

Java 1.1 Applets

<http://worgtsone.scienceontheweb.net/worgtsone/> - <mailto:worgtsone@hush.com>

16. August 2006 – 13. Oktober 2011

Inhaltsverzeichnis

1	What an Applet is	2
2	My first Applet	3
2.1	Screenshot	3
2.2	HTML	3
2.3	Source	3
3	Animated Applet	4
3.1	Screenshot	4
3.2	HTML	4
3.3	Source	5
4	Animated Applet with Mouse Input	6
4.1	Screenshot	6
4.2	HTML	6
4.3	Source	7
5	Sound - 2 B done	8
6	Self-Drawing Objects	9
6.1	Screenshot	9
6.2	HTML	9
6.3	Ball.java	9
6.4	BallGod.java	10
7	Interaction: Slurping	12
7.1	Screenshot	12
7.2	HTML	12
7.3	BallSlurp.java	12
7.4	BallSlurpGod.java	14
8	Interaction: Bumping	16
8.1	Screenshot	16
8.2	BallBump.java	16
8.3	BallBumpGod.java	17
9	MouseTracking	19
9.1	Screenshot	19
9.2	MouseTracker.java	19

10 MouseCube	21
10.1 Screenshot	21
10.2 MouseCube.java	21
11 MouseCube2	24
11.1 Screenshot	24
11.2 MouseCube2.java	24
12 Dodecaeder	27
12.1 Screenshot	27
12.2 Dodecaeder.java - Changes from MouseCube2.java	27
13 Sphere	29
13.1 Screenshot	29
13.2 makePoints() - renewed	29
14 A Deeper Look into Threads	30
14.1 TestG.java	30
14.2 TestG.java	30
15 Web-Application : Visualisation of Sorting Algorithms	32
15.1 Three Screenshots	32
15.2 Some Sad Words about Layouts	33
15.3 Action Detection	34
15.4 Threading is necessary...	35
15.5 SortingA.java	35

Disclaimer

Wissen ist zum Teilen da. Ich teile mein Wissen mit Ihnen, lieber Kollege.
 Ich bin aber nicht perfekt. Unter worgtsone@hush.com
 nehme ich dankbar Ihre Verbesserungsvorschläge entgegen.

*

Legal Blurb: Alle Informationen in diesem Dokument sind falsch, unvollständig,
 irreführend, irrelevant und / oder funktionieren einfach nicht.

Wenn Sie es trotzdem benutzen, und es geht dabei etwas kaputt, ist das Ihr
 Problem, nicht meins.

*

Bitte teilen Sie meine Web-Adresse nicht Ihren Schülern mit.

1 What an Applet is

An Applet is a rectangular graphic on a webpage. It has a height and a width. Its bound to the webpage with the `<applet>` tag.

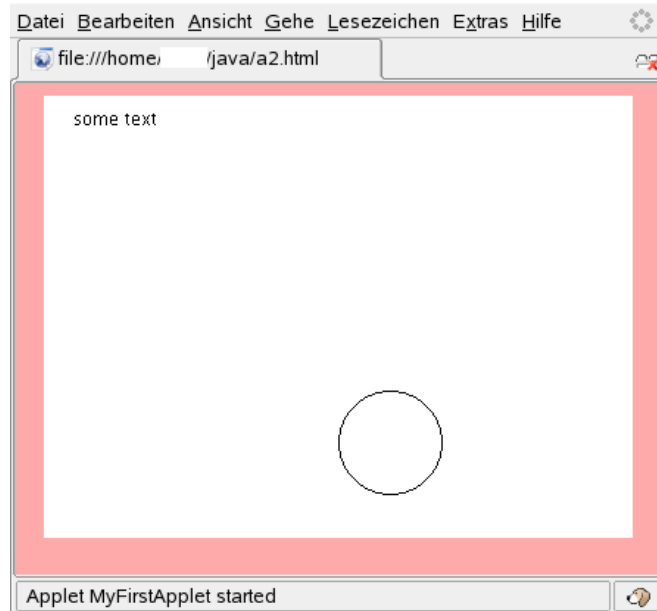
As in all object-orientated programming languages, theres a huuuuuuuuuuuge lot of pre-defined classes that you can inherit.

We will inherit from Applet, a class who knows how to have a height, a width, and how to appear on a webpage. Note: The most specialized classes usually are the most useful.

2 My first Applet

We need – a website, displaying the applet – and an applet that shows a circle and some text. I call this minimalistic artistic.

2.1 Screenshot



2.2 HTML

```
<!-- MyFirstApplet.html - minimalistic, eh? --!>
<applet width=400 height=300 code=MyFirstApplet.class></applet>
```

The above is html for "Hey browser, load and display applet MyFirstApplet, at a height of 300 and a width of 400. Applet end."

2.3 Source

```
//this MUST be MyFirstApplet.java
import java.applet.*;    // import all classes related to applets
import java.awt.*;       // maybe something useful in there

public class MyFirstApplet extends Applet {

    public void start () { // no init needed.
        repaint();       // we DO paint, once.
    }

    public void paint (Graphics g) {
        g.drawOval (200,200,70,70);
        g.drawString ("some text", 20,20);
    }
}
```

3 Animated Applet

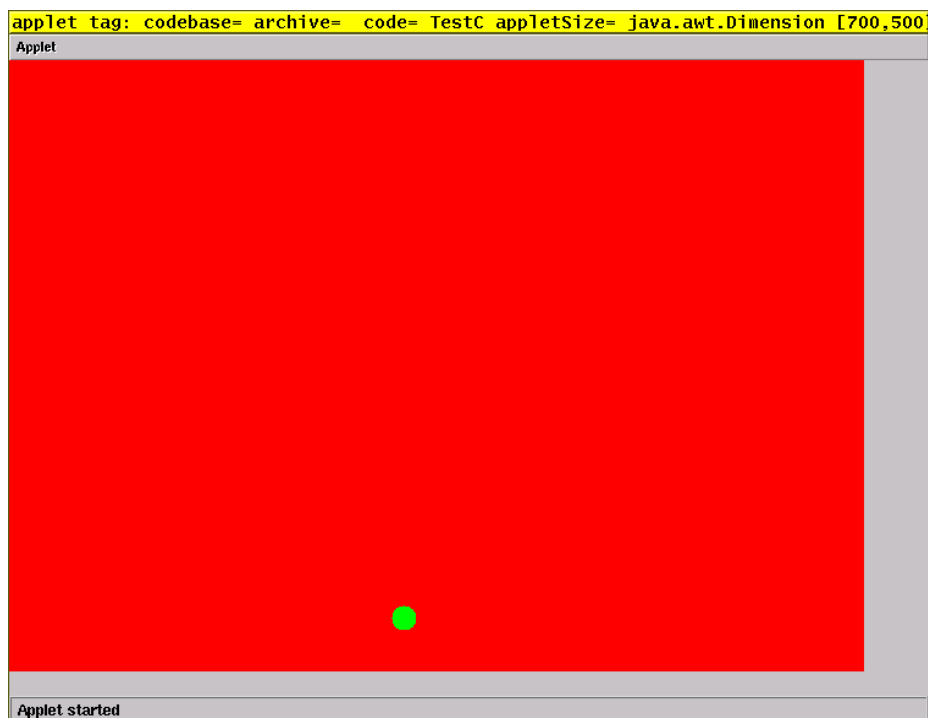
The source should go like:

1. `init();`
2. `for time=1 to 1000 do (`
3. `draw something, depending on time`
4. `sleep a while`
5. `)`

But its not that easy. We MUST implement `Runnable`, that is make the Applet be a Thread, too.

1. Be an Applet. Be a Thread, too;
2. `init` : start, meaning: run yourself;
3. `while true (`
4. `draw something, depending on time`
5. `sleep a while`
6. `)`

3.1 Screenshot



The green ball rotates counter-clockwise.

3.2 HTML

```
<html><applet width=700 height=500 code=TestC.class></applet></html>
```

3.3 Source

```
import java.awt.*;
import java.applet.*;

// is an Applet
// is also a Thread (Runnable)
// guido krueger says: not to eat all browser cpu time
// i say: to be displayed at all
public class TestC extends Applet implements Runnable {
    int x, y;
    Thread th = new Thread (this);          // make it short

// no constructor

    public void init () {
        th.start ();                        // for some queer reason, performs run()...
    }

// no start

    public void run ()    {
        System.out.println ("run...");
        setForeground (Color.green);
        setBackground (Color.red);
        for (int i = 0; i < 500; i++) {
// lets hope its a 700x500 window...
            x = (int) (Math.sin (0.05 * i) * 200 + 350);
            y = (int) (Math.cos (0.05 * i) * 200 + 250);
            repaint ();
// its a thread, so it can sleep()
            try { th.sleep (10); } catch (Exception e) { }
        }
        System.out.println ("loop has finished.");
    }

    public void paint (Graphics g) {
        g.fillOval (x, y, 20, 20);
    }
}
```

4 Animated Applet with Mouse Input

For some queer reason, on my Redhat 6.2 system with jikes and java-version 1.1, Applets can get neither Keyboard input nor actionPerformed input.

