Grafika in animacija (java)

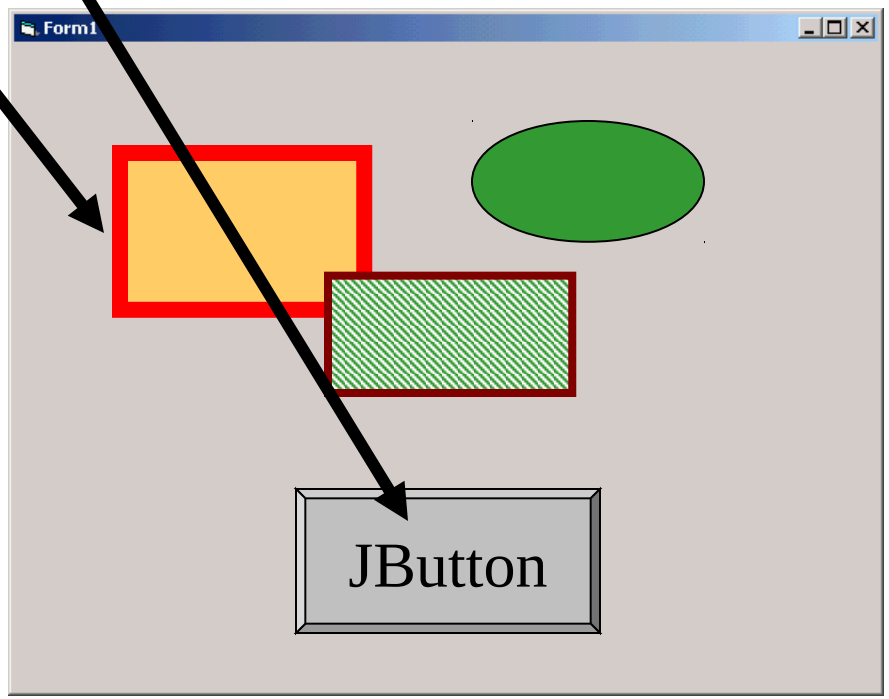
Uporabne podatkovne strukture v Javi

- Vector (dinamično polje)
  - V = new Vector( );
  - V.add(item);
  - V.elementAt(5);
- HashTable (preslika ključe na elemente (slike))
  - H = new HashTable( );
  - H.add(key, item);
  - H.get(key);
- Iteratorji (avtomatične zanke FOR)
  - I = V.iterator( );
  - While (I.hasNext( ))
  - dosomething(I.next( ));

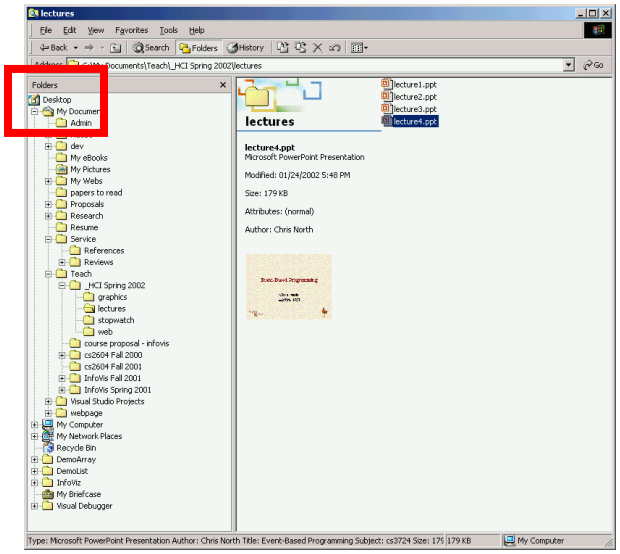
Grafika

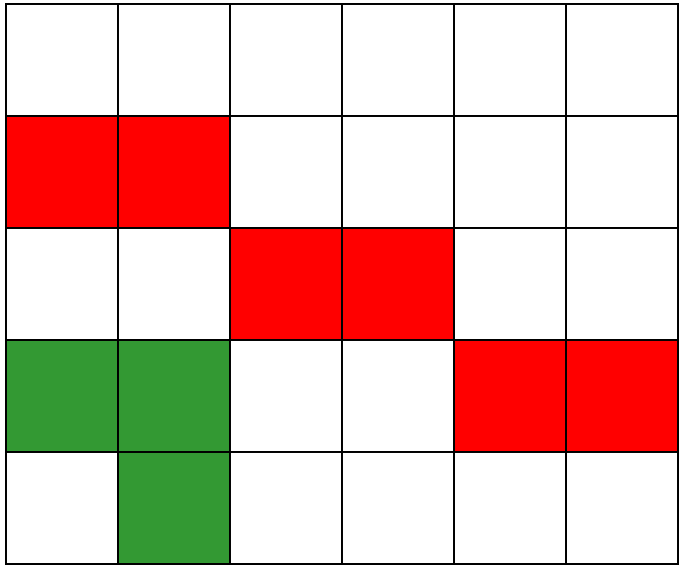
- Okno je kot slikarjevo platno
- Vsebino okna mora narisati aplikacija
  - Komponente se izrišejo same
  - Vse drugo nariše programer

- Kdaj naj rišemo?
- Kako naj rišemo?



