# 2D Graphics in Java

**Introduction**

**Painting of containers and atomic components**

**Custom painting of a container or atomic component**

**Drawing instructions**

**Examples: how to draw**

**Interactive Animation**

**Example: basic video game**

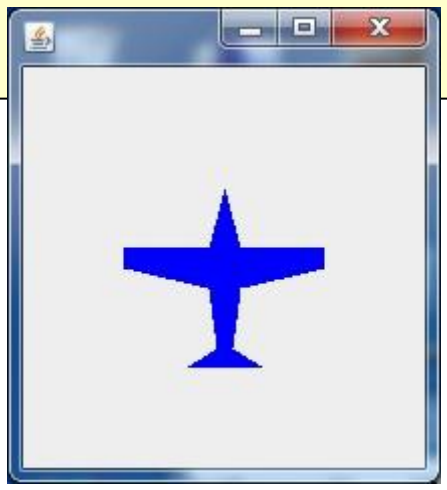
**References:**

http://download.oracle.com/javase/tutorial/2d/

# Introduction

*example: drawing a polygon*

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

public class P
{  public static void main(String[] arg)
   {  Gui gui = new Gui(); } }


class Gui
{
   JFrame f;    DrawingPanel p;

   int n = 13;
   int[] x = new int[] { 100,   92,   50,   50,   92,   96,   80, 120, 104, 108, 150, 150, 108 };
   int[] y = new int[] {   60,   90,   90, 100, 110, 140, 150, 150, 140, 110, 100,   90,   90 };

   class DrawingPanel extends JPanel
   {  public void paintComponent(Graphics g)
      {  super.paintComponent(g);

         g.setColor(Color.blue);
         g.fillPolygon(x , y , n);
      }
   }

   Gui()
   {
      f = new JFrame();    f.setFocusable(true);    f.setVisible(true);
      p = new DrawingPanel();    f.getContentPane().add(p , BorderLayout.CENTER);

      f.setSize(new Dimension(200 + 16, 200 + 38));
   }
}
```



# Painting of containers and atomic components

- the painting of the elements (containers and atomic components of the GUI is performed by the *GUI managing thread*

  for each element, the thread calls over the element its specific version of the instance method **paintComponent**

  **paintComponent** is predefined for each element
  = standard drawing of a button, of a panel (rectangle), . . .

  **paintComponent** receives a **Graphics** object **g** associated to the element

- inside our program, we can call **c.repaint();** for container or atomic component **c**

  → request to the *GUI managing thread* to call as soon as possible:
  - first, the **paintComponent** method of **c**
  - then, the **paintComponent** methods of ALL the elements inside **c**

more generally, **frame.repaint();** requests the repainting of the whole GUI

we must NEVER call ourselves **paintComponent** in our program

# Custom painting of a container or atomic component

- We must subclass the class of the element so as to override its paintComponent method

  → we define a specific way of drawing the element

in general, we should draw only on panels → subclass the class **JPanel**

inside **paintComponent**, first call the overriden **paintComponent** method

= do the standard drawing of the panel (rectangle) → erase previous drawings

→ the first line of **paintComponent** must be **super.paintComponent(g);**

- blueprint for our drawing panel:

```
class Drawing_Panel extends Jpanel
{
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);   // standard drawing of JPanel (rectangle)

      int w = this.getWidth();  int h = this.getHeight();  // height,width of the panel

      // call some drawing methods:
      g.setColor( . . . );
      g.setFont( . . . );
      g.drawLine( . . . );
      g.drawString( . . . );
      . . . . . .
   }
}
```

# Drawing instructions

## **Graphics** object:

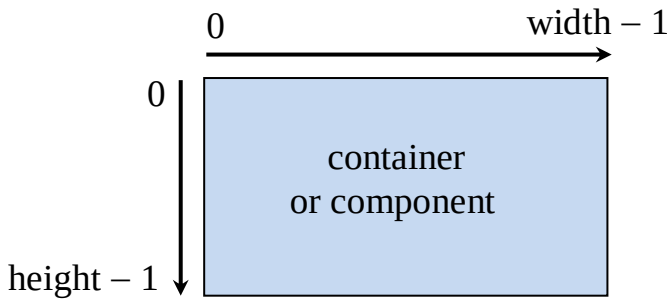
to each element is associated an object **g** instance of **Graphics**
= contains informations for drawing inside the element

**g** is readily available as the argument of **paintComponent**

draw inside the element → apply some drawing methods over **g**

## coordinate system:

all drawings are done relative to the local coordinate system of the element:




the width and height of the element are obtained with **getWidth()** and **getHeight()**

= we call them over the element object (**this** inside **paintComponent**)


coordinates are all integers (**int**)


DO NOT call **getWidth()** or **getHeight()** over **g**

set the current drawing color: **g.setColor(color);**

set the current font: **g.setFont(font);**

draw strings, lines, polygons, rectangles, ovals:

**g.drawString(string , x , y);**

**g.drawLine(x1 , y1 , x2 , y2);**

**g.drawPolyline(array x , array y , n. of points);**

**g.drawPolygon(array x , array y , n. of points);**

**g.fillPolygon(array x , array y , n. of points);**

**g.drawRect(x , y , width , height);**

**g.fillRect(x , y , width , height);**

**g.drawOval(x , y , width , height);**

**g.fillOval(x , y , width , height);**

draw images:

the image coordinates correspond to the upper left corner of the image

the image is stored in a file *.jpeg* or *.gif*

- create an object instance of the class **Image**:

  **Image a = Toolkit . getDefaultToolkit() . getImage(file_name_string);**

- then, draw the image

  **g . drawImage(a , x , y , this);** (**this** is the drawing panel)

geometric transformations:

*device space*: coordinate system of the panel

*user space*: coordinate system in which **x** and **y** are expressed for all drawing instructions

*affine transformation* : straight lines remain straight and parallel lines remain parallel

Java maintains an affine transformation between *device space* and *user space*
= by default, *identity transformation*
*or:* *composition* of a succession of basic affine transformations (translation, rotation, scaling)

- first, obtain the **Graphics2D** object: **Graphics2D g2 = (Graphics2D)g;**
- save the current affine transformation: **AffineTransform at0 = g2.getTransform();**
- translation: **g2.translate(*dx* , *dy*);**
  rotation: **g2.rotate(*rot*);**
  scaling: **g2.scale(*scale_x* , *scale_y*);**
- restore the saved affine transformation: **g2.setTransform(at0);**

