Programmierpraktikum

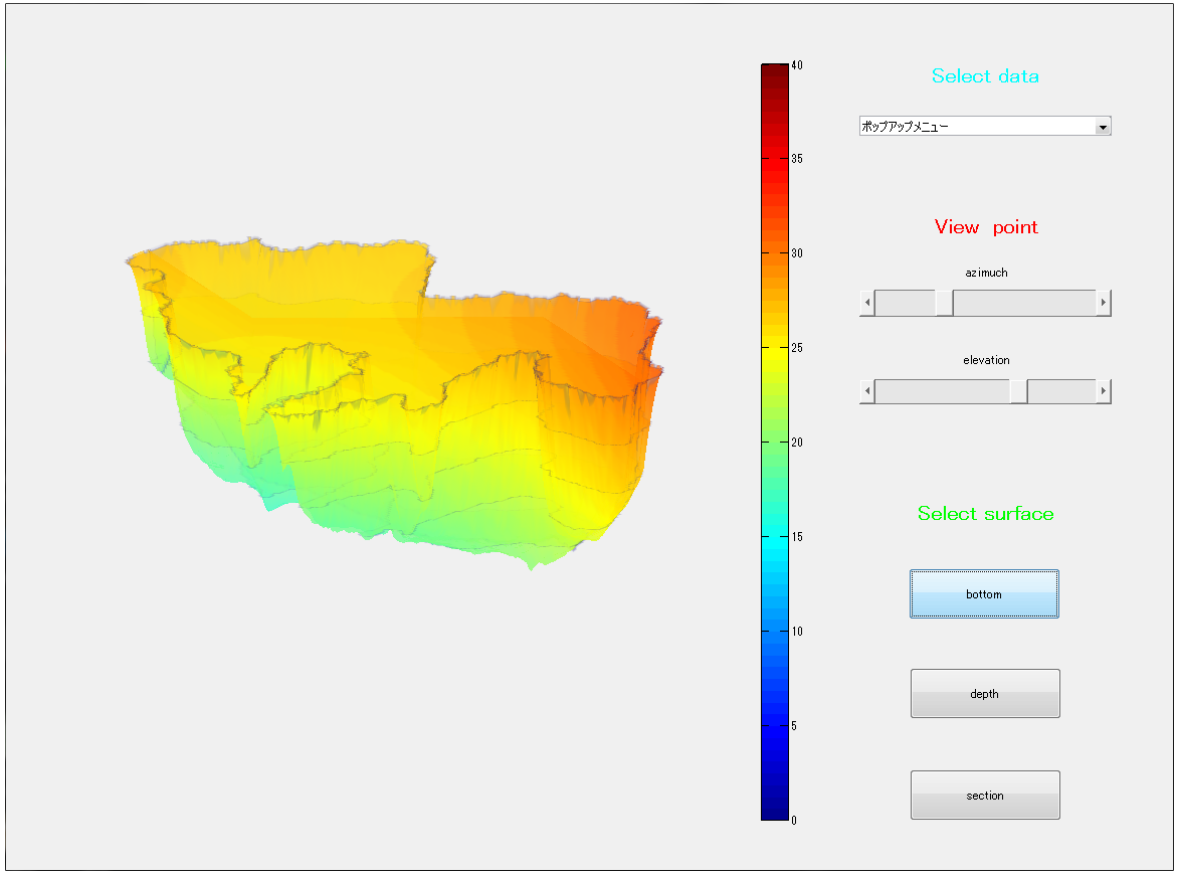
Claudius Gros, SS2012

Institut für theoretische Physik
Goethe-University Frankfurt a.M.

Swing
Graphical User Interface (GUI)

Graphical User Interface (GUI)

- interaction and control of a program via a graphical interface
- for most humans GUI == program


40
35
30
25
20
15
10
5
0
Select data
ポップアップメニュー
View point
azimuch
elevation
Select surface
bottom
depth
section