But Mouse input.

1. be a Applet. Additionally, be a Thread and a MousListener; (???)
2. init : grab the mouse and run yourself;
3. run : while true (
 4. draw something, depending on time
 5. sleep a while
6.)
7. if theres mouse input, do something : change direction.

4.1 Screenshot



The green ball first rotates counter-clockwise. On each Mouse click, it changes direction.

4.2 HTML

```
<html><applet width=700 height=500 code=TestD.class></applet></html>
```

4.3 Source

```

import java.awt.*;
import java.awt.event.*;           // catch events, wontcha
import java.applet.*;

// make it listen to keyboard doesnt work.
// make it listen to action doesnt work, too!!!
// so, make it listen to mouse:
public class TestD extends Applet implements Runnable, MouseListener
{
    int x, y;
    int dir = 1;                   // direction
    Thread th = new Thread (this); // make it short

    public void start () {
        addMouseListener (this);
        th.start ();             // queerly, performs run()...
    }

    public void run () {
        System.out.println ("run...");
        setForeground (Color.green);
        setBackground (Color.red);
        int i = 0;                // counter (what else...)
        while (true) {
            i += dir;
            x = (int) (Math.sin (0.05 * i) * 200 + 350);
            y = (int) (Math.cos (0.05 * i) * 200 + 250);
            repaint ();
            try { th.sleep (10); } catch (Exception e) { }
        }
    }

    public void paint (Graphics g) {
        g.drawString ("Click to change Direction", 20, 20);
        g.fillOval (x, y, 20, 20);
    }

    public void mouseClicked (MouseEvent e) {
        System.out.println ("Something happened.");
        dir = -dir;
    }

    public void mouseEntered (MouseEvent e) { }

    public void mouseExited (MouseEvent e) { }

    public void mouseReleased (MouseEvent e) { }

    public void mousePressed (MouseEvent e) { }
}

```

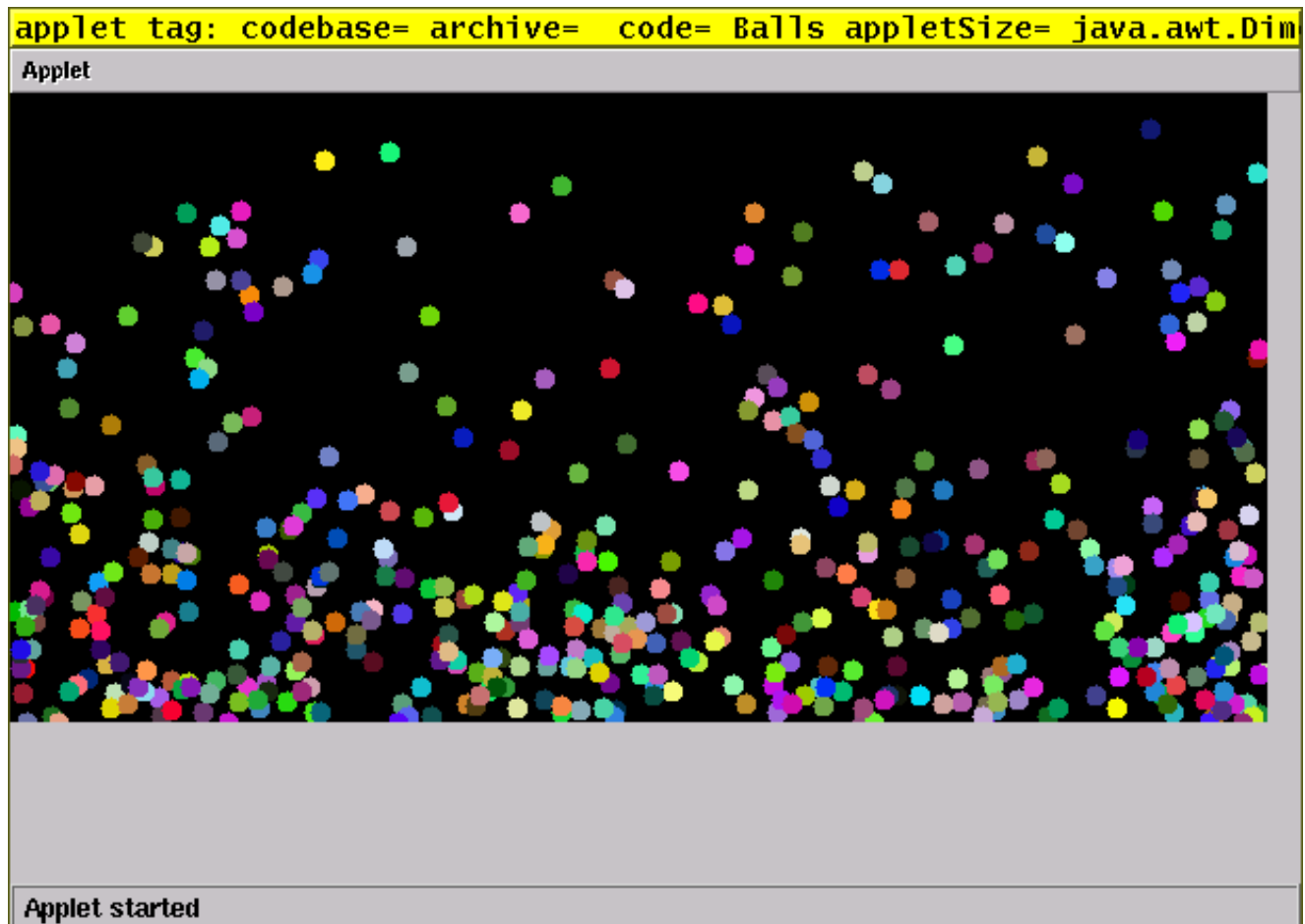
5 Sound - 2 B done

6 Self-Drawing Objects

IMHO, objects should not only know their color and how they move, but also how to draw themselves. We also need a God object to create, destroy and align our objects.

We will create enough Balls of Class Ball, jumping around without colliding.

6.1 Screenshot



6.2 HTML

```
<applet width=600 height=300 code=BallGod.class></applet>
```

6.3 Ball.java

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

// this is the class for ONE ball
public class Ball
{
    double x, y, vx, vy;
    int r = 5;
    Color c;
    double damp; // elasticity of ball
    int X, Y; // screen size, given by Balls
}
```

```

public Color randomColor () {
    return new Color ((int) (Math.random () * 255),
                     (int) (Math.random () * 255),
                     (int) (Math.random () * 255));
}

public Ball (int X, int Y) {
    this.X = X;
    this.Y = Y;
    x = y = 0;
    vx = Math.random ();
    vy = Math.random ();
    c = randomColor ();
    damp = 1 - Math.random () / 10;           // near 1
}

public void drawYourself (Graphics g) {
    g.fillOval ((int) (x - r), (int) (y - r), 2 * r, 2 * r);
}

public void moveYourself () {
    vy = vy + 0.01;
    x = x + vx;
    y = y + vy;
    if (x > X) {
        x = X;                               // paranioa setting
        vx = -vx;
    }
    if (x < 0) {
        x = 0;
        vx = -vx;
    }
    if (y > Y) {
        y = Y;
        vy = -vy * damp;
    }
    if (y < 0) {
        y = 0;
        vy = -vy;
    }
}
}

```

6.4 BallGod.java

```

// this is the class for ALL BALLS, a god-like object
// you may put as much class definitions as you like in one file.
public class BallGod extends Applet implements Runnable {
    int N = 500;                               // number of balls
    int X = 600;                               // screen size
    int Y = 300;
    Ball[] b = new Ball[N];
    Thread th = new Thread (this);
}

```

```
public void init () {
    setForeground (Color.yellow);
    setBackground (Color.black);

    for (int i = 0; i < N; i++)
        b[i] = new Ball (X, Y);
    th.start (); // queerly, performs run()...
}

public void run () {
    while (true) {
        for (int i = 0; i < N; i++)
            b[i].moveYourself ();
        repaint ();
        slep (5); // sleeping is essential, or ani will hassle
    }
}

