Laboration 3.

Syfte

Syftet med denna laboration är att få övning i att strukturera sina program genom att använda metoder och klasser, samt att få övning i att använda sig av fält och **for**-satsen.

Redovisning

Uppgifterna skall vara demonstrerade för och godkända av en handledare senast måndag 24/2.

Uppgift 1

Skriv ett program som läser in ett heltal N och genererar en multiplikationstabell med N rader och N kolumner enligt exemplet nedan:

```
Ange gradtal: 4
1 * 1 = 1
1 * 2 = 2       2 * 2 = 4
1 * 3 = 3       2 * 3 = 6       3 * 3 = 9
1 * 4 = 4       2 * 4 = 8       3 * 4 = 12      4 * 4 = 16
```

Tips 1: Använd två nästlade **for**-satser - en yttre som styr antalet rader och en inre som styr antalet kolumner inom raderna.

Tips 2: Hur många kolumner skall den inre **for**-satsen skriva ut på den första raden? På den andra raden? På den tredje raden?

Uppgift 2

Skriv en klassmetod

public static boolean match(String str)

som kontrollerar parentesmatchning i strängar. Underprogrammet skall som indata ha en sträng och som utdata skall värdet **true** eller **false** lämnas beroende på om parenteserna matchar eller inte.

Exempel:

I exemplen nedan representerar . . en följd av godtyckliga tecken som inte innehåller vänster- eller högerparenteser:

. .(. .(. .). .).	skall ge värdet **true**
. .(. .(. . .).	skall ge värdet **false**
. .(. . . .). .(. . . .). .	skall ge värdet **true**
. . . .). .). .(. .(. . . .	skall ge värdet **false**
. .(. .). .). .(. .(. .). .	skall ge värdet **false**
. . . . .	skall ge värdet **true**

Givetvis skall du också skriva ett program för att testa metoden.

Tips: Räkna antalet förekomster av vänster- respektive högerparenteser under genomlöpningen av strängen. Vad måste gälla för förhållande mellan dessa värden - vid varje tidpunkt under genomlöpning och vid avslutad genomlöpning - för att parentesmatchningen skall vara korrekt?

Uppgift 3

I denna uppgift skall ni använda klassen GenericDie som ni gjorde i laboration 2.

Ni skall skapa en klass TestDice som innehåller en klassmetod

public static int[] roll(GenericDie theDice, **int** nrRoll)

som får en tärning theDice av klassen GenericDie, kastar denna tärning nrRoll antal gånger och returnerar ett heltalsfält som innehåller hur många gånger var och en av valörerna på tärningen kommit upp.

Skriv också en main-metod som använder metoden roll för att göra 100 kast med en 7-sidig tärning och skriver ut resultatet av dessa kast.

Tips: Fältet som metoden roll returnerar innehåller lika många element som det finns sidor på tärningen theDice. Klassen GenericDie har en metod för att ta reda på antalet sidor på tärningen.

Valörerna på en tärning är numrerade från 1 till max, medan indexen i ett fält är indexerade från 0 till max-1. Var i fältet är det lämpligt att lagra valören 1, valören 2, . . , valören max?

Uppgift 4

Utöka klassen TestDice i föregående uppgift med ytterligare en klassmetod

public static int[] valueOf(**int**[] number)

som tar ett heltalsfält number innehållande antal gånger som var och en av valörerna på en tärning kommit upp. Metoden skall returnera ett nytt fält som för varje valör innehåller den *sammanlagda poängsumman* för antalet gånger valören kommit upp. Har t.ex. valören 6 kommit upp 4 gånger skall värdet i returfältet som associeras med valören 6 vara 24.

Skriv sedan en main-metod som gör 100 kast med en 9-sidig tärning och skriver den totala poängsumman som varje valör erhållit under dessa kast, d.v.s. programmet skall först anropa metoden roll (från förra uppgiften) och därefter metoden valueOf.

Tips: Fältet som metoden valueOf returnerar innehåller lika många element som indatafältet number. Hur kan man ta reda på storleken på ett fält?

