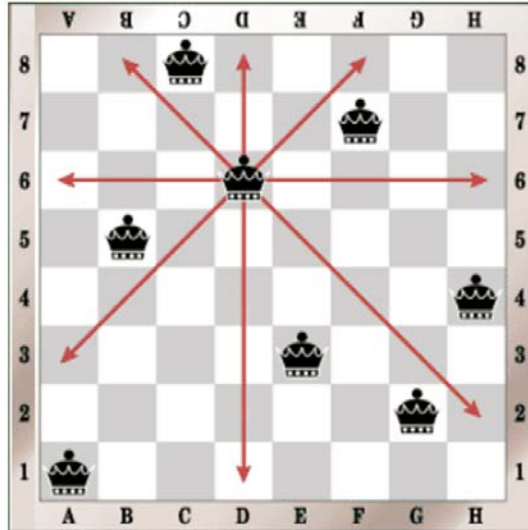


# DRONNINGER (QUEENS)

I denne opgave vil vi beskæftige os med det såkaldte 8-dronningeproblem, hvor man skal placere 8 dronninger på et 8 x 8 skakbræt, således at ingen af dronningerne kan slå hinanden. Det betyder, at den enkelte dronning ikke må have andre dronninger i den række, søjle og de to diagonaler, som går igennem dens position.



Mere generelt vil vi beskæftige os med  $n$ -dronningeproblemet, hvor man for  $n \geq 1$  skal placere  $n$  dronninger på et  $n \times n$  skakbræt, således at ingen af dronninger kan slå hinanden. Problemet er beskrevet på websiden: [en.wikipedia.org/wiki/Eight\\_queens\\_puzzle](https://en.wikipedia.org/wiki/Eight_queens_puzzle)

Umiddelbart kan man tro, at skakbrættet bør repræsenteres ved et 2-dimensionelt array **queens = new boolean[n][n]**, hvor den boolske værdi angiver, om der står en dronning på feltet eller ej. Da vi imidlertid ved, at der aldrig kan være mere end én dronning i hver række, er det simpelt at repræsentere brættet ved et 1-dimensionelt array **queens = new int[n]**, hvor **queens[i]** angiver positionen af dronningen i række  $i$  (dvs. at **queens[3] == 5** betyder, at dronningen i række 3 står i søjle 5). Som sædvanlig starter vi nummereringen med 0, dvs. række 0 og søjle 0.

Testserveren bruges ikke i denne opgave, men ud fra de prints, som jeres program producerer i terminalvinduet, er det let at se, om alt fungerer som det skal.

Husk at I fra og med denne opgave skal dokumentere jeres programmer, sådan som det er vist i BlueJ bogen og ved en forelæsning. Ellers får I genaflevering.

## Opgave 1

Lav en **Solver** kasse, som har tre feltvariabler:

- **int noOfQueens**
- **int[] queens**
- **int noOfSolutions**

samt fem metoder, hvoraf den første er public, mens de fire sidste er private

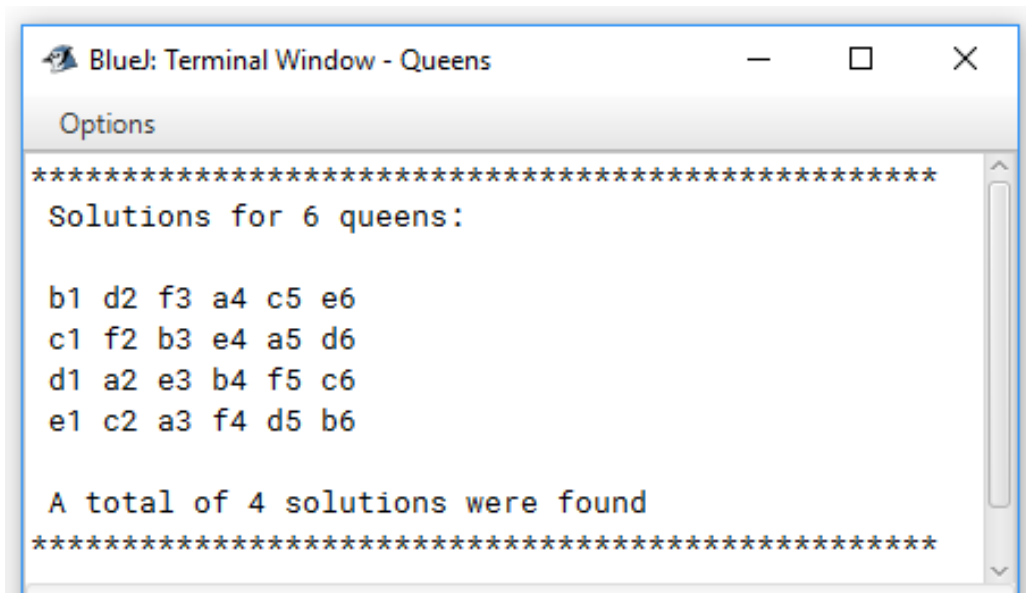
- **void findAllSolutions(int noOfQueens)**
- **void positionQueens(int row)**
- **boolean legal(int row, int col)**
- **void printSolution()**
- **String convert(int row, int col)**

Den vigtigste metode er ***positionQueens***, som er rekursiv.

- Et kald med parameterværdien 0 placerer en dronning i række 0, for derefter at kalde metoden med parameterværdien 1, hvilket placerer en dronning i række 1, for derefter at kalde metoden med parameterværdien 2, osv.
- I hvert trin bruges ***legal*** metoden til at tjekke, hvor i rækken dronningen kan placeres. Metoden returnerer ***true***, hvis det er legalt, at placere en dronning på positionen ***(row,col)***.
- Når et af de rekursive kald har afprøvet alle felterne i sin række, returnerer kaldet, og dermed er man tilbage til det foregående kald, som forsøger at placere sin dronning på de resterende felter i sin række.
- Når den rekursive metode kaldes med parameterværdien ***noOfQueens***, har man fået placeret alle dronningerne, og den fundne løsning udskrives på terminalen ved hjælp af metoden ***printSolution***, som bruger ***convert*** metoden til at konvertere fra ***(row,col)*** notation til den sædvanlige skaknotation for felter, (hvor ***(0,0)*** skrives som ***a1***, mens ***(5,3)*** skrives som ***d6*** og ***(7,7)*** som ***h8***).

Metoden ***findAllSolutions*** er den metode som stilles til rådighed for brugerne.

- Den opdaterer feltvariablerne, samt udskriver noget start info på terminalen (de tre første linjer på nedenstående udskrift).
- Herefter kaldes ***positionQueens*** med parameterværdien 0, hvilket placerer den første dronning og starter rekursionen.
- Når det første kald til ***positionQueens*** returnerer, er alle løsninger fundet og udskrevet på terminalen, og der udskrives noget end info på terminalen (de tre sidste linjer på nedenstående udskrift).



```
Blue: Terminal Window - Queens
Options
*****
Solutions for 6 queens:

b1 d2 f3 a4 c5 e6
c1 f2 b3 e4 a5 d6
d1 a2 e3 b4 f5 c6
e1 c2 a3 f4 d5 b6

A total of 4 solutions were found
*****
```

## Opgave 2

Kald metoden **findAllSolutions** med parameterværdier fra 1 til 10 (begge inklusive) og tjek, at antallet af løsninger matcher det, der er angivet på ovennævnte webside om 8-dronningeproblemet. Tjek også, at nogle tilfældigt udvalgte løsninger er korrekte, dvs. at ingen dronninger kan slå hinanden.

Brug metoden **currentTimeMillis** fra **System** klassen til at måle, hvor mange millisekunder, det tager at finde alle løsninger for en given problemstørrelse. Varigheden udskrives i næstsidste linje af udskriften fra **findAllSolutions**, således at den nu slutter med teksten "were found in XX ms".

Undersøg, hvor højt du kan sætte antallet af dronninger, og stadig finde alle løsninger inden for 3 minutter.