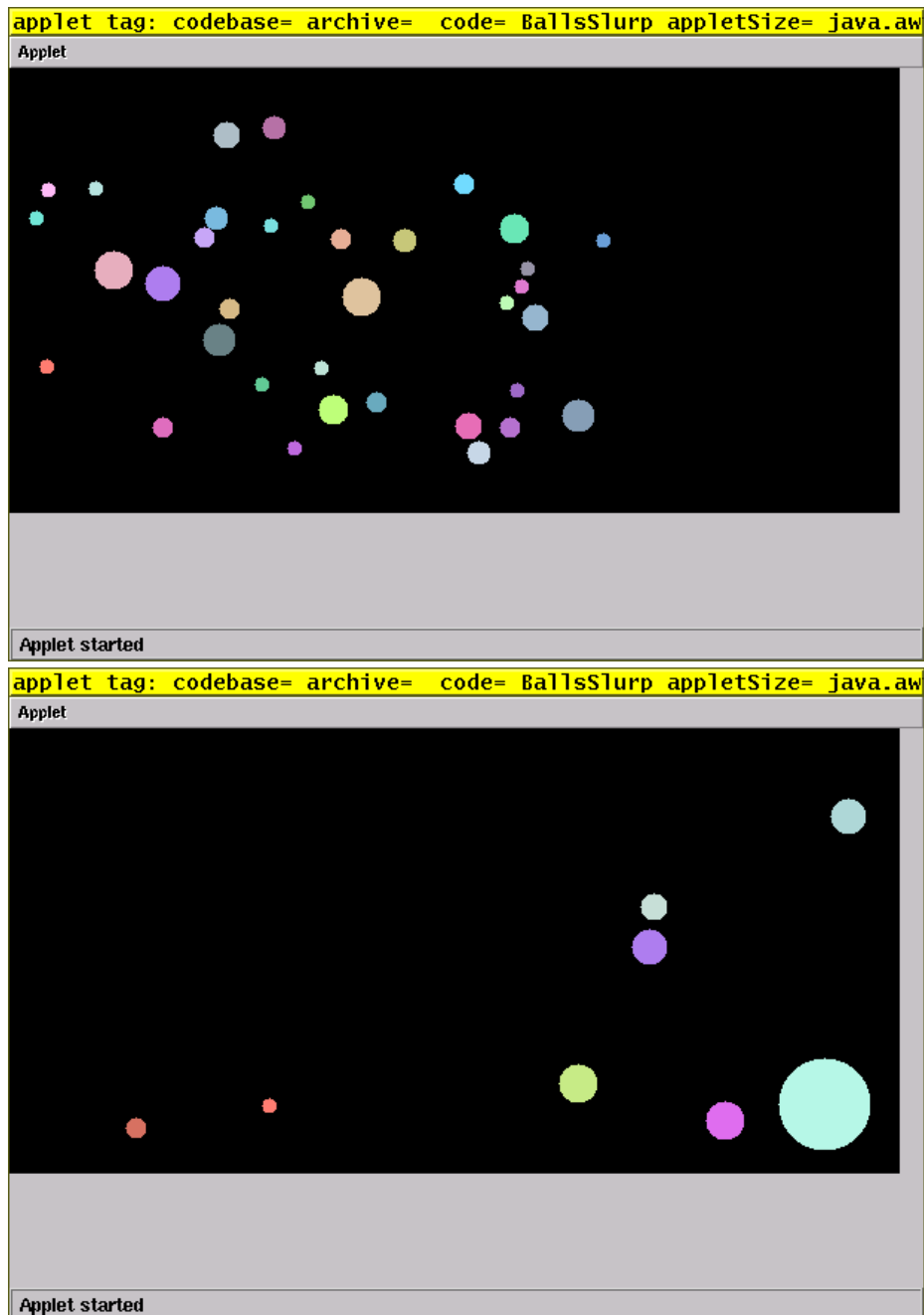
public void paint (Graphics g) {
    for (int i = 0; i < N; i++) {
        g.setColor (b[i].c);
        b[i].drawYourself (g);
    }
}

public void slep (int l) {
    try { th.sleep (l); } catch (Exception e) { }
}
}
```

7 Interaction: Slurping

Collision detection and Slurp commencing is done by god. Slurp means : two balls become one.

7.1 Screenshot



7.2 HTML

```
<applet width=700 height=300 code=BallSlurpGod></applet>
```

I omit HTML from now on.

7.3 BallSlurp.java

```
import java.awt.*;
import java.applet.*;
```

```
public class BallSlurp {

    double x, y, vx, vy;
    int r = 5;
    Color c;
    int X, Y;                // screen size, given by BallsSlurp
    boolean isDrawn = true;

    public Color randomColor () {
        return new Color ((int) (Math.random () * 155 + 100),
                           (int) (Math.random () * 155 + 100),
                           (int) (Math.random () * 155 + 100));
    }

    public BallSlurp (int X, int Y) {
        this.X = X;
        this.Y = Y;
        x = Math.random () * X / 2;
        y = Math.random () * Y / 2;
        vx = Math.random () * 2;
        vy = Math.random () * 2;
        c = randomColor ();
    }

    public void drawYourself (Graphics g) {
        if (isDrawn)
            g.fillOval ((int) (x - r), (int) (y - r), 2 * r, 2 * r);
    }

    public void moveYourself () {
        vy = vy + 0.01;
        x = x + vx;
        y = y + vy;
        if (x > X - r)
        {
            x = X - r;
            vx = -vx;
        }
        if (x < 0 + r)
        {
            x = r;
            vx = -vx;
        }
        if (y > Y - r)
        {
            y = Y - r;
            vy = -vy;
        }
        if (y < 0 + r)
        {
            y = r;
            vy = -vy;
        }
    }
}
```

```

}
}

```

7.4 BallSlurpGod.java

```

public class BallSlurpGod extends Applet implements Runnable {
    int N = 100;                // number of balls
    int X = 600;                // screen size
    int Y = 300;

    BallSlurp[] b = new BallSlurp[N];
    Thread th = new Thread (this);    // make it short

    public void init () {
        setForeground (Color.yellow);
        setBackground (Color.black);

        for (int i = 0; i < N; i++)
            b[i] = new BallSlurp (X, Y);
        th.start ();                // queerly, performs run()...
    }

    public void run () {
        while (true)
        {
            for (int i = 0; i < N; i++)
                b[i].moveYourself ();

            // Collision detection
            // if two balls meet, they become one. That is, one of them
            // becomes them two, with speed, the other will disappear

            for (int i = 0; i < N; i++)
                for (int j = 0; j < N; j++)
                    if (i != j && b[i].isDrawn && b[j].isDrawn)
                    {
                        double xd = b[i].x - b[j].x;
                        double yd = b[i].y - b[j].y;
                        double dist = Math.sqrt (xd * xd + yd * yd);
                        if (dist < b[i].r + b[j].r)
                        {
                            b[i].r = (int) (Math.sqrt (b[i].r * b[i].r +
                                                            b[j].r * b[j].r));

                            b[i].vx =
                                (b[i].vx * b[i].r + b[j].vx * b[j].r) / (b[i].r +
                                                                              b[j].r);

                            b[i].vy =
                                (b[i].vy * b[i].r + b[j].vy * b[j].r) / (b[i].r +
                                                                              b[j].r);

                            BallSlurp bb = new BallSlurp (X, Y);
                            b[i].c = bb.randomColor ();
                            b[j].isDrawn = false;
                        }
                    }
            repaint ();
        }
    }
}

```

```
        sleep (10);                // sleeping is essential, or ani will hassle
    }
}

public void paint (Graphics g) {
    for (int i = 0; i < N; i++) {
        g.setColor (b[i].c);
        b[i].drawYourself (g);
    }
}

public void sleep (int l) {
    try {
        th.sleep (l);
    } catch (Exception e) {
    }
}
}
```

8 Interaction: Bumping

Collision detection and Bumping is done by god.
Some anti-coagulation tricks needed.

8.1 Screenshot



8.2 BallBump.java

```
import java.awt.*;
import java.applet.*;

public class BallBump {

    double x, y, vx, vy;
    int r = 9;
    Color c;
    int X, Y; // screen size, given by BallsBump
    boolean isDrawn = true;

    public Color randomColor () {
        return new Color ((int) (Math.random () * 155 + 100),
                          (int) (Math.random () * 155 + 100),
                          (int) (Math.random () * 155 + 100));
    }

    public BallBump (int X, int Y) {
        this.X = X;
        this.Y = Y;
        x = Math.random () * X / 2;
        y = Math.random () * Y / 2;
        vx = Math.random () * 2;
        vy = Math.random () * 2;
        c = randomColor ();
    }
}
```