Nekaj o pikslih





Običajno imamo rastersko grafiko

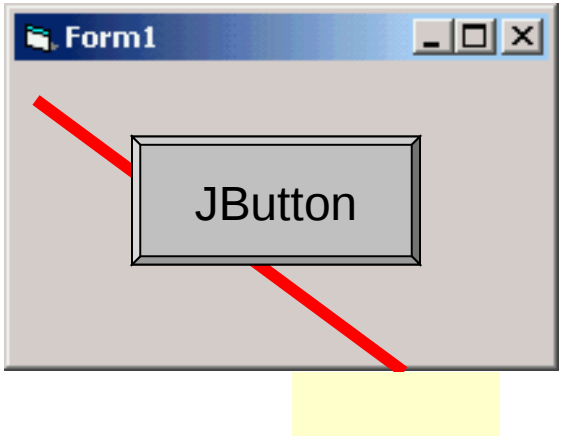
Koordinatni sistem

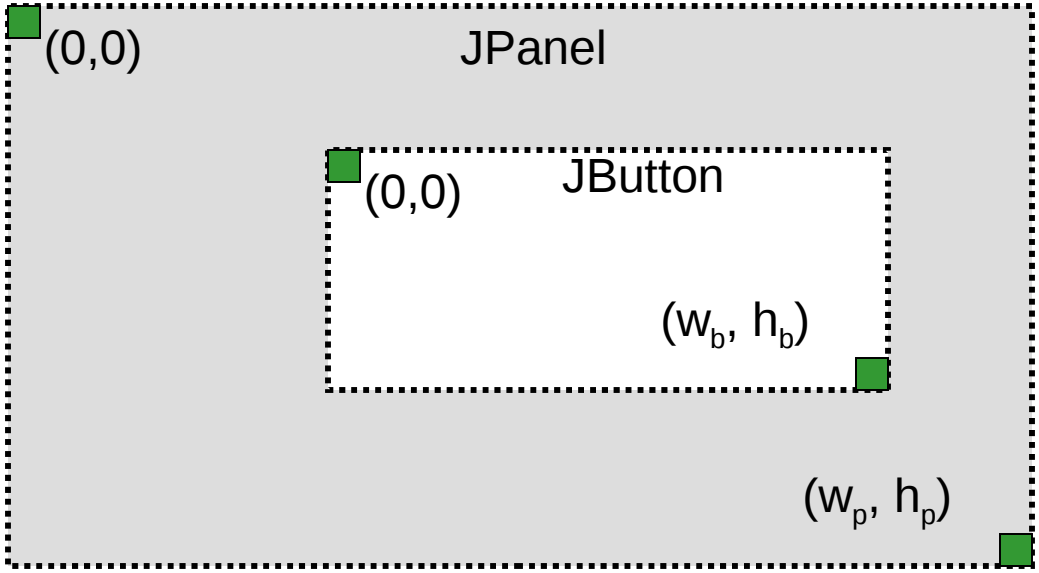
- Kartezični, od zgoraj navzdol

- $Y_{okna} = višina - Y_{kartezični}$

Hierarhija komponent

- Vsaka komponenta ima svoje "podokno"
  - podokno = pravokotnik znotraj starševske komponente
  - Ima lastni koordinatni sistem
- Obrezovanje (Clipping):
  - Ne moremo risati izven podokna
  - ne moremo risati preko vgrajenih komponent (otrok)?





Risanje v Javi

```
import java.awt.Graphics
import java.awt.Graphics2D       // Java2
```

1. *pridobimo "grafični konekst" komponente*

```
Graphics g = myJPanel.getGraphics( );
Graphics2D g2 = (Graphics2D) g;
```

