

INF1010 2015 — Obligatorisk oppgave 9, 10 og 11 — SUDOKU

Versjon 26. mars. Ved behov for presiseringer i del 10 og 11, kan det komme ny versjon 16. april, men ingen endringer i selve oppgaven.

Denne obligatoriske oppgaven skal løses individuelt. Det betyr at alt som leveres for retting skal du selv ha skrevet inn. Det er ikke lov å ta inn kode som er laget av andre. Hverken andre studenter, fra nettet eller andre kilder. Oppdages slik kode i noe som er levert vil det bli oppfattet som forsøk på fusk og fulgt opp i tråd med regelverket om forsøk på fusk.

Denne obligen dekker følgende deler av pensum (stikkord): objektorientert programmering, rekursjon, GUI, lenkelister, ...

Obligen er delt inn i tre innleveringer (9, 10 og 11), slik at det er mulig å få tilbakemeldinger underveis. Oppgave 9 og 10 er igjen delt i to, slik at arbeidet fordeles på en 5-ukers periode med ca. 2 poengs arbeid pr uke. Tilsammen skal du ende opp med et program som kort fortalt løser sudokuoppgaver. Men før vi beskriver de enkelte delene, tar vi en gjennomgang av hele oppgaven for oversiktens skyld:

Hva er sudoku?

Se denne wikipediartikkelen:

<http://no.wikipedia.org/wiki/Sudoku>. (For mange fler detaljer, gå til den engelske wikipediaartikkelen <http://en.wikipedia.org/wiki/Sudoku>.)

Sudokubrettet

Et sudokubrett består av $n \times n$ ruter. Vi bruker følgende begreper i oppgaven:

		1					6	
			6			1		4
6		5	4				3	
	2	4		3				
			9		6			
				5		9	1	
	8				9	4		5
5		3			7			
	1					7		

- **rute** er feltet som det kan stå ett tall (eller én bokstav) i.
- **brett** er alle $n \times n$ ruter.
- **rad** er en vannrett (fra venstre mot høyre på brettet) rekke med n ruter.
- **kolonne** er en loddrett (ovenfra og nedover) rekke med n ruter.
- **boks** er flere vannrette og loddrette ruter, markert med tykkere strek i oppgavene, ofte med vekslende bakgrunnsfarge; i 9×9 -sudoku er en boks på 3×3 ruter, mens i linsudoku består en boks av 2×3 ruter.

Brettet ovenfor er et 9×9 -brett ($n = 9$), dvs. rader, kolonner og bokser har alle 9 ruter. Det er også 9 rader, 9 kolonner og 9 bokser i brettet. 2 kolonner er markert med rød og grønn bakgrunnsfarge. Nederste (niende) rad er markert med fiolett, mens boksen nederst til høyre er rammet inn med gult. Det er vanlig å si at øverste rad er første rad, mens kolonnen lengst til venstre er første kolonne.

Disse begrepene vil i javaprogrammet finnes igjen som objekter. F.eks. vil hele brettet representeres av et objekt som bl.a. inneholder en todimensjonal array av ruter. Hver rute vil igjen inneholde informasjon om hvilken rad, kolonne og boks ruta ligger i.

Du skal lage programmet så generelt at det også kan løse sudokubrett som ikke har kvadratiske bokser. Brettet ovenfor er eksempel på et brett med kvadratiske bokser. Merk at 9×9 ikke er noen øvre grense for størrelsen på brettet. Eksempel på et ikke-kvadratisk brett ser du nedenfor (6×6).

En **sudokuoppgave** er et delvis utfylt brett som kan ha tre løsningsmuligheter:

1. én løsning (slike finner vi i aviser, bøker og blader)
2. ingen løsning (tallene er plassert slik at det ikke finnes en løsning)
3. flere løsninger (for få forhåndsutfylte tall)

Et tomt 9×9 brett har 6.670.903.752.021.072.936.960 løsninger.

Hovedtappene under utførelsen av programmet

- lese inn brettet
- opprette en datastruktur som tilsvare det innleste brettet
- finne alle løsninger og lagre disse i en beholder
- skrive ut løsningene til skjerm og fil
- skrive ut løsningene én etter én, når brukeren trykker på en knapp

Men utviklingen trenger ikke skje i denne rekkefølgen. Mer avansert GUI, f.eks. kan utvikles uavhengig av resten. Det samme gjelder innlesing fra fil, og beholderen som lagrer løsninger. Datastruktur og metoden som finner løsninger, derimot, henger tett sammen. Vi anbefaler at man begynner med datastrukturen. Dette er av erfaring den delen man trenger lengst tid på. Derfor er løsningen av denne skilt ut som det første du skal gjøre.

