Data och Informationsteknik / Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Erland Holmström

Göteborg 8 feb 2017

Examination i

PROGRAMMERINGSTEKNIK F1/TM1 TIN212

Dag: Måndag Datum: 2017-02-13 Tid: 8.30-13.30 (OBS 5 tim) Rum: ??

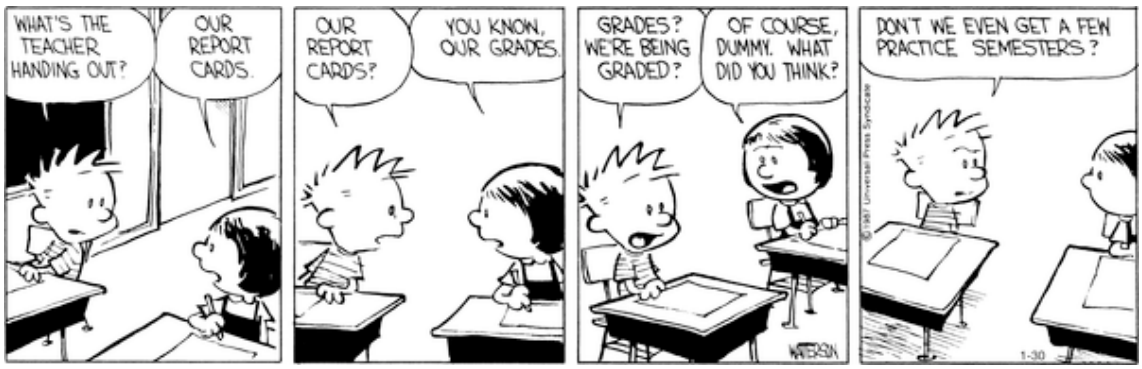
Ansvarig lärare:	Erland Holmström tel. 1007, 0708-710600
Resultat:	Skickas med mail från Ladok.
Lösningar:	Läggs eventuellt på hemsidan.
Tentagranskning:	Tentan finns efter att resultatet publicerats på vår expedition. Tid för klagomål på rättningen annonseras på hemsidan efter att resultatet är publiserat (eller maila mig så försöker vi hitta en tid).
Betygsgränser:	CTH: 3=28p, 4=38p, 5= 48p, max 60p
Hjälpmedel:	**Bravaco, Simonson: Java Programming From the Grounf Up**

Observera:

- Börja med att läsa igenom alla problem så du kan ställa frågor när jag kommer. Jag brukar komma efter ca 1-2 timmar (men det är typ 6 salar så det kan bli senare).
- Alla svar måste motiveras där så är möjligt/lämpligt.
- Skriv läsligt! Rita figurer. Lösningar som är svårlästa bedöms inte.
- Skriv kortfattat och precist.
- Råd och anvisningar som getts under kursen skall följas.
- Programs skall skivas i Java, skall vara indenterade och lämpligt kommenterade.
- Börja nya problem på ny sida.
-
- ***Lämna inte in tesen med tentan utan behåll den.*** Lämna inte heller in kladdpapper/skisser.

Lycka till!

Småfel rättade



Problem 1. Sant eller falskt? (Glöm inte motivera.)

a) Värdena i ett fält (en array) med primitiva data måste vara av samma typ.

b) Värdena i ett fält med objekt måste vara av samma dynamiska typ.

c) Värdena i ett fält med objekt måste vara av samma statiska typ.

d) En matris dvs ett tvådimensionellt fält i Java är ett fält av fält.

e) Av ett objekt kan man bygga flera klasser.

f) En metod kan inte returnera en klass.

g) Namnet på ett objekt är en pekare (referens).

7p)

Problem 2. Testar: aritmetik, talrepresentation

Följande program ger utskriften "false", förklara varför det blir så.

```
public class NineDivEleven  {
  public static void main(String[] args) {
    double f = 1/10.0 + 1/10.0 + 1/10.0;
    double g = 3/10.0;
    System.out.println(f==g);
  }
}
```

(3p)

Problem 3. Testar: klasser, toString, equals, fält, loopar

Du skall skriva en klass, `MyArray`, som innehåller och hanterar ett heltalsfält.

a) Det skall finnas två konstruktorer, den ena har ingen parameter och skapar ett fält med 10 platser. Den andra har en parameter som är fältets storlek.
Bägge fyller fältet med slumptal i intervallet [0..100[.
Bägge skall kolla att storleken är positiv annars kastas en lämplig exception med lämplig text.

b) Det skall finnas en (effektiv) toString metod som skriver ut fältet på formen
[35, 51, 33, 74, 41]
dvs kommaseparerade tal omslutna av [ ]. Inget komma efter sista talet.
Ett tomt fält skall skrivas som [tom].

c) Det skall finnas en equals metod.

d) Det skall finnas en "get" metod `public int[] getArray()` som returnerar det heltalsfält som lagras i klassen.