Uppgift 5

A är en kvadratisk heltalsmatris med N rader och N kolonner. I första kolonnen i A är alla elementen lika med 1. I första raden i A är alla övriga element lika med 0. För resterande element i A gäller att

$$a_{i\,j} = a_{i-1\,j-1} + a_{i-1\,j}$$

I fallet N = 5 får matrisen utseendet:

$$A = \begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 1 & 3 & 3 & 1 & 0 \\ 1 & 4 & 6 & 4 & 1 \end{matrix}$$

Skriv ett program som läser in ett positivt heltal N ≤ 10, beräknar matrisen A, samt skriver ut alla element i och till vänster om diagonalen hos A enligt nedan:

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
```

Utforma programmet på så sätt att det förutom main-metoden innehåller följande tre klassmetoder

public static int readGrad()	som läser in och returnerar ett gradtal ≤ 10
public static int[][] createArray(**int** n)	som skapar och returnerar en matris av gradtal n
public static void writeElement(**int**[][] m)	som skriver ut elementen i matrisen m

Tips1: Börja vid beräkningen av matrisen att initialisera alla element i första raden till 0 och sedan alla element i första kolumnen till 1.

Tips2: Uppgift 1 är till hjälp vid implementationen av metoden writeElement.

Tips3: Utveckla och testa en metod i taget. Börja med den metod du tror är lättast att utveckla.

Uppgift 6

Spelet Yatzy spelas med 5 vanliga 6-sidiga tärningar. Spelet består av 15 ronder och spelet går ut på att få högsta möjliga poäng. I varje rond får man kasta tärningarna tre gånger. Första gången tärningarna kastas i en rond måste samtliga tärningar kastas, medan man i de två resterande kasten i en rond får kasta ett godtyckligt antal av de fem tärningarna, dvs man kan avstå från att kasta vissa tärningar.

Efter varje rond måste man föra in sitt resultat i en av 15 olika poängkategorier. Den poängkategori som väljs blir inte längre möjlig att välja i en senare rond. Nedan anges vilka poängkategorier som finns och vilken poäng som erhålls:

Ettor	summan av poängen för tärningarna med värdet 1
Tvåor	summan av poängen för tärningarna med värdet 2
Treor	summan av poängen för tärningarna med värdet 3
Fyror	summan av poängen för tärningarna med värdet 4
Femmor	summan av poängen för tärningarna med värdet 5
Sexor	summan av poängen för tärningarna med värdet 6
Par	summan av de två högsta tärningarna med samma värde
Två par	summan av poäng för två par av tärningar med inbördes samma värde
Triss	summan av poäng för tre tärningar med samma värde
Fyrtal	summan av poäng för fyra tärningar med samma värde
Liten stege	15 poäng erhålles om tärningarna är i följd från 1 till 5
Stor stege	20 poäng erhålles om tärningarna är i följd från 2 till 6
Kåk	summan av poäng för tre respektive två tärningar med inbördes samma värde
Chans	summan av samtliga fem tärningar
Yatzy	50 poäng om alla tärningar har samma värde.

Er uppgift är att skriva en metod

```
public int[] evaluate(int[] dicePoints)
```

som från en given uppsättning av värden på 5 tärningar beräknar vilken poängsumma som erhålls för var och en av de 15 olika poängkategorierna ovan. Metoden skall finnas i en klass med namnet PlayEngine.

Observera att poängen för en poängkategori är oberoende av poängen i de andra poängkategorierna, d.v.s. att om t.ex. samtliga tärningar har värdet 4 erhålls följande poäng:

Fyror	20
Par	8
Två par	16
Triss	12
Fyrtal	16
Kåk	20
Chans	20
Yatzy	50

Under utvecklingen av metoden evaluate kan ni testa de individuella poängkategorierna genom att använda programmet TestScore.java som finns att kopiera från kursens hemsida. För att testa att er slutversion av metoden evaluate fungerar korrekt *skall* ni använda TestPlayEngine.java och RefPlayEngine.class, som ni kopierar från kursens hemsida.

Tips 1: Om ni sorterar fältet dicePoints innan ni börjar beräkna poängen för de olika poängkategorierna blir flertalet poängberäkningar mycket enklare. Om fältet är sorterat kan man t.ex. konstatera att *alla värden är lika om det första och sista värdet är lika*, dvs villkoret för att erhålla Yatzy.

Tips 2: En metod för att sortera fält finns i klassen Arrays.

Tips 3: *Implementera och testa en poängkategori i taget*. Börja med de enklaste poängberäkningarna först.

Tips 4: Använd privata hjälpmetoder för att få en överskådlig struktur på metoden evaluate.

Uppgift 7

I laboration 2 skrev ni en klass Guest för att representera gäster på hälsoinstitutet Svett&Fett.

Skriv en klass Diet som har som innehåller klassmetoden