```
import java.awt.geom.*;

class DrawingPanel extends JPanel
{  public void paintComponent(Graphics g)
   {  super.paintComponent(g);
      Graphics2D g2 = (Graphics2D)g;

      drawing instruction(s) over other non transformed geometric object(s)

      AffineTransform at0 = g2.getTransform();

      g2.translate(dx , dy);    g2.rotate(rot);    g2.scale(scale_x , scale_y);

      drawing instruction(s) over transformed geometric object(s)

      g2.setTransform(at0);

      drawing instruction(s) over other non transformed geometric object(s)
   }
}
```

test polygon contains point:

- test if the polygon **(X , Y , n)** contains or not the point **(x , y)**

  first, create a polygon object:

  **Polygon polygon = new Polygon(X , Y , n);**

  then, apply method **contains** over the polygon (returns **true** or **false**):

  **polygon.contains(x , y)**

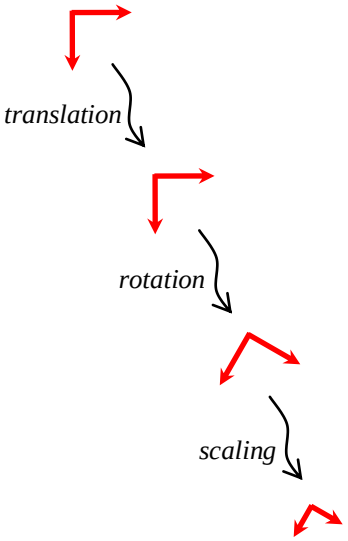
- can be used to select a polygon by clicking on it:

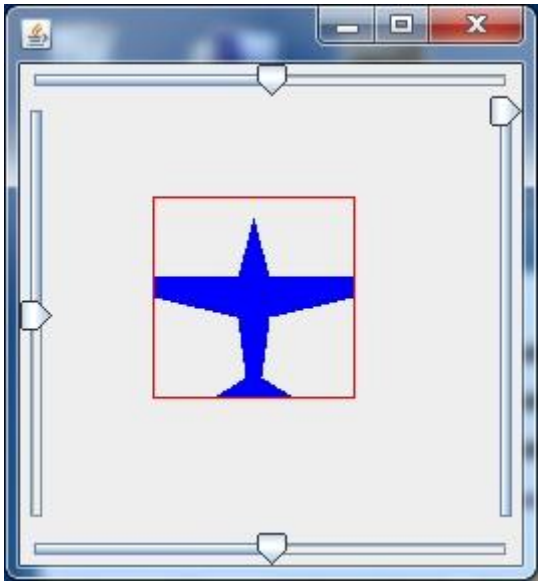
```
p.addMouseListener(new MouseAdapter()
{  public void mouseClicked(MouseEvent e)
   {
      Polygon polygon = new Polygon(X , Y , n);
      if (polygon.contains(e.getX() , e.getY()))
         . . . . . .
```

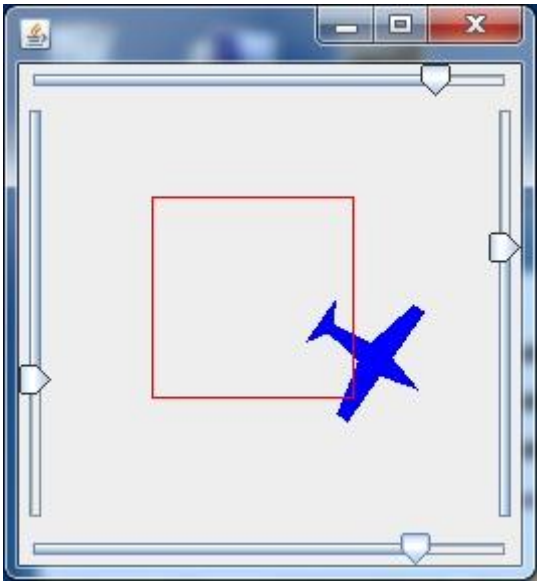
*problem:* how to apply this test easily on a translated / rotated / scaled polygon ?

# Examples: how to draw

*example 1: geometric transformations*



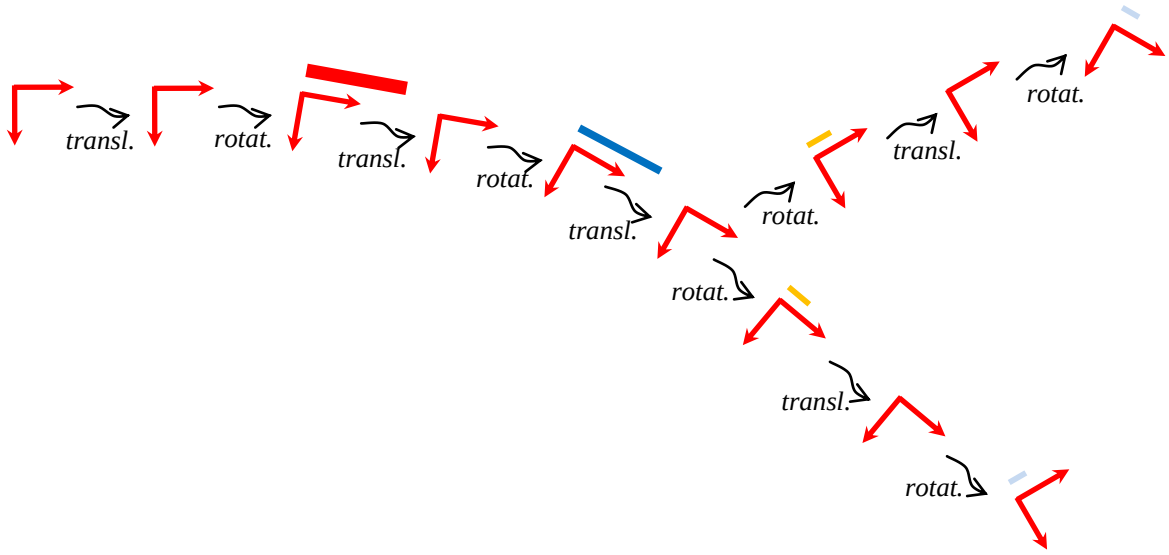


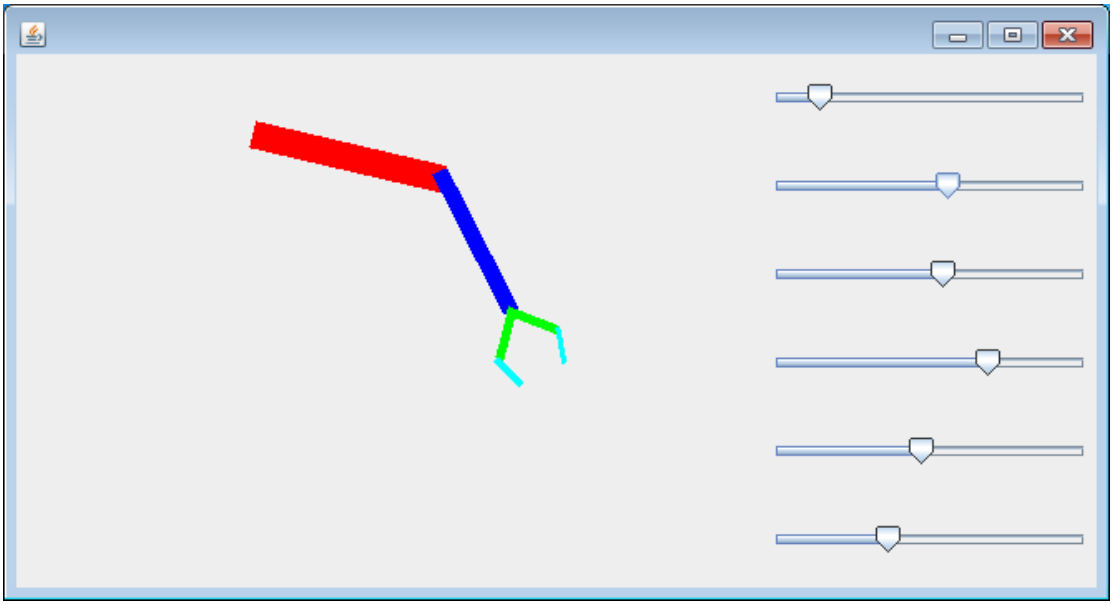
```
import java.awt.geom.*;

