Inlämningsuppgifter omgång 5

Allmänna instruktioner: Det finns två delar i detta dokument: Del 1 som innehåller uppgifter som måste göras för att få godkänt, och Del 2 som innehåller uppgifter för att få bonuspoäng till tentan för högre betyg. För att bli godkänd på inlämningsuppgifterna räcker det således att göra Del 1.

Döp era Javaprogram till de filnamn som specificeras i uppgifterna.

Spara alla java-filer i en katalog ”<användarnamn>_in5”, t.ex. johthu19_in5. Komprimera denna katalog till en zip-fil och skicka in filen.

Deadline för att lämna in uppgifterna på Blackboard är söndag 6/12 klockan 23:59:59.

Uppgifterna presenteras på examinationstillfället för din grupp 7/12-8/12.

Uppgift 1: BenfordsLaw.java

Den amerikanska astronomen Simon Newcomb observerade ett fenomen när han läste en matematikbok som han lånade: startsidorna var mycket mer använda än slutsidorna. Han misstänkte att forskare utförde fler beräkningar med siffror som började med 1-or än med 8-or eller 9-or och formulerade en hypotes som säger att under generella omständigheter är den ledande siffran mycket mer sannolik att vara 1 (ungefär 30%) än siffran 9 (mindre än 4%).

Detta fenomen är känt som Benfords lag och används nu ofta som ett statistiskt test. Ett exempel är IRS-forensic som använder detta för att upptäcka skattebedrägerier eller potentiellt valfusk.

Komponera ett program som läser in en fil med numeriska data (se bifogad fil data.txt) och skriver ut en tabell med statistiska data innehållande ”ledande siffra” och ”antalet förekomst i procent”

Se exempel på output:

```
# 1:   30.8%
# 2:   19.3%
# 3:   13.0%
# 4:    9.9%
# 5:    7.4%
# 6:    5.9%
# 7:    5.2%
# 8:    4.4%
# 9:    4.1%
```

Programmet skall delas i olika relevanta metoder.

Förslag:

1. **getLeadingDigit(…)** # returnerar första siffran i ett tal.

2. **calculateOccurencies(…)**# läser in data och fyller en array med förekomster för varje ledande siffra t.ex

7	**5**	**2**	**1**	**0**	**2**	**3**	**1**	**1**

Dvs. första elementet i listan är antal förekomster av tal med en 1:a som ledande siffra (vilket var 7 i detta fall). Det andra elementet i listan är förekomsten av tal med en 2:a som ledande siffra (vilket var 5 i detta fall) osv.

3. **calculateStatistics(…) #** beräknar förekomst av ett tal med en viss ledande siffra i %. Spara sedan datan i en annan lista.
4. **printTable()** # skriver ut informationen.

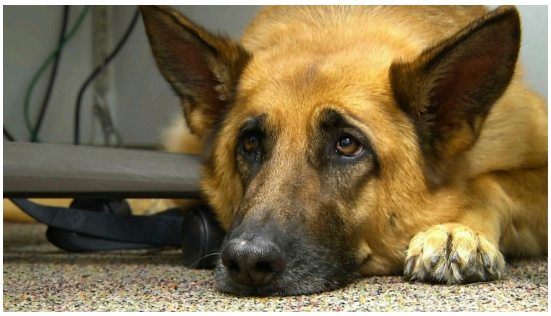
Extra (inte obligatoriskt): Hitta andra filer på internet och testa. Testa med slumpmässig data, stämmer Benfords lag i detta fall?

Uppgift 2: Tiling

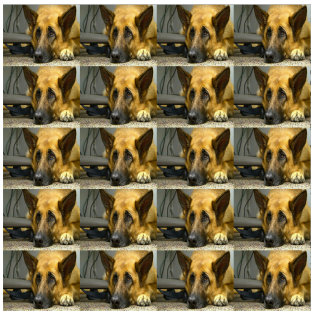
Givet en bild så är er uppgift att skapa en ny bild med multipla kopior av den första bilden. Mer precist så ska användaren ange

- ett filnamn för en fil med en bild,
- M: hur många gånger bilden ska förekomma i x-led i den nya bilden.
- N: hur många gånger bilden ska förekomma i y-led i den nya bilden.
- width: höjden (antalet pixlar i x-led) på den nya bilden.
- height: bredden (antalet pixlar i y-led) på den nya bilden.

Ett exempel ges nedan:



dog.jpg



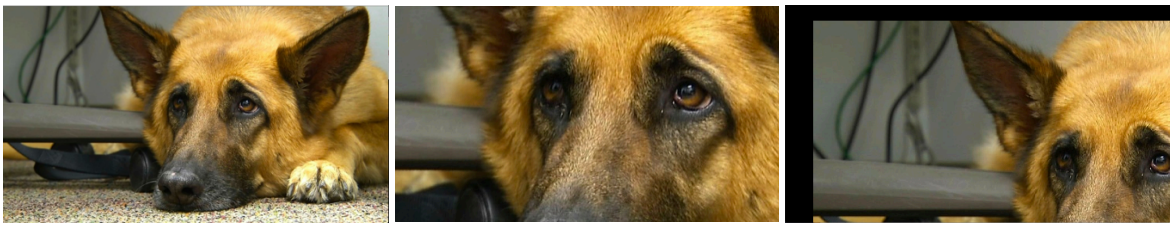
java Tiling dog.jpg 4 5 800 800

Tips: Använd klassen Picture. Ska ett Picture-objekt för ursprungsbilden (t.ex. dog.jpg eller annan bildfil) och ett annat Picture-objekt för bilden du ska skapa. Använd loopar för att gå igenom punkterna i bilden du ska skapa och placera ut pixlarna från ursprungsbilden på rätt ställe.

