodligen märkt en allvarlig brist -- man kan inte ångra en filtrering. Några filtreringar kan göras ogjorda genom att de har en invers; **Flip Horizontal**, **Flip Vertical** och **Invert** är alla sina egna inverser, men har man gjort **Detect Edges** eller **Black and White** finns ingen återvändo. I extrauppgiften ska ni åtminstone delvis åtgärda detta på följande sätt:

Ett nytt menyalternativ **Undo** ska läggas till i **Filter**-menyn. Att välja detta alternativ efter en filtrering ska göra filtreringen ogjord, dvs bilden före filtreringen ska visas. Däremot är det inte möjligt för användaren att göra mer än en Undo-operation i sträck. Denna begränsning gör tillägget mycket lättare eftersom den gamla bilden fortfarande finns kvar åtkomlig för programmet.

Att göra **Undo** då ett filteralternativ med skalning är valt ska innebära att bilden från starten av denna filtrering återställs.

När **Undo** inte är möjlig, dvs direkt efter inläsning av en bild och då **Undo** just gjorts, ska det synas på menyalternativet att det inte kan väljas, genom att texten **Undo** visas med grå text. Det finns en särskild metod i klassen **JMenuItem** med vars hjälp man kan sätta (och ta bort) detta tillstånd.

Att implementera detta tillägg bör inte innebära mer än cirka tjugo rader ny kod i klassen **ImagEFrame**. Ett nödvändigt första steg är att noga studera klassen och försöka förstå hur den fungerar.