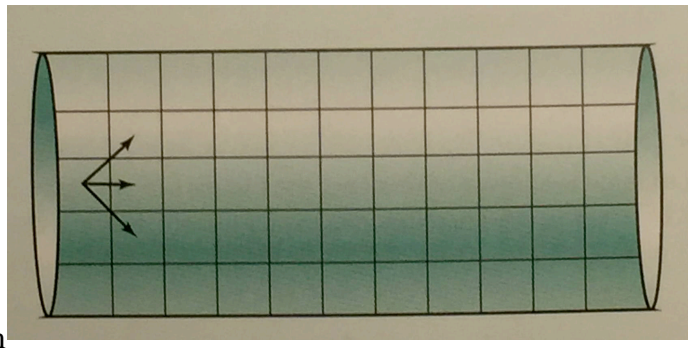
e) Skriv också en mainmetod som skapar två `MyArray` objekt med 10.000 platser, skriver ut det ena objektets innehåll och om fälten är lika.

För samtliga metoder gäller att regler och råd som gåtts igenom skall följas tex skall kod inte dubbleras och kod skall följa ”god sed” för inkapsling av objekt mm dvs sånt vi pratat om under kursen.

(Naturligtvis skulle det finnas fler metoder men vi nöjer oss med dessa)

(15p)

Problem 4. *Testar: rekursion och iteration, metoder, (detta är problem 3 på sid 344 i boken)*



Vi kan tänka oss att vi lägger ut en 2-dimensionell matris på en cylinder så att överdelen på matrisen sätts ihop med nederdelen enligt figuren. Varje ruta motsvarar ett tal i matrisen.

Uppgiften är att skriva två metoder som tillsammans hittar kostnaden för den kortaste vägen från vänstra sidan till högra sidan om man bara kan gå åt höger eller snett uppåt eller snett nedåt enl. figuren. En väg kan börja var som helst på vänster sida och sluta var som helst på höger sida. Kostnaden för vägen är summan av talen som passeras.

Den ena metoden skall vara en (iterativ) wrapper som i tur och ordning prövar att starta i alla positioner till vänster (använd en loop för det) och som anropar den andra metoden för varje rad. Den andra metoden skall då rekursivt beräkna kostnaden från rad i, kolumn 0 till höger sida. Antag att matrisen har dimensionen n x m.
Du behöver alltså inte redovisa vägen, endast dess kostnad.

Du behöver inte skriva någon klass men skriv hur du skulle anropa dina metoder och skriva ut resultatet.

Ett exempel:

12	9	15	**−7**	**−5**	28	−24	17
27	−19	16	−2	3	**7**	19	1
−1	12	23	14	10	45	**7**	−5
−10	**8**	12	71	−23	34	12	**−9**
26	17	**−3**	24	42	56	18	2

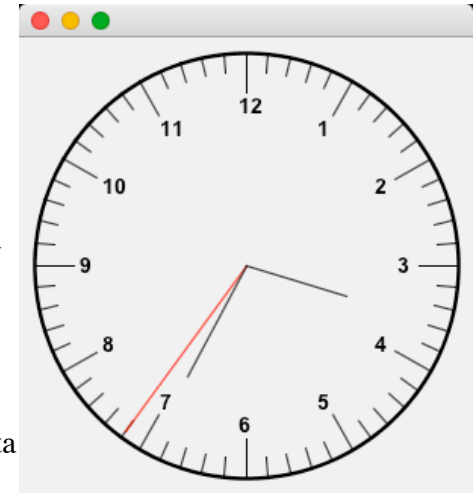
Här är en 5 x 8 matris där man får tänka sig att den översta raden ligger jämte den understa som om man lägger ut matrisen runt en cylinder. Talen i fetstil representerar en väg med kostnad -12. Det är inte den billigaste utan den billigaste har kostnad -33. Det finns två vägar med den kostnaden som börjar i kolumn 2 respektive 3.