```
public static ArrayList<Guest> putOnDiet(ArrayList<Guest> list)
```

Metoden tar en lista list av Guest och returnerar en ny lista, som innehåller de element i list som är överviktiga.

Ni kan testa med den färdiga klassen TestDiet, som på kursens hemsida.

Uppgift 8

En digital bild kan representeras som ett tvådimensionellt fält av bildpunkter.

I en digital gråskalebild är en bildpunkt ett heltalsvärde i intervallet 0-255, där värdet 0 betecknar svart och värdet 255 betecknar vitt. En bild har en viss storlek, dvs ett visst antal bildpunkter i höjdled och ett visst antal bildpunkter i sidled. En gråskalebild kan således representeras med ett tvådimensionellt fält av typen **int**[][], där den första dimensionen definierar bildens höjd och den andra dimensionen definierar bildens bredd. En variabel som representera en gråskalebild med höjden HEIGHT pixlar och bredden WIDTH pixlar kan skapas med satsen

int[][] grayImage = **new int**[HEIGHT][WIDTH];

Elementet grayImage[20][40] anger således gråtonen i den pixel som återfinns 40 pixlar åt höger och 20 pixlar nedåt från bildens övre vänstra hörn. Observera att grayImage[0][0] är pixeln i övre vänstra hörnet i bilden och att grayImage[HEIGHT-1][WIDTH-1] är pixeln i nedre högra hörnet.

En digital färgbild lagras på RGB-format, där varje bildpunkt utgörs av *tre* heltalsvärden i intervallet 0-255. De enskilda värdena representerar intensiteten av färgerna rött, grönt och blått. En färgbild kan avbildas med ett tredimensionellt fält av typen **int**[][][], är den första dimensionen definierar bildens höjd, den andra dimensionen definierar bildens bredd och den tredje dimensionen representerar färgerna rött, grönt och blått. En variabel som representera en färgbild med höjden HEIGHT pixlar och bredden WIDTH pixlar kan skapas med satsen

int[][][] colorImage = **new int**[HEIGHT][WIDTH][3];

Elementet colorImage[18][56][1] anger således intensiteten av grönt i den pixel som återfinns 56 pixlar åt höger och 18 pixlar nedåt från bildens övre vänstra hörn.

I denna uppgift skall ni skriva några enkla metoder för bildbehandling. På kursens hemsida finns ett antal färdiga klasser som ni skall använda.

<u>Gråskalebilder</u>.

I denna uppgift skall ni skriva metoder för behandling av gråskalebilder. På kursens hemsida finns klasserna GrayImage, GrayImagePanel, GrayImageWindow och MyGrayProgram, som ni skall använda.

Klassen GrayImage tillhandahåller de publika klassmetoderna

static int[][] read(String fileName)	som läser in en bild på gif -, jpg-, jpeg- eller png-format från filen fileName. Om ursprungsbilden i filen är en färgbild översätts denna till en gråskalebild.
static void write(String fileName, **int**[][] picture)	som skriver ut fältet picture som en gråskalebild på formatet gif, jpg, jpeg eller png till en fil med namnet fileName.

Klassen GrayImageWindow tillhandahåller en konstruktor

GrayImageWindow(**int**[][] picture1, **int**[][] picture2)	som skapar ett fönster i vilket fälten picture1 och picture2 ritas ut som gråskalebilder.

Klassen MyGrayScaleProgram innehåller nedanstående kod och producerar fönstret som visas ovan.