```

public void drawYourself (Graphics g)    {
    if (isDrawn)
        g.fillOval ((int) (x - r), (int) (y - r), 2 * r, 2 * r);
}

public void moveYourself ()    {
    vy = vy + 0.01;
    x = x + vx;
    y = y + vy;
    if (x > X - r)
        {
            x = X - r;
            vx = -vx;
        }
    if (x < 0 + r)
        {
            x = r;
            vx = -vx;
        }
    if (y > Y - r)
        {
            y = Y - r;
            vy = -vy;
        }
    if (y < 0 + r)
        {
            y = r;
            vy = -vy;
        }
}
}

```

8.3 BallBumpGod.java

```

public class BallBumpGod extends Applet implements Runnable {
    int N = 100;                // number of balls
    int X = 600;                // screen size
    int Y = 300;
    boolean ac = true;          // anti-coagulation
    BallBump[] b = new BallBump[N];
    Thread th = new Thread (this);

    public void init ()    {
        setForeground (Color.yellow);
        setBackground (Color.black);

        for (int i = 0; i < N; i++)
            b[i] = new BallBump (X, Y);
        th.start ();
    }

    public void run () {
        while (true) {
            // never-ending loop

```

```

        for (int i = 0; i < N; i++)
            b[i].moveYourself ();

// Collision detection
// if two balls meet, they swap speeds.

        for (int i = 0; i < N; i++)
            for (int j = 0; j < i; j++)
                if (i != j && b[i].isDrawn && b[j].isDrawn)
                    {
                        double xd = b[i].x - b[j].x;
                        double yd = b[i].y - b[j].y;
                        double dist = Math.sqrt (xd * xd + yd * yd);
                        double rges = b[i].r + b[j].r;
                        if (dist < rges)
                            {
                                double vx = b[i].vx;
                                b[i].vx = b[j].vx;
                                b[j].vx = vx;
                                double vy = b[i].vy;
                                b[i].vy = b[j].vy;
                                b[j].vy = vy;
                                if (ac)
                                    {
                                        b[i].moveYourself ();
                                        b[j].moveYourself ();
                                    }
                            }
                    }
            repaint ();
            slep (5);
        }

public void paint (Graphics g) {
    for (int i = 0; i < N; i++) {
        g.setColor (b[i].c);
        b[i].drawYourself (g);
    }
}

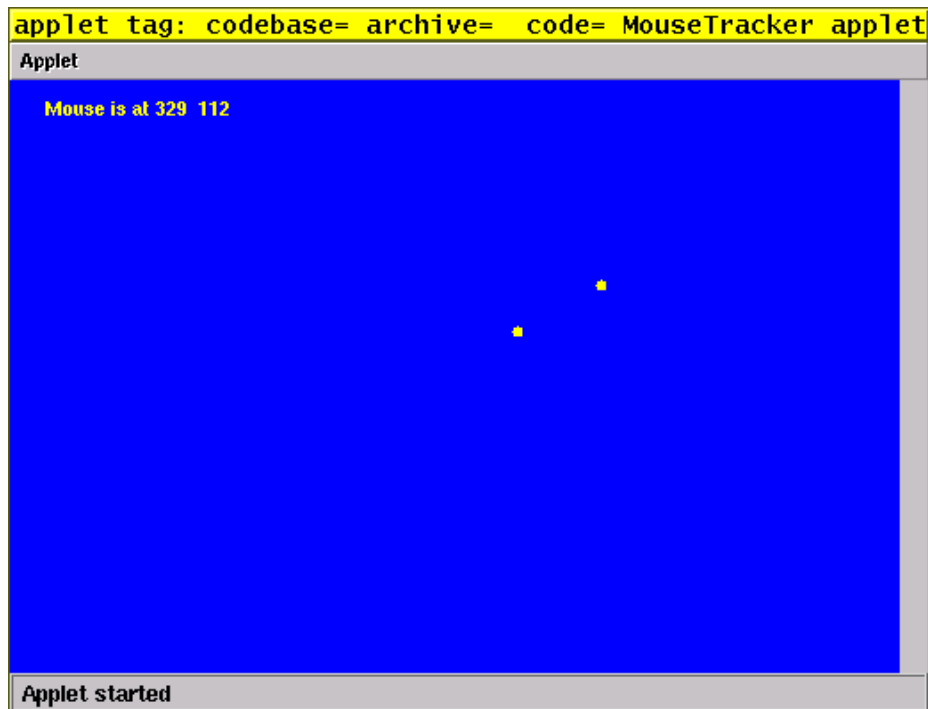
public void slep (int l) {
    try { th.sleep (l); } catch (Exception e) { }
}
}

```

9 MouseTracking

Mouse Tracking is done via MouseMotionListener and its 2 methods.

9.1 Screenshot



9.2 MouseTracker.java

```
import java.awt.*;
import java.awt.event.*;           // catch Mouse Events
import java.applet.*;

public class MouseTracker extends Applet
    implements Runnable, MouseMotionListener {

    double x, y;
    int r;
    int mx, my;                     // mouse position

    Thread th = new Thread (this);

    public void init () {
        x = 0;
        y = 0;
        r = 3;
        mx = 9;
        my = 9;
        addMouseMotionListener (this);    // add listener, or it wont listen
        this.setForeground (Color.yellow);
        this.setBackground (Color.blue);
        th.start ();
    }

    public void run () {
```

```

while (true) {
    x = x + (mx - x) / 19;           // x and y run to mx and my
    y = y + (my - y) / 19;
    repaint ();
    sleep (50);
}

// its totally bad idea to put sleep() into paint() - dunno why
public void paint (Graphics g) {
    g.fillOval ((int) (x - r), (int) (y - r), 2 * r, 2 * r);
    g.fillOval (mx, my, 2 * r, 2 * r);
    g.drawString ("Mouse is at " + mx + " " + my, 20, 20);
}

public void sleep (int l) {
    try { th.sleep (l); } catch (Exception e) { }
}

/* on every mouse move, this is called automagically. */
public void mouseMoved (MouseEvent me) {
    mx = me.getX ();
    my = me.getY ();
}

/* on every mouse drag, this is called automagically. */
public void mouseDragged (MouseEvent me) {
}

}

```

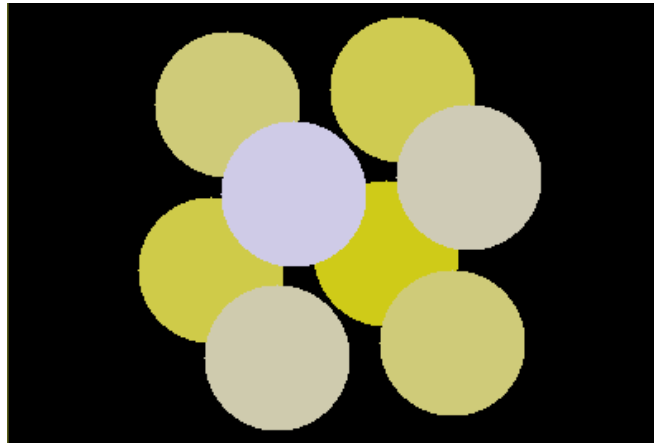
Yo see, its easy to grow this to a 2-dim-screen game: track the successor, make him collect the goodies (of class Goodie, green) and avoid the baddies. Add/subtract score somewhere in the corner.

On the other hand, you might program a 3-dim-figure, and rotate and display its points while taking xpos and ypos as angle degrees. Make the points greyer the deeper they are...

Queer idea...

10 MouseCube

10.1 Screenshot



10.2 MouseCube.java

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class MouseCube extends Applet
    implements Runnable, MouseMotionListener {
    int N = 8;                // number of balls
    int X = 400;              // screen size
    int Y = 400;
    int x2 = X / 2;          // half screen size
    int y2 = Y / 2;
    int r = Y / 9;           // part of screen size
    double wx = 0;           // its a winkel (angle)
    double wy = 0;
    double d = 1.3;          // its nicer
    int mx, my;

    // the Points should REALLY know how to care for themselves
    // actually, they should compose Lines who care for themselves
    double[] x = new double[N];    // x to right
    double[] y = new double[N];    // y to down
    double[] z = new double[N];    // z into depth
    Thread th = new Thread (this);

    public void init () {
        setForeground (Color.yellow);
        setBackground (Color.black);
        addMouseMotionListener (this);
        int c = 0;                // counter
        for (int xx = -1; xx < 2; xx++)
            for (int yy = -1; yy < 2; yy++)
                for (int zz = -1; zz < 2; zz++)
                    if (xx * yy * zz != 0) {
                        x[c] = xx * r * d;
                        y[c] = yy * r * d;
```

```

        z[c] = zz * r * d;
        c++;
    }
    th.start ();
}

public void run () {
    while (true) {
        for (int i = 0; i < N; i++) {
// wx is for rotation around y
            double xx = x[i] * Math.cos (wx) + z[i] * Math.sin (wx);
            z[i] = z[i] * Math.cos (wx) - x[i] * Math.sin (wx);
            x[i] = xx;

// wy is for rotation around x
            double yy = y[i] * Math.cos (wy) + z[i] * Math.sin (wy);
            z[i] = z[i] * Math.cos (wy) - y[i] * Math.sin (wy);
            y[i] = yy;
        }
// oops - must sort by z - so lets bubblesort...
        double h;
        for (int i = 1; i < N; i++)
            for (int j = 0; j < i; j++) {
                if (z[j] < z[i]) {
                    h = x[i]; x[i] = x[j]; x[j] = h;
                    h = y[i]; y[i] = y[j]; y[j] = h;
                    h = z[i]; z[i] = z[j]; z[j] = h;
                }
            }
        repaint ();
        slep (200); // sleeping is essential
    }
}

public void paint (Graphics g) {
    for (int i = 0; i < N; i++) {
// weird
        g.setColor (new Color (200, 200, 127 - (int) (z[i] / 2 / r / d * 120)));
        g.fillOval (x2 + (int) x[i] - r, y2 + (int) y[i] - r, 2 * r, 2 * r);
    }
}

public void slep (int l) {
    try { th.sleep (l); } catch (Exception e) { }
}

public void mouseMoved (MouseEvent me) {
    mx = me.getX ();
    my = me.getY ();
// zero rotation at (30,30)
    wx = (mx - 30) / 1000.0; // make a rotation angle from
    wy = (my - 30) / 1000.0; // mouse position
}