## Opgave 3

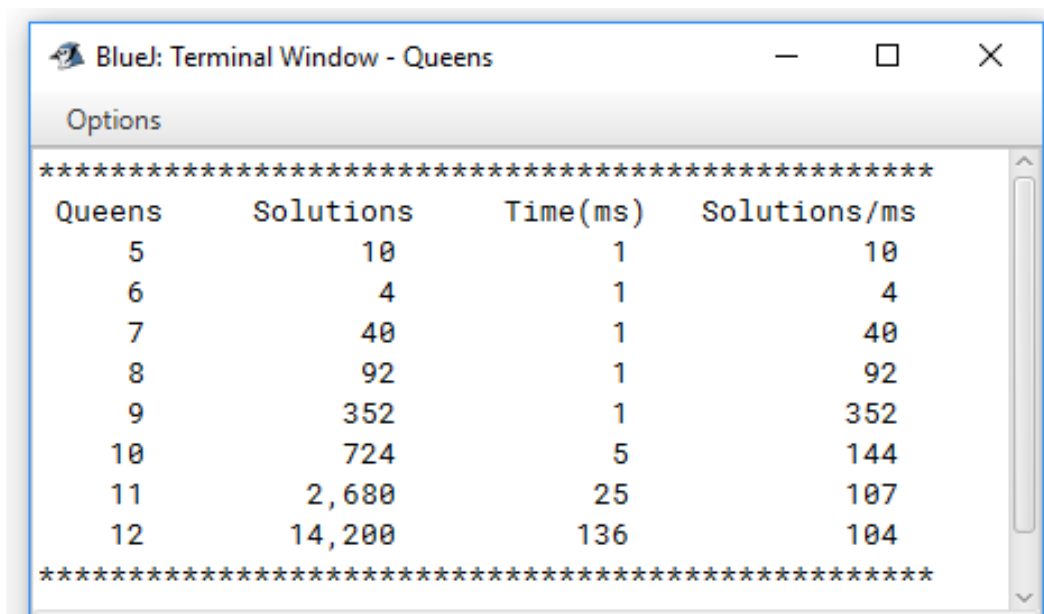
Forsyn **Solver** klassen med en ekstra feltvariabel

- **boolean showSolutions** // styrer om løsninger udskrives på terminalen

samt en ekstra public metode

- **void findNoOfSolutions(int min, int max)**

der finder antallet af løsninger for **noOfQueens** i intervallet **min** til **max** (begge inklusive) og udskriver disse, som vist nedenfor.



Queens	Solutions	Time(ms)	Solutions/ms
5	10	1	10
6	4	1	4
7	40	1	40
8	92	1	92
9	352	1	352
10	724	5	144
11	2,680	25	107
12	14,200	136	104

## Opgave 4

Implementér en testmetode

***public static void testSolver()***

Metoden skal skabe et ***Solver*** objekt, hvori den laver tre successive kald af metoden ***findAllSolutions*** med parameterværdierne 1, 2 og 6 efterfulgt af et kald af metoden ***findNoOfSolution*** med parameterværdierne (1, 12).

Klassemetoder skal producere nedenstående output på terminalen:

```
Blue: Terminal Window - Queens
Options
*****
Solutions for 1 queens:

a1

A total of 1 solutions were found in 0 ms
*****

*****
Solutions for 2 queens:

A total of 0 solutions were found in 0 ms
*****

*****
Solutions for 6 queens:

b1 d2 f3 a4 c5 e6
c1 f2 b3 e4 a5 d6
d1 a2 e3 b4 f5 c6
e1 c2 a3 f4 d5 b6

A total of 4 solutions were found in 0 ms
*****

*****
Queens      Solutions      Time(ms)      Solutions/ms
1           1             1             1
2           0             1             0
3           0             1             0
4           2             1             2
5          10             1            10
6           4             1             4
7          40             1            40
8          92             1            92
9         352             1           352
10         724             5           144
11        2,680           23           116
12       14,200          135           105
*****
```

Tjek at testmetoden producerer det forventede terminaloutput – og tilret dit projekt, hvis det ikke er tilfældet. Det forbrugte antal millisekunder vil selvfølgelig variere fra computer til computer (og fra kørsel til kørsel), men det øvrige output skal være som angivet ovenfor.

