INF1010 2014 — Obligatorisk oppgave 5 — SUDOKU

Versjon 1.0—Feil vil bli rettet. Det vil komme en kommentar om hva som er rettet her, og versjonsnummeret vil endres hvis endringen kan ha betydning for forståelsen.

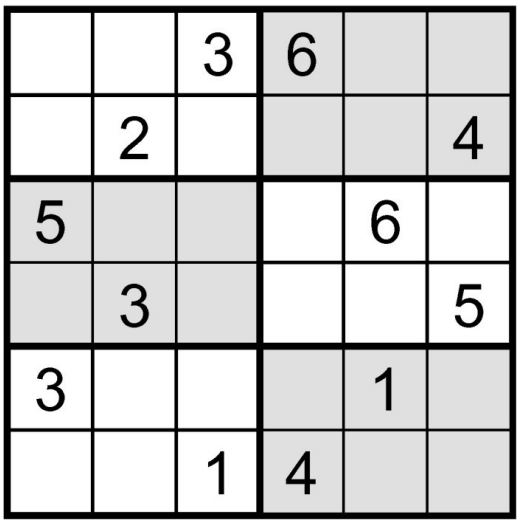
Denne obligatoriske oppgaven skal løses individuelt. Det betyr at alle skal levere fullstendig innlevering i Devilry og at alt som leveres skal du selv ha skrevet inn. Det er ikke lov å ta inn kode som er laget av andre. Hverken andre studenter, fra nettet eller andre kilder. Oppdages slik kode i noe 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): rekursjon, GUI, objektorientering, samspill mellom objekter, lenkelister, MVC ...

I denne oppgaven skal du lage et program som løser sudokuoppgaver. Oppgavene leses fra fil, og løsningen(e) skal skrives ut til terminalen og til fil. Løsningene skal finnes ved å gå gjennom alle rutene på brettet og prøve alle mulige (lovlige) verdier i hver eneste rute. Dette kalles en «rå kraft»-metode («brute force» på engelsk). I andre del skal løsningene vises på skjermen ved hjelp av et grafisk brukergrensesnitt (GUI).

Sudokubrettet

Et sudokubrett består av n × n ruter. Vi bruker følgende begreper i oppgaven:



- **rute** er er navnet vi bruker om den minste enheten på brettet; feltet som det kan stå ett tall (eller én bokstav) i.
- **brett** er alle n × 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 lynsudoku består en boks av 2×3 ruter.

Du skal lage programmet så generelt at det kan løse sudokubrett som ikke har kvadratiske bokser, som brettet ovenfor. Det har 6×6 ruter som har bokser på 2×3 ruter. Merk at 9×9 ikke er noen øvre grense for størrelsen på brettet.

Del 1—uten GUI

I denne obligatoriske oppgaven er en sudokuoppgave 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)

Hint: Under utviklingen kan det være lurt å først lage et program som genererer alle løsninger for et tomt brett (brett- og boksstørrelse eneste inndata),

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.

Besvarelsen din *skal* følge disse retningslinjene og den *skal* inneholde disse delene:

Programmet *skal* inneholde class SudokuBeholder som igjen inneholder de tre offentlige metodene settInn (eventuelt insert hvis du foretrekker engelske navn på identifikatorer) taUt (get), og hentAntallLosninger (getSolutionCount). Du kan lage beholderklassen selv (da lærer du best), men kan også gjenbruke en klasse fra lenkelistenotatet, eller fra Javas API. (Dette er et unntak fra regelen om bruk av andres kode).

Den første delen av programmet skal finne løsninger og legge dem inn i et objekt av klassen SudokuBeholder. Hvis det finnes flere løsninger enn det er plass til i sudokubeholderen, skal beholderen holde orden på hvor mange løsninger som er funnet, men ikke ta vare på flere løsninger en maksantallet, 750 løsninger.

Programmet ditt *skal* inneholde klassene Rute (Square) og Brett (Board) . Klassen Brett skal inneholde en todimensjonal tabell med pekere til alle rutene. Klassen Rute *skal* ha to subklasser, én for ruter som har en forhåndsutfylt verdi, og én for ruter der du skal finne en mulig verdi.

I tillegg *skal* du ha tre klasser som du kaller Boks (Box), Kolonne (Column) og Rad (Row). Du skal lage ett objekt av disse klassene for hver rad, kolonne og boks på brettet. Disse tre klassene *skal* ha en felles superklasse, og subklassene skal gjenbruke mest mulig av koden fra superklassen. Når en rute sjekker om den kan bruke en verdi, *skal* ruten kalle metoder som gjør dette i rutens kolonne-objekt, rad-objekt og boks-objekt. Klassen Rute *skal ikke* ha noen datastruktur (annet enn sin egen verdi) som sier noe om hvilke verdier som er lovlige i denne ruten. Dette siste «*skal ikke*»-kravet kan du fravike under gitte betingelser, se eget avsnitt lenger nede om dette.