```

```
public void mouseDragged (MouseEvent me) {  
    }  
  
}
```

11 MouseCube2

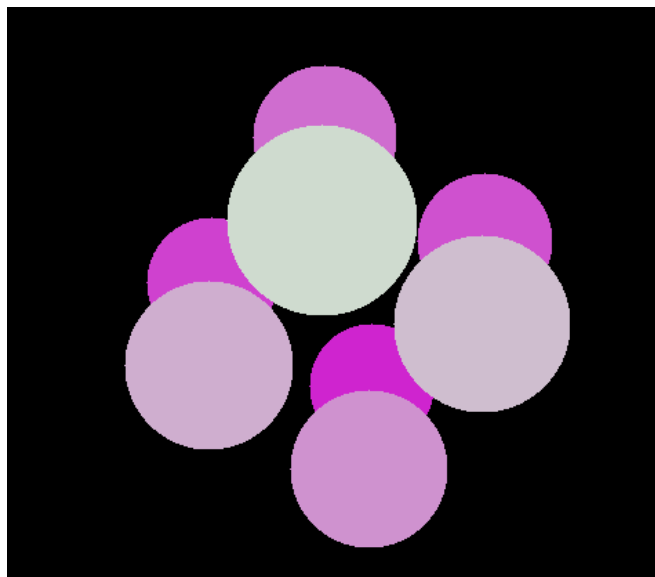
This section contains few new:

- I put Point creation into method makePoints().
- The balls are smaller the deeper they are.
- The infamous flicker of viewing applets is erased:

There is an Image (with, of course, a graphics area) where paint() paints on, then the Image is copied to the screen graphics.

This is done by overwriting update(). Update would normally erase the screen, and then call paint(). Now it does: clear a framebuffer – call paint() on that framebuffer – copy framebuffer to screen.

11.1 Screenshot



11.2 MouseCube2.java

I give only the changes here.

```
// added perspective and Framebuffer

// inserted before init() (actually, between each method you choose)
public void makePoints(){
    int c = 0; // counter
    for (int xx = -1; xx < 2; xx++)
        for (int yy = -1; yy < 2; yy++)
            for (int zz = -1; zz < 2; zz++)
                if (xx * yy * zz != 0)
                {
                    x[c] = xx * r * d;
                    y[c] = yy * r * d;
                    z[c] = zz * r * d;
                    c++;
                }
}
// end inserted
```

```

// call makePoints() where - sic - points are made
  fbg = fb.getGraphics ();
  makePoints();          // makePoints
  th.start ();
}
// end call makePoints()

// added a variable boolean sort=false near start,
// to switch sorting off and on
  if (sort)
// oops - must sort by z
// end added a variable

//////////////////////////////////// overwrite update() //////////////////////////////////
// added, that is overwrote method update()
// we will NOT erase and repaint the screen
// but we will erase and paint on a framebuffer, and then copy it
// to graphics
// i could put these calls into update(), too, but theyre really
// expensive, so my athlon 400 gets hooked.
// 2 needed global variables

Image fb;          // framebuffer
Graphics fbg;     // framebuffer graphics

// 2 calls in init()
fb = this.createImage (X,Y);
fbg = fb.getGraphics();

// finally, update() itself
public void update (Graphics g) {
// erase fb
  fbg.setColor (getBackground ());
  fbg.fillRect (0, 0, X, Y);
// call paint with Image's graphics
  paint (fbg);
// copy Image to screen graphics
  g.drawImage (fb, 0, 0, this);
}
//////////////////////////////////// end overwrite update() //////////////////////////////////

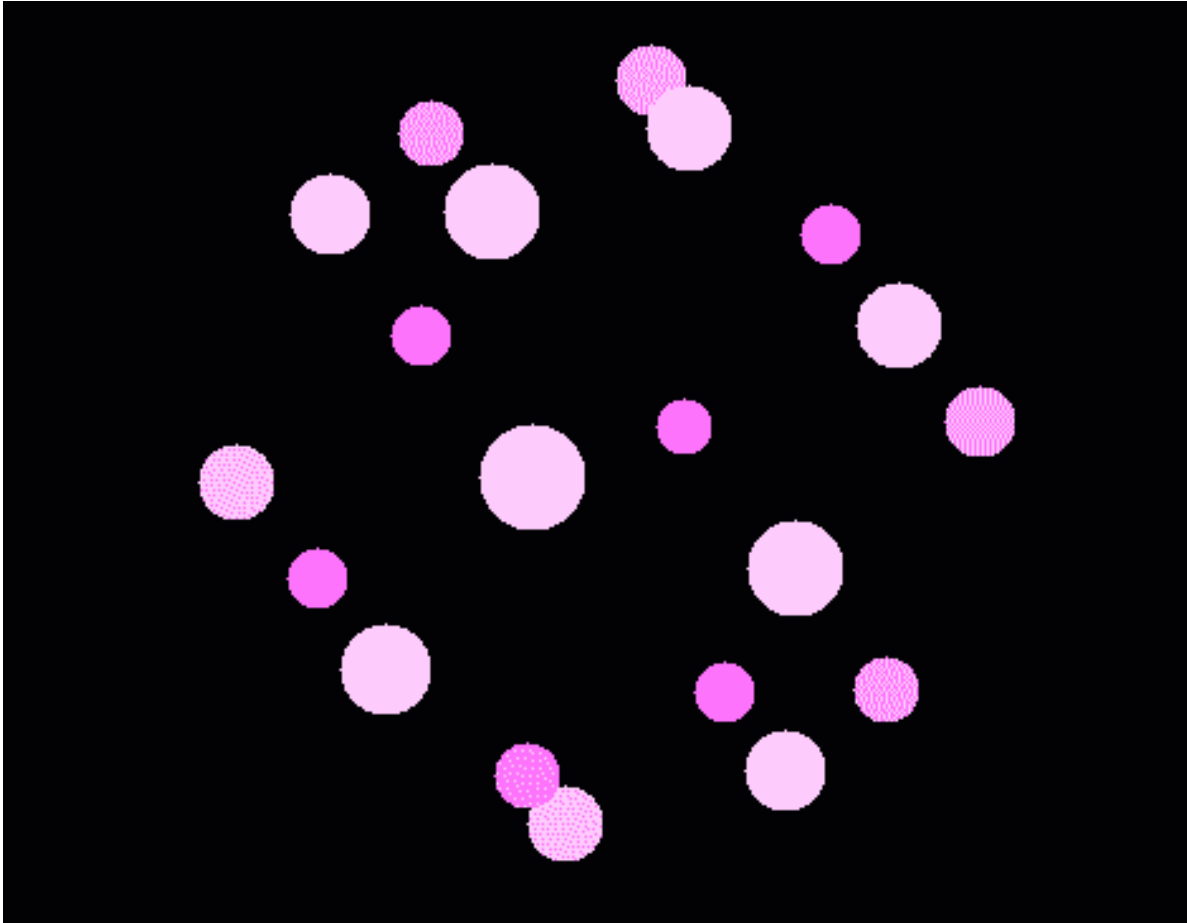
/// in paint() : added balls become bigger or smaller, and a new color
/* perspective:
  if z<=w draw nothing
  if z in between w and 0 draw double size
  if z=0 draw normal size
*/
if (z[i] > w * 9) {
  int r2 = (int) (r * (-w) / (z[i] - w));
  g.fillOval (x2 + (int) x[i] - r2, y2 + (int) y[i] - r2, 2 * r2, 2 * r2);
}
}

```


12 Dodecaeder

This section contains nearly nothing new but Dodecaeder points. A Dodecaeder is a regular 3-d-shape, constructed of 12 pentagons.

