# Inlämningsuppgifter omgång 3

**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>_in3”, t.ex. johthu19_in3. Komprimera denna katalog till en zip-fil och skicka in filen.

Deadline för att lämna in uppgifterna på Blackboard är söndag 22/11 klockan 23:59:59.

Uppgifterna presenteras på examinationstillfället för din grupp 23/11-24/11.

# Del 1 – uppgifter för godkänt

Dessa inlämningsuppgifter behandlar: Fält eller ”arrayer”.

**Uppgift 1: CheckDate.java**

I denna uppgift ska du skriva ett program där användaren anger en månad och en dag. Sedan ska programmet skriva ut om detta är en giltig dag på året. Vi antar att året är 2019, vilket inte är ett skottår. Om användaren t.ex. anger månad: 3 och dag: 21, så är detta en giltig dag, men om användaren anger månad: 2 och dag: 30, så är detta inte en giltig dag då 30:e februari inte är ett giltigt datum.

**Uppgift 2: Names.java**

I filen namn.txt återfinns de tio vanligaste namnen för nyfödda flickor. Skriv ett Javaprogram som skriver ut på skärmen det namn av de tio som är först i den alfabetiska ordningen.

**Tips:** Ni kan läsa in filen genom att göra följande

```
Java Names < namn.txt
```

Ni kan t.ex. skapa ett Scanner objekt

```
Scanner s = new Scanner(System.in);
```

Om ni har en String-array namn av längd 10, så sparar ni namnen i denna genom att köra en loop. I iteration i så sparas det i:te namnet i arrayen namn genom:

```
namn[i] = s.next();
```

Efter detta kan sedan använda (s1.compareTo(s2) >= 0) för att undersöka om strängen s2 kommer före strängen s1 i alfabetet.

**Extra (inte obligatoriskt):** Utöka programmet så att alla de 10 namnen skrivs ut på skärmen i alfabetisk ordning.

**Uppgift 3: DrawPattern.java**

En grafisk designer saknar kompetensen att dra linjer mellan punkter. Personen ifråga besitter kompetensen att generera punkter och specificera mellan vilka punkter det ska vara linjer. Er uppgift är att hjälpa denna person att rita ut de objekt som denne vill ha utritat.

Mer specifikt så finns det en textfil data som innehåller en specifikation på 16 mönster som ska ritas upp med hjälp av 86 punkter. De 16 första raderna i textfilen data (rad 0 till 15) representerar mönstren och de resterande raderna (rad 16 till 98) representerar punkterna. Punkterna representeras av två tal som utgör koordinater.

Första raden i data är ”0 2”. Detta innebär att man ska dra en linje mellan punkten 0 (som finns på rad 16 i data) till punkten 1 (som finns på raden 17 i data). Sen ska man dra en linje mellan punkten 1 (som finns på raden 17 i data) till punkten 2 (som finns på raden 18 i data).

Andra raden i data är ”3 4”. Detta innebär att man ska dra en linje mellan punkten 3 (som finns på raden 19 i data) till punkten 4 (som finns på raden 20 i data).

Den generella principen är att man går igenom alla de 16 första raderna i data och drar linjer mellan efterföljande punkter i det intervall som anges respektive rad. Om det står ”10 17” på en rad så ritar ni en linje mellan punkt 10 och punkt punkt 11, sen mellan punkt 11 och punkt 12, osv… , sen tillslut mellan punkt 16 och punkt 17.

**Tips:** För att läsa in en fil, se tips under Uppgift 2.

De sexton första raderna i data är alla på formen ”start stop”, där start och stop är heltal. Ni kan spara dessa värden i två arrayer av längd 16 eller en tvådimensionell array.

De resterande raderna (från rad 16 och ramåt) representerar punkter och är på formen ”x y”, där x och y är heltal. Ni kan spara dessa punkter med hjälp av två arrayer eller en tvådimensionell array.

Ni kan använda StdDraw för att rita ut figurerna. I början av programmet kan ni ställa in t.ex.

```
StdDraw.setXscale(0, 300);
```

```
StdDraw.setYscale(0, 300);
```

För att rita ut en linje mellan punkterna (x1,y1) och (x2,y2) kan man göra följande:

```
StdDraw.line(x1, y1, x2, y2);
```

Efter ni har ritat ut figurerna, om ni tycker att designen ser lite upp och nervänd så att säga, så kan ni enkelt korrigera detta genom att ändra y-koordinaterna när ni ritar linjerna genom:

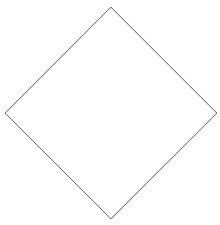
```
StdDraw.line(x1, 300-y1, x2, 300-y2);
```

# Del 2 – uppgifter för bonuspoäng

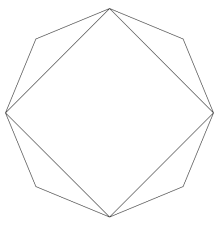
**Uppgift 1: CircleApproximation.java (2p)**

I denna uppgift ska du skriva ett program som ritar upp approximationer av en cirkel. Användaren anger ett tal N, t.ex. 3. Sedan ska N stycken approximationer av cirkeln ritas upp i samma bild. Den första approximationen består av en kvadrat, den andra av en åttahörning, den tredje av en sextonhörning, osv. N sådana figurer ska ritas ovanpå varandra i samma bild, där figurerna har $4 * 2^0$, $4 * 2^1$, $4 * 2^2$, $4 * 2^3$ … $4 * 2^{N-1}$hörn.