Filformatet vi bruker

For å lagre sudokuoppgaver har vi derfinert et eget filformat. Dette 6×6 -brettet beskrives av filen til høyre. Merk at filen ikke har blanke tegn i seg:

		3	6		
	2				4
5				6	
	3				5
3				1	
		1	4		

```

2
3
..36..
.2...4
5...6.
.3...5
3...1.
..14..

```

Første tall (2) er antall rader i hver boks, og neste tall (3) er antall kolonner i hver boks. $2 \times 3 = 6$. Brettet er da $6 \times 6 = 36$ ruter. 6 er også antall ruter i kolonne, boks og rad. Så følger selve brettet. Punktum (.) betyr tom rute. Hvis vi trenger mer enn 10 siffer, bruker vi $A = 10$, $B = 11$, osv. (Se sudokusiden for eksempler).

Programmodulen som leser fra fil og oppretter brettet kan løses tidlig eller senere. For å få poeng for denne delen, må den seinest tas med i leveringen

av oblig 10.

Del 9a—Les inn fra fil og datastruktur for brett og ruter

Denne delen kan du få rettet og testet i perioden 8. april - 28. april.

Størrelsen brettet og selve oppgaven (de sifrene som alt er fylt inn) leses fra fil.

Det er mulig å gjøre selve innlesningsdelen etter del 9b, men for å ha data å jobbe med er det greit å lage en metode som leser inn sudokuoppgaven fra fil med en gang. Den første oppgaven består derfor av å lage en metode som løper gjennom filen, bestemmer størrelsen på brett og boks og oppretter datastrukturen for brettet.

Klassen **Brett** må inneholde info om alle rutene i sudokubrettet. Info om hver rute lagres i objekter av klassen **Rute**.

Klassen **Rute** må inneholde en verdi (husk at noen av rutene ikke har en verdi ennå). For hver rute du leser inn; opprett et objekt av klassen **Rute** og lagre verdien. Lagre så hver rute i brettet (f.eks. i et array).

Lag så en utskriftsmetode som løper gjennom rutene i brettet ditt og skriver ut verdiene til skjerm. Utskriften fra eksempelfilen i oppgaven bør se sånn ut:

```
. . 3 6 . .  
. 2 . . . 4  
5 . . . 6 .  
. 3 . . . 5  
3 . . . 1 .  
. . 1 4 . .
```

Oppsummering: Lag en metode `lesFil()` som leser inn en fil og lagrer infoen i et objekt av klassen **Brett**. Lag så en utskriftsmetode som skriver ut det innleste brettet til skjerm.

Del 9b—utvid datastrukturen. Leveringsfrist 22. april

Denne delen kan du få rettet og testet i perioden 8. april - 28. april.

For å gjøre det mulig å løse sudokubrettet må klassen **Rute** i tillegg til verdi inneholde info om hvilken rad, kolonne og boks den tilhører.

Dette gjøres ved at *hver rute* inneholder pekere til ett objekt av klassen **Boks**, ett objekt av klassen **Rad** og ett objekt av klassen **Kolonne**. Ruter i samme rad skal peke på samme radobjekt osv.

Disse klassene skal brukes for å bestemme hvilke verdier som er «opptatt» i raden, kolonnen og boksen. Klassen **rad**, **kolonne** og **boks** må derfor inneholde info om hvilke verdier som allerede er tatt i den raden/kolonnen/boksen.

Lag en metode

```
void delInnRuter ()
```

som oppretter riktig antall rader, kolonner og bokser, for så å få hvert ruteobjekt i brettet til å peke på sine respektive rader, kolonner og bokser. For at inndelingen i bokser skal bli riktig må du ha tatt vare på de 2 første verdiene i fila som sier antall rader i hver boks og antall kolonner i hver boks.

Tegn datastrukturen til ruta i 3. rad og 2. kolonne i figuren under beskrivelsen

av filformatet. *Denne tegningen skal leveres i Devilry hvis din retter ikke har gitt fritak i forbindelse med retting.*

Skriv tilslutt en metode i klassen `Rute`

```
int [] finnAlleMuligeTall()
```

som returnerer en array med de tallene som er mulige løsningstall i en blank rute. F.eks. skal metoden kalt i øverste rute til venstre i eksempelbrettet på første side returnere en array med tallene 2, 3, 4, 7, 8 og 9.