Hver enkelt rute *skal* ha en metode, fyllUtRestenAvBrettet (fillInnRemainingOfBoard), som prøver å sette alle tall i seg selv (den prøver først med 1, så 2, så 3 osv.), og lykkes dette for et tall, kalles samme metode (fyllUtRestenAvBrettet) i neste rute (dvs den rett til høyre). Når en vannrett rad er ferdig (det finnes ingen rute rett til høyre), kalles metoden i ruten helt til venstre i neste rad, osv. Når et kall på fyllUtRestenAvBrettet-metoden i neste rute returnerer, prøver ruten neste tall som enda ikke er prøvd, osv. helt til alle tall er prøvd i denne ruten. Main-metoden starter det hele ved å kalle fyllUtRestenAvBrettet i den øverste venstre ruten. (Hint: Du kan gjerne lenke sammen alle rutene med en neste-peker, slik at en rute bare kan kalle neste.fyllUtRestenAvBrettet). Når metoden fyllUtRestenAvBrettet har funnet en lovlig verdi i den siste ruten (den nederst til høyre) på brettet, legges denne løsningen inn i beholderen.

Størrelsen brettet og selve oppgaven (de sifrene som alt er fylt inn) *skal* leses fra fil. Filnavnet oppgis som parameter til programmet (på kommandolinja). Filformatet skal være slik som beskrevet under (gruppelæren som skal godkjenne oppgaven din 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 er beskrevet nedenfor.

Om å fravike *skal ikke*-kravet om datastrukturen i klassen Rute

Hvis du ønsker å ha variable i klassen Rute som kan brukes til å midlertidig begrense gyldige verdier i denne ruten, bør du først diskutere dette med gruppelærer. Det er mer komplisert enn man tror og det advares mot å gjøre dette før man har laget ferdig en enklere versjon som virker.

Del 2 - med GUI

Når del 1 er ferdig, skal du lage et grafisk brukergrensesnitt (GUI) for å kommunisere bedre med brukeren og for å skrive ut løsninger. Altså 2 nye *skal*-krav:

1. Programmet *skal* bruke JFileChooser til å finne/velge filen med oppgaven (innfilen gitt som parameter i del 1).

2. Et annet vindu *skal* vise fram løsningen(e) ved å hente den/dem fra beholderen etter at oppgaven er løst. GUIet skal programmeres slik at løsningene kan vises fram en etter en. Brukeren trykker en knapp for å se neste løsning, maks 750. Hvis mer enn en løsning, skal vinduet også si hvor mange løsninger som er funnet.

Delene som har med GUI å gjøre bli gitt som ukeoppgaver i uka etter 26. mars og gjennomgått på vanlig måte. Programmet du lager da bør du lage slik at du lett kan bruke det i din egen løsning.

Om tråder (foreleses 26. mars)

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, så snakk med gruppelæreren som skal rette oppgaven din. Hvis du får skriftlig (på e-post) tillatelse så kan du løse oppgaven på en annen måte.

Tilbud om hjelp underveis

Du bør jevnlig diskutere med gruppelæreren 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 alle fall kontakte gruppelæreren din for å få en tilbakemelding:

• 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

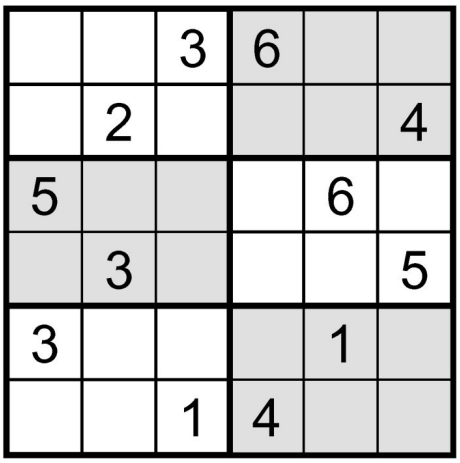
• Når du har laget en skisse av klassen Rute og dens subklasser

• 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.

Filformat

Dette 6×6-brettet beskrives av filen til høyre. Merk at filen ikke har blanke tegn i seg:




```
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).

Når det er få løsninger, kan man 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
3
..1..3
......
....2.
26....
...3..
3..1.2
```

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

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/v14/oblig/5/sudoku.html`

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

leveres i Devilry innen tirsdag 29. april 2014 kl. 23.59