(14p)

Note1: Den som kan sin %-operator vet att den inte funkar för negativa tal som vi vill här men för tentan så kan vi anta att den gör det så den kan användas för wraparound.

Note2: detta är en förenklad version av ett väldigt känt problem som kallas för ”Travelling salesperson”.

Problem 5. *Testar: händelsehantering, klasser, arv,*



I den här uppgiften tänkte jag att ni skall skriva delar av koden för en klocka enligt figuren.

Själva ritandet av klockan är ganska kul men också ganska tekniskt och passar inte så bra som tentauppgift så den delen får ni, se kod för `ClockView.java` i slutet. Ni behöver inte förstå speciellt mycket av den koden (så lägg inte tid på det) det räcker att man vet 2 saker (dessa är utmärkta i koden med en kommentar enligt: // ######### om du vill titta) och dom är

1. Konstruktorn vill ha 3 stycker objekt som indata av typen `ChainedCounterModel. En ChainedCounterModel` är en räknare och dessa tre räknare håller reda på timme, minut respektive sekund se mer nedan.

2. När klockan skall rita om sig så frågar den dessa objekt efter aktuella värden på timme, minut och sekund och ritar sedan upp sig enligt figuren ovan.

Det finns också en färdig modellklass, `CounterModel.java`, för räknaren enligt Javadoc dokumentation sist i tesen. Det är samma räknare som vi använt under kursen.

De två klasserna du skall skriva är:

a) `ChainedCounterModel`: Den ärver CounterModel men lägger till funktionalitet för att göra increment på en annan ChainedCounterModel. Konstruktorn har tre parametrar, `init` som är räknarens startvärde, `modulus` som anger hur långt den skall kunna räkna innan den gör wraparound och increment på räknaren som finns i parametern `counter`. För övrigt så överskuggar den bara increment-metoden från CounterModel. Övrig funktionalitet finns ju redan i CounterModel.

b) En `ClockMain` som lämpligen är en Jframe. Här skall du huvudsakligen:

- hämta aktuell tid från systemet så du kan starta klockan, se nedan,
- skapa räknare och initiera dom med den aktuella tiden,
- skapa en vy för klockan med hjälp av den givna klassen ClockView,
- skapa en timer och en klass som lyssnar på timern och ”tickar” klockan en gång varje sekund,
- och det man för övrigt brukar göra med en Jframe.

Aktuell tid kan man få fram genom följande satser. Värdena är som man förväntar sig dvs 0<=hour<12 osv. Du behöver inte skriva av satserna skriv bara ”hämta aktuell tid” på rätt ställe i den.

```
Calendar cal = Calendar.getInstance();
int hour = cal.get(Calendar.HOUR);
int min = cal.get(Calendar.MINUTE);
int sec = cal.get(Calendar.SECOND);
```

(Note: Det är inte jättemycket kod som skall skrivas, min ClockMain är 28 effektiva rader dvs exklusive kommentarer och blankrader men inklusive slutparenteser och min ChainedCounterModel är 14 rader.)

(16p)

Problem 6. *Testar: swing*

I sista uppgiften skall du skapa en multiplikationstabell enligt figur.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Strorleken på tabellen ges när man startar programmet enligt

```
java Multab 10
```

och är inte begränsat uppåt. Varje liten ruta med ett tal skall vara en knapp (JButton) och varje knapp skall ha storleken 30x30. Du behöver inga lyssnare, enbart det grafiska behövs.

Du skall använda en gridlayout i din kod och fönstret med tabellen skall skrivas ut med övre vänstra hörnet i position 40,40 på skärmen.

Man kan åstadkomma utseendet på knapparna ovan genom att ge en dom en ram enligt

```
butt.setBorder(new BevelBorder(BevelBorder.RAISED));
```

men det är inte nödvändigt.