class Gui
{
   JFrame f;    DrawingPanel p;    JSlider stx , sty , srot , sscale;
   int tx = 100 , ty = 100;    double rot = 0.0 , scale = 1.0;

   int n = 13;
   int[] x = new int[] {    0,   -8,  -50,  -50,   -8,    -4,  -20,   20,    4,    8,   50,   50,    8 };
   int[] y = new int[] {  -40, -10,  -10,    0,   10,   40,   50,   50,   40,   10,    0,  -10, -10 };

   class DrawingPanel extends JPanel
   {  public void paintComponent(Graphics g)
      {  super.paintComponent(g);
         Graphics2D g2 = (Graphics2D)g;
         AffineTransform at0 = g2.getTransform();

         g2.translate(tx , ty);    g2.rotate(rot);    g2.scale(scale , scale);

         g.setColor(Color.blue);    g.fillPolygon(x , y , n);

         g2.setTransform(at0);

         g.setColor(Color.red);    g.drawRect(50 , 50 , 100 , 100);
      }
   }
```

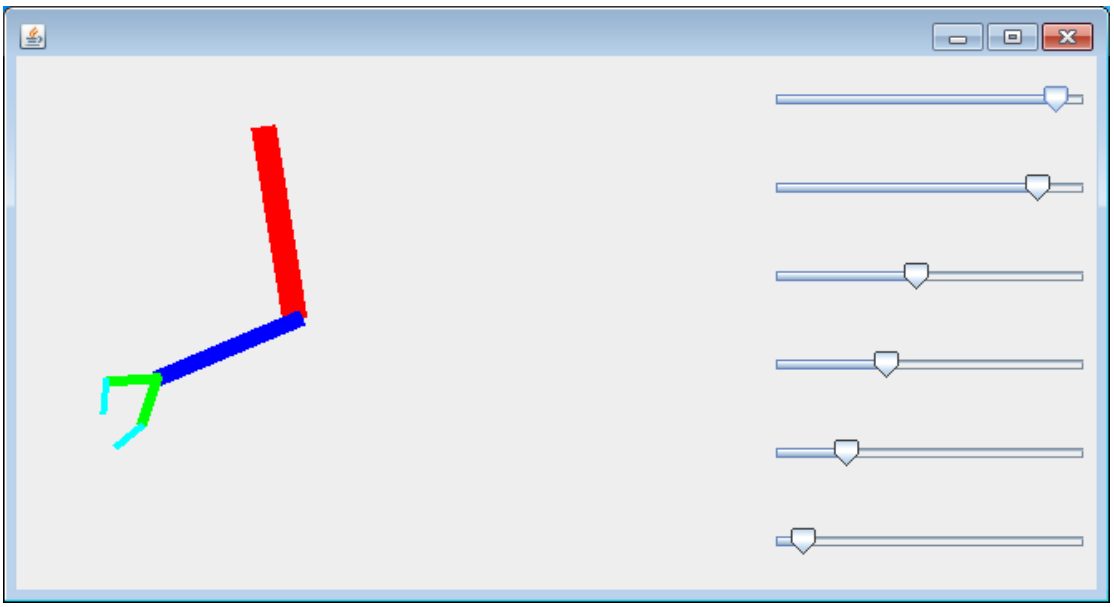
*example 1 (continued):*

```
   Gui()
   {
      f = new JFrame();    f.setFocusable(true);    f.setVisible(true);
      p = new DrawingPanel();    f.getContentPane().add(p , BorderLayout.CENTER);

      stx = new JSlider(JSlider.HORIZONTAL , 0 , 200 , 100);
      sty = new JSlider(JSlider.VERTICAL       , 0 , 200 , 100);    sty.setInverted(true);
      srot    = new JSlider(JSlider.HORIZONTAL , -180 , +180 , 0);
      sscale = new JSlider(JSlider.VERTICAL , 0 , 100 , 100);

      f.getContentPane().add(stx      , BorderLayout.SOUTH);
      f.getContentPane().add(sty      , BorderLayout.WEST);
      f.getContentPane().add(srot     , BorderLayout.NORTH);
      f.getContentPane().add(sscale , BorderLayout.EAST);

      stx.addChangeListener(new ChangeListener()
      {  public void stateChanged(ChangeEvent e)
         {  tx = stx.getValue();    f.repaint();  } } );

      sty.addChangeListener(new ChangeListener()
      {  public void stateChanged(ChangeEvent e)
         {  ty = sty.getValue();    f.repaint();  } } );

      srot.addChangeListener(new ChangeListener()
      {  public void stateChanged(ChangeEvent e)
         {  rot = (Math.PI / 180) * srot.getValue();    f.repaint();  } } );

      sscale.addChangeListener(new ChangeListener()
      {  public void stateChanged(ChangeEvent e)
         {  scale = (1.0 / 100) * sscale.getValue();    f.repaint();  } } );

      f.setSize(new Dimension(250 + 16, 250 + 38));
   }
}
```

*example 2: robotic arm*





*example 2 (continued):*