Oppsummering: Utvid klassen `Rute` til å inneholde info om rad, kolonne og boks. Lag så metoden `int[] finnAlleMuligeTall()`. Husk tegning av datastrukturen til en rute.

Tips: Gi hver rad/kolonne/boks en unik ID, på denne måten kan du enkelt lage en testutskrift som løper gjennom brettet og skriver ut hvilken rad/kolonne/boks hver rute tilhører. Da ser man fort om man har gjort en feil. *Eller:* Lag en testmetode som løper igjennom alle rutene og som for hver rute skriver ut info om verdi, rad, kolonne og hvilke tall som er mulige løsningstall (ved kall på `finnAlleMuligeTall`)

Del 10a—løsningsmetoden og utskrift av løsninger

Del 10 og 11 kan bli endret noe ved behov. Det vil ikke bli noen endringer i hva som skal gjøres, men evt. bedre formuleringer, presiseringer og flere detaljer. Det skjer i såfall i ny versjon av obligteksten som publiseres seinest 16. april.

Denne delen kan du få rettet og testet i perioden 22. april - 5. mai

Løsningene skal finnes ved å gå gjennom alle rutene på brettet og prøve alle mulige (lovlige) verdier i hver eneste rute. Dette kalles «rå kraft»-metode («brute force» på engelsk).

Utvid klassen `Rute` med en metode, `fillUtDenneRuteOgResten`, som prøver å sette alle mulige løsningstall i seg selv. Det aller første denne metoden gjør er å kalle på metoden `finnAlleMuligeTall`. Deretter prøver den å sette alle disse tallene i denne ruta, ett om gangen. For hvert tall som settes i ruta kalles samme metode (`fillUtDenneRuteOgResten`) i neste rute (dvs den rett til høyre). Når en vannrett rad er ferdig (det finnes ingen rute rett til høyre), kalles metoden i ruta helt til venstre i neste rad, osv. Når et kall på `fillUtDenneRuteOgResten`-metoden i neste rute returnerer, prøver ruta neste tall som enda ikke er prøvd, osv. helt til alle tall er prøvd i denne ruta. Main-metoden starter det hele ved å kalle `fillUtDenneRuteOgResten` i den øverste venstre ruta. (Hint: Du kan gjerne lenke sammen alle rutene med en nestepeker, slik at en rute bare kan kalle `neste.fillUtDenneRuteOgResten`). Når metoden har funnet en lovlig verdi i den siste ruta (den nederst til høyre) på brettet, er en løsning funnet.

Løsningen(e) skal skrives ut på skjermen.

Hint: Under utviklingen kan det være lurt å først lage et program som genererer alle løsninger for et tomt brett for så senere å utvide med at noen av rutene kan ha forhåndsutfylte verdier. Ikke bruk et tomt brett med mer enn 9×9 ruter, da dette kan ta fryktelig lang tid.

Del 10b—utskrift og lagring av løsninger. *Leveringsfrist 6. mai*

Denne delen kan du få rettet og testet i perioden 9. april - 13. mai.

Filnavnet oppgis som parameter til programmet (på kommandolinja). Filformatet skal være slik som beskrevet over (retter vil teste programmet ditt med andre filer). Hvis det oppgis ett filnavn skal løsningen(e) skrives til skjerm. Hvis det oppgis to filnavn skal oppgaven løses fra den første filen, og løsningen(e) skrives på den andre filen (og ikke skrives til skjerm).

Filformatene for utskrift er beskrevet nedenfor.

Programmet skal inneholde class `SudokuBeholder` som igjen inneholder de tre offentlige metodene `settInn`, `taUt` og `hentAntallLosninger`. Du kan lage beholderklassen selv (da lærer du best), men kan også gjenbruke klassen fra en tidligere oblig, eller fra Javas API.

Programmet skal finne løsninger og legge dem inn i et objekt av klassen `SudokuBeholder`. Hvis det finnes flere løsninger enn 2500, skal beholderen holde orden på hvor mange løsninger som er funnet, men ikke ta vare på flere enn 2500 løsninger.

Oblig 11—framvisning av løsninger med vindusbasert GUI. *Leveringsfrist 13. mai*

Denne delen kan du få rettet og testet i perioden 6. - 20. mai.