(5p)

```
public class ClockView extends JPanel  {
  private final static int BORDER_WIDTH = 10;
  private BasicStroke bs;
  private Dimension d;
  private Font font;
  private int width;
  ChainedCounterModel ch, cm, cs, cms;

  public ClockView(ChainedCounterModel ch,  // #########
           ChainedCounterModel cm,
           ChainedCounterModel cs) {
    this.ch = ch; this.cm = cm; this.cs = cs;
    bs = new BasicStroke(2.5f);
    d = new Dimension(300, 300);
    setPreferredSize(d);
    font = new Font("Arial", Font.BOLD, 14);
    width = d.width-2*BORDER_WIDTH;
  }

  public void paintComponent(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);
    g2d.translate(BORDER_WIDTH, BORDER_WIDTH);
    Stroke stroke = g2d.getStroke();
    g2d.setStroke(bs);   / Paint oval.
    g2d.drawOval(0, 0, width, width);
    g2d.setStroke(stroke);
    int tickEnd = width/2;   // Paint tick marks.
    for (int second = 0; second < 60; second++) {
      int tickStart;
      if (second%5 == 0)
        tickStart = tickEnd-26; // long tick
      else
        tickStart = tickEnd-13;  // short tick
      drawSegment(g2d, second/60.0, tickStart, tickEnd);
    }
    g2d.setFont(font); // Paint hour labels.
    for (int hour = 1; hour <= 12; hour++) {
      double angle = (hour-3)*2*Math.PI/12;
      int x = (int) (Math.cos(angle)*(width/2-35))+width/2-5;
      int y = (int) (Math.sin(angle)*(width/2-35))+width/2+5;
      g2d.drawString(""+hour, x, y);
    }
    int hour = ch.getValue();     //##########
    int min  = cm.getValue();
    int sec  = cs.getValue();
    int ms   = 1; // vi behöver inte den upplösniingen
    int secHandMaxRad = width/2-5; // Paint hands.
    double fracSec = (sec+ms/1000.0)/60.0;
```

```
    g2d.setColor(Color.RED);
    drawSegment(g2d, fracSec, 0, secHandMaxRad);
    g2d.setColor(Color.BLACK);
    int minHandMaxRad = width/3-10;
    double fracMin = (min+fracSec)/60.0;
    drawSegment(g2d, fracMin, 0, minHandMaxRad);
    int hrHandMaxRad = width/4;
    drawSegment(g2d, (hour+fracMin)/12.0, 0, hrHandMaxRad);
  }
  private void drawSegment(Graphics2D g2d,
                           double fraction, int start, int end) {
    double angle = fraction*Math.PI*2-Math.PI/2.0;
    double _cos = Math.cos(angle);
    double _sin = Math.sin(angle);
    double minx = width/2+_cos*start;
    double miny = width/2+_sin*start;
    double maxx = width/2+_cos*end;
    double maxy = width/2+_sin*end;
    g2d.drawLine((int) minx, (int) miny,
                 (int) maxx, (int) maxy);
  }
}
```

Class CounterModel

java.lang.Object
    CounterModel

```
public class CounterModel
extends java.lang.Object
```

A class for a simple counter. Author: Erland H. Last changed: 2017-02-09

Constructor Summary

Constructors

Constructor and Description
CounterModel() Create a counter that can count up to the default value 10.
CounterModel(int modulus) Create a counter that can count between [0..modulus[.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	**decrement()** Decreases the counter one step.
int	**getModulus()** Returns the modulus for this counter.
int	**getValue()** Returns the value of the counter.
void	**increment()** Increase the counters value.
void	**reset()** Set the counter to 0.
void	**set(int v)** Give the counter the initial value v % modulus.
java.lang.String	**toString()** Returns the counter as a string

Lösningsskisser tenta TIN212 2017-02-13

Uppg 1.
a) sant, man deklarerer ju tex ett heltalsfält som int[] namn och
här finns inga möjligheter att lägga in något annat än int

b) falsk. Man kan mycket väl ha ett fält Person[] som innehåller studenter om studenter ärver Person. Alla subtyper till Person går bra, inklusive Person.

c) sant även om det låter motstridigt b. Detta kontrollerar kompilatorn så i ett fält Person[] kan vi bara lägga in objekt som har statisk typ Person.
b och c handlar alltså om vilken sida typen står
Person[] tmp = new Student[storlek]
Person är den statiska typen, Student är den dynamiska typen. Ser man det ur den statiska typens roll så måste alla objekt vara av den typen dvs Person. Som dynamisk typ dvs vid körningen, funkar alla som är subtyper till Person.

