Data och Informationsteknik / Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Erland Holmström Göteborg 3 jan 2012

Examination in

PROGRAMMERINGSTEKNIK F1 TIN211

DAY: THURSDAY DATE: 2011-12-15 TIME: 8.30-13.30 ROOM: M

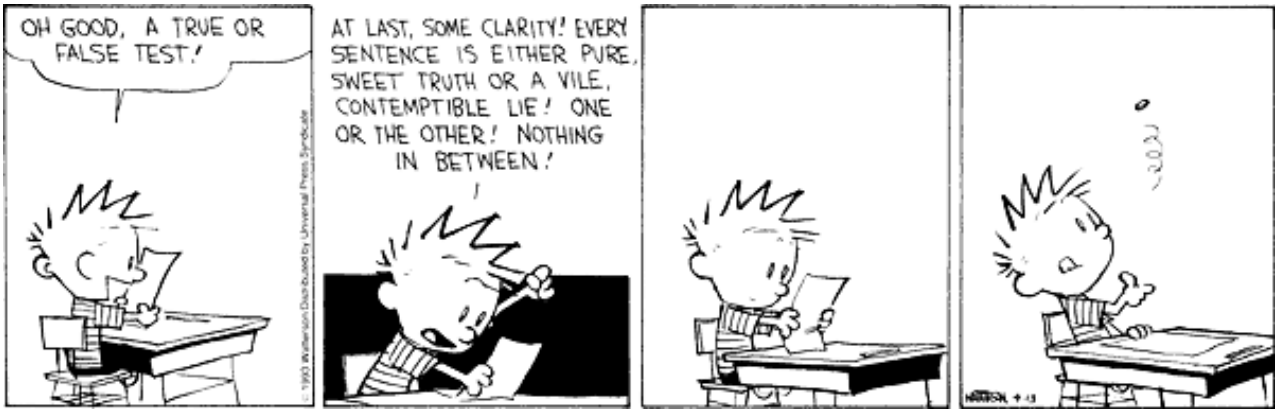
Responsible teacher:	Erland Holmström tel. 1007, home 270358
Results:	Are sent by mail from Ladok.
Solutions:	Are eventually posted on homepage.
Inspection of grading:	The exam can be found in our study expedition after posting of results.
	Time for complaints about grading are announced on homepage after the result are published or mail me and we find a time.
Grade limits:	CTH: 3=26p, 4=36p, 5= 46p, max 60p
Aids on the exam:	**Bravaco, Simonson Java Programming From the Grounf Up**
	or Horstmann Java Concepts and A print of Chapter 18. or Skansholm Java direkt.

(Observera att de gamla böckerna kommer att vara ok även i påsk, sen är det nog slut.

Observe:

- Start by reading through all questions so you can ask questions when I come. I usually will come after appr. 2 hours.
- All answers must be motivated when applicable.
- Write legible! Draw figures. Solutions that are difficult to read are not evaluated!
- Answer concisely and to the point.
- The advice and directions given during course must be followed.
- Programs should be written in Java, indent properly, use comments and so on.
- Start every new problem on a new sheet of paper.

Good Luck!



Problem 1. Sant eller falskt? Förklara. (förklaringen är vanligen viktigare än svaret)

a) En instansmetod kan alltid referera statiska variabler.

b) När ett objekt skickas som argument till en metod så kan metoden ändra argumentet.

c) När ett objekt skickas som argument till en metod så kan metoden ändra objektet.

d) En metod kan inte returnera en referens till ett objekt.

Svara och motivera:

e) Vilka av a,b och c i uttrycket `a.b(c);` kan vara null utan att kompilator/intrepretator klagar?

f) När man skriver en klass så skall man inte skriva en metod (en getter) som returnerar en referens till en privat variabel. Varför?

g) Beskriv kort vad som avses med en "har" relation och en "är" relation. Hur implementeras dessa relationer i Java?

4x0.5+1+2+2 (7p)

Problem 2. *Testar: enkla klasser, synbarhet, konstanter, exceptions, val, booleska uttryck, javadoc*

Ett vanligt sätt att beskriva en färg är att ange intensiteten för var och en av basfärgerna rött, grönt och blått med ett reellt tal mellan 0.0 och 1.0 eller ett heltal mellan 0 och 255. Java har en färdig klass för detta ändamål (`Color`), men här är det meningen att du skall göra en egen klass `MyColor` för detta.