2. *Rišemo vanj*

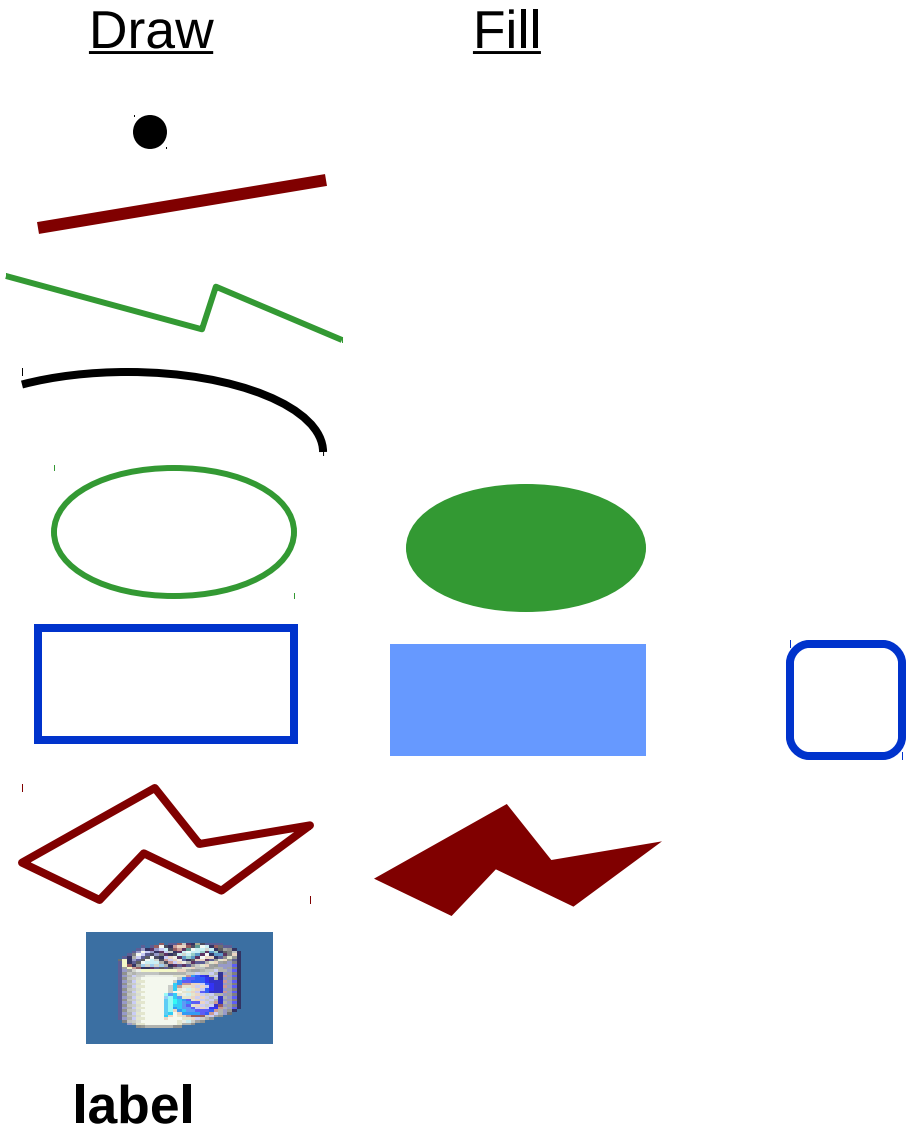
```
g2.drawLine(x1,y1, x2,y2);
```

Grafični gradniki

- Point (x,y)
- Line (pt1,pt2)
- PolyLine (pt list)
- Arc

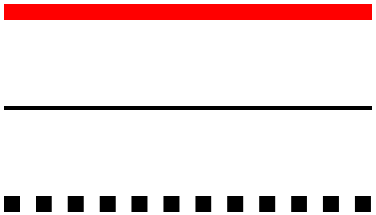
- Oval (pt, w,h)
- Rectangle (pt, w,h)
  - RoundRectangle

- Polygon (pt list)
- Image (file, x,y)
- Text (string, x,y)



Grafični atributi

- Color
- Font
- Atributi črt:
  - debelina, črtkanost, markerji končnih točk,
- Atributi risanja:
  - Color, gradient, texture
- Composite:
  - Mešanje (Blending)
- Transformacije:
  - Translacija, rotacija, "flip", striženje, skaliranje



To velja za Java 2D

Java2D demo

Barva (Color)

- Kombinacije rdeče, zelene in modre
- Vsaka v območju [0, 255]

- Tvorba nove barve v Javi: *new Color(255, 150, 0)*

Oranžna

V Javi velja

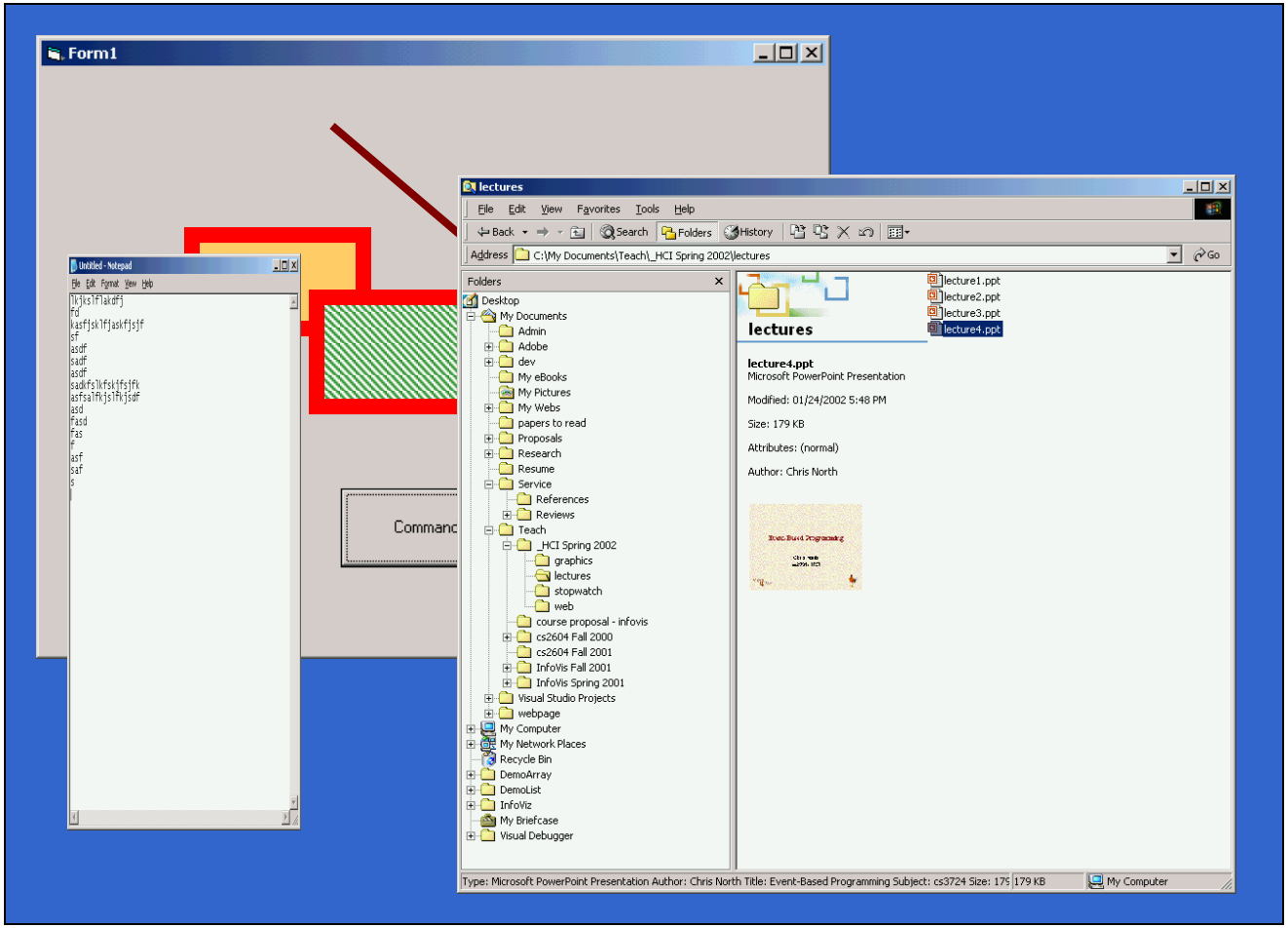
- Risanje gradnikov:
  - g2.drawLine( ), .drawRect( ), …
  - g2.fillRect( ), …
- Objektna usmerjenost:
  - Tvorba likov:
    - import java.awt.geom.*
    - shape = new Point2D.Float(x, y);
    - Line2D, Rect2D, CubicCurve2D, …
  - Risanje likov:
    - g2.draw(shape);
    - g2.fill(shape);
- Barve in fonti:
  - g2.setColor(new Color(r,g,b));
  - g2.setFont(new Font(…));
- Napredne metode:
  - g2.setStroke(…);
  - g2.setPaint(…);
  - g2.setComposite(…);
  - g2.setTransform(…);

