Institutionen för
Data- och informationsteknik
JSk

TENTAMEN
2008-08-29

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 08.30-12.30

Ansvarig: Jan Skansholm

Förfrågningar: 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 uppgift på nytt blad. Skriv endast på en sida av papperet
- Skriv ditt personnummer 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)

a) Vad blir det för utskrift när detta program exekveras?

```
public class C {
  public static void main(String[] argv) {
    int k = 1;
    int[] x = {0, 1, 2, 3};
    int[] y = x;
    lurig(x, y, k);
    System.out.println(x[0] + k + y[0]);
  }

  public static void lurig(int[] p, int[] q, int r) {
    p[0] = 1;
    q[0] = 2;
    r = 3;
  }
}
```

(2 p)

b) Vad skrivs ut när följande program exekveras?

```
public class EnKlass {
  private int x, y;
  private static int z;

  public EnKlass(int a, int b, int c) {
    x = a;
    y = b;
    z = c;
  }

  public String toString() {
    return x + " " + y + " " + z;
  }
}

public class StatTest {
  public static void main(String[] s) {
    EnKlass a = new EnKlass(1, 2, 3);
    EnKlass b = new EnKlass(4, 5, 6);
    System.out.println(a);
    System.out.println(b);
  }
}
```

(2 p)

c) Ange för var och en av följande rader om den är korrekt eller inte. Om en rad är felaktig så ange vad felet är.

```
/* rad 1 */ List<Integer> l1 = new List<Integer>();
/* rad 2 */ List<double>  l2 = new ArrayList<double>();
/* rad 3 */ LinkedList<String> l3 = new LinkedList<String>();
/* rad 4 */ List<l3> l4 = new LinkedList<l3>();
/* rad 5 */ LinkedList<String> l5 = new ArrayList<String>();
/* rad 6 */ List<String> l6 = new LinkedList<String>(l3);
```

(3 p)

Uppgift 2) Konstruera en klassmetod med namnet `tidsintervall` vilken får två tidpunkter `t1` och `t2` som parametrar och som räknar ut hur lång tid det är från `t1` till `t2`. Placera metoden i en klass med namnet `Tidklass`. Parametrarna och resultatet skall vara av typen `String` och ha formen `tt:mm:ss`. Observera att timmar, minuter och sekunder *alltid* anges med två siffror, t.ex. 08:17:05. Detta skall gälla även i resultatet. Metoden skall fungera även om `t2` anger en tid som är mindre än `t1`. I så fall skall man anta att `t2` har inträffar dygnet efter `t1`. Tips: Räkna om tiderna till sekunder.

(10 p)

Uppgift 3) För att kunna hålla ordning på alla deltagare i en löpartävling har man samlat information om dem i en textfil. Raderna i filen kan t.ex. se ut så här:

```
15:03:00 17:45:07 37 Anders Olsson 1972 Elitlöparna
15:08:00 16:59:59 4503 Nils Persson 1959 IK Spring
15:03:00 17:39:01 3506 Brandman Larsson 1965
```

Först kommer starttiden skriven enligt `tt:mm:ss`. (Eftersom det är så många deltagare startar inte alla samtidigt). Därefter kommer sluttiden (också enligt `tt:mm:ss`). För dem som inte kom i mål har sluttiden angivits som `00:00:00`. Resten av raden består av startnumret (varje deltagare har ett unikt startnummer), namn, födelseår och eventuell klubbtillhörighet. Mellan varje uppgift finns ett eller flera blanka tecken. Du kan anta att alla rader i filen följer detta mönster. Däremot vet du inte exakt hur många rader filen innehåller.

Man vill bestämma deltagarnas placering och behöver därför beräkna tiden det tog för varje deltagare att springa runt banan och sortera alla deltagarna efter denna tid. *Skriv ett program som läser innehållet i filen och som skapar en ny fil där personerna är sorterade efter hur fort de sprungit* (den som sprang fortast först). I den nya filen ska det först på varje rad stå tiden det tog att springa runt banan, sedan startnummer, namnet på personen, födelseår och eventuell klubbtillhörighet. Det kan t.ex. se ut på följande sätt

```
01:51:59 4503 Nils Persson 1959 IK Spring
02:36:01 3506 Brandman Larsson 1965
02:42:07 37 Anders Olsson 1972 Elitlöparna
```

Endast de löpare som kommit i mål skall finnas med i den nya filen. Namnen på den ursprungliga filen och den nya filen skall läsas in från dialogrutor.

Tips1: Du får gärna använda dig av metoden `tidsintervall` från uppgift 2 (även om du inte löst den uppgiften.)
Tips2: Objekt av klassen `String` är naturligt jämförbara. Man kan därför, innan man skriver ut raderna till den nya filen, lägga dem i en objektsamling som man antingen sorterar när alla rader lagts dit eller som har egenskapen att raderna sorteras automatiskt när de läggs in. (Använd en standardklass.)

(10 p)

Uppgift 4) Som bekant finns det i Java ett generisk standardgränssnitt med namnet `SortedSet` samt en generisk standardklass `TreeSet` som implementerar detta gränssnitt. I denna uppgift skall du emellertid anta att gränssnittet `SortedSet` och klassen `TreeSet` inte finns. (Däremot får du fortfarande anta att standardgränssnittet `List` samt de standardklasser som implementerar detta gränssnitt finns.)

Din uppgift är att konstruera en (vanlig icke-generisk) klass med namnet `SortedIntSet`. Med hjälp av denna klass skall man kunna skapa mängder i vilka elementen är av klassen `Integer`. Mängderna skall ha egenskapen att varje element bara får förekomma en gång samt att elementen ligger sorterade i mängden. (Klassen skall alltså kunna användas för samma ändamål som klassen `TreeSet`.)