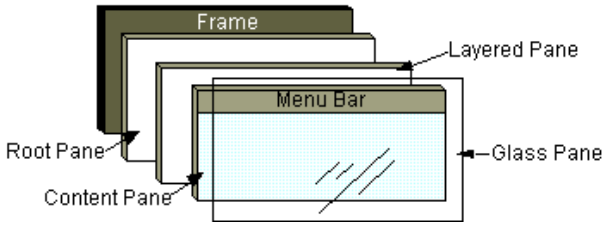
Swing example

```
 1. import javax.swing.*;                          // 'x' for extension
 2. import java.util.*;                            // for sleeping
 3.
 4. public class SimpleFrame {
 5. public static void main(String args[]) throws InterruptedException
 6.                                                // needed for sleeping
 7.   {
 8.   JFrame jf = new JFrame();                    // instantiation of a JFrame
 9.   JLabel jlb = new JLabel("Hello World");      // instantiation of a JLabel
10.   jf.getContentPane().add(jlb);                // add label to content of frame
11.   jf.setSize(300, 200);                        // set frame size
12.   jf.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
13. //  what to to when clicken away the window: remove it, free resources
14.   jf.setVisible(true);                         // show frame
15.   System.out.println("JFrame panel lauched");
16. //
17.   Thread.sleep(3000);                          // wait 3 seconds
18.   jf.setLocationRelativeTo(null);              // move to center of screen
19.   System.out.println("JFrame panel relocated");
```

```
20. //
21.   Thread.sleep(3000);
22.   jlb.setFont(jlb.getFont().deriveFont(20f)); // set font size
23.   System.out.println("JLabel font resized");
24. //
25.   }
26. }    // end of class SimpleFrame
```

layers (panes)

- computer graphics images have several layers
- `JFrame` contains several `pane` objects:
  root, content, layered, glass
- normal applications only use the *content* `pane`,
  it contains all content



frame and window hierarchy

- `JFrame extends Frame` (with title and border)
  `Frame extends Window` (no title and border)
  `Window extends Container`
- `Container extends Component`
  `Component extends Object`

- `Container`: Abstract Window Toolkit (AWT),
  contains component objects
- `Component`: interactive graphical object,
  buttons, checkboxes, scrollbars, ...
- many things can be done using a `Frame` or a `Window`,
  a `JFrame` offers most utilities

moving colored JFrame

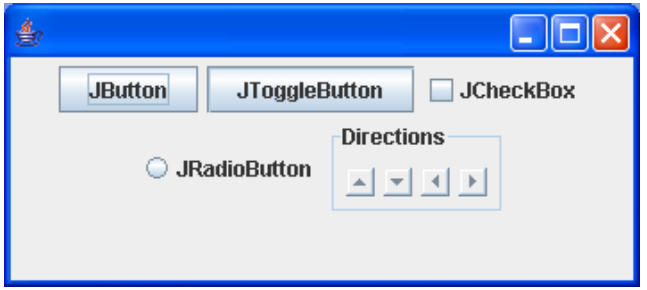
- `Color(red,green,blue)`: rgb color $\in [0, 255]$
- `Point point comp.getLocation(null)`:
  a point object is a 2D vector (`point.x`,`point.y`)

```
 1. import javax.swing.*;                          // for the frame
 2. import java.awt.*;                             // for colors
 3. import java.util.*;                            // for sleeping
 4.
 5. public class MovingScreen extends JFrame {
 6.
 7. /* Constructor of MovingScreen.
 8.  * Using 'this' is not necessary but increases readability.
 9.  * 'title', 'size', .. are member variables of JFrame.
10.  **/
11. public MovingScreen() {
12.   this.setTitle("moving colors");
13.   this.setSize(300, 200);
14.   this.setDefaultCloseOperation(EXIT_ON_CLOSE); // exit program on close
```

```
15.   }
16.
17. public static void main(String[] args) throws InterruptedException {
18.   MovingScreen ex = new MovingScreen();          // instantiation
19.   ex.setLocationRelativeTo(null);                // relative to screen center
20.   ex.setVisible(true);
21.   int screenCenter_x = ex.getLocation(null).x;  // Point.x
22.   int screenCenter_y = ex.getLocation(null).y;  // Point.y
23.   for (int i=0;i<255;i++)
24.     {
25.     Thread.sleep(40);
26.     ex.getContentPane().setBackground(new Color(0,i,255-i));
27.     ex.setLocation(screenCenter_x+i,screenCenter_y+i);
28.     }
29.   } // end of MovingScreen.main()
30. }   // end of class MovingScreen
```

adding content to a frame

adding content using a container



- `container.add(Component comp)`
  is inherited by the `JFrame`
- the `JPanel` container is convenient
  (comes with a layout manager):
  `jframe.add(JPanel jp)`
- `JPanel extends JComponent`
  `JComponent extends Container`

adding content directly to the ContentPane

- `jFrame.getContentPane.add(Component comp)`

- doese not come with a predefined layout manger:
  `jFrame.getContentPane.setLayout(layoutManger)`