```
public class MyGrayProgram {
    public static void main(String[] args) throws Exception{
        int[][] original = GrayImage.read("svamp.jpeg");
        int[][] manipulated = upDown(original);
        GrayImage.write("upDownSvamp.jpeg", manipulated);
        GrayImageWindow iw = new GrayImageWindow(original, manipulated);
    }
    public static int[][] upDown(int[][] samples) {
        int[][] newSamples = new int[samples.length][samples[0].length];
        for (int row = 0; row < samples.length; row = row + 1)
            for (int col = 0; col < samples[row].length; col = col + 1)
                newSamples[row][col] = samples[samples.length - row - 1 ][col];
        return newSamples;
    }//upDown
}//MyGrayProgram
```

I klassen finns metoden

public static int[][] upDown(**int**[][] samples)

som tar ett tvådimensionellt fält samples (som representerar en gråskalebild) och returnerar ett nytt tvådimensionellt fält (som också representerar en gråskalebild). Det nya fältet som metoden returnerar är en spegelvänd kopia av fältet samples roterad runt den horisontella axeln, dvs första raden i det nya fältet är sista raden i samples, andra raden i det nya fältet är näst sista raden i samples, osv. Metoden kan alltså användas för att vända en bild upp och ner.

Metoden main läser in en fil (svamp.jpeg) som innehåller en gråskalebild och lagrar denna i det tvådimensionella fältet original, skapar ett nytt fält manipulated genom att anropa metoden upDown med fältet original, skriver ut fältet manipulated som en bild till filen upDownSvamp.jpeg och skapar slutligen ett fönster där filerna original och manipulated ritas ut som bilder.

Din uppgift är att utöka klassen MyGrayScaleProgram med följande metoder:

public static int[][] leftRigt(**int**[][] samples)	som returnerar en spegelvänd kopia av samples roterad runt den vertikala axeln.
public static int[][] invert(**int**[][] samples)	som returnerar en kopia av samples där värdet $s_{i,j}$ för varje element i samples har ersätts med värdet 255- $s_{i,j}$.
public static int[][] toBlackWhite(**int**[][] samples)	som returnerar en kopia av samples där värde $s_{i,j}$ för varje element i samples har ersätts med värdet 0 om $s_{i,j}$ är mindre än 128 och med värdet 255 om $s_{i,j}$ är större eller lika med 128.


orginal


leftRight


invert


toBlackWhite

För att rita ut t.ex. den spegelvända kopian ändrar ni satsen

int[][] manipulated = upDown(original);

till

int[][] manipulated = leftRight(original);

i main-metoden i klassen MyGrayScaleProgram. Vill ni lagra den nya bilden som en jpeg-bild ändrar ni också filnamnet i satsen

GrayImage.write("upDownSvamp.jpeg", manipulated);

Färgbilder.

För färgbilder finns klasserna ColorImage, ColorImagePanel, ColorImageWindow och MyColorProgram.

Klassen ColorImage tillhandahåller klassmetoderna

static int[][][] read(String fileName)	som läser in en bild på gif -, jpg-, jpeg- eller png-format från filen fileName.
static void write(String fileName, **int**[][][] picture)	som skriver ut fältet picture som en bild på formatet gif, jpg, jpeg eller png till en fil med namnet fileName.

Klassen ColorImageWindow tillhandahåller en konstruktor

ColoImageWindow(**int**[][][] picture1, **int**[][][] picture2)	som skapar ett fönster i vilket fälten picture1 och picture2 ritas ut som färgbilder.

Klassen MyColorProgram har utseendet:

```
public class MyColorProgram {
    public static void main(String[] args) throws Exception {
        int[][][] original = ColorImage.read("svamp.jpeg");
        int[][][] manipulated = upDown(original);
        ColorImage.write("upDownSvamp.jpeg", manipulated);
        ColorImageWindow iw = new ColorImageWindow(original, manipulated);
    }
    public static int[][][] upDown(int[][][] samples) {
        int[][][] newSamples = new int[samples.length][samples[0].length][3];
        for (int row = 0; row < samples.length; row = row + 1)
            for (int col = 0; col < samples[row].length; col = col + 1)
                for (int c = 0; c < samples[0][0].length; c = c + 1)
                    newSamples[row][col][c] = samples[samples.length-row-1][col][c];
        return newSamples;
    }//upDown
}
```

Din uppgift är att utöka klassen MyColorProgram med följande metoder:

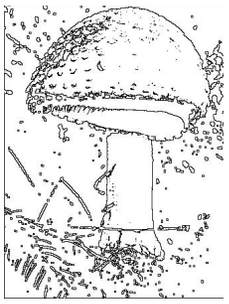
public static int[][][] leftRight(**int**[][][] samples)	som returnerar en spegelvänd kopia av samples roterad runt den vertikala axeln.
public static int[][][] invert(**int**[][][] samples)	som returnerar inversen av samples
public static int[][][] toGray(**int**[][][] samples)	som returnerar en gråbildskopia av samples
public static int[][][] toBlackWhite(**int**[][][] samples)	som returnerar en svartvit kopia av samples
public static int[][][] contour(**int**[][][] samples)	som returnerar konturen av samples


leftRight


invert


toGray


toBlackWhite


contour

Inversen till en färgbild erhålls genom att för varje pixel i orginalbilden ersätta intensiteten $s_{i,j}$ *för var och en av färgerna* röd, grön och blå med värdet 255- $s_{i,j}$.

För att få bästa kvalitén på den gråskalebild (och därmed den svartvita bild) som erhålls från en färgbild skall ni beräkna bildpunkternas *luminans*. Luminansen L är den för ögat upplevda ljusheten hos en yta och definieras som

$$L = 0.299\ r + 0.587g + 0.114b.$$

där r är intensiteten av rött, g är intensiteten av grönt och b är intensiteten av blått. I den gråskalebild som beräknas från en färgbild, sätts intensiteten av färgerna rött, grön och blått i varje pixel till värdet L i motsvarande pixel i färgbilden. I den svartvita bild som erhålls från en färgbild, sätt intensiteten av färgerna rött, grön och blått i varje pixel till värdet 0 om L i motsvarande pixel i färgbilden är mindre än 128 och till 255 om L är större eller lika med 128.

En *kontur* i en svartvit bild kan definieras enligt nedan:

> I konturen av bilden picture, skall en pixel vara svart om och endast om picture[i][j] är svart och antingen (i, j) ligger på *randen* i picture eller (i, j) har minst en *granne* som är vit i picture. Randen i bilden utgörs av de yttre raderna och kolumnerna. Grannar till (i, j) är alla (i_k, j_k) sådan att $i_k \in$ i-1..i+1 och $j_k \in$ j-1..j+1.

Bildpunkter som ligger på randen kan inte direkt beräknas med detta uttryck eftersom dessa bildpunkter saknar en eller flera närliggande punkter. Ni kan bortse från bildpunkterna på randen och låta dessa ha samma värden som i den svartvita varianten av originalbilden.

Frivilliga extrauppgifter

I bildanalys används ofta så kallade *faltningsfilter* för att lyfta fram sådant som är intressant i bilden. Det kan t.ex. röra sig om att ta bort brus, eller att reducera eller framhäva konturer i bilden. Vid användning av faltningsfilter beräknas ett nytt värde på en bildpunkt genom att, förutom att beakta bildpunkten själv, även beakta närliggande bildpunkter. Ett faltningsfilter kan således anses som en matris. Faltningsfiltret