- Klassen skall ha två konstruktorer

```
public MyColor(int red, int green, int blue)
public MyColor(double red, double green, double blue)
```

som skapar en färg med de angivna intensiteterna. Om någon av argumenten är utanför sina respektive intervall tex

```
new MyColor(256, 1, 257)
```

så skall en `IllegalArgumentException` kastas med texten

```
"MyColor: parameter outside of expected range:" färger
```

där ”färger” är en lista med de färger som var utanför intervallet, i exemplet ovan skulle det bli ”red blue” (obs alla som är utanför intervallet skall vara med inte bara en).

- Klassen skall ha en publik instansmetod `getRed` som returnerar den röda intensiteten. - Dessutom skall det finnas en metod med lämpligt namn som gör att

```
System.out.println(new MyColor(125, 125, 128));
```

skriver ut (talen skall naturligtvis kunna ändras)

```
[r=125, g=125, b=128].
```

- Det skall också finnas konstanter för de vanliga färgerna: (det räcker om du skriver kod för en av dem)

WHITE 255,255,255, BLACK 0,0,0, RED 255,0,0, GREEN 0,255,0, BLUE 0,0,255

- För den första av konstruktorerna skall du också skriva javadoc med lämpliga taggar (mer än en)

(12p)

Problem 3. *Testar: abstrakta klasser, polymorfism, dynamisk bindning, equals*

Antag att vi har en abstrakt klass `Shape` med variablerna `x` och `y` som representerar objektets placering och att klassen `Circle` med variabeln `radius` ärver den. Klassen `Cylinder`, med variabeln `length` ärver i sin tur klassen `Circle`.

```
public abstract class Shape{...
public class Circle extends Shape {...
public class Cylinder extends Circle {...
```

a) Vilka av följande deklarationer är ok? Motivera som vanlig.

```
Shape a = new Shape();
Shape b = new Circle();
Shape c = new Cylinder();
Cylinder d = new Circle();
```

Antag nu att `Shape` inte har någon equals metod implementerad. I `Circle` klassen finns

```
public boolean equals(Circle rhs) {
  return  getX() == rhs.getX()
       && getY() == rhs.getY()
       && radius == rhs.radius;
}
```

Vi skriver också följande klass för att söka efter ett objekt i ett fält

```
public class Search {
  public static int search(Object[] f, Object o) {
    for (int i=0; i<f.length; i++) {
      if ( o.equals(f[i]) ) {return i;}
    }
    return -1;
  }
} // end Search
```

och följande huvudprogramm

```
public class TestaGeom2 {
  public static void main(String[] args) {
    Circle c1 = new Circle(3.5, 1, 3);
    Circle c2 = new Circle(3.5, 2, 3);
    Circle c3 = new Circle(3.5, 3, 3);
    Circle c4 = new Circle(3.5, 4, 3);
    Circle c5 = new Circle(3.5, 5, 3);
    Object[] f = {c1, c2, c3, c4, c5};
    System.out.println(
                Search.search(f, new Circle(3.5, 5, 3)));
    System.out.println(Search.search(f, c5));
  }
}
```

b) Vad skriver huvudprogrammet ut? Du måste förklara varför det blir som det blir (och som vanligt ligger det stor vikt vid förklaringen).

c) Hur borde equals sett ut i `Circle`?

d) Varför bör det finnas en `equals` i `Shape` också? Hur bör den se ut?

(15p)

Problem 4. Simulering av skogsbrand

(Mycket att läsa men inte riktigt lika mycket att skriva) Stora skogsbränder är ett återkommande problem på många ställen (tex. i många medelhavsländer, Australien och delar av framförallt södra USA). Du ska skriva ett program som simulerar en skogsbrand, dvs. hur elden sprider sig över ett område under en viss tid.
(Att simulera hur en skogsbrand sprider sig är naturligtvis väldigt komplicerat, därför kommer vår modell att vara förenklad.)

Ett ganska vanligt sätt att simulera ett förlopp över ett område över en viss tid är att dela in området i mindre delar (celler) . Varje cell motsvarar en viss area i skogen (kanske så lite som ett träd), och varje cell har en viss status (tex. brinner, brinner inte, kan inte brinna). Med hjälp av regler beskrivs hur en cell byter från en status till en annan över tiden.