Kako obnovimo sliko (Re-Paint)

- Zaslon je kot platno slikarja
  - Vsa okna se rišejo na isto ploskev!
  - Okna si ne zapomnijo, kaj je pod njimi
- Do ponovnega izrisa (re-paint) pride, ko dele spet prikažemo

- Dogodki, ki sprožijo ponoven izris
  - Odprtje okna, sprememba velikosti, premik iz ozadja
  - Kadar se okna v ospredju premaknejo, spremene velikost, zaprejo

Moja aplikacija



Ponoven izris okna aplikacije

Ponoven izris statične grafike

1. Podokno zbrišemo (zapolnimo ga z barvo ozadja)
2. Ponovno narišemo komponente v podoknu

Pri Java Swing komponente "ujamejo" dogodek "repaint" in kličejo svojo metodo paintComponent( ).

Koda

```
public class MyPanel extends JPanel {

     public void paintComponent(Graphics g){
    super.paintComponent(g);         // pocisti ozadje
    Graphics2D g2 = (Graphics2D)g;   //casta za java2

    // moja grafika:
    g2.setColor(new Color(255,0,0));
    g2.fillRect(10,10,200,50);
    g2.setColor(new Color(0,0,0));
    g2.drawString("Hello World", 10, 10);
  }
}
```

Tipična struktura programa za dinamično grafiko

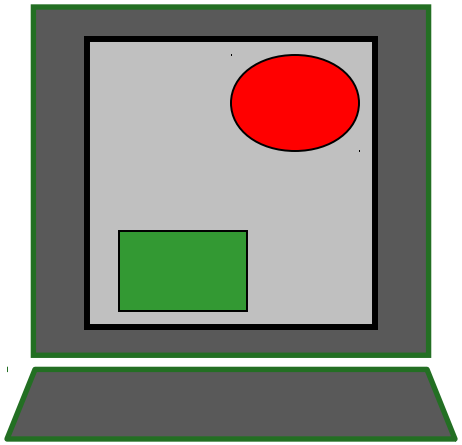
- Pomnimo podatkovno strukturo vsebine okna
  - Na pr. sliko, ko smo jo narisali v programu
- Dogodek "repaint ":
  - Počistimo okno (pobarvamo ga z barvo ozadja)
  - S pomočjo podatkovne strukture vsebino ona spet narišemo
- Drugi dogodki, ki spreminjajo vsebino okna:
  - Spremenijo podatkovno strukturo
  - pošljejo dogodek "repaint"

Kako shranimo vsebino okna

2 načina:

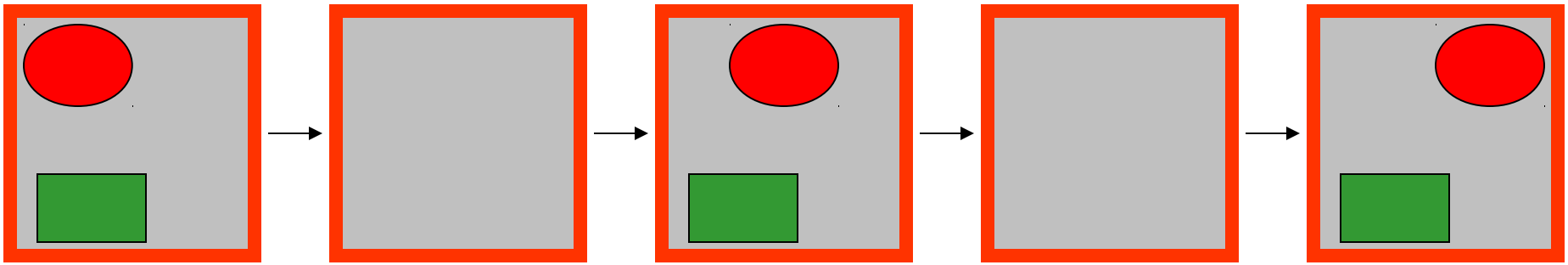
- Pomnimo logično vsebino v podatkovno strukturo
  - » na pr. risarski program: črte, like, barve, …
  - »
- Pomnimo vidno vsebino kot sliko ("off-screen image", bitmap)
  - » Pomnimo na primer piksle
  - » Nato sliko spet narišemo z
  - » g2.drawImage( ) v paintComponent( )