Swing components

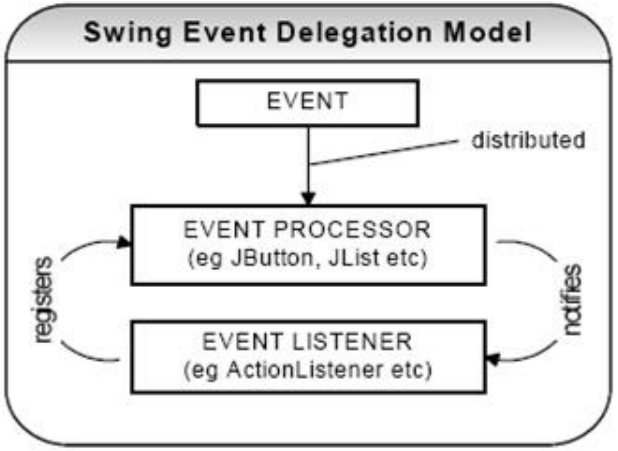
- there are several layout managers for arranging components on *contentPane*

```
 1. import java.awt.*;                                  // for the container
 2. import javax.swing.*;                               // for the frame
 3.
 4. public class AddContentPane {
 5.
 6. public static void main(String[] args) {
 7.   JFrame frame = new JFrame("Various Components");
 8.   frame.setSize(600, 150);
 9.   frame.setLocationRelativeTo(null);
10.   frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
11.
12.   Container content = frame.getContentPane();      // add to contentPane
13.   content.setLayout(new FlowLayout());             // one is necessary
14.
15.   content.add(new JButton("JButton"));
16.   content.add(new JToggleButton("JToggleButton"));
```

```
17.   content.add(new JCheckBox("JCheckBox"));
18.   content.add(new Label("Label"));
19.   content.add(new JRadioButton("JRadioButton"));
20.
21.   Font font = new Font("Arial Black",Font.PLAIN,20); // select a new font
22.   Component[] allComp = content.getComponents();     // get all components
23.
24.   for (Component comp : allComp)                     // iteration
25.     {
26.     comp.setBackground(new Color(153,204,153));      // of component
27.     comp.setForeground(new Color(204,51,51));        // change font color
28.     comp.setFont(font);                              // change font sizes
29.     }
30.   content.setBackground(new Color(204,204,204));     // of contentPane
31.
32.   frame.setVisible(true);                          // show
33.   }
34. }
```

events - making things happen

- **a GUI needs to be interactive**
  react on user action



Java listener

- listen to what happens in the world:
  - *other components of the program*
  - *communication with remote hosts*
  - *incoming emails, ...*
- `ActionListener`: waits until event occurs and then performs `.actionPerformed()`
- register action listener with component

```
1. public class MyClass implements ActionListener {
```

```
2. //   ActionListener is an Interfaces not a class and hence
3. //   implemented and not extended
4.  public void actionPerformed (){
5. //   mandatory: do stuff when action has been triggered
6.  }
7. }
```