Vår modell

Vi tänker oss att skogsområdet är uppdelat i nxn celler. Varje cell motsvarar en liten area i skogsområdet, och varje cell kan anta en av följande värden (status)

- 0 - indikerar att det finns ingenting i cellen som kan börja brinna (cellen består tex. av sand, eller träd som redan har brunnit upp).
- 1 - indikerar att cellen brinner inte just nu (men det finns tex. träd i cellen som skulle kunna börja brinna).
- 2 indikerar att cellen brinner.

0	1	1	1	1
0	1	1	1	1
1	1	1	1	1
1	1	1	2	2
1	1	2	0	0

Exempel:

Området har här delats in i 5x5 celler. Tre av cellerna brinner (motsvaras av de rutor som innehåller siffran 2), fyra av cellerna kan inte börja brinna (de rutor som innehåller 0) och resten av cellerna brinner inte.

Regler för hur branden sprider sig över området

Om en cell i området brinner, kommer elden att sprida sig till omgivande celler med en viss sannolikhet efter en viss tid (Hur elden sprider sig beror ju på vindhastighet, hur torrt det är, om det finns något i omkringliggande celler som kan börja brinna, etc).

I vår (förenklade) modell gäller följande regler:

- Om en cell har status 0 (dvs det finns inget i cellen som kan börja brinna) - kommer den alltid att ha status 0.
- Om en cell har status 2 (dvs brinner)
  - kommer den i nästa tidssteg att ha brunnit upp och få status 0.
- Om en cell har status 1 och minst en granncell brinner (dvs. har status 2)
  - kommer cellen att börja brinna med p% chans. Med grannceller menas cellen rakt ovanför, rakt nedanför, till höger och till vänster.

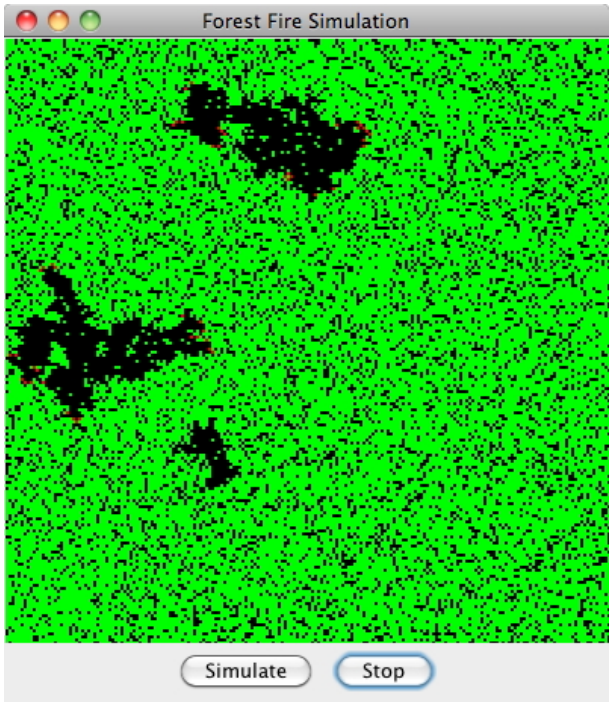
Du ska skriva ett program som simulerar skogsbränder enligt modellen ovan.

Storleken på matrisen anges på kommandoraden enligt

```
java Main 200
```

I figuren nedan är rutorna med status 1 gröna (kan börja brinna), status 0 är svarta (har brunnit upp) och de som har status 2 är röda (brinner).

Den lilla branden nertill har slutat brinna, de två andra brinner fortfarande, man ser kanske att det är en del röda punkter på randen.



Under själva figuren (brandsimuleringen) finns två knappar.

Genom att klicka på `Simulate`knappen körs en simulering som kan stoppas med `stop`knappen. Klickar man igen på `Simulate` så startas simuleringen om igen från början, samma sak om simuleringen gått färdigt. Simuleringen är färdig när inga celler brinner.

Programmet skall använda sig av Model-View-Control mönstret.