## Opgave 5 (til dem der har mod på mere)

I denne opgave skal I bruge BlueJ's debugger til at undersøge **positionQueens** metoden fra opgave 1.

Indsæt et breakpoint på det sted i koden, hvor **positionQueens** kalder sig selv rekursivt. Kald derefter **findAllSolutions** med parameterværdien **8**.

I vil nu få en udførelse, der svarer til den animation, der blev vist på en slide i en af forelæsningerne. Hver gang breakpointet nås, placeres en dronning (svarende til en grøn prik i animationen). Kontroller at feltvariablen **queens** indeholder de korrekte værdier (ved at trykke på den lille røde firkant) samt at de lokale variabler **row** og **column** (eller hvad I nu har kaldt de variabler hvori række- og søjlenummeret opbevares) har værdier, der svarer til positionen af den første grønne prik i animationen.

Efterfølgende trykkes på debuggerens *Continue* knap, hvorefter I når en tilstand, der svarer til den anden grønne prik i animationen. Bemærk, at *Call Sequence* i debuggerens venstre side viser, at der nu er to aktive kald af **positionQueens** metoden, og at I for hver af disse kan tjekke værdien af deres feltvariabler og lokale variabler. Tjek igen, at **queens**, **row** og **column** (for det seneste kald af **positionQueens**) har de forventede værdier.

Fortsæt med at trykke på *Continue* knappen, og tjek hver gang værdierne af **queens**, **row** og **column** op mod de grønne prikker i animationen. Bemærk at der i række 4 først placeres en dronning i søjle 3 og dernæst (efter backtracking) en dronning i søjle 7. Fortsæt med at trykke på *Continue* knappen indtil I har tjekket hele animationen på sliden.

## Opgave 6 (til dem der har mod på mere)

Fjern det breakpoint, som I indsatte i opgave 5, og kald igen **findAllSolutions** med parameterværdien **8**. Tag et screendump eller lignende af BlueJ's terminal, således at I kan huske de første fem fundne løsninger

Indsæt dernæst et breakpoint på det sted i **positionQueens**, hvor I har fundet en løsning, og kald derefter igen **findAllSolutions** med parameterværdien **8**.

I vil nu få en udførelse, der stopper hver gang, der er fundet en løsning. Tjek for de første fem stop, at værdierne af feltvariabler og lokale variabler i debuggeren svarer til de løsninger som blev udskrevet i terminalen, da I kørte metoden uden breakpoints. Hvor mange aktive kald af **positionQueens** er der, hver gang udførelsen stopper? Kan I forklare, hvorfor det er tilfældet?

## Opgave 7 (til dem der har mod på mere)

Den i opgave 5 nævnte slide med animationen, indeholder nogle tal for antal positioneringer og antal rekursive kald, der er nødvendige for at finde første løsning for **noOfQueens** lig med 8. Forsøg om I kan eftervise disse tal, ved at indsætte nogle ekstra feltvariabler / lokale variabler, hvori disse tal kan opsamles.

Sliden indeholder også cirka tal for antal rekursive kald, der er nødvendige for at finde samtlige løsninger for **noOfQueens** lig med 8, 12 og 16. Forsøg om I kan eftervise disse tal, ved at indsætte nogle ekstra feltvariabler / lokale variabler, hvori disse tal kan opsamles.

Den efterfølgende slide om **legal** metoden indeholder nogle cirka tal for, hvor ofte denne metode kaldes. Forsøg om I kan eftervise disse tal, ved at indsætte nogle ekstra feltvariabler / lokale variabler, hvori disse tal kan opsamles. For den sidste af målingerne vil det være nødvendigt at bruge **long** i stedet for **int**.

Prøv også om I kan eftervise påstanden om, at **legal** metoden bruger godt halvdelen af den samlede beregningstid.