Del 2 – uppgifter för bonuspoäng

Uppgift 1: DigitalZoom.java (2p)

java Zoom dog.jpg 1.0 0.5 0.5 java Zoom dog.jpg 0.5 0.5 0.5 java Zoom dog.jpg 0.7 0.3 0.3

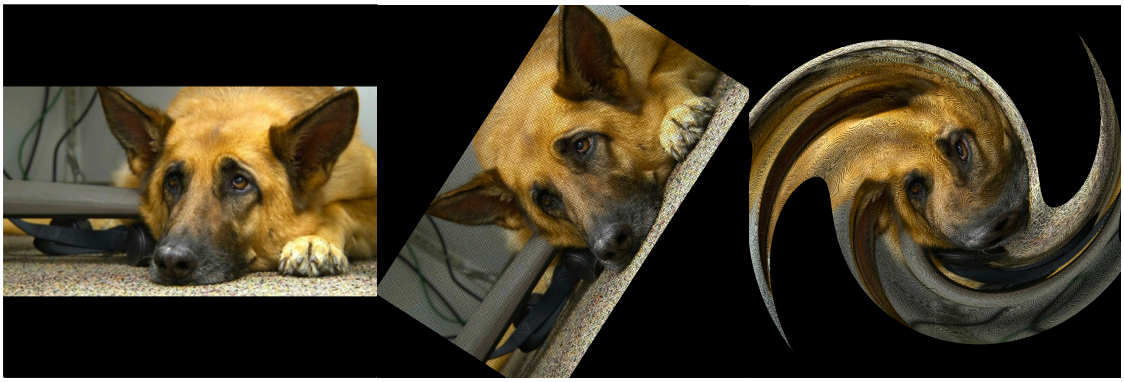


Skriv ett program (som t.ex. använder sig av klassen Picture) för att zooma i bilder. Användaren anger namnet på filen, en zoomfaktor mellan 0 och 1 (inversen av den om vi ska vara strikta), samt centerpunkten runt vilken man ska zooma. (0.0,0.0) är övre vänstra hörnet på bilden och (1.0,1.0) nedre högra hörnet på bilden. I de tre exemplen ovan så används filen dog.jpg. De zoomade bilderna ska vara av samma dimension som den ursprungliga bilden (dvs. om den ursprungliga bilden t.ex. har 800x600 pixlar ska den zoomade bilden även ha det).

I den första bilden ovan så har en zoomfaktor på 1.0 angetts, och centerpunkten har angetts till (0.5, 0.5), dvs. mitt i bilden. Med dessa inställningar så visas den ursprungliga bilden. I den andra bilden ovan så har zoomfaktorn angetts till 0.5 (dubbel förstoring) och centerpunkten till (0.5, 0.5). I den tredje bilden så är zoomfaktorn 0.7 (vilket innebär 1/0.7 förstoring) och centerpunkten är angiven till (0.3, 0.3).

Uppgift 2: Filters.java

Normal Rotation Swirl



I denna uppgift ska ni implementera två så kallade filter förklarade nedan.

- **(1p)** Det första filtret vi betraktar, *Rotation*, tar en bild (se första bilden till vänster ovan) och roterar denna runt mittpunkten på bilden med hjälp av en vinkel v (se mittenbilden ovan). För varje pixel (x, y) i den ursprungliga bilden, så fås den roterade pixeln (x_r, y_r) i den nya bilden via följande ekvationer:

$$x_r = c_x + (x - c_x)\cos(v) + \left(y - c_y\right)\sin(v)$$
$$y_r = c_y - (x - c_x)\sin(v) + \left(y - c_y\right)\cos(v)$$

  där (c_x, c_y) är mittpunkten på bilden. Notera att man kan behöva utöka den ursprungliga bilden med enfärgade pixlar (valda som svarta ovan) så att man garanterar att det finns en lösning till ekvationerna ovan.

- **(1p)** Det andra filtret, *Swirl*, fungerar som *Rotation* förutom att man låter vinkeln v vara avståndsoberoende. Mer precist så är

$$v(d) = \frac{\pi d}{256}$$

  där d är avståndet från mittpunkten av bilden (ni kan även, om ni vill, byta ut 256 mot andra tal och se vad resultatet blir).

  Ni behöver inte ha två separata metoder för de två filtren. Det andra filtret, *Swirl*, är endast en enkel modifiering av det första filtret.