Problem: utripanje

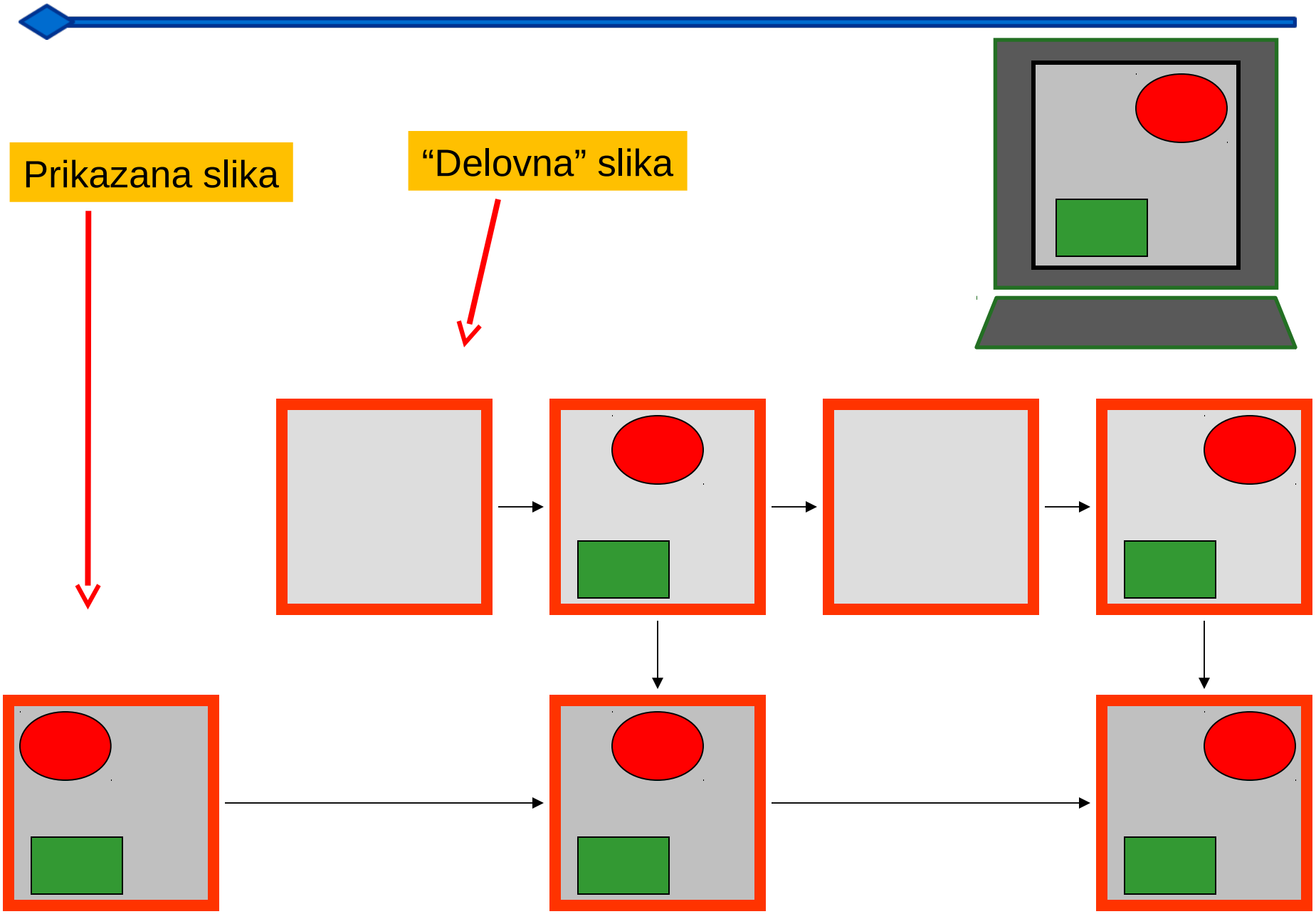


- Pri ponovnem izrisovanju pride do utripanja:

- Narišemo ozadje
- Ponovno izrišemo like



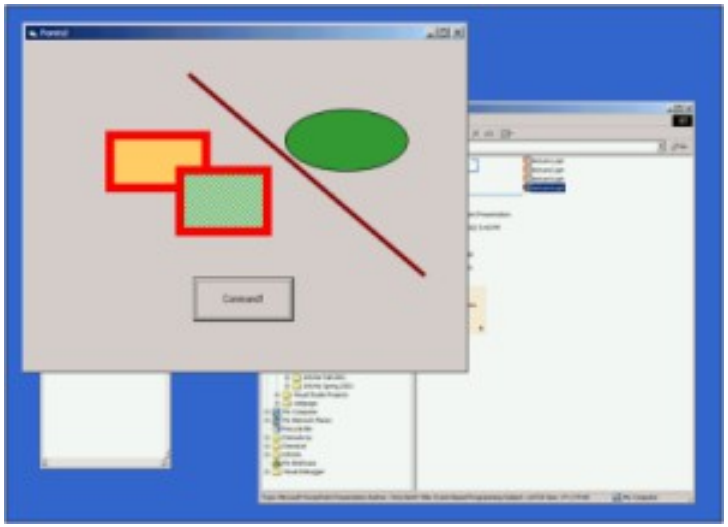
Rešitev: dvojno pomnenje (Double buffering)



Kaj delajo podokna?