12.1 Screenshot



12.2 Dodecaeder.java - Changes from MouseCube2.java

```

/// Only method makepoints() has changed.

public void makePoints() {
    int c = 0; // counter
    // create a dodecaeder semi-automatic
    double xx, yy, zz, rr;
    xx = 0.85;
    rr = Math.sqrt (1 - xx * xx);
    for (int i = 0; i < 5; i++)
    {
        double w = Math.PI * 2 / 5 * i;
        yy = rr * Math.cos (w);
        zz = rr * Math.sin (w);
        x[c] = xx * r * d;
        y[c] = yy * r * d;
        z[c] = zz * r * d;
        c++;
    }
    xx = 0.23;

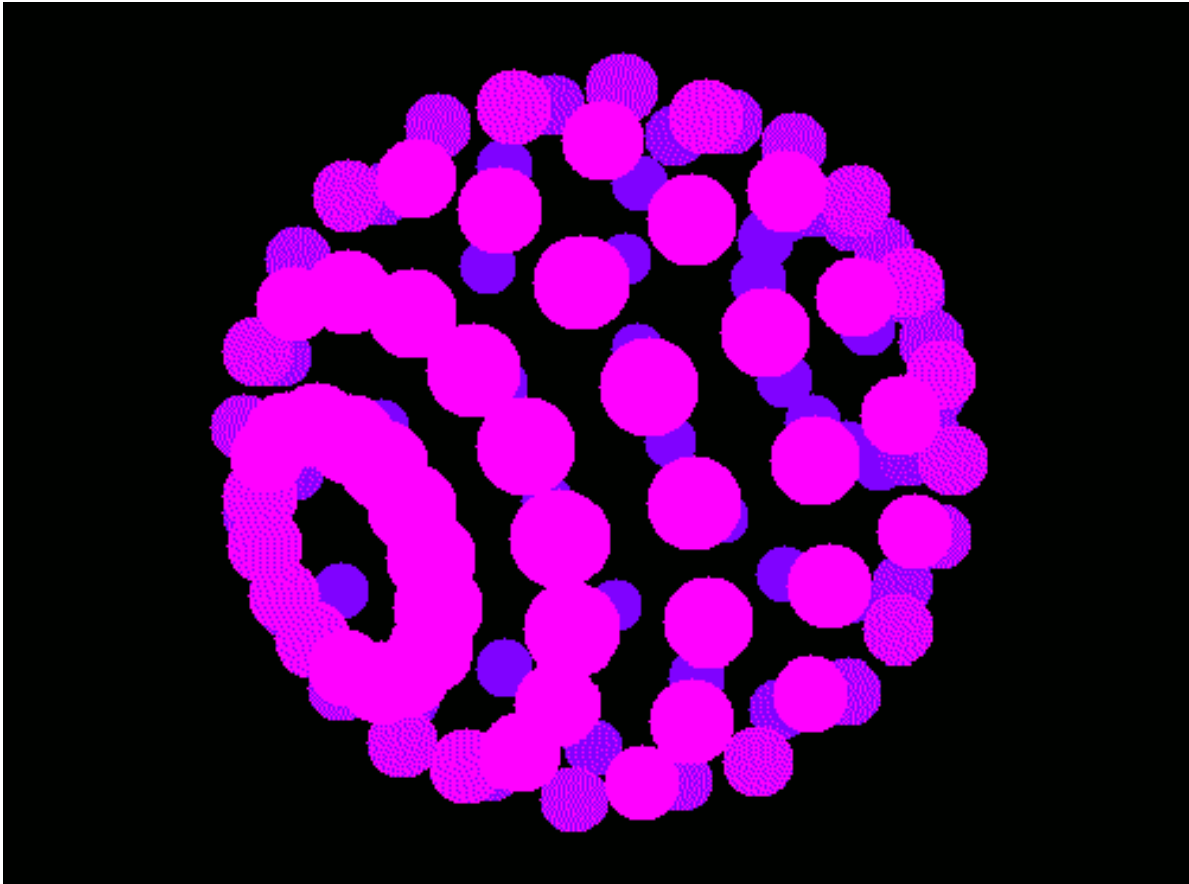
```

```
    rr = Math.sqrt (1 - xx * xx);
for (int i = 0; i < 5; i++)
{
    double w = Math.PI * 2 / 5 * i;
    yy = rr * Math.cos (w);
    zz = rr * Math.sin (w);
    x[c] = xx * r * d;
    y[c] = yy * r * d;
    z[c] = zz * r * d;
    c++;
}
xx = -0.23;
rr = Math.sqrt (1 - xx * xx);
for (int i = 0; i < 5; i++)
{
    double w = Math.PI * 2 / 5 * i;
    yy = -rr * Math.cos (w);
    zz = rr * Math.sin (w);
    x[c] = xx * r * d;
    y[c] = yy * r * d;
    z[c] = zz * r * d;
    c++;
}
xx = -0.85;
rr = Math.sqrt (1 - xx * xx);
for (int i = 0; i < 5; i++)
{
    double w = Math.PI * 2 / 5 * i;
    yy = -rr * Math.cos (w);
    zz = rr * Math.sin (w);
    x[c] = xx * r * d;
    y[c] = yy * r * d;
    z[c] = zz * r * d;
    c++;
}
System.out.println (c + " points.");
N=c;
}
```

13 Sphere

I couldnt resist writing a sphere...

13.1 Screenshot



13.2 makePoints() - renewed

```
public void makeBallPoints () {
    int c = 0;
    int M = 8;
    for (int i = 0; i < M; i++)    {
        double u = Math.PI / M * i;
        for (int j = 0; j < 2 * M; j++)    {
            double v = Math.PI / M * j;
            x[c] = r * d * Math.cos (u);
            y[c] = r * d * Math.sin (u) * Math.cos (v);
            z[c] = r * d * Math.sin (u) * Math.sin (v);
            c++;
        }
    }
    N = c;
    System.out.println (c + " points.");
}
```

14 A Deeper Look into Threads

A thread is some line of fabric. A Thread in Java is something that is run() by saying start() (???) and then runs parallel to the main Thread (that, so far, we were not aware of, and usually never will be). Consider the following 2 Applets. They seem to do the same thing. But one of them wont work.

14.1 TestG.java

```
// doesnt work
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class TestG extends Applet {           // implements Runnable

    public void start () {
        while (true) {
            repaint ();
            try { Thread.sleep (100); } catch (Exception e) { }
        }
    }

    public void paint (Graphics g) {
        int z = (int) (Math.random () * 200);
        g.fillOval (z, z, z, z);
    }
}
```

14.2 TestH.java

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class TestH extends Applet implements Runnable {

    public void start () {
        //////////////// the following lines were added ////////////////
        Thread th = new Thread (this);
        th.start ();
    }

    public void run () {
        //////////////// the above lines were added ////////////////
        while (true) {
            repaint ();
            try { Thread.sleep (100); } catch (Exception e) { }
        }
    }

    public void paint (Graphics g) {
        int z = (int) (Math.random () * 100 + 100);
        g.fillOval (z, z, z, z);
    }
}
```