Din klass skall egentligen ha samma uppsättning metoder som gränssnittet `SortedSet<Integer>` plus gränssnittet `Collection<Integer>`, men du behöver inte skriva alla dessa. De metoder du skall implementera är:

- `clear`
- `size`
- `contains`
- `remove`
- `first`
- `last`
- `add`
- `addAll`

Metoderna `contains`, `remove` och `add` skall ha en parameter av typen `Integer` och metoderna `first` och `last` skall ge ett resultat av typen `Integer`. Metoden `addAll` skall ha en parameter av typen `Collection<Integer>`.

Uppgiften är mycket lättare än den kanske verkar vid första anblick. Man kan nämligen använda sig av en teknik som kallas *delegering*. Då låter man en redan färdig, liknande klass göra det mest av jobbet. Om du internt i din klass använder dig av en lista av typen `List<Integer>` kan t.ex. de första fyra metoderna ovan implementeras genom att man helt enkelt anropar motsvarande metoder i den interna listan. De övriga metoderna som skall implementeras kräver lite mer kod men kan också konstrueras med hjälp av färdiga metoder för den interna listan. Tänk på att i metoden `add` så måste man se till att ett element bara får förekomma en gång och att elementen skall vara sorterade.

(10 p)

Uppgift 5) Skriv ett program som låter oss spela det s k 15-spelet. När programmet startas skall det komma upp ett fönster med knappar enligt vänstra figuren. Knappen längst ned till höger skall vara blank, medan övriga är försedda med nummer enligt figuren.

Man skall sedan oupphörligt kunna trycka på knapparna. Om man trycker på en knapp som har den blanka knappen som granne (dvs rakt ovanför eller nedanför sig eller till vänster eller höger om sig), skall texten på knapparna byta plats. Om man t.ex. trycker på knappen med texten "4" i den vänstra figuren så skall fönstret ändras så att det kommer att se ut som i den högra figuren. Om man trycker på den blanka knappen eller på en knapp som inte har blank granne skall programmet generera ett pipande ljud.

Tips: Låt varje ruta representeras av ett objekt av en egen klass `Ruta`, vilken skall vara en subklass till `JButton`. Om du låter varje ruta själv hålla reda på sin placering på spelplanen (sin rad och sin kolumn) så blir lösningen enkel. Om du dessutom hela tiden ser till att ha en referens till den tomma rutan så blir det ännu enklare.

(12 p)

Uppgift 6) Du har säkert kommit i kontakt med standardprogram som komprimerar filer så att de inte tar så stor plats eller går fortare att överföra via nätet. I denna uppgift skall du skriva ett program som komprimerar textfiler på ett mycket enkelt sätt. Programmet skall läsa en textfil och skapa en annan fil som innehåller samma information som den ursprungliga filen, men i komprimerad form. Den nya filen skall ha samma namn som den ursprungliga filen, men med tillägget ".szip". (Står för 'simple zip'.) Namnet på filen som skall komprimeras skall läsas från kommandoraden.

I en vanlig textfil lagras som du vet varje tecken som en 8-bitars teckenkod, en s.k. *byte*. Vid komprimeringen skall varje sekvens av fler än 3 likadana tecken översättas till en kompaktare form. Om den ursprungliga filen t.ex. innehåller sekvensen `aaaaaa` så skall denna i den komprimerade filen översättas till följande tre bytes: `¥•a`. Byte nr 1 innehåller teckenkoden för yen-tecknet (unicode \u00A5). Denna byte används som en markör för att signalera att de två bytes som följer skall tolkas på ett speciellt sätt. (Naturligtvis blir det problem om den ursprungliga filen skulle råka innehålla ett yen-tecken, men det kan du bortse från här.) Byte nr 3 innehåller teckenkoden för det tecken som ingick i sekvensen (i det här exemplet tecknet `a`). Byte nr 2, den som ovan markerats som •, innehåller sekvensens längd, i detta exempel värdet 6 eftersom sekvensen innehöll 6 tecken. Byte nr 2 innehåller *inte* någon teckenkod, utan den innehåller helt enkelt ett binärt värde i intervallet 0 till 255. En sekvens kan alltså bestå av högst 255 tecken. (Den fil som generas av programmet kan alltså inte läsas av en vanligt texteditor eftersom en sådan skulle försöka tolka det som finns i de bytes som markerats med • som vanliga tecken. För att kunna läsa filen måste man skriva ett annat program som översätter filen tillbaka till vanlig form, men det ingår inte i uppgiften.)

Här kommer ett exempel. Om den ursprungliga filen innehåller

```
Thisssssssss is aaaaaa
testtt 2222255555555555
```

så skall den komprimerade filen innehålla

```
Thi¥•s is ¥•a
testtt ¥•2¥•5
```

där de bytes som markerats med • innehåller värdena 8, 6, 5 resp. 11. Du ser att om en sekvens består av färre tecken än 4 så lönar det sig inte att komprimera den.

Använd standardklasserna `BufferedReader` och `BufferedWriter`. *Tips*: I klassen `BufferedWriter` finns metoden `write`. Denna skriver ut en byte. Metoden får som parameter en **`int`**, vars värde kommer att skrivas till utströmmen. (I normala fall innehåller parametern en teckenkod.) På motsvarande sätt finns det i klassen `BufferedReader` en metod med namnet `read` som läser en byte i taget från inströmmen och som returnerar det värde som finns i denna byte, eller -1 vid filslut. Returtypen är **`int`**.

(11 p)