```
import java.awt.geom.*;

class Gui
{
   JFrame f;    DrawingPanel p;    JPanel ps;    JSlider sa0 , sa1 , sa2 , sa3 , sa4 , sa5;
   double a0 =     0 , a1 =   0 , a2 =    0 , a3 =   0 , a4 =   0 , a5 =   0;
   int       b0 = 110 , b1 = 90 , b2 = 30 , b3 = 20 , b4 = 30 , b5 = 20;
   int       h0 =   16 , h1 = 10 , h2 =   6 , h3 =   4 , h4 =   6 , h5 =   4;

   class DrawingPanel extends JPanel
   {  public void paintComponent(Graphics g)
      {  super.paintComponent(g);
         Graphics2D g2 = (Graphics2D)g;

         g2.translate(150 , 50);    g2.rotate(a0);
         g.setColor(Color.red);    g.fillRect( - h0 / 2 , - h0 / 2 , b0 + h0 / 2 , h0);

         g2.translate(b0 , 0);    g2.rotate(a1);
         g.setColor(Color.blue);    g.fillRect( - h1 / 2 , - h1 / 2 , b1 + h1 / 2 , h1);

         g2.translate(b1 , 0);

         AffineTransform at = g2.getTransform();

            g2.rotate(a2);
            g.setColor(Color.green);    g.fillRect( - h2 / 2  , - h2 / 2 , b2 + h2 / 2 , h2);

            g2.translate(b2 , 0);    g2.rotate(a3);
            g.setColor(Color.cyan);    g.fillRect( - h3 / 2  , - h3 / 2 , b3 + h3 / 2 , h3);

         g2.setTransform(at);

         g2.rotate(a4);
         g.setColor(Color.green);    g.fillRect( - h4 / 2  , - h4 / 2 , b4 + h4 / 2 , h4);

         g2.translate(b4 , 0);    g2.rotate(a5);
         g.setColor(Color.cyan);    g.fillRect( - h5 / 2  , - h5 / 2 , b5 + h5 / 2 , h5);
      }
   }
```

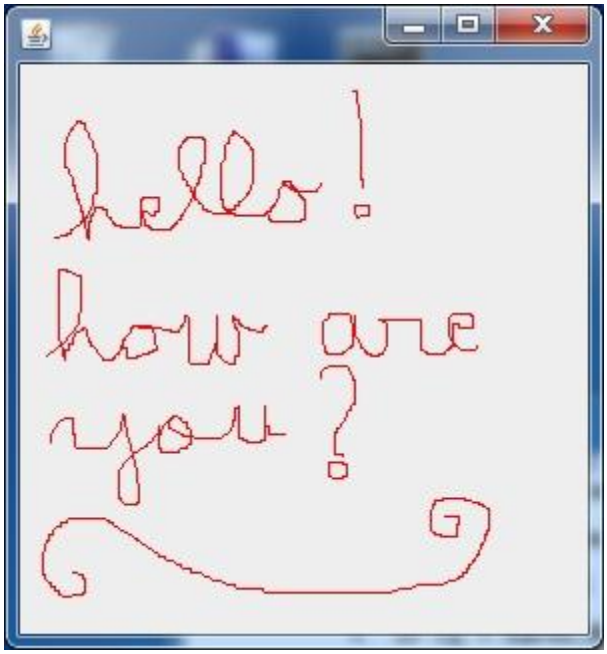
*example 2 (continued):*

```
  Gui()
   {
      f = new JFrame();    f.setFocusable(true);    f.setVisible(true);
      p = new DrawingPanel();
      f.getContentPane().add(p   , BorderLayout.CENTER);
      ps = new JPanel();    ps.setLayout(new GridLayout(0 , 1));
      f.getContentPane().add(ps , BorderLayout.EAST);

      sa0 = new JSlider(JSlider.HORIZONTAL ,    0 , +90 , 0);    ps.add(sa0);
      sa1 = new JSlider(JSlider.HORIZONTAL ,    0 , +90 , 0);    ps.add(sa1);
      sa2 = new JSlider(JSlider.HORIZONTAL , -90 ,     0 , 0);    ps.add(sa2);
      sa3 = new JSlider(JSlider.HORIZONTAL ,    0 , +90 , 0);    ps.add(sa3);
      sa4 = new JSlider(JSlider.HORIZONTAL ,    0 , +90 , 0);    ps.add(sa4);
      sa5 = new JSlider(JSlider.HORIZONTAL , -90 ,     0 , 0);    ps.add(sa5);

      sa0.addChangeListener(new ChangeListener()
      {  public void stateChanged(ChangeEvent e)
         {  a0 = (Math.PI / 180) * sa0.getValue();    f.repaint();  } } );

      sa1.addChangeListener(new ChangeListener()
      {  public void stateChanged(ChangeEvent e)
         {  a1 = (Math.PI / 180) * sa1.getValue();    f.repaint();  } } );

      sa2.addChangeListener(new ChangeListener()
      {  public void stateChanged(ChangeEvent e)
         {  a2 = (Math.PI / 180) * sa2.getValue();    f.repaint();  } } );

      sa3.addChangeListener(new ChangeListener()
      {  public void stateChanged(ChangeEvent e)
         {  a3 = (Math.PI / 180) * sa3.getValue();    f.repaint();  } } );

      sa4.addChangeListener(new ChangeListener()
      {  public void stateChanged(ChangeEvent e)
         {  a4 = (Math.PI / 180) * sa4.getValue();    f.repaint();  } } );

      sa5.addChangeListener(new ChangeListener()
      {  public void stateChanged(ChangeEvent e)
         {  a5 = (Math.PI / 180) * sa5.getValue();    f.repaint();  } } );

      f.setSize(new Dimension(650 + 16, 320 + 38));
   }
}
```

*example 3: scribbling with the mouse*

```
class Gui
{
   JFrame f;    DrawingPanel p;

   static final int M = 1000;    int m = 0;
   Curve[] curve = new Curve[M];

   class Curve
   {  static final int N = 1000;    int n = 0;
      int[] x = new int[N] , y = new int[N];  }

   class DrawingPanel extends JPanel
   {  public void paintComponent(Graphics g)
      {  super.paintComponent(g);

         g.setColor(Color.red);
         for (int i = 0 ; i < m ; i++)
            g.drawPolyline(curve[i].x , curve[i].y , curve[i].n);
      }
   }

