Institutionen för TENTAMEN
Data- och informationsteknik 2018-03-03

PROGRAMMERINGSTEKNIK, del B
TIN213

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 - 11:30 SAL: SB-huset

Ansvarig: Jan Skansholm, tel 0707 163230

Betygsgränser: Sammanlagt maximalt 30 poäng.
På tentamen ges graderade betyg:
3:a 15 poäng, 4:a 20 poäng, 5:a 25 poäng

Hjälpmedel: Skansholm, *Java direkt med Swing*, Studentlitteratur, eller
Bravaco, Simonson, *Java Programming: From the Ground up*
(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 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) Utgå från följande två klasser `Klockslag` och `Observation`:

```
public class Klockslag implements Comparable<Klockslag> {
  int tim;
  int min;

  public int compareTo(Klockslag k) {
    if  (tim < k.tim || (tim == k.tim && min < k.min))
      return -1;
    else if (tim > k.tim || (tim == k.tim && min > k.min))
      return 1;
    else
      return 0;
  }

  @Override
  public String toString() {
    return String.format("%02d:%02d", tim, min);
  }
}

public class Observation implements Comparable<Observation> {
  int nr;
  Klockslag tid = new Klockslag();
  double temp;
  double vindhast;
  int vindrikt;

  public int compareTo(Observation annan) {
    return tid.compareTo(annan.tid);
  }
}
```

Antag sedan att du har en textfil `obs.txt` som innehåller väderobservationer. Varje rad i filen innehåller information om en observation och har formen:

```
stationsnr tim min temperatur vindhast vindriktning
```

Skriv ett program som läser informationen i filen och lagrar den i en lista med element av typen `Observation`. Låt programmet sortera listan så att observationerna hamnar i tidsordning. Skriv sedan ut väderstationens nummer, tidpunkten och temperaturen för alla observationer. När detta gjorts skall listan sorteras om så att den istället i första hand sorteras efter väderstationens nummer. *Tips*: Använd en extern jämförare. Programmet skall slutligen åter skriva ut stationsnummer, tidpunkt och temperatur för alla observationer.

(10 p)

Uppgift 2) I en del programspråk, C och C++ t.ex., kan man ha pekare till funktioner. Detta är inte tillåtet i Java. Man kan ju inte ha referenser till metoder. Men det är i Java möjligt att kapsla in en metod i en klass. Man kan sedan skapa ett objekt av denna klass och referera till objektet. Det går då att anropa metoden via referensen.

Deklarera ett gränssnitt (interface) med namnet `Function` som innehåller en metod med namnet `apply`. Denna metod skall beskriva en allmän matematisk funktion. Den skall alltså ha ett reellt tal som parameter och ge ett resultat av samma typ.

Deklarera därefter två klasser `Square` och `Root` som implementerar gränssnittet `Function`. Klassen `Square` skall kapsla in en metod vilken som resultat ger kvadraten på sin parameter och klassen `Root` skall kapsla in en metod som på motsvarande sätt ger kvadratroten ur sin parameter som resultat.

Skriv sedan en klassmetod med namnet `MakeList` Du kan placera metoden antingen i en egen klass eller i gränssnittet `Function`. (Detta är tillåtet från version 8 av Java.) Metoden `MakeList` skall som första parameter ha en lista `a` med reella tal. Metodens uppgift är att skapa och returnera en ny lista som har lika många element som listan `a`. Komponenternas värden i den nya listan skall vara avbildningar av motsvarande värden i listan `a`. Avbildningen skall ske med hjälp av en godtycklig matematisk funktion. Vilken funktion som skall användas skall styras med hjälp av ytterligare en parameter till metoden `MakeList`.