En vy (`ForestView(ForestModel m)`) finns redan skriven. Det är en `JPanel` som ritar upp matrisen (dvs inte knapparna) genom att anropa "`model.getModelValue(i, j)`" och jämföra innehållet med konstanter i modellen: "`desert`", "`tree`" och "`fire`".

a) *Testar: fält, metoder, enkla klasser, slumptal, synbarhet, loopar och val*

Skriv en modellklass (`ForestModel`) för att hantera "skogen" och själva simuleringen.

Modellklassen skall ha en metod som utför ett steg i simuleringen

```
boolean spreadOneGeneration()
```

som givet reglerna ovan räknar ut hur matrisen ändras under *ett* tidssteg. Kanske behöver man två matriser för att inte skriva sönder ursprungsmatrisen men du får inte kopiera i onödan.

Utgå från en slumpmässigt genererad matris och låt p=0.6. (Matrisen skapas av konstruktorn genom anrop av en metod `reset` som du också skriver. Glöm inte att det måste brinna nånstans också.)

Du behöver inte felhantera.

b) *Testar: swing, händelsehantering*

Skriv kontrollklassen (Control) som tillsammans med huvudprogrammet nedan skapar ett program enligt figuren ovan. Tiden styrs såklart med en Timer.

```
import javax.swing.*;
public class MainC {
    public static void main(String[] args) {
       ForestModel model = new ForestModel
                         (Integer.parseInt(args[0].trim()));
       Control control = new Control(model);
       JFrame frame = new JFrame("Forest Fire Simulation");
       frame.setLocation(50, 50);
       frame.add(control);
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.pack();
       frame.setVisible(true);
    }
}
```

(26p)

Programmeringsteknik F
Tenta 20111215
Uppg 1
4x0.5+1+2+2
a) ja den har tillgång till alla variabler.
b) Luring! både ja och nej kan accepteras med rätt motivering.
- Ja det går bra men bara inom metoden.
- Nej argumentet skickas med värdeanrop så utanför metoden ändras ingenting.
c) Ja det går att ändra "det pekaren pekar på""
d) metoder kan returnera vad som helst, även referenser.
e) endast c. a måste vara ett objekt, b ett metodnamn, bägge måste existera
f) För då kan mottagaren ändra objetets värden och det var ju det man ville förhindra.
g) se OH bilder. "Är" relationen avbildas med arv, "har" med referenser
==========
Uppg 2 se separat fil
==========
Uppg 3
a 4p
b 4
c 3
d 4
a)

```
Shape a = new Shape();  // inte ok, man kan inte instansiera en abstrakt klass
Shape b = new Circle();  // ok
Shape c = new Cylinder();  // ok
Cylinder d = new Circle();  // nej en circle kan inte vara en cylinder
```

b)
-1
4
I bägge anropen kommer systemet att leta efter en equals metod med signaturen
boolean equals(Object o).
Det finns en sådan i Objectklassen men den som finns i Cirkel klassen hittas
inte eftersom den har signatur med en Circle som parameter.
equals i Object klassen använder identitetstest dvs "==" och därför blir det
"hittas ej" -1 på första anropet eftersom där skapas ett nytt objekt.
I andra anropet däremot så är ju objekten identiska och då får vi positionen för
den cirkel vi söker efter.

c) equals i Circle borde haft signatur med Object och sett ut såhär

```
    public boolean equals(Object rhs) {
        return super.equals(rhs) && radius == ((Circle)rhs).radius;
    }
```

d)
Varje klass bör ta hand om "sina" parametrar.
Den bör se ut så här

```
    public boolean equals(Object rhs) {
        if( rhs == null || getClass() != rhs.getClass()) {
            return false;
        } else {
            Shape tmp = (Shape)rhs;
            return x == tmp.x && y == tmp.y;
        }
    }
```