d) sant. Tänk på hur dekaration mm går till, man kan tänka sig parenteser i deklarationen och då blir det kanske tydligare (int[])[]

e) falskt. Man bygger inte klasser av objekt utan objekt av klasser.

f) falskt.
Egentligen borde man kanske fundera över vad en klass är här och vad frågan avser.
Är en klass klassnamnet eller texten som beskriver klassen eller en pekare till klassen eller något annat? Jag kan tänka mig att vi även kommer att ge poäng för "sant" här med lämplig motivering.
Jag tänker mig att ett svar som säger att metoden "Point metod()" returnerar en klass (även om den egentligen returnerar ett objekt av klassen Point) får poäng. Likaså ett exempel med metoden getClass som returnerar namnet. Det finns även metoder som returnerar pekare till klasser men det är utanför den här kursen (men får såklart också poäng).

g) sant. Namnet är en variabel som innehåller en pekare till själva objektet.

Uppg 2.
========== ========== ========== ========== ==========
Om man lägger in utskrift av f och g enligt
System.out.println(f);
System.out.println(g);
så blir det
false
0.30000000000000004
0.3
och då ser man att dom inte är lika.
Att f blir som den blir beror på att 0.1 inte kan representears exakt i datorn
på samma sätt som 0.3 inte kan representeras exakt i decimal talbas utan blir 0.333333...
Det är för övrigt samma sak med klassiska 9/11 = 0.8181818181818182 som också blir en oändlig decimalutveckling som avrundas i representationen, därav namnet på metoden.

Uppg 3.
========== ========== ========== ========== ==========
Den första konstruktorn och kravet att man skulle "testa 10an" gav många frågor. Jag tänker att ni skall veta att man skall återanvända kod och inte initialisera fältet med exakt samma kod på två ställen utan man skall göra this(10). Då testas värdet även om det är onödigt.

```
public class MyArray  {

    private int[] data;  // måste vara privat
// a) =========== ==========
    public MyArray() {
        this(10);      // återanvänd kod!
    }

    public MyArray(int size) {
        if (size<0) {
            throw new IllegalArgumentException("Ej negativ storlek");
        }
        data = new int[size];
        for (int i = 0; i<data.length; i++) {
            data[i] = (int) (Math.random()*100);
        }
    }
// b) =========== ==========
    public String toString() {
        StringBuilder buf = new StringBuilder(); // förutsättning för effektivitet
        buf.append("[ ");
        for (int p=0; p<data.length-1; p++){
            buf.append(data[p] );
            buf.append(", ");
        }
        if (data.length > 0) {
            buf.append( data[data.length-1] );
        } else {
            buf.append("tom");
        }
        buf.append(" ]");
        return buf.toString();
    }
// c) =========== ==========
    public boolean equals (Object o){
        if (this == o) {
            return true;
        } else if ( o == null
                    || getClass() !=
                        o.getClass()){
            return false;
        } else {
            MyArray rhs = (MyArray)o;
            if (data.length != rhs.data.length) return false;
            for(int i=0; i<data.length; i++) {
                if ( data[i] != rhs.data[i] ) {
                    return false;
                }
            }
            return true;
        }
    }
// d) =========== ==========
    public int[] getArray() {
        // som vi pratat om på föreläsning så måste man göra en kopia här annars
        // lämnar man ut den privata variabeln och då är det ju inte privat längre
        int[] tmp = new int[data.length];
        for(int i=0; i<data.length; i++) {
            tmp[i] = data[i];
        }
        return tmp;
    }
```

```
    public static void main(String[] args) {
// e) ========== ==========
        MyArray a = new MyArray(10000);
        MyArray c = new MyArray(10000);
        System.out.println(a);
        System.out.println("är a.equals(c) är " + a.equals(c));
// ) ========== ==========
```