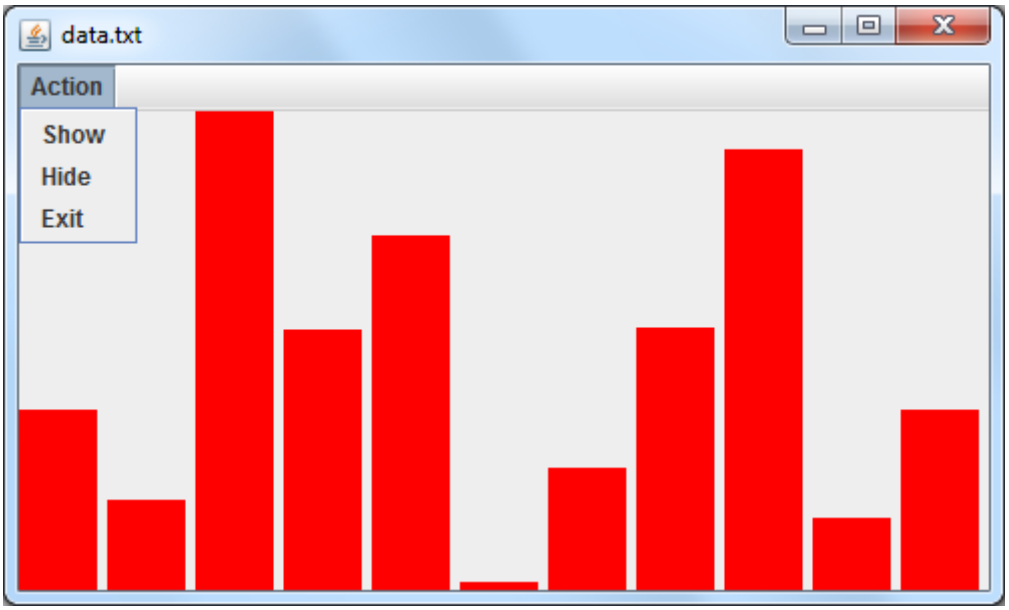
Skriv slutligen ett program som först läser in ett antal reella tal från kommandofönstret och som lägger de inlästa talen i en lista med namnet `l`. Låt programmet sedan skapa två nya listor, `kv` och `ro`. Dessa listor skall ha lika många element som listan `l`. Listan `kv` skall innehålla kvadraterna på alla talen i `l` och listan `ro` kvadratrötterna ur alla talen i `l`. Du *måste* använda dig av metoden `MakeList` för att skapa de två listorna `kv` och `ro`.

(10 p)

Uppgift 3) Antag att det finns en *färdigskriven* klass `Histogram` som är en subklass till `JPanel` och som har förmågan att visa histogram på skärmen. Klassen `Histogram` har följande publika egenskaper:

- `Histogram()` skapar ett tomt histogram (utan staplar),
- `Histogram(double[] a)` skapar ett histogram med lika många staplar som antalet element i `a`. Staplarnas höjder bestäms automatiskt av elementens värden.
- `void setValues(double[] a)` tar bort diagrammets ev. tidigare staplar och lägger dit lika många staplar som antalet element i `a`. Staplarnas höjder bestäms automatiskt av elementens värden.
- `void addValue(double d)` lägger till en ny stapel sist i diagrammet. Stapelns höjd bestäms av `d`:s värde. (Eventuellt skalas då också tidigare staplar om.)

Din uppgift är att skriva ett fullständigt program som, med användning av klassen `Histogram`, kan läsa in data från en fil och presentera de data filen innehåller i form av ett histogram. Det kan t.ex. se ut som i nedanstående figur. Längst upp i fönstret skall det finnas en meny med namnet *Action*. Menyn ska ha de tre alternativen *Show*, *Hide* och *Exit*.



Om användaren väljer alternativet *Show* skall programmet först ta bort ett eventuellt tidigare diagram från fönstret. (Fönstret skall alltså bli tomt.) Därefter skall det visa en dialogruta i vilken användaren kan skriva in namnet på en textfil. Filen förväntas innehålla ett godtyckligt antal mätvärden. Det kan stå ett eller flera mätvärden på varje rad i filen. Om filen existerar skall programmet läsa indata från filen och visa ett histogram i fönstret med lika många staplar som antalet värden i filen. Dessutom skall filens namn visas i fönstrets ram. Om filen inte existerar skall programmet ge ett felmeddelande i en dialogruta.