==========
Uppg 4 se separat fil

```java
// Essentially copied (and simplified) from java.awt.Color
public class MyColor {

    public final static MyColor WHITE      = new MyColor(255, 255, 255);
    public final static MyColor LIGHT_GRAY = new MyColor(192, 192, 192);
    public final static MyColor GRAY       = new MyColor(128, 128, 128);
    public final static MyColor DARK_GRAY  = new MyColor(64, 64, 64);
    public final static MyColor BLACK      = new MyColor(0, 0, 0);
    public final static MyColor RED        = new MyColor(255, 0, 0);
    public final static MyColor GREEN      = new MyColor(0, 255, 0);
    public final static MyColor BLUE       = new MyColor(0, 0, 255);

    private int red   = 255;
    private int green = 255;
    private int blue  = 255;

    /**
    * Checks the color integer components supplied for validity.
    * Throws an IllegalArgumentException if the value is out of
    * range.
    * @param r the Red component
    * @param g the Green component
    * @param b the Blue component
    **/
    private static void testColorValueRange(int r, int g, int b) {
        boolean rangeError = false;
        String badComponentString = "";

        if ( r < 0 || r > 255) {
            rangeError = true;
            badComponentString = badComponentString + " Red";
        }
        if ( g < 0 || g > 255) {
            rangeError = true;
            badComponentString = badComponentString + " Green";
        }
        if ( b < 0 || b > 255) {
            rangeError = true;
            badComponentString = badComponentString + " Blue";
        }
        if ( rangeError ) {
            throw new IllegalArgumentException("MyColor: parameter outside of expected range:
                + badComponentString);
        }
    }

    /**
    * Creates an opaque sRGB color with the specified red, green, and blue
    * values in the range (0.0 - 1.0).
    *
    * @throws IllegalArgumentException if <code>r</code>, <code>g</code>
    *         or <code>b</code> are outside of the range 0.0 to 1.0, inclusive
    * @param r the red component
    * @param g the green component
    * @param b the blue component
    */
    public MyColor(double r, double g, double b) {
        this( (int) (r*255+0.5), (int) (g*255+0.5), (int) (b*255+0.5));
    }
```

```
    public MyColor(int intensity) {
        this(intensity, intensity, intensity);
    }

    /**
    * Creates an sRGB color with the specified red, green, blue, and alpha
    * values in the range (0 - 255).
    *
    * @throws IllegalArgumentException if <code>r</code>, <code>g</code>,
    *         or <code>b</code> are outside of the range 0 to 255, inclusive
    * @param r the red component
    * @param g the green component
    * @param b the blue component
    */
    public MyColor(int r, int g, int b) {
        red = r;
        green = g;
        blue = b;
        testColorValueRange(r, g, b);
    /*
    så här görs det i java.awt.Color dvs man lagrar färgen i *en* int där
    varje delfärg upptar en byte. Komponenten a är genomskinlighet.
     value = ((a & 0xFF) << 24) |
                ((r & 0xFF) << 16) |  // << är vänstershift
                ((g & 0xFF) << 8)  |
                ((b & 0xFF) << 0);
    testColorValueRange(r,g,b,a);
    */
    }

    public int getRed() {return red;}
    public int getGreen() {return green;}
    public int getBlue() {return blue;}

        /**
     * Returns a string representation of this <code>MyColor</code>. This
     * method is intended to be used only for debugging purposes.  The
     * content and format of the returned string might vary between
     * implementations. The returned string might be empty but cannot
     * be <code>null</code>.
     *
     * @return  a string representation of this <code>MyColor</code>.
     */
    public String toString() {
        return getClass().getName() + "[r=" + getRed() + ", g=" + getGreen() + ", b=" + getBl
    }

    public static void main(String[] args) {
        MyColor a = new MyColor(255, 0, 0);
        System.out.println(a);
        MyColor b = new MyColor(1.0, 0.0, 0.0);
        System.out.println(b);
        MyColor c = new MyColor(255, 0, 255);
        System.out.println(c);
    }

} // end MyColor
```

```
public class ForestModel {
    private int size = 0;
    private int[][] model;      // the forest - detta är modellen
    private int[][] shadowM;    // fältet i vilket vi gör uppdateringar
    private int[][] tmp;        // bara för att kunna swapa de ovan

    private int nbrOfFireCells = 0;

    public final int desert = 0;
    public final int tree   = 1;
    public final int fire   = 2;

    public ForestModel(int size) {
        this.size = size;
        model = new int[size+2][size+2]; // marginal för att slippa testa gränser
        shadowM = new int[size+2][size+2];
        //reset();
    }

    // getters
    public int getSize() {
        return model.length-2; // or return size-2;
    }