- Usmerjajo vhod miške k pravi komponenti
- Ugotavljajo dogodke "repaint"
- Si lastijo koordinatni sistem
- Pri premikanju ne zahtevajo ponovnega izrisa
- Obrezovanje: skrivajo slike za oknom

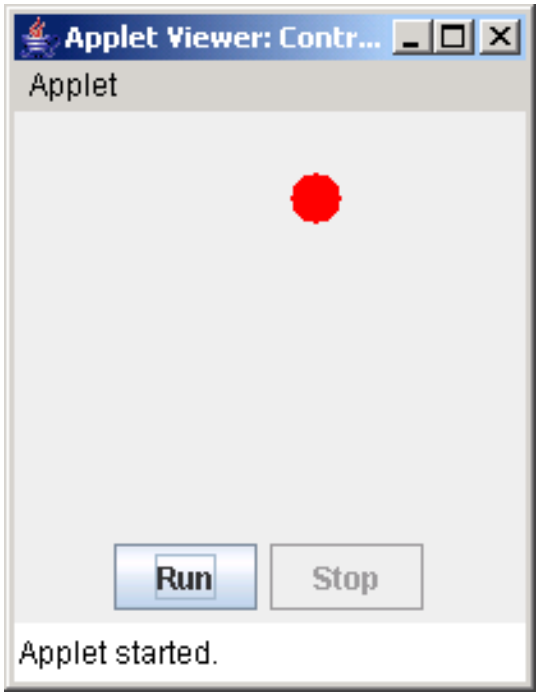
- Nekatera okna si zapomnijo vsebino pod njimi:
    - Popup menus



Animacija z Javo

Animirana žogica

Animirana žogica je “Hello World” animacije



- Spoznajmo ob tem:
  - Koncept Model-View-Controller
  - Koncept Observer-Observable
  - Niti
  - Časovnike (timers)

Demo

Kaj je že MVC?

- MVC pomeni Model-View-Controller
  - Model je trenutna interna predstavitev
    - Mora biti neodvisna od drugih razredov
    - Priročno je, če lahko ta razred podeduje Observable
  - View je način pogleda oziroma prikaza modela
    - Priročno je, če ta razred implementira Observer
  - Controller nudi vhode in spremembe uporabnika
- Te tri komponente običajno implementramo kot ločene razrede (ali skupine razredov)

In kaj sta Observer in Observable?

- java.util.Observable je razred
  - Ko naredi nekaj, kar bi morali opaziti, reče:
    - setChanged();
    - notifyObservers(); /* ali */ notifyObservers(***arg***);

- java.util.Observer je vmesnik
  - Registrirati se mora nekemu Observable:
    - ***myObservable***.addObserver(***myObserver***);
  - Zahteva implementacijo:
    - public void update(Observable ***obs***, Object ***arg***)
    - Ta metoda je avtomatično klicana, ko so na to "observerji" (opazovalci) opozorjeni
    - ***obs*** je objekt, ki ga opazujemo
    - Če Observable izvede notifyObservers(), je ***arg*** enak null

Observer = opazovalec
Observable = nekaj, kar lahko opazujemo

Primer Observer in Observable

```
import java.util.Observable;
import java.util.Observer;

public class MessageBoard extends Observable {
  private String message;

  public String getMessage() {  return message;  }

  public void changeMessage(String message) {
    this.message = message;
   setChanged();
    notifyObservers(message);
  }

 public static void main(String[] args) {
    MessageBoard board = new MessageBoard();
    Student janko = new Student("Janko");
    Student metka= new Student("Metka");
    board.addObserver(janko);
    board.addObserver(metka);
    board.changeMessage("Dobra novica, jutri je petek!");
    board.changeMessage("Slaba novica, to je pomota!");

  }
```

Primer Observer in Observable (nadaljevanje)

```
class Student implements Observer {
  String ime;

  public Student(String ime){
     this.ime = ime;
  }
  public void update(Observable o, Object arg) {
    System.out.println(ime + " sporoca: Novo obvestilo na tabli: " + arg);
  }
}
```

```
Metka sporoca: Novo obvestilo na tabli: Dobra novica, jutri je petek!
Janko sporoca: Novo obvestilo na tabli: Dobra novica, jutri je petek!
Metka sporoca: Novo obvestilo na tabli: Slaba novica, to je pomota!
Janko sporoca: Novo obvestilo na tabli: Slaba novica, to je pomota!
```

DEMO

Poslušalci (Listeners)

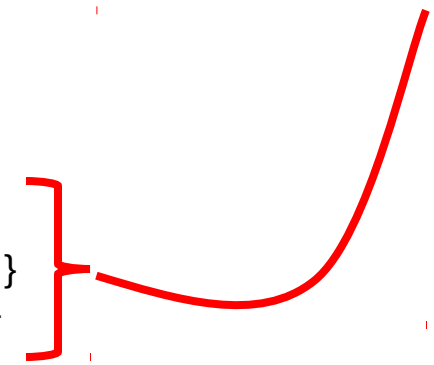
```
public class MouseEvents extends Applet
   implements MouseListener, MouseMotionListener {
   String msg = "";
   int mouseX = 0, mouseY = 0;

    public void init() {
       addMouseListener(this);
       addMouseMotionListener(this);
    }


 public void paint(Graphics g) {
        setBackground(new Color(255,255,200));
        g.drawString(msg, mouseX, mouseY);
    }
```

