Institutionen för
Data- och informationsteknik
JSk

TENTAMEN
2009-01-12

OBJEKTORIENTERAD PROGRAMMERING
för Z1 (TDA540)

OBS! Det kan finnas kurser med samma eller liknande namn på olika utbildningslinjer. Denna tentamen gäller *endast* för den eller de utbildningslinjer som anges ovan. Kontrollera därför noga att denna tentamen gäller för den utbildningslinje du själv går på.

TID 14.00 - 18.00

Ansvarig: Jan Skansholm, tel. 772 10 12 eller 0707-163230

Betygsgränser: Sammanlagt maximalt 60 poäng.
På tentamen ges graderade betyg:.
3:a 24 poäng, 4:a 36 poäng och 5:a 48 poäng

Hjälpmedel: Skansholm, *Java direkt med Swing*, valfri upplaga, Studentlitteratur.
(Understrykningar och mindre anteckningar i boken är tillåtna.)

Inga kalkylatorer är tillåtna.

Tänk på:

- att skriva tydligt och disponera papperet på ett lämpligt sätt.
- att börja varje ny (del)uppgift på nytt blad. Skriv endast på en sida av papperet
- Skriv den (anonyma) kod du fått av tentamensvakten på *alla* blad.

De råd och anvisningar som givits under kursen skall följas vid programkonstruktionerna. Det innebär bl.a. att onödigt komplicerade, långa och/eller ostrukturerade lösningar i värsta fall ej bedöms.

Uppgift 1) När man skall spela in ett TV-program på video är det ibland lite knöligt att räkna ut om bandet räcker. Detta skulle bli enklare om man hade ett program som räknade ut hur många minuter det är mellan två klockslag. Skriv ett sådant program. Programmet skall från två dialogrutor läsa in starttiden respektive sluttiden för programmet som skall spelas in. Tiderna skall anges med formen *tt.mm* i dialogrutorna. Som resultat skall programmet visa hur många minuter det är mellan de två tidpunkterna. Programmet skall även klara av det fall när man påbörjar en inspelning före midnatt och avslutar den efter. Tips: Räkna om de båda tiderna till antalet minuter som gått sedan det aktuella dygnets start.

(10 p)

Uppgift 2) En lärare brukar ha fem prov varje termin. För att hålla reda på elevernas resultat har han skrivit in alla resultaten i en textfil. I filen har han två rader för varje elev. På den första raden står elevens namn och på den andra elevens resultat på vart och ett av de fem olika proven (fem heltal). Vid läsårets slut vill han, för att kunna sätta betyg, få ut statistik från filen.

Skriv ett program som läser in informationen från textfilen. (Antag att filen heter `prov.txt`.) Elevernas namn skall sparas i ett vanligt endimensionellt fält inne i programmet. Provresultaten skall sparas i ett tvådimensionellt fält. Du kan anta att det finns högst 30 elever i en klass. Programmet skall sedan för var och en av eleverna skriva ut elevens namn följt av hans eller hennes genomsnittspoäng på de fem proven. Programmet skall slutligen för vart och ett av proven skriva ut genomsnittspoängen för alla elever.

(10 p)

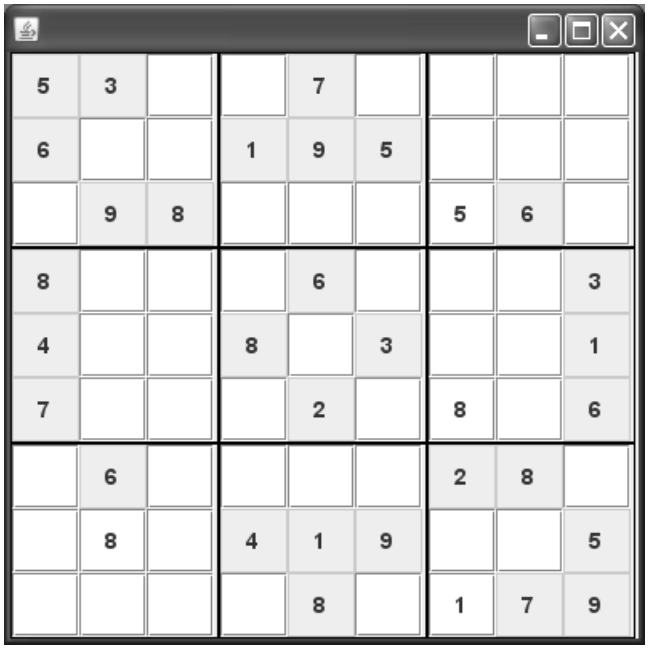
Uppgift 3) Antag att filen `ord.txt` innehåller ett antal ord. Det kan finnas ett eller flera ord på varje rad i filen och orden separeras med blanka tecken. Uppgiften är att skriva ett program som läser orden i filen och skriver ut alla orden i kommandofönstret. Orden skall skrivas ut med ett ord på varje rad. Utskriften skall ske i längdordning så att de kortaste orden skrivs ut först och de längsta sedan. Om flera ord har samma längd spelar det ingen roll i vilken ordning dessa skrivs ut inbördes. Först skrivs alltså alla ord med längden ett, därefter alla ord med längden två osv.

I denna uppgift får du *inte* använda dig av något fält (array). Du får inte heller skriva någon egen sorteringsmetod. Du skall istället läsa in orden i filen till en lista av någon klass som implementerar standardgränssnittet `List<String>` och sedan, så långt det är möjligt, utnyttja färdiga standardmetoder. (Tips: extern jämförare)

(10 p)

Uppgift 4) I Sudoku använder man normalt en spelplan som består av 9x9 rutor vilka kan innehålla siffrorna 1-9. Spelplanen är indelad i 9 s.k. regioner bestående av 3 rader och 3 kolumner, så som framgår av figuren.