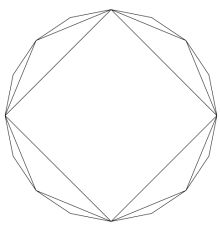
Bilder för N = 1, N = 2, och N = 3 finns nedan.



N=1



N=2

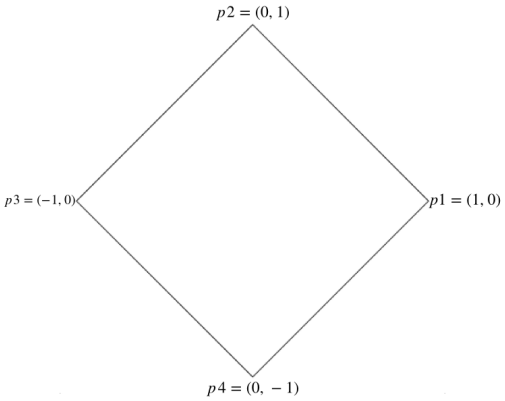


N=3.

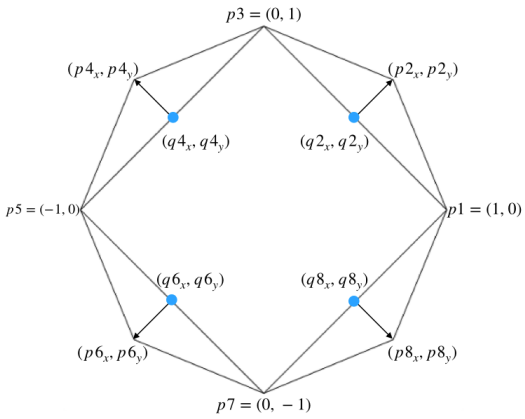
För att få 1 poäng så krävs endast att programmet gör det som det ska.

För att få 2p så får inte lösningen använda sig av fördefinierade trigonometriska funktioner. Istället så gör ni följande:

Ni börjar med fallet N=1, och skapar de punkter som ni ska dra linjer mellan samt ritar ut linjerna mellan dem. Se nedan:




Sen, när N är större än 1, så utgår ni från fallet N-1, och för varje linje räknar ni ut mittpunkten på linjen och flyttar denna mittpunkt ut på cirkeln. Sen drar ni linjer mellan de gamla och de nya punkterna. Se exempel nedan för N = 2. Mittpunkterna på linjerna är blå.




För fallet N=2 kan ni först räkna ut mittpunkterna på de fyra tidigare linjerna, blå prickar i bilden. Mittpunkten räknas ut t.ex. genom att addera den första punkten med skillnaden mellan den andra och den första punkten.

Punkten q2, mitt emellan p1 och p3 är således q2 = p1 + 0.5*(p3-p1). Detta innebär att x- respektive y-koordinaten för q2 är

```
q2x = 1 + 0.5*(0-1) // = 0.5
```

```
q2x = 0 + 0.5*(1-0) // = 0.5
```

Punkten p2 = (p2x, p2y) på cirkeln fås sedan från mittpunkten på linjen genom:

```
p2x = q2x/Math.sqrt(q2x*q2x + q2y*q2y);

p2y = q2y/Math.sqrt(q2x*q2x + q2y*q2y);
```

**Uppgift 2: Permutations.java (2p)**

En permutation är en array (fält) av längd *n* som innehåller alla heltalen mellan 0 och n-1. Detta innebär att varje sådant heltal mellan 0 och n-1 finns i arrayen och det förekommer endast en gång. Se nedan tre exempel på permutationer för *n* = 5.

```
int p1[] = {0,1,2,3,4};

int p2[] = {1,3,4,2,0};

int p2[] = {1,0,2,4,3};
```

Permutationer används inom massor av tillämningar; inom datorseende, mönsterigenkänning, tilldelningsproblem, etc. Vissa permutationsproblem är svårlösliga och så kallade NP-fullständiga såsom handelsresandeproblemet. Det problem vi betraktar här är däremot inte så svårt (som tur är).

Inversen till en permutation a är en annan permutation b av samma längd sådan att för varje i är följande uttryck true:

```
(a[b[i]] == i) && (b[a[i]] == i)
```

Skriv ett program där användaren anger ett heltal n. Därefter

- Skapas en double-array numbers av längd n där varje element är ett slumptal mellan 0 och 1.
- Skapa en permutation perm av längd n som är på formen {0,1,2,…,n-1}, dvs det första elementet är 0, det andra 1, det tredje 2, osv. upp till sista elementet som är n-1.
- Spara numbers i en array numbersOld. Sortera därefter arrayen numbers så att elementen efter sorteringen är i stigande ordning, dvs. det första elementet är det minsta talet, det andra elementet är det näst minsta osv. Detta göres genom att använda bubbelsortering. Det innebär att

man, n gånger, går igenom alla tal i numbers ett efter ett och byter plats på två på varandra följande tal om det första talet är större än det andra talet. Varje gång man byter plats på två tal i numbers så byter man även plats på motsvarande tal i perm.

- Inversen, vi kallar den permInv till permutationen perm räknas nu ut.
- Sedan verifieras att när vi permuterar numbers med permInv, så erhåller vi numbersOld. Dvs. för varje i är följande uttryck true:

```
numbers[permInv[i]] == numbersOld[i]
```