   Gui()
   {
      f = new JFrame();    f.setFocusable(true);    f.setVisible(true);
      p = new DrawingPanel();    f.getContentPane().add(p , BorderLayout.CENTER);

      p.addMouseListener(new MouseAdapter()
      {  public void mousePressed(MouseEvent e)
         {  if (m < M)
            {  m++;    curve[m - 1] = new Curve();  }  } } );

      p.addMouseMotionListener(new MouseMotionAdapter()
      {  public void mouseDragged(MouseEvent e)
         {  Curve c = curve[m - 1];
            if (c.n < Curve.N)  {  c.n++;    c.x[c.n - 1] =  e.getX();
                                             c.y[c.n - 1] = e.getY();  }
            f.repaint(); } } );

      f.setSize(new Dimension(300 + 16 , 300 + 38));
   }
}
```

*example 3(continued):*



*example 4: image drawing, file chooser*

```
class Gui
{
   String back_image_file , front_image_file;
   Image back_image = null , front_image = null;

   boolean drag = false;    int x_drag , y_drag;

   JFrame f;    DrawingPanel p;    JFileChooser fc;
   JMenuBar mb;    JMenu m0;    JMenuItem mi00 , mi01 , mi02;

   class DrawingPanel extends JPanel
   {   public void paintComponent(Graphics g)
       {   super.paintComponent(g);

           if (back_image != null)
              g.drawImage(back_image  , 0 , 0 , this);

           if (front_image != null && drag)
              g.drawImage(front_image , x_drag - 15 , y_drag - 15 , this);
       }
   }

  Gui()
  {
     f = new JFrame();    f.setFocusable(true);    f.setVisible(true);
     p = new DrawingPanel();    f.getContentPane().add(p , BorderLayout.CENTER);
     fc = new JFileChooser();

     mb = new JMenuBar();    f.setJMenuBar(mb);
     m0 = new JMenu();    m0.setText("load");    mb.add(m0);
     mi00 = new JMenuItem();    mi00.setText("load back image");    m0.add(mi00);
     mi01 = new JMenuItem();    mi01.setText("load front image");    m0.add(mi01);

     mi00.addActionListener(new ActionListener()
     {  public void actionPerformed(ActionEvent e)
        {  if (fc.showDialog(f , "enter") == JFileChooser.APPROVE_OPTION)
              back_image = Toolkit.getDefaultToolkit().getImage(
                                                    fc.getSelectedFile().getPath());
        f.repaint(); } } );
```

*example 4 (continued):*

```
    mi01.addActionListener(new ActionListener()
    {  public void actionPerformed(ActionEvent e)
       {  if (fc.showDialog(f , "enter") == JFileChooser.APPROVE_OPTION)
             front_image = Toolkit.getDefaultToolkit().getImage(
                                                    fc.getSelectedFile().getPath());  } } );

    p.addMouseMotionListener(new MouseMotionAdapter()
    {  public void mouseDragged(MouseEvent e)
       {  drag = true;  x_drag = e.getX();  y_drag = e.getY();    f.repaint(); } } );

    p.addMouseListener(new MouseAdapter()
    {  public void mouseReleased(MouseEvent e)
       {  drag = false;    f.repaint(); } } );

    f.setSize(new Dimension(400 + 16 , 300 + 60));
   }
}
```

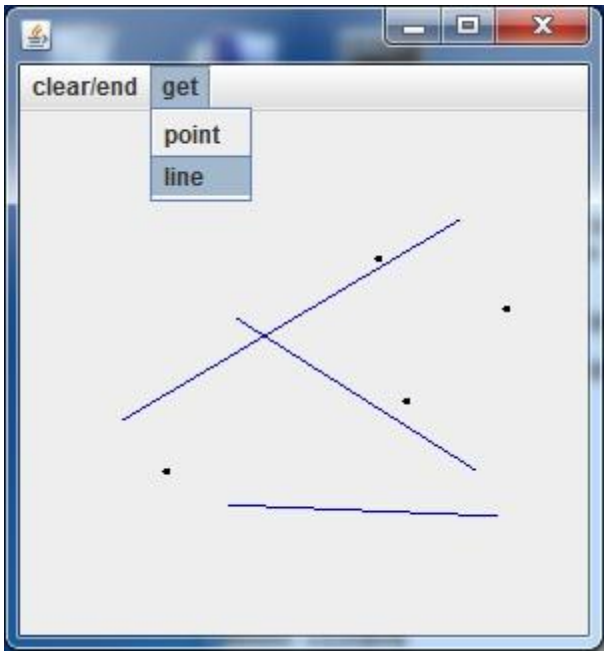
*example 5: drawing program (points, lines)*

```
class Point {  int x , y;  }
class Line  {  int x1 , y1;    int x2 , y2;  }


class Gui
{
   static final int N = 20;
   JFrame f;    JPanel p;    JMenuBar mb;    JMenu m0 , m1;
   JMenuItem mi00 , mi01 , mi10 , mi11;

   Point[] point = new Point[N];    Boolean mode_getpoint = false;    Point in_point;
   Line[]  line    = new Line[N];      int         mode_getline   = 0;         Line  in_line;

   class DrawingPanel extends JPanel
   {  public void paintComponent(Graphics g)
      {  super.paintComponent(g);

         g.setColor(Color.black);
         for (int i = 0 ; i < N ; i++)
            if (point[i] != null  &&  (!mode_getpoint  ||  point[i] != in_point))
               g.fillOval(point[i].x - 2 , point[i].y - 2 , 4 , 4);

         g.setColor(Color.blue);
         for (int i = 0 ; i < N ; i++)
            if (line[i] != null    &&  (mode_getline == 0  ||  line[i]  != in_line))
               g.drawLine(line[i].x1 , line[i].y1 , line[i].x2 , line[i].y2);
      }
   }

   Gui()
   {
      f = new JFrame();    f.setFocusable(true);    f.setVisible(true);
      p = new DrawingPanel();     f.getContentPane().add(p , BorderLayout.CENTER);

      mb = new JMenuBar();  f.setJMenuBar(mb);
      m0 = new JMenu();    m0.setText("clear/end");    mb.add(m0);
      mi00 = new JMenuItem();    mi00.setText("clear");    m0.add(mi00);
      mi01 = new JMenuItem();    mi01.setText("end");      m0.add(mi01);
      m1 = new JMenu();    m1.setText("get");            mb.add(m1);
      mi10 = new JMenuItem();    mi10.setText("point");    m1.add(mi10);
      mi11 = new JMenuItem();    mi11.setText("line");     m1.add(mi11);
```

*example 5 (continued):*