Från början är några av rutorna ifyllda med siffror och kan inte ändras. Dessa är gråa i figuren. Det gäller att fylla i de övriga rutorna på så sätt att varje rad, kolumn och region kommer att innehålla siffrorna 1-9 och så att varje siffra bara förekommer en gång i en viss rad, kolumn eller region. I denna uppgift är målet att konstruera ett program som låter användaren spela Sudoku på datorn. När spelet börjar skall programmet visa ett fönster där de givna siffrorna redan är ifyllda. För att uppgiften skall bli lite mer lätthanterlig har den delats i ett antal *deluppgifter vilka kan lösas helt oberoende av varandra*. (Du behöver alltså inte ha löst en viss deluppgift för att få använda dig av den i en annan deluppgift.)



Följande klass är given och *skall* användas i deluppgifterna:

```
class Ruta extends JTextField {
   private int rad, kol;    // rutans plats
   private char tecken;      // tecken som skall visas

   public Ruta(int r, int k, char c) {
     rad = r; kol = k;
     setHorizontalAlignment(JTextField.CENTER);
     setFont(new Font("SansSerif", Font.BOLD, 14));
     setTecken(c);
   }

   public int getRad() { return rad; }

   public int getKol() { return kol; }

   public char getTecken() { return tecken; }

   public void setTecken(char c) {
     tecken = c;
     setText("" + c);
   }
 }
```

a) Skriv ett program som visar en spelplan enligt figuren ovan. Fönstrets storlek skall vara 400x400 pixlar. I denna deluppgift skall varje ruta i spelplanen vara blank. (Det skall alltså inte ännu visas några siffror i rutorna.) Klassen som programmet ligger i skall heta `Sudo`. Deklarera i klassen `Sudo` en instansvariabel med namnet `rutor` vilken skall vara ett tvådimensionellt fält med 9 rader och 9 kolumner, där komponenterna är av typen `Ruta`. I konstruktorn till klassen `Sudo` skall variabeln `rutor` initieras och det skall finnas kod som genererar ett fönster som ser ut som i figuren. Det skall finnas gränslinjer i fönstret så att man tydligt ser de nio regionerna, precis som i figuren.

Resten av klassen `Sudo` beskrivs i de följande deluppgifterna och behöver inte skrivas ännu.

(8 p)

b) Skriv i klassen `Sudo` en privat instansmetod med namnet `nyttSpel`. Metoden skall vara så utformad att man kan anropa den varje gång man vill påbörja ett nytt spel. Du kan t.ex. tänka dig att användaren har valt alternativet *Nytt spel* på en meny. Du kan också anta att denna metod anropas sist i konstruktorn i klassen `Sudo`. Metoden `nyttSpel` skall fylla i alla rutorna i spelplanen `rutor`. Först skall den i en dialogruta fråga efter namnet på en textfil. Filen förväntas innehålla de givna siffrorna. De rutor som skall vara blanka från början markeras i filen med nollor. Spelplanen i figuren ovan representeras t.ex. av en fil med innehållet:

```
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
```

De rutor som markeras med nollor skall i spelplanen innehålla ett blankt tecken och vara ändringsbara. De övriga rutorna skall visa den rätta siffran och vara icke-ändringsbara (Tips. metoden `setEditable`).

Du behöver inte kontrollera att den fil användaren anger finns. (Låt det bli en exception om den inte finns.) Däremot skall du kontrollera att filen bara innehåller enstaka siffror (dvs. tal i intervallet 1 till 9 med blanka tecken mellan) och att det finns minst 81 siffror i filen. Skulle så inte vara fallet skall en dialogruta med ett felmeddelande visas och programmet avbrytas.

(8 p)

c) Skriv i klassen `Sudo` en privat instansmetod med namnet `OK`. Metoden skall användas för att undersöka om det går att lägga in ett viss tecken i en viss ruta. Metoden skall ha tre parametrar: en **`char`** som innehåller det tecken man försöker lägga in och två **`int`**-parametrar vilka anger i vilken rad resp. kolumn man vill lägga in tecknet. Metoden skall ge ett returvärde av typen **`boolean`**. Ett blankt tecken skall alltid ge värdet **`true`**. Om tecknet inte är blankt och inte är en siffra i intervallet ’1’ till ’9’ skall metoden ge värdet **`false`**. Om tecknet är en siffra i intervallet ’1’ till ’9’ skall metoden, genom att undersöka fältet `rutor`, se om det går bra att lägga in siffran i den angivna rutan. (Den aktuella siffran får då inte redan finnas i den rad, kolumn eller region det gäller.) Går det bra skall värdet **`true`** returneras annars returneras värdet **`false`**.

(8 p)

d) Det sista steget är att lägga till en hanterare så att användaren kan skriva in siffror i de tomma rutorna. Uppgiften är att skriva en hanterare som tar hand om händelser av typen `KeyReleased`. (Anledningen till att man inte skall använda `ActionPerformed` är att användaren då skulle varit tvungen att trycka på Enter-tangenten efter varje inmatad siffra.) I hanteraren skall man använda metoden `getKeyChar` för att få reda på vilket tecken användaren skrivit in. Om det går bra att lägga in detta tecken i den ruta händelsen inträffade i (testa med metoden `OK`) så skall det inmatade tecknet läggas in i och visas i rutan. Om det inte går att lägga in det inmatade tecknet skall istället ett blankt tecken läggas in och visas i rutan. Programmet kommer i och med detta att förhindra att användaren skriver in siffror så att det blir dubbletter i någon rad-, kolumn eller region.

Visa också här vilken kod som måste läggas in i konstruktorn i klassen `Sudo` för att man skall komma till hanteraren när användaren skriver något i en ruta.

(6 p)