    public int getModelValue(int x, int y) {
        x=x+1;
        y=y+1;
        if ( !(x>=1 && x<=size && y>=1 && y<=size) ) {
            throw new IllegalArgumentException("getModelValue: outside forest");
        } else {
            return model[x][y];
        }
    }

    public void reset() {
        int emptyCell = 2; // ca 20% tomma celler
        for(int i=1; i<model.length-1; i++) {
            for(int j=1; j<model.length-1; j++) {
                model[i][j] = tree;
                if( 10*Math.random() < emptyCell ) {
                    model[i][j] = desert;
                }
            }
        }
        startAFire(); startAFire(); startAFire();
    }
    /*
    private void plantATree(int x, int y) {
        if ( !(x>=1 && x<=size && y>=1 && y<=size) ) {
            throw new IllegalArgumentException("plantATree: outside forest");
        } else {
            if (model[x][y] == 2) {
                nbrOfFireCells--;
            }
            model[x][y] = tree;
        }
    }
    */
    private void startAFire() {
        int x = (int)(size*Math.random()+1);
        int y = (int)(size*Math.random()+1);
```

```
        model[x][y] = fire;
        nbrOfFireCells++;
    }

    private boolean anyNeighboursOnFire(int x, int y) {
        if ( !(x>=1 && x<=size && y>=1 && y<=size) ) {
            throw new IllegalArgumentException("plantATree: outside forest");
        } else {
            return
                (  model[x+1][y]   == fire // right
                || model[x]  [y-1] == fire // down
                || model[x-1][y]   == fire // left
                || model[x]  [y+1] == fire // up
                );
        }
    }

    // ingick inte i tentan
    public boolean spread(int sp) {
        while ( sp>0 ) {
            sp--;
            if ( !spreadOneGeneration() ) {
                return false;
            }
        }
        return true;

    }

    public boolean spreadOneGeneration() {
        for(int i=1; i<model.length-1; i++) {
            for(int j=1; j<model[i].length-1; j++) {
                if ( model[i][j] == desert ) {
                    shadowM[i][j] = desert;
                } else if ( model[i][j] == fire ) {
                    shadowM[i][j] = desert;
                    nbrOfFireCells--;
                } else if ( model[i][j] == tree ) {
                    if ( anyNeighboursOnFire(i, j) && Math.random()<0.7 ) {
                        shadowM[i][j] = fire;
                        nbrOfFireCells++;
                    } else {
                        shadowM[i][j] = tree;
                    }
                }
            }
        }
        //swap matrix
        tmp = model;
        model = shadowM;
        shadowM = tmp;
        return nbrOfFireCells!=0; // false when nbrOfFireCells is == 0
    }

    public String toString() {
        StringBuffer s = new StringBuffer();
        for(int i=0; i<model.length+2; i++) {
            s.append("-");
        }
        s.append("\n");
        for(int i=0; i<model.length; i++) {
```

```
            s.append("|");
            for(int j=0; j<model.length; j++) {
                if (model[i][j] == tree) { // tree not burning
                        s.append("^");
                } else if (model[i][j] == fire) { //  burning
                        s.append("*");
                } else {  // empty cell
                    s.append(" ");
                }
            }
            s.append("|");
            s.append("\n");
        }
        for(int i=0; i<model.length+2; i++) {
            s.append("-");
        }
        s.append("\n");
        return s.toString();
    }
}
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Control extends JPanel {

    private ForestModel m;
    private ForestView v;
    private Timer timer;
    //private int speed = 5;

    public Control(ForestModel model) {
        m = model;
        v = new ForestView(m);
        setLayout(new BorderLayout());
        add(v, BorderLayout.NORTH);
        JButton runButton = new JButton("Simulate");
        JButton stopButton = new JButton("Stop");
        JPanel bp = new JPanel();
        bp.add(runButton);
        bp.add(stopButton);
        runButton.addActionListener(new RunListener());
        stopButton.addActionListener(new StopListener());
        //changeSpeedButton.addActionListener(new ChangeSpeedListener());
        add(bp, BorderLayout.SOUTH);
        timer = new Timer(5, new StepListener());
    }

    private class RunListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            m.reset(); // börja om
            v.repaint();
            timer.start();
        }
    }

    private class StopListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            timer.stop();
        }
    }


    private class StepListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if ( !m.spread(2) ) {
                timer.stop();
            }
            v.repaint();
        }
    }
}
```