```
    mi00.addActionListener(new ActionListener()
      {  public void actionPerformed(ActionEvent e)
        {  for (int i = 0 ; i < N ; i++)  {  point[i] = null;    line[i] = null;  }
          f.repaint();  } } );

      mi01.addActionListener(new ActionListener()
      {  public void actionPerformed(ActionEvent e)
        {  System.exit(0); } } );

      mi10.addActionListener(new ActionListener()
      {  public void actionPerformed(ActionEvent e)
        {  if (mode_getline == 0)
          {  mode_getpoint = true;
            int i;    for (i = 0 ; i < N ; i++)
                  if (point[i] == null)  {  in_point = point[i] = new Point();   break;  }
            if (i == N)  mode_getpoint = false;  } } } );

      mi11.addActionListener(new ActionListener()
      {  public void actionPerformed(ActionEvent e)
        {  if (mode_getpoint == false)
          {  mode_getline = 1;
            int i;    for (i = 0 ; i < N ; i++)
                  if (line[i] == null)  {  in_line = line[i] = new Line();    break;  }
            if (i == N)  mode_getline = 0;  } } } );

      p.addMouseListener(new MouseAdapter()
      {  public void mouseClicked(MouseEvent e)
        {
          if      (mode_getpoint)      {  in_point.x = e.getX();    in_point.y = e.getY();
                                          mode_getpoint = false;    f.repaint();  }

          else if (mode_getline == 1)  {  in_line.x1 = e.getX();    in_line.y1 = e.getY();
                                          mode_getline = 2;  }

          else if (mode_getline == 2)  {  in_line.x2 = e.getX();    in_line.y2 = e.getY();
                                          mode_getline = 0;          f.repaint();  }
        } } );

      f.setSize(new Dimension(300 + 16 , 300 + 60));
    }
  }
```

*example 5(continued):*



# Interactive Animation

Principle:

- calculate the evolution over time of a given "world"
  made up of objects following certain rules

  → succession of world states at times $t_0$ , $t_0 + \Delta t$ , $t_0 + 2\,\Delta t$ , $t_0 + 3\,\Delta t$ , $t_0 + 4\,\Delta t$ , ...

  the variables describing the state at $t_0 + i\,\Delta t$ are incremented → state at $t_0 + (i + 1)\,\Delta t$

  $$x \leftarrow x + v_x\,\Delta t$$
  $$y \leftarrow y + v_y\,\Delta t$$

- the human operator can influence the world's evolution
  with instructions given through a GUI

animation loop:

```
while (not over)
{
   evolve the world over one time step

   repaint the drawing panel

   synchronize real time with animation time
}
```

```
while (not over)
{
   long t_start = System.currentTimeMillis();

   world.evolve();
   gui.drawingpanel.repaint();