```
}
```

So, if you want to write a `SortAlgorithmVisualisation`, you must

- tell your Applet it is also a Thread (“implements Runnable”);
- write a single `run()` method;
- in this `run()`, distinguish which sort method you are currently using.

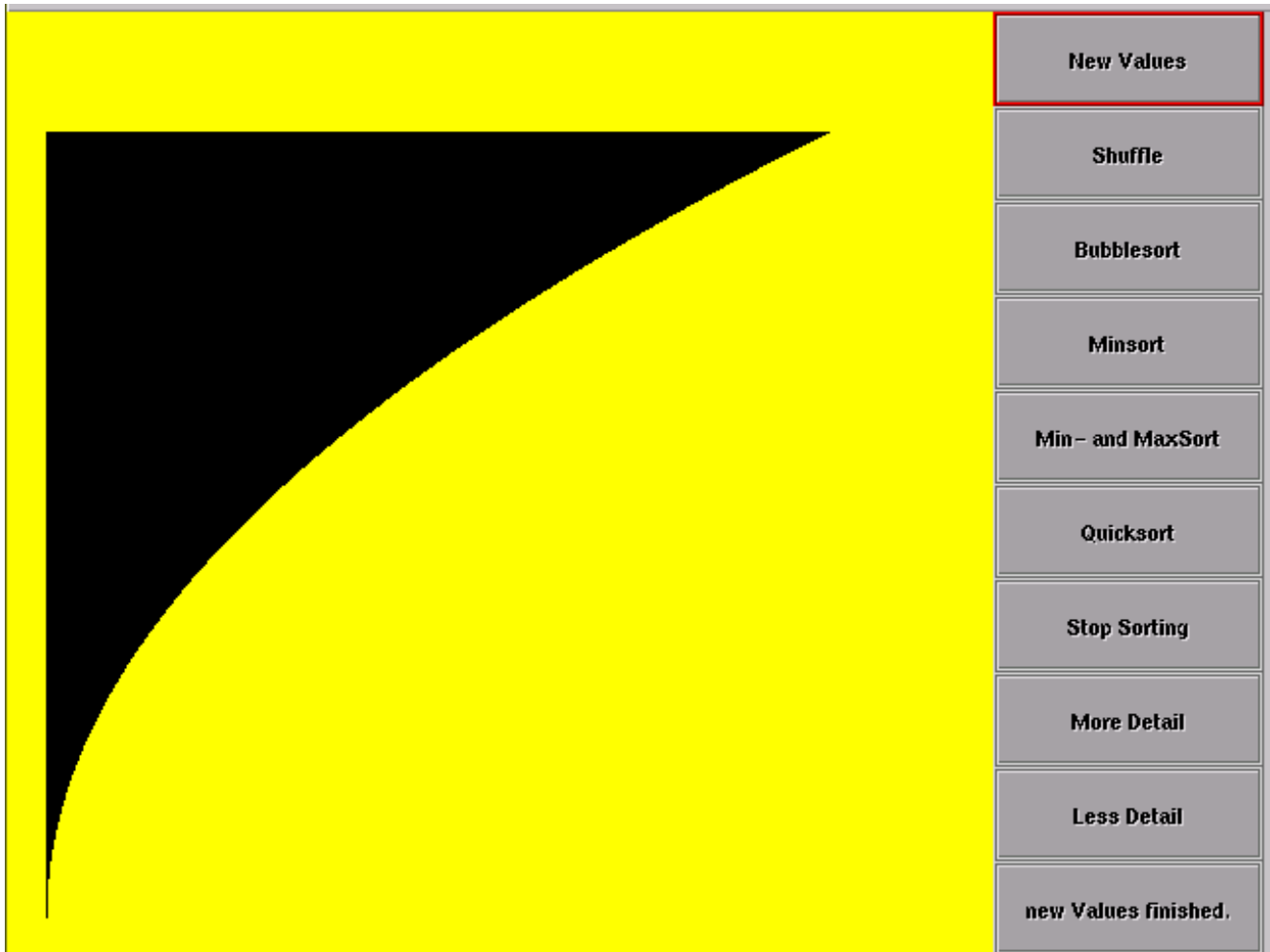
Well then...

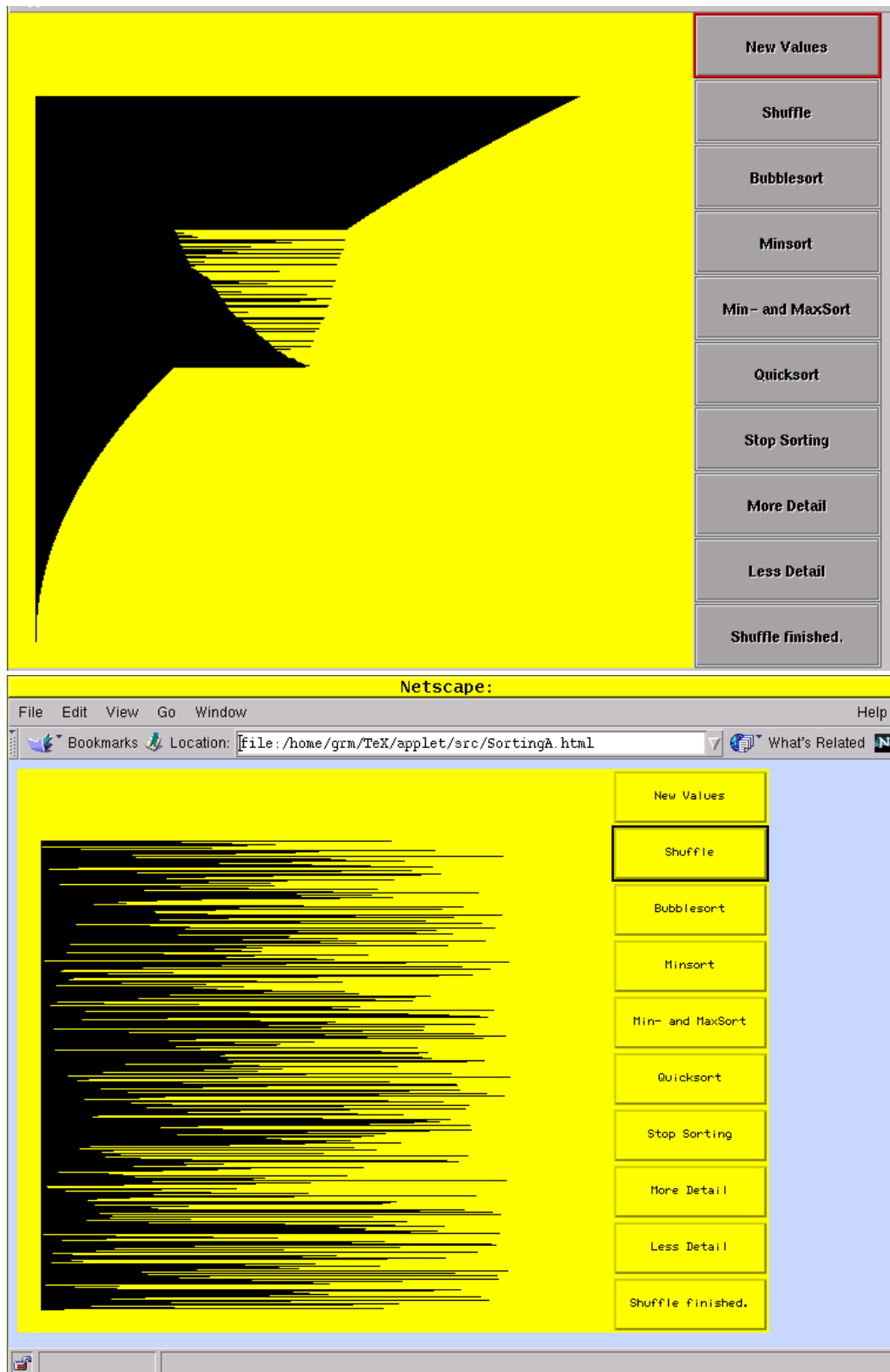
15 Web-Application : Visualisation of Sorting Algorithms

Okay this is a web application. On the right, you have your choices: NewValues - Bubblesort - Minsort - Min- and MaxSort - StopSorting - Shuffle - moreDetail - less Detail. The last Button is to display the time in ms (Milliseconds) or other status info.

I think, the Buttons' names are quite self-explaining.

15.1 Three Screenshots



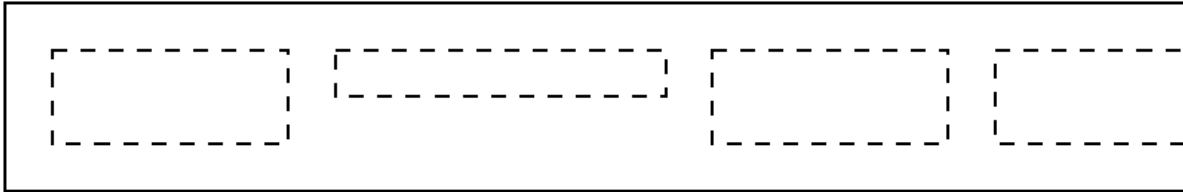


15.2 Some Sad Words about Layouts

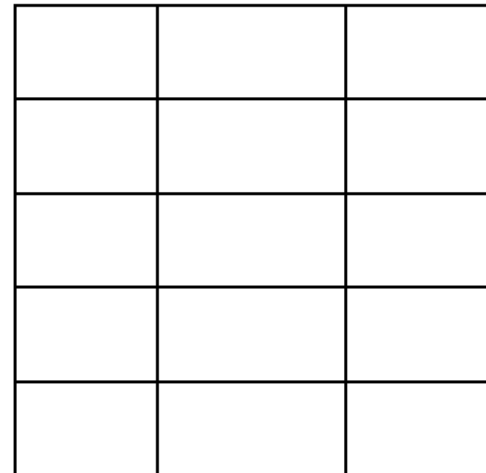
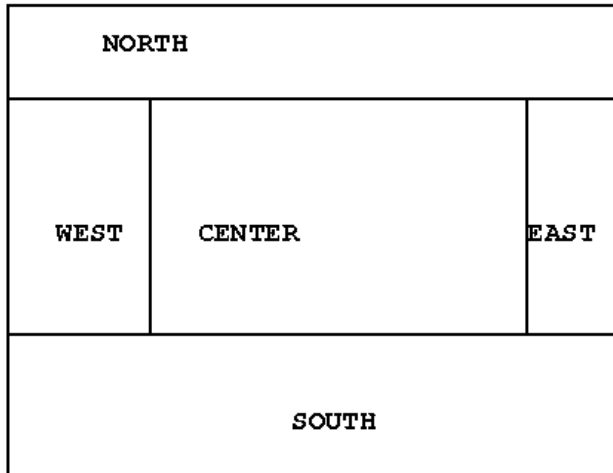
Java does not force absolute coordinates for anything, but instead wants the following:
Say what you want, and say how you want it relative to the other things.

That's why we have to learn some things:

An applet, and each other Container for Things, can have one of the following Layouts:



FlowLayout



BorderLayout

GridLayout (5, 3)

Theres GridBagLayout and CardLayout available, too, but they're hard to use.

FlowLayout lines the Items up horizontally - insufficient space leads to unseeable Buttons.

At GridLayouts, all Buttons have the same size. If you dont care about the number of rows / columns, put a 0 (Zero).

In BorderLayout, you must tell Java in which region your Item shall go.

For a 2-handed screen with a graphics on the left and stacked buttons on the right, i will use the CENTER and EAST regions of a BorderLayout.

To stack the buttons, I will put one Panel (whatever that is - in fact, its something where you can put things on) with GridLayout (1,0) in the EAST region of the applet's BorderLayout.

Scared? Think of it like stacking cards:

1. the applet itself;
2. the BorderLayout, covering the whole applet
3. the Panel, in EAST region;
4. the GridLayout (1,0) on the Panel;
5. the Buttons in the GridLayout.

15.3 Action Detection

We will catch Button's events with the all-powerful ActionListener. We will bind him to each and every button.

We must, then, write a method actionPerformed. It is automatically called whenever an ActionEvent (e.g.ButtonClick) is performed by the user. We then retrieve the Command (= the Button's Label) from the ActionEvent and act logically.

At least, that's the plan.

15.4 Threading is necessary...

... and I still dislike it...

... but I consider them as the only known, and reliable workaround. See previous chapter.

15.5 SortingA.java

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class SortingA extends Applet implements ActionListener, Runnable {

    String sort;                // to see in Thread which way to go
    int X = 640;
    int Y = 480;
    int N = 400;
    Thread th;
    int detail = 1;            // allowed: 0 no / 1 few / 2 full detail
    int parabol = -1;        // see newValues()
    Button bu;
    int sleepms = 1;        // sleepinterval in ms
    int b[] = new int[N];
    String[] buttons =
    {
        "New Values", "Shuffle", "Bubblesort", "Minsort", "Min- and MaxSort",
        "Quicksort", "Stop Sorting", "More Detail", "Less Detail", "0 msec"};

    public void init () {
        this.setLayout (new BorderLayout ());
        Panel p1 = new Panel ();
        this.add (p1, BorderLayout.EAST);
        p1.setLayout (new GridLayout (0, 1));        // (rows, columns)
        for (int i = 0; i < buttons.length; i++)    {
            bu = new Button (buttons[i]);
            p1.add (bu);
            bu.addActionListener (this);
        }
        setBackground (Color.yellow);
        newValues ();
    }

    public void run () {
        if (sort == "bubble")    bubbleSort ();
        if (sort == "min")        minSort ();
        if (sort == "minmax")    minMaxSort ();
        if (sort == "quick")    quickSort ();
        repaint ();
    }

    public void paint (Graphics g) {
        for (int i = 0; i < N; i++)    {
            g.drawLine (20, Y - 20 - i, 20 + b[i], Y - 20 - i);
        }
    }
    // doesnt work here
```

```

//  slep (1);
}

public void newValues () {
    parabol = -parabol;
    for (int i = 0; i < N; i++)    {
        if (parabol == 1)
            b[i] = i * i / N;
        else
            b[i] = i;
    }
    bu.setLabel ("new Values finished.");
}

public void shuffle () {
    for (int i = 0; i < 2 * N; i++)    {
        int j = (int) (Math.random () * N);
        int k = (int) (Math.random () * N);
        int l = b[j];
        b[j] = b[k];
        b[k] = l;
        if (detail == 2) rePaint (0);
        bu.setLabel ("Shuffle finished.");
    }
}

public void bubbleSort () {
    boolean sorted = false;
    while (!sorted)    {
        sorted = true;
        for (int j = 0; j < N - 1; j++)    {
            if (b[j] > b[j + 1])    {
                int h = b[j + 1];
                b[j + 1] = b[j];
                b[j] = h;
                sorted = false;
                if (detail == 2) rePaint ();
            }
        }
        if (detail == 1) rePaint ();
    }
    bu.setLabel ("BubbleSort finished.");
}

public void minMaxSort () {
    for (int j = 0; j < N / 2; j++)    {
// b[j] shall become miminum. b[N-1-j] shall become maximum.
        for (int i = j; i < N - j - 1; i++)    {
            if (b[j] > b[i])    {
                int h = b[i];
                b[i] = b[j];
                b[j] = h;
                if (detail == 2) rePaint ();
            }
        }
    }
}