DEMO

Poslušalci (nadaljevanje)

```
public void mouseEntered (MouseEvent me)  {
      mouseX = 0; mouseY = 10;
      msg = "Vstop miske.";
      repaint();
   }

   public void mouseClicked (MouseEvent me)  {
      mouseX = me.getX();   mouseY = me.getY();
      msg = "Klik miske.";
     repaint();
   }

   public void mouseExited(MouseEvent me)  {
     mouseX = 0;   mouseY = 10;
     msg = "Izstop miske.";
     repaint();
   }

   public void mousePressed(MouseEvent me)  {    }
   public void mouseReleased(MouseEvent me)  {    }
   public void mouseDragged(MouseEvent me)  {    }

   public void mouseMoved(MouseEvent me) {
      showStatus("Premik miske" + me.getX() + ", " + me.getY());
   }
```

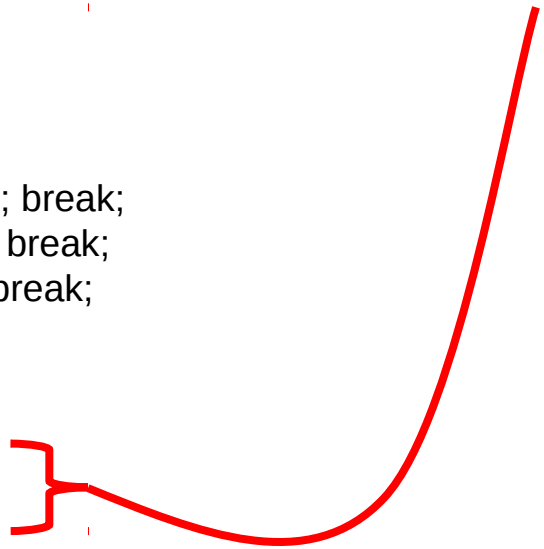
Poslušalci so vmesniki
Navesti moramo vse metode, četudi jih ne uporabljamo.



Poslušanje tipkovnice

```
public class KeyboardEvents extends Applet implements KeyListener  {
    int stevec = 0;
    String msg = "";
    Font f;          // font  izpisa niza lahko nastavimo

    public void init() {
        this.addKeyListener (this);  // This class has its own key listeners.
        this.setFocusable(true);    // Allow panel to get focus
    }

     public void paint(Graphics g) {
        f = new Font("Arial", Font.PLAIN, 24);
        msg = new Integer(stevec).toString();
        g.setFont(f);
        g.drawString(msg, 100, 300);
    }

    public void keyPressed ( KeyEvent e){
        switch (e.getKeyCode()) {
            case KeyEvent.VK_DELETE: stevec=0; break;
            case KeyEvent.VK_PLUS : stevec++;   break;
            case KeyEvent.VK_MINUS : stevec--; break;
        }
        repaint();
    }
    public void keyReleased ( KeyEvent e ){ }
    public void keyTyped ( KeyEvent e ){ }
}
```

Poslušalci so vmesniki
Navesti moramo vse metode, četudi jih ne uporabljamo.



DEMO

Virtual Key Codes

Alphanumerične tipke:

VK_0, VK_1, ..., VK_9, VK_A, VK_B, ..., VK_Z

Kontrolne tipke (Control keys):

VK_ENTER, VK_BACKSPACE, VK_TAB, VK_ESCAPE

Funkcijske tipke (Function keys):

VK_F1, VK_F2, VK_F3, VK_F4 VK_F5, VK_F6, VK_F7, VK_F8, VK_F9,
VK_F10, VK_F11, VK_F12,
VK_SCROLL_LOCK, VK_PRINTSCREEN, VK_PAUSE,
VK_DELETE, VK_INSERT,
VK_PAGE_UP, VK_PAGE_DOWN, VK_HOME, VK_END

Puščice (Arrow keys):

VK_LEFT, VK_RIGHT, VK_UP, VK_DOWN

In kako je z nitmi?

- Podedujemo lahko razred Thread:
  - class Animation extends Thread {…}
  - Omejuje nas možnost dedovanja enega samega razreda
  - Prekriti moramo metodo public void run( )
  - In nit pognati
    - Animation anim = new Animation( );
      anim.start( );
- Ali pa implementiramo vmesnik Runnable :
  - class Animation implements Runnable {…}
  - Implementirati moramo metodo public void run( )
  - In nit pognati
    - Animation anim = new Animation( );
      Thread myThread = new Thread(anim);
      myThread.start( );

Časovniki (Timers)

- Za časovni razpored izvajanja uporabimo java.util.Timer
- Timer lahko:
  - Razporedi enkratno izvajanje
  - Razporedi ponovna izvajanja v regularnih časovnih intervalih
- Konstruktorji Timer
  - Timer()
  - Timer(boolean ***isDaemon***)
  - Timer(String ***name***)
  - Timer(String ***name***, boolean ***isDaemon***)
- Timer lahko uporablja nit s podanim imenom. Ta nit je lahko tudi demonska (nudi servis drugim nitim).
  - Demonsko nit za časovnik tvorimo z
    - new Timer(true) ali
    - new Timer(***name***, true)