I denne skal du lage et grafisk brukergrensesnitt (GUI) med `Swing` og `Awt` for å kommunisere bedre med brukeren og for å skrive ut løsninger. Altså 2 nye krav:

1. Programmet skal bruke `JFileChooser` til å finne/velge filen med oppgaven.
2. Et annet vindu skal vise fram løsningen(e) ved å hente den/dem fra beholderen etter at oppgaven er løst. Er det flere enn en løsning vises bare den første fram, men informasjon om hvor mange løsninger som totalt ble funnet.

I den avsluttende delen skal du utvide GUI-et med *lyttende knapper* slik at løsningene (hvis flere) kan vises fram en etter en når brukeren tryker på en knapp. Her kan du også legge inn andre nyttige GUI-funksjoner som f.eks. muligheten til å legge inn en oppgave direkte i vinduet.

Om progresjonen underveis

Retteperiodene under hver del er veiledende. Den enkelte retter kan avvike fra dette, men vær oppmerksom på at **retteperiodene strekker seg etter fristen!** De som ikke møter til retting i retteperioden kan ikke regne med å få tilbakemelding før siste del skal leveres.

Om tråder

I denne oppgaven trenger du ikke programmere med andre tråder enn maintråden og GUI-tråden. Maintråden starter med å opprette et eller flere vinduer for å lese inn data. Disse vinduene skal så lukkes når data er lest inn. Da startes selve løsningsalgoritmen. Når maintråden har funnet en løsning legges denne inn i en beholder. Når alle løsningene er funnet skal maintråden åpne et vindu der brukeren kan be om at en og en løsning blir vist fram ved å trykke på en knapp i vinduet.

Prøv å lage et robust program, dvs. et som ikke kræsjer når filformatet er feil eller noe annet uventet skjer.

Om du synes at noen av disse kravene er urimelige, eller du synes du kan løse oppgaven mer elegant eller bedre på en annen måte, ta det opp med din retter i forbindelse med rettingen av de tidlige delene. En forutsetning for å gjøre dette, er at man har møtt til retting underveis.

Om formatet for utskrift til skjerm/fil

Når det er få løsninger, kan du bruke samme format for å skrive løsningene ut til skjerm og fil. Punktum byttes ut med løsningstallet. Når det er mange løsninger kan du bruke det alternative utformatet som vises nedenfor. Denne sudokuoppgaven har 28 løsninger, de første 9 av dem er listet opp til høyre i alternativt utformat:

2	1: 421563//653214//134625//265431//512346//346152//
3	2: 421653//536214//143526//265431//612345//354162//
..1..3	3: 421653//536214//153426//264531//612345//345162//
.....	4: 421653//635214//513426//264531//142365//356142//
....2.	5: 451263//623415//135624//264531//512346//346152//
26....	6: 451263//623514//134625//265431//512346//346152//
...3..	7: 451263//632415//513624//264531//125346//346152//
3..1.2	8: 521463//643215//135624//264531//412356//356142//
	9: 521643//436215//143526//265431//612354//354162//

Retting underveis

Du bør minst ukentlig diskutere med retteren din hvordan du skal løse denne oppgaven. Pass på å hele tiden ha et program som kompilerer og kjører (men som i starten ikke gjør særlig mye). Du bør i det minste kontakte retteren din for å få tilbakemelding etter hver av de fem delene.

I tillegg kan det være greit å få tilbakemeldinger:

- når du har laget **main** og mange tomme klasser og har en første grove skisse av hele programmet ditt
- når du har bestemt formatet på løsningene slik de skal lagres i sudokubeholderen
- når du har laget en skisse av klassen **Rute** og dens subklasser
- når du har laget en skisse av klassene **Boks**, **Kolonne** og **Rad** og superklassen til disse klassene.
- når du har laget en skisse av GUI-programmet som henter ut løsninger fra sudokubeholderen og tegner dem ut.

Flere eksempler, eksempler på brett laget med **Swing**, flere sudokuoppgaver, samt utfyllende informasjon om sudoku finner du på [INF1010s Sudoku-side](http://heim.ifi.uio.no/inf1010/v15/oblig/91011/sudoku.html):

<http://heim.ifi.uio.no/inf1010/v15/oblig/91011/sudoku.html>

Her vil vi også legge ut ekstraoppgaver til dem som ønsker flere utfordringer og informasjon om vår programmeringskonkurranse!

Om alternativ datastruktur

Hvis du ønsker å ha variable i klassen **Rute** som kan brukes til å midlertidig begrense gyldige verdier i denne ruta, eller gjøre andre større endringer i datastruktur, skal du først diskutere dette med din retter og få tillatelse.

Lykke til med programmeringen!

Stein Michael Storleer