Om användaren väljer alternativet *Hide* skall programmet se till att fönstret blir tomt och att det inte står något filnamn i fönstrets ram.

Om användaren väljer alternativet *Exit* skall programmet avslutas.

(10 p)

```
// L?sningar till tentamen, del B 2018-03-03

// Uppgift 1
import java.io.*;
import java.util.*;

class ObsProgram {
  public static void main(String[] arg) throws IOException {
    List<Observation> l = new  ArrayList<>();
    Scanner fil = new Scanner(new File("obs.txt"));
    while (fil.hasNext()) {
      Observation obs = new Observation();
      obs.nr  = fil.nextInt();
      obs.tid.tim = fil.nextInt();
      obs.tid.min = fil.nextInt();
      obs.temp = fil.nextDouble();
      obs.vindhast = fil.nextDouble();
      obs.vindrikt = fil.nextInt();
      l.add(obs);
    }
    Collections.sort(l);
    for (Observation ob : l)
      System.out.println(ob.nr + "  " + ob.tid + "  " + ob.temp);
    Collections.sort(l,new JfrObs());
     for (Observation ob : l)
      System.out.println(ob.nr + "  " + ob.tid + "  " + ob.temp);
  }
}

class JfrObs implements Comparator<Observation> {
  public int compare(Observation o1, Observation o2) {
    if (o1.nr < o2.nr)
      return -1;
    else if (o1.nr == o2.nr)
      return o1.tid.compareTo(o2.tid);   // i andra hand efter tid
    else
      return 1;
  }
}


 // Uppgift 2
     import java.util.*;

     interface Function {
       public double apply(double x);

       public static List<Double> MakeList(List<Double> a, Function f) {
         List<Double>  b = new ArrayList<>();
         for (Double d: a)
           b.add(f.apply(d));
         return b;
       }

     }

     class Square implements Function {
       public double apply(double x) {
         return x*x;
       }
     }
```

```
    class Root implements Function {
      public double apply(double x) {
        return Math.sqrt(x);
      }
    }

      public static void main(String[] arg) {
        List<Double> l = new ArrayList<>();
        Scanner s = new Scanner(System.in);
        while (s.hasNext())
          l.add(s.nextDouble());
        List<Double> kv = Function.MakeList(l, new Square());
        List<Double> ro = Function.MakeList(l, new Root());
          }
        }


// Uppgift 3
  import java.io.*;

   public class HistogramProg extends JFrame implements ActionListener {
     JMenuItem[] mi = {new JMenuItem("Show"), new JMenuItem("Hide"), new
JMenuItem("Exit")};
    Histogram h = new Histogram();

    public HistogramProg() {
      JMenuBar mb = new JMenuBar();
      setJMenuBar(mb);
      JMenu amen = new JMenu("Action");
      mb.add(amen);
      for (JMenuItem it : mi) {
        amen.add(it);
        it.addActionListener(this);
      }
      add(h);
      setSize(500, 300);
      setVisible(true);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    private void clear() {
      setTitle(null);
      remove(h);
      h = new Histogram();
      add(h);
      repaint();
    }

    public void actionPerformed(ActionEvent e) {
      if (e.getSource() == mi[0]) {  // Show
        clear();
        String s = JOptionPane.showInputDialog("File name? ");
        if (s == null)      // St�ngningsrutan
          return;
        Scanner sc = null;
        try {
          sc = new Scanner(new File(s));
        }
        catch (FileNotFoundException ex) {
          JOptionPane.showMessageDialog(null, "File not found");
```

```
      return;
    }
    setTitle(s);
    while (sc.hasNextDouble())
      h.addValue(sc.nextDouble());
  }
  else if (e.getSource() == mi[1])    // Hide
    clear();
  else if (e.getSource() == mi[2])    // Exit
    System.exit(0);
}
```