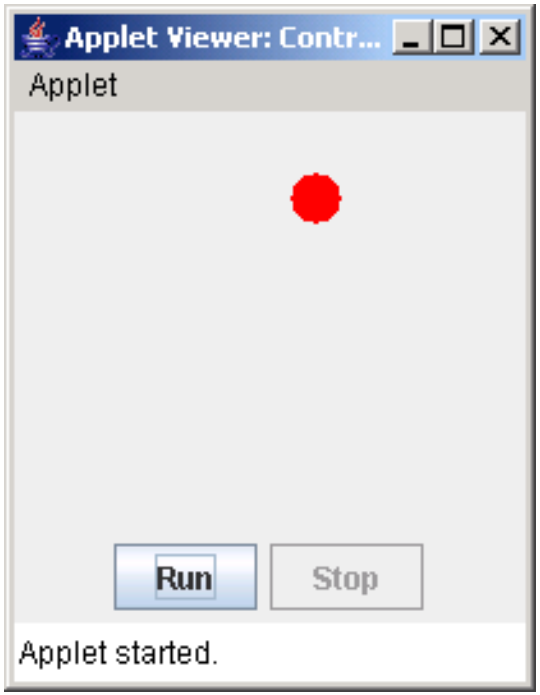
Uporaba časovnika v animaciji

- public void schedule(TimerTask ***task***, long ***delay***, long ***period***)
  - Razporedi določeno nalogo za ponavljanje v fiksnih časovnih intervalih, zakasnjeno za določeno zakasnitvijo (ki je lahko tudi nič)
  - Naslednja izvajanja bodo v približno regularnih intervalih
  - Časi so definirani v milisekundah

- Opomba: schedule terja kot argument TimerTask
  - TimerTask je abstraktni razred, ki ga moramo razširiti (podedovati) in implementirati metodo
    public void run()
  - TimerTask ima še (že implementirano) metodo public boolean cancel()
    - Ta vrne false , če ni potrebno brisati nobenih izvajanih metod

Animirana žogica (še enkrat)

Animirana žogica je “Hello World” animacije



- Spoznali smo ob tem:
  - Koncept Model-View-Controller
  - Koncept Observer-Observable
  - Niti
  - Časovnike (timers)

Demo

Če se gremo MVC, kje je tu model?

```
class Model extends Observable {
        public final int BALL_SIZE = 20;
        private int xPosition = 0;
        private int yPosition = 0;
        private int xLimit, yLimit;
        private int xDelta = 6;
        private int yDelta = 4;

        // metode (na naslednji prosojnici)

    }
```

Razred Model (nadaljevanje 1)

```
public void setLimits(int xLimit, int yLimit) {
        this.xLimit = xLimit - BALL_SIZE;
        this.yLimit = yLimit - BALL_SIZE;
    }

    public int getX() {
        return xPosition;
    }

    public int getY() {
        return yPosition;
    }

    public void makeOneStep() {
        // Koda za izvedbo enega koraka (na naslednji prosojnici)
    }
```

Razred Model (nadaljevanje 2)

```
public void makeOneStep() {
        // Do the work
        xPosition += xDelta;
        if (xPosition < 0 || xPosition >= xLimit) {
            xDelta = -xDelta;
            xPosition += xDelta;
        }
        yPosition += yDelta;
        if (yPosition < 0 || yPosition >= yLimit) {
            yDelta = -yDelta;
            yPosition += yDelta;
        }
        // Notify observers
        setChanged();
        notifyObservers();
    }
```

Razred View

```
import java.awt.*;
    import java.util.*;

    class View extends Panel implements Observer {
        Model model;

        View(Model model) {
            this.model = model;
        }

        public void paint(Graphics g) {
            g.setColor(Color.red);
            g.fillOval(model.getX(), model.getY(),
                    model.BALL_SIZE, model.BALL_SIZE);
        }

        public void update(Observable obs, Object arg) {
            repaint();
        }
    }
```

Razred Controller

```
import java.applet.*;
    import java.awt.*;
    import java.awt.event.*;
    import java.util.Timer;
    import java.util.TimerTask;
    import javax.swing.*;

    public class Controller extends JApplet {
        JPanel buttonPanel = new JPanel();
        JButton runButton = new JButton("Run");
        JButton stopButton = new JButton("Stop");
        Timer timer;

        Model model = new Model();
      View view = new View(model); // View must know about Model
```

Razred Controller (nadaljevanje 1)

```
public void init() {
        layOutComponents();
        attachListenersToComponents();

        // Connect model and view
        model.addObserver(view);
    }
private void layOutComponents() {
        setLayout(new BorderLayout());
          this.add(BorderLayout.SOUTH, buttonPanel);
             buttonPanel.add(runButton);
             buttonPanel.add(stopButton);
             stopButton.setEnabled(false);
          this.add(BorderLayout.CENTER, view);
     }
```

Razred Controller (nadaljevanje 2)

```
private void attachListenersToComponents() {
        runButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                runButton.setEnabled(false);
                stopButton.setEnabled(true);
               timer = new Timer(true);
               timer.schedule(new Strobe(), 0, 40); // 25 times a second
            }
        });
        stopButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
               runButton.setEnabled(true);
               stopButton.setEnabled(false);
               timer.cancel();
            }
        });
    }
```

Razred Controller (nadaljevanje 3)

```
private class Strobe extends TimerTask {
        public void run() {
            model.setLimits(view.getWidth(), view.getHeight());
            model.makeOneStep();
        }
    }
```

Povzetek primera z animacijo

V tem primeru smo uporabili:

- Model-View-Controller
  - Dober načrtovalski vzorec (design pattern) za različne : loči “poslovno logiko” (Model) od razredov, ki v osnovi predstavljajo vhod-izhod
- Observer-Observable
  - Dober načrtovalski vzorec, ki izolira Model od prikaza (View)
- Threads
  - Če hočemo imeti animacijo, ki jo lahko nadzorujemo, so niti bistvene. Animacija teče v eni niti, kontrole tečejo v drugi
- Timers
  - Časovniki so pripraven način razporejanja nalog, ki se ponavljajo v časovnih intervalih
  - Seveda bi lahko uporabili tudi drugačen koncept in namesto tega uporabili Thread.sleep(***ms***)

Primeri z Java2D

WEB

Afine transformacije likov
Afine transformacije slike