```

```

        if (b[N - j - 1] < b[i]) {
            int h = b[i];
            b[i] = b[N - j - 1];
            b[N - j - 1] = h;
            if (detail == 2) repaint ();
        }
    }
    if (detail == 1) repaint ();
}
bu.setLabel ("minMaxSort finished.");
}

public void minSort () {
    for (int i = 0; i < N; i++) {
        for (int j = i + 1; j < N; j++)
            if (b[j] < b[i]) {
                int h = b[j];
                b[j] = b[i];
                b[i] = h;
                if (detail == 2) repaint ();
            }
        if (detail == 1) repaint ();
    }
    bu.setLabel ("minSort finished.");
}

public void quickSort () {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < i; j++)
            if (b[j] < b[i]) {
                int h = b[i];
                b[i] = b[j];
                b[j] = h;
                if (detail == 2) repaint ();
            }
    bu.setLabel ("quickSort isnt implemented yet.");
}

public void actionPerformed (ActionEvent e) {
// "New Values", "Shuffle", "Bubblesort", "Minsort", "Min- and MaxSort",
// "Quicksort", "Stop Sorting", "More Detail", "Less Detail", "0 msec";
    String cmd = e.getActionCommand ();
    System.out.println (cmd);
    if (cmd == buttons[0]) newValues ();
    if (cmd == buttons[1]) shuffle ();
    if (cmd == buttons[2]) {
        sort = "bubble";
        th = new Thread (this);
        th.start ();
    }
// starts this Applet's/Thread's run()
    if (cmd == buttons[3]) {
        sort = "min";
        th = new Thread (this);
        th.start ();
    }
}

```

```

    } // starts this Applet's/Thread's run()
if (cmd == buttons[4]) {
    sort = "minmax";
    th = new Thread (this);
    th.start ();
} // starts this Applet's/Thread's run()
if (cmd == buttons[5]) {
    sort = "quick";
    th = new Thread (this);
    th.start ();
} // starts this Applet's/Thread's run()
if (cmd == buttons[6]) th.stop (); // stop sorting Thread
if (cmd == buttons[7]) {
    if (detail < 2) detail++;
    bu.setLabel ("detail = " + detail);
}
if (cmd == buttons[8]) {
    if (detail > 0) detail--;
    bu.setLabel ("detail = " + detail);
}
repaint ();
}

// to have a repaint that sleeps a while
public void rePaint () {
    repaint ();
    try { Thread.sleep (sleepms); } catch (Exception e) { }
}

}

```

Thats all folks.