   long dt_real = System.currentTimeMillis() - t_start;
   if (dt_real < dt)  try {Thread.sleep(dt  - dt_real);} catch(InterruptedException e){}
                 else  System.out.println("PC too slow; please increase dt");
}
```

# Example: basic video game

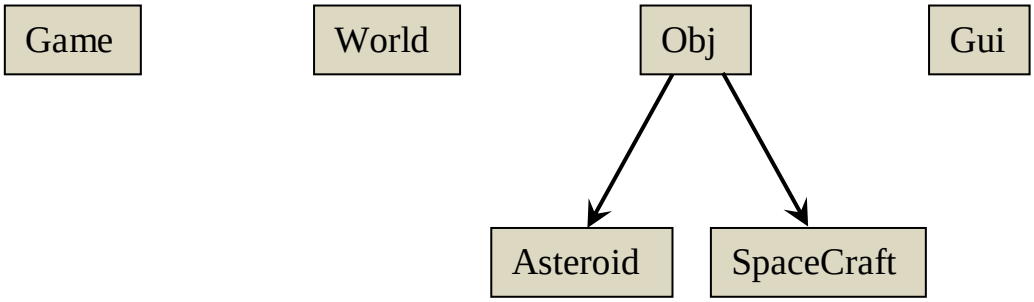
## overview:

world = { asteroids with constant speed (initially fixed at random) ; spacecraft whose speed (*direction*, *throttle*) is controlled through the GUI }

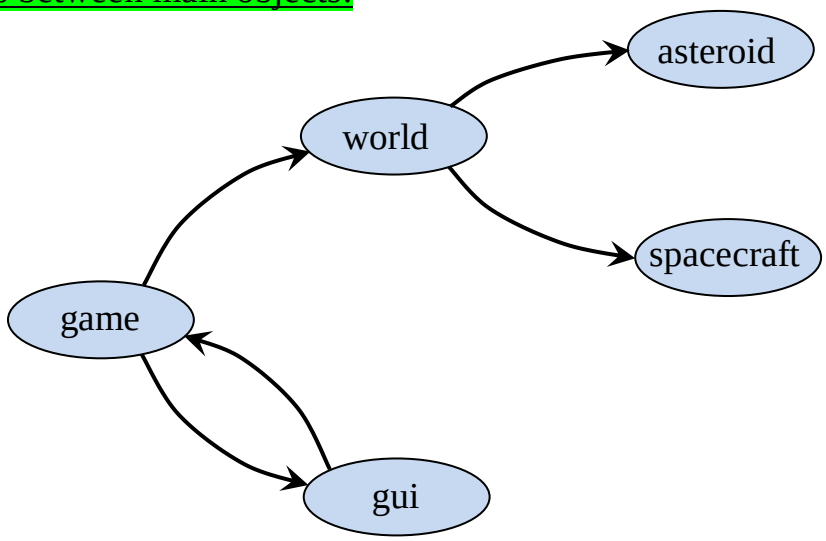
control over *direction*: **COMdir** = -1 , 0 , +1 <=> turn left , nothing , turn right

control over *throttle*: **COMthr** = -1 , 0 , +1 <=> throttle down , nothing , throttle up

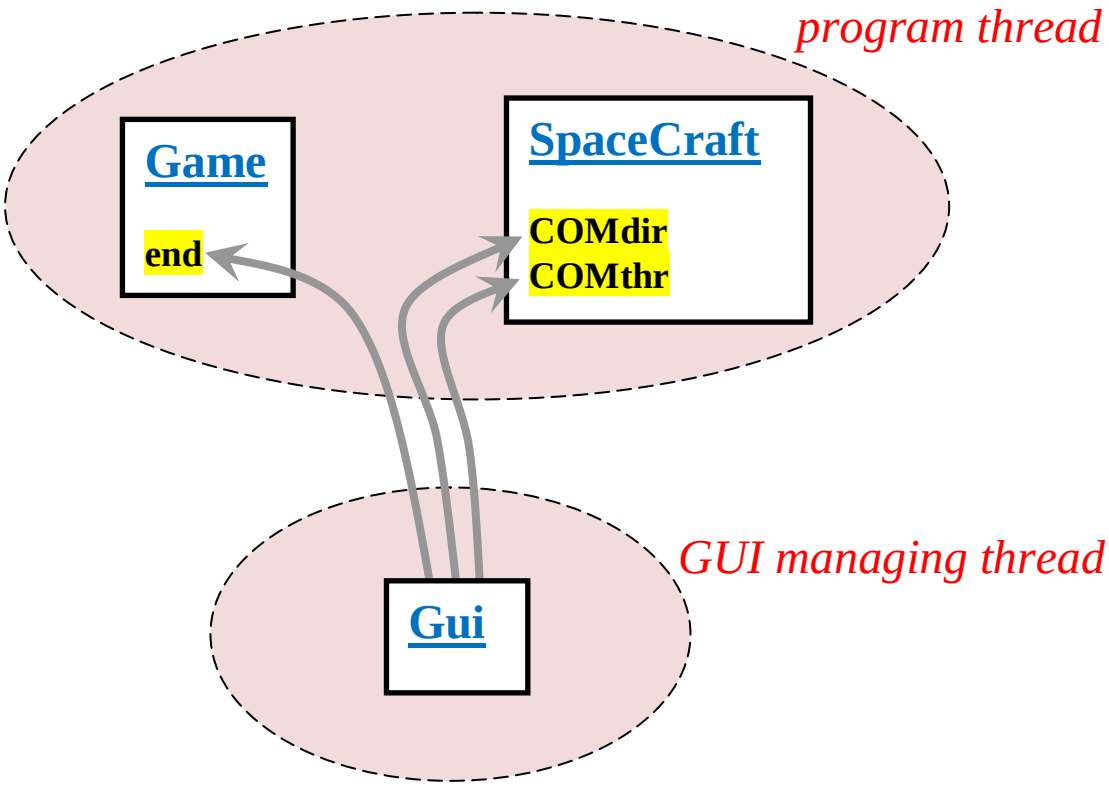
## classes:




## referencings between main objects:



threads and shared data:



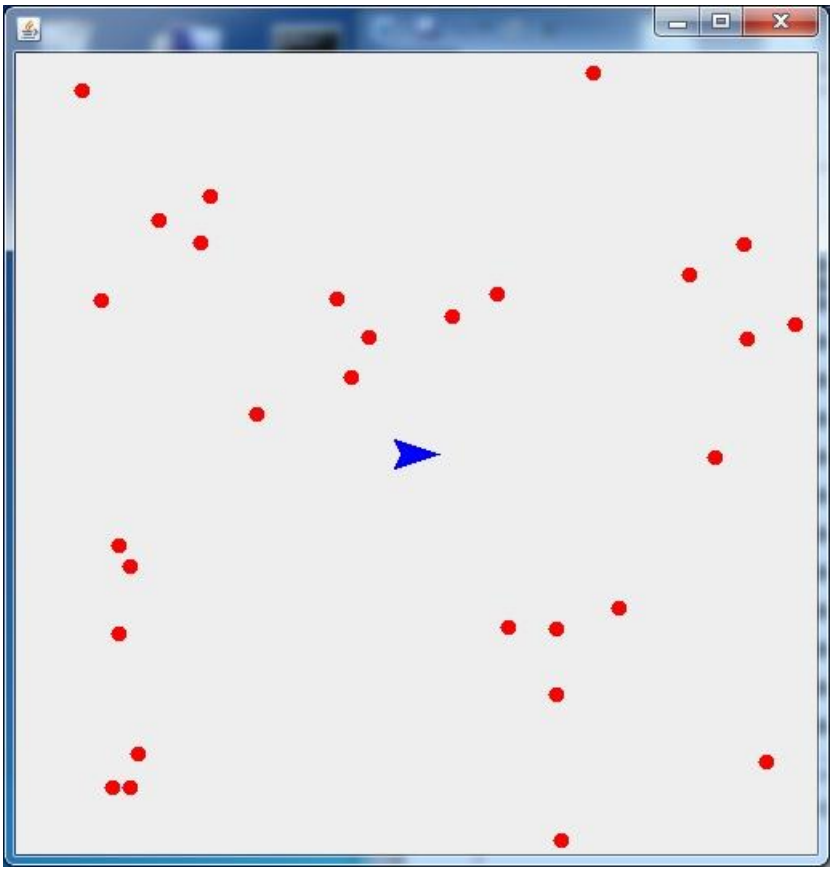

we must preserve the coherence of shared variables **end** , **COMdir** , **COMthr**

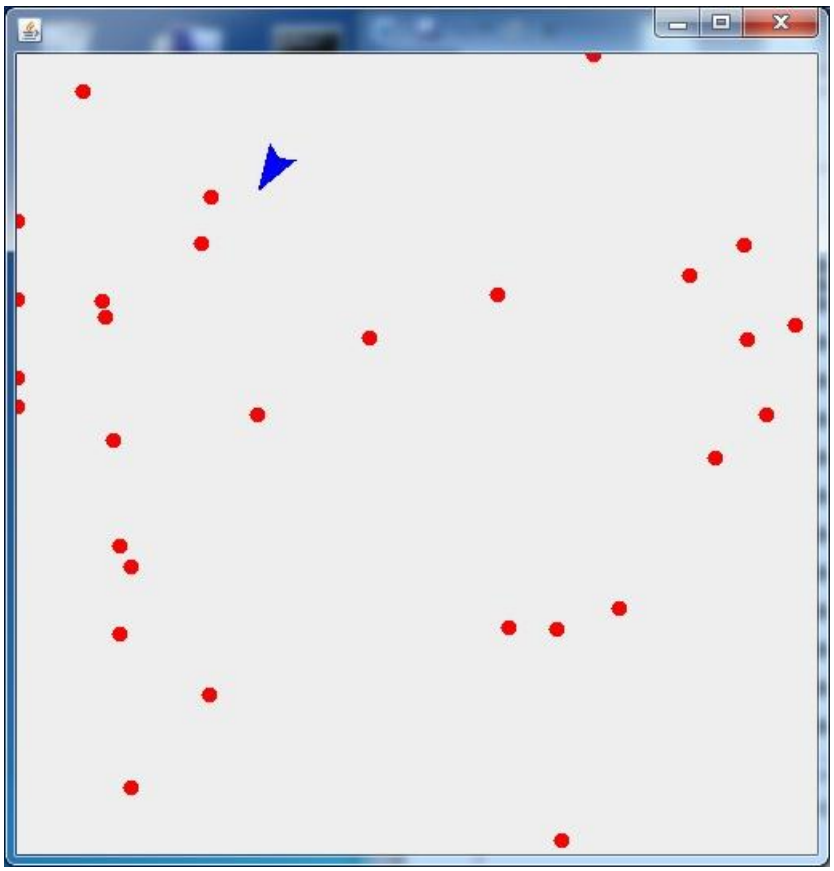
→ access **end** through synchronized methods **set_end** , **get_end**

→ access **COMdir** through synchronized methods **set_COMdir** , **evolve** (of **SpaceCraft**)

→ access **COMthr** through synchronized methods **set_COMthr** , **evolve** (of **SpaceCraft**)

*example6: spacecraft and asteroids video game*





*example 6 (continued):*

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


public class P
{  public static void main(String[] arg)
   {  Game game = new Game();
      game.play();  }  }


class Game
{
   boolean end = false;    synchronized void set_end(boolean v)  {  end = v;  }
                           synchronized boolean get_end()  {  return end;  }
   static int size;    static long dt;
   World world;    Gui gui;

   Game()
   {  size = 500;    dt = 100;
      world = new World();    gui = new Gui(this);  }

   void play()
   {
      while (!get_end())
      {
         long t_start = System.currentTimeMillis();

         world.evolve();    gui.p.repaint();

         long dt_real = System.currentTimeMillis() - t_start;
         if (dt_real < dt)  try {Thread.sleep(dt  - dt_real);} catch(InterruptedException e){}
                       else  System.out.println("PC too slow; please increase dt");
      }

      System.exit(0);
   }
}
```