Uppg 4.

```
========== ========== ========== ========== ==========
    //fältet antas vara typat int[][] cyl ≈ ....
    private static int n = cyl.length;      // nbr of rows
    private static int m = cyl[1].length;   // nbr of columns
    // return the smallest of a,b,c
    private static int min(int a, int b, int c) {
        if (a<b && a<c) {
            return a;
        } else if (b<a && b<c) {
            return b;
        } else {
            return c;
        }
    } // min

    // ************************************************************
    // a solution that that compute the cost of one of the best path
    // from the i'th row, j'th column
    // Du skall INTE undersöka/jämföra talen hos dina grannar, det ger inte
    // optimala lösningar, utan skall jämföra vägen från respektive granne till
    // andra sidan. Vägen kan man fråga sina grannar om.
    private static int path(int i, int j) {
        if (j == m-1) {
            return cyl[i][j];
        } else {
            return cyl[i][j] +
                min(path (mod(i-1, n), j+1),
                    path (i,           j+1),
                    path (mod(i+1, n), j+1)
                    );
        }
    } // end path

    public static int findMinPath() { // man kan ha matrisen som parameter om man vill
        //search for the smallest in the first column
        int smallest = Integer.MAX_VALUE;
        for (int i=0; i<=n-1; i++) {
            smallest = Math.min(smallest, path(i, 0));
        }
        return smallest;
    }
    // ************************************************************

    public static void main(String[] args) {
        System.out.println("ineffefficient best path is: " + findMinPath());
```

Uppg 5.

```
========== ========== ========== ========== ==========
public class ChainedCounterModel extends CounterModel {
    private ChainedCounterModel next;   // inget mer får deklareras här
                                        // speciellt så är det ett allvarligt fel att
                                        // ha en ny modulus eller en ny variabel för
                                        // en räknare här, dom finns i CounterModel

    public ChainedCounterModel(int init, int modulus, ChainedCounterModel counter) {
        super(modulus);
        next = counter;
        set(init);
    }

    public void increment() {
        super.increment();
        if ( getValue()==0 && next!=null ) {
            next.increment();
        }
    }
} // end ChainedCounter
===========
public class ClockMain extends JFrame {

    private ChainedCounterModel secCounter;

    public ClockMain () {
        Calendar cal = Calendar.getInstance();
        int hour = cal.get(Calendar.HOUR);
        int min = cal.get(Calendar.MINUTE);
        int sec = cal.get(Calendar.SECOND);
        //int ms = cal.get(Calendar.MILLISECOND);
        //int ampm = cal.get(Calendar.AM_PM);
        ChainedCounterModel hoursCounter = new ChainedCounterModel(hour ,24,null);
        ChainedCounterModel minsCounter = new ChainedCounterModel(min ,60,hoursCounter);
        secCounter = new ChainedCounterModel(sec ,60,minsCounter);
        //secsCounter = new ChainedCounterModel(ms ,10,secCounter);

        // skapa en vy
        ClockView clock = new ClockView(hoursCounter, minsCounter, secCounter );
        add(clock);

        Timer timer = new Timer(1000N, new mSecListener());
        timer.start();

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocation(100,100);
        pack(); //0,5
        setVisible(true);
    }

    // this is the controler
    private class mSecListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            secCounter.increment();
            repaint();
        }
    }
    public static void main(String[] args) {
        // det enda som behövs på tentan i main är
        // new ClockMain();
        // men så här borde det se ut
        Runnable r = new Runnable() {
            @Override
            public void run() {
                new ClockMain();
            }
        };
```

```
        EventQueue.invokeLater(r);
    }
}
```

Uppg 6.

```
========== ========== ========== ========== ==========
public class Multabell extends JFrame {
    // man skall inte spara om man inte behöver göra det. Här finns ingen anledning
    // att deklarera knapparna i ett fält, speciellt som det står att det inte finns
    // någon begränsning uppåt.
    public Multabell(int size) {
        setTitle("Multiplikationstabell");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(40,40);                        // flyttar fönstret
        setLayout(new GridLayout(size, size));
        JButton butt;
        for (int i = 1; i <= size; i++) {
            for (int j = 1; j <= size; j++) {
                butt = new JButton("" + i*j);
                butt.setBorder(new BevelBorder(BevelBorder.RAISED)); // behövdes ej
                butt.setPreferredSize(new Dimension(30,30));// här måste storleken sättas
                                    // setSize fungerar säkert också men då utan pack
                add(butt);
            }
        }
        pack();
        setVisible(true);
    }

    public static void main(String[] args) {
        new Multabell(Integer.parseInt(args[0]));
    }
} // end
========== ========== ========== ========== ==========
```