```
-1  -1 -1
-1   9 -1
-1  -1 -1
```

är ett filter som *förändrar skärpan* i bilden. Filtret anger att det nya värdet i en bildpunkt beräknas genom att multiplicera bildpunktens gamla värde med 9 och subtrahera med varje närliggande bildpunkt. Detta kan också uttryckas på följande sätt:

> Alla bildpunkter $new_{i,j}$ i den bild som erhålls med faltningsfiltret och som *inte ligger på randen* i bilden beräknas av uttrycket
>
> $$new_{ij} = -1*old_{i-1,j-1} -1*old_{i-1,j} -1*old_{i-1,j+1} -1*old_{i,j-1} + 9*old_{i,j} -1*old_{i,j+1} -1*old_{i+1,j-1} -1*old_{i+1,j} -1*old_{i+1,j+1}$$
>
> där $old_{i,j}$ är värdet för bildpunkten i,j i bilden som filtreras. I en färgbild appliceras faltningsfiltret på varje färgkomponent. Värdet av den nya bildpunkten $new_{i,j}$ kan bli negativt eller större än 255. Detta måste kontrolleras. Negativa värden sätts till 0 och värden större än 255 sätts till 255.

Ett annat faltningsfilter som förändrar skärpan är

```
 0  -1   0
-1   5  -1
 0  -1   0
```

För att beräkna det nya värdet av en bildpunkt används i detta filter endast 4 av de 8 närbelägna bildpunkterna. Resultatet av de båda filtren ser ni nedan.

För att detektera kanter i bilder appliceras två filter på original bilden (ett filter för att få fram kanterna i x-led och ett filter för att få fram kanterna i y-led). Ett välkänt kantdekteringsfilter är Sobel-filtret, vars faltningsmatriser har följande utseende:

```
-1  -2  -1          1   0   -1
 0   0   0          2   0   -2
 1   2   1          1   0   -1
```

Om vi för varje enskild färgkomponent betecknar resultatet från dessa båda filter för bildpunkten (i, j) som $d_x(i, j)$ respektive $d_y(i,j)$, beräknas slutresultatet från Sobel-filteringen som $s(i,j)=\sqrt{d_x(i,j)^2+d_y(i,j)^2}$.

Färgbilden har först översatts till en gråskalebild, varefter gråskalebilden filtrerats med Sobel-filtren

Varje enskild färg i färgbilden har filtrerats med Sobel-filtren, varefter det erhållna värdet inverterats (dvs satts till 255 – värde).

Antalet bearbetningar man kan göra på bilder är näst intill obegränsat. Ni kan säkert komma på egna förslag på metoder att implementera. Den intresserade kan också hitta många ytterligare exempel på bildbehandling och filter genom sökning på nätet. Några lämpliga adresser att börja med är:

http://en.wikipedia.org/wiki/Image_editing

http://en.wikipedia.org/wiki/Sobel_operator