*example 6 (continued):*

```
class World
{
   static final int N = 30;
   Obj[] obj = new Obj[N];    SpaceCraft spacecraft;

   World()
   {  obj[0] = spacecraft = new SpaceCraft();
      for (int i = 1 ; i < N ; i++)  obj[i] = new Asteroid();  }

   void evolve()
   {  for (int i = 0 ; i < N ; i++)  if (obj[i] != null)  obj[i].evolve();  }

   void draw(Graphics g)
   {  for (int i = 0 ; i < N ; i++)  if (obj[i] != null)  obj[i].draw(g);  }
}


abstract class Obj
{  abstract void evolve();
   abstract void draw(Graphics g);  }


class Asteroid extends Obj
{
   int x , y , vx , vy;

   Asteroid()
   {  x = (int)(Math.random() * Game.size);
      y = (int)(Math.random() * Game.size);
      vx = (int)((2*Math.random()-1) * Game.size / 300.0);
      vy = (int)((2*Math.random()-1) * Game.size / 300.0);  }

   void evolve()
   {  x += vx * Game.dt / 1000.0;    y += vy * Game.dt / 1000.0;  }

   void draw(Graphics g)
   {  g.setColor(Color.red);    g.fillOval(x - 5 , y - 5 , 10 , 10);  }
}
```

*example 6 (continued):*

```
class SpaceCraft extends Obj
{
   int x , y;
   double dir;    int COMdir;    synchronized void set_COMdir(int c)  { COMdir = c; }
   double thr;    int COMthr;    synchronized void set_COMthr(int c)  { COMthr = c; }

   SpaceCraft()
   {  x = y = Game.size / 2;
      dir = 0.0;    thr = 0.0;  }

   synchronized void evolve()
   {
      dir += COMdir * (2.0 * Math.PI / 5) * Game.dt / 1000.0;
      thr += COMthr * (1.0 / 5) * Game.dt / 1000.0;
      if (thr < 0.0)  thr = 0.0;    if (thr > 1.0)  thr = 1.0;

      double v = thr * Game.size / 10.0;
      x += (int)(v * Math.cos(dir) * Game.dt / 1000.0);
      y += (int)(v * Math.sin(dir)  * Game.dt / 1000.0);
   }


   int n = 4;
   int[] X = new int[] {   15 , -15 , -10 ,  -15 };
   int[] Y = new int[] {     0 ,  10 ,     0 ,  -10 };

   void draw(Graphics g)
   {
      Graphics2D g2 = (Graphics2D)g;
      java.awt.geom.AffineTransform at0 = g2.getTransform();

      g2.translate(x , y);    g2.rotate(dir);

      g.setColor(Color.blue);    g.fillPolygon(X , Y , n);

      g2.setTransform(at0);
   }
}
```

*example 6 (continued):*

```
class Gui
{
   Game game;    JFrame f;    DrawingPanel p;

   class DrawingPanel extends JPanel
   {  public void paintComponent(Graphics g)
      {  super.paintComponent(g);
         game.world.draw(g);  }  }

   Gui(Game game)
   {
      this.game = game;
      f = new JFrame();    f.setFocusable(true);    f.setVisible(true);
      p= new DrawingPanel();    f.getContentPane().add(p , BorderLayout.CENTER);

      f.addKeyListener(new KeyAdapter()
      {  public void keyPressed(KeyEvent e)
         {   int c = e.getKeyCode();    SpaceCraft sc = Gui.this.game.world.spacecraft;
             switch (c)
            { case KeyEvent.VK_LEFT:     sc.set_COMdir(-1);  break;
              case KeyEvent.VK_RIGHT:  sc.set_COMdir(+1);  break;
              case KeyEvent.VK_DOWN:  sc.set_COMthr(-1);  break;
              case KeyEvent.VK_UP:        sc.set_COMthr(+1);  break;
              case KeyEvent.VK_ESCAPE:  Gui.this.game.set_end(true);  break;  } } } );

      f.addKeyListener(new KeyAdapter()
      {  public void keyReleased(KeyEvent e)
         {  int c = e.getKeyCode();    SpaceCraft sc = Gui.this.game.world.spacecraft;
             switch (c)
             { case KeyEvent.VK_LEFT:     sc.set_COMdir(0);  break;
               case KeyEvent.VK_RIGHT:  sc.set_COMdir(0);  break;
               case KeyEvent.VK_DOWN:  sc.set_COMthr(0);  break;
               case KeyEvent.VK_UP:        sc.set_COMthr(0);  break;  } } } );

      f.setSize(new Dimension(Game.size + 16, Game.size + 38));
   }
}
```