simple Swing events

- *actionEvent*.getSource()
  returns an Object: the source of the event

```
 1. import java.awt.event.*;                // for events
 2. import java.awt.*;                      // for colors
 3. import javax.swing.*;                   // for JFrame
 4.
 5. public class SimpleEvent
 6.         extends JFrame                  // extending an class
 7.         implements ActionListener       // implementing an interface
 8. {
 9. JButton button1, button2, button3;     // needed several times
10. JLabel label;
11.
12. /* Constructor.
13.  **/
14.
15. public SimpleEvent(){
16.   this.setTitle("click buttons");       // title
```

```
17.   this.setSize(400, 200);               // size
18.   this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
19.   this.setLocationRelativeTo(null);     // position
20.
21.   this.label = new JLabel();            // initially empty
22.
23.   this.button1 = new JButton("text");
24.   this.button2 = new JButton("color");
25.   this.button3 = new JButton("exit");
26.
27.   this.button1.addActionListener(this); // register listeners
28.   this.button2.addActionListener(this); // 'this' implements
29.   this.button3.addActionListener(this); // an ActionListener
30.
31.   JPanel panel = new JPanel();          // for the content
32.   panel.add(button1);
33.   panel.add(button2);
34.   panel.add(button3);
35.   panel.add(label);
36.   panel.validate();
37.   this.add(panel);
38. }    // end of constructor
39.
```

```
40. /* Main.
41.  **/
42.
43. public static void main(String[] args) {
44.   SimpleEvent window = new SimpleEvent();
45.   window.setVisible(true);
46. }
47.
48. /* ActionPerformed is mandatory since SimpleEvent
49.  * implements ActionListener.
50.  **/
51.
52. public void actionPerformed (ActionEvent ae){
53.   if(ae.getSource() == this.button1){
54.     label.setText("you successfully clicked button 1");
55.     }
56.
57.   if(ae.getSource() == this.button2){
58.     label.setText(("changing colors of button 2"));
59.     button2.setBackground(new Color(230,230,130));
60.     }
61.
62.   if (ae.getSource() == this.button3){
```

```
63.      System.exit(0);
64.      }
65.    }  // end of SimpleEvent.actionPerformed()
66. }    // end of SimpleEvent
```

list of events

event typ	event processor	event trigger
Action	JButton	button activation
Adjustment	JScrollBar	on change
CaretEvent	JTextComponent	moving cursor
Component	Component	changing visbility, size
Container	Container	changing content
Focus	JComponent	new focus (for typing)
Key	JComponent	keyboard activation
Menu	JMenu	on selection
Mouse	JComponent	mouse in/out
MouseMotion	JComponent	mouse motion

graphics and drawing

- all graphics operations onto a graphics object
- `Graphics2D extends Graphics`: more options
- `JComponent.paintComponent(Graphics g)`
  graphics objects automatically painted/repainted
- interactive components / graphics

```
 1. import javax.swing.*;           // for JFrame, JCompoent
 2. import java.awt.*;              // for graphics objects
 3.
 4. public class DrawingColor {
 5.   public static void main(String[] args) {
 6.   JFrame frame = new JFrame("drawing colors");
 7.   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 8.   frame.setSize(500,300);
 9.   frame.setLocationRelativeTo(null);
10.   frame.getContentPane().add(new MyComponent());  // add content
11.   frame.setVisible(true);
```

```
12.   }
13. }   // end of class DrawingColor
14.
15. /** User defined component, extending the
16.   * abstract class JComponent.
17.   */
18. class MyComponent extends JComponent{
19. /** Paints the component when first shown or needed.
20.   * Overrides the super.paintComponent of the
21.   * JComponent. Painting and drawing is down on a
22.   * Graphics object. Graphics2D extends Graphics and
23.   * has more options.
24.   */
25.   public void paintComponent(Graphics g){
26.   int width  = 200;
27.   int height = 120;
28. //
29.   g.setColor(Color.blue);                // select color
30.   g.fillOval(20,20,width,height);        // filled elipse
31.   g.setColor(Color.cyan);
32.   g.fillOval(24,24,width,height);
33. //
34.   g.setColor(Color.gray);
```

```
35.   g.fillRect(270+width/2,20+height/2,width/2,height/2);  // filled rectangle
36. //
37.   Graphics2D g2d = (Graphics2D)g;          // cast Graphics to Graphics2D
38.   g2d.setStroke(new BasicStroke(5));       // only availabe in Graphics2D
39.   g2d.setColor(Color.red);
40.   g2d.drawRect(270,20,width/2,height/2);   // borders thickness is stroke
41. //
42.   int[] xPoints = new int[200];
43.   int[] yPoints = new int[200];
44.   for (int x=0; x<200;x++)
45.     {
46.     xPoints[x] = 50 + 2*x;
47.     yPoints[x] = 200 + (int)(40.0*Math.sin(Math.PI*x/25.0));
48.     }
49.   g.setColor(Color.black);
50.   g.drawPolyline(xPoints,yPoints,xPoints.length);
51.   } // end of MyComponent.paintComponent()
52. }   // end of class MyComponent
```

exercise: GUI for Kepler problem

Write a GUI for the numerical solution of the Kepler problem with Runge-Kutta integration.

- add an JFrame with suitable layout manager to your Kepler program
- the GUI should allow the user to select all parameters
  - *starting position/velocity*
  - *power α of the force field*
  - *integration method*
- the orbits are drawn to a reserved part of the GUI